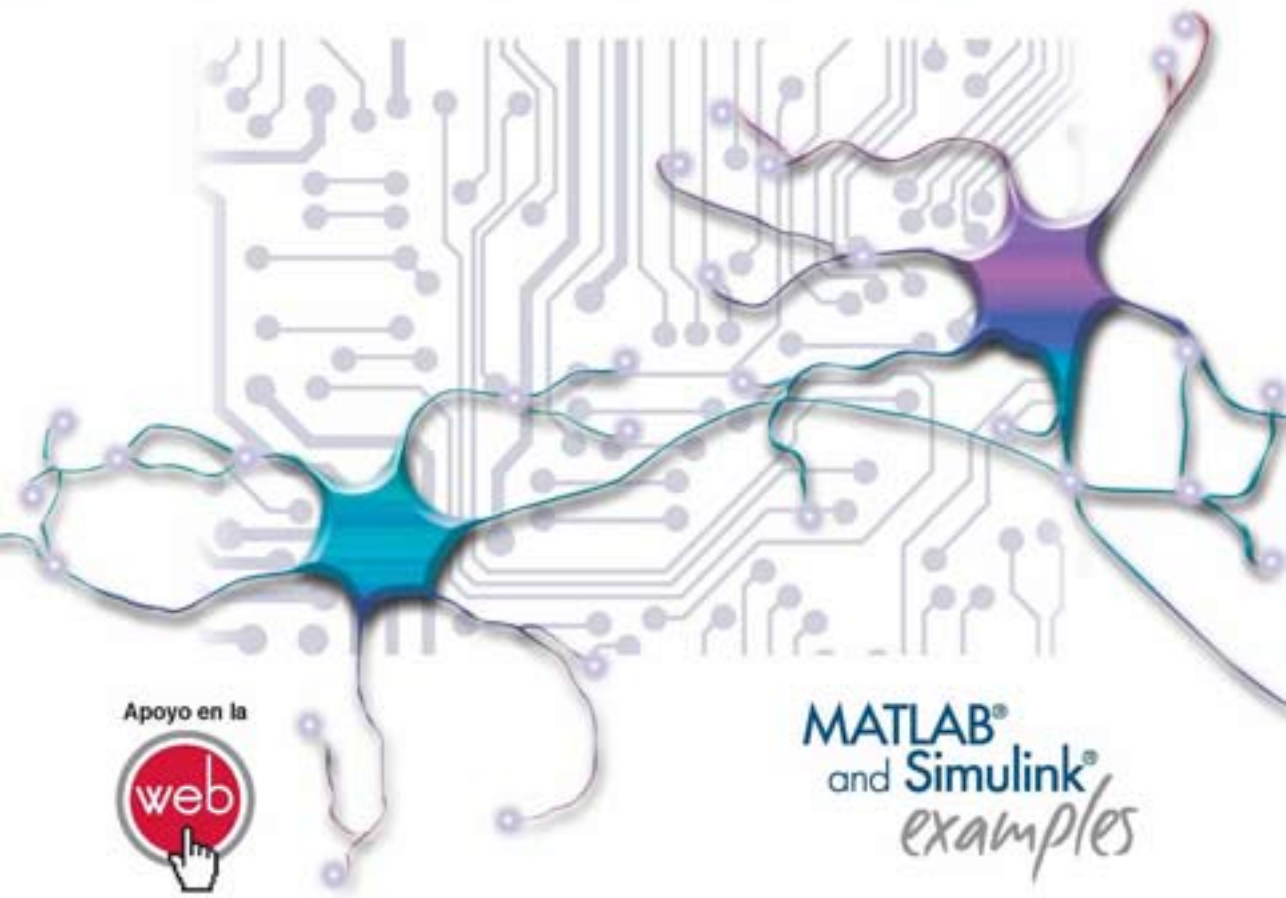
Pedro Ponce Cruz

INTELIGENCIA ARTIFICIAL

CON APLICACIONES A LA INGENIERÍA



Apoyo en la web

MATLAB® and Simulink® examples

Alfaomega

INTELIGENCIA ARTIFICIAL
CON APLICACIONES A LA INGENIERÍA

INTELIGENCIA ARTIFICIAL CON APLICACIONES A LA INGENIERÍA

Dr. Pedro Ponce Cruz

Alfaomega

Buenos Aires • Bogotá • México DF • Santiago de Chile

Editor
Alejandro Herrera

Revisor
Dr. David Moisés Terán Pérez

Corrector de estilo
Alejandro Cruz Ulloa

Gerente editorial
Marcelo Grillo Giannetto
mgrillo@alfaomega.com.mx

Diagramación
Editec

Datos catalográficos
Ponce, Pedro
Inteligencia artificial con aplicaciones a la ingeniería
Primera Edición
Alfaomega Grupo Editor, S.A. de C.V., México
ISBN: 978-607-7854-83-8
Formato: 17 × 23 cm Páginas: 376

Inteligencia artificial con aplicaciones a la ingeniería
Pedro Ponce Cruz

Primera edición: Alfaomega Grupo Editor, México, Julio 2010

Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

ISBN: 978-607-7854-83-8

Impreso en México. Printed in México.
Empresas del grupo:
México: Alfaomega Grupo Editor, S.A. de C.V. – Pitágoras 1139, Col. Del Valle, México, D.F. – C.P. 03100.
Tel.: (52-55) 5089-7740 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396
E-mail: atencionalcliente@alfaomega.com.mx
Colombia: Alfaomega Colombiana S.A. – Carrera 15 No. 64 A 29 – PBX (57-1) 2100122, Bogotá, Colombia,
Fax: (57-1) 6068648 – E-mail: scliente@alfaomega.com.co
Chile: Alfaomega Grupo Editor, S.A. – General del Canto 370-Providencia, Santiago, Chile
Tel.: (56-2) 235-4248 – Fax: (56-2) 235-5786 – E-mail: agechile@alfaomega.cl
Argentina: Alfaomega Grupo Editor Argentino, S.A. – Paraguay 1307 P.B. "11", Buenos Aires, Argentina,
C.P. 1057 – Tel.: (54-11) 4811-7183 / 8352, E-mail: ventas@alfaomegaeditor.com.ar

AGRADECIMIENTOS

- Quiero agradecer a cada uno de los profesores que me enseñaron dentro del aula el verdadero significado de ser un ingeniero.
- A todas las personas que se toman el tiempo para darme un buen consejo, para mejorar el desarrollo de mi vida profesional.
- A todas las personas que conforman mi familia y que me apoyan siempre de manera incondicional, ya que siento su afecto sin importar el lugar.
- A mi hijo Pedro Ponce, que es parte fundamental de mi existir.
- A mi madre por dejarme usar sus electrodomésticos como juguetes y siempre darme una nueva oportunidad de hacerlo mejor.
- A la memoria de la pequeña Sandy quien me enseñó que el tiempo que uno vive es valioso cuando deja recuerdos de las cosas lindas que se hacen. ¡Te extrañaré cada día por siempre Sandy!
- A veces las palabras no reflejan sentimientos pero el corazón siembre encuentra la forma más bella de trasmitir cada uno de ellos. Para mi compañero Fluffy.
- A Alejandro Herrera, mi editor, por hacer del manuscrito inicial esta obra bibliográfica.
- Al Instituto Tecnológico de Estudios Superiores de Monterrey Campus Ciudad de México por todo el apoyo brindado para generar esta obra y cada uno de mis proyectos de investigación.

Dr. Pedro Ponce Cruz

Ciudad de México, julio de 2010.

NOTA IMPORTANTE

Todas las marcas registradas utilizadas en este documento son propiedad de sus respectivos propietarios.

El uso de cualquier marca registrada en este texto, no confiere al autor o al editor ningún derecho de propiedad sobre tales marcas.

MATLAB®, Real-Time Workshop®, Simulink® son marcas registradas de The MathWorks™, se mencionan únicamente con fines didácticos y de identificación.

Mayor información de MATLAB® puede solicitarse a

The Mathworks., Inc.
3 Apple Hill Drive
Natick Ma. 017660-2098 USA
O en ***http://www.mathworks.com***.

El lector podrá solicitar:

Versión para estudiantes de MATLAB® y Simulink® en:
http://www.mathworks.com/academia/student_version

Copia de evaluación para el instructor en:
http://www.mathworks.com/programs/academia/eval.html

MENSAJE DEL EDITOR

Los conocimientos son esenciales en el desempeño profesional. Sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecerles a estudiantes y profesionales conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (TIC) para facilitar el aprendizaje. Libros como éste tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales con información actualizada.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento. Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente que lo guía hacia los objetivos y metas propuestas con diversas actividades pedagógicas para asegurar la asimilación del conocimiento y su extensión y actualización futuras.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos guía en diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.

EL AUTOR

Dr. Pedro Ponce Cruz

Estudió la carrera de Ingeniería en Control y Automatización en el Instituto Politécnico Nacional (IPN), graduándose 1995. Posteriormente realizó estudios de posgrado en el mismo Instituto, donde obtuvo los grados de Maestro en Ciencias y de Doctor en Ciencias, logrando en este último el mejor promedio del programa en 2001, ambos grados con especialidad en ingeniería eléctrica con opción en control automático.

Se desempeñó como ingeniero de campo y diseño en el Departamento de Speed Control AC, así como ingeniero de proyectos de desarrollo industrial Nivel II. Fue profesor del Departamento de Control y Automatización y coordinador de laboratorios de control automático en el Instituto Politécnico Nacional.

Es especialista en las áreas de: automatización de sistemas industriales, máquinas eléctricas, accionamientos eléctricos, electrónica de potencia, control convencional continuo y digital, sistemas inteligentes y redes neurales artificiales.

Cuenta en su haber con más de 60 publicaciones en revistas y congresos de renombre académico y es autor del libro *Máquinas eléctricas y técnicas modernas de control*, publicado por este sello editorial.

Fue miembro del Sistema Nacional de Investigadores y ha recibido numerosos reconocimientos, entre los que destacan:

- Mejor profesor en el área de la División de Ingeniería y Arquitectura del Tecnológico de Monterrey, Campus Ciudad de México (2006).
- Medalla al mérito académico como mejor profesor de posgrado de la Escuela de Graduados en Ingeniería y Arquitectura, Tecnológico de Monterrey, Campus Ciudad de México (2005).
- Reconocimiento por labor docente en la ESIME, Instituto Politécnico Nacional, México, D.F.
- Primer lugar en el Graphical System Design National Instruments 2009 en Austin, Texas, en el área de biotecnología, siendo el primer latinoamericano en ganar el premio.
- Primer lugar en el III Concurso Iberoamericano 2009, Nuevo México

Actualmente se desempeña como director de la Maestría en Ciencias de la Ingeniería del Instituto Tecnológico de Estudios Superiores de Monterrey (I.T.E.S.M.), Campus Ciudad de México.

CONTENIDO

A quién está dirigido *Inteligencia artificial con aplicaciones a la ingeniería* XIX
Acceso al material complementario XX
Prefacio XXI
Por qué la inteligencia artificial XXII

CAPÍTULO 1: INTELIGENCIA ARTIFICIAL 1

INTRODUCCIÓN 1
ANTECEDENTES DE LA INTELIGENCIA ARTIFICIAL 1
RAMAS QUE COMPONEN LA INTELIGENCIA ARTIFICIAL 2
LÓGICA DIFUSA 3
Introducción 3
Historia de la lógica difusa 3
REDES NEURALES ARTIFICIALES 6
Introducción 6
Historia de las redes neurales 6
Perceptrón 8
Redes de retropropagación (backpropagation) 9
ALGORITMOS GENÉTICOS 12
Introducción 12
Historia de los algoritmos genéticos 12
Definiciones 13
Herencia 14
¿Qué es herencia? 14
El código genético 15
Selección natural 15
Operaciones genéticas en cadenas binarias 17
Selección 17
Cruzamiento 17
Mutación 19
RESUMEN 19
EJEMPLOS 20
APLICACIONES 21
1. Desentrelazado de señales de video con lógica difusa 21
Procedimiento 22
Conclusiones 25
2. Marcadores anatómicos de los ventrículos del corazón 25
Procedimiento 25
Resultados 26
Conclusiones 27
3. Segmentación de imágenes cerebrales de resonancia magnética basada en redes neuronales 27
Procedimiento 28
Resultados y conclusiones 29
Referencias 29
4. Optimización de sistemas para tratamiento de agua (Austria, lógica difusa) 29
5. Monitoreo de glaucoma a través de redes neuronales 30
6. Algoritmos genéticos para el diseño de sistemas de MRI (magnetic resonance imaging) 32

CAPÍTULO 2: LÓGICA DIFUSA **33**

INTRODUCCIÓN **33**
Qué es una variable lingüística 34
Aplicaciones 34
Cámaras de video 34
Reconocimiento 34
Controladores 35
Sistemas de control en lazo abierto 35
Sistema de control en lazo cerrado 35
Uso de lógica difusa en el Transporte 35
Uso de lógica difusa en los sistemas de control 36
CONCEPTOS DE LÓGICA BOOLEANA Y DIFUSA **36**
LÓGICA BOOLEANA **38**
Axiomas de los conjuntos convencionales 39
Operaciones en la lógica convencional 39
Leyes de De Morgan 40
LÓGICA DIFUSA **40**
Lógica simbólica 41
Tautologías y quasi-tautologías 42
Representación de conjuntos difusos discretos 42
Operaciones en la lógica difusa empleando conjuntos difusos 43
Ejemplo de programa de operación difusa realizado en MATLAB® 43
Norma triangular (T) 44
Co-normas T (normas S) 45
Aseveraciones booleanas aplicadas a la lógica difusa 45
Operaciones entre conjuntos difusos 47
Producto de dos conjuntos difusos 47
Potencia de un conjunto difuso 47
Concentración 47
Dilación 48
Intensificación de contraste 48
Corte alfa 49
Propiedades de los conjuntos difusos 49
Funciones de membresía y sus partes básicas 49
Función de saturación 50
Función hombro 50
Función triangular 51
Función trapecio o Pi 52
Función “S” o sigmoidal 52
Descripción matemática de las funciones de membresía 53
Aplicaciones reales de las distintas funciones de membresía 54
Partes de una función de membresía 56
Cálculo de función de pertenencia 56
1. Método HORIZONTAL 57
2. Método VERTICAL 57
3. Método de comparación de parejas (Saaty, 1980) 57
4. Método basado en la especificación del problema 58
5. Método basado en la optimización de parámetros 58
6. Método basado en la agrupación difusa (Fuzzy Clustering) 58
El principio de extensión: generalización 58

PRINCIPIO DE EXTENSIÓN **61**
NÚMEROS DIFUSOS **61**
Suma de números difusos 62
RELACIONES NÍTIDAS Y DIFUSAS **64**
Producto cartesiano 64
Relaciones nítidas 64
Relaciones difusas 65
Composición 65
Composición sup-estrella 68
Operaciones con relaciones difusas 69
Unión 69
Intersección 69
Complemento 69
Producto cartesiano difuso y composición 69
Reglas difusas 69
Modus ponens y modus tollens 70
CONTROLADORES DIFUSOS **71**
Interfaz de difusificación 72
Base de conocimientos 72
Lógica de decisiones 73
Interfaz de desdifusificación 73
Método de centro de área o gravedad 74
Método de centro máximo 75
Método de izquierda máximo 76
Método de derecha máximo 76
Aproximación de sistemas difusos 77
Definición de las entradas y salidas del sistema 78
Ejemplo de un sistema difuso con retardos en la información para aproximaciones difusas 80
Funciones de membresía 80
Reglas lingüísticas 80
Superficie de salida 81
Diseño de controladores con base en Mamdani 81
Ejemplo 82
Aplicaciones reales de controladores difusos 85
Controlador difuso clásico 86
Ejemplo 90
Controladores P 91
Controladores PD 92
Controladores PI 94
Controlador PID 95
Simulación de un Control PID difuso 103
Controlador difuso con PID convencional como respaldo 104
Controlador difuso como sintonizador de PID convencional 104
Concepto de estabilidad 104
Punto de equilibrio 104
Asintóticamente estable 105
Entrada-cero de estabilidad 105
Teorema 1. (Estabilidad de Lyapunov para sistemas autónomos) 106
Estabilidad de Lyapunov para sistemas difusos 107
Teorema 107
La construcción para muestreo de datos 107

Controlador difuso-convencional autosintonizable por lógica difusa 108
Método Ziegler-Nichols 109
Controlador proporcional difuso 109
PD autosintonizable 112
PI autosintonizable 113
PID autosintonizable 114
Análisis de resultados 115
Autosintonización vs. Ziegler-Nichols 116
Controlador difuso como programador de ganancias para PID 120
Estabilidad 120
Diseño con base en Sugeno 121
Ejemplo 121
ALGORITMO DEL RAZONAMIENTO 122
Ejemplo 122
Diseño digital con base en estabilidad 123
Ejemplo 126
EJEMPLO SISTEMA DIFUSO SUGENO 127
EJEMPLO DE MOTOR DC 129
EJEMPLO DE SISTEMA DE 2 ENTRADAS 131
MÉTODOS DE INFERENCIA 132
Método de Tsukamoto 132
Método de Larsen 132
Resumen de mecanismos de inferencia 132
AGRUPAMIENTOS DIFUSOS 133
Validez de un cluster 133
Clusters nítidos 134
Ejemplo 135
Clusters difusos 138
Ejemplo 139
Aplicaciones reales de los agrupamientos difusos 141
Aproximaciones de sistemas reales por el método de Sugeno 143
Aproximación de un deshumidificador desecante 146
Aproximación de un potenciómetro 147
Aproximación de un sensor óptico 149
Ejemplo de aplicación de método para optimización de clusters con lógica difusa tipo Mamdani 150
Calculadora difusa por método Mamdani 155
Caracterización de un controlador tipo PID mediante un controlador tipo Sugeno 156
Controlador difuso basado en control directo del par (DTC) 159
Control de velocidad sin sensores usando control directo del par (DTC) basado en la modulación del ancho de pulso mediante vectores espaciales (SVPWM) 162
Agrupamientos difusos con pesos 164
Segmentación de imágenes médicas a través de agrupamientos difusos 166
Aproximación de un modelado de sentimientos humanos basado en el reconocimiento de expresiones faciales con lógica difusa 168
Aproximación a los sentimientos humanos a través de lógica difusa 169
PROGRAMAS BÁSICOS EN MATLAB® 172
SATURACIÓN 172
HOMBRO 172
TRIANGULAR 173
TRAPEZOIDAL 174
SIGMOIDAL 175

CLUSTERS DIFUSOS Y SISTEMA SUGENO **176**
CALCULADORA DIFUSA MATLAB® **179**

CAPÍTULO 3 REDES NEURALES ARTIFICIALES **193**

REDES NEURALES BIOLÓGICAS **193**
Fundamentos biológicos de las redes neurales naturales 193
Máquinas inteligentes 194
Sistema eléctrico neuronal 195
MODELOS DE NEURONAS **196**
Ruido 197
Neuronas de dos estados 197
La neurona genérica 197
APLICACIONES DE LAS REDES NEURALES ARTIFICIALES (RNA) **198**
DEFINICIÓN DE UNA RED NEURONAL ARTIFICIAL **198**
FUNCIONES DE ACTIVACIÓN **199**
Función escalón 200
Función lineal y mixta 200
Función tangente hiperbólica 201
Función sigmoidal 201
Función de Gauss 202
TOPOLOGÍAS DE LAS REDES NEURALES **202**
Elementos de una red neuronal artificial 202
ENTRENAMIENTO DE LAS REDES NEURALES **203**
REDES DE UNA CAPA **203**
Perceptrón 204
Separación de variables linealmente separables con el perceptrón 206
ADALINE (Adaptive Linear Neuron) **208**
Problema del operador lógico XOR por uso del perceptrón 210
Desarrollo 210
OR **211**
AND **212**
XOR **212**
Control de un motor de pasos con un grupo de perceptrones 216
Teorema de Kolmogorov 224
REDES MULTICAPA **224**
Perceptrón multicapa 225
Redes de retropropagación (backpropagation) 225
Principios para entrenar una red multicapa empleando el algoritmo de retropropagación 225
Redes neurales - Retropropagación del error 228
Capas intermedias 230
Algoritmo de retropropagación con momento (Backpropagation with Momentum) 231
DISEÑO DE FILTROS FIR CON REDES NEURALES ARTIFICIALES **232**
Filtro 232
Filtros adaptativos digitales 233
Emulación del filtro empleando una red neuronal programada en MATLAB® 234
EJEMPLO RECONOCIMIENTO DE LETRAS EMPLEANDO ENTRENAMIENTO DE RETROPROPAGACIÓN DEL ERROR **234**
Resultados 235
REDES AUTOORGANIZABLES **236**
Aprendizaje asociativo 236

Red de una sola neurona 237
Tipos de estímulos 237
Ejemplo 237
Interpretación de la Regla de Hebb en asociadores lineales 238
TOPOLOGÍA DE REDES NEURONALES EMPLEADAS PARA LA CLASIFICACIÓN, PREDICCIÓN Y RECONOCIMIENTO DE PATRONES 238
Red Instar 238
Red Outstar 239
Redes Competitivas 239
Red de Kohonen 240
Red de Hamming 241
Mapas de Autoorganización (SOM) 241
Learning Vector Quantization (LVQ) 241
Redes Recurrentes 242
Red Hopfield 242
Redes ANFIS 244
Algoritmo de un sistema ANFIS 247
Algoritmo de entrenamiento para ANFIS 248
Arquitectura de ANFIS 256
Método de mínimos cuadrados 258
Mínimos cuadrados recursivos 259
Ejemplo ANFIS con línea de comandos 259
Ejemplo sistema ANFIS empleando ANFIS EDIT de MATLAB® 260
Empleo de función Genfis1 265
EJEMPLO DE UN SISTEMA ANFIS Y DIFUSO PARA EL MODELADO DE MÁQUINAS DE CORRIENTE ALTERNA, EN UN ESQUEMA DE CONTROL VECTORIAL 267
Introducción 267
Etapas del control difuso tipo Sugeno 268
Fusificación 268
Evaluación de reglas 268
Desarrollo 268
Control vectorial 268
Modelo difuso del motor de inducción 271
Fusificación 271
Edición de reglas 272
Modelo ANFIS del motor de inducción 273
Control vectorial difuso 275
APROXIMADOR NEURO-DIFUSO CON CLUSTERS Y REDES NEURALES TRIGONOMÉTRICAS 277
Entrenamiento de retropropagación 279
Redes neurales basadas en Fourier 280
Cálculo de la función de la red neuronal basada en Fourier 281
Establecimiento de los pesos 282

CAPÍTULO 4 ALGORITMOS GENÉTICOS 285

CHARLES DARWIN Y LA TEORÍA DE LA EVOLUCIÓN 285
ALGORITMOS GENÉTICOS 287
Introducción 287
Algoritmos genéticos 289
Definiciones 290

Operaciones genéticas en cadenas binarias 292
Selección 292
Cruzamiento 293
Mutación 294
Algoritmo 294
Resumen 295
Ejemplo 296
Análisis 297
El Teorema del Schema 298
La óptima asignación de los procesos 300
Paralelismo implícito 300
Conjunto difuso de sintonización 301
Codificación de un subconjunto difuso en un intervalo 301
Funciones de aptitud estándar 302

CAPÍTULO 5 EJEMPLO DE AG EN MATLAB® 306

DETERMINAR LA IMPEDANCIA NECESARIA DE UN COMPONENTE PARA QUE UN CIRCUITO AC LE TRANSFIERA LA MÁXIMA POTENCIA DE ENERGÍA 306
Introducción 306
Aplicaciones 306
Problema de máxima transferencia de potencia 306
El algoritmo genético 307
1. Problema de optimización 307
2. Representación 308
3. Población inicial 308
4. Evaluación 309
5. Crear una nueva población 310
ALGORITMOS GENÉTICOS 313
ALGORITMO GENÉTICO BÁSICO CONVENCIONAL BINARIO 313
ALGORITMO GENERACIÓN DE NUEVOS INDIVIDUOS MEDIANTE OPERACIONES DE CRUZA Y MUTACIÓN 314
ALGORITMO DE SELECCIÓN PROPORCIONAL O RULETA 315
ARCHIVOS M DE MATLAB® PARA EL ALGORITMO 315
MAIN 315
FUNCIÓN OBJETIVO 316
EVAL. POBLACIÓN 319
EVAL. EACH 319
CONVERTIR BIT2NUM 320
NEXT POPULATION 320

Anexo A MATLAB® GENETIC ALGORITHMS TOOLBOX

Introducción 323
Sección 1. Declaración de función aptitud y restricciones 324
Fitness function/Función aptitud 324
Number of variables/ Número de variables 324
Constraints/ Restricciones 326
Linear inequalities/ Desigualdades lineales 326
Sección 2. Área de gráficos 327
Plot interval/Intervalo de trazado 327
Best fitness/Mejor aptitud 328

Best individual/Mejor individuo 328
Distance/Distancia 329
Expectation/Expectativa 329
Genealogy/Genealogía 330
Range/Rango 330
Score diversity/Diversidad de puntuación 331
Scores/Puntuación o ponderación 331
Selection/Selección 332
Stopping/Detención 332
Max constrain/Máxima violación 333
Custom function/Función personalizada 333
Sección 3. Resultados de la función aptitud 333
Use random status from previous run/ Uso aleatorio de la corrida anterior 334
Current generation/Generación actual 334
Status and results/ Estado y resultado 334
Sección 4. Alternativas de optimización para la función aptitud 335
Population /Población 335
Population Type /Tipo de la población 335
Creation function/Función de la creación (CreationFcn) 336
InitialPopulation/Población inicial 336
InicialScores/Puntuación inicial 336
PopInitRange /Rango inicial 336
Fitness Scaling/Escala de la función de ajuste o evaluación 336
FitnessScalingFcn/Función del escalamiento 336
Shift linear/Cambio Lineal (@fitscalingshiftlinear) 337
Custom/Personalizado 337
Selection/Selección 338
Stochastic Uniform/ Estocástico uniforme (@ selectionstochunif) 338
Remainder/Resto (@ selectionremainder) 338
Uniform/Uniforme (@ selectionuniform) 338
Roulette/Ruleta (@ selectionroulette) 338
Tournament/Torneo (@ selectiontournament) 338
Custom/Personalizado 338
Reproduction/Reproducción 339
Elite Count/Conteo elite 339
Crossover fraction/Fracción de cruzamiento 339
Mutation/Mutación 339
Mutation function/Función mutación 340
Uniform/Uniforme (mutación uniforme) 341
Custom/personalizado 341
Crossover/Cruzamiento 341
Scattered/Dispersos (@ crossoverscattered) 341
Single Point/Un solo punto (@crossoversinglepoint) 342
Two point/Dos puntos (@ crossovertwopoint) 342
Intermediate/Intermedio (@crossoverintermediate) 342
Heuristic/Heurística (@crossoverheuristic) 343
Custom/Personalizado 343
Migration/Migración 343
Direction/Dirección (MigrationDirection) 344
Interval/Intervalo 344
Fraction/Fracción 344

Algorithm settings/Parámetros del algoritmo 344
Initial penalty/Penalidad inicial 344
Penalty factor/Factor de penalización 344
Hybrid Function/Función de hibridación 345
Stopping Criteria/Criterio de detención 345
Generations/Generaciones 345
Time Limit/Tiempo límite 345
Fitness Limit/Límite de ajuste 345
Stall Generations/ Generaciones recesivas 345
Stall Time/ Tiempo de retardo (StallTimeLimit) 346
Output Function/Función de salida 346
History to new window/Historia de una nueva ventana (@gaoutputgen) 346
Custom/Personalizado 346
Estructura de la función de salida 346
Display to command window/Despliegue en la ventana de comandos 346
Vectorize/Vectorizar 347
Referencias 347
Bibliografía 348

A QUIÉN ESTÁ DIRIGIDO

Inteligencia artificial con aplicaciones a la ingeniería

Este libro está dirigido a profesores, alumnos y profesionistas de cualquier disciplina de la ingeniería, que busquen entender y aplicar los conocimientos de inteligencia artificial en su área de acción correspondiente.

A lo largo de la obra se explican con mucho detalle los principios básicos de cada uno de los métodos de inteligencia artificial, los cuales se ejemplifican con diferentes aplicaciones para que el lector pueda diseñar y desarrollar soluciones empleando estas herramientas. Además, en cada ejemplo se expone claramente las posibilidades y limitaciones del método respectivo.

Como se dará cuenta el lector, estas herramientas son un acercamiento a lo que muchos científicos han dedicado sus vidas: el entendimiento del ser humano para la generación de nuevas herramientas que se aproximen a la forma de razonamiento, aprendizaje y evolución humana.

Asimismo, estas herramientas se pueden combinar con las existentes para generar nuevas características que puedan proporcionar mejores resultados. En esta obra se plantea la solución a diferentes problemas como el modelado de sistemas, predicción de comportamientos dinámicos, métodos de control empleando inteligencia artificial y se abre una gama de posibilidades con los cuales el lector podrá proponer diferentes aplicaciones.

En la actualidad el empleo de la inteligencia artificial crea el mundo en que vivimos, por tanto no conocer estos métodos es una gran desventaja para cualquier ingeniero.

Dr. Pedro Ponce Cruz

ACCESO AL MATERIAL COMPLEMENTARIO

Para tener acceso al material complementario de *Inteligencia artificial con aplicaciones a la ingeniería*, se deben realizar los siguientes pasos:

1. Ir a la página: ***http://virtual.alfaomega.com.mx/***

2. En el la sección 

situada en el panel derecho de la página seleccionar el botón


3. Una vez ubicado en la sección Libros con material adicional, seleccionar la imagen correspondiente a este libro y descargar el material adicional.

4. Ahora disfrute su trabajo con *Inteligencia artificial con aplicaciones a la ingeniería*.

PREFACIO

En general, los métodos de inteligencia artificial (IA) son una respuesta al deseo de aproximar el comportamiento y el pensamiento humano a diversos sistemas para la solución de determinadas problemáticas. Por ello, no es de sorprender que actualmente se tiene sistemas muy avanzados que pueden emular ciertas características humanas, sin embargo aún nos encontramos muy lejos de poder recrear algunas otras.

En la actualidad los métodos de la inteligencia artificial (IA) tienen un gran auge y muchos investigadores se encuentran estudiando nuevas alternativas en el área. Hoy en día es común el empleo de sistemas que utilizan la IA para su funcionamiento cotidiano, entre ellos los equipos electrodomésticos como lavadoras, hornos de microondas, cámaras de video, e inclusive sistemas de transporte.

Lo que se pretende con estos métodos en ingeniería es resolver los problemas, no sólo de una manera novedosa, sino sobre todo tener mejores soluciones, más eficientes y mejor planeadas.

El presente libro aborda los temas más importantes de la IA que se pueden emplear en ingeniería, y los expone de una manera sencilla y accesible. Entre estos temas destacan la lógica difusa, las redes neurales, los sistemas neuro-difusos y los algoritmos genéticos. Cada uno de éstos se trata en forma detallada para que el lector interesado pueda realizar de manera natural la solución de problemas de ingeniería, comprendiendo además la teoría que sustenta al método respectivo.

Como se sabe, los sistemas difusos tienen la capacidad de emular la forma de la inferencia humana y además pueden almacenar la experiencia en forma lingüística. Por su parte, las redes neurales artificiales se sustentan básicamente en que pueden realizar el aprendizaje y la clasificación de patrones a través de la simulación de neuronas biológicas. Por otro lado, los sistemas neuro-difusos tienen la capacidad de aprender y almacenar conocimiento en forma lingüística. Por último, los sistemas genéticos tratan de imitar la selección natural en la que se busca tener al individuo más fuerte, el cual posibilitará la realización de determinados procesos de optimización.

En esta obra, cada uno de estos métodos y técnicas se adecua a la solución de diferentes problemas mediante una serie de ejercicios, por lo que en cada capítulo se tiene un conjunto de problemas de distinta naturaleza, los cuales sirven de apoyo para completar el proceso de aprendizaje. En muchos de estos casos se emplea MATLAB® para elaborar el diseño del sistema de IA que se requiere. Cabe señalar que la mayoría de estos casos se realizó durante los cursos que imparto, los cuales tienen el objetivo de que los estudiantes participen de manera activa en la solución de problemas de ingeniería mediante el uso de los métodos de la IA, proponiendo mejores alternativas de solución.

Dr. Pedro Ponce Cruz, Ciudad de México.

POR QUÉ LA INTELIGENCIA ARTIFICIAL

El hombre se ha aplicado a sí mismo el nombre científico de hombre sabio (*homo sapiens*) como una valoración de la trascendencia de sus (nuestras) habilidades mentales, tanto para la vida cotidiana como para el propio sentido de identidad. Los esfuerzos de la Inteligencia Artificial (IA), por su parte, se enfocan en lograr la comprensión de entidades inteligentes. Por ello, una de las razones de su estudio y análisis es aprender acerca de los propios seres humanos, pero a diferencia de la Filosofía y de la Psicología (que también se ocupan de la *inteligencia*), los esfuerzos de la IA están encaminados tanto a la construcción de entidades inteligentes, como a su comprensión.

Otra razón más por la que se estudia la IA es porque las entidades inteligentes así construidas son interesantes y útiles. No obstante las fases tempranas en las que todavía se encuentra, mediante la IA ha sido posible crear diversos productos de trascendencia y sorprendentes. Si bien es imposible pronosticar con precisión lo que se puede esperar de esta disciplina en el futuro, es evidente que las computadoras que posean una inteligencia a nivel humano o superior, tendrán repercusiones importantes en la vida diaria, así como en el devenir de la civilización (Russell y Norving, 1996).

Inteligencia artificial es una acepción acuñada a mediados del siglo XX, cuyo desarrollo se ha caracterizado por una sucesión de periodos alternativos de éxito y abandono de la misma. Al principio, la idea intuitiva de la IA creó expectativas que no siempre han sido cubiertas, y desde luego no en el grado que se había esperado, de manera un tanto ilusoria. Pero actualmente se puede considerar que el enfoque computacional inteligente no depende de inmediatos y probados resultados, sino que está avalado por sus logros y su desarrollo a lo largo de varias décadas, por lo cual se ha consolidado en el ámbito de la computación como una acepción totalmente asumida, aunque sometida todavía a controversia en algunos sectores científicos (Pajares Martinsan y Santos Peñas, 2006).

Lo que hoy se conoce como IA empezó hacia 1960 cuando en el Instituto Tecnológico de Massachusetts (MIT, por sus siglas en inglés), John McCarthy creó el *LISP* (el primer lenguaje de investigación dentro de la IA). Sin embargo, el término IA suele atribuírsele a Marvin Minsky, también del MIT, quien en 1961 escribió un artículo titulado “Hacia la Inteligencia Artificial” (The Institute of Radio Engineers Proceedings). Los años sesenta del siglo pasado fueron un intenso periodo de optimismo hacia la posibilidad de hacer que una computadora pensase. Después de todo, esos años contemplaron la primera computadora que jugaba ajedrez, las primeras pruebas matemáticas informatizadas, y el ya famoso e igualmente bien conocido Programa ELIZA que fue escrito en el MIT por Joseph Weisenbaum en 1964. El programa ELIZA actuaba como un psicoanalizador. En este tipo de análisis, el psiquiatra toma un papel pasivo generalmente repitiendo las propias declaraciones del paciente, en vez de llevar el peso de la conversación. Posteriormente, en la década de los años setenta se creó el *PROLOG*, obra de Alain Colmerauer, en Masella, Francia, en 1972. PROLOG era un lenguaje diseñado para ayudar a resolver problemas relativos a la IA. Este lenguaje poseía un gran número de características especiales tales como una base de datos incorporada y una sintaxis bastante simple (Schildt, 1990).

De hecho, el problema que aborda la IA es uno de los más complejos: ¿cómo es posible que un diminuto cerebro (sea biológico o electrónico), tenga capacidad para percibir, comprender, predecir y manipular un mundo que en tamaño y complejidad lo excede?

La IA siempre ha tenido como modelo natural las funcionalidades inteligentes del hombre, enfocándose en distintos aspectos. Su primera motivación fue intentar construir máquinas que pudieran pensar como el ser humano, o al menos emularle en alguna capacidad de tal modo que denotara cierta inteligencia.

La IA es una de las disciplinas más nuevas. Formalmente se inicia en 1956 cuando se acuñó el término, no obstante que ya para entonces se había estado trabajando en ello durante cinco años. Junto con la genética moderna, la IA es el campo en que la mayoría de los científicos de otras disciplinas les gustaría trabajar. El estudio de la inteligencia es una de las disciplinas más antiguas. Desde hace más de 2 000 años los filósofos se han esforzado por comprender cómo se ve, se aprende, se recuerda y se razona, así como la manera en que esas actividades deberían realizarse.

Los temas fundamentales de la Inteligencia Artificial

El campo de la IA se compone de varias áreas de estudio, las más comunes e importantes son:

- Búsqueda de soluciones
- Sistemas expertos
- Procesamiento del lenguaje natural
- Reconocimiento de modelos
- Robótica
- Aprendizaje de las máquinas
- Lógica
- Incertidumbre y “lógica difusa”

La llegada de las computadoras a principios de la década de los años cincuenta del siglo pasado, permitió pasar de la especulación de las charlas de café a su abordaje mediante una auténtica disciplina teórico-experimental. En realidad, la IA ha resultado ser mucho más compleja de lo que inicialmente se pensó al principio, ya que las ideas modernas que se tienen de ella se caracterizan por su gran riqueza, sutilidad y por lo interesantes que son. Las tablas 1(a) y 1(b) muestran algunas de las definiciones de la IA. Estas definiciones varían en torno a dos dimensiones principales. Las que se establecen en la tabla 1(a) se refieren a procesos mentales y al razonamiento, en tanto que las de la tabla 1(b) se correlacionan con la conducta. Las definiciones que están a la izquierda de la tabla 1(a) miden la condición deseable en función de la eficiencia humana, mientras que las de la derecha lo hacen según el concepto de inteligencia ideal denominado **racionalidad**. Se considera que un sistema es racional si hace lo correcto.

A lo largo de la historia se han adoptado los cuatro enfoques que se aprecian en las tablas a y b. Desde luego existe una tensión entre enfoques que se centran en lo humano y los que se enfocan en *la racionalidad*. El enfoque centrado en el comportamiento humano constituye una ciencia empírica que entraña el empleo de hipótesis y de su confirmación mediante experimentos, mientras que el enfoque racionalista combina matemáticas e ingeniería (Russell y Noving, 1996).

Tabla a. Algunas definiciones de las IA.

"La interesante tarea de lograr que las computadoras piensen … máquinas con mente, en su amplio sentido literal" (Haugeland, 1985).	"El estudio de las facultades mentales mediante el uso de modelos computacionales" (Charniak y McDermott, 1985).
"[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades tales como toma de decisiones, resolución de problemas, aprendizaje, …" (Bellman, 1978).	"El estudio de los cálculos que permiten percibir, razonar y actuar" (Winston, 1992).
"El arte de crear máquinas con capacidad de realizar funciones que realizadas por personas requieren de inteligencia" (Kurzweil, 1990).	"Un campo de estudio que se enfoca en la explicación y emulación de la conducta inteligente en función de procesos computacionales" (Schalkoff, 1990).
"El estudio de cómo lograr que las computadoras realicen tareas que, por el momento, los humanos hacen mejor" (Rich y Knight, 1991).	"La rama de la ciencia de la computación que se ocupa de la automatización de la conducta inteligente" (Luger y Stubblefield, 1993).

Tabla b. Categorías en que se clasifica la IA.

Sistemas que piensan como humanos.	Sistemas que piensan racionalmente.
Sistemas que actúan como humanos.	Sistemas que actúan racionalmente.

A continuación se explica cada uno de los cuatro enfoques con que es posible estudiar y analizar la IA.

1. Actuar como humano: el enfoque de la prueba de Turing

Mediante la prueba de Turing, propuesta por Alan Turing (1950), se intenta ofrecer una satisfactoria definición operativa de lo que es la inteligencia.

Turing definió una conducta inteligente como la capacidad de lograr eficiencia a nivel humano en todas las actividades de tipo cognoscitivo, suficiente para engañar a un elevador. Brevemente, la prueba que Turing propuso consistía en que un humano interrogase a una computadora por medio de un teletipo; la prueba se consideraba aprobada si el evaluador era capaz de determinar si una computadora o un humano era quien había respondido las preguntas en el otro extremo de la terminal.

Cabe señalar que hoy por hoy, el trabajo que entraña programar una computadora para pasar la prueba es considerable. La computadora debería ser capaz de lo siguiente:

- **Procesar un lenguaje natural.** Para así poder establecer comunicación satisfactoria, sea en inglés o en cualquier otro idioma humano.
- **Representar el conocimiento.** Para así guardar toda la información que se le haya dado antes o durante el interrogatorio.
- **Razonar automáticamente.** Con el propósito de utilizar la información guardada al responder preguntas y obtener nuevas conclusiones.
- **Autoaprendizaje de la máquina.** Para que se adapte a nuevas circunstancias y para detectar y extrapolar esquemas determinados.

En la prueba de Turing deliberadamente se evitó la interacción física directa entre evaluador y computadora, dado que para medir la inteligencia era innecesario simular físicamente a un humano. Sin embargo, en la denominada prueba total de Turing se utiliza una señal de video para que el evaluador pueda calificar la capacidad de percepción del evaluado, y también para que aquél pueda pasar objetos físicos "a través de la trampa".

Para aprobar la prueba total de Turing, es necesario que la computadora esté dotada de:

- **Vista.** Que le permita percibir objetos.
- **Robótica.** Para desplazar los objetos.

En el campo de la IA no se han hecho muchos esfuerzos para pasar la prueba de Turing. La necesidad de actuar como los humanos se presenta básicamente cuando los programas de IA deben interactuar con gente, por ejemplo cuando un sistema experto de procesamiento de lenguaje natural entabla diálogo con un usuario.

Programas como los anteriores deberán comportarse de acuerdo con ciertas convenciones normales de las interacciones humanas con el objetivo de poder entenderlos.

Por otra parte, la manera de elaborar representaciones y de razonar en qué se basan estos sistemas podrá o no conformarse de acuerdo con un modelo humano.

2. Pensar como humano: el enfoque del modelo cognoscitivo

Para poder afirmar que un programa determinado utiliza algún tipo de razonamiento humano, previamente habrá que definir cómo piensan los seres humanos. Habrá que penetrar en el funcionamiento de la mente humana.

Hay dos formas de hacer esto:

- Mediante la introspección (para intentar atrapar nuestros propios pensamientos conforme éstos se van dando).
- Mediante la realización de experimentos psicológicos.

Una vez que se cuente con una teoría bastante precisa de la mente, puede procederse a expresar tal teoría en un programa de computadora. Si los datos de entrada/salida del programa

y la duración de su comportamiento corresponden a los de la conducta humana, entonces existe evidencia de que algunos de los mecanismos del programa también funcionan en los seres humanos. En el caso de Newell y Simon (1961), creadores del Solucionador General de Problemas (SGP), no bastó con que su programa resolviera correctamente los problemas propuestos.

Lo que a estos investigadores les interesaba fundamentalmente era seguir la pista de los pasos del razonamiento y compararla con la ruta seguida por sujetos humanos a los que se propuso los mismos problemas. Esta actitud contrasta fuertemente con la de otro investigador de la época, Wang (1960), a quien le importaba obtener respuestas correctas independientemente de cómo las obtendría un ser humano. De modo que en el campo interdisciplinario de la ciencia cognitiva concurren modelos computacionales de IA y técnicas experimentales de psicología, para intentar elaborar teorías precisas y verificables del funcionamiento de la mente humana.

3. Pensar racionalmente: el enfoque de las leyes del pensamiento

El filósofo griego Aristóteles fue uno de los primeros en intentar codificar "*la manera correcta de pensar*", es decir, establecer procesos de pensamiento irrefutables. Sus famosos silogismos son esquemas de estructuras de argumentación mediante las cuales siempre se llega a conclusiones correctas si se parte de premisas correctas. Por ejemplo: "*Sócrates es un hombre; todos los hombres son mortales; por lo tanto, Sócrates es hombre y es mortal*". Dichas leyes del pensamiento debían gobernar la manera de operar de la mente. Así se inauguró el campo de la *Lógica*.

El desarrollo de la lógica formal a fines del siglo XIX y a principios del siglo XX permitió tener una notación precisa para representar aseveraciones relacionadas con todo lo que existe en el mundo, así como sus relaciones mutuas[1]. Ya para 1965 existían programas que, teniendo tiempo y memoria suficiente, podían describir un problema en notación lógica y encontrarle solución, siempre y cuando dicha solución existiera (de no existir dicha solución, el programa continuaría sin cesar buscándola).

En la IA, la tradición logicista se esfuerza por elaborar programas como el anterior para crear sistemas inteligentes.

Este enfoque presenta dos obstáculos. En primer lugar, no es fácil recibir un conocimiento informal y expresarlo en los términos formales que exige la notación lógica, especialmente cuando el conocimiento tiene menos de 100% de certidumbre. En segundo lugar, hay una gran diferencia entre la posibilidad de resolver un problema "en principio", y realmente hacerlo en la práctica. Incluso problemas que entrañan una docena de elementos agotarían la capacidad de cómputo de cualquier computadora a menos que se cuente con lineamientos sobre los pasos de razonamiento que hay que utilizar primero. Si bien los dos obstáculos anteriores están presentes en todo intento de construir sistemas de razonamiento computacional, fue en la tradición logicista donde surgieron por primera vez debido a que la capacidad de los sistemas de representación y de razonamiento está bien definida y estudiada a profundidad.

[1] A diferencia de lo que sucede con la notación de la Aritmética común, en cuyo caso prácticamente sólo se representan aseveraciones acerca de la igualdad y desigualdad entre números.

4. Actuar en forma racional: el enfoque del agente racional

Actuar racionalmente implica actuar de manera tal que se logren los objetivos deseados con base en ciertos supuestos. Un *agente* es algo capaz de percibir y actuar. De acuerdo con este enfoque, se considera la IA como el estudio y construcción de agentes racionales.

En el caso del enfoque de la IA según las *"leyes del pensamiento"*, todo el énfasis se ponía en hacer inferencias correctas. La obtención de estas inferencias a veces forma parte de lo que se considera un agente racional, puesto que una manera de actuar racionalmente es el razonamiento lógico que asegure la obtención de un resultado determinado, con lo que se actuará de conformidad con tal razonamiento. Sin embargo, el efectuar una inferencia correcta no siempre depende de la racionalidad, pues hay situaciones en las que no existe algo que se pueda considerar lo que correctamente debería hacerse, y sin embargo hay que decidirse por un curso de acción. Existen también maneras de actuar racionalmente que de ninguna manera entrañan inferencia alguna.

Todas esas "habilidades cognoscitivas" que se necesitan en la prueba de Turing permiten emprender acciones racionales. Por lo tanto, es necesario tener la capacidad para representar el conocimiento y razonar con base en él, pues de esta manera se podrían tomar decisiones correctas en una amplia gama de situaciones. Es necesario ser capaces de generar oraciones comprensibles en lenguaje natural, puesto que la enunciación de tales oraciones permite desenvolverse en una sociedad compleja. El aprendizaje no se emprende por pura erudición, sino porque el profundizar en el conocimiento de cómo funciona el mundo facilitará concebir mejores estrategias para manejarse en él. La percepción visual no sólo es algo divertido, sino que es algo necesario para darse una mejor idea de lo que una acción determinada puede producir.

Estudiar la IA adoptando el enfoque del diseño de un agente racional ofrece dos ventajas. Primera, es más general que el enfoque de las "leyes del pensamiento", dado que el efectuar inferencias correctas es sólo un mecanismo útil para garantizar la racionalidad, pero no es un mecanismo necesario. Segunda, es más afín a la manera en que se ha producido el avance científico que los enfoques basados en la conducta o pensamiento humanos, toda vez que se define claramente lo que será la norma de la racionalidad, norma que es de aplicación general. Por el contrario, la conducta humana se adapta bien sólo en un entorno específico y, en parte, es producto de un proceso evolutivo complejo y en gran parte ignoto, cuya perfección todavía se ve distante.

Hay que tener presente desde ahora que más tarde que temprano podrá constatarse que lograr la racionalidad perfecta (siempre hacer lo correcto) no es posible en entornos complejos. El cómputo que implican es excesivo.

David Moisés Terán Pérez **Luis Alejandro Herrera García**

1

INTELIGENCIA ARTIFICIAL

INTRODUCCIÓN

Para abordar el concepto de inteligencia artificial, tal vez cabría plantearse primero la siguiente pregunta: "¿qué es la inteligencia?" Sin duda, se trata de una pregunta difícil cuya respuesta aún no ha sido resuelta totalmente, la cual sigue desconcertando tanto a los biólogos como a los psicólogos y filósofos de nuestra época. Por supuesto que el objetivo de este libro no es zanjar la discusión. Más bien la intención es presentar algunas ideas en torno a la noción de inteligencia que nos ayuden a identificar ciertas características distintivas de la denominada inteligencia artificial (IA).

Se podría comenzar por destacar algunas propiedades generales que presenta la inteligencia humana, como por ejemplo la habilidad de enfrentar nuevas situaciones, la habilidad de resolver problemas, de responder preguntas, elaborar planes, etc. Desde sus inicios, el hombre se representó el mundo real mediante símbolos, los cuales constituyen la base del lenguaje humano. En este sentido, se podría considerar a la IA como un dialecto simbólico constituido por cadenas de caracteres que representan conceptos del mundo real. De hecho, los procesos simbólicos son una característica esencial de la IA. A partir de lo expuesto es posible formular una definición más aproximada de nuestro objeto de estudio: la IA es una rama de las ciencias computacionales que se ocupa de los símbolos y métodos no algorítmicos para la resolución de problemas.

ANTECEDENTES DE LA INTELIGENCIA ARTIFICIAL

Se podría considerar que unos de los primeros pasos hacia la IA fueron dados hace mucho tiempo por Aristóteles (384-322 a.C.), cuando se dispuso a explicar y codificar ciertos estilos de razonamiento deductivo que él llamó *silogismos*. Otro intento sería el de Ramón Llull (d.C. 1235-1316), místico y poeta catalán, quien construyó un conjunto de ruedas llamado *Ars Magna*, el cual se suponía iba a ser una máquina capaz de responder todas las preguntas.

Por su parte, Martin Gardner [Gardner 1982] atribuye a Gottfried Leibniz (1646-1716) el sueño de "un álgebra universal por el cual todos los conocimientos, incluyendo las verdades morales y metafísicas, pueden algún día ser interpuestos dentro de un sistema deductivo único". Sin embargo, no existió un progreso sustancial hasta que George Boole [Boole 1854] comenzó a desarrollar los fundamentos de la lógica proposicional. El objeto de Boole fue, entre otros: "recoger... algunos indicios probables sobre la naturaleza y la constitución de la mente humana". Poco después, Gottlob Frege propuso un sistema de

notación para el razonamiento mecánico y al hacerlo inventó gran parte de lo que hoy conocemos como cálculo proposicional (lógica matemática moderna) [Frege 1879].

En 1958, John McCarthy, responsable de introducir el término "inteligencia artificial", propuso utilizar el cálculo proposicional como un idioma para representar y utilizar el conocimiento en un sistema que denominó la "Advice Taker". A este sistema se le tenía que decir qué hacer en vez de ser programado. Una aplicación modesta pero influyente de estas ideas fue realizada por Cordell Green en su sistema llamado QA3.

Lógicos del siglo XX, entre ellos Kart Codel, Stephen Kleene, Emil Post, Alonzo Church y Alan Turing, formalizaron y aclararon mucho de lo que podía y no podía hacerse con los sistemas de lógica y de cálculo. En fechas más recientes, científicos de la computación como Stephen Cook y Richard Karp, descubrieron clases de cálculos que aunque parecían posibles en principio, podrían requerir cantidades totalmente impracticables de tiempo y memoria (almacenamiento).

Algunos filósofos [Lucas 1961, Penrose 1989, Penrose 1994] interpretaron como una confirmación que la inteligencia humana nunca será mecanizada. Warren McCulloch y Walter Pitts escribieron teorías acerca de las relaciones entre los elementos de cálculo simple y las neuronas biológicas [McCuylloch y Pitts 1943]. Otro trabajo realizado por Frank Rosenblatt [1962] exploró el uso de redes llamadas *perceptrones*. Otras corrientes de trabajo, entre ellos la cibernética [Wiener 1948], la psicología cognitiva, la lingüística computacional [Chomsky 1965] y la teoría de control adaptable, contribuyeron a la matriz intelectual de la IA y su desarrollo.

Gran parte del trabajo inicial de la IA se desarrolló en la década de 1960 y principios de los setenta en programas como General Problem Solver (GPS) de Allan Newell, Cliff Show y Herbert Simon [Newell, Show y Simon 1959, Newell y Show 1963]. Otros sistemas que influyeron son: la integración simbólica [Slagle 1963], álgebra word [Bobrow 1968], analogy puzzles [Evans 1968] y control y robots móviles [Nilsson 1984b]. Muchos de estos sistemas son el tema de un artículo llamado *Computers and Thought* [Feigenbaum y Feldman 1963].

Hacia finales de los setenta y principios de los ochenta, algunos programas que se desarrollaron contenían mayor capacidad y conocimientos necesarios para imitar el desempeño humano de expertos en varias tareas. El primer programa que se le atribuye la demostración de la importancia de grandes cantidades de conocimiento y dominio específico es DENDRAL, un sistema de predicción de la estructura de las moléculas orgánicas que considera su fórmula química y el análisis de espectrograma de masa [Feigenbaum, Buchanan and Lederberg 1971]. Le siguieron otros "sistemas expertos" como por ejemplo [Shortliffe 1976, Millar Pople y Myers 1982] sistemas computacionales configurables [McDermott 1982] y otros más.

En mayo 11 de 1997, un programa de IBM llamado *Deep Blue* derrotó al actual campeón mundial de ajedrez, Garry Kasparov. Por otra parte, Larry Roberts desarrolló uno de los primeros programas de análisis de escena [Roberts 1963]. Este trabajo fue seguido por una amplia labor de máquinas de visión (visión artificial) [Nalga 1993]. Otros proyectos que se pueden mencionar son CYC [Goha and Lenat 1990, Lenat y Goha 1990, Lenat 1995], una de cuyas metas era recolectar e interpretar gran cantidad de información para su conocimiento. Aunque el interés en las redes neurales se estancó un poco después de los trabajos pioneros de Frank Rosenblatt en los últimos años de la década de 1950, se reanudó con energía en los años ochenta. En la actualidad hay distintas aplicaciones con la IA.

Softbots [Etzioni y Weld 1994] son agentes de software que deambulan por la Internet, encontrando información que piensan será útil a sus usuarios al acceder a Internet. La presión constante para mejorar las capacidades de los robots y los agentes de software motivarán y guiarán mayores investigaciones de IA en los años venideros.

RAMAS QUE COMPONEN LA INTELIGENCIA ARTIFICIAL

Como se mencionó, existen varios elementos que componen la ciencia de la IA, dentro de los cuales se pueden encontrar tres grandes ramas:

- Lógica difusa
- Redes neurales artificiales
- Algoritmos genéticos

Cada una consta de características especiales, así como de una función específica. En las siguientes secciones se exponen más específicamente dichas tecnologías.

LÓGICA DIFUSA

Introducción

Las computadoras manejan datos precisos que se reducen a cadenas de unos (1) y ceros (0) y proposiciones que son ciertas y falsas. El cerebro humano puede razonar con información que involucra incertidumbre o juicios de valor como: "el aire es frío" o "la velocidad es rápida". Además, las personas tienen un sentido común que les permite razonar en un mundo donde las cosas son parcialmente ciertas.

La lógica difusa es una rama de la IA que le permite a una computadora analizar información del mundo real en una escala entre lo falso y verdadero. Los matemáticos dedicados a la lógica en la década de 1920 definieron un concepto clave: *todo es cuestión de grado*. La lógica difusa manipula conceptos vagos como "caliente" o "húmedo" y permite a los ingenieros construir televisores, acondicionadores de aire, lavadores y otros dispositivos que juzgan información difícil de definir. Los sistemas difusos son una alternativa a las nociones de pertenencia y lógica que se iniciaron en la Grecia antigua.

El lenguaje natural maneja conceptos no precisos como "hace frío" o "el precio es alto". Cuando se traduce el lenguaje humano al contexto de la lógica clásica se pierde la riqueza del significado, pérdida que puede ser importante si estamos diseñando un sistema experto. Suponiendo que se diseña un sistema experto en reproducir la capacidad de diagnóstico de un médico, el ingeniero sabe que el médico se basa en medidas exactas, pero el diagnóstico y la receta de las medicinas están llenos de razonamiento difuso.

Cuando los matemáticos carecen de algoritmos que dictan cómo un sistema debe responder a ciertas entradas, la lógica difusa puede controlar o describir el sistema usando reglas de sentido común que se refieren a cantidades indefinidas. Los sistemas difusos frecuentemente tienen reglas tomadas de expertos, pero cuando no hay experto los sistemas difusos adaptivos aprenden las reglas observando cómo la gente manipula sistemas reales.

Historia de la lógica difusa

El estudio moderno de la lógica difusa y de las contradicciones parciales tiene sus orígenes en el siglo XX, cuando Bertrand Russell retomó una antigua paradoja griega. Según ésta, un cretense afirma que todos los cretenses mienten. Así que ¿miente o dice la verdad? Si miente, dice la verdad y no miente. Si no miente, entonces dice la verdad y miente. Ambos casos conducen a una contradicción porque la frase es cierta y falsa a la vez. En este caso la lógica clásica se rinde. Pero la lógica difusa dice que la respuesta es mitad verdad y mitad falsa, 50% de la frase del cretense es cierta y 50% de frase es falsa. El cretense miente 50% del tiempo y no miente la otra mitad. Cuando la pertenencia es menor del total, un sistema bivalente puede simplificar el problema aproximado hacia cero o hacia el 100%. Pero 50% no se puede aproximar hacia arriba o hacia abajo.

En la década de 1920 e independientemente de Rusell, el lógico polaco Jan Lukasiewicz trabajó los principios de la lógica multivaluada, en éstos las proposiciones pueden tomar valores verdaderos fraccionales entre los unos (1) y los ceros (0) de la lógica clásica.

En 1965 Lotfi A. Zadeh, en aquel entonces director del Departamento de Ingeniería Eléctrica de la Universidad de California en Berkeley, publicó *Fuzzy Sets*. Este artículo describe las matemáticas de los

conjuntos difusos y por extensión de la lógica difusa, y este trabajo le dio nombre a su campo. Zadeh aplicó la lógica de Lukasiewicz a cada objeto en un conjunto y creó un álgebra completa para conjuntos difusos. Esta teoría propone funciones de pertenencia (o los valores falso y verdadero) sobre el rango [0.0, 1.0].

La lógica difusa se aplicó a mediados de la década de 1970 por Ebrahim H. Mamdani en el Queen Mary Collage en Londres. Mamdani diseñó un controlador difuso para un motor a vapor. Desde entonces el término *lógica difusa* es sinónimo de cualquier sistema matemático o computacional que razona con lógica difusa. La noción de sistemas difusos consiste en que los valores verdaderos (en lógica difusa) o valores de pertenencia (en conjuntos difusos) se indican en un número entre [0.0, 1.0], donde 0.0 representa falsedad total y 1.0 significa verdad absoluta. Por ejemplo, la frase "Lisa tiene 1 año de vida" tiene un valor verdadero, por ejemplo, de 0.9. La frase se puede trabajar en términos de conjuntos como: "Lisa es un miembro del conjunto de gente pequeña", en términos de conjuntos difusos.

$$\mu_{gente\_pequeña}(\text{Lisa}) = 0.9, \text{ donde } \mu \text{ es la función de pertenencia al conjunto de gente pequeña.}$$

Es importante distinguir entre sistemas difusos y probabilidad: los dos operan sobre el mismo rango numérico, pero los conceptos son distintos. La frase arriba mencionada en términos probabilísticos es: "Hay un 90% de probabilidad que Lisa sea pequeña".

La diferencia semántica es importante. La frase probabilística supone que "Lisa es o no es pequeña", hay un 90% de probabilidad de conocer la categoría en que se encuentra. En contraste, la frase difusa supone que "Lisa es más o menos pequeña". Los grados difusos no son lo mismo que probabilidades. Las probabilidades miden si algo va a ocurrir o no. Los niveles difusos miden el grado en el cual algo ocurre o alguna condición existe.

La lógica difusa hoy en día es muy común y se halla en diferentes sectores de la tecnología, tanto en la electrónica como el control, las matemáticas, la robótica, etc. El objetivo principal de la lógica difusa es crear un sistema basado en el comportamiento y pensamiento humanos. Esto se logra gracias al planteamiento de un modelo en cualquier contexto y traducirlo a reglas gramaticales o lenguaje humano. La clave de la lógica difusa se basa en la experiencia. El sistema toma el banco de conocimiento del experto, ya sea de mecánica, construcción, fotografía, computación, etc., y con él crea sus reglas para desarrollar una propuesta.

Esta metodología se considera compleja, pero el ser humano asimila todos los días las instrucciones con este tipo de enseñanza. Un ejemplo muy sencillo es como el niño aprende a patear un balón, donde la persona que lo instruye le indica con qué fuerza debe golpear el objeto. Nunca el padre, por ejemplo, le dirá al niño el número newtons necesarios para que el objeto sea movido. Tan sólo hace falta decir "mucha" o "poca fuerza", y entonces el niño entenderá de qué se trata. De igual forma el sistema entenderá cuánto es mucho y cuánto es poco si en el banco de datos se determina un rango de fuerza, en el que poco sea 0-3 newtons y "mucho" sea 3-8 newtons para mover la pelota.

Tener un rango de "mucho", "poco" abre una gama de posibilidades de la fuerza con que puede ser golpeado el objeto. Los términos "mucho" y "poco" no se encasillan en una sola cantidad, aspecto que algunos sistemas en la vida cotidiana sí lo realizan. Los interruptores de la luz son un claro ejemplo ya que sólo funcionan de dos maneras: apagado o encendido, lo que se traduciría en la lógica booleana en 1 o 0. En la figura 1.1 se muestra las operaciones básicas de la lógica booleana.

En la lógica difusa no se puede tomar valores únicos para saber si un elemento pertenece o no. Se debe tomar valores entre el 0-1, por ello las opciones de respuesta son mayores y a veces infinitas si no se acota adecuadamente el sistema o fenómeno en estudio. La figura 1.2 ejemplifica mejor la diferencia entre ambas propuestas; también se aprecia el abanico de posibilidades que maneja la lógica difusa. Del mismo modo se puede plantear una situación similar para distintas áreas de conocimiento, ciencia o tecnología.

La lógica difusa consta de tres etapas para obtener el resultado deseado. La primera etapa se basa en un proceso donde las variables tienen un grado de incertidumbre metalingüístico. Por lo tanto, el rango de valores (*universo de discurso*) de cada variable puede clasificarse por conjuntos difusos, por ejemplo

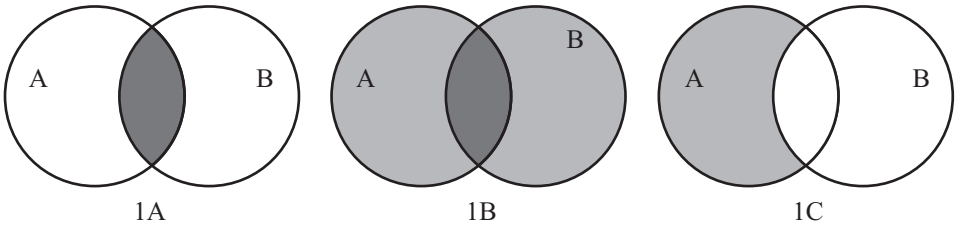


Fig. 1.1 Diagramas Venn, donde AND es unión (1A), OR es intersección (1B) y NOT complemento (1C). Son las operaciones básicas de la lógica booleana, con las cuales se puede conocer si un elemento pertenece o no a un grupo A o B; también se da el caso de que pertenezca a ambos.

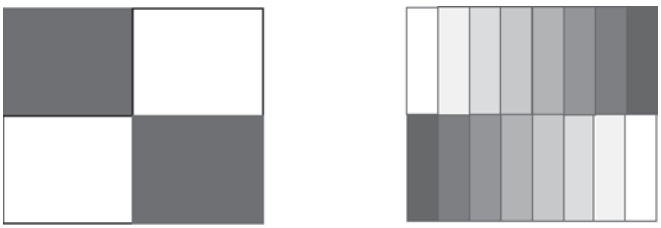


Fig. 1.2 En el primer cuadro de la izquierda (2A) se puede distinguir entre blanco y negro (0 o 1), así funciona la lógica booleana o un sistema binario. En la imagen de la derecha (2B), a excepción de los extremos es muy difícil definir lo que es negro o blanco, por ello existe una gama de respuestas (0-1); de igual forma funciona la lógica difusa, por lo cual se le tiene que dar un grado de pertenencía a los elementos, en este caso qué cantidad de color negro existe en un elemento, así como de blanco.

baja, media, alta. Cuando los sensores miden las variables, los valores pasan a un proceso de *fusificación* que consiste en pasar dichos valores a un rango de pertenencia entre cero (0) y uno (1). Se busca determinar en qué grado el valor que se está adquiriendo pertenece a un conjunto difuso. Los conjuntos difusos son caracterizados mediante funciones de membresía, las cuales están sintonizadas al punto de operación adecuado para el funcionamiento del sistema.

En la segunda etapa se proponen reglas lingüísticas (*inferencia*) que servirán de guía para que el sistema se comporte de manera más adecuada, idónea o deseada según el modelo de referencia o los objetivos del usuario. El grado de pertenencia de cada una de las variables se evalúa en un conjunto de reglas de inferencia. Dichas reglas de inferencia fueron determinadas con ayuda de un experto. El conjunto de reglas de inferencia determina una consecuencia, es decir, asigna un grado de pertenencia a un conjunto difuso que caracteriza a las salidas.

Una vez obtenidas las consecuencias, la tercera etapa es un proceso para determinar los valores óptimos de salida, conocido como *desfusificación*, y que consiste en pasar el grado de pertenencia, proveniente de la consecuencia de la regla de inferencia, a un valor nítido o real. Para hacer eso, previamente se sintonizaron funciones de membresía de cada una de las salidas con el fin de obtener un valor cuantificable. La figura 1.3 muestra el diagrama esquemático del controlador difuso.

Al final el control entregará valores nítidos o reales, consecuencia de las reglas lingüísticas previamente estructuradas, con lo cual este sistema interpretará las órdenes y realizará las acciones pertinentes.

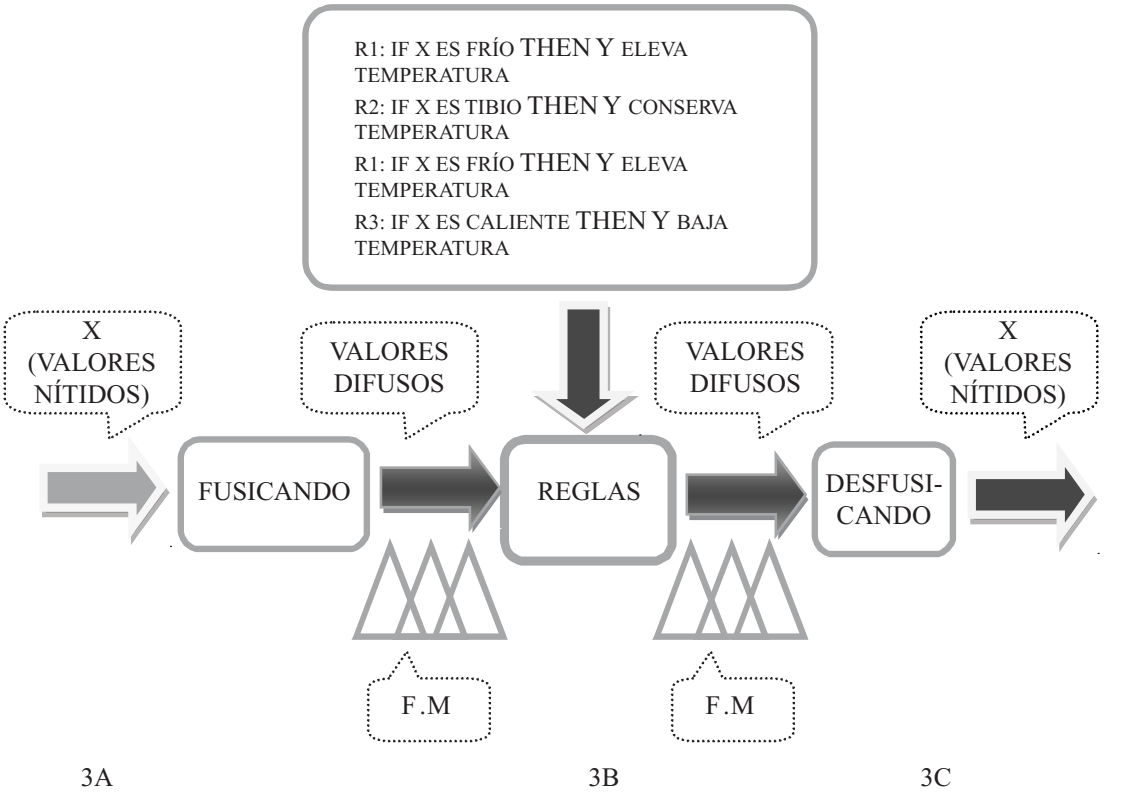


Fig. 1.3 El diagrama de bloques para desarrollar la metodología de lógica difusa muestra las tres etapas que constituyen el control. Tomando como entradas X (3A), se lleva a cabo el proceso conforme al método (3B), obteniendo una respuesta de salidas Y (3C).

REDES NEURALES ARTIFICIALES

Introducción

Como se dijo antes, la tecnología neural trata de reproducir el proceso de solución de problemas del cerebro. Así como los humanos aplican el conocimiento ganado con la experiencia a nuevos problemas o situaciones, una red neural toma como ejemplos problemas resueltos para construir un sistema que toma decisiones y realiza clasificaciones. Los problemas adecuados para la solución neural son aquellos que no tienen solución computacional precisa o que requieren algoritmos muy extensos como en el caso del reconocimiento de imágenes.

Historia de las redes neurales

Alrededor de 1943 los investigadores Warren McCulloch y Walter Pitts propusieron el primer modelo simple de la neurona. En las décadas de los cincuenta y los setenta, el movimiento en redes neurales fue liderado por B. Widrow y M. E. Hoof., quienes trabajaron con una máquina llamada *Adaline* (Adaptive Linear Element).

Otro pionero fue el psicólogo Frank Rosenblatt de la Universidad de Corell. En 1959, Rosenblatt construyó una máquina neural simple que llamó *perceptrón*. Ésta tenía una matriz con 400 fotoceldas

que se conectaban aleatoriamente a 512 unidades tipo neurona. Cuando se representaba un patrón a las unidades sensoras, éstas enviaban una señal a un banco de neuronas que indicaba la categoría del patrón. El perceptrón de Rosenblatt reconoció todas las letras del alfabeto.

Al final de los años setenta Minsky y Papert demostraron que los *perceptrones* eran incapaces de hacer tareas simples tales como sintetizar la función lógica XOR. Las matemáticas del libro *Perceptrons* eran indiscutibles y su tono dio el mensaje que los perceptrones eran un camino sin salida. Uno de los científicos que continuó trabajando durante los años oscuros de las redes neurales fue Stephen Grossberg, ahora director del Centro para Sistemas Adoptivos de la Universidad de Boston. Grossberg junto con Gail Carpenter de la Universidad de Northeastern han propuesto un modelo de red neural llamado ART (Adaptive Resonance Theory). Otros investigadores que trabajaron durante los años setenta fueron Teuvo Kohonen, de la Universidad de Helsinki, y Jim Anderson, de la Universidad de Brown, que trabajó con alternativas de semillas de conexionismo y junto con Geoff Hinto, quien presentó trabajos matemáticos y de aplicación de redes neuronales organizaron el primer encuentro neoconexionista en 1979, al cual asistieron David Rumelhart, Mc Clelland, Geoff Hinton, Anderson, Jerry Feldman y Terry Sejnowski.

En 1986 Mc Clelland y Rumelhart publicaron un libro en dos volúmenes titulado: *Parallel Distributed Processing: Explorations in the Microstruture of Cognition*. Este libro se considera un clásico en el área de redes neurales y se puede decir que su aparición significó un nuevo impulso a la investigación en sistemas neurales al mostrar las ventajas y desventajas de las redes neurales artificiales (RNA).

Algunas ventajas de las RNA frente a otros sistemas de procesamiento de información son:

- Las RNA pueden sintetizar algoritmos a través de un proceso de aprendizaje.
- Para utilizar la tecnología neural no es necesario conocer los detalles matemáticos. Sólo se requiere estar familiarizado con los datos del trabajo.
- La solución de problemas no lineales es uno de los fuertes de las RNA.
- Las RNA son robustas, pueden fallar algunos elementos de procesamiento pero la red continúa trabajando; esto es contrario a lo que sucede en programación tradicional.

Las desventajas de las redes neurales son:

- Las RNA se deben entrenar para cada problema. Además, es necesario realizar múltiples pruebas para determinar la arquitectura adecuada. El entrenamiento es largo y puede consumir varias horas de la computadora (CPU).
- Debido a que las redes se entrenan en lugar de programarlas, éstas necesitan muchos datos.
- Las RNA representan un aspecto complejo para un observador externo que desee realizar cambios. Para añadir nuevo conocimiento es necesario cambiar las iteraciones entre muchas unidades para que su efecto unificado sintetice este conocimiento. Para un problema de tamaño considerable es imposible hacer esto manualmente, por lo tanto una red con representación distribuida debe emplear algún esquema de aprendizaje.

Las redes neurales se basan en generalizar información extraída de datos experimentales, tablas bibliográficas o bases de datos, los cuales se determinan por expertos humanos. Dichas redes neurales toman en cuenta las entradas (corriente, voltaje) y como salidas las señales del sistema (velocidad, temperatura, torque). La red neural utilizada es una red multicapa de diez neuronas en la capa de entrada, diez neuronas en la capa oculta y cinco neuronas en la capa de salida. Por lo tanto, se tienen 250 pesos ajustables mediante un control retroalimentado o de lazo cerrado. En la figura 1.4 se presenta un diagrama de red neural. Los parámetros de inicialización se obtuvieron mediante un conjunto de datos experimentales y una base de datos.

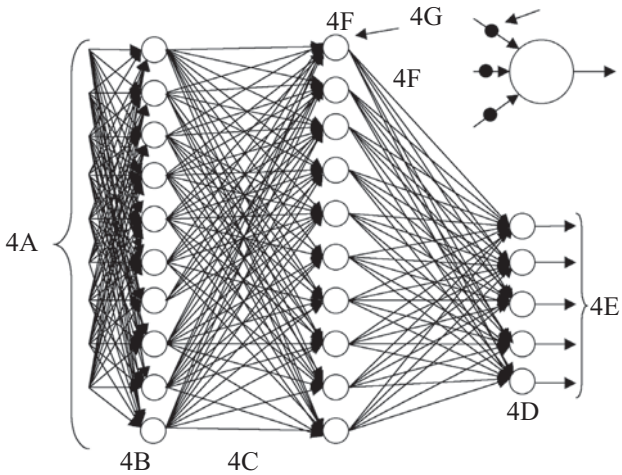


Fig. 1.4 Esquema de la red neural multicapas 10-10-5 para el controlador inteligente. Entradas (4A): Representa cualquier variable. Capa de entradas (4B), capa oculta (4C) y capa de salidas (4D). Salidas (4E): Representa también cualquier variable de interés para el usuario. Los pesos entre cada neurona (4F) están representados por un punto negro (4G).

El entrenamiento está basado en el algoritmo de "retropropagación del error" por el método del gradiente descendiente, en donde los pesos se actualizan mediante el uso de un conjunto ordenado de entradas y salidas deseadas y la comparación entre dicha salida y la salida real de la red neural. También se utiliza para el entrenamiento otra metodología alterna que es el "perceptrón". Es un clasificador de forma binaria: sólo existe la posibilidad de ser parte de un grupo A o B; funciona con sistemas lineales. Ambas tecnologías antes mencionadas se explican con más detalle a continuación.

Perceptrón

Es un tipo de red neural artificial. También puede entenderse como perceptrón la neurona artificial y unidad básica de inferencia en forma de discriminador lineal. Éste consiste en una suma de las señales de entrada, multiplicadas por unos valores de pesos escogidos inicialmente en forma aleatoria. En una fase en la que éste aprende, la entrada se compara con un patrón preestablecido para determinar la salida de la red. Si en la comparación la suma de las entradas multiplicadas por los pesos es mayor o igual que el patrón preestablecido, la salida de la red es uno (1); en caso contrario la salida es cero (0). El perceptrón es un dispositivo que, en su configuración inicial, no está en capacidad de distinguir patrones de entrada muy complejos, sin embargo mediante un proceso de aprendizaje es apto para adquirir esta capacidad. En esencia, el entrenamiento implica un proceso de refuerzo a través del cual los pesos que codifican las sinapsis se incrementan o se disminuyen. La red tipo perceptrón fue inventada por el psicólogo Frank Rosenblatt en el año de 1957. Su intención era ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, sin entrar en mayores detalles con respecto a condiciones específicas y desconocidas para organismos biológicos concretos.

En la figura 1.5 se representa una neurona "artificial", la cual intenta modelar el comportamiento de la neurona biológica. Aquí el cuerpo de la neurona se representa como un sumador lineal de los estímulos externos z_j, seguida de una función no lineal $y_j = f(z_j)$. La función $f(z_j)$ es llamada la función de activación y es la función que utiliza la suma de estímulos para determinar la actividad de salida de la neurona.

Este modelo es la base de la mayoría de las arquitecturas de las RNA que se interconectan entre sí. Las neuronas emplean funciones de activación diferentes según la aplicación. Algunas veces son funciones lineales, otras son funciones sigmoidales (por ejemplo la tanh x) y otras son funciones de umbral de

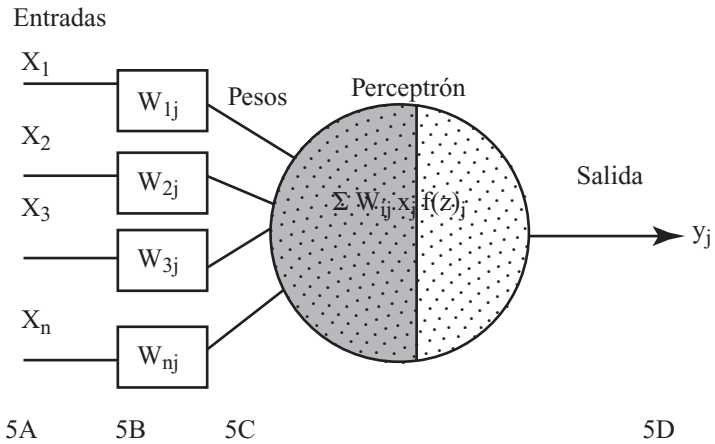


Fig. 1.5 Estructura del perceptrón, la más simple en las RNA. Es un discriminador binario lineal y puede ser entrenado para mejorar su desempeño. Las entradas de la neurona (5A), los pesos aleatorios (5B), la sumatoria de la multiplicación de los pesos por sus respectivas entradas (5C) y la salida que es el cálculo de todos los pesos y sus entradas (5D).

disparo. La eficiencia sináptica se representa por factores de peso de interconexión w_{ij}, desde la neurona *i* hasta la neurona *j*.

Los pesos pueden ser positivos (excitación) o negativos (inhibición). Los pesos junto con las funciones $f(z)$ dictan la operación de la red neural. Normalmente las funciones no se modifican de tal manera que el estado de la red neural depende del valor de los factores de peso (sinapsis) que se aplica a los estímulos de la neurona. En un perceptrón, cada entrada se multiplica por el peso *w* correspondiente y los resultados se suman, siendo evaluados contra el valor de umbral; si el resultado es mayor al mismo, el perceptrón se activa.

El perceptrón sólo es capaz de resolver funciones definidas por dos dimensiones. Un ejemplo de una función que no puede ser resuelta es el operador lógico. El entrenamiento de un perceptrón es por medio de la regla de aprendizaje delta:

Para cada peso *w* se realiza un ajuste *dw* según la regla:

$$dw = \eta(x - Y)X$$

Donde η es la razón de aprendizaje, *y* el valor deseado, *Y* el valor obtenido y *X* la entrada aplicada al perceptrón.

Redes de retropropagación (backpropagation)

Principios para entrenar una red multicapa empleando el algoritmo de retropropagación

El algoritmo Backpropagation para redes multicapa es una generalización del algoritmo de mínimos cuadrados. Ambos algoritmos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. La red Backpropagation trabaja bajo aprendizaje supervisado y por tanto necesita un conjunto de instrucciones de entrenamiento que le describa cada salida y su valor de salida esperado.

Si se considera la red de tres capas con dos entradas y una salida de la figura 1.6, es posible apreciar que cada neurona está compuesta de dos unidades, donde la primera suma los productos de las entradas por sus respectivos pesos, y la segunda unidad contiene la función de activación. La señal *e* corresponde a la salida de la suma y $y = f(e)$ es la señal de salida del elemento no lineal de la función de activación, así como la salida de la neurona.

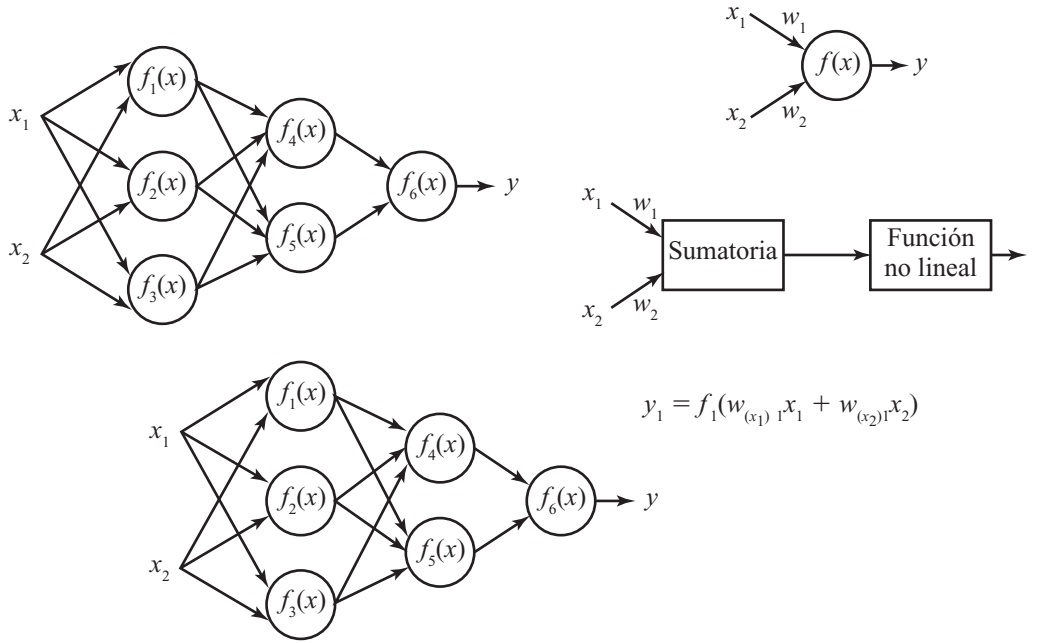


Fig. 1.6 Red de dos capas.

Para "enseñarle" a la red neural es necesario entrenar un conjunto de datos, el cual consiste en señales de entradas x_1 y x_2 asignadas con objetivos correspondientes (salidas deseadas) denominados *z*. El entrenamiento es un proceso iterativo. En cada iteración los pesos de los nodos se modifican usando nuevos datos del conjunto para el entrenamiento. Las modificaciones de los pesos se calculan empleando el algoritmo de retropropagación del error para el entrenamiento supervisado.

Cada paso del entrenamiento comienza al forzar ambas entradas de salida del conjunto de entrenamiento. Después es posible determinar los valores de salida de las señales de cada neurona en cada capa de la red. La figura 1.7 muestra dos ejemplos de cómo se propaga la señal a través de la red, donde los pesos w_{mn} corresponden a la conexión de la salida de la neurona *m* con la entrada de la neurona *n* en la capa siguiente.

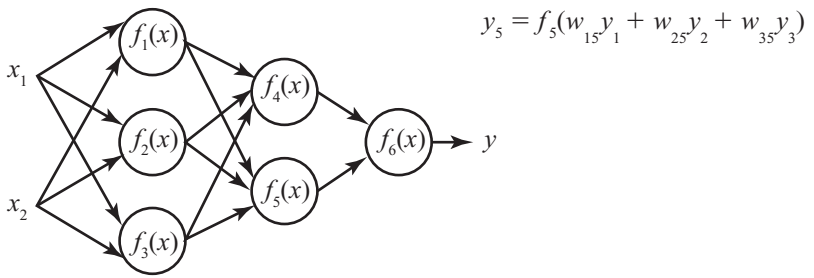


Fig. 1.7 Propagación de las señales en las neuronas.

En el siguiente paso del algoritmo, la salida de la red se compara con el valor objetivo deseado. La diferencia se denomina error de la señal ( ). Es imposible conocer el error en las neuronas de las capas

internas directamente, debido a que los valores de salida de estas neuronas son desconocidos. El algoritmo de retropropagación propaga el error de regreso a todas las neuronas, cuya salida fue la entrada de la última neurona; esto se puede apreciar en la figura 1.8.

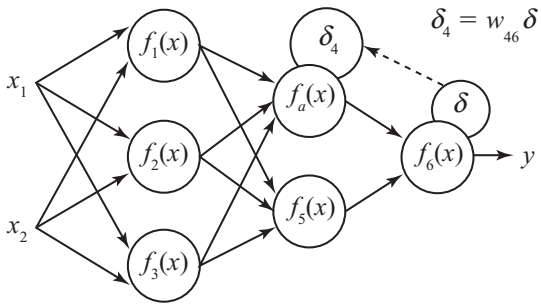


Fig. 1.8 Retropropagación del error, capa de entrada, salida y ocultas.

Posteriormente el error se va propagando a las neuronas de capas anteriores considerando los pesos de las conexiones, según se muestra en la figura 1.9.

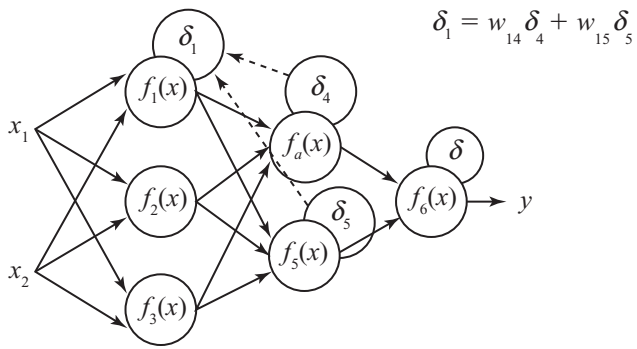


Fig. 1.9 Retropropagación del error en capas intermedias.

Cuando se calcula el error para cada neurona, los pesos de entrada pueden modificarse según los ejemplos que se presentan en la figura 1.10. Los coeficientes η afectan la velocidad de aprendizaje y pueden seleccionarse por distintos métodos. Uno de ellos implica que al inicio del proceso de entre-

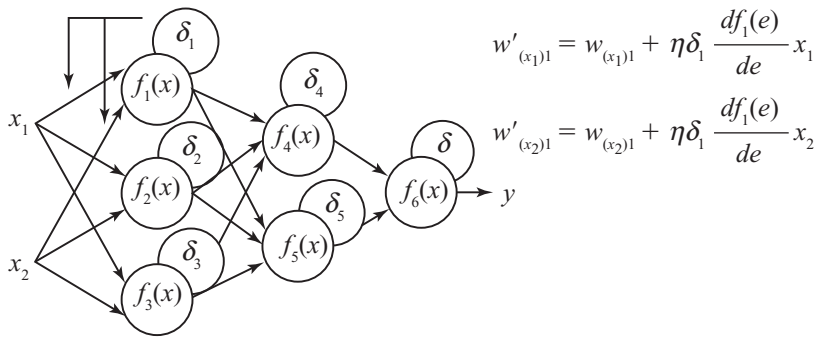


Fig. 1.10 Actualización de los pesos.

namiento se elige un valor grande, el cual va descendiendo gradualmente conforme avanza el proceso. Otro método comienza con parámetros pequeños que aumentan a medida que el proceso avanza y nuevamente disminuye en la etapa final. Comenzar el proceso con un parámetro pequeño permite el establecimiento de los signos de los pesos.

Finalmente, tanto para el controlador difuso como para la red neural artificial, las acciones de control son sencillamente pasar los datos de salida a los dispositivos que se conecten o, en su caso, plantas virtuales previamente cargadas en el Sistema Didáctico de Control Inteligente Multipropósito.

ALGORITMOS GENÉTICOS

Introducción

Un algoritmo genético (AG) es una técnica de búsqueda iterativa inspirada en los principios de selección natural. Los AG no buscan modelar la evolución biológica sino derivar estrategias de optimización. El concepto se basa en la generación de poblaciones de individuos mediante la reproducción de los padres.

Durante el curso de la evolución, los genes con evolución lenta fueron remplazados por genes con mejor estrategia evolutiva. Por lo tanto, se esperan estrategias altamente eficientes en la fauna y la flora modernas.

Muchos problemas tiene funciones objetivo complejas y la optimización tiende a finalizar en mínimos/máximos locales. La idea de los AG es optimizar (hallar el máximo o mínimo) una función objetivo utilizando los principios de la selección natural sobre los parámetros de la función.

Historia de los algoritmos genéticos

La primera idea surgió en la tesis de J. D. Bagley: "El funcionamiento de los sistemas adaptables empleando algoritmos genéticos y correlativos", en 1967. Esta tesis influyó decisivamente en J. H. Holland, quien se puede considerar como el pionero de los AG.

En rigor, los AG pueden concebirse como métodos de optimización. En general, los problemas de optimización se plantean de la siguiente manera:

$$x_0 \in X \text{ tal que } f \text{ es un máximo en } x_0, \text{ donde } f: X \to \Re, \text{ por lo tanto:}$$
$$f(x_0) = \text{máx}_{x \in X} f(x)$$

Es casi imposible obtener una solución en sentido estricto. Dependiendo del problema planteado, puede ser suficiente encontrar el máximo valor o el más cercano al valor máximo; f es una función asignada para definir el valor de "aptitud" para cada individuo (esto es, por supuesto, una gran ayuda para simplificar).

Definición. Se asumirá que S es un arreglo de cadenas (es un caso no trivial en algunos aspectos de la gramática). Proponiendo que X es el espacio deseado para la optimización, se tendrá la siguiente función:

$$e: X \to S$$
$$x \to e(x)$$

esta función se llama *función codificación*. Inversamente, la función

$$\tilde{e}: S \to X$$
$$s \to \tilde{e}(s)$$

se denomina *función de decodificación*. Estas funciones se deben especificar dependiendo de las necesidades del problema que se plantea, y no necesariamente son biyectivas. Pero es conveniente usar funciones biyectivas. Además se debe cumplir la siguiente igualdad:

$$(e \; o \; \widetilde{e}) \equiv id_s$$

Definiciones

La siguiente lista contiene diferentes expresiones utilizadas en la genética y su estructura equivalente en AG:

Evolución natural	Algoritmo genético
genotipo	código de cadena
fenotipo	punto sin codificar
cromosoma	cadena
gen	posición de cadena
alelo	valor en una posición determinada
función de aptitud o aptitud	valor de la función objetivo

Genotipo: expresión genética de un organismo o estructura genética del organismo. La información contenida en el genoma.

Fenotipo: características físicas de un organismo, atribuibles a la expresión de su fenotipo. Contiene tanto los rasgos físicos como los conductuales. Es el resultado de la interacción entre el genotipo y el ambiente; se interpreta como la suma de los caracteres observables en un individuo. Es la manifestación externa del genotipo.

Cromosoma: es la molécula única de ADN, unida a histonas (proteínas básicas) y otras proteínas que se condensa durante la mitosis (proceso de división celular- reparto equitativo del material hereditario) y la meiosis (proceso de fragmentación —divisiones pequeñas), formando una estructura compacta.

Gen: especifica la herencia de un carácter; está formado por una secuencia de aminoácidos de una o más cadenas de ARN (ácido ribonucleico: interviene en diferentes neuronas, en la expresión de la información genética), que realizan diferentes funciones en la cadena.

Alelo: el valor de un gen. Una de las dos o más formas alternativas de un gen; determina el carácter controlado por el gen. Un ejemplo es el *diploide* que contiene dos juegos de cromosomas, por lo tanto tiene dos copias de cada gen.

Función de aptitud: es un tipo especial de función que cuantifica la optimalidad de una solución. Se traduce en un cromosoma óptimo para que sus bases sean combinadas con cualquier otra técnica para la producción de una nueva generación que sea mejor a las anteriores.

En términos generales, las cadenas de los AG son análogas a los *cromosomas* en el sistema biológico. En los organismos naturales, uno o más cromosomas se combinan para formar la prescripción genética total, para la construcción y operación del organismo. En los organismos naturales el "paquete total" de genética se denomina genotipo. En los AG el "paquete total" de las cadenas se llama *estructura* (la estructura está compuesta por varias cadenas). En los organismos naturales, la creación de los organismos se realiza mediante la interacción de "paquetes" genéticos con su medio y se llama *fenotipo*.

En la terminología natural, se dice que los cromosomas están compuestos por genes, los cuales permiten tomar distintos valores llamados *alelos*. En la genética la posición del gen (llamado *locus*) se identifica en forma separada de la función del gen. Así, se puede hablar de un gene en particular, por ejemplo el gen que da color a los ojos de los animales: su lugar es la posición 10 y el valor del alelo es "ojos azules".

En los AG las cadenas están compuestas por características, que toman diferentes valores. Estas características se localizan en distintas posiciones de la cadena.

Los AG también fueron desarrollados por John Holland y sus colegas en la Universidad de Michigan (1975). Estos algoritmos se basan en la mecánica de la selección natural, la cual afirma que sólo los organismos que *mejor se adapten* sobreviven. Parte de la historia de los algoritmos se describió en los antecedentes de la IA, de manera que ahora se abordarán otros aspectos importantes para entender mejor esta parte de la ciencia.

Los componentes de un algoritmo genético son:

- Una función que se desea optimizar.
- Un grupo de candidatos para la solución.
- Una función de evaluación que mida cómo los candidatos optimizan la función.
- Función de reproducción.

El AG funciona porque hay competencia por los recursos y se hereda la mejor configuración genética en cada generación. Ciertas características de un organismo de la misma especie, en las que el éxito se manifiesta como éxito reproductivo, lo convierte en el mejor adaptado, por lo que deja más hijos que los otros. De esta manera, los rasgos hereditarios que favorecen el éxito tienden a estar más representados en la población.

Herencia

Desde tiempos inmemoriales, el hombre se ha percatado de la herencia de los rasgos físicos o psicológicos. Advirtió fácilmente que, en general, los hijos se parecen a los padres, y que esto no sólo ocurre en el hombre, sino también en los demás organismos; ya los filósofos griegos de la Antigüedad meditaban sobre estos hechos. Pero el desarrollo de la genética no se inició sino hasta el comienzo del siglo XX, época en que los científicos conocían ya numerosos detalles de la constitución de la célula. Antes, en el año de 1860, tuvieron lugar los famosos experimentos del monje austriaco Gregor Mendel, quien cruzó diversas plantas de jardín de su convento y pudo establecer algunas de las leyes más importantes de la genética. Sin embargo, la importancia de la obra de Mendel no fue reconocida hasta que los resultados obtenidos por éste no se compararon con los que el estadounidense Thomas Morgan realizó con la mosca de fruta. La genética, es decir, la ciencia de la herencia, junto con la investigación de la biología celular, ha podido establecer que los genes, portadores de los rasgos hereditarios, se encuentran en los cromosomas del núcleo celular. Incluso se ha mostrado que las cualidades dependientes de dichos genes se heredan según ciertas leyes derivadas de las posibles combinaciones de los genes de las células sexuales. Hoy se sabe que la información sobre la herencia está almacenada en una especie de "código" en los ácidos nucleicos (ADN y ARN). La moderna genética dedica gran atención al estudio de estos ácidos. Se ha demostrado que aunque sean muy estables, pueden ser transformados mediante ciertos productos químicos, rayos X, etc., de tal manera que al transmitirse a las células descendientes, provocan en éstas la aparición de nuevas cualidades. Estas modificaciones, positivas o negativas, se llaman "mutaciones" y pueden surgir también en forma espontánea.

¿Qué es herencia?

En la reproducción sexual los genes se combinan de diferentes maneras. Los cromosomas paternos se reparten entre las células sexuales, y cuando tiene lugar la fecundación surgen combinaciones de factores hereditarios diferentes de los que poseen los padres. A través de constantes combinaciones, la masa hereditaria se transmite por medio de las células sexuales, de generación en generación; podría decirse que es potencialmente inmortal.

En las plantas y animales superiores, los cromosomas de las células normales se agrupan por parejas. En cada pareja, uno de los cromosomas procede del padre y el otro de la madre. Al formarse las células sexuales, los dos cromosomas de cada pareja van a parar a células distintas, de modo que los

gametos sólo reciben la mitad de los cromosomas contenidos en una célula normal. En el momento de la fecundación, cuando los gametos se fusionan y reúnen sus cromosomas, el número de éstos vuelve a ser el de una célula normal. En los seres humanos, este numero es 46 (23 parejas).

Al juntarse las dos células sexuales, cada cromosoma se reúne con su pareja y cada gen con su homólogo del otro cromosoma. Los genes así emparejados no siempre son iguales. Uno puede ser *dominante*, impidiendo que se manifieste el otro, que se llama *recesivo*. Se debe considerar que no se heredan directamente las cualidades, sino sólo los genes que las determinan; la aparición de un rasgo depende de la interacción entre los genes y la influencia del medio. El conjunto de los genes o predisposiciones hereditarias de un individuo es su *genotipo*, mientras que el conjunto de sus caracteres externos, es decir, el resultado de la interacción del genotipo con el medio, es el *fenotipo*. Las cualidades adquiridas sólo por la influencia del medio no pueden heredarse.

El código genético

El material genético está formado por ácidos nucleicos. Éstos se hallan en todas las células vivas, en las que determinan la constitución de las proteínas y de los genes. Se distinguen dos tipos de ácidos nucleicos: ácido desoxirribonucleico (ADN) y ácido ribonucleico (ARN). Ambos están formados por fosfatos, azúcares y bases nitrogenadas. Alrededor de 1940 se consiguió demostrar que la función de los cromosomas, como portadores de los factores hereditarios, corresponde a los ácidos nucleicos. En 1962 se otorgó el premio Nobel por el descubrimiento de la estructura y función de la molécula de ADN. Ésta tiene forma de hélice doble. Las dos hélices de la molécula están unidas mediante bases nitrogenadas enlazadas por puentes de hidrógeno. Cada unión está integrada por dos bases, formando cuatro tipos de uniones cuya ordenación en la molécula constituye el código genético. Éste debe ser transmitido, con precisión, de célula a célula, en sucesivas generaciones. Dicha transmisión tiene lugar cuando, al dividirse la célula, las hélices de la molécula de ADN se separan y cada una de ellas sintetiza de nuevo su cadena complementaria hasta formar otra vez hélices dobles que luego se reparten entre las células hijas.

Las moléculas de ARN, homólogas a las del ADN, se forman en el núcleo celular. Después se trasladan a los centros de elaboración de proteínas, los ribosomas del citoplasma, donde se dirige la formación de las diferentes clases de proteínas. Hoy el "código genético" se ha descifrado. Se conoce con mucha exactitud el mecanismo por el cual los ácidos nucleicos (núcleos) lo transportan a los lugares de síntesis para producirlo en proteínas (citoplasmas). A medida que aumente el conocimiento humano sobre la materia viva, se podrá comprender mejor cómo surgió la vida en la Tierra.

Selección natural

Mientras reflexionaba acerca de la selección artificial, Darwin leyó un ensayo del economista Thomas Malthus. El ensayo sugería que de no controlarse la población humana, sobrepasaría con el tiempo a la producción de alimentos, ocasionando una fuerte competencia por la existencia. Darwin se dio cuenta de que las ideas de Malthus podían aplicarse al mundo natural. Razonó que algunos competidores en la lucha por la existencia estarían mejor equipados para sobrevivir que otros. Los menos equipados morirían. Allí estaba, finalmente, el marco de trabajo para una nueva teoría sobre el origen de las especies.

La teoría de Darwin tiene cuatro principios básicos que explican cómo pueden cambiar los rasgos de una población con el tiempo. Primero, los individuos de una población muestran diferencias o variaciones. Segundo, las variaciones pueden heredarse, lo cual significa que pasan de padre a hijo. Tercero, los organismos tienen más descendientes de los que pueden sobrevivir con los recursos disponibles. El cardenal promedio, por ejemplo, pone nueve huevos cada verano. Si cada pichón de cardenal sobrevive y se reproduce una sola vez, tomaría sólo siete años para que el primer par produjera un millón de aves. Finalmente, las variaciones que aumentan el éxito reproductivo tendrán mayor oportunidad de transmitirse, que aquellas que no lo aumentan. Si tener una cola de abanico ayuda a las palomas a reproducirse exitosamente, futuras generaciones incluirán más palomas con este tipo de cola.

Darwin llamó a su teoría selección natural. Razonó que, dado el tiempo necesario, la selección natural podría modificar una población lo suficiente como para producir nuevas especies.

Para las estructuras básicas de un algoritmo genético también es necesario conocer la transición de una generación a otra, la cual consta de cuatro elementos básicos:

Selección: mecanismo de selección individual (cadena) para la reproducción acorde con la *función de aptitud* (valor de la función objetivo). Los algoritmos de selección serán los encargados de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no. La idea básica de selección está asociada con la *función de aptitud* y el sistema original; para su implementación es comúnmente conocida como *roulette-wheel* (RWS); ésta utiliza una distribución de probabilidad, donde la probabilidad de selección de una cadena es directamente proporcional a su *aptitud*.

Cruzamiento: método de fusión sobre la información genética de dos individuos; si la codificación es elegida apropiadamente, dos progenitores saludables producirán descendientes sanos. Es el principal operador genético; provee un mecanismo para heredar características a su descendencia; interviene en ambos progenitores.

Mutación: en la evolución real, el material genético puede ser alterado en forma aleatoria debido a un error en la reproducción o la deformación de genes; un ejemplo es la radiación de los rayos gama. En los algoritmos genéticos, la mutación se realiza, con gran probabilidad, como una deformación aleatoria de las cadenas. Produce cambios incrementales al azar en la descendencia, efectuando cambios aleatorios en los valores del alelo en algunos genes. En el caso de cromosomas binarios le corresponde hacer los cambios de posiciones en cada bit. No afecta a toda la población, pero es probable que dañe a algunos. La mutación tiene el efecto de perturbar de manera segura a los cromosomas a fin de introducir nuevas características que no estaban presentes en ningún elemento de los progenitores.

Reemplazo: procedimiento para calcular (crear) una nueva generación de la anterior y sus descendientes. Se crea un espacio a la descendencia en la población eliminando de ella a los padres.

Los algoritmos genéticos trabajan con un número fijo de cadenas binarias de longitud fija. Para este fin, se asumirá que las cadenas que se va a considerar son del mismo conjunto

$$S = \{0,1\}^n;$$

Por lo tanto, la generación en el tiempo t es una lista de m cadenas donde se denotará como:

$$B_1 = (b_{1,t}, b_{2,t}, \ldots, b_{m,t}).$$

Los AG, en su mayoría, tienen la estructura siguiente:

Algoritmo
Para t: = 0
Se computa la población inicial definida por $B_0 = (b_{1,0}, b_{2,0}, \ldots, b_{m,0})$;
Mientras WHILE se alcanza la condición deseada DO
Inico
FOR i:=1 hasta m DO
Se selecciona un individuo $b_{i,t+1}$ *de* B_t;
FOR i:=1 hasta m-1 Paso 2 DO
IF Random [0, 1]$\leq pc$ *THEN*
cruza $b_{i,t+1}$ *con* $b_{i,t+1,\,t+1}$;
FOR i:=1 hasta m DO
eventualmente mutar
se crea la descendencia cruzando a los individuos;
eventualmente se mutan $b_{i,t+1}$;
se realiza un incremento t:=t+1
FIN

La selección, el cruzamiento (se realiza sólo con una probabilidad crítica de percolación p^c) y la mutación tienen ciertos grados de libertad, mientras que la operación de reemplazo ha sido especificada. Como es fácil de apreciar, todo individuo seleccionado se reemplaza por su sucesor después del cruzamiento y la mutación; los individuos no seleccionados mueren inmediatamente.

Operaciones genéticas en cadenas binarias

Selección

La *selección* es la componente que guía el algoritmo para encontrar la solución, prefiriendo dentro de un grupo de baja *función de aptitud* a los más altos. Puede ocuparse una operación determinista; en la mayoría de las implementaciones tiene componentes aleatorios. La probabilidad de escoger el individuo adecuado es directamente proporcional a su *función de aptitud*. Se puede observar como un experimento aleatorio con

$$P[b_{j,t \text{ es seleccionada}}] = \frac{f(b_{j,t})}{\sum_{k=1} f(b_{k,t})}$$

sólo funciona para los valores positivos de las *funciones de aptitudes*. Si éste no es el caso, se debe aplicar (un cambio en el caso más simple) una transformación de no-decremento $\varphi : \Re \rightarrow \Re^+$. Entonces la probabilidad se puede expresar por

$$P[b_{j,t \text{ es seleccionada}}] = \frac{\varphi(f(b_{j,t}))}{\sum_{k=1} \varphi(f(b_{k,t}))}$$

las diferentes salidas se obtendrán con diferentes probabilidades. La programación del algoritmo se puede proponer como sigue, y la configuración analógica se representa con la expresión matemática anterior.

Algoritmo
x: ≡ Random[0,1];
i: ≡ 1
Mientras $i<m \;\&\; x< \sum_{j=1}^{i} f(b_{j,t}) / \sum_{j=1}^{m} f(b_{j,t})$ *hacer*
i: i+1;
selecciona $b_{j,t}$ *;*

Este método a menudo se denomina selección proporcional, no es el mejor para la convergencia rápida pero puede funcionar en problemas de optimización básicos.

Cruzamiento

En la reproducción sexual como se desempeña en el mundo real, la materia genética de los progenitores se mezcla cuando los gametos de los progenitores se fusionan. Por lo general los cromosomas son aleatoriamente divididos y fusionados, con la consecuencia de que algunos genes de los descendientes provienen de un progenitor, mientras que otros provienen del otro progenitor. Este mecanismo se llama *cruzamiento*. Es una herramienta muy potente para introducir nuevos materiales genéticos y mantener

la diversidad genética, pero con la notable propiedad de que progenitores saludables también producen buen rendimiento en los descendientes, o incluso mejores. Varias investigaciones han llegado a la conclusión que el cruzamiento es la razón por la que las especies de reproducción sexual se adaptan más rápido que las de reproducción asexual.

Básicamente, el *cruzamiento* es el intercambio de genes entre los cromosomas de los dos progenitores. En un caso sencillo, se puede realizar este proceso cortando dos cadenas en una posición elegida al azar e intercambiarlas en sus extremos. Este proceso, que se nombra *cruzamiento de único-punto*, se aprecia en la figura 1.11.

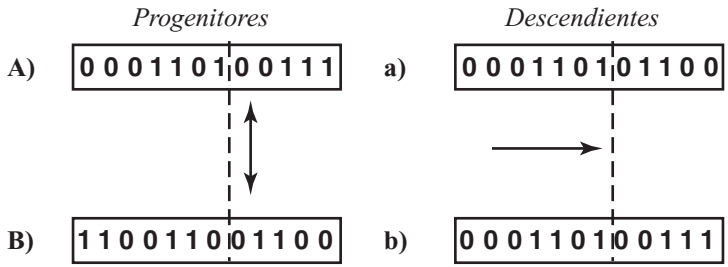


Fig. 1.11 Cruzamiento de único-punto.

A continuación se muestra la programación para el cruzamiento.

```
Algoritmo
Seleccionar de forma aleatoria
pos:=Random{1, . . . ,n-1};
FOR i:=1 hasta pos DO
Inicio
Child1[i]:= Parent1[i];
Child2[i]:= Parent2[i]
Fin
FOR i:=pos+1 TO n DO
Inicio
Child1[i]:= Parent1[i];
Child2[i]:= Parent2[i]
Fin
```

Para otros problemas o diferentes codificaciones, otros métodos de cruzamiento pueden ser útiles o incluso necesarios. Se mencionan algunos de ellos a continuación.

Cruzamiento de N-puntos: en lugar de un único punto, se eligen al azar N puntos de ruptura. Cada segunda sección se intercambia. Entre estas clases, la de *dos-puntos* es particularmente importante.

Cruzamiento segmentado: esta técnica es muy parecida al cruzamiento de N-puntos, con la diferencia de que el número de puntos de ruptura puede variar.

Cruzamiento uniforme: para cada posición, se decide al azar si se intercambian las posiciones.

Cruzamiento aleatorio: primero se escoge una permutación aleatoria que se aplicará a los progenitores, después el cruzamiento de N-puntos se aplica a los progenitores aleatorios, y finalmente los descendientes aleatorios son transformados de nuevo con permutación inversa.

Mutación

El último ingrediente de los AG es la *mutación*, esto es la deformación aleatoria de la información genética en un individuo como las radiaciones radioactivas u otros medios de influencia. En la reproducción real, la probabilidad de que determinado gen sea mutado es casi igual para todos los genes. Así, está al alcance de la mano usar las siguientes técnicas de mutación para una determinada cadena binaria *s*, donde p_M es la probabilidad de que un solo gen sea modificado:

```
Algoritmo
FOR i:=1 TO n DO
IF Random [0,1]< p_M THEN
invert s[i];
```

Por supuesto que se pueden encontrar muchas alternativas y con más detalles. Algunas de estas técnicas se muestran a continuación.

Inversión de un solo bit: la probabilidad de mutación p_M de que un bit elegido al azar sea negado.

Inversión por fragmentos: toda la cadena es invertida bit a bit con una probabilidad de mutación p_M.

Selección aleatoria: la probabilidad de mutación p_M de que una cadena elegida al azar sea remplazada.

RESUMEN

Si se desea emplear los métodos descritos, podemos escribir un algoritmo genético universal para la solución de problemas de optimización en el espacio $S = \{0,1\}^n$.

```
Algoritmo
t:=0
Se crea la población inicial B_0 = (b_1,0, b_2,0, . . . , b_m,0);
WHILE se detiene la condición que no se cumple DO
BEGIN
(* selección proporcional *)
FOR i:=1 TO m DO
BEGIN
x:= Random [0, 1];
k:=1;
WHILE k<m & x< ∑_j=1^i f(b_j,t)/∑_j=1^m f(b_j,t) DO
k: k+1;
b_i,t+1 = b_k,t
Fin
(* único-punto de cruzamiento *)
FOR i:=1 TO m-1 STEP 2 DO
BEGIN
```

```
IF Random [0, 1]=<pc THEN
BEGIN
pos:=Random{1,...,n-1};
FOR k:=pos+1 TO n DO
BEGIN
aux:= b_{i,t+1} [k];
b_{i,t+1} [k]:= b_{i,t+1, t+1} [k];
b_{i,t+1, t+1} [k]:=aux
Fin
Fin
Fin
( * mutación* )
FOR i:=1 TO m DO
FOR k:=1 TO n DO
IF Random [0,1]< p_M THEN
invertb_{i,t+1} [k];
t:=t+1
Fin
```

EJEMPLOS

Encontrar el máximo de la función siguiente:

$$f_1 : \{0, \ldots, 31\} \to \Re$$
$$x \to x^2$$

Para especificar una cadena con el espacio a lo largo de un sistema de codificación y decodificación. En este ejemplo está al alcance de la mano considerar $S = \{0,1\}^5$, donde el valor de $\{0, \ldots, 31\}$ se codifica por su representación binaria. De manera correspondiente, una cadena se decodifica como

$$\tilde{c}(s) = \sum_{i=0}^{4} s[4 - i] \cdot 2^{i}.$$

Con una población de $m = 4$, una probabilidad de cruzamiento $pc = 1$ y una probabilidad de mutación $p_M = 0.001$, la primera generación al azar con distribución uniforme de $\{0,1\}^5$, se obtendrá lo siguiente en el primer paso:

Núm. del individuo	cadena (genotipo)	valor x (fenotipo)	$f(x) = x^2$	P seleccionada$_i$ $f_i/\sum f_i$
1	01101	13	169	0.14
2	11000	24	576	0.49
3	01000	8	64	0.06
4	10011	19	361	0.31

La suma de la *función de aptitud* es 1170, donde el promedio es 293 y el máximo es 576. Se puede apreciar que en las últimas columnas la selección proporcional favorece a los de *alta-aptitud* (como el núm. 2) sobre los de *baja-aptitud* (como el núm. 3).

Realizando un experimento aleatorio se ha dejado morir al individuo núm. 3 y seleccionando doble vez al núm. 2, con lo que se obtuvo una segunda generación como sigue:

Conjunto de individuos seleccionados	Lugar de cruzamiento (aleatorio)		Nueva población	valor x	$f(x)\ x^2$
0110|1	(1)	4	01101	12	144
1100|0	(2)	4	11001	25	625
11|000	(2)	2	11011	27	729
10|011	(4)	2	10000	16	256

APLICACIONES

Existen diversas aplicaciones en la industria; a continuación se mencionan varias de ellas para que el lector tenga un panorama más amplio de todas las técnicas mencionadas y aprecie la potencia que desempeñan como herramienta en nuestra realidad. Estos ejemplos se tomaron de diferentes fuentes y la descripción de cada proyecto se explica de manera más extensa y detallada.

1. *Desentrelazado de señales de video con lógica difusa.*
2. *Marcadores anatómicos de los ventrículos del corazón.*
3. *Segmentación de imágenes cerebrales de resonancia magnética basadas en redes neuronales.*
4. *Optimización de sistemas para tratamiento de agua (Austria).*
5. *Monitoreo de glaucoma a través de redes neuronales.*
6. *Algoritmos genéticos para el diseño de sistemas de MRI (magnetic resonance imaging).*

1. Desentrelazado de señales de video con lógica difusa

La eficacia de la lógica difusa para manejar la ambigüedad e imprecisión que aparece en numerosos problemas ha motivado en los últimos años una creciente aplicación de dichas técnicas al procesado de imágenes. Muchos de los trabajos realizados se han centrado en dos aplicaciones que son objeto de una gran demanda en la actualidad: el desentrelazado de señales de video y el incremento de resolución de imágenes. La conversión de señales de video entrelazadas a progresivas se requiere en numerosos dispositivos como proyectores, DVD's, televisores de alta definición o monitores LCD, para adecuar su formato de presentación al formato de transmisión entrelazado empleado por los sistemas convencionales de televisión.

Esto ha fomentado el desarrollo de numerosos algoritmos de desentrelazado que implementan desde simples esquemas de interpolación espacio-temporal, hasta complejas técnicas de compensación de movimiento. Por otro lado, la ampliación de tamaño de imágenes es especialmente necesaria en aplicaciones de análisis e interpretación donde un aumento de la resolución en ciertas áreas de la imagen puede resultar crucial. En este trabajo se proponen nuevas técnicas de interpolación basadas en lógica difusa que proporcionan soluciones eficaces para las dos aplicaciones anteriores, sin que ello implique un incremento excesivo en su costo.

Procedimiento

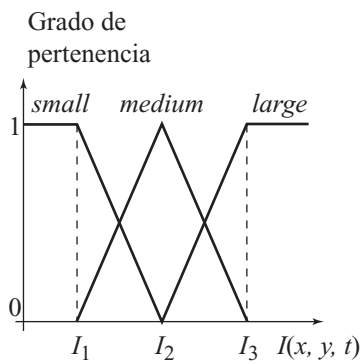
El formato entrelazado empleado por los actuales sistemas de transmisión de señales de televisión permite reducir a la mitad el ancho de banda de la señal transmitida sin afectar significativamente la calidad de las imágenes. Por este motivo, dicho formato sigue empleándose a pesar de que muchos de los dispositivos actuales realizan un barrido de presentación progresivo que requiere disponer del número total de líneas de la imagen. Los algoritmos de desentrelazado aplican diferentes técnicas de interpolación para generar la imagen completa a partir de la información transmitida en los sucesivos campos. De entre ellos, los más eficientes son aquellos que adaptan la estrategia de interpolación a las características específicas de la imagen, como el grado de movimiento o la presencia de bordes.

La detección del grado de movimiento permite dirigir y optimizar el proceso de desentrelazado. Si no existe movimiento, las líneas no transmitidas en un campo pueden obtenerse a partir de la información del campo anterior mediante lo que se llama "una técnica de interpolación temporal". Por el contrario, cuando existe movimiento, la información de las líneas del campo anterior no es fiable, siendo preferible aplicar una interpolación espacial entre los píxeles del campo actual. La idea básica de los algoritmos adaptativos al grado de movimiento fue propuesta en y puede expresarse matemáticamente mediante la expresión:

$$I_0(x, y, t) = (1 - \gamma(x, y, t)) \cdot I_T(x, y, t) + \gamma(x, y, t) \cdot Is(x, y, t)$$

donde $I_o(x, y, t)$ representa la luminancia de un determinado píxel, las variables x e y son las coordenadas espaciales del píxel en el fotograma, t indica el número de orden del fotograma en la secuencia, I_S e I_T representan los valores de luminancia obtenidos al utilizar técnicas de interpolación espacial y temporal, respectivamente, y γ indica el grado de movimiento mediante un valor comprendido entre 0 y 1.

Los trabajos más recientes se centran en determinar el valor de γ para que sea lo suficientemente robusto y, por tanto, no se produzcan detecciones erróneas de movimiento. Este trabajo consiste en utilizar un sistema basado en lógica difusa para estimar el grado de movimiento. La entrada al sistema (I) es la diferencia bidimensional de valores de luminancia entre dos campos con el mismo orden de paridad. Dicha variable de entrada se evalúa utilizando el conjunto de funciones de pertenencia triangulares que se muestra en la grafica 1.1. La tabla 1.1 tiene la base de reglas del sistema difuso. Su interpretación es: cuando el grado de movimiento es pequeño se realizará una interpolación temporal, y cuando es grande, una interpolación espacial; en cualquier otro caso se llevará a cabo una combinación de ambas. Los parámetros (I_1, I_2, I_3) que definen las funciones de pertenencia de la figura 1, así como los factores (α_1, α_2) que determinan la combinación entre I_S e I_T, se ajustaron mediante técnicas de aprendizaje supervisado.



Gráfica 1.1 Funciones de pertenencia utilizadas por el sistema difuso para realizar el desentrelazado de video con el grado de movimiento.

Tabla 1.1 Base de reglas del sistema difuso

SI	Antecedente	ENTONCES	CONSECUENTE
1)	$I(x, y, t)$ es pequeña	---	$I_T(x, y, t)$
2)	$I(x, y, t)$ es mediana	---	$\alpha_1 I_T(x, y, t) + \alpha_2 I_S(x, y, t)$
3)	$I(x, y, t)$ es grande	---	$I_S(x, y, t)$

Para la implementación del algoritmo se seleccionó como método de interpolación temporal (I_T) la inserción del píxel del campo anterior con las mismas coordenadas espaciales. Para realizar la interpolación espacial (I_S) se propone un algoritmo que utiliza un sistema basado en lógica difusa para detectar de manera robusta la presencia de bordes en la imagen. Las entradas del sistema son las diferencias en valor absoluto de la luminancia en las tres direcciones (a, b, c) que se muestran en la figura 1.1. La tabla 1.2 tiene la base de reglas del sistema. Las dos primeras reglas se activan cuando la correlación es grande en una dirección y al mismo tiempo es pequeña en las direcciones contrarias. En ambos casos el resultado interpolado se obtiene como valor medio de la luminancia en la dirección donde se encuentra el borde. La tercera regla describe una situación en la que no existe un borde porque la correlación es muy grande en dos direcciones al mismo tiempo. En este caso el resultado se obtiene interpolando los cuatro píxeles de las direcciones *a* y *c*. En cualquier otro caso el valor más adecuado se obtiene interpolando en la dirección vertical. Las etiquetas de la base de reglas (*small, strongly small* and *large*) se describen mediante las funciones de pertenencia de la gráfica 1.2. Los parámetros que las definen (*a1, a2, a3*) también se ajustaron mediante técnicas de aprendizaje supervisado hechas con anterioridad.

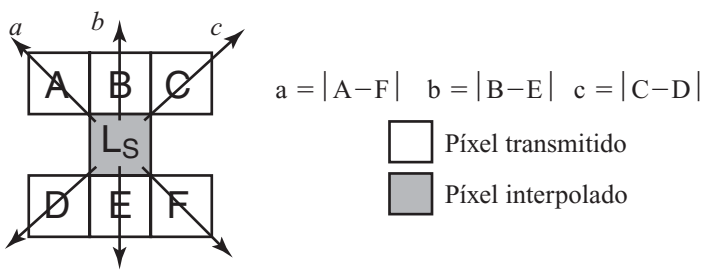
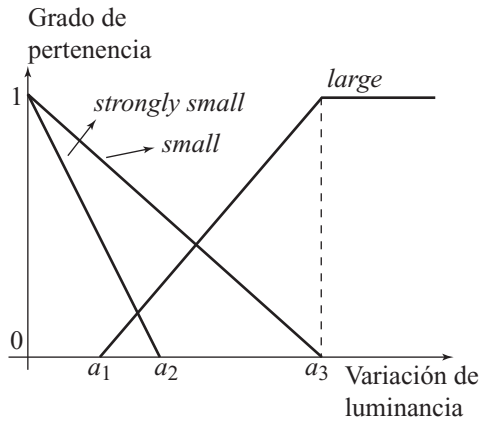


Fig. 1.1 Píxeles utilizados por el sistema difuso para implementar la interpolación espacial adaptativa con los bordes de la imagen.

Tabla 1.2 Base de reglas del sistema difuso

SI	Antecedente	ENTONCES	CONSECUENTE
1)	A es pequeña, b y c son grandes	---	$(A + F)/2$
2)	A y b son grandes y c es pequeña	---	$(C + D)/2$
3)	A es muy pequeña, b grande y c muy pequeña	---	$(A + F + C + D)/2$
4)	Otros	---	$(B + E)/2$



Gráfica 1.2 Funciones de pertenencia utilizadas.

Las dos aplicaciones descritas se han probado utilizando una amplia batería de imágenes y secuencias estándares de video. Para obtener los ficheros de entrenamiento se ha partido de imágenes en formato progresivo. Los píxeles eliminados son interpolados aplicando no sólo las técnicas descritas en los apartados anteriores, sino también técnicas convencionales. La tabla 1.3 muestra los errores obtenidos al comparar las imágenes interpoladas con las originales. Para la aplicación de desentrelazado se muestran los valores medios obtenidos tras procesar varios fotogramas de las secuencias. Se observa que los errores más pequeños corresponden en todos los casos al método propuesto.

El estudio realizado se ha extendido también al análisis de distintas opciones de implementación de los algoritmos, como por ejemplo observar la eficacia de las técnicas basadas en lógica difusa cuando las imágenes contienen ruido.

Tabla 1.3 Valores medios de los errores obtenidos al procesar distintas imágenes de varias secuencias de video.

Secuencia Formato	Missa CIF	París CIF	Trevor CIF	Salesman CIF	News QCIF	Mother QCIF	Carphone QCIF
Repetición línea	14.75	283.19	51.05	68.87	197.27	42.86	97.29
V, medio líneas	5.83	139.98	20.37	28.84	77.28	16.56	36.56
Píxel anterior	9.48	67.15	23.82	15.706	31.62	15.81	60.12
VT 2 fields [2]	6.13	54.96	14.19	14.42	18.49	7.11	25.41
VT 3 fields [2]	5.76	47.43	12.504	13.12	17.62	2.301	22.86
ELA [2]3+3	7.31	182.01	25.23	40.01	141.27	18.79	45.92
ELA [2]5+5	9.05	223.39	30.34	62.52	166.37	24.72	35.32
Técnica [4]	6.48	31.701	18.83	11.24	21.88	7.31	38.55
Técnica [7]	6.23	19.27	13.93	9.64	11.53	4.22	21.63
Propuesta	5.42	16.907	11.35	8.87	8.51	3.92	20.75

Conclusiones

Como se observó en la última tabla, el método con base en lógica difusa arroja mejores resultados para desentrelazado de video que las técnicas hasta ahora empleadas. Este trabajo es importante por la constante innovación en tecnologías de video y búsqueda de mejores resoluciones. Con este proyecto se pretende estar a la par de este crecimiento e implementarlo en las nuevas tecnologías.

Ahora bien, ¿por qué se utilizó lógica difusa? Esto se debe en gran medida a que se establecieron varias técnicas de interpolación de video que mostraron funciones de pertenencia entre los distintos parámetros (ej.: luminescencia). Los sistemas de lógica difusa pueden establecer de manera adecuada la interpolación a través de las funciones de membresía. Ahora bien, este proyecto también se pudo haber implementado usando redes neurales y/o algoritmos genéticos. No obstante, recordemos que ambas requieren de periodos de entrenamiento y adaptación, lo que podría atrasar la implementación de la solución. Por otra parte, podría ser más complicado elaborar el diseño de una red neural para la evaluación de interpolación de los píxeles. Los algoritmos genéticos podrían crear una población con las distintas soluciones que se muestran en la tabla 1.3 y hacer un entrecruzamiento para optimizar la solución.

2. Marcadores anatómicos de los ventrículos del corazón

La ventriculografía es una técnica utilizada para visualizar las cavidades cardiacas. Su objetivo principal es definir el tamaño y la forma del ventrículo izquierdo, así como también visualizar la forma y la movilidad de estructuras asociadas con las válvulas del corazón. Extraer la forma ventricular ha sido uno de los principales problemas que se encuentran al aplicar las técnicas de procesamiento digital de imágenes a imágenes de las cavidades cardiacas.

Aunque en la actualidad se desarrollan técnicas para detectar automáticamente los contornos ventriculares, no se ha encontrado aún ningún método capaz de resolver este problema de manera satisfactoria; además las técnicas existentes no han sido validadas clínicamente.

Muchos de los métodos o técnicas empleados para la obtención de los contornos ventriculares establecen dos etapas o fases de trabajo. Una primera fase o etapa previa a la detección, mediante la cual se define un conjunto de puntos que pueden utilizarse para construir un contorno inicial, y una segunda fase que busca optimizar el contorno obtenido en la etapa anterior. Muchos autores coinciden en que la solución del problema de generación automática de un contorno aproximado debe estar relacionada con la aplicación de técnicas basadas en inteligencia artificial [1]. En tal sentido, en el presente trabajo se propone el uso de las redes neurales para identificar los marcadores anatómicos necesarios para establecer una representación de un contorno inicial sobre imágenes angiográficas del ventrículo izquierdo.

En este trabajo se establece un conjunto de consideraciones tanto teóricas como prácticas que permiten realizar el diseño y la verificación del comportamiento de redes neurales, para la detección de tres marcadores anatómicos del corazón. A partir de imágenes angiográficas se construyó un conjunto de variables discriminantes compuestas por matrices de tamaño 31×31 tanto de las zonas de interés del ventrículo izquierdo como de otras regiones anatómicas. Una vez obtenidas las mencionadas matrices, se configuraron las bases de datos que constituyen el conjunto de patrones de entrada para las fases de entrenamiento, validación y prueba de las redes neurales, las cuales son sometidas posteriormente a procesos de simulación hasta obtener resultados aceptables.

Procedimiento

Las variables discriminantes se obtienen a partir de secuencias de imágenes dinámicas del corazón. Usando esta secuencia de imágenes cardiacas, se aplicó un proceso manual para extraer un conjunto de subimágenes de tamaño 31×31, con tentativas de los marcadores anatómicos más importantes al momento de caracterizar el ventrículo izquierdo.

Luego se procedió a generar las bases de datos necesarias para cada red neural, se configuró una matriz **P**, de 180 vectores en relación 1:4, es decir, por cada vector ($\mathbf{a}_i$) que representa un marcador anatómico, se introducen cuatro vectores (**Ii**) relativos a no marcadores; esto es, la estructura parcial de **P** es

la siguiente: $\mathbf{P} = [a_1\ I_1\ I_2\ I_3\ I_4\ a_2\ I_5\ I_6\ I_7\ I_8\ a_3\ I_9\ I_{10}\ I_{11}\ I_{12}. . .]$. Con el objeto de indicarle a la red cuáles vectores de **P** constituyen no marcadores y cuáles representan ápex, se construyó una matriz de 180 etiquetas **t**, respetando la misma relación establecida para **P**, asignándole el valor de "-1" a los vectores ápex y "1" a los vectores no ápex; por tanto la estructura parcial de t es: **t** = [-1 1 1 1 1 -1 1 1 1 1 -1 1 1 1 1. . .]

Es importante señalar que para clasificar a un vector como no marcador, se consideró el grado de diferencia que existe entre el mencionado vector —establecido mediante histogramas— y los que representan marcadores. Las matrices **P** y **t** constituyen los patrones de entrenamiento de nuestra red neuronal.

Ahora bien, las redes neuronales comúnmente se pueden configurar para representar funciones o reconocer patrones. La ventaja de la segunda configuración es que el tiempo de entrenamiento se reduce considerablemente en relación con otros algoritmos y además se garantiza una buena velocidad de convergencia, por lo que ésta se eligió para este trabajo.

Posteriormente se efectuó la fase de *entrenamiento*, que consiste en presentar a la entrada de la red diseñada el conjunto de patrones de entrenamiento mencionado anteriormente. Después de ello se identificó cada uno de los vectores que reconocía dicha red como ápex, detectando que la misma se estaba equivocando en un gran número de imágenes que eran no ápex. Luego se aplicó el proceso de *boostrapping*, que consiste en presentarle a la red los vectores no marcadores que erróneamente estaba identificando como marcadores. Debido a esto se generó una nueva red que reflejó un mejor comportamiento que la anterior. Agotado el proceso de boostrapping, se obtuvo redes mejor entrenadas pero que identificaban un número exagerado de vectores para cada zona de interés, entonces se procedió a modificar el número de neuronas y se aplicó el proceso de *adaptación*, donde de manera heurística se le presentó a las redes mejoradas un mismo conjunto de patrones de entrada hasta obtener una respuesta satisfactoria en la salida de cada red, en la que no se apreciaron problemas de sobreentrenamiento. En la diagrama 1.1 se muestra el proceso descrito.

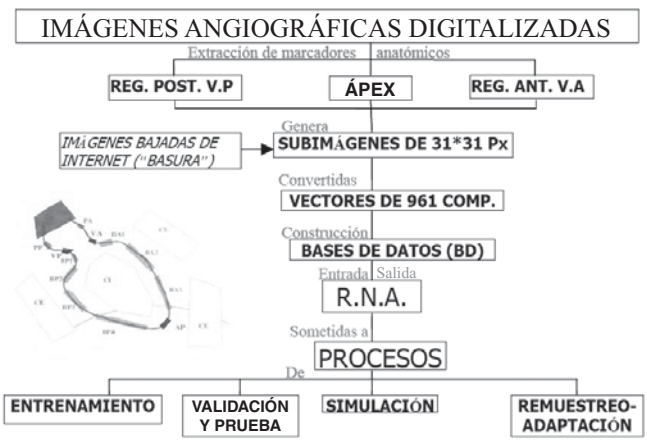


Diagrama 1.1 Proceso seguido durante el desarrollo de la red neural.

Resultados

En las primeras pruebas hechas a las redes neuronales implementadas para distinguir los ventrículos del corazón, de un total de 21 imágenes muestra, las redes desempeñaron un trabajo satisfactorio en 15 de ellas dando un rendimiento de 70%. Se pretende mejorar el diseño de estas redes aumentando el número de marcadores anatómicos o de unidades neuronales para poder utilizar este software confiablemente. En la figura 1.12 se muestra una imagen en la cual se identificó el ventrículo izquierdo; el software da las coordenadas del centroide de las regiones cuadradas.

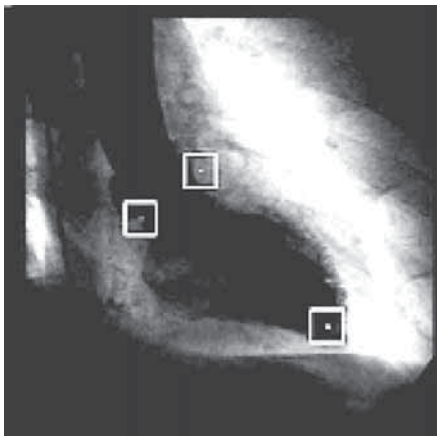


Fig. 1.11 Imagen del corazón, que resalta los marcadores anatómicos.

Conclusiones

Las redes neuronales generadas identificaron los marcadores anatómicos primordiales para la detección automática de los contornos ventriculares del corazón: la región del ápex y las regiones anterior y posterior a la válvula aórtica. Se verifica que tales patrones pueden ser identificados sin la necesidad de utilizar técnicas de preprocesamiento de los patrones de entrenamiento de cada red neuronal.

Los resultados obtenidos permiten la construcción de aproximaciones iniciales de los contornos ventriculares, los cuales pueden ser optimizados usando alguna técnica de detección de contornos, como por ejemplo un modelo de cuerpos deformables.

¿Por qué se usaron redes neuronales? Comúnmente se utilizan las redes neuronales en situaciones para las cuales no existe alguna clase de modelo o aproximación matemática, o cierto proceso iterativo. Más bien, éstas se usan cuando el problema se resuelve por observación continua y se entrena dicha red para identificar lo que se busca, y debido a la inexistencia de cierta relación entre parámetros y/o funciones que modelen el problema, que sería inapropiado el uso de lógica difusa o algoritmos genético en este trabajo.

3. Segmentación de imágenes cerebrales de resonancia magnética basada en redes neuronales

El estudio a través de imágenes de los cambios estructurales del cerebro puede proveer información útil para el diagnóstico y el manejo clínico de pacientes con demencia. Las imágenes de resonancia magnética (R.M.) pueden mostrar anormalidades que no son visibles en la tomografía computarizada. Asimismo tienen el potencial de detectar señales de anormalidad, lo que permite un diagnóstico diferencial entre la enfermedad de Alzheimer y la demencia vascular.

Se ha desarrollado numerosas técnicas con el propósito de obtener una segmentación a partir de las imágenes multiespectrales de R.M. El uso de técnicas de patrones para segmentar conjuntos de datos en imágenes de R.M. se ha desarrollado ampliamente. La mayoría son computacionalmente costosas, pero sobre todo lo más difícil es encontrar simultáneamente automatización y exactitud en los resultados. Aquellas técnicas que segmentan con mayor exactitud son semiautomáticas, es decir son operador dependientes.

En este trabajo se presenta un método de segmentación de imágenes de R.M. cerebrales basada en la utilización de redes neurales utilizando algoritmos genéticos para el ajuste de los parámetros. Esta propuesta permite detectar y cuantificar los diferentes tejidos cerebrales de una manera automática, más

sensible y esencialmente menos subjetiva y con ello dar un primer paso hacia el diagnóstico temprano de la atrofia cerebral.

Procedimiento

El método propuesto presenta los siguientes pasos:

Etapa de entrenamiento

- Paso 1: El operador selecciona entre 3 y 5 puntos de cada sustancia que se va a segmentar: líquido cefalorraquídeo (LCR), sustancia gris y sustancia blanca, de sólo una imagen de muestra, representativa del tipo de imágenes que se desearán segmentar.
- Paso 2: La red neural se construye y al mismo tiempo se entrena a partir de los valores de gris de los píxeles seleccionados en las diferentes imágenes (datos de entrenamiento, entrada), indicándosele los valores deseados en la salida, diferentes para cada sustancia.

El valor óptimo del parámetro spread que se utiliza para entrenar la red (que corresponde a la dispersión de las gaussianas presentes a la red en su capa de entrada) se calcula mediante AG.

Se le indicó al AG un tamaño de población de 50 individuos. Se generó una población inicial aleatoria que se modificó en las sucesivas iteraciones con base en la función de evaluación. En esta aplicación dicha función se determinó a través del error calculado comparando la imagen clasificada obtenida como salida de la red y la imagen resultado previamente clasificada tomada como referencia. Se determinó la cantidad de píxeles clasificados erróneamente, siendo éste el valor que se intenta reducir.

En estos dos pasos queda el sistema preparado para segmentar nuevas imágenes.

Etapa de consulta

- Paso 3: Se consulta a la red ya entrenada indicando como patrones de entrada a los niveles de gris de un nuevo conjunto de imágenes. Se obtendrá así la clasificación para cada píxel de este nuevo caso.
- Paso 4: Se crea una nueva imagen con intensidades diferentes para cada píxel según la clase de sustancia a la que ha sido asignado.
- Paso 5: Se visualizan las estructuras segmentadas con diferentes colores.

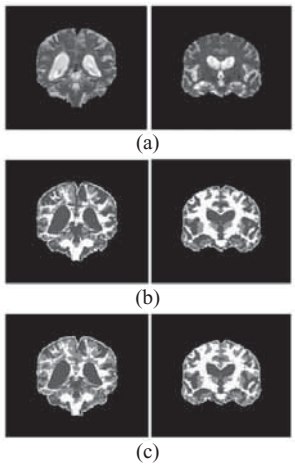


Fig. 1.13 a) Imágenes originales tomadas de R.M. b) Imágenes clasificadas con el software BRAINS; son imágenes de referencia. c) Imágenes clasificadas con la red neuronal.

Resultados y conclusiones

Este trabajo presenta un método de segmentación automática de imágenes de R.M. cerebrales. El método es preciso y eficiente, siendo los resultados obtenidos prácticamente independientes del experto.

La red neuronal involucrada en el proceso se construye y entrena a partir de una única imagen, lo cual permite clasificar otras imágenes tomadas con el mismo protocolo con un error menor que el 1% y con un bajísimo tiempo de cálculo, característica propia de la etapa de consulta del tipo de redes neuronales utilizadas (de regresión).

Del éxito de la segmentación dependen la posterior cuantificación de la materia gris, blanca y LCR, y por lo tanto la medición de la evolución de la atrofia cerebral.

¿Por qué se usó este tipo de inteligencia artificial? En esta aplicación podemos observar que se utilizó tanto las redes neurales como algoritmos genéticos. Esto es porque, como se sabe, las redes neurales son más fáciles de implementar para situaciones que sólo se resuelven mediante observación y comparación, y que no pueden ser modeladas matemáticamente. No obstante se incluyeron algoritmos genéticos para mejorar el diseño de la red neural (mas no para analizar las imágenes) de acuerdo con su rendimiento. Ahora bien, en este caso no se aplicó lógica difusa porque como se dijo, no se conoce una relación específica o modelo entre los parámetros analizados, y éstos a su vez se utilizan más dentro del control de procesos.

Referencias

1. Abras, G., V. Ballarin y M. González, "Aplicación de reconocimiento de patrones en la clasificación de tejido cerebral," *Actas del Congreso Argentino de Bioingeniería. Tafí del Valle.* Septiembre de 2008.

4. Optimización de sistemas para tratamiento de agua (Austria, lógica difusa)

Este caso trata de una solución lógica difusa en la producción de bioquímicos en la industria más grande de producción de penicilina en Austria. Después de extraer la penicilina a partir de los microorganismos que la generan, un sistema de tratamiento de aguas residuales procesa las biomasas sobrantes. Ahora bien, los lodos fermentados que se obtienen en el curso de este tratamiento contienen microorganismos y restos de sales nutrientes, los cuales son el material base para fertilizantes de alta calidad que se venden como coproductos de la producción de penicilina.

Para hacer el abono, el lodo fermentado se decanta y el agua sobrante se vaporiza. El objetivo es reducir los costos del proceso de vaporización, por lo que el proceso de decantación del agua y las sustancias para fertilizante deben ser optimizados. Así, se implementó una solución utilizando lógica difusa para dejar de operar manualmente este proceso.

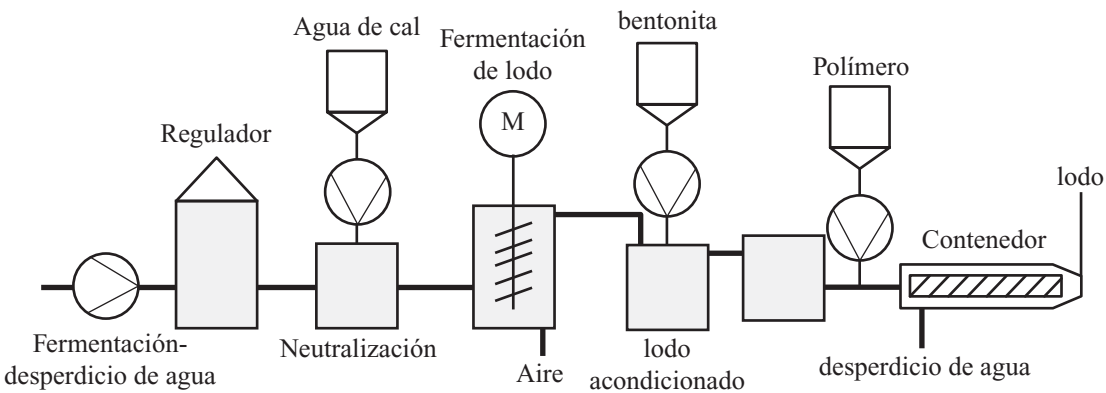


Diagrama 1.2 Proceso de separación de la biomasa del agua para producción de fertilizantes.

Con el objeto de reducir los costos del proceso antes mencionado, se debe considerar (ver diagrama 1.2):

- Reducir el uso de químicos precipitantes (para separar la mezcla del agua más fácil) debido a que el polímero usado es muy costoso.
- Extraer el máximo de agua al momento de decantarla ya que la vaporización del agua restante consume grandes cantidades de energía.
- El contenido de biomasa en la decantación debe ser de 0.7g/l, si excede los 1.5g/l se reduce el rendimiento del sistema y hasta puede detenerlo debido a la limitada capacidad del equipo para procesar masa biológica.

Este último punto es crítico y es difícil para los operadores ajustarlo manualmente durante todo el tiempo del funcionamiento del equipo. A su vez, la determinación de un modelo matemático podría ayudar a mejorar el proceso; sin embargo, debido a la gran cantidad de variables esto requiere gran tiempo y esfuerzo. Es por ello que se usa lógica difusa para automatizar el proceso sin grandes esfuerzos; además de que al ser un proceso cuyas variables cambian constantemente, se necesita una regulación continua y que se ajuste a cualquier cambio (más o menos polímero de acuerdo con la biomasa entrante).

Ahora bien, ¿por qué no se usaron redes neuronales o algoritmos genéticos? Tanto las redes neuronales como los AG requieren de "entrenamiento" o periodos de prueba para llegar a una solución óptima y por lo general se aplican a problemas con muchas más variables o con un gran flujo de información; es decir, se preocupan más por cómo se adaptarán las soluciones al problema y no tanto por la medición de las variables. En este caso, la medición de las sustancias como el polímero y la biomasa, así como la regulación de la potencia del vaporizador lo son todo, porque el objetivo es aumentar el rendimiento y reducir los costos; el proceso de "entrenamiento" resultaría muy costoso.

- [Recurso Electrónico]. Recuperado el 18 de agosto de 2009 de: http://www.fuzzytech.com/
- Von Altrock, C., B. Krause y H. J. Zimmermann, "Advanced Fuzzy Logic Control Technologies in Automotive Applications", *IEEE Conference on Fuzzy Systems*, ISBN 0-7803-0237-0, pp. 831-842.

5. Monitoreo de glaucoma a través de redes neuronales

El monitoreo de glaucoma a través de los cambios oftalmológicos de un paciente, requiere de una decisión basada en si los datos médicos representan una situación crítica o sospechosa de glaucoma como diagnóstico diferencial. Este trabajo presenta cómo, por medio de redes neuronales, se ayuda con el diagnóstico.

La decisión de los médicos acerca del ojo del paciente se basa en los siguientes conjuntos de datos:

- Mediciones directas
- Estimaciones paramétricas
- Descripciones verbales del paciente.

Las mediciones directas se refieren a la presión intraocular (PIO) que figura como un número real en una unidad de presión y los conjuntos de datos perimétricos. Estos últimos datos describen el estado del campo visual del paciente, medido por dispositivos especiales que detectan la pérdida de sensibilidad a la luz en diferentes puntos de la retina. Estos lugares son estimulados por un haz de luz de diferentes intensidades en distintos lugares dentro de un hemisferio con una iluminación de fondo mientras el paciente observa al centro. La "respuesta" de si se percibió un estímulo la da el paciente oprimiendo un botón. Por lo tanto, en general esta medición es subjetiva.

Otro aporte para el proceso de decisión son las descripciones de la pupila. Éstas se realizan principalmente al ver la pupila durante el examen de los ojos y la estimación de varios parámetros, como la razón copa-disco (RCD), la ubicación de la excavación o una comparación de la RCD de los ojos derecho e izquierdo. La diferencia entre ellos se clasifica por los médicos como "normal"o "en aumento".

Y el último aporte son las descripciones del paciente mismo que proporcionan información acerca de cambios entre las visitas al doctor y ofrecen una línea de tiempo de dichos cambios.

Con base en los datos anteriores, el oftalmólogo tiene que decidir si la información apunta hacia un posible glaucoma o no, y de ser así y de la severidad de los síntomas, decidir qué acción tomar (lo que llevará a otro proceso de decisiones más adelante). Estas decisiones pueden influir en el tiempo entre las visitas que debe hacer el paciente, la toma de medicamentos o alguna cirugía.

Ahora bien, al usar redes neuronales se pretende simular la diferente toma de decisiones que se pueden hacer y ayudar al médico a tomar una resolución final. Debido a la subjetividad de los datos, como la RCD y las respuestas a los estímulos de luz, se usan las redes neuronales las cuales modelan posibles clasificaciones a los síntomas de acuerdo con casos de glaucoma ya estudiados.

Por ejemplo, en el diagrama 1.3 se muestra un árbol de clasificaciones sólo para los datos perimétricos; el objetivo es refinar las clasificaciones para no sólo tener opciones como "normal/patológico", sino también respuestas como "cuestionable/probable". De esta manera se pretende hacer un árbol de decisiones para toda medición y evaluación hecha al paciente, cuyos resultados formarán parte de otra red más grande para el diagnóstico y recomendaciones del paciente.

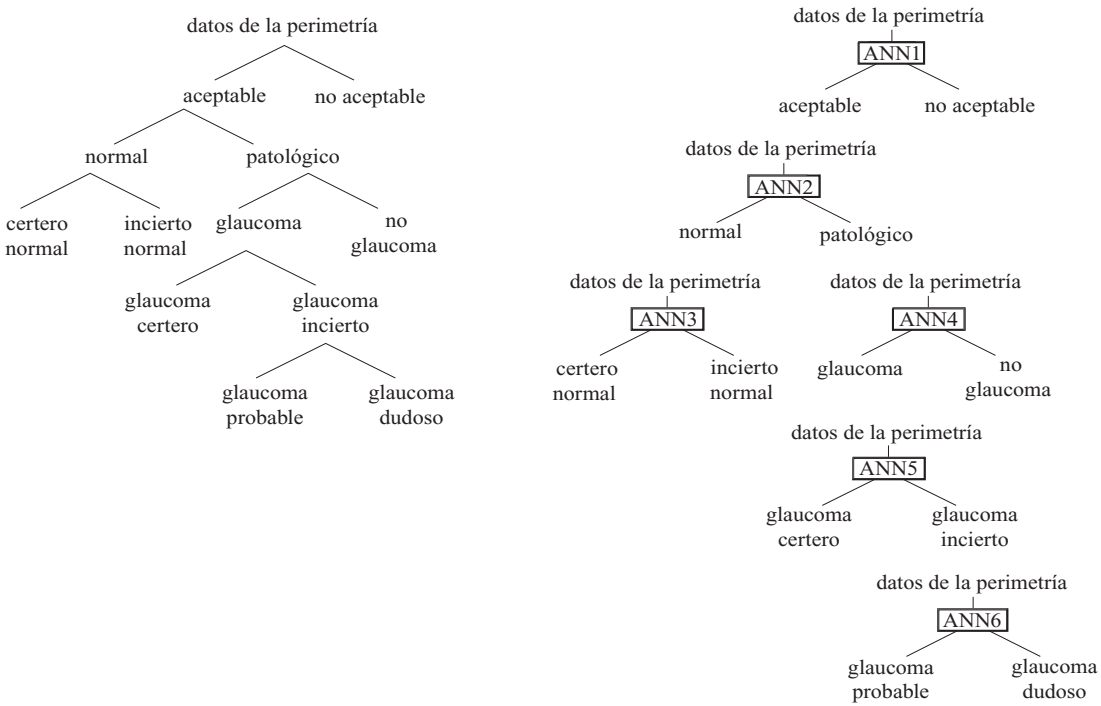


Diagrama 1.3 Árbol de clasificaciones para los datos paramétricos.

¿Por qué no se usó otro tipo de inteligencia artificial? Como se mencionó, las evaluaciones oftalmológicas no proporcionan datos numéricos sino se expresan de acuerdo con las observaciones del doctor o la perspectiva del mismo paciente. Por esta razón, el uso de lógica difusa no sería muy conveniente debido a que no existe algún tipo evaluación cuantificable o relación directa entre las diferentes observaciones. No obstante, se considera que en vez de tener una gran red neural conformada por subredes de cada evaluación y medición, se podría manejar los resultados de las subredes con lógica difusa ya que sólo cierta combinación de resultados apunta hacia probable glaucoma o no. Por otra parte, los algoritmos genéticos se utilizan en problemas con un mayor número de eventos y probabilidades; el diagnóstico del glaucoma al fin y al cabo se puede reducir a un número no tan grande de toma de decisiones, además de

que ya se tiene una idea de qué elementos conforman un diagnóstico positivo o no. Los AG se frecuentan más cuando no se tiene una idea muy clara de la mejor combinación para cierto resultado o cuando el flujo de información es muy cambiante. En este caso, la información es subjetiva pero no cambiante.

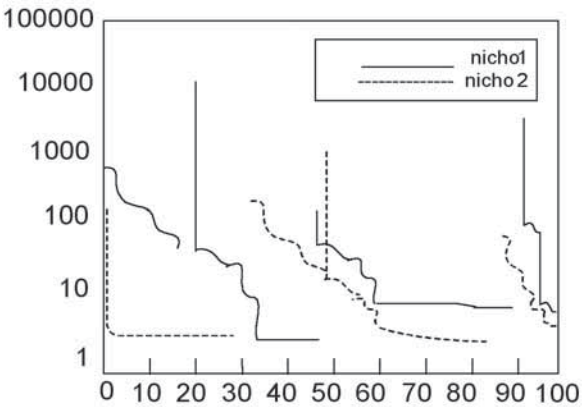
- G. Zahlmann, M. Schubert, M. Obermaier, G. Mann, "Concept of a knowledge-based monitoring system for glaucoma and diabetic retinopathy using a telemedicine approach", Proc. of the 13th Annual International Conf. IEEE-EMBS, IEEE Press, 111 pp.

6. Algoritmos genéticos para el diseño de sistemas de MRI (magnetic resonance imaging)

Esta aplicación describe un paquete de software para el diseño de equipo médico (MRI) llamado CamGASP, el cual usa algoritmos genéticos (AG). Como se sabe, estos algoritmos permiten a una población de varios diseños evolucionar hacia uno que sea el más apto. El objetivo principal de estos algoritmos es optimizar la función de costo, la cual busca homogeneizar al máximo la imagen de interés en el MRI; por otra parte, se considera factores ingenieriles importantes como el ancho, volumen y esfuerzos de las espirales electromagnéticas (coils), así como la intensidad del campo magnético que producen. Lo anterior, de acuerdo con las especificaciones de cada comprador.

En este caso, cada "gen" de nuestro código representa una espiral de nuestro magneto, los cuales se combinarán y mutarán de la manera más conveniente para producir la mejor resolución de imágenes en el MRI. Esto aplica no sólo para MRI de cuerpo completo: se daría la posibilidad de construir equipos más pequeños utilizados sólo para el procesamiento de imágenes de ciertas partes del cuerpo, u obtener a su vez PET scans (tomografías a base de emisiones de positrones).

La ventaja de este software entre otras es que fue diseñado para correr en paralelo en varias unidades que comparten la información de las mutaciones de los "genes" de información, lo que acelera el proceso de evolución del diseño. La gráfica 1.3 muestra cómo estos algoritmos se hicieron cada vez más óptimos y eficientes.



La evolución de la función aptitud con un número de generaciones. Se puede apreciar que la función aptitud mejora rápidamente al inicio de la optimización. Después, las curvas se aplanan hacia fuera mientras que otras mejoras en el diseño llegan a ser difíciles. Una vez que un nicho permanece inmóvil por más de 2000 generaciones, el nicho es reinicializado.

Gráfica 1.3 Comportamiento de las funciones de aptitud, cuando se aplican algoritmos genéticos

¿Por qué algoritmos genéticos? El diseño de los MRI varía de acuerdo a las necesidades de los compradores y su especialidad, por lo que las variables de diseño cambian conforme a cada uno de ellos. Por este motivo utilizar una solución con lógica difusa nos sería eficiente (tomaría más tiempo desarrollar una relación entre las espirales magnéticas y la función costo, y cada una variaría con el usuario). Por otra parte, también se podría utilizar redes neuronales; no obstante se tendría que comparar en cuánto tiempo se llega a una solución óptima con respecto a los algoritmos genéticos.

2

LÓGICA DIFUSA

INTRODUCCIÓN

Hace más de 30 años de la primera publicación de Lotfi Asker Zadeh, *Conjuntos difusos,* la cual sienta las bases de una nueva forma de lógica. El ser humano muestra dificultad para tomar decisiones cuando se tiene información imprecisa. La lógica difusa fue creada para emular la lógica humana y tomar decisiones acertadas a pesar de la información. Es una herramienta flexible que se basa en reglas lingüísticas dictadas por expertos. Por ejemplo, la velocidad de un automóvil es una variable que puede tomar distintos valores lingüísticos, como "alta", "media" o "baja". Estas variables lingüísticas están regidas por reglas que dictan la salida del sistema.

En otras palabras, la lógica difusa es un conjunto de principios matemáticos basados en grados de membresía o pertenencia, cuya función es modelar información. Este modelado se hace con base en reglas lingüísticas que aproximan una función mediante la relación de entradas y salidas del sistema (composición). Esta lógica presenta rangos de membresía dentro de un intervalo entre 0 y 1, a diferencia de la lógica convencional, en la que el rango se limita a dos valores: el cero o el uno.

Lotfi Asker Zadeh

Matemático azerbaiyano, profesor de la Universidad de Berkeley, nació en 1921 en Bakú, una ciudad en el mar Caspio de la antigua República Soviética de Azerbaiyán. Después de emigrar a Irán y de estudiar en la Universidad de Teherán, llegó a Estados Unidos donde continuó sus estudios en el MIT, en la Universidad de Columbia y finalmente en la Universidad de Berkeley. Es famoso por introducir en 1965 la teoría de conjuntos difusos o lógica difusa. Se le considera el padre de la teoría de la posibilidad, campo en el que ha recibido varios galardones, entre los que destaca la Medalla Richard W. Hamming en 1992 y doctorados honoris causa en varias instituciones del mundo, entre ellas la Universidad de Oviedo (1995), la Universidad de Granada (1996) y la Universidad Politécnica de Madrid (2007).

Mediante el uso de lógica difusa se puede representar la forma de la lógica humana, por ejemplo en afirmaciones como "el día es caluroso", "el automóvil va muy rápido", etc. En el primer caso, se sabe que hay alta temperatura, pero no se sabe a qué temperatura exactamente nos estamos refiriendo; en el segundo caso, se dice que "el automóvil va rápido", sin embargo nunca se especifica su velocidad.

¿Por qué usar un sistema difuso? Si se requiere automatizar un proceso que controla un trabajador, el sistema difuso tendrá la tarea de emular a dicho trabajador. Además, si se toma en cuenta que el tra-

bajador hace juicios con base en su criterio y experiencia, y que estos juicios y decisiones se realizan en forma lingüística (como "alto", "lento", etc.), se puede notar que un sistema convencional no maneja este tipo de entradas, mientras que el sistema difuso sí lo hace.

Otra ventaja del sistema de control basado en lógica difusa es que no es necesario conocer un modelo matemático del sistema real, pues se puede ver como una caja negra a la cual se le proporcionan entradas, y a través del sistema esta planta generará la salida deseada. En el control convencional sí es necesario conocer la planta del sistema.

Qué es una variable lingüística

Una variable lingüística adopta valores con palabras que permiten describir el estado de un objeto o fenómeno; estas palabras se pueden representar mediante conjuntos difusos. Una variable numérica toma valores numéricos, por ejemplo: edad = 65, mientras que una variable lingüística toma valores lingüísticos: edad es "viejo".

Todos los valores lingüísticos forman un conjunto de términos o etiquetas.

Para desarrollar un control con estas características, es necesario un experto, en este caso el trabajador, del cual se tomará un registro de las situaciones que se le presentan, así como de la solución que él les da. Esta experiencia se traduce en reglas que usan variables lingüísticas.

Para hacer este control es necesario tener las entradas del sistema y éstas se van a mapear a variables lingüísticas. A este mapeo se le llama difusificación. Con estas variables se forman reglas, las cuales serán las que regirán la acción de control que será la salida del sistema.

La anatomía básica de un controlador difuso consta de tres partes:

Reglas: estas son reglas que dictan la acción de control que se va a tomar. Éstas se derivan de un experto. Dichas reglas tiene la estructura de relaciones. La lógica difusa se basa en relaciones, las cuales se determinan por medio de cálculo de reglas "SI-ENTONCES" (con las cuales se puede modelar aspectos cualitativos del conocimiento humano, así como los procesos de razonamiento sin la necesidad de un análisis cuantitativo de precisión). Un ejemplo de una regla sería:

Si la temperatura es alta entonces se debe de encender el ventilador.

Difusificador: es el nexo entre las entradas reales y difusas. Todas las entradas necesitan ser mapeadas a una forma en que las reglas puedan utilizarlas.

Desdifusificador: toma un valor difuso de las reglas y genera una salida real.

Aplicaciones

Las aplicaciones de la lógica difusa se realizan en áreas multidisciplinarias que van desde la evolución tecnológica de los electrodomésticos, hasta programas computacionales para tomar decisiones y se han extendido a diversas áreas específicas que se mencionan a continuación.

Cámaras de video

La lógica difusa se emplea en los electrodomésticos con dos variantes: software y hardware. Las aplicaciones que contemplan el hardware incluyen el uso de tres sensores para lograr un enfoque automático del lente para captar al objeto indicado.

Reconocimiento

En áreas de seguridad que requiere la identificación, por ejemplo, de actividad volcánica a partir del monitoreo de anomalías en líneas largas de registro, o en el reconocimiento de caracteres y en los sis-

temas de vigilancia de video han sido analizados y probados para ofrecer alternativas paralelas a las tradicionales, mediante el almacenamiento de conocimiento de imágenes. Debido a que el criterio para determinar actividad volcánica peligrosa es vago, algoritmos basados en la lógica difusa permiten el reconocimiento automático de la actividad, así como la identificación de la morfología respectiva.

En la actualidad se han diseñado diversos dispositivos para el reconocimiento de caracteres impresos, denominados Asociadores Ópticos de Caracteres (OCA), los cuales clasifican entradas de tipo óptico en letras, números y otros caracteres mediante dispositivos de difusificación e inferencias. En el reconocimiento aplicado en cámaras de vigilancia se emplea conocimiento de expertos o aprendido a partir de imágenes previamente grabadas, para determinar mediante la función de asociación cuáles píxeles agrupados en una zona pertenecen a un objeto.

Controladores

De la misma manera, la lógica difusa se aplica a través de controladores difusos para la calidad del agua, los sistemas de operación automática de trenes, los sistemas automáticos de operación de contenedores, los elevadores, los reactores nucleares, las transmisiones de automóviles y las computadoras, por mencionar diversos ejemplos interesantes.

Sistemas de control en lazo abierto

Son aquellos en que la variable de salida (variable controlada) no tiene efecto sobre la acción de control (variable de control).

No se compara la salida del sistema con el valor deseado de la salida del sistema (referencia).

Para cada entrada de referencia le corresponde una condición de operación fijada.

La exactitud de la salida del sistema depende de la calibración del controlador.

En presencia de perturbaciones estos sistemas de control no cumplen su función adecuadamente.

Suele aparecer en dispositivos con control secuencial, en el que no hay una regulación de variables sino que se realizan una serie de operaciones de una manera determinada. Esa secuencia de operaciones puede venir impuesta por eventos (event-driven) o por tiempo (timedriven). Se programa utilizando PLC (controladores de lógica programable).

Ejemplo: Una lavadora de ropa

http://www.isa.cie.uva.es/~felipe/docencia/ra12itielec/tema1_trasp.pdf

Sistema de control en lazo cerrado

Son aquellos en que la señal de salida del sistema (variable controlada) tiene efecto directo sobre la acción de control (variable de control). Cuando opera en presencia de perturbaciones tiende a reducir la diferencia entre la salida de un sistema y alguna entrada de referencia. Esta reducción se logra manipulando alguna variable de entrada del sistema, siendo la magnitud de dicha variable de entrada función de la diferencia entre la variable de referencia y la salida del sistema.

Uso de lógica difusa en el Transporte

La aplicación más famosa es el controlador del tren subterráneo usado en Sendai, que supera a operadores humanos y a controladores automatizados convencionales. Los reguladores convencionales encienden o paran un tren reaccionando a los marcadores de la posición que demuestran qué tan lejos está el tren de una estación. Dado que los reguladores son programados rígidamente, el trayecto puede ser desigual; el regulador automatizado aplicará la misma presión de los frenos cuando un tren está, por ejemplo, a 100 metros de una estación, incluso si el tren va cuesta arriba o cuesta abajo. A mediados de los años ochenta, ingenieros de Hitachi utilizaron reglas difusas Hitachi para acelerar, retardar y frenar

los trenes de subterráneo de manera más suave que un operador humano hábil. Las reglas abarcaron una amplia gama de variables sobre el funcionamiento en curso del tren, tales como con qué frecuencia y por cuánto cambió su velocidad y qué tan cercana está la velocidad real a la velocidad máxima. En pruebas simuladas el controlador difuso venció a un controlador automático en cuanto a la comodidad de los pasajeros, en trayectos más cortos y en la reducción del 10% del consumo de energía del tren. Hoy en día el sistema difuso funciona en el subterráneo de Sendai durante horas pico y controla algunos trenes de Tokio también. Los seres humanos operan el subterráneo durante las horas de poco tráfico, para continuar con el desarrollo de sus habilidades.

Uso de lógica difusa en los sistemas de control

Este conjunto de aplicaciones ha sido motivado por el deseo satisfacer uno o más de los siguientes objetivos:

1. Mejorar la robustez que se obtiene con los métodos clásicos de control lineales.
2. Diseño de control simplificado para modelos complejos.
3. También se obtiene una implementación simplificada.
4. Autonomía.
5. Adaptabilidad.
6. En el caso del control difuso, no es necesario un modelo matemático de la planta.

Una de las bondades de los sistemas difusos es que no necesitan el modelado de una planta ni la ejecución de la identificación en tiempo real. La esencia del control difuso es que convierte la estrategia de control lingüístico, la cual se basa en el conocimiento de un experto, en una estrategia de control automático.

Es muy importante tener presente el hecho de que los controles difusos se basan en reglas de control empíricas, ya que el objetivo de esta sección es esbozar de una manera generalizada la lógica difusa.

Para este punto es bien sabido que el modelado difuso es un método para describir las características de un sistema usando reglas de inferencia difusas. Se hace especial referencia en el manejo de las reglas de control difuso, comúnmente extraídas de un experto, y de la sintonización de éstas. Para una mejor idea de por qué se suscitan algunos problemas en estos aspectos, en la siguiente sección se ofrece un panorama general de lo que es la lógica difusa usando como medio de explicación algunas comparaciones con la lógica tradicional.

CONCEPTOS DE LÓGICA BOOLEANA Y DIFUSA

La lógica booleana se basa en la idea de conjuntos nítidos (o convencionales), con pertenencia de unos (1) y ceros (0).

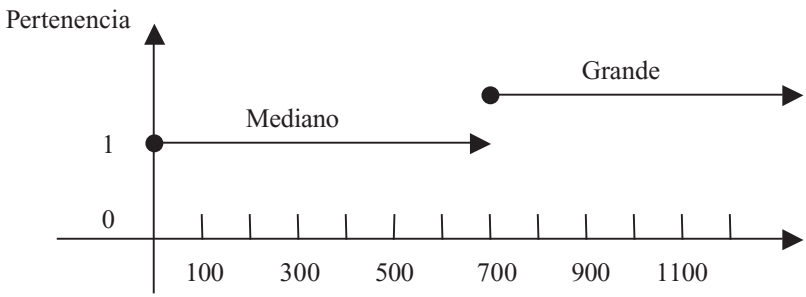


Fig. 2.1 Gráfica de valores numéricos en un conjunto nítido.

En la figura 2.1 se aprecia el concepto de nitidez, el cual se puede definir con base en un número preestablecido. Este número es el 700. ¿Por qué? Porque antes del 700 no se considera un número grande, y del 700 en adelante sí se considera un número grande. Hay "nitidez". Con base en esto se puede decir que el 699.999 es un número mediano y el 700.0001 es un número grande.

En otras palabras, en esta gráfica hay una escala de pertenencia, no hay ambigüedad. O es un número "grande" o es "mediano".

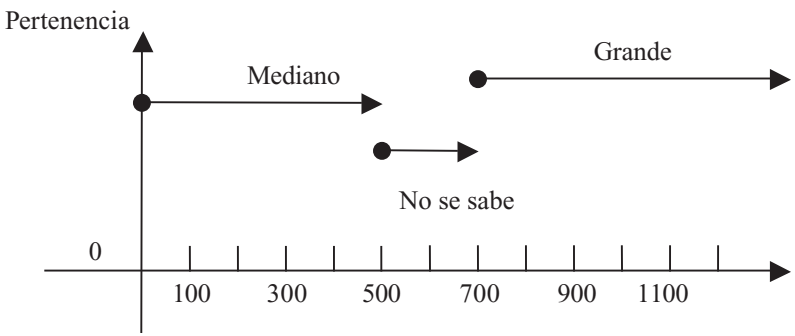


Fig. 2.2 Gráfica de valores numéricos en un conjunto semi-nítido.

En esta gráfica hay una ambigüedad. ¿Por qué? Es muy evidente que antes del 500 es un número mediano, y después del 700 es un número grande, pero en el rango entre 500 y 700 no se ha definido si es un número grande o mediano. Hay ambigüedad, no está clara la pertenencia. En este conjunto de elementos hay algunos que "medio pertenecen" tanto a los grandes como a los pequeños.

Con esta idea se puede uno imaginar la necesidad de la lógica difusa para representar casos de la vida cotidiana del ser humano. En la naturaleza es muy común ver distintos grados de pertenencia de distintos elementos. En un conjunto difuso hay grados de pertenencia que están entre cero (0) y uno (1).

A continuación analizaremos la idea básica de la lógica difusa desde la perspectiva de un diagrama de Venn.

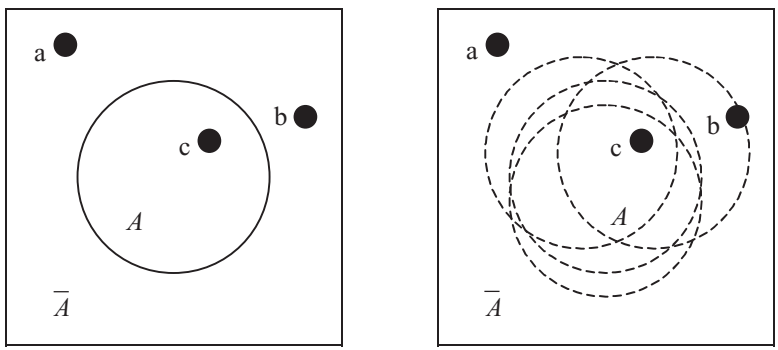


Fig. 2.3 Diagramas de Venn.

En estos diagramas se observa dos universos, los cuales contienen el conjunto A y su complemento. Cada universo consta de tres elementos. En el primer universo se puede ver que los elementos a y b pertenecen al complemento de $\overline{A}$, mientras que c pertenece a A. Sin embargo, en el segundo universo no se puede apreciar si el elemento b está en A o bien en $\overline{A}$ complemento. Por esta razón se dice que el primer universo es *convencional* (nítido) y el segundo es *difuso*.

Si analizamos más de cerca estos universos, *podremos darnos cuenta que dentro del conjunto difuso está el convencional*. En otras palabras:

Un *conjunto convencional* se define como aquel conjunto que sólo tiene dos grados de pertenencia, 0 o 1.

Un *conjunto difuso* se define como aquel conjunto que muestra todos los grados de pertenencia entre 0 y 1, incluidos éstos.

LÓGICA BOOLEANA

En la lógica booleana existe una función característica:

$$\chi_A = \begin{cases} 1, & X \in A \\ 0, & X \notin A \end{cases}$$

Esta función característica (χ_A) nos dice que si el elemento X pertenece al conjunto ($X \in A$), entonces la pertenencia es 1; si no pertenece $X \notin A$, entonces la pertenencia es 0.

Esta función expresa que en este universo nítido sólo hay dos posibles valores de pertenencia: 1 o 0.

Los conectores básicos dentro de la lógica booleana son los siguientes:

Tabla 2.1 Conectores básicos

$\cup, \vee$	unión, disyunción
$\cap, \wedge$	intersección, conjunción
- o ~	negación
$\in, \notin$	pertenencia, no pertenencia
$\forall$	para todo
|	tal que

***Lógica booleana**: es una lógica de conjuntos y nos sirve, principalmente, para definir formas de intersección entre conjuntos.*

En este caso, los conjuntos serían lo que quedan definidos por una palabra, es decir, serían conjuntos definidos por intensión. Si uso la palabra "psicoanálisis", ésta recubre todo el conjunto de elementos, para el caso, páginas web, en las que dicha palabra se encuentre incluida. Así, a partir de diferentes palabras se definen conjuntos de páginas agrupadas por el hecho de incluir (o no) esa determinada palabra. Estos conjuntos tendrán entre sí elementos en común y elementos que no. Una manera de afinar nuestra búsqueda consistirá en utilizar estos operadores booleanos para precisar el campo de nuestro interés.

Las principales opciones son:

***OR** - se suman los conjuntos definidos por dos palabras, es decir la respuesta será todas aquellas referencias donde aparezcan, indistintamente, UNA U OTRA de las palabras indicadas para búsqueda.*

***AND** - se trata de la intersección de los conjuntos definidos por las dos palabras, es decir sólo aquellas referencias que contengan AMBAS palabras a la vez.*

***NOT** - en este caso, aquellas referencias que tengan la primer palabra y no la segunda, es decir un primer conjunto, amputado de su parte común con otro.*

***NEAR** - como el AND, pero con la exigencia suplementaria de una cercanía entre las palabras.*

http://www.psiconet.com/enlaces/internet/boole.htm

Axiomas de los conjuntos convencionales

$X \in X$: X pertenece al universo X

$X \in A$: X pertenece al conjunto A

$X \notin A$: X no pertenece al conjunto A

$A \subset B$: A está contenido en B

$A \subseteq B$: A está contenido en o es equivalente a B

$\emptyset$: Conjunto vacío

$P(X)$: Conjunto de potencias

Un conjunto de potencias se define como todos los conjuntos posibles que se pueden formar con los elementos de un conjunto.

Ejemplo:

Teniendo el conjunto: $X = \{a, b, c\}$

Se tienen tres elementos, lo que nos da un posible de ocho combinaciones distintas (2^n, donde $n = 3$). Recordemos que en las combinaciones no importa el orden de los elementos en un conjunto dado, mientras que en las permutaciones sí importa.

A continuación se muestra el conjunto de potencias de X:

$$P(X) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$$

Operaciones en la lógica convencional

Las operaciones entre distintos conjuntos convencionales se muestran gráficamente en las siguientes imágenes.

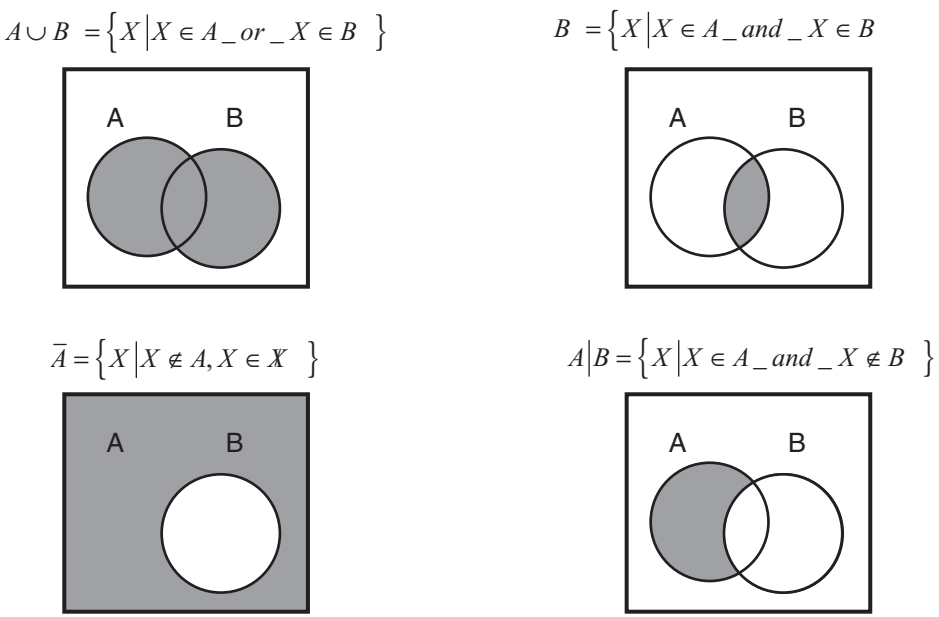


Fig. 2.4 Operaciones entre conjuntos convencionales.

Leyes de De Morgan

$A \cup \overline{A} = X$: A unión $\overline{A}$ complemento es igual al universo.

$A \cap \overline{A} = \emptyset$: A intersección $\overline{A}$ complemento es igual a el conjunto vacío.

$\overline{A \cap B} = \overline{A} \cup$: el complemento de la intersección de A con B es igual al complemento de $\overline{A}$ en unión con el complemento de $\overline{B}$.

$\overline{A \cup B} = \overline{A} \cap \overline{B}$: el complemento de la unión de A con B es igual a la intersección del complemento de $\overline{A}$ en intersección con el complemento de $\overline{B}$.

Las leyes de De Morgan establecen equivalencias. En otras palabras, la *negación de la conjunción* es equivalente a la *disyunción de las negaciones* y la negación de la disyunción es equivalente a la conjunción de las negaciones.

Ejemplo:

Haciendo una analogía con una cadena, si se tienen eslabones (E_i)

$$\overline{E_1 \cup E_2 \cup E_3 ..... \cup E_n} = \overline{E}_1 \cap \overline{E}_2 \cap \overline{E}_3 ..... \cap \overline{E}_n$$

Donde cada elemento es E_i, y donde $\overline{E}$ es que se rompa el eslabón, se puede decir:

"La unión rota de todos los eslabones (nótese que los eslabones están completos) es igual a la intersección de cada eslabón roto".

LÓGICA DIFUSA

Primero es indispensable establecer cierta nomenclatura y terminología. Cuando se habla de conjuntos nítidos, la variable típica a usar es la X. En conjuntos difusos la función de pertenencia que se utiliza es la μ. Ésta toma los valores entre cero (0) y uno (1); como se mencionó, la forma de representación de los conjuntos difusos puede ser de dos maneras: de forma continua o discreta, como se presenta a continuación.

Un conjunto difuso se escribe con una tilde arriba del nombre del conjunto:

$$\tilde{A} = \{a, b, c\}$$

Ésta se utiliza para diferenciarlos de los conjuntos nítidos.

En la lógica difusa los conjuntos se pueden presentar en forma continua o discreta.

Conjunto difuso discreto:

$$\tilde{A} = \left\{ \frac{\mu_A(X_1)}{X_1} + \frac{\mu_A(X_2)}{X_2} + ... \right\} = \left\{ \sum_i \frac{\mu_A(X_i)}{X_i} \right\}$$

En este punto es importante recordar que el signo (+) no indica suma sino unión. Dicha forma de representación es muy empleada en los sistemas digitales como los microcontroladores, computadoras, etcétera.

Conjunto difuso continuo:

$$\tilde{A} = \left\{ \int \frac{\mu_A(X)}{X} \right\}$$

Un conjunto convencional se define por una función característica, que se conoce también como función de pertenencia. El símbolo de integral denota $\int$ unión de elementos del conjunto.

Lógica simbólica

La lógica difusa tiene sus bases en la lógica simbólica. La lógica simbólica permite el establecimiento de un lenguaje artificial empleando símbolos para de esta forma representar argumentos lógicos complicados. Partiendo de *proposiciones*, es decir, de oraciones verdaderas o falsas, es posible traducirlas a un lenguaje de símbolos y representaciones, para posteriormente simplificar y ejecutar operaciones, e incluso traducir nuevamente hacia proposiciones de lenguaje ordinario.

Una proposición puede ser *simple*, con valor de Verdadero o Falso, o *compuesta*, dependiendo de los valores de verdad de componentes simples conectados a partir de operadores como *y*, *o*, *no*, entre otros. El operador *y* se denomina *conjunción*, y se simboliza con $\wedge$. El operador *o* se denomina *disyunción*, y se simboliza con $\vee$. A continuación se muestran las definiciones de la conjunción y disyunción para dos proposiciones simples p y q.

Tabla 2.2 Tabla de verdad de la conjunción y la disyunción

p	q	$p \wedge q$	$p \vee q$
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

La negativa de una proposición se denomina *negación*.

Ejemplo:

"Tengo un lápiz y una pluma" es una conjunción.
"Tengo un lápiz o una pluma" es una disyunción.
"No tengo un lápiz" es una negación.

Existen proposiciones *condicionales* de la forma "si p, entonces q"; p es el *antecedente* o *hipótesis*, y q es el *consecuente* o la *conclusión*. Una forma de simbolizar las proposiciones condicionales es: $p \rightarrow q$, y su tabla de verdad se muestra a continuación.

Tabla 2.3 Tabla de verdad de una proposición condicional

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

Ejemplo:

"Si $1 + 1 = 2$, entonces $4 > 0$" es una proposición condicional verdadera.
"Si $1 + 1 = 3$, entonces $4 > 0$" es una proposición condicional verdadera.
"Si $1 + 1 = 3$, entonces $4 < 0$" es una proposición condicional verdadera.
"***Si* $\mathbf{1 + 1 = 2}$*, entonces* $\mathbf{4 < 0}$*" es una proposición condicional falsa.***

Tautologías y quasi-tautologías

Las proposiciones que siempre son verdaderas se denominan *tautologías*. Para probar que una proposición es una tautología se construye su tabla de verdad y se verifica que todos los casos sean verdaderos. Si una proposición condicional es una tautología se denomina *implicación*.

En la lógica convencional existen ocho tautologías: trivial, ley de la doble negación, ley del medio excluido, razonamiento directo, razonamiento indirecto, ley de transitividad, ley de la contrarrecíproca y silogismo disyuntivo.

En la lógica difusa no existen tautologías pero sí quasi-tautologías, ya que las proposiciones no adquieren valores de 0 o 1 como en la lógica booleana, sino que contemplan valores de pertenencia entre 0 y 1.

Ejemplo:

Es muy claro que las tablas de verdad de elementos difusos se encuentran determinadas por las operaciones básicas de dichos elementos, como son la unión, intersección y complemento. Este ejemplo se puede utilizar para denotar una tabla de implicación difusa que parte de valores entre cero y uno.

Determinar $A \rightarrow B$

Tabla 2.4 Tabla de verdad de una quasi-tautología

A	B	$A \rightarrow B = \text{máx}(\bar{A}, B)$
0.3	0.2	0.7
0.3	0.8	0.8
0.7	0.2	0.3
0.7	0.8	0.8

Después de revisar las operaciones básicas de unión, intersección y complemento que se presentan más adelante, se recomienda realizar la tabla de verdad de la implicación.

Representación de conjuntos difusos discretos

Se puede expresar en forma gráfica un conjunto difuso discreto como la unión de sus elementos; el número de elementos depende del problema que se va a resolver. Es muy común que a través de una interpolación de cada uno de los elementos se pueda construir una trayectoria con cada uno de ellos, la cual es la base de una función de membresía o pertenencia. Tomemos como ejemplo el conjunto difuso que se muestra a continuación:

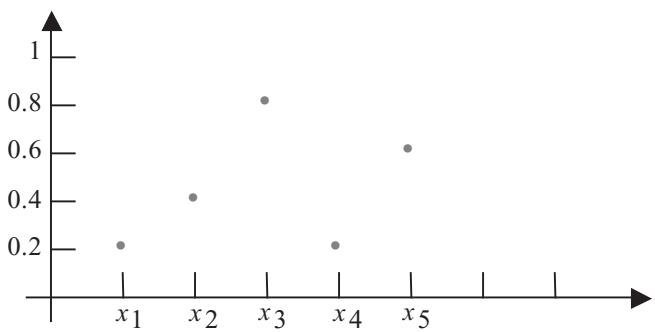


Fig. 2.5 Gráfica difusa.

$$\tilde{A} = \left\{ \frac{0.2}{x_1} + \frac{0.4}{x_2} + \frac{0.8}{x_3} + \frac{0.2}{x_4} + \frac{0.6}{x_5} \right\}$$

El conjunto difuso $\tilde{A}$ se define a partir de la gráfica. Como se puede observar, los elementos de este conjunto están formados por fracciones, en las cuales el denominador es el elemento y el numerador es el grado de pertenencia del elemento, pero con una cierta función de membresía.

Operaciones en la lógica difusa empleando conjuntos difusos

Existen operaciones básicas que se definen por operadores binarios para el caso de lógica booleana; en el caso de la lógica difusa se presentan a continuación los operadores que se emplean para realizar las operaciones básicas (unión, intersección y complemento).

Las operaciones básicas entre conjuntos difusos que se definirán son la unión, intersección y complemento, de la siguiente forma:

$$\mu_{\tilde{A}\cup\tilde{B}}(X) = \mu_{\tilde{A}}(X) \vee \mu_{\tilde{B}}(X)$$

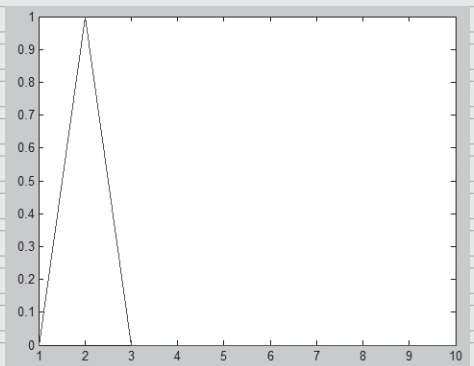
$$\mu_{\tilde{A}\cap\tilde{B}}(X) = \mu_{\tilde{A}}(X) \wedge \mu_{\tilde{B}}(X)$$

$$\mu_{\bar{\tilde{A}}}(X) = 1 - \mu_{\tilde{A}}(X)$$

La intersección se clasifica como una norma triangular (norma T) y la unión es una co-norma T (o norma S).

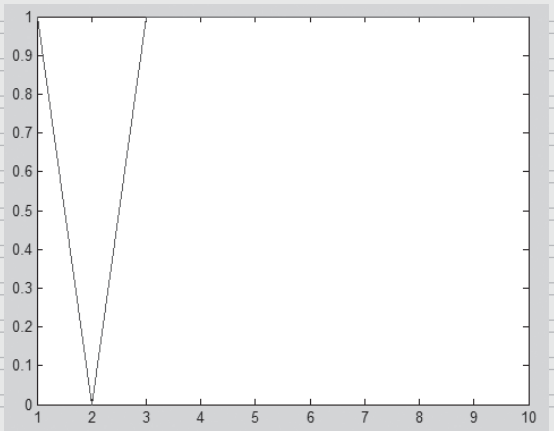
Ejemplo de programa de operación difusa realizado en MATLAB®

```
Gráfica triangular


for time=1:10,
if (time<alfa)
        y(time)=0;
elseif and(alfa<=time,time<=beta)
        y(time)=(time-alfa)/(beta-alfa);elseif and (beta <= time, time <= gamma)
        y(time)=((gamma-time)/(gamma-beta));
elseif (time>gamma)
        y(time)=0;
end
end
plot (y)
```



Gráfica triangular.

```
Complemento de gráfica triangular

for time=1:10,
if (time<alfa)
        y(time)=1;
elseif and(alfa<=time,time<=beta)
        y(time)=1-(time-alfa)/(beta-alfa);
elseif and (beta <= time, time <= gamma)
        y(time)=1-((gamma-time)/(gamma-beta));
elseif (time>gamma)
        y(time)=1;
end
end
plot (y)
```



Gráfica triangular inversa.

Nomenclatura	
Unión	$\cup \Rightarrow \vee \Rightarrow$ *or* $\Rightarrow$ *máximo* (supremum)
Intersección	$\cap \Rightarrow \vee \Rightarrow$ *or* $\Rightarrow$ *mínimo* (infimum)
La pertenencia de X en el conjunto $\tilde{A}$	$\mu_{\tilde{A}}(X)$

Norma triangular (T)

Las condiciones que definen a la norma T son exactamente las mismas del monoide abeliano parcialmente ordenado en el intervalo unitario real [0,1]. La operación monoidal de cualquier monoide abeliano parcialmente ordenado L se denomina una norma triangular sobre L. Una norma T es una función $T : [0,1] \times [0,1] \rightarrow [0,1]$ que satisface las siguientes propiedades:

Conmutatividad: $T(a, b) = T(b, a)$
Monotonía: $T(a, b) \leq T(c, d)$ si $a \leq c$ y $b \leq d$
Asociatividad: $T(a, T(b, c) = T(T(a, b), c)$
El número 1 actúa como elemento identidad: $T(a, 1) = a$

Considerando que una norma T es una operación algebraica binaria en el intervalo [0,1], la notación algebraica es común y la norma T se denota con un asterisco (*).

Una norma T se denomina *continua* si es continua al igual que una función, en la topología de intervalo usual de [0,1]2 (similar para la continuidad derecha e izquierda). Una norma T* se considera *arquimediana* si posee la propiedad arquimediana: para cada x, y en el intervalo abierto (0,1) hay un

número natural *n* tal que *x** . . . **x* (*n* veces) es menor o igual a *y*. Una norma T continua y arquimediana se llama *estricta* si 0 es su elemento único nilpotente, de otra forma se denomina *nilpotente*.

El orden parcial usual para las normas T se considera a partir de un punto:

$T_1 \leq T_2$ si $T_1(a, b) \leq T_2(a, b)$ para todo a, b en [0, 1].

Como funciones, las normas T largas se consideran más fuertes que las menores. En la semántica de la lógica difusa, no obstante, entre más grande sea una norma T, menor será la conjunción que representa.

Las normas T se emplean para representar intersección en la teoría de conjuntos difusos.

Co-normas T (normas S)

Las co-normas T, también denominadas normas S, son duales a las normas T bajo la operación de reversión de orden, la cual asigna $1 - x$ a x en el intervalo [0,1]. Dada una norma T, la co-norma complementaria se define como $\perp (a,b) = 1 - T(1-a, 1-b)$. Esto generaliza las leyes de De Morgan.

Una co-norma T satisface las siguientes condiciones, las cuales pueden emplearse para una definición equivalente axiomática independientemente de las normas T:

Conmutatividad: $S(a, b) = S(b, a)$
Monotonía: $S(a, b) \leq s(c, d)$ si $a \leq c$ y $b \leq d$
Asociatividad: $S(a, S(b, c)) = S(S(a, b), c)$
Elemento identidad: $S(a, 0) = a$

Las co-normas T se usan para representar *disyunción* en la lógica difusa, y *unión* en la teoría de conjuntos difusos.

Aseveraciones booleanas aplicadas a la lógica difusa

A continuación se muestra dos enunciados que en la lógica booleana son correctos, pero que en la lógica difusa no lo son.

$$\tilde{A} \cup \overline{\tilde{A}} = X$$
$$\tilde{A} \cap \overline{\tilde{A}} = \emptyset$$

En seguida se analiza más de cerca estas aseveraciones desde los principios y operaciones difusas antes expuestos. Siendo:

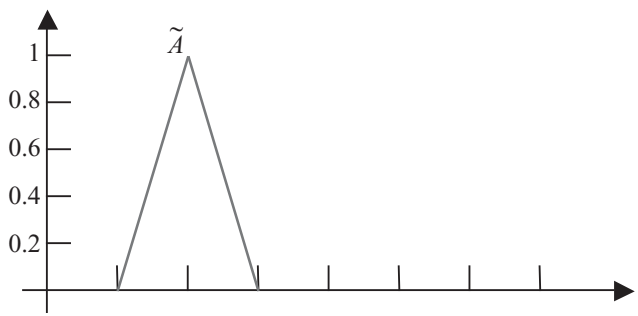


Fig. 2.6 Gráfica difusa.

Con base en:

$$\mu_{\bar{\tilde{A}}}(X) = 1 - \mu_{\tilde{A}}(X)$$

Sabemos que el complemento de $\bar{\bar{\tilde{A}}}(\tilde{A})$ es $1 - \tilde{A}$, por lo que si desarrollamos de manera gráfica podemos ver que:

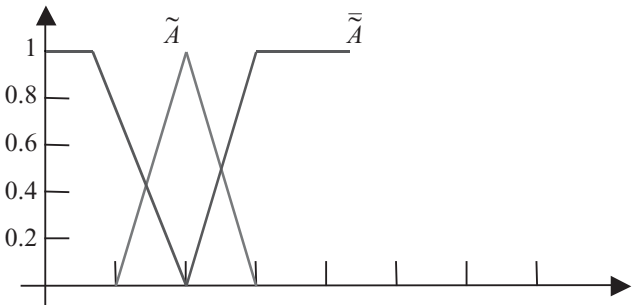


Fig. 2.7 Gráfica difusa.

Por lo que si hacemos la unión de $\tilde{A}$ y $\bar{\bar{\tilde{A}}}$ vemos que no da *X*, es decir, el universo.

De igual forma, si hacemos la intersección podremos observar que no queda un conjunto vacío, como asevera el enunciado, dicho lo cual, tampoco se cumple.

En conclusión, se puede observar que a primera instancia pareciera que las dos aseveraciones son correctas, pero, si bien en lógica convencional son ciertas, hay unos axiomas que en lógica difusa no se cumplen.

En el siguiente ejemplo se propone realizar las operaciones básicas de conjuntos difusos para su buen manejo.

Ejemplo:

Con base en los conjuntos:

$$\tilde{A} = \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} \right\}$$

$$\tilde{B} = \left\{ \frac{0.5}{x_1} + \frac{0.6}{x_2} \right\}$$

Encontrar:

a) $\mu_{\tilde{A} \cup \tilde{B}}(x) =$ b) $\mu_{\tilde{A} \cap \tilde{B}}(x) =$ c) $\bar{\mu}_{\tilde{A}}(x) =$

Solución:

a) $\mu_{\tilde{A} \cup \tilde{B}}(x) = \left\{ \frac{0.8}{x_1} + \frac{0.7}{x_2} \right\}$ b) $\mu_{\tilde{A} \cap \tilde{B}}(x) = \left\{ \frac{0.5}{x_1} + \frac{0.6}{x_2} \right\}$ c) $\mu_{\tilde{A}}(x) = \left\{ \frac{0.2}{x_1} + \frac{0.3}{x_2} \right\}$

Ejemplo:

De acuerdo con los siguientes conjuntos difusos:

$$\tilde{A} = \left\{ \frac{1}{1} + \frac{0.75}{1.5} + \frac{0.3}{2} + \frac{0.15}{2.5} + \frac{1}{3} \right\}$$

$$\tilde{B} = \left\{ \frac{1}{1} + \frac{0.6}{1.5} + \frac{0.8}{2} + \frac{1}{2.5} + \frac{0}{3} \right\}$$

Realizar las operaciones:

a) $\tilde{A} \cup \tilde{B} =$ b) $\tilde{A} \cap \tilde{B} =$ c) $\tilde{A} \cap \bar{\tilde{B}} =$ d) $\tilde{B} \cap \bar{\tilde{A}} =$ e) $\bar{\tilde{A}} \cup \tilde{A} =$ f) $\bar{\tilde{B}} \cap \tilde{B} =$

Solución:

$$\bar{\tilde{A}} = \left\{ \frac{0}{1} + \frac{0.25}{1.5} + \frac{0.7}{2} + \frac{0.85}{2.5} + \frac{0}{3} \right\}$$

$$\bar{\tilde{B}} = \left\{ \frac{0}{1} + \frac{0.4}{1.5} + \frac{0.2}{2} + \frac{0}{2.5} + \frac{1}{3} \right\}$$

a) $\tilde{A} \cup \tilde{B} = \left\{ \frac{1}{1} + \frac{0.75}{1.5} + \frac{0.8}{2} + \frac{1}{2.5} + \frac{1}{3} \right\}$

b) $\tilde{A} \cap \tilde{B} = \left\{ \frac{1}{1} + \frac{0.6}{1.5} + \frac{0.3}{2} + \frac{0.15}{2.5} + \frac{0}{3} \right\}$

c) $\tilde{A} \cap \bar{\tilde{B}} = \left\{ \frac{0}{1} + \frac{0.4}{1.5} + \frac{0.2}{2} + \frac{0}{2.5} + \frac{1}{3} \right\}$

d) $\tilde{B} \cap \bar{\tilde{A}} = \left\{ \frac{0}{1} + \frac{0.25}{1.5} + \frac{0.7}{2} + \frac{0.85}{2.5} + \frac{0}{3} \right\}$

e) $\bar{\tilde{A}} \cup \tilde{A} = \left\{ \frac{1}{1} + \frac{0.75}{1.5} + \frac{0.7}{2} + \frac{0.85}{2.5} + \frac{1}{3} \right\}$

f) $\bar{\tilde{B}} \cap \tilde{B} = \left\{ \frac{0}{1} + \frac{0.4}{1.5} + \frac{0.2}{2} + \frac{0}{2.5} + \frac{0}{3} \right\}$

Operaciones entre conjuntos difusos

A continuación se describe las operaciones más representativas entre conjuntos difusos, ejemplificadas algunas de ellas gráficamente para su mejor apreciación.

Es posible considerar que el empleo de estas operaciones de lógica difusa afecta directamente el grado de pertenencia de los elementos que conforman el conjunto difuso, por lo que se tendría descripciones diferentes al emplear estos operadores. Dichas descripciones permiten realizar un mapeo entre los valores nítidos y los valores difusos. Por esta razón, éstas reciben el nombre de funciones de membresía, que son las responsables de mapear los elementos de un conjunto nítido con su grado correspondiente de membresía. Algunas operaciones son el resultado de operaciones con dos o más conjuntos, como es el caso del producto del conjunto difuso.

Producto de dos conjuntos difusos

$$\mu_{\tilde{A} \cdot \tilde{B}}(x) = \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

Mientras que la potencia es la manipulación de un solo conjunto, esto es elevarlo a una potencia alterando los valores de membresía de cada elemento del conjunto.

Potencia de un conjunto difuso

$$\mu_{\tilde{A}^{\alpha}}(x) = \left[\mu_{\tilde{A}}(x) \right]^{\alpha}$$

Donde:

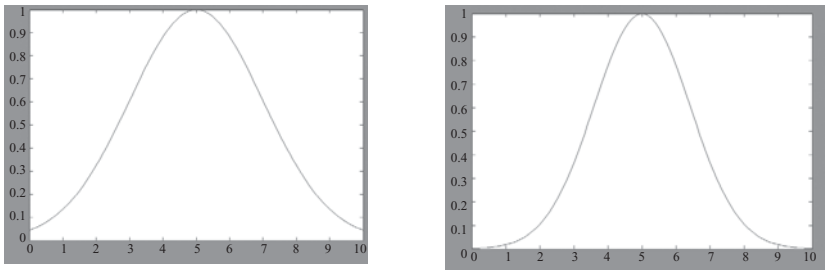
α : Número real positivo

$\tilde{A}^{\alpha}$: Conjunto difuso nuevo

Concentración

$$\mu_{CON(\tilde{A})}(x) = \left(\mu_{\tilde{A}}(x) \right)^2$$

Esta operación es el equivalente al término lingüístico "muy" ("very") que modifica la característica de una función de pertenencia o membresía, concentrando los valores de la misma, como se aprecia en el ejemplo de la figura 2.8.



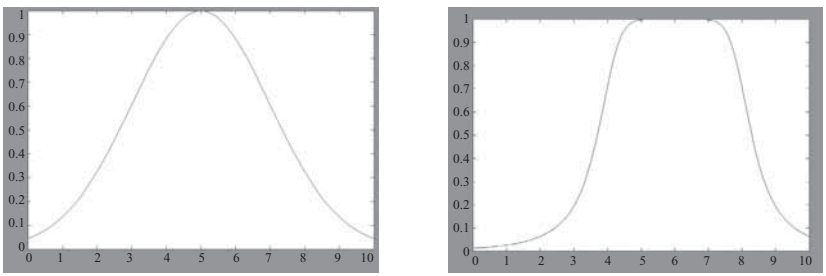
a) *b*)

Fig. 2.8 *a*) Función $\mu_{\tilde{A}}(x)$ original y *b*) su concentración.

Dilación

$$\mu_{DIL(\tilde{A})}(x) = \sqrt{\mu_{\tilde{A}}(x)} = \left(\mu_{\tilde{A}}(x)\right)^{1/2}$$

Equivale a la expresión lingüística de "más o menos" ("more or less"), modificando los parámetros de la función según se muestra en el ejemplo de la figura 2.9.



a) *b*)

Fig. 2.9 *a*) Función $\mu_{\tilde{A}}(x)$ original y *b*) su dilación.

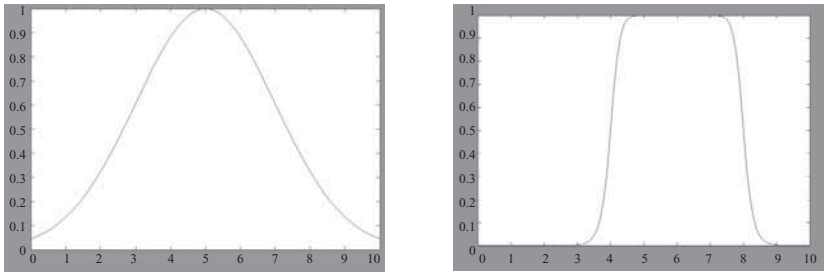
Intensificación de contraste

Incrementa la función de membresía donde los valores sean mayores a 0.5 y la disminuye para valores menores a 0.5.

$$\mu_{INT(\tilde{A})}(x) = 2\left(\mu_{\tilde{A}}(x)\right)^2 \text{ para } 0 \le \mu_{\tilde{A}}(x) \le 0.5$$

$$\mu_{INT(\tilde{A})}(x) = 1 - 2\left(1 - \mu_{\tilde{A}}(x)\right)^2 \text{ para } 0.5 \le \mu_{\tilde{A}}(x) \le 1$$

En la siguiente figura se presenta el efecto de dicha operación; cabe destacar que el punto de variación se fija en 0.5. Pero no es la única alternativa, puede tener varias opciones de selección para diferentes puntos.



a) *b*)

Fig. 2.10 *a*) Función $\mu_{\tilde{A}}(x)$ original y *b*) su intensificación de contraste.

Corte alfa

El corte alfa se define como un conjunto nítido o difuso que contiene a los elementos del conjunto $\tilde{A}$ que tengan un valor de membresía mayor o igual al valor escalar de la constante alfa; esta constante se define entre los valores de 0 a 1. En donde la constante alfa define el valor de referencia, valor de membresía alfa, para decidir cuáles son los elementos que pertenecen o no a dicho conjunto. En algunos casos se usan cortes alfa inversión, que son todos los elementos con valor de membresía menor o igual al valor escalar alfa.

$$A_\alpha = \left\{ x \in X \mid \mu_{\tilde{A}}(x) \geq \alpha \right\}$$

Ejemplo

Encontrar el corte alfa para el siguiente conjunto difuso tomando como valor alfa 0.5.

$$\tilde{A} = \left\{ \frac{1}{1} + \frac{1}{2} + \frac{0.75}{3} + \frac{0.5}{4} + \frac{0.3}{5} + \frac{0.3}{6} + \frac{0.1}{7} + \frac{0.1}{8} \right\}$$

$$A_{0.5} = \left\{ 1, 2, 3, 4 \right\}$$

Propiedades de los conjuntos difusos

Doble negación:	$\tilde{\tilde{\tilde{A}}} = \tilde{A}$
Idempotencia:	$\tilde{A} \cup \tilde{A} = \tilde{A}$
	$\tilde{A} \cap \tilde{A} = \tilde{A}$
Conmutatividad:	$\tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A}$
Asociación:	$\left(\tilde{A} \cap \tilde{B}\right) \cap \tilde{C} = \tilde{A} \cap \left(\tilde{B} \cap \tilde{C}\right)$
Absorción:	$\tilde{A} \cap \left(\tilde{A} \cup \tilde{B}\right) = \tilde{A}$
Leyes de De Morgan:	$\overline{\tilde{A} \cup \tilde{B}} = \tilde{\tilde{A}} \cap \tilde{\tilde{B}}$
	$\overline{\tilde{A} \cap \tilde{B}} = \tilde{\tilde{A}} \cup \tilde{\tilde{B}}$

Funciones de membresía y sus partes básicas

Para la representación de los grados de pertenencia de cada uno de los elementos que conforman el conjunto difuso, lo más natural es extraer los datos de los fenómenos que se va a representar y con ellos

definir la forma de la función de membresía. De otra manera existen metodologías que permiten asignar grados de membresía a cada uno de los elementos del conjunto.

Existen funciones de membresía convencionales y no convencionales que permiten realizar un mapeo de un universo nítido a un universo difuso (grados de membresía entre 0 y 1). Entre las funciones de membresía convencionales se tienen las siguientes.

Función de saturación

La función de saturación es la más sencilla de ellas. Tiene un valor de 0 hasta cierto punto y después crece con pendiente constante hasta alcanzar el valor de 1, en donde se estaciona. La figura 2.11 muestra la gráfica de esta función de membresía. Se puede notar que esta gráfica tiene sus cambios de pendiente en los valores 5 y 10.

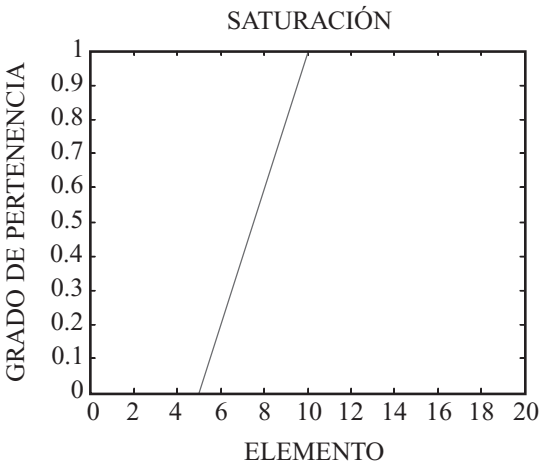


Fig. 2.11 Función de saturación.

Este tipo de funciones describe muy bien situaciones en donde se alcanza un nivel máximo a partir de cierto punto, por ejemplo la estatura de las personas o el rendimiento académico de un alumno. Se podría considerar que una persona es alta con grado de pertenencia unitario a partir de 1.90 m, o que un alumno es excelente con grado de pertenencia unitario a partir de un promedio general de 95.

El código de la gráfica de la función de membresía de saturación se encuentra en el Anexo A.

Función hombro

La siguiente función que se muestra es la función hombro, que es, por decirlo de alguna manera, la contraparte de la función saturación. En este tipo de funciones se inicia en un valor unitario y se desciende con constante pendiente hasta alcanzar el valor de 0. La figura 2.12 muestra la gráfica de esta función de membresía, la cual tiene sus cortes en 9 y 18.

Este tipo de función es útil cuando el grado de pertenencia es total en valores pequeños y decae conforme el valor de la variable aumenta; por ejemplo: el nivel de oxígeno en una pecera; mientras el número de peces no sobrepase un límite contemplado, el oxígeno es suficiente; a medida que el número de peces aumente, el oxígeno será más limitado hasta que llegue el punto en que no sea suficiente y los peces mueran.

El código de la gráfica de la función de membresía hombro se encuentra en el Anexo B.

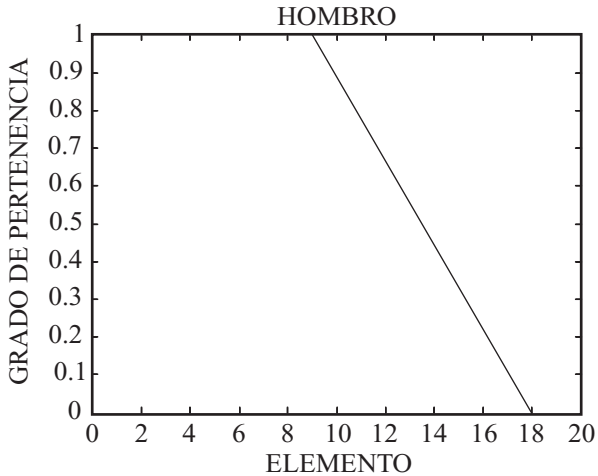


Fig. 2.12 Función hombro.

Función triangular

A continuación se muestra la función triangular. Su forma, como su nombre lo indica, consta de una parte de pendiente positiva constante hasta alcanzar la unidad, y una vez que lo ha logrado desciende de manera uniforme. La figura 2.13 muestra un ejemplo de esta función, la cual tiene comienzo en el valor 8 y termina en 12, teniendo el pico en el valor 10.

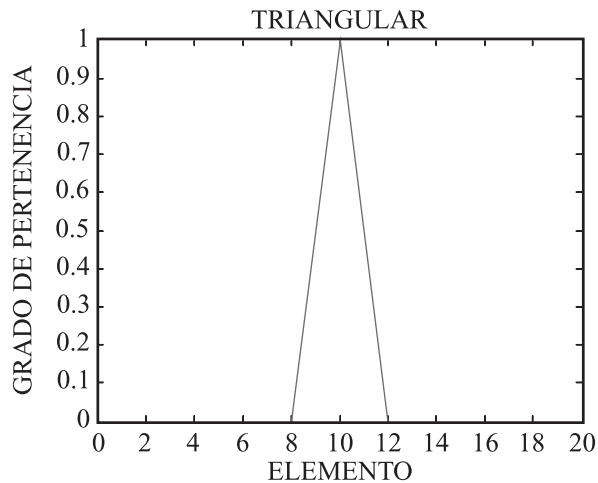


Fig. 2.13 Función triangular.

La función triangular es muy adecuada para definir situaciones en las que se tiene un valor óptimo central, el cual se va perdiendo conforme uno se aleja de él. Un ejemplo de esta situación es la temperatura corporal, que tiene un valor óptimo de 37º centígrados, pero que por debajo de 35º o por encima de 39º se considera peligrosa, es decir, el nivel de pertenencia al conjunto de temperaturas seguras en el cuerpo humano es 0.

El código de la gráfica de la función de membresía triangular se encuentra en el Anexo C.

Función trapecio o Pi

Una generalización de la función triangular es la función trapecio o función Pi. En el caso de esta función de membresía, no sólo se tiene un valor para el cual la pertenencia es unitaria, sino toda una franja que varía su ancho dependiendo del fenómeno observado. En la figura 2.14 se aprecia que la gráfica empieza a crecer de manera constante en 6, llega al valor unitario en 8 donde se conserva hasta 10 y decrece de manera uniforma hasta 12.

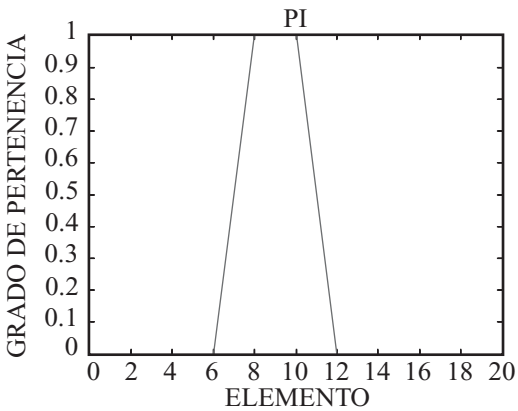


Fig. 2.14 Función trapecio o Pi.

La forma de esta función es muy utilizada, ya que como se mencionó se emplea cuando hay un rango de valores óptimos, alrededor de los cuales las condiciones no son adecuadas. Un buen ejemplo de esto es la iluminación de un salón de clases. Existe un rango en el cual la iluminación es agradable para las personas, pero por debajo de dicho rango la luz no es suficiente para leer el pizarrón, y por encima de él es molesto para la vista de los estudiantes.

El código de la gráfica de la función de membresía trapecio o Pi se encuentra en el Anexo D.

Función "S" o sigmoidal

Finalmente tenemos la función "S". La forma de esta función es similar a la de saturación. Sin embargo, como su nombre lo indica, el segmento de subida no es una línea recta, sino una curva de segundo orden,

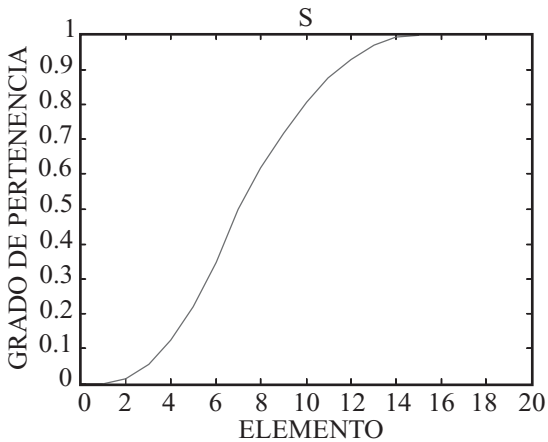


Fig. 2.15 Función "S".

la cual cambia de concavidad en un punto dado, y una vez que llega a 1 se mantiene en este valor. En la figura 2.15 se muestra una función de este tipo: la gráfica comienza en 1, tiene un cambio de concavidad en 7 y alcanza el valor máximo en 15.

Esta función también define fenómenos como los definidos por la función de saturación. La diferencia principal radica precisamente en que los cambios de pertenencia a cierto conjunto no son tan drásticos, por lo que se apega más a la realidad. La pertenencia a la clase media basada en el ingreso monetario mensual es un ejemplo que puede ser definido por esta función.

Descripción matemática de las funciones de membresía

Función de pertenencia del tipo hombro o saturación derecha:

$$f(x)=\begin{cases} x \le \alpha \rightarrow & 0 \\ \alpha \le x \le \beta \rightarrow & \dfrac{x-\beta}{\alpha-\beta} \\ x \le \beta \rightarrow & 1 \end{cases}$$

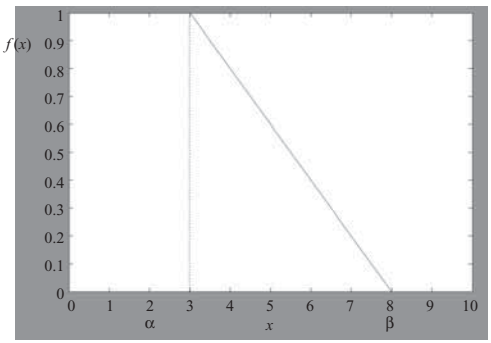


Fig. 2.16 Función tipo saturación derecha.

Función de pertenencia del tipo hombro o saturación izquierda:

$$f(x)=\begin{cases} x \le \alpha \rightarrow & 1 \\ \alpha \le x \le \beta \rightarrow & \dfrac{x-\alpha}{\beta-\alpha} \\ x \le \beta \rightarrow & 0 \end{cases}$$

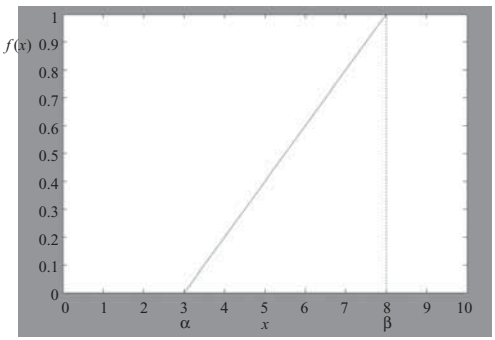


Fig. 2.17 Función tipo saturación izquierda.

Función PI:

$$f(x)=\begin{cases} \alpha \le x \le \beta \rightarrow & \dfrac{x-\alpha}{\beta-\alpha} \\ \beta \le x \le \varphi \rightarrow & 1 \\ \varphi \le x \le \delta \rightarrow & \dfrac{x-\varphi}{\delta-\varphi} \\ 0 \ \textit{de otra manera} & \end{cases}$$

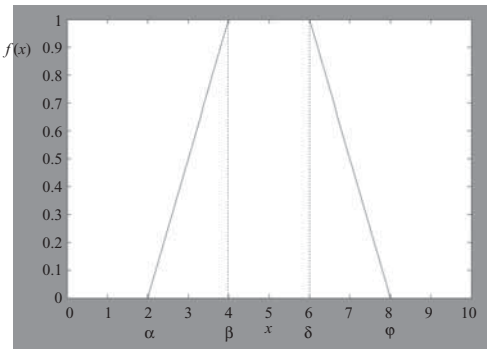


Fig. 2.18 Función tipo PI.

Característica tipo triangular:

$$f(x)=\begin{cases}\alpha \le x \le \beta \rightarrow & \dfrac{x-\alpha}{\beta-\alpha}\\ \beta \le x \le \delta \rightarrow & \dfrac{x-\delta}{\beta-\delta}\\ 0 \ de\ otra\ manera\end{cases}$$

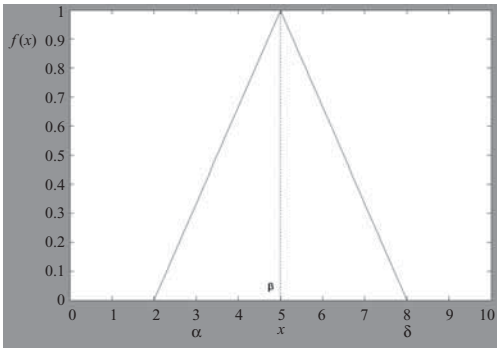


Fig. 2.19 Función tipo triangular.

Tipo "S" o sigmoidal:

$$f(x)=\begin{cases}x \le \alpha \rightarrow & 0\\ \alpha \le x \le \beta \rightarrow & 2\left(\dfrac{x-\alpha}{\gamma-\alpha}\right)^2\\ \alpha \le x \le \gamma \rightarrow & 1-2\left(\dfrac{x-\gamma}{\gamma-\alpha}\right)^2\\ 1\ de\ otra\ manera\end{cases}$$

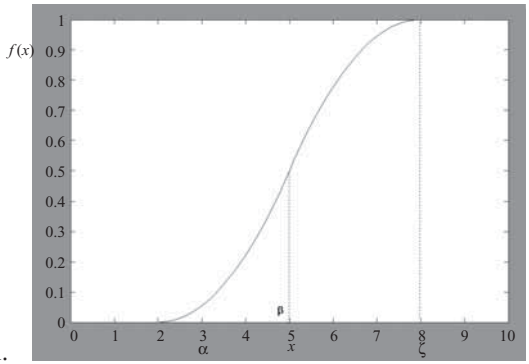


Fig. 2.20 Función tipo sigmoidal.

Aplicaciones reales de las distintas funciones de membresía

Ángulos de giro de un volante, clasificados de acuerdo con su magnitud en derecha, izquierda y centro.

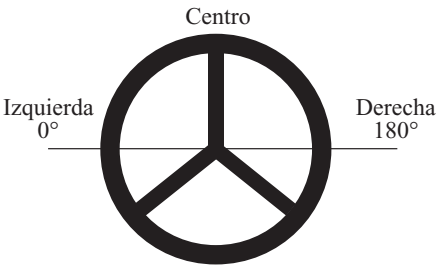
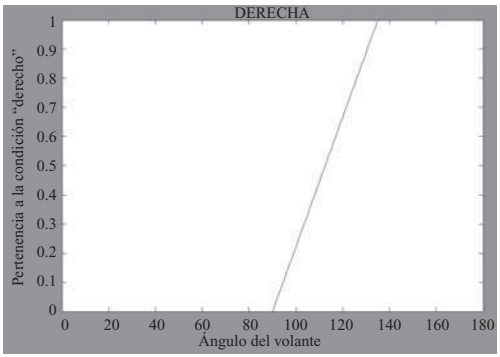


Fig. 2.21 Función hombro izquierdo aplicada al ángulo de un volante.



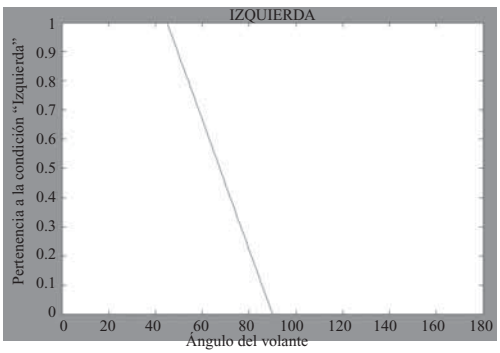


Fig. 2.22 Función hombro derecho aplicada al ángulo de un volante.

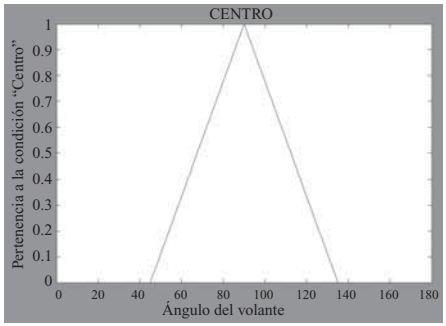


Fig. 2.23 Función triangular aplicada al ángulo de un volante.

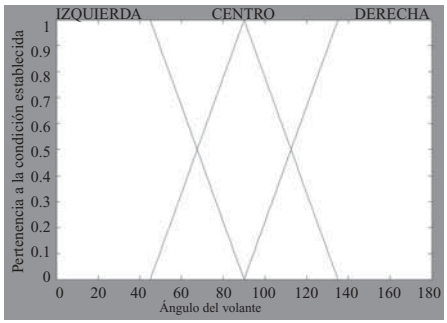


Fig. 2.24 Traslape de las funciones de pertenencia empleadas en el ejemplo del volante.

El estado físico del agua en transición (donde coexisten dos distintos estados) durante un experimento clasificado en hielo, agua y vapor dependiendo de la temperatura de la misma, implica el uso de una función de pertenencia trapezoidal, además de las funciones de hombros derecho e izquierdo.

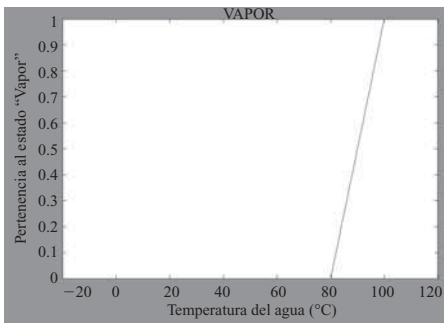


Fig. 2.25 Función hombro izquierdo aplicada a la temperatura del agua.

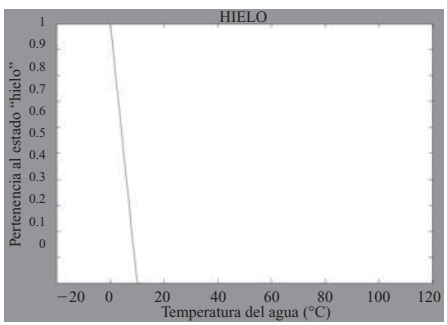


Fig. 2.26 Función hombro derecho aplicada a la pertenencia al estado "hielo".

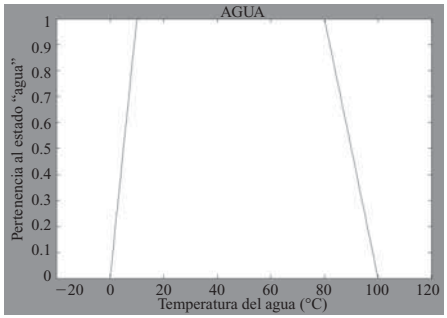


Fig. 2.27 Función trapezoidal o Pi aplicada a la pertenencia al estado "agua".

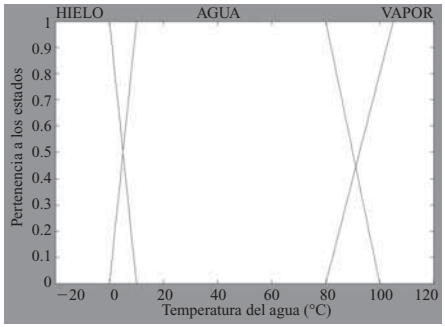


Fig. 2.28 Traslape de las funciones de pertenencia correspondientes a los estados físicos del agua en transición.

Partes de una función de membresía

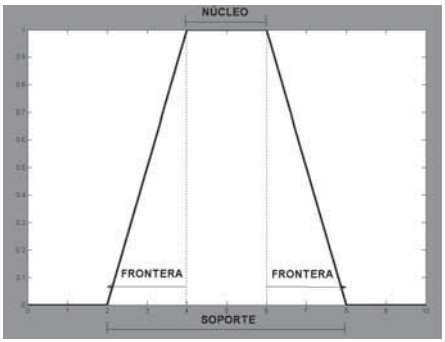


Fig. 2.29 Partes de una función de membresía.

Núcleo: elementos donde $x \rightarrow \mu_{\tilde{A}}(x) = 1$

Soporte: $\mu_{\tilde{A}}(x) > 0$

Fronteras: $0 < \mu_{\tilde{A}}(x) < 1$

Cálculo de función de pertenencia

Las *funciones de pertenencia* pueden calcularse de diversas formas. El método a elegir depende de la aplicación en particular, del modo en que se manifieste la incertidumbre y en el que ésta sea medida durante los experimentos.

1. Método HORIZONTAL:

- Se basa en las respuestas de un grupo de N "expertos".
- La pregunta tiene el formato siguiente:
  "¿Puede x ser considerado compatible con el concepto A?
- Sólo se acepta un "SÍ" o un "NO", de manera que:

$$A(x) = \left(Respuestas \quad Afirmativas \right) \Big/ N$$

2. Método VERTICAL:

- Se escogen varios valores para α, para construir sus $\alpha - cortes$.
- Ahora la pregunta es la siguiente, efectuada para esos valores de α predeterminados: "Identifique los elementos de X que pertenecen a A con grado no menor que α".
- A partir de esos $\alpha - cortes$ se identifica el conjunto difuso A (usando el llamado "principio de identidad" o teorema de representación).

3. Método de comparación de parejas (Saaty, 1980):

- Supóngase que tenemos ya el conjunto difuso A sobre el universo X de n valores $(x_1, x_2, \cdots, x_n,)$.
- Calcular la matriz recíproca $M = [a_{hi}]$, matriz cuadrada nxn:

  a) La diagonal principal es siempre 1.
  b) Propiedad de reciprocidad:

$$a_{hi} a_{ih} = 1$$

  c) Propiedad transitiva:

$$a_{hi} a_{ik} = a_{hk}$$

- El proceso es el inverso:

  — Se calcula la matriz M.
  — Se calcula a partir de M.

$$M = \begin{bmatrix} \frac{A(x_1)}{A(x_1)} & \frac{A(x_1)}{A(x_2)} & \cdots & \frac{A(x_1)}{A(x_n)} \\ \frac{A(x_2)}{A(x_1)} & \frac{A(x_2)}{A(x_2)} & \cdots & \frac{A(x_2)}{A(x_n)} \\ \vdots & \ddots & \frac{A(x_i)}{A(x_j)} & \vdots \\ \frac{A(x_n)}{A(x_1)} & \frac{A(x_n)}{A(x_2)} & \cdots & \frac{A(x_n)}{A(x_n)} \end{bmatrix}$$

- Para calcular M se cuantifica numéricamente el nivel de prioridad o mayor pertenencia de una pareja de valores: x_i con respecto a x_j.

  — Número de comparaciones: $n(n-1)/2$

— La transitividad es difícil de conseguir (el autovalor más grande de la matriz sirve para medir la consistencia de los datos: si es muy bajo, deberían repetirse los experimentos).

4. Método basado en la especificación del problema:

- Requiere una función numérica que quiera ser aproximada.
- El error se define como un conjunto difuso: mide la calidad de la aproximación.

5. Método basado en la optimización de parámetros:

- La forma de un conjunto difuso A depende de unos parámetros denotados por el vector p: representado por $A(x; p)$.
- Se obtiene algunos resultados experimentales en la forma de parejas (elemento, grado de pertenencia): (E_k, G_k) con $k = 1, 2, \ldots, N$.
- El problema consiste en optimizar el vector p, por ejemplo reduciendo el error cuadrático:

$$\min_p \sum_{k=1}^{N} \left[G_k - A\left(E_k; p\right) \right]^2$$

6. Método basado en la agrupación difusa (Fuzzy Clustering):

- Se trata de agrupar los objetos de un universo en grupos (solapados) cuyos niveles de pertenencia a cada grupo son vistos como grados difusos.
- Existen varios algoritmos de Fuzzy Clustering, pero el más aceptado es el algoritmo de "fuzzy isodata" (Bezdek, 1981).

El principio de extensión: generalización

Se puede *generalizar el principio de extensión* para el caso en el que el universo sea el producto cartesiano de n universos:

- $X = X_1 x X_2 x \ldots x X_n$
- La función de transformación: $f : X \to Y, y = f(x)$, con $x = (x_1, x_2, \ldots, x_n)$.
- El principio de extensión transforma n conjuntos difusos $A_1, A_2, \ldots y \quad A_n$, de los universos $X_1, X_2, \ldots y \quad X_n$ respectivamente, en un conjunto difuso $B = f(A_1, A_2, \ldots y \quad A_n)$ definido como:

$$B\left(y\right) = \sup\left\{ \min\left[A_1, A_2, \ldots y \quad A_n \right] \mid x \in X, y = f(x) \right\}$$

Ejemplos: Sean X e Y, ambos, el universo de los números naturales.

Función **sumar 4**: $y = f(x) = x + 4$:

- $A = 0.1/2 + 0.4/3 + 1/4 + 0.6/5;$
- $B = f\left(A\right) = 0.1/6 + 0.4/7 + 1/8 + 0.6/9;$

Función **suma**: $y = f(x_1, x_2) = x_1 + x_2$:

- $A_1 = 0.1/2 + 0.4/3 + 1/4 + 0.6/5;$
- $A_2 = 0.4/5 + 1/6;$
- $B = f\left(A_1, A_2\right) = \{0.1/7 + 0.4/8 + 0.4/9 + 1/10 + 0.6/11\};$

Para formar el 9 en la función suma se puede hacer sumando 5 + 4 y 6 + 3, donde *el mínimo de mín*{5,4} *es* 0.4 *y el mínimo de mín*{6,3} *es* 0.1, *ahora el máximo de ambos será máx*(0.4,0.4) → 0.4

Para formar el 8 en la función suma se puede hacer sumando 5 + 3 y 6 + 2, donde *el mínimo de mín*{5,3} *es* 0.4 *y el mínimo de mín*{6,2} *es* 0.1, *ahora el máximo de ambos será máx*(0.4,0.1) → 0.4

Las operaciones aritméticas con números difusos se pueden encontrar en un intervalo cerrado que cumpla con $\left[a,b\right]*\left[d,e\right]=[f*g \mid a\le f\le b, d\le g\le e]$

Operaciones aritméticas, +, − , / y *.

$\left[a,b\right]*\left[d,e\right]=[a+d,b+e]$

$\left[a,b\right]*\left[d,e\right]=[a-e,b-d]$

$\left[a,b\right]*\left[d,e\right]=[\text{mín}(ad,ae,bd,be),\text{máx}(ad,ae,bd,be)]$

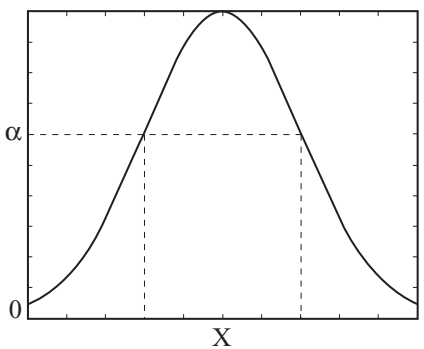
$[a,b]/[d,e]=[a,b]\cdot[1/e \; 1/d]$

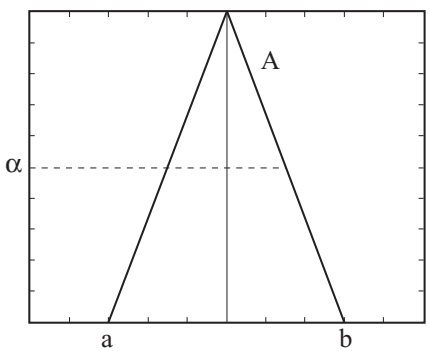
Ejemplo:

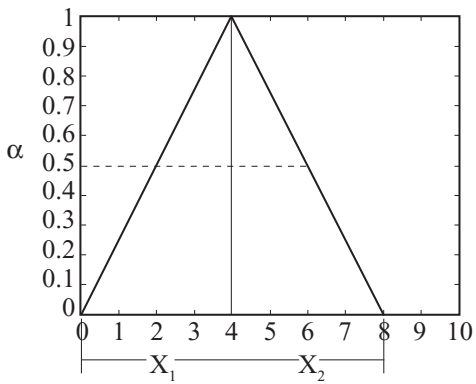
$$\text{Para } A(x)=\begin{cases}0 & para \quad x\le -1 \quad y \quad x>3\\ (x+1)/2 & para \quad -1<x\le 1\\ (3-x)/2 & para \quad 1<x\le 3\end{cases} \qquad \text{Para } B(x)=\begin{cases}0 & para \quad x\le 1 \quad y \quad x>5\\ (x-1)/2 & para \quad -1<x\le 3\\ (5-x)/2 & para \quad 3<x\le 5\end{cases}$$

$$A^{\alpha}=[2\alpha-1,3-2\alpha],\ B^{\alpha}=[2\alpha+1,5-2\alpha]$$

$$(A+B)^{\alpha}=\left[4\alpha,8-4\alpha\right] \quad para \quad \alpha\in(0,1)$$







$$\alpha = \frac{x_1}{4}, \alpha = \frac{-x_2 + 8}{4}$$

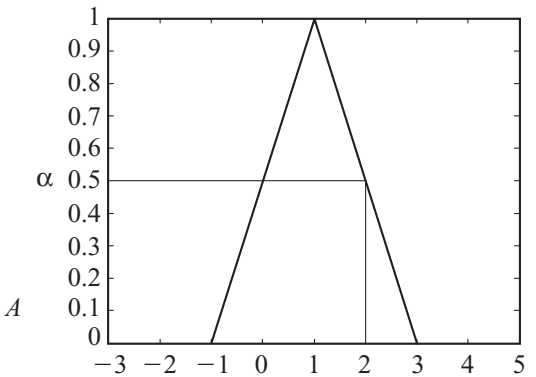
Por lo tanto

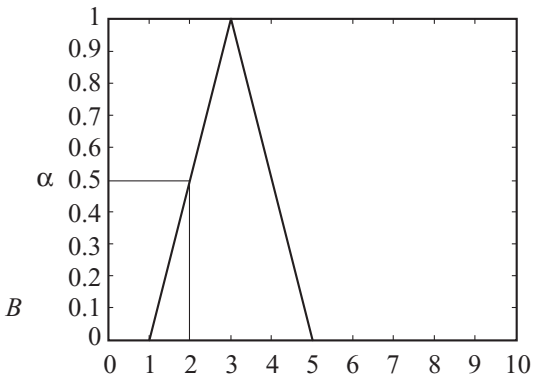
$$(A+B)(x) = \begin{cases} 0 \quad para \quad x \le 0 \quad y \quad x > 8 \quad /4 \quad para \quad 0 < x \le 4 \\ x/4 \quad para \quad 0 < x \le 4 \\ (8-x)/4 \quad para \quad 4 < x \le 8 \end{cases}$$

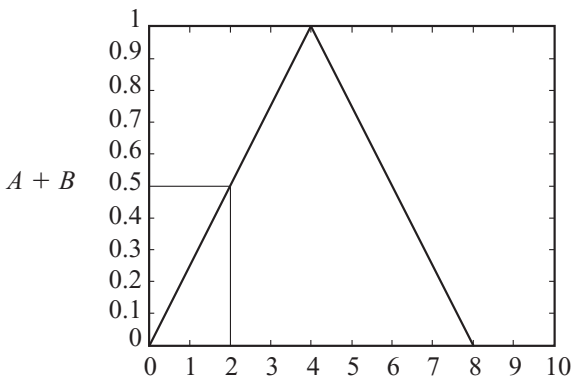
A = *centrado en* 1

B = *centrado en* 3

$(A + B)$ = *centrado en* 4







$$C = \underset{A+B}{\vee}\left[\mu_A(x) \wedge \mu_B(x)\right] = \left[0.5 \wedge 0.5\right] = \wedge\left[0.5\right] = 0.5$$

$$(A+B)^{\alpha} = \left[4\alpha, 8-4\alpha\right] \rightarrow \frac{x}{a} \; para \quad 0 < x \le 4 \quad y \quad para \quad 2 \quad con \quad F.M \quad de \quad 0.5.$$

PRINCIPIO DE EXTENSIÓN

El principio de extensión es una herramienta matemática para extender nociones y operaciones matemáticas del mundo nítido al difuso.

Suponiendo que una función *f* mapea elementos $x_1, \ldots, x_n$ de un universo de discurso *X* a otro universo de discurso *Y*, esto es:

$$\begin{aligned} y_1 &= f(x_1) \\ y_2 &= f(x_2) \\ &\vdots \\ y_n &= f(x_n) \end{aligned}$$

Si tenemos un conjunto difuso $\tilde{A}$ definido en $x_1, \ldots, x_n$:

$$\tilde{A} = \left\{ \frac{\mu_{\tilde{A}}(x_1)}{x_1} + \ldots + \frac{\mu_{\tilde{A}}(x_n)}{x_n} \right\}$$

Entonces:

$$\tilde{B} = f(\tilde{A}) = \left\{ \frac{\mu_{\tilde{A}}(x_1)}{f(x_1)} + \ldots + \frac{\mu_{\tilde{A}}(x_n)}{f(x_n)} \right\}$$

Donde cada imagen de x_i bajo *f*, es decir que $y_i = f(x_i)$ se convierte en difusa con un grado de pertenencia $\mu_{\tilde{A}}(x_i)$.

$$\mu_{\tilde{B}}(y_o) = \mu_{\tilde{A}}(x_2) \vee \mu_{\tilde{A}}(x_3)$$

$$\mu_{\tilde{B}}(y) = \int_{u \times v \times \ldots \times w} \left[\mu_{\tilde{A}1}(u) \wedge \mu_{\tilde{A}2}(u) \wedge \ldots \wedge \mu_{\tilde{A}m}(w) \right] \Big|_{f(u,v,\ldots w)}$$

$$\mu_{\tilde{B}}(y) = \bigvee_{f(.)} \left[\mu_{\tilde{A}}(x) \wedge \mu_{\tilde{B}}(y) \right]$$

NÚMEROS DIFUSOS

Los números difusos deben ser normales y convexos. Son normales si al menos un punto en el universo de discurso de la función de pertenencia alcanza el valor unitario (1). Son convexos si la función de pertenencia de forma gráfica no aumenta y disminuye más de una vez en todo su universo de discurso. Esto es, un número difuso *F* en un universo continuo *U*:

$$\max_{u \in U} \mu_F(u) = 1 \qquad \text{(normal)}$$

$$\mu_F(\lambda u_1 + (1-\lambda)u_2) \geq \min(\mu_F(u_1), \mu_F(u_2)) \qquad \text{(convexo)}$$

$$u_1, u_2 \in U, \ \lambda \in [0,1]$$

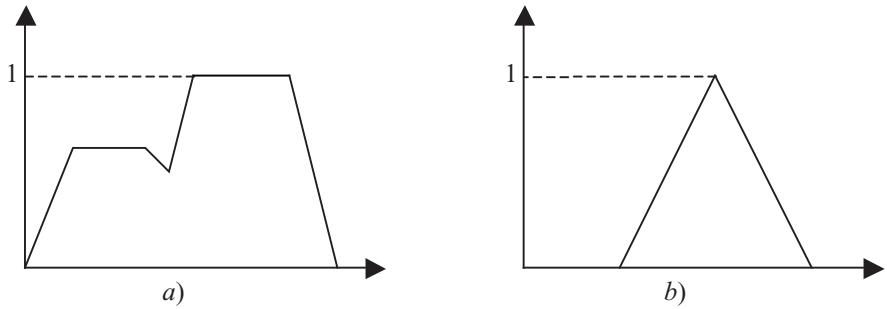


Fig. 2.30 *a*) No es convexo porque la función aumenta y disminuye más de una vez, sin embargo es normal. *b*) Es normal y convexo, y por lo tanto un número difuso.

Un número difuso puede representar valores numéricos para expresiones como: "aproximadamente", "cercano a" y "casi".

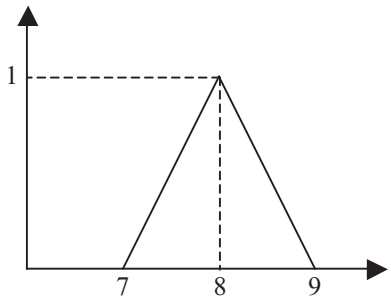


Fig. 2.31 Número difuso "casi" 8.

En los números difusos es posible realizar distintas operaciones como suma, resta, multiplicación y división, empleando el principio de extensión.

Suma de números difusos

A partir del uso de números difusos y del principio de extensión, es posible realizar diversas relaciones como la suma entre los mismos. Para demostrar esto se ha empleado un ejemplo con dos funciones de pertenencia triangulares que se muestran en las figuras 2.32 y 2.33, las cuales representan los números difusos 5 y 8, respectivamente.

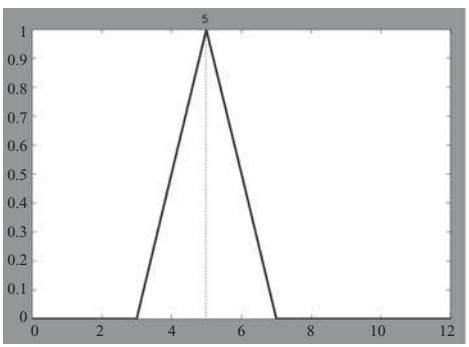


Fig. 2.32 Número difuso 5.

La suma de dos números difusos $\tilde{A}$ y $\tilde{B}$, de tal forma que $\tilde{A} + \tilde{B} = \tilde{C}$ es una relación de la siguiente forma:

$$\tilde{C} = \bigvee_{\tilde{A}+\tilde{B}} \left[\mu_{\tilde{A}}(x) \wedge \mu_{\tilde{B}}(y) \right]$$

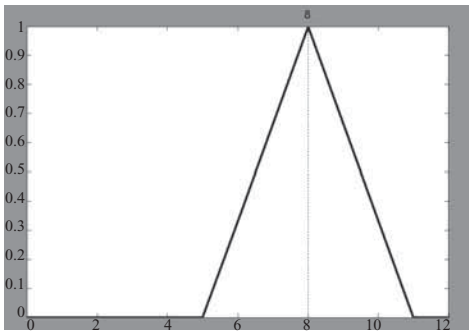


Fig. 2.33 Número difuso 8

Para realizar la operación se generó una tabla que relaciona la función de pertenencia del número 5 con la del número 8, la cual se muestra en la figura 2.34.

		B											
		1	2	3	4	5	6	7	8	9	10	11	12
A	1	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	2	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	3	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	4	0.5, 0	0.5, 0	0.5, 0	0.5, 0	0.5, 0	0.5, 0.3	0.5, 0.7	0.5, 1	0.5, 0.7	0.5, 0.3	0.5, 0	0.5, 0
	5	1, 0	1, 0	1, 0	1, 0	1, 0	1, 0.3	1, 0.7	1, 1	1, 0.7	1, 0.3	1, 0	1, 0
	6	0.5, 0	0.5, 0	0.5, 0	0.5, 0	0.5, 0	0.5, 0.3	0.5, 0.7	0.5, 1	0.5, 0.7	0.5, 0.3	0.5, 0	0.5, 0
	7	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	8	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	9	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	10	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	11	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0
	12	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0.3	0, 0.7	0, 1	0, 0.7	0, 0.3	0, 0	0, 0

Fig. 2.34 Tabla para $\tilde{A} + \tilde{B} = \tilde{C}$.

Para ejemplificar que el programa es correcto, se probó para el número 10, teniendo las combinaciones de $\tilde{A}$ y $\tilde{B}$: 1 y 9, 2 y 8, 3 y 7, 4 y 6, 5 y 5, 6 y 4, 7 y 3, 8 y 2, 9 y 1. Con esto se realiza el mínimo de los dos valores de cada casilla de la figura 2.34 correspondiente a las combinaciones mencionadas:

$$\mu_{\tilde{A}}(1) \wedge \mu_{\tilde{B}}(9) = 0 \wedge 0.7 = 0$$

$$\mu_{\tilde{A}}(2) \wedge \mu_{\tilde{B}}(8) = 0 \wedge 1 = 0$$

$$\mu_{\tilde{A}}(3) \wedge \mu_{\tilde{B}}(7) = 0 \wedge 0.7 = 0$$

$$\mu_{\tilde{A}}(4) \wedge \mu_{\tilde{B}}(6) = 0.5 \wedge 0.3 = 0.3$$

$$\mu_{\tilde{A}}(5) \wedge \mu_{\tilde{B}}(5) = 1 \wedge 0 = 0$$

$$\mu_{\tilde{A}}(6) \wedge \mu_{\tilde{B}}(4) = 0.5 \wedge 0 = 0$$

$$\mu_{\tilde{A}}(7) \wedge \mu_{\tilde{B}}(3) = 0 \wedge 0 = 0$$

$$\mu_{\tilde{A}}(8) \wedge \mu_{\tilde{B}}(2) = 0 \wedge 0 = 0$$

$$\mu_{\tilde{A}}(9) \wedge \mu_{\tilde{B}}(1) = 0 \wedge 0 = 0$$

Finalmente, se obtiene el máximo de los resultados anteriores, por lo que $\tilde{C} = 0.3$.

RELACIONES NÍTIDAS Y DIFUSAS

Producto cartesiano

Como se mencionó, la lógica difusa se basa en relaciones entre nítidos que se puede expandir a conjuntos difusos, los cuales pueden tener como base el producto cartesiano, que se puede definir a través de la colección de elementos que confirman un conjunto y se relacionan de manera directa con elementos de otro conjunto.

Siendo el producto cartesiano una relación directa entre los dos conjuntos, sean nítidos o difusos, entendiendo por producto cartesiano ($X \times Y$) de dos conjuntos X y Y, un conjunto de todos los pares ordenados en los que el primer componente está contenido en el primer conjunto y el segundo componente pertenece al segundo (pertenece a X y el segundo a Y):

$$X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}$$

Un par ordenado se tiene cuando dos elementos pertenecen a una relación; pueden existir relaciones binarias, ternarias hasta llegar a n-arias. En el producto cartesiano, se toma cada uno de los elementos y se le relaciona con todos los demás. Sin embargo, deben considerarse ciertas restricciones, como por ejemplo que el producto cartesiano no es conmutativo.

Relaciones nítidas

Una relación es una correspondencia. En una relación convencional (nítida) si existe la relación es de 1; si no, es 0.

Algunos ejemplos de relaciones nítidas si se tiene un conjunto X y un conjunto Y formados por distintos elementos, son:

Fig. 2.35 Ejemplos de relaciones.

Otra forma de escribir las relaciones son:

$$X \times Y = \{(X, Y) \mid X \in X, Y \in Y\}$$

$$X_{X \times Y} = \begin{cases} 1, (X, Y) \in X \times Y \\ 0, (X, Y) \notin X \times Y \end{cases}$$

En las relaciones nítidas, si se desea relacionar elementos del conjunto X con el conjunto Y, se debe de realizar un mapeo por medio de una función. Entiéndase por mapeo el pasar de un elemento de un conjunto a su equivalente en otro conjunto.

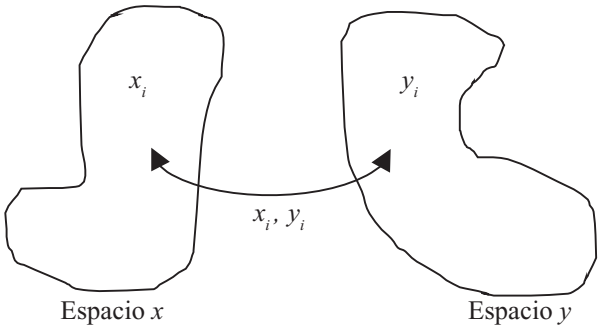


Fig. 2.36 Mapeo.

Ejemplo

$$A = \{0,1\} \qquad B = \{a,b,c\}$$
$$A \times B = \{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$$
$$B \times A = \{(a,0),(a,1),(b,0),(b,1),(c,0),(c,1)\}$$
$$A \times A = A^2 = \{(0,0),(0,1),(1,0),(1,1)\}$$

Relaciones difusas

Las relaciones difusas siguen ciertas características que permiten establecer diferentes grados de valor relación en cada una de ellas. Por ejemplo, en la naturaleza existen relaciones en que sólo animales de la misma especie pueden cruzarse teniendo relaciones restringidas y en ocasiones no restringidas, clasificándose éstas en los conjuntos nítidos, mientras que en las relaciones difusas existen valores entre 0 y 1 que establecen el valor de la relación.

Composición

La composición es una relación usada en la lógica difusa. Se basa en los máximos y mínimos. Haciendo una analogía con una serie de cadenas como la que muestra la figura 2.37, en la composición se busca la cadena más fuerte y la más débil.

Fig. 2.37 Cadenas.

Las cadenas representan las posibles trayectorias del punto A al punto B, es decir los posibles nexos entre un elemento y otro. El eslabón más débil se determina con los mínimos; el eslabón más fuerte se encuentra con los máximos.

El objetivo de la composición es relacionar conjuntos que aparentemente no tienen relación. En la lógica difusa esto tiene mucho sentido pues hay diferentes ponderaciones o grados de pertenencia.

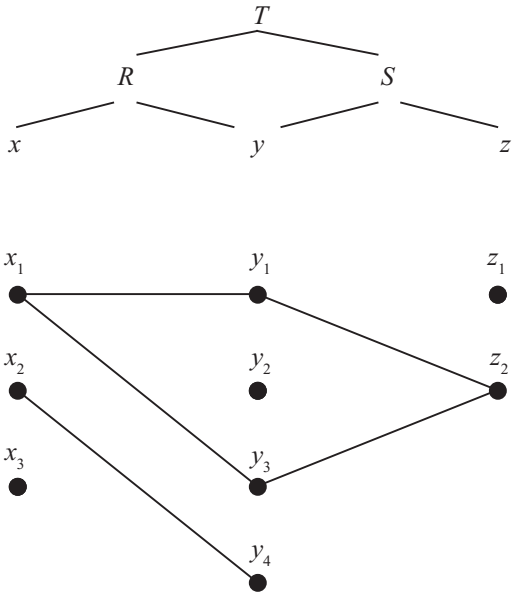


Fig. 2.38 Diagrama de la composición de T.

Analizando esto con conjuntos:

$$T = R \cdot S$$

Donde T es la composición de R con S

$$X_T(x,z) = \vee_{y \in Y}(X_R(x,y) \wedge X_S(y,z))$$

La oración anterior encierra la forma en que se hace la composición. Quiere decir que se toman los valores mínimos de los conjuntos "pequeños", y una vez que se obtuvieron estos valores, de éstos se determina el máximo, siendo este valor final la composición.

A continuación se ejemplifica este enunciado. Para una explicación más detallada del mismo véase el anexo A.

Se determinará, con base en el diagrama anterior (diagrama de la composición T), el conjunto:

$$\mu(x_1, z_1)$$

Primero se determinan las matrices:

$$R = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$T = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cc} z_1 & z_2 \end{array} \\ \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array} \qquad S = \begin{array}{c} \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{array} \begin{array}{c} \begin{array}{cc} z_1 & z_2 \end{array} \\ \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \end{array}$$

Con base en la composición:

$$X_T(x,z) = \vee_{y \in Y}(X_R(x,y) \wedge X_S(y,z))$$

Se acomodan los términos de la siguiente manera:

$$\mu(x_1,z_1) = \text{máx}[\text{mín}((x_1,y_1),(x_1,z_1)), \text{mín}((x_1,y_2),(y_2,z_1)), \text{mín}((x_1,y_3),(y_3,z_1)), \text{mín}((x_1,y_4),(y_4,z_1))\ ]$$

Esto con el propósito de encontrar la relación o el camino que hay entre x_1 y z_1. Debido a que no están relacionados de una manera directa, deben pasar antes por su relación común, en este caso "y".

Finalmente se sustituyen valores y se determina que:

$$\mu(x_1,z_1) = \text{máx}\left[1\right] = 1$$

Aplicando esto a un ejemplo difuso:

A continuación se determina, con base en las matrices R, S, la composición T.

$$R = \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cc} y_1 & y_2 \end{array} \\ \begin{bmatrix} 0.7 & 0.5 \\ 0.8 & 0.4 \end{bmatrix} \end{array}$$

$$S = \begin{array}{c} \\ y_1 \\ y_2 \end{array} \begin{array}{c} \begin{array}{ccc} z_1 & z_2 & z_3 \end{array} \\ \begin{bmatrix} 0.9 & 0.6 & 0.2 \\ 0.1 & 0.7 & 0.5 \end{bmatrix} \end{array}$$

Solución:

$$T = \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} z_1 & z_2 & z_3 \end{array} \\ \begin{bmatrix} 0.7 & 0.6 & 0.5 \\ 0.8 & 0.6 & 0.4 \end{bmatrix} \end{array}$$

$$\mu_T(x,y) = \vee\left(\mu(x,y) \bullet \mu(y,z)\right)$$

Para la composición, máx- composición del producto:

$$\mu_T(x_2,z_2) = \text{máx}\left(\left(0.8 \cdot 0.6\right),\left(0.4 \cdot 0.8\right)\right) = 0.48$$

$$T = \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} z_1 & z_2 & z_3 \end{array} \\ \begin{bmatrix} 0.63 & 0.42 & 0.25 \\ 0.72 & 0.48 & 0.20 \end{bmatrix} \end{array}$$

Hasta este punto se ha planteado algunas relaciones difusas, con el propósito de poder establecer la fuzzificación, la cual, como se verá más adelante, es una de las etapas del control difuso en la que se pasa de valores reales a valores difusos mediante funciones de membresía.

Una relación más real sería una analogía con un motor. Suponiendo que la relación que existe entre la resistencia de la planta y la corriente es "R", y suponiendo que la relación entre la corriente y la velocidad es "S", la relación entre la resistencia y la velocidad no es directa, pero se puede determinar por medio de la función composición.

Ejemplo

Para un motor se sabe que:

$$R = R_s \times I,\ S = I \times N,\ T = R_s \circ N$$

Considerando las mediciones:

$$R_s = \left\{ \frac{0.3}{30} + \frac{0.7}{60} + \frac{1}{100} + \frac{0.2}{120} \right\}$$

$$I = \left\{ \frac{0.2}{20} + \frac{0.4}{40} + \frac{0.6}{60} + \frac{0.8}{80} + \frac{1}{100} + \frac{0.1}{120} \right\}$$

$$N = \left\{ \frac{0.33}{500} + \frac{0.67}{1000} + \frac{1}{1500} + \frac{0.15}{1800} \right\}$$

Se obtienen las matrices:

$$R = \begin{array}{c} \\ 30 \\ 60 \\ 100 \\ 120 \end{array} \begin{array}{c} \begin{array}{cccccc} 20 & 40 & 60 & 80 & 100 & 120 \end{array} \\ \begin{bmatrix} 0.2 & 0.3 & 0.3 & 0.3 & 0.3 & 0.1 \\ 0.2 & 0.4 & 0.6 & 0.7 & 0.7 & 0.1 \\ 0.2 & 0.4 & 0.6 & 0.8 & 1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.1 \end{bmatrix} \end{array}$$

$$S = \begin{array}{c} \\ 20 \\ 40 \\ 60 \\ 80 \\ 100 \\ 120 \end{array} \begin{array}{c} \begin{array}{cccc} 500 & 1000 & 1500 & 1800 \end{array} \\ \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.15 \\ 0.33 & 0.4 & 0.4 & 0.15 \\ 0.33 & 0.6 & 0.6 & 0.15 \\ 0.33 & 0.67 & 0.8 & 0.15 \\ 0.33 & 0.67 & 1 & 0.15 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix} \end{array}$$

$$T = \begin{array}{c} \\ 30 \\ 60 \\ 100 \\ 120 \end{array} \begin{array}{c} \begin{array}{cccc} 500 & 1000 & 1500 & 1800 \end{array} \\ \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0.15 \\ 0.33 & 0.67 & 0.7 & 0.15 \\ 0.33 & 0.67 & 1 & 0.15 \\ 0.2 & 0.2 & 0.2 & 0.15 \end{bmatrix} \end{array}$$

Composición sup-estrella

Siendo R y S relaciones difusas en $U \times V$ y $V \times W$, respectivamente, la composición de R y S es una relación difusa $R \circ S$ definida por:

$$R \circ S = \left\{ \left[(u, w), \sup(\mu_R(u, v) * \mu_S(v, w)) \right],\ u \in U,\ v \in V,\ w \in W \right\}$$

Donde * puede ser cualquier operador dentro de las normas triangulares, es decir: mínimo, producto algebraico, producto drástico o producto relacionado.

Operaciones con relaciones difusas

Unión

$$\mu_{R\cup S} = \text{máx}\left(\mu_R(x,y), \mu_S(x,y)\right)$$

Intersección

$$\mu_{R\cap S} = \text{mín}\left(\mu_R(x,y), \mu_S(x,y)\right)$$

Complemento

$$\mu_R(x,y) = 1 - \mu_R(x,y)$$

Producto cartesiano difuso y composición

$$\mu_R(x,y) = \mu_{A\times B}(x,y) = \text{mín}\left(\mu_A(x), \mu_B(y)\right)$$

Ejemplo

Encontrar $T = R \cdot S$

$$A = \left\{\frac{0.2}{x_1} + \frac{0.5}{x_2} + \frac{1}{x_3}\right\} \quad y \quad B = \left\{\frac{0.3}{y_1} + \frac{0.9}{y_2}\right\}$$

Reglas difusas

Ahora viene la segunda etapa del proceso, es decir: crear un conjunto de reglas que permitan utilizar las composiciones que se han realizado. Las reglas son los conectores que unen los antecedentes con la conclusión.

Analizando primero las reglas que existen en lógica convencional:

Si se tienen:

Premisa 1	T(P)=1…..(1)
Premisa 2	T(P)=0…..(2)
----------	----------

Se genera:

Conclusión	T: u→{0,1}

Conectores:

Tabla 2.5 Conectores.

V	Unión
A	Intersección
- o ~	Negación
→	Implicación
↔	Equivalencia

Su aplicación en tablas de verdad:

Tabla 2.6 Tabla de verdad con reglas difusas.

P	Q	$P \rightarrow Q$	$P \leftrightarrow Q$
V	V	V	V
V	F	F	F
F	V	V	F
F	F	V	V

Modus ponens y modus tollens

Las reglas pueden formularse en dos modos denominados *modus ponens* y *modus tollens*, respectivamente. El *modus ponens* parte de los antecedentes para encontrar la consecuencia, y el *modus tollens* parte de la consecuencia recíproca para determinar los antecedentes.

Modus ponens:

$A \rightarrow B$

A

B

Modus tollens:

$A \rightarrow B$

$\overline{B}$

$\overline{A}$

"Tautología" (*modus ponens*)

A	B	$A \rightarrow B$	$(A \wedge (A \rightarrow B))$	$(A \wedge (A \rightarrow B)) \rightarrow B$
0	0	1	0	1
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

"Tautología" (*modus tollens*)

A	B	$A \rightarrow B$	$(A \wedge (A \rightarrow B))$	$(A \wedge (A \rightarrow B)) \rightarrow B$
1	1	1	0	1
1	0	0	0	1
0	1	1	0	1
0	0	1	1	1

Lógica difusa

Proposición $\rightarrow$ P

$T : u \in U \rightarrow \{0,1\}$

$T(P) = \mu_A(x)$ *donde* $0 \leq \mu_A \leq 1$

Negación $T(\bar{P}) = 1 - T(\bar{P})$

Disyunción $P \vee Q$: x *es* A *or* B

Conjunción $P \wedge Q$: x *es* A *and* B

Implicación $P \rightarrow Q$: x *es* A, *entonces* x *es* B

$$T(P \rightarrow Q) = \text{máx}\left(T(\bar{P}), T(Q)\right)$$

$$T(P \vee Q) = \text{máx}\left(T(\bar{P}), T(Q)\right)$$

$$T(P \wedge Q) = \text{mín}\left(T(\bar{P}), T(Q)\right)$$

"Quasi-tautología" (*modus ponens*)

A	B	$A \rightarrow B$	$(A \wedge (A \rightarrow B))$	$(A \wedge (A \rightarrow B)) \rightarrow B$
0.3	0.2	0.7	0.3	0.7
0.3	0.8	0.8	0.3	0.8
0.7	0.2	0.3	0.3	0.7
0.7	0.8	0.8	0.7	0.8

"Quasi-tautología" (*modus ponens*)

A	B	$A \rightarrow B$	$(A \wedge (A \rightarrow B))$	$(A \wedge (A \rightarrow B)) \rightarrow B$
0.3	0.2	0.7	0.7	0.7
0.3	0.8	0.8	0.2	0.8
0.7	0.2	0.3	0.3	0.7
0.7	0.8	0.8	0.2	0.8

CONTROLADORES DIFUSOS

Un controlador difuso está compuesto de cuatro partes principales: interfaz de difusificación, base de conocimientos, lógica de decisiones e interfaz de desdifusificación, las cuales se detallan a continuación.

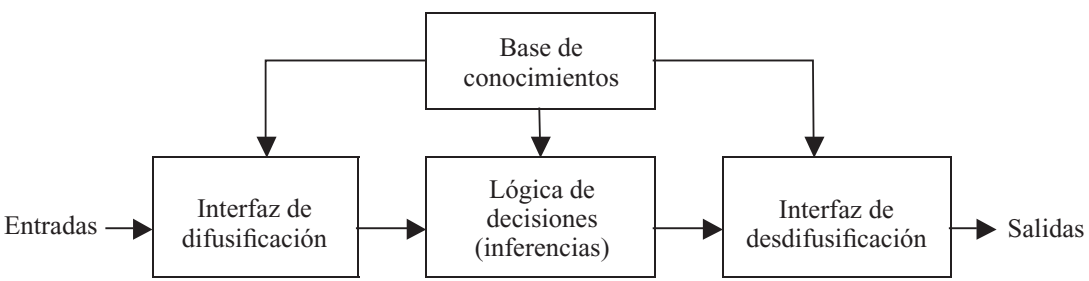


Fig. 2.39 Estructura de un controlador difuso.

Interfaz de difusificación

Mide los valores de las variables de entrada para realizar un mapeo a escala que transfiere el rango de valores de las variables a un universo de discurso difuso. La difusificación convierte los datos de entrada en valores lingüísticos que son las etiquetas de las funciones de pertenencia o conjuntos difusos.

La representación de información a través de conjuntos difusos puede realizarse en forma discreta. Al "discretizar" información, es decir, segmentar un universo en un número definido de partes, es posible definir un conjunto difuso asignando un grado de pertenencia a cada elemento genérico del nuevo universo discreto. Los niveles de "discretización" tienen una gran influencia en la obtención de un control. Para la "discretización" es necesario realizar un mapeo a escala para transformar valores medidos en las variables a valores del universo discreto, ya sea de manera uniforme o no uniforme, o combinaciones de ambos.

Una variable lingüística en general se asocia a un conjunto de términos, definido en el mismo universo de discurso. Para encontrar cuántos términos son necesarios en un conjunto se emplean particiones difusas. El número de conjuntos difusos determina la complejidad del controlador, y éstos tienen un significado lingüístico como "negativo grande", "cero", "positivo pequeño". La figura 2.40 muestra ejemplos de dos particiones difusas en el mismo universo, normalizado de -1 a $+1$.

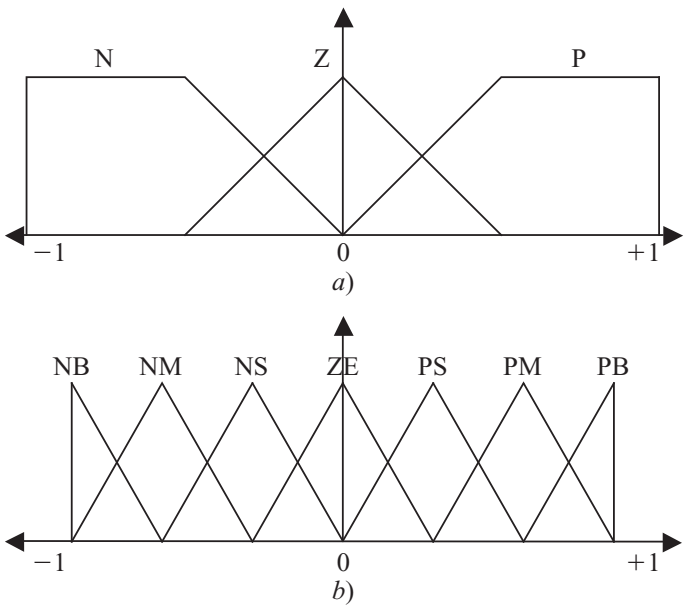


Fig. 2.40 Particiones difusas con distinto número de términos: *a*) tres términos *N, Z* y *P*; *b*) siete términos *NB, NM, NS, ZE, PS, PM* y *PB*.

Base de conocimientos

La base de conocimientos contiene toda la información de la aplicación que se va a controlar, así como las metas del controlador. Consta de una base de datos y una base de reglas lingüísticas para controlar la variable. La base de datos proporciona las definiciones para el establecimiento de reglas y la manipulación de datos difusos. La base de reglas caracteriza las metas de control y la política que utilizan los expertos para llevar a cabo el control, empleando proposiciones.

Un algoritmo de control difuso debe ser capaz de inferir una acción de control correspondiente para cada estado del proceso que se va a controlar, propiedad que se denomina *unidad*. La estrategia de la base de datos comprende los soportes de la definición de los conjuntos difusos.

Existen cuatro modos de derivación de las reglas difusas de control, las cuales contemplan la experiencia de expertos, el conocimiento de ingeniería de control y las acciones de control de un operador, con la forma de proposiciones condicionales que relacionan las variables de estado en el antecedente del proceso con las variables de control del proceso en las consecuencias.

El conjunto de reglas se define como:

$$(\divideontimes_1 \text{ es } L_{X_1}{}^{(k)} \text{ y } (\ldots) \ldots (\ldots) \text{ y } (\divideontimes_n \text{ es } L_{\divideontimes_n}{}^{(k)}), \text{ k} = 1,2,\ldots, \text{ m}$$

Donde:

X : Dominio físico actual donde tienen sentido los valores lingüísticos.

L_X : Conjunto de valores lingüísticos que $\divideontimes$ puede tomar, un elemento arbitrario es $L_{\divideontimes}$.

$\divideontimes$: Nombre simbólico.

μ_x : Función semántica de interpretación de un valor lingüístico en términos cuantitativos.

$$\mu_x : L_{\divideontimes} \rightarrow \tilde{L}_X.$$

Lógica de decisiones

La lógica utilizada para tomar decisiones dentro de un controlador difuso es el núcleo del mismo. A partir de la misma se simula la lógica que utilizan las personas para tomar decisiones, con base en conceptos difusos y en la inferencia de acciones de control, empleando implicaciones y las reglas establecidas según la base de conocimientos.

El resultado de disparar estas reglas con valores de entrada físicos nítidos X_1^k, X_2^k,..., X_n^k será m conjuntos difusos recortados, los cuales se denotarán por:

$$C_{\tilde{L}u}{}^{(1)}, \ldots\ldots, C_{\tilde{L}u}{}^{(m)}$$

O m conjuntos escalados:

$$S_{\tilde{L}u}{}^{(1)}, \ldots\ldots, S_{\tilde{L}u}{}^{(m)}$$

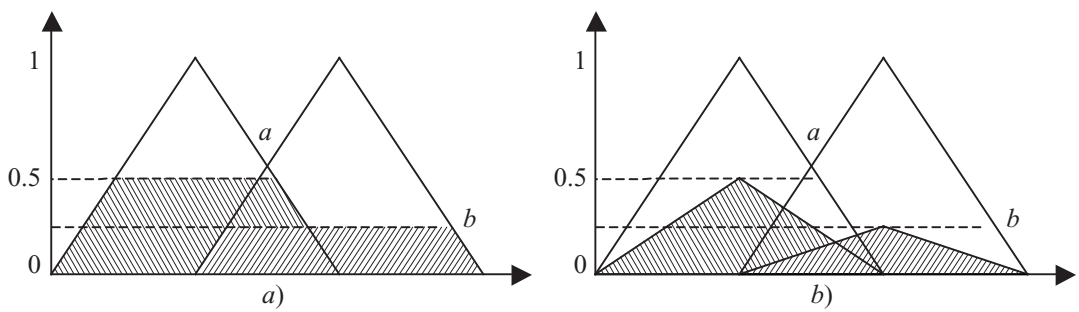


Fig. 2.41 *a*) Conjuntos recortados y *b*) conjuntos escalados.

Interfaz de desdifusificación

La interfaz de desdifusificación se encarga del mapeo a escala que convierte el rango de valores de las variables de salida a sus universos de discurso correspondientes. La desdifusificación es la herramienta para obtener la acción de control nítida a partir de una acción de control difusa.

La salida de control $\tilde{u}$ o μ_u se obtiene como la unión de las salidas de control cortadas o escalonadas:

$$\tilde{u} = \underset{k=1}{\overset{m}{u}} c\tilde{L}u^k$$

El valor nítido se denota por u^* y el área del conjunto se define por:

$$\int_u \mu_u(u)du$$

Algunos métodos para encontrar el valor nítido son los siguientes:

Método de centro de área o gravedad

El método del centro de área o centro de gravedad se puede representar en forma discreta:

$$Salida = \frac{\sum_{x=a}^{b} \mu(x) \cdot x}{\sum_{x=a}^{b} \mu(x)}$$

o continua:

$$Salida = \frac{\int_a^b \mu(x) \cdot x dx}{\int_a^b \mu(x) \cdot dx}$$

Este método se utiliza para obtener el valor real de la salida. Su metodología es sencilla: corta la función de membresía al grado de la membresía respectiva, es decir, segmenta las funciones de membresía, generando en cada función dos áreas. El área inferior que se forma es la que se toma para hacer el cálculo. Se sobreponen todas estas áreas y se saca el centroide de la superposición, el cual nos dice la salida real del sistema. Para explicar mejor esto, se analizará un ejemplo.

La variable lingüística de salida es la velocidad de un ventilador. Los valores positivos son giros a la derecha y los negativos son a la izquierda. Las unidades del universo discurso están dadas en m/seg. Este ventilador tiene como salidas difusas 0.75 y 0.25.

En la figura 2.42 se ven las funciones de membresía, así como las proyecciones.

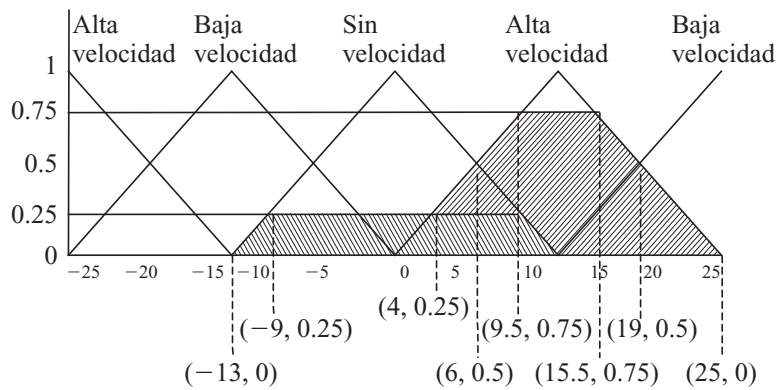


Fig. 2.42 Corte de la función de membresía con el grado de membresía respectivo.

Ahora bien, sacando el centroide de esta área (el área rayada) se obtiene el valor real de la salida. Para sacar el área se utilizan los puntos más significativos de la misma. En este caso, están señalados con las líneas y sus coordenadas.

$$Salida = \frac{\sum_{x=a}^{b} \mu(x) \cdot x}{\sum_{x=a}^{b} \mu(x)} = \frac{(0 * -13) + (0.25 * -9) + (0.25 * 4) + (0.5 * 6) + (0.75 * 9.5) + (0.75 * 15.5) + (0.5 * 19) + (0 * 25)}{(0 + 0.25 + 0.25 + 0.5 + 0.75 + 0.75 + 0.5 + 0)}$$

$$Salida = \frac{\sum_{x=a}^{b} \mu(x) \cdot x}{\sum_{x=a}^{b} \mu(x)} = 10$$

Por lo tanto, el centroide está en 10 y la salida será de 10.

Método de centro máximo

Este método es un método simplificado. Primero se calcula el valor más común de cada etiqueta; esto se hace con el máximo de la respectiva función de membresía. Ahora bien, en el caso de las funciones trapezoidales se debe escoger el medio maximizado de dicha función.

Ya con estos valores típicos se les asigna un peso o valor, el cual es proporcional al grado de membresía, y la salida se determina haciendo un balance de los distintos valores.

A continuación se verá este método aplicado al ejemplo anterior.

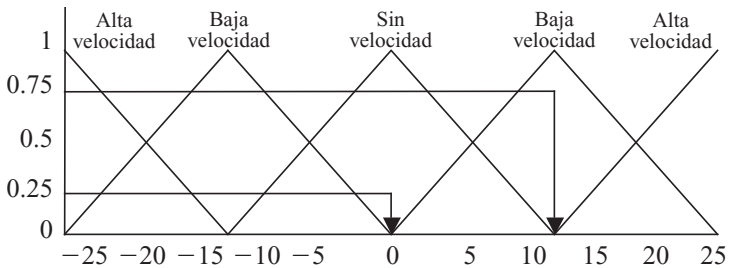


Fig. 2.43 Método del centro máximo.

Como se aprecia en la figura 2.43, el grado de membresía de 0.75 es el que tiene mayor peso y su valor típico es de 12.5 m/seg. El otro grado de membresía de 0.25, su valor típico es de cero, por lo que el cálculo de este centroide quedaría:

$$Salida = \frac{(0.75 * 12.5) + (0 * 0.25)}{(0.25 + 0.75)} = 9.375$$

Sin embargo, este método se aplica más cuando se utilizan funciones de membresía trapezoidales, en cuyo caso se emplea por lo general el valor típico de la mitad del máximo de la función.

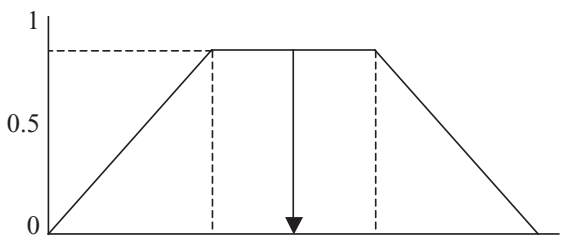


Fig. 2.44 Método del centro máximo de una función trapezoidal.

Método de izquierda máximo

El valor de salida real se obtiene de la misma manera que el método máximo, pero en vez de usar el valor más común de la mitad del máximo, se utiliza el máximo izquierdo.

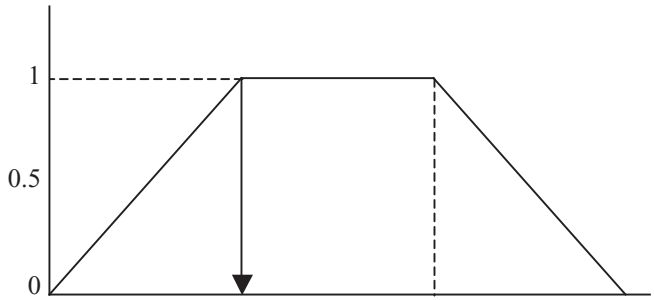


Fig. 2.45 Método de izquierda máximo de una función trapezoidal.

Método de derecha máximo

El valor de salida real se obtiene de la misma manera que el método máximo, pero en vez de usar el valor más común de la mitad del máximo, se utiliza el máximo derecho.

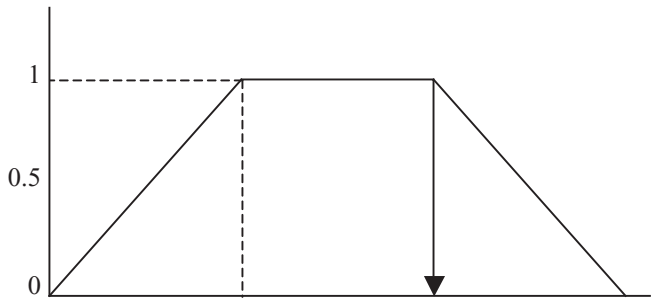


Fig. 2.46 Método de derecha máximo de una función trapezoidal.

Los parámetros principales de diseño de un controlador difuso se enlistan a continuación, no obstante no existen procedimientos sistemáticos para el diseño del controlador.

1. Estrategias de difusificación y la interpretación de un operador de difusificación.
2. Bases de datos: dominio discreto, normalización, partición del espacio, elección de las funciones de pertenencia.
3. Base de reglas: elegir las variables de entrada y las de salida, fuente y derivación de las reglas, consistencia, interactividad y abarcar todos los casos.
4. Lógica de decisiones: definición de las implicaciones difusas, interpretación de los conectores, definición de las composiciones, mecanismo de inferencia.
5. Estrategias de desdifusificación y la interpretación de un operador de desdifusificación.

Los controladores difusos (FLC, por las siglas en inglés de Fuzzy Logic Controller) pueden clasificarse en clásicos, con PID convencional como respaldo, sintonizador de PID convencional, programador de ganancias para PID. Todas estas topologías se estudiarán a continuación con más detalle.

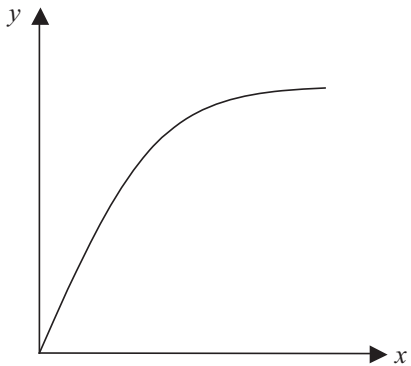


Fig. 2.47 Sistema no lineal.

Para diseñar un sistema difuso, por ejemplo del sistema de la figura 2.47, el primer paso consiste en la definición de las funciones de pertenencia de los espacios de entrada y salida, denominados *x* y *y* para el sistema no lineal. Partiendo de los puntos arbitrarios de la figura 2.48, para el eje *x* se seleccionan funciones de pertenencia cuyos máximos (con pertenencia igual a 1) corresponden a los puntos. Asimismo, en el eje *y* también se seleccionan funciones de pertenencia con máximos en las intersecciones.

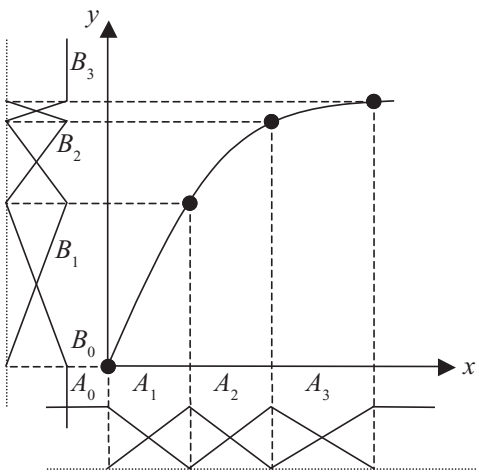


Fig. 2.48 Definición de las funciones de pertenencia para la entrada y la salida.

Finalmente se eligen las reglas para la aproximación, establecidas usualmente de la siguiente forma, para obtener la aproximación que se ve en la figura 2.49:

Si *x* pertenece al dominio A_0, entonces *y* pertenece al dominio B_0

Si *x* pertenece al dominio A_1, entonces *y* pertenece al dominio B_1

Si *x* pertenece al dominio A_2, entonces *y* pertenece al dominio2 B_2

Si *x* pertenece al dominio A_3, entonces *y* pertenece al dominio B_3

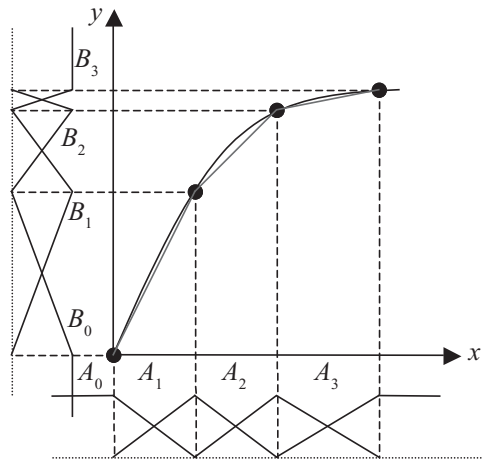


Fig. 2.49 Aproximación de una función mediante un sistema difuso.

Ejemplo

Suponiendo que se desea aproximar una función seno por medio de lógica difusa, a continuación se hará el desarrollo para la solución de este problema, con base en la metodología propuesta.

Definición de las entradas y salidas del sistema

Para hacer la aproximación es necesario hacer un muestreo de distintos puntos que sean parte de la función seno. Esto con el propósito de tomar valores representativos por medio de los cuales se pueda aproximar la función. Es obvio que entre más valores de muestra, mayor será la calidad de la aproximación a la función, sin embargo también se volverá más complejo el sistema y el objetivo de usar la lógica difusa en este problema es simplificar lo más posible el modelo que se va a emplear para el desarrollo de la solución.

Para definir las entradas y salidas, se debe analizar lo que se desea alcanzar, en este caso la función seno; por lo tanto, con base en la grafica de seno es posible observar puntos clave que describen la forma básica. Esto se puede apreciar más en la gráfica, en donde el eje *x* muestra los distintos valores de π, y el eje *y* muestra los radianes:

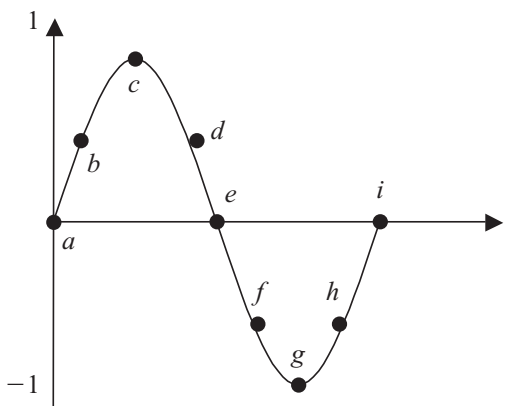


Fig. 2.50 Dibujo de la función seno.

Con base en la gráfica de seno se escogieron los puntos característicos de una manera arbitraria. Estos puntos están marcados con letras en la figura 2.50. Después se procedió a realizar una tabla en la cual se muestran dichos puntos. Ahora se establecen las entradas; en este caso se optó por establecer como entradas los distintos valores de p y como salidas los valores de rad. Para simplificar se decidió poner etiquetas para cada entrada y salida. A continuación se muestra la tabla.

Tabla 2.7 Selección de etiquetas de entrada y salida para el modelo difuso de la función seno.

Puntos	Entradas	Salidas	Etiq.Entrada	Etiq.Salida
a	0	0	E1	S5
b	$\pi/4$	0.7071	E2	S4
c	$\pi/2$	1	E3	S3
d	$(3/4)\,\pi$	0.7071	E4	S4
e	π	0	E5	S5
f	$(5/4)\,\pi$	-0.7071	E6	S2
g	$(3/2)\,\pi$	-1	E7	S1
h	$(7/4)\,\pi$	-10.7071	E8	S2
i	2π	0	E9	S5

Como se puede apreciar, se necesita nueve funciones de entrada y sólo cinco funciones (en este caso serán constantes, aunque hay casos en los que no lo son) de salida, ya que se repiten.

Reglas lingüísticas que regirán el sistema

Si la entrada es E1 entonces la salida es S5
Si la entrada es E2 entonces la salida es S4
Si la entrada es E3 entonces la salida es S3
Si la entrada es E4 entonces la salida es S4
Si la entrada es E5 entonces la salida es S5
Si la entrada es E6 entonces la salida es S2
Si la entrada es E7 entonces la salida es S1
Si la entrada es E8 entonces la salida es S2
Si la entrada es E9 entonces la salida es S5

Al simularlo, esta fue la aproximación que generó:

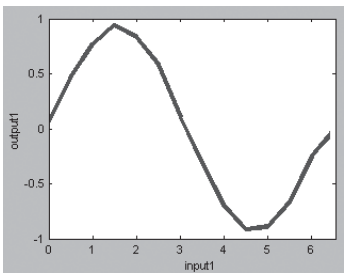


Fig. 2.51 Aproximación de la función seno usando lógica difusa en MATLAB®.

Este ejemplo demuestra que con lógica difusa se puede aproximar de manera muy sencilla una función; en este caso se generó una buena aproximación de la función seno con tan sólo nueve puntos de referencia.

Ejemplo de un sistema difuso con retardos en la información para aproximaciones difusas

En este ejemplo se presentan los resultados para obtener una función senoidal a través de reglas lingüísticas de predicción, en donde a partir del valor de la salida actual y la salida anterior se podrá obtener la entrada siguiente. Se presentan las funciones de membresía, las reglas y la superficie de salida que demuestran la validez del sistema difuso planteado.

Funciones de membresía

Para obtener las reglas lingüísticas capaces de predecir, necesitamos dos funciones de membresía para las entradas, así como una función de membresía de salida. Las funciones de membresía de las entradas contienen los distintos ángulos a los que se evaluará la función seno. Las funciones de membresía pueden observarse en las figuras 2.52 y 2.53 para el valor anterior y el valor actual.

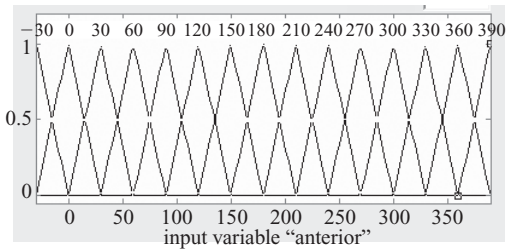


Fig. 2.52 Funciones de membresía para el valor anterior.

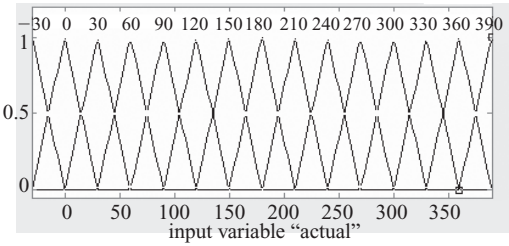


Fig. 2.53 Funciones de membresía para el valor actual.

Para el caso de la salida, tenemos los valores que se obtienen al evaluar el seno para cada uno de los ángulos que componen las entradas. Con esto armamos las funciones de membresía de la figura 2.54.

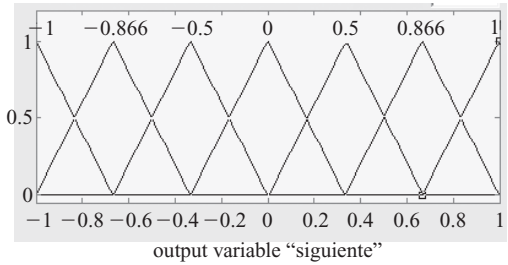


Fig. 2.54 Funciones de membresía para la salida.

Reglas lingüísticas

Las reglas lingüísticas necesarias para que el sistema pueda predecir el valor siguiente a través de la información de las salidas anteriores, quedan entonces de la siguiente forma:

Si anterior es −30 y actual es 0 entonces siguiente es 0.5
Si anterior es 0 y actual es 30 entonces siguiente es 0.866
Si anterior es 30 y actual es 60 entonces siguiente es 1
Si anterior es 60 y actual es 90 entonces siguiente es 0.866
Si anterior es 90 y actual es 120 entonces siguiente es 0.5
Si anterior es 120 y actual es 150 entonces siguiente es 0
Si anterior es 150 y actual es 180 entonces siguiente es −0.5
Si anterior es 180 y actual es 210 entonces siguiente es −0.866
Si anterior es 210 y actual es 240 entonces siguiente es −1
Si anterior es 240 y actual es 270 entonces siguiente es −0.866
Si anterior es 270 y actual es 300 entonces siguiente es −0.5
Si anterior es 300 y actual es 330 entonces siguiente es 0

Superficie de salida

Al evaluar estas reglas se debe obtener la superficie en la cual se observa el seno. La superficie de salida se observa en la figura 2.55.

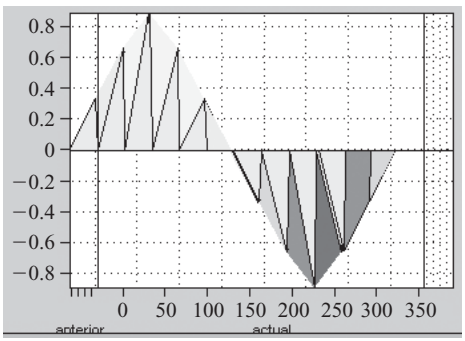


Fig. 2.55 Superficie de salida para el seno.

Como puede verse, la superficie es similar a la resultante de la función seno, por lo que se puede concluir que el sistema responde con las características deseadas.

Diseño de controladores con base en Mamdani

Mamdani propuso el primer controlador difuso en los años setenta y comprende el siguiente procedimiento básico:

1. Siendo el error la diferencia entre el valor deseado y el valor real de la variable que se va a controlar, esto es: $\varepsilon = V_{deseado} - V_{real}$, se seleccionan las funciones de pertenencia que realizarán la difusificación.
2. Se establecen las reglas a partir de proposiciones condicionales, y el dispositivo de inferencia será una composición máx-mín que ha sido previamente definida como:

$$X_T(x,z) = \vee_{y \in Y}(X_R(x,y) \wedge X_S(y,z))$$

3. Se seleccionan las funciones de pertenencia para la desdifusificación y el método que se va a usar para encontrar el valor nítido de la salida, normalmente correspondiente al método del centroide:

$$\frac{\sum \mu(x) \cdot x}{\sum \mu(x)} \left(\text{discreto}\right) \qquad \frac{\int \mu(x) \cdot x\, dx}{\int \mu(x)\, dx} \left(\text{continuo}\right)$$

Ejemplo

Se desea automatizar un cultivo hidropónico de vegetales. Considerando que para cada tipo de vegetal existen diferentes requerimientos de temperatura, humedad e intensidad luminosa, y que además dependen de la hora del día, es necesario que el controlador tenga cuatro entradas correspondientes a cada una de las variables mencionadas, como se aprecia en la figura 2.56.

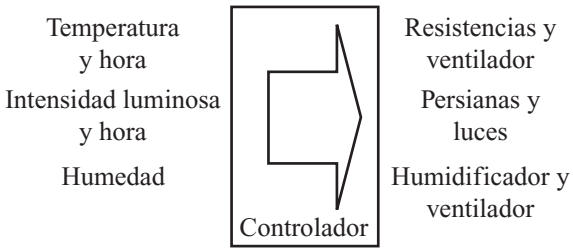


Fig. 2.56 Entradas y salidas del controlador del ejemplo.

Asimismo, el diseño propuesto implica el uso de cinco actuadores para el sistema: resistencias, persianas, luces, un humidificador y un ventilador, los cuales proporcionarán las características ambientales que requiere el cultivo.

Para poder difusificar los voltajes obtenidos a partir de los sensores que miden las variables del sistema, se han establecido las funciones de pertenencia correspondientes a cada variable, con sus respectivas etiquetas y universos de discurso, para la temperatura, la intensidad luminosa, la humedad y la hora, respectivamente en las figuras 2.57 a 2.60.

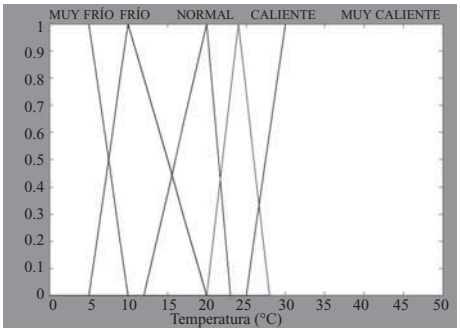


Fig. 2.57 Funciones de pertenencia para la temperatura.

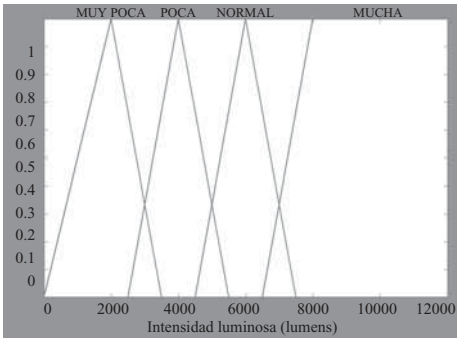


Fig. 2.58 Funciones de pertenencia para la intensidad luminosa.

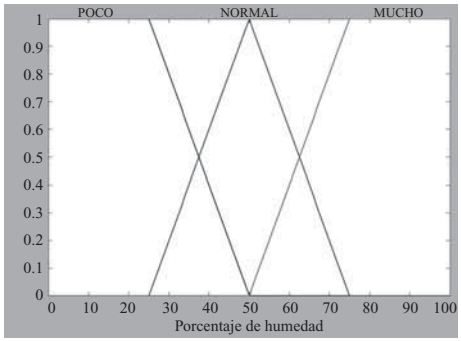


Fig. 2.59 Funciones de pertenencia para la humedad.

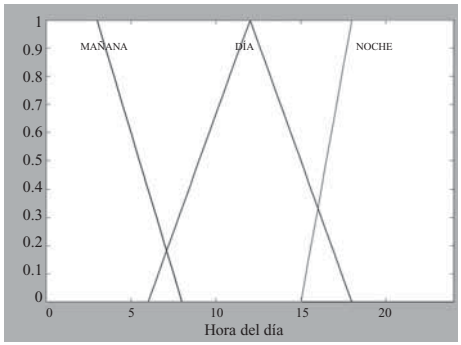


Fig. 2.60 Funciones de pertenencia para la hora.

Con las funciones de pertenencia elegidas y respectivamente etiquetadas, se procedió al diseño de la FAM o matriz de reglas difusas para relacionar las variables de entrada con el comportamiento de los actuadores a la salida del sistema. Para esto se crearon tres distintas matrices que se muestran en las tablas 2.8, 2.9 y 2.10, para el control de la temperatura, la intensidad luminosa y la humedad del ambiente, respectivamente.

Tabla 2.8 FAM para el control de la temperatura

		Temperatura									
		Muy frío		Frío		Normal		Caliente		Muy caliente	
Hora	Mañana	ON	OFF	OFF	OFF	OFF	MEDIO	OFF	ON	OFF	ON
	Día	ON	OFF	MEDIO	OFF	OFF	MEDIO	OFF	ON	OFF	ON
	Noche	ON	OFF	OFF	OFF	OFF	MEDIO	OFF	ON	OFF	ON

Resistencia Ventilador

Tabla 2.9 FAM para el control de la intensidad luminosa

		Intensidad luminosa							
		Muy poca		Poca		Normal		Mucha	
Hora	Mañana	CIERRA	OFF	CIERRA	OFF	CIERRA	OFF	CIERRA	OFF
	Día	ABRE	ON	ABRE	ON	MEDIO	OFF	CIERRA	OFF
	Noche	CIERRA	OFF	CIERRA	OFF	CIERRA	OFF	CIERRA	OFF
		Persianas			Luces				

Tabla 2.10 FAM para el control de la humedad

Humedad (%)			
Poco	Normal	Mucho	
OFF	OFF	ON	Ventilador
ON	OFF	OFF	Humidificador

Para las funciones de desdifusificación se decidió emplear del tipo singleton o barra para simplificar la obtención del valor nítido de voltaje para los actuadores. De esta manera, la selección se observa en las figuras 2.61, 2.62, 2.63, 2.64 y 2.65, para el ventilador, las resistencias, las persianas, las luces y el humidificador, respectivamente. Para encontrar el valor nítido se utilizó el método del centroide.

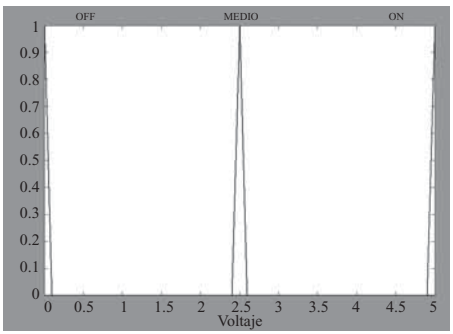


Fig. 2.61 Funciones de pertenencia para el ventilador.

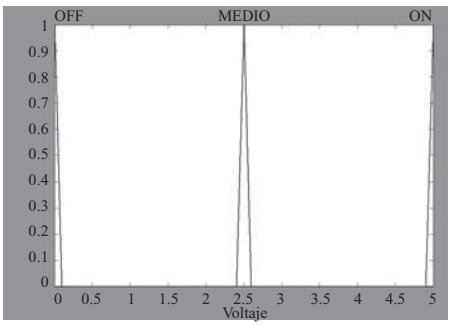


Fig. 2.62 Funciones de pertenencia para las resistencias.

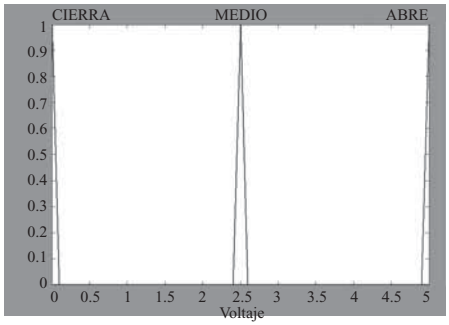


Fig. 2.63 Funciones de pertenencia para las persianas.

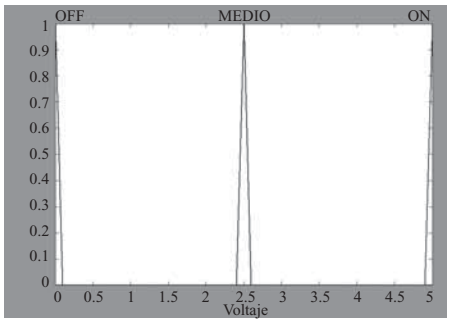


Fig. 2.64 Funciones de pertenencia para las luces.

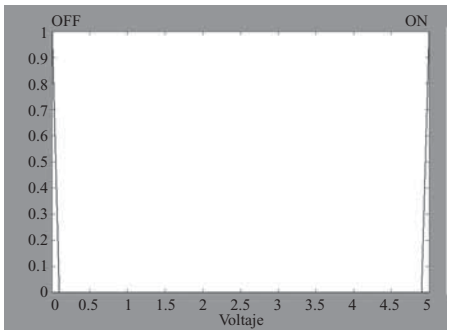


Fig. 2.65 Funciones de pertenencia para el humidificador.

Aplicaciones reales de controladores difusos

Para controlar condiciones ambientales como la temperatura en distintos lugares cerrados, principalmente casas habitación, se propuso y diseñó un controlador difuso[1] que incorpora microprocesadores comunes de la familia PIC en conjunto con lenguaje C obtenido a partir de un intérprete que convierte funciones difusas predeterminadas a C, denominado Fuzz-C. En este preprocesador se establecen los parámetros de las funciones de pertenencia de tipo convencional para las entradas y salidas y las reglas difusas en forma lingüística a través de proposiciones, y se realiza la desdifusificación empleando el método del centroide.

El código C resultante se compila en el PIC, cuyo sistema mínimo incorpora además una interfaz de conversión de señal analógica a digital que mide la temperatura a partir de termistores, una pantalla

[1] Walter Banks, Ashok Patel y Sherif Abdel-Kader (1995). "A Home Control System Based on Fuzzy Logic" en *Home Automation & Building Control.*

LCD con botones para ajustar el rango deseado de temperatura a distintas horas del día, memorias para almacenar los datos obtenidos y los deseados, así como una interfaz serial para comunicar la información a la computadora en caso de que se desee monitorear. De esta manera es posible controlar el sistema de aire acondicionado, ventiladores y humidificadores para obtener las condiciones deseadas en distintas regiones del planeta.

Controlador difuso clásico

Un controlador difuso clásico tiene un lazo de control como el de la figura 2.66. Al igual que los controladores industriales clásicos, contempla el uso de una acción proporcional (controlador P), proporcional-derivativa (controlador PD), proporcional-integral (controlador PI) y proporcional-integral-derivativa (Controlador PID).

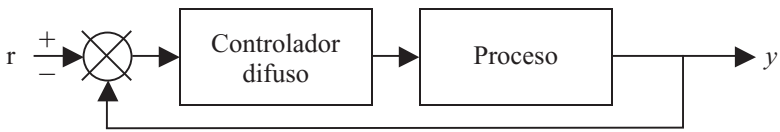


Fig. 2.66 Lazo de un controlador difuso clásico.

Partiendo de la respuesta de un sistema a un escalón de ganancia *r*, que es el valor deseado del sistema, el cual se muestra en la figura 2.67, y con funciones de pertenencia para la difusificación y desdifusificación de tipo triangular, con tres términos y las mismas etiquetas (figura 2.68), es posible diseñar las matrices de reglas para cada tipo de controlador según su acción: controlador P, controlador PD, controlador PI y controlador PID.

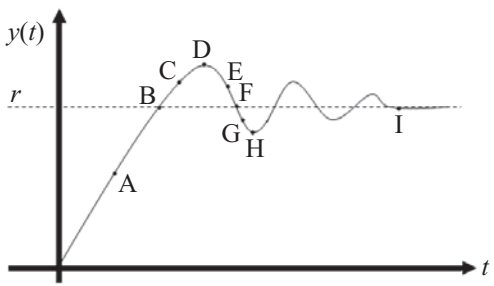


Fig. 2.67 Respuesta de un sistema a un escalón r.

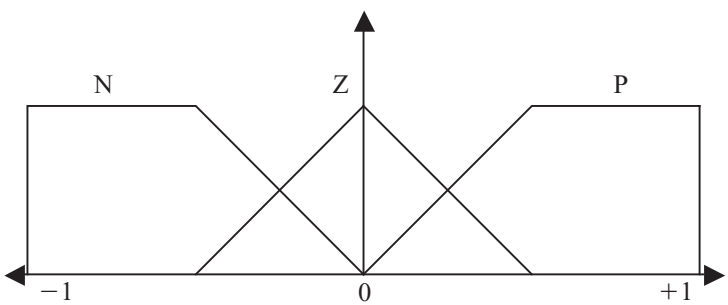


Fig. 2.68 Funciones de pertenencia propuestas para la difusificación y desdifusificación.

Controlador P

La ley de control en un controlador tipo P se define como $u = K_P\varepsilon$, donde $\varepsilon = r - y$. A partir de esto, se proponen las siguientes reglas, representadas de forma matricial en una tabla denominada matriz de asociación difusa o FAM.

Tabla 2. 11 FAM para un controlador P.

$\dot{\varepsilon}$ \ ε	N	Z	P
u	N	Z	P

Controlador PD

La ley de control en un controlador tipo PD se define como $u = K_P\varepsilon + K_D\dot{\varepsilon}$, donde $\varepsilon = r - y$. Se propone la siguiente FAM.

Tabla 2.12 FAM para un controlador PD.

ε	N	Z	P
N	N	N	P
Z	N	Z	P
P	N	P	P

Controlador PI

La ley de control en un controlador tipo PI se define como $u = K_P\varepsilon + K_I\int \varepsilon\, d\varepsilon$, donde $\varepsilon = r - y$. Siendo que la derivada de la ley de control es $\dot{u} = K_P\dot{\varepsilon} + K_I\varepsilon$, si se integra la salida de un controlador PD es posible obtener un controlador PI empleando la FAM del controlador PD.

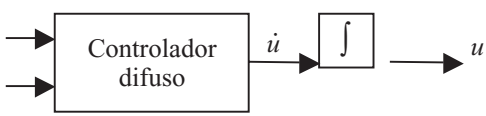


Fig. 2.69 Obtención de un controlador PI difuso a partir de uno tipo PD.

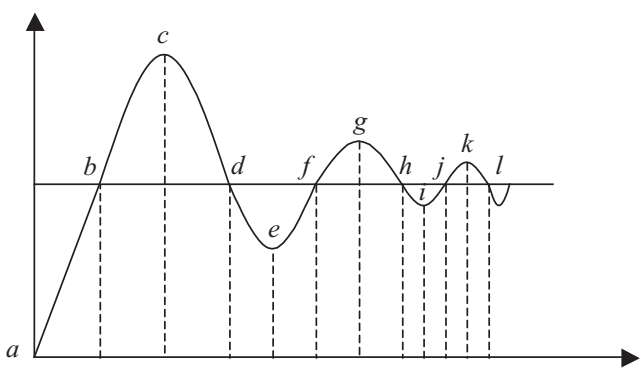


Fig. 2.70 Respuesta a escalón de una planta.

Para la planta de la figura 2.70 se obtienen las siguientes reglas prototipo para conjuntos difusos denominados negativo (N), cero (Z) y positivo (P).

Tabla 2.13 FAM para un controlador PD con tres funciones de pertenencia.

Regla	Error	Derivada del error	Salida	Punto de referencia
1	P	Z	P	a, e, i
2	Z	N	N	b, f, j
3	N	Z	N	c, g, k
4	N	P	P	d, h, l
5	Z	Z	Z	Establecimiento

Para el mismo sistema de la figura 2.70 se obtienen las siguientes reglas para conjuntos difusos denominados negativo grande (NG), negativo medio (NM), negativo pequeño (NP), cero (ZE), positivo pequeño (PP), positivo medio (PM), positivo grande (PG).

Tabla 2.14 FAM para un controlador PD con siete funciones de pertenencia.

Regla	Error	Derivada del error	Salida	Punto de referencia
1	PG	ZE	PG	a
2	PM	ZE	PM	e
3	PP	ZE	PP	i
4	ZE	NG	NG	b
5	ZE	NM	NM	f
6	ZE	NS	NS	j
7	NG	ZE	NG	c
8	NM	ZE	NM	g
9	NP	ZE	NP	k
10	ZE	PG	PG	d
11	ZE	PM	PM	h
12	ZE	PP	PP	l
13	ZE	ZE	ZE	Establecimiento

Estabilidad

Se puede considerar en un sistema dos tipos de estabilidad:

- *Estabilidad interna: cuando se observa el comportamiento del sistema con condiciones iniciales distintas a cero, y condiciones iniciales nulas.*

- *Estabilidad externa: cuando el sistema tiene condiciones iniciales nulas y entradas diferentes de cero.*

La estabilidad en un sistema lineal se puede analizar a través de la matriz de transición, esto para sistemas invariantes en el tiempo y la estabilidad externa con la matriz de función de transferencia. Para lo cual se pueden definir puntos de equilibrio a través de la definición de sistema estable de Lyapunov, donde el estado de equilibrio $x_e = 0$ *es estable en el sentido de Lyapunov, si cumple que para cualquier escalar dado* $\varepsilon > 0$*, existe un escalar* $\delta(t_0, \varepsilon) > 0$ *tal que* $\|x_0\| < \delta$*, donde* $\|x(t)\| < \varepsilon$*, para todo* $t \geq t_0$

Es muy útil el plano de estados o plano de fase, donde se puede describir una familia de trayectorias para cada uno de los estados y verificar entre otras cosas las condiciones de estabilidad del sistema.

Las reglas pueden probarse mediante un plano de fase, graficando al error y a la derivada del error, donde el segundo cuadrante corresponde al decrecimiento del sobrepaso y el cuarto al decrecimiento del tiempo de impulso, como muestra la figura 2.71.

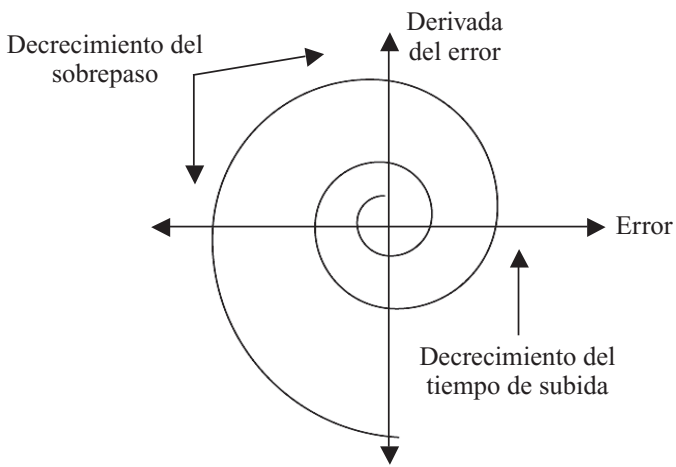


Fig. 2.71 Plano de fase.

De manera que se obtienen nuevas reglas para las tablas mostradas anteriormente:

Tabla 2.15 Continuación de la FAM para un controlador PD con tres funciones de pertenencia.

Regla	Error	Derivada del error	Salida	Punto de referencia
6	P	N	P	i, v
7	N	N	N	ii, vi
8	N	P	N	iii, vii
9	P	P	P	iv, viii
10	P	N	Z	ix
11	N	P	Z	xi

Tabla 2.16 Continuación de la FAM para un controlador PD con siete funciones de pertenencia.

Regla	Error	Derivada del error	Salida	Punto de referencia
14	PG	NP	PM	i
15	PP	NG	NM	ii
16	NG	PP	NM	iii
17	NP	PG	PM	iii
18	PP	NP	ZE	ix
19	NP	PP	ZE	xi

El plano de fase lingüístico para el controlador de la segunda tabla se ve en dos versiones en la figura 2.72.

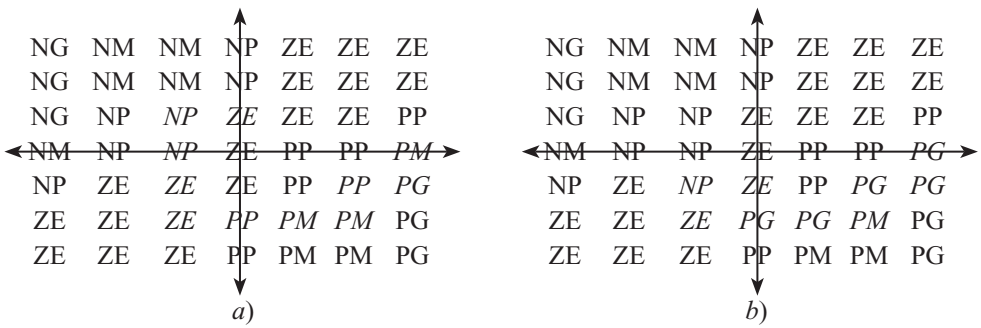


Fig. 2.72 Plano de fase lingüístico *a*) con reglas iniciales y *b*) con reglas modificadas.

Ejemplo

Diseñar controladores tipo P, PD, PI y PID para tres y cinco funciones de pertenencia empleando MATLAB® para la planta de segundo orden:

$$G(s) = \frac{25}{s^2 + 4s + 25}$$

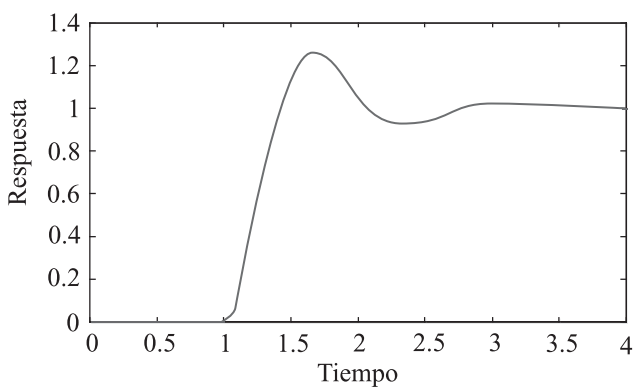


Fig. 2.73 Respuesta de la planta del ejemplo ante un escalón unitario de entrada.

Controladores P

Se diseñó dos controladores tipo P, uno con tres funciones de pertenencia triangulares, y otro con cinco, para evaluar el comportamiento de los mismos. Para la difusificación y la desdifusificación en el primer controlador se seleccionó tres conjuntos difusos que se ven en la figura 2.74. Para el segundo se propuso las funciones de pertenencia que se aprecian en la figura 2.75.

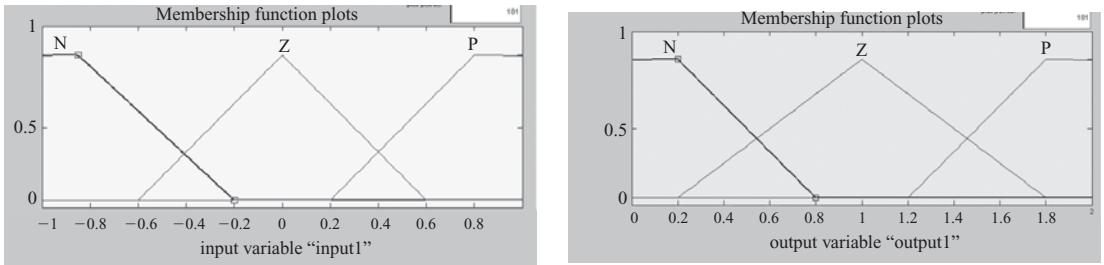


Fig. 2.74 Funciones de pertenencia propuestas para el controlador 1.

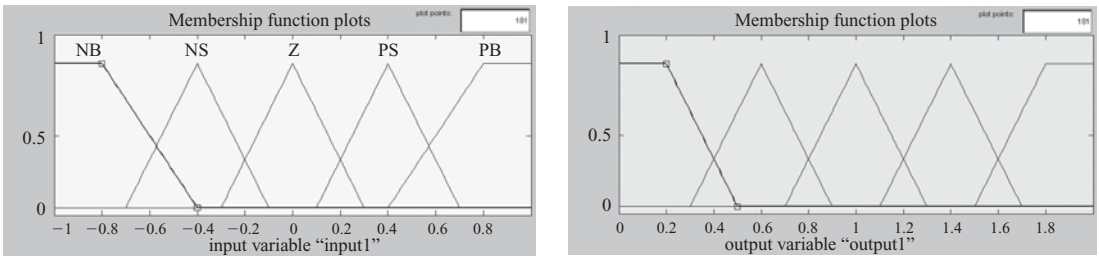


Fig. 2.75 Funciones de pertenencia propuestas para el controlador 2.

Asimismo, se diseñó las siguientes matrices de asociación difusa para las reglas, las cuales se listan en las tablas 2.17 y 2.18 para el controlador 1 y el controlador 2, respectivamente, que se programaron en MATLAB® empleando el Toolkit de controladores difusos, al igual que las funciones de pertenencia para la entrada y la salida.

Tabla 2.17 FAM para el controlador 1

Entrada (error)	**N**	**Z**	**P**
Salida (corrección)	N	Z	P

Tabla 2.18 FAM para el controlador 2

Entrada (error)	**NB**	**NS**	**ZE**	**PS**	**PB**
Salida (corrección)	NB	NS	ZE	PS	PB

Finalmente se comprobó el desempeño de los controladores mediante un lazo de control en Simulink® de MATLAB®, resultando las imágenes de las figuras 2.76 y 2.77, respectivamente.

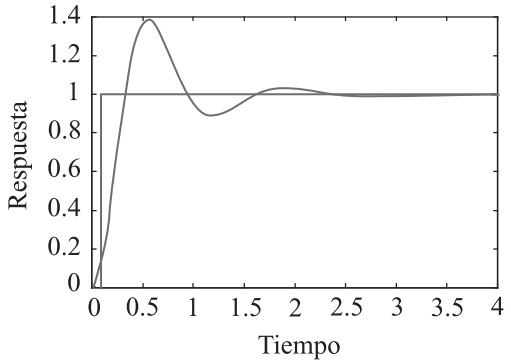


Fig. 2.76 Resultado del controlador 1.

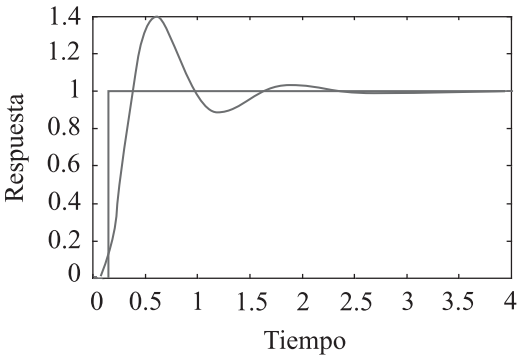


Fig. 2.77 Resultado del controlador 2.

Controladores PD

También se propuso dos controladores tipo PD, uno con tres funciones de pertenencia triangulares y otro con cinco. Para el controlador 3 se seleccionó tres conjuntos difusos que se ven en la figura 2.78. Para el controlador 4 se propuso las funciones de pertenencia que muestra la figura 2.79.

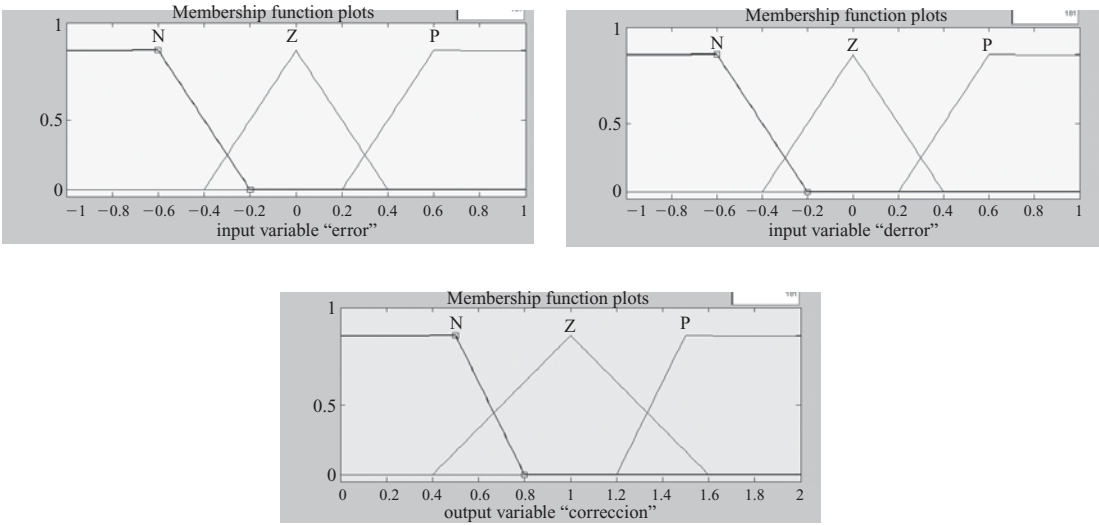
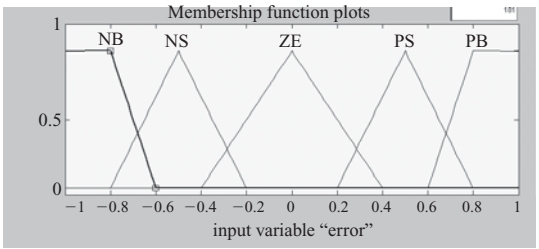
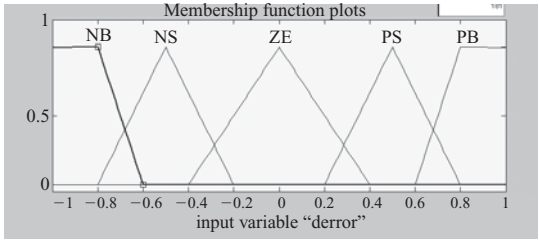


Fig. 2.78 Funciones de pertenencia propuestas para el controlador 3.





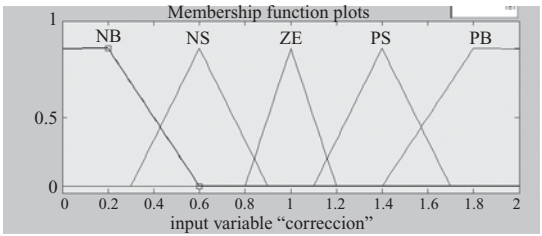


Fig. 2.79 Funciones de pertenencia propuestas para el controlador 4.

Se diseñó las siguientes matrices de asociación difusa para las reglas, listadas en las tablas 2.19 y 2.20 para el controlador 3 y el controlador 4, correspondientemente.

Tabla 2.19 FAM para el controlador 3.

		Error		
		N	**Z**	**P**
Derivada del error	**N**	N	Z	P
	Z	N	Z	P
	P	N	Z	P

Tabla 2.20 FAM para el controlador 4.

		Error				
		NB	**NS**	**ZE**	**PS**	**PB**
Derivada del error	**NB**	NB	NS	ZE	PS	PB
	NS	NB	NS	ZE	PS	PB
	ZE	NB	NS	ZE	PS	PB
	PS	NS	NS	ZE	PS	PB
	PB	NB	NS	ZE	PS	PB

Se comprobó el desempeño de los controladores mediante un lazo de control en Simulink® de MATLAB®, resultando las imágenes de las figuras 2.80 y 2.81, respectivamente.

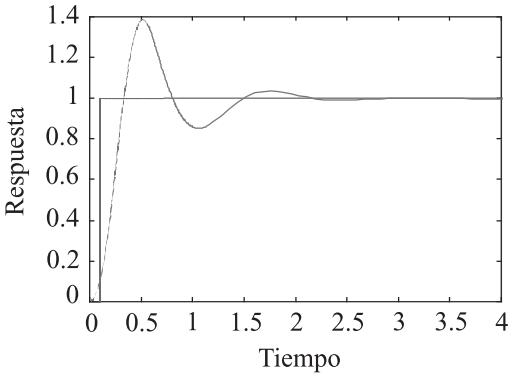


Fig. 2.80 Resultado con el controlador 3.

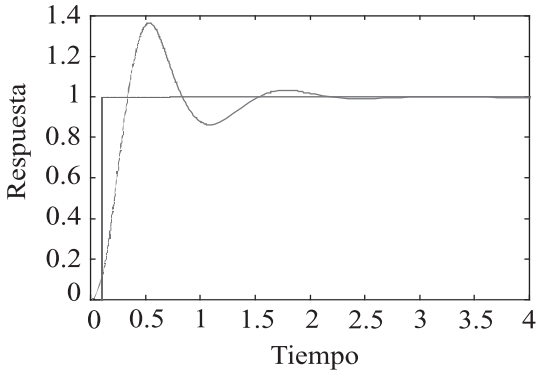


Fig. 2.81 Resultado con el controlador 4.

Controladores PI

Para diseñar los controladores tipo PI, denominados con los números 5 y 6, se utilizó las mismas funciones de pertenencia que en los controladores PD para las entradas, así como las matrices de asociación difusa. En la salida de los controladores se colocó bloques de integración para convertirlos en PI, como se muestra en la figura 2.82 para el controlador 5.

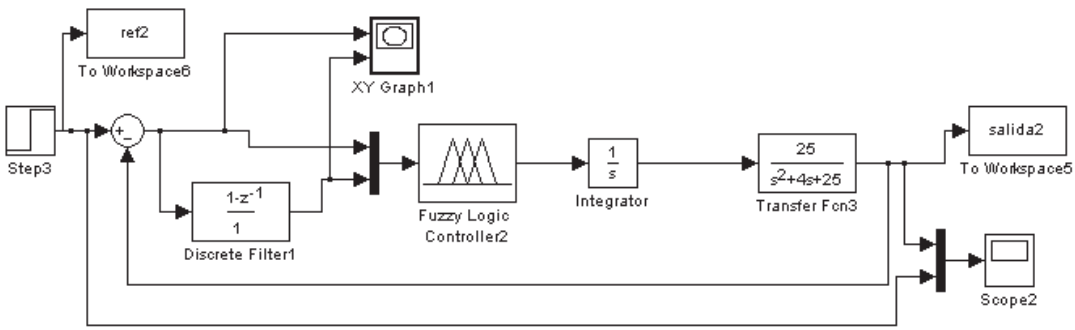


Fig. 2.82 Diagrama de bloques del controlador 5 en Simulink®.

Sin embargo, las funciones de pertenencia de la salida fueron distintas. Las correspondientes al controlador 5 se aprecian en la figura 2.83 y las del controlador 6 en la figura 2.84.

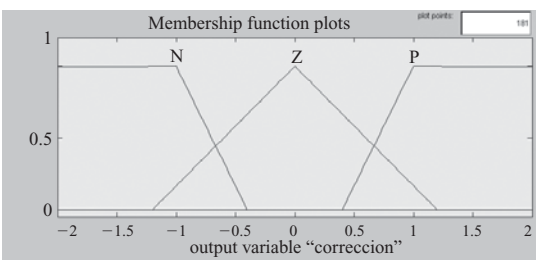


Fig. 2.83 Funciones de pertenencia del controlador 5.

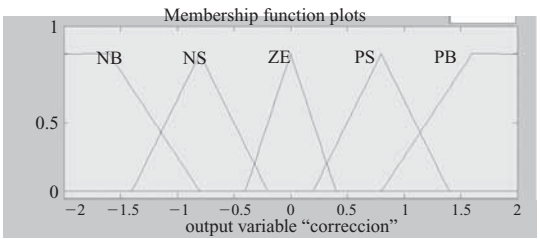


Fig. 2.84 Funciones de pertenencia del controlador 6.

Las figuras 2.85 y 2.86 muestran el comportamiento de los controladores con la planta propuesta.

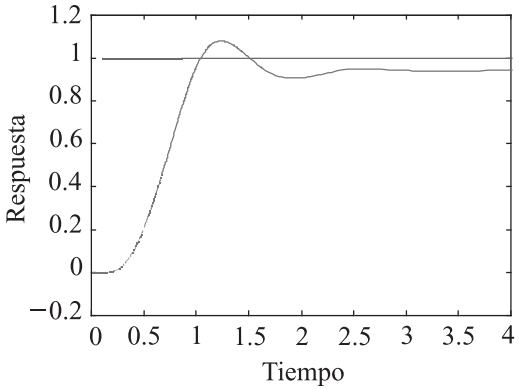


Fig. 2.85 Resultado con el controlador 5.

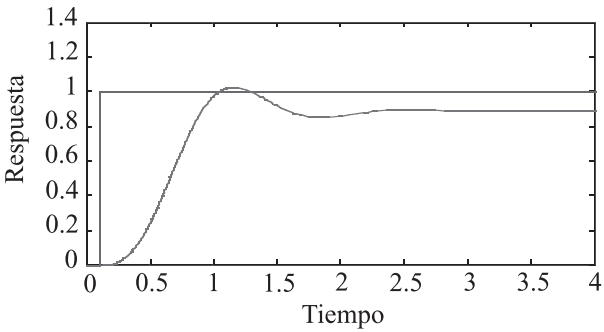


Fig. 2.86 Resultado con el controlador 6.

Controlador PID

Se diseñó dos controladores tipo PID empleando las funciones de pertenencia de la figuras 2.87 y 2.88.

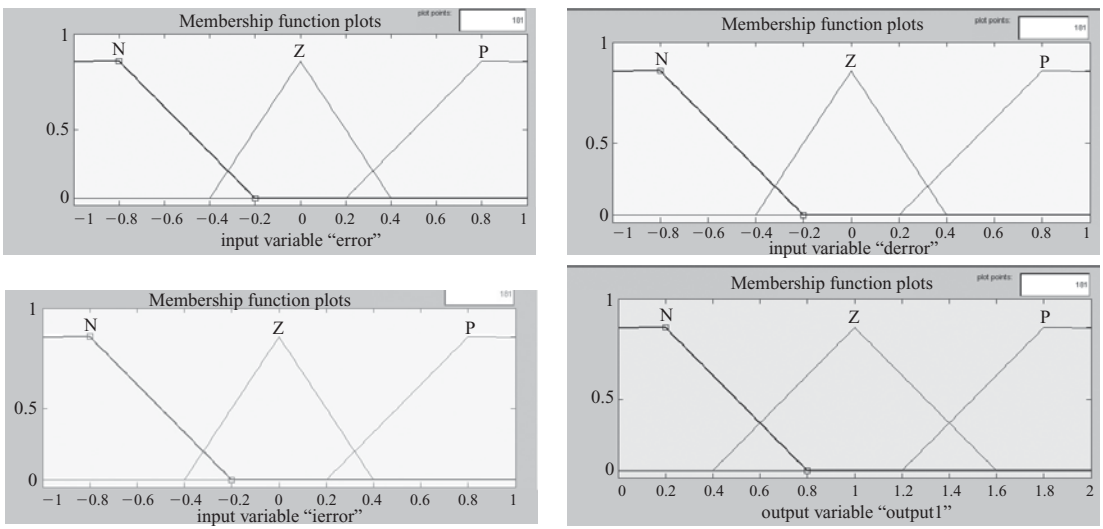


Fig. 2.87 Funciones de pertenencia para el controlador 7.

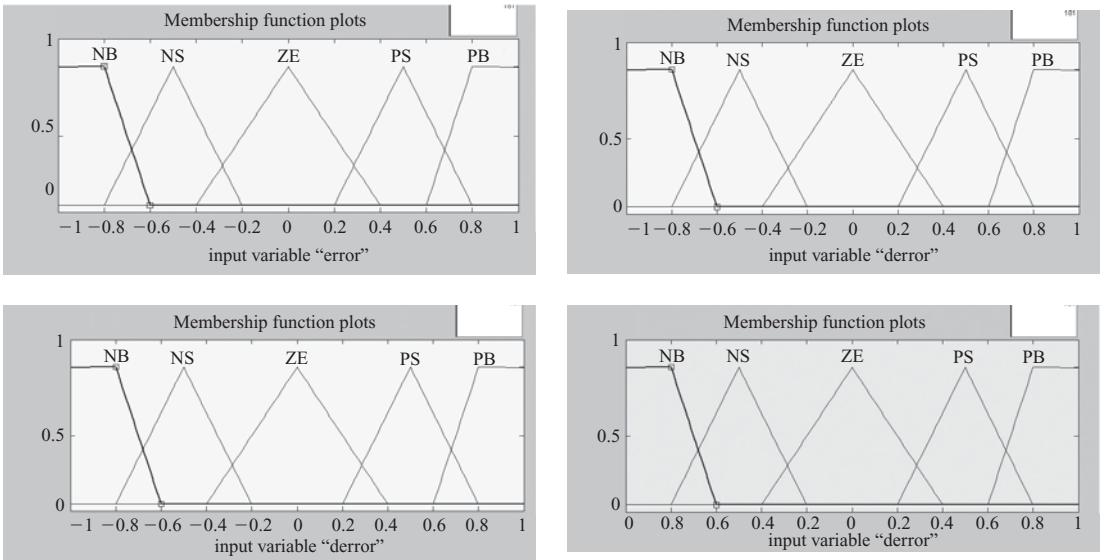


Fig. 2.88 Funciones de pertenencia para el controlador 8.

Se propuso dos FAM cúbicas, las cuales se muestran en forma vertical en las tablas 2.21 y 2.22, para el controlador 7 y 8, respectivamente.

Tabla 2.21 FAM para el controlador 7

Error	Derivada	Integral	Corrección
N	N	N	N
N	N	Z	N
N	N	P	N
N	Z	N	N
N	Z	Z	N
N	Z	P	N
N	P	N	N
N	P	Z	N
N	P	P	N
Z	N	N	Z
Z	N	Z	Z
Z	N	P	Z
Z	Z	N	Z
Z	Z	Z	Z
Z	Z	P	Z
Z	P	N	Z

(*continúa*)

Tabla 2.21 FAM para el controlador 7 (*continuación*)

Error	Derivada	Integral	Corrección
Z	P	Z	Z
Z	P	P	Z
P	N	N	P
P	N	Z	P
P	N	P	P
P	Z	N	P
P	Z	Z	P
P	Z	P	P
P	P	N	P
P	P	Z	P
P	P	P	P

Tabla 2.22 FAM para el controlador 8

Error	Derivada	Integral	Corrección
NB	NB	NB	NB
NB	NB	NS	NB
NB	NB	ZE	NB
NB	NB	PS	NB
NB	NB	PB	NB
NB	NS	NB	NB
NB	NS	NS	NB
NB	NS	ZE	NB
NB	NS	PS	NB
NB	NS	PB	NB
NB	ZE	NB	NB
NB	ZE	NS	NB
NB	ZE	ZE	NB
NB	ZE	PS	NB
NB	ZE	PB	NB

(*continúa*)

Tabla 2.22 FAM para el controlador 8 (*continuación*)

Error	Derivada	Integral	Corrección
NB	PS	NB	NS
NB	PS	NS	NS
NB	PS	ZE	NS
NB	PS	PS	NS
NB	PS	PB	NS
NB	PB	NB	NB
NB	PB	NS	NB
NB	PB	ZE	NB
NB	PB	PS	NB
NB	PB	PB	NB
NS	NB	NB	NS
NS	NB	NS	NS
NS	NB	ZE	NS
NS	NB	PS	NS
NS	NB	PB	NS
NS	NS	NB	NS
NS	NS	NS	NS
NS	NS	ZE	NS
NS	NS	PS	NS
NS	NS	PB	NS
NS	ZE	NB	NS
NS	ZE	NS	NS
NS	ZE	ZE	NS
NS	ZE	PS	NS
NS	ZE	PB	NS
NS	PS	NB	NS
NS	PS	NS	NS
NS	PS	ZE	NS
NS	PS	PS	NS
NS	PS	PB	NS

(*continúa*)

Tabla 2.22 FAM para el controlador 8 (*continuación*)

Error	Derivada	Integral	Corrección
NS	PB	NB	NS
NS	PB	NS	NS
NS	PB	ZE	NS
NS	PB	PS	NS
NS	PB	PB	NS
ZE	NB	NB	ZE
ZE	NB	NS	ZE
ZE	NB	ZE	ZE
ZE	NB	PS	ZE
ZE	NB	PB	ZE
ZE	NS	NB	ZE
ZE	NS	NS	ZE
ZE	NS	ZE	ZE
ZE	NS	PS	ZE
ZE	NS	PB	ZE
ZE	ZE	NB	ZE
ZE	ZE	NS	ZE
ZE	ZE	ZE	ZE
ZE	ZE	PS	ZE
ZE	ZE	PB	ZE
ZE	PS	NB	ZE
ZE	PS	NS	ZE
ZE	PS	ZE	ZE
ZE	PS	PS	ZE
ZE	PS	PB	ZE
ZE	PB	NB	ZE
ZE	PB	NS	ZE
ZE	PB	ZE	ZE
ZE	PB	PS	ZE
ZE	PB	PB	ZE

(*continúa*)

Tabla 2.22 FAM para el controlador 8 (*continuación*)

Error	Derivada	Integral	Corrección
PS	NB	NB	PS
PS	NB	NS	PS
PS	NB	ZE	PS
PS	NB	PS	PS
PS	NB	PB	PS
PS	NS	NB	PS
PS	NS	NS	PS
PS	NS	ZE	PS
PS	NS	PS	PS
PS	NS	PB	PS
PS	ZE	NB	PS
PS	ZE	NS	PS
PS	ZE	ZE	PS
PS	ZE	PS	PS
PS	ZE	PB	PS
PS	PS	NB	PS
PS	PS	NS	PS
PS	PS	ZE	PS
PS	PS	PS	PS
PS	PS	PB	PS
PS	PB	NB	PS
PS	PB	NS	PS
PS	PB	ZE	PS
PS	PB	PS	PS
PS	PB	PB	PS
PB	NB	NB	PB
PB	NB	NS	PB
PB	NB	ZE	PB
PB	NB	PS	PB
PB	NB	PB	PB

(*continúa*)

Tabla 2.22 FAM para el controlador 8 (*continuación*)

Error	Derivada	Integral	Corrección
PB	NS	NB	PS
PB	NS	NS	PS
PB	NS	ZE	PS
PB	NS	PS	PS
PB	NS	PB	PS
PB	ZE	NB	PB
PB	ZE	NS	PB
PB	ZE	ZE	PB
PB	ZE	PS	PB
PB	ZE	PB	PB
PB	PS	NB	PB
PB	PS	NS	PB
PB	PS	ZE	PB
PB	PS	PS	PB
PB	PS	PB	PB
PB	PB	NB	PB
PB	PB	NS	PB
PB	PB	ZE	PB
PB	PB	PS	PB
PB	PB	PB	PB

Las respuestas del sistema con los controladores propuestos pueden apreciarse en la figura 2.89 para el controlador 7, y para el 8 en la figura 2.90.

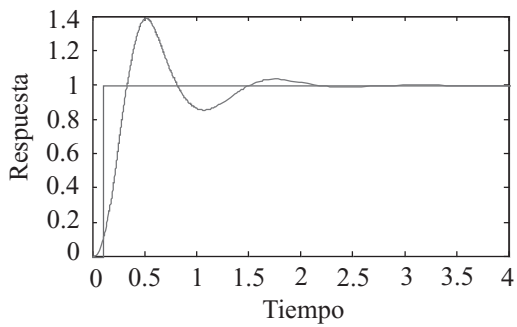


Fig. 2.89 Respuesta con el controlador 7.

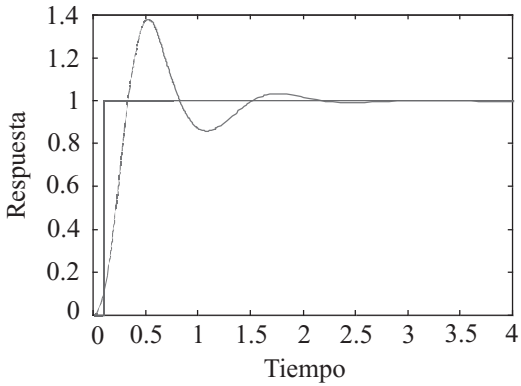


Fig. 2.90 Respuesta con el controlador 8.

Las tablas que se muestran a continuación presentan otras configuraciones propuestas para un controlador PID de siete funciones de pertenencia por entrada y otras siete de salida: positivo grande (PG), positivo medio (PM), positivo pequeño (PP), cero (ZE), negativo grande (NG), negativo medio (NM) y negativo pequeño (NP), para dos configuraciones distintas de la integral del error.

Tabla 2.23 FAM para un controlador PID, integral del error = PG

Integral del error: PG							
$\dot{\varepsilon}$ ε	**NG**	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
NG	PG	PM	PS	Z	NP	NM	NG
NM	PM	PP	ZE	NP	NM	NG	NG
NP	PP	ZE	NP	NM	NG	NG	NG
ZE	ZE	NP	NM	NG	NG	NG	NG
PP	NP	NM	NG	NG	NG	NG	NG
PM	NM	NG	NG	NG	NG	NG	NG
PG	NG	NG	NG	NG	NG	NG	NG

Tabla 2.24 FAM para un controlador PID, integral del error = PM

Integral del error: PM							
$\dot{\varepsilon}$ ε	**NG**	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
NG	PG	PG	PM	PP	ZE	NP	NM
NM	PG	PM	PP	ZE	NP	NM	NG
NP	PM	PP	ZE	NP	NM	NG	NG
ZE	PP	ZE	NP	NM	NG	NG	NG
PP	ZE	NP	NM	NG	NG	NG	NG
PM	NP	NM	NG	NG	NG	NG	NG
PG	NM	NG	NG	NG	NG	NG	NG

Simulación de un Control PID difuso

$$U(t) = k_p\left[e(t) + \frac{1}{T_I}\int_0^T e(t)\,dt + T_d\frac{de(t)}{dt}\right]$$

$$= k_p e(t) + k_I\int_0^t e(t)\,dt + k_d\frac{de(t)}{dt}$$

Donde k_p, k_1 y k_d son constantes

$$U(k) = k_p e(k) + k_p\frac{T_d}{T_I}\sum_{i=0}^{k} e(i) + k_p\frac{T_D}{T_d}\left[e(k) - e(k-1)\right]$$

$$= k_p e(k) + k_{Id}\sum_{i=0}^{k} e(i) + k_{Dd}\Delta e(k)$$

$$\Delta U(k) = U(k) - U(k-1)$$

$$U(k) = U(k-1) + \Delta U(k)$$

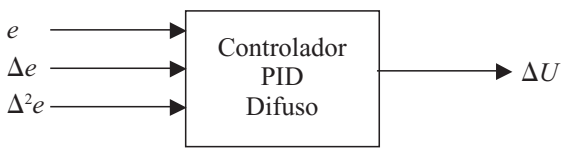
$$\Delta U(k) = k_{Id}e(k) + k_{pd}\Delta e(k) + k_{Dd}\left[\Delta e(k) - \Delta e(k-1)\right]$$

$$\Delta^2 e(k) = \Delta e(k) - \Delta e(k-1)$$

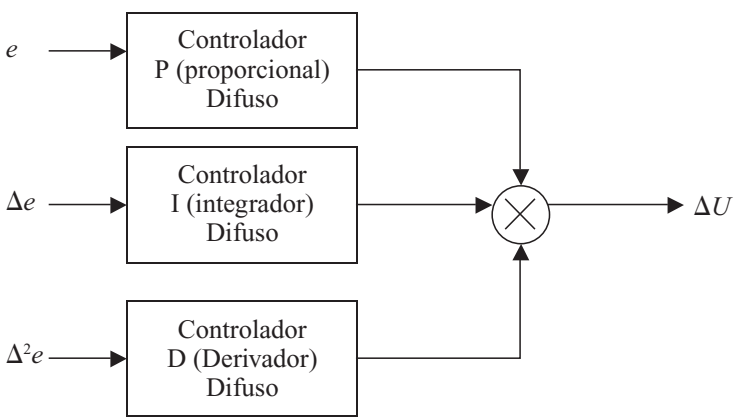
Existen dos maneras de armar un Control PID Difuso. La *primera propuesta* es colocando en un solo bloque las variables, error, la derivada del error y la doble derivada del error.

La segunda propuesta es creando cada variable por separado (su propio bloque), sumado los bloques y obtener así el PID.

La *primera propuesta*



La *segunda propuesta*



Otro modo de apreciar el PID es:

$$U(t) = k_p e(t) + k_I \int_0^t e(t)\,dt + k_d \frac{de(t)}{dt}, \text{ derivando esta expresión}$$

$$\text{queda } \frac{du}{dt} = \dot{U}(t) = k_p \dot{e}(t) + k_I e(t) + k_d \frac{d^2 e(t)}{dt}.$$

Controlador difuso con PID convencional como respaldo

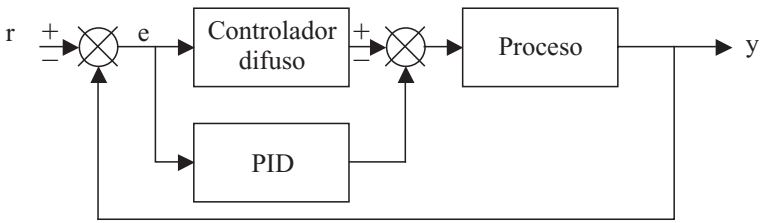


Fig. 2.91 Lazo de un controlador difuso con PID convencional como respaldo.

Controlador difuso como sintonizador de PID convencional

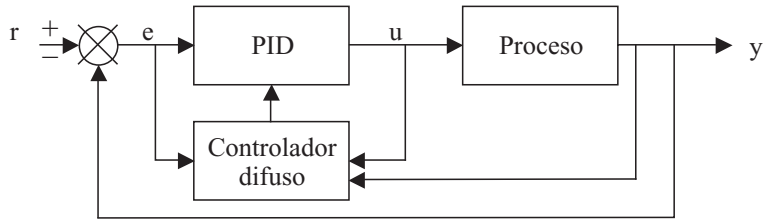


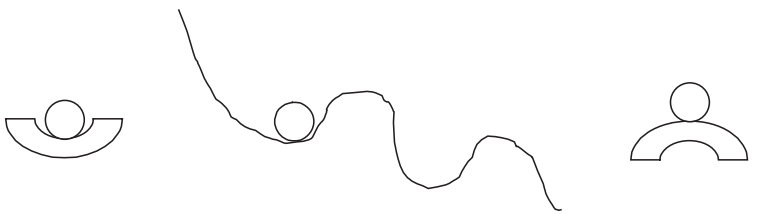
Fig. 2.92 Lazo de un controlador difuso como sintonizador de PID convencional.

Concepto de estabilidad

Punto de equilibrio

Estabilidad: un sistema no puede ir más allá del punto de equilibrio si tiene pequeñas perturbaciones.

1) Punto de equilibrio estable.
2) Condición de punto de equilibrio estable.
3) Punto de equilibrio inestable.

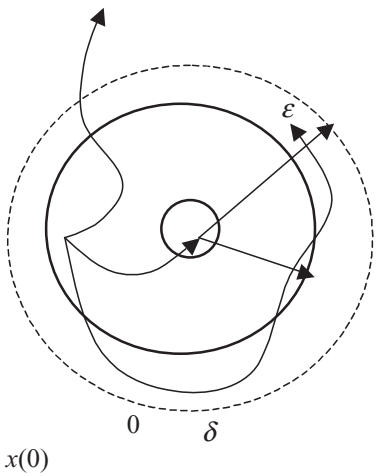


(1) (2) (3)

Asintóticamente estable

Un sistema es asintóticamente estable si éste cumple con el sentido de Lyapunov, y

$\exists \quad \delta(t_0) > 0, \quad \left\|x\left(t_0\right)\right\| < \delta(t_0) \;\text{ con }\; \lim_{t\to\infty}\left\|x(t_0)\right\| = 0.$



El estado de equilibrio $x_e = 0$ de un sistema $\dot{x}\left(t\right) = A\left(t\right)x\left(t\right)$ es estable si para cualquier ε $\exists, \quad \left\|x\left(t_0\right)\right\| < \delta(t_0, \varepsilon)$ implica $\left\|x\left(t\right)\right\| < \varepsilon \quad \forall \quad t \geq t_0$

Donde $\dot{x}\left(t\right)$ se puede representar como variables de estado

$\dot{X} = Ax + Bu$

$Y = Cx + Du$

Los tipos de movimiento *libre de estabilidad* son:

- Pequeños disturbios.
- Grandes disturbios.

Los tipos de movimiento *forzado* de *estabilidad* son:

- Entrada-limitada salida-limitada (*bounded-input bounded-output*) o sus siglas BIBO.
- Un sistema estable es un sistema *dinámico* cuando existe una respuesta limitada debida a una entrada limitada de estabilidad, en el sentido de estabilidad asintótica de Lyapunov.

Entrada-cero de estabilidad

Se llama así a un estado de equilibrio $x_e : x\left(t\right) = x_e$ para todo $t \geq t_0$ si:

- $x_e = 0$
- x_e es un eigenvector correspondiente a un eigenvalor cero, ejemplo:

  $A(x)x_e = \lambda I = 0$

  Suponiendo $\dot{x}\left(t\right) = A\left(t\right)x\left(t\right)$ con $x\left(t_0\right) = x_0$, donde $\det\left(\lambda I - A\right) = 0$

Considerando $\dot{x}(t) = A(t)$ con $x(t_0) = x_0$ se puede definir que:

- El sistemas es estable en el sentido de Lyapunov si $\Re e\{\lambda_i\} \leq 0$, y los eigenvalores con partes reales cero son distintos.
- El sistema es asintóticamente estable si $\Re e\{\lambda_i\} < 0$, para todo *t*.
- Para un sistema linealmente invariante en el tiempo, un sistema asintóticamente estable implica estabilidad exponencial.
- Para sistemas variantes, los sistemas pueden ser inestables aun si $\Re e\{\lambda_i\} < 0$, para todo *t*.

Teorema 1. (Estabilidad de Lyapunov para sistemas autónomos)

Siendo $x = 0$ un punto de equilibrio para un sistema descrito por $\dot{x} = f(x)$ donde $f : U \to \Re^n$ es una *localidad Lipschitz* y $U \subset \Re^n$ es un dominio que contiene el origen. Y sabiendo que $V : U \to \Re$ es una función positiva definida en *U*, continua y diferenciable.* *La propiedad de localidad Lipschitz implica que una función sea continua y diferenciable en más de un dominio*.

Por lo tanto:

1) Si $\dot{V}(x) = \left[\frac{\partial V}{\partial X}\right] f$ es negativo-semidefinido, entonces $x = 0$ es un punto de equilibrio.

2) Si $\dot{V}(x)$ es negativo definido, entonces $x = 0$ es asintóticamente estable.

En ambos casos se llama *función de Lyapunov*.

3) Se cumple que una función escalar de los estados *V*(*x*), la cual es positiva definida en una región *U* alrededor del punto de equilibrio, es una *función de Lyapunov* si:

$$V(x) > 0, excepto \quad si \quad V(0) = 0$$

4) La función *V* es una *función de Lyapunov* si $\dot{V}(x)$ es negativo y semi-definido en

$$U : \dot{V}(x) \leq 0$$

5) La existencia de una *función de Lyapunov* es suficiente para probar la estabilidad (en el sentido de Lyapunov) en una región *U*.

Ejemplo:

Suponiendo que el siguiente es un sistema escalar:

$$\dot{x} = -x^3, \quad x \in \Re$$

Investigar la estabilidad del origen $x = 0$ para la *función de Lyapunov*: $V(x) = 0.5x^2$
La derivada de *V*(*x*) está dada por:

$$\dot{V}(x) = \frac{\partial V}{\partial x}(-x^3) \text{ de lo anterior } V(x) = 0.5x^2 \quad \rightarrow \quad \frac{\partial V}{\partial x} = x$$

por consiguiente $\dot{V}(x) = (x)(-x^3) = \underline{-x^4}$

es negativa definida para todas las $x \in \Re$, el origen es globalmente asintóticamente estable.

Estabilidad de Lyapunov para sistemas difusos

La implicación de Takagi-Sugeno para sistemas difusos dice:

$$P^i : si \quad x(k) \quad es \quad A_1^i \quad y \quad x(k-n+1) \quad es \quad A_n^i$$

$$entonces \quad x^i(k+1) = a_0^i + a_1^i x(k) + \cdots a_n^i x(k-n+1)$$

$$por \quad lo \quad tanto \quad x(k+1) = A^i x(k), \quad donde$$

$$A_i = \begin{bmatrix} a_1^i & a_2^i & \cdots & a_{n-1}^i & a_n^i \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \qquad x(k+1) = \frac{\sum_{i=1}^{l} w^i A_i x(k)}{\sum_{i=1}^{l} w^i}$$

el punto de equilibrio de un sistema difuso es globalmente asintóticamente estable si existe una matriz P común positiva definida para todos los subsistemas tal que

$$A_i^T P A_i - P < 0 \quad para \quad i = 1, \ldots l.$$

Cualquier sistema no lineal puede ser aproximado con segmentos de una función lineal si se satisface la condición de estabilidad.

Teorema

Siendo A_i matrices estables y no-singulares para $i = 1, \ldots, l$ entonces $A_i A_j$ son matrices estables si existe una matriz P común positiva definida tal que

$$A_i^T P A_i - P < 0 \quad para \quad i = 1, \ldots l.$$

Ejemplo de sistema difuso:

Si $x(k)$ es A^1 $entonces$ $x^1(k+1) = 1.2x(k) - 0.6(k-1)$

Si $x(k)$ es A^2 $entonces$ $x^2(k+1) = x(k) - 0.4(k-1)$

Verificando la estabilidad del sistema. Solución:

$$A_1 = \begin{bmatrix} 1.2 & -0.6 \\ 1 & 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1 & -0.4 \\ 1 & 0 \end{bmatrix} \qquad A_1 A_2 = \begin{bmatrix} 0.6 & -0.48 \\ 1 & -0.4 \end{bmatrix}$$

La construcción para muestreo de datos

En esta sección se aborda el problema de la construcción de funciones de membresía por muestreo de datos.

$$X = \Re$$

Con n muestreo de datos

$$\langle x_1, a_1 \rangle, \langle x_2, a_2 \rangle, \cdots, \langle x_n, a_n \rangle$$

Para $x_i \in X(=\Re)$ el problema es determinar todas las funciones de membresía $\wedge$.

Interpolación de Lagrange (método *ajuste de curvas*)

$$f(x) = a_1 L_1(x) + a_2 L_2(x) + \cdots + a_n L_n(x)$$

Donde:

$$L_i(x) = \frac{(x-a_1)\cdots(x-a_{i-1})(x-a_{i+1})\cdots(x-a_n)}{(x_i-a_1)\cdots(x_i-a_{i-1})(x_i-a_{i+1})\cdots(x_i-a_n)}$$

La fórmula fue publicada por Waring (1779), descubierta por Euler (1783) y pública también por Lagrange (1785). Los valores $f(x)$ no necesariamente se hallan en [0, 1) para algunas cierto $x \in \Re$, $f(x)$ puede no ser considerado como una función de membresía, pero se puede convertir f en A para cada $x \in \Re$ de la manera siguiente:

$$A(x) = \max\left[0, \min\left[1, f(x)\right]\right]$$

La desventaja del orden destaca cuando se incrementa el número de datos.

$$\langle 0,0\rangle, \langle 0.5,0.2\rangle, \langle 0.8,0.9\rangle, \langle 1,1\rangle, \langle 1.2,0.9\rangle, \langle 1.5,0.2\rangle, \langle 2,0\rangle$$

$$f(x) = 6.5x^6 + 39x^5 - 109.6x^3 + 64.2x^2 - 13.6x$$

También se puede ocupar el método de ajuste de curvas mediante mínimos cuadrados

$$E = \sum\left[f(x_i;\alpha,\beta\cdots) - a_i\right]^2$$

Controlador difuso-convencional autosintonizable por lógica difusa

La lógica difusa ha surgido recientemente como una herramienta importante para el control de sistemas y procesos industriales complejos. Sin embargo, es muy probable que en aplicaciones caseras encontremos la leyenda "Fuzzy Logic" impresa. El desarrollo de sistemas controlados difusamente está en constante crecimiento, encontrando día a día nuevas aplicaciones sedientas de un control como el difuso. Hornos de microondas, lavadoras de ropa, refrigeradores, medidores de presión arterial, son sólo algunos ejemplos de aplicaciones donde la lógica difusa ha encontrado una tierra fértil.

Lo que la lógica difusa nos permite es encontrar valores intermedios para poder calificar enunciados convencionales como verdaderos o falsos, de esta manera es posible modelar matemáticamente enunciados como "muy caliente", "demasiado frío", etc. y procesarlos con una computadora la cual los evaluará utilizando como base la lógica difusa.

En esencia un controlador lógico difuso contiene un algoritmo que es capaz de convertir una estrategia de control lingüística en una estrategia de control automático. Con la lógica difusa se puede diseñar aplicaciones para que las máquinas respondan con mayor inteligencia a la imprecisión y a las condiciones del mundo exterior, con lo que se busca imitar el comportamiento humano. La creación de una máquina con lógica difusa es forjar un sistema experto, en donde el comportamiento de la máquina va a estar basado totalmente en el conocimiento del experto o de la persona que aporta sus conocimien-

tos empíricos para el funcionamiento de ésta. El conocimiento del experto es el conocimiento empírico de cómo controlar el fenómeno, sin conocer ningún modelo del sistema que va a controlar.

Es posible también que el conocimiento experto sea tomado de una ley de control tan popular como el control PID (Proporcional Integral Diferencial). Este tipo de controlador se utiliza para llevar la salida de una planta a un estado estable. Requiere de una retroalimentación para conocer si existe diferencia alguna entre la entrada al sistema y su salida; en caso de que sí la haya, a ésta se le llama "error". El controlador PID está formado por tres términos, uno de ellos esencial para su funcionamiento y los dos restantes que son complementarios y opcionales. El término esencial es el término proporcional, el cual multiplica directamente al "error" por una constante Kp (ganancia) y la alimenta a la planta; conforme el error decrece, el producto del mismo por la Kp también lo hace. En ocasiones es imposible que un sistema llegue al estado deseado simplemente con un término de Kp; es ahí cuando entra la acción integradora, es decir, el término I. Para esto es necesario integrar el error durante cierto tiempo y sumar a la acción proporcional esta acción integradora. De este modo el error de estado estable se elimina. Por último, la acción derivativa o término "D" es alimentado por la derivada o cambio del error, de este modo el control puede hasta cierto punto predecir la tendencia del sistema para anticiparse y eliminar el sobretiro del sistema. Tanto los términos P, I y D requieren de una ganancia para actuar en forma apropiada.

Método Ziegler-Nichols

Ziegler y Nichols determinaron de manera empírica un método para determinar el valor de las ganancias Ki y Kd en función de la ganancia crítica del sistema con el controlador Kp. Uno de los ejemplos de este trabajo es un controlador PID difuso en donde los criterios de Ziegler y Nichols se usan para determinar el valor de la Ki y Kd del sistema.

Ziegler-Nichols Stability Margin			
Controller Type	K_c	τ_I	τ_D
P	$0.5\ K_{cu}$	—	—
PI	$0.45\ K_{cu}$	$\frac{P_u}{1.2}$	—
PID	$0.6\ K_{cu}$	$\frac{P_u}{2.0}$	$\frac{P_u}{8.0}$

Como se mencionó, el controlador difuso requiere de un experto quien le dirá qué debe hacer para cada tipo de entrada al sistema. Es posible entonces hacer un controlador proporcional-difuso, de manera que sea una ley proporcional la que relacione, en forma difusa, las entradas con las posibles salidas, y si esto es posible, entonces también es posible alimentar al mismo controlador difuso con las entradas del cambio en el error y de la integral del error, para así lograr un controlador PID-difuso; esto es el objetivo principal de este método: el diseño de un controlador PID-difuso.

Controlador proporcional difuso

Se realizó un control P difuso con el propósito de que el controlador sintonizara en tiempo real los valores de Kp. En esta metodología se usó como entrada el error y la salida era el valor de Kp. Las funciones de membresía del error se muestran en la figura 2.93, con cinco funciones gaussianas en un universo de discurso de −1 a 1; las etiquetas son negalto, negmed, cero, posmed, posalto. Y para la salida (Kp), en la figura 2.94 hay tres funciones tipo triangulares con un universo de discurso de 0 a 30, y cuyas etiquetas son cero, med y alto.

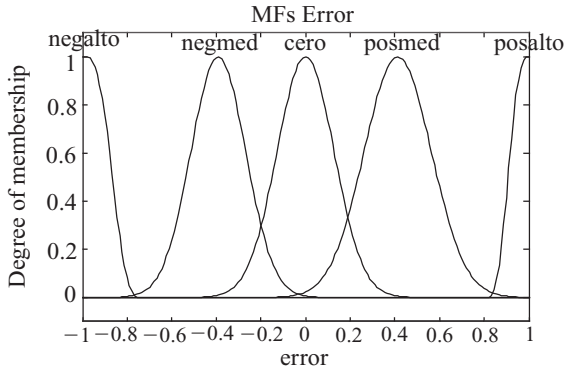


Fig. 2.93 Funciones de membresía del error.

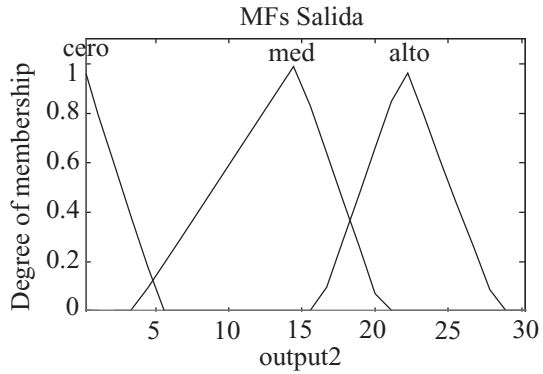


Fig. 2.94 Funciones de membresía de la salida Kp.

Las reglas if-then tipo modus ponens utilizadas comprendían tanto los valores positivos como los negativos, pero donde realmente lo que importa es su valor absoluto; por ello, las reglas son:

Si error es NegAlto entonces salida es Alto
Si error es Cero entonces salida es Cero
Si error es PosAlto entonces salida es Alto
Si error es NegMed entonces salida es Med
Si error es PosMed entonces salida es Med

Cabe señalar que se utilizó un criterio mín-máx para evaluar las reglas y la defuzzificación se realizó con el método del centroide. Como referencia presentamos en la figura 2.95 la evaluación en un momento dado.

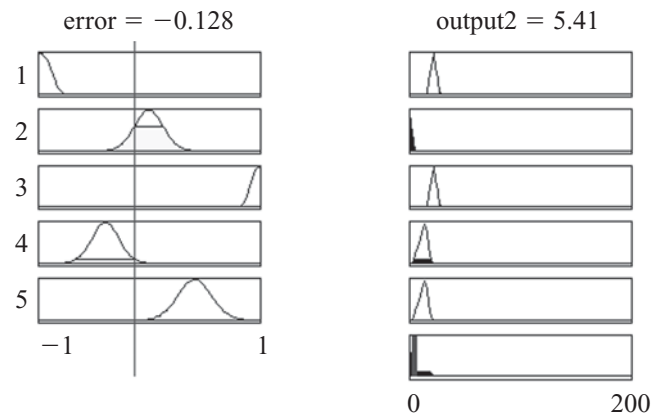


Fig. 2.95 Evaluación de reglas del controlador difuso.

Con el controlador definido, la curva creada por el controlador se muestra en la figura 2.96, donde observamos que no es tan ligero el cambio, pero ello se realizó para incrementar mucho el valor de Kp al tener extremos del error, para hacer aún más rápido al sistema, y esperando trabajar en errores de −0.5 a 0.5.

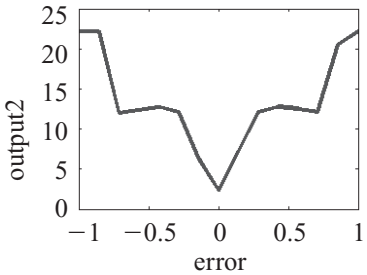


Fig. 2.96 Curva del controlador difuso.

Ya que se hubo configurado el controlador en su totalidad, se implementó con un sistema como el que se muestra en la figura 2.97. La planta se simula con una función de transferencia de segundo orden y que representa un motor de corriente directa. El valor de referencia es 1, con condiciones iniciales cero, por lo que el error máximo esperado es 1.

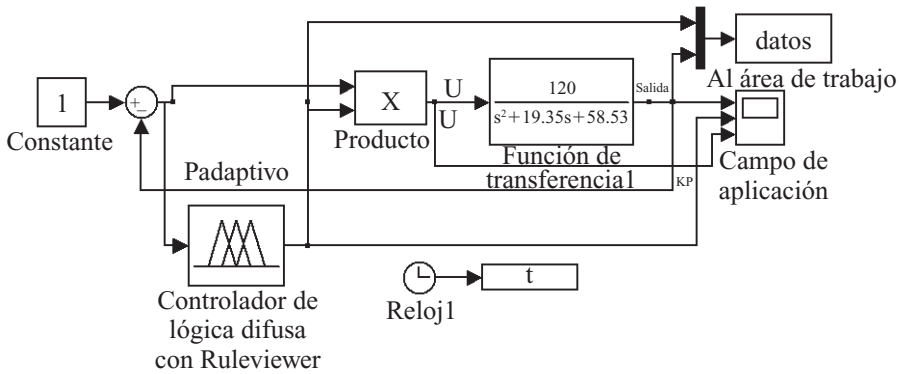


Fig. 2.97 Diagrama de bloques del sistema de control tipo P.

Al simular, encontramos las siguientes gráficas (figura 2.98) que muestran cómo se adapta el valor de Kp sobre todo en el periodo transitorio del sistema, pero que no puede evitar el sobrepaso y presenta un error de estado estable.

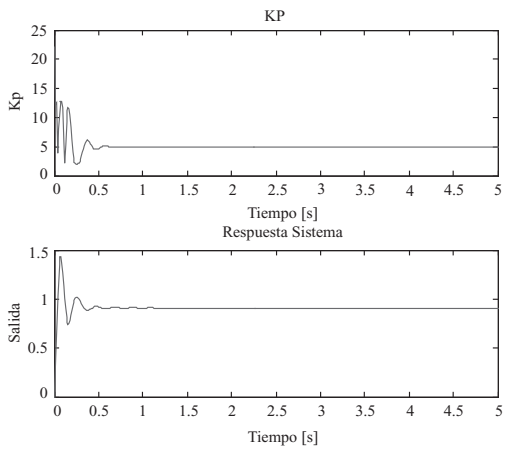


Fig. 2.98 Simulación con el controlador difuso tipo P autosintonizable.

PD autosintonizable

Siguiendo la misma lógica, se realizó un control PD adaptivo mediante dos controladores difusos, uno para estimar el valor de Kp y otro para el de Kd; las características de ambos controladores eran iguales excepto por el universo de discurso de la variable de salida. Para el controlador que emula el valor de Kd, las funciones de membresía se muestran en la figura 2.99.

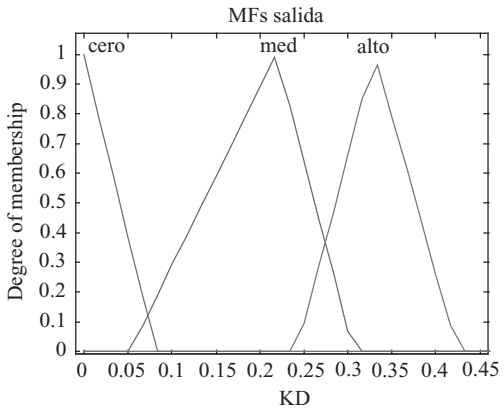


Fig. 2.99 Funciones de membresía de la salida Kd.

En el caso del controlador difuso que emula Kd, la entrada consiste en la derivada del error, como se muestra en la figura 2.100. Al simular, obtenemos las gráficas de la figura 2.101, notando que el valor de Kd también se adapta, trabajando especialmente en la zona transitoria, pero metiendo ruido al sistema. La salida ya no presenta sobreimpulso, como buen controlador PD, pero continúa con un error de estado estable.

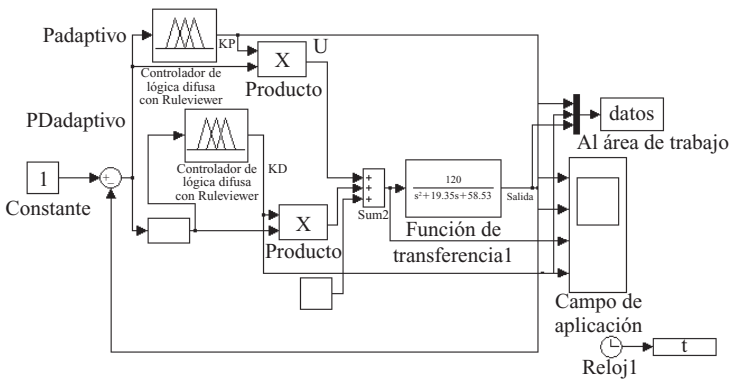


Fig. 2.100 Diagrama de bloques del sistema de control tipo PD.

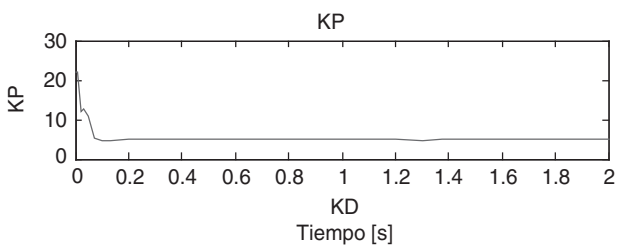


Fig. 2.101 Simulación con el controlador difuso tipo PD autosintonizable. (*continúa*)

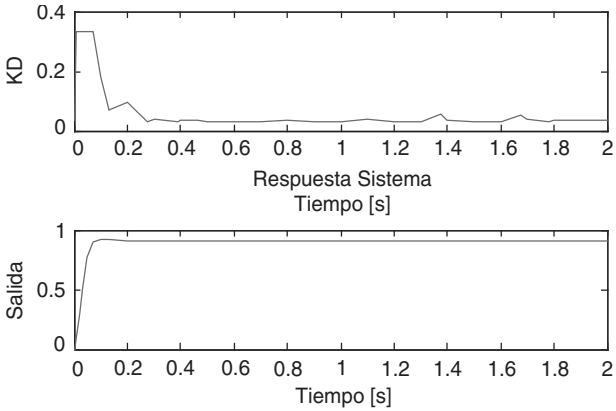


Fig. 2.101 Simulación con el controlador difuso tipo PD autosintonizable. (*continuación*)

PI autosintonizable

De manera idéntica se creó el estimador difuso de Ki, con la particularidad de que su universo de discurso va de 0 a 15 (figura 2.102) y que la entrada será la derivada del error (figura 2.103). La salida muestra que se elimina el error de estado estable, pero se observa sobreimpulso (figura 2.104). Nótese que Ki adquiere importancia conforme el sistema se estabiliza.

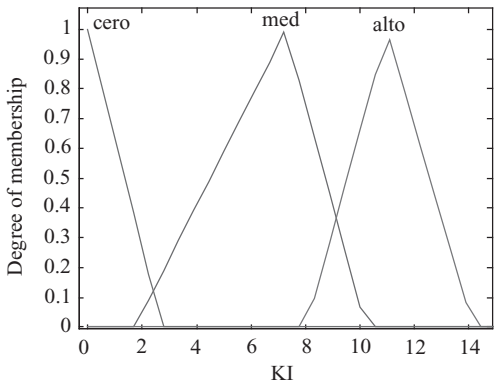


Fig. 2.102 Funciones de membresía de la salida Ki.

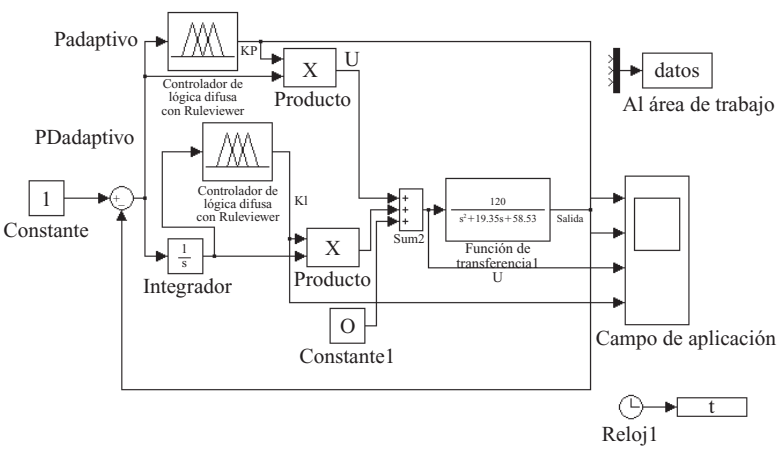


Fig. 2.103 Diagrama de bloques del sistema de control tipo PD.

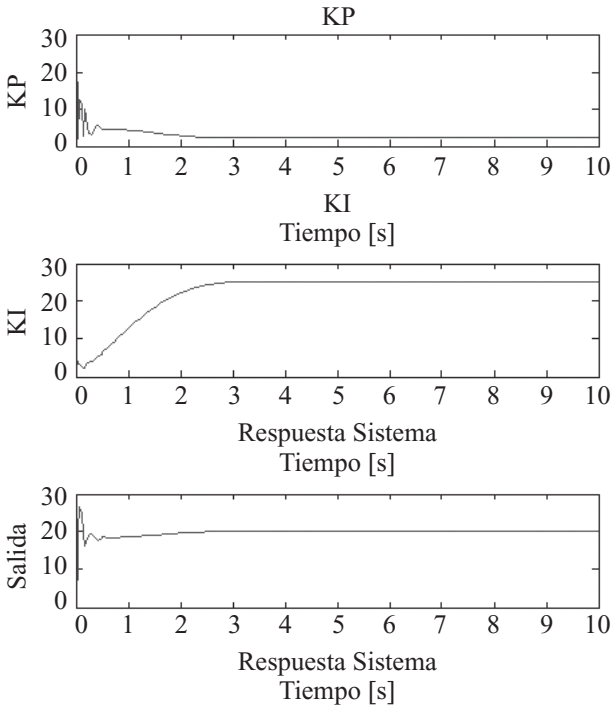


Fig. 2.104 Simulación con el controlador difuso tipo PI autosintonizable.

PID autosintonizable

Por último, con los mismos controladores difusos que estiman Kp, Kd y Ki, sin modificar, se realizó un controlador PID adoptivo como se muestra en el diagrama de la figura 2.105.

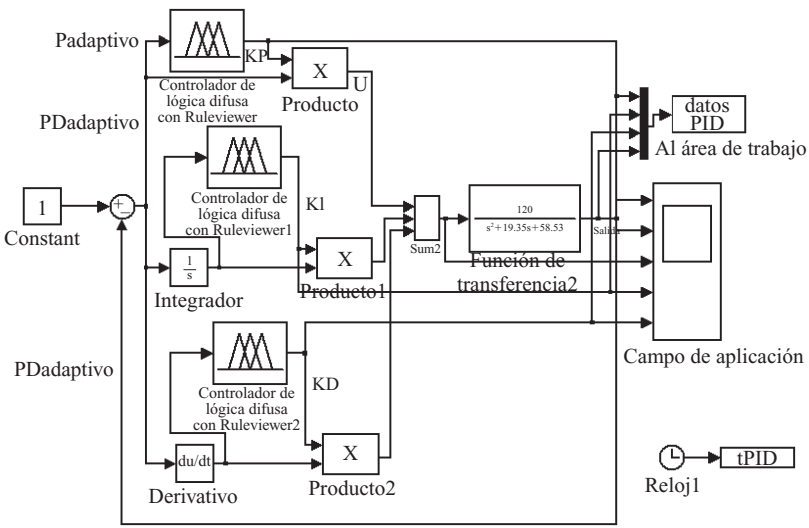


Fig. 2.105 Diagrama de bloques del sistema de control tipo PID.

La respuesta es bastante buena, sin sobreimpulso ni error de estado estable, con los tres controladores autosintonizables funcionando adecuadamente (figura 2.106).

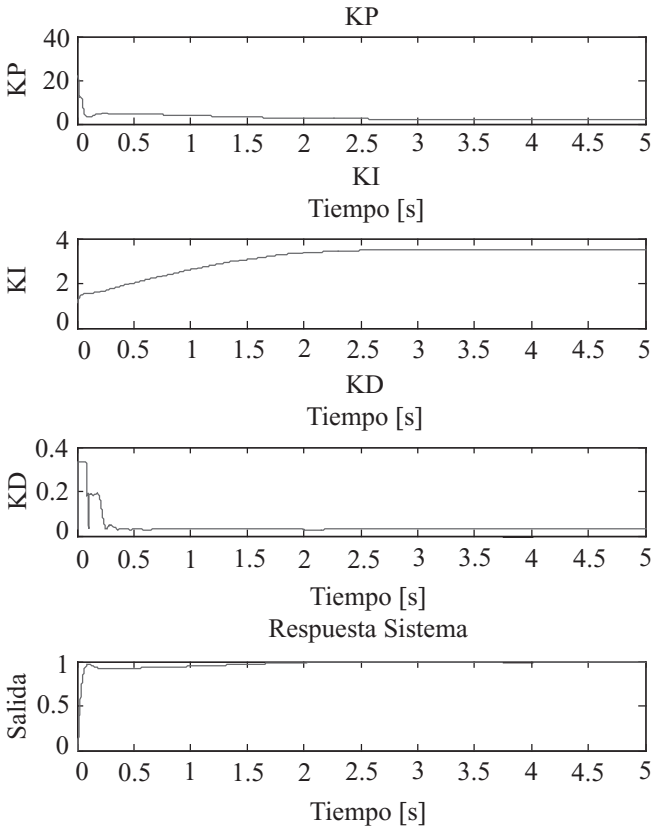


Fig. 2.106 Simulación con el controlador difuso tipo PID autosintonizable.

Análisis de resultados

Las características del controlador P, PD, PI y PID se muestran claramente en la gráfica comparativa de la figura 2.107. Sin lugar a dudas, el mejor control lo realiza el controlador PID.

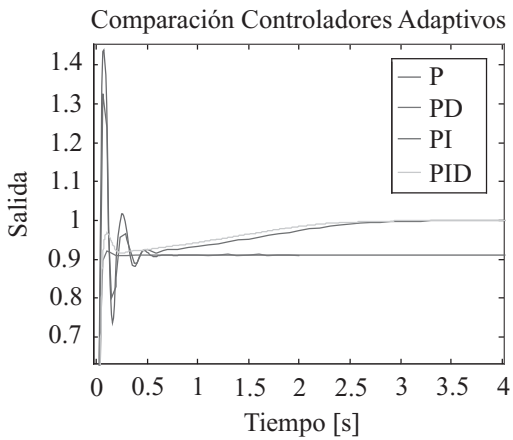


Fig. 2.107 Gráfica comparativa de los cuatro controladores autosintonizables.

También se deseó comparar el controlador PID autosintonizable con uno convencional bien sintonizado. La comparación se muestra en la figura 2.108 y hace notar que el controlador difuso también necesita ser sintonizado (que en este caso no se hizo con gran detalle) en un rango para que funcione en forma ideal.

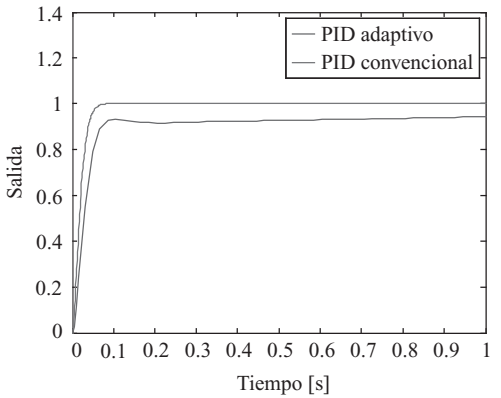


Fig. 2.108 Gráfica comparativa de controladores PID convencional y autosintonizable.

Autosintonización vs. Ziegler-Nichols

Como vimos antes, existen dos maneras de obtener las constantes kd y ki. La primera es mediante un controlador difuso y sus tres etapas, y la otra es mediante los factores establecidos en el método de Ziegler-Nichols.

En este caso estamos aproximando los factores del método de Ziegler-Nichols, ya que para el método exacto no se requiere únicamente la kp, sino también el periodo.

Los resultados que se obtienen para el controlador P son exactamente los mismos ya que no hay variación en la kd y la ki.

Los resultados del controlador PD con el método de Ziegler-Nichols se muestran en la figura 2.110. Cabe señalar que los resultados comparados son para la misma planta y cuentan con el mismo controlador difuso. Al igual que en el control autosintonizable, se obtiene una respuesta sin sobreimpulso, cuya respuesta estable no llega al valor de referencia; sin embargo, la respuesta es significativamente más lenta.

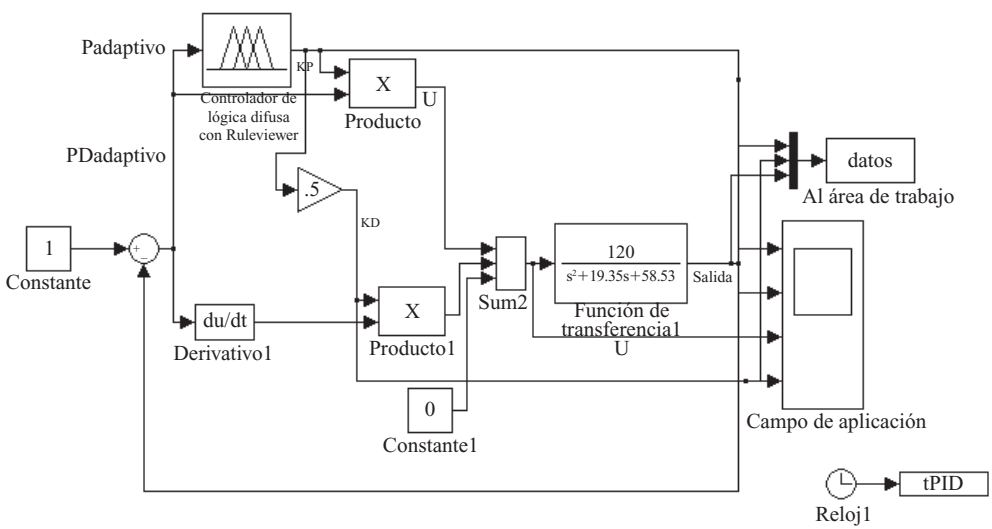


Fig. 2.109 Diagrama de bloques del sistema de control tipo PD mediante Ziegler-Nichols.

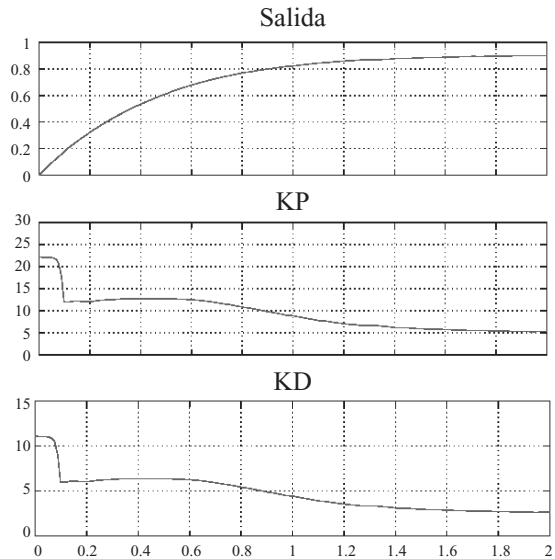


Fig. 2.110 Simulación con el controlador difuso tipo PD por Ziegler-Nichols.

Para el controlador PI con el método de Ziegler-Nichols se obtiene los resultados que se muestran en la figura 2.112. En este caso, la respuesta que se obtiene es muy parecida a la del método anterior, incluyendo el tiempo de respuesta.

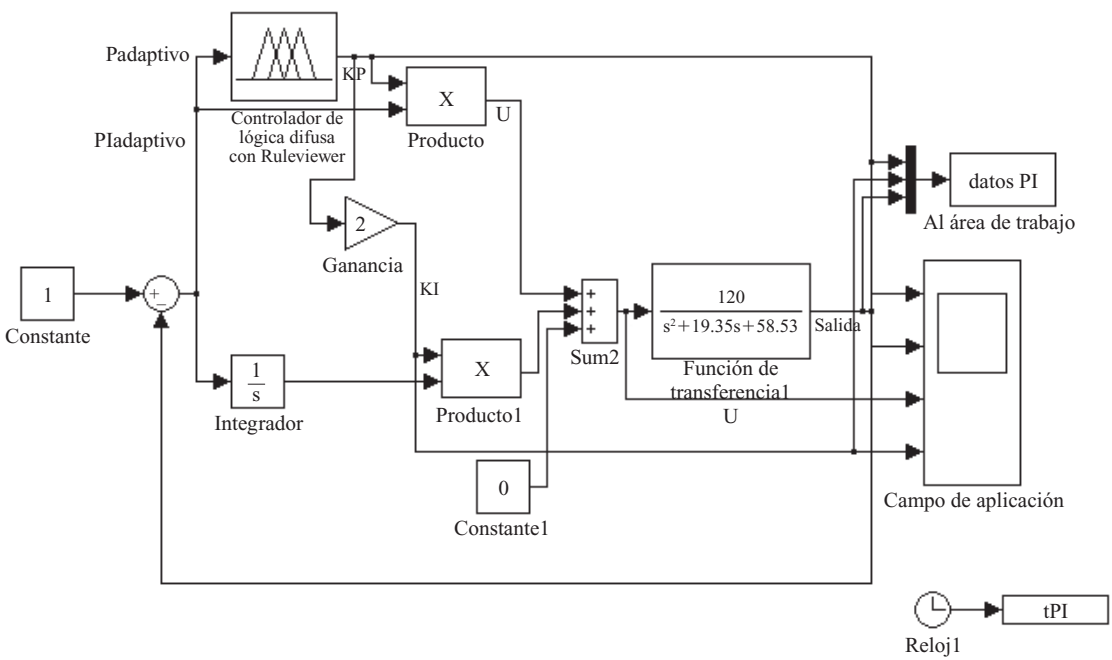


Fig. 2.111 Diagrama de bloques del sistema de control tipo PI mediante Ziegler-Nichols.

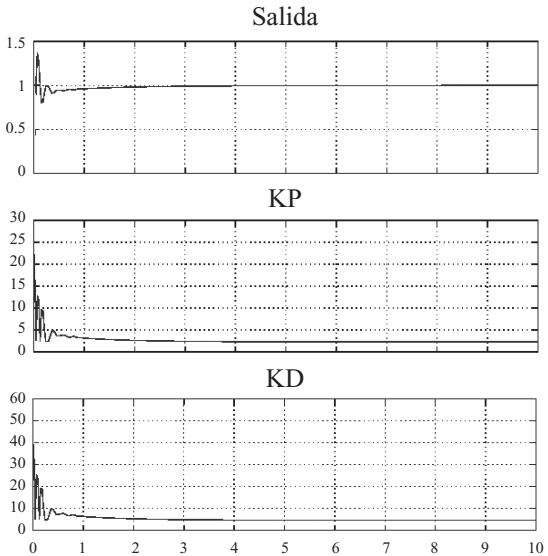


Fig. 2.112 Simulación con el controlador difuso tipo PI por Ziegler-Nichols.

En la figura 2.114 puede observarse los resultados del método de Ziegler-Nichols aplicado para un control PID. Como puede verse, son mucho menos efectivos que el método de autosintonización o un PID tradicional bien sintonizado. Como se aprecia en la gráfica la respuesta incluye un sobreimpulso que no es propio de este tipo de controlador. Considerando este resultado, podríamos decir que no se justifica la aplicación de este método, salvo que los recursos sean realmente limitados y no exista la posibilidad de contar con tres controladores.

Por otra parte, quedaría pendiente analizar la posibilidad de integrar el método de Ziegler-Nichols completo, para poder obtener los resultados más exactos.

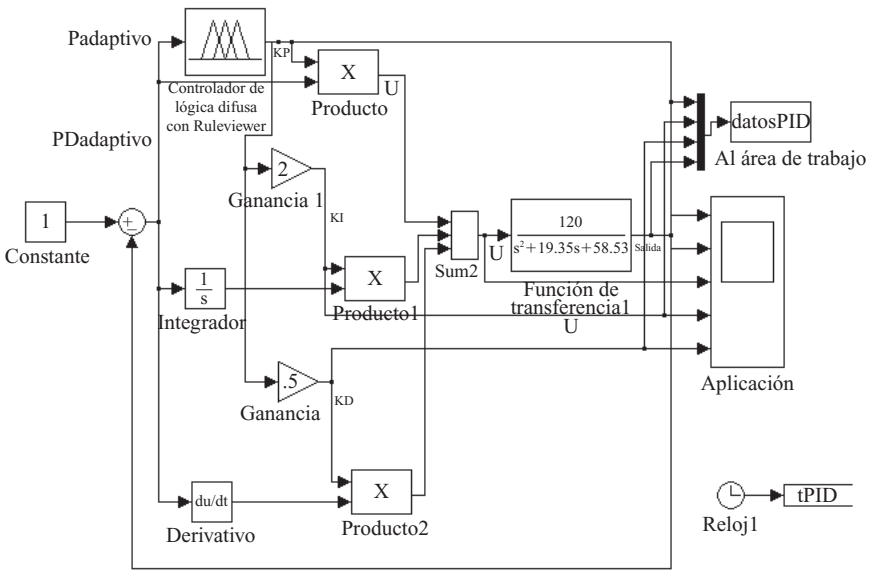


Fig. 2.113 Diagrama de bloques del sistema de control tipo PID mediante Ziegler-Nichols.

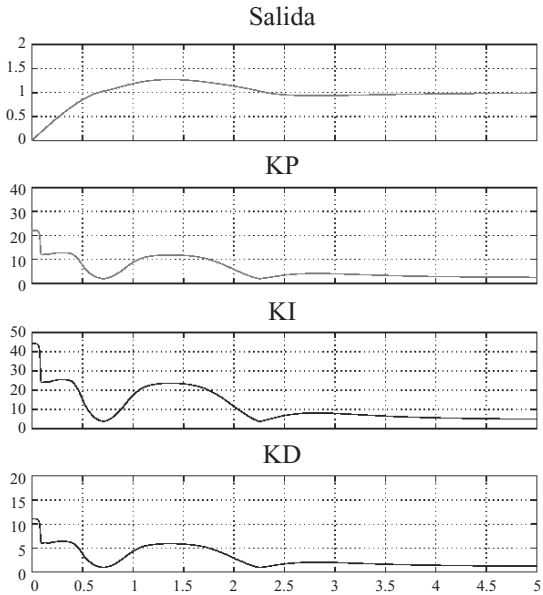


Fig. 2.114 Simulación con el controlador difuso tipo PID por Ziegler-Nichols.

El controlador autosintonizable por medio de lógica difusa funciona correctamente. En métodos convencionales es necesario encontrar valores de Kp, Kd y Ki exactos para un buen funcionamiento, mientras que para el controlador difuso es posible elegir un rango y que la lógica del sistema decida qué valor exacto aplicar. La lógica se obtiene por un experto en controladores PID.

Se comprobó las características de los controladores tipo P, PD, PI, PID, con sus defectos y ventajas:

- El controlador tipo P ayuda a reaccionar rápidamente al sistema, pero provoca sobreimpulso grande y error de estado estable.
- El controlador tipo PD reacciona en forma más lenta pero sin sobrepaso al trabajar en la zona transitoria.
- El tipo PI reacciona rápidamente, con sobreimpulso, pero ahora evitando el error de estado estable.
- La integración de los tres métodos en un PID ayuda al sistema a reaccionar rápidamente, sin sobreimpulso y sin error de estado estable.

La autosintonización difusa de constantes Kp, Kd y Ki puede hacerse de muchas formas; aquí se eligió dos de ellas, ambas eficientes pero con características especiales, a saber:

- El controlador por Ziegler-Nichols tiene la ventaja de necesitar únicamente un controlador difuso, con la desventaja de que las constantes dependen únicamente de dicho controlador, y el método de Ziegler-Nichols es una aproximación que no en todos los casos es óptima.
- El otro controlador es más complejo pues requiere sintonizar tres controles difusos (específicamente el universo de discurso de la variable de salida), pero tiene la ventaja de que cada constante se adapta independientemente con mayor precisión (no está expuesta al error de la aproximación en Ziegler-Nichols).

Los controladores difusos pueden emplearse en diferentes plantas, pero continúa siendo necesario resintonizar, no es universal, pero sí continúa siendo autosintonizable en el rango predeterminado.

Los métodos convencionales y difusos pueden complementarse, no están peleados, y su integración puede resultar en sistemas más eficientes y robustos que por sí mismos.

Controlador difuso como programador de ganancias para PID

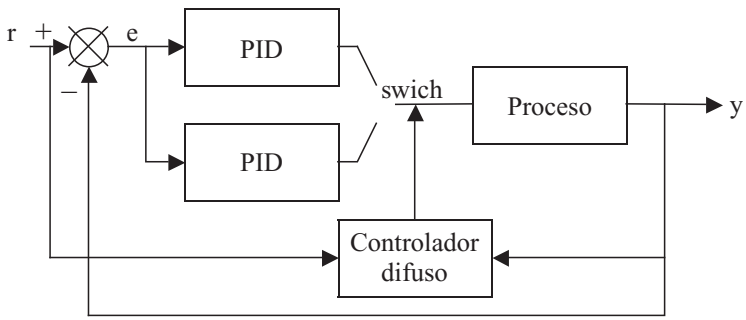


Fig. 2.115 Lazo de un controlador difuso como programador de ganancias para PID.

Estabilidad

Un método para mejorar la estabilidad de un sistema de control difuso consiste en el ajuste de las reglas difusas después de observar la respuesta del sistema, antes de realizar un análisis formal de la estabilidad. En este método, se divide la tabla de reglas difusas en cinco grupos (figura 2.116) de acuerdo con sus efectos en un lazo cerrado. Estos efectos se muestran en la figura 2.117, con su correspondiente diagrama de fase.

Cuando la curva de respuesta exhibe características no deseadas, como un tiempo de subida muy grande, las acciones de alguno de los cinco grupos se ajusta con relación al error.

Error

Derivada del error	NG	NP	ZE	PP	PG
NG	B	B	A	A	A
NP	B	B	A	A	A
ZE	C	C	E	A	A
PP	C	C	C	D	D
PG	C	C	C	D	D

NG = Negativo Grande
NP = Negativo Pequeño
ZE = Cero
PP = Positivo Pequeño
PG = Positivo Grande
CI = Cambio en la entrada del proceso

Fig. 2.116 Reglas divididas en cinco grupos.

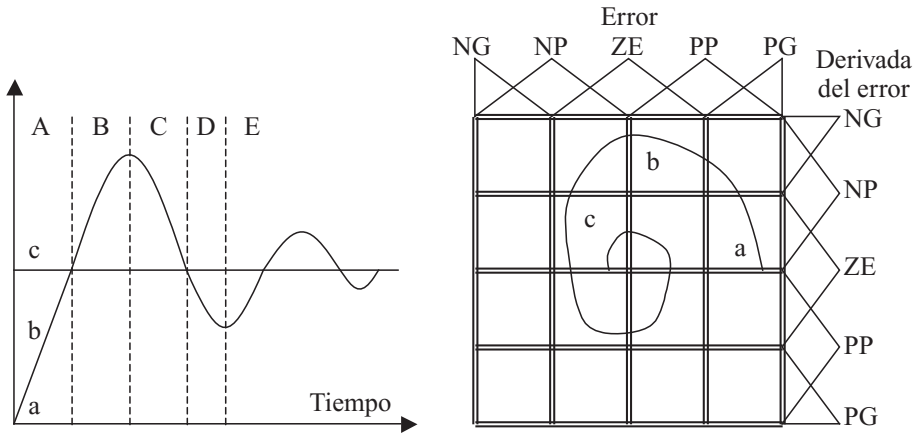


Fig. 2.117 Curva de respuesta y plano de fase.

Diseño con base en Sugeno

Sugeno propuso el diseño de controladores difusos a partir de encontrar un modelo difuso de la planta que relacionara las entradas con sus salidas. Una vez que ya se ha diseñado un modelo difuso de la planta, es posible generar un controlador que parte de la salida deseada para determinar la entrada necesaria en el sistema, con un proceso de deducción inversa.

A continuación se explica un método propuesto por Takagi y Sugeno para la identificación difusa de sistemas y su aplicación en modelado y control. Este es un método matemático para hacer un modelo difuso de un sistema.

El primer paso es definir la estructura de las implicaciones. A continuación se define una implicación difusa (Z):

$$Z: \text{Si } f(x_1 \text{ es } A_1, \ldots, x_k \text{ es } A_k) \text{ entonces } y = g(x_1, ..., x_k)$$

Donde:

y : Variable de la consecuencia, cuyo valor es inferido.

$x_1 ... x_k$: Variables de la premisa, además de que también aparecen en la consecuencia.

$A_1 \ldots A_k$: Conjuntos difusos con funciones de membresía lineales, representando un subespacio difuso en el cual la implicación Z puede ser usada para el razonamiento.

f : Función lógica que conecta las proposiciones en la premisa.

g : Función que implica el valor de y cuando $x_1 ... x_k$ satisface la premisa.

Ejemplo

$$Z: \text{Si } x_1 \text{ es Alto y } x_2 \text{ es Bajo entonces } y = x_1 + x_2 + 6x_3$$

Esta implicación dice que si x_1 es Alto y x_2 es Bajo, entonces y es la suma de x_1, x_2 y $6x_3$, pero x_3 no está condicionado en la premisa.

Entonces, la forma general de una implicación es:

$$Z: \text{Si } x_1 \text{ es } A_1 \text{ y } \ldots \text{ y } x_k \text{ es } A_k \text{ entonces } y = p_0 + p_1 x_1 + ... + p_k x_k$$

Analizando esta estructura, se puede generalizar que en las premisas se usan conectores “and” (y); en la consecuencia se debe usar una función lineal.

ALGORITMO DEL RAZONAMIENTO

Suponiendo que se tienen z^n implicaciones, y además $x_1 \ldots x_k$ son funciones tipo singleton, si se desea inferir la salida "y", ésta se calcula con los siguientes pasos:

1) Para cada implicación Z^i, y^i se calcula por g^i la consecuencia.

$$y^i = g^i(x_1^0, \ldots . x_i^0)$$

$$y^i = p_0^i + p_1^i \cdot x_1^0 + p_2^i \cdot x_2^0 + \ldots . + p_k^i \cdot x_k^0$$

2) El valor de verdad de $y = y^i$ se calcula por:

$$\left| y = y^i \right| = \left| A_1^i(x_1^0) \wedge \ldots . \wedge A_k^i(x_k^0) \right| \wedge \left| Z^i \right|$$

Donde:

$\left| \ \right|$: Valor de verdad

$\wedge$: Se refiere al operador "y", es decir usa la relación mín (máx-mín)

$\left| x^0 es\_A \right| = A(x^0)$

Ahora bien, por simplificación, se asume:

$$\left| R^i \right| = 1$$

3) Generalizando, la salida final inferida, y para n implicaciones está dada como el promedio de todas las y^i con sus respectivos pesos $\left| y = y^i \right|$

$$y = \frac{\sum \left| y = y^i \right| \cdot y^i}{\sum \left| y = y^i \right|}$$

A continuación se verá un ejemplo.

Ejemplo

Si se tiene tres implicaciones:

R_1 : Si x_1 es frío1 y x_2 es frío2 entonces $y = x_1 + x_2$

R_2 : Si x_2 es caliente1 entonces $y = 4x_1$

R_3 : Si x_2 es caliente2 entonces $y = 8x_2$

El razonamiento a seguir para las entradas $x_1 = 3$ y $x_2 = 7$ se puede ver en la tabla 2.25.

Tabla 2.25 Tabla del razonamiento del método Takagi-Sugeno para el ejemplo

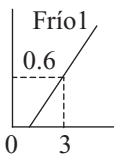
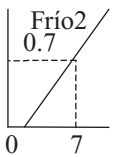
Implicación	Premisa		Consecuencia	Valor de verdad
R_1			$y = 3 + 7 = 10$	$0.6 \wedge 0.7 = 0.6$

Tabla 2.25 Tabla del razonamiento del método Takagi-Sugeno para el ejemplo (*continuación*)

Implicación	Premisa	Consecuencia	Valor de verdad
R_2	Caliente1 0.5 0 3	$y = (4)(3) = 12$	0.5
R_3	Caliente2 0.45 0 7	$y = (8)(7) = 56$	0.45

Ahora bien, la diferencia entre este método y una aproximación lineal es que las conexiones entre las relaciones son más suaves con este método.

Diseño digital con base en estabilidad

Para el diseño digital se consideran reglas para construir un modelo difuso, primer paso del método Takagi-Sugeno, siendo éstas de la forma:

Regla i del modelo: Si $y(k)$ es A_{i1} y . . . y $y(k-n+1)$ es $A_{i(n-1)}$ entonces $y(k+1) = y_i(k+1)$

$$y(k+1) = a_{i0}y(k) + a_{i1}y(k-1) + ... + a_{i(n-1)}y(k-n+1) + b_{i0}u(k) + b_{i1}u(k-1) + ... + b_{im}u(k-m)$$

Donde:

$u(k-m)$: variables de entrada.

$y_i(k+1)$: salida de la regla *I*.

$A_{i1}, ..., A_{i(n-1)}$: niveles difusos.

A partir de esto, se obtiene un modelo difuso:

$$y\left(k+1\right) = \frac{\sum_{i=1}^{r} w_i y_i\left(k+1\right)}{\sum_{i=1}^{r} w_i}$$

Donde:

$w_i = \mu\left(A_{i0}\right) \times ..... \times \mu\left(A_{i(n-1)}\right)$ para $i = 1, 2, ..., r$

$\times$: Denota la operación "y"

$\mu\left(A_{ij}\right)$: Grado de pertenencia de "$y(k-j)$ es A_{ij}" para $j = 0,1,2, \ldots, n-1$

Considerando la propiedad del coeficiente cambiable, a partir de la cual se deriva que

$$y(k+1) = \frac{1}{\sum_{i=1}^{r} w_i}[\sum_{i=1}^{r} w_i a_{i0} y(k) + \sum_{i=1}^{r} w_i a_{i1} y(k-1) + ... + \sum_{i=1}^{r} w_i a_{i(n-1)} y(k-n+1) + \sum_{i=1}^{r} w_i b_{i0} u(k)$$

$$+ \sum_{i=1}^{r} w_i b_{i1} u(k-1) + ... + \sum_{i=1}^{r} w_i b_{im} u(k-m)]$$

$$y(k+1) = a_{d0}y(k) + a_{d1}y(k-1) + ... + a_{d(n-1)}y(k-\mathbf{n}+1) + b_{d0}u(k) + b_{d1}u(k-1) + ... + b_{dm}u(k-m)$$

Donde:

$$a_{dj} = \frac{\sum_{i=1}^{r} w_i a_{ij}}{\sum_{i=1}^{r} w_i} \text{ para } j = 0,1,2, \ldots, n-1$$

$$b_{dl} = \frac{\sum_{i=1}^{r} w_i b_{il}}{\sum_{i=1}^{r} w_i} \text{ para } l = 0,1,2, \ldots, m$$

Estos coeficientes cambian de manera rápida, por lo que el sistema se convierte en altamente no lineal. Cada coeficiente a_{dj}, $j = 0,1,2, \ldots, n-1$ cambia de acuerdo con w_b y w_i, dependiendo de las variables de estado.

La señal de un controlador difuso se define como:

$$u(k) = \frac{\sum_{i=1}^{r} w_i u_i(k)}{\sum_{i=1}^{r} w_i}$$

Y sustituyendo en las ecuaciones anteriores el término $u(k)$ se obtiene un término:

$$\frac{\sum_{i=1}^{r} w_i b_{i0} \sum_{i=1}^{r} w_i u_i(k)}{\sum_{i=1}^{r} w_i \sum_{i=1}^{r} w_i}$$

Esta ecuación implica la multiplicación de dos sumatorias iguales, lo cual se conoce como propiedad de doble sumatoria. El número de subsistemas difusos obtenidos es r^2.

Las reglas de control resultan de la siguiente forma:

Regla de control i: Si < la premisa es > entonces

$$u(k) = \frac{num[u_i(k)]}{den[u_i(k)]}$$

$$u(k) = \frac{\dfrac{\sum_{i=1}^{r} w_i u_i(k)}{\sum_{i=1}^{r} w_i}}{\dfrac{\sum_{i=1}^{r} w_i b_{i0}}{\sum_{i=1}^{r} w_i}} = \frac{\sum_{i=1}^{r} w_i u_i(k)}{\sum_{i=1}^{r} w_i b_{i0}}$$

$$\text{numerador}\,[u_i(k)] = \begin{array}{l}[a_{fi0}y(k) + a_{fi1}y(k-1) + ... + a_{fin}y(k-n) + b_{fi1}u(k-1) \\ \quad + ... + b_{fim}u(k-m) + R]\end{array}$$

$$\text{denominador}\,[u_i(k)] = b_{i0}$$

Donde:

$a_{fij} = a_{cj} - a_{ij}$ para todo $j = 0, 1, 2, ..., n$.

$b_{fil} = b_{cl} - b_{il}$ para todo $l = 0, 1, 2, ..., m$.

a_{cj} y b_{cl} : Coeficientes independientes de *i* que son diseñados para el controlador.

R : Entrada de referencia en el sistema a lazo cerrado.

Sustituyendo la señal de control $u(k)$ en el modelo de la planta $y(k + 1)$ resulta:

$$y(k+1) = \frac{1}{\sum_{i=1}^{r} w_i}[\sum_{i=1}^{r} w_i a_{i0} y(k) + \sum_{i=1}^{r} w_i a_{i1} y(k-1) + ... + \sum_{i=1}^{r} w_i a_{in} y(k-n) + \sum_{i=1}^{r} w_i a_{di0} y(k)$$
$$+ \sum_{i=1}^{r} w_i a_{fi1} y(k-1) + ... + \sum_{i=1}^{r} w_i a_{fin} y(k-n) + \sum_{i=1}^{r} w_i b_{fi1} u(k-1) + ... + \sum_{i=1}^{r} w_i b_{fim} u(k-m)$$
$$+ \sum_{i=1}^{r} w_i b_{i1} u(k-1) + ... + \sum_{i=1}^{r} w_i b_{im} u(k-m) + \sum_{i=1}^{r} w_i R]$$

Agrupando coeficientes:

$$y(k+1) = \frac{1}{\sum_{i=1}^{r} w_i}[\sum_{i=1}^{r} w_i (a_{i0} + a_{fi0}) y(k) + \sum_{i=1}^{r} w_i (a_{i1} + a_{fi1}) y(k-1) + ... + \sum_{i=1}^{r} w_i (a_{in} + a_{fin}) y(k-n)$$
$$+ \sum_{i=1}^{r} w_i (b_{i1} + b_{fi1}) u(k-1) + ... + \sum_{i=1}^{r} w_i (b_{im} + b_{fim}) u(k-m)] + R$$

Considerando que a_{cj} y b_{cl} son independientes de *i*:

$$y(k+1) = \frac{1}{\sum_{i=1}^{r} w_i}[a_{c0} \sum_{i=1}^{r} w_i y(k) + a_{c1} \sum_{i=1}^{r} w_i y(k-1) + ... + a_{cn} \sum_{i=1}^{r} w_i y(k-n) + b_{c1} \sum_{i=1}^{r} w_i u(k-1)$$
$$+ ... + b_{cm} \sum_{i=1}^{r} w_i u(k-m)] + R$$

$$y(k+1) = a_{c0} y(k) + a_{c1} y(k-1) + ... + a_{cn} y(k-n) + b_{c1} u(k-1) + ... + b_{cm} u(k-m) + R$$

Si se elige b_{fil} para que sea 0 el sistema se convierte en:

$$y(k+1) = a_{c0} y(k) + a_{c1} y(k-1) + ... + a_{cn} y(k-n) + R$$

Ejemplo

Teniendo un sistema definido por las siguientes reglas:

Regla 1 del modelo: Si $y(k)$ es M_1 y $y(k-1)$ es M_1 entonces
$$y(k+1) = -2y(k) - 3y(k-1) - y(k-2) + 0.2u(k) + 0.1u(k-1)$$

Regla 2 del modelo: Si $y(k)$ es M_1 y $y(k-1)$ es M_2 entonces
$$y(k+1) = -y(k) - 2y(k-1) - 2y(k-2) + 2u(k) + 1.5u(k-1)$$

Regla 3 del modelo: Si $y(k)$ es M_2 y $y(k-1)$ es M_1 entonces
$$y(k+1) = -y(k) - 2.5y(k-1) - y(k-2) + 0.3u(k) + 0.1u(k-1)$$

Regla 4 del modelo: Si $y(k)$ es M_2 y $y(k-1)$ es M_2 entonces
$$y(k+1) = -1.5y(k) - 2y(k-1) - 2y(k-2) + u(k) + 0.8u(k-1)$$

Los conjuntos M_1 y M_2 pueden observarse en la figura 2.118.

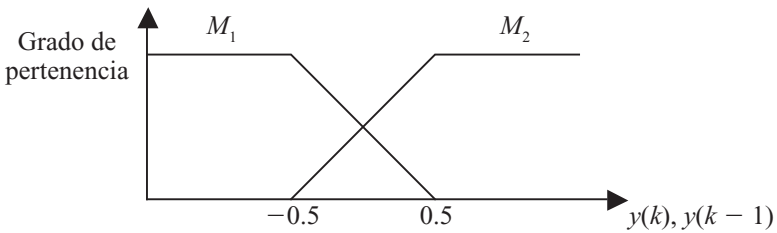


Fig. 2.118 Funciones de pertenencia M_1 y M_2.

De las reglas se obtienen los siguientes coeficientes:

$[a_{10}, a_{11}, a_{12}] = [-2, -3, -1]$, $[b_{10}, b_{11}] = [0.2, 0.1]$
$[a_{20}, a_{21}, a_{22}] = [-1, -2, -2]$, $[b_{20}, b_{21}] = [2, 1.5]$
$[a_{30}, a_{31}, a_{32}] = [-1, -2.5, -1]$, $[b_{30}, b_{31}] = [0.3, 0.1]$
$[a_{40}, a_{41}, a_{42}] = [-1.5, -2, -2]$, $[b_{40}, b_{41}] = [1, 0.8]$

Se desea que el sistema tenga un comportamiento en lazo cerrado descrito como:

$$y(k+1) = 0.3y(k) + 0.2y(k-1) - 0.35y(k-2) + R$$

Con esta información se calculan los coeficientes $a_{c0} = 0.3$, $a_{c1} = 0.2$ y $a_{c3} = -0.35$. Las reglas de control quedarían como se muestra a continuación:

Regla de control 1: Si $y(k)$ es M_1 y $y(k-1)$ es M_1 entonces
$$num[u(k)] = 2.3y(k) + 3.2y(k-1) + 0.65y(k-2) - 0.1u(k-1) + R$$
$$den[u(k)] = 0.2$$

Regla de control 2: Si $y(k)$ es M_1 y $y(k-1)$ es M_2 entonces
$$num[u(k)] = 1.3y(k) + 2.2y(k-1) + 1.65y(k-2) - 1.5u(k-1) + R$$
$$den[u(k)] = 2$$

Regla de control 3: Si $y(k)$ es M_2 y $y(k-1)$ es M_1 entonces

$$num[u(k)] = 1.3y(k) + 2.7y(k-1) + 0.65y(k-2) - 0.1u(k-1) + R$$
$$den[u(k)] = 0.3$$

Regla de control 4: Si $y(k)$ es M_2 y $y(k-1)$ es M_2 entonces

$$num[u(k)] = 1.8y(k) + 2.2y(k-1) + 1.65y(k-2) - 0.8u(k-1) + R$$
$$den[u(k)] = 1$$

EJEMPLO SISTEMA DIFUSO SUGENO

En este trabajo se utiliza MATLAB® para linealizar una curva que representa una función de membresía. Para hacer esto utilizamos el método de mínimos cuadrados que es una aproximación de todos los valores posibles obteniendo las ecuaciones necesarias de las líneas para aplicar el método de Sugeno.

- A partir de la medición de los datos de un sistema real y simular su comportamiento por medio de un sistema difuso utilizando MATLAB®.
- Aplicar un sistema de dos entradas y una salida para un sistema general.

El método de inferencia es uno de los más utilizados. En este trabajo nos ayudamos con el FIS editor de MATLAB® en el modo Mamdani.

Con esta herramienta seguimos los pasos del método comenzando con la definición de las funciones de membresía de la entrada y salida.

Posteriormente pueden fijarse las reglas lingüísticas y observar el comportamiento de la salida de acuerdo con la entrada. También podemos ver la gráfica de relación entrada/salida.

Más adelante veremos cómo varía esa relación de acuerdo con el cambio en la forma de las funciones de membresía de entrada.

De acuerdo con la tabla, en que nos indica en el eje X un rango de -1.5 hasta 1.5 con desplazamientos de 0.5. Para cada uno de estos valores, se tienen salidas desde 0.07 hasta 1. Por lo tanto, se definió la entrada como cada uno de los puntos en X, es decir 7 funciones de membresía triangulares y 4 salidas ya que algunos puntos se repiten.

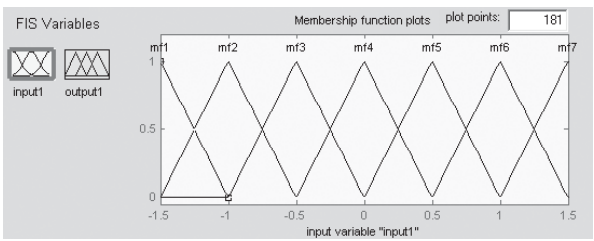


Fig. 2.119 Entrada del sistema difuso.

Por lo tanto, se asignaron las reglas que nos indican a qué salida pertenece cada entrada. Esto se muestra en la figura 2.120.

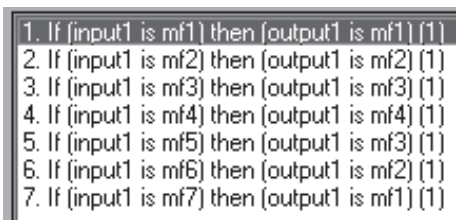


Fig. 2.120 Reglas del sistema difuso.

La primera salida que se propuso fue Singleton, donde las salidas son de manera puntual.

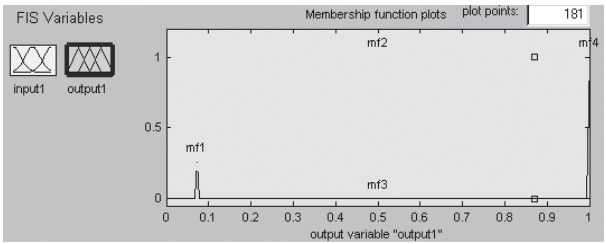


Fig. 2.121 Salida Singleton.

El contorno con esta salida nos da la gráfica de la figura 2.122.

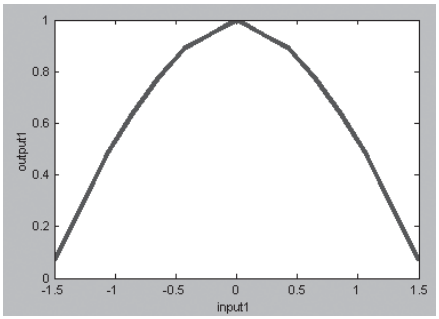


Fig. 2.122 Contorno Singleton.

La siguiente salida que se propuso fue con curvas de Gauss (figura 1.123).

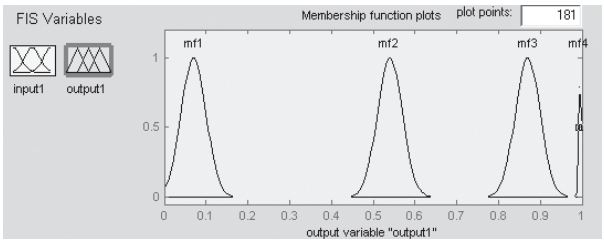


Fig. 2.123 Salida en forma de Gauss.

El contorno que nos da con esta salida es de la siguiente manera (figura 2.124):

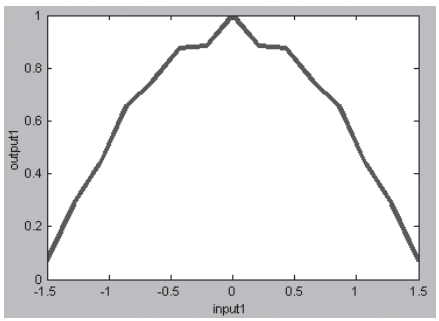


Fig. 2.124 Contorno de Gauss.

La tercera y última salida que se propuso fue con gráficas de tipo PI (figura 2.125).

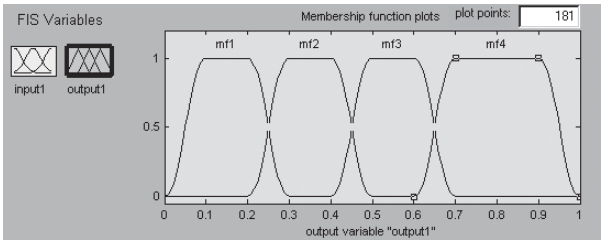


Fig. 2.125 Salida tipo PI.

El contorno que nos da con una respuesta a este tipo es de la siguiente manera (figura 2.126):

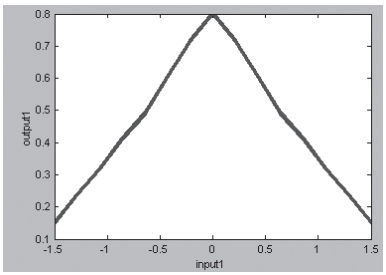


Fig. 2.126 Contorno tipo PI.

Tomando en consideración que los datos obtenidos representan una función coseno, el contorno más próximo a éste fue el de tipo Singleton, de manera puntual.

EJEMPLO DE MOTOR DC

Se obtuvo datos a la respuesta de velocidad de un motor tipo Pittman de la serie 14000 de corriente directa. Esto se hizo utilizando un encoder y un DSP para interpretar los valores del encoder. Los datos se representan en la tabla 2.26.

Tabla 2.26 Tabla de valores de velocidad/tiempo.

Tiempo	Velocidad
0	0
1	27.197157
2	41.19372905
3	48.39683594
4	52.1037971
5	54.01152403
6	54.9933045
7	55.49856175

Tabla 2.26 Tabla de valores de velocidad/tiempo (*continuación*).

Tiempo	Velocidad
8	55.75858414
9	55.89240041

Graficando la velocidad contra el tiempo se obtuvo la gráfica de la figura 2.127.

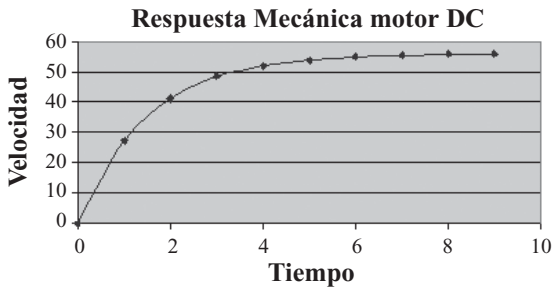


Fig. 2.127 Respuesta mecánica del motor.

Para poder simular el comportamiento del motor de corriente directa de manera difusa, se aplicó el mismo método que en el ejercicio anterior. La entrada del sistema son gráficas triangulares donde el pico se encuentra en cada valor de tiempo medido (cada segundo). Véase la figura 2.128.

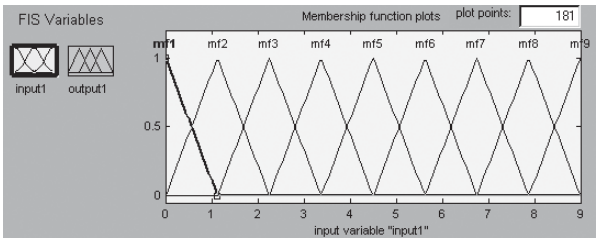


Fig. 2.128 Entrada del sistema difuso.

La salida del sistema se aproximaron funciones triangulares lo más próximo a cada punto medido en la tabla 2.25, por lo que la salida del sistema difuso quedó de la manera que muestra la figura 2.129:

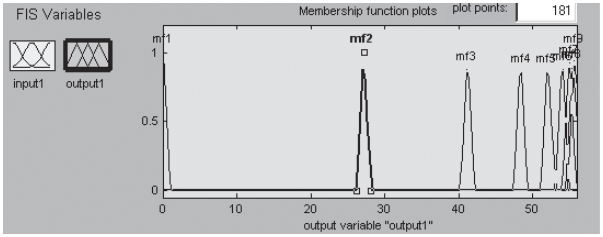


Fig. 2.129 Salida del sistema difuso.

Como reglas del sistema difuso se asignó cada una de las entradas a cada una de las salidas, por lo que las reglas quedaron como muestra la figura 2.130:

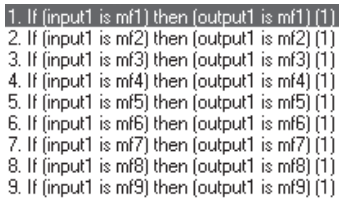


Fig. 2.130 Reglas del sistema difuso.

Utilizando la función de contorno en MATLAB®, se encontró que la respuesta a este sistema es el de la figura 2.131:

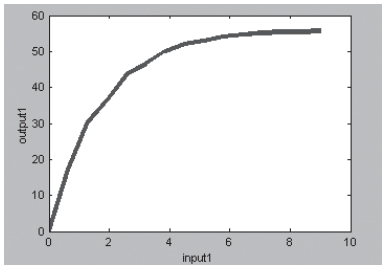


Fig. 2.131 "Contorno del sistema difuso".

Comparando las figuras 2.129 y 2.131 podemos ver claramente que el sistema difuso representa el comportamiento de la respuesta mecánica del motor utilizado de corriente directa.

EJEMPLO DE SISTEMA DE 2 ENTRADAS

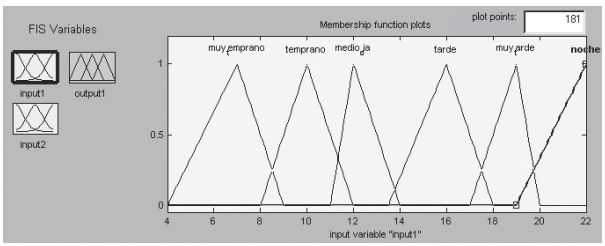


Fig. 2.132 Entrada 1. hora del día.

Nuestra primera entrada es la hora del día donde consideramos las horas desde: muy temprano, temprano, medio día, tarde, muy tarde y noche. Consideramos los rangos de: muy temprano, muy tarde y noche como las horas pico del tráfico en la ciudad de la esperanza. Véase la figura 2.134.

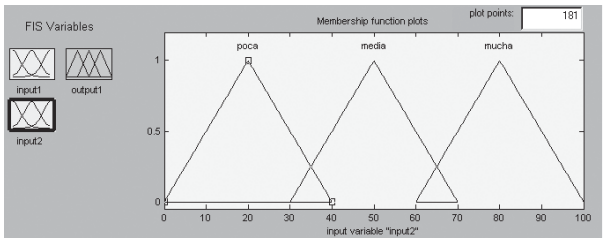


Fig. 2.133 Entrada 2. Humedad relativa.

La segunda es la cantidad de lluvia en el día, considerándola como porcentaje de humedad de 0 a 100. Dividimos esa humedad en poca, media y mucha. Véase la figura 2.133.

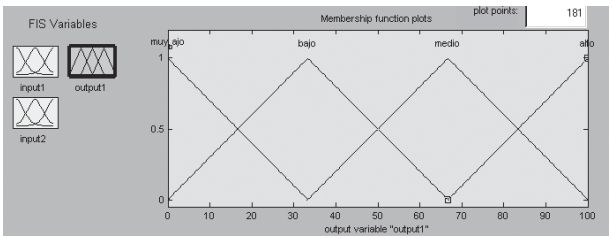


Fig. 2.134 Salida. Porcentaje de tráfico.

La salida está definida como el porcentaje de tráfico encontrado en la ciudad de acuerdo con la hora y cantidad de lluvia, de manera que puede haber tráfico muy bajo, bajo, medio y alto.

Las reglas las definimos de la siguiente manera (figura 2.135):

```
1. If (input1 is muy_temprano) and (input2 is poca) then (output1 is medio) (1)
2. If (input1 is muy_temprano) and (input2 is mucha) then (output1 is alto) (1)
3. If (input1 is temprano) and (input2 is poca) then (output1 is bajo) (1)
4. If (input1 is temprano) and (input2 is mucha) then (output1 is medio) (1)
5. If (input1 is medio_dia) and (input2 is media) then (output1 is medio) (1)
6. If (input1 is medio_dia) and (input2 is poca) then (output1 is muy_bajo) (1)
7. If (input1 is tarde) and (input2 is poca) then (output1 is bajo) (1)
8. If (input1 is tarde) and (input2 is mucha) then (output1 is medio) (1)
9. If (input1 is muy_tarde) and (input2 is poca) then (output1 is medio) (1)
10. If (input1 is muy_tarde) and (input2 is mucha) then (output1 is alto) (1)
11. If (input1 is noche) and (input2 is poca) then (output1 is medio) (1)
12. If (input1 is noche) and (input2 is mucha) then (output1 is alto) (1)
```

Fig. 2.135 Reglas lingüísticas.

En general las reglas se basan en que si tenemos mucha lluvia el tráfico aumenta, y además que si estamos en una hora pico el tráfico es aun mayor (véase la figura 2.136).

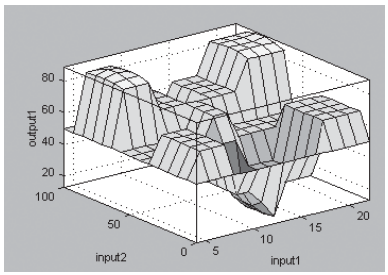


Fig. 2.136 Relación de salida y sus dos entradas.

MÉTODOS DE INFERENCIA

Método de Tsukamoto

Tsukamoto propone el uso de funciones monótonas (ascendentes o descendentes) para los consecuentes, siguiendo el método propuesto por Takagi-Sugeno. La salida de cada regla es un valor concreto generado por la fuerza de activación de dicha regla.

Los modelos que resultan de este método no son muy utilizados, ya que no son transparentes. No obstante, puede utilizarse cuando los datos son muestras.

Método de Larsen

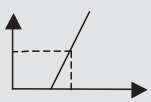
A diferencia del método propuesto por Mamdani o el de Takagi-Sugeno, Larsen propuso en 1980 una multiplicación aritmética entre dos funciones de pertenencia con diferentes universos de discurso. Siendo A y B dos funciones de pertenencia, la implicación de producto resulta:

$$\phi\left[\mu_A(x), \mu_B(y)\right] = \mu_A(x) \cdot \mu_B(y).$$

Resumen de mecanismos de inferencia

A continuación se muestra una tabla que resume las diferencias entre los distintos mecanismos de inferencias propuestos por Mamdani, Larsen, Takagi-Sugeno y Tsukamoto.

Tabla 2.27 Resumen de los mecanismos de inferencia

	Mamdani	Larsen	Takagi-Sugeno	Tsukamoto
Método de implicación	Operador mínimo	Operador producto		
Composición de proposiciones	Operador máximo	Operador máximo		
Activación de reglas $i = 1,2$	$\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$	$\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$	$\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$	$\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$
Salidas individuales de reglas	$C'_1(w) = \alpha_1 \wedge C_1(w)$ $C'_2(w) = \alpha_2 \wedge C_2(w)$ (mínimo)	$C'_1(w) = \alpha_1 C_1(w)$ $C'_2(w) = \alpha_2 C_2(w)$ (producto)	$Z_1^* = a_1 x_0 + b_1 y_0 A$ $Z_2^* = a_2 x_0 + b_2 y_0$	$C'_1(w) = C_1(w)$ $C'_2(w) = C_2(w)$ (nítido)
Salida total del sistema	$C(w) = C'_1(w) \vee C'_2(w)$	$C(w) = C'_1(w) \vee C'_2(w)$		
Salida determinística total	Método de desdifusificación	Método de desdifusificación	$Z_0 = \frac{(\alpha_1 Z_1^* + \alpha_2 Z_2^*)}{(\alpha_1 + \alpha_2)}$	$Z_0 = \frac{(\alpha_1 Z_1^* + \alpha_2 Z_2^*)}{(\alpha_1 + \alpha_2)}$
Base de n reglas, α_1 es el nivel de activación de la regla i, $i = 1,2, \ldots, n$	$C(w) = \bigvee_{i=1}^{n} (\alpha_i \wedge C_i(w))$	$C(w) = \bigvee_{i=1}^{n} (\alpha_i C_i(w))$	$Z_0 = \frac{\sum_{i=1}^{n} a_i Z_i^*}{\sum_{i=1}^{n} a_i}$	$Z_0 = \frac{\sum_{i=1}^{n} a_i Z_i^*}{\sum_{i=1}^{n} a_i}$
Representación gráfica de los métodos				

AGRUPAMIENTOS DIFUSOS

Un agrupamiento o cluster se refiere a la identificación de c subclases en un universo X con n muestras de datos, siendo $2 \leq c < n$, ya que $c = 1$ implica que todos los datos son una sola clase y $c = n$ implica que cada muestra es un cluster por sí misma. Existen dos tipos de particiones de datos: *duros* o nítidos y *suaves* o difusos.

Si es posible medir la distancia entre todos los pares de observación en un espacio, es posible esperar que la distancia entre los puntos del mismo cluster sea considerablemente menor que la distancia entre los puntos de distintos clusters. Uno de los métodos que define particiones óptimas a través de una función de criterio global que mide cuáles particiones candidato optimizan una suma de errores al cuadrado entre puntos y centros de partición en un espacio. Este método aplicado a datos difusos fue propuesto por Bezdek en 1981.

Validez de un cluster

Es necesario identificar el valor de c que proporcione el mejor número de clusters para el análisis que se desea realizar.

Clusters nítidos

Para explicar el método se considera un conjunto de *n* datos a clasificar:

$$X = \{x_1, x_2, ..., x_n\}.$$

Cada muestra x_i es definida por *m* características:

$$x_i = \{x_{i1}, x_{i2}, ..., x_{im}\}.$$

En los clusters duros o *Hard c-Means* cada punto será asignado a uno y sólo un cluster, de tal forma que también se les denomina *particiones*. Definiendo una familia de conjuntos $\{A_i, i = 1, 2, ..., c\}$ como particiones de *X*, se aplican las siguientes expresiones:

$\bigcup_{i=1}^{c} A_i = X$: Familia de conjuntos.

$A_i \cap A_j = \phi$ para todo $i \neq j$: No existe traslape.

$\phi \subset A_i \subset X$ para toda *i*: Las clases no pueden estar vacías y no contienen todas las muestras de datos.

De la misma manera, para el caso de funciones es posible extrapolar las expresiones asociadas:

$\bigvee_{i=1}^{c} \chi_{A_i}(x_k) = 1$ para toda *k*.

$\chi_{A_i}(x_k) \wedge \chi_{A_j}(x_k) = 0$ para toda *k*.

$0 < \sum_{k=1}^{n} \chi_{A_i}(x_k) < n$ para toda *i*.

Donde la función característica es definida como:

$$\chi_{A_i(x_k)} = \begin{cases} 1 & x_k \in A_i \\ 0 & x_k \notin A_i \end{cases}.$$

Una asignación de pertenencia del punto *j* en el cluster *i* se define como $\chi_{ij} (i = 1, 2, ..., c; j = 1, 2, ..., n)$. Y un espacio para una partición dura para *X* se define como la siguiente matriz:

$$M_c = \left\{ U \mid \chi_{ij} \in \{0,1\}, \sum_{i=1}^{c} \chi_{ik} = 1,\ 0 < \sum_{k=1}^{n} \chi_{ik} < n \right\}.$$

Donde *U* es una matriz de *c* renglones y *n* columnas.
La cardinalidad de M_c es

$$\eta_{M_c} = \left(\frac{1}{c!} \left[\sum_{i=1}^{c} \binom{c}{i} (-1)^{c-i} \cdot i^n \right] \right).$$

Donde $\binom{c}{i}$ es el coeficiente binomial de *e* elementos tomando *i* a la vez.

El algoritmo propuesto para las particiones es una aproximación a partir de la suma de los errores cuadrados empleando una norma euclidiana para caracterizar la distancia entre dos puntos. Este algorit-

mo se denota como $J(U, v)$, donde U es la matriz de partición y v un vector de los centros de los clusters. La función objetivo se define entonces como:

$$J(U,v) = \sum_{k=1}^{n}\sum_{i=1}^{c} \chi_{ik}(d_{ik})^2.$$

Donde d_{ik} es una distancia euclidiana medida en un espacio de m dimensiones entre la muestra o punto k y el centro v_i del cluster i, descrita de la forma

$$d_{ik} = d(x_k - v_i) = \|x_k - vi\| = \left[\sum_{j=1}^{m}(x_{ij} - v_{ij})^2\right]^{1/2}.$$

Los m elementos del vector v_i se calculan a partir de

$$v_{ij} = \frac{\sum_{k=1}^{n} \chi_{ik} \cdot x_{kj}}{\sum_{k=1}^{n} \chi_{ik}}.$$

Para encontrar la partición óptima U^* que produce el valor mínimo de la función objetivo J es necesario emplear el siguiente método iterativo:

1. Fijar c tal que $2 \leq c < n$ e inicializar la matriz $U : U^{(0)} \in M_c$.
2. Calcular los vectores de cada centro para los c conjuntos: $\left\{v_i^{(r)} \; con \; U^{(r)}\right\}$, $r = 0,1,2, \ldots$ iteraciones.
3. Actualizar $U^{(r)}$ y calcular las funciones características actuales para toda i, k :

$$\chi_{ik}^{r+1} = \begin{cases} 1 & d_{ik}^{(r)} = \text{mín}\left\{d_{jk}^{(r)}\right\} \text{ para toda } j \in c \\ 0 & \text{de otra forma} \end{cases}.$$

4. Si $\left\|U^{(r+1)} - U^{(r)}\right\| \leq \varepsilon$ (nivel de tolerancia) detener, si no establecer $r = r + 1$ y regresar al paso 2.

Ejemplo

Se desea agrupar el conjunto de puntos de la figura 2.137 en dos clusters:

$x_1 = \{1,1\}$
$x_2 = \{4,1\}$
$x_3 = \{4,2\}$
$x_4 = \{5.5, 1\}$

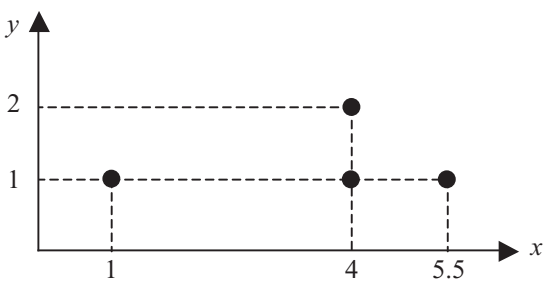


Fig. 2.137 Muestra de puntos que se va a agrupar.

Se ha propuesto una matriz inicial para dos clusters:

$$U^{(0)} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

A continuación se calculan los centroides v_i de cada cluster, los cuales tienen dos elementos cada uno (los puntos tienen una coordenada x y una y):

Cluster 1:

$$v_{11} = \frac{(0)(1)+(0)(4)+(1)(4)+(1)(5.5)}{0+0+1+1} = 4.75$$

$$v_{12} = \frac{(0)(1)+(0)(1)+(1)(2)+(1)(1)}{0+0+1+1} = 1.5$$

$$v_1 = \{4.75, 1.5\}$$

Cluster 2:

$$v_{21} = \frac{(1)(1)+(1)(4)+(0)(4)+(0)(5.5)}{1+1+0+0} = 2.5$$

$$v_{22} = \frac{(1)(1)+(1)(1)+(0)(2)+(0)(1)}{1+1+0+0} = 1$$

$$v_2 = \{2.5, 1\}$$

Ahora se obtienen las distancias entre los centroides y los datos:

Cluster 1:

$$d_{11} = \sqrt{(1-4.75)^2+(1-1.5)^2} = 3.783$$

$$d_{12} = \sqrt{(4-4.75)^2+(1-1.5)^2} = 0.901$$

$$d_{13} = \sqrt{(4-4.75)^2+(2-1.5)^2} = 0.901$$

$$d_{14} = \sqrt{(5.5-4.75)^2+(1-1.5)^2} = 0.901$$

Cluster 2:

$$d_{21} = \sqrt{(1-2.5)^2+(1-1)^2} = 1.5$$

$$d_{22} = \sqrt{(4-2.5)^2+(1-1)^2} = 1.5$$

$$d_{23} = \sqrt{(4-2.5)^2+(2-1)^2} = 1.803$$

$$d_{24} = \sqrt{(5.5-2.5)^2+(1-1)^2} = 3$$

Se actualiza $U^{(1)}$:

Para $k = 1$: mín(d_{11}, d_{21}) = 1.5, por lo que $\chi_{11} = 0$ y $\chi_{21} = 1$
Para $k = 2$: mín(d_{12}, d_{22}) = 0.901, por lo que $\chi_{12} = 1$ y $\chi_{22} = 0$
Para $k = 3$: mín(d_{13}, d_{23}) = 0.901, por lo que $\chi_{13} = 1$ y $\chi_{23} = 0$
Para $k = 4$: mín(d_{14}, d_{24}) = 0.901, por lo que $\chi_{14} = 1$ y $\chi_{24} = 0$

$$U^{(1)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Se recalculan los centroides:

Cluster 1:

$$v_{11} = \frac{(0)(1)+(1)(4)+(1)(4)+(1)(5.5)}{0+1+1+1} = 4.5$$

$$v_{12} = \frac{(0)(1)+(1)(1)+(1)(2)+(1)(1)}{0+1+1+1} = 1.333$$

$$v_1 = \{4.5, 1.33\}$$

Cluster 2:

$$v_{21} = \frac{(1)(1)+(0)(4)+(0)(4)+(0)(5.5)}{1+0+0+0} = 1$$

$$v_{22} = \frac{(1)(1)+(0)(1)+(0)(2)+(0)(1)}{1+0+0+0} = 1$$

$$v_2 = \{1,1\}$$

Y nuevamente las distancias:

Cluster 1:

$$d_{11} = \sqrt{(1-4.5)^2 + (1-1.333)^2} = 3.516$$
$$d_{12} = \sqrt{(4-4.5)^2 + (1-1.333)^2} = 0.6$$
$$d_{13} = \sqrt{(4-4.5)^2 + (2-1.333)^2} = 0.833$$
$$d_{14} = \sqrt{(5.5-4.5)^2 + (1-1.333)^2} = 1.054$$

Cluster 2:

$$d_{21} = \sqrt{(1-1)^2 + (1-1)^2} = 0$$
$$d_{22} = \sqrt{(4-1)^2 + (1-1)^2} = 3$$
$$d_{23} = \sqrt{(4-1)^2 + (2-1)^2} = 3.162$$
$$d_{24} = \sqrt{(5.5-1)^2 + (1-1)^2} = 4.5$$

Se actualiza $U^{(2)}$:

Para $k = 1$: mín(d_{11}, d_{21}) = 0, por lo que $\chi_{11} = 0$ y $\chi_{21} = 1$
Para $k = 2$: mín(d_{12}, d_{22}) = 0.6, por lo que $\chi_{12} = 1$ y $\chi_{22} = 0$
Para $k = 3$: mín(d_{13}, d_{23}) = 0.833, por lo que $\chi_{13} = 1$ y $\chi_{23} = 0$
Para $k = 4$: mín(d_{14}, d_{24}) = 1.054, por lo que $\chi_{14} = 1$ y $\chi_{24} = 0$

$$U^{(2)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Ya que no existe cambio alguno entre $U^{(1)}$ y $U^{(2)}$ el algoritmo se detiene y se ha encontrado una respuesta, donde $v_1 = \{4.5, 1.33\}$ y $v_2 = \{1, 1\}$, para los conjuntos descritos en:

$$U^* = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Clusters difusos

En el caso de los clusters difusos, se define una familia de conjuntos difusos $\left\{\tilde{A}_i, i = 1, 2, ..., c\right\}$ en un universo de puntos X. Es posible asignar una pertenencia a los distintos datos en cada conjunto difuso, de tal forma que el punto k tiene la siguiente pertenencia a la clase i:

$$\mu_{ik} = \mu_{\tilde{A}_i}(x_k) \in [0,1].$$

Adicionalmente, existen las siguientes restricciones extrapoladas de los clusters nítidos:

$$\sum_{i=1}^{c} \mu_{ik} = 1 \text{ para toda } k = 1, 2, \ldots, n.$$

$$0 < \sum_{k=1}^{n} \mu_{ik} < n.$$

$$\bigvee_{i=1}^{c} \mu_{A_i}(x_k) = 1 \text{ para toda } k.$$

Una familia de matrices de particiones difusas M_{fc} se define como:

$$M_{fc} = \left\{ \tilde{U} \mid \mu_{ik} \in [0,1]; \sum_{i=1}^{c} \mu_{ik} = 1; \; 0 < \sum_{k=1}^{n} \mu_{ik} < n \right\}, \quad i = 1, 2, ..., c, \quad k = 1, 2, ..., n.$$

Para este algoritmo se define una función objetivo J_m con una matriz de partición $\tilde{U}$ para agrupar n datos en c clases:

$$J_m(\tilde{U}, v) = \sum_{k=1}^{n} \sum_{i=1}^{c} (\mu_{ik})^{m'} (d_{ik})^2.$$

Donde:

$$d_{ik} = d(x_k - v_i) = \left[\sum_{j=1}^{m} (x_{kj} - v_{ij})^2 \right]^{1/2}.$$

El parámetro m' controla la cantidad de difusificación en el proceso de clasificación, y aunque puede tener valores entre $[1, \infty)$, normalmente se emplean valores de 1 o 2.

Los centros de cada cluster se calculan con la expresión

$$v_{ij} = \frac{\sum_{k=1}^{n} \mu_{ik}^{m'} \cdot x_{kj}}{\sum_{k=1}^{n} \mu_{ik}^{m'}}.$$

El algoritmo se detalla a continuación:

1. Fijar c tal que $2 \leq c < n$, seleccionar m' e inicializar la matriz $\tilde{U}$.

2. Calcular los centros $\left\{v_i^{(r)}\right\}$ para los c conjuntos, $r = 0, 1, 2, \ldots$ iteraciones.
3. Actualizar la matriz de partición $\tilde{U}^{(r)}$ para la iteración r:

$$\mu_{ik}^{(r+1)} = \left[\sum_{j=1}^{c}\left(\frac{d_{ik}^{(r)}}{d_{jk}^{(r)}}\right)^{2/(m'-1)}\right]^{-1}, \text{ para } I_k = \phi$$

$$\text{O } \mu_{ik}^{(r+1)} = 0 \text{ para todas las clases } i \text{ donde } i \in \tilde{I}_k,$$

$$I_k = \left\{i \mid 2 \le c < n; d_{ik}^{(r)} = 0\right\} \tilde{I}_k = \{1, 2, ..., c\} - I_k \sum_{i \in I_k} \mu_{ik}^{(r+1)} = 1$$

4. Si $\left\|\tilde{U}^{(r+1)} - \tilde{U}^{(r)}\right\| \le \varepsilon_L$ detener, si no establecer $r = r + 1$ y regresar al paso 2.

Ejemplo

Para los mismos puntos de la figura 2.137, se propone una matriz inicial para dos clusters:

$$\tilde{U}^{(0)} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

A continuación se calculan los centroides v_i de cada cluster:

Cluster 1:

$$v_{11} = \frac{(0)^2(1) + (0)^2(4) + (1)^2(4) + (1)^2(5.5)}{0^2 + 0^2 + 1^2 + 1^2} = 4.75$$

$$v_{12} = \frac{(0)^2(1) + (0)^2(1) + (1)^2(2) + (1)^2(1)}{0^2 + 0^2 + 1^2 + 1^2} = 1.5$$

$$v_1 = \{4.75, 1.5\}$$

Cluster 2:

$$v_{21} = \frac{(1)^2(1) + (1)^2(4) + (0)^2(4) + (0)^2(5.5)}{1^2 + 1^2 + 0^2 + 0^2} = 2.5$$

$$v_{22} = \frac{(1)^2(1) + (1)^2(1) + (0)^2(2) + (0)^2(1)}{1^2 + 1^2 + 0^2 + 0^2} = 1$$

$$v_2 = \{2.5, 1\}$$

Ahora se obtienen las distancias entre los centroides y los datos:

Cluster 1:

$$d_{11} = \sqrt{(1 - 4.75)^2 + (1 - 1.5)^2} = 3.783$$

$$d_{12} = \sqrt{(4 - 4.75)^2 + (1 - 1.5)^2} = 0.901$$

$$d_{13} = \sqrt{(4 - 4.75)^2 + (2 - 1.5)^2} = 0.901$$

$$d_{14} = \sqrt{(5.5 - 4.75)^2 + (1 - 1.5)^2} = 0.901$$

Cluster 2:

$$d_{21} = \sqrt{(1-2.5)^2 + (1-1)^2} = 1.5$$
$$d_{22} = \sqrt{(4-2.5)^2 + (1-1)^2} = 1.5$$
$$d_{23} = \sqrt{(4-2.5)^2 + (2-1)^2} = 1.803$$
$$d_{24} = \sqrt{(5.5-2.5)^2 + (1-1)^2} = 3$$

Se actualiza $\tilde{U}^{(1)}$ empleando la ecuación para $m' = 2$

$$\mu_{ik}^{(r+1)} = \left[\sum_{j=1}^{c} \left(\frac{d_{ik}^{(r)}}{d_{jk}^{(r)}} \right)^2 \right]^{-1} :$$

$$\mu_{11} = \left[\left(\frac{d_{11}}{d_{11}} \right)^2 + \left(\frac{d_{11}}{d_{21}} \right)^2 \right]^{-1} = \left[\left(\frac{3.783}{3.783} \right)^2 + \left(\frac{3.783}{1.5} \right)^2 \right]^{-1} = 0.136$$

$$\mu_{12} = \left[\left(\frac{d_{12}}{d_{12}} \right)^2 + \left(\frac{d_{21}}{d_{22}} \right)^2 \right]^{-1} = \left[\left(\frac{0.901}{0.901} \right)^2 + \left(\frac{0.901}{1.5} \right)^2 \right]^{-1} = 0.735$$

$$\mu_{13} = \left[\left(\frac{d_{13}}{d_{13}} \right)^2 + \left(\frac{d_{13}}{d_{23}} \right)^2 \right]^{-1} = \left[\left(\frac{0.901}{0.901} \right)^2 + \left(\frac{0.901}{1.803} \right)^2 \right]^{-1} = 0.8$$

$$\mu_{14} = \left[\left(\frac{d_{14}}{d_{14}} \right)^2 + \left(\frac{d_{14}}{d_{24}} \right)^2 \right]^{-1} = \left[\left(\frac{0.901}{0.901} \right)^2 + \left(\frac{0.901}{3} \right)^2 \right]^{-1} = 0.917$$

$$\mu_{21} = \left[\left(\frac{d_{21}}{d_{11}} \right)^2 + \left(\frac{d_{21}}{d_{21}} \right)^2 \right]^{-1} = \left[\left(\frac{1.5}{3.783} \right)^2 + \left(\frac{1.5}{1.5} \right)^2 \right]^{-1} = 0.864$$

$$\mu_{22} = \left[\left(\frac{d_{22}}{d_{12}} \right)^2 + \left(\frac{d_{22}}{d_{22}} \right)^2 \right]^{-1} = \left[\left(\frac{1.5}{0.901} \right)^2 + \left(\frac{1.5}{1.5} \right)^2 \right]^{-1} = 0.265$$

$$\mu_{23} = \left[\left(\frac{d_{23}}{d_{13}} \right)^2 + \left(\frac{d_{23}}{d_{23}} \right)^2 \right]^{-1} = \left[\left(\frac{1.803}{0.901} \right)^2 + \left(\frac{1.803}{1.803} \right)^2 \right]^{-1} = 0.2$$

$$\mu_{24} = \left[\left(\frac{d_{24}}{d_{14}} \right)^2 + \left(\frac{d_{24}}{d_{24}} \right)^2 \right]^{-1} = \left[\left(\frac{3}{0.901} \right)^2 + \left(\frac{3}{3} \right)^2 \right]^{-1} = 0.083$$

$$\tilde{U}^{(1)} = \begin{bmatrix} 0.136 & 0.735 & 0.8 & 0.917 \\ 0.864 & 0.265 & 0.2 & 0.083 \end{bmatrix}$$

Se recalculan los centroides:

Cluster 1:

$$v_{11} = \frac{(0.136)^2(1) + (0.735)^2(4) + (0.8)^2(4) + (0.917)^2(5.5)}{0.136^2 + 0.735^2 + 0.8^2 + 0.917^2} = 4.591$$

$$v_{12} = \frac{(0.136)^2(1) + (0.735)^2(1) + (0.8)^2(2) + (0.917)^2(1)}{0.136^2 + 0.735^2 + 0.8^2 + 0.917^2} = 1.313$$

$$v_1 = \{4.591, 1.313\}$$

Cluster 2:

$$v_{21} = \frac{(0.864)^2(1) + (0.265)^2(4) + (0.2)^2(4) + (0.083)^2(5.5)}{0.864^2 + 0.265^2 + 0.2^2 + 0.083^2} = 1.418$$

$$v_{22} = \frac{(0.864)^2(1) + (0.265)^2(1) + (0.2)^2(2) + (0.083)^2(1)}{0.864^2 + 0.265^2 + 0.2^2 + 0.083^2} = 1.046$$

$$v_2 = \{1.418, 1.046\}$$

El algoritmo continuará hasta que $\left\|\tilde{U}^{(r+1)} - \tilde{U}^{(r)}\right\| \leq \varepsilon_L$.

Aplicaciones reales de los agrupamientos difusos

Para clasificar células de cierta parte del cuerpo como el cuello del útero, y de esta forma poder determinar la presencia de células cancerígenas, es posible realizar clasificaciones mediante métodos de agrupamientos que se apoyen en bases de datos de características celulares, a partir de imágenes tomadas de muestras. Diversos programas computacionales como CHAMP miden las características de las células, y la clasificación se ha realizado experimentalmente con diversos métodos para la detección del cáncer cervico-uterino:[2] Fuzzy C-Means, agrupamientos de Gustafson-Kessel, agrupamientos del Vecino Cercano, redes neuronales tipo ANFIS.

Para analizar las células, en primer lugar se preparan los especímenes y posteriormente se escanean en forma digital para ser procesados por la computadora. A continuación se buscan las células que se van a analizar y se segmentan en núcleo y citoplasma. Se miden las diversas características de las células (área, color, forma) y se clasifican según una base de datos proporcionada por el experto conforme a sus características. De acuerdo con la clasificación es posible aplicar métodos de agrupamientos y determinar la presencia de células cancerígenas.

Los dispositivos eléctricos para controlar el flujo y el par de un motor de corriente directa se han usado comúnmente en la industria, siendo los más frecuentes aquellos que contienen un controlador de velocidad de lazo cerrado. Sin embargo, los sensores de velocidad son caros y espaciosos, por lo que emplear lazos abiertos reduce el costo y la complejidad del sistema. Empleando técnicas de control vectorial es posible obtener modelos para motores de inducción que resultan similares a los de motores de corriente directa.

Se ha diseñado diversos controladores difusos para dispositivos eléctricos de motores de corriente directa, que implican el control de velocidad en lazo abierto, los cuales se han empleado en aplicaciones industriales y de investigación, como el control de velocidad y posición de robots móviles.

[2] Jens Byriel (1999). *Neuro-fuzzy Classification of Cells in Cervical Smears*. Technical University of Denmark.

Un ejemplo de estos controladores difusos se logró implementar a través de la obtención del modelo difuso de un dispositivo eléctrico para un motor de corriente directa, mediante el método de Sugeno. Posteriormente, se diseñó el controlador con base en el modelo del dispositivo eléctrico. La figura 2.138 muestra la curva de velocidad del motor empleado.

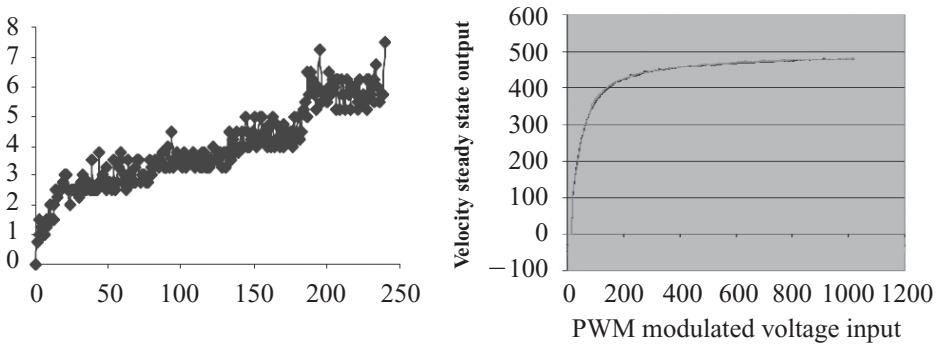


Fig. 2.138 Curva de velocidad del motor de corriente directa.

Posteriormente, en el diseño del controlador difuso se empleó el método de agrupamientos difusos Fuzzy C-Means para sintonizar tres funciones de pertenencia de la entrada, teniendo como resultado el que muestra la figura 2.139.

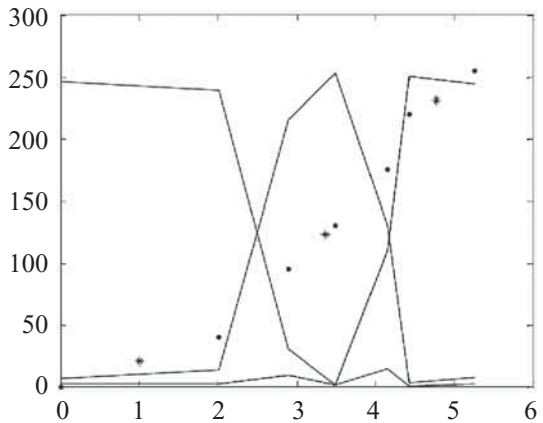


Fig. 2.139 Funciones de pertenencia obtenidas por el método de agrupamientos difusos Fuzzy C-Means.

Finalmente, se simuló el sistema total propuesto en la figura 2.140, obteniendo los resultados para distintos escalones de entrada de la figura 2.141.

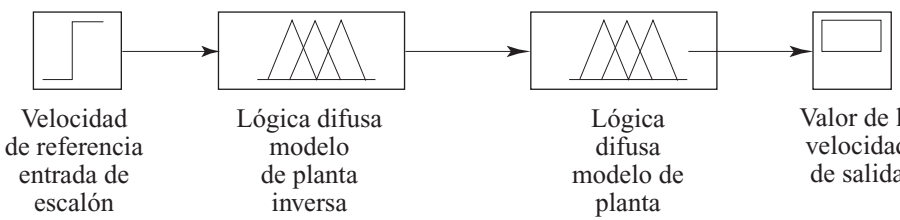


Fig. 2.140 Sistema propuesto para el control a lazo abierto de un motor de corriente directa.

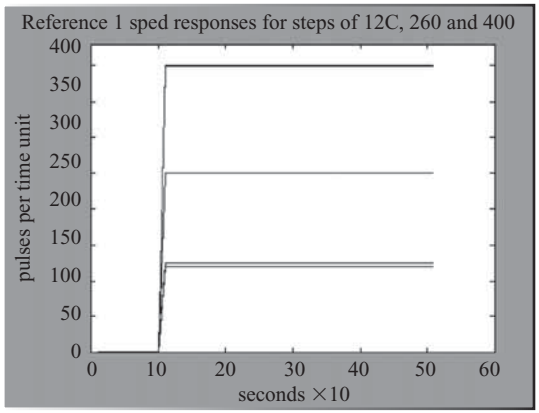


Fig. 2.141 Respuestas a escalón obtenidas con sus respectivas referencias para el sistema propuesto.

Entre las aplicaciones posteriores del controlador propuesto se encuentra el control de navegación autónoma de un hexápodo, incorporando redes neuronales.

Aproximaciones de sistemas reales por el método de Sugeno

A continuación se muestran diversas aproximaciones de funciones matemáticas o sistemas reales obtenidas mediante el uso del método Sugeno antes mencionado.

Aproximación de funciones matemáticas

Se aproximó la función $y = x^3 - 10x^2$ empleando nueve funciones de pertenencia de la tabla 2.28 y aproximando las funciones de salida de las reglas difusas mediante el método de mínimos cuadrados, con el resultado que se ve en la figura 2.142.

Tabla 2.28 Puntos característicos de la función y funciones de pertenencia empleadas para la obtención del modelo difuso

Funciones de pertenencia	*x*	*y*
1	−10	−2000
2	−7.5	−984.375
3	−5	−375
4	−2.5	−78.125
5	0	0
6	2.5	−46.875
7	5	−125
8	7.5	−140.625
9	10	0

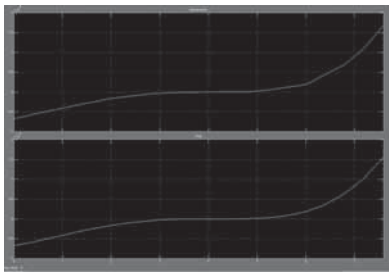


Fig. 2.142 Inferior: función propuesta. Superior: modelo difuso con base en Sugeno.

Se aproximó la función $y = \text{sen } x$ empleando nueve funciones de pertenencia de la tabla 2.29 y aproximando las funciones de salida de las reglas difusas mediante el método de mínimos cuadrados, con el resultado que se muestra en la figura 2.143.

Tabla 2.29 Puntos característicos de la función y funciones de pertenencia empleadas para la obtención del modelo difuso

Funciones de pertenencia	*x*	*y*
1	0	0
2	0.7	0.6442
3	1.4	0.9854
4	2.1	0.8632
5	2.8	0.3350
6	3.5	−0.3508
7	4.2	−0.8716
8	4.9	−0.9825
9	5.6	−0.6313

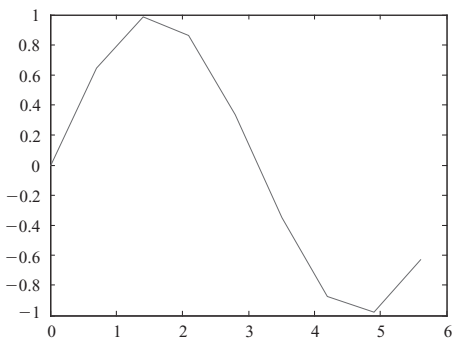


Fig. 2.143 Modelo difuso obtenido con base en Sugeno.

Se aproximó la función $z = \sqrt{x^2 + y^2}$ empleando once funciones de pertenencia de la tabla 2.30 y la figura 2.144, y aproximando las funciones de salida de las reglas difusas mediante el método de mínimos cuadrados, con el resultado que muestra la figura 2.145 en forma de superficies.

Tabla 2.30 Puntos característicos de la función y funciones de pertenencia empleadas para la obtención del modelo difuso

Funciones de pertenencia	*x*	*y*	*z*
1	0	−1	1
2	0.1	−0.8	0.80623
3	0.2	−0.6	0.63246
4	0.3	−0.4	0.5
5	0.4	−0.2	0.44721
6	0.5	0	0.5
7	0.6	0.2	0.63246
8	0.7	0.4	0.80623
9	0.8	0.6	1
10	0.9	0.8	1.2042
11	1	1	1.4142

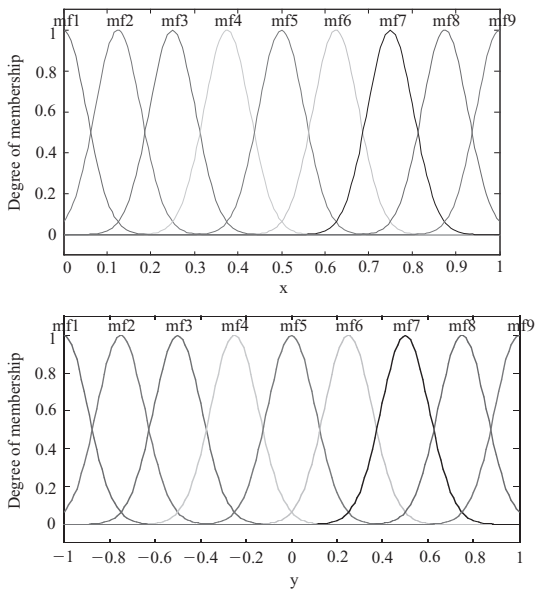


Fig. 2.144 Funciones de pertenencia propuestas para las entradas *x* y *y*.

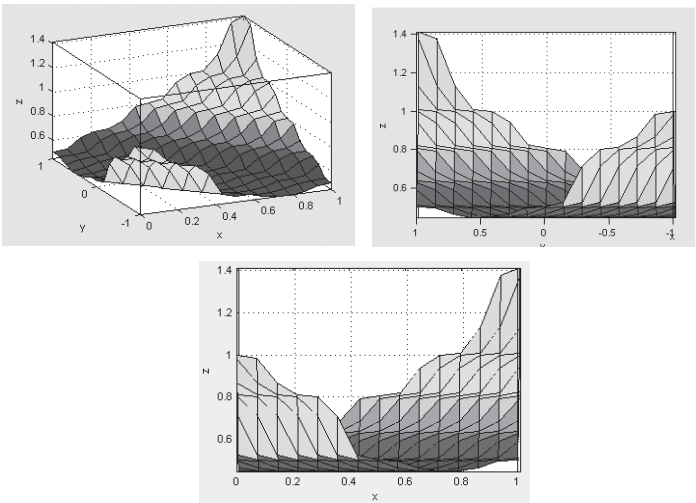


Fig. 2.145 Resultados vistos a manera de curvas.

Aproximación de un deshumidificador desecante

El deshumidificador usa un material desecante como el gel de silicio. El aire que proviene de un proceso pasa a través de la rueda y el vapor de agua que contiene queda atrapado en el material. En la cámara de reactivación se aplica aire caliente a la rueda para evaporar la humedad que contiene (reactivación de la rueda desecante). Las partes de un deshumidificador desecante se ven en la figura 2.146.

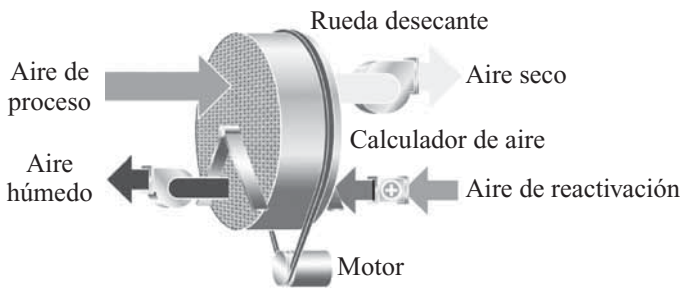


Fig. 2.146 Componentes de un deshumidificador desecante.

Se plantearon nueve funciones de pertenencia a partir de los puntos característicos del funcionamiento del aparato contenidos en la tabla 2.31, cuyo resultado se muestra en la figura 2.147.

Tabla 2.31 Puntos característicos de la función y funciones de pertenencia empleadas para la obtención del modelo difuso

Funciones de pertenencia	Temperatura de entrada	Humedad de salida
1	55	7.8
2	60	8.9
3	65	11

Tabla 2.31 Puntos característicos de la función y funciones de pertenencia empleadas para la obtención del modelo difuso (*continuación*)

Funciones de pertenencia	Temperatura de entrada	Humedad de salida
4	70	15
5	75	19
6	80	22
7	85	25
8	90	28
9	95	30

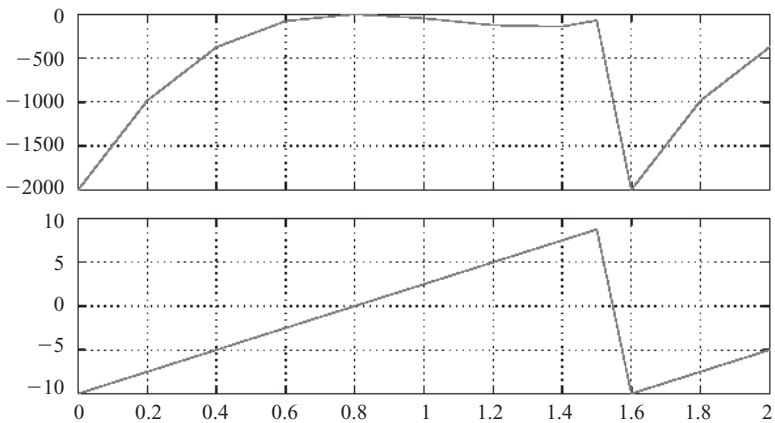


Fig. 2.147 Modelo difuso obtenido con base en Sugeno.

Aproximación de un potenciómetro

Considerando la curva de un potenciómetro de la figura 2.148, que relaciona el ángulo de giro con la resistencia, se consideraron ocho puntos (tabla 2.32) característicos para formar funciones de pertenencia respectivas (figura 2.149). El resultado de la aproximación se observa en la figura 2.150.

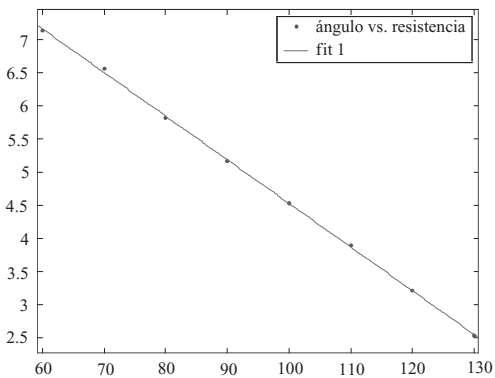


Fig. 2.148 Curva del potenciómetro.

Tabla 2.32 Puntos característicos de la función y funciones de pertenencia empleadas para la obtención del modelo difuso

Funciones de pertenencia	Ángulo	Resistencia en kΩ
1	60	7.13
2	70	6.56
3	80	5.81
4	90	5.16
5	100	4.53
6	110	3.89
7	120	3.21
8	130	2.53

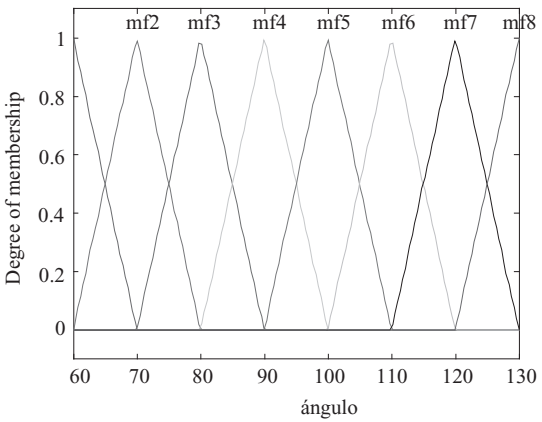


Fig. 2.149 Funciones de pertenencia propuestas.

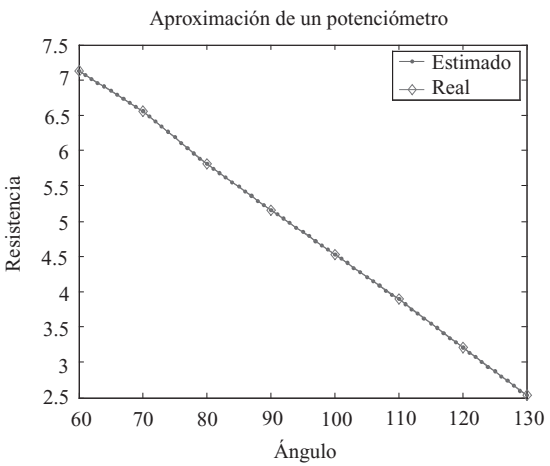


Fig. 2.150 Modelo difuso obtenido.

Aproximación de un sensor óptico

Algunos robots emplean sensores ópticos para determinar de manera indirecta su inclinación. Este proceso es no lineal debido a que se emplean convertidores analógicos digitales y existen diversos efectos ópticos de reflejo y de temperatura. La figura 2.151 muestra el funcionamiento de un sensor óptico para determinar la inclinación.

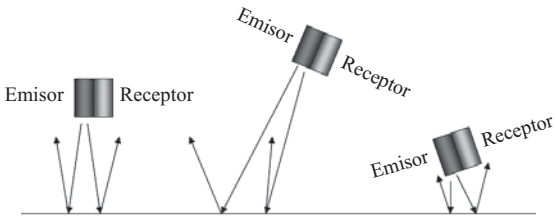


Fig. 2.151 Funcionamiento del sensor óptico para conocer la inclinación de un robot.

Para la obtención del modelo difuso del sensor se graficó la posición angular del servo contra la salida del sensor (figura 2.152) y se tomó valores de pruebas para determinar las funciones de pertenencia. El resultado del modelo puede apreciarse en la figura 2.153.

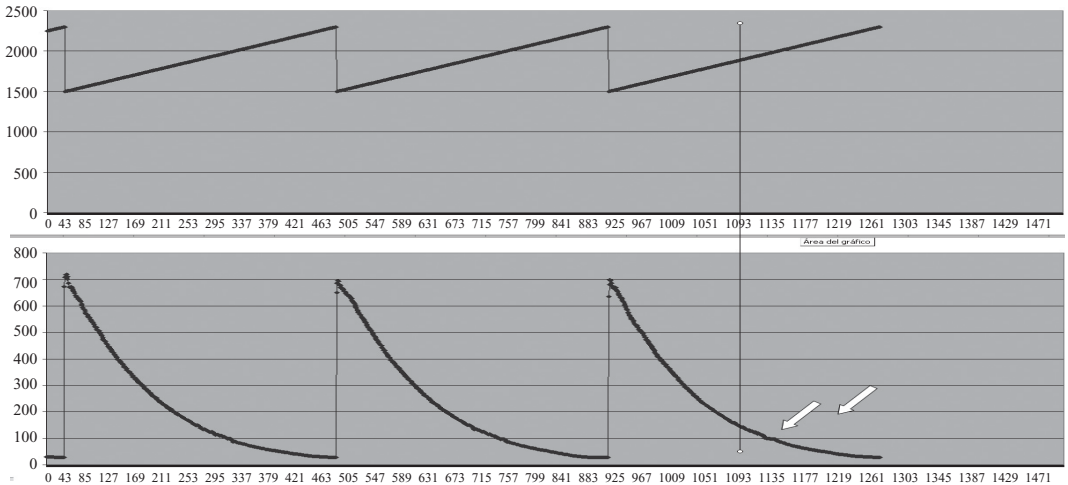


Fig. 2.152 Gráficas de la posición angular respecto a la salida del sensor.

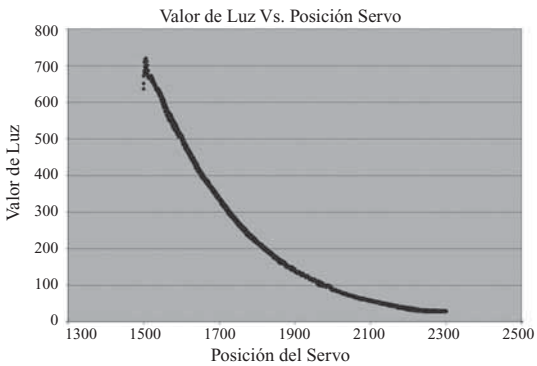


Fig. 2.153 Modelo difuso de un sensor óptico.

Ejemplo de aplicación de método para optimización de clusters con lógica difusa tipo Mamdani

Las apuestas deportivas tienen un gran auge en nuestro país, pero sobre todo las carreras de caballos, ya que es una de las pocas que pueden verse en vivo en el Hipódromo de las Américas de la ciudad de México. La pista de dicho hipódromo es un óvalo con una longitud total de 1508 m, con 326.2 m en las rectas y 428 m en las curvas; el escape mayor tiene 296.6 m y el menor 140 m, lo que permite realizar carreras de hasta una milla de distancia. Asimismo, un ancho de 25 m con un ligero peralte, y dos caídas: una hacia el exterior de 7 m de ancho y otra hacia el interior de 18 m de ancho. El riel externo es tubular de 5 pulgadas de diámetro de aluminio, el interno es de forma semicircular de 20 pulgadas de ancho del mismo material para absorber el posible impacto de algún jinete.

La figura 2.154 muestra de manera gráfica lo anterior:

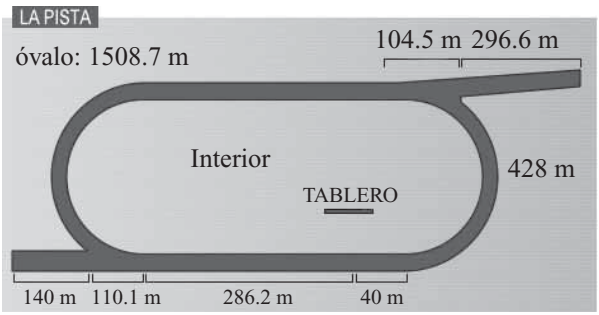


Fig. 2.154 Pista del Hipódromo de las Américas.

Existen distintos tipos de apuestas que se pueden realizar en las diferentes carreras, las cuales se dividen en apuestas directas y apuestas exóticas. Las directas son Primeras, Segundas y Terceras. Las apuestas exóticas son Exacta, Quinela, Trifecta, Triple, Superfecta, Ve por cuatro, Ve por seis, Ve por siete, Candado, Llave y Ve por todas. A continuación se da una breve explicación de cada una de ellas.

- Primero: El apostador elige el caballo que cree que llegue en primer lugar.
- Segundo: El apostador elige a un caballo que llegue en primero o segundo lugar.
- Tercero: El apostador elige a un caballo que llegue en primero, segundo o tercer lugar.
- Exacta: El apostador elige a los caballos que lleguen en primero y segundo lugar en el orden exacto.
- Quinela: El apostador elige a los caballos que lleguen en primero y segundo lugar sin importar el orden.
- Trifecta: El apostador elige a los caballos que lleguen en primero, segundo y tercer lugar en el orden exacto.
- Triple: El apostador elige el primer lugar de tres carreras consecutivas.
- Superfecta: El apostador elige el primero, segundo, tercero y cuarto lugar en orden exacto
- Ve por cuatro: El apostador elige el primer lugar de cuatro carreras consecutivas.
- Ve por seis: El apostador elige el primer lugar de seis carreras consecutivas.
- Ve por siete: El apostador elige el primer lugar de siete carreras consecutivas.
- Ve por todas: El apostador elige el primer lugar de todas las carreras programadas para ese día en específico.
- Candado: Esta apuesta permite seleccionar todas las combinaciones posibles con los caballos que haya elegido para una apuesta. Cada combinación se cobrará como una apuesta. Por ejemplo, si cree que los caballos 1, 3 y 5 van a terminar en los primeros tres lugares pero no está

seguro del orden, puede pedir una Trifecta candado 1, 3, 5 (6 apuestas) y no tendrá que preocuparse del orden en que lleguen mientras estos caballos queden entre los primeros tres lugares.

- Llave: Esta apuesta permite seleccionar a un caballo ganador con las demás combinaciones posibles según los caballos y el tipo de apuesta que haya elegido. En este caso son menos apuestas que en el candado. Tomando el ejemplo anterior, si está seguro que el caballo 1 ganará pero tiene dudas de los otros dos, deberá pedir una Trifecta llave 1 con 3, 5 (2 apuestas).

Dependiendo del tipo de apuesta es la cantidad pagada debido a su dificultad. Por ejemplo, la apuesta de Tercero paga mucho menos que la de Va por todas, que es la apuesta más difícil de ganar. En esta última puede ganar una cantidad acumulada, que está alrededor de los 100,000 MXP, más el premio dado según los momios. Además, el costo mínimo de las apuestas varía según el tipo de apuesta que se elija. A continuación se muestra dichos costos:

Primero, Segundo, Tercero y Ve por cuatro/ seis/ siete.	$10.00
Exacta, Quinela, Trifecta y Triple	$5.00
Superfecta	$3.00

Para realizar las apuestas se puede adquirir el "Programa de carreras" del día, en el que se muestran todas las carreras, caballos participantes y sus historiales, jinete, peso del jinete, etc. Es muy recomendable adquirirlo para poder tener una guía oficial para realizar las apuestas.

Para darnos una idea más clara de cómo se debe realizar las apuestas y en qué hay que fijarse, consultamos a algunos expertos en apuestas de caballos. A continuación se muestra una de las entrevistas realizadas:

Entrevista con un experto

"Lo primero en que te fijas es la fecha de las últimas carreras para ver si el caballo trae ritmo o ha estado muy descansado y por lo tanto fuera de forma. Luego analizas la distancia de la carrera y si el caballo ha corrido esa distancia regularmente o si por lo general corre más o menos distancia; es decir, saber si el caballo es de sprint o de fondo. Luego ves el coeficiente de velocidad, que es un número que describe el desempeño del caballo en comparación con el récord de esa distancia en esa pista; esto es, si el récord de la carrera de 6 furlongs es de 1 minuto y 30 segundos, y el caballo que corre ahora hace exactamente ese tiempo, entonces su coeficiente de velocidad es 100, si corre en menos tiempo (1:29) entonces su coeficiente de velocidad sube según la proporción en que baja su tiempo. Ese es el indicador más importante; luego hay que comparar si los caballos han competido entre ellos antes y quién ha ganado y en qué distancia; también hay que ver la clasificación del caballo, es decir, si es Allowance (la mejor clasificación, no hay precio de compra definido) y compite en la categoría de los que valen $50,000, entonces normalmente tendría una ventaja; es como si tú juegas con mejores jugadores y luego te bajan de liga, pues tú sigues teniendo un mejor nivel. Otras cosas qué analizar son las medicinas legales que le dan al caballo como la Butozolidina o la FT, hay varias, y ésas tienen distintos efectos. También hay que considerar que cuando un caballo que es evidentemente mejor, normalmente le ponen un kilo o un kilo y medio de más de peso para que la carrera sea más pareja, y eso es súper importante porque el caballo puede cansarse antes. El jockey también es importante: si el jockey ya ha montado antes a ese caballo, entonces lo conoce mejor y puede hacer una mejor carrera. Y hay muchísimos otros elementos como si es un caballo que corre mejor en pista lodosa o rápida, o menos, etc. El chiste es que si quieres hacer un modelo, entonces pones coeficientes y ponderas el resultado para saber cuál tendrá la mayor probabilidad de ganar."

"Básicamente te fijas en esos datos y luego le apuestas al que más "te lata". También, otra manera de jugar es considerando los momios, es decir: saber quién tiene más probabilidades de ganar según la gente en general, pero como siempre, la gente es influenciable y esos momios pueden no ser objetivos.

En fin, hay muchísimas cosas, el chiste es que lo hagas lo más simple y que bases tus premisas en lo más lógico y congruente."

A partir de ésta y otras entrevistas realizadas, consideramos cuáles son los datos más relevantes que debemos tomar en cuenta para realizar nuestro proyecto. Cabe destacar que sólo se consideró los datos más relevantes, ya que existen demasiadas variables que tendrían que ser tomadas en cuenta.

Para la realización del proyecto tomamos las siguientes variables:

- Historial del caballo para las cuatro últimas carreras
- Coeficientes de velocidad
- Pesos de los jinetes anteriores
- Peso del jinete actual
- Favoritos de los expertos

Los datos están en un archivo de Excel, de donde se toman y manipulan con MATLAB®.

Desarrollo

Para obtener la respuesta final es necesario introducir una gran cantidad de información. Para trasladar los datos desde Excel a MATLAB® es posible agregar una barra de herramientas a Excel que simplifica el intercambio de información (startmatlab), como se muestra en la figura 2.155.

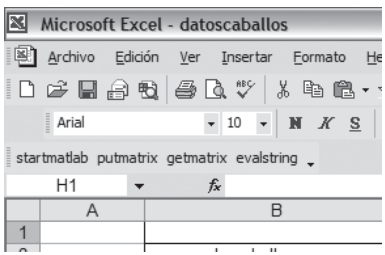


Fig. 2.155 Barra de herramientas para comunicación entre Excel y MATLAB®.

Se puede emplear el botón "putmatrix", el cual envía los datos seleccionados en el documento de Excel a MATLAB® en forma de una matriz. Es factible leer la matriz creada desde el "Command Window" y por tal motivo se corre todo el programa desde ahí. Cuando se inicia el programa se muestra de inmediato el siguiente comentario (figura 2.156):

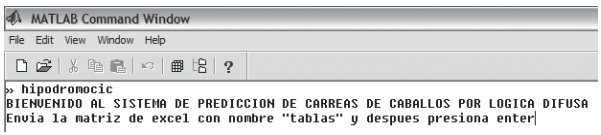


Fig. 2.156 Comentario al inicio del programa.

Se pide que el usuario mande la tabla con la información de todos los caballos. Para cada carrera es necesario proporcionar cierta información para todos los caballos que participan, la cual se obtiene del "Programa de carreras" que se da en el hipódromo. La figura 2.157 muestra la información necesaria para uno de los caballos.

Datos carrera actual				
numero de caballo	1			
peso del ginete	51			
Datos Historicos				
	posición final	coef. de velocidad	peso jinete	número de caballos
ultima carrera	2	71	51	7
penultima carrera	2	65	53	6
antepenultima carrera	4	62	53	8
anteantepenultima carrera	3	66	51	8

Fig. 2.157 Información considerada del caballo para el análisis difuso.

Para cada uno de los caballos que participan se toma la información de: sus cuatro últimas carreras, la posición de llegada en cada una de ellas, coeficiente de velocidad y el peso del jinete. También se incluye en la tabla el número de caballos que participaron en la carrera, dato que se usará después en el programa, y el peso del jinete para la carrera actual. Los datos se envían a MATLAB® con el nombre de "tablas" y de ahí en adelante se identifica en esa matriz la información individual por medio de renglones y columnas.

Lo siguiente que pide el programa una vez que se le envió la tabla, es el número de caballos que participan en la carrera y los tres favoritos. Lo anterior es toda la información que necesita el programa para llevar a cabo su análisis.

En la figura 2.158 se introduce un pequeño segmento del programa para explicar un poco cómo se encuentra ciclado y el funcionamiento general. Se crea en principio una variable w que sirve como apuntador para obtener información de los distintos caballos. En el documento de Excel y por tanto también en la matriz existe un desplazamiento de 10 renglones entre la información de cada caballo. Es por eso que w incremente de 10 en 10 y llega a un máximo que se obtiene multiplicando el número de caballos por 10. De ahí en adelante se puede obtener un dato de la matriz a partir del valor actual de w, y se mantiene el orden de los caballos por medio de i. Casi al final del programa se muestra que i aumenta de uno en uno cada ciclo, permitiendo referenciar correctamente la información a cada caballo. En la figura 2.158 se inicializa i con valor de 1, y w igual, posteriormente se obtiene el número del caballo actual, que ocupa la posición (3,2) dentro de la matriz. Ya con lo anterior se posible saber si el caballo favorito para primer lugar es el primer caballo, y así continúa el programa determinando si cada caballo es favorito o no y para qué lugar.

```
i=1;
for w=1:10:totalcaballos*10;


%favoritos
z=w+1;
numc=tablas(z,2);
numcaballo=cell2mat(numc);


pertfav(i)=0;
        if favorito1==numcaballo;
                pertfav(i)=1;
        end
if favorito2==numcaballo;
      pertfav(i)=0.85;
       end


   if favorito3==numcaballo;
     pertfav(i)=0.70;
   end
```

Fig. 2.158 Fracción del programa donde se comienza con el ciclo principal.

Las entradas se fusificaron en nuestro programa por medio de funciones de membresía tipo sigmoidal, como la de la figura 2.159.

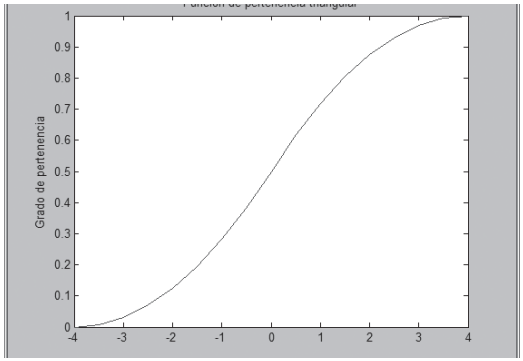


Fig. 2.159 Ejemplo de gráfica tipo sigmoidal.

Básicamente para cada dato de entrada obtuvimos una pertenencia que indica qué tan bueno o malo era. Por ejemplo, en el caso del peso del jinete consideramos que 55 era un peso malo, con valor de pertenencia de 0, y 48 tendría un valor de 1 por ser un muy buen peso. Cada entrada de peso se fusifica para obtener un grado de pertenencia a una función que podría denominarse "excelente". Lo mismo sucede con los coeficientes de velocidad y con las posiciones en que llega el caballo. Para el caso de los coeficientes 40 es nuestro valor malo y 80 el bueno con pertenencia de 1. En cuanto a las posiciones es variable: un primer lugar indica una pertenencia de 1, pero el programa toma de la matriz la cantidad de competidores por cada carrera y con eso determina el rango de la gráfica, asignándole una pertenencia de 0 al último lugar.

Una vez que se obtuvo las entradas se aplican reglas. Para cada caballo se toma la columna completa de cada variable, por ejemplo de la posición de llegada para las últimas cuatro carreras. Se toma entonces el valor mínimo de todas las pertenencias, y lo mismo pasa con las columnas de peso y coeficiente. Teniendo eso, lo siguiente es una etapa de difusificación por medio de una gráfica de barras. Nuestra idea fue identificar las posiciones esperadas de los caballos por medio de una score. La fórmula de nuestra score se basa en centros de masa y se puede ver las salidas en la figura 2.160.

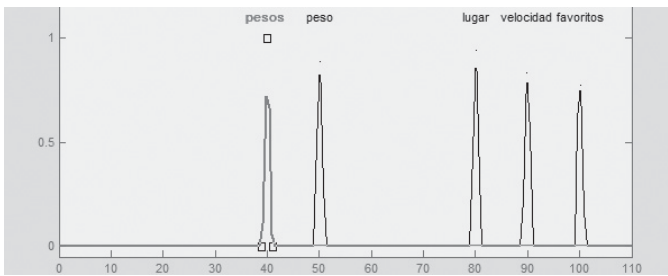


Fig. 2.160 Gráfica de barras de las salidas para un caballo.

La gráfica se obtuvo por medio del ayudante Fuzzy de MATLAB®, pero en realidad sólo sirvió de referencia visual; la fórmula con todas sus entradas se encuentra en el programa. La idea parte de que decidimos darle distinta prioridad a las diferentes características de entrada. Es decir, consideramos que la opinión de cuáles son favoritos tiene una gran importancia, por lo que su función de membresía tiene su centro en 100. El coeficiente de velocidad tiene su centro en 90, la posición en 80, el peso actual del jinete tiene su centro en 50 y el peso de las carreras anteriores 40. Los valores anteriores se multiplican por las pertenencias que se habían obtenido previamente en el programa. El resultado es que se va a tener una score o puntuación relativa, que mientras más grande es mejor, mejora las probabilidades del caballo de llegar en primer lugar. Las características importantes como la posición y velocidad pueden incrementar mucho la score final, dependiendo de su pertenencia que indica qué tan buenas han sido o son las características.

Calculadora difusa por método Mamdani

En esta sección se presenta el desarrollo de una calculadora difusa a través de un sistema Mamdani. Esta calculadora que se pretende generar podrá obtener la suma, multiplicación o resta de dos números que se pedirán por el teclado al usuario de la misma, y estas entradas sólo podrán ser valores entre 0 y 9 contando todos su decimales, de lo contrario no generará un valor de salida.

Como hemos explicado, la primera etapa es generar la base de datos en la cual se basa nuestro control difuso y para ello se ha propuesto una función triangular para cada numero, tal como se ve en la figura 2.161.

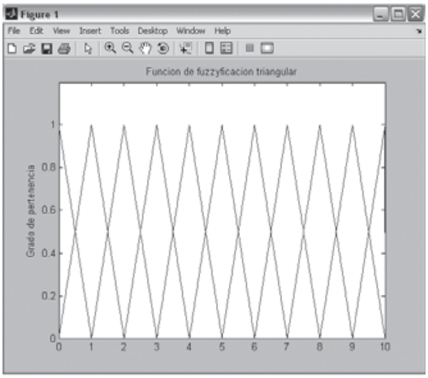


Fig. 2.161 Funciones de membresía para los números de entrada entre cero y diez.

Como se puede ver en la figura 2.161 se generó códigos para cada uno de los números dando así un total de 11 triángulos, ya que se necesitó tomar en cuenta el número 0 y el número 10 para poder obtener los números decimales de los extremos, es decir, los decimales para el número 9 y los decimales antes del número 1.

Cada triángulo es un triángulo isósceles en donde la cúspide coincide obviamente con el grado de pertenencia de uno y está justamente arriba de cada valor de número entero; este triángulo termina exactamente en el siguiente número entero y este tipo de parámetros se siguió para todos los triángulos excepto los dos de la esquina, que en cuanto a área serían exactamente la mitad por sus dimensiones.

Para seguir el método de Mamdani debemos tener una entrada de valor nítido la cual se fuzzifíca a través de las funciones de membresía que hemos generado; para eso se generó en MATLAB® una llamada a pedir un número al usuario que debe de asignar por medio del teclado, el cual debe ser entre 0 y 9.99999 abarcando los decimales que se deseen.

Una vez obtenida dicha entrada, se mide el grado de pertenencia de este valor con respecto a cada uno de los triángulos de membresía y los que tengan grado registrarán valor distinto de creo, de lo contrario será cero, lo cual quiere decir que no pertenece al número. Cada vez que se obtiene una entrada se activarán forzosamente dos funciones de pertenencia, las cuales servirán para sacar el centro de masa que nos dirá exactamente cuál es el número nítido de salida.

```
% fuzzificacion entrada arbitraria
E=input (´primer numero=´)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%GRADOS DE PERTENENCIA EN TRIANGULOS
    If    (E<a0) | (E>c0)
          Pertentri0=0;
    end
    If    (E>a0) &  (E<b0)
          Pertentri0=(E-a0)/ (b0-a0);
    end
```

```
If    (E>b0) &  (E<c0)
      Pertentri0=1-(E-b0)/ (c0-b0);
```

Fig. 2.162 Parte de código en MATLAB® para el primer número de entrada.

La figura 2.162 muestra cómo se pide al usuario el primer número de entrada y empezamos a medir el grado de pertenencia en cada triángulo para que quede más claro.

Este mismo procedimiento se hizo de igual modo para el segundo número de entrada, al cual se pidió de la misma manera, se le asignó los grados de pertenencia a todas las funciones de membresía para, finalmente, por medio del cálculo de su centro de masa obtener su valor nítido de salida.

$$\text{Salida Real} = \frac{\sum_{x=a}^{b} \mu(x) \cdot x}{\sum_{x=a}^{b} \mu(x)} \qquad \text{Ec(3)}$$

La ecuación 3 es la que se usó para poder obtener el valor nítido de nuestra salida, es decir, se hizo la suma de todas las funciones de membresía, cada una con su grado de pertenencia, y ese valor se dividió por la suma de todos los grados de pertenencia que se obtuvieron.

Una vez recuperados los valores nítidos de nuestras entradas, se guardaron en una variable dentro de nuestro programa y se pidió al usuario que especificara el tipo de operación que quería realizar a través de un número específico en el teclado: si desea sumar los dos números deberá presionar la tecla 1, 2 sería para la operación de resta, de lo contrario sería multiplicación.

Al definir la operación que se haría, dentro del programa se especificaba una salida dependiendo de la operación para mostrar el resultado en pantalla.

El proceso se muestra a continuación; en la parte final del proceso se observa el resultado final. El código completo de la calculadora difusa se presenta en la sección G de los Programas básicos en MATLAB®, al final del capítulo.

```
Máximo
primer numero = 2
E =
     2
Segundo  numero = 4
E =
     4
Tipode operación? (1.) suma (2.) resta (3.) multiplicación = 3
OP =
     3
Salidafin
     8
```

Caracterización de un controlador tipo PID mediante un controlador tipo Sugeno

Se realizó la sustitución de una ley de control tipo PID, sintonizada para un sistema de segundo orden cuya función de transferencia es:

$$\frac{1}{s^2 + 10s + 20}.$$

Para llevar esto a cabo, se introdujo una entrada tipo rampa con pendiente de 0.5 al sistema con el controlador PID. Se obtuvo los datos en cada instante del error y la corrección generada por el controlador, los cuales se muestran en la figura 2.163.

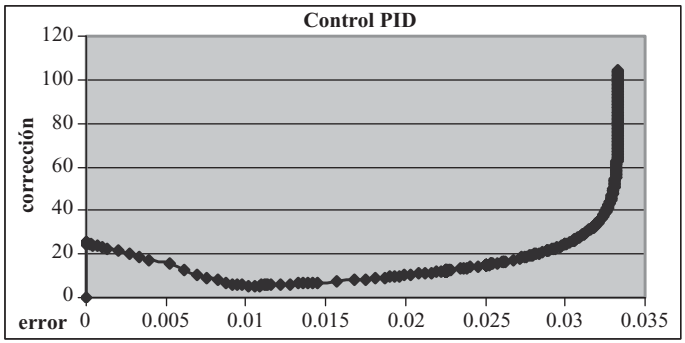
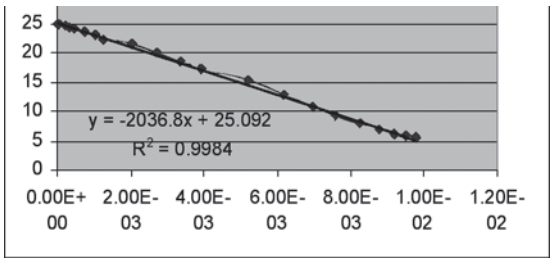
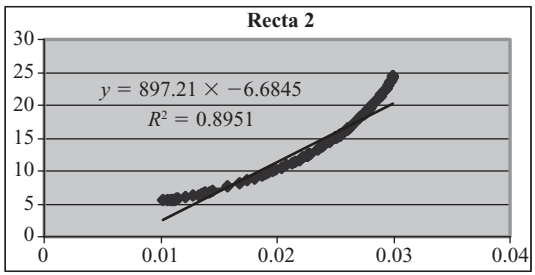


Fig. 2.163 Error y corrección generada por el controlador PID.

Se establecieron tres segmentos (figura 2.164) para hacer una aproximación lineal para las funciones correspondientes a las salidas de las reglas difusas.





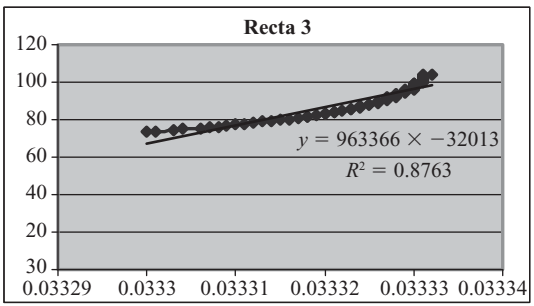


Fig. 2.164 Rectas propuestas para el modelo difuso.

Adicionalmente, se empleó funciones de pertenencia del tipo trapezoidal para las entradas (figura 2.165). Para este modelo difuso el resultado de la aproximación de la señal de corrección del controlador PID puede apreciarse en la figura 2.166.

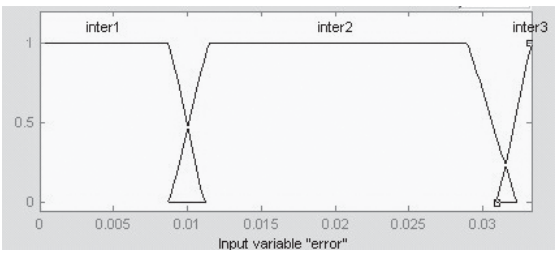


Fig. 2.165 Funciones de pertenencia para las entradas del modelo difuso.

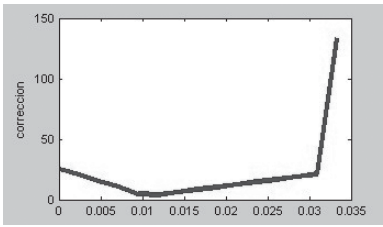


Fig. 2.166 Aproximación de la señal de corrección del controlador PID.

Además, se agregó otra recta al modelo anterior para la sección del codo (figura 2.167). Las funciones de pertenencia entonces resultaron como se aprecia en la figura 2.168. El resultado de este modelo se muestra en la figura 2.169.

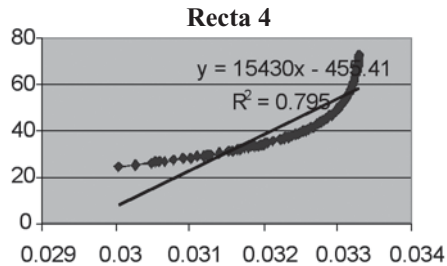


Fig. 2.167 Recta adicional propuesta para el modelo difuso.

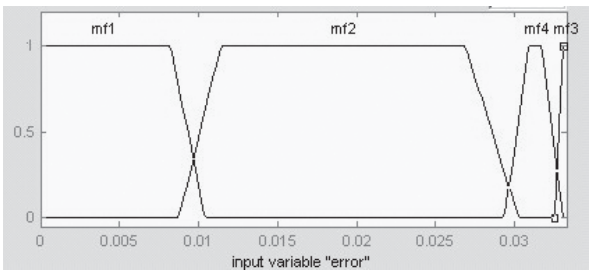


Fig. 2.168 Funciones de pertenencia para las entradas del modelo difuso.

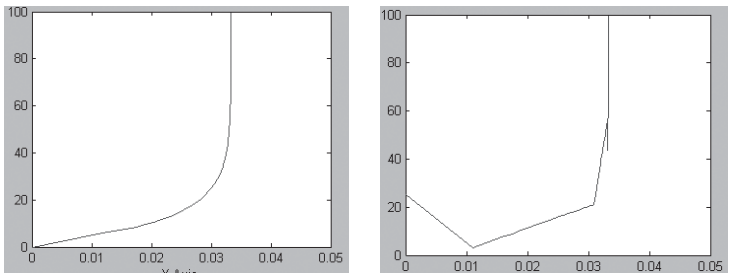


Fig. 2.169 *a*) Señal de corrección del controlador PID original y
b) aproximación de la señal de corrección del controlador PID.

Controlador difuso basado en control directo del par (DTC)

Un controlador directo del par (DTC) para un motor de inducción a partir de un inversor de voltaje (VSI) tiene el esquema general que muestra la figura 2.170.

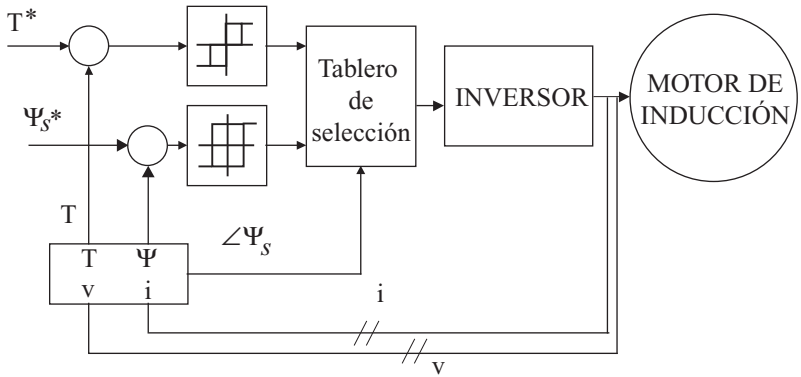


Fig. 2.170 Esquema de un DTC para motor de inducción a partir de un VSI.

En este sistema, el flujo del estator está definido por las siguientes ecuaciones:

$$\overline{\psi_s} = \int \left(\overline{V_s} - \overline{I_s} R_s \right) dt.$$

$$\angle \psi_s = \tan^{-1} \left(\frac{\psi_{sq}}{\psi_{sd}} \right)$$

Por otra parte, el par resulta:

$$T = \frac{3}{2} P \left[\overline{\psi_s} \right] \left[\overline{I_s} \right] \text{sen } \delta.$$

El voltaje tiene una influencia en el par y en el flujo del estator, lo cual en vectores espaciales puede apreciarse en la tabla 2.33, considerando la representación vectorial de la figura 2.171.[3]

Tabla 2.33 Influencia de los vectores espaciales de voltaje en el par y en el vector espacial de flujo del estator

	V_k	V_k+1	V_k+2	V_k+3	V_k−2	V_k−1	V_0 V_7
Stator Flux	↑ ↑	↑	↓	↓ ↓	↓	↑	
Torque	↑	↑	↑	↓	↓ ↓	↓ ↓	↓

[3] Para entender mejor el manejo de los vectores espaciales en las máquinas eléctricas, véase Ponce, Pedro, *Máquinas eléctricas*, Alfaomega, 2008.

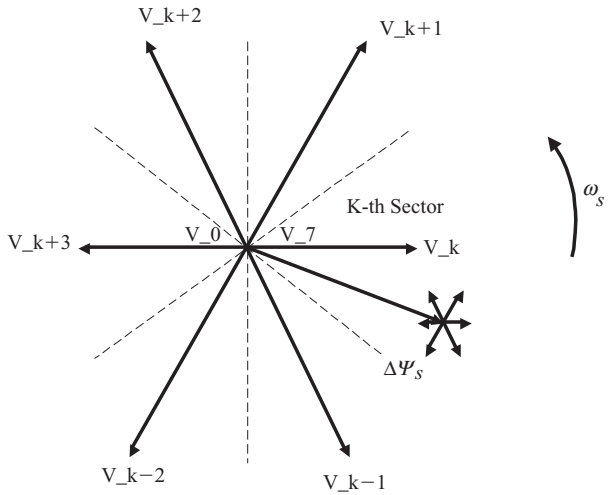


Fig. 2.171 Vectores espaciales de voltaje.

Existen diversos problemas relacionados con el flujo del estator, como la presencia de armónicos que muestra la figura 2.172.

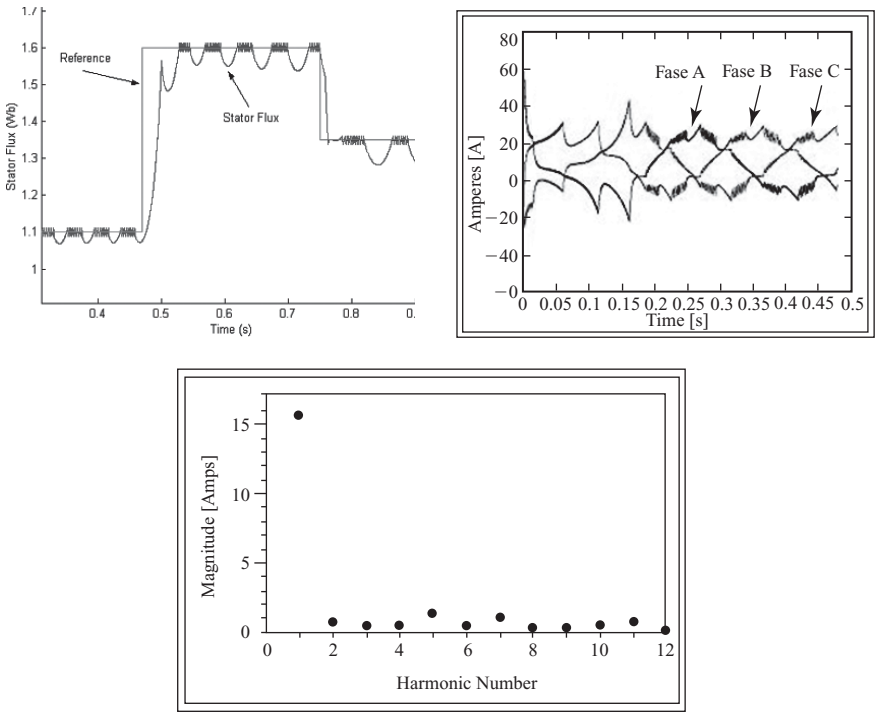


Fig. 2.172 Presencia de armónicos en el flujo del estator y en la corriente de fase.

Para reducir los armónicos se propone el esquema de la figura 2.173 incorporando un controlador difuso de sectores variables de activación de interruptores. El controlador difuso se basa en el ángulo a partir de la medición del error en el flujo del estator y en el par.

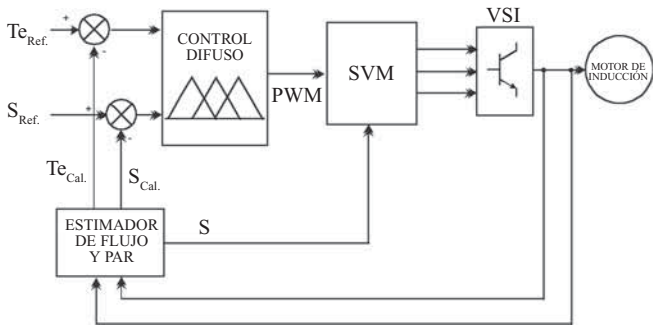


Fig. 2.173 Esquema propuesto para el DTC.

Por último, se simuló los resultados para el par y para el flujo del estator, los cuales se presentan en las figuras 2.174 y 2.175, respectivamente.

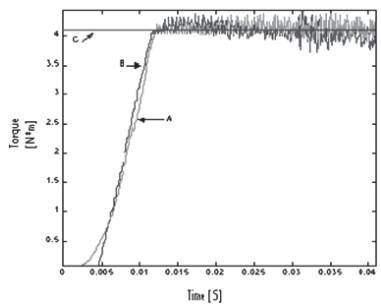


Fig. 2.174 Par para un ángulo de 0º (A), 30º (B) y la referencia (C).

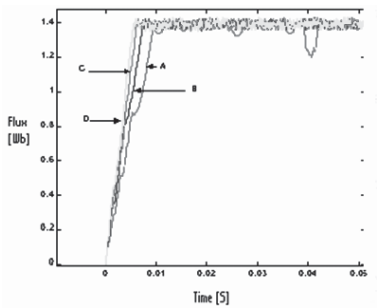


Fig. 2.175 Respuesta del flujo del estator para un ángulo de 0º (A), 15º (B), 25º (C) y 30º (D).

Adicionalmente se graficó la respuesta del flujo del estator en par constante y las regiones de debilitamiento del campo para un ángulo de 300º y HB=0.002 (figura 2.176).

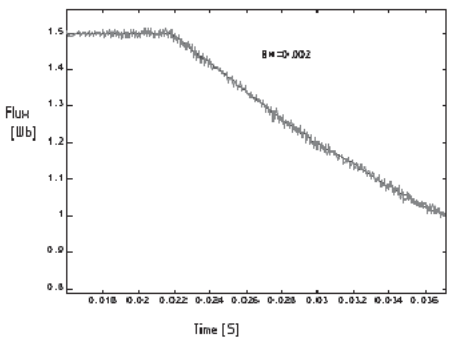


Fig. 2.176 Respuesta del flujo del estator en par constante y regiones de debilitamiento del campo para un ángulo de 300º y HB=0.002.

La figura 2.177 muestra el flujo del estator y las corrientes de fase con reducción de armónicos, resultado de la incorporación del controlador difuso.

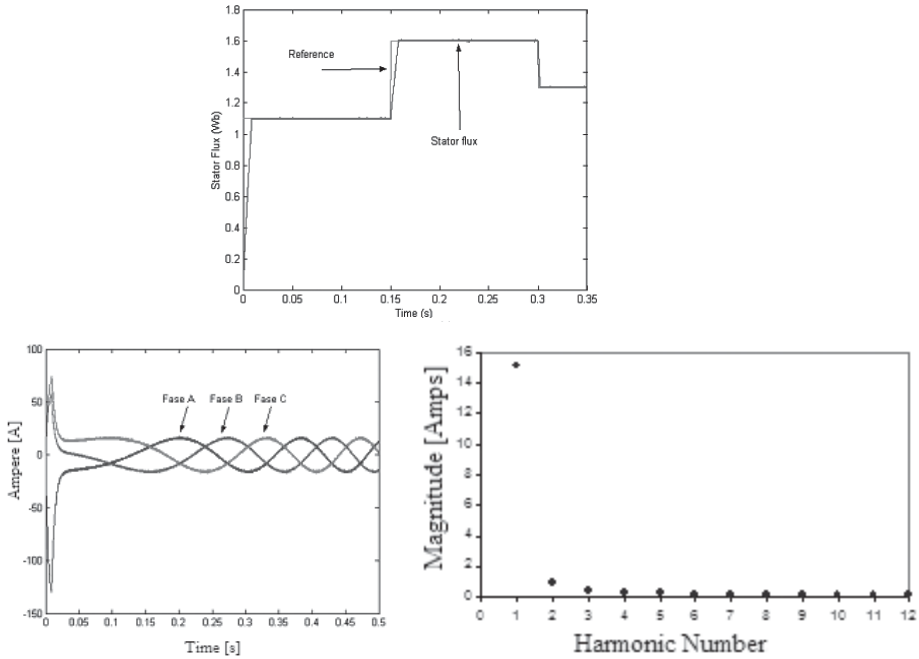


Fig. 2.177 Eliminación de armónicos en el flujo del estator y en la corriente de fase.

Control de velocidad sin sensores usando control directo del par (DTC) basado en la modulación del ancho de pulso mediante vectores espaciales (SVPWM)

Otra propuesta para llevar a cabo un DTC emplea la modulación del ancho de pulso (PWM) a través de vectores espaciales (SVPWM), y se propuso incorporar un controlador difuso de la velocidad de un motor de inducción, como se muestra en la figura 2.178.

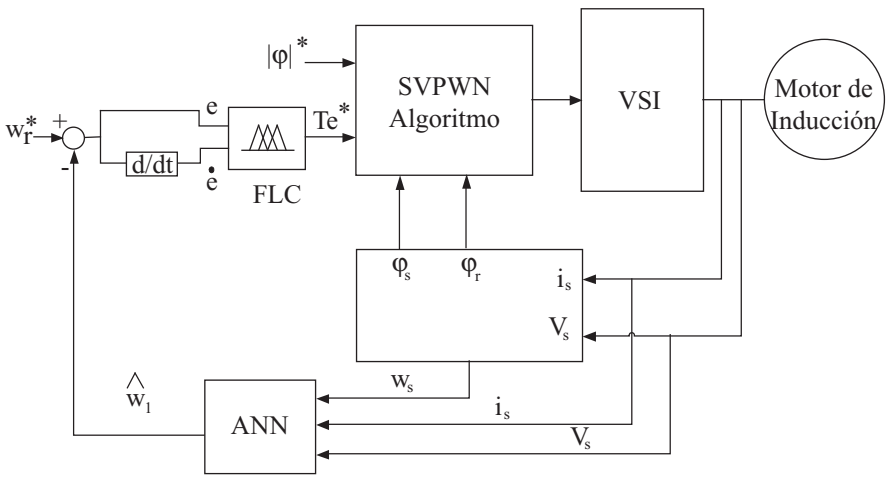


Fig. 2.178 Topología propuesta para el control SVPWM.

Se incorporó un controlador difuso tipo Mamdani, cuya topología se presenta en la figura 2.179.

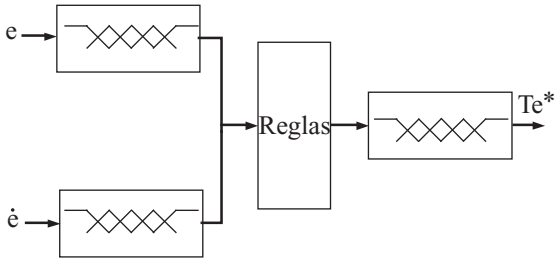


Fig. 2.179 Controlador difuso tipo Mamdani propuesto.

Los resultados obtenidos a partir del uso de un controlador DTC convencional contra la topología propuesta para SVPWM se muestran en la figura 2.180, donde se aprecia la reducción del rizado en el par y en el flujo para una frecuencia de 5 Hz y una frecuencia de encendido de los interruptores del inversor de 20 kHz. Además, se graficó la corriente del estator para observar la presencia de armónicos y la reducción de los mismos con el SVPWM (figura 2.181).

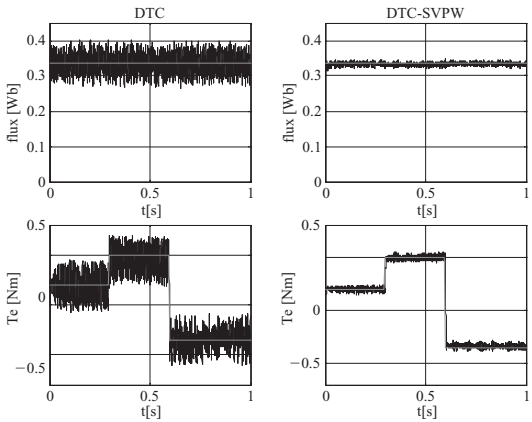


Fig. 2.180 Comparación del rizado en el flujo y en el par de un motor de inducción, con un controlador DTC convencional (izquierda) y el SVPWM propuesto (derecha).

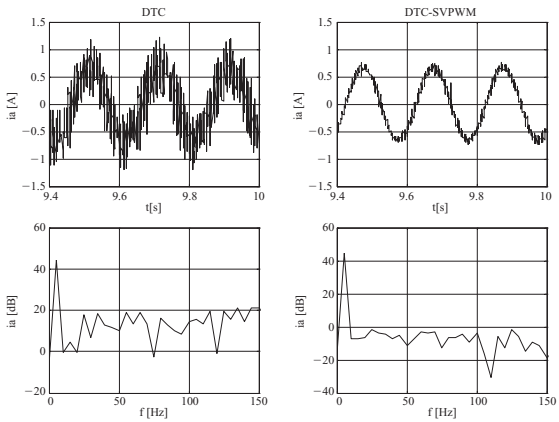


Fig. 2.181 Comparación de los armónicos en la corriente del estator, con un controlador DTC convencional (izquierda) y el SVPWM propuesto (derecha).

La figura 2.182 muestra el comportamiento de un controlador DTC convencional con respecto al de un SVPWM ante cambios en la dinámica de la entrada.

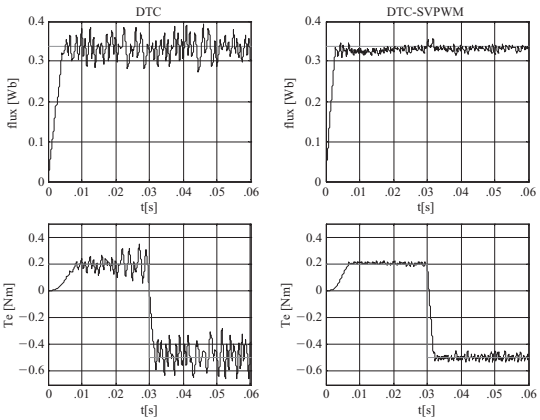
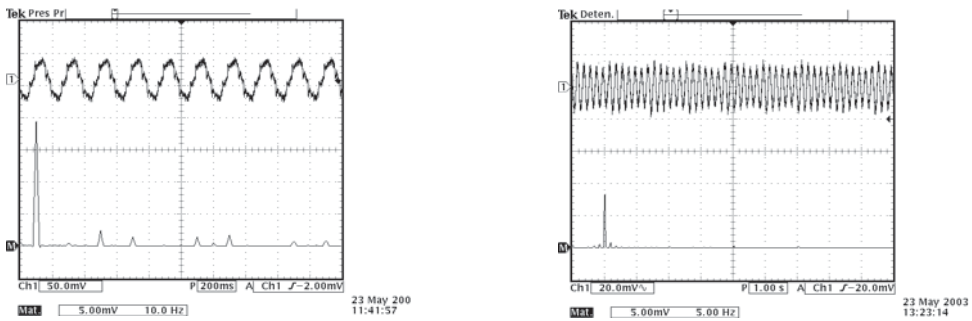


Fig. 2.182 Comparación de la respuesta dinámica en el flujo y en el par, con un controlador DTC convencional (izquierda) y el SVPWM propuesto (derecha).

Finalmente, se implementó el controlador tipo SVPWM con un motor de inducción real y el uso de componentes electrónicos, y la figura 2.183 muestra la reducción de armónicos en la corriente comparada con un control a partir de un inversor de seis pasos, medida en el osciloscopio.



a) *b*)

Fig. 2.183 Comparación de la corriente y los contenidos armónicos de: *a*) un controlador por inversor de seis pasos y *b*) el SVPWM, a 5 Hz de frecuencia.

Agrupamientos difusos con pesos

El agrupamiento denominado Fuzzy c-Means considera que cada cluster tenga aproximadamente el mismo número de elementos, y sin embargo existen algoritmos en los cuales es posible que tengan distintas cantidades a partir de ponderaciones y una media generalizada, definida como:

$$M_m(Y) = \left(\sum_{i=1}^{c} \alpha_i y_i^{\frac{1}{1-m}} \right)^{1-m}$$

Donde:

$$\sum_{i=1}^{c} \alpha_i = 1.$$

Se sabe que

$$\lim_{m\to 1}\left(\sum_{i=1}^{c}\alpha_i\left\|x_k-v_i\right\|^{\frac{2}{1-m}}\right)^{1-m}=\min_j\left\{\left\|x_k-v_i\right\|^2\right\}.$$

Por lo tanto:

$$J(u,v)=\left(\sum_{k=1}^{n}\min_{1\le i\le c}\left\{\left\|x_k-v_i\right\|^2\right\}\right)$$

$$J(u,v)=\sum_{k=1}^{n}\lim_{m\to 1}\left(\sum_{i=1}^{c}\alpha_i\left\|x_k-v_i\right\|^{\frac{2}{1-m}}\right)^{1-m}.$$

Redefiniendo:

$$J^{(m)}(X,v)=\sum_{k=1}^{n}\left(\sum_{i=1}^{c}\alpha_i\left\|x_k-v_i\right\|^{\frac{2}{1-m}}\right)^{1-m}$$

La nueva función objetivo para el algoritmo Fuzzy c-Means con peso (WFCM) resulta:

$$J_m^{\alpha}(u,v)=\sum_{k=1}^{n}\sum_{i=1}^{c}\alpha_i^{1-m}u_{ik}^{\ m}\left\|x_k-v_i\right\|^2$$

Garantizando para minimizar que:

$$v_i=\frac{\sum_{k=1}^{n}u_{ik}^{n}x_k}{\sum_{k=1}^{n}u_{ik}^{n}}$$

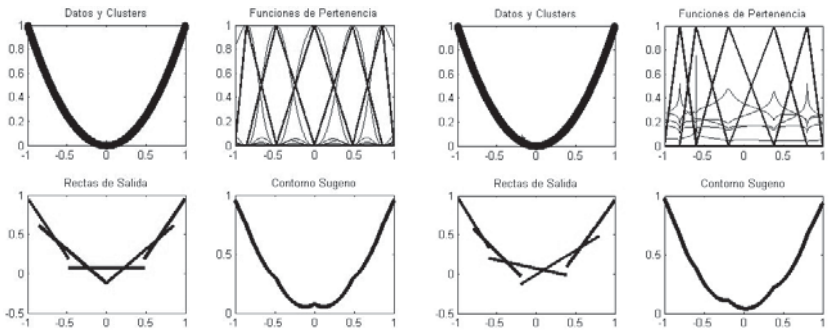
$$u_{ik}=\frac{\alpha_i\left\|x_k-v_i\right\|^{\frac{2}{m-1}}}{\sum_{j=1}^{c}\alpha_j\left\|x_k-v_j\right\|^{-\frac{2}{m-1}}}$$

$$\alpha_i=\frac{\left(\sum_{k=1}^{n}u_{ik}^{m}\left\|x_k-v_i\right\|^2\right)^{\frac{1}{m}}}{\sum_{i=1}^{c}\left(\sum_{k=1}^{n}u_{ik}^{m}\left\|x_k-v_i\right\|^2\right)^{\frac{1}{m}}}.$$

El algoritmo del WFCM es el siguiente:

1. Establecer los parámetros m, $V^{(0)}$, c, α_0 y el límite ζ. El contador de iteraciones l se inicializa en 0.
2. Se encuentra la matriz de pertenencia $U^{(0)}$ con los vectores de centro $V^{(0)}$ y los pesos $\alpha^{(0)}$.
3. Encontrar $\alpha^{(l+1)}$ con $V^{(0)}$ y $U^{(0)}$
4. Encontrar $V^{(l+1)}$ con $U^{(0)}$.
5. Si máx $\left\{\left\|v_i^{l+1}-v_i^{(l)}\right\|,\left\|\alpha_i^{(l+1)}-\alpha_i^{(l)}\right\|\right\}<\xi$ detener el proceso, si no, $l=l+1$ y se repiten los pasos desde el 2.

Para ejemplificar la incorporación de este algoritmo al de FCM se realizó el modelo matemático de una parábola con el método Sugeno, agrupando los datos de entrada para sintonizar las funciones de pertenencia, obteniendo los resultados comparativos que se aprecian en la figura 2.184.



a) *b*)

Fig. 2.184 Uso del algoritmo FCM *a*) sin peso y *b*) con peso.

Segmentación de imágenes médicas a través de agrupamientos difusos

Para el procesamiento de imágenes en medicina, es posible emplear la minimización difusa, la cual implica una mejora en el contraste de colores mediante el uso de intensificadores difusos, por lo que cada píxel en una imagen se difusifica y se reduce el grado de difusión de la misma. Para realizar la reducción difusa se emplea un parámetro difusificador F_d definido como:

$$\mu(g) = \left[1 + \frac{g_{\text{máx}} - g}{F_d}\right]^{-F_e}.$$

Adicionalmente se emplea el intensificador:

$$\mu'(g) = \begin{cases} 2\mu(g)^2 & si\ 0 \le \mu(g) \le 0.5 \\ 1 - 2(1-\mu(g))^2 & si\ 0.5 \le \mu(g) \le 1 \end{cases}.$$

Para desdifusificar de acuerdo con:

$$g' = g_{\text{máx}} - F_d\left(\mu'(g)^{-\frac{1}{F_e}} - 1\right), \quad \left(1 + \frac{g_{\max}}{F_d}\right)^{-F_e} \le \mu'(g) \le 1.$$

El operador difusificador se calcula a partir de:

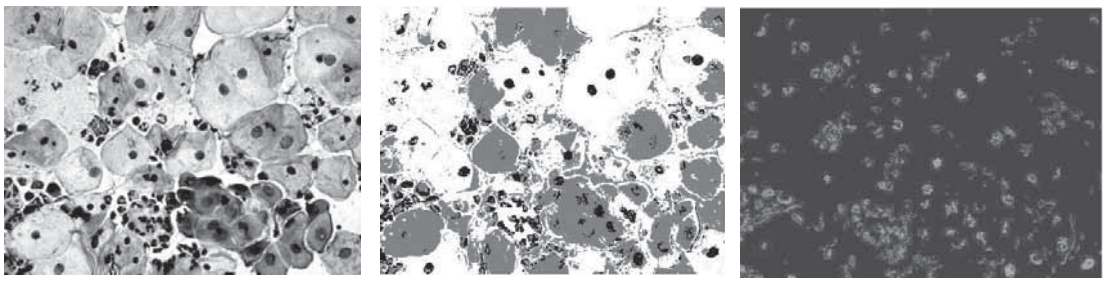
$$F_d^{\,2} = \frac{1}{2}\frac{\sum_{k=0}^{g_{\text{máx}}} (g_{\text{máx}} - g)^4 p(g)}{\sum_{k=0}^{g_{\text{máx}}} (g_{\text{máx}} - g)^2 p(g)}.$$

La figura 2.185 muestra el resultado del uso de este proceso de reducción difusa para una imagen convencional.

a) *b*)

Fig. 2.185 *a*) Imagen original. *b*) Imagen contrastada.

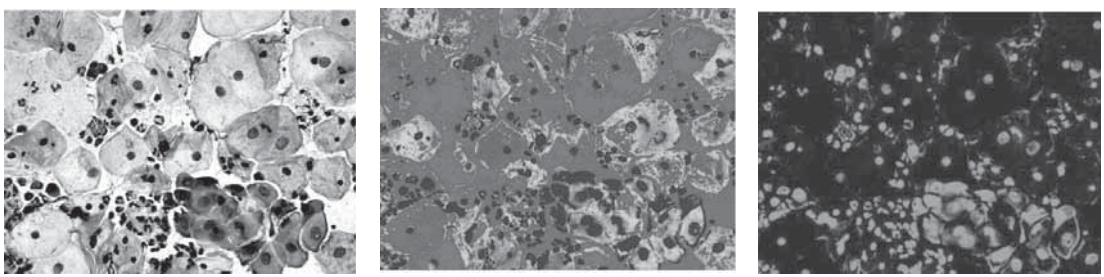
Para la identificación de células es posible utilizar segmentaciones por color (nítidas) o segmentaciones difusas empleando el método FCM. El resultado de la segmentación por color empleando el método K Means puede verse en la figura 2.186, obteniendo tres agrupamientos a partir de la identificación de los núcleos, citoplasma y fondo.



a) *b*) *c*)

Fig. 2.186 *a*) Imagen original. *b*) Separación de núcleos, citoplasmas y fondos. *c*) Núcleos en azul.

Para realizar la segmentación difusa de células se busca que la imagen sea una matriz tridimensional de $m \times n \times 3$ elementos, empleando el algoritmo de FCM, para generar tres conjuntos con tres vectores de sus respectivos centros, ya que se desea discernir entre el núcleo, el citoplasma y el fondo de contraste de la imagen. Los resultados de este agrupamiento pueden observarse en la figura 2.187.



a) *b*) *c*)

Fig. 2.187 *a*) Imagen original. *b*) Separación de núcleos, citoplasmas y fondos. *c*) Núcleos en azul.

La clasificación difusa es más suave respecto a la nítida, debido a que rescata detalles y células que en la imagen no están completas, ofreciendo mejores resultados en general.

Aproximación de un modelado de sentimientos humanos basado en el reconocimiento de expresiones faciales con lógica difusa

En esta época interactuamos diariamente con sistemas computacionales, automatizados, electrónicos, etc. Sin embargo, la mayoría de los usuarios no se sienten cómodos al momento de comunicarse con una computadora o similar. El descontento entre el humano y la computadora se debe básicamente a la falta de empatía entre ambos. Esta falta de empatía por parte de la computadora se debe a la inhabilidad de reconocer los sentimientos o emociones del usuario y comprenderlos para poder transmitir otro tipo de sentimientos.

La comunicación no verbal, por medio del uso del lenguaje corporal o expresiones faciales, es una parte esencial para la comunicación humana. Los humanos usamos de manera efectiva este tipo de comunicación como un medio de expresar información. Es decir, se crean y usan expresiones faciales para transmitir emociones durante una conversación. Una gran cantidad de información no-verbal viene de expresiones faciales.

La creación de un sistema modular para el reconocimiento y transmisión de emociones mejoraría en gran medida la interacción hombre-computadora, la cual sería más placentera para el usuario. Para realizar la construcción de una interfase afectiva, la computadora primero debe ser capaz de reconocer ciertas emociones a partir de estudios hechos por Ekman en los años setenta, los cuales comprueban la existencia de algunas expresiones universales, especialmente la existencia de seis expresiones básicas. Estas expresiones son generales y están más allá de las razas, culturas, sexos, etc.; son las de felicidad, tristeza, sorpresa, disgusto, temor y enojo.

Las expresiones faciales pueden brindar información no-verbal acerca del estado de ánimo mental y físico de la persona que esté hablando. Normalmente el Sistema de Codificación de Acción Facial (FACS, por sus siglas en inglés) consta de 44 “sets” de unidades de acción (AU’s) visualmente reconocibles. Pero en general podemos clasificar 17 puntos clave para la expresión facial. Encontramos cuatro puntos para las cejas conformados por los extremos izquierdos y derechos de cada una de éstas. Para los ojos tenemos ocho puntos clave conformados por los extremos izquierdo y derecho, así como los extremos superior e inferior de cada ojo, es decir cuatro puntos por ojo. Para la boca contamos con otros cuatro puntos ubicados en cada uno de los extremos de ésta, igual a la distribución de los ojos. Con todos estos puntos sumamos un total de 16 puntos. El último punto está ubicado entre los ojos y el punto central de la cara, que se toma como punto de simetría. A partir de los puntos clave creamos un total de 21 valores usando distancia euclideana, mediciones de ángulo y promedios de valores en ambos lados de los ojos y las cejas. Con este procedimiento obtenemos combinaciones de distintos valores los cuales se utilizan para crear o simular expresiones faciales. De esta manera podemos normalizar las expresiones con base en ciertos movimientos o parámetros obtenidos a partir de los AU´s. Esto quiere decir que con ciertos parámetros de distintas AU´s obtendremos una expresión determinada, ya sea felicidad, sorpresa, temor, etcétera.

En resumen, la cara se divide en ciertos puntos clave y a partir de éstos obtenemos ciertos valores. Se combinan uno o más valores para crear una unidad de accion (AU), las cuales se usan para determinar la expresión facial.

Otro punto que debe considerarse es la evaluación de la acústica del tono de voz. Esto es muy importante ya que también a partir del tono de voz podemos saber el estado de ánimo de una persona. Un método para realizar esto es obtener el sentido del ritmo de las palabras, es decir, la variación en el sonido de éstas. En el análisis de emociones, las conversaciones verbales pueden dividirse en dos categorías: emociones naturales o emociones artificiales. Las conversaciones naturales incluyen las charlas relajadas, por ejemplo una plática con amigos o familiares. Las conversaciones artificiales pueden ser cuando alguien se encuentra frente a un micrófono o ha memorizado algún dialogo.

De esta manera, ahora es posible la investigación que involucra tanto la medición como las comparaciones de complicadas características psicológicas humanas, así como estados y transiciones de estados de ánimo.

Aproximación a los sentimientos humanos a través de lógica difusa

Para poder leer las emociones a partir de expresiones faciales es necesario seguir un protocolo. Algunos investigadores han planteado una serie de etapas, comenzando por el escaneo del rostro e identificación de puntos clave. Dichos puntos se denominan "feature points". Una vez que se tienen esos puntos se puede tomar relaciones entre ellos para formar ángulos y distancias, los "feature values". Ya con esos valores es posible reconocer acciones como por ejemplo abrir la boca o los ojos, que posteriormente pueden unirse para definir una emoción.

Nosotros decidimos utilizar las siete expresiones que define el artículo, las cuales se muestran en la figura 2.188. Dichas emociones son las salidas dentro del proceso difuso y se estableció funciones de membresía para cada una de ellas.

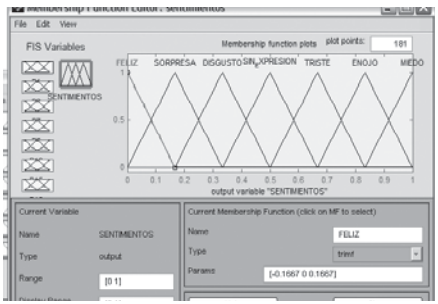


Fig. 2.188 Funciones de pertenencia para la salida, las siete expresiones básicas.

Las entradas son los puntos característicos del rostro. En el artículo se proponen 17 puntos, pero para nuestro análisis únicamente utilizamos siete que consideramos muy útiles. La tabla 2.34 muestra nuestra selección, y aunque mencionamos en la tabla los 17 puntos, sólo algunos tienen valores para las expresiones. Básicamente definimos que cada uno de los puntos podía estar arriba, abajo o en posición neutral. Los valores no llenados en la tabla corresponden a posiciones neutrales. Así, por ejemplo, un ojo abierto lo indicamos con "ojo izq superior arriba" y "ojo izq inferior abajo". Es importante mencionar que consideramos expresiones simétricas, por lo que muchos puntos se definen sólo en una sección del rostro.

Puntos	Definicion	Felicidad	Tristeza	Sorpresa	Disgusto	Temor	Enojo	Sin expresion
P1	Ceja izq exterior			arriba			arriba	
P2	Ceja izq interior		arriba	arriba		arriba		
P3	Ceja der interior							
P4	Ceja der exterior							
P5	ojo izq exterior							
P6	ojo izq interior							
P7	ojo izq superior	arriba	abajo	arriba		arriba		
P8	ojo izq inferior	abajo	arriba	abajo		abajo		
P9	ojo der exterior							
P10	ojo der interior							
P11	ojo der superior							
P12	ojo der inferior							
P13	boca izq	arriba	abajo	arriba	abajo			
P14	boca der							
P15	boca superior			arriba	abajo		abajo	
P16	boca inferior	abajo		abajo	abajo		abajo	
P17	punto central cara							

Tabla 2.34 Puntos necesarios con su correspondiente valor para cada una de las expresiones.

Definimos por lo tanto siete entradas, los siete puntos que utilizamos. Posteriormente, a cada entrada se le asignó tres funciones para que tuviera sus tres posibles valores (figura 2.189).

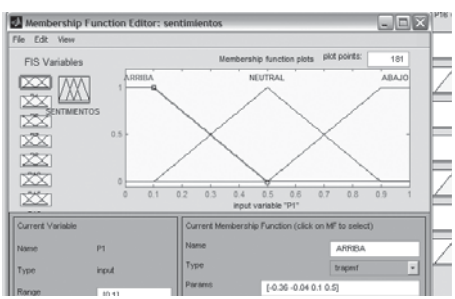


Fig. 2.189 Funciones de entrada utilizadas para los puntos.

Una vez que se tenían las entradas y salidas, lo siguiente fue crear las reglas. Como se puede ver en la figura 2.190 se utilizó siete reglas, cada una de ellas para definir una salida. La forma de crearlas fue siguiendo la tabla 2.34 de tal manera que teniendo las entradas adecuadas se puede tener la salida correspondiente.

Fig. 2.190 Reglas implementadas en el proceso.

Debido a la forma en que funciona la aplicación fuzzy de MATLAB®, no se pueden aplicar entradas literales como "punto 1 arriba". Para poder realizar una entrada es necesario poner valores numéricos, debido a la forma en que se encontraban las funciones de entrada se debe poner un valor de 0.5 para indicar neutral, 1 para indicar abajo y 0 para arriba. La figura 2.191 es un ejemplo de la lectura de felicidad; se introduce un vector con las entradas y la salida es un valor numérico, el cual debe compararse con la gráfica de salidas para ver a qué sentimiento corresponde.

Fig. 2.191 Expresión de felicidad generada a partir de las entradas necesarias.

Se debe tener cuidado de colocar las salidas con cierta coherencia en el orden, de manera que si no se introducen los valores precisos para tener cierto sentimiento, tendríamos una salida que indicaría que se aproxima a una expresión y tal vez un poco más a otra (figura 2.192).

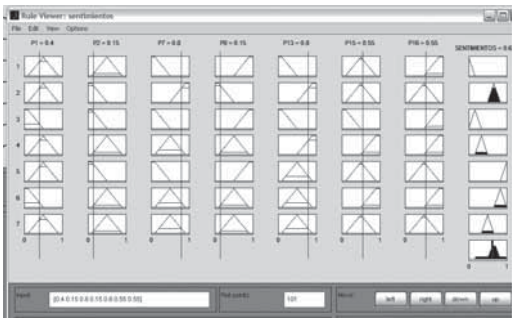
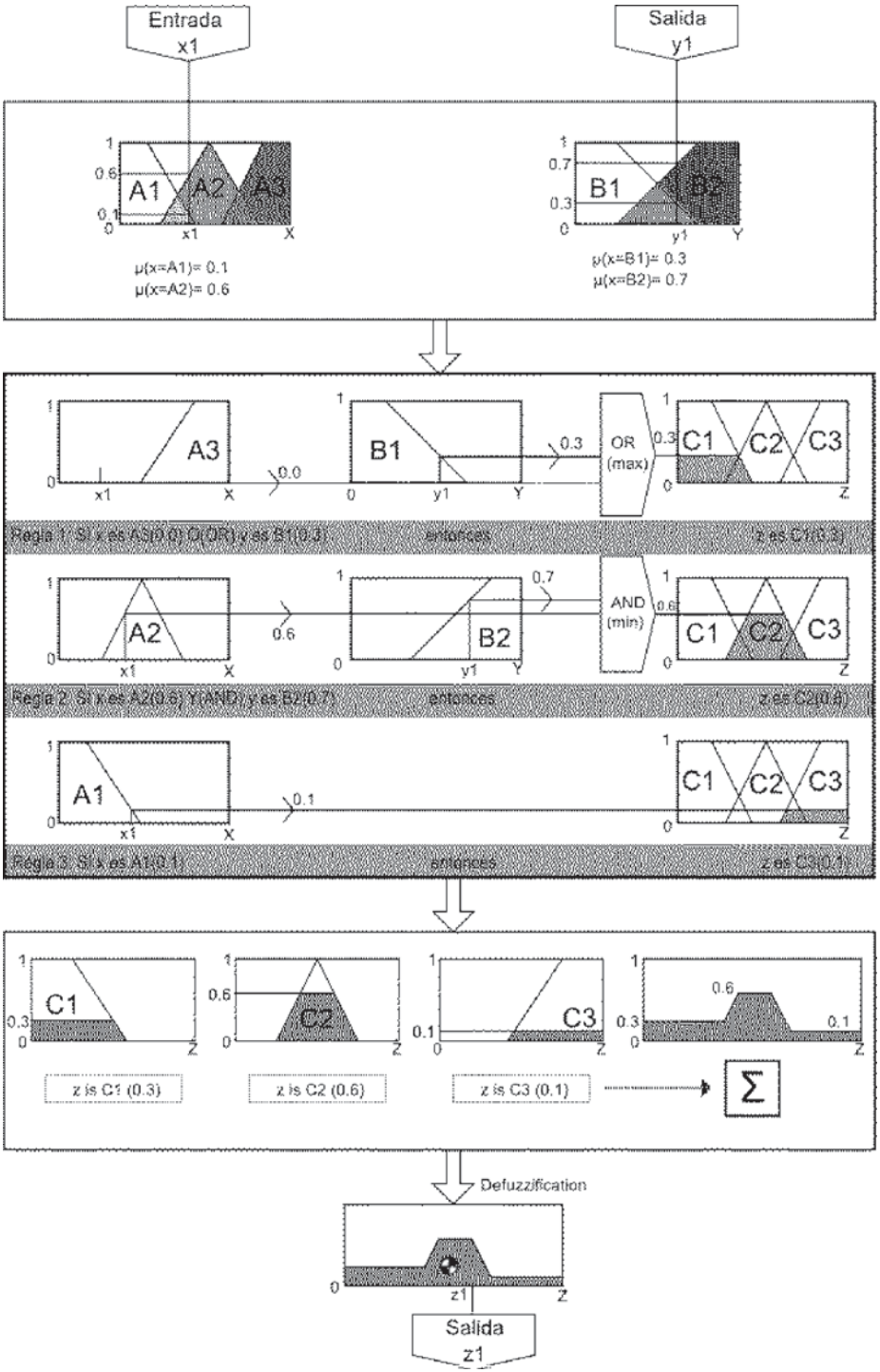


Fig. 2.192 Ejemplo de salida para una expresión no exacta.



PROGRAMAS BÁSICOS EN MATLAB

```
A

%SATURACIÓN
%Parámetros de la función
params=[3 8];
x=0:.1:10;
i=1:101;
%Función rampmf
a = params(1); b = params(2);
if a > b,
        error('Condicion Ilegal: a > b');
end
y = zeros(size(x));
index = find(x >= b);
if ~isempty(index),
        y(index) = ones(size(index));
end
index = find(x < a);
if ~isempty(index),
        y(index) = zeros(size(index));
end
index = find(a <= x & x <= b);
if ~isempty(index) & a ~= b,
        y(index) = (x(index)-a)/(b-a);
end
index = find(x > b);
if ~isempty(index),
        y(index) = ones(size(index));
end
Y(i)=y;
%Gráfica
figure, plot(x,Y),title ('RAMP');



B

%HOMBRO
%Parámetros de la función
```

```
params=[3 8];
x=0:.1:10;
i=1:101;
%Función rampmf
a = params(1); b = params(2);
if a > b,
        error('Condicion Ilegal: a > b');
end
y = zeros(size(x));
index = find(x >= b);
if ~isempty(index),
        y(index) = zeros(size(index));
end
index = find(x < a);
if ~isempty(index),
        y(index) = ones(size(index));
end
index = find(a <= x & x <= b);
if ~isempty(index) & a ~= b,
        y(index) = 1-(a-x(index))/(a-b);
end
Y(i)=y;
%Gráfica
figure, plot(x,Y),title ('RAMPneg');



C


%TRIANGULAR
%Parámetros de la función
params=[2 5 8];
x=5;
%Función trimf
a = params(1); b = params(2); c = params(3);
if a > b,
        error('Illegal parameter condition: a > b');
elseif b > c,
        error('Illegal parameter condition: b > c');
elseif a > c,
        error('Illegal parameter condition: a > c');
```

```
end
y = zeros(size(x));
index = find(x <= a | c <= x);
y(index) = zeros(size(index));
if (a ~= b)
        index = find(a < x & x < b);
        y(index) = (x(index)-a)/(b-a);
end
if (b ~= c)
        index = find(b < x & x < c);
        y(index) = (c-x(index))/(c-b);
end
index = find(x == b);
y(index) = ones(size(index));
%Gráfica
x=0:0.1:10;
figure, plot(x,trimf(x,[2 5 8])),title ('TRI');



D


%TRAPEZOIDAL
%Parámetros de la función
params=[2 4 6 8];
x=5;
%Función trapmf
a = params(1); b = params(2); c = params(3); d = params(4);
if a > b,
        error('Condicion Ilegal: a > b');
elseif c > d,
        error('Condicion Ilegal: c > d');
end
y1 = zeros(size(x));
y2 = zeros(size(x));
index = find(x >= b);
if ~isempty(index),
        y1(index) = ones(size(index));
end
index = find(x < a);
if ~isempty(index),
```

```
        y1(index) = zeros(size(index));
    end
    index = find(a <= x & x < b);
    if ~isempty(index) & a ~= b,
        y1(index) = (x(index)-a)/(b-a);
    end
    index = find(x <= c);
    if ~isempty(index),
        y2(index) = ones(size(index));
    end
    index = find(x > d);
    if ~isempty(index),
        y2(index) = zeros(size(index));
    end
    index = find(c < x & x <= d);
    if ~isempty(index) & c ~= d,
        y2(index) = (d-x(index))/(d-c);
    end
    y = min(y1, y2);
    %Gráfica
    x=0:0.1:10;
    figure, plot(x,trapmf(x,[2 4 6 8])), title ('TRAP');



    E

    %SIGMOIDAL
    %Parámetros de la función
    params=[2 8];
    x=5;
    %Función smf
    x0 = params(1); x1 = params(2);
    if x0 >= x1,
        y = x >= (x0+x1)/2;
        return;
    end
    y = zeros(size(x));
    index1 = find(x <= x0);
    if ~isempty(index1),
        y(index1) = zeros(size(index1));
```

```
end
index2 = find((x0 < x) & (x <= (x0+x1)/2));
if ~isempty(index2),
        y(index2) = 2*((x(index2)-x0)/(x1-x0)).^2;
end
index3 = find(((x0+x1)/2 < x) & (x <= x1));
if ~isempty(index3),
        y(index3) = 1-2*((x1-x(index3))/(x1-x0)).^2;
end
index4 = find(x1 <= x);
if ~isempty(index4),
        y(index4) = ones(size(index4));
end
%Gráfica
x=0:0.1:10;
figure, plot(x,smf(x,[2 8])), title ('S');



F

%%%%%%%% clusters Difusos y sistema Sugeno %%%%%%%%%%%%%%%%%%%%

clear all

disp('ENTRADAS')
disp(' ')
disp('BIENVENIDO AL METODO SUGENO')
disp('RANGO DE DATOS')
xi=input('Dame el primer dato (X inicial): ');
xf=input('Dame el ultimo dato (X final): ');
tmuestra=(xf-xi)/10000;
x=xi:tmuestra:xf;
disp(' ')
disp('FUNCION DE ENTRADA')
f=input('Dame la funcion de entrada (f(x)): ');
disp(' ')
disp('FUNCIONES DE MEMBRESIA Y SALIDA')
nm=input('Dame el numero de clusters que deseas (clusters, membresias y rectas): ');
nm=round(nm);
n=nm;
```

```
%%%%%%%%%%%%%% PROGRAMA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%% Ajustando el tiempo de muestreo

intervalos=floor((length(x)-1)/n);
valormaximo=intervalos*n;
xcolchon=x;
fcolchon=f;
clear x
clear f
for i=1:valormaximo
        x(i)=xcolchon(i);
        f(i)=fcolchon(i);
end

%%%%%%%%%%%%%%%%% Haciendo las funciones de clusters

valoresclu=[x' f'];
[center,U,obj_fcn] = fcm(valoresclu,n);
uu=U';
maxu=max(uu);
for i=1:n
        vector(i)=find(uu(:,i)==maxu(i));
end
vec=sort(vector);

%%%%%%%%%%%%%%%% Crea funciones de pertenencia triangulares

for i=1:nm
        if (i-1)==0
                pt(i,1)=x(1);
        else
                pt(i,1)=x(vec(i-1));
        end
        pt(i,2)=x(vec(i));
        if (i+1)>nm
                pt(i,3)=x(length(x));
        else
                pt(i,3)=x(vec(i+1));
```

```
        end
        funciont(i,:)=trimf(x,[pt(i,1) pt(i,2) pt(i,3)]);
    end

    %%%%%%%%%%% calcula n pendientes de salida (minimos)

    for i=1:n
        clear xxx
        clear fff
        clear fppp
        xxx=pt(i,1):tmuestra:pt(i,3);
        orales=find(xxx(1)==x);
        namas=0;
        for ii=orales:(orales+length(xxx)-1)
            namas=namas+1;
            fff(namas)=f(ii);
        end
        pendientes=polyfit(xxx,fff,1);
        fppp=polyval([pendientes(1) pendientes(2)],xxx);
        subplot(2,2,3), plot(xxx,fppp,'r-','LineWidth',3)
        hold on;
        fp(i,:)=polyval([pendientes(1) pendientes(2)],x);
    end
    title('Rectas de Salida')
    hold off;

    %%%%%%%%%%% Aplicacion de Sugeno

    for ii=1:length(x)
        for i=1:nm
            evaluaciones(i)=fp(i,ii)*funciont(i,ii);
            sumas(i)=funciont(i,ii);
        end
        yt(ii)=sum(evaluaciones)/(sum(sumas)+0.000001);
    end
    yt(1)=yt(2);
    yt(length(yt))=yt(length(yt)-1);

    %%%%%%%%%%%%% Graficas
    subplot(2,2,1),
```

```
plot(x,f,'o','LineWidth',2)
hold on;
for i=1:n
    plot([center(i,1)],[center(i,2)],'k*')
    hold on;
end
title('Datos y Clusters')
hold off;

subplot(2,2,2),
for i=1:n
    plot(x, funciont(i,:),'LineWidth',2)
    hold on;
    plot(x, U(i,:),'r')
    hold on;
end
title('Funciones de Pertenencia')
hold off;
subplot(2,2,4), plot(x,yt,'k.','LineWidth',2)
title('Contorno Sugeno')
clear all



G

Código del programa de calculadora difusa  MATLAB

%%%%%%%%%%%%%%%%%%%%%%
%%% Calculadora Difusa
%%%%%%%%%%%%%%%%%%%%%%

%definición de parámetros
clear all;
xmin=0;% minimo_eje_x=');
xmax=10; %maximo_eje_x=');
res=0.01;
ymin=0;
ymax=1.2;

ejex1=xmin:res:xmax;
```

```
%parámetros triángulo 0
b0=0; %('centro2=');
a0=0; %('inicio2=');
c0=1; %('fin2=');
i=1;
for x=ejex1;
    if (x<a0)|(x>=c0)
        pertentri0(i)=0;
    end
    if (x>=b0)&(x<c0)
        pertentri0(i)=1-(x);
    end
    i=i+1;
end

%parámetros triángulo1
b1=1; %('centro1=');
a1=0; %('inicio1=');
c1=2; %input('fin1=');
i=1;
for x=ejex1;
    if (x<a1)|(x>=c1)
        pertentri1(i)=0;
    end
    if (x>=a1)&(x<b1)
        pertentri1(i)=(x-a1)/(b1-a1);
    end
    if (x>=b1)&(x<c1)
        pertentri1(i)=1-(x-b1)/(c1-b1);
    end
    i=i+1;
end

%parámetros triángulo2
b2=2; %('centro2=');
a2=1.; %('inicio2=');
c2=3.; %('fin2=');
i=1;
for x=ejex1;
    if (x<a2)|(x>=c2)
```

```
            pertentri2(i)=0;
        end
        if (x>=a2)&(x<b2)
            pertentri2(i)=(x-a2)/(b2-a2);
        end
        if (x>=b2)&(x<c2)
            pertentri2(i)=1-(x-b2)/(c2-b2);
        end
        i=i+1;
    end
    %parámetros triángulo3
    b3=3; %('centro2=');
    a3=2; %('inicio2=');
    c3=4; %('fin2=');
    i=1;
    for x=ejex1;
        if (x<a3)|(x>=c3)
            pertentri3(i)=0;
        end
        if (x>=a3)&(x<b3)
            pertentri3(i)=(x-a3)/(b3-a3);
        end
        if (x>=b3)&(x<c3)
            pertentri3(i)=1-(x-b3)/(c3-b3);
        end
        i=i+1;
    end

    %parámetros triángulo4
    b4=4; %('centro1=');
    a4=3; %('inicio1=');
    c4=5; %input('fin1=');
    i=1;
    for x=ejex1;
        if (x<a4)|(x>=c4)
            pertentri4(i)=0;
        end
        if (x>=a4)&(x<b4)
            pertentri4(i)=(x-a4)/(b4-a4);
        end
```

```
        if (x>=b4)&(x<c4)
            pertentri4(i)=1-(x-b4)/(c4-b4);
        end
        i=i+1;
    end

    %parámetros triángulo5
    b5=5; %('centro2=');
    a5=4.; %('inicio2=');
    c5=6.; %('fin2=');
    i=1;
    for x=ejex1;
        if (x<a5)|(x>=c5)
            pertentri5(i)=0;
        end
        if (x>=a5)&(x<b5)
            pertentri5(i)=(x-a5)/(b5-a5);
        end
        if (x>=b5)&(x<c5)
            pertentri5(i)=1-(x-b5)/(c5-b5);
        end
        i=i+1;
    end

    %parámetros triángulo6
    b6=6; %('centro2=');
    a6=5; %('inicio2=');
    c6=7; %('fin2=');
    i=1;
    for x=ejex1;
        if (x<a6)|(x>=c6)
            pertentri6(i)=0;
        end
        if (x>=a6)&(x<b6)
            pertentri6(i)=(x-a6)/(b6-a6);
        end
        if (x>=b6)&(x<c6)
            pertentri6(i)=1-(x-b6)/(c6-b6);
        end
        i=i+1;
```

```
end

%parámetros triángulo7
b7=7; %('centro1=');
a7=6; %('inicio1=');
c7=8; %input('fin1=');
i=1;
for x=ejex1;
    if (x<a7)|(x>=c7)
        pertentri7(i)=0;
    end
    if (x>=a7)&(x<b7)
        pertentri7(i)=(x-a7)/(b7-a7);
    end
    if (x>=b7)&(x<c7)
        pertentri7(i)=1-(x-b7)/(c7-b7);
    end
    i=i+1;
end

%parámetros triángulo8
b8=8; %('centro2=');
a8=7.; %('inicio2=');
c8=9.; %('fin2=');
i=1;
for x=ejex1;
    if (x<a8)|(x>=c8)
        pertentri8(i)=0;
    end
    if (x>=a8)&(x<b8)
        pertentri8(i)=(x-a8)/(b8-a8);
    end
    if (x>=b8)&(x<c8)
        pertentri8(i)=1-(x-b8)/(c8-b8);
    end
    i=i+1;
end
%parámetros triángulo9
b9=9; %('centro2=');
a9=8; %('inicio2=');
```

```
c9=10; %('fin2=');
i=1;
for x=ejex1;
    if (x<a9)|(x>=c9)
        pertentri9(i)=0;
    end
    if (x>=a9)&(x<b9)
        pertentri9(i)=(x-a9)/(b9-a9);
    end
    if (x>=b9)&(x<c9)
        pertentri9(i)=1-(x-b9)/(c9-b9);
    end
    i=i+1;
end
%parámetros triángulo10
b10=10; %('centro2=');
a10=9.; %('inicio2=');
c10=10.; %('fin2=');

i=1;
for x=ejex1;
    if (x<a10)|(x>=c10)
        pertentri10(i)=0;
    end
    if (x>=a10)&(x<b10)
        pertentri10(i)=(x-a10)/(b10-a10);
    end
    if (x>=b10)&(x<c10)
        pertentri10(i)=1-(x-b10)/(c10-b10);
    end
    i=i+1;
end

%%%%%%%%%grafica de triangulos%%%%%%%%%%%%%

plot(ejex1,pertentri0)
title('Funcion de fuzzyficacion triangular');
axis([xmin,xmax,ymin,ymax]);
ylabel('Grado de pertenencia');
hold
```

```
plot (ejex1,pertentri1)
plot (ejex1,pertentri2)
plot (ejex1,pertentri3)
plot (ejex1,pertentri4)
plot (ejex1,pertentri5)
plot (ejex1,pertentri6)
plot (ejex1,pertentri7)
plot (ejex1,pertentri8)
plot (ejex1,pertentri9)
plot (ejex1,pertentri10)

% ahora hay que ver qué triángulo tiene grado de pertenencia

%fuzzificacion entrada arbitraria
E=input('primer numero= ')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GRADOS DE PERTENENCIA EN TRIANGULOS
    if (E<a0)|(E>=c0)
        pertentri0=0;
    end
    if (E>=a0)&(E<b0)
        pertentri0=(E-a0)/(b0-a0);
    end
    if (E>=b0)&(E<c0)
        pertentri0=1-(E-b0)/(c0-b0);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (E<a1)|(E>=c1)
        pertentri1=0;
    end
    if (E>=a1)&(E<b1)
        pertentri1=(E-a1)/(b1-a1);
    end
    if (E>=b1)&(E<c1)
        pertentri1=1-(E-b1)/(c1-b1);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (E<a2)|(E>=c2)
        pertentri2=0;
    end
```

```
if (E>=a2)&(E<b2)
        pertentri2=(E-a2)/(b2-a2);
end
if (E>=b2)&(E<c2)
        pertentri2=1-(E-b2)/(c2-b2);
end
%%%%%%%%%%%%%%%%%%%%
if (E<a3)|(E>=c3)
        pertentri3=0;
end
if (E>=a3)&(E<b3)
        pertentri3=(E-a3)/(b3-a3);
end
if (E>=b3)&(E<c3)
        pertentri3=1-(E-b3)/(c3-b3);
end
%%%%%%%%%%%%%%%%%%%%
if (E<a4)|(E>=c4)
        pertentri4=0;
end
if (E>=a4)&(E<b4)
        pertentri4=(E-a4)/(b4-a4);
end
if (E>=b4)&(E<c4)
        pertentri4=1-(E-b4)/(c4-b4);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a5)|(E>=c5)
        pertentri5=0;
end
if (E>=a5)&(E<b5)
        pertentri5=(E-a5)/(b5-a5);
end
if (E>=b5)&(E<c5)
        pertentri5=1-(E-b5)/(c5-b5);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a6)|(E>=c6)
        pertentri6=0;
end
```

```
if (E>=a6)&(E<b6)
        pertentri6=(E-a6)/(b6-a6);
end
if (E>=b6)&(E<c6)
        pertentri6=1-(E-b6)/(c6-b6);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a7)|(E>=c7)
        pertentri7=0;
end
if (E>=a7)&(E<b7)
        pertentri7=(E-a7)/(b7-a7);
end
if (E>=b7)&(E<c7)
        pertentri7=1-(E-b7)/(c7-b7);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a8)|(E>=c8)
        pertentri8=0;
end
if (E>=a8)&(E<b8)
        pertentri8=(E-a8)/(b8-a8);
end
if (E>=b8)&(E<c8)
        pertentri8=1-(E-b8)/(c8-b8);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a9)|(E>=c9)
        pertentri9=0;
end
if (E>=a9)&(E<b9)
        pertentri9=(E-a9)/(b9-a9);
end
if (E>=b9)&(E<c9)
        pertentri9=1-(E-b9)/(c9-b9);
end
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a10)|(E>=c10)
        pertentri10=0;
end
```

```
        if (E>=a10)&(E<b10)
            pertentri10=(E-a10)/(b10-a10);
        end
        if (E>=b10)&(E<c10)
            pertentri10=1-(E-b10)/(c10-b10);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        num= (0*pertentri0 + 1*pertentri1 + 2*pertentri2 + 3*pertentri3 + 4*pertentri4 + 5*pertentri5 + 6*pertentri6 +
7*pertentri7 + 8*pertentri8 + 9*pertentri9 + 10*pertentri10);
        den= (pertentri0 + pertentri1 + pertentri2 + pertentri3 + pertentri4 + pertentri5 + pertentri6 + pertentri7 + pertentri8 +
pertentri9 + pertentri10);
        salida=(num/den);

        NA=salida
        clear E;
        E=input('segundo numero= ')
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % GRADOS DE PERTENENCIA EN TRIANGULOS
        if (E<a0)|(E>=c0)
            pertentri0=0;
        end
        if (E>=a0)&(E<b0)
            pertentri0=(E-a0)/(b0-a0);
        end
        if (E>=b0)&(E<c0)
            pertentri0=1-(E-b0)/(c0-b0);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if (E<a1)|(E>=c1)
            pertentri1=0;
        end
        if (E>=a1)&(E<b1)
            pertentri1=(E-a1)/(b1-a1);
        end
        if (E>=b1)&(E<c1)
            pertentri1=1-(E-b1)/(c1-b1);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if (E<a2)|(E>=c2)
            pertentri2=0;
```

```
    end
    if (E>=a2)&(E<b2)
        pertentri2=(E-a2)/(b2-a2);
    end
    if (E>=b2)&(E<c2)
        pertentri2=1-(E-b2)/(c2-b2);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (E<a3)|(E>=c3)
        pertentri3=0;
    end
    if (E>=a3)&(E<b3)
        pertentri3=(E-a3)/(b3-a3);
    end
    if (E>=b3)&(E<c3)
        pertentri3=1-(E-b3)/(c3-b3);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (E<a4)|(E>=c4)
        pertentri4=0;
    end
    if (E>=a4)&(E<b4)
        pertentri4=(E-a4)/(b4-a4);
    end
    if (E>=b4)&(E<c4)
        pertentri4=1-(E-b4)/(c4-b4);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (E<a5)|(E>=c5)
        pertentri5=0;
    end
    if (E>=a5)&(E<b5)
        pertentri5=(E-a5)/(b5-a5);
    end
    if (E>=b5)&(E<c5)
        pertentri5=1-(E-b5)/(c5-b5);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (E<a6)|(E>=c6)
        pertentri6=0;
```

```
end
if (E>=a6)&(E<b6)
    pertentri6=(E-a6)/(b6-a6);
end
if (E>=b6)&(E<c6)
    pertentri6=1-(E-b6)/(c6-b6);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a7)|(E>=c7)
    pertentri7=0;
end
if (E>=a7)&(E<b7)
    pertentri7=(E-a7)/(b7-a7);
end
if (E>=b7)&(E<c7)
    pertentri7=1-(E-b7)/(c7-b7);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a8)|(E>=c8)
    pertentri8=0;
end
if (E>=a8)&(E<b8)
    pertentri8=(E-a8)/(b8-a8);
end
if (E>=b8)&(E<c8)
    pertentri8=1-(E-b8)/(c8-b8);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a9)|(E>=c9)
    pertentri9=0;
end
if (E>=a9)&(E<b9)
    pertentri9=(E-a9)/(b9-a9);
end
if (E>=b9)&(E<c9)
    pertentri9=1-(E-b9)/(c9-b9);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (E<a10)|(E>=c10)
    pertentri10=0;
```

```
end
if (E>=a10)&(E<b10)
    pertentri10=(E-a10)/(b10-a10);
end
if (E>=b10)&(E<c10)
    pertentri10=1-(E-b10)/(c10-b10);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num= (0*pertentri0 + 1*pertentri1 + 2*pertentri2 + 3*pertentri3 + 4*pertentri4 + 5*pertentri5 + 6*pertentri6 + 7*pertentri7 + 8*pertentri8 + 9*pertentri9 + 10*pertentri10);
den= (pertentri0 + pertentri1 + pertentri2 + pertentri3 + pertentri4 + pertentri5 + pertentri6 + pertentri7 + pertentri8 + pertentri9 + pertentri10);
salida=(num/den);
NB=salida
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% definicion de operaciones

%E=4;
%while (OP>3)
OP=input('tipo de operación? (1.) suma  (2.) resta  (3.) mutipliacion (4) division= ')
%end
if (OP<=1)
    salidafin=NA+NB;
end
if (OP<=2 & OP>1)
    salidafin=NA-NB;
end
if (OP<=3 & OP>2)
    salidafin=NA*NB;
end
if (OP<=4 & OP>3)
    salidafin=NA/NB;
end
salidafin
```

3

REDES NEURALES ARTIFICIALES

REDES NEURALES BIOLÓGICAS

Las redes neurales artificiales son aproximadores no lineales a la forma en que funciona el cerebro; por lo tanto no deben compararse directamente con el cerebro ni confundir los principios que fundamentan el funcionamiento de las redes neurales artificiales y el cerebro, ni pensar que las redes neurales se basan únicamente en las redes biológicas ya que sólo emulan en una parte muy simple el funcionamiento del cerebro humano. Además se debe considerar que las redes biológicas son generadoras de procesos neurobiológicos en que se establecen relaciones de complejidad muy alta, las cuales no se puede lograr con redes monocapas ni con redes multicapas.

Las RNA pueden estudiarse como aproximadores universales desde el punto de vista matemático. A continuación se presenta un panorama general de los fundamentos biológicos de las redes neurales naturales, sin profundizar en este apasionante campo, ya que el aprendizaje de un tema tan complejo como éste requiere de muchos otros volúmenes para su estudio. Aquí sólo se desarrolla una idea básica que sirve como planteamiento primordial en el estudio de las redes neurales artificiales.

Fundamentos biológicos de las redes neurales naturales

Una neurona biológica es una célula especializada en procesar información. Está compuesta por el cuerpo de la célula (soma) y dos tipos de ramificaciones: el axón y las dendritas. La neurona recibe las señales (impulsos) de otras neuronas a través de sus dendritas y transmite señales generadas por el cuerpo de la célula a través del axón. La figura 3.1 muestra los elementos de una red neuronal natural.

En general, una neurona consta de un cuerpo celular más o menos esférico, de 5 a 10 micras de diámetro, del que salen una rama principal, el axón y varias ramas más cortas que corresponden a las dendritas.

Una de las características de las neuronas es su capacidad de comunicarse. En forma concreta, las dendritas y el cuerpo celular reciben señales de entrada; el cuerpo celular las combina e integra y emite señales de salida. El axón transmite dichas señales a las terminales axónicos, los cuales distribuyen la información. Se calcula que en el cerebro humano existen aproximadamente 10^{15} conexiones.

Las señales que se utilizan son de dos tipos: eléctricas y químicas. La señal generada por la neurona y transportada a lo largo del axón es un impulso eléctrico, mientras que la señal que se transmite entre los terminales axónicos de una neurona y las dendritas de la otra es de origen químico.

Para establecer una similitud directa entre la actividad sináptica y la analogía con las redes neurales artificiales podemos considerar que las señales que llegan a la sinapsis son las entradas a la neurona;

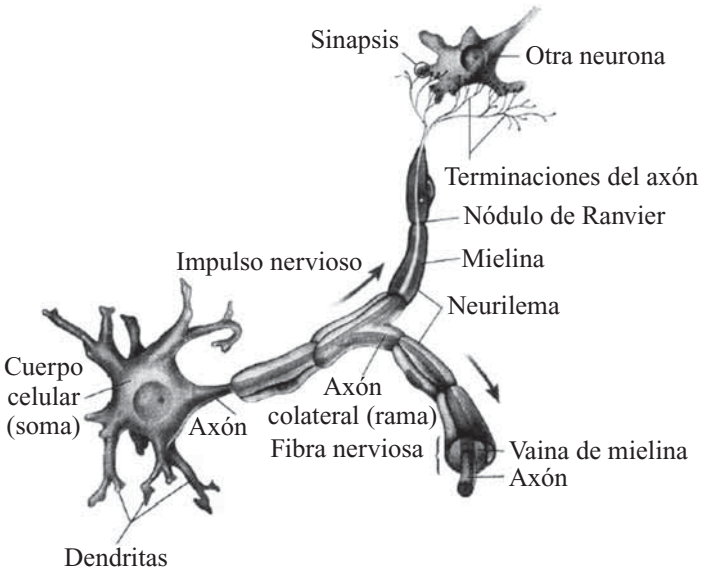


Fig. 3.1 Elementos neurales.

éstas son ponderadas (atenuadas o simplificadas) a través de un parámetro denominado peso, asociado a la sinapsis correspondiente. Estas señales de entrada pueden excitar a la neurona (sinapsis con peso positivo) o inhibirla (peso negativo).

El efecto es la suma de las entradas ponderadas. Si la suma es igual o mayor que el umbral de la neurona, entonces la neurona se activa (da salida). Esta es una situación de todo o nada; cada neurona se activa o no se activa. La facilidad de transmisión de señales se altera mediante la actividad del sistema nervioso. Las sinapsis son susceptibles a la fatiga, deficiencia de oxígeno y la presencia de anestésicos, entre otros. Esta habilidad de ajustar señales es un mecanismo de aprendizaje.

Máquinas inteligentes

La historia de la creación humana de máquinas 'inteligentes' es similar a la historia de la evolución en la naturaleza. Primero surgieron máquinas u organismos cuyo propósito era sólo uno, por lo tanto eran inflexibles. Un ejemplo son las máquinas de cálculo del siglo XIX que sólo servían para una cosa y no eran reprogramables sin cambiar el hardware. Estas máquinas se fueron desarrollando progresivamente hasta hacerse reprogramables, adaptativas e inteligentes.

Las redes neurales artificiales empezaron en 1943. En ese entonces sólo servían como aproximaciones a funciones booleanas. A principios de los años sesenta había muchas expectativas de los alcances que se podrían tener con estas redes, sin embargo no se logró los resultados esperados y la inversión de tiempo y dinero en redes neurales decreció hasta los años ochenta. Durante esta década se empezó a retomar la estructura neuronal, ya que el equipo con que se contaba era muy superior al de los años sesenta. Hoy en día las redes neurales se encuentran en diversas aplicaciones. El equipo computacional que se tiene en la actualidad sigue siendo una restricción al modelaje de estas redes, ya que en realidad un cerebro de humano tiene entre 10^{10} y 10^{11} neuronas. Una neurona consta de cuatro partes principales: dendritas, cuerpo celular, axón y sinapsis. Las dendritas son tejidos delgados en forma de túbulos que se esparcen como si fueran las ramas de un árbol. Generalmente estas dendritas son las receptoras de estímulos.

El cuerpo celular es donde se encuentran los organelos y el ADN y puede funcionar como receptor de estímulos. El axón es otro tejido en forma de túbulo más grande que sirve de conector o transmisor.

Éste se divide al final en varias ramas que acaban en un *bulbo*. Después del bulbo está la sinapsis, que es el espacio entre el bulbo *presináptico* y la dendrita *postsináptica* de otra neurona.

Sistema eléctrico neuronal

El análisis del comportamiento está guiado a los eventos eléctricos o cambios de potenciales entre el interior de la neurona y el exterior.

El interior de la membrana de una neurona tiene un potencial negativo conforme al exterior de la membrana. De igual manera, el interior de la membrana tiene una alta concentración de iones K^+ y baja concentración de iones Na^+, mientras que en el exterior de la membrana es al revés. La alta concentración de K^+ se debe a que como el ion tiene una carga positiva, se siente atraído por la carga negativa del interior de la membrana. La baja concentración de sodio se debe a dos cosas: primero que la membrana es 80 veces más permeable al potasio que al sodio, y segundo porque en el interior de la neurona se encuentra una *bomba de potasio* la cual produce energía cada vez que saca un ion de sodio y mete uno de potasio. Hay que notar que el equilibrio conductivo de los iones de potasio en el axón está por debajo del umbral de activación de la neurona, y que el equilibrio conductivo del sodio está muy por arriba del umbral.

El potencial de membrana cuando se encuentra en reposo es de −60mV genéricamente. Si el potencial llega a subir (hacia lo positivo) hasta un umbral de activación por estímulos externos, se desencadena un proceso retroalimentativo que tiene como resultado un cambio agudo de potencial hacia lo positivo. Si se muestra en un osciloscopio, estos cambios de potencial forman picos ya que el cambio de voltaje conforme al tiempo es exponencial hasta que llega a un límite superior, después del cual cae en picada hasta su valor de reposo o del estímulo presente. Si el estímulo es de 20mV, la neurona genérica (−60mV dentro de la membrana) cambiaría su voltaje a −40mV, que para este caso estaría arriba del umbral de activación (−50mV). En el momento que el voltaje cruza el umbral, empieza la cadena retroalimentativa del sodio hasta llegar a 20mV, donde llega a su límite para recaer a los −40mV, de donde va a volver a empezar.

El proceso del potencial de activación es el siguiente: primero el potencial cambió de −60mV a −40mV. Esto causa que la conductancia (1/Ohm[Ω]) de la membrana aumente, la vuelve menos aislante, permite que aumente la conductancia de los iones sodio del exterior, que a su vez aumentan la conductancia de la membrana. Esto sucede hasta que se alcanza el equilibrio de la conductancia del sodio. Después la concentración del sodio (su conductancia) dentro de la membrana cae drásticamente regresando al estado original.

El hecho de que el potencial sea como un pico, y que éste se active como función del tiempo siempre y cuando el estímulo lleve el voltaje al umbral, nos lleva a imaginar la neurona como un convertidor de voltaje a frecuencia.

Las neuronas son lentas por el ruido, la resistencia y la gran capacitancia que tiene la membrana de la neurona, que hacen más lento el proceso de transmisión. En un sistema en el tiempo se debe tener en cuenta que durante la implementación de este tipo de redes, la transmisión se hace lenta. En el proceso de transmisión se pueden hacer descripciones del comportamiento de las redes a través de circuitos eléctricos empleando elementos básicos como son resistencia, capacitancia e inductancias como lo muestran las siguientes ecuaciones que se emplean para la descripción del proceso de transmisión, dentro de la membrana.

La ecuación de la corriente en función de la resistencia, el voltaje y la capacitancia de la membrana es:

$$i_m = \frac{V}{r_m} + c_m \frac{dV}{dt} \qquad [3.1]$$

donde i_m, es la corriente de la membrana, t es el tiempo, r_m es la resistencia de la membrana y c_m la capacitancia de la membrana.

Es aquí donde se ve que entre más grande sea el capacitor, mayor la corriente o mayor el tiempo requerido para cambiar el voltaje. Esto ayuda a explicar la lentitud de la transmisión ya que la membrana

tiene una gran capacitancia. Análogamente, la ecuación que describe la corriente en términos de una diferencia de longitud o de espacio es la siguiente:

$$i_m = \left(\frac{1}{r_e + r_i} \right) \frac{d^2V}{dx^2} \qquad [3.2]$$

donde i_m es la corriente de la membrana, x es la distancia, r_e es la resistencia de la membrana y r_i la capacitancia de la membrana.
Haciendo un cambio de variable:

$$\tau = r_m c_m$$

$$\lambda^2 = \frac{r_m}{r_e + r_i}$$

Despejando ambas ecuaciones para el voltaje nos da:

$$V = \lambda^2 \frac{d^2V}{dx^2} - \tau \frac{\partial V}{\partial t} \qquad [3.3]$$

Todo este análisis lleva a la demostración de la lentitud de la señal transmitida por los potenciales de acción. Resolviendo la ecuación diferencial y dejando el voltaje en términos del tiempo veremos que la respuesta del circuito a un impulso es de tipo logarítmico, por lo cual si se manda una señal que varía con una frecuencia de 1kHz y la respuesta es tan lenta, entonces la señal resultante va a ser muy distinta a la inicial. Como la resistencia y capacitancia de la membrana es muy alta, y la caída de voltaje conforme a la distancia es exponencial, sería extremadamente ineficiente transmitir el impulso eléctrico tal cual. Para esto están los potenciales de acción que son como piezas de dominó. Una vez que se activa uno, éste hace que se active su vecino más próximo, y éste a su vez el suyo, y así hasta llegar al bulbo.

Lo único malo de este mecanismo de transmisión es que es lento, y aumentando el diámetro del axón sólo aumenta la velocidad en proporciones de raíz cuadrada. Por ello los vertebrados tienen un recubrimiento en sus axones con mielina que aumenta bastante la velocidad de transmisión. La velocidad de transmisión de los humanos es de 120 m/s, mientras que la de los invertebrados llega a 25 m/s.

MODELOS DE NEURONAS

Una sinapsis tiene dos lados: el presináptico (axón) y el postsináptico (dendrita normalmente). Cuando un potencial de activación llega a la sinapsis hay un retardo de alrededor de 2 ms. Esto se debe a los dos procesos de difusión que se llevan a cabo en la sinapsis. Primero iones de calcio (Ca^{2+}) entran al bulbo presináptico para favorecer que la vesículas del bulbo liberen neurotransmisores. Después las vesículas se funden en la membrana presináptica liberando los neurotransmisores en la hendidura sináptica (el espacio entre los dos tejidos), y éstos cuando llegan a la dendrita causan un cambio de voltaje o de conductancia de ciertos iones. El impulso postsináptico (PPS) es inhibitorio (PPSI) o excitador (PPSE) dependiendo de qué iones sean cambiados.

Una sola neurona postsináptica puede tener varios miles de sinapsis en ella. El efecto resultante en modelos cuando hay más de un PPS es la suma algebraica de los PPSI y los PPSE, en donde los PPSI se restan a los PPSE. Sin embargo, en la realidad la integración de varios PPS es más complicada; es una función que depende del tipo de célula y de dónde están localizadas las sinapsis relativamente entre sí. Esto lleva a interacciones no lineales de varios PPS.

Teóricamente, el cálculo de la función integradora varía mucho de un caso a otro y es bastante complejo para obtenerlo en forma teórica para cada caso, por lo cual se trata de obtener valores experimentales. Sin embargo, éstos no son abundantes, de manera que los datos actuales quizás sean sólo una

muestra pequeña de algo más diverso de lo previsto. Aparte de que la integración sináptica realmente no es lineal, la única forma para que se sumen dos PPS idealmente, es que tengan un mismo lugar en el tiempo. Los PPS, a diferencia del potencial de activación, tienen una fase mucho más grande y duran más tiempo decayendo lentamente.[1] Por ello, si se graficara el potencial en un punto de una dendrita en función del tiempo, se vería que los PPS tienen un pico y una caída lenta, por lo que la suma normalmente se daría con PPS desplazados.

Otro efecto importante de los PPS es que mientras más lejos de la fuente, más "aplastados" serán. En su recorrido por las dendritas los PPS pasan por filtros de paso bajo. Esto quiere decir que entre más lejos del soma (cuerpo celular), menor magnitud o amplitud tendrán sus impulsos.

La teoría del potencial lento dice básicamente que lo que la neurona presináptica en verdad transmite es el resultado postsináptico del impulso. Como los impulsos postsinápticos son lentos y alargados, es mucho más fácil que se pueda dar una integración temporal. Por lo tanto, el verdadero procesador analógico dentro de las neuronas son las dendritas con ayuda de los neurotransmisores.

Una vez que la integración temporal se da entre los impulsos, el resultante llega de nuevo al axón, que convierte este PPS resultante en un nuevo potencial de acción con alta frecuencia y poca amplitud.

La teoría del potencial lento entonces también propone que los axones se utilizan para hacer más eficiente la transmisión de datos o impulsos. Si la información no tuviera que viajar tanto, y las neuronas estuvieran más pegadas entre sí, los axones vendrían sobrando.

Ruido

Se dice que la transmisión neuronal es ruidosa por tres razones:

1. La formación de un potencial análogo a través de flujo de iones discreto causa ruido discreto.
2. La diferencia en la liberación de neurotransmisores que está en función del voltaje de entrada a la presinápsis, a los iones en el medio y a la cantidad de neurotransmisores dentro de cada vesícula.
3. La capacidad limitada de los PPS para representar la frecuencia de los potenciales de acción.

Neuronas de dos estados

Desarrolladas por Pitts y McCulloch, son neuronas que aceptan por valores de entrada uno o cero y tienen un umbral de activación de uno o de dos, formando así en el primer caso un OR lógico y en el segundo un AND lógico. La integración de los PPS es totalmente lineal. En su tiempo eran lo más aproximado a una neurona, pero hoy se sabe que esta representación de proposiciones lógicas no es del todo correcta.

En una clase más realista de modelos, los integradores, se distinguen dos modelos: *integración-y-disparo* e *integración-y-disparo olvidadizo*. En el primero se tiene una señal de entrada s(t) que describe el cambio de voltaje u(t) con el tiempo, es decir, su derivada conforme al tiempo. Si se asume que s(t) es una constante, entonces se puede deducir matemáticamente que la frecuencia de picos o potencial es igual a la constante s entre el umbral de activación.

$$f = \frac{s}{\gamma} \qquad [3.4]$$

Donde γ es el umbral de activación, f es la frecuencia de picos y s es la constante que describe la derivada de $u(t)$.

La neurona genérica

Dentro de la neurona se lleva a cabo un proceso de transmisión y procesamiento de datos que se realiza en dos etapas. La primera es la suma algebraica de las entradas de las sinapsis, este es el potencial lento

[1] Lento equivale a aproximadamente 2 ms.

que es alimentado a una función no lineal. La segunda etapa se realiza con la evalución de esta función obteniendo el valor de salida de la neurona.

"*La neurona genérica conectivista no dispara potenciales de acción pero tiene una salida que es un nivel de actividad continuamente graduado.*"

APLICACIONES DE LAS REDES NEURALES ARTIFICIALES (RNA)

Algunas áreas donde se aplican RNA son:

- Automóviles: Sistemas de piloto automático. Detección de fallas por reconocimiento externo de vibraciones.
- Bancos: Lectura de cheques y otros documentos. Evaluación de aplicaciones de créditos.
- Electrónica: Predicción de secuencia de códigos. Distribución de elementos en CI. Control de procesos. Análisis de fallas. Visión artificial. Reconocimiento de voz.
- Finanzas: Tasación real de los bienes. Asesoría de préstamos. Previsión en la evolución de precios. Seguimiento de hipotecas. Análisis de uso de línea de crédito. Evaluación del riesgo en créditos. Identificación de falsificaciones. Interpretación y reconocimiento de firmas.
- Manufactura: Control de la producción y del proceso. Análisis y diseño de productos. Diagnóstico de fallas en el proceso y maquinarias. Identificación de partículas en tiempo real. Inspección de calidad mediante sistemas visuales. Análisis de mantenimiento de máquinas.
- Medicina: Análisis de células portadoras de cáncer mamario. Análisis de electroencefalograma y de electrocardiograma. Reconocimiento de infartos mediante ECG. Diseño de prótesis. Optimización en tiempos de trasplante. Reducción de gastos hospitalarios.
- Robótica: Control dinámico de trayectoria. Robots elevadores. Controladores. Sistemas ópticos.
- Seguridad: Códigos de seguridad adaptivos. Criptografía. Reconocimiento de huellas digitales.
- Telecomunicaciones: Compresión de datos e imágenes. Automatización de servicios de información. Traslación en tiempo real de lenguaje hablado.
- Transporte: Diagnóstico de frenos en camiones. Sistemas de ruteo y seguimiento de flotas.
- Voz: Reconocimiento de voz. Compresión de voz. Clasificación de vocales. Transformación de texto escrito a voz.

DEFINICIÓN DE UNA RED NEURONAL ARTIFICIAL

Las RNA se definen como sistemas de mapeos no lineales cuya estructura se basa en principios observados en los sistemas nerviosos de humanos y animales. Constan de un número grande de procesadores simples ligados por conexiones con pesos. Las unidades de procesamiento se denominan neuronas. Cada unidad recibe entradas de otros nodos y genera una salida simple escalar que depende de la información local disponible, guardada internamente o que llega a través de las conexiones con pesos. Pueden realizarse muchas funciones complejas dependiendo de las conexiones.

Las neuronas artificiales simples fueron introducidas por McCulloch y Pitts en 1943. Una red neuronal se caracteriza por los siguientes elementos:

1. Un conjunto de unidades de procesamiento o neuronas.
2. Un estado de activación para cada unidad, equivalente a la salida de la unidad.
3. Conexiones entre las unidades, generalmente definidas por un peso que determina el efecto de una señal de entrada en la unidad.

4. Una regla de propagación, que determina la entrada efectiva de una unidad a partir de las entradas externas.
5. Una función de activación que actualiza el nuevo nivel de activación basándose en la entrada efectiva y la activación anterior.
6. Una entrada externa que corresponde a un término determinado como bias para cada unidad.
7. Un método para reunir la información, correspondiente a la regla del aprendizaje
8. Un ambiente en el que el sistema va a operar, con señales de entrada e incluso señales de error.

En muchas redes las unidades de proceso tienen respuesta de la forma:

$$y = f\left(\sum_k \omega_k x_k\right) \qquad [3.5]$$

donde:

x_k : señales de salida de otros nodos o entradas externas.
ω_k : pesos de las ligas de conexión.
$f(\bullet)$: función no lineal simple.

La función *f* puede ser sigmoidal, tangente hiperbólica, escalón, entre otras. En MATLAB® se tiene diferentes funciones de activación como tansig, hardlim y purelin, entre otras, lo cual facilita las aproximaciones que se requieran hacer, empleando RNA.

Cada unidad de proceso tiene una tarea simple: recibe la entrada de otras unidades o de fuentes externas y procesa la información para obtener una salida que se propaga a otras unidades.

Una red puede tener una estructura arbitraria, pero las capas que contienen estas estructuras están definidas de acuerdo con su ubicación en la topología de la red neuronal. Las entradas externas son aplicadas en la primera capa, y las salidas se consideran la última capa. Las capas internas que no se observan como entradas o salidas se denominan *capas ocultas*. Por convención, las entradas no se consideran como capa porque no realizan procesamiento.

La entrada total *u* de una unidad *k* es la suma de los pesos de las entradas conectadas, más un bias θ:

$$u = \sum_j \omega_j x_j + \theta. \qquad [3.6]$$

Si el peso ω_j es positivo se habla de una *excitación* y si el peso es negativo se considera una *inhibición* de la entrada. Si consideramos a las entradas como funciones del tiempo, la expresión anterior se convierte en:

$$u(t) = \sum_j \omega_j(t) x_j(t) + \theta(t). \qquad [3.7]$$

FUNCIONES DE ACTIVACIÓN

La regla que logra establecer el efecto de la entrada total $u(t)$ en la activación de la unidad *k* se denomina función de activación (F_k):

$$y(t + 1) = F_k(y(t), u(t)). \qquad [3.8]$$

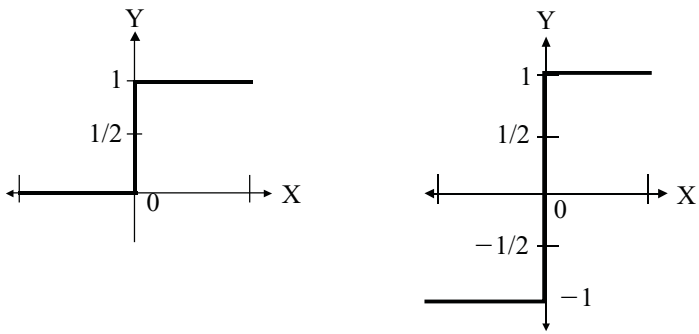
En muchas ocasiones esta función es de la forma no decreciente respecto a la entrada total de la unidad:

$$y(t+1) = F_k\left(\sum_j \omega_j(t) x_j(t) + \theta(t)\right).$$

Algunas de las funciones de activación más usadas son las siguientes:

Función escalón

La función de activación escalón se asocia a neuronas binarias en las cuales, cuando la suma de las entradas es mayor o igual que el umbral de la neurona, la activación es 1; si es menor, la activación es 0 (o −1).

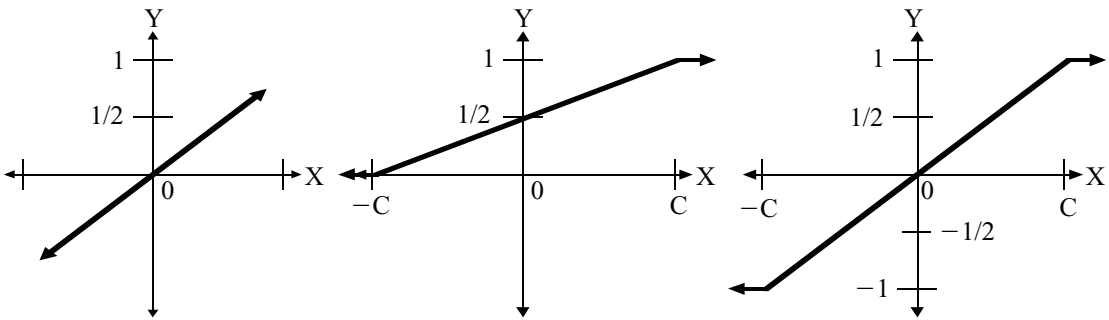


$$F_k(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \qquad F_k(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}$$

Fig. 3.2 Función de activación escalón.

Función lineal y mixta

La función lineal o identidad responde a la expresión $F_k(u) = u$. En las neuronas con función mixta, si la suma de las señales de entrada es menor que un límite inferior, la función se define como 0 (o −1). Si dicha suma es mayor o igual que el límite superior, entonces la activación es 1. Si la suma de entrada está comprendida entre los dos límites, entonces la activación es 1. Si la suma de entrada está comprendida entre ambos límites, superior e inferior, entonces la activación se define como una función lineal de la suma de las señales de entrada (véase la figura 3.3).



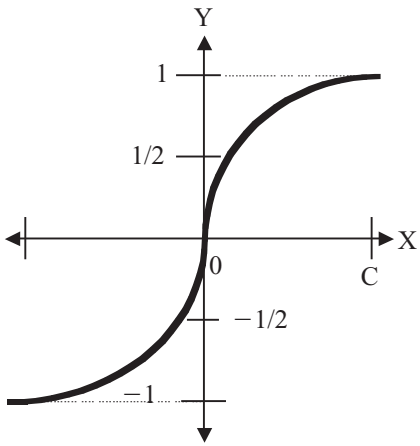
$$F_k(x) = x \qquad F_k(x) = \begin{cases} 0 & \text{si } x < -c \\ 1 & \text{si } x > c \\ \dfrac{x}{2c} + \dfrac{1}{2} & \text{en otro caso} \end{cases} \qquad F_k(x) = \begin{cases} -1 & \text{si } x < -c \\ 1 & \text{si } x > c \\ ax & \text{en otro caso} \end{cases}$$

Fig. 3.3 Función de activación lineal y mixta.

Función tangente hiperbólica

La función de activación tangente hiperbólica se emplea en los casos que presentan variaciones suaves de valores positivos y negativos de la señal a clasificar. Como se puede ver en su descripción en la figura 3.4, es una de las funciones más empleadas en entrenamientos supervisados, como en el caso del entrenamiento de retropropagación del error.

Debe tenerse cuidado de emplear esta figura entre los umbrales postivos y negativos antes de la saturación, de otra forma la salida siempre generara valores saturados iguales a 1 y -1.

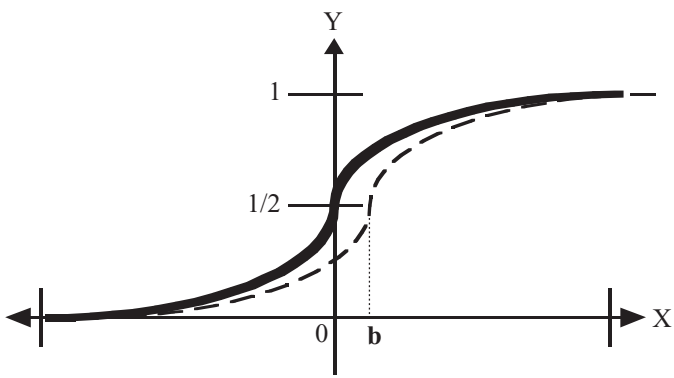


$$F_k(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

Fig. 3.4 Función tangente hiperbólica.

Función sigmoidal

Con la función sigmoidal el valor dado por la función es cercano a uno de los valores asintóticos. Esto hace que en la mayoría de los casos, el valor de salida esté comprendido en la zona alta o baja del sigmoide. De hecho, cuando la pendiente es elevada, esta función tiende a la función escalón. Sin embargo, la importancia de la función sigmoidal es que su derivada siempre es positiva y cercana a cero para los valores grandes positivos o negativos; además, toma su valor máximo cuando $x = 0$. Esto hace que se puedan utilizar reglas de aprendizaje definidas para las funciones escalón, con la ventaja, respecto a esta función, de que la derivada está definida en todo el intervalo (véase la figura 3.5).

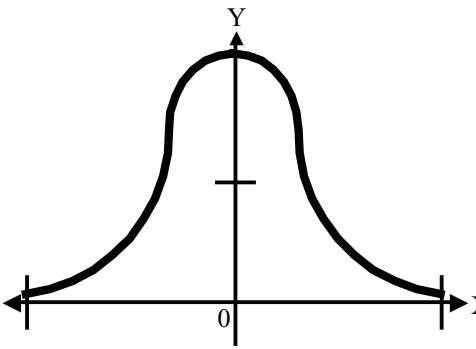


$$F_k(u) = \frac{1}{1 + e^{-u}}$$

Fig. 3.5 Función sigmoidal.

Función de Gauss

Los mapeos ocultos algunas veces pueden realizarse con un solo nivel de neuronas mediante el uso de funciones de activación tipo Gauss, en lugar de funciones tipo sigmoidales (véase la figura 3.6).



$$F_k(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{(\frac{x^2}{2\sigma^2})}$$

Fig. 3.6 Función de activación gaussiana.

TOPOLOGÍAS DE LAS REDES NEURALES

Dos de las topologías más usadas, de acuerdo con las diferencias en la manera de realizar las conexiones, son:

a) Redes de propagación hacia delante (feed-foward): el flujo de información de las entradas a las salidas es exclusivamente hacia delante, extendiéndose por capas múltiples de unidades, pero no hay ninguna conexión de retroalimentación.
b) Redes recurrentes: contienen conexiones de retroalimentación, lo que puede derivarse en un proceso de evolución hacia un estado estable en el que no haya cambios en el estado de activación de las neuronas.

Elementos de una red neuronal artificial

Una RNA consta de un conjunto de elementos de procesamiento conectados entre sí y entre los que se envían información a través de conexiones. Un esquema básico de una red neuronal artificial se observa en la figura 3.7, la cual presenta las diferentes capas que tiene esta topología, que es una estructura que se conoce con el nombre de feed-forward (hacia delante) debido al flujo de la información.

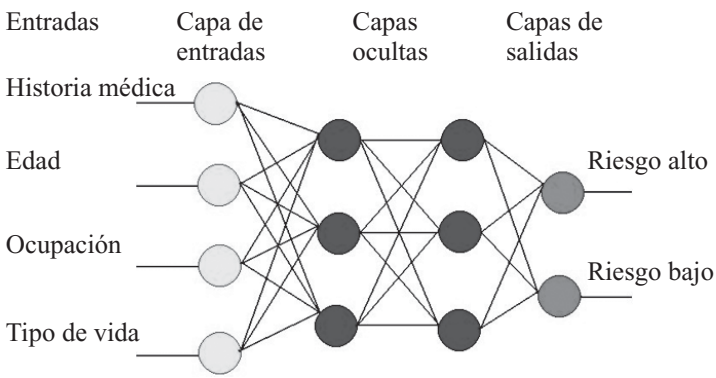


Fig. 3.7 Esquema básico de una RNA.

Los elementos básicos de una RNA son:

- Conjuntos de unidades de procesamiento (neuronas).
- Conexiones entre unidades (asociado a cada conexión un peso o valor).
- Funciones de salida o activación para cada unidad de procesamiento.

El modelo de la neurona y la arquitectura de la red describen cómo esta última transforma sus entradas en las salidas. Los elementos individuales de cálculo que forman la mayoría de los modelos de sistemas neurales artificiales, se conocen como neuronas artificiales.

ENTRENAMIENTO DE LAS REDES NEURALES

Se denomina entrenamiento al proceso de configuración de una red neuronal para que las entradas produzcan las salidas deseadas a través del fortalecimiento de las conexiones. Una forma de llevar esto a cabo es a partir del establecimiento de pesos conocidos con anterioridad, y otro método implica el uso de técnicas de retroalimentación y patrones de aprendizaje que cambian los pesos hasta encontrar los adecuados.

Además, el aprendizaje puede dividirse en *supervisado* o *asociativo* y *no supervisado* o *auto-organizado*. En el primer caso se introducen entradas que corresponden a determinadas salidas, ya sea por un agente externo o por el mismo sistema. En el segundo caso el entrenamiento se enfoca a encontrar características estadísticas entre agrupamientos de patrones en las entradas.

Un tipo de regla que se usa para el entrenamiento mediante el ajuste de pesos es la *Hebbiana*, propuesta por Hebb en 1949 y que ha dado pie a diversas variantes propuestas en el transcurso del tiempo. Si dos unidades j y k están activas al mismo tiempo, la conexión entre las dos debe ser fortalecida mediante la modificación del peso:

$$\Delta\omega_{jk} = \gamma\, y_j y_k\,. \qquad [3.9]$$

donde γ es una constante de proporcionalidad positiva que representa la tasa de aprendizaje.

Otra regla usada comúnmente implica el ajuste de los pesos a través de la diferencia entre la activación actual y la deseada; se le conoce como *Regla Delta*:

$$\Delta\omega_{jk} = \gamma\, y_j (d_k - y_k). \qquad [3.10]$$

donde d_k es la activación deseada.

REDES DE UNA CAPA

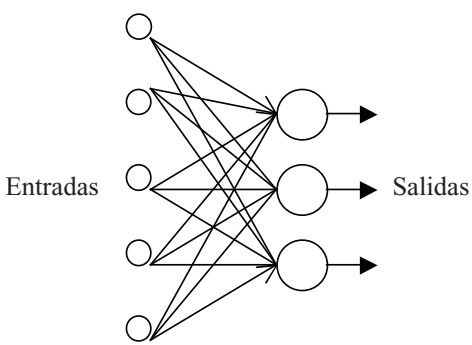


Fig. 3.8 Red de una capa.

Las redes de una capa, como la que se muestra en la figura 3.8, presentan las entradas directamente conectadas a las salidas mediante pesos. Las salidas no interactúan, por lo que una red con N_{out} salidas puede ser analizada como N_{out} redes separadas. Cada unidad como las que muestra la figura 3.9 produce salidas de la forma:

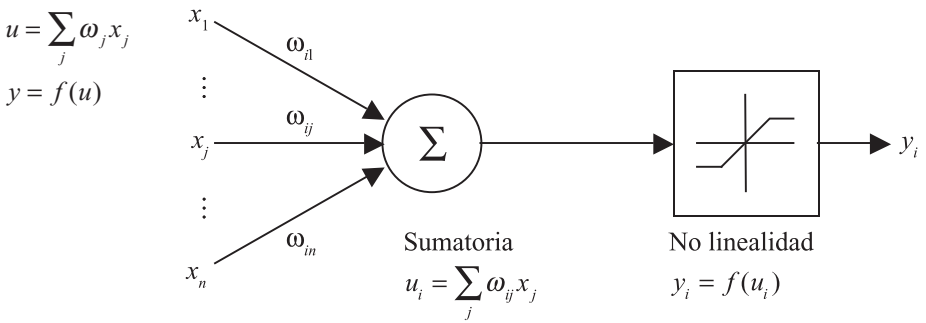


Fig. 3.9 Función del nodo.

En notación vectorial se expresa como:

$$y(\vec{x}) = f(\vec{\omega}^T \vec{x}). \qquad [3.11]$$

Los puntos de $\vec{x}$ con una suma constante $\sum_j \omega_j x$ definen un hiperplano perpendicular al vector $\vec{\omega}$. La orientación del hiperplano se determina por la dirección de $\vec{\omega}$, dependiendo de los pesos pero no de la magnitud de $\vec{\omega}$.

La suma $\sum_j \omega_j x_j = 0$ define un hiperplano que cruza el origen. La introducción de un término θ denominado bias en la sumatoria el hiperplano tiene mayor movilidad y permite la separación de algunos conjuntos de datos, con lo que se obtiene mejores clasificaciones.

Perceptrón

Los perceptrones de una capa pueden clasificar correctamente los conjuntos de datos que son linealmente separables, esto es, que pueden ser separados por un hiperplano. En dos dimensiones esto puede verse como una línea recta que separa dos clases (figura 3.10).

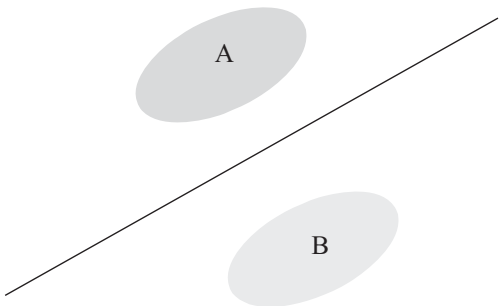


Fig. 3.10 Separación lineal de los conjuntos A y B.

Un perceptrón convencional tiene una función de no linealidad binaria y la topología que muestra la figura 3.11.

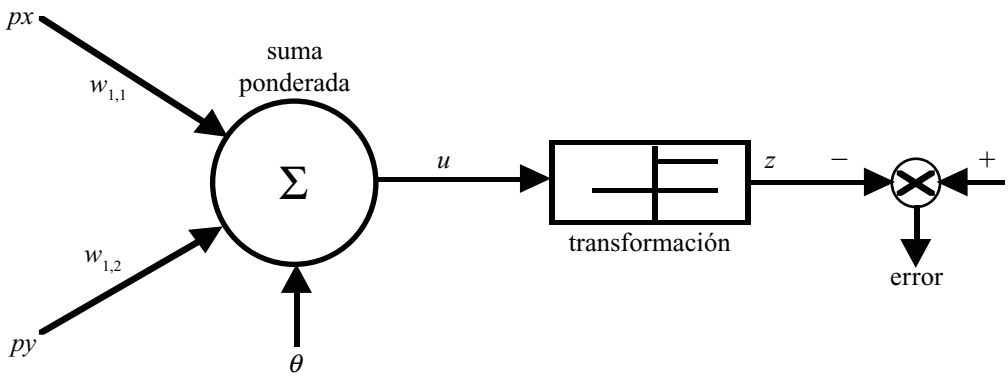


Fig. 3.11 Perceptrón clásico propuesto por McCulloch - Pitts.

El algoritmo de aprendizaje del perceptrón funciona para aprender funciones binarias linealmente separables, ya que de otra manera el algoritmo no convergería ni produciría la mejor solución. Debido a que las salidas son binarias, se emplean unidades lineales de umbral. Cada unidad calcula la suma u con pesos de las N entradas $x_j, j = 1 \ldots N$, y genera una salida binaria y:

$$u = \sum_{j=0}^{N} \omega_j x_j = \vec{\omega}^T \vec{x} \qquad [3.12]$$

$$y = \begin{cases} -1 & u \le 0 \\ +1 & u > 0 \end{cases}.$$

De una forma alterna, se tiene en un intervalo de cero a uno,

$$y = \begin{cases} -1 & u \le 0 \\ 0 & u > 0 \end{cases}$$

La constante de bias es 1, siendo las entradas, salidas y valores objetivo asumidos como ± 1 o cero.

Los pesos son actualizados por un número de reglas simples, comparando las salidas $y(\vec{x})$ con los objetivos $t(\vec{x})$. Los pesos son adaptados con la ecuación:

$$\Delta\vec{\omega} = \begin{cases} 2\eta t\vec{x} & \text{si } t \neq y \\ 0 & \text{otros casos} \end{cases} \qquad [3.13]$$

donde:

η es una constante positiva pequeña que controla la tasa de aprendizaje, usualmente entre 0 y 1.

Al mejorar la exactitud de la clasificación, el sistema comete menos errores y los cambios en los pesos se vuelven menos frecuentes. Una tasa efectiva de aprendizaje puede alentar el proceso, por lo que lograr la clasificación perfecta puede tomar un largo tiempo.

Las redes de una capa y con función de activación tipo sigmoidal se llaman normalmente perceptrones, aunque el perceptrón original de Rosenblatt consistía de una familia de redes, de más de una capa, y con conexiones recurrentes.

La forma de entrenamiento se puede realizar de una manera recursiva empleando la siguiente expresión.

$$\Delta\vec{\omega} = \omega(k+1) = \omega(k) + \eta x(k) e(k)$$
$$\text{donde} \quad e(k) = yd(k) - y(k) \qquad [3.14]$$

Se define el error como $e(k)$, el valor deseado $yd(k)$ y valor actual de salida de la neurona como $y(k)$.

El algoritmo se define como:

1) fijar pesos iniciales, con valores aleatorios
2) establecer los valores de entradas x1,x2..,xn
3) calcular la salida de la neurona

   $y(k) = f(\sum Xiwi - \theta)$

4) Actualizar los pesos

   $\Delta\vec{\omega} = \omega(k+1) = \omega(k) + \eta x(k) e(k)$
   donde $e(k) = yd(k) - y(k)$

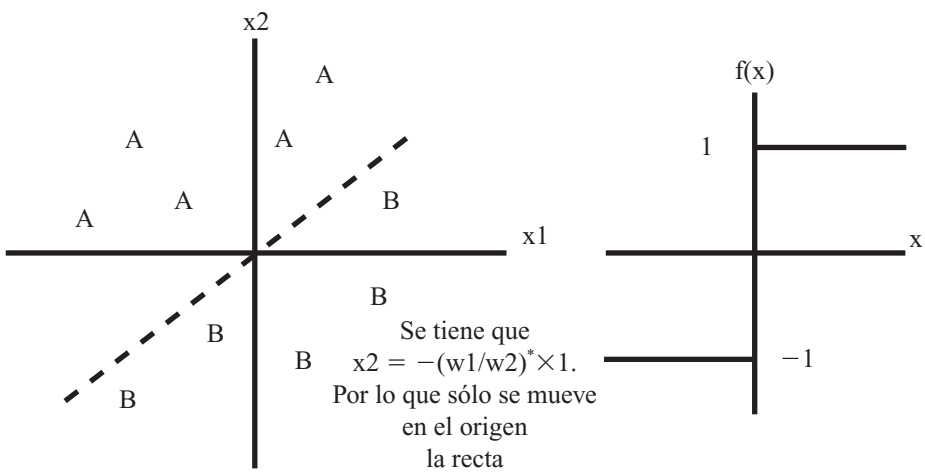
5) continuar hasta

   $e(k) \leq \xi$
   ξ = Tolerancia

En caso de no cumplir con la condición del paso 5, se regresa de nuevo al punto 3.

Separación de variables linealmente separables con el perceptrón

Un perceptrón de dos entradas puede separar un plano en dos secciones, ya que su ecuación forma una línea recta, como muestra la figura 3.12.



El perceptrón puede clasificar un vector analógico de entrada en dos clases A y B.

Fig. 3.12 Clasificación de patrones con el perceptrón.

Considerando las entradas x_1 y x_2, con sus respectivos pesos ω_1 y ω_2 y un bias θ, se obtiene la ecuación:

$$\omega_1 x_1 + \omega_2 x_2 + \theta = 0\,. \qquad [3.15]$$

Despejando resulta una ecuación donde los pesos determinan la pendiente de la línea y el bias determina el desplazamiento vertical de la clasificación lineal:

$$x_2 = -\frac{\omega_1}{\omega_2} x_1 - \frac{\theta}{\omega_2}. \qquad [3.16]$$

Esta recta se denomina frontera de decisión y es ortonormal al vector de los pesos de la red.

Ejemplo

Para clasificar patrones linealmente separables, se propone un perceptrón de dos entradas para ser entrenado, con pesos iniciales $\omega_1 = -0.1$, $\omega_2 = 0.8$ y un bias de $\theta = -0.5$, que se muestra en la figura 3.13 con los puntos que se desea clasificar a través de una línea recta.

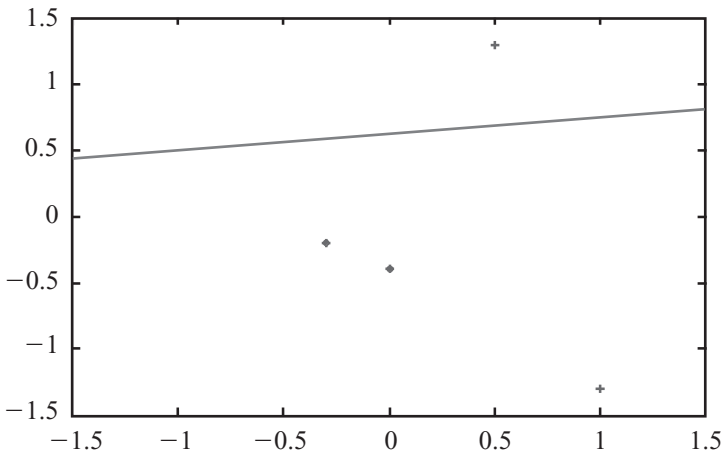


Fig. 3.13 Perceptrón original propuesto.

Después de entrenar el perceptrón en MATLAB, se obtuvo una línea que se ve en la figura 3.14, con pesos $\omega_1 = 2.7$, $\omega_2 = 0.5$.

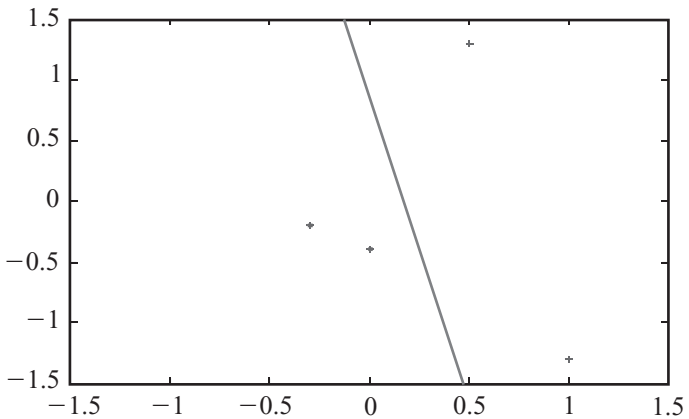


Fig. 3.14 Perceptrón entrenado.

ADALINE (Adaptive Linear Neuron)

La red ADALINE tiene una topología similar a la del perceptrón, excepto porque emplea una función de activación lineal, usando para su entrenamiento un método de mínimos cuadrados (LMS).

Entre sus aplicaciones se encuentran:

- Procesamiento de señales.
- Canceladores de ECO, en señales telefónicas.
- Sistemas de predicción.

Un esquema de predicción básico es el siguiente, en el cual se presenta un entrenamiento con datos del sistema tanto de la entrada como de la salida. Después de realizado el entrenamiento se elimina la entrada de datos de la salida del sistema, como lo muestra la figura 3.15.

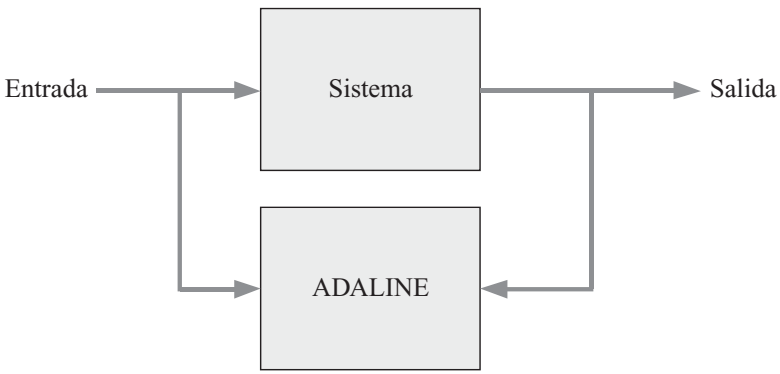


Fig. 3.15 Sistema de entrenamiento para una red ADALINE.

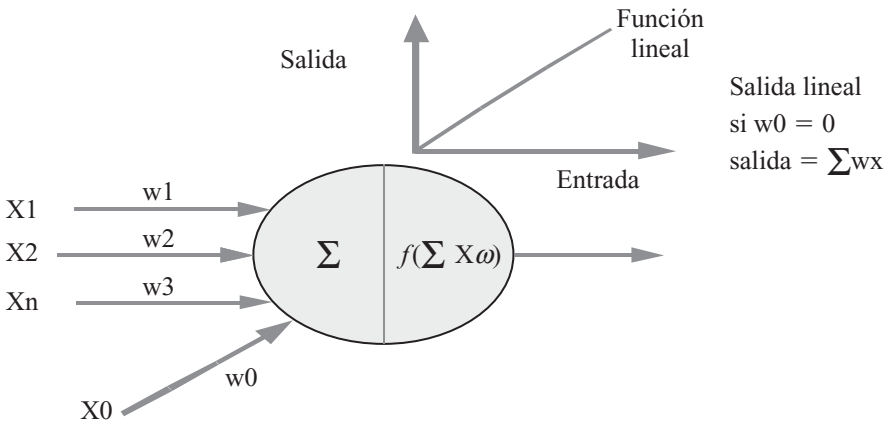


Fig. 3.16 ADALINE (con salida de activación lineal).

Las redes ADALINE son estructuras de una capa con una función de escalón como no linealidad. Adicionalmente, las entradas pueden ser de forma continua, mientras que en el perceptrón son binarias de ± 1. Los pesos son ajustados mediante la regla de Widrow-Hoff para minimizar la diferencia entre la salida y el objetivo. Este algoritmo es una implementación iterativa de la regresión lineal, reduciendo el error cuadrado de un ajuste lineal.

Los pesos se actualizan de manera general por:

$$\Delta\vec{\omega} = \eta(t-u)\frac{\vec{x}}{\left\|\vec{x}\right\|^2}. \qquad [3.17]$$

Empleando el error cuadrático medio se tiene

$$e^2 = \frac{1}{2}\sum_{k=1}^{h} e^2 \qquad [3.18]$$

donde h es el número de patrones de entrada; partiendo de la definición básica del error se tiene $e = (d(k) - s(k))$, calculando la salida por $s(k) = \sum_{j=1}^{n} x_j w_j$, como se desea reducir el gradiente descendiente se puede expresar a través de $\Delta\omega_j = -\alpha\left[\frac{\partial e_k^2}{\partial \omega_i}\right]$

donde α = *coeficiente de entrenamiento*

$$\Delta\omega = -\alpha\frac{\partial e_k^2}{\partial s_k}\frac{\partial s_k}{\partial \omega_i} \qquad [3.19]$$

Si S_k, se tiene una salida lineal, $s(k) = \sum_{j=1}^{n} x_j w_j$, se puede $\frac{\partial e_k^2}{\partial \omega_i} = -e_k x_k$

La actualización de los pesos se puede realizar por la siguiente expresión:

$$\Delta\omega = \omega(k) + \alpha x(k) e(k) \qquad [3.20]$$

En el caso del perceptrón con una salida que contiene umbrales, es posible emplear esta metodología y una vez que se ajustan los pesos cambiar por la función de umbral. Una de las funciones más empleadas en esta topología de entrenamiento es la sigmoidal.

En caso de sólo tomar el error cuadrático se describe por $e^{medio} = \sum_{k=1}^{h} e^2$, el valor deseado depende del gradiente del error con respecto a los pesos

$$\frac{\partial e^{medio}}{\partial \omega} = \frac{\partial e_k^{medio}}{\partial s_k}\frac{\partial s_k}{\partial \omega_i},$$

tomando en cuenta la salida lineal $s(k) = \sum_{j=1}^{n} x_j w_j$, se tiene que $\frac{\partial s_k}{\partial \omega_i} = X$

Desarrollando $\frac{\partial e_k^{medio}}{\partial s_k} X = \frac{\partial e_k}{\partial \omega_k}$,

asumiendo que $\frac{\partial e_k^{medio}}{\partial s_k} = -2(d-f)\frac{\partial f}{\partial s}$,

se define $\frac{\partial e^{medio}}{\partial \omega} = \frac{\partial e_k^{medio}}{\partial s_k}\frac{\partial s_k}{\partial \omega_i} = -2(d-f)\frac{\partial f}{\partial s}X$,

hay que recordar que para el caso del perceptrón no se puede calcular esta derivada de la función, mientras que si aproximamos $f = s, \quad \therefore \quad \frac{\partial f}{\partial s} = 1$

que implica que $\frac{\partial e^{medio}}{\partial \omega} = \frac{\partial e_k^{medio}}{\partial s_k}\frac{\partial s_k}{\partial \omega_i} = -2(d-f)X$,

si se modifican teniendo en cuenta que es un gradiente negativo y el valor de 2 se incluye dentro de la constante de entrenamiento, se considera la misma expresión de actualización de los pesos.

$$\Delta\omega = \omega(k) + \alpha x(k) e(k).$$

El procedimiento propuesto por Werbos se basa en emplear la función sigmoidal

$$f(s) = \frac{1}{1+e^{-s}}, \qquad [3.21]$$

donde S es entrada.

Un punto importante a resaltar es que en el empleo de un algoritmo iterartivo se puede calcular la derivada de la función sigmoidal como $f(s)' = \frac{\partial f}{\partial s} = f(1-f)$ que facilita el proceso iterativo de la derivada en un programa de entrenamiento.

Si sustituimos la derivada en el cálculo del gradiente se tiene

$$\frac{\partial e^{medio}}{\partial \omega} = \frac{\partial e_k^{medio}}{\partial s_k} \frac{\partial s_k}{\partial \omega_i} = -2(d-f) f(1-f) X$$

La actualización de los pesos se puede hacer empleando

$$\Delta\omega = \omega(k) + \alpha x(k) e(k) f(1-f) \qquad [3.22]$$

Problema del operador lógico XOR por uso del perceptrón

De acuerdo con la lógica booleana existen operadores como AND, OR, XOR y NOT donde sólo el operador XOR y XNOR son linealmente indivisibles. Al graficar en un plano cartesiano se puede ver la operación sobre dos operandos booleanos y sus salidas, donde se aprecia claramente que una línea recta no podría clasificar en dos conjuntos la salida del operador, con lo cual resulta imposible para un perceptrón simple realizar dicho aprendizaje.

En este trabajo se presentan dos opciones para resolver este problema, al generar un perceptrón de dos capas cuyo aprendizaje fue hecho elemento por elemento para llegar a una clasificación tanto funcional como gráfica del comportamiento del XOR.

Por otro lado se muestra una aproximación con sólo un perceptrón que incluye una tercer entrada que es función de las entradas originales, con lo cual se genera un mapeo en tres dimensiones que permite hacer la clasificación correctamente.

El perceptrón multicapa se presenta implementado físicamente en un arreglo de amplificadores operacionales que muestran cómo se lleva a cabo la clasificación correspondiente a partir de los pesos obtenidos del entrenamiento correspondiente.

Un perceptrón se evalúa a partir de la suma de las entradas ponderadas por sus pesos asociados correspondientes. Dicha suma se evalúa después dentro de una función de activación, la cual de manera estándar para un perceptrón resulta de un comparador contra cero, dado que:

$$f(\mu) = \begin{cases} 1 & \mu > 0 \\ 0 & \mu < 0 \end{cases} \qquad [3.23]$$

Desarrollo

En la gráfica de la figura 3.17 se muestra la combinación de entradas de un operador XOR y sus respectivas salidas. En ella se observa que al estar las salidas iguales en polos opuestos sobre el conjunto de posibilidades, resulta imposible trazar una línea recta que divida los conjuntos de manera que el perceptrón simple caracterice esto en forma sencilla.

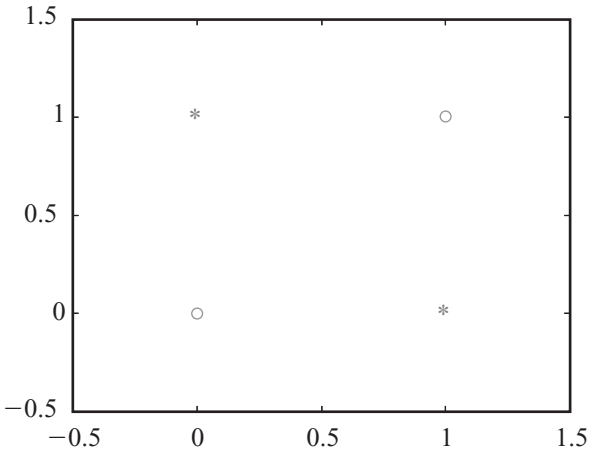


Fig. 3.17 Gráfica del XOR.

Sin embargo, es posible definir dos operadores alternos a través de los cuales normalicemos dos separaciones diferentes cuya conjunción genere a su vez una clasificación pertinente al problema que buscamos resolver.

Recordemos de nuevo las combinaciones necesarias para el operador XOR para la siguiente salida, dadas como:

I1	Io	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Podemos definir un operador intermediario como el operador OR, el cual sabemos que es posible obtener a través de un perceptrón donde su tabla está dada como:

OR

I1	Io	OR
0	0	0
0	1	1
1	0	1
1	1	1

Ahora bien, un segundo operador de interés es el operador AND, el cual sabemos es también clasificable a través de un perceptrón. Su tabla de verdad está dada como:

AND

I1	Io	AND
0	0	0
0	1	0
1	0	0
1	1	1

Si tomamos ambas salidas de AND y OR en una tabla y la comparamos contra las salidas que requerimos para XOR en un inicio, tenemos que:

XOR

OR	AND	XOR
0	0	0
1	0	1
1	0	1
1	1	0

Lo cual se reduce a que un tercer perceptrón asimile la clasificación a través de sólo tres patrones, puesto que los dos intermedios son el mismo ya que gráficamente se puede demostrar que dicha división es admisible.

Introduciendo los patrones para cada perceptrón en el algoritmo desarrollado en MATLAB® descrito a continuación, se obtiene los valores de los pesos correspondientes para cada perceptrón como $[w_0 \quad w_1 \quad w_2]$ donde w0 se refiere al peso del valor de ajuste interno del perceptrón $\{-1\}$. Dicho valor de ajuste permite mayor movilidad en el ajuste de los pesos, lo cual repercute gráficamente en que la función de activación de cada perceptrón pueda moverse a lo largo del eje x y no sólo se escale a lo largo del eje y.

```
function perc=perceptron(w,patron,alfa)
xo=-1;
%xo=0;
i=1;
patron=[xo*ones(size(patron,1),1) patron];


errorg=[1 1 1 1];


while sum(errorg) > 0
        for i=1:size(patron,1)


                salida=evalperc(w,patron(i,[1:(size(patron,2)-1)]));
```

```
            error=patron(i,size(patron,2))-salida;
            errorg(i)=0;
            if error ~= 0
                errorg(i)=1;
                while error ~= 0

                        salida=evalperc(w,patron(i,[1:(size(patron,2)-1)]));
                        error=patron(i,size(patron,2))-salida;
                        w = w + alfa*error*patron(i,[1:(size(patron,2)-1)]);
                end
            end
        % errorg
        end
end perc=w;
```

Dicho algoritmo, a su vez, hace uso de una función particular que realiza la evaluación en cada perceptrón como:

```
function salida=evalperc(w,inputs)

            nets=[inputs]*w';

        if nets >= 0
            salida = 1;
        elseif nets < 0
            salida = 0;
        end
```

A partir de esto se obtuvieron los siguientes pesos para cada perceptrón:

Perceptrón	W_0	W_1	W_2
OR	0.5	1	1
AND	1.5	1	1
Y {XOR}	0.5	1	−1

Gráficamente se puede mostrar cómo dicha clasificación se logra adecuadamente sobre las combinaciones originales del operador XOR, incluyendo las gráficas resultantes de las rectas obtenidas para OR y AND en la gráfica de la figura 3.18, donde se muestra que se clasifica lo que está dentro de la banda formada por dichas rectas como un {1} y todo lo que esté fuera como un {0}. Cabe resaltar que por lo mismo, valores mucho mayores a 1 que se pensaría estarían contenidos en la clasificación que da {1}, por el contrario dan {0} ya que se encuentran fuera de la banda original.

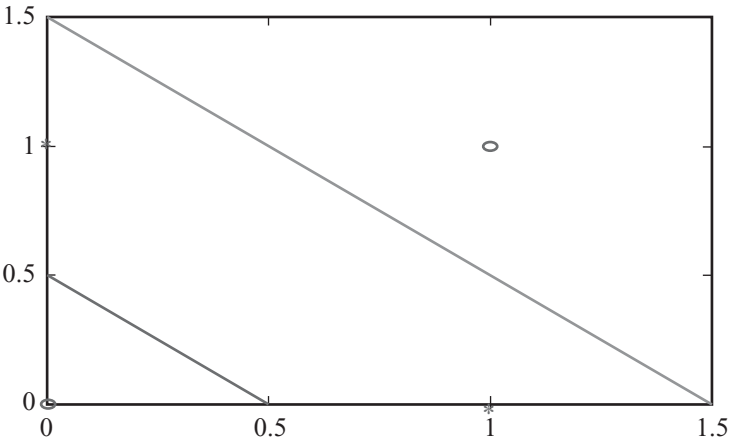


Fig. 3.18 Gráfica del XOR con clasificación.

En la gráfica de la figura 3.19 a su vez encontramos la clasificación que resulta en el tercer perceptrón, el cual a partir de las salidas AND y OR logra una clasificación posible del XOR original.

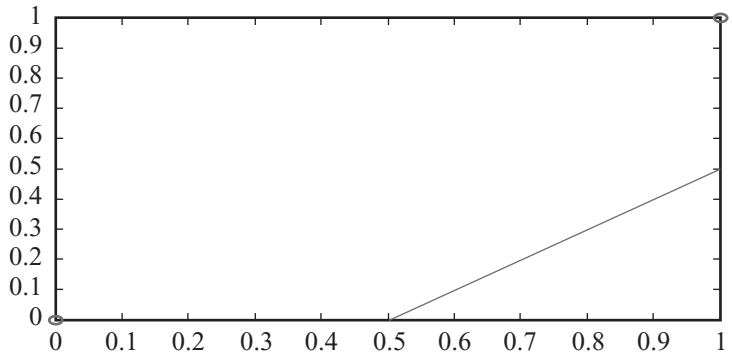


Fig. 3.19 Gráfica del tercer perceptrón.

A diferencia de una red neuronal con aprendizaje por retropropagación del error, cada perceptrón de este esquema fue entrenado de manera individual sin contemplar el entrenamiento de las neuronas adyacentes.

Dicho esquema se puede implementar físicamente a partir de amplificadores operacionales (Op-Amp). Para cada perceptrón se utilizó 3 Op-Amp en cascada. El primero se encargaba de evaluar la suma, el segundo evaluaba la función de activación como un comparador de nivel modelando la función del perceptrón estándar. El tercer Op-Amp se encargaba de acondicionar la salida del comparador a valores de voltaje adecuados entre {0,1} para ser introducidos en el siguiente perceptrón, ya que el entrenamiento se hizo directamente con dichos valores.

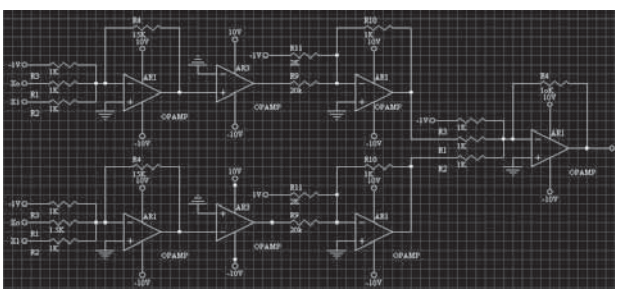


Fig. 3.20 Gráfica del XOR con clasificación.

La figura 3.20 muestra un esquema de la conexión que se evalúa. Este ejemplo sirve para comprobar cómo un mapa de entradas inicialmente no separable por un perceptrón simple puede ser clasificado a partir de la integración de capas posteriores, la cual realiza segmentaciones parciales que en una forma global permiten clasificar correctamente dicho operador inicial.

Por otro lado, se demuestra matemáticamente cómo es posible realizar dicha clasificación a partir de un solo perceptrón al incluir una tercera entrada al perceptrón original. Si agregamos X_3 tal que $x_3 = x_1x_2$, dicha entrada no es una variable nueva pues resulta del producto de una mutua influencia de las entradas originales. Así, la salida del perceptrón queda como:

$$y = f(-w_0 + x_1w_1 + x_2w_2 + x_1x_2w_3)$$

Se puede resolver del problema del XOR. Si entrenamos dicho perceptrón a partir de los patrones:

X1	X2	X1X2	Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

obtenemos pesos como:

$$w = \{1,1,1,-2\}$$

quedando:

$$y = f(-1 + x_1 + x_2 - 2x_1x_2)$$

Se puede apreciar entonces que $x_1 + x_2 - 2x_1x_2 = 1$. A partir de esta distinción en 3D es posible que segmente las entradas originales en dos conjuntos que contienen las salidas {0} y {1} que se buscaban en un inicio.

El algoritmo antes descrito fue probado nuevamente para un nuevo conjunto de patrones alterando el coeficiente de aprendizaje para poder observar su comportamiento.

El programa que se usó fue el siguiente:

```
x=[0 0 0;0 1 1;1 0 1;1 1 1];
w=[-1.5 0.5 1.5];
alfas=[0.001 0.01 0.1 1 10 100];
for i=1:size(alfas)
      ws(i)=perceptron(w,x,alfas(i));
end
```

del cual se tienen los siguientes pesos

Wo	W1	W2
0.75	1	0.75
0.75	1	0.75
0.7	1	0.8
1	1	1.5
1	1	10.5
1	1	100.5

Como se puede apreciar, el factor de aprendizaje permite que los pesos hagan excursiones de mayor extensión a fin de encontrar puntos mínimos de una manera más global, sin embargo se corre el riesgo de que dichas excursiones hagan saltos demasiado grandes en el conjunto de posibilidades, con lo que puede saltarse áreas que contienen el mínimo global de interés. Un esquema de coeficientes dinámicos sería recomendable ya que éste podría ir disminuyendo a medida de que nos aproximemos al mínimo al disminuir una razón del error en el aprendizaje.

Retomando la importancia de la variable de ajuste interna del perceptrón, ésta le da mayor flexibilidad a la función de activación ya que puede moverse a lo largo de los ejes, con lo cual cubre una mayor gama de posibilidades para los pesos y converge en menor tiempo. Se probó en el algoritmo anterior nulificar el efecto de w_o resultando que el algoritmo no logrará converger ya que la naturaleza misma del problema requiere de movimientos en los ejes para lograr asentar una recta que clasifique los patrones de aprendizaje.

Control de un motor de pasos con un grupo de perceptrones

El desarrollo de este ejercicio es a través del entrenamiento de dos neuronas: la primera de ellas será entrenada para decidir la dirección de un sistema que consta de dos sensores de luz y que va a girar hacia la izquierda o hacia la derecha por medio de un motor a pasos, dependiendo qué sensor se active con la incidencia de luz. La segunda neurona será entrenada para decidir a qué velocidad va a girar el sistema antes mencionado de acuerdo con la combinación que presenten dos interruptores.

Más adelante se presentará el código con el cual se puede entrenan a las neuronas que participan en el proceso, así como la implementación con amplificadores operacionales (Op-Amp).

La primera parte es la del entrenamiento y la segunda es la implementación de las dos neuronas que se va a utilizar. El tipo de entrenamiento que se usó para ambas neuronas fue el del perceptrón convencional. Como recordará, el perceptrón es un modelo que nos ayuda a dividir patrones linealmente separables y lo que hace este entrenamiento es deducir cuándo una entrada pertenece a una de las dos clases. Por esta razón decidimos construir tablas de verdad que fueran linealmente separables para cada una de las neuronas, y de acuerdo con dichas tablas realizamos su entrenamiento. Las tablas de verdad requeridas son las siguientes:

Tabla de verdad para neurona de dirección:

V1	V2	Salida
0	0	0
0	1	0
1	0	1
1	1	1

1 → izquierda 0 → derecha

Tabla de verdad para neurona de velocidad:

V1	V2	Salida
0	0	0
0	1	0
1	0	0
1	1	1

1 → rápido 0 → lento

Sabemos que el entrenamiento consiste en encontrar los pesos que están relacionados con cada una de las entradas y con el umbral. Siguiendo el algoritmo para este tipo de entrenamiento se presenta un programa en C++; el código fuente es el que se muestra a continuación:

```
#include <stdio.h>
void main()
{
        float a=0,b=0,d=0,e=1,l,x,y,z,net,sum,c=0,r1,r2,r3,r4,e1=1;
        printf("Dame los valores iniciales\n");
        printf("x=");
        scanf("%f",&x);
        printf("y=");
        scanf("%f",&y);
        printf("z=");
        scanf("%f",&z);
        printf("l=");
        scanf("%f",&l);
        while(e!=0)
        {

                a=0,b=0,d=0;
                printf("%f\n",x);
                sum=(a*x)+(b*y)-(2*z);
                if(sum>2.5)net=5;
                else net=0;
                e1=e;
                e=d-net;
                r1=e;
                x=x+l*e*a;
                y=y+l*e*b;
                z=z+l*e*2;
                a=0,b=5,d=0;
                sum=a*x+b*y-2*z;
                if(sum>2.5)net=5;
                else net=0;
                e=d-net;
                r2=e;
                x=x+l*e*a;
                y=y+l*e*b;
                z=z+l*e*2;
```

```
            a=5,b=0,d=0;
            sum=a*x+b*y-2*z;
            if(sum>2.5)net=5;
            else net=0;
            e=d-net;
            r3=e;
            x=x+l*e*a;
            y=y+l*e*b;
            z=z+l*e*2;
            a=5,b=5,d=5;
            sum=a*x+b*y-2*z;
            if(sum>2.5)net=5;
            else net=0;
            e=d-net;
            r4=e;
            x=x+l*e*a;
            y=y+l*e*b;
            z=z+l*e*2;
            sum=a*x+b*y-z*2;
            if(sum>2.5)net=5;
            else net=0;
            c++;

        }
        printf("x=%f\n",x);
        printf("y=%f\n",y);
        printf("z=%f\n",z);
        printf("e=%f\n",e);
        printf("c=%f\n",c);
```

En cada uno de los casos se tiene un umbral $\theta = 2$.

La inicialización de los pesos de la neurona de dirección fue:

$\omega_1 = 1$
$\omega_2 = 1$
$\omega_\theta = 1$

Los pesos que resultan son:

$\omega_1 = 3$
$\omega_2 = 0.5$
$\omega_\theta = 1.2$

La inicialización de los pesos de la neurona de velocidad fue:

$\omega_1 = 1$
$\omega_2 = 1$
$\omega_\theta = 1$

Los pesos son:

$\omega_1 = 0.55$
$\omega_2 = 0.45$
$\omega_\theta = 1.14$

Para poder implementar las neuronas por medio de Op-Amp se realizó el siguiente análisis: sabemos que para un perceptrón el cálculo de la salida aplicado a nuestro caso se hace por medio de la siguiente ecuación:

$$y(t) = f\left[\omega_1 X_1 + \omega_2 X_2 - \omega_\theta X_\theta\right] \qquad [3.24]$$

Para cada una de las combinaciones de entrada tenemos un resultado que es el que se evalúa en la función. Cabe mencionar que el 0 equivale a 0 V y el 1 a 5V.

Para el caso de la neurona de dirección:

$0\ 0 \rightarrow 0$
$y = -1.2(2) = -2.4$

$0\ 1 \rightarrow 0$
$y = 0.5(5) - 1.2(2) = 0.1$

$1\ 0 \rightarrow 1$
$y = 3(5) - 1.2(2) = 12.6$

$1\ 1 \rightarrow 1$
$y = 3(5) + 0.5(5) - 1.2(2) = 15.1$

Para el caso de la neurona de velocidad:

$0\ 0 \rightarrow 0$
$y = -1.14(2) = -2.2$

$0\ 1 \rightarrow 0$
$y = 0.45(5) - 1.14(2) = 0.03$

$1\ 0 \rightarrow 1$
$y = 0.55(5) - 1.14(2) = 0.47$

$1\ 1 \rightarrow 1$
$y = 0.55(5) + 0.45(5) - 1.14(2) = 2.72$

De acuerdo con la ecuación (3.24), se hace la sumatoria de todos los pesos multiplicados por sus respectivas entradas, esto por medio de Op-Amp´s se logra con un sumador inversor, que aplicado a nuestro caso tiene la siguiente configuración (figura 3.21):

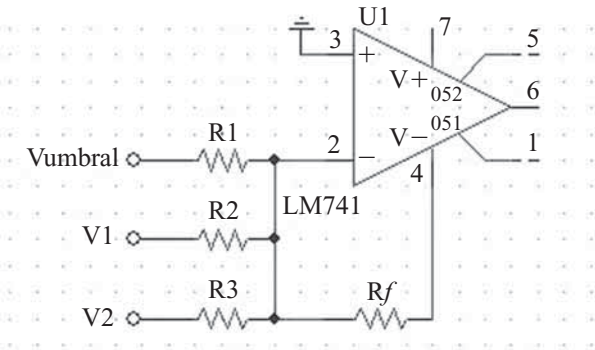


Fig. 3.21 Configuración mediante Op-Amp.

$$V_0 = -V_{in}\left(\frac{R_f}{R_1} + \frac{R_f}{R_2} + \frac{R_f}{R_3}\right) \qquad [3.25]$$

Para la neurona de dirección, el caso de la entrada del V_θ (Vumbral) se resuelve por medio de otro amplificador que logre darnos la salida que deseamos, que en este caso es de -2.4. El Op-Amp que se usó fue un inversor, gracias a que el valor de umbral se resta en la ecuación (3.25). Su configuración es la que se muestra a continuación (figura 3.22):

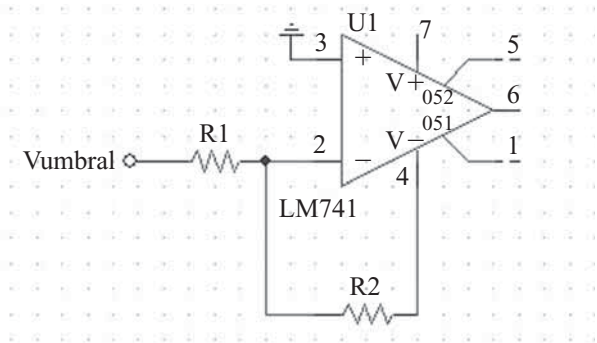


Fig. 3.22 Configuración con inversor Op-Amp.

$$V_o = -\frac{R_2}{R_1}V_{in} \qquad [3.26]$$

Se fija el valor de la resistencia R_2 y obtenemos el valor de la resistencia R_1. El Vin es de 2V, que fue el valor que le damos al umbral, y el de salida es el que tenemos al multiplicarlo por su peso ($\omega_\theta = 1.2$).

$V_{in} = 2V$
$V_{out} = -2.4V$
$R_1 = 10kW$

$$R_1 = -\frac{R_2}{V_o}V_{in} = -\frac{10k}{(-2.4)}(2) = 8.3k\Omega$$

$R_2 = 8.3kW$

Los valores de las resistencias R_2 y R_3 se obtienen igual que en el caso del valor del umbral. La resistencia R_f se fija en 10KΩ.

Para el caso de V_1.

$R_f = 10k\Omega$

$V_{in} = 5V$ (cuando entra un 1 es 5V)

$V_{out} = 15V$ ($\omega_\theta = 3$)

$$R_2 = \frac{R_f}{V_o} V_{in} = \frac{10k}{(15)}(5) = 3.3k\Omega$$

$R_2 = 3.3k\Omega$

Para el caso de V_2.

$R_f = 10k\Omega$

$V_{in} = 5V$ (cuando entra un 1 es 5V)

$V_{out} = 2.5V$ ($\omega_\theta = 0.5$)

$$R_2 = \frac{R_f}{V_o} V_{in} = \frac{10k}{(2.5)}(5) = 20k\Omega$$

$R_2 = 20k\Omega$

Para la neurona de velocidad, el caso de la entrada del Vq (Vumbral) se resuelve también por medio de otro amplificador que logre darnos la salida que deseamos, que en este caso es de −2.28. El Op-Amp que se utilizó fue un inversor, gracias a que el valor de umbral se resta en la ecuación (3.25). Y su configuración es la misma que usó para el umbral de la neurona de dirección.

Fijamos el valor de la resistencia R_2 y obtenemos el valor de la resistencia R_1. El Vin es de 2V, que fue el valor que le damos al umbral, y el de salida es el que tenemos al multiplicarlo por su peso (ω_θ = 1.14).

$V_{in} = 2V$

$V_{out} = -2.28V$

$R_1 = 10k\Omega$

$$R_1 = -\frac{R_2}{V_o} V_{in} = -\frac{10k}{(-2.28)}(2) = 8.7k\Omega$$

$R_2 = 8.7k\Omega$

Los valores de las resistencias R_2 y R_3 se obtienen igual que en el caso del valor del umbral. La resistencia R_f se fija en 10KΩ.

Para el caso de V_1.

$R_f = 10k\Omega$

$V_{in} = 5V$ (cuando entra un 1 es 5V)

$V_{out} = 2.75V$ ($\omega_\theta = 0.55$)

$$R_2 = \frac{R_f}{V_o} V_{in} = \frac{10K}{(2.75)}(5) = 18.181K\Omega$$

$R_2 = 18.181k\Omega$

Para el caso de V_2.

$R_f = 10k\Omega$

$V_{in} = 5V$ (cuando entra un 1 es 5V)

$V_{out} = 2.25V$ ($\omega_\theta = 0.45$)

$$R_2 = \frac{R_f}{V_o} V_{in} = \frac{10K}{(2.25)}(5) = 22.2K\Omega$$

$R_2 = 22.2k\Omega$

En ambos casos (tanto para la neurona de dirección como para la neurona de velocidad) la salida del sumador es de signo contrario al que en realidad debe de tener, debido a la configuración inversora que se utilizó. Por ello se usó un inversor de ganancia unitario para cambiar el valor del voltaje de salida que tiene el sumador. La configuración de dicho inversor es la siguiente (figura 3.23):

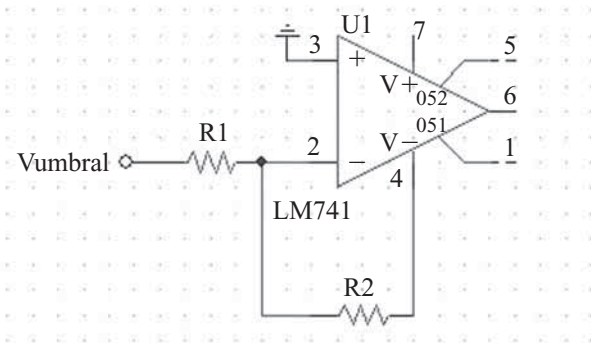


Fig. 3.23 Configuración con inversor de ganancia unitario.

$$V_o = -\frac{R_2}{R_1} V_{in}$$

Donde:

V_o es el que se obtiene de la salida del sumador, el valor depende de lo que se encuentre en V_1 y en V_2.

V_{out} es el mismo valor que V_o pero con signo contrario.

$R_1 = 10k\Omega$

$R_2 = 10k\Omega$

Como se muestra en la ecuación (1), la salida se tiene que evaluar en una función; en nuestro caso esta evolución se hace por medio de un Op-Amp que tiene configuración de comparador. El diseño de un comparador es siguiente (figura 3.24):

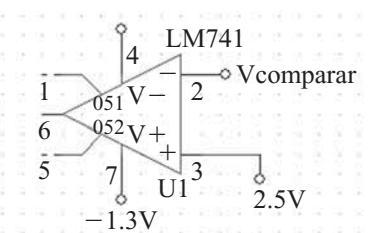


Fig. 3.24 Configuración de comparador.

El valor que sale del último inversor se compara con 2.5V, si el valor es mayor se va a 1 (5V) y si el valor es menor a 2.5V se va a 0 (0V).

El diseño completo de cada una de las neuronas es el que se muestra a continuación (figura 3.25):

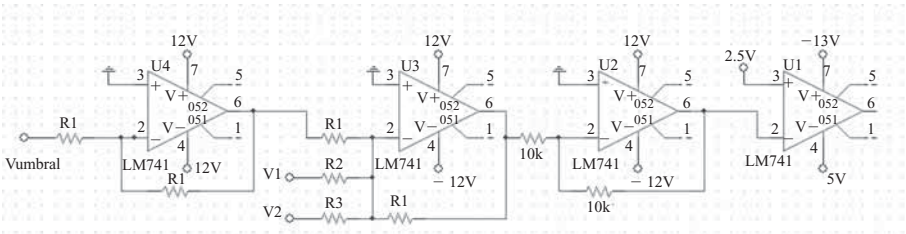


Fig. 3.25 Diseño completo de cada una de las neuronas, Op-Amp LM741.

Una vez que las neuronas se encuentran entrenadas y listas con los amplificadores operacionales, se desarrolla la parte del controlador del motor a pasos.

El modelo del driver del motor a pasos es el MC3479 y tiene la siguiente configuración (figura 3.26):

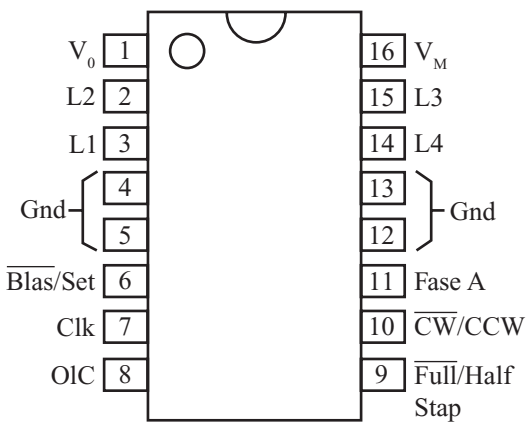


Fig. 3. 26 Configuración del MC3479.

A este controlador se le conectan las dos neuronas previamente diseñadas, de velocidad y dirección, el motor a pasos y el reloj, el cual nos facilita conocer la frecuencia de los pasos del motor. El motor se alimenta con 18V. La configuración es la siguiente (figura 3.27):

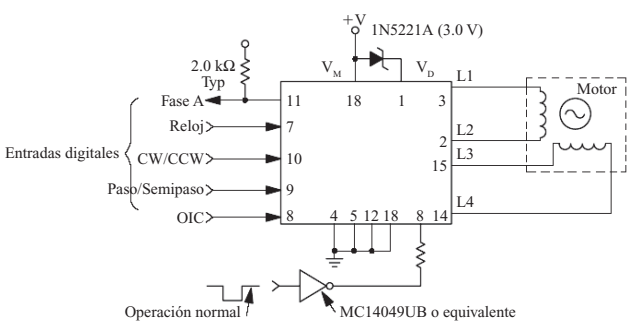


Fig. 3.27 Configuración del controlador.

La neurona de dirección funciona por medio de sensores de luz y la neurona de velocidad mediante dos interruptores. Después de armada cada una de las partes, se procede a la integración total que permite evaluar en diferentes condiciones el desempeño del clasificador lineal (perceptrón).

Teorema de Kolmogorov

Kolmogorov demostró que una función continua de varias variables puede ser representada por la superposición de funciones continuas unidimensionales de las variables de entrada originales. Cualquier función continua mapeada en una entrada de n dimensiones, $n \geq 2$, a una salida de m dimensiones puede ser implementada por una red con una capa oculta. Para $\phi : \bar{I}^n \rightarrow \bar{R}^m$, $\phi(\bar{x}) = \bar{y}$, donde $\bar{I}$ es el intervalo de [0,1] y por lo tanto $\bar{I}^n$ es un hipercubo, la función ϕ se implementa con n entradas, $2n + 1$ elementos en la capa media oculta y m elementos en la capa de salida. Esto es:

$$z_k = \sum_{j=1}^{n} \lambda^k \psi(x_j + \varepsilon k) + k \qquad [3.27]$$

$$y_i = \sum_{k=1}^{2n+1} g_i(z_k) \qquad [3.28]$$

Donde:

λ : constante real

ψ : función continua real y monotónica independiente de ϕ pero dependiente de n

ε : número racional $0 < \varepsilon \leq \delta$

δ : constante positiva arbitraria

Desafortunadamente esta prueba no indica el proceso de selección de pesos para implementar un mapeo en particular, y tampoco especifica las funciones g_i que tienen formas desconocidas y normalmente no lineales. Por lo que el empleo de topología de redes multicapas es inminente para el aumento de la potencialidad de las RNA.

REDES MULTICAPA

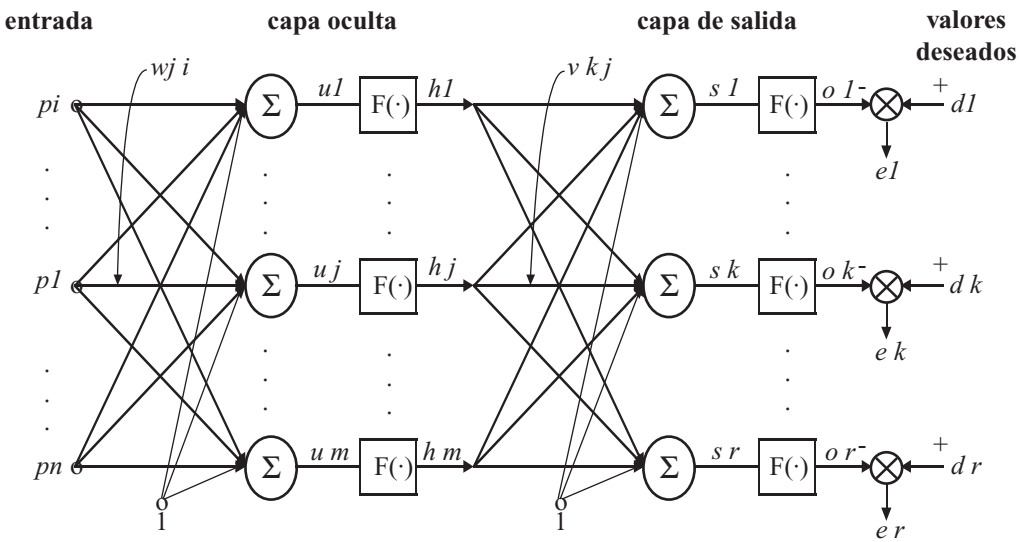


Fig. 3.28 Red neuronal de dos capas.

Perceptrón multicapa

Es uno de los primeros esfuerzos por representar el aprendizaje supervisado, en donde se emplean funciones de activación elementales de forma binaria en la mayoría de las ocasiones. Este elemento es un clasificador lineal con dos entradas y un punto de ajuste, siendo la salida el clasificador.

Redes de retropropagación (backpropagation)

La retropropagación del error es un entrenamiento supervisado que se emplea para redes multicapa, donde se ajusta el valor de los pesos en función del error generado. Esta técnica es muy empleada ya que permite tener un método de optimización que se encuentra al definir el gradiente del error y minimizarlo con respecto a los parámetros de la red neural.

Principios para entrenar una red multicapa empleando el algoritmo de retropropagación

Si se considera la red de tres capas con dos entradas y una salida de la figura 3.29, es posible apreciar que cada neurona está compuesta de dos unidades, donde la primera suma los productos de las entradas por sus respectivos pesos, y la segunda unidad contiene la función de activación. La señal corresponde a la salida de la suma y $y = f(e)$ es la señal de salida del elemento no lineal de la función de activación, así como la salida de la neurona.

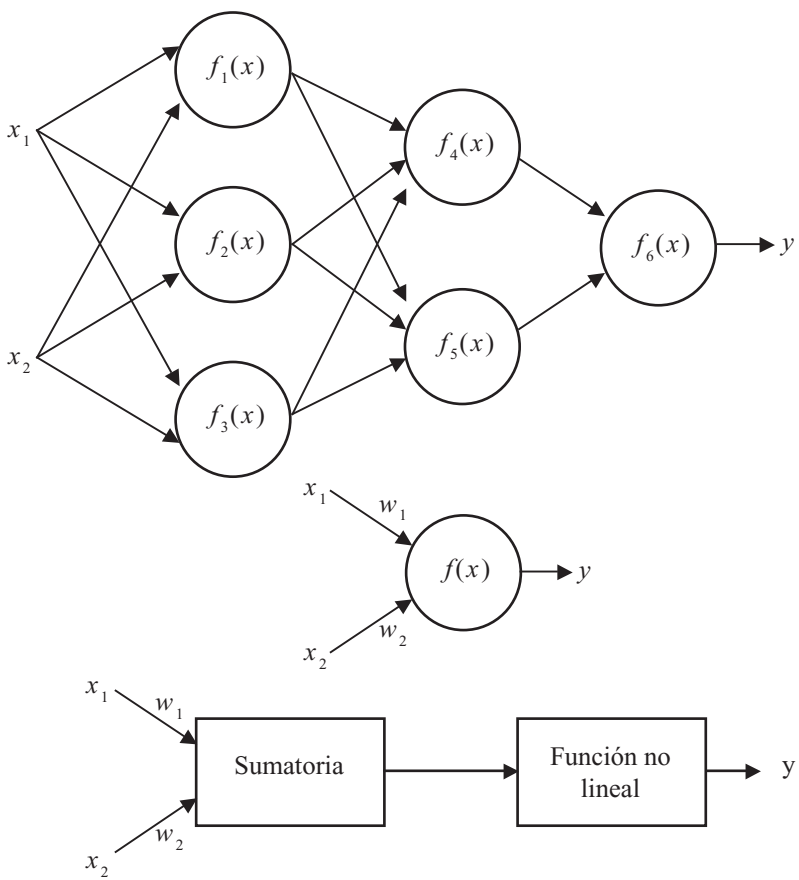


Fig. 3.29 Red de dos capas.

Para enseñarle a la red neuronal es necesario entrenar un conjunto de datos, el cual consta de señales de entradas x_1 y x_2 asignadas con objetivos correspondientes (salidas deseadas) denominados *z*. El entrenamiento es un proceso iterativo. En cada iteración los pesos de los nodos se modifican usando nuevos datos del conjunto para el entrenamiento. Las modificaciones de los pesos se calculan empleando el siguiente algoritmo que se explica a continuación.

Cada paso del entrenamiento comienza al forzar ambas entradas de salida del conjunto de entrenamiento. Después es posible determinar los valores de salida de las señales de cada neurona en cada capa de la red.

La figura 3.30 muestra dos ejemplos de cómo se propaga la señal a través de la red, donde los pesos ω_{mn} corresponden a la conexión de la salida de la neurona *m* con la entrada de la neurona *m* en la capa siguiente.

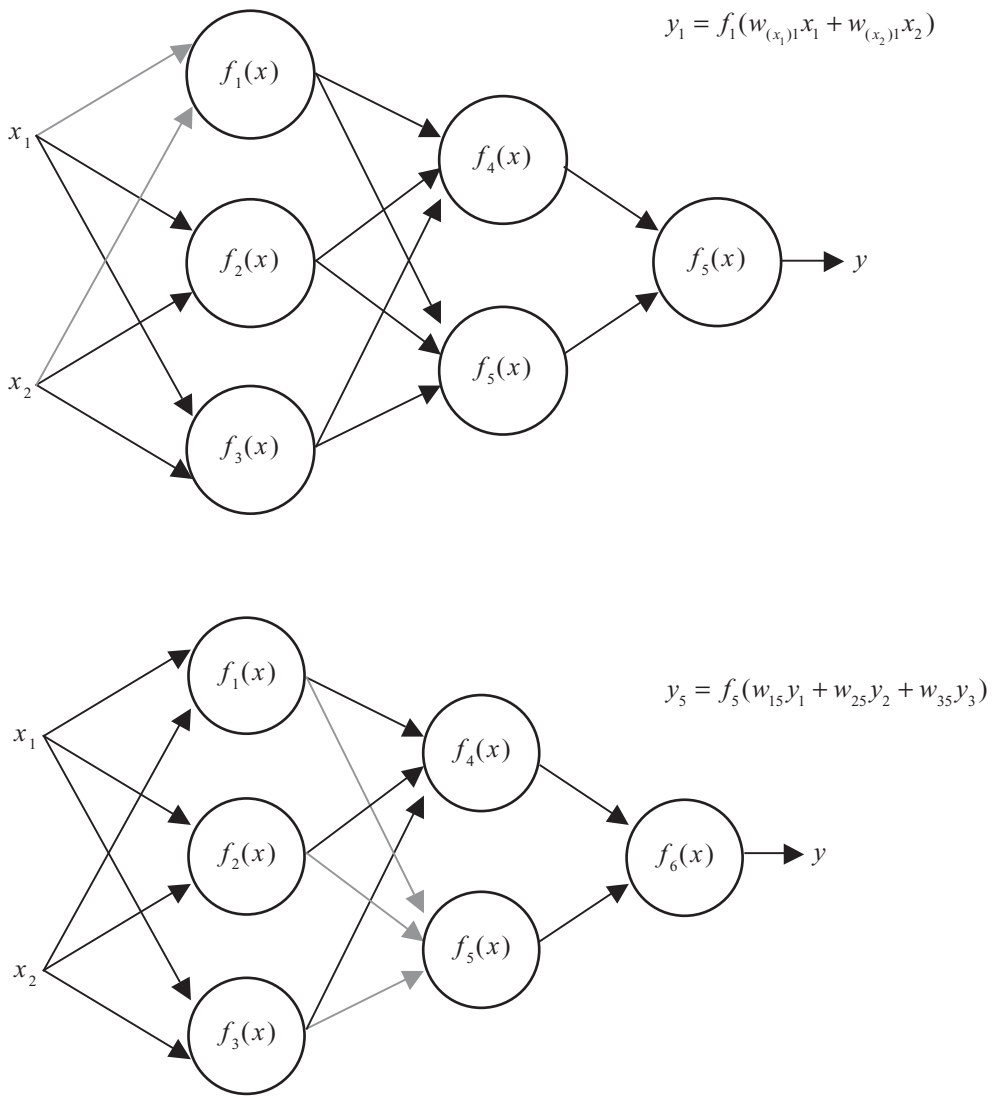


Fig. 3.30 Propagación de las señales en las neuronas.

En el siguiente paso del algoritmo, la salida de la red es comparada con el valor objetivo deseado. La diferencia se denomina error de la señal (δ). Es imposible conocer el error en las neuronas de las capas internas directamente, debido a que se desconocen los valores de salida de estas neuronas. El algoritmo de retropropagación propaga el error de regreso a todas las neuronas, cuya salida fue la entrada de la última neurona, como se muestra en la figura 3.31.

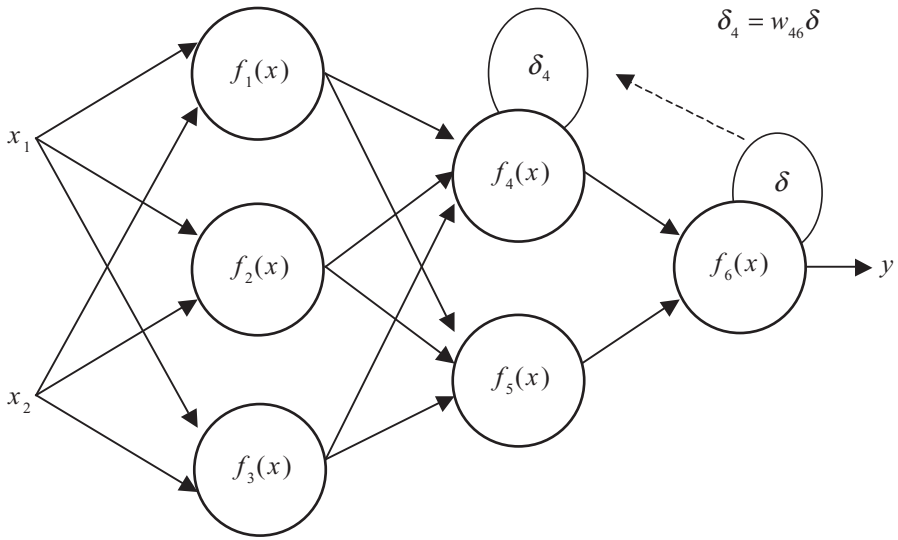


Fig. 3.31 Retropropagación del error.

Posteriormente el error se va propagando a las neuronas de capas anteriores, considerando los pesos de las conexiones, según se aprecia en la figura 3.32.

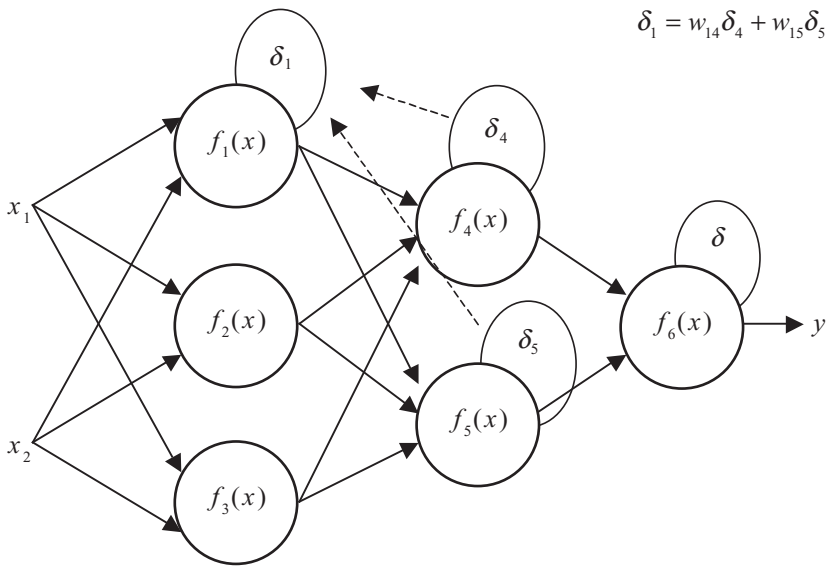


Fig. 3.32 Retropropacación del error.

Cuando se calcula el error para cada neurona, los pesos de entrada pueden modificarse según los ejemplos de la figura 3.33. Los coeficientes η afectan la velocidad de aprendizaje y pueden seleccionarse por distintos métodos. Uno de ellos implica que al inicio del proceso de entrenamiento se elige un valor grande, el cual va descendiendo gradualmente conforme avanza el proceso. Otro método comienza con parámetros pequeños que aumentan a medida que el proceso avanza y nuevamente disminuye en la etapa final. Comenzar el proceso con un parámetro pequeño permite el establecimiento de los signos de los pesos.

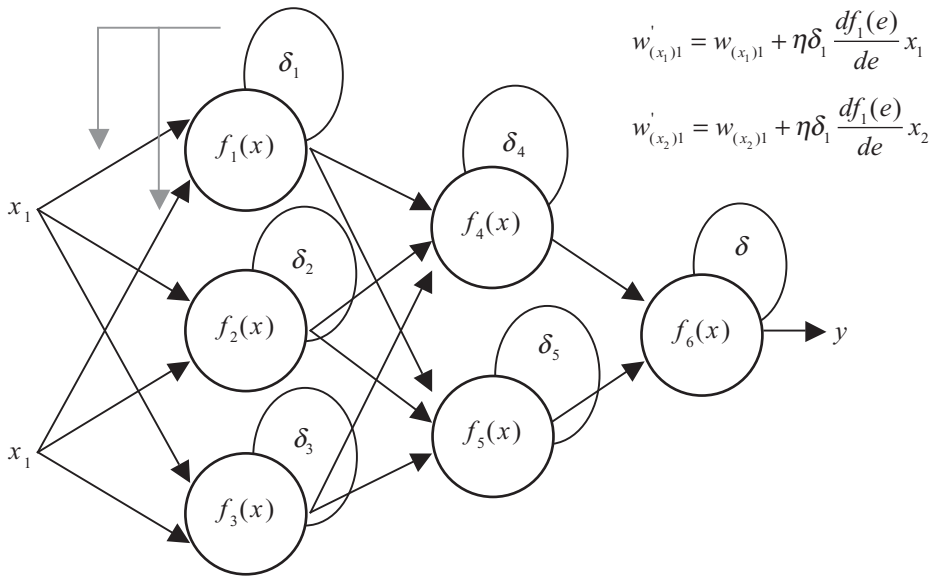


Fig. 3.33 Actualización de los pesos.

Redes neurales – Retropropagación del error

Si tenemos un sistema de redes con la siguiente forma (todas con función sigmoidal) (figura 3.34)

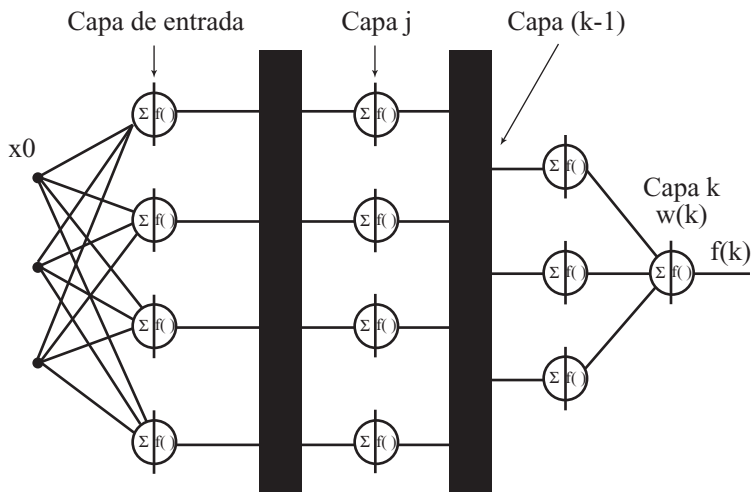


Fig. 3.34 Sistema de redes con función sigmoidal.

Donde

$$S_i^{(j)} = x^{(j-1)} \bullet \omega_i^{(j)}$$

donde

j = capa (j)

i = elemento (i)

Para el método de retropropagación

$$\frac{dE}{d\omega_i^{(j)}} = \left[\frac{dE}{d\omega_i^{(j)}} \cdots \frac{dE}{dm_{j-1}^{(j)}} \right]$$

donde $E = (d - f)^2$

por lo tanto

$$\frac{dE}{d\omega_i^{(j)}} = \frac{dE}{dS_i^{(j)}} \frac{dS_i^{(j)}}{d\omega_i^{(j)}}$$

Si

$$S_i^{(j)} = x^{(j-1)} \bullet \omega_i^{(j)}; \frac{dE}{d\omega_i^{(j)}} = x^{(j-1)}$$

sustituyendo

$$\frac{dE}{d\omega_i^{(j)}} = \frac{dE}{dS_i^{(j)}} x^{(j-1)}$$

teniendo en cuenta que

$$\frac{dE}{dS_i^{(j)}} = \frac{d(d-f)^2}{dS_i^{(j)}} = -\frac{2(d-f)df}{dS_i^{(j)}}$$

$$\frac{dE}{d\omega_i^{(j)}} = -\frac{2(d-f)df}{dS_i^{(j)}} x^{(j-1)}$$

si δ es la sensibilidad del error $\rightarrow$

$$\delta = \frac{(d-f)df}{dS_i^{(j)}}$$

$$\frac{dE}{d\omega_i^{(j)}} = -2\delta_i^{(j)} x^{(j-1)}$$

El gradiente es con sentido negativo $\therefore$

$$\omega_i^{(j)} \leftarrow \omega_i^{(j)} + c_i^{(j)} \delta_i^{(j)} x^{(j-1)}$$

C = factor de aprendizaje

—Cálculo de los pesos de última capa

$$\delta_i^{(j)} = (d-f)\frac{df}{dS_i^{(j)}}$$

Para la capa de salida:

$$\frac{df}{dS^k}$$

Para f como sigmoide y su entrada

$$\frac{df}{dS^k} = f(1-f)$$

$$S^k = (d-f)f(1-f)$$

$$\omega^k \leftarrow \omega^k + c^k(d-f)f(1-f)x^{(j-1)}$$

Capas intermedias

$$\delta_i^{(j)} = (d-f)\left[\frac{df}{dS_1^{(j+1)}}\frac{dS_1^{(j+1)}}{dS_i^{(j)}} + \cdots + \frac{df}{dS_{m(j+1)}^{(j+1)}}\frac{dS_{m(j+1)}^{(j+1)}}{dS_i^{(j)}}\right]$$

$$= \sum_{l=1}^{m(j+1)} (d-f)\frac{df}{dS_l^{(j+1)}}\frac{dS_l^{(j+1)}}{dS_i^{(j)}}$$

Nota: (d-f) $\rightarrow \frac{df}{dS_1^{(j+1)}} \rightarrow \delta$

$$= \sum_{l=1}^{m(j+1)} \delta_l^{(j+1)}\frac{dS_l^{(j+1)}}{dS_i^{(j)}}$$

Empleando derivadas parciales $\frac{dS_l^{(j+1)}}{dS_i^{(j)}}$ $\therefore$

$$S_l^{(j+1)} = x^{(j+1)} \bullet \omega_l^{(j+1)}$$

$$= \sum_{v=1}^{m(j+1)} f_v^{(j)}\omega_{vl}^{(j+1)} \rightarrow \text{Pesos no dependen de } S$$

$$\frac{dS_i^{(j+1)}}{dS_i^{(j)}} = \frac{\sum_{v=1}^{m(j+1)} f_v^{(j)}\omega_{vl}^{(j+1)}}{dS_i^{(j)}}$$

$$= \sum_{v=1}^{m(j+1)} \frac{df_v^{(j)}}{dS_i^{(j)}}\omega_{vl}^{(j+1)}$$

Nota: la ecuación anterior es igual a cero excepto cuando v = i

$$\frac{df_v^{(j)}}{dS_i^{(j)}} = f_v^{(j)}\left(1-f_v^{(j)}\right)$$

$$\frac{dS_i^{(j+1)}}{dS_i^{(j)}} = \omega_{il}^{(j+1)} f_i^{(j)}\left(1-f_i^{(j)}\right)$$

Donde $\delta_i^{(j)} = f_i^{(j)}\left(1-f_i^{(j)}\right)\sum_{l=1}^{m(j+1)} \delta_l^{(j+1)}\omega_{il}^{(j+1)}$ NO DEPENDE DEL ERROR

Para la capa de salida

$\delta^{(k)} = (d - f) f (1 - f)$

La regla genérica

$\omega_i^{(j)} \leftarrow \omega_i^{(j)} + c_i^{(j)} \delta_i^{(j)} x^{(j-1)}$

Ejemplo:

Para c = 1

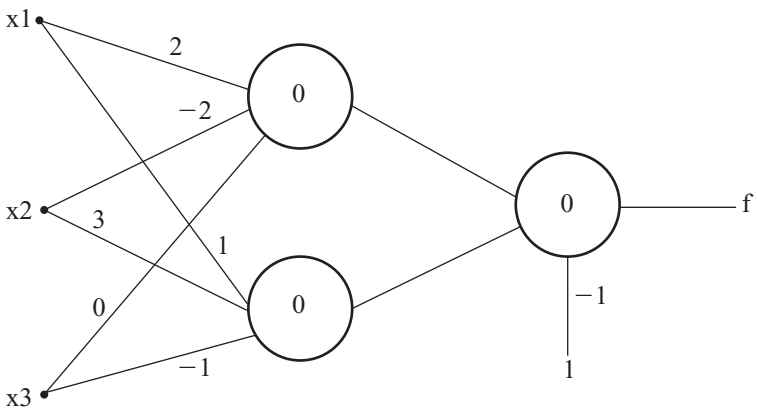


Fig. 3.35 Entrenamiento de retropropagación del error.

Para el primer vector de entrada (1,0,1) se tiene

$f_1^{(1)} = 0.881$

$f_2^{(1)} = 0.5$ Primera capa

Capa de salida f = 0.665 como salida final. Propagando hacia atrás:

$\delta^{(2)} = -0.148$

$\delta_1^{(1)} = -0.047$

$\delta_2^{(1)} = 0.074$

$\omega_1^{(1)} = (1.963; -2; -0.047)$

$\omega_2^{(1)} = (1.074; 3; -0.926)$

$\omega^{(2)} = (2.870; -2.074; -1.148)$

Algoritmo de retropropagación con momento (Backpropagation with Momentum)

Este algoritmo puede interpretarse como un gradiente descendente con factor de suavidad, expresado de la forma siguiente:

$$\Delta\omega(t) = -\eta \frac{\partial E}{\partial \omega}(t) + \alpha \Delta\omega(t-1), 0 \le \alpha < 1$$

Se denomina la regla delta generalizada.

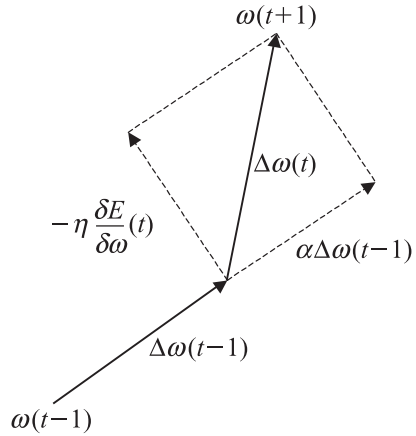


Fig. 3.36 Forma gráfica de la ecuación del algoritmo de retropropagación con momento.

El momento tiene los siguientes efectos:

1. Suaviza los cambios en los pesos filtrando las variaciones de alta frecuencia.
2. Cuando una secuencia larga de pesos cambia en la misma dirección, el momento tiende a amplificar la tasa de aprendizaje efectivo en un factor de $\eta' = \eta/(1 - \alpha)$, lo cual conduce a una convergencia más rápida.
3. El momento puede ayudar al sistema en ocasiones para que escape de mínimos locales proporcionándole al estado del vector la inercia suficiente en la superficie del error.
4. Efectos en la trayectoria (figura 3.37).

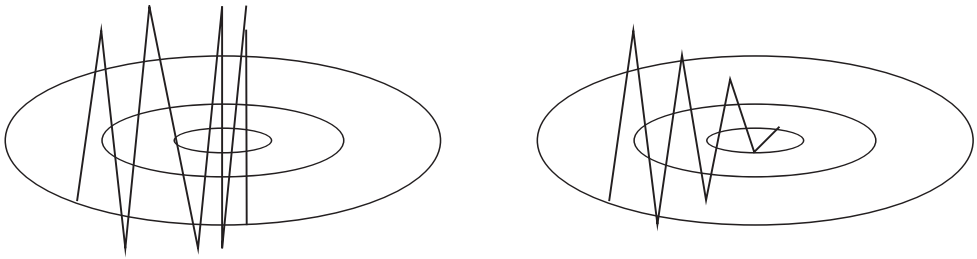


Fig. 3.37 Trayectorias obtenidas con y sin momento en la retropropagación.

DISEÑO DE FILTROS FIR CON REDES NEURALES ARTIFICIALES

Filtro

Es un dispositivo (hardware o software) que se aplica a un conjunto de datos para poder extraer información sobre una variable de interés. En el área de las señales, el filtrado es un proceso mediante el cual se modifica el contenido espectral de una señal.

Cuando hablamos de filtrado nos referimos a un *proceso que puede ser lineal*, diseñado para alterar el contenido espectral de una señal de entrada (o una secuencia de datos) de un modo específico. El filtrado se realiza por filtros cuya magnitud y/o fase satisfacen ciertas especificaciones en el dominio de la frecuencia.

El término filtrado adaptativo implica que los parámetros que caracterizan al filtro, tales como ancho de banda, frecuencias de los ceros, cambian con el tiempo, esto es, los coeficientes, también llamados pesos, de los filtros adaptativos cambian con el tiempo, en contraposición a los coeficientes de los filtros fijos que son invariantes con el tiempo.

Filtros adaptativos digitales

Son aquellos en los que la entrada, la salida y los pesos del filtro están cuantificados y codificados en forma binaria. El tener los coeficientes del filtro no fijos sino variables es necesario cuando no se conocen a priori las características estadísticas de la señal que se va a filtrar, o cuando se conocen y se sabe que son cambiantes con el tiempo, y así, es en esos casos donde se precisa de un filtrado adaptativo.

La ecuación de entrada-salida de un filtro adaptativo digital es:

$$y(n) = \Sigma_{i=0}^{N} a_i(n)x(n-i) - \Sigma_{j=1}^{M} b_j(n)y(n-j) \qquad [3.29]$$

Para $n \geq 0$ (señales y filtros causales)

Donde x(n) e y(n) son las muestras de entrada y salida, respectivamente, en el instante n, ai(n) y bj(n) son los pesos del filtro i-ésimo y j-ésimo en el instante n, y N+M+1 es el número total de coeficientes del filtro. Si en lugar de usar $a_i(n)$ y $b_j(n)$ se utilizan a_i y b_j, los coeficientes ya no serían variantes con el tiempo, y nos encontraríamos ante un filtro fijo en lugar de ante un filtro adaptativo.

Si $b_j(n) = 0$ para $1 \leq j \leq M$, resulta un filtro adaptativo FIR, esto es, de respuesta impulso finita.

$$y(n) = \Sigma_{i=0}^{N} ai(n)x(n-i) \; n \geq 0$$

La salida es una combinación lineal de los valores presentes y pasados de la señal de entrada.

Se trata de un filtro no recursivo. Tiene memoria finita. Véase la figura 3.38.

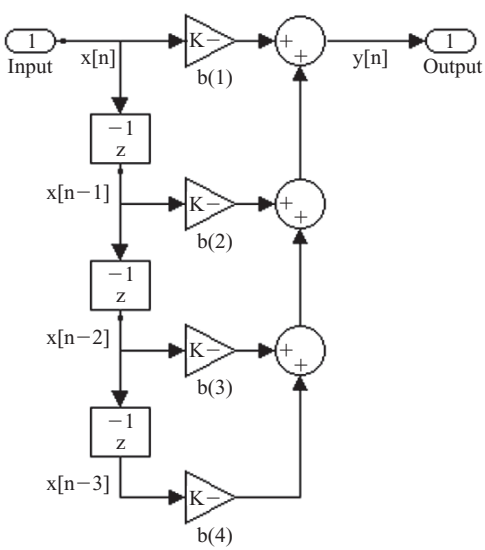


Fig. 3.38 Diagrama de bloques FIR de Simulink®.

El diseño de filtros se complica cuando no se conoce por completo la información. Una opción interesante es el uso de un filtro adaptativo, que es un dispositivo que se autodiseña a través de un algoritmo recursivo.

El filtro tiene condiciones iniciales y después de algunas iteraciones converge a la solución óptima.
En un proceso de filtrado: se obtiene un dato de salida en respuesta a datos de la entrada.
En un proceso adaptativo: se ajustan los coeficientes del filtro de acuerdo con un algoritmo.
Estos dos procesos trabajan de manera interactiva.

Emulación del filtro empleando una red neuronal programada en MATLAB®

P: son las entradas

Pi: son los atrasos en las entradas iniciales

T: son las salidas deseadas

newlind: diseña una red empleando P, Pi, T, usando funciones purelin en las neuronas

sim: simula la red para las entradas ya con los pesos calculados

Ejemplo:

Tomando los datos de una entrada senoidal con tres atrasos, desde 0 a 2Pi tenemos entradas de datos:

```
P={0  0.5000  1.0000  1.5000  2.0000  2.5000  3.0000  3.5000  4.0000  4.5000  5.0000  5.5000  6};
```

Para utilizar tres atrasos usamos:

```
Pi = {1 2 3};
```

Las salidas de una onda senoidal para P tenemos:

```
T={0  0.4794  0.8415  0.9975  0.9093  0.5985  0.1411  −0.3508  −0.7568  −0.9775
                                −0.9589  −0.7055  −0.2794};
```

aplicando la red:

```
net = newlind(P,T,Pi);
```

Para simular la señal de salida de la red tenemos:

```
Y = sim(net,P,Pi)
```

EJEMPLO RECONOCIMIENTO DE LETRAS EMPLEANDO ENTRENAMIENTO DE RETROPROPAGACIÓN DEL ERROR

En este ejemplo se entrena una red neuronal para reconocer las letras del alfabeto. Las primeras 16 letras (de la A a la P) están definidas como una plantilla de 7×5 y cada una se guarda como un vector de longitud 35. Los archivos de datos son una matriz de entrada x (tamaño = 35×16) y una matriz objetivo (tamaño = 4×16). Las entradas en una columna de la matriz t son el código binario de la letra del alfabeto en la columna correspondiente de x. La columna en la matriz son los índices de las casillas rellenas de la plantilla de 7×5.

La red neuronal se entrena para identificar una letra. La red tiene 35 entradas correspondientes a las casillas de la plantilla y tiene 4 salidas correspondientes al código binario de la letra. Se utiliza una tasa

de aprendizaje de 0.5, un entrenamiento máximo de 5000 ciclos y un error mínimo cuadrado de 0.05. Una vez entrenada la red se procede a realizar las pruebas.

Resultados

La primera prueba que se hizo fue con una versión modificada de la letra J. Escogimos esta letra debido a que es muy diferente al resto de las letras. La figura 3.39 muestra el esquema de la letra que dibujamos, la cual es similar a la letra J, pero cuenta con tres modificaciones.

Fig. 3.39 Letra J modificada.

El código para la letra J es 1 0 0 1. Cuando se proporciona a la red las 35 entradas para la plantilla de la letra J original, el código que la red arroja es 0.97 0.02 0.03 0.96. Cuando introdujimos el código de la letra J modificada que se muestra en la figura 1, el código que nos dio la red neuronal fue 0.61 0.00 0.08 0.99.

La segunda prueba realizada fue con dos letras que se parecen. Escogimos la letra B y la letra E, las cuales difieren en la última columna de la plantilla. Hicimos una combinación de las dos letras, de tal forma que la parte superior se pareciera a la letra B y la de abajo a la letra E como se muestra en la figura 3.40.

Fig. 3.40 Combinación de las letras B y E.

El código binario de la letra B es 0 0 0 1 y el de la letra E es 0 1 0 0. Al introducir el código de la letra combinada, la red neuronal entregó como resultado 0.21 0.33 0.73 0.96. Si consideramos que los números menores a 0.5 se toman como 0 binario y los mayores o iguales como 1, entonces el código obtenido se puede leer como una letra D.

Por último, realizamos una prueba con una plantilla que no se asemejara a ninguna letra del alfabeto. Dicha plantilla se generó de manera aleatoria por MATLAB®. La imagen resultante que se insertó en la red neuronal aparece en la figura 3.41.

Fig. 3.41 Letra aleatoria.

El código que obtuvimos con esta letra aleatoria fue 0.99 0.12 0.00 0.92, el cual puede interpretarse como una letra J.

Por lo anterior, podría decirse que las redes neurales funcionan de manera correcta en tareas de clasificación y reconocimiento.

REDES AUTOORGANIZABLES

Las redes de aprendizaje no supervisado no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La red no recibe ninguna información por parte del entorno y se dice que son capaces de autoorganizarse.

Los algoritmos de aprendizaje no supervisado se clasifican en dos tipos:

- Aprendizaje asociativo, donde se mide la familiaridad o se extraen características de los datos de entrada.
- Aprendizaje competitivo, orientado hacia la clasificación de los datos de entrada.

Aprendizaje asociativo

De acuerdo con el estudio psicológico del aprendizaje asociativo, Donald Hebb propuso la Regla de Hebb: "Cuando un axón de una celda A está lo suficientemente cerca de otra celda B como para excitarla y repetidamente ocasiona su activación, un cambio metabólico se presenta en una o ambas celdas, tal que la eficiencia de A como celda excitadora de B, se incrementa".

Para Hebb, una celda es una red neuronal con conexiones de estructura compleja y la eficiencia es el peso de la conexión, mientras que la actividad coincidente es crucial para fortalecer las conexiones. Este postulado marcó el inicio del aprendizaje no supervisado.

Red de una sola neurona

La red más sencilla que realiza una asociación se muestra en la figura 3.34, con una sola neurona de entrada y una función de transferencia tipo limitador fuerte (hardlim):

$$a = hard\ \lim(\omega p + b). \qquad [3.30]$$

Se considera a p con dos valores posibles: 0 para la ausencia del estímulo y 1 para la presencia del estímulo. Si a es 1 hay respuesta por parte de la red, y si es 0 no hay respuesta.

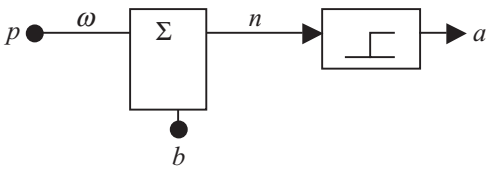


Fig. 3.42 Red de una sola neurona que realiza una asociación.

Tipos de estímulos

Se han definido dos tipos de estímulos:

- Estímulo no condicionado: refuerza el aprendizaje y ayuda a hacer la asociación con la salida deseada; se presenta intermitentemente para simular un proceso real de aprendizaje y memorización en la red.
- Estímulo condicionado: se presenta siempre a la red y ésta debe asociarlo con la salida deseada, para entregar la respuesta correcta cada vez que se introduce este estímulo.

Ejemplo

Asociar la forma de una fruta pero no su olor, mediante una neurona.

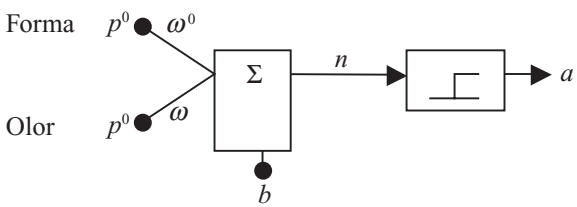


Fig. 3.43 Red para asociación de frutas.

El estímulo no condicionado p^0 será 1 si la forma es detectada y 0 si no lo es. El estímulo condicionado p será 1 si el olor es detectado y 0 si no lo es.

$$a = hard\ \lim(\omega^0 p^0 + \omega p + b). \qquad [3.31]$$

Para iniciar con el asociador se asigna $\omega^0 = 1$, $\omega = 0$ y $b = -0.5$, por lo que la función de transferencia se simplifica como $a = hard\ \lim(p^0 - 0.5)$. La red responderá sólo si $p^0 = 1$, es decir, sólo con la forma.

Interpretación de la Regla de Hebb en asociadores lineales

Para un asociador, la Regla de Hebb se interpreta de la siguiente manera: si dos neuronas en cualquier lado de la sinapsis son activadas simultáneamente, la longitud de la sinapsis se incrementará. La figura 3.36 muestra la estructura de un asociador lineal con un vector de entrada *p*, que se representa con la ecuación:

$$a_i = \sum_{j=1}^{R} \omega_{ij} p_j . \qquad [3.32]$$

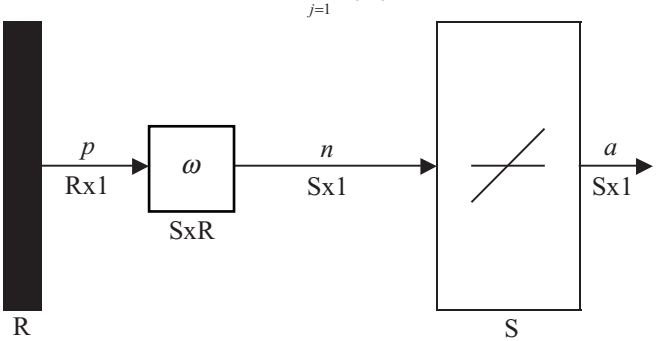


Fig. 3.44 Estructura de un asociador lineal.

Donde la conexión o sinapsis entre la entrada p_j y la salida a_i es el peso ω_{ij}, por lo que si un valor positivo en la entrada produce un valor positivo en la salida el peso debe incrementarse:

$$\omega_{ij}^{\ nuevo} = \omega_{ij}^{\ anterior} + \alpha(a_{iq})(p_{jq}). \qquad [3.33]$$

Para un vector q-ésimo de j elementos de la entrada p e i elementos de la salida a y una tasa de aprendizaje α que es un valor positivo constante. Los pesos serán incrementados si la entrada y la salida son positivas o negativas, y serán disminuidos si la entrada y la salida tienen signos contrarios.

Una manera de mejorar la Regla de Hebb es mediante la incorporación de una tasa de olvido γ, positiva y menor a 1, que controla el crecimiento de la matriz de pesos:

$$\omega(q) = \omega(q-1) + \alpha a(q) p^T(q) - \gamma\omega(q-1) = (1-\gamma)\omega(q-1) + \alpha a(q) p^T(q) \qquad [3.34]$$

TOPOLOGÍA DE REDES NEURONALES EMPLEADAS PARA LA CLASIFICACIÓN, PREDICCIÓN Y RECONOCIMIENTO DE PATRONES

Red Instar

Este tipo de redes se emplea para el reconocimiento de patrones, a partir de una entrada vectorial como muestra la figura 3.45 y asociaciones y aprendizaje no supervisado.

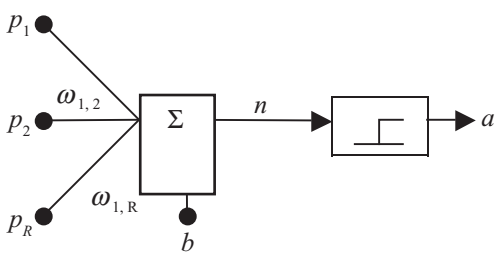


Fig. 3.45 Red Instar.

La ecuación de transferencia de la red Instar es la siguiente:

$$a = hard\lim(\omega^T p + b). \quad [3.35]$$

La red se activará si el producto punto entre el vector de pesos y la entrada sea mayor o igual al bias con signo negativo. El mayor producto punto se presentará cuando los dos vectores (el del peso y el de las entradas) apunten en la misma dirección. El mayor valor del bias se presentará cuando la red esté activa.

Para obtener los beneficios del término de peso con tasa de olvido se agrega un nuevo término proporcional a las salidas:

$$\omega_{ij}(q) = \omega_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma a_i(q)\omega_{ij}^{anterior}. \quad [3.36]$$

Si la tasa de olvido es igual a la de aprendizaje se obtiene la ***Regla de Instar*** de forma vectorial:

$$\omega(q) = \omega(q-1) + \alpha(p(q) - \omega(q-1)) = (1-\alpha)\omega(q-1) + \alpha p(q). \quad [3.37]$$

Red Outstar

La red Outstar (figura 3.46) tiene una entrada escalar y una salida vectorial y recuerda patrones por asociación de un estímulo de entrada con un vector de respuesta. La expresión de salida corresponde a:

$$a = satlins(\omega p).$$

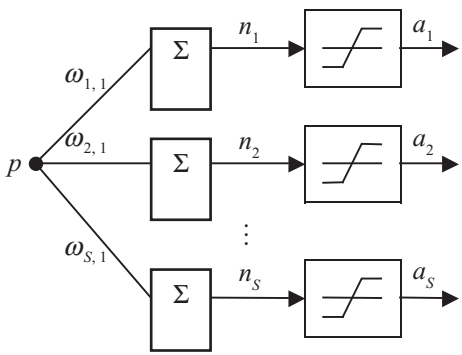


Fig. 3.46 Red Outstar.

Se desea recordar un vector de valores entre -1 y 1 por lo que se emplea la función de saturación simétrica (satlins). Para incorporar el coeficiente del olvido se multiplica éste por la entrada de la red para simular el estímulo no condicionado, por lo que:

$$\omega_{ij} = \omega_{ij}(q-1) + \alpha a_i(q) p(q) - \gamma p(q)\omega_{ij}(q-1). \quad [3.38]$$

Haciendo a los coeficientes de aprendizaje y olvido iguales se obtiene:

$$\omega_{ij} = \omega_{ij}(q-1) + \alpha(a_i(q) - \omega_{ij}(q-1)) p(q). \quad [3.39]$$

Redes Competitivas

En este tipo de redes las neuronas compiten para activarse quedando finalmente una, o una por grupo, como neurona vencedora; el resto quedarán anuladas y siendo forzadas a sus valores de respuesta mínimos. La competencia se realiza en todas las capas de la red, existiendo conexiones de autoexcitación y

conexiones de inhibición. El objetivo de la competencia es categorizar o agrupar los datos de entrada, de tal manera que las entradas de la misma categoría activen la misma neurona de salida.

Red de Kohonen

Existen evidencias que demuestran que en el cerebro hay neuronas organizadas en zonas, de tal forma que la información captada del entorno se representa internamente en forma de mapas bidimensionales. Estas teorías se sustentan en que algunas áreas del cerebro pueden crear y ordenar neuronas especializadas y construir mapas especiales para atributos y características.

El modelo propuesto por Tuevo Kohonen en los años ochenta tiene dos variantes:

- Learning Vector Quantization (LVQ)
- Topology Preserving Map (TPM) o Self Organizing Map (SOM)

Las dos variantes se basan en la formación de mapas topológicos para establecer características comunes entre la información de la entrada (vectores). En el caso del LVQ los mapas son de una dimensión, y en el SOM son bidimensionales o tridimensionales.

El aprendizaje en las redes de Kohonen es de tipo offline, es decir existe una etapa de aprendizaje donde se fijan los pesos de las conexiones y otra de funcionamiento.

Un concepto muy importante es la zona de vecindad o vecindario alrededor de la neurona vencedora i^*, ya que los pesos de las neuronas que se encuentran en esa zona denominada $X(q)$ serán actualizados en conjunto con el de la vencedora (aprendizaje cooperativo). El algoritmo de aprendizaje es el siguiente:

1. Se inicializan los pesos con valores aleatorios pequeños y se fija la vecindad entre las neuronas de salida.
2. Se presenta la entrada p en forma de vector con valores continuos.
3. Se determina la neurona vencedora de la capa de salida, cuyos pesos de conexión con las entradas son los más parecidos a la entrada p. Para esto se calculan las distancias euclidianas entre los dos vectores, entradas y pesos:

$$d_i = \sum_{j=1}^{N} (p_j - \omega_{ij})^2, 1 \le i \le M. \qquad [3.40]$$

4. Con la neurona vencedora se actualizan los pesos entre las entradas y ésta, y entre las vecinas y las entradas, con la siguiente expresión:

$$\omega(q) = \omega(q-1) + \alpha(q)(p(q) - \omega(q-1)) \text{ para } i \in X(q). \qquad [3.41]$$

El tamaño de $X(q)$ puede reducirse en cada iteración y el término $\alpha(q)$ es el coeficiente de aprendizaje, con un valor entre 0 y 1 el cual decrece con el número de iteraciones q del proceso de entrenamiento. Para encontrar α se utilizan:

$$\alpha(q) = \frac{1}{q} \text{ ó } \alpha(q) = \alpha_1\left(1 - \frac{q}{\alpha_2}\right) \qquad [3.42]$$

5. El proceso se repite con todos los patrones de aprendizaje hasta obtener la salida deseada. El aprendizaje ocurre cuando la neurona i sea miembro del conjunto $X(q)$.

Este tipo de redes es especialmente útil para establecer relaciones desconocidas previamente entre conjuntos de datos.

Red de Hamming

La red de Hamming se basa en el aprendizaje competitivo y consta de dos capas: la primera es una red Instar que realiza la correlación entre el vector de entrada y los vectores protitpo, y la segunda realiza la competición para determinar cuál de los vectores prototipo está más cercano al vector de entrada. La competición se implementa por inhibición lateral, es decir, por un conjunto de conexiones negativas entre las neuronas de la capa de salida. Véase la figura 3.47.

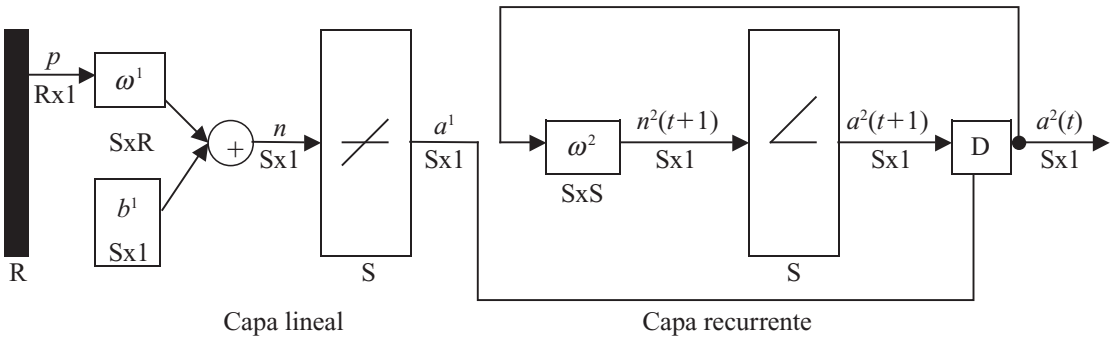


Fig. 3.47 Red de Hamming.

Problemas con las redes competitivas

1. Elección de la tasa de aprendizaje de tal manera que permita encontrar un punto de equilibrio entre la velocidad de convergencia y la estabilidad final de los vectores de peso.
2. Cuando las clases están muy juntas ocurre un problema de estabilidad, ya que si un vector de pesos trata de apuntar hacia una clase determinada, puede entrar al territorio de otro vector de pesos.
3. Cuando el vector de pesos inicial de una neurona se encuentra muy lejos de cualquiera de los vectores de entrada y por lo tanto no se obtiene un ganador de la competición, por lo que la neurona "morirá", lo cual no es deseable.

Mapas de Autoorganización (SOM)

Estas redes comprenden un patrón de conexión entre neuronas donde cada una se refuerza a sí misma (on-center) mientras inhibe a todas las neuronas a su alrededor (off-surround). Se determina primero la neurona ganadora *i** y luego los vectores de pesos de las neuronas que se encuentren en una región cercana denominada vecindario N_{i^*}, dentro de un radio d, serán actualizados mediante la Regla de Kohonen:

$$\omega_i(q) = \omega_i(q-1) + \alpha(p(q) - \omega_i(q-1)) \text{ para } i \in N_{i*}(d),\ N_i(d) = \left\{ j, d_{ij} \le d \right\}. \qquad [3.43]$$

Learning Vector Quantization (LVQ)

Esta red es un híbrido que emplea aprendizaje no supervisado y supervisado para la clasificación de patrones. Cada neurona de la primera capa es asignada a una clase y después cada clase es asignada a una neurona en la segunda capa. El número de neuronas en la primera capa debe ser mayor o al menos igual que el número de neuronas en la segunda. Cada neurona de la primera capa aprende un vector prototipo que permite clasificar una región del espacio de entrada, pero en lugar de calcular la distancia entre la entrada y el vector de pesos por un producto punto, se calcula directamente. De esta manera los vectores no tienen que normalizarse.

Redes Recurrentes

Las redes recurrentes son redes dinámicas por naturaleza, como la red de Hopfield, la red de Jordan y la red de Elman, o de naturaleza estática pero realimentadas de sus salidas hacia las entradas. Son herramientas poderosas para simular e identificar sistemas dinámicos no lineales.

Red Hopfield

Esta red surge a partir de la teoría de control geométrico basado en la geometría diferencial. El modelo básico se presentó por primera vez como un circuito eléctrico de amplificadores operacionales como neuronas y redes de capacitores y resistencias. La entrada de cada amplificador es la suma de las corrientes en las entradas, más las realimentaciones de otros amplificadores. Es una red recurrente que normalmente tiene retroalimentación de las señales de salida.

El proceso de entrenamiento es un proceso iterativo en el que se aplican las señales de entrada y la salida se calcula; el proceso se repite hasta que la señal de salida es constante, que es cuando se dice que la red es una red estable. En caso de que la salida no sea una salida constante y sea una salida variable, se tiene una red inestable.

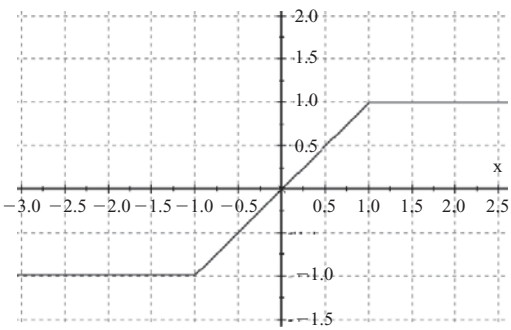
En los años ochenta J. Hopfield propuso el principio de operación de una red recurrente estable, la cual consta de un grupo de neuronas (n) en donde la salida de cada neurona sirve de retroalimentación en las entradas como muestra la figura 3.48, excepto para su propia entrada.

Cada una de las neuronas es normalmente es del tipo perceptrón, en donde se emplea una función harlim o sign que cumple la siguiente expresión.

$$Salida = Y = \begin{cases} +1, si \quad entrada \quad x > 0 \\ -1, si \quad entrada \quad x < 0 \\ Y, si \quad es \quad entrada \quad x = 0 \end{cases}$$

Se puede emplear también una función lineal con saturación, la cual implica que el estado de cero transición es más grande, pudiéndose representar como:

$$Salida = Y = \begin{cases} +1, si \quad entrada \quad x \geq 1 \\ -1, si \quad entrada \quad x \leq -1 \\ X, si \quad -1 < x < 1 \end{cases}$$



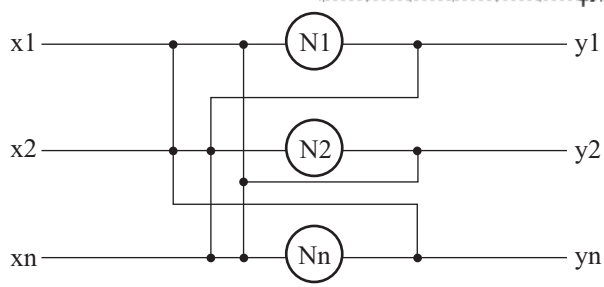


Fig. 3.48 Red Hopfield.

Tener $\theta = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

$$\therefore \omega = \begin{bmatrix} 0 & w_{12} & w_{1i} & w_{1n} \\ w_{21} & 0 & w_{2i} & w_{2n} \\ w_{i1} & w_{i2} & 0 & w_{in} \\ w_{n1} & w_{n2} & w_{ni} & 0 \end{bmatrix}$$

Para $w_{ij} = w_{ji}$ (simétrica)

Una vez que se define la función de activación se debe realizar el proceso de ajuste de pesos, en donde se tiene un vector de entrada y de salida (Y). Si tomamos este vector de salida definido por los patrones de entrada

$$Y = \begin{bmatrix} y1 \\ y2 \\ y3 \\ yn \end{bmatrix}$$

se pueden encontrar los pesos de acuerdo con $W = YmY_m^T - MI$

donde Y es el vector de entrada, I es la matriz identidad, Y^T es la transpuesta del vector de entrada y M es el número de estados que se desean memorizar.

Por ejemplo, se tiene los siguientes patrones para memorizar

$$Y1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, Y2 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \qquad \text{se tiene } Y1^T = [111],\ Y2^T = [-1-1-1]$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad W = YmY_m^T - 2I$$

$$W = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}\begin{bmatrix} -1 & -1 & -1 \end{bmatrix} - 2\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}$$

Para evaluar la red se tiene lo siguiente:

$$Ym = \text{sign}(WXm - \theta);\ M = 1,2,3\ldots M$$

Donde θ es threshold Matriz

$$Y_1 = \left\{ \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$Y_2 = \left\{ \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

¿Qué pasa con los siguientes estados?:

Estado			It/0−1	Entradas			Salidas			Memoria		
1	1	1	0	1	1	1						
−1	1	1	0	−1	1	1	1	1	1	1	1	1
			1	1	1	1						
1	1	−1	0	1	1	−1	1	1	1	1	1	1
			1	1	1	1	1	1	1	1	1	1
−1	−1	−1	0	−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	1	0	−1	−1	−1	−1	−1	−1	−1	−1	−1
			1	−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	1	−1										
1	−1	−1										

Nota: Recordar que los estados (0) en la función sign permanecen.

Corrector de error
Se puede ver que para un solo elemento como

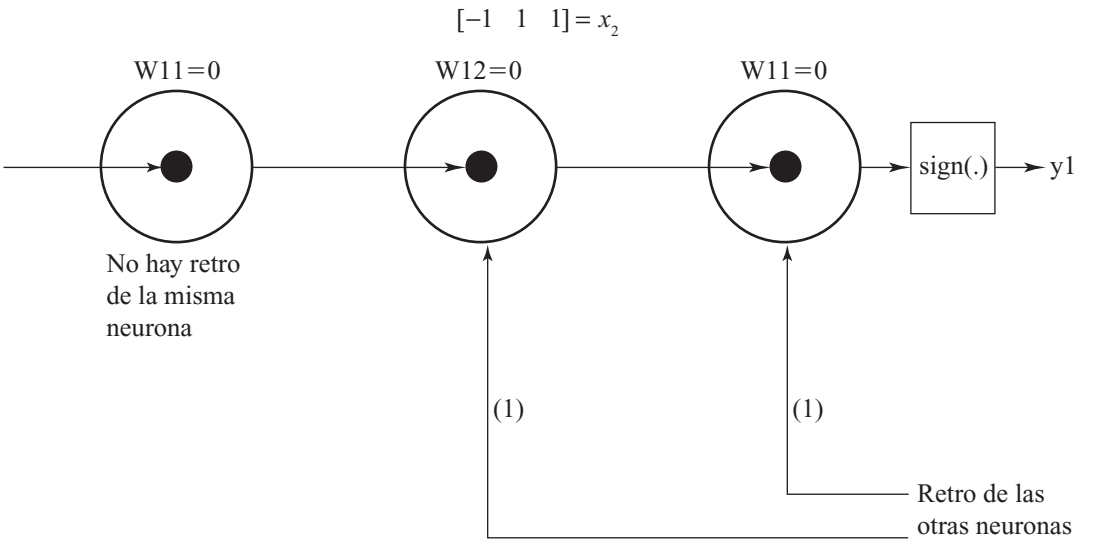


Fig. 3.49 Red Hopfield con retroalimentación de neuronas vecinas.

Redes ANFIS

Los sistemas ANFIS son sistemas basados en redes neurales adaptables con inferencia difusa. En dichos sistemas se tiene como principio el uso de diferentes métodos para el ajuste de los parámetros de ajuste,

entre ellos se encuentra el método de retropropagación del error. El principio de funcionamiento de este tipo de topologías fue propuesto por R. Jang en su trabajo de tesis neuro-fuzzy modeling.

La idea principal de este método es el empleo de los parámetros de las funciones de activación como los parámetros de ajuste, esto es, los pesos de las redes neurales convencionales, lo cual permite el ajuste automático de las reglas, así como el ajuste de las funciones de membresía.

El mapeo difuso puede respresentarse a través de una matriz de asociación difusa y después trasladarse a regiones de activación; esto se puede representar por redes tipo ANFIS las cuales pueden ser empleadas para el mapeo correcto de las decisiones en un sistema de inferencia difuso bajo un proceso de entrenamiento. La figura 3.50 muestra el diagrama básico de un sistema de inferencia.

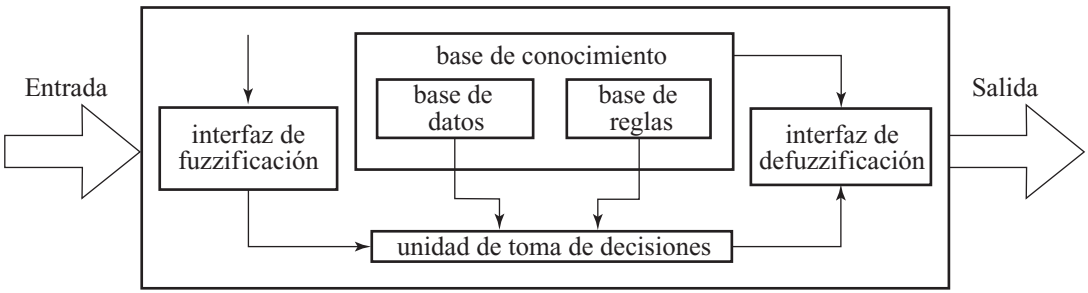


Fig. 3.50 Diagrama básico de un sistema de inferencia.

Un mapeo de las entradas se muestra en la figura 3.51 en que se aprecia el subespacio difuso correspondiente a nueve reglas.

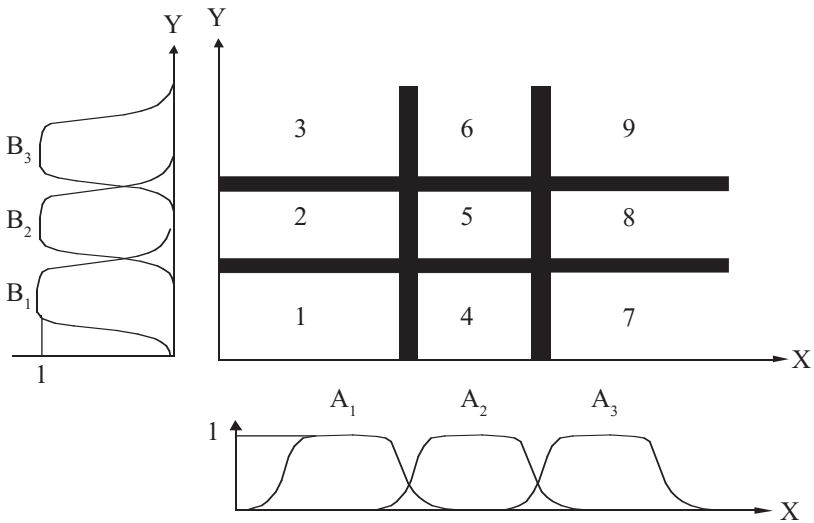


Fig. 3.51 Mapeo de entrada y salida.

Bajo este principio se extienden las redes a topologías ajustables como son las redes adaptables con inferencia difusa; en este tipo de redes aparecen nodos fijos y nodos adaptables.

La figura 3.52 denota los nodos fijos con círculos y los nodos adaptables con cuadros; en el proceso de entrenamiento se ajustan sólo los nodos adaptables.

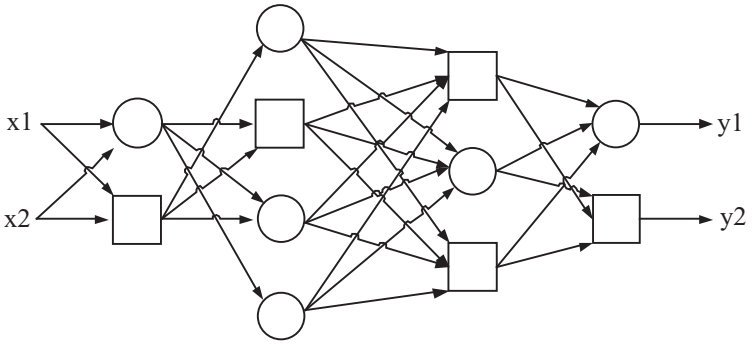


Fig. 3.52 Topología de una RED adaptable con nodos fijos y adaptables.

La figura 3.53 muestra cómo se tiene una segmentación de las partes de un sistema difuso emulado a través de una red neuronal adaptable. Se puede ver que la capa 1 es la responsable de ajustar los parámetros de la función de membresía de entrada; las capas 2 y 3 de la red son las encargadas de realizar las operaciones difusas, mientras que las capas 4 y 5 realizan una ponderación para obtener la salida del sistema (f).

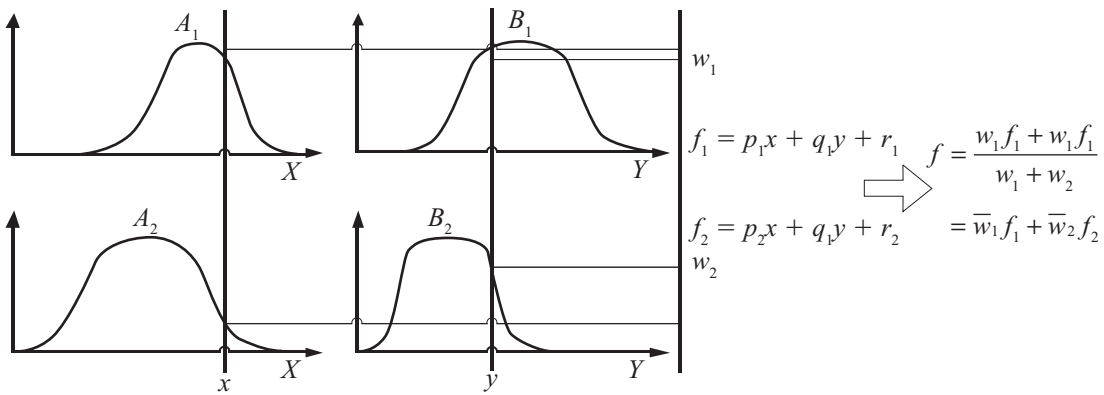


Fig. 3.53 Evaluación del sistema de inferencia difuso.

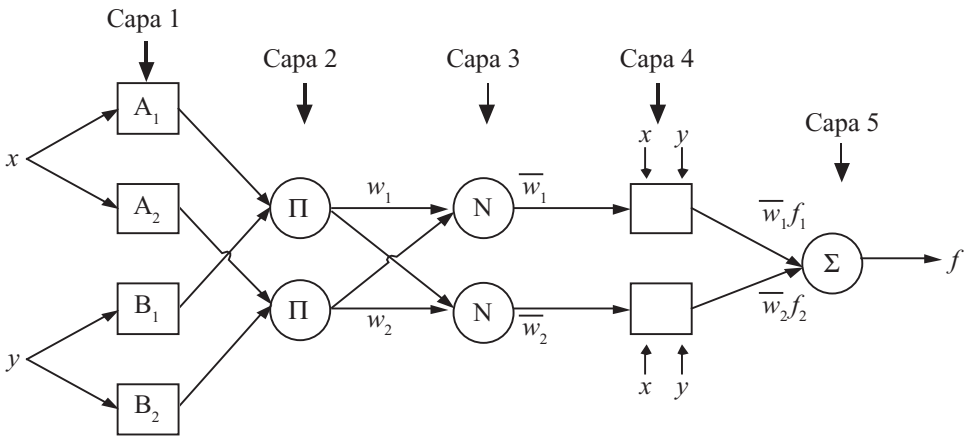


Fig. 3.54 Sistema difuso y red ANFIS para la emulación del sistema de inferencia.

Existen diferentes posibilidades de tener sistemas ANFIS; esto se puede observar en la topología ANFIS tipo I (figura 3.55); las topologías cambian de acuerdo con los requerimientos del sistema de inferencia.

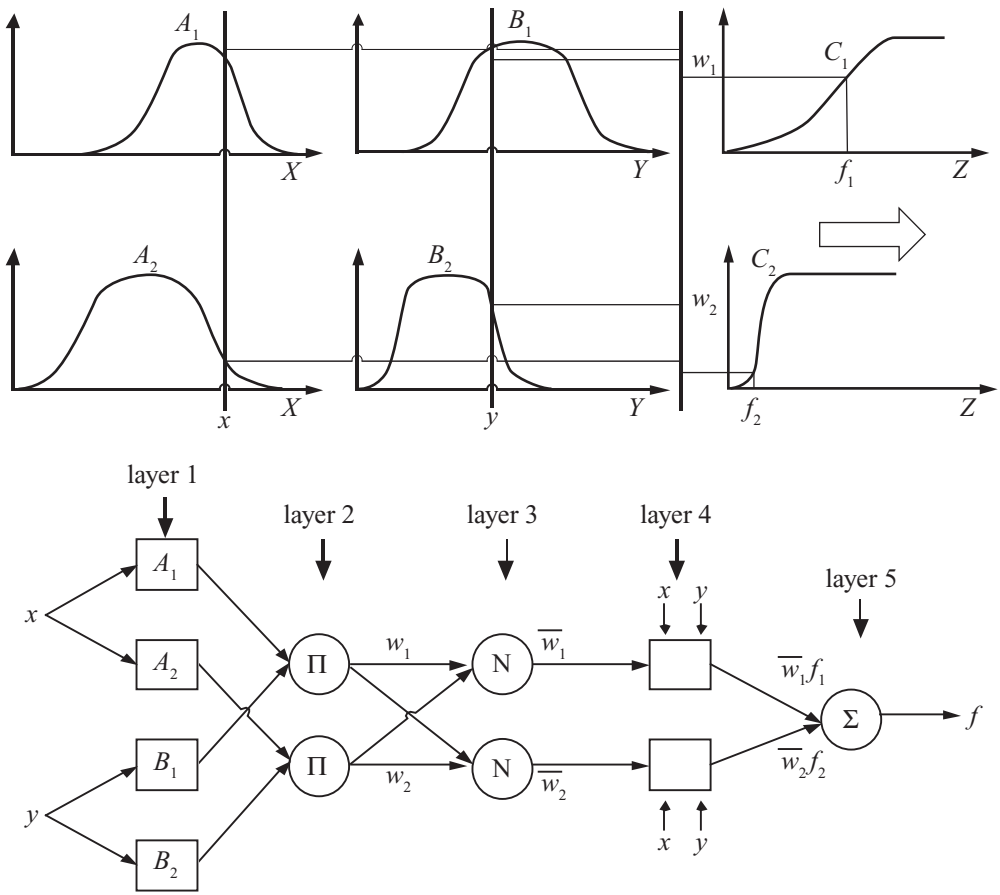


Fig. 3.55 Topología de una red ANFIS- tipo I.

Algoritmo de un sistema ANFIS

El algoritmo del sistema ANFIS presentado por Jang se basa en la generación del error definido por la siguiente ecuación, en donde *T* es el objetivo a alcanzar y *O* son las salidas generadas. Se puede ver que el error es una función de energía que se tiende a reducir durante el proceso de entrenamiento.

$$Ep = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^{L})^2 \qquad [3.44]$$

Como resultado el error acumulado se puede definir como $E = \sum_{p=1}^{p} E_p$

este error sirve para poder emplear el método del gradiente descendiente

$$\frac{\partial E_p}{\partial O_{i,p}^{L}} = -2(T_{i,p} - O_{i,p}^{L}) \qquad [3.45]$$

a través de ajuste de los valores de salida para los nodos externos, mientras que para los nodos internos se tiene

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,\mathbf{p}}^k} \qquad [3.46]$$

dicha expresión resulta de aplicar la regla de la cadena. Teniendo un parámetro de ajuste alfa se tiene

$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha}$,

resultando para cada conjunto de nodos relacionados con alfa $\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha}$

por lo tanto, se tiene como ecuación genérica para el parámetro alfa $\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha}$

y como coeficiente de aprendizaje $\eta = \frac{k}{\sqrt{\sum \alpha \left(\frac{\partial E}{\partial \alpha} \right)^2}}$.

donde *k* es el coeficiente que determina el paso de avance para el coeficiente de aprendizaje.

Esto se puede resumir en el algoritmo de entrenamiento para ANFIS.

Algoritmo de entrenamiento para ANFIS

Cada iteración o época está compuesta por un adelanto y un atraso. En el adelanto una etapa de entrenamiento del patrón de entrada (vector de entrada) está presente en las ANFIS; las neuronas de salida son calculadas en la capa mediante los parámetros de ésta, y las reglas de parámetros consecuencia son identificadas por el estimador de mínimos cuadrados.

En la técnica de interferencia difusa tipo Sugeno, cada salida puede ser una función lineal, por lo que pueden formarse *p* ecuaciones lineales en términos de los parámetros consecuencia, de la siguiente forma:

$$yd(1) = \bar{\mu}_1(1) f_1(1) + \bar{\mu}_2(1) f_2(1) + ... + \bar{\mu}_n(1) f_n(1)$$
$$yd(2) = \bar{\mu}_1(2) f_1(2) + \bar{\mu}_2(2) f_2(2) + ... + \bar{\mu}_n(2) f_n(2)$$
$$Yd(P) = \bar{\mu}_1(P) f_1(P) + \bar{\mu}_2(P) f_2(P) + ... + \bar{\mu}_n(P) f_n(P)$$

también se puede expresar como:

$$Yd(1) = \bar{\mu}_1(1)[k_{10} + k_{11}x_1(1) + k_{12}x_2(1) + ... + k_{1m}x_m(1)]$$
$$+\bar{\mu}_2(1)[k_{20} + k_{21}x_1(1) + k_{22}x_2(1) + ... + k_{2m}x_m(1)]$$
$$+\bar{\mu}_n(1)[k_{n0} + [k_{n1}x_1(1) + k_{n2}x_2(1) + ... + k_{nm}x_{\mathbf{m}}(1)]$$

donde m = variables de entrada

n = número de neuronas en la capa de las reglas

En la notación de la matriz se tiene:

$$Yd = Ak$$

Yd es de las dimensiones *Px*1

$Yd = \begin{bmatrix} yd(1) \\ yd(2) \\ yd(P) \end{bmatrix}$; donde *A* es una matriz de $Pxn(1 + m)$

$$A=\begin{bmatrix} \bar{\mu}_1(1)\bar{\mu}_1(1)X_1(1)\cdots\bar{\mu}_1(1)X_m(1)\cdots\bar{\mu}_n(1)X_m(1) \\ \bar{\mu}_1(2)\bar{\mu}_1(2)X_1(2)\cdots \quad \cdots \quad \cdots\bar{\mu}_n(2)X_m(2) \\ \vdots \\ \bar{\mu}_1(P)\bar{\mu}_1(P)X_1(P)\cdots \quad \cdots \quad \cdots\bar{\mu}_n(P)X_m(P) \end{bmatrix}$$

$k =$ es un vector $n(1 + m)x1$ consecuencia, de parámetros desconocidos

$$k=[k_{10}k_{11}k_{12}\cdots k_{1m}k_{20}k_{21}k_{22}\cdots k_{2m}\cdots k_{n0}k_{n1}k_{n2}\cdots k_{nm}]^T$$

Normalmente el número de patrones P de entradas-salidas usado en el entrenamiento es mayor que el número de parámetros consecuencia $n(1 + m)$. Se está tratando con un sistema sobrederterminado, donde puede no existir una solución exacta.

Se tiene que hallar una estimación de mínimos cuadrados k, k^T que reduzca el error cuadrático. $\|Ak - yd\|^2$ *está compuesta por la técnica pseudoinversa*

$$k^T=(A^TA)^{-1}A^Tyd$$

la *pseudoinversa* es $(A^TA)^{-1}A^T$ de A, si (A^TA) no es singular.
tan pronto como los parámetros consecuencia de las reglas son definidos, se puede computar el vector salida (y) y determinar el vector error e.

$$e=Yd-y$$

el algoritmo de back-propagation es aplicado en el atraso.

Permitamos, por ejemplo, considerar aplicar una corrección al parámetro a de la función de activación de campana usado en la neurona $A1$.

$$\Delta\alpha=-\alpha\frac{\partial E}{\partial\alpha}=-\alpha\frac{\partial E}{\partial\alpha}\frac{\partial e}{\partial y}\frac{\partial y}{\partial(\bar{\mu}_i f_i)}\frac{\partial(\bar{\mu}_i f_i)}{\partial\bar{\mu}_i}\frac{\partial\bar{\mu}_i}{\partial\bar{\mu}_{Ai}}\frac{\partial\bar{\mu}_{Ai}}{\partial\alpha}$$

Donde $\alpha =$ es grado de aprendizaje

$E =$ es el valor inmediato del error cuadrático para las ANFIS

$$E=\frac{1}{2}e^2=\frac{1}{2}(Yd-y)^2$$

Así se obtendrá

$$\Delta a=-\alpha(Yd-y)(-1)f_i\frac{\bar{\mu}_i(1-\bar{\mu}_i)}{\bar{\mu}_i}\frac{\bar{\mu}_i}{\bar{\mu}_{A1}}\frac{\partial\bar{\mu}_{A1}}{\partial a}$$

o también

$$\Delta a=-\alpha(Yd-y)f_i\,\bar{\mu}_i(1-\bar{\mu}_i)\frac{1}{\bar{\mu}_{A1}}\frac{\partial\bar{\mu}_{A1}}{\partial a}$$

donde

$$\frac{\partial\bar{\mu}_{A1}}{\partial a}=-\left[\frac{1}{\left(1+\left(\frac{x_1-a}{c}\right)^{2b}\right)^2}\frac{1}{c^2b}2b(x_1-a)^{2b-1}(-1)\right]$$

$$=\bar{\mu}_{A1}2\frac{2b}{c}\left[\frac{x_1-a}{c}\right]^{2b-1}$$

De manera similar se puede obtener la corrección para los parámetros *b* y *c*.

A continuación se presentan varios ejercicios de ANFIS apoyados por MATLAB®.

En el primero se define el conjunto de instrucciones que posteriormente se ejecutarán para conocer el comportamiento del sistema:

```
clc
clear

% Ejemplo Anfis

% Number of input data points numPts = 51 and input data x

Con esto generamos un vector con los datos separados de manera equidistantes.
numPts = 51; x=linspace(-1,1,numPts)';

%Output data y

Generamos el vector de salidas, en este caso una suma de senos con diferentes frecuencias.
y=0.6*sin(pi*x)+0.3*sin(3*pi*x)+0.1*sin(5*pi*x);

%Store data in matrix called data; use part for trainning part for checking

Se genera la matriz de entrenamiento de entrenamiento.

data = [x y]; % total data set

Se divide el vector data en dos vectores, uno de entrenamiento y uno de pruebas

trndata = data(1:2:numPts,:); %training data set
chkdata = data(2:2:numPts,:); %checking data set

%Plot training data (o) and checking data(x)

Se grafican los vectores de entrenamiento y pruebas

plot(trndata(:,1),trndata(:,2), 'o', chkdata(:,1), chkdata(:,2),'x')
grid
title('Training data (x) and checking data(o)')
xlabel('x'); ylabel('Measured data for y');

%Next apply genfis1 to generate initial set of membership functions.
```

```
 Información del sistema difuso, 5 funciones de membresía de tipo Jang
nummfs=5;% number of membership functions
mftype = 'gbellmf'; % membership functions type is generalized bell

Se genera el sistema difuso con los datos anteriores.
fismat = genfis1(trndata, nummfs, mftype);

Gráfica de las funciones de membresía iniciales
plotmf(fismat, 'input', 1);

%Determine number of iteratios.

Número de iteraciones para entrenamiento
numepochs = 40;

%Start the optimization

Se ajustan los parámetros del sistema difuso utilizando el método de entrenamiento de ANFIS
[fismat1, trnerr, ss, fismat2, chkerr] = ...
anfis(trndata, fismat, numepochs, NaN, chkdata);

Se grafican las nuevas funciones de membresía
plotmf(fismat1,'input',1)

Se evalúa el resultado del sistema difuso entrenado con los datos de entrada del conjunto de pruebas

out=evalfis(chkdata(:,1), fismat1);

%Evaluates the output of the fuzzy system fismat1 - input checking

Se grafican los datos de prueba con el resultado obtenido por el sistema difuso

hold;

plot(chkdata(:,1), out);

%Plots out vs x

plot(chkdata(:,1), chkdata(:,2));
```

```
%Plots y vs x (training data)

plot(x,y); grid
```

En el segundo ejercicio se modifica el vector de salidas y se ejecuta el programa. Se cambió la función a aproximar por la siguiente:

$$y=0.6*\cos(pi*x)+0.3*\sin(3*pi*x)+0.1*\cos(5*pi*x);$$

El sistema logra entrenar de manera exitosa la función propuesta, como se muestra en las gráficas de la figura 3.56.

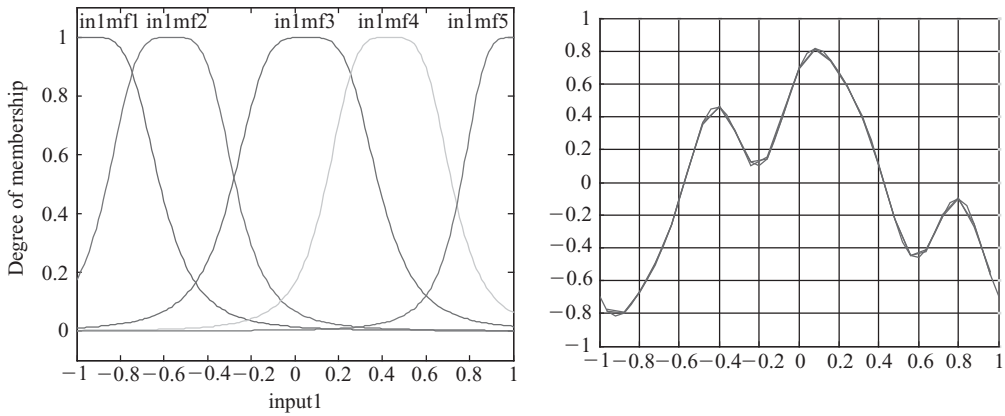


Fig. 3.56 Entrenamiento del sistema ANFIS.

El tercer ejercicio tiene el objetivo que el usuario conozca las herramientas de apoyo para poder obtener un mejor entendimiento sobre las ANFIS, por lo que se utilizará el editor de ANFIS de MATLAB®, que se abre con el siguiente comando: Anfisedit, con lo cual aparece la siguiente pantalla (figura 3.57):

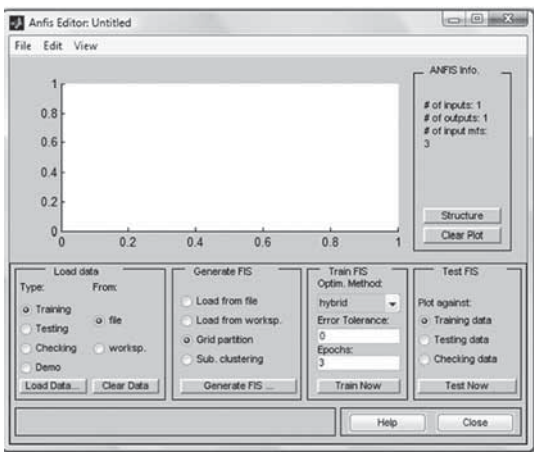


Fig. 3.57 Editor ANFIS de MATLAB®.

Se cargan los datos de entrenamiento y pruebas que ya teníamos anteriormente, dando click en load data form workspace e introduciendo el nombre de la función. Se selecciona el tipo de sistema difuso con 5 funciones de membresía de tipo campana de jang y salidas lineales (figura 3.58):

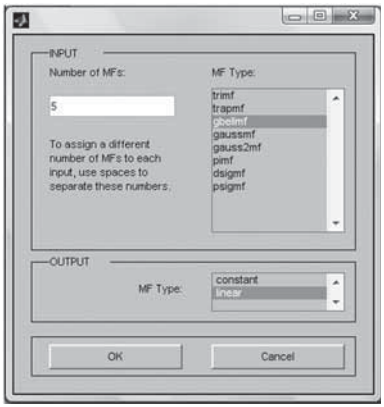


Fig. 3.58 Selección del tipo de sistema difuso.

Se selecciona el entrenamiento híbrido y 40 iteraciones y se procede a entrenar; puede observarse que desde la época 25 el error converge a 0.02.

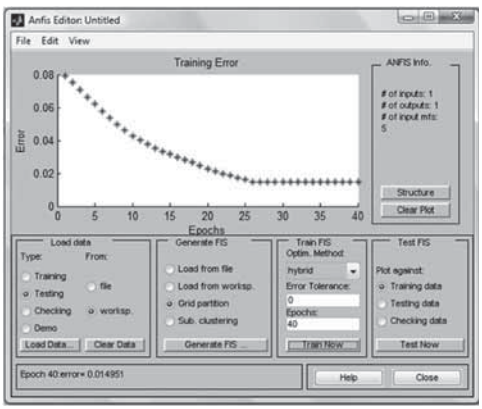


Fig. 3.59 Selección del entrenamiento híbrido y 40 iteraciones.

Se compara el desempeño del sistema con el conjunto de pruebas; puede verse que el resultado es casi idéntico (figura 3.60).

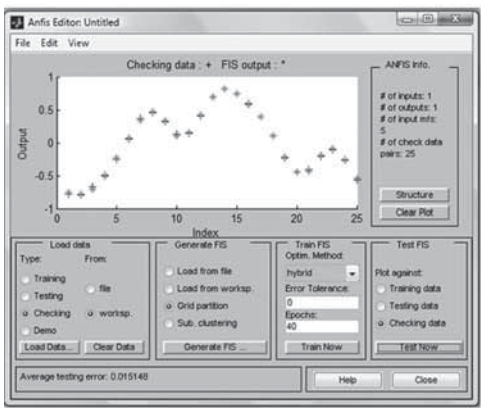


Fig. 3.60 Comparación entre el desempeño del sistema y el conjunto de pruebas.

Finalmente nos vamos a Edit >> Fis Properties.. y vemos la forma del sistema difuso entrenado, donde se puede apreciar las funciones de membresía de salida y el resultado del sistema (figura 3.61)

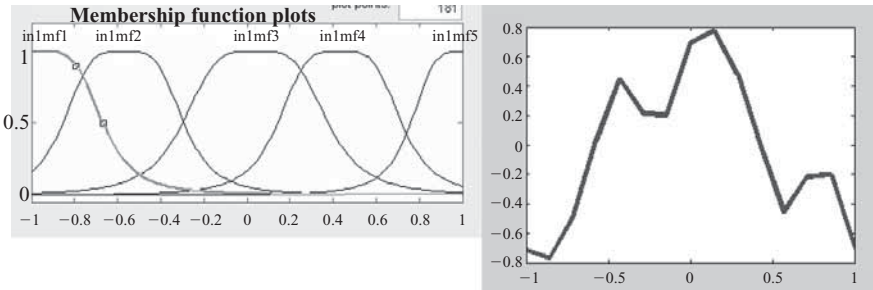


Fig. 3.61 Funciones de membresía de salida y resultado del sistema.

El cuarto ejercicio es un sistema de ecuaciones *sobrederterminado* con el cual se busca que el elector comprenda con mayor sencillez la *técnica pseudoinversa*.

Se tiene lo siguiente:

$$x_1 - x_2 = 2$$
$$x_1 + x_2 = 4$$
$$2x_1 + x_2 = 8$$

Se definen las matrices:

```
A =
     1    −1
     1     1
     2     1

>> yd = [2 4 8]'
yd =
     2
     4
     8
```

Se aplica el método de la pseudoinversa en el producto de las matrices, obteniendo los valores aproximados de x1 y x2. Se debe recordar que el sistema está sobredeterminado, ya que existen más ecuaciones que incógnitas, por lo que se puede obtener un sinfín de valores, los cuales pueden o no satisfacer el sistema propuesto.

```
>> pinv(A)*yd
ans =
   3.2857
   1.1429

>> k = ans
k =
   3.2857
   1.1429
```

Una vez obtenidos los valores sustituimos en las ecuaciones para comprobar su aproximación.

```
>> k(1) − k(2)

ans =
    2.1429

>> k(1) + k(2)

ans =
    4.4286

>> 2*k(1) + k(2)

ans =
    7.7143
```

Finalmente se puede apreciar que en los resultados, 2.1429 es cercano a 2, mientras que el valor 4.4286 se encuentra un poco alejado de 4 y 7.7143 es más próximo a 8.

El quinto ejercicio también es un sistema de ecuaciones *sobrederterminado*. La finalidad es encontrar en qué punto, si es que lo hay, se intersecan las tres rectas; se tiene:

$$x_1 + x_2 = 1$$
$$2x_1 + x_2 = 1$$
$$3x_1 + x_2 = 3$$

Se plantean las matrices:

```
>> A = [1 1;2 1;3 2]

A =
    1     1
    2     1
    3     2

>> yd = [1 1 3]'

yd =
    1
    1
    3
```

Aplicando el método de la pseudoinversa se obtiene:

```
>> k = pinv(A)*yd

k =
    0.0000
    1.3333
```

Obtenidos los valores sustituimos en las ecuaciones. Se grafica dichas ecuaciones para conocer si existe el punto de intersección (figura 3.62).

```
>> k(1) + k(2)

ans =
   1.3333

>> 2*k(1) + k(2)

ans =
   1.3333

>> 3*k(1) + k(2)

ans =
   1.3333
```

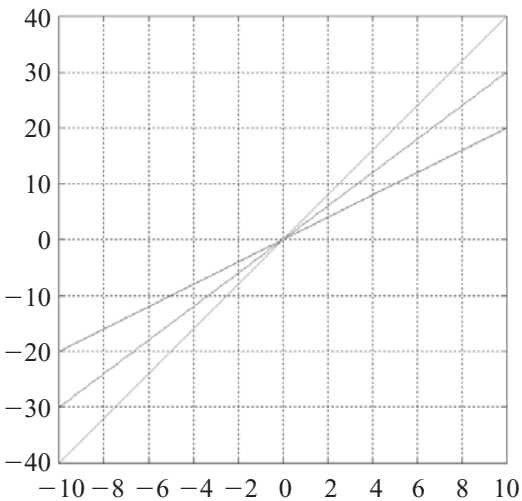


Fig. 3.62 Gráfica para ver los puntos de intersección.

La respuesta se puede apreciar que en 0 se intersecan las tres rectas pero en 3 nunca lo harán, por lo que 1.3 es la reducción del error.

Arquitectura de ANFIS

Para simplificar, se asumirá que el sistema de Interferencia difusa en estudio tiene dos entradas x y y con una salida z. Supóngase que las reglas base contienen dos reglas difusas *si-entonces* tipo Takagi y Sugeno.

Regla 1: Si x es A_1 y y es B_1, *entonces* $f_1 = p_1x + q_1y + \tau_1$,
Regla 2: Si x es A_2 y y es B_2, *entonces* $f_1 = p_1x + q_2y + \tau_2$,

Entonces el razonamiento difuso tipo 3 de la figura 3.54 y su equivalente correspondiente en arquitectura ANFIS (tipo-3 ANFIS) se muestra en la figura 3.55. Las funciones nodo en la misma capa pertenecen a la misma familia de funciones como se describe a continuación.

Capa 1: Todo nodo *i* en esta capa es un nodo cuadrático con una función nodo

$$O_i^1 = \mu_{A_i}(x)$$

donde x es la entrada al nodo i y A_i es la etiqueta lingüística (*largo, corto*, etc.) asociada a la función nodo. En otras palabras, O_i^1 es la función membresía de A_i y especifica el grado de satisfacción que dada la x, cumplirá con el cuantificador A_i. Usualmente se elige una $\mu_{A_i}(x)$ para representar el máximo valor gráficamente de la función campana, equivalente a 1 y un mínimo valor para que sea equivalente a 0, tal que:

$$\mu_{A_i}(x) = \frac{1}{1 + \left[\left(\frac{x - c_i}{a_i}\right)^2\right] a_i}$$

o

$$\mu_{A_i}(x) = \exp\left\{-\left(\frac{x - c_i}{a_i}\right)^2\right\}$$

donde $\{a_i, b_i, c_i\}$ son el conjunto de parámetros. Cuando los valores de estos parámetros cambian, la forma de la campana variará por consiguiente, así como la diversidad de formas que las funciones de membresía puedan presentar en las etiquetas lingüísticas A_i. De hecho, las funciones continuas derivables (en cualquier segmento), así como las funciones de membresía más usadas, la trapezoidal y triangular, son también fuertes candidatas para funciones nodo en esta capa. Los parámetros en esta capa son conocidos como la *base (premisa) de los parámetros*.

Capa 2: Todo nodo en esta capa es un nodo circular con etiqueta Π que multiplica las señales de entrada y manda el resultado del producto fuera de esta capa. Por ejemplo,

$$w_i(x) = \mu_{A_i}(x) \mathbf{x} \mu_{B_i}(y), \qquad i = \mathbf{1, 2.}$$

Cada nodo de salida representa la fuerza de relación de cada regla. (De hecho, otro operador *T-norm* que generaliza el desempeño AND puede ser usado como función nodo en esta capa.)

Capa 3: Todo nodo en esta capa es un nodo circular con etiqueta N. El *enésimo* nodo calcula la proporción de fuerza de relación de la *enésima* regla como el total de fuerza de relación de todas las reglas:

$$w_i(x) = \frac{w_i}{w_1 + w_2}, \qquad i = 1, 2.$$

Por comodidad, las salidas de capa se llaman *normalización de fuerza de relación*.

Capa 4: Todo nodo *i* en esta capa es un nodo cuadrado con una función nodo

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i)$$

donde $\overline{w}_i$ es la salida de la capa 3 y $\{p_i, q_i, r_i\}$ son el conjunto de parámetros. Los parámetros en esta capa se conocen como parámetros de consecuencia.

Capa 5: El único nodo en esta capa es un nodo circular con etiqueta $\sum$ que computará el total de las salidas como la sumatoria de todas las señales de entrada, por ejemplo.

$$O_i^5 = \textit{overall output} = \sum_i \overline{w}_i f_i = \frac{\sum_i \overline{w}_i f_i}{\sum_i wi}$$

Método de mínimos cuadrados

Si consideramos que se tienen sistemas lineares y datos de entrada y salida, y g es el sistema físico que deseamos identificar, G entonces será el conjunto de entrenamiento (definido por los datos experimentales de entrada y salida generados desde este sistema). Podemos usar un modelo (para sistemas lineales)

$$y(k) = \sum_{i=1}^{q} \theta_{ai} y(k-i) + \sum_{i=0}^{p} \theta_{bi} u(k-i)$$

donde u(k) y y(k) son las entradas y salidas del sistema en el tiempo k.

Si:

$f(x,\theta) = \theta^t x(k)$

para

$x(k) = [y(k-1), \ldots, y(k-q), u(k) \ldots, u(k-p)]^t$

y

$\theta = [\theta_a, \ldots \theta_{aq}, \ldots \theta_{bp}]^t$

En un grupo del método de mínimos cuadrados definimos

$y(m) = [y^1, y^2, \ldots y^m]^t$

$$\theta(m) = \begin{bmatrix} [(x]_1)^t \\ [(x]_2)^t \\ [(x]_m)^t \end{bmatrix}$$

$\in_i = y^i - [(x]^i)^t \theta$ (es el error en aproximar los datos (x^i, y^i)

$\in$ G usando θ

$E(m) = [\in_1, \in_2, \ldots \in_m]^t$

entonces

$E = Y - \phi\theta$

si

$$V(\theta) = \frac{1}{2} E^t E$$

Lo que se quiere es tomar θ para reducir V(θ). Si tomamos la derivada parcial de V con respecto a θ y lo igualamos a cero, tenemos un ecuación para θ'.

Otro acercamiento a la derivación es notar que

$zv = E^t E = Y^t Y - Y^t \phi\theta - \theta^t \phi^t Y + \theta^t \phi^t \phi\theta$

Entonces se completa el cuadrado ($\phi^t \phi$ es invertible)

$zv = Y^t Y - Y^t \phi\theta - \theta^t \phi^t Y + \theta^t \phi^t \theta + Y^t \phi[(\phi]^t \phi)^{-1} \phi^t Y$

$zv = Y^t (I - \phi[(\phi]^t \phi)^{-1} \phi^t) Y + [(\theta - [(\phi]^t \phi)]^{-1} \phi^t Y)^t \cdots \phi^t \phi(\theta - [(\phi]^t \phi^{-1} \phi^t Y)$

El primer término es independiente de θ, entonces no podemos reducir V a través de este término, por lo cual puede ser ignorado. El segundo término es cero, por lo que resulta:

$\phi' = (\phi^t \phi)^{-1} \phi^t Y$

En grupos con pesos de mínimos cuadrados

$$V(\theta) = \frac{1}{2} E^t W E$$

$\theta' = (\phi^t W \phi)^{-1} \phi^t W Y$

Ejemplo: ajuste una línea al conjunto de datos (nuestro modelo parametrizado es)
$y = x_1\theta_1 + x_2\theta_2$

Si escogemos $x_2 = 1$, "y" es una línea

$$\left\{\left(\begin{bmatrix}1\\1\end{bmatrix},1\right),\left(\begin{bmatrix}2\\1\end{bmatrix},1\right),\left(\begin{bmatrix}3\\1\end{bmatrix},3\right)\right\}$$

Reduciendo la suma de la distancias cuadradas entre la línea y los datos, tomamos que

$$\phi = \begin{bmatrix}1 & 1\\ 2 & 1\\ 3 & 1\end{bmatrix}$$

$$Y = \begin{bmatrix}1\\1\\3\end{bmatrix}$$

entonces

$$\phi' = (\phi^t\phi)^{-1}\phi^t Y = \left(\begin{bmatrix}14 & 6\\ 6 & 3\end{bmatrix}\right)^{-1}\begin{bmatrix}12\\5\end{bmatrix} = \begin{bmatrix}1\\ -\frac{1}{3}\end{bmatrix}$$

$$y = x_1 - \frac{1}{3}$$

Mínimos cuadrados recursivos

Nos permite actualizar θ' estimándolo cada vez que recibimos un nuevo par de entradas (datos) sin usar todos los datos anteriores, además de reducir el costo computacional de la pseudoinversa, para evitar el cálculo del inverso de $\phi^t\theta$, podemos partir de que:

$$V(\theta,k) = \frac{1}{2}\sum_{i=1}^{k}\lambda^{k-1}(y^i - (x^i)^t\theta)^2$$

donde se tiene que

$$p(k) = \frac{1}{\lambda}\Big(I - p(k-1)\Big)x^k\ldots$$

Lo que resulta en la expresión de mínimos cuadrados recursivos:

$$\theta'(k) = \theta'(k-1) + p(k)x^k\left(y - (x^k)\right)^t\theta'(k-1)$$

Ejemplo ANFIS con línea de comandos

```
% Number of input data points numPts = 51 and input data x
numPts=51;x=linspace(-1,1,numPts)';
```

```
% Output data y
y=0.6*sin(pi*x)+0.3*sin(3*pi*x)+0.1*sin(5*pi*x);
% Store data in matrix called data; use part for training part for checking
data=[x y]; % total data set
trndata = data(1:2:numPts,:); % training data set
chkdata = data(2:2:numPts,:); % checking data set
% Plot training data (o) and checking data (x)
plot (trndata(:,1),trndata(:,2), 'o',chkdata(:,1),chkdata(:,2),'x')
grid
title('Training data (x) and checking data (o)')
xlabel('x');ylabel('Measured data for y');
 % Next apply genfis1 to generate initial set of membership functions.
nummfs=5; % number of membership functions
mftype='gbellmf'; % membership functions type is generalized bell
fismat = genfis1(trndata,nummfs,mftype);
plotmf(fismat,'input',1);
 % Determine number of iterations
numepochs = 40;
% Start the optimization
[fismat1, trnerr, ss, fismat2, chkerr]= ...
anfis(trndata, fismat, numepochs, NaN, chkdata);
plotmf(fismat1,'input',1)
out=evalfis(chkdata(:,1),fismat1);
% Evaluates the output of the fuzzy system fismat1 - input checking
hold;
plot(chkdata(:,1),out);
% Plots out vs x
plot(chkdata(:,1),chkdata(:,2));
% Plots y vs x (training data)
plot(x,y); grid
```

Ejemplo sistema ANFIS empleando ANFIS EDIT de MATLAB®

Para el siguiente ejemplo se entrenaron tres redes difusas para que simularan el comportamiento de tres sistemas, uno de los cuales era un sistema real. Para comenzar, se propuso un sistema para que la red ANFIS fuera entrenada y tratara de emularlo. El sistema propuesto es una ecuación matemática, en este caso una exponencial. El sistema es el siguiente:

$$y = e^{-x}$$

Si lo graficamos en MATLAB® de 0 a 5 nos queda como muestra la figura 3.63.

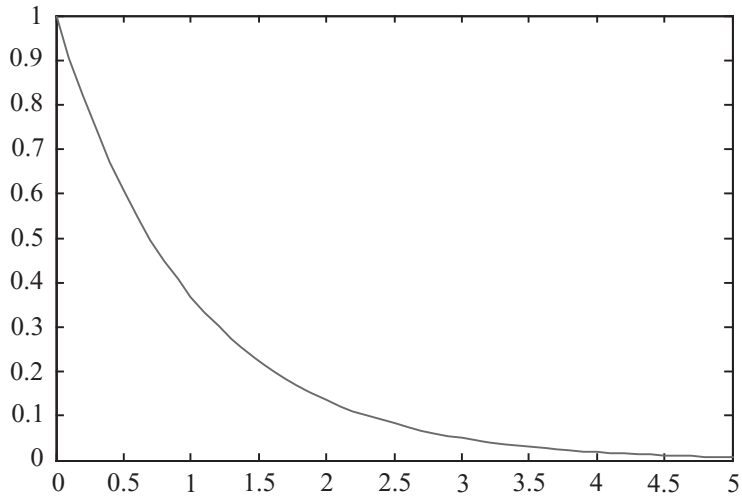


Fig. 3.63 Gráfica de sistema ANFIS en MATLAB®.

Esta función la usamos para entrenar la red ANFIS en la que proponemos cinco funciones de membresía tipo campana para la entrada. Después de aproximadamente 20000 iteraciones llegamos a un error de 0.38 en el que el sistema permanece estacionario (figura 3.64).

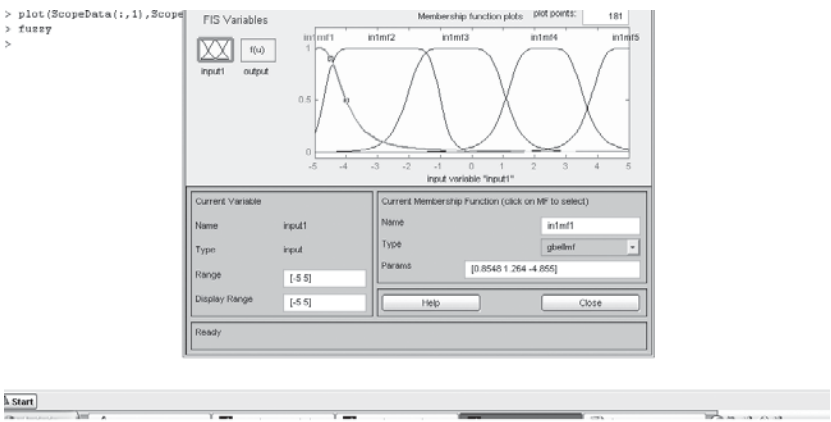


Fig. 3.64 Funciones de membresía de salida.

Para verificar el buen funcionamiento del sistema se simuló la red con el siguiente diagrama de bloques en simulink (figura 3.65).

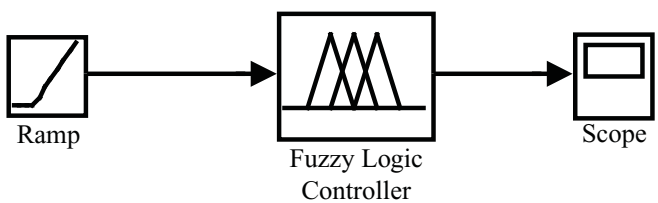


Fig. 3. 65 Diagrama.

En la figura 3.66 se puede observar la salida a la siguiente respuesta.

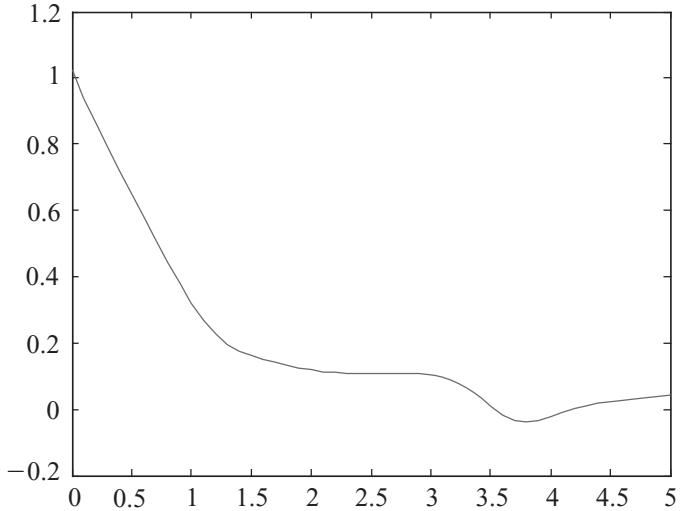


Fig. 3. 66 Respuesta a la salida.

El segundo sistema propuesto es una función tipo *sinc* recorrida cinco posiciones a la derecha, que representa la siguiente función:

$$y = \frac{sen(\pi(x-5))}{\pi(x-5)}$$

En este caso la propusimos de 0 a 10 y al graficarla en MATLAB® vemos la siguiente gráfica (figura 3.67).

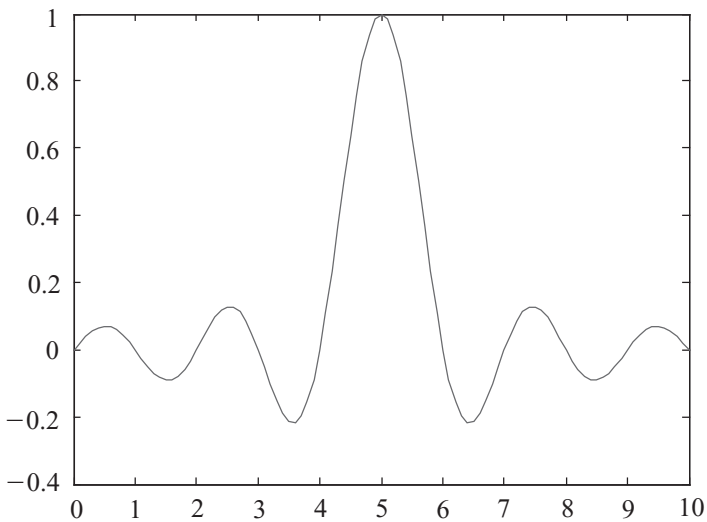


Fig. 3.67 Gráfica en MATLAB®.

Esta función la usamos para entrenar la red ANFIS en la que proponemos cinco funciones de membresía tipo campana para la entrada. Después de aproximadamente 300 iteraciones llegamos a un error de 0.02428 en el que el sistema se estanca.

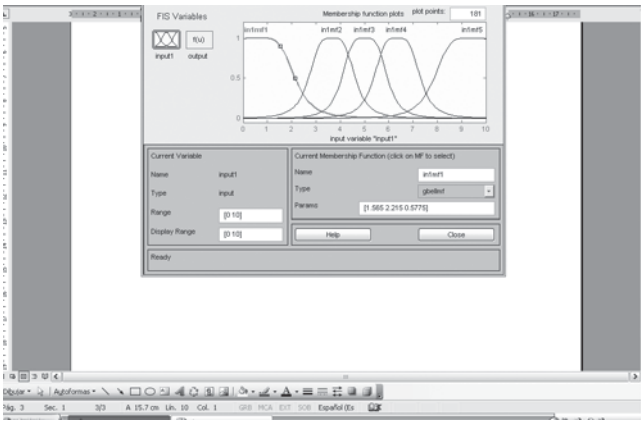


Fig. 3.68 Funciones de membresía de salida. Muestra el estancamiento.

Entonces simulamos esta red para comprobar su funcionamiento. La simulación se hace para 10 segundos, en cada segundo la rampa incrementa en uno su valor comenzando desde cero, el diagrama se muestra en la figura 3.69.

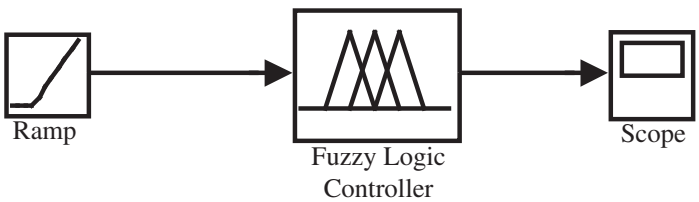


Fig. 3. 69 Diagrama.

Después de simular el sistema, se observa en la figura 3.70 la siguiente respuesta a la salida.

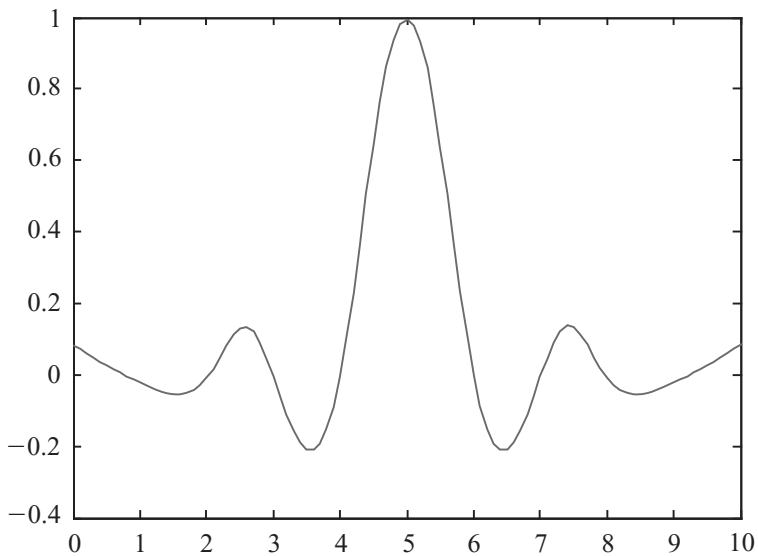


Fig. 3. 70 Respuesta a la salida.

El tercer sistema propuesto es un sistema real del cual se obtienen sus gráficas de entrada y salida con un osciloscopio y se convierten en vectores de MATLAB® para poder entrenar a la red para que simule su comportamiento. El sistema es un circuito RC en serie con valores de 1kΩ y 10μF.

Sus gráficas de entrada y salida son las siguientes (figura 3.71).

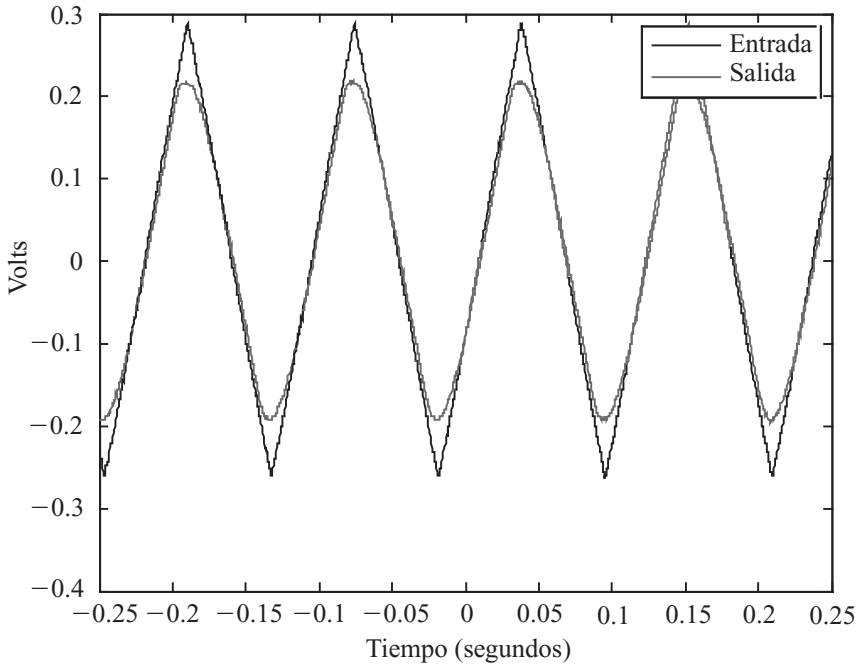


Fig. 3. 71 Gráfica en MATLAB®.

Con estos vectores entrenamos a la red difusa ANFIS en la que proponemos tres funciones de membresía tipo campana. En aproximadamente 100 iteraciones se llegó a un error de 0.1. Las funciones de membresía resultantes después del entrenamiento de la red se muestran en la figura 3.72.

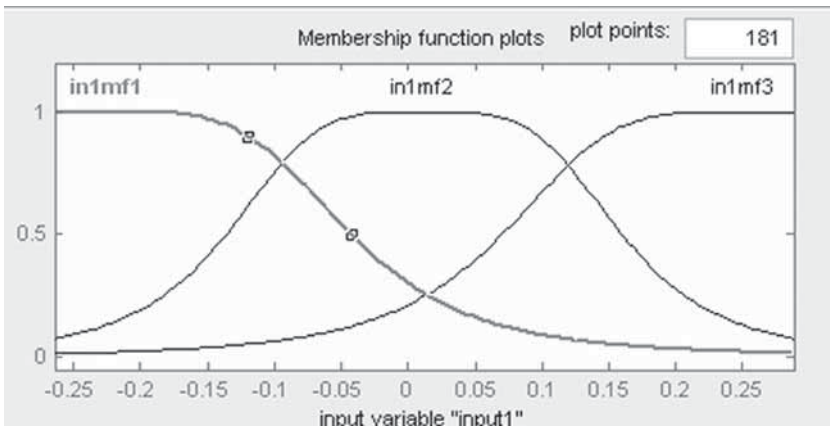


Fig. 3. 72 Funciones de membresía de salida después del entrenamiento.

Entonces en Simulink® simulamos el funcionamiento de nuestra red para la entrada real pero solamente de 0 a 0.25 segundos, véase la figura 3.73.

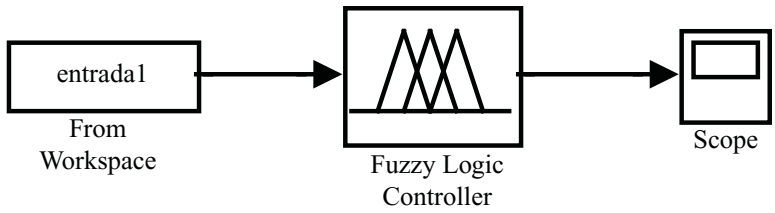


Fig. 3. 73 Diagrama.

La simulación del sistema muestra la siguiente gráfica (figura 3.74).

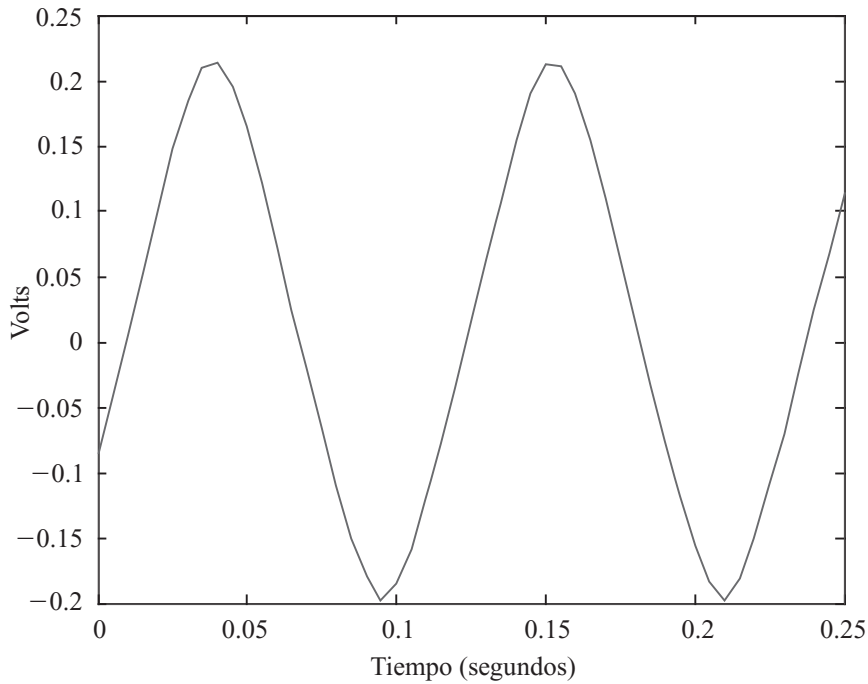


Figura 3.74 Gráfica en MATLAB®.

Empleo de función Genfis1

La función genfis1 de MATLAB® es una herramienta para generar un sistema de inferencia difuso (FIS) en el workspace con base en características que el usuario define. El sistema es una red ANFIS tipo Sugeno. Se definen previamente las entradas, la salida, el número de funciones de membresía para cada entrada, el tipo de funciones de membresía para cada entrada y el tipo de función para la salida, ya sea constante o lineal. Nosotros definimos como entradas el sobretiro y el tiempo de establecimiento de una planta x y la salida es una calificación. Esto es para generar un sistema evaluador del desempeño de una planta. El vector de datos de entrada entonces fue el siguiente:

datos =		
2.0000	20.0000	10.0000
1.6000	15.0000	20.0000
1.5000	13.0000	30.0000
1.3000	11.0000	40.0000
1.0000	10.0000	50.0000
0.9000	9.0000	60.0000
0.8000	8.0000	70.0000
0.7000	7.0000	80.0000
0.5000	6.5000	90.0000
0.3000	6.2500	100.0000

Pedimos que se usaran tres funciones de membresía gaussianas para cada entrada y que las salidas fueran constantes. Después de generar el FIS lo entrenamos para llegar a un error de 0.06994, pero tuvimos que reingresar los datos al editor de ANFIS. Después del entrenamiento, las funciones de membresía se aprecian en las figuras 3.75 y 3.76:

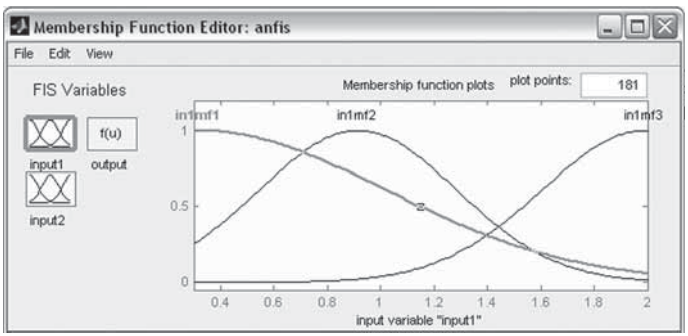


Fig. 3. 75 Funciones de membresía de la entrada sobretiro.

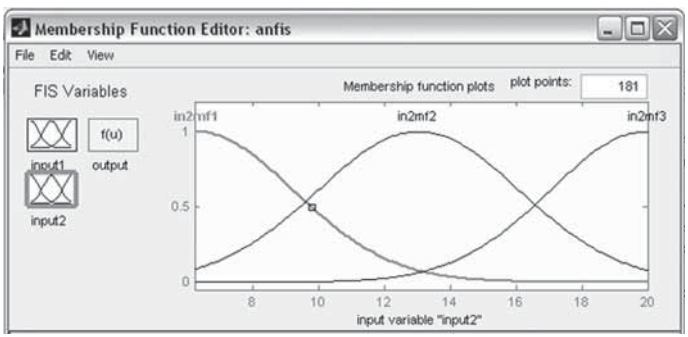


Fig. 3. 76 Funciones de membresía para la entrada tiempo de establecimiento.

Entonces la superficie del sistema queda del siguiente modo (figura 3.77):

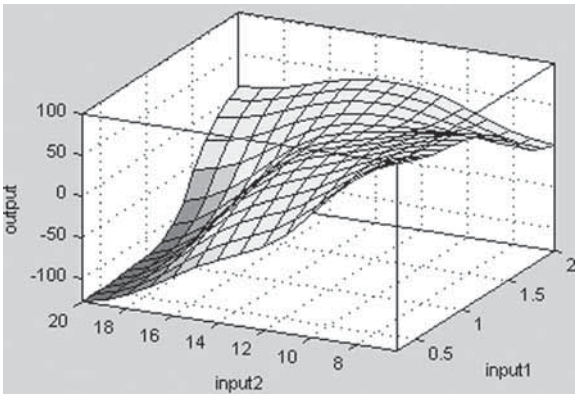


Fig. 3. 77 Superficie del sistema.

EJEMPLO DE UN SISTEMA ANFIS Y DIFUSO PARA EL MODELADO DE MÁQUINAS DE CORRIENTE ALTERNA, EN UN ESQUEMA DE CONTROL VECTORIAL

Introducción

El control vectorial permite tener un control desacoplado del flujo y del par electromagnético, por lo que se puede controlar a la máquina de inducción como si fuera una máquina de corriente continua. Algunos trabajos reportados en la literatura demuestran que se puede tener una respuesta dinámica muy buena. Esta técnica de control exige disponer de un buen modelo del motor que se va a controlar y de los recursos computacionales necesarios para su realización [1].

Los problemas que presenta esta estrategia de control son:

- Requiere tener un PWM.
- Depende de parámetros del motor.
- Presenta un algoritmo complejo.
- Presenta también problemas para controlar las variables con referencias senoidales, porque es difícil filtrar el ruido generado por el ancho de banda de los controladores de histéresis.
- La estructura del control debe realizarse de acuerdo con el tipo de máquina que se tenga, la cual puede ser síncrona (rotor y estator giran al mismo tiempo) y asíncrona (rotor y estator giran en tiempos diferentes).

La lógica difusa es un conjunto de principios matemáticos basados en grados de pertenencia o membresía para modelar información. Por medio de reglas lingüísticas se aproxima una función mediante la relación de las entradas y salidas del sistema y presenta rangos de validación dentro de un intervalo entre 0 y 1, en lugar de asignar valores binarios como en la lógica convencional.

En los sistemas no lineales, las técnicas de modelación e identificación suelen ser difíciles de implementar. Sin embargo, otras técnicas basadas en la lógica difusa son cada vez más utilizadas para el modelado de este tipo de sistemas o procesos. Entre los diferentes métodos de modelación se encuentran los métodos de Takagi-Sugeno.

El modelo mencionado consiste en reglas if-then con antecedentes difusos y funciones matemáticas para las consecuencias.

Un sistema difuso tipo Sugeno característico consiste en una base de reglas, funciones de membresía y un procedimiento de inferencia.

Etapas del control difuso tipo Sugeno

Fusificación

Es la primera etapa, donde se reciben los valores reales de entrada del sistema, los cuales toman un grado de pertenencia en las funciones de membresía; es así como se obtienen las entradas difusas que han de ser evaluadas por las reglas [2].

Evaluación de reglas

Es la segunda etapa; las reglas se componen de oraciones lingüísticas tipo "si-entoces" (if…then) por un método de inferencia de *modus ponens*. Estas reglas son dadas por un experto del sistema o bien por un control convencional. A la salida de esta etapa se tienen funciones matemáticas que representan las salidas.

Desarrollo

Control vectorial

Primero se presenta el algoritmo de control vectorial que se realizó con la herramienta de Simulink de MATLAB®, como muestra la figura 3.78.

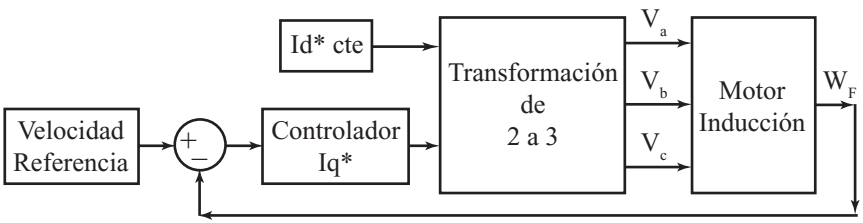


Fig. 3. 78 Diagrama a bloques del control vectorial.

Se construye el diagrama a bloques de un motor de inducción trifásico en un marco de referencia rotatorio (d-q), con el objetivo de simplificar el manejo de las ecuaciones y de esta manera emular un motor de corriente continua.

Para la realización de esta actividad se utilizó las siguientes ecuaciones:

$$V_{qs}^{e} = r_s i_{qs}^{e} + \omega_e \lambda_{ds}^{e} + p\lambda_{qs}^{e} \quad (1)$$

$$V_{ds}^{e} = r_s i_{ds}^{e} - \omega_e \lambda_{qs}^{e} + p\lambda_{ds}^{e} \quad (2)$$

$$V_{qr}^{e} = r_r i_{qr}^{e} + \left(\omega_e - \omega_r\right)\lambda_{dr}^{e} + p\lambda_{qr}^{e} \quad (3)$$

$$V_{dr}^{e} = r_r i_{dr}^{e} - \left(\omega_e - \omega_r\right)\lambda_{qr}^{e} + p\lambda_{dr}^{e} \quad (4)$$

$$\lambda_{qs}^{e} = L_s i_{qs}^{e} + L_m i_{qr}^{e} \quad (5)$$

$$\lambda_{ds}^{e} = L_s i_{qs}^{e} + L_m i_{dr}^{e} \quad (6)$$

$$\lambda_{qr}^{e} = L_r i_{qr}^{e} + L_m i_{qs}^{e} \quad (7)$$

$$\lambda_{dr}^{e} = L_r i_{dr}^{e} + L_m i_{ds}^{e} \quad (8)$$

$$T_e = \frac{3}{2}\frac{P}{2}\left(\lambda_{qr}^{e} i_{dr}^{e} - \lambda_{dr}^{e} i_{qr}^{e}\right) \quad (9)$$

El inversor se simuló con switches ideales para tener un comportamiento similar al de los semiconductores empleados en la realidad.

Se construyó un PWM Banda de Histéresis el cual compara las corrientes reales del motor con las generadas por el algoritmo de control, limitando los valores de corriente con un ancho de banda. La intersección entre ambas genera un tren de pulsos que es enviado al inversor.

Para lograr emular un motor de DC, se necesita tener solamente dos variables independientes para controlar par y flujo, lo cual se logra por medio de transformaciones de dos variables invariantes en el tiempo a dos variables variantes en el tiempo (transformación inversa de Park) y de dos a tres variables variantes en el tiempo (transformada inversa de Clarke).

Para validar el algoritmo, se creó un lazo cerrado de control de velocidad del motor, por lo que se utilizó un controlador tipo PID sintonizado por un método convencional.

Los resultados de la simulación del control vectorial antes mencionado son los siguientes:

En la figura 3.79 se muestra la salida de corriente por parte del motor, la cual presenta picos en el estado transitorio; esto se debe al consumo de corriente en el arranque; dichos picos son de alrededor de 17 amperes, los cuales se presentan durante los primeros 5 segundos de operación, observándose también que conforme avanza el tiempo el consumo de corriente disminuye, estabilizándose entre los 5 amperes.

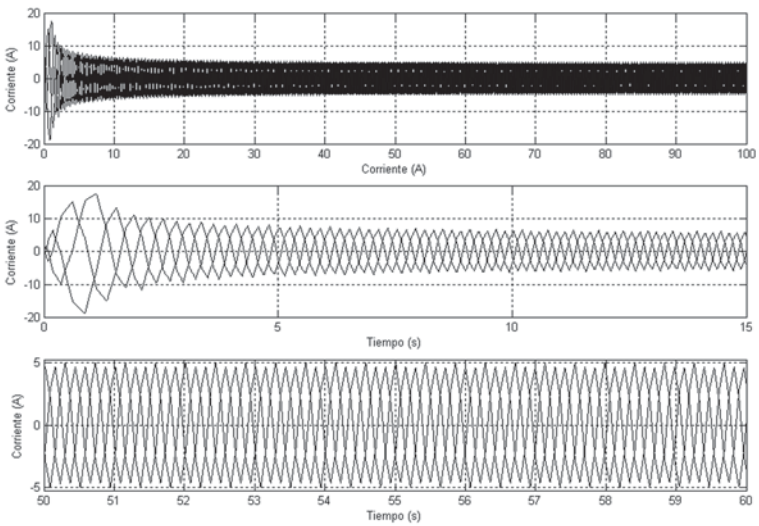


Fig. 3. 79 a) Corrientes de salida del motor de inducción trifásico; b) y c) acercamientos.

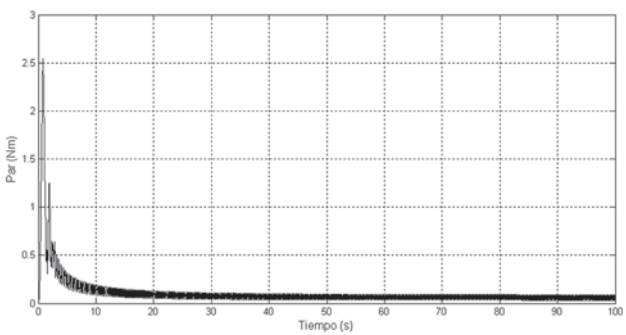


Fig. 3. 80 Par electromagnético.

En el modelo se trabajó sin carga (figura 3.80), por lo que el par electromagnético llega a la referencia presentando un pico pequeño, el cual es común en el arranque de un motor de inducción.

La figura 3.81 muestra la velocidad que presenta el motor: representa un sistema un poco lento en su respuesta (70 s), sin embargo lo que se buscó fue tener una validación por medio de simulación para comprobar que el algoritmo es correcto.

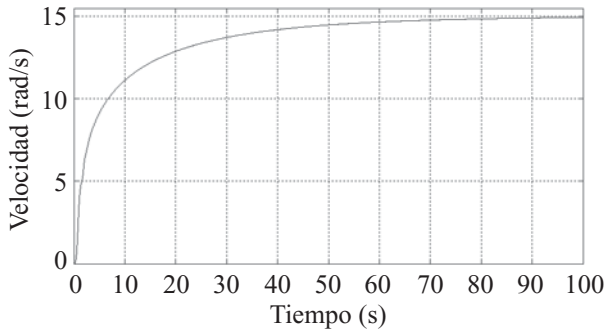


Fig. 3. 5.81 Velocidad.

La figura 3.82 muestra los pulsos a la salida del inversor.

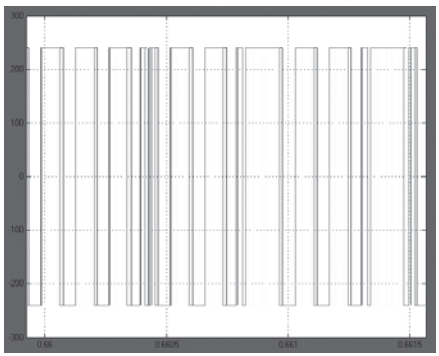


Fig. 3. 82 Señales de salida del inversor.

La banda de histéresis se muestra en la figura 3.83: la señal con mayor número de armónicos es la señal de salida del motor, es decir, las corrientes $i_{a\text{-}b\text{-}c}$, y la señal con menor distorsión es la generada por la transformada inversa de Clarke.

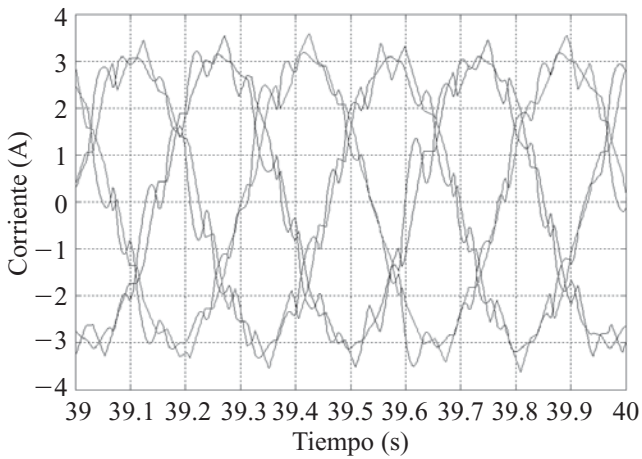


Fig. 3. 83 PWM Banda de histéresis.

Modelo difuso del motor de inducción

Se realizó el modelo difuso de un motor de inducción utilizando como datos experimentales la señal de voltaje de entrada y la señal de corriente de la salida. Se graficó los datos de entrada contra los de salida, obteniéndose la figura 3.84.

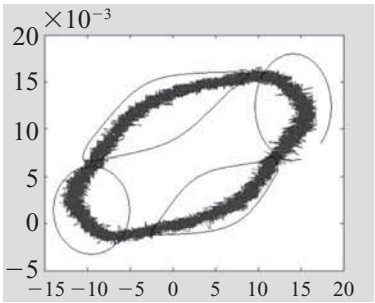


Fig. 3. 84 Gráfica voltaje vs. corriente.

La gráfica anterior se seccionó en 4 partes para poder linealizarla. Así, cada sección linealizada proporciona un polinomio que posteriormente servirá de salida para el sistema difuso tipo Sugeno:

Fusificación

Para la fusificación se utilizó dos entradas: X y yn-1. Para la entrada X se seleccionó 7 funciones de membresía (FM) que van desde Negativo Grande (NG), Negativo Mediano (NM), Negativo Pequeño (NP), Cero (Z), Positivo Pequeño (PP), Positivo Mediano (PM) y Positivo Grande (PG). Las FM son de tipo triangular en un universo de discurso (UD) de -15 a 15, ya que fue el rango de los valores de voltaje con los que se trabajó el motor. La distribución de las FM se muestra en la figura 3.85.

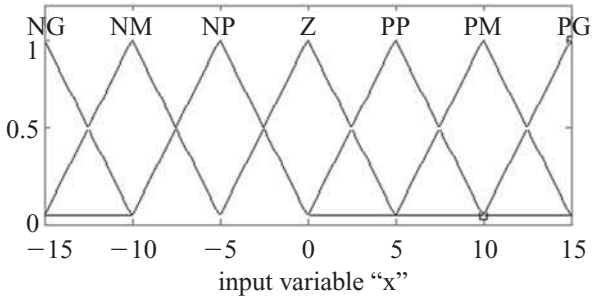


Fig. 3. 85 Gráfica voltaje vs. corriente.

Para la entrada Yn-1 se seleccionó 4 FM: Yn1, Yn2, Yn3 y Yn4 que toman los rangos que se obtuvieron de las secciones 1 a 4. Las FM son de tipo triangular en un UD de -0.003 a 0.018, ya que fue el rango de los valores de corriente que entregó a la salida el motor. La distribución de las FM se muestra en la figura 3.86:

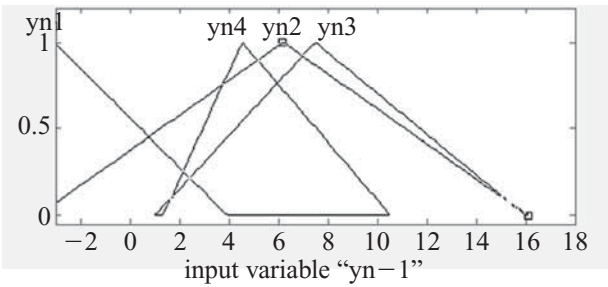


Figura 3.86 Gráfica voltaje vs. corriente.

Edición de reglas

Para la edición de las reglas lingüísticas se utilizó un modo de inferencia Tollens donde a partir de los antecedentes se tienen las consecuencias. Las 18 reglas se muestran a continuación:

- Si x es NG y yn-1 es Y1, entonces yn es Y1
- Si x es NM y yn-1 es Y2, entonces yn es Y1
- Si x es NP y yn-1 es Y4, entonces yn es Y1
- Si x es NM y yn-1 es Y2, entonces yn es Y2
- Si x es NM y yn-1 es Y4, entonces yn es Y4
- Si x es NP y yn-1 es Y2, entonces yn es Y2
- Si x es NP y yn-1 es Y4, entonces yn es Y4
- Si x es Z y yn-1 es Y2, entonces yn es Y2
- Si x es Z y yn-1 es Y4, entonces yn es Y4
- Si x es PP y yn-1 es Y2, entonces yn es Y2
- Si x es PP y yn-1 es Y4, entonces yn es Y4
- Si x es PM y yn-1 es Y2, entonces yn es Y2
- Si x es PM y yn-1 es Y4, entonces yn es Y4
- Si x es PG y yn-1 es Y2, entonces yn es Y3
- Si x es PG y yn-1 es Y3, entonces yn es Y3
- Si x es PG y yn-1 es Y4, entonces yn es Y3

Los polinomios que se obtuvo son de segundo orden como se muestra a continuación:
Polinomio 1:

$$Y1 = 4E-05x^2 - 0.0002x - 0.0056$$

Polinomio 2:

$$Y2 = 4E-05x^2 + 0.0005x + 0.0139$$

Polinomio 3:

$$Y3 = 4E-06x^2 - 0.001x + 0.0271$$

Polinomio 4:

$$Y4 = 3E-05x^2 + 0.0002x - 0.0008$$

Las gráficas que se obtienen de la simulación de este modelo difuso son las siguientes (figuras 3.87 y 3.88):

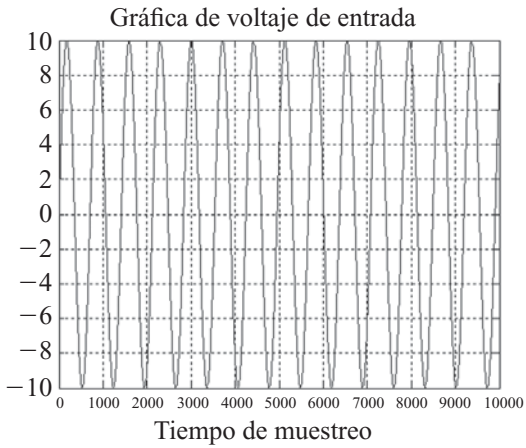


Fig. 3. 87 Voltaje de entrada.

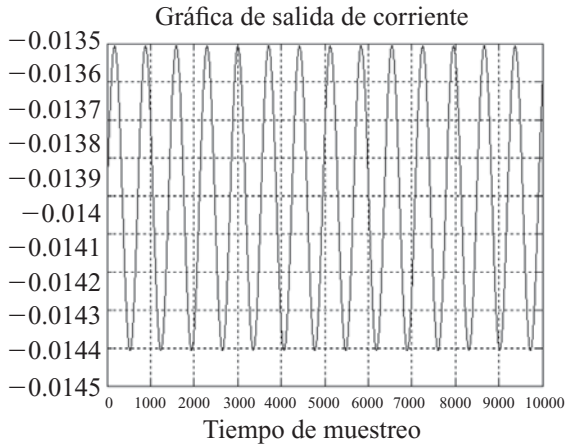


Fig. 3. 88 Corriente de salida.

La gráfica de salida real del motor de inducción se muestra en la figura 3.89.

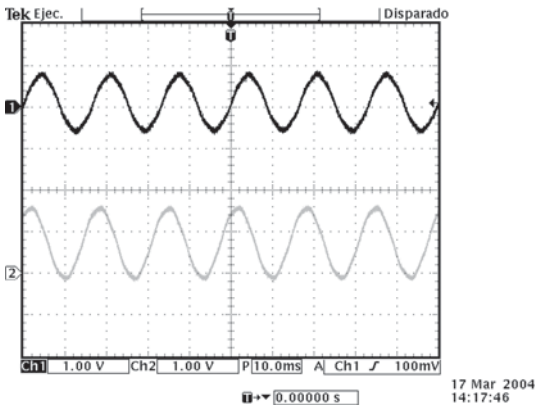


Fig. 3. 89 Parte superior (voltaje de entrada); parte inferior (corriente de salida).

Al comparar las señales experimentales y de simulación, se observa que la señal de corriente experimental está dentro del rango de los 28 mA, mientras que la señal en simulación se encuentra en el rango de 27 mA, por lo que el modelo es válido (el voltaje de entrada es de 100 V).

Modelo ANFIS del motor de inducción

El modelo de motor difuso presentado anteriormente entregó resultados satisfactorios, sin embargo se sintonizó de manera experimental. En esta parte del documento se muestra un modelo difuso sintonizado por redes neurales artificiales utilizando el editor ANFIS de MATLAB® (Adaptive Neural based Fuzzy Inference System).

Tomando datos experimentales de entrada de voltaje (V) y datos de salida de corriente (I) del motor a diferentes amplitudes, se entrenó con ANFIS para obtener el modelo. La finalidad del entrenamiento es obtener un modelo difuso sintonizado con las ventaja adaptiva de las redes neurales artificiales, buscando una convergencia de error a cero, por lo cual se entrenó con 10 000 iteraciones para lograr dicha convergencia. La figura 3.90 muestra el entrenamiento que se realizó, donde la gráfica en tono más

oscuro (azul) representa los datos experimentales de salida del sistema y la gráfica en tono más claro (rojo) representa la respuesta del entrenamiento.

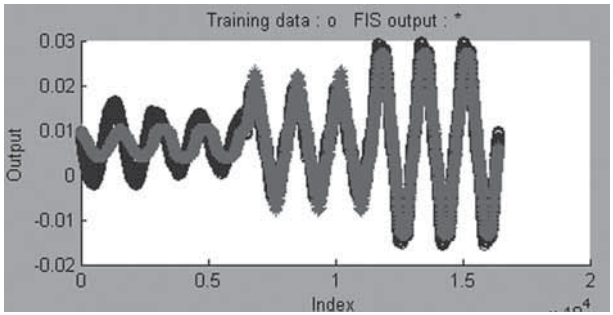


Fig. 3. 90 Entrenamiento del ANFIS.

En la figura 3.91 se tiene la red neuronal artificial que se genera para sintonizar el sistema difuso.

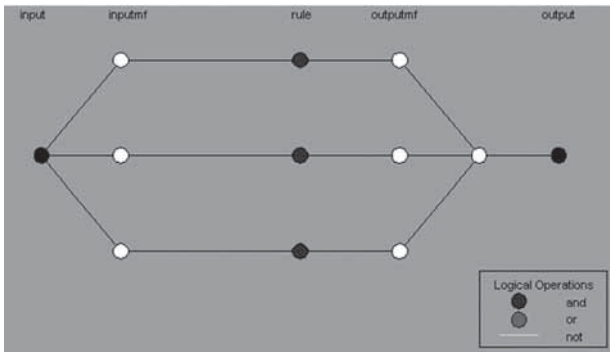


Fig. 3. 91 Red neuronal artificial.

Empleando los mismos datos experimentales anteriores, usados para el modelo difuso del motor, se obtuvo los siguientes resultados de simulación con el sistema sintonizado con ANFIS (figuras 3.92 y 3.93):

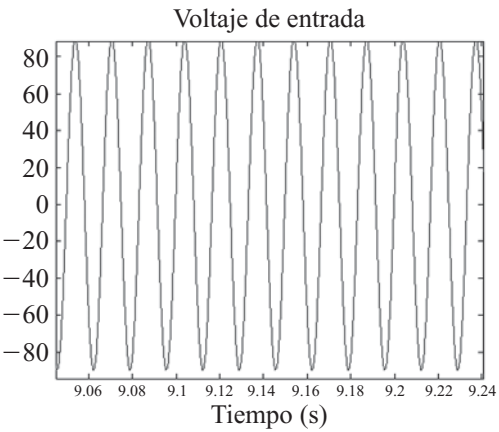


Figura 3.92 Voltaje de entrada.

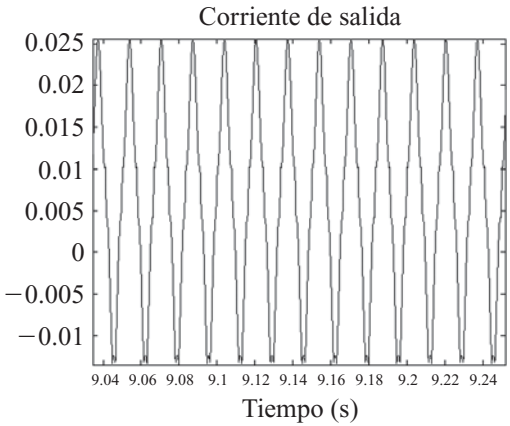


Fig. 3. 93 Corriente de salida.

En la figura 3.93 se observa que se tiene una amplitud de corriente de alrededor de 25 mA como dato real, mientras que en la simulación la salida de corriente que entrega el modelo de motor bajo el entrenamiento utilizando ANFIS es de 25 mA, con lo cual se valida que la corriente de simulación es correcta y que el entrenamiento fue el adecuado ya que se encuentra dentro del rango de amplitud.

Control vectorial difuso

Se realizó un control vectorial difuso, es decir que se sustituyó todo el diagrama del control vectorial que se observa en la figura 3.78 por un bloque de lógica difusa como se muestra en la figura 3.94.

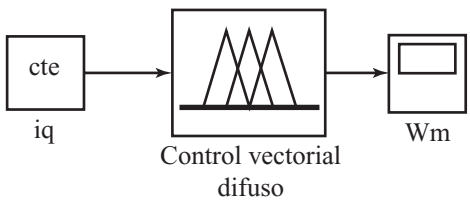


Fig. 3. 94 Diagrama a bloques del control vectorial difuso para un motor de inducción.

Para poder realizar el control vectorial difuso, se obtuvo datos de simulación de la entrada iq y la salida Wm (ver figura 3.95) manteniendo como un valor constante la entrada id que es la variable que controla el flujo, a la cual se le asignó el valor 1.

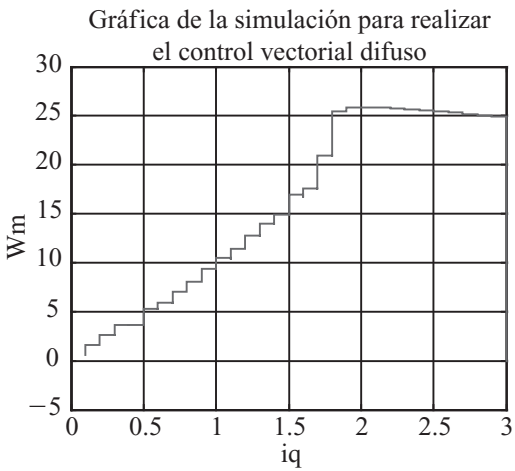


Fig. 3. 95 Se tomó datos de la entrada contra la salida para encontrar los rangos de las funciones de membresía.

Las nueve funciones de membresía que se utilizó son de tipo triangular en un universo de discurso entre 0.5 y 3 con la distribución que muestra la figura 3.96:

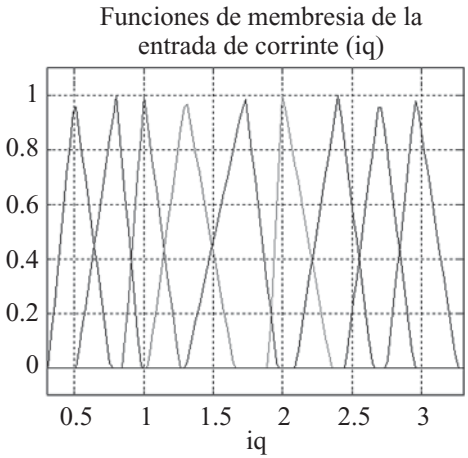


Fig. 3. 96 Universo de discurso del control vectorial difuso con nueve funciones de membresía de tipo triangular.

Las reglas lingüísticas que se propuso son las siguientes:

- Si iq es mf1, entonces Wm es y1
- Si iq es mf2, entonces Wm es y2
- Si iq es mf3, entonces Wm es y3
- Si iq es mf4, entonces Wm es y4
- Si iq es mf5, entonces Wm es y5
- Si iq es mf6, entonces Wm es y6
- Si iq es mf7, entonces Wm es y7
- Si iq es mf8, entonces Wm es y8
- Si iq es mf9, entonces Wm es y9

Para las nueve salidas se utilizó funciones lineales ya que se usó un sistema difuso tipo Sugeno, siendo éstas:

- Y1= 0+3.9
- Y2=0+7.55
- Y3=0+10
- Y4=0+13.2
- Y5=0+17.1
- Y6=0+25.2
- Y7=0+25
- Y8=0+24.5
- Y9=0+24.15

Empleando este sistema difuso se puede realizar un mapeo de la entrada de la corriente en el eje q contra la velocidad del rotor; para establecer esta relación no lineal es necesario el empleo del sistema difuso. Se puede realizar un barrrido de los valores de corriente para encontrar los valores de la velocidad como se puede ver en la figura 3.97.

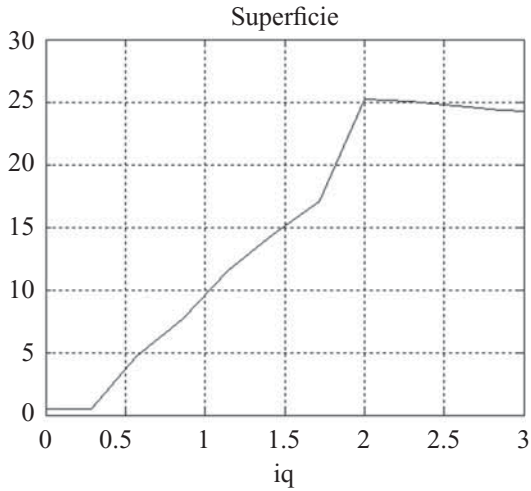


Fig. 3. 97 Gráfica de superficie que se genera a partir del sistema difuso.

Para comprobar que nuestro sistema difuso fuera una buena aproximación, se asignó valores de iq tanto a la simulación del control vectorial como al bloque difuso y se comparó los valores, encontrando un error de 0.005332%.

Tomando en cuenta los resultados anteriores se puede decir lo siguiente: se realizó un modelo del motor de inducción trifásico utilizando lógica difusa, el cual al ser probado en simulación en varios valores en un rango de 0 a 100 volts, genera valores de corriente en un rango de 0 a 27mA, que comparados con los valores experimentales son correctos, ya que se encuentran dentro del rango de consumo de corriente del motor. Con esto validamos en simulación el modelo difuso obtenido del motor de inducción sintonizado de manera heurística.

En el resultado obtenido en el modelo del motor utilizando ANFIS, se observa que el entrenamiento para la red tuvo una tendencia del error a cero, el cual bajo un entrenamiento de 10 000 iteraciones, llegó a un valor de 0.0032603, lo cual nos muestra que el sistema generado se aproxima a las señales obtenidas experimentalmente. Esto es, que si se tiene un voltaje de entrada de amplitud de 160 volts, se genera una corriente de 35 mA, datos que al ser comparados con los experimentales, están dentro del rango de operación.

Se realizó un control vectorial difuso tipo Sugeno con salidas de primer orden, en el cual se hizo la sustitución de todo el algoritmo de control, incluyendo el motor, por un solo bloque de lógica difusa. Se sintonizó nueve funciones de membresía de tipo triangular con base en los datos de entrada contra salida, de manera que se obtuvo un promedio de error de 0.005332%, lo cual presenta una opción muy buena de solución.

APROXIMADOR NEURO-DIFUSO CON CLUSTERS Y REDES NEURALES TRIGONOMÉTRICAS

Este aproximador tiene la ventaja de poder entrenarse en un tiempo corto; gracias a esta característica se puede implementar en un sistema en línea.

El estudio de este sistema se puede iniciar con el teorema de Kolmogorov, que menciona que una capa oculta es suficiente para una aproximación universal. Kolmogorov mostró que una función continua de muchas variables se puede representar exactamente por la superposición de funciones continuas unidimensionales de las variables de entrada originales. Cualquier mapeo de una función de n-dimensiones a la entrada (n>=2) hacia una salida de m-dimensiones puede ser implementada exactamente por una red (figura 3.98) con una capa oculta para ϕ: $I^n \rightarrow R^m$, $\phi(x) = y$ donde I es el intervalo cerrado (0,1].

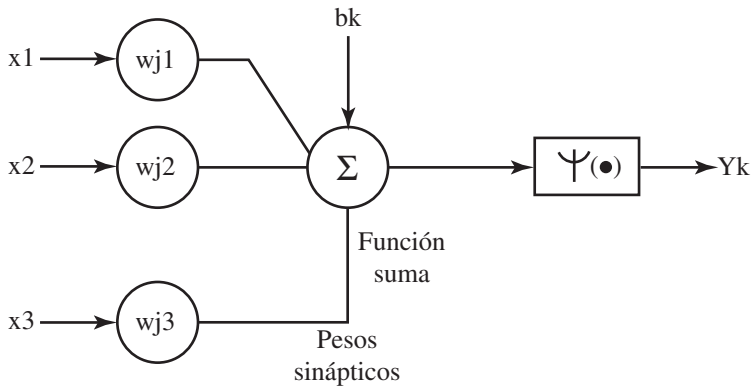


Fig. 3. 98 Modelo de una neurona.

$$Y_k = \Psi(u_k + b_k)$$

$$Z_k = \sum_{j=1}^{n} \lambda^n \Psi(x_j + \varepsilon k) + k$$

$$Y_i = \sum_{k=1}^{2n+1} g_i(Z_k)$$

$$\lambda = cte$$

$$\Psi = función\_continua$$

$$g_i = nodo\_de\_salida\_real\_y\_continua$$

Se puede usar una serie trigonométrica de Fourier como una función de activación. Por lo tanto, el método propuesto por el Dr. Pedro Ponce a través de funciones trigonométricas a la salida y que se puedan adaptar a la forma de las funciones de membresía de la entrada, se puede redefinir como se muestra en la figura 3.99.

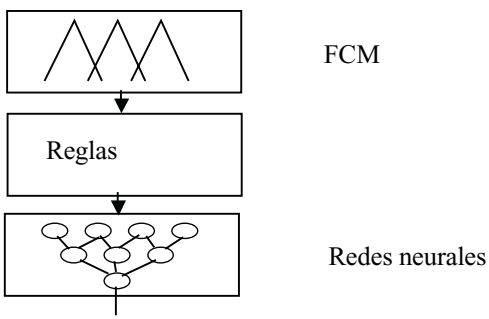


Fig. 3. 99 Método neuro–difuso adaptable.

Entrenamiento de retropropagación

Función de energía (error)

$$E = \frac{1}{2}\sum_p \sum_i (d_{pi} - Y_{pi})^2, E = \sum_p E_p$$

Donde

$$E_p = \frac{1}{2}\sum_i (d_{pi} - Y_{pi})^2$$

$$\frac{\partial E}{\lambda_i} = \sum_p \frac{\partial E_p}{\partial \lambda_i}$$

$$\frac{\partial E_p}{\partial \lambda_i} = \sum_k \frac{\partial E_p}{\partial a_k}\frac{\partial a_k}{\partial \lambda_i}$$

$$\delta i = \frac{\partial Ep}{\partial ai} = \frac{\partial E_p}{\partial Y_i}\frac{\partial Y_i}{\partial a_i}$$

$$\delta k = -(d_{pk} - Y_{pk})\delta$$

De igual forma que el método Sugeno con clusters difusos, este controlador estará basado en una curva de operación, sólo que los valores de los casos son diferentes. El método consta de tres etapas: defusificación, evaluación de reglas y evaluación con redes neurales trigonométricas a la salida (figura 3.100).

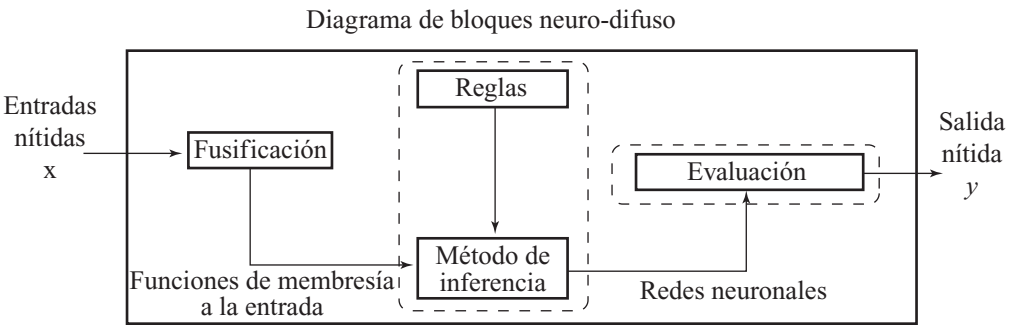


Fig. 3. 100 Diagrama a bloques del método neuro-difuso.

De manera general se puede decir que si los grados de pertenencia se evalúan con el valor de salida de cada cluster, donde dicho valor de salida estará dado por una red neuronal evaluada con los valores de entrada. Recordemos que cualquier función es posible aproximarla con series de Fourier [25], por lo que para acercarnos a la curva de operación se realizó una aproximación con la ecuación (47).

$$f(x,y,z) = \lambda_1 + \sum_{n=1}^{\infty} \lambda_2 \sin(nx + \lambda_5) + \sum_{n=1}^{\infty} \lambda_3 \sin(ny + \lambda_6) + \sum_{n=1}^{\infty} \lambda_4 \sin(nz + \lambda_7) \quad (3.47)$$

Dicha ecuación será representada por una red neuronal, donde x, y, z son las entradas y donde $\lambda_l = (a_0, C, D, E, \theta, \alpha, \beta)$ estarán dados por los pesos de las redes neurales. Dichos pesos se obtuvieron

entrenando a las redes neurales, con la retropropagación del error (20) basado en el método del gradiente.

$$w_{ij} = w_{ij} - \mu \cdot \frac{\partial E}{\partial w_{ij}} \tag{3.48}$$

Donde w_{ij} es el peso de la red, μ es el coeficiente de aprendizaje, mientras que el otro término es la derivada parcial del error con respecto a la derivada del peso anterior. Se usó para aproximar la función 10 componentes o armónicos. La figura 35 muestra el diagrama general de la red neuronal utilizada para el control respectivo.

Redes neurales basadas en Fourier

Sin duda, una de las ventajas más contundentes de las series de Fourier es que puede modelar cualquier señal periódica como una suma de funciones trigonométricas. De acuerdo con el teorema de Fourier, podemos modelar cualquier señal de periodo 2*l* de acuerdo con las ecuaciones:

$$para\ f(x) = f(x+2l)$$

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}\left(a_n \cos(n\omega x) + b_n sen(n\omega x)\right), \qquad \omega = \frac{\pi}{l}$$

donde,

$$a_0 = \frac{1}{l}\int_0^{2l} f(x)dx$$

$$a_n = \frac{1}{l}\int_0^{2l} f(x)\cos(n\omega x)dx$$

$$b_n = \frac{1}{l}\int_0^{2l} f(x)sen(n\omega x)dx$$

En este sentido, la expresión de Fourier tiene en sí la forma de la expresión de una red neuronal expresada antes; también se trata de una suma de funciones multiplicadas por un coeficiente más una constante.

Por ende, se podría construir una red neuronal basada en esta expresión de acuerdo con el diagrama de la figura 3.101.

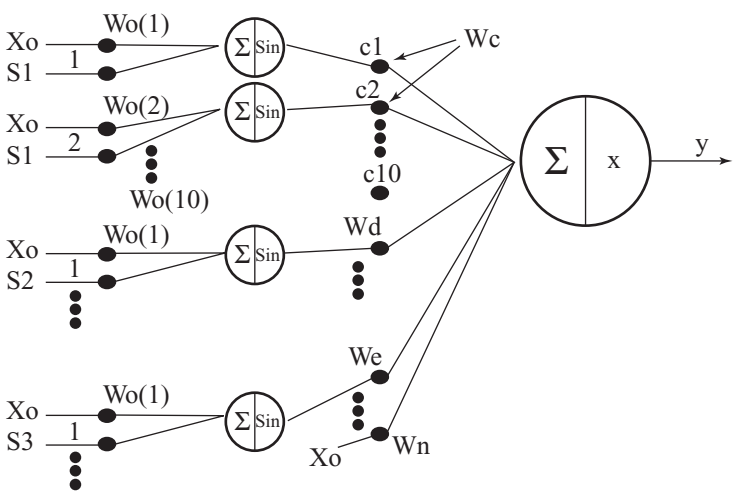


Fig. 3. 101 Diagrama de una red neuronal basada en Fourier.

La topología propuesta tendría tan sólo dos capas de neuronas: en la primera capa habría un conjunto determinado de neuronas que tendrían una función de activación trigonométrica y un peso que determinaría la frecuencia de la misma; en la segunda capa se efectuaría la suma de cada una de estas funciones multiplicada por su coeficiente específico más la constante final.

De inmediato se puede observar las ventajas de una topología antes descrita: se trata de una topología de fácil implementación y bajo costo; además, su ventaja esencial es que es posible calcular los pesos de la red neuronal de un modo analítico y se tiene un control del grado de exactitud que se alcanza conforme se aumenta más neuronas o armónicos dentro de la serie de Fourier.

Cálculo de la función de la red neuronal basada en Fourier

En la búsqueda de un "aprendizaje" de la red, se plantean los datos de entrada y de salida deseados. Podemos representarlos como una función donde para un cierto dominio se tiene un condominio objetivo. Normalmente, el dominio que nos interesa modelar está claramente acotado pues las variables que se desea modelar lo suelen estar. Debido a lo anterior, nosotros sólo estamos interesados en el comportamiento de la señal dentro del dominio, y si éste es acotado podemos suponer que es una señal periódica con un periodo del doble del rango del dominio. Con el propósito de explicar esto en forma más clara, supóngase que se quiere que una red neuronal de una sola salida y una sola entrada tome los valores de la figura 3.102.

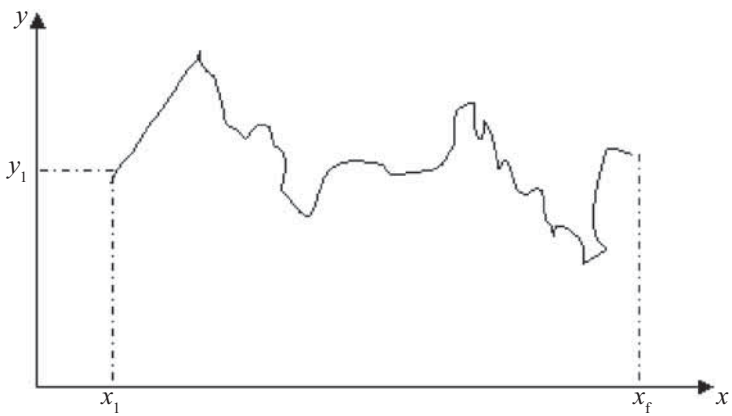


Fig. 3. 102 Salida que queremos modele la red neuronal.

Los valores objetivo tienen un dominio bien acotado; podemos calcular el rango, l del modo $l = X_f - X_i$. Además es posible construir una señal periódica con base en la función que queremos modelar y tomar ésta como medio periodo de la señal. De este modo podemos también hacer que nuestra función sea par o impar. Asimismo es factible hacer los cambios de variable $X_S = X - X_i$ y $Y = Y - Y_i$ para así lograr que nuestra señal inicie en el origen. La construcción de una señal periódica impar o par queda ejemplificada en las figuras 3.103 y 3.104.

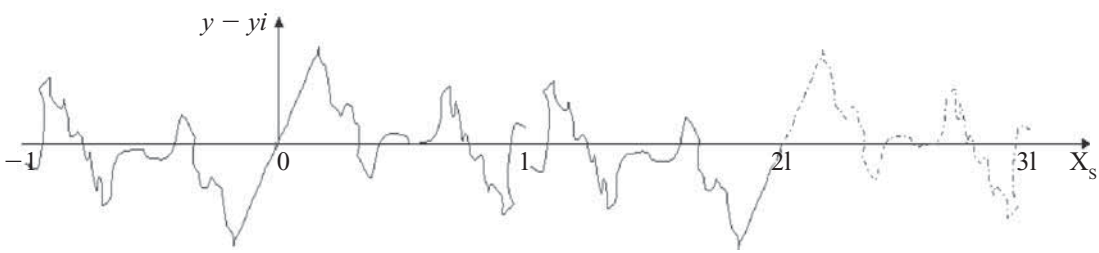


Fig. 3. 103 Construcción de una señal periódica impar.

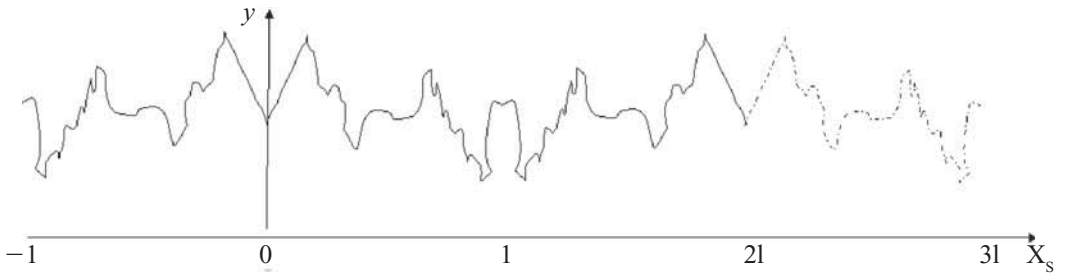


Fig. 3. 104 Construcción de una señal periódica par.

De acuerdo con lo que se observa en las figuras 3.103 y 3.104, se puede observar la ventaja de modelar la señal periódica como una señal del tipo par, pues de este modo la señal no presenta discontinuidades que podrán requerir de una gran cantidad de armónicos para ser aproximadas. A continuación podemos usar las series de Fourier para modelar nuestra señal periódica y con ello implementar los coeficientes trigonométricos de mayor peso dentro de la red neuronal.

Dado que podemos controlar que la señal periódica sea par o impar, podemos con esto reducir la serie de Fourier. De acuerdo con lo que escojamos, encontraremos:

Señal impar	Señal par
$y_s = \sum_{n=1}^{p} b_n sen(n\omega x_s)$	$y_s = \frac{1}{2}a_0 + \sum_{n=1}^{p} a_n \cos(n\omega x_s)$
Además,	Además,
$b_n = \frac{2}{l}\int_0^1 f(x) sen(n\omega x) dx$	$a_n = \frac{2}{l}\int_0^1 f(x) \cos(n\omega x) dx$
Una vez estimados los coeficientes b_n, es posible expresar la función total de la red neuronal como:	Una vez estimados los coeficientes a_n, es posible expresar la función total de la red neuronal como:
$y = y_i + \sum_{n=1}^{p} b_n sen\left(n\omega (x - x_i)\right)$	$y = \frac{1}{2}a_0 + \sum_{n=1}^{p} a_n \cos\left(n\omega (x - x_i)\right)$

Establecimiento de los pesos

El método tradicional de las redes neurales para establecer los pesos en los nodos entre neuronas, es el de darles valores aleatorios y esperar a que converjan por el método de gradiente descendente. Gracias a la topología estudiada es ahora posible estimar con determinismo el valor que deberán de tener estos coeficientes.

Debido a que en muchas ocasiones la unión entre elementos del dominio y condominio no está establecida como función, sino que se tiene tan sólo una relación de correspondencia, es posible que no se pueda realizar la integral analítica para el cálculo de los coeficientes.

Se pueden determinar los coeficientes de la serie a través del método de mínimos cuadrados, y observar que la estimación de los coeficientes de la serie puede ser calculada como una aproximación lineal que puede reducirse a un número determinado de coeficientes y cuya solución reside en un problema de álgebra lineal del tipo Ax = b.

Determinación de los pesos por mínimos cuadrados

Se parte de que modelaremos nuestra función de la forma $g(x) = b_0 + \sum_{n=1}^{p} b_n sen(n\omega x)$ y estamos interesados en encontrar los coeficientes b_n que aproximen lo mejor posible el conjunto de puntos (x_i, y_i). El uso de senos en la función de aproximación hace referencia a que se está modelando la señal como una función impar. Si se desea modelar la función como función impar, basta con sustituir los senos por cosenos.

Para el método de mínimos cuadrados definimos la funcional F en la que buscaremos los coeficientes que tienda a su mínimo.

$$F(b_0, b_1, ..., b_p) = \sum_{i=1}^{m} \omega_i \left[g(b_0, b_1, ..., b_p; x_i) - y_i \right]^2$$

Podemos encontrar su mínimo de obtener su derivada e igualarla a cero

$$\frac{\partial F}{b_j} = \sum_{i=1}^{m} \omega_i \left[g(b_0, b_1, ..., b_p; x_i) - y_i \right] \frac{\partial g}{b_j} = 0$$

De la expresión anterior encontramos: 0

$$\frac{\partial F}{b_j} = \sum_{i=1}^{m} \omega_i \left[g(b_0, b_1, ..., b_p; x_i) - y_i \right] sen(j\omega x) = 0 \qquad \text{para } j \neq 0$$

$$\sum_{i=1}^{m} \omega_i g(b_0, b_1, ..., b_p; x_i) sen(j\omega x) = \sum_{i=1}^{m} \omega_i y_i sen(j\omega x) \text{ para } j \neq 0$$

Por lo anterior, podemos expresar el sistema de p ecuaciones de la forma Ax = b y se puede encontrar una aproximación a la solución.

$$\begin{pmatrix} \sum \omega_i & \sum \omega_i sen(\omega x_i) & \sum \omega_i sen(2\omega x_i) & \cdots & \sum \omega_i sen(p\omega x_i) \\ \sum \omega_i sen(\omega x_i) & \sum \omega_i sen^2(\omega x_i) & \sum \omega_i sen(2\omega x_i) sen(\omega x_i) & \cdots & \sum \omega_i sen(p\omega x_i) sen(\omega x_i) \\ \sum \omega_i sen(2\omega x_i) & \sum \omega_i sen(\omega x_i) sen(2\omega x_i) & \sum \omega_i sen^2(2\omega x_i) & \ddots & \sum \omega_i sen(p\omega x_i) sen(2\omega x_i) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \sum \omega_i sen(p\omega x_i) & \sum \omega_i sen(\omega_i x) sen(p\omega_i x) & \sum \omega_i sen(2\omega x_i) sen(p\omega x_i) & \cdots & \sum \omega_i sen^2(p\omega x_i) \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix} = \begin{pmatrix} \sum \omega_i y_i \\ \sum \omega_i y_i sen(\omega x) \\ \sum \omega_i y_i sen(2\omega x) \\ \vdots \\ \sum \omega_i y_i sen(p\omega x) \end{pmatrix}$$

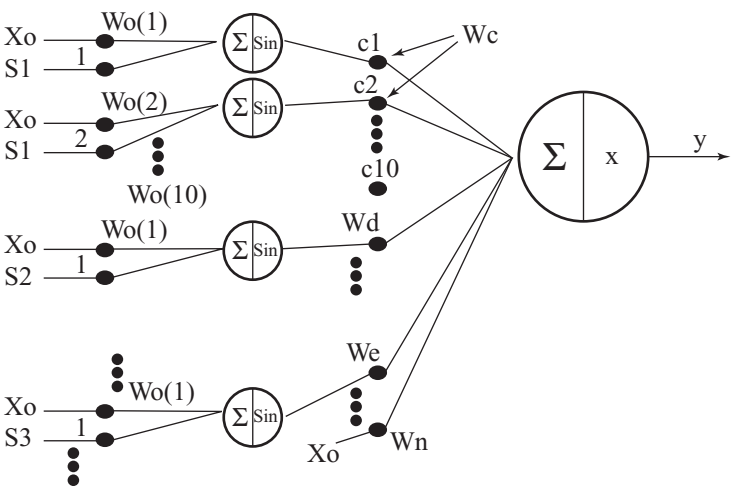


Fig. 3. 105 Diagrama general de la red neuronal para el sistema neuro-difuso.

Una vez entrenada la red con los patrones de la curva de operación, finalmente la salida estará dada por (3.49).

$$Salida = \frac{\mu_{x1} * x_{x1} + \mu_{x2} * x_{x2} + \mu_{x3} * x_{x3}}{\mu_{x1} + \mu_{x2} + \mu_{x3}} \tag{3.49}$$

Donde μxi es el grado de pertenencia del elemento evaluado al cluster i y x_{xi} es el valor de salida de la red neuronal correspondiente al cluster i evaluada con las entradas del punto x. Las ecuaciones de salida de cada cluster fueron obtenidas con redes neurales con (3.47).

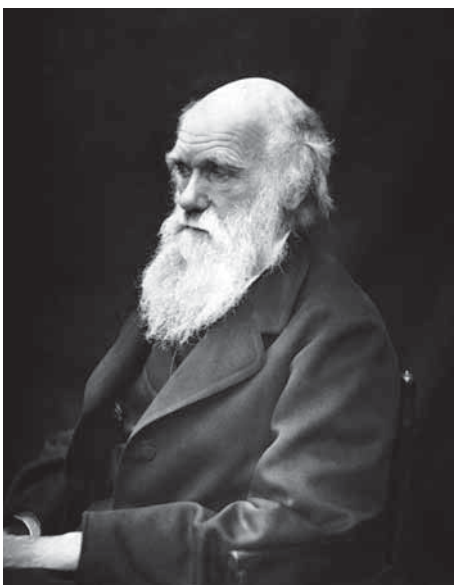
4

ALGORITMOS GENÉTICOS

CHARLES DARWIN Y LA TEORÍA DE LA EVOLUCIÓN

Científico británico que sentó las bases de la moderna teoría evolutiva, al plantear el concepto de que todas las formas de vida se han desarrollado a lo largo de un proceso de selección natural que involucra millones de años. Su trabajo tiene una influencia decisiva en varias disciplinas científicas, la religión y el pensamiento moderno en general.

En 1831 inició una expedición científica alrededor del mundo en el barco de reconocimiento HMS Beagle, que duró cinco durante los cuales recopiló una cantidad inmensa de datos y anotaciones sobre geografía, geología, botánica y zoología, así como un gran número de muestras de las que obtendría información suficiente para escribir varios libros. En 1838 ya había perfilado su teoría de la evolución y



Darwin, Charles Robert (1809 - 1882)

del mecanismo de la selección natural, que presentó 20 años después, el 1 julio de 1858 en la Linnean Society de Londres. El 24 de noviembre de 1859 se puso a la venta la primera edición de *El origen de las especies por selección natural*, la cual se agotó ese mismo día.

La teoría de Darwin sostiene que la variación entre las especies ocurre al azar y que la supervivencia o extinción de cada organismo está determinado por la capacidad de dicho organismo a adaptarse a su medio ambiente, con los siguientes puntos clave:

- Las especies tienen gran fertilidad.
- Las poblaciones permanecen aproximadamente del mismo tamaño, con fluctuaciones muy pequeñas.
- Los alimentos son limitados, pero relativamente constantes la mayor parte del tiempo.
- En ciertas condiciones habrá lucha por la supervivencia entre individuos.
- En la reproducción sexual, dos individuos no serán idénticos.
- La variación es extensa, muchas de estas variaciones serán heredadas.

De esto se desprende que en un mundo de poblaciones estables, donde cada individuo debe luchar para sobrevivir, aquellos con las mejores características tendrán mayores probabilidades de sobrevivir, cuyos rasgos y características ventajosos serán pasados a sus descendientes y heredadas por las siguientes generaciones, tornándose predominantes entre la población a través del tiempo. Este proceso se conoce como selección natural, la cual si se le lleva lo suficientemente lejos quizá realice cambios en la población y eventualmente dará lugar a nuevas especies.

Elementos en la teoría de la evolución:

- Variación: Existe una variación en cada población.
- Competición: Los organismos compiten por recursos limitados.
- Procreación: Los organismos procrean más de lo que pueden vivir.
- Genética: Los organismos traspasan rasgos genéticos a sus crías.
- Selección natural: Aquellos organismos con los rasgos más beneficiosos tendrán más probabilidades de sobrevivir y reproducirse.

Algunas publicaciones de Charles Darwin son:

- Fertilización de las orquídeas, 1862
- Variación de animales y plantas bajo domesticación 1868
- El origen del hombre, 1871
- La expresión de las emociones en el hombre y los animales, 1872
- Las plantas insectívoras, 1875
- Sobre los movimientos y costumbres de las plantas trepadoras, 1875
- Efectos de la autofertilización y de la fertilización cruzada en el reino vegetal, 1876
- Las diferentes formas de las flores, 1877
- Vida de Erasmus Darwin, 1879
- El poder del movimiento de las plantas, 1880
- La formación del mantillo vegetal por la acción de las lombrices, 1881

Darwin falleció el 19 de abril de 1882. Está sepultado en la catedral de Westminster, al lado de Isaac Newton.

ALGORITMOS GENÉTICOS

Los algoritmos genéticos parten de la premisa de emplear la evolución natural como un procedimiento de optimización que se caracteriza por tener operaciones básicas que son:

- Selección
- Cruzamiento
- Mutación

Para poder evaluar las operaciones antes mencionadas es necesario que la información a optimizar se encuentre en cadena de bits (0,1), y que esta representación sea una cadena finita siendo cada cadena un individuo que conforma una población. Existen diferentes representaciones, pero esta es una de las más empleadas. La figura 4.1 muestra individuos que conforman la población, cada individuo tiene una cadena de 12 bits.

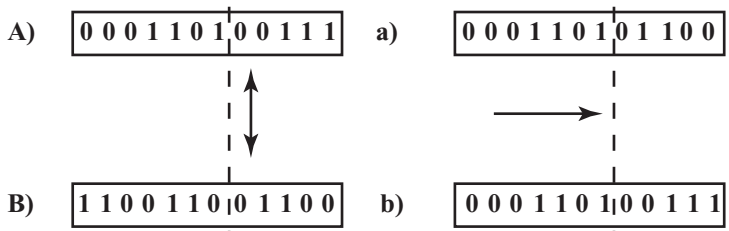


Fig. 4.1 Individuos que forman la población inicial A y B los cuales se encuentran binarizados. A partir de estos individuos iniciales A y B, se generan nuevos individuos a y b, llamados descendientes.

Por lo tanto, el algoritmo es una búsqueda empleando dichas operaciones; por ejemplo, se puede plantear una familia de individuos los cuales se seleccionan y después se considera a los más óptimos para realizar un cruzamiento entre ellos, y la forma de evitar caer en mínimos locales es el empleo de la mutación. Por ello la mutación puede considerarse una operación de fuga de mínimos locales. Esta búsqueda se define como una búsqueda de un espectro amplio.

La forma de la selección se puede realizar de manera lógica empleando el criterio de elegir al individuo más óptimo tomando como base el siguiente criterio.

Introducción

Herencia. Desde tiempos inmemoriales, el hombre ha reflexionado acerca de la herencia de los rasgos físicos o psicológicos. Advirtió fácilmente que en general los hijos se parecen a los padres y que esto no sólo ocurre en el hombre, sino también en los demás organismos. Los filósofos griegos de la Antigüedad clásica ya meditaban sobre estos hechos. Pero el desarrollo de la genética no se inició hasta comienzos del siglo xx, época en que los científicos conocían ya numerosos detalles de la constitución de la célula. Antes, en el año 1860 tuvieron lugar los famosos experimentos del monje austriaco Gregor Mendel, quien cruzó diversas plantas del jardín de su convento y supo establecer algunas de las leyes más importantes de la genética. Sin embargo, la importancia de la obra de Mendel no fue reconocida hasta que los resultados obtenidos por él se compararon con los que el estadounidense Thomas Morgan realizó con la mosca de fruta. La genética, es decir, la ciencia de la herencia, junto con la investigación de la biología celular, ha podido establecer que los genes, portadores de los rasgos hereditarios, se encuentran en los *cromosomas* del núcleo celular. Incluso se ha mostrado que las cualidades dependientes de dichos genes se heredan según ciertas leyes derivadas de las posibles combinaciones de los genes de las células sexua-

les. Hoy se sabe que la información sobre la herencia está almacenada en una especie de "código" en los ácidos nucleicos (ADN y ARN). La moderna genética dedica gran atención al estudio de estos ácidos. Se ha demostrado que aunque son muy estables, pueden transformarse mediante ciertos productos químicos, rayos X, etc., de tal manera que al transmitirse a las células descendientes, provocan en éstas la aparición de nuevas cualidades. Estas modificaciones, positivas o negativas, se llaman "mutaciones" y pueden surgir también en forma espontánea.

¿Qué es herencia? En la reproducción sexual los genes se combinan de diferentes maneras. Los cromosomas paternos se reparten entre las células sexuales, y cuando tiene lugar la fecundación, surgen combinaciones de factores hereditarios diferentes de los que poseen los padres. A través de constantes combinaciones, la masa hereditaria se transmite por medio de las células sexuales, de generación en generación; se puede decir que es potencialmente inmortal.

En las plantas y animales superiores, los cromosomas de las células normales se agrupan por parejas. En cada pareja, uno de los cromosomas procede del padre y el otro de la madre. Al formarse las células sexuales, los dos cromosomas de cada pareja van a parar a células distintas, de modo que los gametos sólo reciben la mitad de los cromosomas contenidos en una célula normal. En el momento de la fecundación, cuando los gametos se fusionan y reúnen sus cromosomas, el número de éstos vuelve a ser el de una célula normal. En los seres humanos, este número es 46 (23 parejas).

Al juntarse las dos células sexuales, cada cromosoma se reúne con su pareja, y cada gen con su homólogo del otro cromosoma. Los genes así emparejados no siempre son iguales. Uno puede ser *dominante,* impidiendo que se manifieste el otro, que se llama *recesivo.*

Se debe considerar que no se heredan directamente las cualidades, sino sólo los genes que las determinan; la aparición de un rasgo depende de la interacción entre los genes y la influencia del medio.

El conjunto de los genes o predisposiciones hereditarias de un individuo es su *genotipo,* mientras que el conjunto de sus caracteres externos, es decir, el resultado de la interacción del genotipo con el medio, es el *fenotipo.* Las cualidades adquiridas sólo por la influencia del medio no pueden heredarse.

El código genético. El material genético está formado por ácidos nucleicos. Éstos se hallan en todas las células vivas, en las que determinan la constitución de las proteínas y de los genes. Se distinguen dos tipos de ácidos nucleicos: ácido desoxirribonucleico (ADN) y ácido ribonucleico (ARN). Ambos están formados por fosfatos, azúcares y bases nitrogenadas. Alrededor de 1940 se consiguió demostrar que la función de los cromosomas, como portadores de los factores hereditarios, corresponde a los ácidos nucleicos. En el año 1962 se otorgó el premio Nobel por el descubrimiento de la estructura y función de la molécula de ADN. Ésta tiene forma de hélice doble. Las dos hélices de la molécula están unidas mediante bases nitrogenadas enlazadas por puentes de hidrógeno. Cada unión está integrada por dos bases, formando cuatro tipos de uniones cuya ordenación en la molécula constituye el código genético. Éste debe ser transmitido con precisión, de célula a célula, en sucesivas generaciones. Dicha transmisión tiene lugar cuando, al dividirse la célula, las hélices de la molécula de ADN se separan y cada una de ellas sintetiza de nuevo su cadena complementaria, hasta formar otra vez hélices dobles que luego se reparten entre las células hijas.

Las moléculas de ARN, homólogas a las del ADN, se forman en el núcleo celular. Después se trasladan a los centros de elaboración de proteínas, los ribosomas del citoplasma, donde se dirige la formación de las diferentes clases de proteínas. Hoy el "código genético" se ha descifrado. Se conoce con mucha exactitud el mecanismo por el cual los ácidos nucleicos (núcleos) lo transportan a los lugares de síntesis para producirlo en proteínas (citoplasmas). A medida que aumenta el conocimiento humano sobre la materia viva, se podrá comprender mejor cómo surgió la vida en la Tierra.

Selección natural. Es un mecanismo evolutivo que se define como la reproducción diferencial de los genotipos en el seno de una población biológica. En su forma inicial, la teoría de la evolución por selección natural constituye la gran aportación de Charles Darwin, la cual fue posteriormente reformulada

en la actual teoría de la evolución: la Síntesis moderna. En Biología evolutiva se considera la principal causa del origen de las especies y de su adaptación al medio.

La formulación clásica de la selección natural establece que las condiciones de un medio ambiente favorecen o dificultan, es decir, seleccionan la reproducción de los organismos vivos según sean sus peculiaridades. La selección natural fue propuesta por Darwin como medio para explicar la evolución biológica. Esta explicación parte de dos premisas. La primera de ellas afirma que entre los descendientes de un organismo hay una variación aleatoria, no determinista, que es en parte heredable. La segunda premisa sostiene que esta variabilidad puede dar lugar a diferencias de supervivencia y de éxito reproductor, haciendo que algunas características de nueva aparición se puedan extender en la población. La acumulación de estos cambios a lo largo de las generaciones produciría todos los fenómenos evolutivos.

La selección natural se puede expresadar como la siguiente ley general, tomada de la conclusión de *El origen de las especies*:

Existen organismos que se reproducen y la progenie hereda características de sus progenitores; existen variaciones de características si el medio ambiente no admite a todos los miembros de una población en crecimiento. Entonces aquellos miembros de la población con características menos adaptadas (según lo determine su medio ambiente) morirán con mayor probabilidad. En consecuencia, aquellos miembros con características mejor adaptadas tendrán mayores posibilidades de sobrevivir.

El resultado de la repetición de este esquema a lo largo del tiempo es la evolución de las especies.

La evolución biológica es un fenómeno natural real, observable y comprobable empíricamente. La llamada Síntesis evolutiva moderna es una sólida teoría que actualmente proporciona explicaciones y modelos matemáticos respecto de los mecanismos generales de la evolución o los fenómenos evolutivos, como la *adaptación*. Cualquier teoría científica tiene su hipótesis y está sujeta a crítica y comprobación experimental.

Uno de los fundadores de la Síntesis moderna, Dobzhansky, definió la evolución del siguiente modo: "La evolución es un cambio en la composición genética de las poblaciones". La Síntesis moderna de la evolución se basa en tres aspectos fundamentales:

1. La ascendencia común de todos los organismos de un único ancestro.
2. El origen de nuevos caracteres en un linaje evolutivo.
3. Los mecanismos por los que algunos caracteres persisten mientras que otros desaparecen.

Algoritmos genéticos

La primera idea surgió en la tesis de J. D. Bagley, "El funcionamiento de los sistemas adaptables empleando algoritmos genéticos y correlativos", en 1967. Dicha tesis influyó decisivamente en J. H. Holland, quien se puede considerar como el pionero de los algoritmos genéticos.

En rigor, se puede ver a los AG como métodos de optimización. En general, los problemas de optimización se plantean de la siguiente forma:

$x_0 \in X$ tal que f es un máximo en x_0, donde $f : X \to \Re$, por lo tanto:

$$f(x_0) = \max_{x \in X} f(x)$$

Es casi imposible obtener una solución en sentido estricto. Dependiendo del problema planteado, puede ser suficiente encontrar el máximo valor o el más cercano al valor máximo. f es una función asignada para definir el valor de "aptitud" para cada individuo (esto es, por supuesto, una gran ayuda para simplificar).

Definición. Se asumirá que S es un arreglo de cadenas (es un caso no trivial en algunos aspectos de la gramática). Proponiendo que X es el espacio deseado para la optimización, se tendrá la siguiente función:

$$c : X \to S$$
$$x \to c(x)$$

esta función se llama *función codificación*. Inversamente, la función

$$\tilde{c} : S \rightarrow X$$
$$s \rightarrow \tilde{c}(s)$$

se llama *función de decodificación*. Estas funciones deben ser especificadas dependiendo de las necesidades del problema que se plantea, y no necesariamente son biyectivas. Pero es conveniente usar funciones biyectivas. Además se debe cumplir la siguiente igualdad:

$$(c \circ \tilde{c}) \equiv id_s$$

Definiciones

La siguiente lista contiene diferentes expresiones que se usan en la genética y su estructura equivalente en AG:

Evolución natural	Algoritmo genético
genotipo	código de cadena
fenotipo	punto sin codificar
cromosoma	cadena
gen	posición de cadena
alelo	valor en una posición determinada
función de aptitud o aptitud	valor de la función objetivo

- ***Genotipo***: expresión genética de un organismo o estructura genética del organismo. Es la información contenida en el genoma.
- ***Fenotipo***: características físicas de un organismo, atribuibles a la expresión de su fenotipo. Contiene tanto los rasgos físicos como los conductuales. Es el resultado de la interacción entre el genotipo y el ambiente; se interpreta como la suma de los caracteres observables en un individuo. Es la manifestación externa del genotipo.
- ***Cromosoma***: es la molécula única de ADN unida a histonas (proteínas básicas) y otras proteínas que se condensa durante la mitosis (proceso de división celular - reparto equitativo del material hereditario) y la meiosis (proceso de fragmentación - divisiones pequeñas), formando una estructura compacta.
- ***Gen***: especifica la herencia de un carácter; está formado por una secuencia de aminoácidos de una o más cadenas de ARN (ácido ribonucléico- interviene en diferentes neuronas, en la expresión de la información genética), que realizan diferentes funciones en la cadena.
- ***Alelo***: el valor de un gen. Una de las dos o más formas alternativas de un gen; determina el carácter controlado por el gen. Un ejemplo es el *diploide* que contiene dos juegos de cromosomas, por lo tanto tiene dos copias de cada gen.
- ***Función de aptitud***: es un tipo especial de función que cuantifica la optimalidad de una solución. Se traduce en un cromosoma óptimo para que sus bases sean combinadas con cualquier otra técnica para la producción de una nueva generación que sea mejor a las anteriores.

En términos generales, las cadenas de los algoritmos genéticos son análogas a los *cromosomas* en el sistema biológico. En los organismos naturales, uno o más cromosomas se combinan para formar la prescripción genética total, para la construcción y operación del organismo. En los organismos naturales el "paquete total" de genética se llama genotipo. En los algoritmos genéticos el "paquete total" de las cadenas se denomina estructura (la estructura está compuesta por varias cadenas). En los organismos naturales, la creación de los organismos se realiza mediante la interacción de "paquetes" genéticos con su medio y se le nombra *fenotipo*.

En la terminología natural, se dice que los cromosomas están compuestos por genes, los cuales permiten tomar distintos valores llamados *alelos*. En la genética la posición del gen (llamado *locus*) se identifica en forma separada de la función del gen. Así, se puede hablar de un gene en particular, por ejemplo el gen que da color a los ojos de los animales: su lugar es la posición 10 y el valor del alelo es ojos azules.

En los algoritmos genéticos las cadenas están compuestas por características que toman diferentes valores. Estas características se localizan en distintas posiciones de la cadena.

Se escribirá a continuación la estructura básica de un AG.

```
Algoritmo
t:=0
Se computa la  población inicial Bo.
WHILE se detiene la condición que no se cumple DO
BEGIN
Se selecciona a los individuos para la reproducción;
Se crea la descendencia cruzando a los individuos;
Eventualmente se mutan los individuos;
Se computa la nueva generación
END
```

Para las estructuras básicas de un AG también es necesario conocer la transición de una generación a otra, que consta de cuatro elementos básicos:

- ***Selección***: mecanismo de selección individual (cadena) para la reproducción acorde con la *función de aptitud* (valor de la función objetivo). Los algoritmos de selección serán los encargados de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no. La idea básica de selección está asociada con la *función de aptitud* y el sistema original; para implementación es comúnmente conocida como *roulette-wheel* (RWS); ésta utiliza una distribución de probabilidad, donde la probabilidad de selección de una cadena es directamente proporcional a su *aptitud*.
- ***Cruzamiento***: método de fusión sobre la información genética de dos individuos; si la codificación se elige apropiadamente, dos progenitores saludables producirán descendientes sanos. Es el principal operador genético; provee un mecanismo para heredar características a su descendencia; interviene en ambos progenitores.
- ***Mutación***: en la evolución real, el material genético puede ser alterado en forma aleatoria debido a un error en la reproducción o la deformación de genes; un ejemplo es la radiación de los rayos gama. En los AG, la mutación se realiza, con gran probabilidad, como una deformación aleatoria de las cadenas. Produce cambios incrementales al azar en la descendencia, efectuando cambios aleatorios en los valores del alelo en algunos genes. En el caso de cromosomas binarios, le corresponde hacer los cambios de posiciones en cada bit. No afecta a toda la población,

pero es probable que dañe a algunos. La mutación tiene el efecto de perturbar de manera segura a los cromosomas a fin de introducir nuevas características que no estaban presentes en ningún elemento de los progenitores.

- ***Reemplazo***: procedimiento para calcular (crear) una nueva generación de la anterior y sus descendientes. Se crea un espacio a la descendencia en la población eliminando de ella a los padres.

Los AG trabajan con un número fijo de cadenas binarias de longitud fija. Para este fin, se asumirá que las cadenas que se va a considerar son del mismo conjunto:

$$S = \{0,1\}^n;$$

Por lo tanto, la generación en el tiempo t es una lista de m cadenas donde se denotará como:

$$B_t = (b_{1,t}, b_{2,t}, \ldots, b_{m,t}).$$

Los AG, en su mayoría, tienen la estructura siguiente:

```
Algoritmo
t:=0
Se computa la  población inicial B0 = (b1,0, b2,0, ..., bm.0);
WHILE se detiene la condición que no se cumple DO
BEGIN
FOR i:=1 TO m DO
Se selecciona un individuo bi,t+1 de Bt;
FOR i:=1 TO m-1 STEP 2 DO
IF Random [0, 1]≤pC THEN
cruza bi,t+1 con bi,t+1,t+1;
FOR i:=1 TO m DO
eventualmente mutar
se crea la descendencia cruzando a los individuos;
eventualmente se mutan bi,t+1;
t:=t+1
END
```

La selección, el cruzamiento (se realiza sólo con una probabilidad crítica de percolación p_C) y la mutación tienen ciertos grados de libertad, mientras que la operación de reemplazo ha sido especificada. Como es fácil de apreciar, todo individuo seleccionado es reemplazado por su sucesor después del cruzamiento y la mutación; los individuos no seleccionados mueren inmediatamente.

Operaciones genéticas en cadenas binarias

Selección

Es la componente que guía el algoritmo para encontrar la solución, prefiriendo dentro de un grupo de baja *función de aptitud* a los más altos. Puede ocuparse una operación determinista; en la mayoría de las implementaciones tiene componentes aleatorios. La probabilidad de escoger el individuo adecuado es directamente proporcional a su *función de aptitud*. Se puede observar cómo un experimento aleatorio con

$$P[b_{j,t} \text{ es seleccionada}] = \frac{f(b_{j,t})}{\sum_{k=1} f(b_{k,t})}$$

sólo funciona para los valores positivos de las *funciones de aptitudes*. Si éste no es el caso, debe aplicarse una transformación de no-decremento $\varphi : \Re \to \Re^+$ (un cambio en el caso más simple). Entonces la probabilidad puede ser expresada por

$$P[b_{j,t} \text{ es seleccionada}] = \frac{\varphi(f(b_{j,t}))}{\sum_{k=1} \varphi(f(b_{k,t}))}$$

las diferentes salidas se obtendrán con diferentes probabilidades. La programación del algoritmo se puede proponer como sigue, y la configuración analógica se representa con la expresión matemática anterior.

```
Algoritmo
x: =Random[0,1];
i: =1
WHILE i<m & x< Σ_{j=1}^{i} f(b_{j,t}) / Σ_{j=1}^{m} f(b_{j,t}) DO
i: i+1;


selecciona b_{i,t+1};
```

Este método a menudo se llama selección proporcional.

Cruzamiento

En la reproducción sexual, como se desempeña en el mundo real, la materia genética de los progenitores se mezcla cuando los gametos de los progenitores se fusionan. Por lo general, los cromosomas son aleatoriamente divididos y fusionados, con la consecuencia de que algunos genes de los descendientes provienen de un progenitor, mientras que otros provienen del otro progenitor. Este mecanismo se denomina *cruzamiento*. Es una herramienta muy potente para introducir nuevos materiales genéticos y mantener la diversidad genética, pero con la notable propiedad de que progenitores saludables también producen buen rendimiento en los descendientes, o incluso mejores. Varias investigaciones han llegado a la conclusión de que el cruzamiento es la razón por la que las especies de reproducción sexual se adaptan más rápido que las de reproducción asexual.

Básicamente, el *cruzamiento* es el intercambio de genes entre los cromosomas de los dos progenitores. En un caso simple, se puede realizar este proceso cortando dos cadenas en una posición elegida al azar e intercambiándolas en sus extremos. Este proceso, que se llama *cruzamiento de único-punto*, se muestra en la figura 4.2.

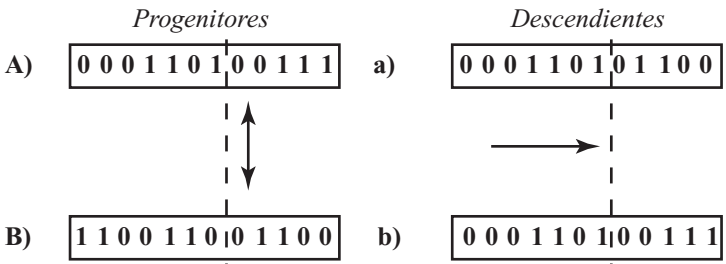


Fig. 4.2 Cruzamiento de único-punto

A continuación se muestra la programación para el cruzamiento

```
Algoritmo
pos:=Random{1,…,n-1};
FOR i:=1 TO pos DO
BEGIN
Child1[1] := Parent1 [i];
Child2[1] := Parent2 [i]
END

FOR i:=pos+1 TO n DO
BEGIN
Child1[1] := Parent1 [i];
Child2[1] := Parent2 [i]
END
```

Para otros problemas o diferentes codificaciones, pueden ser útiles o incluso necesarios otros métodos de cruzamiento. A continuación se menciona algunos de ellos:

- ***Cruzamiento de N-puntos***: en lugar de un único punto, se eligen al azar N puntos de ruptura. Cada segunda sección se intercambia. Entre estas clases, la de *dos-puntos* es particularmente importante.
- ***Cruzamiento segmentado***: esta técnica es muy parecida al cruzamiento de N-puntos, con la diferencia de que el número de puntos de ruptura puede variar.
- ***Cruzamiento uniforme***: para cada posición se decide al azar si se intercambian las posiciones.
- ***Cruzamiento aleatorio***: primero se escoge una permutación aleatoria que se aplicará a los progenitores, después el cruzamiento de N-puntos se aplica a los progenitores aleatorios y, finalmente, los descendientes aleatorios se transforman de nuevo con permutación inversa.

Mutación

El último ingrediente de los AG es la *mutación*, es decir, la deformación aleatoria de la información genética en un individuo, como las radiaciones radioactivas u otros medios de influencia. En la reproducción real, la probabilidad de que un determinado gen sea mutado es casi igual para todos los genes. Así, está al alcance de la mano usar las siguientes técnicas de mutación para una determinada cadena binaria s, donde p_M es la probabilidad de que un sólo gen sea modificado:

Algoritmo

```
FOR i:=1 TO n DO
IF Random [0,1]< pM THEN
invert s[i];
```

Por supuesto, se puede encontrar muchas alternativas y con más detalles. Algunas técnicas se muestran a continuación:

- ***Inversión de un solo bit***: la probabilidad de mutación p_M de que un bite elegido al azar sea negado.
- ***Inversión por fragmentos***: toda la cadena es invertida bit a bit con una probabilidad de mutación p_M.
- ***Selección aleatoria***: la probabilidad de mutación p_M de que una cadena elegida al azar sea reemplazada.

Resumen

Si se desea ocupar los métodos descritos, podemos escribir un algoritmo genético universal para la solución de problemas de optimización en el espacio $S = \{0,1\}^n$.

```
Algoritmo

t:=0
Se crea la población inicial B_0 = (b_{1,0}, b_{2,0}, ..., b_{m.0});
WHILE se detiene la condición que no se cumple DO
BEGIN
(* selección proporcional *)

    FOR i:=1 TO m DO
    BEGIN
    x:= Random [0, 1];

    k:=1;
    WHILE k<m & x< ∑_{j=1}^{i} f(b_{j,t}) / ∑_{j=1}^{m} f(b_{j,t}) DO
        k: k+1;
        b_{i,t+1} = b_{k,t}
    END
(* único-punto de cruzamiento *)

    FOR i:=1 TO m-1 STEP 2 DO
    BEGIN
    IF Random [0, 1]  THEN
    BEGIN
        pos:=Random{1,…,n-1};

        FOR k:=pos+1 TO n DO
        BEGIN
            aux:=  [k];
            b_{i,t+1} [k]:=  [k];
            b_{i,t+1,t+1} [k]:=aux
```

```
            END
        END
    END
    ( * mutación* )

    FOR i:=1 TO m DO
        FOR k:=1 TO n DO
    IF Random [0,1]< p_M THEN
    invert b_{i,t+1} [k];

    t:=t+1
    END
```

Ejemplo

Encontrar el máximo de la función siguiente:

$$f_1 : \{0, \ldots, 31\} \to \Re$$
$$x \to x^2$$

para especificar una cadena con el espacio a lo largo de un sistema de codificación y decodificación. En este ejemplo, se puede considerar $S = \{0,1\}^5$, donde el valor de $\{0, \ldots, 31\}$ se codifica por su representación binaria. De manera correspondiente, una cadena es decodificada como

$$\tilde{c}(s) = \sum_{i=0}^{4} s[4-i] \cdot 2^i.$$

Con una población de $m = 4$ y una probabilidad de cruzamiento $p_C = 1$ y una probabilidad de mutación $p_M = 0.001$, la primera generación al azar con distribución uniforme de $S = \{0,1\}^5$, se obtendrá lo siguiente en el primer paso:

Núm. del individuo	cadena (genotipo)	valor x (fenotipo)	$f(x)$ x^2	P seleccionada$_i$ $f_i / \sum f_i$
1	01101	13	169	0.14
2	11000	24	576	0.49
3	01000	8	64	0.06
4	10011	19	361	0.31

La suma de la *función de aptitud* es 1170, donde el promedio es 293 y el máximo es 576. Se puede apreciar que en las últimas columnas la selección proporcional favorece a los de *alta-aptitud* (como el individuo núm. 2) sobre los de *baja-aptitud* (como el individuo núm. 3).

Realizando un experimento aleatorio, se ha dejado morir al individuo núm. 3 y seleccionado doble vez al núm. 2, con lo cual se obtuvo una segunda generación como sigue:

Conjunto de individuos seleccionados	Lugar de cruzamiento (aleatorio)	Nueva población	valor x	$f(x)$ x^2
0110|1 (1)	4	01101	12	144
1100|0 (2)	4	11001	25	625
11|000 (2)	2	11011	27	729
10|011 (4)	2	10000	16	256

Ahora el interés está enfocado en el máximo de una función

$$f_2 : [-1,1] \to R$$
$$x \to 1 + e^{-x^2}.\cos(36x).$$

Teniendo una cadena binaria, lo primero es discretizar el espacio buscado con valores entre $[-1,1]$. Una técnica común para hacerlo es realizar una graduación uniforme 2^n puntos, después enumerar los puntos graduados y usar la representación binaria del índice de puntos como codificación. La forma general (para un intervalo arbitrario *[a, b]*) se muestra a continuación:

$$C_{n[a,b]} : [a,b] \to \{0,1\}^n$$
$$x \to bin_n\left(\text{round}\left[\left((2n-1).\frac{x-a}{b-a}\right)\right]\right),$$

Donde bin_n es una función que convierte el números desde $\{0, \ldots 2^{n-1}\}$ a su representación binaria con una longitud n. Este operador no es biyectivo, la información se pierde debido al redondeo. Su correspondiente decodificación se define como

$$\widetilde{C}_{n,[a,b]} : \{0,1\}^n \to [a,b]$$
$$s \to a + bin_n^{-1}(s[\,).\frac{b-a}{2^n-1}].$$

La función de decodificación $\widetilde{C}_{n,[a,b]}$ es inyectiva.

Análisis

Para métodos de optimización determinística convencional, tanto el método del gradiente, Newton o el método cuasi-Newton, etc., son usualmente empleados para obtener resultados que garanticen que la secuencia de la iteración converja. Los métodos de optimización probabilística sólo pueden dar información de expectativas o comportamiento aproximado.

Desde que la transición de una generación a la siguiente es una combinación de tres operadores probabilísticos (selección, cruzamiento y mutación), la estructura interna del algoritmo genético es bastante complicada. Para cada uno de los operadores probabilísticos existen diferentes variantes que se han propuesto; es imposible de dar un resultado de convergencia general por el hecho de que las elecciones realizadas por los operadores influyen esencialmente en la convergencia.

Definición. Una cadena $H = (h_1, \cdots, h_n)$ mayor que el alfabeto $\{0,1\ *\}$ se llama *modelo schema* (binario) con longitud n. Una $h_i = *$ se denomina *especificación de H*, una $h_i = *$ se nombra *wildcard*.

No es difícil de apreciar que la *schemata* (plural de *schema*) puede ser considerada como un subconjunto específico de $\{0,1\}^n$ si se considera que la siguiente función donde el mapeo del *modelo schema* para ésta, se encuentra asociada con el subconjunto.

$$i:\{0,1,*\}^n \rightarrow [P(\{0,1\}]^n)$$

$$\{s \mid \forall 1 \le i \le n : (h]_i = *)(\Rightarrow)(h_i = s_i)\}$$

Definición. Una cadena $S = (s_1, \cdots, s_n)$ mayor que el alfabeto $\{0,1\}$ *cumple el modelo schema* $H = (h_1, \cdots, h_n)$ *si y sólo si* ésta coincide con H en que todas sus posiciones son *non-wildcard*:

$$[\forall i \in \{i \mid h]_j \neq * \} : s_i = h_i$$

El *número de especificaciones* del *modelo schema H* se llama *orden* y se denota como

$$O(H) = \left| \{ i \in \{1, \cdots, n\} \mid h_i \neq * \} \right|.$$

La distancia entre los primeros y las últimas especificaciones es

$$\delta(H) = \max\{i \mid h_i \neq *\} - \min\{i \mid h_i \neq *\}$$

se denomina la *longitud definida* del *modelo schema H*.

El Teorema del Schema

El llamado *Teorema del Schema* provee una idea profunda y valiosa del principio intrínseco de los AG. Asumiendo que se tiene un AG del tipo *cadena binaria* con parte proporcional de *selección* (operador) y una *función de aptitud f* arbitraria pero arreglada. Se realizarán las siguientes notaciones.

1. El número de individuos que cumplen el tiempo *t* para *H* se denotan como

$$r_{H,t} = \left| B_t \cap H \right|.$$

2. La expresión $(\bar{f}\, t)$ hace referencia a *aptitud* promedio observada para un tiempo *t*:

$$(\bar{f}\, t) = \frac{1}{m} \sum_{i=1}^{m} f(b_{i,t})$$

3. El término $(\bar{f}\, H, t)$ representa la *aptitud* promedio observada en el *modelo schema H* en un tiempo *t*:

$$(\bar{f}\, H, t) = \frac{1}{r_{H,t}} \sum_{i \in \{j \mid b_{i,t} \in H\}}^{m} f(b_{i,t})$$

Teorema del *Schema* –Holland, 1975. *Asumiendo que se puede considerar un algoritmo genético del tipo universal (antes visto, para problemas de optimización), la siguiente desigualdad cumple para todo modelo schema H.*

$$E[r_{H,t+1}] \ge r_{H,t} \cdot \frac{(\bar{f}\, H, t)}{(\bar{f}\, t)} \cdot \left(1 - p_c \cdot \frac{\delta(H)}{n-1}\right) \cdot (1 - p_c)^{o(H)}$$

La probabilidad de que se ha seleccionado a un individuo que cumpla con *H* es:

$$\frac{\sum_{i \in \{j \mid b_{i,t} \in H\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})}$$

Esta probabilidad no cambia durante la ejecución del lazo de *selección*. Además, cada una de las *m* es completamente seleccionada de manera individual de otras. Con una cantidad de muestras *m* y la probabilidad anteriormente definida, se puede obtener, por lo tanto, que el número esperado de individuos seleccionados que cumple con *H* es:

$$m.\frac{\sum_{i\in\{j|b_{i,t}\in H\}}^{m} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})} = m.\frac{r_{H,t}}{r_{H,t}}\frac{\sum_{i\in\{j|b_{i,t}\in H\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})}$$

$$= r_{H,t}.\frac{\frac{\sum_{i\in\{j|b_{i,t}\in H\}} f(b_{i,t})}{r_{H,t}}}{\frac{\sum_{i=1}^{m} f(b_{i,t})}{m}} = r_{H,t}.\frac{(\bar{f}H,t)}{(\bar{f}t)}$$

Si dos individuos que se cruzan cumplen con *H*, entonces las dos descendencias también cumplirán con *H*. El número de cadenas que cumple con *H* puede disminuir si uno de los individuos que cumple con *H* se cruza con otro que no lo cumpla; pero en caso de que el espacio de cruzamiento se halle dentro de las especificaciones de *H*, la probabilidad de este espacio (dentro de la longitud definida por *H*) es:

$$\frac{\delta(H)}{n-1}$$

Partiendo de este punto la probabilidad de supervivencia p_S de *H* puede ser calculada como se muestra (el cruzamiento se calcula sólo con p_C):

$$p_s \geq 1 - p_c.\frac{\delta(H)}{n-1}$$

La selección y el cruzamiento son calculados de manera independiente, así al computar los números de cadenas esperadas que cumplen con *H* después de la cruza, se calcularán como:

$$\frac{(\bar{f}H,t)}{(\bar{f}t)}.r_{H,t}.p_s \geq \frac{(\bar{f}H,t)}{(\bar{f}t)}.r_{H,t}.\left(1 - p_c.\frac{\delta(H)}{n-1}\right).$$

Después de la cruza, el número de cadenas que cumplen con *H* puede decrecer si la probabilidad de que todas las especificaciones de *H* queden intactas por la mutación. Una cadena que cumple con *H* es alterada por mutación debido a las especificaciones de probabilidad

$$(1 - p_c)^{o(H)}$$

Para un algoritmo genético del tipo *cadena binaria* con una ruleta de *selección* (roulette wheel), la desigualdad se cumple

$$E[r_{H,t+1}] \geq \frac{(\bar{f}H,t)}{(\bar{f}t)} \cdot P_c(H) \cdot P_M(H)$$

para cualquier modelo schema *H*, donde $p_C(H)$ es una constante que depende del modelo schema *H* y del método de cruzamiento; $p_M(H)$ es una constante que depende únicamente de *H* y el operador mutación implicado.

Existe una pregunta fundamental que no se ha abordado del todo. Indudablemente, los AG operan en cadenas binarias, pero no en schemata (plural de schema). El Teorema del *Schema* casi provee una

observación de toda la schemata, donde todo crece y decae acorde con su valor de *aptitud* promedio observado en paralelo. ¿Cuál es realmente la interpretación de este comportamiento y por qué hacer esto?

Finalmente, se formula una pregunta más: ¿dónde el rol crucial de la schemata sobrepasa el promedio de la *aptitud* y de dónde proviene la longitud definida corta y qué son la influencia de la función de aptitud y la codificación del *modelo schema*?

La óptima asignación de los procesos

La Teoría del Schema ha proporcionado estructuras de bloques que reciben procesos de crecimiento exponencialmente para generaciones futuras. La pregunta es, ¿por qué ésta puede ser una buena estrategia? Esto conlleva a un problema de gran importancia y análisis sobre toma *de decisiones estadístico* y saber cuál es la estrategia adecuada. Por ello se llega a la conclusión, apoyada por los estudios de Holland, de que la óptima estrategia se encuentra dada en la siguiente ecuación:

$$N - n^{\bullet} = \sqrt{8\pi b^4 \ln N^2} \cdot e^{n^4/2b^2}$$

Paralelismo implícito

Hasta ahora sea han encontrado dos puntos de vista diferentes sobre AG:

1. El punto de vista de los algoritmos como operadores de AG con cadenas.
2. La interpretación basada en el *modelo schema*.

Por ello, se puede preguntar si los AG son realmente procesos de cadenas o esquemas. La respuesta es que son ambas. Hoy en día, la interpretación común es que en los procesos sobre AG se encuentra implícita una gran cantidad de schemata.

Se puede buscar información sobre diferentes regiones de interés de manera continua a través de una búsqueda en paralelo.

Esta notable propiedad se llama comúnmente *paralelismo implícito* para los AG.

Una representación simple del AG es como se ha venido presentado (el Teorema del *Schema*), sólo una estructura *m* en un tiempo *t*, sin ninguna memoria o contabilidad de generaciones previas. Ahora se intentará obtener una aproximación de cuántas schemata puede procesar actualmente un AG.

Por ello, existen 3^n schemata de longitud *n*. Una sola cadena binaria que cumple *n* schemata de primer orden, $((_{\perp}2^t n))$ schemata de segundo orden y el caso general cuando $((_{\perp}k^t n))$ schemata de *k* orden. Partiendo de este punto, una cadena debe cumplir

$$\sum\nolimits_{\perp}(i=1)^t m \equiv [((_{\perp}k^t n)) = 2^t n]$$

schemata. Así, para cualquier generación entre 2^n y $m \cdot 2n$ schemata, existe al menos una representación. ¿Pero actualmente cuántas schemata se procesan? Holland dio una cantidad estimada de schemata que se pueden manejar durante la siguiente generación. Aunque el resultado parece un tanto desconcertante, aun así provee información importante de la gran cantidad de schemata que difícilmente son procesadas por el momento; de hecho, se consideró una cantidad relativa muy pequeña de cadenas para la estimación.

Si se considera que una generación de población se crea de manera aleatoria por algoritmo genético simple (como el de tipo universal) y se dice que $\varepsilon \in (0, 1)$ *es el error corregido vinculado, entonces la longitud de la schemata es*

$$l_s \leq \varepsilon \cdot (n-1) + 1$$

tiene la probabilidad por lo menos $1 - \varepsilon$ de sobrevivir al cruzamiento de un-punto. Si el tamaño de la población es escogida como $m = 2^{l_s/2}$, el número de schemata que sobreviven para la siguiente generación es del orden $O(m^3)$.

Conjunto difuso de sintonización

Se debe comenzar este tema aprendiendo la primera tarea: cómo se encuentra la configuración óptima de un conjunto difuso. Anteriormente, se tenía el *algoritmo genético universal*, el cual resuelve una clase general para problemas de optimización. Ahora se estudiará qué tanto el *algoritmo genético simple* se le puede aplicar una optimización con conjuntos difusos. Lo que se requiere es una codificación apropiada, un operador genético (en caso que las variables estándar no sean suficientes) y una escala de *aptitud*.

Codificación de un subconjunto difuso en un intervalo

Desde que este es uno de los casos más importantes en aplicaciones de sistemas difusos, se puede restringir un subconjunto difuso, dado en un intervalo [*a*, *b*] de números reales. Cabe mencionar que nunca se podrá encontrar un posible *conjunto difuso* que satisfaga la codificación. Es común en aplicaciones arreglar una subclase en particular que pueda representar un promedio finito de parámetros. La descripción de dicho *conjunto difuso* puede ser entonces encriptado codificando estos parámetros.

Existen segmentos en las funciones de membresía lineales que contienen conjuntos de arreglos formados por redes de puntos $(a = x_0, x_1, \cdots, x_{n-1}, x_n = b)$, y una red de espacio igual en los casos simples. Los programas populares de computación para control difuso como fuzzyTECH o TILShell utilizan estas técnicas para su representación interna de conjuntos difusos. Es fácil de apreciar que su forma de la función de membresía es únicamente determinada por el grado de pertenencia que puede tener esta red de puntos. Por lo tanto, se puede encriptar de manera sencilla un conjunto difuso, tal que se incluya en la codificación todos los valores de la función en una gran cadena (figura 4.2):

$C_{n,[0,1]}(\mu(x_0))$	$C_{n,[0,1]}(\mu(x_1))$	$\cdots$	$C_{n,[0,1]}(\mu(x_n))$

Fig. 4.2 Codificación de los valores de una función.

Una solución práctica para la encriptación del grado de pertenencia es $n = 8$. Tanto que los 8-bits de codificación se usan en varios programas de computación también.

Para la mayoría de los problemas, sin embargo, una sencilla representación de conjuntos difusos es suficiente. Muchas aplicaciones reales usan las funciones de membresía triangulares y trapezoidales. No es de sorprenderse que una función triangular pueda ser encriptada como (figura 4.3):

$C_{n,[a,b]}(r)$	$C_{n,[0,\delta]}(u)$	$C_{n,[0,\delta]}(v)$

,

Fig. 4.3 Función triangular encriptada.

donde δ es el límite superior para el tamaño del offset (compensador), por ejemplo $\delta = (b - a)/2$. Lo mismo puede ser realizado análogamente para una función trapezoidal (figura 4.4):

$C_{n,[a,b]}(r)$	$C_{n,[0,\delta]}(q)$	$C_{n,[0,\delta]}(u)$	$C_{n,[0,\delta]}(v)$

Fig. 4.4 Función trapezoidal.

En aplicaciones específicas de control, donde el comportamiento suave de éste juega un rol importante, los conjuntos difusos con cambios notables deben usarlo. La función más prominente es la de campana, cuya función de membresía está dada por la función campana de Gauss:

$$\mu(x) = e^{\frac{(x-r)^2}{2u^2}}$$

El análogo de la función campana a trapezoidal también se denomina funciones de base radial.

$$\mu(x) = \begin{cases} e^{\frac{(x-r)^2}{2u^2}} & \text{si } |x - r| > q \\ 1 & \text{si } |x - r| > q \end{cases}$$

La forma común de la función de membresía campana se muestra abajo. De nuevo el método de codificación es sencillamente (figura 4.5)

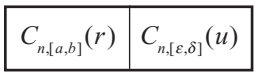


Fig. 4.5 Función de membresía campana.

donde $\in$ es el límite inferior para la extensión u. Análogamente para la función de base radial se tiene (figura 4.6):

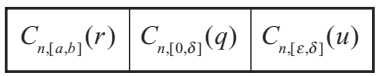


Fig. 4.6 Función de base radial.

El paso final es simple y sencillo: para definir la codificación de toda la configuración, con todos los conjuntos difusos que intervienen, es suficiente codificar de todos los conjuntos difusos relevantes, una gran cadena.

Funciones de aptitud estándar

Aún es difícil de formular una receta general que cumpla con todos los tipos de aplicaciones; existe una situación estándar importante: en el caso donde un ejemplo conjunto entradas-salidas representativo es dado. Se puede asumir que $F(\vec{v}, x)$ es una función que computa las salidas obtenidas por las entradas x proporcionadas con respecto a los parámetros del vector $\vec{v}$. Una muestra de información es provista en forma de lista como pares (x_t, y_t) con intervalo $1 \leq t \leq K$ (K es el número de muestras de información). Obviamente la meta es encontrar una configuración de parámetros $\vec{v}$ tal que las salidas correspondientes $F(\vec{v}, x)$ coincidan con el muestro de salidas y_t lo mejor posible. Esto se puede lograr reduciendo la función de error.

$$f(\vec{v}) = \sum_{i=1}^{K} d(F(\vec{v}, x_i), y_i),$$

donde $d(*, *)$ es una distancia calculada que se define en el espacio de salida. En caso de que la salida se conforme por números reales, una muestra prominente (aproximación) es la *bien conocida* suma de errores cuadráticos:

$$f(\vec{v}) = \sum_{i=1}^{K} (F(\vec{v}, x_i) - y_i)^2.$$

Para un mejor empleo de la caja de herramientas MATLAB® Toolkit Algoritmos genéticos en el desarrollo de AG empleando AG, vea el anexo A.

5

EJEMPLO DE AG EN MATLAB®

DETERMINAR LA IMPEDANCIA NECESARIA DE UN COMPONENTE PARA QUE UN CIRCUITO AC LE TRANSFIERA LA MÁXIMA POTENCIA DE ENERGÍA

Introducción

El AG tiene entre sus ventajas funcionar bien para optimización global de funciones especialmente cuando la función objetivo es discontinua o con varios mínimos locales. Sin embargo, esto conduce a posibles inconvenientes. Dado que no se utiliza información adicional, como la evaluación de gradientes y pendientes, el AG tiene una tasa de convergencia lenta con funciones objetivo sin muchas variaciones.

El AG se puede utilizar tanto en la optimización con o sin restricciones. Es posible aplicarlo a la programación lineal, la programación estocástica (el caso donde los problemas de programación matemática tienen variables aleatorias, introduciendo así un grado de incertidumbre) y los problemas de optimización combinatoria.

Aplicaciones

Dado que es factible emplear el AG para resolver problemas tanto con restricciones como sin ellas, simplemente es una manera de obtener una solución a un problema de optimización estándar. Por lo tanto, se puede utilizar para resolver problemas de optimización clásicos tales como la maximización del volumen o la reducción al mínimo de la cantidad de material necesario para producir un recipiente. Mediante la aplicación de AG a los problemas de programación lineal y no lineal, es posible resolver los problemas comunes como el problema de la dieta (eligiendo la más barata de un conjunto de alimentos que deben cumplir con ciertos requerimientos nutricionales). También se pueden aplicar a los problemas de optimización combinatoria en la ciencia de la computación, como es el manejo de horarios.

Problema de máxima transferencia de potencia

Se considera el circuito de la figura 5.1, donde un circuito de corriente alterna (ac) se conecta al componente Z_L y se representa mediante su equivalente de Thévenin. El componente generalmente se represen-

ta por su impedancia que puede modelar un motor eléctrico, una antena, una TV, entre otros. En forma rectangular, la impedancia de Z_{Th} y Z_L son:

$$Z_{Th} = R_{Th} + jX_{Th}$$
$$Z_L = R_L + jX_L$$

donde, R es la resistencia y X la reactancia.

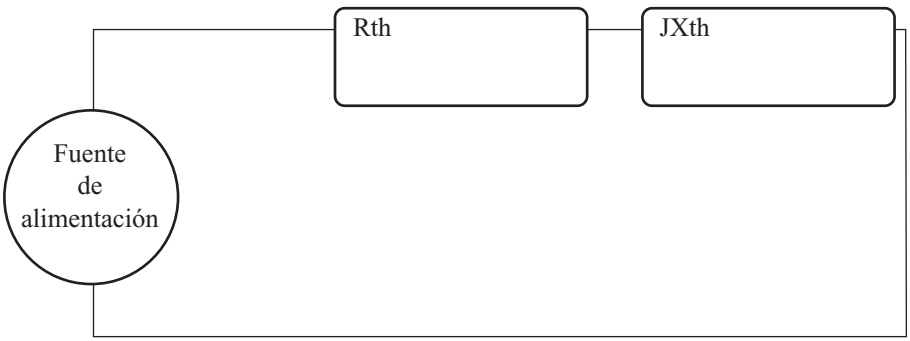


Fig. 5.1 Circuito alimentado con corriente alterna.

Entonces la corriente a través de componente es

$$I = \frac{V_{Th}}{Z_{Th} + Z_{Th}} = \frac{V_{Th}}{\left(R_{Th} + jX_{Th}\right) + \left(R_L + jX_L\right)}$$

y la potencia promedio

$$P = \frac{1}{2}\left|I^2\right| R_L = \frac{1}{2} R_L \frac{\left(V_{Th}\right)^2}{\left(R_{Th} + R_L\right)^2 + \left(X_{Th} + X_L\right)^2}$$

El objetivo es variar los parámetros R_L y X_L para obtener P máxima. Matemáticamente, esto se puede hacer derivando de manera parcial con respecto a cada parámetro e igualando a cero. Con esto se obtiene:

$$\frac{\partial P}{\partial X_L} = -\frac{\left(V_{Th}\right)^2 R_L \left(X_{Th} + X_L\right)}{\left[\left(R_{Th} + R_L\right)^2 + \left(X_{Th} + X_L\right)^2\right]^2}$$

$$\frac{\partial P}{\partial R_L} = -\frac{\left(V_{Th}\right)^2 \left[\left(X_{Th} + X_L\right)^2 + \left(R_{Th} + R_L\right)^2 - 2R_L\left(R_{Th} + R_L\right)\right]}{\left[\left(R_{Th} + R_L\right)^2 + \left(X_{Th} + X_L\right)^2\right]^2}$$

Igualando a cero $\frac{\partial P}{\partial X_L}$ obtenemos

$$X_L = -X_{Th}$$

E igualando a cero $\frac{\partial P}{\partial R_L}$,

$$R_L = \sqrt{\left(R_L\right)^2 + \left(X_{Th} + X_L\right)^2}$$

Y si $X_L = -X_{Th}$, $R_L = R_{Th}$. Por lo que para obtener la máxima potencia la impedancia del componente debe ser:

$$Z_L = R_L + jX_L = R_{Th} - jX_{Th}$$

Ahora se pretende encontrar estos valores por medio de un AG que maximice la función P de potencia variando los parámetros R_L y X_L, dados R_{th} y X_{Th} fijos.

El algoritmo genético

1. Problema de optimización

La función objetivo es la ecuación de la potencia, la cual se muestra a continuación junto con una gráfica representativa (figura 5.2).

$$P = \frac{1}{2} R_L \frac{\left(V_{Th}\right)^2}{\left(R_{Th} + R_L\right)^2 + \left(X_{Th} + X_L\right)^2} \text{ Función objetivo}$$

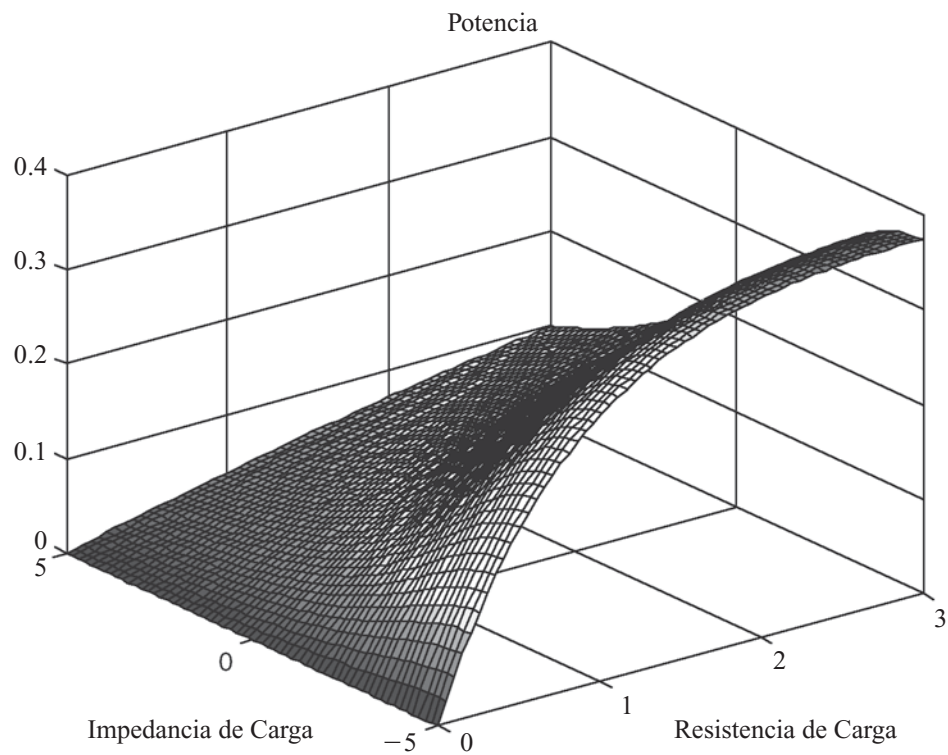


Fig. 5.2 Función de potencia.

2. Representación

Es necesario codificar las variables de decisión en una cadena binaria. Aquí se utilizan 16 bits para representar una variable. La asignación de una cadena binaria a un número real para las variables R_L y X_L se completa como sigue:

variable = valor decimal*((rango(sup)-rango(inf))/(2^length(bit)-1)) + range(inf)

Por ejemplo, la longitud total de un cromosoma es de 32 bits, 16 para la variable R_L y 16 para X_L:

1110011011001100 0110101110000010

Entonces:

Número binario	**Valor decimal**
1110011011001100	59076
0110101110000010	27522

Suponiendo que para ambas variables el rango superior es 2 y el inferior -2,

$$\text{Variable}R_L = \left(59706 \times \frac{4}{2^{16}-1}\right) - 2 = 1.606$$

$$\text{Variable}X_L = \left(27522 \times \frac{4}{2^{16}-1}\right) - 2 = -0.320$$

3. Población inicial

En cada generación el tamaño de la población es 20. La población inicial se genera aleatoriamente como se ve en la tabla 5.1.

Tabla 5.1 Población inicial

#	Población inicial (Binario)	Pob. Inicial (Decimal)
1	0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1	f(1.342870, -1.197604)=0.181367
2	1 0 0 1 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 0	f(1.710826, 2.100938)=0.106021
3	0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0 0 1	f(0.362737, 0.462882)=0.039666
4	1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 1 0 0 1	f(2.515084, 1.395514)=0.158694
5	1 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 0 1 1 1 1 1 0 1 1 0 0 1 1 0 1	f(2.763928, -2.546807)=0.316956
6	1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1	f(2.251087, -2.055161)=0.279476
7	0 1 0 0 0 0 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1	f(0.790753, -1.404211)=0.130346
8	0 0 0 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1	f(0.246738, 2.004654)=0.018628
9	1 0 1 0 0 1 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 0	f(1.961547, -2.599145)=0.290542
10	0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 1 0	f(0.097276, 2.119554)=0.007262
11	1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 0 0 0 1 0	f(2.486015, 2.349050)=0.133013
12	0 1 0 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 0 0 1 1	f(0.902220, -1.615702)=0.152169
13	1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 0 0	f(2.449256, -4.546502)=0.368616
14	1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 0 1	f(2.551888, 3.133211)=0.118420
15	0 1 1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0	f(1.147356, 4.861601)=0.045112
16	1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0	f(2.191485, -0.472648)=0.207840
17	1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0	f(2.155871, -4.804379)=0.364424
18	1 1 1 0 1 0 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 0 1	f(2.755459, 4.443046)=0.101389
19	0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 1 1	f(0.488029, 3.070039)=0.028413
20	0 0 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 1 1	f(0.360403, -3.819104)=0.127834

4. Evaluación

El primer paso después de la creación de una generación es calcular el valor de aptitud de cada miembro. El proceso de evaluación de la aptitud de un cromosoma consta de tres pasos:

1) Convertir el genotipo del cromosoma a su fenotipo. Esto significa convertir a las cadenas binarias en los correspondientes valores reales.
2) Evaluar la función objetivo.
3) Convertir el valor de la función objetivo en un valor de ajuste. Aquí, a fin de sólo tener valores positivos, la aptitud de cada cromosoma es igual a la función objetivo evaluada para cada cromosoma, menos el mínimo valor obtenido para la función objetivo.

Los valores de la función objetivo F y la aptitud de los cromosomas de los valores Eval de la población que se muestra en la tabla 5.1, son los que presenta la tabla 5.2.

Tabla 5.2 Ajuste de la población inicial

#	Función objetivo	Eval (Ajuste) = F(#) − Fmin
1	f(1.342870, -1.197604)=0.181367	0.174
2	f(1.710826, 2.100938)=0.106021	0.099
3	f(0.362737, 0.462882)=0.039666	0.032
4	f(2.515084, 1.395514)=0.158694	0.151
5	f(2.763928, -2.546807)=0.316956	0.310
6	f(2.251087, -2.055161)=0.279476	0.272
7	f(0.790753, -1.404211)=0.130346	0.123
8	f(0.246738, 2.004654)=0.018628	0.011
9	f(1.961547, -2.599145)=0.290542	0.283
10	f(0.097276, 2.119554)=0.007262	0.000
11	f(2.486015, 2.349050)=0.133013	0.126
12	f(0.902220, -1.615702)=0.152169	0.145
13	f(2.449256, -4.546502)=0.368616	0.361
14	f(2.551888, 3.133211)=0.118420	0.111
15	f(1.147356, 4.861601)=0.045112	0.038
16	f(2.191485, -0.472648)=0.207840	0.201
17	f(2.155871, -4.804379)=0.364424	0.357
18	f(2.755459, 4.443046)=0.101389	0.094
19	f(0.488029, 3.070039)=0.028413	0.021
20	f(0.360403, -3.819104)=0.127834	0.121

En esta primera generación, el cromosoma con mejor ajuste es el 13 y el peor el 10, ya que se quiere un máximo.

5. Crear una nueva población

Después de la evaluación, tenemos que crear una nueva población de la generación actual. Se utilizan los tres operadores: reproducción, cruce y mutación.

Reproducción: Los dos cromosomas (cadenas) con la aptitud pueden "vivir" y producir descendencia en la próxima generación. Por ejemplo, en la primera población, a los cromosomas 13 y 16 se les permite vivir en la segunda población.

Selección y cruce: La probabilidad acumulada se utiliza para decidir qué cromosomas se seleccionarán para cruce. La probabilidad acumulada se calcula en los siguientes pasos:

a) Calcular el ajuste total de la población.

$$\text{AjusteTotal} = \sum_{i=1}^{\text{TamañoPob.}} \text{Ajuste}(i)$$

b) Calcular la probabilidad P_i para cada cromosoma.

$$P_i = \frac{\text{Ajuste}(i)}{\text{AjusteTotal}}$$

c) Calcular la probabilidad acumulada Q_i para cada cromosoma.

$$Q_k = \sum_{k=0}^{i} P_k$$

Para la población mostrada anteriormente el ajuste total es de 3.03096 y la probabilidad de selección y acumulada para cada individuo se muestra en la tabla 5.3:

Tabla 5.3 Probabilidad de selección y probabilidad acumulada para cada miembro.

#	P_i	Q_i
1	0.057	0.057
2	0.033	0.090
3	0.011	0.101
4	0.050	0.151
5	0.102	0.253
6	0.090	0.343
7	0.041	0.383
8	0.004	0.387
9	0.093	0.480
10	0.000	0.480
11	0.041	0.522
12	0.048	0.570
13	0.119	0.689
14	0.037	0.726
15	0.012	0.738
16	0.066	0.804
17	0.118	0.922
18	0.031	0.953
19	0.007	0.960
20	0.040	1.000

El cruce que se utiliza aquí es el método de punto de corte (Xover point), que selecciona al azar un punto de corte e intercambia la parte derecha de los padres para generar descendencia.

a) Generar un número aleatorio *r* en el intervalo [0,1].

b) Si Qi-1 $<$ *r* $<=$ Qi, se selecciona el i-enésimo cromosoma para ser el padre 1.

c) Repita el paso a y b para reproducir los padres dos.

d) Generar un número aleatorio *r* en el intervalo [0,1]. Si *r* es menor que la probabilidad de cruce (elegimos la probabilidad de cruce igual a 1,0), el cruce se realiza, el punto de corte se selecciona detrás del gen cuyo lugar es el entero más cercano (mayor o igual) a *rx* (longitud(bit)-1). En este caso, la longitud es de 32 bits.

e) Se repite del paso a al paso d, en total nueve veces para terminar el cruce. La creación de 18 hijos más 2 cromosomas (lo más aptos de la anterior) mantiene la población de 20 en cada generación.

Por ejemplo, el primer cruce de la población mostrada es:

Punto de cruce = 22

Padre1

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

Padre2

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

Nueva población1

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

Nueva población2

1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1

Mutación: La mutación se realiza después del cruce. La mutación altera uno o más genes con una probabilidad igual a la tasa de mutación. En este caso, la probabilidad de mutación es 0.01.

a) Generar una secuencia de números aleatorios de r_k ($k = 1, \ldots, 640$) (El número de bits total en una población 20 _ 32 = 640).

b) Si r_i es 1, se cambia el bit i-enésimo de 1 a 0 o de 0 a 1.

c) Los cromosomas elite de la población anterior no están sujetos a mutación, así que después de la mutación deben ser restaurados.

Una nueva población se crea después de una iteración del AG. Este procedimiento se puede repetir las veces deseadas, pero en este ejemplo se cierra el ciclo cuando no se muestra una mejoría en el mejor miembro de cada generación después de 20 iteraciones. Para este ejemplo, se tomó el voltaje de Thévenin como 3.0 volts, la resistencia de Thévenin como 3 ohms y la reactancia como 5*j*. Se muestra al individuo con el mejor ajuste en cada iteración.

$$V_{Th} = 3.0\left[V\right] \qquad R_{Th} = 3.0\left[\Omega\right] \qquad X_L = 5.0\,j\left[\Omega\right]$$

Generación 2: f(2.449256, -4.546502)=0.368616
Generación 3: f(2.449256, -4.649958)=0.369644
Generación 4: f(2.824262, -4.533989)=0.372275
Generación 5: f(2.824262, -4.533989)=0.372275
Generación 6: f(2.825727, -4.533837)=0.372281
Generación 7: f(2.825727, -4.649958)=0.373317
Generación 8: f(2.825727, -4.649958)=0.373317
Generación 9: f(2.825727, -4.649958)=0.373317
Generación 10: f(2.918013, -4.649958)=0.373621
Generación 20: f(2.966354, -4.962463)=0.374973
Generación 30: f(2.966354, -4.985657)=0.374986
Generación 40: f(2.967269, -4.998474)=0.374989
Generación 50: f(2.973953, -4.996033)=0.374993
Generación 60: f(2.976608, -4.995422)=0.374994
Generación 70: f(2.997391, -4.998779)=0.375000
Generación 80: f(2.998535, -4.998779)=0.375000

Generación 90: f(2.998718, -5.000000)=0.375000
Generación 100: f(2.999268, -5.000000)=0.375000
Generación 110: f(3.000000, -5.000000)=0.375000
Generación 111: f(3.000000, -5.000000)=0.375000
Generación 112: f(3.000000, -5.000000)=0.375000
Generación 113: f(3.000000, -5.000000)=0.375000
Generación 114: f(3.000000, -5.000000)=0.375000
Generación 115: f(3.000000, -5.000000)=0.375000
Generación 116: f(3.000000, -5.000000)=0.375000
Generación 117: f(3.000000, -5.000000)=0.375000
Generación 118: f(3.000000, -5.000000)=0.375000
Generación 119: f(3.000000, -5.000000)=0.375000
Generación 120: f(3.000000, -5.000000)=0.375000
Generación 121: f(3.000000, -5.000000)=0.375000
Generación 122: f(3.000000, -5.000000)=0.375000
Generación 123: f(3.000000, -5.000000)=0.375000
Generación 124: f(3.000000, -5.000000)=0.375000
Generación 125: f(3.000000, -5.000000)=0.375000
Generación 126: f(3.000000, -5.000000)=0.375000
Generación 127: f(3.000000, -5.000000)=0.375000

Máxima potencia: 3.750000e-001 Watts

Cuando: Rth = 3.000000 ohms, Xth = -5.000000j ohms

Así comprobamos lo que matemáticamente sabíamos: la máxima potencia se alcanza cuando la impedancia del componente conectado a un circuito ac es el conjugado de la impedancia de dicho circuito.

Las figuras 5.3 y 5.4 muestran, respectivamente, el mapa de contornos de la función potencia y la ubicación de los miembros creados por el AG para la décima generación y la última para ver su convergencia.

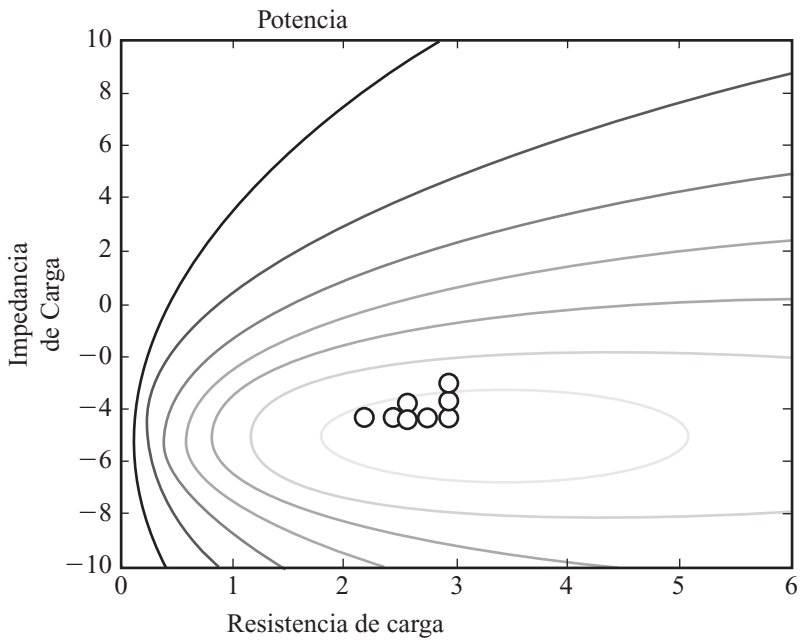


Fig. 5.3 Mapa de contornos de la función objetivo con la población inicial.

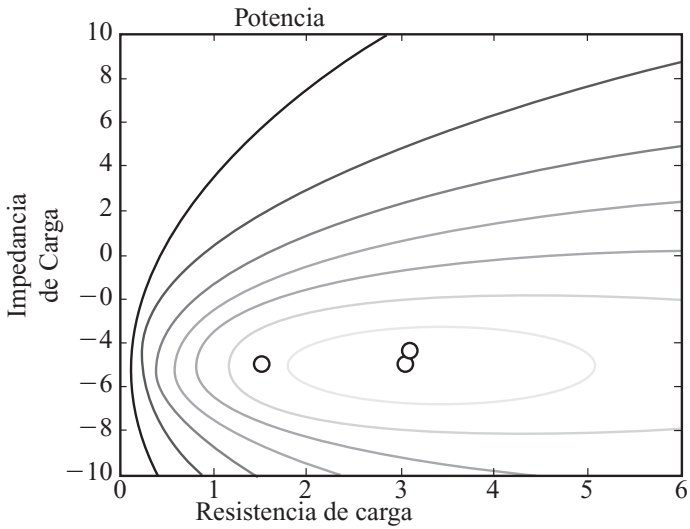


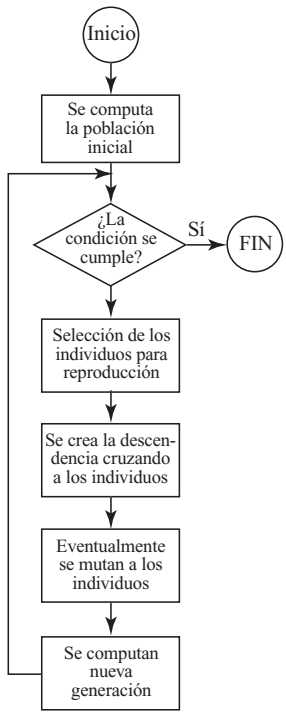
Fig. 5.4 Mapa de contornos de la función objetivo con la última generación.

Algoritmos genéticos

A continuación el lector encontrará los diagramas de flujo y programas de los algoritmos genéticos presentados en este capitulo, desarrollados para MATLAB®.

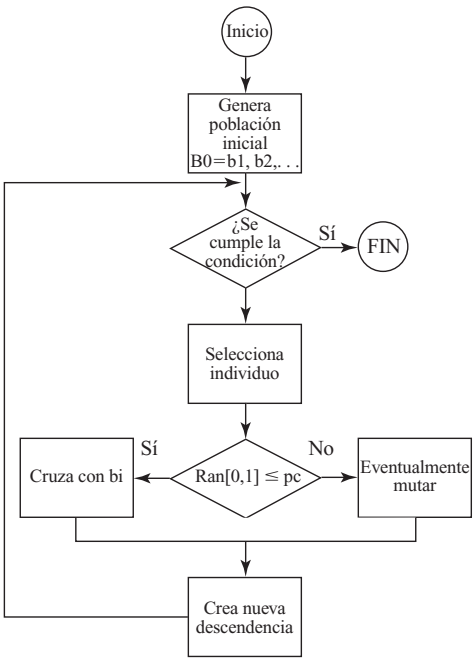
Empleando estos códigos como base siéntase el lector en la libertad de modificar los diagramas de flujo y programas, para generar el proceso de optimización de diferentes casos y condiciones, así como tomar en cuenta diferentes límites de optimización.

Algoritmo genético básico convencional binario



```
Algoritmo
t:=0
Se computa la población inicial Bo.
WHILE definir la condición que no se cumple DO
BEGIN
Se selecciona a los individuos para la reproducción;
Se crea la descendencia cruzando a los individuos;
Eventualmente se mutan a los individuos;
Se computa la nueva generación
END
```

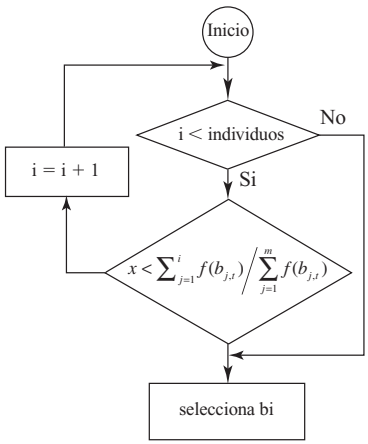
Algoritmo generación de nuevos individuos mediante operaciones de cruza y mutación




```
Algoritmo
t:=0
Se computa la población inicial B0 = (b1,0, b2,0,…);
WHILE se define la condición que no se cumple DO
BEGIN
FOR i:=1 TO m DO
Se selecciona un individuo bi,,t de Bt;
FOR j:=1 TO m-1 STEP 2 DO
```

```
IF Random [0,1] ≤pc THEN
cruza bi,t con bj,t;
FOR i:=1 TO m DO
Eventualmente mutar
Se crea descendencia cruzando a los individuos;
Eventualmente se mutan bi,t+1;
t:=t+1
END
```

Algoritmo de selección proporcional o ruleta




```
Algoritmo
x: = Random [0,1];
i: = 1
WHILE i<m & x < Σ_{j=1}^{i} f(b_{j,t}) / Σ_{j=1}^{m} f(b_{j,t}) DO
i: i+1;
selecciona bi,t;
```

ARCHIVOS M DE MATLAB PARA LOS ALGORITMOS

Main

```
popuSize = 20; % Población inicial
xover_rate = 1.0; % Tasa de cruzamiento
mutate_rate = 0.01; % Tasa de mutación
bit_n = 16; % Número de bits para cada variable
limit = 0;
```

```
global Vth;
global Rth;
global Xth;
%Se pide por los valores de entrada
Vth=input('Valor del voltaje Vth: ');
if isempty(Vth)
    Vth=0;
end
Rth=input('Valor de la resistencia Rth: ');
if isempty(Rth)
    Rth=0;
end
Xth=input('Valor de la impedancia Xth: ');
if isempty(Xth)
    Xth=0;
end

obj_fcn = 'G_Pfunction2';% Función objetivo
var_n = 2; % Número de variables de entrada
range = [0, Rth; -Xth, Xth]; % Rango de la entradas
% Gráfica de la función de potencia
syms RL;
syms XL;
z = (0.5*(Vth^2)*RL)/((Rth+RL)^2+(Xth+XL)^2);
figure;
ezsurf(z,[0,Rth,-Xth,Xth]);
xlabel('Load Resistance'); ylabel('Load Impedance'); title('Potencia');
% Mapa de contornos de la función potencia
figure;
ezcontour(z,[0,2*Rth,-2*Xth,2*Xth]);
hold off;
xlabel('Load Resistance'); ylabel('Load Impedance'); title('Potencia');

% Creación de la población inicial
popu = rand(popuSize, bit_n*var_n) > 0.5;

fprintf('Población inicial.\n');
for i=1:popuSize
for j=1:bit_n*var_n
```

```
fprintf('%1.0f ',popu(i,j));
end
fprintf('\n');
end

upper = zeros(50, 1); %Creación de la matriz de mejor individuo

% Loop Principal
i=0;
%Cuando el ind. más apto no mejora después de 20 generaciones el GA
%termina
while(limit <=20)
i=i+1;
k=i;
% Reseteo de variables
delete(findobj(0, 'tag', 'member'));
delete(findobj(0, 'tag', 'count'));
% Evaluación de la función para cada miembro de la población
fcn_value = evalpopu(popu, bit_n, range, obj_fcn);
if (i==1),
fprintf('Población inicial\n ');
for j=1:popuSize
fprintf('f(%f, %f)=%f\n', ...
bit2num(popu(j, 1:bit_n), range(1,:)), ...
bit2num(popu(j, bit_n+1:2*bit_n), range(2,:)), ...
fcn_value(j));
end
end

% Llenar la matriz con los mejores individuos
upper(i) = max(fcn_value);

%Registro de si el ind. más apto mejora o no
if(i>=2)
    if(upper(i)==upper(i-1))
        limit = limit +1;
    else
        limit=0;
    end
end
```

```
 % Gráfica de los mejores individuos
[best, index] = max(fcn_value);
fprintf('Generación %i: ', i);
fprintf('f(%f, %f)=%f\n', ...
bit2num(popu(index, 1:bit_n), range(1,:)), ...
bit2num(popu(index, bit_n+1:2*bit_n), range(2,:)), ...
best);
% Creación de la siguiente población con selección, cruce y mutación
popu = nextpopu(popu, fcn_value, xover_rate, mutate_rate,k);
if(mod(i,10)==0)
fprintf('Población después de la %d º generación.\n',i);
fprintf('Presione cualquier tecla...\n');
pause;
end
%end
end
[best, index] = max(fcn_value);
fprintf('\nMaxima potencia: %i Watts\n ', best);
fprintf('Cuando: Rth = %f ohms, Xth = %fj ohms\n', ...
bit2num(popu(index, 1:bit_n), range(1,:)), ...
bit2num(popu(index, bit_n+1:2*bit_n), range(2,:)));
```

Función objetivo

```
function z = G_Pfunction2(input)
%Regresa el valor de la función de acuerdo al input.

global PREV_PT % Salida de la función anterior
global Vth;
global Rth;
global Xth;
RL= input(1); XL = input(2);
z = (0.5*(Vth^2)*RL)/((Rth+RL)^2+(Xth+XL)^2);
% Gráfica de la función potencia
property='Marker';
line(RL, XL, property, 'o', 'markersize', 10, ...
'clipping', 'off', 'erase', 'xor', 'color', 'k', ...
'tag', 'member', 'linewidth', 2);
```

```
if ~isempty(PREV_PT),% Actualiza la gráfica
line([PREV_PT(1) RL], [PREV_PT(2) XL], 'linewidth', 1, ...
'clipping', 'off', 'erase', 'none', ...
'color', 'w', 'tag', 'traj');
end

PREV_PT = [RL XL];
drawnow;
```

Eval. Población

```
function fitness = evalpopu(popu, bit_n, range, obj_fcn)
%EVALPOPU Evaluación de ajuste de la población.

global count %Contador global de individuos
pop_n = size(popu, 1);
fitness = zeros(pop_n, 1);
%Evaluación miembro por miembro
for count = 1:pop_n,
fitness(count) = evaleach(popu(count, :), bit_n, range, obj_fcn);
end
```

Eval. Each

```
function out = evaleach(string, bit_n, range, obj_fcn)
% EVALEACH Evaluación de cada individuo.
% string: cadena de bits que representan un individuo.

var_n = length(string)/bit_n;
input = zeros(1, var_n);
%Ciclo para convertir cada individuo de bits a decimal
for i = 1:var_n,
input(i) = bit2num(string((i-1)*bit_n+1:i*bit_n), range(i, :));
end
out = feval(obj_fcn, input);%Regresa el valor de salida
```

Convertir BIT2NUM

```
function num = bit2num(bit, range)
% BIT2NUM Conversión de cadena de bits a número decimal.
integer = polyval(bit, 2);
num = integer*((range(2)-range(1))/(2^length(bit)-1)) + range(1);
```

NEXT Population

```
function new_popu = nextpopu(popu, fitness, xover_rate, mut_rate,k)
new_popu = popu;
popu_s = size(popu, 1);
string_leng = size(popu, 2);
% ====== ELITISMO: Se conservan los mejores 2 individuos
tmp_fitness = fitness;
[a, index1] = max(tmp_fitness); % Encuentra al mejor
tmp_fitness(index1) = min(tmp_fitness);
[a, index2] = max(tmp_fitness); % Encuentra al segundo mejor
new_popu([1 2], :) = popu([index1 index2], :);
% Se reescala la function de ajuste
fitness = fitness - min(fitness);% Positiva
total = sum(fitness);
if(k==1)
fprintf('Función de ajsute despues de nueva escala\n');
for i=1:popu_s
fprintf('%10.3f \n',fitness(i));
end
fprintf('La suma de cada ajuste %10.5f\n',total);
end
if total == 0,
fprintf('=== Error ===\n');
fitness = ones(popu_s, 1)/popu_s;% sum is 1
else
fitness = fitness/sum(fitness); % sum is 1
end
cum_prob = cumsum(fitness);
if(k==1)
```

```
fprintf('La probabilidad de cada cromosoma, y su prob. acumulada \n');
for i=1:popu_s
fprintf('%10.3f %10.3f\n',fitness(i),cum_prob(i));
end
end
% ====== SELECCIÓN Y CRUZAMIENTO
for i = 2:popu_s/2,
% === Se seleccionan dos padres de acuerdo con su nivel de ajuste
tmp = find(cum_prob - rand > 0);
parent1 = popu(tmp(1), :);
tmp = find(cum_prob - rand > 0);
parent2 = popu(tmp(1), :);
% === Se determina si se cruza o no
if rand < xover_rate,
% Operación de cruzamiento
xover_point = ceil(rand*(string_leng-1));
new_popu(i*2-1, :) = ...
[parent1(1:xover_point) parent2(xover_point+1:string_leng)];
new_popu(i*2, :) = ...
[parent2(1:xover_point) parent1(xover_point+1:string_leng)];
end
if(k==1)
fprintf('Punto de cruce = %d \n', xover_point);
fprintf('Padre1\n');
for j=1:string_leng
fprintf('%d ',parent1(j));
end
fprintf('\n');
fprintf('Padre2\n');
for j=1:string_leng
fprintf('%d ',parent2(j));
end
fprintf('\n');
fprintf('Nueva población1\n');
for j=1:string_leng
fprintf('%d ',new_popu(i*2-1,j))
end
fprintf('\n');
fprintf('Nueva población2\n');
```

```
for j=1:string_leng
fprintf('%d ',new_popu(i*2,j))
end
fprintf('\n');
end
end

if(k==1)
fprintf('El resultado después del cruce de la primera población\n');
for i=1:popu_s
for j=1:string_leng
fprintf('%d ',new_popu(i,j))
end
fprintf('\n');
fprintf('\n');
end
end
% ====== MUTACION (los elites no se mutan)
mask = rand(popu_s, string_leng) < mut_rate;
new_popu = xor(new_popu, mask);
if(k==1)
fprintf('El resultado de la mutación de la 1a población\n');
for i=1:popu_s
for j=1:string_leng
fprintf('%d ',new_popu(i,j))
end
fprintf('\n');
fprintf('\n');
end
end
% Se restauran los miembros elite
new_popu([1 2], :) = popu([index1 index2], :);
```

Anexo A

MATLAB® GENETIC ALGORITHMS TOOLBOX

INTRODUCCIÓN

Este documento tiene la finalidad de conjuntar las funciones de *gatool* algoritmos genéticos de MATLAB® para que al usuario le sea más fácil encontrar algunas respuestas a sus dudas sobre el empleo de esta caja de herramientas. Para mayor documentación se recomienda al usuario revisar el manual de MATLAB®.

Genetic Algorithm Tool. Es una interfaz gráfica que le permite al usuario utilizar y diseñar algoritmos genéticos (AG) sin trabajar en la línea de comandos (también se puede configurar todas las opciones que se presentan en este documento mediante la línea de comandos). Para abrir esta herramienta se escribe:

gatool

La interfaz de Genetic Algorithm Tool se divide en cuatro secciones como se aprecia en la figura A.1.

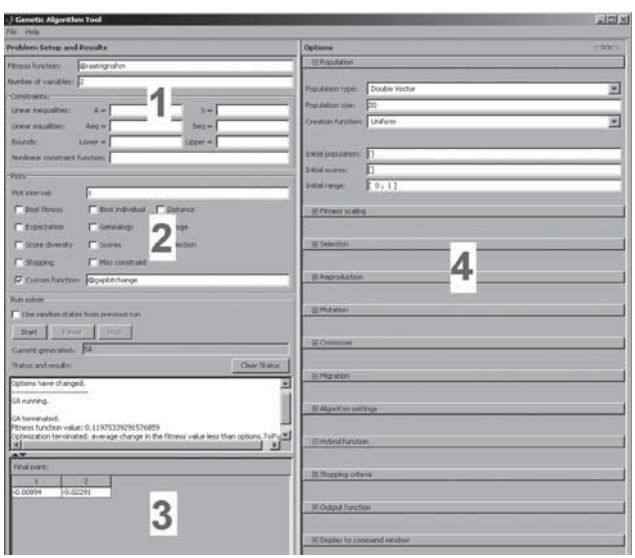


Fig. A.1 Secciones de Genetic Algorithm Tool.

1. Es la parte donde se declara la función que se va a evaluar, con sus respectivas restricciones.
2. Área de gráficos, donde existen diversos trazos que se pueden realizar.
3. Área donde se muestra los resultados de la función aptitud.
4. En esta sección se encuentran las opciones para mejorar y alterar la función, optimizando los AG.

SECCIÓN 1. DECLARACIÓN DE FUNCIÓN APTITUD Y RESTRICCIONES

Fitness function/Función aptitud. Es la función objetivo que se desea minimizar o maximizar. Antes de evaluar la función es necesario crear la función con sus respectivas características en un archivo M-file. Se guarda el archivo con el nombre deseado, por ejemplo "new"; es muy importante recordar el nombre, ya que al evaluar la función es necesario escribir el nombre del archivo para evaluar dicha función en el campo correspondiente, como se muestra en la figura A.2.

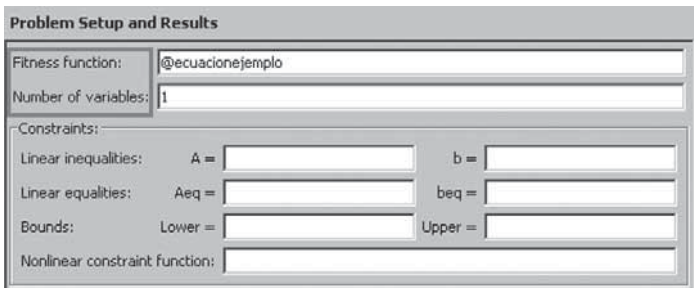


Fig. A.2 Ventana para la declaración de la función que se va a optimizar y su número de variables.

Number of variables/ Número de variables. En la opción se escribe la cantidad de variables independientes que se desea tener para la función aptitud; el campo correspondiente también se indica en figura A.2.

Para que el usuario tenga una mayor comprensión del uso de esta herramienta, se realizará un ejemplo con el cual sea posible interpretar los resultados y gráficas obtenidos.

Ejemplo

Se desea obtener el valor mínimo de la función aptitud:

$$y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1;$$

para ello se necesita crear un *archivo .m*, el cual se declarará de la siguiente manera:

```
function [y]=ejemexpag(x)
y=(1/3)*(x.^3)+(3/2)*(x.^2)-(4*x)-1
```

Después se guarda el documento con el nombre deseado, en este caso "*ecuacionejemplo*"; se puede abrir la herramienta de algoritmos genéticos gatool. Se declara la función que se va a optimizar y el número de variables, que en este caso es "*x*".

Antes se graficará la función para conocer de qué tipo es y conocer su comportamiento. Primero se declara los límites de la función como se muestra en el siguiente código.

```
x=−1:0.1:10;
y=ecuacionejemplo(x);
plot(x,y)
```

El resultado debe ser como el de la figura A.3.

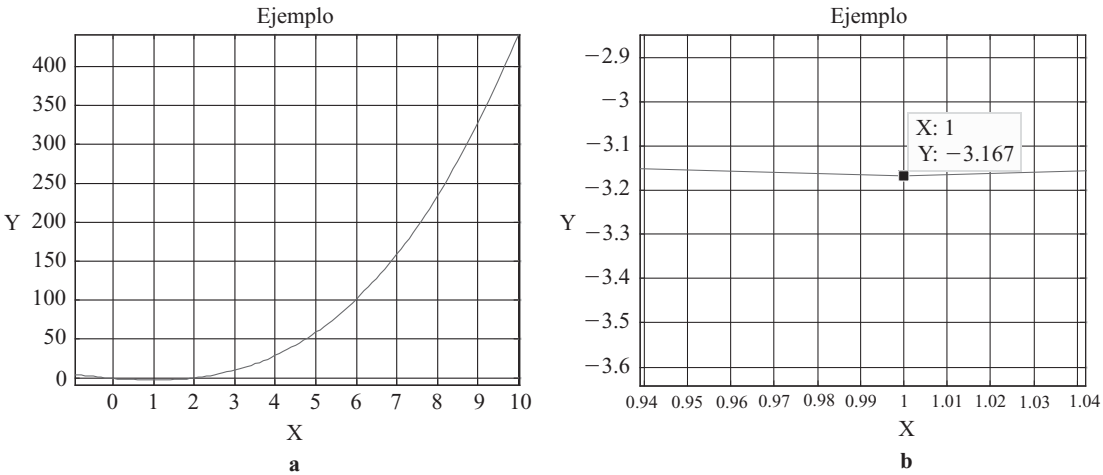


Fig. A.3 a) Gráfica de la función aptitud $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$
b) Acercamiento al punto mínimo de la función.

Los algoritmos genéticos son metodologías para optimizar las funciones, ya que buscan al mejor individuo que cumpla con las condiciones de restricción que se desea; esto puede significa encontrar el máximo o el mínimo a la redonda de la función que se evalúe. Se debe considerar que existen mínimos y máximos relativos que no son los máximos y mínimos globales de las funciones.

En este caso en particular se puede resolver en forma analítica este problema, ya que su complejidad no es alta y la funcionalidad del ejemplo es poder realizar de manera analítica y después validar el resultado encontrado a través de algoritmos genéticos.

Se tiene entonces $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$ derivando

$$\frac{dy}{dx} = x^2 + 3x - 4$$

Igualando con cero la derivada y encontrando las raíces se obtiene:

$(x + 4)(x - 1) = 0$

Esto muestra que existen dos raíces respuestas las cuales pueden ser máximo, mínimos o ninguno de éstos.

Se evalúa en la segunda derivada para conocer el máximo y mínimo. En las raíces encontradas

$$2x+3=\frac{d^2y}{dx}$$

Evaluando la segunda derivada en $x = 1$ y $x = -4$, que son las raíces encontradas, se puede aplicar el criterio siguiente.

Cuando $x < 0$ se obtiene un máximo; por el contrario, si se tiene un valor $x > 0$ se tendrá un mínimo. Basados en los resultados se deduce que el valor mínimo está en $x_1 = 1$.

Si se desea conocer en qué punto se interseca con *y*, se evalúa la función inicial

$y=\frac{1}{3}x^3+\frac{3}{2}x^2-4x-1$ con $x = -4$ y $x = 1$. Entonces:

$$y=\frac{1}{3}x^3+\frac{3}{2}x^2-4x-1=\frac{1}{3}(-4)^3+\frac{3}{2}(-4)^2-4(-4)-1=\frac{53}{3}=17.666$$

Se obtendrá el mínimo evaluando $y=\frac{1}{3}x^3+\frac{3}{2}x^2-4x-1$ en $x_2 = 1$ entonces se tiene un valor igual a -3.1667.

Por lo tanto, las coordenadas son *máx* (-4,17.666) y *mín* (1,-3.1666): En la figura A.4 se muestran respectivamente los puntos.

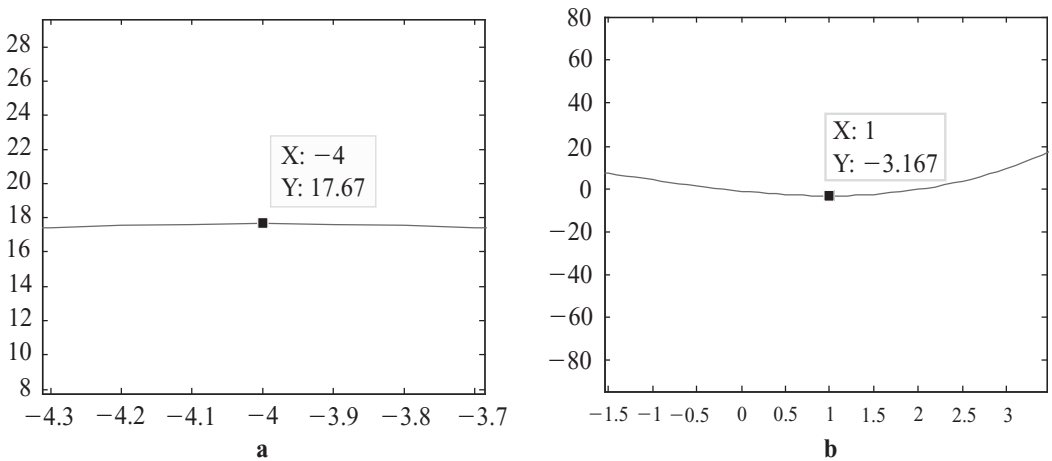


Fig. A.4 La gráfica (a) muestra el máximo de la función y (b) el mínimo.

Regresando a la herramienta gatool la respuesta será -3.1666 y el punto 0.9934, lo cual es una aproximación muy buena porque nosotros encontramos un valor de -3.1667 con un valor de x=1, pero se debe recordar que la ecuación propuesta es muy simple y avanzado en complejidad la aproximación del resultado variará, por lo que debemos tener en cuenta que los algoritmos genéticos son aproximaciones al resultado que buscamos, son métodos de búsqueda no puntuales, son métodos de búsqueda expandidos.

Constraints/Restricciones

Linear inequalities /Desigualdades lineales. Se expresan en la forma A * $x \leq b$; se especifican con la matriz A y el vector *b*.

Linear equalities/Igualdades lineales. Se expresan en la forma Aeq * x = beq; se especifican por la matriz Aeq y el beq vector.

Las restricciones anteriores se muestran en la figura A.5.

La función GA asume que la función de restricción tendrá una entrada x, donde x tiene tantos elementos como el número de variables en el problema. La función de restricción calcula los valores de la desigualdad y de todas las restricciones de igualdad, devolviendo dos vectores c y CEQ, respectivamente.

Los límites para la función aptitud se definen como parte inferior y superior, los cuales se señalan en la figura A.5 como Bounds Lower y Upper, respectivamente.

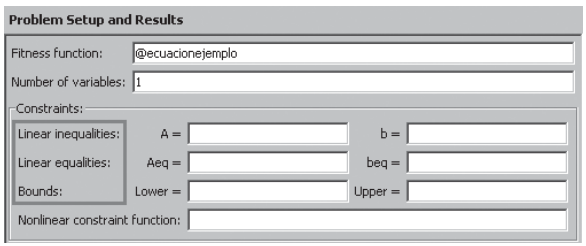


Fig. A.5 Declaración de intervalos para la función new.

A continuación se presentan algunos ejemplos de restricciones y limitaciones.

$(x_1)(x_2) + x_1 - x_2 + 1.5 \leq 0,$ (restricción no lineal)

$10 - (x_1)(x_2) \leq 0,$ (restricción no lineal)

$0 \leq x_1 \leq 1,$ y (límite)

$0 \leq x_2 \leq 13$ (límite)

SECCIÓN 2. ÁREA DE GRÁFICOS

Plot interval/Intervalo de trazado. Especifica el número de generaciones entre las que se llaman de forma consecutivas y las funciones que se grafican. En esta sección se hallan las alternativas que puede graficar el usuario, dependiendo del enfoque de estudio que desee. A continuación se presentan cada una de ellas y su función; el usuario puede encontrar estas opciones en la parte del GA Tool que se muestra en la figura A.6.

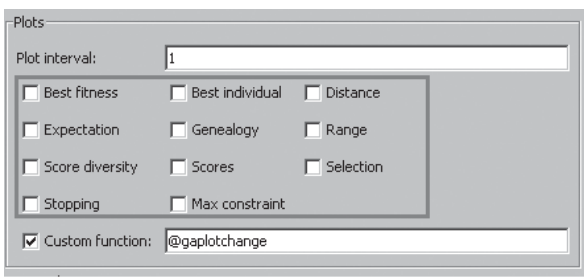


Fig. A.6 Área de gráficos.

Best fitness/Mejor aptitud. Grafica el mejor valor de la función aptitud de cada generación contra el número de iteraciones. Esto se traduce en que muestra la mejor característica de cada generación contra el número de épocas (figura A.7).

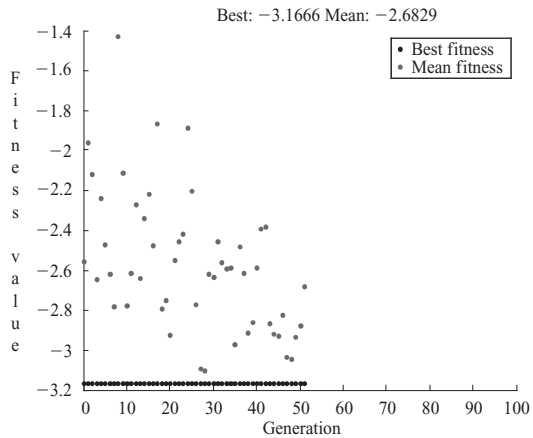


Fig. A.7 Muestra el resultado de la mejor aptitud en la función $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$. En este caso, como ya se sabe el resultado fue -3.1666.

Best individual/Mejor individuo. Grafica las entradas del vector del individuo con el mejor valor de la función aptitud de cada generación. Muestra al individuo más apto de cada generación, lo que se ejemplifica en la figura A.8.

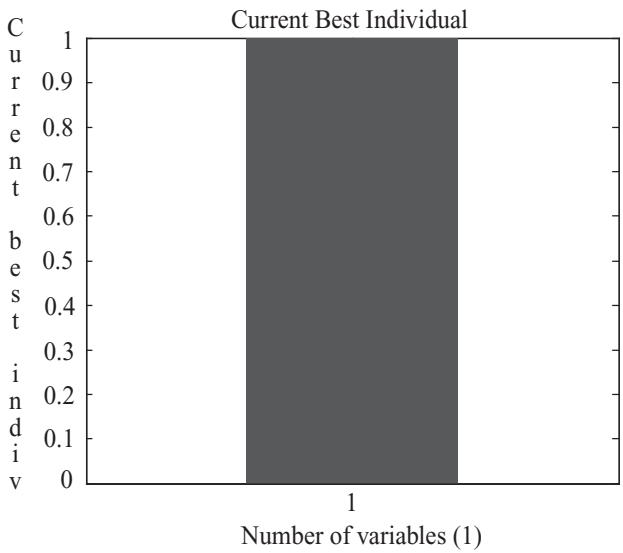


Figura A.8 Muestra al individuo más apto de $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$, donde basados en los cálculos anteriores se sabe que es .9999, o con exactitud 1.

Distance/Distancia. La distancia que existe entre los individuos se le conoce como diversidad. En una población donde se tiene gran diversidad, se refiere a la gran distancia existente entre los individuos. La diversidad es esencial para el AG, ya que permite al algoritmo buscar en regiones más amplias del espacio. Si la distancia entre los individuos es muy grande tardará mayor tiempo en converger o, en caso extremo, divergir. Por el contrario, si los individuos guardan una distancia cercana, éstos tendrán características similares, lo cual no es conveniente para los AG. La figura A.9 muestra la distancia entre los individuos en cada generación, con lo que se puede observar su diversidad.

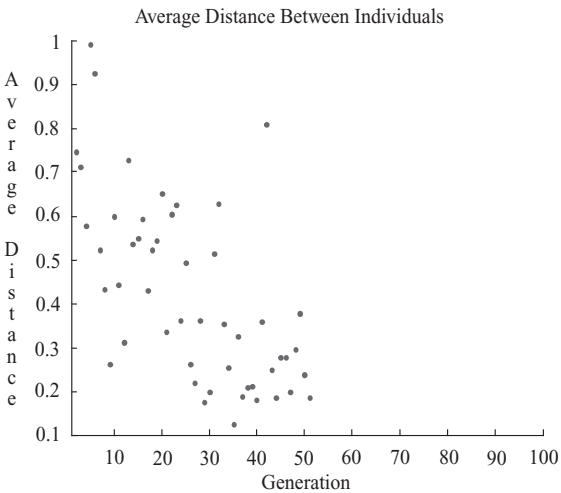


Fig. A.9 Distintas distancias existentes en las generaciones para $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$.

Expectation/Expectativa. Traza el número previsto de hijos contra la puntuación (de baja calidad) en cada generación. Ejemplo en la figura A.10.

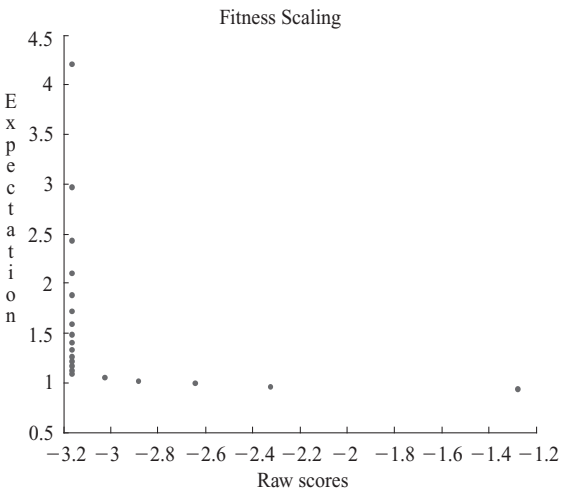


Fig. A.10 Se aprecian los hijos creados en cada época *versus* su puntuación.

Genealogy/Genealogía. Grafica la genealogía de los individuos. Las líneas a partir de una generación a la siguiente son de colores codificados; que se presentan en pantalla, a continuación se mencionan los colores y su significado:

- El trazado rojo indica niños mutados.
- El trazado azul indica niños cruzados.
- El trazado negro indica niños elite.

Las distintas líneas que representan la genealogía se pueden ver en la figura A.11.

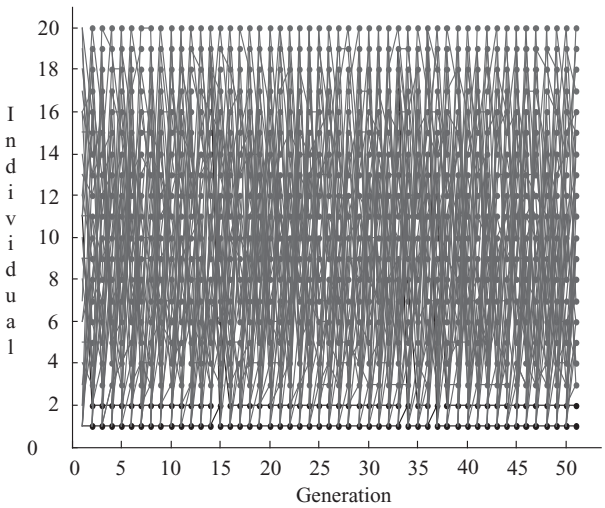


Fig. A.11 El árbol genético de los individuos (hijos) de $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$.

Range/Rango. Traza el máximo, el mínimo y los valores medios de la función aptitud en cada generación, como se observa en la figura A.12.

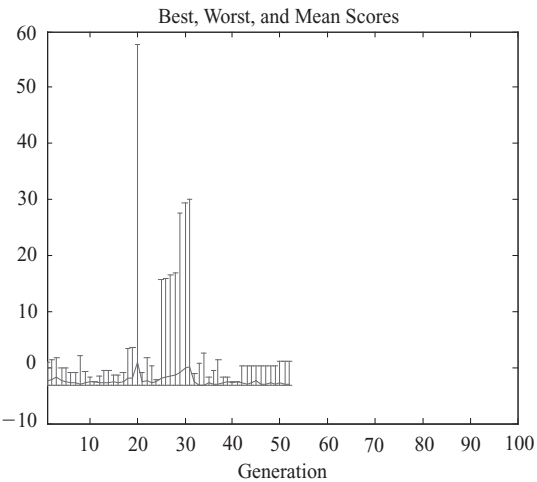


Fig. A.12 Muestra el máximo, mínimo y la media de la función propuesta $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$.

Score diversity/Diversidad de puntuación. Traza los valores (puntuación) de cada generación en lo que se conoce como un histograma, el cual se ejemplifica en la figura A.13.

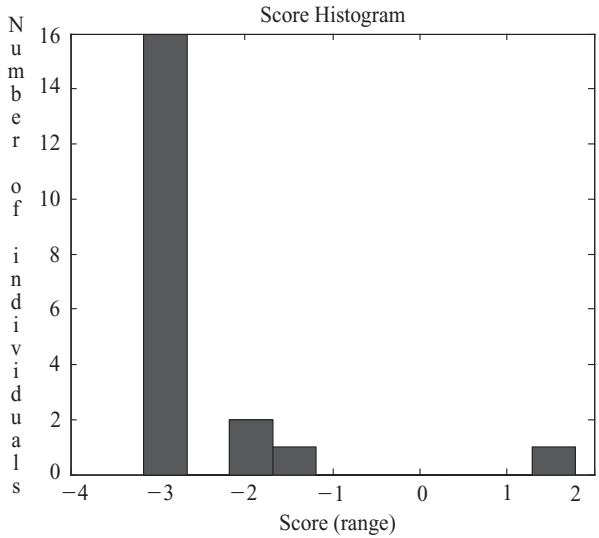


Fig. A.13 Histograma de la función $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$.

Scores/Puntuación o ponderación. Traza los valores (puntuación) de los individuos en cada generación (figura A.14).

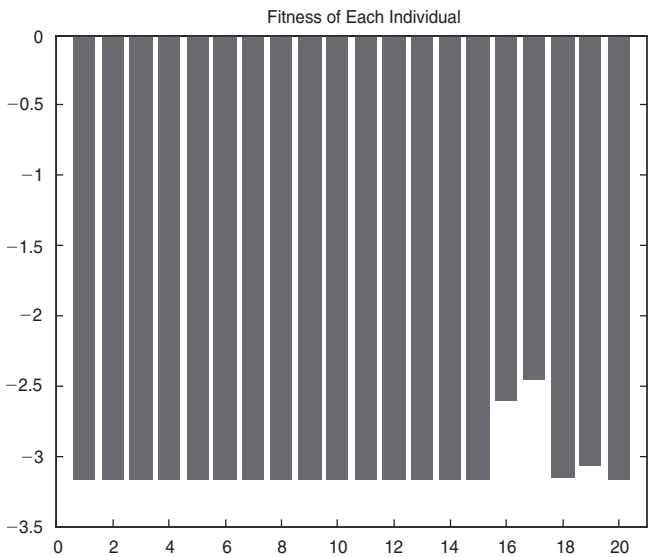


Fig. A.14 Valores de los individuos en $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$.

Selection/Selección. Grafica un histograma de los padres. Esto muestra qué contribuciones han hecho los padres en cada generación (figura A.15).

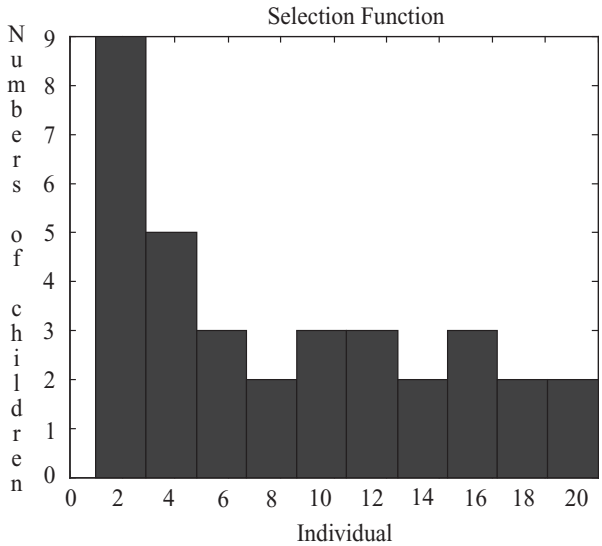


Fig. A.15 El histograma de padres de la función $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$.

Stopping/Detención. Grafica paro o sesiones de niveles según su criterio de acuerdo con el histograma de los padres, como en la figura A.16.

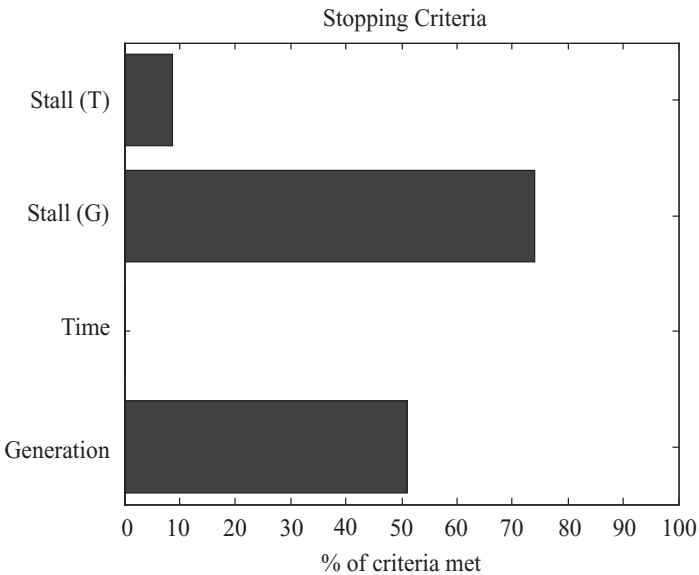


Fig. A.16 Comportamiento de generaciones dependiendo de la aptitud de los padres.

Max constrain/Máxima violación. Traza la máxima violación en una restricción no lineal (figura A.17).

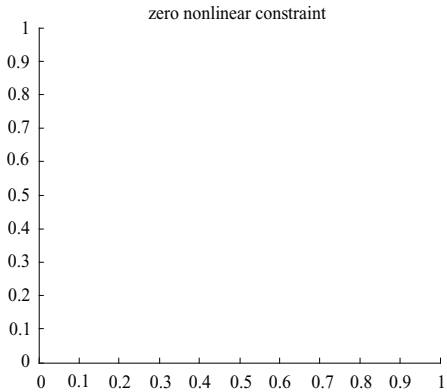


Fig. A.17 Este caso de $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$ no existe ninguna violación dentro del proceso, pero en funciones más complejas se pueden apreciar estas alteracio-

Custom function/Función personalizada. Esta herramienta tiene el objetivo de aplicar una función a la función aptitud, apreciando su comportamiento de la función aptitud mediante la gráfica.

SECCIÓN 3. RESULTADOS DE LA FUNCIÓN APTITUD

La siguiente sección que se analizará es Run solver y Final point.

En la figura A.18 se muestra un ejemplo para comprender las características que poseen las herramientas gráficas de AG en MATLAB®. Se resalta con un recuadro el valor de la función de ajuste y el número de generación actual.

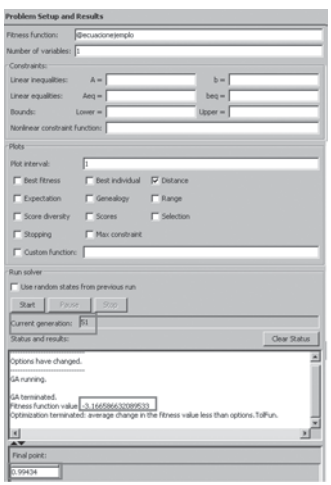


Fig. A.18 Muestra los resultados de la función aptitud.

Use random status from previous run/ Uso aleatorio de la corrida anterior. La opción sirve para tomar resultados de generaciones pasadas formando nuevos resultados con los AG.

Current generation/Generación actual. En ésta se muestra la cantidad de iteraciones o generaciones o épocas que se tuvo que evaluar para llegar al resultado.

Status and results/ Estado y resultado. Se muestra el resultado de la función aptitud, en conjunto con los puntos en los cuales se debe evaluar la función aptitud para hallar dicho resultado; éstos se muestran en la ventana de "Final point". Cabe mencionar que el número de variables que se declara en la sección 1 debe ser igual al número de incógnitas contenidas en la ecuación, ya que MATLAB® no tiene restricción en cuanto a número de variables. Se plantea tal situación debido a que el usuario debe recordar que el número de variables que se va a evaluar no debe exceder el número de variables en la ecuación; si es el caso, se indeterminaría el sistema, aun cuando MATLAB® entregue un resultado. Se tiene que pensar en forma análoga a un sistema de ecuaciones, donde debe existir igual número de incógnitas a ecuaciones; si no se respeta esta restricción el sistema nunca lograría converger o simplemente sería divergente.

Es necesario recordar que cada vez que se vuelva ejecutar la herramienta gatool, se tendrá un resultado distinto pero muy próximo a la solución buscada, a pesar de que la ecuación que se optimizará sea la misma.

Con respecto a la opción de *Export to Workspace*, como su nombre lo indica sirve para exportar todos los datos obtenidos en la herramienta de "*gatool*" a *Workspace*. Hay distintas opciones a escoger: toda la información creada o sólo ciertas partes del proceso. El botón para realizar esta exportación se muestra en la figura A.19.

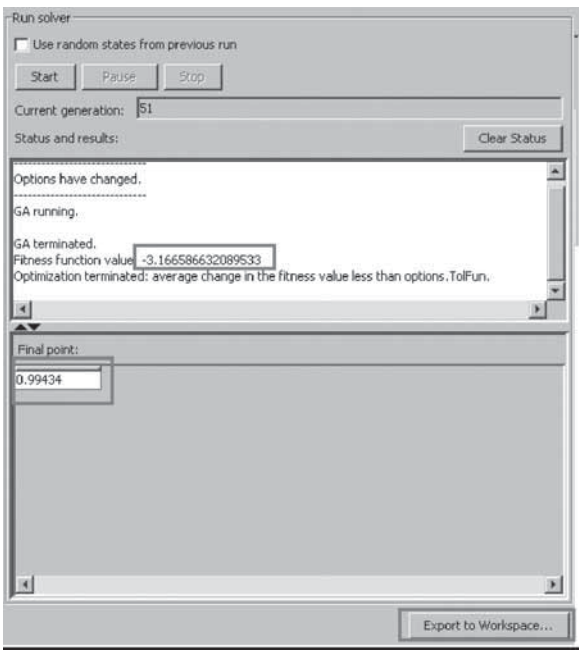


Fig. A.19 Resultados obtenidos en el planteamiento del ejemplo $y = \frac{1}{3}x^3 + \frac{3}{2}x^2 - 4x - 1$.

Donde se muestran las funciones de Fitness function value, Final point y Export to Workspace.

SECCIÓN 4. ALTERNATIVAS DE OPTIMIZACIÓN PARA LA FUNCIÓN APTITUD

Esta sección es de alta prioridad, ya que las opciones que ofrece la herramienta de algoritmos genéticos son muy potentes, donde se puede cruzar, mutar, crear un híbrido y demás aplicaciones para optimizar la función aptitud.

Population/Población

Las opciones de población permiten especificar los parámetros de la población que utiliza el AG, las cuales se muestran en la figura A.20.

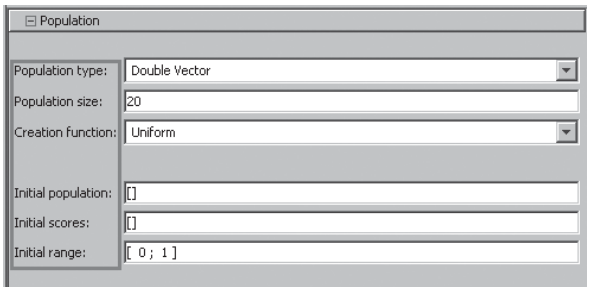


Figura A.20 Opciones que existen en el apartado de Population.

Population Type/Tipo de la población. Especifica el tipo de datos en la entrada de la función aptitud. El usuario puede fijar el tipo de la población teniendo las siguientes opciones:

- *Double Vector (Vector doble)*. Utilice esta opción si los individuos en la población son de tipo doble (número con decimal). Esta opción es la que viene por default.
- *Bit string (Cadena binaria)*. Use esta opción si los individuos en la población son de tipo cadena binaria.
- *Custom (Personalizado o configurable)*. Emplee esta opción para crear una población cuyo tipo de datos no sea ninguna opción de las anteriores. Si el usuario utiliza un tipo personalizado de la población, debe escribir su propia creación, mutación o funciones de cruce que acepten entradas en ese tipo de población, especificando estas funciones en los campos siguientes, respectivamente:
  - CreationFcn (Función de creación)
  - MutationFcn (Función de mutación)
  - CrossoverFcn (Función de cruce)
- *Population Size (Tamaño de la población)*. Especifica cuántos individuos hay en cada generación. Con un tamaño grande de la población, el AG busca en el espacio de solución más a fondo, de tal modo que reduce la oportunidad de que el algoritmo regrese a un mínimo local (no

[1] Recuérdese que en programación *double* se refiere a un número largo de tipo flotante que puede contener una cantidad grande y precisa. Por ejemplo, 2.2250738585072020_10_308. La precisión de un número se refiere a cuántos dígitos puede almacenar (antes y después del punto), sin que exista algún error.

siendo lo mismo que un mínimo global). Sin embargo, un tamaño grande de población también hace que el algoritmo funcione en forma lenta.

Si se fija el tamaño de la población (Population size) con un vector, el AG creará subpoblaciones múltiples, igual al número de longitud del vector. El tamaño de cada subpoblación es la entrada correspondiente del vector.

Creation function/Función de la creación (CreationFcn). Especifica la función que crea la población inicial para los AG.

El usuario puede elegir de las siguientes funciones:

- *Uniform (Población uniforme) (@gacreationuniform)*: crea una población inicial al azar con una distribución uniforme. Esta opción es la que se halla por default si existen restricciones o límites.
- *Feasible (Población factible) (@gacreationlinearfeasible)*: crea una población inicial al azar que satisfaga todos los límites y restricciones lineales. Se predispone para crear a los individuos que estén dentro de los límites (restricciones) y crea poblaciones aceptables dispersas. Esta opción estará por default si hay restricciones lineales.
- *Población personalizada*: le permite escribir su propia función de creación, que debe generar los datos del tipo que uno especifica en *tipo de población*.

InitialPopulation/Población inicial. Especifica una población inicial para el AG. El valor por default es [], en este caso se utiliza la *función de creación* por default para crear la población inicial. Si usted incorpora un arreglo no vacío en el campo inicial de la población, el arreglo no debe tener mayor cantidad de filas que el tamaño de la población, y exactamente el número de columnas de las variables. En este caso, el AG llama una función de creación para generar a los individuos restantes, si procede.

InicialScores/Puntuación inicial. Especifica las puntuaciones iniciales para la población inicial. La puntuación inicial puede ser también parcial.

PopInitRange/Rango inicial. Especifica el rango de los vectores en la población inicial que es generada por una función de creación. El usuario puede fijar el rango inicial para formar una matriz con dos filas y columnas cuyo valor es el número de las variables. Cada columna tiene la forma [lb; ub], donde "lb" es el límite más bajo y "ub" es el límite superior para las entradas en esa coordenada. Si se especifica como rango inicial un vector 2X1, cada entrada se expande como una fila de constantes con longitud igual al número de las variables.

FITNESS SCALING/Escala de la función de ajuste o evaluación

Convierte las puntuaciones (de baja calidad) que devuelve la función aptitud de algoritmos genéticos en valores con un rango adecuado para ser procesados por la función selección. El usuario puede especificar las opciones para el escalamiento de la aptitud en el panel de escala Fitness que muestra la figura A.21.

FitnessScalingFcn/Función del escalamiento. Especifica la forma en que se realiza la conversión; ésta tiene las siguientes opciones:

- El *rango(@fitscalingrank)*. La opción por default, Rango, convierte las puntuaciones (de baja calidad) con base en el rango de cada individuo dentro de la población. El rango de cada individuo es su posición en la lista de puntuaciones ordenadas.

Ejemplo

El rango del individuo más apto es 1, del siguiente más óptimo es 2, y así sucesivamente. Esta opción elimina el efecto de la propagación o desviación estándar de las puntuaciones (de calidad más baja) de la población.

Fig. A.21 Muestra una de las opciones a las que se puede acceder en el panel Fitness scaling.

- Proporcional (@fitscalingprop). La escala proporcional hace que el valor de la puntuación de individuo adquiera un valor proporcional.
- El Tope (@fitscalingtop). La escala *Top* clasifica a los individuos más óptimos con el mismo valor. Al seleccionar *Top* se muestra un campo adicional, *Quantity*, que determina el número de individuos a los que se les asigna valores positivos. *Quantity* puede ser un número entero entre 1 y el tamaño de la población, o una fracción entre 0 y 1 que especifica una parte del tamaño de la población. El valor predeterminado es 0.4. A su vez, es importante aclarar que cada uno de los individuos que producen descendencia se les asigna un mismo valor, mientras que al resto se les da un valor 0. Los valores dentro de esta escala tienen la forma [0, 1 / n, 1 / n, 0, 0, 1 / n, 0, 0, 1 / n].

Para reemplazar el valor por default de Quantity en la línea de comandos, utilice la siguiente sintaxis:

options = gaoptimset('FitnessScalingFcn',{@fitscalingtop,quantity})

Shift linear/Cambio lineal (@fitscalingshiftlinear). Esta opción convierte los valores de las puntuaciones de tal manera que el valor asignado para el individuo más apto sea igual a una constante multiplicada por la puntuación promedio de la población. La constante se especifica en el campo Max Survival rate (Tasa de supervivencia de Max), que se muestra cuando se selecciona Shift linear. Su valor predeterminado es 2.

Para reemplazar el valor por default de *Quantity* en la línea de comandos, utilice la siguiente sintaxis:

options = gaoptimset('FitnessScalingFcn', {@fitscalingshiftlinear, rate})

Custom/Personalizado. Permite escribir una función propia de conversión o escala. Si la función no tiene argumentos de entrada, se escribe en el cuadro de texto en la forma @myfun. Si su función tiene argumentos de entrada, se escribe como un arreglo de la forma {@ myfun, P1, P2, ...}, donde P1, P2, ... son los argumentos de entrada.

Así, la función de escala o conversión debe tener la siguiente sintaxis:

function expection = myfun(scores, nParents, P1, P2, ...)

Los argumentos de entrada de la función son:

- Scores (puntuaciones). Un vector de escalares, una para cada miembro de la población.
- nParents. El número de padres que necesita la población usada.
- P1, P2, ... Argumentos de entrada adicionales, si los hay, que se desea pasar a la función.

La función regresa un acercamiento o estimación en la forma de un vector de escalares con la misma longitud que Scores; cada valor es el asignado a cada individuo de acuerdo con la función de conversión creada. La suma de los valores del vector de estimación debe ser igual a nParents.

Selection/Selección

Las opciones de selección determinan cómo el algoritmo genético escoge a los padres para la próxima generación. El usuario puede especificar la función que el algoritmo utiliza en el campo *Selection Function* (SelectionFcn) desplegable que se ve en la figura A22. Las opciones disponibles son las siguientes:

Fig. A 22 Se muestra el panel de Selection y la opción por default.

Stochastic Uniform/ Estocástico uniforme (@ selectionstochunif). La función de selección por default, *Stochastic Uniform*, traza una línea cuyas secciones corresponden a cada uno de los padres; la longitud de cada sección es proporcional al valor de escala (Fitness scaling) de cada padre. El algoritmo se mueve a lo largo de la línea en pasos de igual tamaño. En cada paso, el algoritmo asigna el padre que le corresponde a la sección. El primer paso es un número aleatorio menor al tamaño del paso de los algoritmos.

Remainder/Resto (@ selectionremainder). *Remainder* asigna a los padres de manera determinista de acuerdo con la parte entera del valor de escala de cada individuo y luego utiliza la selección de ruleta para la parte fraccional. Por ejemplo, si el valor de escala de un individuo es de 2.3, es listado dos veces como padre porque la parte entera es 2. Después de que los padres han sido asignados conforme a las partes enteras de los valores de escala, el resto de los padres son elegidos estocásticamente. La probabilidad de que un padre sea elegido en este paso es proporcional a la parte fraccional de su valor de escala.

Uniform/Uniforme (@ selectionuniform). La selección uniforme asigna a los padres con base en su puntuación y el número total de padres. Este tipo de selección es útil para depurar y hacer pruebas, pero no es una estrategia de búsqueda muy eficaz.

Roulette/Ruleta (@ selectionroulette). Esta selección elige a los padres mediante la simulación de una ruleta, en la que el área de sección de la rueda correspondiente a un individuo es proporcional a la puntuación del individuo. El algoritmo utiliza un número aleatorio para seleccionar una de las secciones, con una probabilidad igual a su área.

Tournament/Torneo (@ selectiontournament). Esta opción elige a cada uno de los padres primero escogiendo el *Tournament Size* (tamaño del torneo o competencia), después escoge cierta cantidad de jugadores al azar igual al valor del *Tournament Size* y se selecciona al más apto de este conjunto. El tamaño del torneo de tamaño debe ser de al menos 2. El valor predeterminado del tamaño del torneo es 4.

Para sobrescribir el valor predeterminado de Tournament Size en la línea de comandos se escribe:

options = gaoptimset('SelectionFcn', {@selecttournament, size})

Custom/Personalizado. Permite escribir una función propia de selección. Si la función no tiene argumentos de entrada, se escribe en el cuadro de texto en la forma @myfun. Si su función tiene argumentos de entrada, se escribe como un arreglo de la forma {@ myfun, P1, P2, ...}, donde P1, P2, ... son los argumentos de entrada.

La función de selección debe tener la siguiente sintaxis:

function parents = myfun(expectation, nParents, options, P1, P2, ...)

Los argumentos de entrada de la función son:

- Expectation (expectación). El número esperado de descendientes de cada miembro de la población.
- nParents. El número de padres a seleccionar.
- options (opciones). La estructura opciones del AG.
- P1, P2, ... Argumentos de entrada adicionales, si los hay, que se desea pasar a la función.
- La función regresa parents en la forma de un vector de longitud nParents con los índices de los padres seleccionados.

Reproduction/Reproducción

Las opciones de reproducción determinan cómo el AG crea descendientes para la próxima generación; ésta tiene dos opciones: Elite Count y Crossover fraction que se muestran en la figura A23.

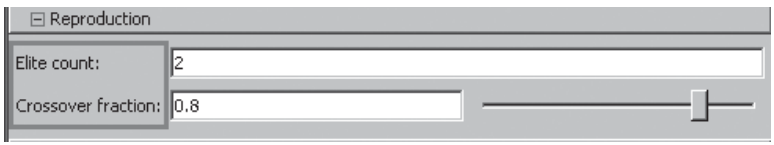


Fig. A23 Opciones del panel Reproduction.

Elite Count/Conteo elite (EliteCount). Especifica el número de individuos garantizados para sobrevivir en la próxima generación. El valor de *Elite Count* debe ser un número entero positivo menor o igual al tamaño de la población. El valor predeterminado es 2.

Crossover fraction/Fracción de cruzamiento (CrossoverFraction). Determina la fracción de la próxima generación, siempre que no sean descendientes élite, que es producida por entrecruzamiento. El valor de Crossover Fraction debe ser una fracción entre 0 y 1, el cual ser introducido en el campo o moviendo la barra deslizable. El valor predeterminado es 0.8.

Mutation/Mutación

Las opciones de mutación especifican cómo el algoritmo genético hace pequeños cambios aleatorios en los individuos de la población para crear descendencia mutada. La mutación proporciona diversidad genética y permite que el algoritmo genético busque una solución en un espacio más amplio. El usuario puede determinar la función de la mutación en el campo Mutation function (MutationFcn) que se aprecia en la figura A24.

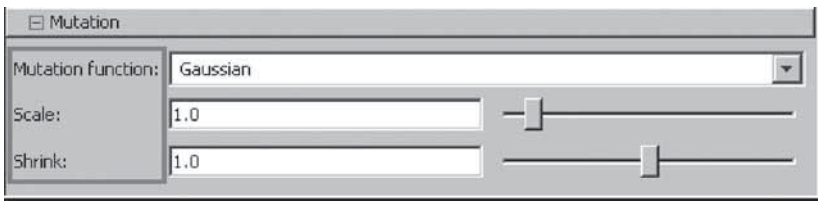


Fig. A24 Opciones del panel Mutation.

Mutation function/Función mutación. La función de la mutación por default *Gaussian* añade un número al azar tomado de una distribución de Gaussiana con una media de 0, a cada entrada del vector de padres (*parent vector*). La variación de esta distribución es determinada por los parámetros *Scale* (escala) y *Shrink* (contracción), los cuales se muestran cuando se selecciona *Gaussian*, y también cuando se elige la opción *Initial* range en las opciones *Population*.

- El parámetro *Scale* (escala) determina la variación en la primera generación. Si se establece *Initial range* como un vector V 2×1, la variación inicial es la misma en todas las coordenadas del vector padre (vector parent), cuyo valor es Scale * [V(2)-V(1)].

Si establece Initial range por un vector V, con dos filas y Number of variables en las columnas, la variación inicial en la coordenada i del vector padre es dada por la Scale*[V(i,2)-V(i,1)].

- El parámetro **Shrink** (contracción) controla como la variación conforme se pasa de generación a generación. Si establece **Initial range** como un vector V 2×1, la variación en la generación de k-ésima, vark, es la misma en todas las coordenadas del vector padre, y es determinada por la fórmula recursiva:

$$\text{var}_k = \text{var}_{k-1}\left(1 - Shrink\,\frac{k}{Generations}\right)$$

o

$$\text{var}_k = \text{var}_{k-1}\left(1 - Contracción\,\frac{k}{Generaciones}\right)$$

Si establece Initial range por un vector V, con dos filas y Number of variables en las columnas, la variación inicial en la coordenada i del vector padre en la generación de k-ésima, vark, es determinada por la fórmula recursiva:

$$\text{var}_{i,k} = \text{var}_{i,k-1}\left(1 - Shrink \cdot \frac{k}{Generations}\right)$$

o

$$\text{var}_{i,k} = \text{var}_{i.k-1}\left(1 - Contracción \cdot \frac{k}{Generaciones}\right)$$

Los valores predeterminados de Scale y Shrink son de 0.5 y 0.75, respectivamente. Si se establece Shrink a 1, el algoritmo reduce la varianza en cada uno de coordenada en forma lineal hasta llegar a 0 en la última generación. Un valor negativo de Shrink causa que la variación aumente.

Para reemplazar el valor por default de Scale y Shrink en la línea de comandos, se escribe:

options = gaoptimset('MutationFcn', {@mutationgaussian, scale,shrink}

Uniform/Uniforme (mutación uniforme). La mutación uniforme es un proceso de dos pasos. En primer lugar, el algoritmo selecciona una fracción de las entradas del vector de un individuo para la mutación, donde cada entrada tiene un índice de probabilidad de ser mutado. El valor por default de la probabilidad es 0.01. En el segundo paso, el algoritmo reemplaza cada entrada seleccionada por un número aleatorio seleccionado de manera uniforme del rango que rige a esa entrada.

Para reemplazar el valor por default de Rate en la línea de comandos, se escribe:

options = gaoptimset('MutationFcn', {@mutationuniform, rate})

Custom/Personalizado. Permite escribir una función propia de mutación. Si la función no tiene argumentos de entrada, se escribe en el cuadro de texto en la forma @myfun. Si su función tiene argumentos de entrada, se escribe como un arreglo de la forma {@ myfun, P1, P2, ...}, donde P1, P2, ... son los argumentos de entrada.

Así, la función de mutación debe tener la siguiente sintaxis:

function mutationChildren = myfun(parents, options, nvars, FitnessFcn, state, thisScore, thisPopulation, P1, P2, ...)

Los argumentos de entrada de la función son:

- Parents. Vector de padres escogido por la función de selección.
- Options. La estructura opciones del AG.
- nvars. Número de variables.
- FitnessFcn. Fitness Function (función de ajuste).
- state – Structure State (Estructura de estado) [ver el final del documento].
- thisScore. Vector que contiene la puntuación de la población actual.
- thisPopulation. Matriz de los individuos de la población.
- P1, P2, ... Argumentos de entrada adicionales, si los hay, que se desea pasar a la función.

La función regresa mutationChildren (descendencia mutada) como un arreglo cuyas filas corresponden a los individuos mutados. Las columnas del arreglo son inicializadas de acuerdo con el *Number of Variables*.

Crossover/Cruzamiento

Las opciones Crossover especifican cómo el AG combina dos individuos, o padres, para formar un descendiente para la próxima generación. **Crossover Function** o función de cruzamiento (CrossoverFcn) determina la función que realiza el cruce. Se puede elegir entre las funciones de la lista desplegable que se observa en la figura A25.

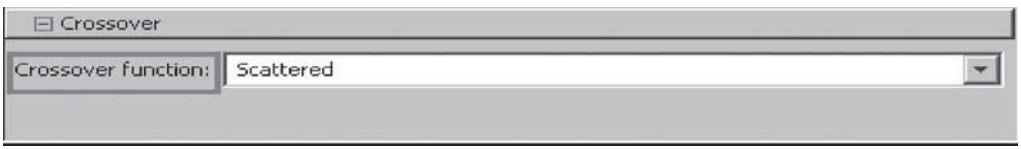


Fig. A25 Panel de Crossover.

Scattered/Dispersos (@ crossoverscattered). La función de cruce por default crea un vector binario aleatorio y selecciona los genes del primer padre donde el vector es 1 y los genes del segundo padre donde el vector es 0, y combina estos genes para formar un descendiente.

Ejemplo

Si P1 y P2 son los padres

P1 = [a b c d e f g h]
P2 = [1 2 3 4 5 6 7 8]

y el vector binario es *[1 1 0 0 1 0 0 0]*, la función devuelve el hijo siguiente:

child1 = [a b 3 4 e 6 7 8]

Single Point/Un solo punto (@crossoversinglepoint). Elige un número entero n al azar entre 1 y el valor de *Number of variables* y luego:

- Selecciona las entradas del vector, menores o iguales que n del primer padre.
- Selecciona las entradas del vector, mayores o iguales que n del segundo padre.
- Concatena o une estas entradas para formar un vector hijo.

Ejemplo

Si P1 y P2 son los padres

P1 = [a b c d e f g h]
P2 = [1 2 3 4 5 6 7 8]

y el punto de cruce es de 3, la función devuelve el descendiente siguiente.

child = [A B C 4 5 6 7 8]

Two point/Dos puntos (@ crossovertwopoint). Selecciona dos enteros al azar m y n entre 1 y el valor de *Number of Variables*. Luego la función elige:

- Entradas del vector en posiciones menores o iguales a m del primer padre.
- Entradas del vector en posiciones de m+1 a n, inclusive, del segundo padre.
- Entradas del vector en posiciones mayores que n del primer padre.

El algoritmo entonces concatena estos genes para formar un único gen.

Ejemplo

Si P1 y P2 son los padres

P1 = [a b c d e f g h]
P2 = [1 2 3 4 5 6 7 8]

y los puntos de cruce son 3 y 6, la función devuelve el hijo siguiente.

child = [a b c 4 5 6 g h]

Intermediate/Intermedio (@crossoverintermediate). Crea a los descendientes al obtener un promedio de los padres. El usuario puede especificar los pesos mediante un solo parámetro, **R**, que puede ser un escalar o un vector de una fila con longitud *Number of variables*. El valor predeterminado es un vector con todos sus elementos igual a 1. La función crea el hijo del parent1 (padre1) y el parent2 (padre2) utilizando la siguiente fórmula.

child = parent1 + rand * Ratio * (parent2 : parent1)

Si todas las entradas de Ratio (razón) se encuentran en el intervalo [0,1], los hijos resultantes se encuentran dentro del hipercubo definido al colocar a los padres en vértices opuestos. Si Ratio no está en ese rango, los hijos podrían quedar fuera del hipercubo. Si Ratio es un escalar, entonces todos los hijos se encuentran en la línea entre los padres.

Para reemplazar el valor predeterminado de Ratio en la línea de comandos, se utiliza la sintaxis siguiente:

options = gaoptimset ('CrossoverFcn', (@ crossoverintermediate, relación));

Heuristic/Heurística (@crossoverheuristic). Devuelve un hijo que se encuentra en la línea que contiene a los dos padres, a poca distancia del padre con el mejor valor de ajuste en dirección opuesta al padre con el peor valor de ajuste. El usuario puede determinar la distancia entre el hijo y el padre más óptimo modificando el parámetro *Ratio*, que aparece cuando se selecciona Heuristic. El valor predeterminado del *Ratio* es de 1.2. Si parent1 y parent2 son los padres, y parent1 tiene el mejor valor de ajuste, la función devuelve al hijo:

child = parent2 + R * (parent1 : parent2);

Para reemplazar el valor predeterminado Ratio en la línea de comandos, se escribe:

options = gaoptimset ('CrossoverFcn', (@ crossoverheuristic,relación));

Custom/Personalizado. Permite escribir una función propia de cruzamiento. Si la función no tiene argumentos de entrada, se escribe en el cuadro de texto en la forma @myfun. Si su función tiene argumentos de entrada, se escribe como un arreglo de la forma {@ myfun, P1, P2, ...}, donde P1, P2, ... son los argumentos de entrada.

Así, la función de cruzamiento debe tener la siguiente sintaxis:

xoverKids = myfun(parents, options, nvars, FitnessFcn, unused,thisPopulation)

Los argumentos de entrada de la función son:

- Parents : Vector de padres escogido por la función de selección.
- Options : La estructura opciones del AG.
- Nvars : Número de variables.
- FitnessFcn : Fitness Function (función de ajuste).
- Unused : Posición que no es utilizada.
- thisPopulation : Matriz de los individuos de la población.
- P1, P2, ... : Argumentos de entrada adicionales, si los hay, que se desea pasar a la función.

La función regresa *xoverKids* (descendencia resultante del cruce), como un arreglo cuyas filas corresponden a los descendientes. Las columnas del arreglo son indizadas de acuerdo con el *Number of Variables.*

Migration/Migración

Las opciones de migración especifican cómo los individuos se mueven entre las subpoblaciones. La migración se produce si se establece el tamaño de la población (Population Size) como un vector de longitud superior a 1. Cuando se produce la migración, los mejores individuos de una subpoblación sustituyen a los peores de otra subpoblación. Los individuos que emigran de una subpoblación a otra se copian, no se elimina su subpoblación original.

Se puede controlar cómo la migración ocurre con los siguientes tres campos en el panel de opciones de migración, figura A26.

Fig. A26 Panel de Migration.

Direction/Dirección (MigrationDirection). La migración puede tener lugar en una o ambas direcciones.

- Si se establece la Dirección de *Forward* (hacia delante), la migración ocurre hacia la última subpoblación. Es decir, la n-ésima subpoblación migra hacia la (n+1) subpoblación.
- Si se establece la Dirección como *Both* (ambos), la n subpoblación emigra hacia la subpoblación (n−1) y (n+1).

La migración une los extremos de las subpoblaciones. Es decir, la última subpoblación emigra a la primera, y la primera a la última. Para evitar esta unión, se debe especificar una subpoblación de tamaño 0 añadiendo una entrada de 0 al final del vector del tamaño de la población que se ingresó en *Population Size*.

Interval/Intervalo. Controla cuántas generaciones se producen entre migraciones.

Ejemplo

Si establece un intervalo de 20, la migración tiene lugar cada 20 generaciones.

Fraction/Fracción. Controla el número de individuos que se mueven entre las subpoblaciones. Especifica la fracción de la subpoblación más pequeña de entre dos.

Ejemplo

Si los individuos emigran de una subpoblación de 50 a una subpoblación de 100 personas y se establece la fracción a 0.1, el número de personas que emigran es de 0.1* 50 = 5.

Algorithm Settings/Parámetros del algoritmo

En esta opción se definen parámetros específicos del algoritmo encerrados en un recuadro en la figura A27.

Fig. A27 Panel de Algorithm settings.

Initial penalty/Penalidad inicial. Especifica un valor inicial para que lo use el algoritmo. Debe ser igual o mayor a 1. El valor predeterminado es 10.

Penalty factor/Factor de penalización. Incrementa en un factor igual al *Penalty factor* el valor de *Initial penalty* cuando la solución no tiene la precisión requerida o cuando no cumple alguna condición inicial. Este debe ser mayor que 1 y su valor por default es de 100.

Hybrid Function (Función de hibridación)

Una función híbrida es otra función de reducción que se ejecuta después de que el algoritmo genético termina. Puede especificar una función híbrida en *Hybrid function*.

Las opciones son las siguientes y se accede a ellas en el panel de la figura A28

- *None* (Ninguno)
- *Fminsearch*. Utiliza la función de MATLAB fminsearch.
- *Patternsearch*. Utiliza un patrón de búsqueda.
- *Fminunc*. Utiliza la función *fminunc* de **Optimization Toolbox** (Herramientas de Optimización).

Fig. A28 Panel de Hybrid function.

Stopping Criteria/Criterio de detención

Criterios de detención que determinan qué causa que el algoritmo termine. Puede especificar las siguientes opciones (figura A29).

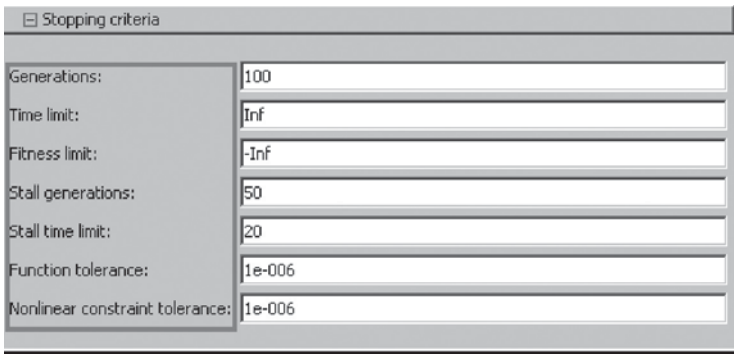


Fig. A29 Panel de Stopping criteria con sus diferentes opciones.

Generations/Generaciones. Especifica el número máximo de iteraciones que el AG lleva a cabo. El valor por default es 100.

Time Limit/Tiempo límite. Especifica el tiempo máximo en segundos en el que el AG se ejecuta antes de detenerse.

Fitness Limit/Límite de ajuste. El algoritmo se detiene si el mejor valor de ajuste es menor o igual a este valor límite.

Stall Generations/Generaciones recesivas (StallGenLimit). El algoritmo se detiene si no hay mejoría en el mejor valor de ajuste para el número de generaciones especificado por el valor de *Stall Generations*.

Stall Time/ Tiempo de retardo (StallTimeLimit). El algoritmo se detiene si no hay mejoría en el valor de ajuste en un intervalo de tiempo en segundos especificado por el valor de *Stall Time*.

Output Function/Función de salida

Las funciones de salida devuelven una salida del AG a la línea de comandos en cada generación. Las opciones disponibles se señalan en la figura A30.

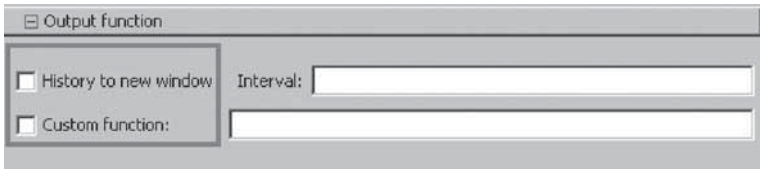


Fig. A30 Panel de Output function.

History to new window/Historia de una nueva ventana (@gaoutputgen). Muestra la historia de los puntos calculados por el algoritmo en una nueva ventana en cada intervalo (*Interval*).

Custom/Personalizado. Permite escribir una función de salida propia. Para especificar la función de salida de *History to new window* en la línea de comandos, se deben configurar las opciones de la siguiente manera:

options = gaoptimset ('OutputFcns', @ gaoutputgen)

Estructura de la función de salida

La función del AG pasa los siguientes argumentos de entrada a la función de salida en cada generación:

- **Options /Opciones.** La estructura de opciones del AG.
- **State /Estado.** Estructura que contiene información sobre la población actual.
- **Flag /Marcador o bandera.** Cadena que indica el estado actual del algoritmo de la siguiente manera:
  - 'init' : fase inicial
  - 'iter' : Algoritmo de ejecución
  - 'done' : Algoritmo terminado
- La función de salida devuelve los siguientes argumentos State, Optnew y Optchanged a AG:
- State
- Optnew : La estructura options (opciones) modificada por la función de salida. Este argumento es opcional.
- Optchanged : Marcador (flag) que indica los cambios en options en optnew.

Display to command window/Despliegue en la ventana de comandos

Esta opción indica el nivel de información que se muestra en el Command Window cuando el algoritmo se está ejecutando; el panel de esta opción se observa en la figura A31. Se puede elegir de entre las siguientes:

- **Off :** No se muestra ninguna información.
- **Iterative (Iterativo) :** Muestra información en cada iteración del algoritmo.
- **Diagnose (diagnóstico) :** Muestra información en cada iteración. Además indica información acerca de problemas en la ejecución y las opciones que se han cambiado en lugar de las predeterminadas.
- **Final :** Indica sólo la razón por la que se dejó de ejecutar el algoritmo.

Fig. A31 Ventana de Display to command window.

Vectorize/Vectorizar

La opción de vectorizar especifica si el cálculo de la función de ajuste es vectorizado o no; el campo para modificar esta opción se muestra en la figura A32. Cuando se establece que la función de ajuste es vectorizada en **Off**, el AG calcula los valores de la función de evaluación en un bucle o ciclo.

Cuando se establece como **On**, el algoritmo calcula los valores de la función de una nueva generación con una llamada a dicha función de ajuste, lo cual es más rápido que calcular los valores en un bucle. Sin embargo, para utilizar esta opción, la función de evaluación especificada debe ser capaz de aceptar matrices de entrada con un número arbitrario de filas.

Fig. A32 Venatana de Vectorize.

REFERENCIAS

Recurso electrónico recuperado el 18 de mayo de 2010 http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01018506

Baber, *Consultor Enciclopedia Combi Visual*, t. 4, RANDSA S.A., Barcelona, 1981.

Bankman Isaac, "HandBook of Medical Images", Academic Press, EU, 2000.

Biggs, Alton, *et al. Biology: The Dynamics of life*, McGraw-Hill, EU, 2000.

Brox, Piedad,"Fuzzy motion adaptive algorithm for video de-interlacing", Springer, Berlín, 2010.

Coppin, Ben, *Artificial Intelligence Illuminated*, Jones & Bartlett, EU, 2004.

Ejemplos de aplicaciones de GA Tool box en www.mathworks.com

Delgado, Alberto, *Inteligencia Artificial y mini robots*. ECOE, Colombia, 1998.

Diccionario Enciclopédico Planeta, ts. 4 y 5, 2a. ed., Planeta, 1985.

Manual de algoritmos genéticos Tool box. MATLAB®

ANEXO B BIBLIOGRAFÍA

El lector podrá descargar de http://virtual.alfaomega.com.mx/ el anexo B con más de 20 páginas de referencias bibliográficas actualizadas.