UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE INGENIERÍA MECÁNICA

IDENTIFICACIÓN DE SISTEMAS DINÁMICOS MEDIANTE EL MODELO NARMAX USANDO REDES NEURONALES RECURRENTES Y ALGORITMOS GENÉTICOS

TESIS DE GRADO

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO MECATRÓNICO

PRESENTADO POR:

DANNY PERALTA BRAVO

PROMOCIÓN
2001–II

LIMA – PERÚ
2003

IDENTIFICACIÓN DE SISTEMAS DINÁMICOS MEDIANTE EL MODELO NARMAX USANDO REDES NEURONALES RECURRENTES Y ALGORITMOS GENÉTICOS

ÍNDICE

prologo 1

CAPÍTULO I.

INTRODUCCIÓN 4

1.1. Fondo y Motivación 5
1.2. Planteamiento de la Tesis 6
1.3. Organización de la Exposición 7

CAPÍTULO II.

IDENTIFICACIÓN PARAMÉTRICA DE SISTEMAS 8

2.1. ¿Qué es la Identificación de Sistemas? 9
2.2. Técnicas Tradicionales para la Identificación de Sistemas 9
2.2.1. Métodos de la Respuesta Transitoria 10
2.2.2. Métodos de la Respuesta en Frecuencia 14
2.3. Aproximaciones Funcionales 16
2.4. Optimización Estructural 19
2.5. Optimización Paramétrica 21
2.6. Determinación de la Influencia de las Variables de Entrada 21
2.7. Validación de Modelo 28
2.8. Esquema General de la Aproximación Funcional 30

CAPÍTULO III.

MODELADO DE SISTEMAS DINÁMICOS NO LINEALES 34

3.1. Aspectos Generales 35
3.2. Estructura del modelo 36
3.3. Aproximación mediante modelos lineales 37
3.3.1. Modelo OE 39
3.3.2. Modelo ARX 40
3.3.3. Modelo ARMAX 42
3.4. Método de los Mínimos Cuadrados 43
3.4.1. Método de los Mínimos Cuadrados no recursivo (Algoritmo "One-Shot") 44
3.4.2. Método de los Mínimos Cuadrados Recursivo 47
3.4.3. Modificaciones del Algoritmo de RLS Recursivo 51
3.5. Identificación de sistemas utilizando RLS 53

3.6. Aproximación mediante Modelos No Lineales 60
3.6.1. Modelo Hammerstein 61
3.6.2. Modelo Wiener 62
3.6.3. Modelo Polinomial NARMAX 63

CAPÍTULO IV.

APROXIMACIÓN FUNCIONAL A UN MODELO NARMAX USANDO REDES NEURONALES RECURRENTES 70

4.1. Breve Historia 71
4.2. Fundamentos de las Redes Neuronales 74
4.2.1. El Modelo Biológico 74
4.2.2. Las Redes Neuronales Artificiales 77
4.3. El Perceptron Multicapa *MLP* 82
4.3.1. Función de costo 84
4.3.2. BPEA para redes neuronales recurrentes 84
4.4. Aproximación Funcional usando Redes Neuronales recurrentes . . . 92

CAPÍTULO V.

APROXIMACIÓN FUNCIONAL A UN MODELO NARMAX USANDO ALGORITMOS GENÉTICOS 100

5.1. Breve Historia 101
5.2. Algoritmo Genético Simple 102
5.2.1. Criterios para Implementar un AG 107
5.3. Operadores Genéticos 113
5.3.1. Selección 113
5.3.2. Cruzamiento 115
5.3.3. Mutación 118
5.4. Ejemplo de Aplicación 119
5.5. Fundamento Matemático de los Algoritmos Genéticos 122
5.5.1. Teorema de los Esquemas 125
5.5.2. Propiedad del Paralelismo Implícito de los AGs 128
5.6. Templado Simulado (*Simulated Annealing*) 129
5.6.1. Teoría de la Información 129
5.6.2. Conceptos de Mecánica Estadística 130
5.6.3. Funcionamiento del Templado Simulado 132
5.7. Aproximación Funcional Usando Algoritmos Genéticos 132
5.8. Aproximación Funcional Usando el Algoritmo Híbrido *SA-GA* . . . 140

CAPÍTULO VI.

IDENTIFICACION PARAMÉTRICA *"ON-LINE"* APLICADO A UN MOTOR DC 147

6.1. Principios y Funcionamiento del Motor DC 147
6.2. Diagrama General para la Identificación *on line* 150

6.3. Etapa de Sensado 151
6.4. Etapa de Generación de la Señal de Entrada 154
6.5. Etapa de Amplificación de Potencia 154
6.6. Etapa de Comunicación 156
6.6.1. Etapa de Comunicación Serial 157
6.6.2. Etapa de Comunicación Paralela 162
6.7. Selección del Periódo de Muestreo y Diagrama de Tiempos 162
6.8. Programación 164
6.9. Diagrama de Flujo de la Programación 168

CAPÍTULO VII.

ANÁLISIS DE RESULTADOS 169

7.1. Aproximación Funcional frente a una entrada Escalón 170
7.2. Aproximación Funcional frente a una entrada Senoidal 171
7.3. Aproximación Funcional frente a una entrada Cuadrada 171

CONCLUSIONES 174

RECOMENDACIONES PARA TRABAJOS FUTUROS 178

APÉNDICE A: PROGRAMAS EN MATLAB 180

APÉNDICE B: PROGRAMAS CON EL GUI DE MATLAB 182

APÉNDICE C: CIRCUITOS IMPRESOS 190

APÉNDICE D: CARACTERÍSTICAS FÍSICAS Y ELÉCTRICAS DEL MOTOR DC 192

APÉNDICE E: CARACTERÍSTICAS FÍSICAS Y ELÉCTRICAS DEL ENCODER 195

BIBLIOGRAFÍA 196

PRÓLOGO

La identificación de sistemas puede definirse como el área de la teoría de sistemas que estudia metodologías para la obtención de modelos matemáticos de sistemas dinámicos a partir de mediciones sobre el sistema. La identificación de sistemas se ha convertido en una herramienta fundamental en muchas ramas de la ingeniería y otras áreas tan diversas como la bio-tecnología y la economía, las cuales requieren la existencia de modelos precisos de sistemas que posibiliten el análisis, la simulación, el diseño e implementación de estrategias de control. En aplicaciones de control, la obtención de un modelo matemático más o menos preciso del sistema es fundamental ya que la mayoría de los métodos de diseño de controladores parten de la hipótesis de que un modelo parametrizado del proceso está disponible.

En la presente tesis se pretende dar un panorama general de los métodos más difundidos para la identificación de sistemas dinámicos, tanto en sus aspectos teóricos como en los de implementación mediante el uso de software interactivo, a través de la creación de interfaces gráficas GUI (Graphical User Interface) de Matlab. Veremos también como emplear las técnicas más innovadoras de la Inteligencia Artificial para la identificación paramétrica de sistemas. Los distintos métodos se ilustrarán con ejemplos de aplicación. Luego se realizará la identificación paramétrica de sistemas a un motor DC en línea, utilizando redes neuronales recurrentes.

La presente tesis consta de siete capítulos con los siguientes contenidos:

- Capítulo 1: Comienza con una breve introducción de los modelos dinámicos. Se plantea el problema a resolver.

- Capítulo 2: Se presenta los conceptos sobre la identificación de sistemas, los "métodos tradicionales" utilizados para la realizar la identificación. Además se explica la teoría sobre los aproximadores funcionales y su fundamento matemático.

- Capítulo 3: Se presenta los diferentes aproximadores lineales y no lineales de los sistemas dinámicos, dando un especial énfasis al modelo polinomial NARMAX. También se presenta el método de los mínimos cuadrados recursivos utilizado para la identificación paramétrica de sistemas.

- Capítulo 4: Se presenta a las redes neuronales, una reseña histórica y su fundamento matemático. Se muestra su eficacia en la identificación de parámetros de un aproximador funcional (polinomio NARMAX) tanto en una planta lineal como en una no lineal.

- Capítulo 5: Se presenta un tema relativamente nuevo, como son los algoritmos genéticos, se muestra una breve historia, fundamento matemático y un pequeño ejemplo para describir su funcionamiento. Luego se demuestra su eficacia en la identificación de los parámetros de un aproximador funcional (de un polinomio NARMAX) tanto en una planta lineal como en una no lineal. También hemos agregado el simulated annealing que es una búsqueda heurística que nos permite llegar a una convergencia mucho más rápida del algoritmo genético cuando trabajan juntos (híbrido SAGA).

- Capítulo 6: Se presenta las diferentes etapas que se siguieron para realizar la identificación de un motor DC "on line", realizando una breve explicación de cada una de las diferentes etapas y los diagramas de flujo de cada una de ellas, describiendo los diferentes componentes utilizados para la realización de cada una de las etapas. También se presenta la interfaz gráfica que es utilizada en el Capítulo 7 para la identificación paramétrica de un motor DC en línea.

- Capítulo 7: Se presentan los resultados obtenidos de la identificación en línea mediante redes neuronales recurrentes (por medio del algoritmo Back-Propagation Error Algorithm BPEA) de un motor DC.

Por último se muestran cuatro secciones adicionales, que contienen las conclusiones de la tesis, las recomendaciones para trabajos futuros, una serie de apéndices, los cuales contienen: *listado de los programas en Matlab, las interfaces gráficas utilizando el GUI y los circuitos impresos utilizados para realizar la identificación en línea de un motor DC* y la bibliografía.

CAPÍTULO I
INTRODUCCIÓN

El diseño de un controlador contínuo o discreto, ya sea mediante técnicas clásicas o en variables de estado, requiere de un modelo de la planta a controlar que caracterice su comportamiento dinámico. Este modelo permite al diseñador realizar y validar mediante las respectivas simulaciones, el ajuste de parámetros del controlador que permiten obtener una respuesta que cumpla con las especificaciones de diseño. En la presente tesis se estudian diferentes alternativas para obtener el modelo de un sistema que represente completamente su dinámica como paso previo al diseño de un controlador.

Cuando se hace necesario conocer el comportamiento de un sistema en determinadas condiciones y ante determinadas entradas, se puede recurrir a la experimentación sobre dicho sistema y a la observación de sus salidas. Sin embargo, en muchos casos la experimentación puede resultar compleja o incluso imposible de llevar a cabo, lo que hace necesario trabajar con algún tipo de representación que se aproxime a la realidad, a la que se conoce como *modelo*.

Básicamente, un modelo es una herramienta que permite predecir el comportamiento de un sistema sin necesidad de experimentar sobre él, por lo que generalmente la definición de modelo se da a partir de la representación, a menudo en términos matemáticos, de las características importantes del objeto o sistema bajo estudio. El desarrollo de computadoras de alta velocidad ha hecho posible incrementar las aplicaciones del modelado, mediante la representación de un sistema como un modelo matemático, convirtiendo al modelo en instrucciones para una computadora.

Actualmente es posible modelar sistemas más grandes y complejos que antes. Esto ha resultado en considerar el estudio de las técnicas de modelado

para computadoras, es decir, las computadoras están involucradas en el modelado en dos maneras: como herramienta computacional para modelado y como sujetos de modelado.

1.1. Fondo y Motivación

En este trabajo de tesis se planteó las siguientes interrogantes ¿Cómo realizar un buen control? ¿Qué técnica es la más adecuada para que cumpla con las condiciones de diseño?, partiendo de estas dos interrogantes podemos decir que para realizar un buen control sobre cualquier sistema (mecánico, hidraúlico, neumático, económico, biológico, social[1], etc) es necesario contar con una representación de ese sistema, generalmente a través de un modelo matemático. Este modelo matemático hallado debe representar la dinámica completa del proceso, a lo largo de todos los puntos de operación del proceso (la cual abarca las no linealidades del sistema - ver figura 1.1).

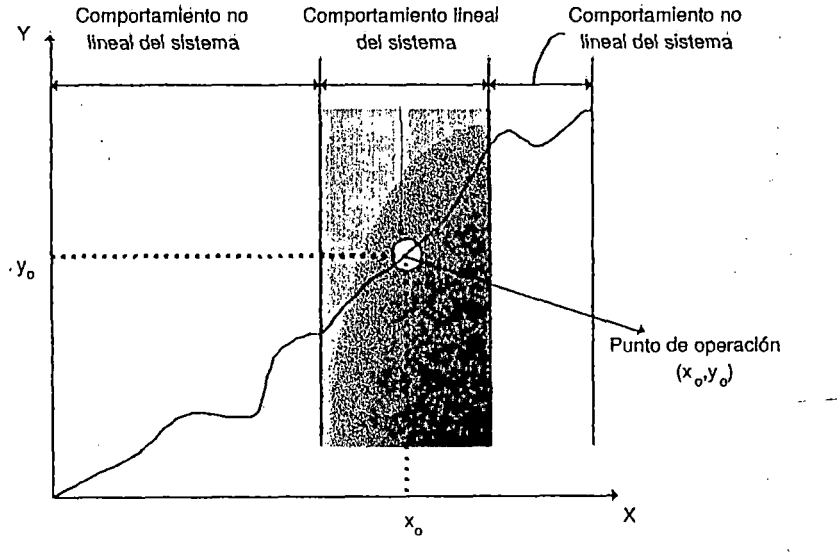

Figura 1.1: Representación del comportamiento del sistema al variar su punto de operación.

Diversos métodos y algoritmos para la identificación de sistemas han sido estudiados desde 1960. Muchos procedimientos han sido propuestos y muy usa-

[1]Modelamientos Sociales realizados en el Departamento de Matemática Aplicada Universidad de Valencia a cargo del Dr.Moliner rachid.temre@uv.es

dos en la identificación de sistemas lineales, sin embargo su aplicabilidad para la identificación de sistemas no lineales es muy limitada. Actualmente se ha incrementado el estudio de modelos no lineales para la representación de procesos. Se han desarrollado herramientas matemáticas que permiten modelar la dinámica completa del proceso incluyendo sus no linealidades.

1.2. Planteamiento de la Tesis

Como se explicó en la sección anterior, el problema fundamental para un buen control radica en la necesidad de tener un modelo matemático que represente la dinámica completa del sistema (incluyendo las no linealidades que puede presentar el sistema). Generalmente se trata de obtener un modelo lineal que se asemeje al proceso, pero este modelo lineal sólo es válido para un rango de trabajo del proceso, pero que sucede cuando los parámetros del sistema varían o cuando los sensores se saturan, entonces el sistema empieza a presentar no linealidades, por lo tanto, el modelo lineal que se había elegido queda obsoleto debido a que ya no es capaz de representar esas no linealidades del proceso.

Entonces surge la necesidad de obtener un método que nos permita modelar la dinámica completa del proceso, incluyendo la no linealidades que pueda presentar dicho proceso. En esta tesis se propone el modelamiento de la dinámica de sistemas mediante aproximadores funcionales, por medio de la utilización del modelo polinomial NARMAX y la estimación de los parámetros de este modelo polinomial a través de técnicas de la inteligencia artificial: redes neuronales recurrentes y algoritmos genéticos.

1.3. Organización de la Exposición

El objetivo de la exposición es, primero presentar a los aproximadores funcionales como una alternativa de solución para la identificación de sistemas. Luego se explicará los diferentes métodos desarrollados en la tesis, para la identificación de sistemas dinámicos utilizando el modelo polinomial NARMAX. Nos centraremos en la identificación paramétrica utilizando redes neuronales recurrentes y se expondrá los resultados obtenidos de la identificación en línea de un motor DC. El diagrama de flujo de la exposición es presentada en la figura 1.2.

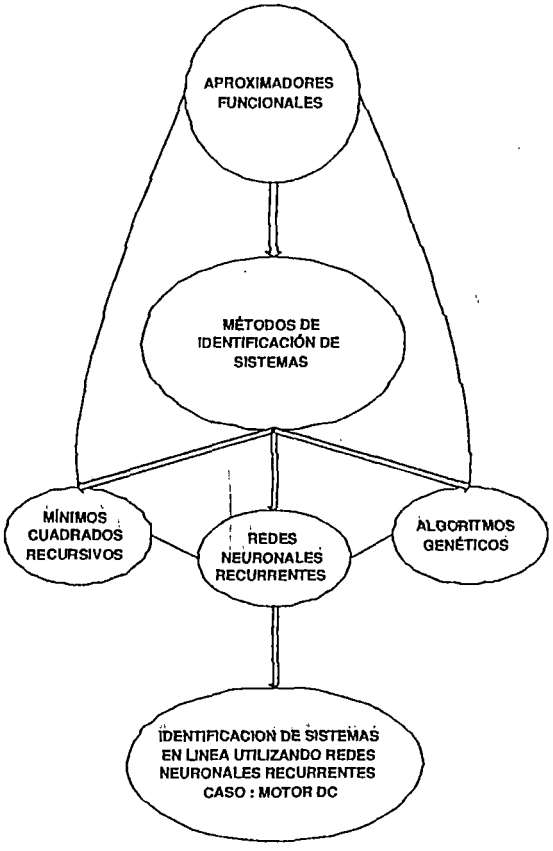

Figura 1.2: Organización de la exposición.

CAPÍTULO II
IDENTIFICACIÓN PARAMÉTRICA DE SISTEMAS

El modelado de procesos dinámicos a partir de un conjunto de muestras de la relación entrada/salida puede ser considerado como un problema de aproximación funcional, en el que se trata de hallar una expresión matemática que sea capaz de reproducir la relación inyectiva que existe entre las entradas y las salidas del proceso.

Un modelo matemático de un sistema dinámico se define como un conjunto de relaciones matemáticas que representan la dinámica del sistema. Hay que tener presente que el modelo matemático se obtenga puede que no sea el único para el sistema analizado. El modelo matemático hallado no sólo debe ser un mimetismo de la respuesta analizada del sistema, es decir la ecuación encontrada no debe sólo representar los datos hallados, sino que además ese modelo matemático encontrado debe ser capaz de representar la dinámica del sistema, es decir que frente a una señal que no ha sido utilizada para la evaluación del modelo, la representación matemática hallada debe reproducir con cierta precisión la respuesta real del sistema[1].

En las últimas tres décadas se ha consolidado la teoría de la identificación de sistemas. En los inicios los modelos matemáticos usados para representar los sistemas eran lineales y considerados suficientes a pesar de que no conseguían reproducir el comportamiento dinámico de los sistemas no lineales. La mayor parte de la teoría de control de sistemas fue desarrollada para modelos lineales. Con el avance tecnológico e industrial y el interés por el modelamiento no lineal, el desarrollo de herramientas matemáticas que nos permitieran entender mejor los fenómenos no lineales creció significativamente.

[1]Este procedimiento es conocido como *Validación del Modelo*, el cual se verá en los apartados siguientes.

En este capítulo se tratará la identificación de sistemas, algunos de los métodos tradicionales usados para la identificación, para luego centrarnos en las aproximaciones funcionales analizando sus fundamentos matemáticos. Finalmente veremos como es que se valida la aproximación funcional hallada.

2.1. ¿Qué es la Identificación de Sistemas?

Como se mencionó al iniciar este capítulo, la identificación de sistemas consiste en encontrar una relación matemática entre la entrada y salida de un sistema dinámico (ya sea un sistema mecánico, eléctrico, neumático, biológico, social, económico [10] etc). Pero tal relación matemática deberá no solo representar la relación de entrada y salida del sistema analizado, sino que además debe ser capaz de representar la salida del sistema frente a entradas que no han sido utilizadas para la evaluación del modelo, este proceso se conoce como *validación del modelo*. Debemos tener presente que existen una infinidad de curvas que pueden representar al sistema, pero sólo unos pocos modelo matemáticos pueden lograr representar la dinámica completa del sistema.

2.2. Técnicas Tradicionales para la Identificación de Sistemas

La mayoría de técnicas de identificación de sistemas están basados en modelos lineales. Los modelos hallados a través de estas técnicas representan en una forma aceptable la dinámica del sistema cuando este trabaja alrededor de un punto de trabajo (punto de operación), es decir la eficiencia de modelo matemático va ha depender de la zona en la cual está trabajando el sistema (para una mejor ilustación de esta idea ver la figura 2.1).

En la práctica mucho sistemas electroneumáticos, hidraúlicos, neumáticos, etc., involucran relaciones no lineales entre las variables. Por ejemplo, la salida de un componente puede saturarse para señales de entradas grandes. Puede haber una zona muerta[2] que afecta las señales pequeñas, los amotiguadores que

[2]La zona muerta de un componente es un rango pequeño de variaciones de entrada ante las cuales el componente es insensible

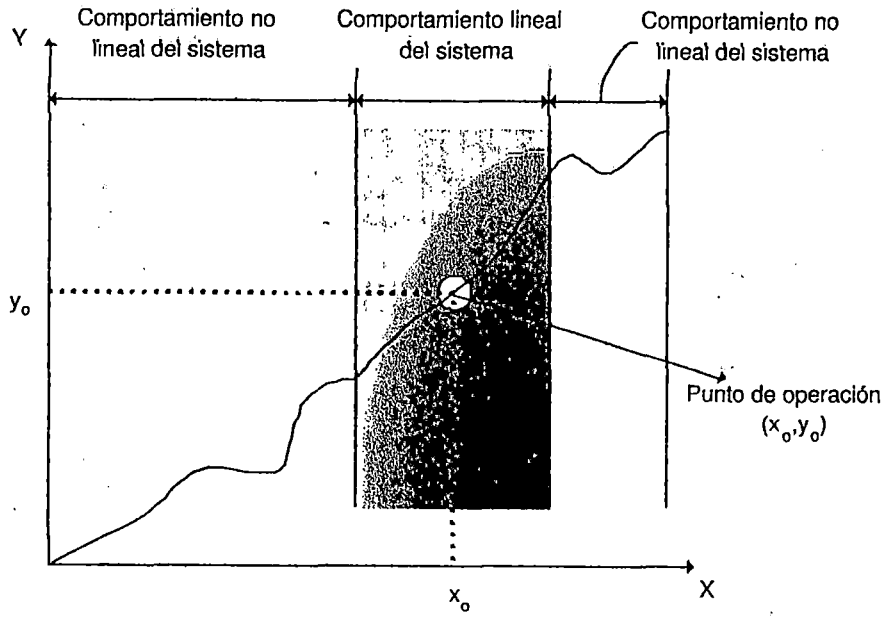

Figura 2.1: Zona de trabajo de un modelo lineal.

se utilizan en los sistemas físicos pueden ser lineales para operaciones a baja velocidad, pero pueden volverse no lineales a altas velocidades [9], y la fuerza de amortiguamiento se hace proporcional al cuadrado de la velocidad de operación. Algunos ejemplos de estas no linealidades son mostrados en la figura 2.2.

2.2.1. Métodos de la Respuesta Transitoria

Las técnicas de la respuesta transitoria están usualmente basadas en la respuesta en lazo abierto a una señal de prueba. Las señal de prueba más usada es la señal escalón unitario. Muchos procesos industriales tienen respuestas en lazo abierto como se muestra en las figuras 2.3, 2.5 ó 2.7.

Los sistemas autoregulados (SRS: Self-Regulating Systems) muestran una respuesta debido a una entrada escalón unitario (ver 2.3). Los procesos de control de temperatura son ejemplos de este tipo de sistemas. Un simple modelo matemático para tal sistema puede ser descrito por medio de la siguiente función de trasferencia:

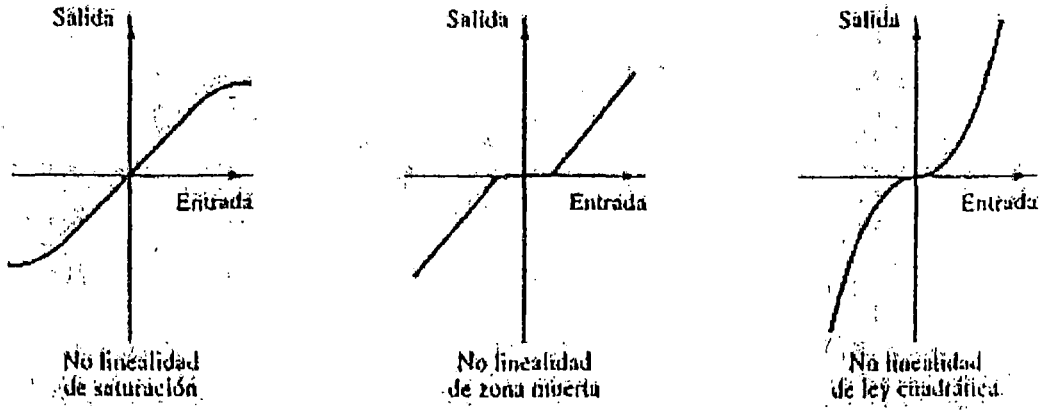

Figura 2.2: Curvas características para diferentes no linealidades.

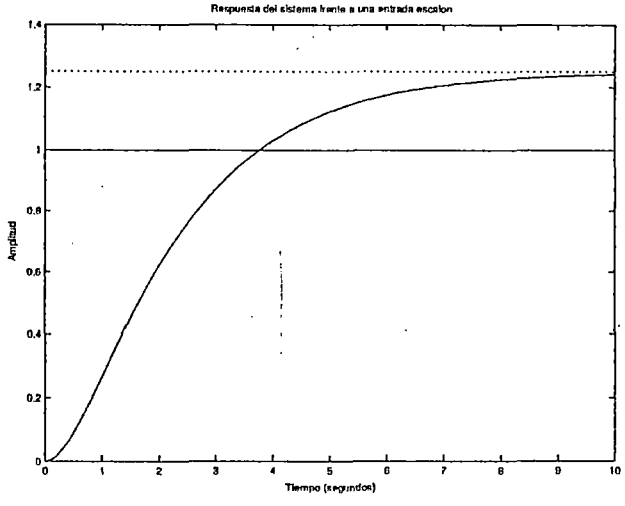

Figura 2.3: Respuesta de un proceso autoregulado ante una entrada escalón.

$$G(s) = \frac{K}{1+\tau s}\, e^{-sL} \tag{2.1}$$

Se tiene que K es la *ganancia del proceso*, τ es la *constante de tiempo* y L es el *tiempo de retardo*. Este modelo describe el comportamiento del proceso en el tiempo y donde L es la constante de tiempo de más alto orden en el proceso.

La determinación de los tres parámtros K, τ y L está basado en la construcción gráfica, tal como se muestra en la figura 2.4 donde se ha dibujado también la tangente a la respuesta del sistema (ocurre en el punto de inflexión de la curva).

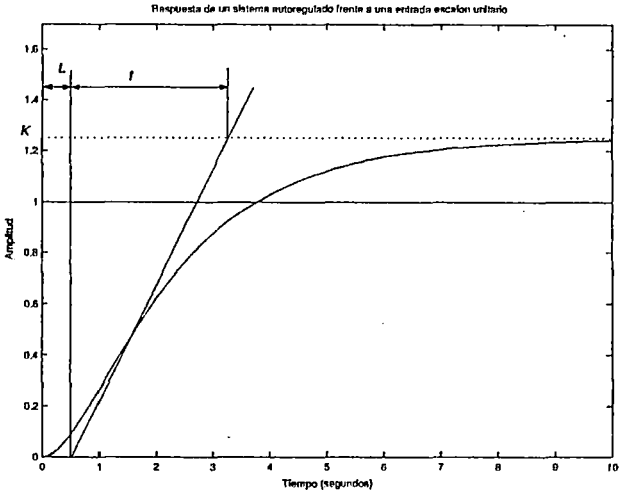

Figura 2.4: Determinación de los parámetros K, L y τ para un proceso autoregulado.

Los procesos no autoregulados (NSRP: Nonself-regulating processes) tienen una respuesta tal como se muestra en la figura 2.5 frente a un escalón unitario, donde se puede observar que la señal de entrada ocasiona que el sistema dé una respuesta que se incrementa o decrementa indefinidamente. Los procesos de control de nivel son de este tipo y ellos usualmente contienen un término integrador. La dinámica de este proceso puede ser aproximado mediante la siguiente función de tranferencia:

$$G(s) = \frac{K}{s} e^{-sL} \tag{2.2}$$

donde:

$$K = \frac{R}{L}$$

Se tiene que L es el *tiempo de retardo*, R es la *razón de cambio unitario*, y K es la *ganancia*. Esos dos parámetros pueden ser fácilmente obtenidos gráficamente

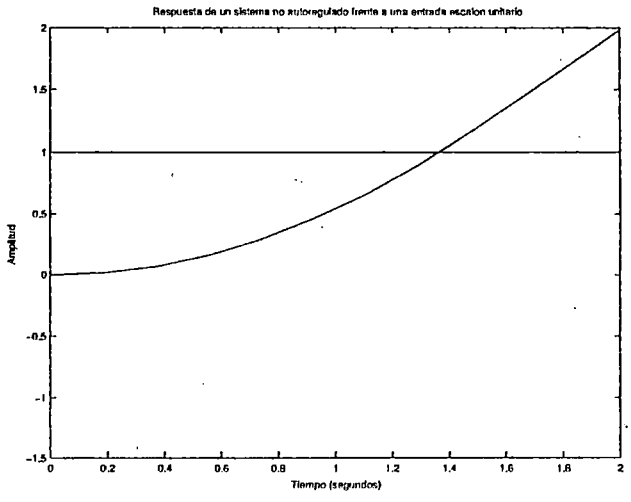

Figura 2.5: Respuesta de un proceso no autoregulado ante una entrada escalón.

de la respuesta de proceso a la entrada escalón unitario tal como se muestra en la figura 2.6

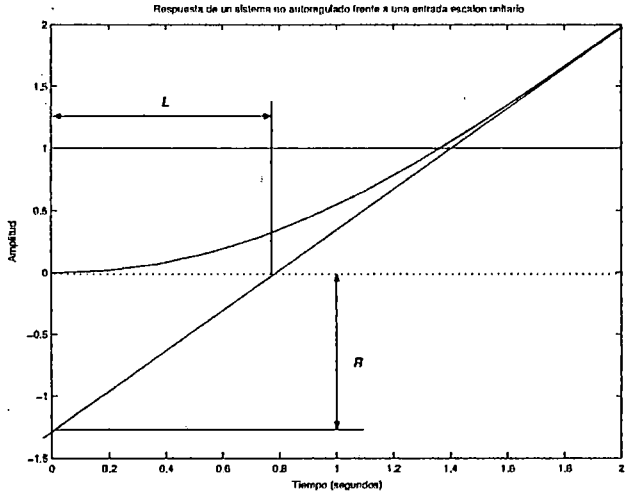

Figura 2.6: Determinación de los parámetros K y L para un proceso No autoregulado.

Un sistema oscilatorio que es sometido a una entrada escalón unitario se muestra en la figura 2.7. La dinámica puede ser aproximada a través de la siguiente función de tranferencia prototipo de segundo orden:

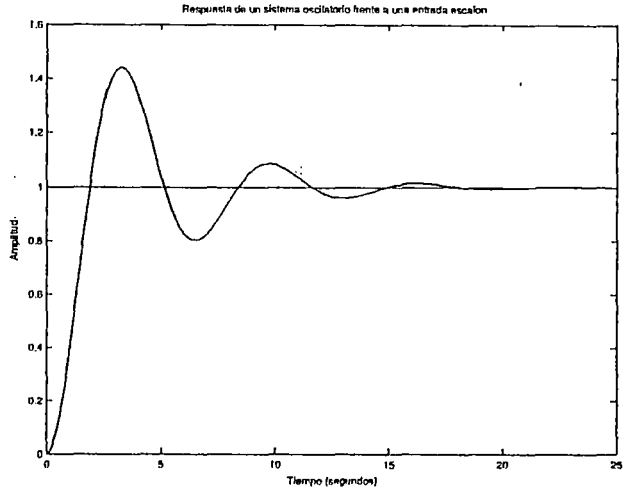

Figura 2.7: Respuesta de un proceso oscilatorio ante una entrada escalón.

$$G(s) = \frac{K \cdot \omega^2}{s^2 + 2\zeta\omega s + \omega^2} \tag{2.3}$$

Donde ω es la *frecuencia natural*, K es la *ganancia* y ζ es el *factor de amortiguamiento*. La frecuencia relativa, puede ser aproximado gráficamente (ver figura 2.8) y a través de las siguientes relaciones:

$$\begin{cases} \zeta = & \left[\sqrt{1 + (\frac{2\pi}{\ln d})^2}\right]^{-1} \\ \omega = & \frac{2\pi}{T_p\sqrt{1-\zeta^2}} \end{cases} \tag{2.4}$$

Una forma alterna para la ecuación (2.3) es incluir el tiempo de retardo L:

$$G(s) = \frac{K \cdot \omega^2}{s^2 + 2\zeta\omega s + \omega^2} e^{-sL} \tag{2.5}$$

donde L puede ser determinado de la misma forma que la ecuación (2.2).

2.2.2. Métodos de la Respuesta en Frecuencia

En este caso, el modelo resultante es una representación de la respuesta en frecuencia del sistema, obtenida mediante la aplicación de señales de entrada

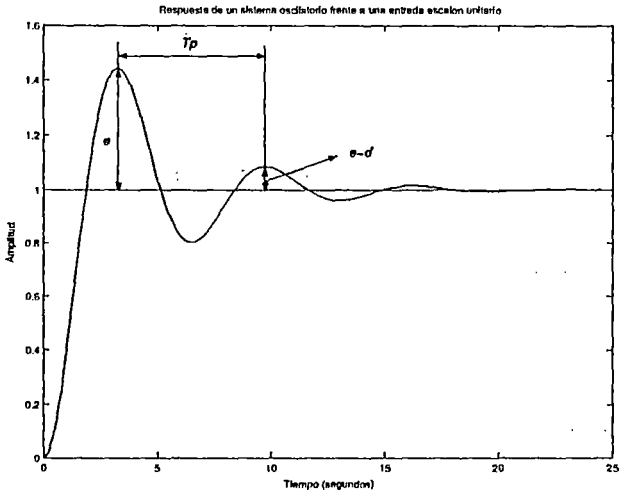

Figura 2.8: Determinación de los parámetros d y T_p para un proceso oscilatorio.

sinusoidales de distintas frecuencias. Cuando no sea posible aplicar este tipo de entradas, puede recurrirse a la aplicación de un ruido blanco[3], que permite obtener la respuesta en frecuencia mediante el conocido análisis espectral. Este análisis se basa en realizar las Transformadas de Fourier de las funciones de *covarianza de la entrada y la salida* y la *correlación entre la entrada y la salida*. Por tanto, definiendo las siguientes funciones de correlación:

$$R_u(\tau) = E[u(t-\tau)\cdot u(t)] = \frac{1}{N}\sum_{t=1}^{N} u(t-\tau)u(t) \tag{2.6}$$

$$R_{yu}(\tau) = E[y(t-\tau)\cdot u(t)] = \frac{1}{N}\sum_{t=1}^{N} y(t-\tau)u(t) \tag{2.7}$$

y sus transformadas de Fourier:

$$\Phi_u(\omega) = \sum_{t=-\infty}^{\infty} R_u(\tau)e^{-j\omega\tau} \tag{2.8}$$

$$\Phi_{yu}(\omega) = \sum_{t=-\infty}^{\infty} R_{yu}(\tau)e^{-j\omega\tau} \tag{2.9}$$

[3]Secuencia de números aleatorios de media nula y varianza constante.

Se demuestra que puede obtenerse la respuesta en frecuencia del sistema mediante la siguiente expresión:

$$G(e^{-j\omega\tau}) = \frac{\Phi_{yu}(\omega)}{\Phi_u(\omega)} \tag{2.10}$$

Este método no requiere un procesamiento complejo de los datos, ni ningún tipo de conocimiento previo sobre la planta, a excepción de que ésta sea lineal. Además, permite concentrar los datos obtenidos en torno al margen de frecuencias de interés. El principal inconveniente es que el modelo resultante no puede usarse directamente para simulación.

2.3. Aproximaciones Funcionales

El modelado de procesos dinámicos a partir de un conjunto de muestras de la relación entrada/salida puede ser considerado como un problema de aproximación funcional, en el que se trata de hallar un modelo matemático que sea capaz de reproducir la relación inyectiva (o en algunos casos biyectiva, como se verá en en Capítulo III) que existe entre las entradas y las salidas del proceso.

En esta sección se presentará un esquema general de aproximación funcional que pretende sistematizar el procedimiento de ajuste de aproximadores funcionales. Este procedimiento descompone el problema global del modelado de relaciones entrada/salida en dos optimizaciones parciales: la *optimización estructural* y la *optimización paramétrica*. La primera de ellas determinará la estructura del aproximador, mientras que la segunda ajustará sus parámetros.

Supongamos que se tiene un proceso estático[4] e invariante en el tiempo, que transforma un vector de entradas x en un vector de salidas y, y que la única información disponible del proceso es un conjunto de muestras de la forma:

$$S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

[4]En un proceso estático las salidas del proceso en el instante t sólo dependen de sus entradas en el mismo instante t y no de valores pasados.

El objetivo de la aproximación funcional es obtener una expresión matemática (modelo) que reproduzca lo más fielmente posible la transformacion $x \longrightarrow y$, esto es, que sea capaz de generar salidas correctas para vectores de entrada no contenidos en el conjunto de muestras S.

Podemos considerar el problema de la aproximación funcional como un problema de aprendizaje supervisado, en el que el ajuste y la adaptación del aproximador se interpreta como un proceso de aprendizaje, que es supervisado por el conjunto de muestras S.

De esta forma podemos definir dos niveles de adaptación del aproximador: *un nivel estructural* que determina la estructura interna del aproximador y por tanto su capacidad de representación, y *un nivel paramétrico* gobernado por el vector de parámetros w, que da forma al aproximador funcional, una vez determinada la estructura de dicha función.

Un aproximador funcional queda determinado por la siguiente ecuación:

$$F = \{y = f_h(x, w)/x \in R^n, y \in R^m, w \in W_h; h = 1, \ldots, H\} \qquad (2.11)$$

Donde y es la salida del sistema en respuesta al vector de entradas x, h es un índice estructural que permite identificar la estructura interna del aproximador y w es el vector de parámetros libres que el aproximador funcional ha tomado del espacio de parámetros W_h.

El problema del aprendizaje es seleccionar, de un conjunto dado de transformaciones de entrada/salida, aquella función $f_{h^*}(x, w^*)$ que aproxime de alguna forma óptima el vector de respuestas deseadas y. Para ello habrá que determinar la estructura óptima del aproximador funcional (dada por el índice estructural h^*), y el vector óptimo de parámetros w^* para esa estructura.

Para llevar a cabo este proceso de optimización es necesario establecer una función de error de la aproximación que permita evaluar la bondad del aproximador. La función de error ideal que se desea evaluar en todo problema de

aproximación funcional, es la función de costo. Esta función queda definida de la siguiente manera:

Sea $L(d, f_h(x,w))$ una medida de la discrepancia entre el vector de respuestas deseadas y, correspondientes al vector de entradas x, y la salida $f_h(x,w)$ del aproximador funcional. El valor esperado de la discrepancia queda definido por la función de costo:

$$R(h,w) = L(y, f_h(x,w)) \tag{2.12}$$

Como medidas de discrepancia más comunes tenemos el error cuadrático:

$$L(y, f_h(x,w)) = \|y - f_h(x,w)\|^2 \tag{2.13}$$

y el error absoluto:

$$L(y, f_h(x,w)) = |y - f_h(x,w)| \tag{2.14}$$

En el presente trabajo de tesis, utilizaremos como medida de la discrepancia al error cuadrático.

Bajo estas condiciones, el objetivo del aprendizaje es minimizar la función de costo $R(h,w)$ sobre el conjunto de funciones $f_h(x,w)$, con $w \in W_h$.

En la mayoría de los problemas prácticos de aproximación funcional, toda la información disponible se encuentra contenida en un conjunto de N pares de entrenamiento, al que llamaremos conjunto de entrenamiento:

$$S_N = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

Por lo tanto tendremos que estimar la función de costo a partir de las discrepancias cometidas sobre este conjunto. Para ello se puede utilizar el *Principio*

de Inducción de Minimización Empírica del Riesgo.

La idea básica de este método es utilizar el conjunto de entrenamiento $[(x_1, y_1), \ldots, (x_N, y_N)]$ para construir la función empírica de costo:

$$R_{emp}(h, w) = \frac{1}{N} \sum_{i=1}^{N} L(y(i), f_h(x(i), w)) \tag{2.15}$$

La función de costo empírica $R_{emp}(h, w)$ puede ser minimizada, al menos en teoría, respecto del vector de parámetros w para cada valor de h.

2.4. Optimización Estructural

La solución propuesta en esta tesis como se mencionó en la sección anterior, descompone el problema del aprendizaje en dos optimizaciones parciales: *la optimización estructural*, que determina la estructura óptima del aproximador funcional (dada por el índice estructural h^*) y por tanto su capacidad de representación, y *la optimización paramétrica*, que determina para cada estructura considerada de índice h, el óptimo vector de parametros $w^* \in W_h$.

De esta forma el optimizador estructural irá proponiendo una serie de estructuras candidatas según una estrategia determinada, luego encargará la evaluación de estos aproximadores al optimizador paramétrico y detendrá la optimización estructural cuando se cumplan ciertas condiciones de finalización.

La minimización estructural del riesgo tiene como objetivo determinar la estructura óptima del aproximador, en base a su capacidad de generalización.

La capacidad de generalización del aproximador se mide mediante el error de test (R_{test}), donde el error de generalización o error de test (R_{test}) se define como la frecuencia relativa de errores cometidos por el aproximador funcional sobre un conjunto de ejemplos no utilizados para el ajuste del aproximador, que denominaremos conjunto de test.

Para obtener la estructura óptima del aproximador funcional debemos basarnos en el siguiente lema:

> *A medida que la capacidad de representación funcional del aproximador funcional va creciendo, la capacidad de generalización del mencionado sistema, ajustado sobre el mismo conjunto de entrenamiento, se hace más incierta.*

Este lema establece como estrategia de optimización estructural el tomar como punto de partida estructuras sencillas e ir aumentando la complejidad del aproximador hasta alcanzar un mínimo en su capacidad de generalización.

Para ello, se formará una familia anidada de funciones de aproximación:

$$\{y = f_h(x, \omega)/x \in R^n, y \in R^m, w \in W_h; h = 1, \ldots, H\} \tag{2.16}$$

donde:

$$f_1 \subset f_2 \subset \ldots \subset f_H$$

Esto quiere decir que la función f_2 es capaz de realizar todas las transformaciones de entrada/salida realizables por f_1, la función f_3 es capaz de realizar todas las transformaciones de entrada/salida realizables por f_2 y f_1.

El optimizador estructural se limita a proponer como estructuras candidatas la secuencia de funciones $f_1, f_2, ...$, hasta detectar que el error de test devuelto por el optimizador paramétrico ha alcanzado un mínimo o se ha estabilizado.

La forma más usual y la que se usará en la presente tesis es cuando la propia estructura del aproximador permite definir un orden del mismo ligado directamente a su capacidad de representación ("Control directo de la estructura"). Por ejemplo, si se trata de aproximar una onda periódica por una serie de senos y cosenos de frecuencias múltiplos de la fundamental (Series de Fourier), el número

de armónicos considerado controlará directamente la capacidad de representación del aproximador.

2.5. Optimización Paramétrica

Una vez definida la estructura del aproximador funcional $\widehat{y} = f(x, w)$, la optimización paramétrica tiene como objetivo hallar el vector de parámetros óptimos w^*, que mejor asemeja la función aproximadora $\widehat{y}$ a la función subyacente y. En el siguiente capítulo haremos referencia a los métodos utilizados para la optimización paramétrica basados en la miniminización de la función de costo J:

$$J = \frac{1}{N} \sum_{i=1}^{N} e(i)^T e(i) \tag{2.17}$$

donde $e(i)$ es definido como $e(i) = y(i) - \widehat{y}(i)$ y N es el número total de muestras.

2.6. Determinación de la Influencia de las Variables de Entrada

Una vez ajustado el aproximador funcional, es posible analizar la influencia que cada una de las variables de entrada tiene en la salida del mismo, con el fin de identificar aquellas variables no relevantes, es decir que no aportan ninguna información a la salida. La eliminación de estas variables disminuye la complejidad del aproximador, aumentando su capacidad de generalización. Es posible comparar la influencia que cada una de las variables de entrada x_i tiene en la salida y, analizando las sensibilidades, definidas como:

$$\zeta_i = \frac{\partial y_i}{\partial x_i} \tag{2.18}$$

En el caso particular de un aproximador funcional lineal se tendría lo siguiente:

Sea $\widehat{y}(x,w)$ el aproximador funcional definido por:

$$\widehat{y}(x,w) = W \cdot X^T \tag{2.19}$$

donde:

$$W = [w1, \ldots, w_N]$$
$$X = [x1, \ldots, x_N]$$

Entonces, derivando la ecuación (2.19) respecto a W, tenemos:

$$\zeta = \frac{\partial \widehat{y}(x,w)}{\partial X^T} = \frac{\partial (W \cdot X^T)}{\partial X^T} = W \tag{2.20}$$

Normalizando las sensibilidades dividiendo por su valor máximo, obtendremos una medida relativa de la influencia de cada variable. De esta forma, cuanto menor sea el valor absoluto de estos ratios, menor será la influencia de la variable de entrada correspondiente, pudiendo eliminar aquellas variables que tienen una influencia relativa despreciable.

Por ejemplo, supongamos que se tiene el siguiente sistema lineal de segundo orden, expresado en ecuación de diferencias:

$$\begin{aligned} y_{k+1} \quad &= A_1 y_k + A_2 y_{k-1} + A_3 y_{k-2} + \ldots \\ &+ B_1 u_k + B_2 u_{k-1} + B_3 u_{k-2} \end{aligned}$$

donde:

$A_1 = 2{,}6277, A_2 = -2{,}3332, A_3 = 0{,}6976, B_1 = 0{,}0172, B_2 = 0{,}0308, B_3 = 0{,}0140$

Ahora supongamos que se tiene un aproximador funcional de la forma siguiente:

$$\begin{aligned}\widehat{y}_{k+1} &= a_1\widehat{y}_k + a_2\widehat{y}_{k-1} + a_3\widehat{y}_{k-2} + a_4\widehat{y}_{k-3} + a_5\widehat{y}_{k-4} + \dots \\ &+ b_1u_k + b_2u_{k-1} + b_3u_{k-2} + b_4u_{k-3} + b_5u_{k-4} + \dots \\ &+ e(k)\end{aligned}$$

donde: $e(k) = y(k) - \widehat{y}(k)$

Ahora si se somete ambos modelos a una entrada aleatoria de media cero y de varianza constante (ruido blanco) obtendremos las respuestas que se muestran en la figura 2.9 (se debe aclarar que los parámetros del vector W han sido hallados a través del método de los mínimos cuadrados que se verá en el siguiente capítulo).

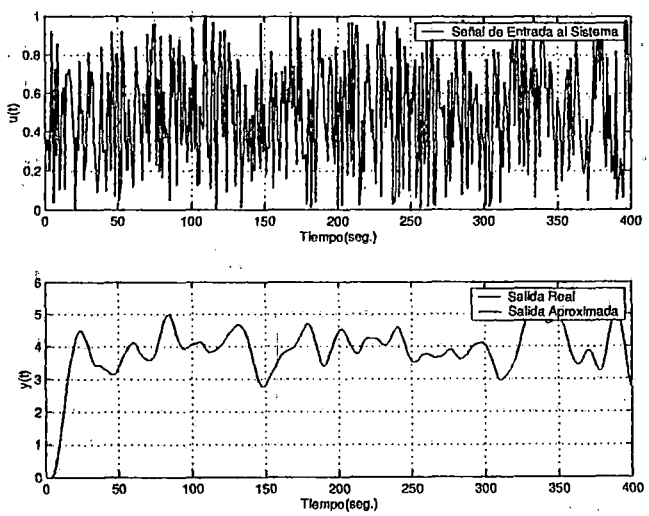

Figura 2.9: Salida del sistema y salida del aproximador funcional.

Donde el vector W tiene los siguientes valores:

$$\begin{aligned}W = [\ & -1{,}7572 \quad 0{,}3991 \quad 0{,}4061 \quad 0{,}2147 \quad -0{,}2452 \quad \dots \\ & 0{,}0172 \quad 0{,}0459 \quad 0{,}0470 \quad \dots \\ & 0{,}0232 \quad 0{,}0050 \quad 2{,}0364 \quad]\end{aligned} \tag{2.21}$$

Ahora normalizando el vector W dividiendo entre el máximo valor de dicho vector:

$$W_{normalizado} = [\quad 0{,}8629 \quad 0{,}1959 \quad 0{,}1994 \quad 0{,}1054 \quad 0{,}1204 \quad \ldots$$
$$0{,}0084 \quad 0{,}0225 \quad 0{,}0231 \quad \ldots$$
$$0{,}0114 \quad 0{,}0024 \quad 1{,}0000 \quad] \tag{2.22}$$

Se elimina los parámetros que tienen un *ratio* pequeño en comparación con los demás, es decir se va a eliminar las variables no influyentes de la estructura. Por lo tanto eliminaremos el w_5 (b_1u_k), w_6 (b_2u_{k-1}), w_7 (b_3u_{k-2}) y el w_8 (b_4u_{k-3}) entonces la estructura del modelo es el siguiente:

$$\widehat{y}_{k+1} = a_1\widehat{y}_k + a_2\widehat{y}_{k-1} + a_3\widehat{y}_{k-2} + a_4\widehat{y}_{k-3} + a_5\widehat{y}_{k-4} \tag{2.23}$$

Sometemos esta nueva estructura a la misma entrada a la que fue sometida la estructura anterior, los resultados son mostrados en la figura 2.10.

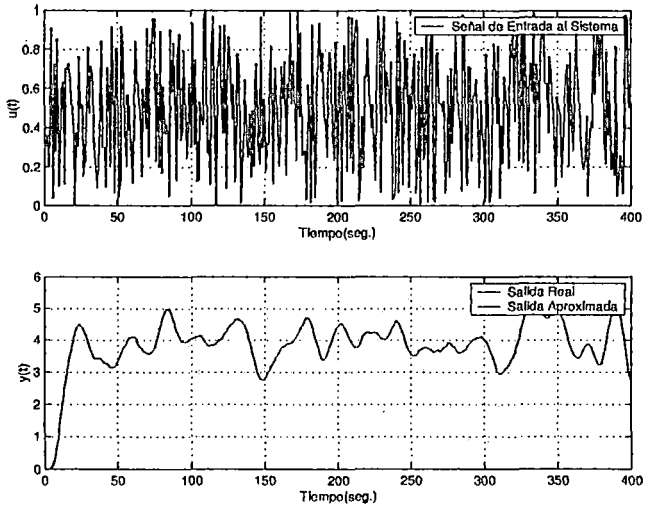

Figura 2.10: Salida del sistema y salida del aproximador funcional simplificado.

Como se puede observar en la figura 2.9 y en la figura 2.10, tanto la estructura sin simplificar como la simplificada cumplen con seguir a la respuesta del sistema con gran precisión, por lo que nos podemos quedar como una estructura candidata a la simplicada (de acuerdo al lema expuesto en la sección anterior) y además esta estructura tendría a las variables más influyentes, con las que se permite lograr una buena respuesta y obtener así una mayor generalización, es decir, una mejor aproximación frente a entradas que no han sido utilizadas para su entrenamiento. El programa de estos cálculos se encuentra en el *Apéndice A* con el nombre de *AES_1.m*

Cuando la relación entrada/salida no es lineal, las sensibilidades dejan de ser constantes y dependen del vector de entradas x. El único recurso que queda entonces para cuantificar la importancia de cada una de las variables de entrada, es el llamado *Análisis de las Distribuciones Estadísticas de las Sensibilidades.* Para ello podemos utilizar el conjunto de entrenamiento para evaluar el vector de sensibilidades con cada uno de los ejemplos y construir una matriz donde cada fila corresponda a un ejemplo y cada columna a una variable de entrada. Esta matriz puede ser filtrada eliminando las sensibilidades correspondientes a ejemplos que dan un error de aproximación superior a dos veces la desviación típica del mismo. Las variables de entrada no relevantes en la salida tendrán una distribución de sensibilidades centrada en el origen y de pequeña varianza. Las variables relevantes generarán sensibilidades no nulas en distintas regiones del espacio de entrada.

Por ejemplo, supongamos que se tiene el siguiente sistema no lineal [8]:

$$y_{k+1} = \frac{y_k}{1+y_k^2} + u_k^3$$

Ahora se desea encontrar un aproximador funcional, para ello partimos del siguiente modelo no lineal:

$$\widehat{y}_{k+1} = a_1\widehat{y}_k + a_2u_k + a_3e_k + a_4\widehat{y}_ku_k + \ldots$$

$$+ a_5\widehat{y}_k e_k + a_6 u_k e_k + a_7 \widehat{y}_k \widehat{y}_{k-1} + \ldots$$
$$+ a_8 u_k u_{k-1} + a_9 e_k e_{k-1}$$

Ahora partiendo de este modelo como principal, se empieza a crear una serie de estructuras, de la siguiente forma:

$$\begin{aligned}
\widehat{y}_{k+1} &= a_1\widehat{y}_k + a_3 e_k + a_4 \widehat{y}_k u_k + \ldots \\
&\quad + a_5\widehat{y}_k e_k + a_7 \widehat{y}_k \widehat{y}_{k-1} + \ldots \\
&\quad + a_8 u_k u_{k-1} \\
\widehat{y}_{k+1} &= a_1\widehat{y}_k + a_2 u_k + a_3 e_k + a_4 \widehat{y}_k u_k + \ldots \\
&\quad + a_6 u_k e_k + a_7 \widehat{y}_k \widehat{y}_{k-1} + \ldots \\
&\quad + a_9 e_k e_{k-1}
\end{aligned}$$

Luego a todas las estructuras que se ha creado las sometemos a una entrada cuadrada. Después con los diferentes pesos de cada estructura $W_1, W_2, \ldots, W_H$ se halla la media y la desviación estándar de cada parámetro y obtendremos la figura 2.11.

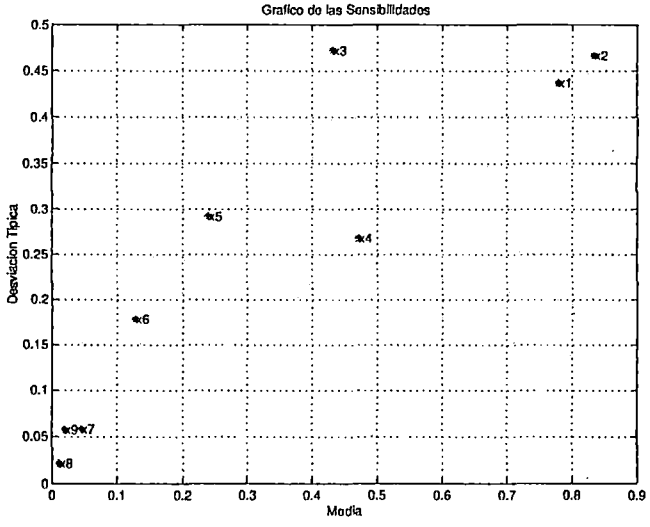

Figura 2.11: Gráfico de la influencia de las variables de entrada.

En la figura 2.11 los valores x_i indican el término w_i del aproximador funcional. Además las variables que se alejan de cero son las variables influyentes, y las que oscilan alrededor de cero se desprecian, por lo tanto las varibles de los parámetros x_1, x_2, x_3, x_3, x_5 y x_6 son las variables influyentes: y_k, u_k, e_k, $y_k u_k$, $y_k e_k$ y $u_k e_k$.

Entonces nuestro modelo estructural es el siguiente:

$$\begin{aligned} \widehat{y}_{k+1} &= a_1\widehat{y}_k + a_2 u_k + a_3 e_k + a_4 \widehat{y}_k u_k \ldots \\ &+ a_5 \widehat{y}_k e_k + a_6 u_k e_k \end{aligned}$$

La respuesta del sistema frente a la entrada de una señal cuadrada es mostrada en la figura 2.12.

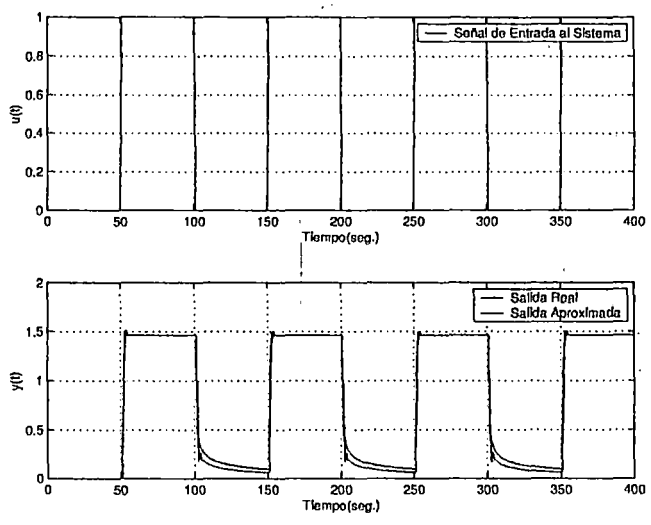

Figura 2.12: Respuesta del aproximador funcional.

Ahora para comprobar (validación del sistema) si el aproximador funcional hallado cumple con representar la dinámica del sistema real vamos a someter a ambos sistemas a una entrada senoidal (ver gráfica 2.13).

Como se observa en la figura 2.13 el aproximador funcional encontrado cumple con la dinámica del sistema, por lo tanto el modelo es válido.

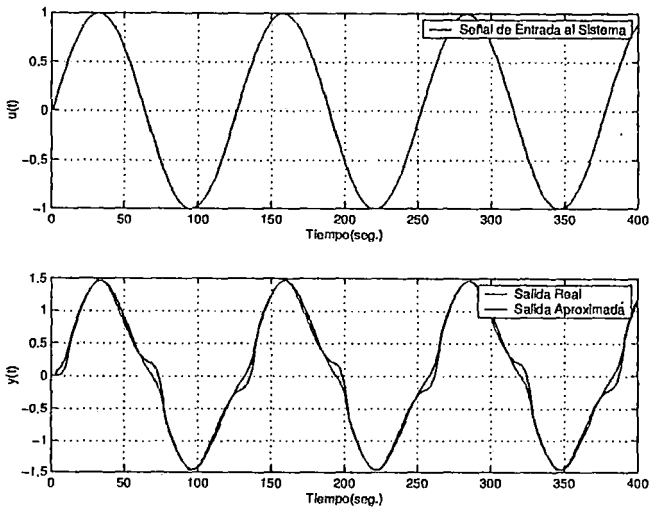

Figura 2.13: Respuesta del aproximador funcional (validación del aproximador).

2.7. Validación de Modelo

Una vez que se tiene definido la estructura del aproximador funcional, el siguiente paso será el de encontrar los parámetros óptimos de W que minimizen el error entre la salida real y la salida aproximada. En los capítulos posteriores se detallarán los diferentes métodos utilizados para lograr este fin. Luego de haber hallado la estructura óptima y los parámetros W óptimos, se procederá a validar dicho aproximador funcional, para ello se recurre a dos procedimientos:

1. *Análisis de Residuos.*- Se conocen como residuos de un sistema a los errores de predicción obtenidos según la expresión: $e_k = y_k - \widehat{y}_{(k,W)}$ siendo W el vector de parámetros del modelo, y_k la respuesta real del sistema e $\widehat{y}_{(k,W)}$ la respuesta estimada por el modelo para la misma entrada. Idealmente, estos residuos deben ser independientes de la entrada. Si no sucede así, significa que hay componentes en e_k que proceden de la entrada u_k, lo cual a su vez significa que el modelo no es capaz de describir completamente la dinámica del sistema. Para realizar el estudio anterior, suele comprobarse la correlación entre el error de predicción y la entrada al sistema, según la expresión:

$$R_{eu} = \frac{1}{N}\sum_{t=1}^{N} e(t+\tau)u(t) \tag{2.24}$$

El modelo por tanto será tanto más exacto cuanto más se acerquen a cero los términos de la correlación anterior. Puede demostrarse que si $e(t)$ y $u(t)$ son realmente independientes, la expresión anterior (para valores grandes de N) es una distribución normal, con media cero y varianza constante:

$$P_\tau = \frac{1}{N}\sum_{k} R_e(k)R_u(k) \tag{2.25}$$

Donde R_e y R_u son las covarianzas de $e(t)$ y $u(t)$ respectivamente. Generalmente, $R_{eu}(t)$ se representa en un diagrama junto con las líneas $\pm 3\sqrt{P_\tau}$. Si todos los coeficientes de R_{eu} están dentro de este intervalo, aceptaremos la hipótesis de independencia de los residuos. Resultados valederos son cuando uno de cada veinte coefientes salen fuera de rango (confianza del 95 %).

Por ejemplo, del problema anterior su grafico de la correlación cruzada se muestra en la figura 2.14

En la figura 2.14 podemos observar que el número de puntos que se salen de los límites establecidos son menores que el 5 % del total de datos por tanto podemos aseverar que el aproximador funcional encontrado representa fielmente al sistema real (el programa con los cálculos se encuentra en el *Apéndice A* con el nombre de *AES_2.m*).

2. *Análisis de Datos No utilizados.*- Es cuando luego de hallar el aproximador funcional, lo evaluamos con una entrada que no halla sido utilizada para las pruebas (en forma análoga al ejemplo anterior - ver gráfico 2.13).

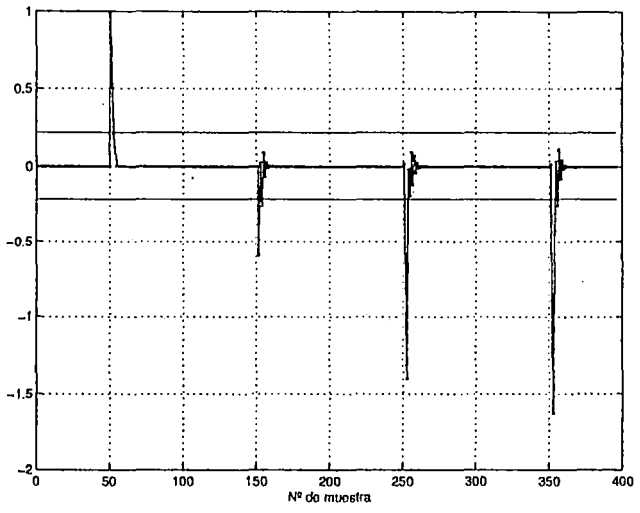

Figura 2.14: Función de correlación cruzada.

2.8. Esquema General de la Aproximación Funcional

A continuación se presenta tres diagramas de flujo en los cuales se pretende sistematizar cada uno de los procesos para la identificación paramétrica de sistemas por medio del ajuste de aproximadores funcionales.

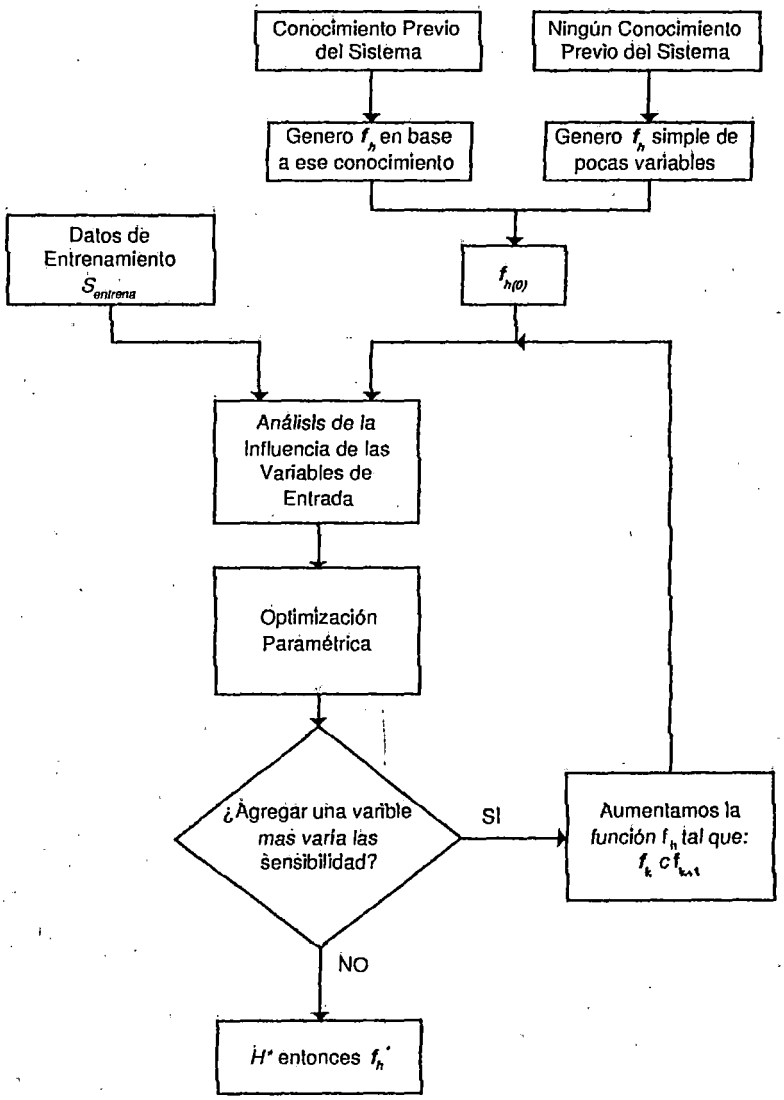

Figura 2.15: Diagrama de flujo para la optimización estructural.

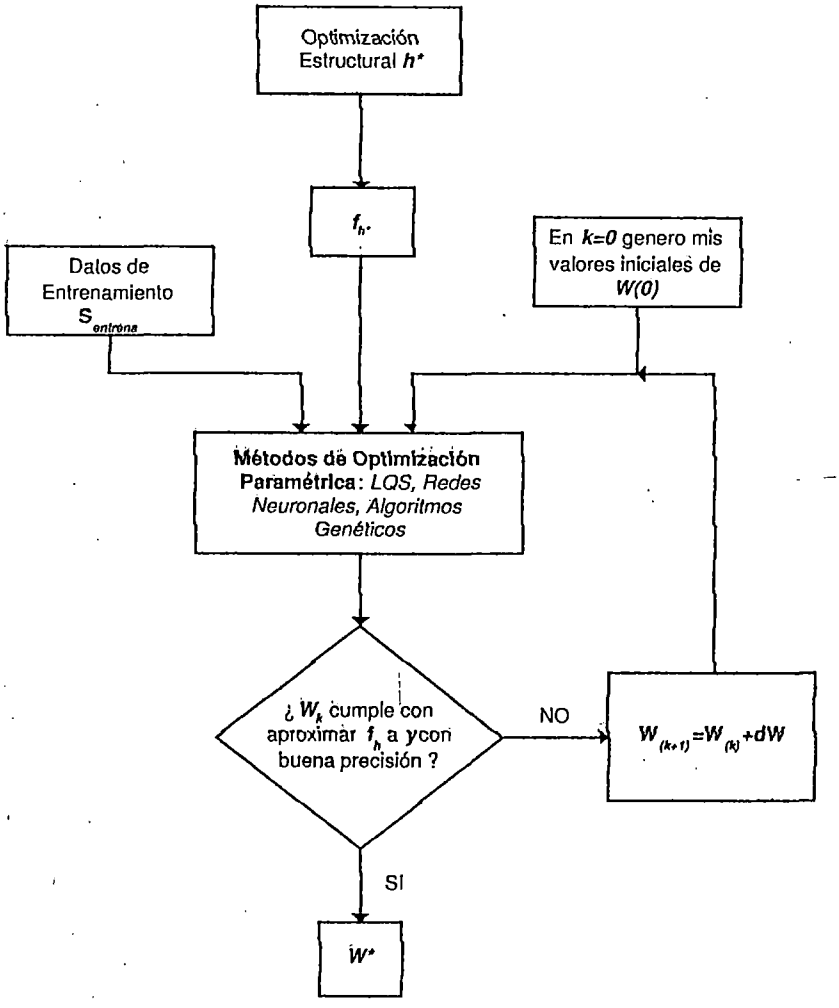

Figura 2.16: Diagrama de flujo para la optimización paramétrica.

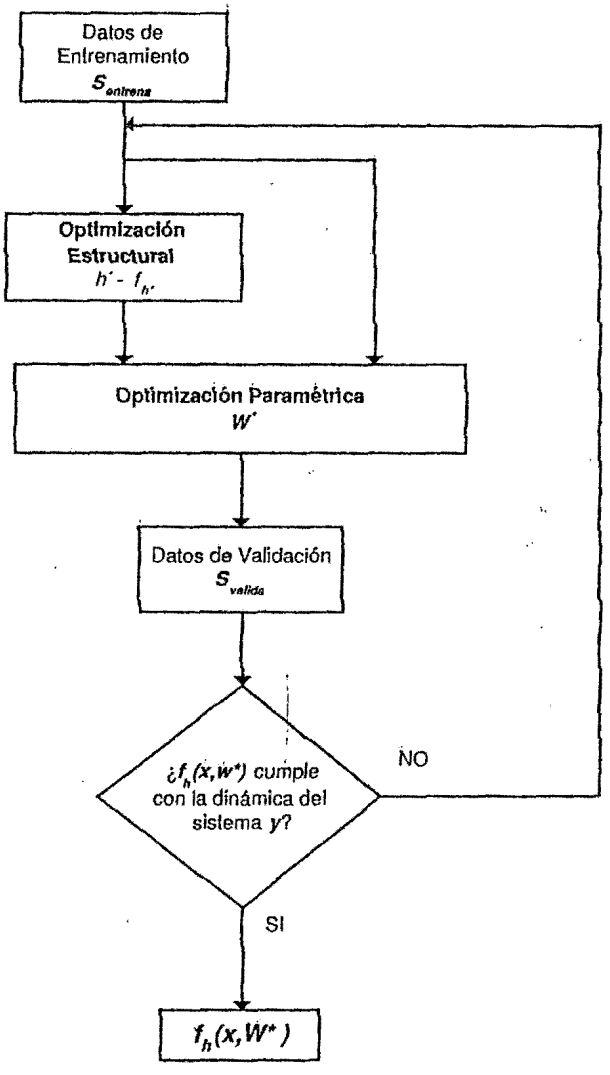

Figura 2.17: Diagrama de flujo para la identificación paramétrica de sistemas mediante aproximadores funcionales.

CAPÍTULO III
APROXIMACIÓN DE SISTEMAS DINÁMICOS NO LINEALES

En las últimas décadas se ha consolidado la teoría y la práctica para la identificación de sistemas. En los inicios los modelos matemáticos para representar los sistemas eran lineales a pesar de que no conseguían reproducir el comportamiento dinámico no lineal de los sistemas. Con el avance tecnológico e industrial, un interés por el modelamiento no lineal fue en aumento, ayudados por el creciente desarrollo de herramientas matemáticas (debido también a que las técnicas existentes para modelos lineales no conseguían reproducir toda la gama de comportamientos dinámicos de los sistemas reales).

Dentro de las representaciones no lineales podemos destacar los modelos de bloques interconectados que representan a los sistemas a través de bloques. La dinámica del sistema es representada por un modelo dinámico lineal y un modelo dinámico no lineal. Una gran ventaja de tal representación es la posibilidad de utilizar técnicas de identificación lineal. Los modelos de bloques interconectados fueron utilizados a mediados de los ochenta. Por aquellos años Billings y Leontaritis presentaron el modelo NARMAX (Nonlinear AutoRegressive Moving Average model with eXogenous inputs). Este modelo es capaz de representar una amplia gama de sistemas no lineales. Varias técnicas de selección de estructuras y estimación de parámetros fueron desarrollados para este modelo.

A mediados de la década de los noventa, el interés por los modelos de bloques interconectados resurgió. Un factor principal fue la facilidad con que estos modelos podían trabajar con las diferentes técnicas de control.

En el presente capítulo se explicará los diferentes modelos lineales de identificación de sistemas luego nos centraremos en el análisis del método de los

mínimos cuadrados para la estimación de los parámetros. Se hará un estudio de las diferentes modelos no lineales ahondando en el modelo polimomial NARMAX.

3.1. Aspectos Generales

Se denomina identificación a la técnica de construir un modelo a partir de las variables medidas del proceso: entradas o variables de control, salidas o variables controladas y, posiblemente, perturbaciones. En principio y con el objetivo de modelizar se pueden proponer tres formas distintas de utilizar los métodos de identificación:

- Hacer distintas aproximaciones para estructurar el problema: seleccionar las señales de interés, observar la dependencia entre ellas, estudiar el grado de linealidad del proceso.

- Construir un modelo que describa el comportamiento entre las entradas y las salidas, prescindiendo del comportamiento físico. Hay distintas formas de abordar el problema, según se consideren modelos no paramétricos o modelos paramétricos.

- Utilizar los datos para determinar los parámetros no conocidos del modelo físico obtenido a base del estudio de propiedades y leyes físicas del proceso estudiado.

Otro aspecto a tener en cuenta será el tipo de modelo matemático que se pretende identificar. Hay varias formas de catalogar los modelos matemáticos: deterministas o estocásticos, dinámicos o estáticos, de parámetros distribuidos o concentrados, lineales o no lineales, y de tiempo continuo o tiempo discreto. A continuación se describe algunas de ellas:

- *Deterministas* ya que se quiere estudiar la relación entre la entrada y la salida con una parte no modelada o no conocida (estocástica).

- *Dinámicos* porqué el objetivo es conocer el comportamiento dinámico de un proceso.

- *De parámetros concentrados* no se considera la variación en función del espacio.

- *Lineales o no lineales.*

3.2. Estructura del modelo

Partiendo de la base de que para modelizar un proceso necesitamos los datos observados, en el caso de un sistema dinámico con una entrada en el instante t denominada como *u(t)* y una salida en el instante t denominada como *y(t)* los datos serán una colección finita de observaciones:

$$S^N = \{u(0), y(0), u(1), y(1), \ldots, u(N), y(N)\} \tag{3.1}$$

El problema de los métodos de identificación consiste en encontrar relaciones matemáticas entre las secuencias de entrada y las secuencias de salida.

Ahora definiendo las observaciones de forma más general:

$$S^N = \{y(t), \phi(t)\} \qquad t = 1, \ldots, N \tag{3.2}$$

El problema radica en como determinar *y(t+1)* a partir de $\phi(t)$. En el caso de un sistema dinámico, $\phi(t)$ contendría la información de las entradas y salidas anteriores a t. El problema matemático que se fórmula es la construcción de una función $\widehat{g}_N(t, \phi(t))$ tal que a partir de ella podamos determinar:

$$\widehat{y}(t+1) = \widehat{g}_N(t, \theta, \phi(t)) \tag{3.3}$$

En general se busca una función $\widehat{g}_N(t, \theta, \phi(t))$ que sea parametrizable, es decir que tenga un número finito de parámetros. A estos parámetros se les denomina con θ. A toda la familia funciones candidatas se las denomina estructura del modelo, y en general estas funciones se escriben como $\widehat{g}_N(t, \theta, \phi(t))$. Esta función permite calcular el valor $\widehat{y}(t+1)$:

$$\widehat{y}(t+1) \approx \widehat{g}_N(t, \theta, \phi(t)) \tag{3.4}$$

La búsqueda de una buena función se realiza en términos del parámetro θ y el cálculo del valor $\widehat{\theta}_N$ conduce a que $\widehat{g}_N(t, \phi(t)) = g_N(t, \widehat{\theta}_N, \phi(t))$, por ejemplo en el caso de una estructura de modelo simple como ARX de primer orden (este modelo lo veremos con mayor detalle en las secciones siguientes):

$$\widehat{y}(t) + a_1 y(t-1) = b_1 u(t-1) + b_2 u(t-2)$$

la correspondencia con la formulación general sería:

$$\begin{aligned} \theta &= [a_1, b_1, b_2] \\ \phi(t) &= [y(t-1), u(t-1), u(t-2)] \\ g(t, \theta, \phi(t)) &= -a_1 y(t-1) + b_1 u(t-1) + b_2 u(t-2) \end{aligned}$$

El ejemplo anterior muestra la formulación convencional de los sistemas de identificación, en que la estructura del modelo se corresponde con una regresión lineal.

3.3. Aproximación mediante modelos lineales

La mayoría de los sistemas tienen un comportamiento no lineal, excepto en un determinado rango de operación donde pueden ser considerados lineales. Los modelos lineales aproximan al sistema no lineal alrededor de un punto de operación y la performance del modelo lineal (y de sus características predictivas) se ven deterioradas al variar el punto de operación del sistema no lineal.

De acuerdo a lo expuesto en la sección anterior, supongamos que el sistema puede ser modelado como un proceso estable, invariante en el tiempo y linealizable, con una entrada y una salida, por lo que puede ser descrito por una ecuación lineal en diferencias de la forma:

$$\begin{aligned} y(k) \; &+a_1 y(k-1) + \ldots + a_n y(k-n) = b_1 u(k-d-1) + b_2 u(k-d-2) + \ldots \\ &b_n u(k-d-n) + v(k) + c_1 v(k-1) + c_2 v(k-2) + \ldots + c_n v(k-n) \end{aligned} \quad (3.5)$$

o en forma vectorial:

$$y(k) = \phi^T(k)\theta + v(k) \quad (3.6)$$

donde:

$$\phi^T(k) = [-y(k-1), \ldots, -y(k-n), u(k-d-1), \ldots, u(k-d-n), v(k), \ldots, v(k-n)]$$

$$\theta^T = [a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n]$$

Se tiene que $v(k)$ es una señal de ruido estadísticamente independiente y estacionaria con distribución normal y de media nula. Entonces la función de transferencia en z de este sistema puede describirse como:

$$y(z) = \frac{B(z^{-1})}{A(z^{-1})} z^{-d} u(z) + \frac{C(z^{-1})}{A(z^{-1})} v(z) \quad (3.7)$$

donde:

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n} \\ B(z^{-1}) &= b_1 z^{-1} + b_2 z^{-2} + \ldots + b_n z^{-n} \\ C(z^{-1}) &= 1 + c_1 z^{-1} + c_2 z^{-2} + \ldots + c_n z^{-n} \end{aligned}$$

El primer cociente $B(z^{-1})/A(z^{-1})$ representa el modelo de la planta, y el segundo $C(z^{-1})/A(z^{-1})$ representa el modelo de las perturbaciones.

3.3.1. Modelo OE

En el modelo de error de salida *OE*, se incluyen como entradas valores pasados de la salida estimada. El modelo *OE* (Ouput Error) tiene la estructura que se muestra en la figura 3.1.

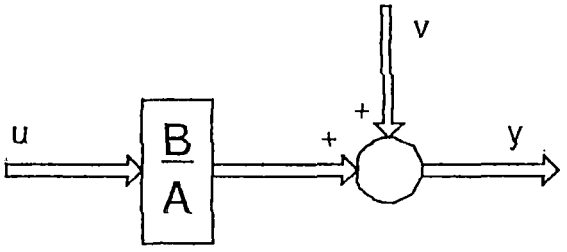

Figura 3.1: Estructura del modelo OE.

De acuerdo a la estrutura mostrada en la figura 3.1 y según la ecuación (3.7) tenemos lo siguiente:

$$G(z^{-1}) = \frac{B(z^{-1})}{A(z^{-1})} \tag{3.8}$$

$$H(z^{-1}) = 1 \tag{3.9}$$

Ahora a partir de la ecuación (3.5) y según las ecuaciones anteriores (3.8) y (3.9) tenemos:

$$\begin{aligned} y(k) &= -a_1 y(k-1) - \ldots - a_n y(k-n) + b_1 u(k-d-1) + b_2 u(k-d-2) + \ldots \\ &\quad + b_n u(k-d-m) + v(k) \end{aligned} \tag{3.10}$$

De la ecuación (3.6):

$$y(k) = \theta^T \phi(k) \tag{3.11}$$

$$\widehat{y}(k/\widehat{\theta}) = \widehat{\theta}^T \phi(k) \tag{3.12}$$

$$e(k) = y(k) - \widehat{y}(k/\widehat{\theta}) = v(k) \tag{3.13}$$

donde:

$\phi^T(k)$=$[-y(k-1), \ldots, -y(k-n), u(k-d-1), \ldots, u(k-d-m), v(k)]$

$\theta^T = [a_1, \ldots, a_n, b_1, \ldots, b_m, 1]$

Las ecuaciones (3.11), (3.12) y (3.13) son las ecuaciones del Modelo OE. Esta estructura permite modelar la mayoría de los procesos lineales, pero no la característica del ruido. Al ser un modelo recurrente, el ajuste del vector de parámetros $\theta(t)$ es más complicado.

3.3.2. Modelo ARX

Una forma de evitar la recurrencia del modelo de error de salida es suministrar como entrada la salida real del proceso, en lugar de la salida estimada. Esto da lugar al modelo autoregresivo con entradas exógenas *ARX*. El modelo *ARX* (**A**uto**R**egressive with e**X**tra input) tiene la estructura que se muestra en la figura 3.2.

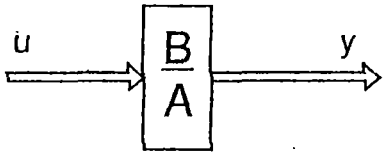

Figura 3.2: Estructura del modelo ARX.

De acuerdo a la estrutura mostrada en la figura 3.2 y según la ecuación (3.7) se tiene lo siguiente:

$$G(z^{-1}) = \frac{B(z^{-1})}{A(z^{-1})} \tag{3.14}$$

$$H(z^{-1}) = 0 \tag{3.15}$$

Ahora a partir de la ecuación (3.5) y según las ecuaciones (3.14) y (3.15), obtenemos:

$$\begin{aligned} y(k) &= -a_1 y(k-1) - \ldots - a_n y(k-n) + \ldots \\ &\quad + b_1 u(k-d-1) + \ldots + b_m u(k-d-m) \end{aligned} \tag{3.16}$$

De la ecuación (3.6):

$$y(k) \triangleq \theta^T \phi(k) \tag{3.17}$$

$$\widehat{y}(k/\widehat{\theta}) = \widehat{\theta}^T \phi(k) \tag{3.18}$$

donde:

$\phi^T(k)$=$[\, -y(k-1), \ldots, -y(k-n), u(k-d-1), \ldots, u(k-d-m)\,]$

$\theta^T = [\, a_1, \ldots, a_n, b_1, \ldots, b_m\,]$

Las ecuaciones (3.17) y (3.18) son las ecuaciones del Modelo ARX. Este modelo permite ajustar todo proceso lineal corrompido por un ruido aditivo, a costa de aumentar los retardos n y m. Su ventaja es que requiere menos parámetros y al ser un modelo no recurrente, la estimación de sus parámetros $\theta(t)$ es más sencilla.

3.3.3. Modelo ARMAX

Es frecuente en la predicción de series temporales que el análisis de correlaciones residuales muestre un alto contenido de información en la serie del error de predicción. Esta información, enmascarada bajo forma de ruido coloreado[1], puede ser realimentada al estimador de tal forma que su utilización mejore considerablemente la calidad de la estimación, sin aumentar de forma desproporcionada el número de parámetros a estimar. El modelo *ARMAX* (AutoRegressive Moving Average with eXtra input) tiene la estructura que se muestra en la figura 3.3.

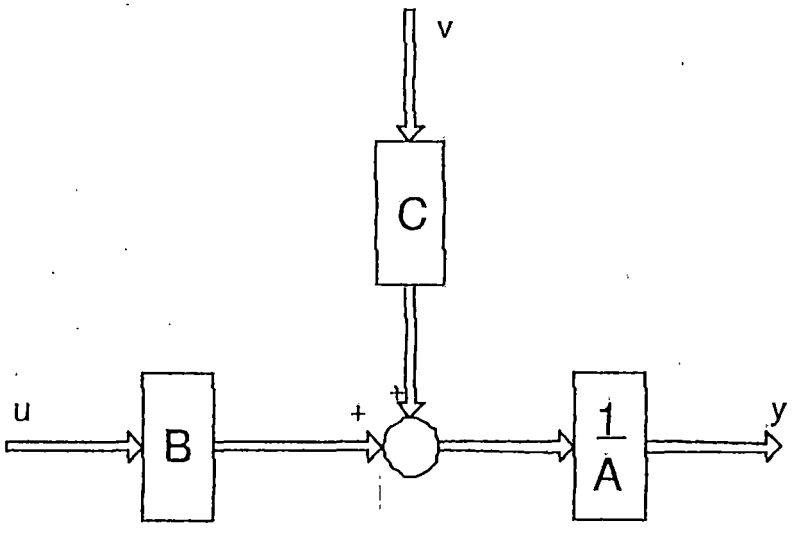

Figura 3.3: Estructura del modelo ARMAX.

De acuerdo a la estrutura mostrada en la figura 3.3 y según la ecuación (3.7) se tiene lo siguiente:

$$G(z^{-1}) = \frac{B(z^{-1})}{A(z^{-1})} \tag{3.19}$$

$$H(z^{-1}) = \frac{C(z^{-1})}{A(z^{-1})} \tag{3.20}$$

[1]Ruido Coloreado es la secuencia de números aleatorios de media cero y varianza uno.

Ahora a partir de ecuación (3.5) y según las ecuaciones anteriores (3.19) y (3.20) se obtiene:

$$\begin{aligned} y(k) &= -a_1 y(k-1) - \ldots - a_n y(k-n) + b_1 u(k-d-1) + b_2 u(k-d-2) + \ldots \\ & b_n u(k-d-m) + v(k) + c_1 v(k-1) + c_2 v(k-2) + \ldots + c_n v(k-p) \end{aligned} \quad (3.21)$$

De la ecuación (3.6):

$$y(k) = \theta^T \phi(k) \quad (3.22)$$

$$\widehat{y}(k/\widehat{\theta}) = \widehat{\theta}^T \phi(k) \quad (3.23)$$

$$e(k) = y(k) - \widehat{y}(k/\widehat{\theta}) = v(k) \quad (3.24)$$

donde:

$$\phi^T(k) = [\, -y(k-1), \ldots, -y(k-n), u(k-d-1), \ldots, u(k-d-m), v(k), \ldots, v(k-p) \,]$$

$$\theta^T = [\, a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n \,]$$

Las ecuaciones (3.22), (3.23) y (3.24) son las ecuaciones del Modelo ARMAX. Esta estructura permite modelar todos los procesos lineales corrompidos por un ruido aditivo, con un modelo mucho más compacto que el modelo ARX (en el caso de ruido coloreado). Su principal problema es que es un modelo recurrente, lo que complica la estimación de sus parámetros $\theta(t)$.

3.4. Método de los Mínimos Cuadrados

Es una identificación paramétrica, es decir se parte de que la estructura de las ecuaciones es conocida y son sus coeficientes lo que se desea determinar. Se

estudiará primero el algoritmo de mínimos cuadrados para obtener unos coeficientes constantes con las N primeras medidas. En la práctica, como el tiempo de cómputo es elevado, la identificación suele realizarse *off-line*.

En el algoritmo de mínimos cuadrados recursivos se irá modificando los coeficientes secuencialmente, con cada nueva medida posterior a N y se obtendrán los valores en régimen permanente de estos coeficientes. Este proceso puede realizarse *on-line*, por lo que los coeficientes serán dinámicos, adaptándose en cada instante para que el error entre el modelo y la planta sea mínimo.

3.4.1. Método de los Mínimos Cuadrados no recursivo (Algoritmo "One-Shot")

La idea central es obtener, por minimización del error de la salida $y(k)$ de un sistema muestreado y del sisteta aproximado $\widehat{y}(k)$, los valores de los parámetros $\theta(t)$ de la ecuación(3.22), suponiendo que son conocidos a priori el retardo d, el orden n y m $(n \geq m)$ de los polinomios A y B de un modelo ARMAX con retardo

Volviendo a escribir las ecuaciones (3.22),(3.23) y (3.24) del modelos ARMAX (pero tomando el retardo de la señal residual[2] $p = 0$), tenemos que:

$$y(k) = \theta^T \phi(k)$$

$$\widehat{y}(k/\widehat{\theta}) = \widehat{\theta}^T \phi(k)$$

$$e(k) = y(k) - \widehat{y}(k/\widehat{\theta}) = v(k)$$

donde:

$$\phi^T(k) = [\ -y(k-1), \ldots, -y(k-n), u(k-d-1), \ldots, u(k-d-m), v(k)\ ]$$

$$\theta^T = [\, a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_m, d_1 \,]$$

[2]El error $e(k)$ también es llamado *error predictivo* o *valor residual* y se asume que su media es cero y su varianza constante.

Si se realizan mediciones de pares de entrada salida en cada instante de muestreo $(y(k), u(k))$ desde $k=1,\ldots,N$ donde $N \gg n$, entonces el vector E de errores de predicción se puede expresar a partir del vector Y de salidas medidas y de la matriz de regresión ϕ:

$$E = Y - \phi^T\theta \tag{3.25}$$

$$Y^T = [y_1, \ldots, y_N] \tag{3.26}$$

$$\phi^T = [\varphi_0, \varphi_1, \ldots, \varphi_{N-1}] \tag{3.27}$$

$$E^T = [e_1, e_2, \ldots, e_{N-1}] \tag{3.28}$$

Ahora definiendo la función de costo $V_N(\theta)$ como:

$$V_N(\theta) = \frac{1}{2}\sum_{i=1}^{N} e_i^2 \tag{3.29}$$

Reemplazando la ecuación (3.25) en la ecuación (3.29) se tiene que:

$$V_N(\theta) = \frac{1}{2}\sum_{i=1}^{N} (Y - \phi^T\theta)^T(Y - \phi^T\theta) \tag{3.30}$$

Luego, el conjunto de parámetros θ se puede obtener por minimización del índice $V_N(\theta)$ (suma de los cuadrados de los valores residules). Entoces derivando la ecuación (3.30)

$$\frac{\partial V_N(\theta)}{\partial \theta} = \frac{\partial(\frac{1}{2}\sum_{i=1}^{N}(Y-\phi^T\theta)^T(Y-\phi^T\theta))}{\partial\theta}$$

$$= -\phi(Y-\phi^T\theta) \tag{3.31}$$

Ahora derivando la ecuación (3.31)

$$\frac{\partial^2 V_N(\theta)}{\partial\theta^2} = \phi\phi^T > 0 \tag{3.32}$$

Para minimizar el índice $V_N(\theta)$ igualamos a cero la ecuación (3.31) y despejamos el vector de parámetros θ:

$$\theta = (\phi\phi^T)^{-1}\phi Y \tag{3.33}$$

$$\theta = P\phi Y \tag{3.34}$$

donde: $P=(\phi\phi^T)^{-1}$

La matriz $\phi\phi^T$ se conoce como la *matriz de covarianza* y a la matriz P se le conoce como *matriz de varianza-covarianza.*

El diagrama de flujo de este método se puede observar en la figura 3.4. Este procedimiento es poco práctico en tiempo real, al exigir un elevado tiempo de computación, debido a que requiere la inversión de una matriz de orden $2N \times 2N$, además en el cálculo de matrices inversas pueden cometerse errores importantes dependiendo de la precisión de la computadora.

La matriz de covarianzas es simétrica positiva (los elementos de su diagonal son todos positivos) definida como:

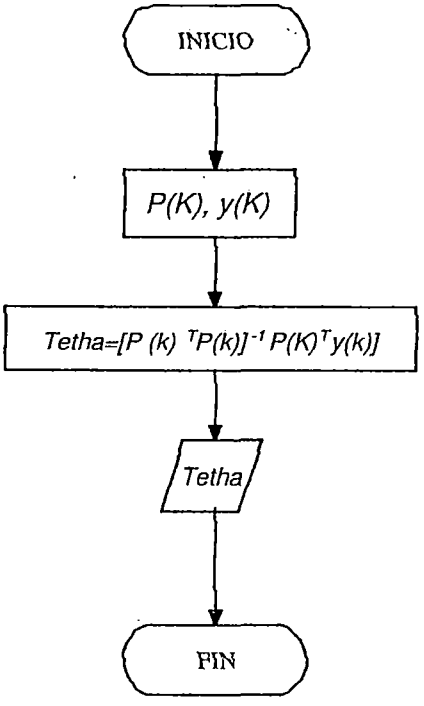

Figura 3.4: Diagrama de flujo del método de Mínimos Cuadrados no recursivo.

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

donde p_{11} y p_{22} representan sumas de cuadrados de salidas y entradas, respectivamente, mientras que p_{12}=p_{21} contiene sumas de cuadrados cruzados de entradas y salidas.

3.4.2. Método de los Mínimos Cuadrados Recursivo

Los métodos recursivos aprovechan parte de los cálculos realizados en un paso anterior, para el realizar el siguiente. Por lo tanto el cálculo de los parámetros en un instante se realiza de la siguiente manera:

$$\theta_{k+1} = \theta_k + \textit{corrección} \tag{3.35}$$

Como el objetivo es encontrar θ_{k+1}, entonces modificando la ecuación (3.33) se tendrá:

$$\theta_{k+1} = (\phi_{k+1}\phi_{k+1}^T)^{-1}\phi_{k+1}Y_{k+1} \tag{3.36}$$

Pero como se sabe de la ecuación (3.27) que:

$$\phi_k^T = [\varphi_0, \varphi_1, \ldots, \varphi_{k-1}]$$

Entonces en $k+1$ se tendrá lo siguiente:

$$\phi_{k+1}^T = [\varphi_0, \varphi_1, \ldots, \varphi_k] \tag{3.37}$$

Por lo tanto:

$$\phi_{k+1}\phi_{k+1}^T = \begin{bmatrix} \varphi_1 & \varphi_2 & \cdots & \varphi_k \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_k \end{bmatrix}$$

$$\phi_{k+1}\phi_{k+1}^T = \phi_k\phi_k^T + \varphi_k\varphi_k^T \tag{3.38}$$

Y también:

$$\phi_{k+1}Y_{k+1}^T = \begin{bmatrix} \varphi_1 & \varphi_2 & \cdots & \varphi_k \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{k+1} \end{bmatrix}$$

$$\phi_{k+1}Y_{k+1}^T = \phi_k Y_k + \varphi_k y_{k+1} \tag{3.39}$$

Ahora reemplazando la ecuación (3.38) y la ecuación (3.39) en la ecuación (3.36) tenemos:

$$\theta_{k+1} = [\,\phi_k\phi_k^T + \varphi_k\phi_k\,]^{-1}\phi_k Y_k + \varphi_k y_{k+1} \tag{3.40}$$

De la teoría de matrices[4] tenemos que:

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

Identificando términos y comparándolos con la ecuación (3.40) se deduce:

$$A = \phi_k\phi_k^T \qquad B = \varphi_k \qquad C = 1 \qquad D = \varphi_k^T$$

Con lo cual se tiene:

$$\theta_{k+1} = [(\phi_k\phi_k^T)^{-1} - (\phi_k\phi_k^T)^{-1}(1 + \varphi_k^T(\phi_k\phi_k^T)^{-1}\varphi_k)^{-1}\varphi_k^T(\phi_k\phi_k^T)^{-1}](\phi_k Y_k + \varphi_k y_{k+1})$$

Pero se sabe que $P_k = (\phi_k\phi_k^T)^{-1}$ y reemplazando en la fórmula anterior se obtiene:

$$\theta_{k+1} = [P_k - P_k(1 + \varphi_k^T P_k\varphi_k)^{-1}\varphi_k^T P_k](\phi_k Y_k + \varphi_k y_{k+1}) \tag{3.41}$$

Pero de la ecuación (3.38) tenemos que:

$$P_{k+1}^{-1} = P_k^{-1} + \varphi_k\varphi_k^T$$

Con lo cual la ecuación (3.41) se reduce a:

$$\theta_{k+1} = \theta_k + \frac{P_k\varphi_k}{1 + \varphi_k^T P_k\varphi_k}(y_{k+1} - \varphi_k^T\theta_k) \tag{3.42}$$

La ecuación (3.41) es conocida como el *Algoritmo de identificación recursiva de Mínimos Cuadrados (RLS).*

$$\theta_{k+1} = \theta_k + L_{k+1}e_{k+1} \tag{3.43}$$

donde:

$$L_{k+1} = \frac{P_k\varphi_k}{1 + \varphi_k^T P_k \varphi_k}$$

$$e_{k+1} = (y_{k+1} - \varphi_k^T \theta_k)$$

La estimación de los parámetros θ está dada por los valores de la estimación anterior corregida por un término lineal del error entre la salida y su predicción siendo L_{k+1} la ganacia de corrección.

El algoritmo RLS se puede resumir de la siguiente forma:

1. Defina valores iniciales para θ_k, P_k.

2. Forme el vector de regresión φ_k.

3. Calcule la matriz L_{k+1}:

$$L_{k+1} = \frac{P_k\varphi_k}{1 + \varphi_k^T P_k \varphi_k}$$

4. Mida el par de entrada-salida (u_{k+1}, y_{k+1}) del sistema muestreado.

5. Calcule el nuevo vector θ_{k+1} y la nueva matriz P_{k+1}:

$$\theta_{k+1} = \theta_k + L_{k+1}e_{k+1}$$

$$P_{k+1} = P_k - L_{k+1}\varphi k \varphi k^T P_k$$

6. Haga P_k=P_{k+1}, θ_k= θ_{k+1} y repita desde el punto 2

3.4.3. Modificaciones del Algoritmo de RLS Recursivo

Se ha supuesto que los parámetros del sistema son invariantes en el tiempo, por lo que el identificador formulado tiene memoria infinita esto quiere decir que se les da un mismo peso a todas las medidas obtenidas.

Sin embargo si los parámetros del sistema varían lentamente (debido a desgaste de los equipos, a que el sistema es no lineal, entre otros) es conveniente reducir la memoria del identificador con el objeto de que éste pueda seguir las variaciones del sistema dando mayor peso a las últimas medidas sobre las más antiguas. Esto puede hacerse introduciendo algunas mejoras [4]:

- Introduciendo un factor de olvido c.
- Sumando una matriz positiva R a la matriz de covarianza P del identificador.
- Utilizando ambas ténicas.

Inclusión del Factor de Olvido. Como se mencionó anteriormente mediante la inclusión del factor de olvido se consigue que el identificador tenga memoria finita. La modificaión consiste en sustituir P_k por P_k/c.

Para $c = 1$ se tiene el algoritmo de mínimos cuadrados normal, mientras que para $c < 1$ el algoritmo olvida las medidas más anitguas. Los valores de c se hacen variar entre 0.95 y 1. Para valores de $c = 1$ se obtiene una gran eliminación del ruido y para valores de $c < 1$ se obtiene un mejor seguimiento de la variación de los parámetros.

El índice $V_N(\theta)$ pasa a ser:

$$V_N(\theta) = \sum_{n=1}^{k} c^{k-n} e^2(k)$$

Suma de una Matriz Positiva. Este método llamado *random walk* [4] consiste en aumentar una matriz R positiva a la matriz de varianza-covarianza P_k, obteniéndose:

$$P_{k+1} = (I - L_{k+1}\varphi_{k+1}^T)P_k + R$$

Teniendo en cuenta estas modificaciones obtendremos el siguiente algoritmo para el método de los mínimos cuadrados recursivos:

1. Seleccionar los valores de P_k y θ_k.

2. Obtener los nuevos valores de y_{k+1} y u_{k+1}.

3. Calcular el error residual:

$$e_{k+1} = y_{k+1} - \varphi_{k+1}^T\theta_k$$

4. Calcular L_{k+1} dado por la expresión:

$$L_{k+1} = \frac{P_k\varphi_k}{1 + \varphi_k^T P_k\varphi_k}$$

5. Calcular los nuevos parámetros estimados dados por:

$$\theta_{k+1} = \theta_k + L_{k+1}e_{k+1}$$

6. Actualizar la matriz de covarianza:

$$W_{k+1} = (I - L_{k+1}\varphi_{k+1}^T)P_k + R$$

$$P_{k+1} = \frac{W_{k+1}}{c}$$

7. Actualizar el vector de medidas φ_{k+1}

3.5. Identificación de sistemas utilizando RLS

En esta sección se va a desarrollar la identificación de dos sistemas, uno lineal y otro no lineal, alimentados por una entrada aleatoria de media cero y varianza constante. Los modelos obtenidos serán validados a través de diferentes señales (escalón unitario, senoidal y cuadrada). El programa para desarrollar esta prueba fue realizado en *Matlab* y el código se encuentra en el *Apéndice A* con el nombre de *lqs_mejorado_V.m*. El sistema lineal es una planta de segundo orden cuya ecuación en diferencias viene dada por:

$$\begin{aligned} y_{k+1} &= A_1 y_k + A_2 y_{k-1} + A_3 y_{k-2} + \dots \\ &\quad + B_1 u_k + B_2 u_{k-1} + B_3 u_{k-2} \end{aligned} \tag{3.44}$$

donde:

$A_1 = 2{,}6277, A_2 = -2{,}3332, A_3 = 0{,}6976, B_1 = 0{,}0172,$
$B_2 = 0{,}0308, B_3 = 0{,}0140$

y nuestro modelo aproximado es el polinomio ARMAX con $n = 2$, $m = 2$, $p = 0$, $d = 0$:

$$\begin{aligned} y(k+1) &= -a_0 y(k) - a_1 y(k-1) - a_2 y(k-2) + \dots \\ &\quad + b_0 u(k) + b_1 u(k-1) + b_2 u(k-2) + e(k) \end{aligned} \tag{3.45}$$

Para la identificación de los parámetros θ_{k+1} se ha utilizado una señal aleatoria de media cero y varianza constante (el número de muestras para el entrenamiento es $N = 400$).

Los parámetros del algoritmo *RLS* son los siguientes:

Tabla 3.1: Parámetros iniciales del RLS

P=Matriz Diagonal con $p_{ij} = 1000 \; \forall$ i=j; $p_{ij} = 0 \; \forall \; i \neq j$
R=Matriz Positiva con $r_{ij} = p_{ij} \; \forall$ i=j; $r_{ij} = 0 \; \forall \; i \neq j$
Factor de Olvido c= 0,95
Parámetros iniciales $\theta_o^T = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]^T$

Nuestro sistema a estimar será el siguiente:

$$\begin{aligned} \widehat{y}_{k+1} &= a_1\widehat{y}_k + a_2 y_{k-1} + a_3 y_{k-2} + \dots \\ &+ a_4 u_k + a_5 u_{k-1} + a_6 u_{k-2} + \dots \\ &+ a_7 e_k \dots \end{aligned} \tag{3.46}$$

Ahora con esa entrada aleatoria entrenamos al algoritmo RLS, obteniendo el resultado que se aprecia en la figura 3.5, en la cual se muestra la respuesta del sistema real y del sistema aproximado frente a dicha entrada aleatoria.

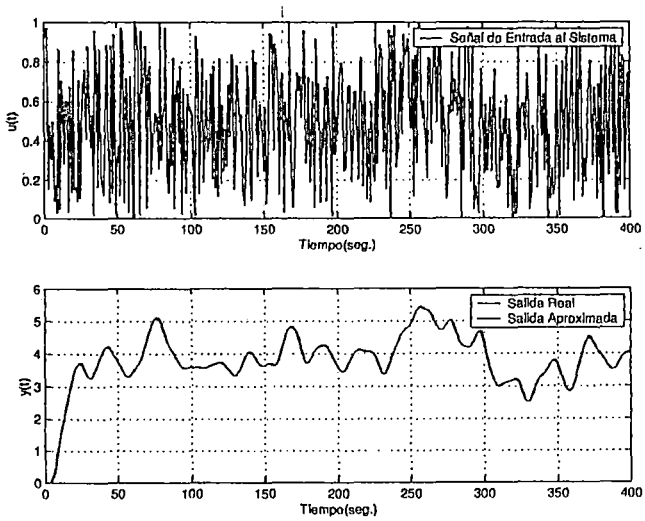

Figura 3.5: Respuesta de un sistema lineal de segundo orden con una entrada aleatoria.

Luego de haber encontrado los parámetros θ realizaremos el análisis de la

correlación cruzada (Capítulo II) para ver si el sistema hallado tiene una estructura válida o no.

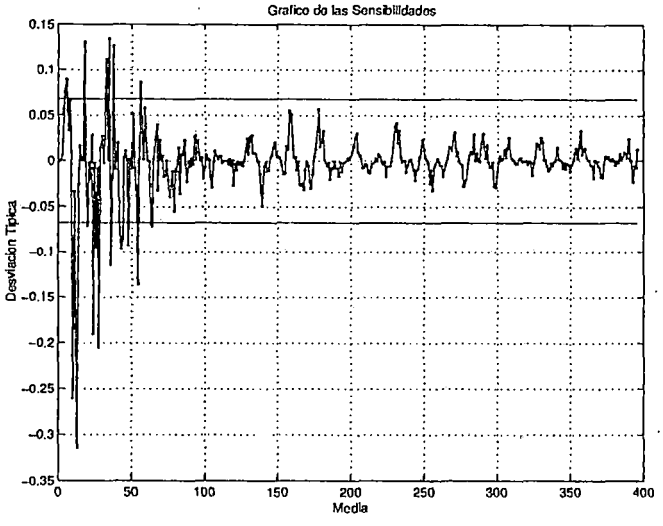

Figura 3.6: Análisis de la correlación cruzada para el modelo propuesto.

Como se observa de la figura 3.6, el número de puntos que se salen de los límites es menor que el 5 % del total de muestras, por lo tanto la estructura que se tomó inicialmente se da como correcta. Ahora toca validar el modelo ARMAX encontrado, sometiendo tanto al modelo aproximado como al modelo a tres tipos de entradas: escalón unitario, senoidal, cuadrada. En las figuras 3.7, 3.8, 3.9, se muestra los resultados de la respuesta tanto del sistema real como del sistema aproximado frente a los diferentes tipos de entrada.

Como se observa en las figuras 3.7, 3.8 y 3.9, el sistema aproximado cumple con seguir al sistema real con gran exactitud, es decir el modelo aproximado y_N ha aprendido la dinámica del sistema.

La tabla 3.2 contiene los errores de aproximación frente a las diferentes entradas a que son sometidos los dos sistemas.

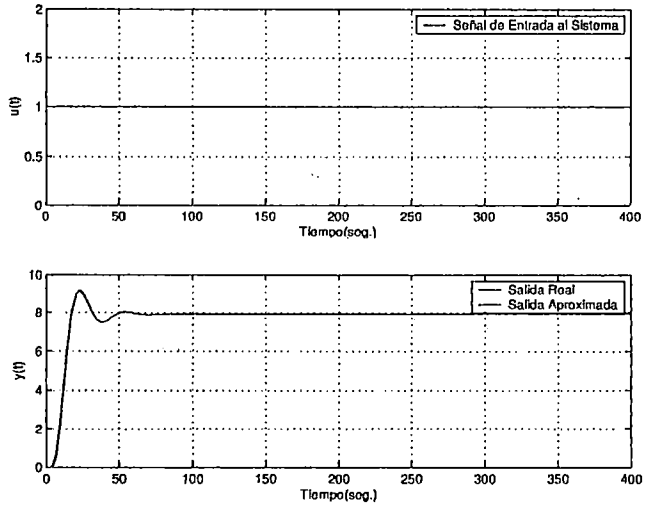

Figura 3.7: Validación del modelo ARMAX aproximado para la planta lineal de segundo orden con una entrada escalón unitario, utilizando *RLS* (N=400).

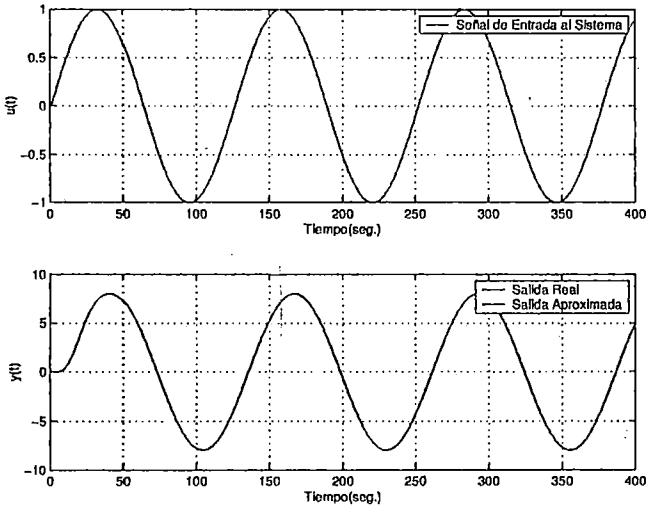

Figura 3.8: Validación del modelo ARMAX aproximado para la planta lineal de segundo orden con una entrada senoidal utilizando *RLS* (N=400).

Ahora vamos a probar el algoritmo del RLS para una planta no lineal. El sistema a estimar será la planta de Narendra y Parthasarathy [8] cuya ecuación en diferencias es:

$$y_{k+1} = \frac{y_k}{(1+y_k)^2} + u_k^3 \tag{3.47}$$

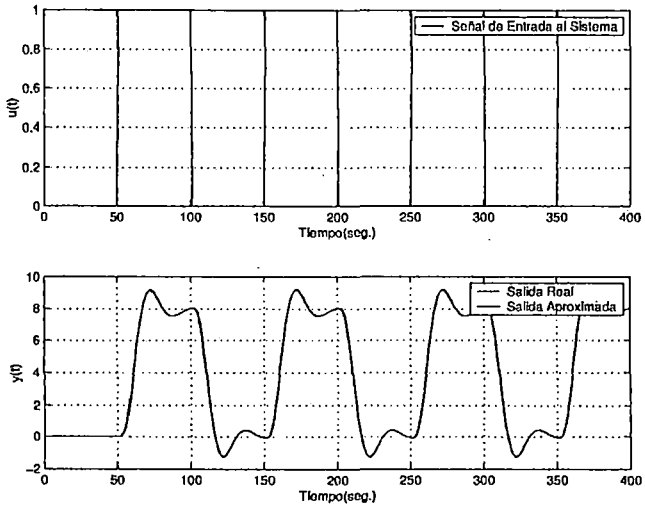

Figura 3.9: Validación del modelo ARMAX aproximado para la planta lineal de segundo orden con una entrada cuadrada utilizando *RLS* (N=400).

Tabla 3.2: Errores del algoritmo RLS frente a diferentes tipos de entradas (sistema lineal)

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N})\cdot 100\%$
Escalón	0.94 %
Senoidal	10.61 %
Cuadrada	17.03 %

El modelo es el mismo que el utilizado para el sistema lineal y además se partirá con las mismas condiciones iniciales.

Al igual que el sistema lineal, se realizará la identificación con 400 muestras de una señal aleatoria (con media=0 y varianza constante) y validará el modelo hallado con las señales escalón unitario, senoidal y cuadrada.

Luego de haber entrenado el modelo aproximado $\widehat{y}_N$ con la señal aleatoria, los resultados los graficamos en la figura 3.10.

Realizando también el análisis de la correlación cruzada se tiene la figura 3.11.

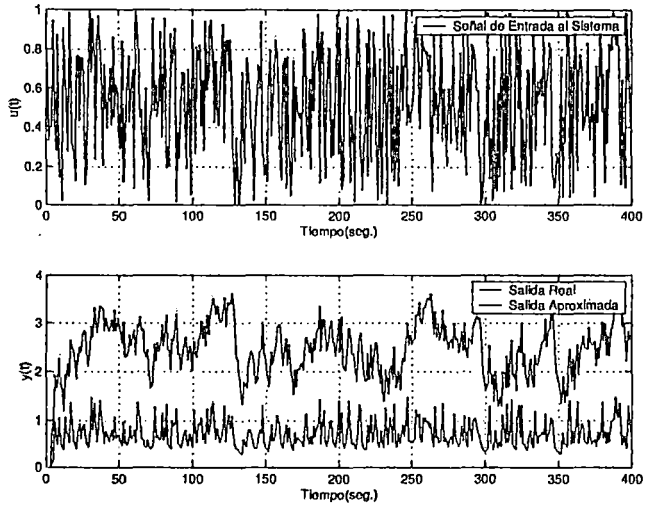

Figura 3.10: Entrenamiento del sistema para la planta No Lineal Narendra-Parthasarathy con una entrada aleatoria utilizando *RLS* (N=400).

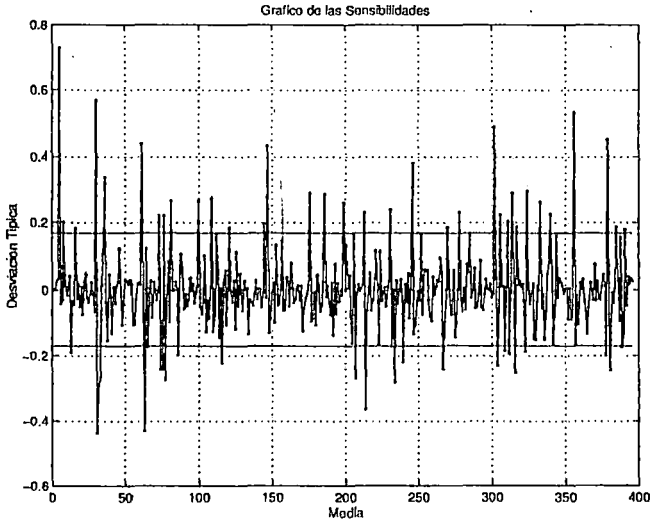

Figura 3.11: Análisis de la correlación cruzada para el modelo propuesto.

En la figura 3.11 se observa que más del 5 % de los puntos de la muestra sobresalen los límites, por lo tanto la estructura propuesta no es confiable.

Ahora después de haber encontrado los parámetros θ vamos validar el modelo ARMAX encontrado, sometiendo tanto al modelo aproximado como al modelo

a tres tipos de entradas: escalón unitario, senoidal, cuadrada. Los resultados de las respuestas, tanto del sistema real como el aproximado frente a las diferentes señales son mostradas en las figuras 3.12, 3.13, 3.14, respectivamente.

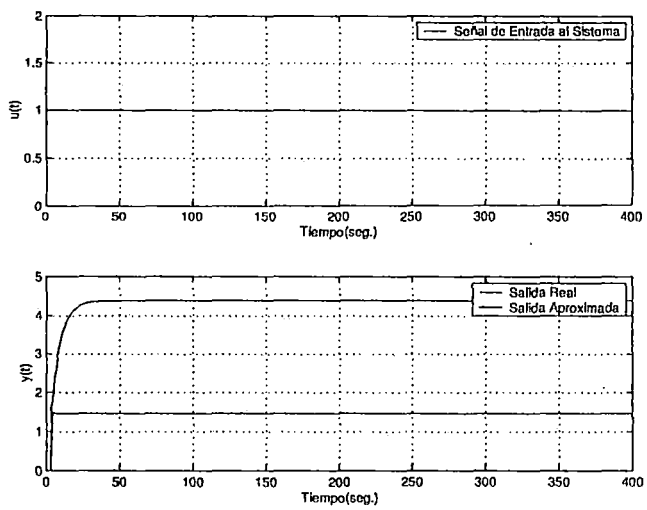

Figura 3.12: Validación del modelo ARMAX aproximado para la planta no lineal de Narendra-Parthasarathy con una entrada escalón unitario utilizando *RLS* (N=400).

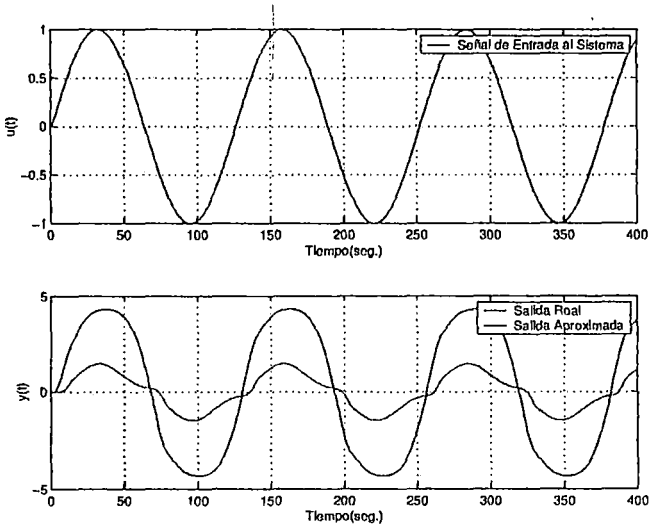

Figura 3.13: Validación del modelo ARMAX aproximado para la planta no lineal de Narendra-Parthasarathy con una entrada senoidal utilizando *RLS* (N=400).

Como se observa en las figuras 3.12,3.13 y 3.14, el sistema aproximado no

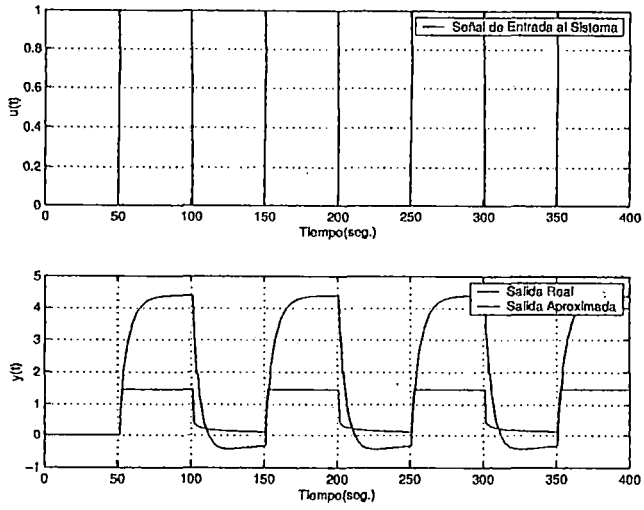

Figura 3.14: Validación del modelo ARMAX aproximado para la planta no lineal de Narendra-Parthasarathy con una entrada cuadrada, utilizando *RLS* (N=400).

cumple con seguir al sistema real (como se aseveró con el análisis de la correlación cruzada la estructura no era la óptima), el error existente entre el modelo real y el modelo aproximado es muy grande para tomar como válido el sistema hallado. Esto quiere decir que el método de RLS no es bueno para aproximar modelos no lineales. En la tabla 3.3 se muestra el error del sistema aproximado en relación a las diferentes entradas.

Tabla 3.3: Errores del algoritmos RLS frente a diferentes tipos de entradas (sistema no lineal)

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N})\cdot 100\,\%$
Escalón	14.40 %
Senoidal	80.51 %
Cuadrada	68.82 %

3.6. Aproximación mediante Modelos No Lineales

Para describir globalmente el comportamiento de los sistemas se debe recurrir a modelos no lineales. La gran variedad de modelos no lineales hace que

no sea posible obtener métodos generales de identificación, sino sólo para determinadas clases de modelos no lineales. Muchos sistemas no lineales pueden ser representados por la interconexión de sistemas lineales estacionarios y no linealidades estáticas[3]. Estos modelos se denominan *orientados a bloques (block-oriented nonlinear models)*. Entre los modelos orientados a bloques los que han sido más estudiados son los *Modelos Hammerstein* y los *Modelos Wiener*.

3.6.1. Modelo Hammerstein

Este modelo consta de una parte no lineal conectada en serie a un modelo lineal. En la figura 3.15 se puede observar la estructura de este modelo.

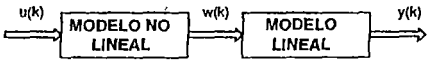

Figura 3.15: Modelo de bloques Hammerstein.

Sea $f(\cdot)$ una función no lineal que representa al modelo no lineal y $g(\cdot)$ una función lineal que representa al modelo lineal. Como podemos ver de la figura 3.15, *w(k)* es la señal intermedia que se obtiene mapeando la señal *u(k)* a través de la función $f(\cdot)$. Entonces tenemos lo siguiente:

$$w(k) = f(u(k)) \tag{3.48}$$

$$w(k-q) = f(u(k-q)) \tag{3.49}$$

Ahora, supongamos que el bloque lineal es un modelo ARX. Entonces la relación entre la entrada y salida para el segundo bloque de acuerdo a la ecuación (3.16) y según la ecuación (3.49) es:

[3]Las no linealidades estáticas aparecen por ejemplo debido a saturación de actuadores, sensores con características no lineales, etc.

$$\begin{aligned} y(k) &= -a_1 y(k-1) - \ldots - a_n y(k-n) + \ldots \\ &\quad + b_1 w(k-d-1) + \ldots + b_m w(k-d-m) \end{aligned} \tag{3.50}$$

Reescribiendo la ecuación (3.50) en forma de sumatorias se tiene:

$$y(k) = \sum_{p=1}^{n_y} \theta_p y(k-p) + \sum_{q=1}^{n_w} \sigma_q w(k-q) \tag{3.51}$$

Reemplazando la ecuación (3.49) en la ecuación (3.51) se tiene lo siguiente:

$$y(k) = \sum_{p=1}^{n_y} \theta_p y(k-p) + \sum_{q=1}^{n_w} \sigma_q f(u(k-q)) \tag{3.52}$$

La ecuación (3.52) muestra que el modelo Hammerstein es un caso particular del polinomio NARX[4], donde la no linealidades son expresadas en función de la señal u.

3.6.2. Modelo Wiener

Este modelo consta de una parte lineal conectada en serie a un modelo no lineal. En la figura 3.16 se puede observar la estructura de este modelo.

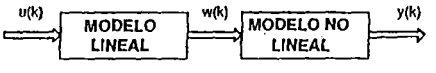

Figura 3.16: Modelo de bloques Wiener.

De acuerdo a la figura 3.16 se puede observar que la señal de salida *y(k)* se obtiene mapeando la señal intermedia *w(k)* a través de la función no lineal $f(\cdot)$:

$$y(k) = f(w(k)) \tag{3.53}$$

[4]NARX: NolinearAutoRegressive with eXtra input

También se observa que la señal intermedia $w(k)$ puede ser obtenida a través de la estimación de la inversa de $f(\cdot)$[5]:

$$w(k) = f^{-1}(y(k)) \tag{3.54}$$

De acuerdo al procedimiento anterior, tomamos el modelo ARX como referencia y obtenemos las siguiente ecuación:

$$w(k) = \sum_{p=1}^{n_w} \theta_p w(k-p) + \sum_{q=1}^{n_u} \sigma_q \, g(u(k-q)) \tag{3.55}$$

Reemplazando la ecuación (3.53) en la ecuación (3.55) se tiene:

$$y(k) = f^{-1}(\sum_{p=1}^{n_w} \theta_p w(k-p) + \sum_{q=1}^{n_u} \sigma_q \, g(u(k-q))) \tag{3.56}$$

Una gran dificultad para obtener las representaciones Hammerstein y Wiener es la de estimar las funciones $f(\cdot)$ y $g(\cdot)$ a paritr de los datos de entrada y salida sin tener disponible la señal intermedia $w(k)$.

3.6.3. Modelo Polinomial NARMAX

Basándonos en las ecuaciones obtenidas para los modelos ARMAX (ecuaciones (3.22), (3.23) y)3.24), vamos a mostrar la estrutura no lineal NARMAX (Non-linear Auto-Regressive with Moving Average and eXogenous inputs).

Las propiedades generales de este modelo hace que se pueda representar una gran variedad de sistemas no lineales. Un modelo NARMAX se puede representar de la siguiente forma:

[5]Para que una función $f : A \longrightarrow B$ admita inversa f debe ser biyectiva es decir $f^{-1} : B \longrightarrow A$

$$y(k) = F^l(y(k-1),\ldots,y(k-n_y),u(k-1),\ldots,u(k-n_u),\ldots \\ v(k-1),\ldots,e(k-n_v)) + e(t) \tag{3.57}$$

donde:

F^l: es una función no lineal cualquiera.
n_y: es el máximo retardo de la señal de salida.
n_u: es el máximo retardo de la señal de entrada.
n_v: es el máximo retardo de la señal de error.

Generalmente la función F^l no es conocida de antemano. La dinámica del sistema puede ser reconstruida utilizándose una representación para aproximar F^l. Algunas aproximaciones para esta función son modelos polinomiales racionales [2], [3]. Así la función F puede ser representada del siguiente modo:

$$y(k) = \theta_0 + \sum_{i_1=1}^{n} \theta_{i_1} x_{i_1}(k) + \sum_{i_1=1}^{n}\sum_{i_2=i_1}^{n} \theta_{i_1}\theta_{i_2} x_{i_1}(k)x_{i_2}(k) + \ldots \\ \sum_{i_1=1}^{n}\sum_{i_l=i_{l-1}}^{n} \theta_{i_i}\ldots\theta_{il} x_{i_1}\ldots x_{il} + e(k) \tag{3.58}$$

donde:

$$\begin{aligned} x_1(k) &= y(k-1) \\ x_2(k) &= y(k-2) \\ &\vdots \quad\quad \vdots \\ x_{n_y+1}(k) &= u(k-1) \\ x_{n_y+2}(k) &= u(k-2) \\ &\vdots \quad\quad \vdots \\ x_{n_y+n_u+1}(k) &= e(k-1) \\ x_{n_y+n_u+2}(k) &= e(k-2) \end{aligned}$$

$$\begin{aligned} \vdots \quad & \quad \vdots \\ x_n(k) &= e(k-n_e) \\ n &= n_y + n_u + n_e \end{aligned}$$

Por ejemplo si se desea modelar con los siguientes valores $n_y = 3$, $n_u = 3$, $n_e = 0$ y $l = 2$ entonces el número de términos que se generaría de la expansión polinomial sería:

$$\binom{3+3+0+2}{4} = \frac{8!}{4!} = 70 \text{ términos}$$

Como se observa, si aumentamos en una unidad los retrasos máximos, tanto de la entrada como de la salida, entonces el número de términos que se generarían de la expansión polinomial crecería factorialmente, por ejemplo si hacemos $n_y = 4$, $n_u = 4$, $n_e = 0$ y $l = 2$, entonces se tendría:

$$\binom{4+4+0+2}{4} = \frac{10!}{4!} = 210 \text{ términos}$$

Donde se observa que resulta engorroso estar expandiendo esa cantidad de términos (además que computacionalmente se consumiría demasiados recursos de memoria en hacer los cálculos respectivos). Aquí resulta crucial la etapa de optimización estructural que se propuso en el Capítulo II, la selección de los términos adecuados que permita representan de una forma precisa la salida de un sistema, pero sin caer en el sobredimencionamiento de la estructura del aproximador funcional.

Por ejemplo, supongamos que tenemos una secuencia de datos generados por el siguiente sistema no lineal, frente a una entrada senoidal:

$$y_{k+1} = \frac{y_k}{1+y_k^2} + u_k^3$$

El objetivo será encontrar un aproximador funcional que represente la dinámica completa del sistema, para ello debemos seleccionar los componentes esenciales para formar nuestra estructura y_{k+1}, para lograr ello nos ayudamos del análisis de la influencia de las variables de entrada.

Primero seleccionamos algunos elemento de la expansión del polinomio NARMAX:

$$\begin{aligned} &y_k, u_k, ek, y_k u_k, y_k e_k, u_k e_k, y_k y_{k-1}, \cdots \\ &u_k u_{k-1}, e_k e_{k-1}, y_{k-1} e_k, u_{k-1} e_{k-1}, \cdots \\ &u_k^2, e_k^2, y_k^2, y_k^2 u_{k-1} \end{aligned} \tag{3.59}$$

Podemos seleccionar más terminos, pero empezaremos con ellos, luego creamos una serie de estructuras con los diferentes términos y los evaluamos con la señal senoidal que afectamos al sistema real. Realizando el análisis de sensibilidades de las varibles de entrada explicadas en el capítulo anterior obtenemos la gráfica que se muestra en la figura 3.17

Donde se observa que las variables más influyentes son las que se encuentran alejadas de $(0,0)$ mientras las que se encuentran cercanas a ese punto son irrelevantes. Se aprecia que los elementos $X_1, X_2, X_3, X_4, X_5, X_6, X_7$ son los más influyentes, estos corresponden a los términos: $y_k, u_k, ek, y_k u_k, y_k e_k, u_k e_k$, con lo cual el aproximador funcional candidato será:

$$y_{k+1} = a_1 y_k + a_2 u_k + a_3 ek + a_4 y_k u_k + a_5 y_k e_k + a_6 u_k e_k$$

Arreglando la función anterior se tiene:

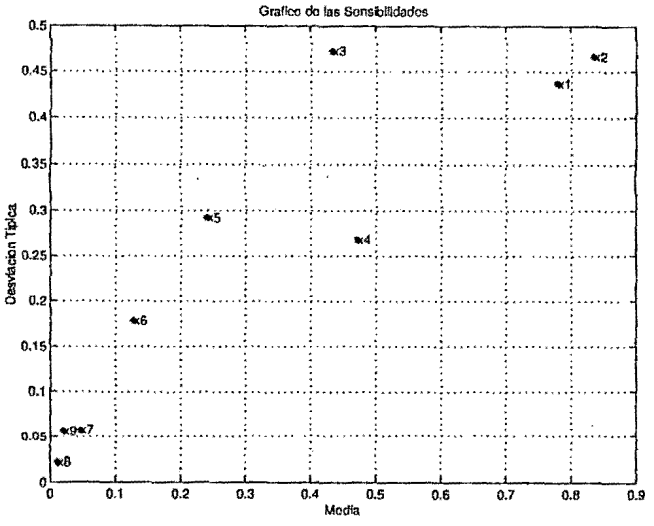

Figura 3.17: Gráfico de la influencia de las variables de entrada.

$$y(k+1) = \phi^T(k)\theta$$

donde:

$$\theta = [\, a_1 \;\; a_2 \;\; a_3 \;\; a_4 \;\; a_5 \;\; a_6 \,]$$
$$\phi = [\, y_k \;\; u_k \;\; ek \;\; y_k u_k \;\; y_k e_k \;\; u_k e_k \,]$$

Los parámetros θ los cálculamos a través del algoritmos RLS. Comparamos la respuesta del sistema aproximado $\widehat{y}_k$ frente al sistema real y_k con la entrada senoidal, obteniendo la gráfica que se muestra en la figura 3.18.

Ahora para validar el modelo vamos a someter a la prueba de la correlación cruzada (ver figura 3.19).

Como se observa en la figura 3.19 menos del 5 % de los puntos sobresalen los límites, por lo tanto la estructura elegida es suficiente. Ahora se procederá a validar el sistema con una entrada cuadrada (ver figura 3.20).

En figura 3.20 se observa que el sistema aproximado sigue al sistema real con una buena precisión, pudiendo afirmar que el sistema encontrado cumple

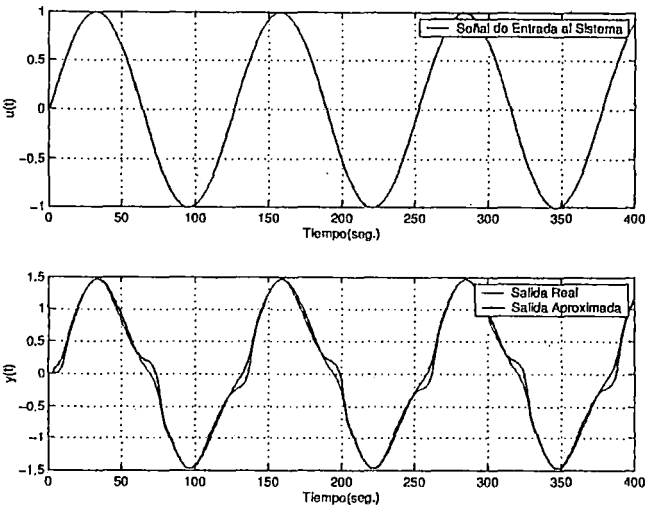

Figura 3.18: Respuesta del sistema real y aproximado frente a una entrada senoidal.

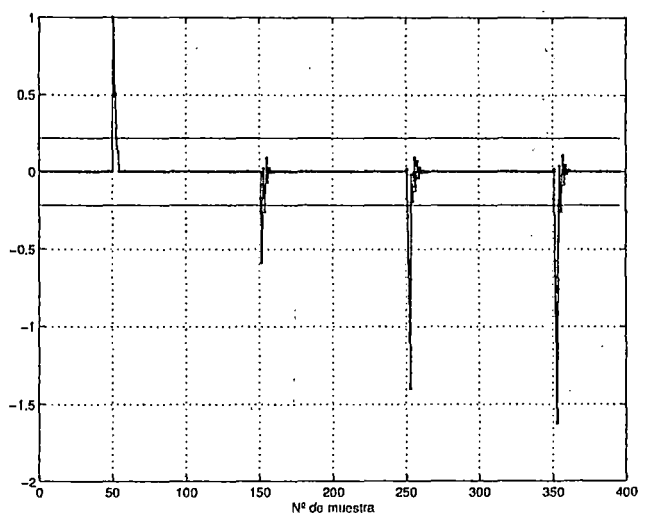

Figura 3.19: Gráfica de la correlación cruzada.

con representar la dinámica completa del sistema. El programa de esta parte se encuentra en el *Apéndice A* con el nombre de *AES_2.m*.

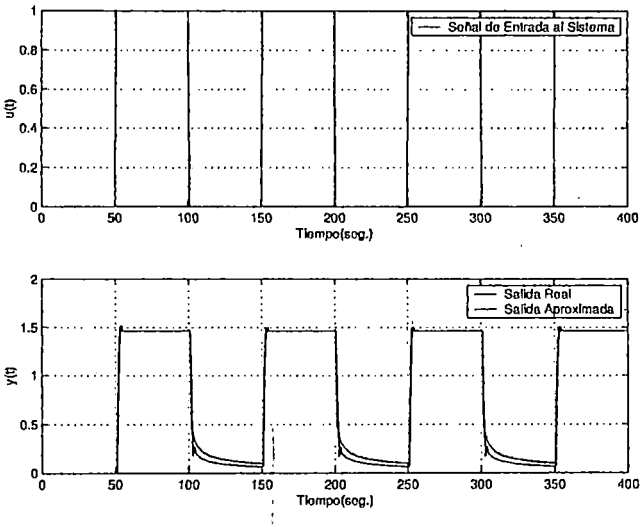

Figura 3.20: Respuesta del sistema real y aproximado frente a una entrada cuadrada.

CAPÍTULO IV
APROXIMACIÓN FUNCIONAL A UN MODELO NARMAX USANDO REDES NEURONALES RECURRENTES

Las redes neuronales son modelos matemáticos que intentan imitar la estructura y funcionamiento del cerebro humano. Una red neuronal está formada por un conjunto de elementos simples interconectados, que es capaz de procesar la información disponible para aprender y así clasificar, predecir, discriminar,etc. Una diferencia esencial entre las redes neuronales y otros modelos matemáticos, que se usan con más frecuencia, reside en el proceso de construcción, puesto que en estos últimos es necesario especificar a priori la función que sigue cualquier proceso que se pretenda modelizar, mientras que una red neuronal desarrolla una aproximación a la relación funcional desconocida que liga unas variables con otras.

En los últimos años, ha habido un creciente interés en aplicar redes neuronales para la identificación (modelamiento) de sistemas dinámicos, predicción y control. La identificación de plantas o sistemas desconocidos, es un tópico extensamente estudiado en la teoría de control clásico. Diversos métodos y algoritmos para la identificación de sistemas han sido estudiados desde 1960, muchos procedimientos han sido propuestos y muy usados en la identificación de sistemas lineales, sin embargo su aplicabilidad para la identificación de sistemas no lineales es muy limitada. Por esta razón y por las ventajas que presentan las redes neuronales para la representación de modelos no lineales, han motivado a muchos investigadores a realizar estudios sobre identificación de sistemas dinámicos por medio de redes neuronales. Esta motivación está incrementada además por la relevancia que tiene la obtención de un modelo que represente la dinámica de un sistema en la implementación de diversos esquemas de control neuronal adaptativo. Se han publicado diversos artículos sobre identificación de sistemas usando redes neuronales [1]. El pionero en la investigación sobre identificación y control

de sistemas por medio de redes neuronales ha sido Kumpati. S. Narendra quien describe en [8], como la dinámica de sistemas no lineales puede ser modelada mediante redes neuronales. Tanto redes neuronales de tipo feedforward como redes recurrentes, se recomiendan para la identificación de sistemas dinámicos.

El desarrollo de redes neuronales para identificación, se ha realizado tradicionalmente sobre arquitecturas clásicas de tipo perceptron multicapa donde la dinámica del sistema se representa por ventanas temporales pasadas, tanto de entradas como de salidas del sistema que son alimentadas a la entrada de la red. En el caso lineal[1] esto equivaldría a un modelo ARX (AutoRegresive with eXogenous input). La identificación de sistemas dinámicos puede realizarse de dos formas, con entrenamiento *off-line* (fuera de línea) o con entrenamiento *on-line* (en línea). Mucha investigación se ha efectuado sobre la identificación de sistemas *off-line* donde se usa un archivo generado con la historia de las entradas y salidas del sistema para entrenar la red, sin embargo para propósitos de control adaptativo de procesos no lineales, se requieren algoritmos de entrenamiento *on-line* para proveer una mejora en la exactitud del sistema modelado y ajustar los parámetros de la red de acuerdo a los cambios que ocurran en el proceso.

En el presente capítulo se explicará el funcionamiento de las redes neuronales recurrentes utilizando el algoritmo BPEA y se aplicará estás redes para la identificación de un sistema lineal y un sistema no lineal mediante el modelo NARMAX.

4.1. Breve Historia

En la tabla 4.1 se describe los inicios y el desarrollo de la redes neuronales[2].

Fue a partir de 1986 en que el panorama para las redes neuronales empezó a cambiar iniciándose una serie de investigaciones. En la actualidad, son numerosos los trabajos que se realizan y publican cada año, las aplicaciones nuevas que surgen (sobretodo en el área de control) y las empresas que lanzan al mercado productos

[1]Función de activación lineal

[2]Facilitada por la Ing. Ana Bollella, e-mail: goica21@adinet.com.uy

Tabla 4.1: Historia de las redes neuronales

1936	**Alan Turing.** Fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas.
1949	**Donald Hebb.** Escribió un importante libro: La Organización del Comportamiento, en el que se establece una conexión entre psicología y fisiología. Fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.
1950	**Karl Lashley.** En sus series de ensayos, encontró que la información no era almacenada en forma centralizada en el cerebro sino que era distribuida encima de él.
1956	**Congreso de Dartmouth.** Este Congreso frecuentemente se menciona para indicar el nacimiento de la inteligencia artificial.
1957	**Frank Rosenblatt.** Comenzó el desarrollo del Perceptrón. Esta es la red neuronal más antigua, utilizándose hoy en día para aplicación como reconocedor de patrones. Este modelo es capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado anteriormente. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función OR-exclusiva y, en general, era incapaz de clasificar clases no separables linealmente. En 1959, escribió el libro Principios de Neurodinámica, en el que confirmó que, bajo ciertas condiciones, el aprendizaje del Perceptrón convergía hacia un estado finito (Teorema de Convergencia del Perceptrón).
1960	**Bernard Widrow/Marcial Hoff.** Desarrollaron el modelo Adaline (ADAptative LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas.
1961	**Karl Steinbeck, Die Lernmatrix.** Red neuronal para simples realizaciones técnicas (memoria asociativa).

Tabla 4.2: Continuacion...

1967	**Stephen Grossberg.** A partir de sus conocimientos fisiológicos, escribió numerosos libros y desarrolló modelos de redes neuronales. Realizó una red: Avalancha, que consistía en elementos discretos con actividad que varía en el tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades como reconocimiento continuo de habla y aprendizaje de los brazos de un robot.
1969	**Marvin Minsky/Seymour Papera.** En este año surgieron críticas que frenaron, hasta 1982, el crecimiento que estaban experimentando las investigaciones sobre redes neuronales. Minsky y Papera, del Instituto Tecnológico de Massachussets (MIT), publicaron un libro Perceptrons. Probaron (matemáticamente) que el Perceptrón no era capaz de resolver problemas relativamente fáciles, tales como el aprendizaje de una función no-lineal. Esto demostró que el Perceptrón era muy débil, dado que las funciones no-lineales son extensamente empleadas en computación y en los problemas del mundo real. A pesar del libro, algunos investigadores continuaron su trabajo. Tal fue el caso de James Anderson, que desarrolló un modelo lineal, llamado Asociador Lineal, que consistía en unos elementos integradores lineales (neuronas) que sumaban sus entradas. Este modelo se basa en el principio de que las conexiones entre neuronas son reforzadas cada vez que son activadas. Anderson diseñó una potente extensión del Asociador Lineal, llamada Brain State in a Box (BSB).
1974	**Paul Werbos.** Desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*), cuyo significado quedó definitivamente aclarado en 1985.
1977	**Stephen Grossberg.** Teoría de Resonancia Adaptada (TRA). La Teoría de Resonancia Adaptada es una arquitectura de red que se diferencia de todas las demás previamente inventadas. La misma simula otras habilidades del cerebro: memoria a largo y corto plazo.
1977	**Teuvo Kohonen.** Ingeniero electrónico de la Universidad de Helsinki, desarrolló un modelo similar al de Anderson, pero independientemente.
1980	**Kunihiko Fukushima.** Desarrolló un modelo neuronal para el reconocimiento de patrones visuales.
1985	**John Hopfield.** Provocó el renacimiento de las redes neuronales con su libro: "Computación neuronal de decisiones en problemas de optimización".
1986	**David Rumelhart/G. Hinton.** Redescubrieron el algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*).

nuevos, tanto hardware como software (sobre todo para simulación) basados en las redes neuronales.

4.2. Fundamentos de las Redes Neuronales

4.2.1. El Modelo Biológico

Se estima que el cerebro humano contiene más de cien mil millones de neuronas, estudios sobre la anatomía del cerebro humano concluyen que hay más de 1000 sinápsis a la entrada y a la salida de cada neurona. Es importante notar que aunque el tiempo de conmutación de una neurona (unos pocos milisegundos) es casi igual (o en algunos casos superior) que en los actuales elementos de las computadoras, ellas tienen una conectividad miles de veces superior que las actuales supercomputadoras. Las neuronas y las conexiones entre ellas (sinápsis) constituyen la clave para el procesado de la información. Algunos elementos que destacan su estructura histológica son mostrados en la figura 4.1:

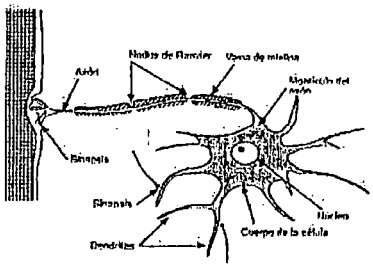

Figura 4.1: Partes principales de una neurona.

Las *dendritas*, que son la vía de entrada de las señales que se combinan en el cuerpo de la neurona. De alguna manera la neurona elabora una señal de salida a partir de ellas. El *axón*, que es el camino de salida de la señal generada por la neurona. Las *sinapsis*, que son las unidades funcionales y estructurales elementales que median entre las interacciones de las neuronas. En las terminaciones de las *sinapsis* se encuentran unas *vesículas* que contienen unas sustancias químicas llamadas *neurotransmisores* que ayudan a la propagación de las señales electroquímicas de una neurona a otra.

Lo que básicamente ocurre en una neurona biológica es lo siguiente: la neurona es estimulada o excitada a través de sus entradas (inputs) y cuando se alcanza un cierto umbral, la neurona se dispara o activa, pasando una señal hacia el axón. La membrana es permeable para ciertas especies iónicas, y actúa de tal forma que se mantenga una diferencia de potencial entre el fluído intracelular y el fluído extracelular. Este efecto se consigue primordialmente mediante la acción de una bomba sodio potasio (ver la figura 4.2).

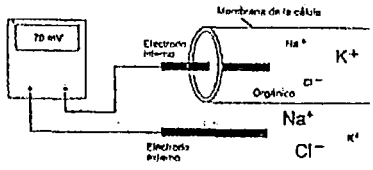

Figura 4.2: Bomba de sodio potasio.

Posteriores investigaciones condujeron al descubrimiento de que estos procesos son el resultado de eventos electroquímicos. Como ya se sabe, el pensamiento tiene lugar en el cerebro, que consta de billones de neuronas interconectadas. Así, el secreto de la "inteligencia" -sin importar como se defina- se sitúa dentro de estas neuronas interconectadas y de la interacción entre ellas. La forma que dos neuronas interactúan no está totalmente conocida, dependiendo además de cada neurona.

En general, una neurona envía su salida a otras por su *axón*. El *axón* lleva la información por medio de diferencias de potencial, u ondas de corriente, que depende del potencial de la neurona. En los siguentes párrafos vamos a describir brevemente como es que se realiza esta interacción.

Como se puede ver en la figura 4.3, el *axón* está cubierto con lo que se denomina *vaina de mielina*. Esta capa es interrumpida en varios puntos por los *nodos de Ranvier*. Las entradas exitatorias que llegan a la célula reducen la diferencia de potencial que existe entre los dos lados de la membrana celular. La

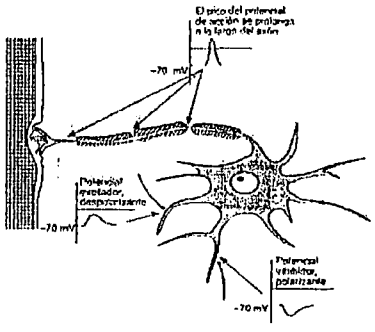

Figura 4.3: Potenciales generados en una neurona.

despolarización resultante en el *montículo del axón* altera la permeabilidad de la membrana celular a efectos de los iones de sodio. Como resultado hay un fuerte flujo entrante de iones positivos de sodio, que penetran en la célula, contribuyendo más aún a la despolarización. Este efecto autogenerado da lugar al *potencial de acción.*

Las fibras nerviosas en sí son malos conductores. La transmición del *potencial de acción* a lo largo de *axón* es el resultado de una serie de despolarizaciones que tienen lugar en los *nodos de Ranvier.* Cuando uno de los nodos se despolariza, se desencadena la despolarización del siguiente modo: el potencial de acción viaja a lo largo de la fibra en forma discontínua, de un nodo a otro. Una vez que el potencial de acción ha pasado por un cierto punto, ese punto no puede volver a ser exitado durante aproximadamente 1 milisegundo, que es el tiempo que tarda en volver a su potencial de reposo. Este periódo refractario limita la frecuencia de transmición a unos 1000 exitaciones por segundo[6].

La comunicación entre dos neuronas tiene lugar como resultado de la liberación de unas sustancias llamadas *neurotransmisores* por parte de la célula presináptica, al ser absorvidas por la célula postsináptica tal como se muestra en la figura 4.4.

Cuando el potencial de acción llega a la membrana presináptica, los cambios de permeabilidad de la membrana dan lugar a un flujo entrante de iones de so-

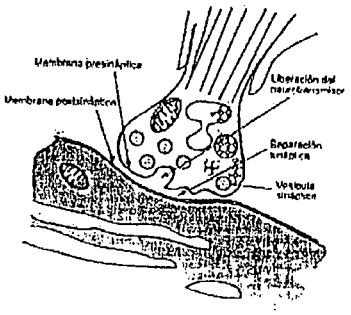

Figura 4.4: Separación sináptica.

dio. Estos iones dan lugar a que las vesículas que contienen los neurotransmisores se fundan con la membrana sináptica, liberando así sus neurotransmisores en la separación sináptica.

Los neurotransmisores se difunden a través de la unión y se unen a la membrana postsináptica en ciertos lugares llamados receptores. La acción química que se produce en los receptores da lugar a cambios de permeabilidad de la membrana postsináptica. Un flujo entrante de iones positivos hacia la célula despolarizará el potencial de reposo, este efecto es exitatorio, por el contrario, si entran iones negativos, se producirá un efecto hiperpolarizante, este efecto es inhibitorio. Estos dos efectos son locales, y actúan tan sólo a lo largo de una pequeña distancia hacia el interior de la neurona. Estos dos efectos (tanto el exitatorio como el inhibitorio) se suman, y si el valor de la suma es mayor que cierto valor umbral entonces se genera un *potencial de acción.*

4.2.2. Las Redes Neuronales Artificiales

Las RNA están inspiradas en la estructura del cerebro y fueron concebidas para resolver cierto tipo de problemas especialmente mal resueltos por las técnicas de programación tradicionales. Formalmente, la computación neuronal es la disciplina tecnológica que trata sistemas paralelos y adaptativos de procesamiento de información distribuida, que desarrollan sus capacidades bajo su exposición a un entorno de información.

Tal como se mostró en la sección anterior, esta definición muestra claramente el paralelismo entre la estructura cerebral biológica y las RNA. Este paralelismo ha motivado a muchos investigadores de diversos campos de la ciencia a profundizar en la interpretación biológica de las RNA, proponiendo nuevas estructuras conexionistas y estrategias de aprendizaje directamente inspiradas de la modelización del cerebro. Estos estudios han dado resultados interesantes en el campo de las RNA, pero desde un punto de vista subjetivo, lo más importante de ellos es el conocimiento que directa ó indirectamente están aportando para esclarecer los complejos procesos de aprendizaje, percepción y gestión del conocimiento que tienen lugar en el cerebro humano.

Con lo expuesto anteriormente podemos afirmar que las RNA son estructuras adaptativas de procesamiento de información inspiradas en la estructura cerebral, donde el procesamiento se lleva a cabo mediante la interconexión de elementos de proceso muy sencillos a lo que llamaremos neuronas. Esta arquitectura da lugar a estructuras altamente paralelizables donde el flujo de información no sigue un camino secuencial, sino que se distribuye a través de las conexiones de los elementos de proceso donde la información es tratada. Durante la fase de aprendizaje, la RNA se expone a un entorno de información para que pueda adaptar sus pesos (parámetros libres que dan forma a las funciones de transferencia de sus elementos de proceso) y posiblemente también su estructura. Durante la fase de evaluación, los pesos de la red se mantienen fijos y la RNA se limita a tratar la información de entrada que se le suministra.

A continuación mostraremos las estructuras de redes neuronales que a mi parecer son las más relevantes, primero empezaremos mostrando la estructura del Adaline, Madaline, luego mostraremos el Perceptron Multicapa, para finalmente mostrar la estructura que más nos interesa, el Perceptron Multicapa Recurrente.

El Adaline y Madaline

Creadores: B. Widrow y M.E. Hoff
Fecha: 1960

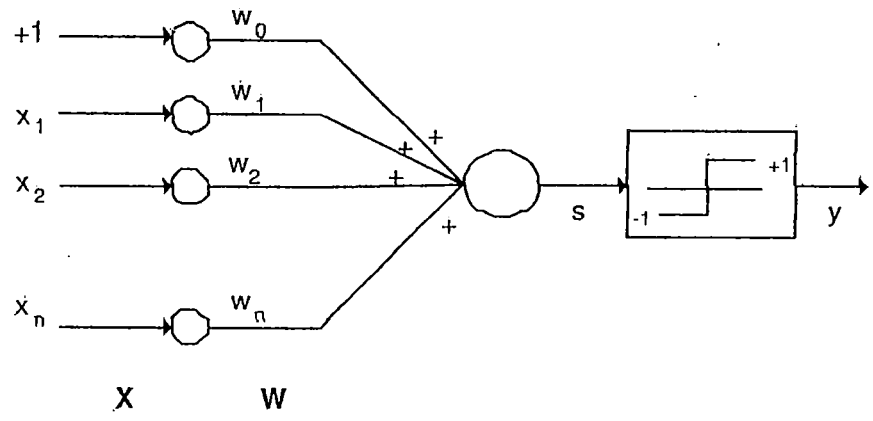

Figura 4.5: Estructura de una red Adaline.

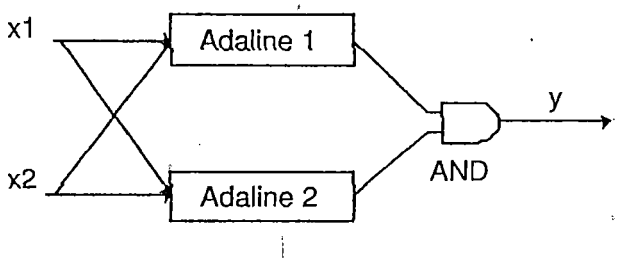

Figura 4.6: Estructura de una red Madaline.

Aplicaciones:

- Filtrado adaptativo de señales.
- Ecualización adaptativa.
- Reconocimiento de patrones.

Ventajas:

- Su gran sencillez y homogeneidad les hacen fácilmente realizables en tecnología VLSI.

Desventajas:

- Sólo son capaces de resolver problemas de clasificación linealmente separables y llevar a cabo transformaciones lineales.

Perceptron Multicapa

Creadores: P.J. Werbos D. Parker D. Rumelhart

Fecha: 1974-1986

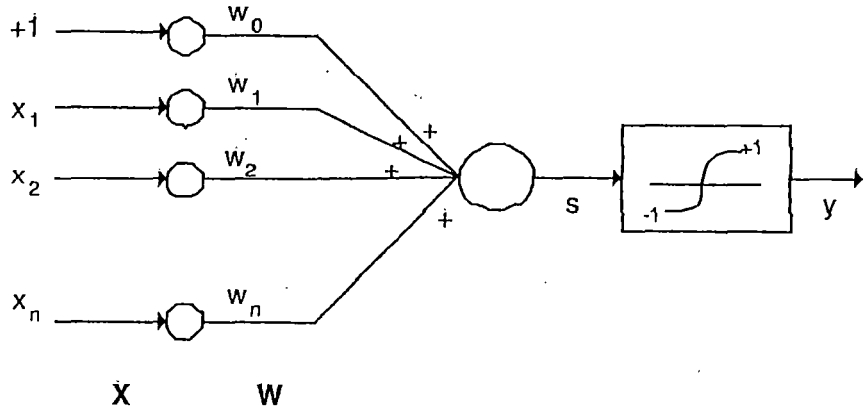

Figura 4.7: Estructura de una red Perceptron.

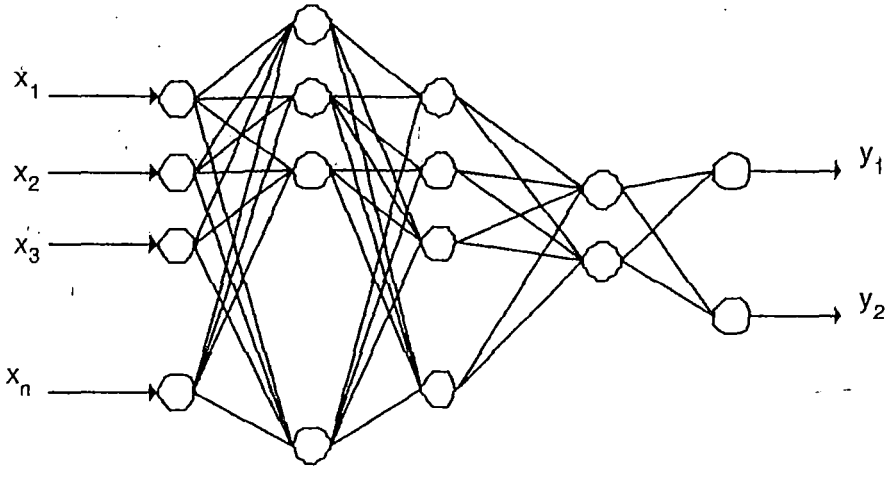

Figura 4.8: Estructura de una red Perceptron multicapa.

Aplicaciones:

- Aproximación funcional.
- Reconocimiento de patrones.
- Filtrado de señales.

- Eliminación de ruido.
- Segmentación de imágenes y señales.
- Control adaptativo.
- Compresión de datos.

Ventajas:

- Capacidad de representación funcional universal.
- Gran rapidez de procesamiento.
- Genera buenas representaciones internas de las características de los datos de entrada. Ampliamente estudiada.
- Es la estructura conexionista que más se ha aplicado en la práctica.

Desventajas:

- Tiempo de aprendizaje elevado para estructuras complejas.

Perceptron Multicapa Recurrente

Creadores: Almeida Pineda

Fecha: 1987

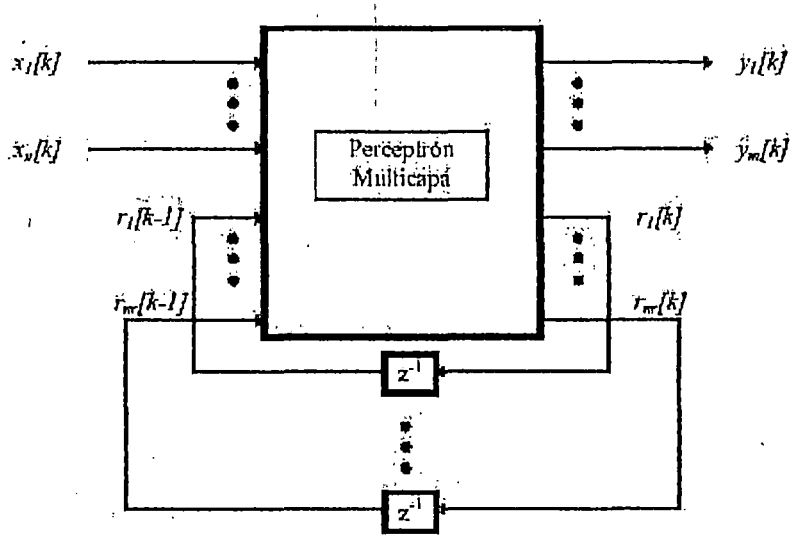

Figura 4.9: Estructura de una red perceptron multicapa recurrente.

Esta estructura recurrente permite estimar los valores de las salidas y en el

instante k, a partir de los valores de las entradas externas x en el mismo instante k, y de los valores de lo que podríamos llamar variables de estado internas r en el instante $(k-1)$, entre las que podría haberse incluído algunas de las variables de salida.

Aplicaciones:

- Control.
- Reconocimiento del habla.
- Predicción de secuencias.

Ventajas:

- Capaz de tratar información temporal.

Desventajas:

- Estructuras muy complicadas.
- El aprendizaje puede resultar muy difícil.

4.3. El Perceptron Multicapa *MLP*

En esta sección describiremos el perceptron multicapa, pero llevándolo al campo de la identificación de sistemas no lineales. Como se puede observar en la figura 4.8, el MLP está compuesto de varias capas de simple perceptrons. En la figura 4.7 la función de activación viene a ser la función signo, pero las más utilizadas son la función de ganancia unitaria, la tangente hiperbólica, la función gausiana y la función sigmoidea.

$$\phi_{lineal}(x) = x \tag{4.1}$$

$$\phi_{tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.2}$$

$$\phi_{gaus}(x) = e^{-x^2} \tag{4.3}$$

$$\phi_{sigm}(x) = \frac{1}{1 + e^{-x}} \tag{4.4}$$

Las respuestas de esas cuatro funciones son mostradas en la figura 4.10. Otras funciones de activación han sido usadas como por ejemplo la función signo y la función de saturación, pero ellas son esencialmente simples variaciones de

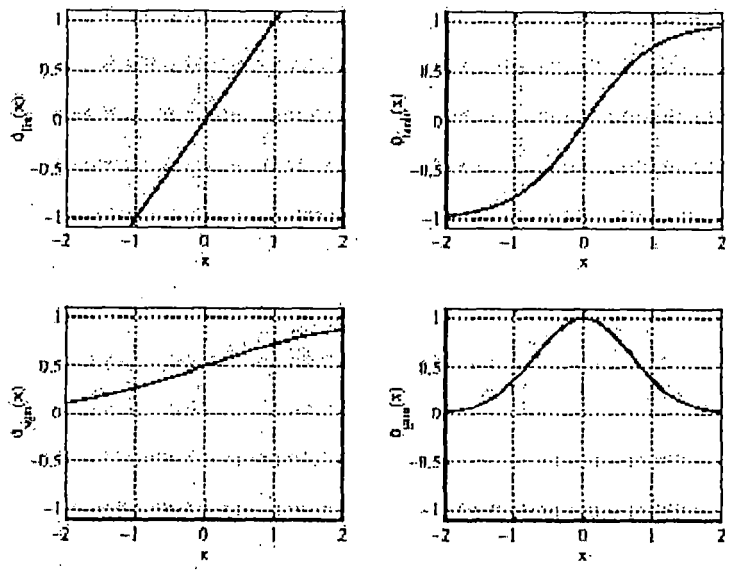

Figura 4.10: Respuesta de las funciones de activación. Superior izquierda: lineal, superior derecha: hiperbólica, inferior izquierda: sigmoidea, inferior derecha: gaussiana

las cuatro funciones mencionadas. Por ejemplo, la función $\tanh \theta(x) \approx sign(\theta(x))$ para valores grandes de θ.

La función tanh y la función sigmoidea tienen un comportamiento lineal para valores cercanos a cero y muestran características saturadas para valores alejadas a cero, esto es muy atractivo para la identificación de sistemas, debido a que en general, los sistemas tienen un comportamiento lineal en un punto de operación determinado y debido a los efectos de la fricción, saturación de los actuadores, es decir cuando el sistema se sale de su punto de operación estos muestran características saturadas.

Una forma más compacta de representar el MLP es a través del diagrama de bloque de matrices, tal como se muestra en la figura 4.11, donde la matriz de pesos es representada por Θ, la bias por el vector θ_b, la función de activación por $\phi(\cdot)$ y los vectores de entrada y salida por Z_{in} y Z_{out}.

La salida del MLP puede ser escrito de la siguiente manera:

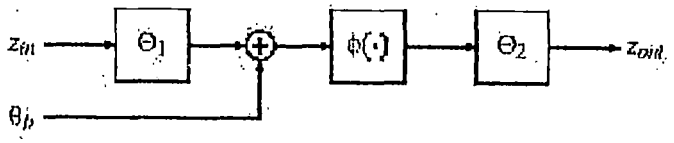

Figura 4.11: Diagrama de bloques de matrices de un MLP

$$z_{out} = \Theta_2 \phi(\Theta_1 z_{in} + \theta_b) \tag{4.5}$$

Todas las señales que ingresa al MLP son agrupadas en el vector Z_{in}, mientras Z_{out} agrupa la salida de la red.

4.3.1. Función de costo

La salida estimada es restada de la salida real del sistema, para calcular el error de predicción. Basados en este error de predicción los pesos en el MLP serán actualizados de acuerdo a los algoritmos conocidos en la literatura de las redes neuronales como *reglas de aprendizaje*. El objetivo entonces será el de minimizar el error cuadrático medio.

En esta tesis usaremos las reglas de aprendizaje recursivo: función de costo recursivo tal como se muestra el la siguiente ecuación:

$$J(k) = \sum_{i=1}^{k} \lambda^{k-i} \frac{1}{2} \varepsilon(k)^T \varepsilon(k) \tag{4.6}$$

donde $\varepsilon(k) = y(k) - \hat{y}(k)$, λ es el factor de olvido, y cumple la misma función expuesta en el capítulo 3, el de que la red "olvide" las informaciones más antiguas.

4.3.2. BPEA para redes neuronales recurrentes

La más simple regla de aprendizaje disponible para el perceptron multicapa (MLP) es el Back Propagation Error Algorithm (BPEA). Esta regla de aprendiza-

je es ampliamente usada debido a su simplicidad y a su robustez. La actualización de los pesos de la red se va a llevar a cabo según la ecuación (4.7):

$$\theta_{i+1} = \theta_i - \eta \frac{\partial J}{\partial \theta_i} \tag{4.7}$$

donde $\eta > 0$ es la longitud de paso de aprendizaje.

El algoritmo consiste de tres pasos: Primero estimamos la salida a través de la ecuación (4.5) (ver figura 4.12).

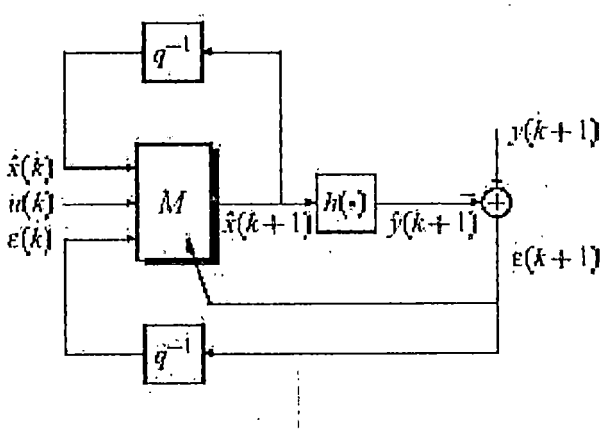

Figura 4.12: Modelo de una red neuronal MLP recurrente.

De acuerdo a la figura 4.12 la salida aproximada del sistema viene a estar dada por:

$$\widehat{y}(k+1) = h(\widehat{x}(k+1)) \tag{4.8}$$

$$\widehat{y}(k+1) = H\Theta_2\phi(\Theta_1 z_{in} + \theta_b) \tag{4.9}$$

Luego el error de predicción $\varepsilon(k+1) = y(k+1) - \widehat{y}(k+1)$, nos va a servir para calcular la derivada de la función de costo con respecto a los pesos actuales

y por último, este gradiente nos será útil para actualizar los pesos.

De la ecuación (4.6) se obtiene lo siguiente:

$$\begin{aligned} J(k) &= \sum_{i=1}^{k} \lambda^{k-i} \frac{1}{2} \varepsilon(k)^T \varepsilon(k) \\ &= \lambda J(k-1) + \frac{1}{2} \varepsilon(k)^T \varepsilon(k) \end{aligned} \tag{4.10}$$

La función de costo es vista como la suma de la perfomance de las muestras actuales más un decremento exponencial de las performance anteriores. Entonces derivando la ecuación (4.10) se obtiene:

$$\begin{aligned} G_j(k) &= \frac{\partial J(k)}{\partial \theta} \\ &= \frac{\partial \lambda J(k-1)}{\partial \theta} + \frac{\frac{1}{2} \varepsilon(k)^T \varepsilon(k)}{\partial \theta} \\ &= \lambda G_j(k-1) - \frac{\partial \hat{y}(k)^T}{\partial \theta} \varepsilon(k) \\ &= \lambda G_j(k-1) - \psi(k) \varepsilon(k) \end{aligned} \tag{4.11}$$

La matriz $\psi(k)$ es llamada gradiente del modelo. De acuerdo a la figura 4.12 se tiene lo siguiente:

$$\hat{y}(k, \theta) = HM(z_{in}, \theta) = HM(\hat{x}(k-1, \theta), \varepsilon(k-1, \theta), \theta) \tag{4.12}$$

El gradiente del modelo $\psi(t)$ puede ser calculado derivando la ecuación anterior respecto a θ y usando la regla de la cadena se tiene que:

$$\frac{\partial \hat{y}(k)^T}{\partial \theta} = \frac{\partial \hat{x}(k)^T}{\partial \theta} H^T = \psi_x(k) H^T$$

donde $\psi_x(k) = (\partial \hat{x}(k)^T / \partial \theta) H^T$. Para este nuevo gradiente se obtiene lo siguiente:

$$\begin{aligned}
\psi_x(k) &= \frac{\partial \hat{x}(k)^T}{\partial \theta} + \frac{\partial \hat{x}(k-1)^T}{\partial \theta} \frac{\partial \hat{x}(k)^T}{\partial \hat{x}(k-1)} + \frac{\partial \varepsilon(k-1)^T}{\partial \theta} \frac{\partial \hat{x}(k)^T}{\partial \varepsilon(k-1)} \\
&= \frac{\partial \hat{x}(k)^T}{\partial \theta} + \frac{\partial \hat{x}(k-1)^T}{\partial \theta} \frac{\partial \hat{x}(k)^T}{\partial \hat{x}(k-1)} - \frac{\partial \hat{x}(k-1)^T}{\partial \theta} H^T \frac{\partial \hat{x}(k)^T}{\partial \varepsilon(k-1)} \\
&= \frac{\partial \hat{x}(k)^T}{\partial \theta} + \psi_x(k-1) \left(\frac{\partial \hat{x}(k)^T}{\partial \hat{x}(k-1)} - H^T \frac{\partial \hat{x}(k)^T}{\partial \varepsilon(k-1)} \right) \qquad (4.13)
\end{aligned}$$

Ahora se tiene que obtener la derivada de la salida de la red respecto a cada uno de los pesos. Para poder calcularlos nos vamos ayudar de la figura 4.13.

Donde x_i son las diferentes entradas, n_e es el número de entradas, n_o es el número de capas ocultas. De la figura 4.13 se observa que:

$$\begin{aligned}
s_1 &= x_1 p_{11} + x_2 p_{21} + \ldots + x_{ne} p_{ne1} \\
s_2 &= x_1 p_{12} + x_2 p_{22} + \ldots + x_{ne} p_{ne2} \\
&\vdots \qquad\qquad \vdots \\
s_{no} &= x_1 p_{1no} + x_2 p_{2no} + \ldots + x_{ne} p_{neno}
\end{aligned}$$

$$S = \Theta_1^T X^T \qquad (4.14)$$

$$\Phi = \Phi(S) \qquad (4.15)$$

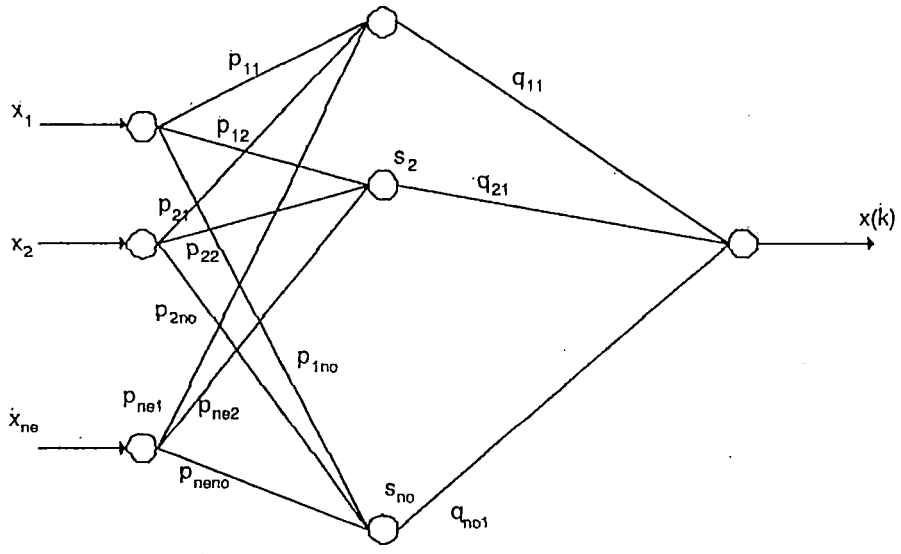

Figura 4.13: Red Neuronal MLP.

$$\hat{x}(k+1) = \phi(s_1)q_{11} + \phi(s_2)q_{21} + \ldots + \phi(s_{no})q_{no1} \tag{4.16}$$

donde:

$$\Theta_1 = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1ne} \\ p_{21} & p_{22} & \cdots & p_{2ne} \\ \vdots & \vdots & \ddots & \vdots \\ p_{nc1} & p_{nc2} & \cdots & p_{none} \end{pmatrix}$$

$$\mathbf{X} = \begin{pmatrix} x_1 & x_2 & \ldots & x_{1ne} \end{pmatrix}^T$$

$$\mathbf{S} = \begin{pmatrix} s_1 & s_2 & \ldots & x_{1no} \end{pmatrix}^T$$

$$\mathbf{\Phi} = \begin{pmatrix} \phi(s_1) & \phi(s_2) & \ldots & \phi(x_{1no}) \end{pmatrix}^T$$

F es la matriz de la función de activación de S. Ahora si se deriva la ecuación (4.16) con respecto a los pesos de salida q_{i1} se obtiene lo siguiente:

$$\frac{\partial \hat{x}(k+1)}{\partial q_{11}} = \phi(s_1)$$

$$\frac{\partial \hat{x}(k+1)}{\partial q_{21}} = \phi(s_2)$$

$$\vdots \qquad \vdots$$

$$\frac{\partial \hat{x}(k+1)}{\partial q_{no1}} = \phi(s_{no})$$

$$\frac{\partial \hat{x}(k+1)}{\partial \Theta_2} = \Phi \tag{4.17}$$

El vector anterior es modificado para mostrarlo como una matriz y será nombrada D_1.

$$\mathbf{D_1} = \begin{pmatrix} \phi(s_1) & 0 & \dots & 0 \\ 0 & \phi(s_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \phi(s_(no)) \end{pmatrix} \tag{4.18}$$

Ahora si derivando la ecuación (4.16) con respecto a cada uno de los pesos de entrada Θ_1 tenemos lo siguiente:

$$\frac{\partial \hat{x}(k+1)}{\partial p_{11}} = \frac{\partial(\phi(s_1)q_{11})}{\partial p_{11}} = q_{11}\frac{\partial \phi(s_1)}{\partial p_{11}} = q_{11}\phi'(s_1)x_1$$

$$\frac{\partial \hat{x}(k+1)}{\partial p_{12}} = \frac{\partial(\phi(s_2)q_{21})}{\partial p_{12}} = q_{21}\frac{\partial \phi(s_2)}{\partial p_{12}} = q_{21}\phi'(s_2)x_1$$

$$\vdots \quad \vdots$$

$$\frac{\partial \hat{x}(k+1)}{\partial p_{1no}} = \frac{\partial(\phi(s_{no})q_{no1})}{\partial p_{1no}} = q_{no1}\frac{\partial \phi(s_{no})}{\partial p_{1no}} = q_{no1}\phi'(s_{no})x_1$$

(4.19)

$$\frac{\partial \hat{x}(k+1)}{\partial p_{21}} = \frac{\partial(\phi(s_1)q_{11})}{\partial p_{21}} = q_{11}\frac{\partial \phi(s_1)}{\partial p_{21}} = q_{11}\phi'(s_1)x_2$$

$$\frac{\partial \hat{x}(k+1)}{\partial p_{22}} = \frac{\partial(\phi(s_2)q_{21})}{\partial p_{22}} = q_{21}\frac{\partial \phi(s_2)}{\partial p_{22}} = q_{21}\phi'(s_2)x_2$$

$$\vdots \quad \vdots$$

$$\frac{\partial \hat{x}(k+1)}{\partial p_{2no}} = \frac{\partial(\phi(s_{no})q_{no1})}{\partial p_{2no}} = q_{no1}\frac{\partial \phi(s_{no})}{\partial p_{2no}} = q_{no1}\phi'(s_{no})x_2$$

(4.20)

$$\frac{\partial \hat{x}(k+1)}{\partial p_{ne1}} = \frac{\partial(\phi(s_1)q_{11})}{\partial p_{ne1}} = q_{11}\frac{\partial \phi(s_1)}{\partial p_{ne1}} = q_{11}\phi'(s_1)x_{ne}$$

$$\frac{\partial \hat{x}(k+1)}{\partial p_{ne2}} = \frac{\partial(\phi(s_2)q_{21})}{\partial p_{ne2}} = q_{21}\frac{\partial \phi(s_2)}{\partial p_{ne2}} = q_{21}\phi'(s_2)x_{ne}$$

$$\vdots \quad \vdots$$

$$\frac{\partial \hat{x}(k+1)}{\partial p_{neno}} = \frac{\partial(\phi(s_{no})q_{no1})}{\partial p_{neno}} = q_{no1}\frac{\partial \phi(s_{no})}{\partial p_{neno}} = q_{no1}\phi'(s_{no})x_{ne}$$

(4.21)

Acomodando los términos hallados en una matriz se tiene:

$$\frac{\partial \hat{\mathbf{x}}(\mathbf{k}+\mathbf{1})}{\partial \Theta_1} = \begin{pmatrix} q_{11}\phi'(s_1)x_1 & q_{11}\phi'(s_1)x_2 & \dots & q_{11}\phi'(s_1)x_{ne} \\ q_{21}\phi'(s_2)x_1 & q_{21}\phi'(s_2)x_2 & \dots & q_{21}\phi'(s_2)x_{ne} \\ \vdots & \vdots & \ddots & \vdots \\ q_{no1}\phi'(s_{no})x_1 & q_{no1}\phi'(s_{no})x_2 & \dots & q_{no1}\phi'(s_{no})x_{ne} \end{pmatrix} \tag{4.22}$$

$$\mathbf{D_2} = \frac{\partial \hat{\mathbf{x}}(\mathbf{k}+\mathbf{1})}{\partial \Theta_1} = \begin{pmatrix} \begin{pmatrix} X \\ 1 \end{pmatrix} & 0 & \dots & 0 \\ 0 & \begin{pmatrix} X \\ 1 \end{pmatrix} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \begin{pmatrix} X \\ 1 \end{pmatrix} \end{pmatrix} \Phi' \Theta_2^T \tag{4.23}$$

Luego de haber hallado las derivadas de $\hat{x}(k+1)$ con respecto a cada una de los parámetros de Θ_1 y Θ_2 tenemos:

$$\frac{\partial \hat{\mathbf{x}}(\mathbf{k}+\mathbf{1})}{\partial \Theta} = \begin{pmatrix} D_2 \\ D_1 \end{pmatrix} \tag{4.24}$$

Teniendo calculado el gradiente de la función de costo respecto a los parámetro Θ_1 y Θ_2, los pesos se pueden actualizar de acuerdo a la ecuación (4.7).

El algoritmo de BPEA puede ser descrito de la siguiente manera:

1. Establecer el vector X.

2. Estimar $\hat{x}(k+1)$. (Ecuación (4.16))

3. Calcular $\psi(k+1)$. (Ecuación (4.13))

4. Calcular el error de predicción: $\varepsilon(k+1) = x(k+1) - \hat{x}(k+1)$

5. Calcular la derivada del gradiente de la función de costo:
   $G_j(k) = \lambda G_j(k-1) - \psi(k)\varepsilon(k)$. (Ecuación (4.11))

6. Actualizar los parámetros Θ. (Ecuación (4.7))

7. Pasar a la siguiente muestra

8. Regresar al punto 1.

4.4. Aproximación Funcional usando Redes Neuronales recurrentes

Una vez que tenemos listo el *algoritmo BPEA* recurrente, ahora lo único que debemos hacer es generar una posible estructura de nuestro aproximador funcional y luego hallar los parámetros por medio del algoritmo BPEA.

La estructura en bloques se va a utliizar para desarrollar el aproximador funcional es el que se muestra en la figura 4.14.

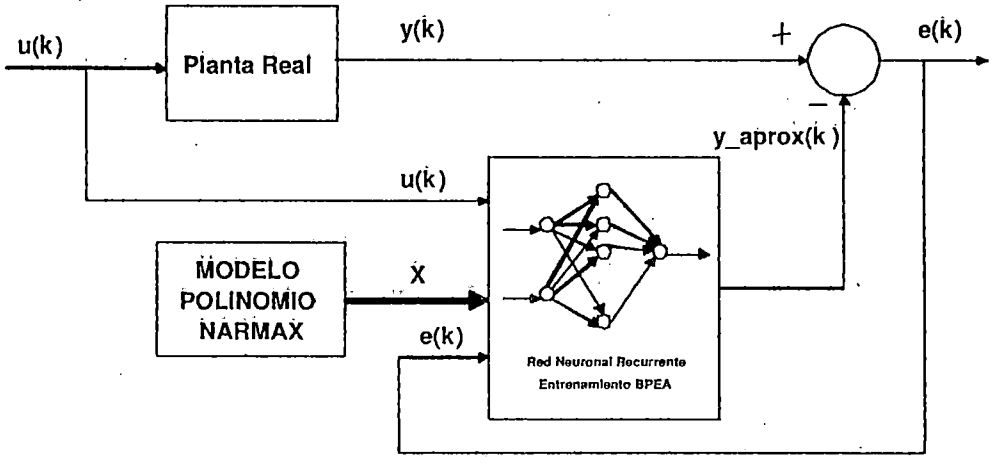

Figura 4.14: Diagraba de bloques de la estructura a utilizar para la identificación mediante aproximadores funcionales.

La estructura del aproximador funcional con la que se va a trabajar es la siguiente:

$$y_{k+1} = a_1 y_k + a_2 u_k + a_3 ek + a_4 y_k u_k + a_5 y_k e_k + a_6 u_k e_k$$

Ahora se aplicará el algoritmo BPEA para ese modelo polinomial NARMAX, para ello primero usaremos la planta lineal de segundo orden y luego la planta no lineal propuesta por Narendra [8].

$$\begin{aligned} y_{k+1} &= A_1 y_k + A_2 y_{k-1} + A_3 y_{k-2} + \ldots \\ &\quad + B_1 u_k + B_2 u_{k-1} + B_3 u_{k-2} \end{aligned} \tag{4.25}$$

donde:

$A_1 = 2{,}6277, A_2 = -2{,}3332, A_3 = 0{,}6976, B_1 = 0{,}0172,$
$B_2 = 0{,}0308, B_3 = 0{,}0140$

Como en el capítulo anterior se va a proceder a entrenar el BPEA con un entrada aleatoria, luego se analizá el g'rafico de las correlaciones cruzadas para verificar que el modelo escogido es suficiente para representar la dinámica del sistema y finalmente se validará el modelo obtenido con entradas que no han sido utilizadas para el entrenamiento de la red neuronal.

Pero antes se debe establecer los parámetros de la red neuronal (ver tabla 4.3).

La respuesta del sistema de segundo orden y la del sistema aproximado es mostrada en la figura 4.15.

Tabla 4.3: Parámetros de la red neuronal (BPEA recursivo)

Número de neuronas de entrada N_e	12
Número de neuronas ocultas N_e	18
Número de neuronas de salida N_e	1
Número de iteraciones	400
Factor de olvido λ	0.5164
Rate de aprendizaje μ	0.1510

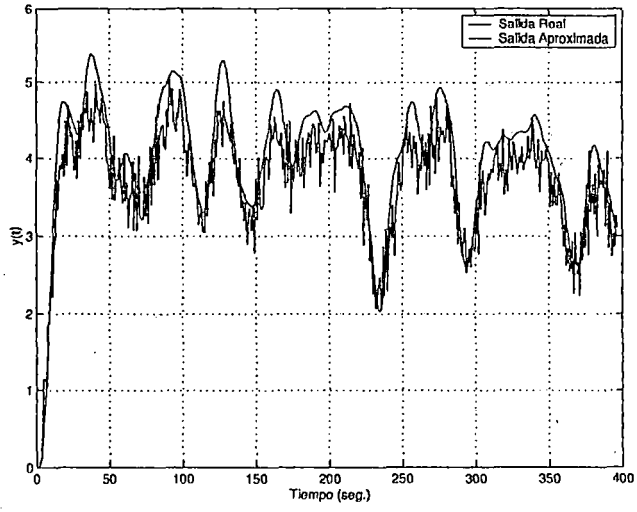

Figura 4.15: Respuesta del sistema real y del sistema aproximado después del entrenamiento de la red neuronal con una entrada aleatoria.

Ahora luego de haber entrenado la red neuronal se procederá a validar la estructura seleccionada. Para ello se aplica el *Análisis de la Correlación Cruzada* que se explicó en el Capítulo II, obteniendo la figura 4.16.

Como se observa de la figura 4.16 del total de puntos analizados menos del 5% del total caen fuera de los límites por lo tanto la estructura seleccionada es suficiente para representar la dinámica del sistema.

Ahora se validará el sistema encontrado exitando tanto al sistema real como al sistema aproximado con entradas: escalón, senoidal y cuadrada (ver figuras 4.17, 4.18, 4.19):

En la tabla 4.4 se muestra los error de aproximación producido por el aproximador funcional entrenado con redes neuronales frente a diferentes tipos de en-

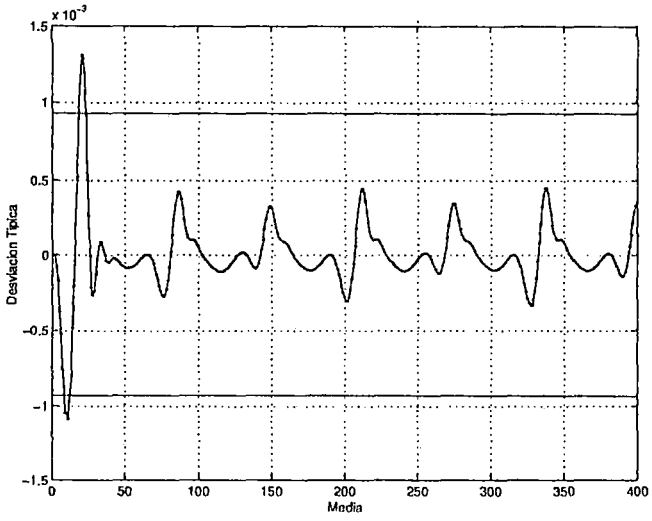

Figura 4.16: Validación de la estructura propuesta (análisis de la correlación cruzada).

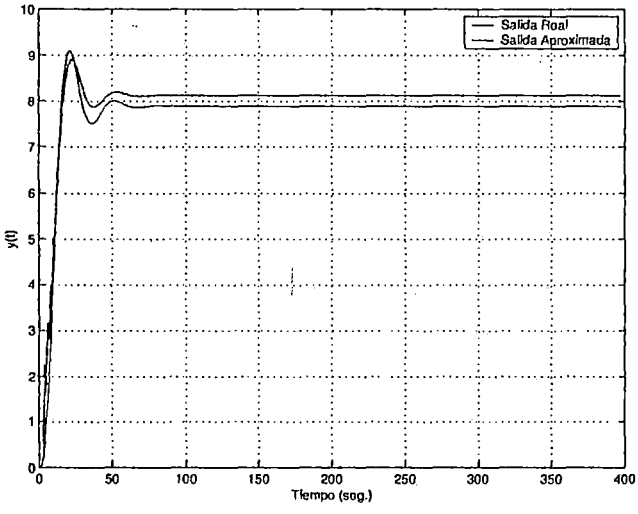

Figura 4.17: Respuesta del sistema real y del sistema aproximado ante una entrada escalón unitario - caso lineal.

tradas. El programa de esta parte se encuentra en el *Apéndice A* con el nombre de $bpea_V.m$.

Ahora se probará la eficiencia de la red neuronal con el modelo no lineal propuesto por Narendra [8]:

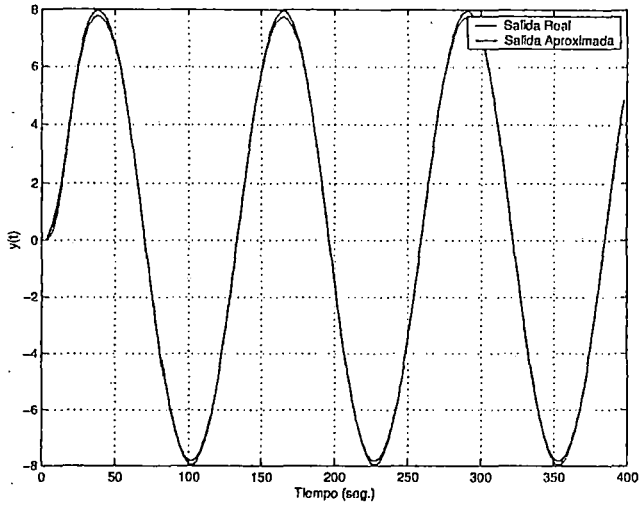

Figura 4.18: Respuesta del sistema real y del sistema aproximado ante una entrada senoidal - caso lineal.

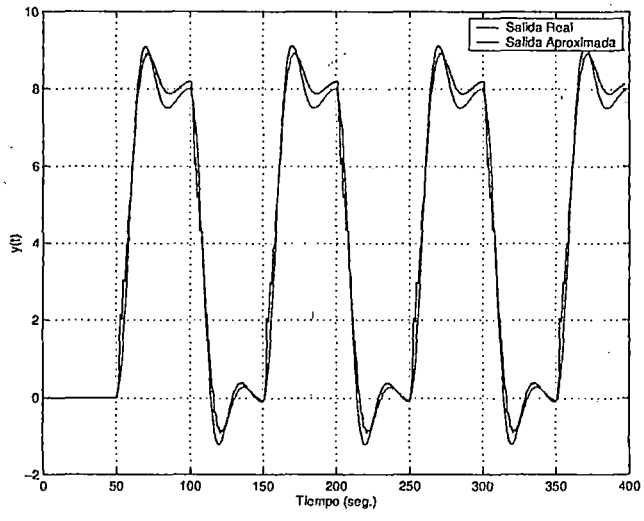

Figura 4.19: Respuesta del sistema real y del sistema aproximado ante una entrada cuadrada - caso lineal.

$$y_{k+1} = \frac{y_k}{(1+y_k)^2} + u_k^3 \tag{4.26}$$

Someteremos al sistema real y al sistema aproximado a una entrada aleatoria para el respectivo entrenamiento de la red neuronal (ver figura 4.20). Los parámetros iniciales utilizados por la red neuronal son los mismos que en el caso anterior.

Tabla 4.4: Errores de la red neuronal frente a diferentes tipos de entradas.

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N}) \cdot 100\,\%$
Escalón	4.5314 %
Senoidal	1.7251 %
Cuadrada	4.7304 %

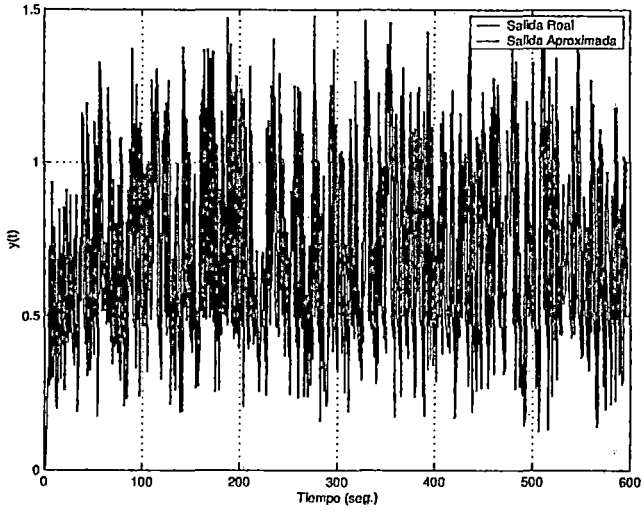

Figura 4.20: Respuesta del sistema real y del sistema aproximado ante una entrada aleatoria (fase de entrenamiento).

Luego de realizar el entrenamiento de la red se procederá a validar la estructura para ver si es suficiente como para representar la dinámica completa del sistema.

Como se observa de la figura 4.21 menos del 5 % de los puntos caen fuera del los límites, por lo tanto afirmamos que el sistema hallado es suficiente para representar la dinámica completa del sistema. Ahora se procederá a validar el modelo con entradas de diferentes tipos: escalón, senoidal y cuadrada (ver figuras 4.22, 4.23, 4.24).

De las figuras 4.22, 4.23, 4.24 se obtiene la tabla 4.5, de la cual se afirma que la estructura del modelo polinomial NARMAX que se ha propuesto y los parámetros para dicha estructura hallado por la red neuronal cumple con seguir

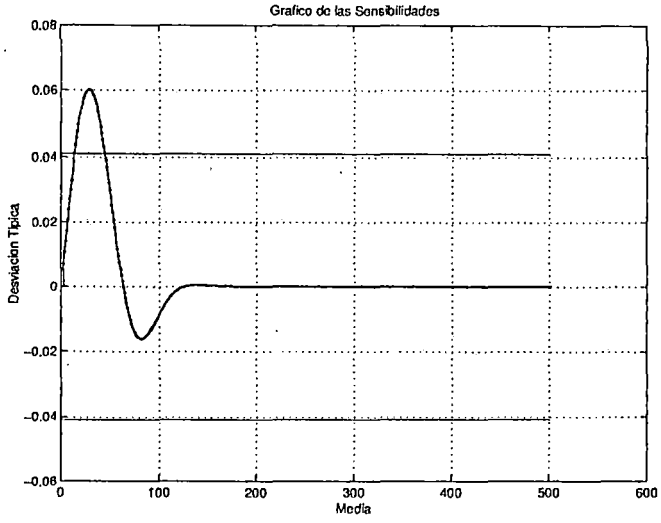

Figura 4.21: Validación de la estructura propuesta (análisis de la correlación cruzada).

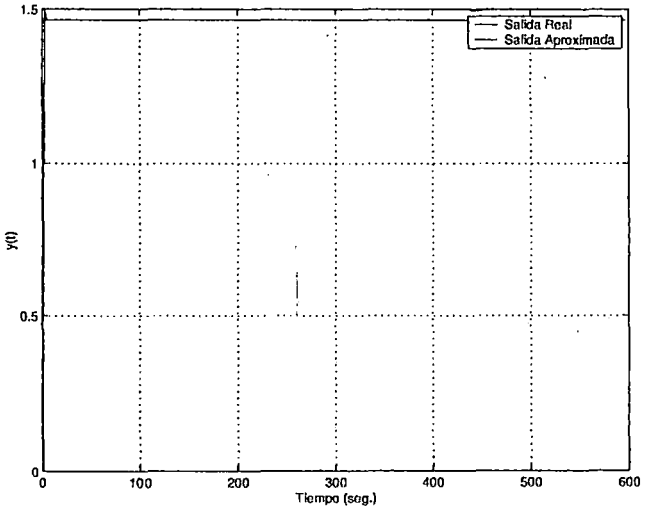

Figura 4.22: Respuesta del sistema real y del sistema aproximado ante una entrada escalón unitario - caso no lineal.

la dinámica del sistema real frente a entradas que no han sido utilizadas para su entrenamiento. El programa de esta parte se encuentra en el *Apéndice A* con el nombre de *bpea_V_no.m*.

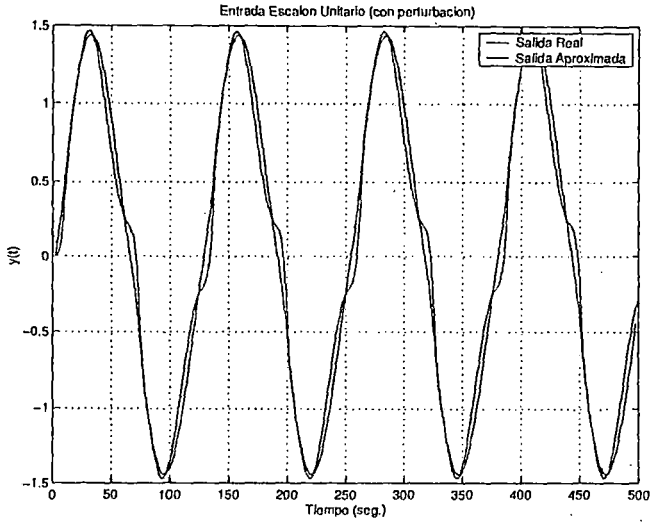

Figura 4.23: Respuesta del sistema real y del sistema aproximado ante una entrada senoidal - caso no lineal.

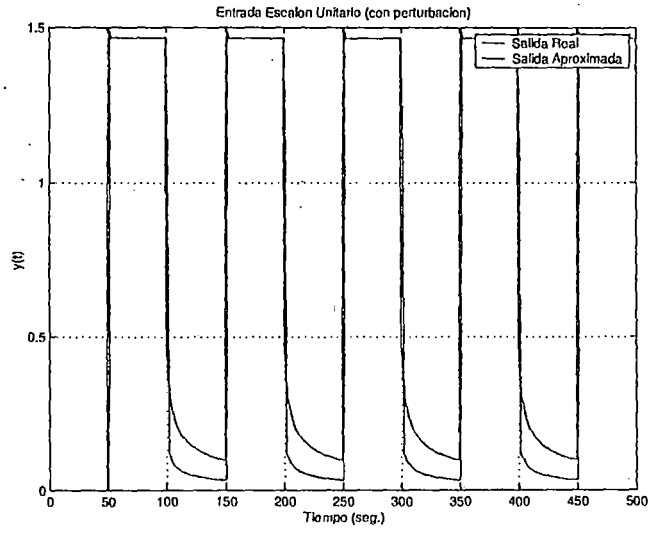

Figura 4.24: Respuesta del sistema real y del sistema aproximado ante una entrada cuadrada - caso no lineal.

Tabla 4.5: Errores de la red neuronal frente a diferentes tipos de entradas.

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N})\cdot 100\,\%$
Escalón	0.0022 %
Senoidal	2.2565 %
Cuadrada	16.0706 %

CAPÍTULO V
APROXIMACIÓN FUNCIONAL A UN MODELO NARMAX USANDO ALGORITMOS GENÉTICOS

En la naturaleza todos los seres vivos se enfrentan a problemas que deben resolver con éxito, como conseguir más luz del sol, o cazar una mosca. La Computación Evolutiva interpreta la naturaleza como una inmensa máquina de resolver problemas y trata de encontrar el origen de dicha potencialidad para utilizarla en nuestros programas. Los Algoritmos Genéticos son una de las más conocidas y originales técnicas de resolución de problemas dentro de lo que se ha definido como *Computación Evolutiva* (o *Algoritmos Evolutivos*), término que agrupa a los Algoritmos Genéticos, las Estrategias Evolutivas y la Programación Evolutiva[12]. En realidad todas estas técnicas son muy parecidas y comparten muchos aspectos.

Un Algoritmo Genético (AG) es una técnica de resolución de problemas inspirada en la evolución de los seres vivos. En un Algoritmo Genético (AG) se define una estructura de datos que admita todas las posibles soluciones a un problema. Cada uno de los posibles conjuntos de datos admitidos por esa estructura será una solución al problema. Unas soluciones serán mejores, otras peores. Solucionar el problema consistirá en encontrar la solución óptima, y por tanto, los Algoritmos Genéticos (AG) son en realidad un método de búsqueda. Pero un método de búsqueda muy especial, en el que las soluciones al problema son capaces de reproducirse entre sí, combinando sus características y generando nuevas soluciones. En cada ciclo se seleccionan las soluciones que más se acercan al objetivo buscado, eliminando el resto de soluciones. Las soluciones seleccionadas se reproducirán entre sí, permitiendo de vez en cuando alguna mutación o modificación al azar durante la reproducción.

En este capítulo, primero daremos una breve reseña de la historia de los Al-

goritmos Genéticos (AG), luego resolveremos una gran interrogante *¿porqué funcionan los algoritmos genéticos?*, analizaremos su fundamento matemático (*Teorema de los Esquemas*), veremos también los diferentes operadores genéticos con los cuales trabaja un AG. Luego mostraremos el desempeño del *AG* a través de un pequeño ejemplo de optimización de una función. Luego hablaremos acerca de una técnica heurística conocida como *Templado Simulado*[1]. Este procedimiento se basa en una analogía con el comportamiento de un sistema físico al someterlo a un baño de agua caliente. El *Templado Simulado* (SA[2]) ha sido probado con éxito en numerosos problemas de optimización, mostrando gran *"habilidad"* para evitar quedar atrapado en óptimos locales. Además analizaremos el comportamiento de un algoritmo híbrido: *Algoritmo Genético y Simulated Annealing SA-GA*[3]. Finalmente realizaremos las diferentes respuestas de la aproximación funcional usando el modelo *NARMAX*, para un modelo lineal de segundo orden y de un modelo no lineal, aplicando algoritmos genéticos y el algoritmo híbrido SA-GA.

5.1. Breve Historia

Desarrollados por Holland en la Universidad de Michigan, los algoritmos genéticos (AG) son algoritmos de búsqueda basados en la genética natural. Estos algoritmos combinan la supervivencia de los más aptos, con un intercambio de información estructurada y aleatoria a la vez, con el fin de formar mejores individuos.

Actualmente los algoritmos genéticos están siendo tomados con gran interés en las diferentes ramas de ingeniería, en la bio-tecnología, en economía, etc. debido a su gran eficiencia y robustez en resolver situaciones en lo que no se tiene un conocimiento claro o se tiene poca información acerca de la naturaleza del problema.

En la tabla 5.1 se describe los inicios y el desarrollo de la computación

[1]Templado Simulado, Simulated Annealing, Templado Recocido son términos que engloban al mismo concepto

[2]SA son las siglas en inglés de Simulated Annealing

[3]SA-GA proviene de las siglas en inglés Simulated Annealing - Genetic Algorithm

evolutiva y de los algoritmos genéticos.

5.2. Algoritmo Genético Simple

Los Algoritmos Genéticos son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas. Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor ó puntuación, relacionado con la bondad de dicha solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos, descendientes de los anteriores, los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergerá hacia

Tabla 5.1: Historia de la computación evolutiva y de los algoritmos genéticos

1932	**Canon** visualiza la evolución natural como un proceso de aprendizaje muy similar al proceso mediante el cual una persona aprende por ensayo y error.
1950	**Alan Turing** también reconoció una conexión *obvia* entre la evolución y el aprendizaje de máquina.
1957	**Friedberg** se dió a la tarea de mejorar gradualmente un programa de computadora manipulando sus instrucciones en forma similar al proceso de evolución (programación automática). Sus resultados sin embargo no fueron alentadores, pues se demostró que una búsqueda puramente aleatoria podía proporcionar mejores resultados que su método.
1960	**Campell** conjeturó que *en todos los procesos que nos conducen a la expansión del conocimiento, está involucrado un proceso de variación ciega y supervivencia selectiva.*
1962	**Bremermann** fue tal vez el primero en ver a la evolución como un proceso de optimización e introdujo el uso de la representación binaria de un cromosoma y la noción de función de aptitud.
1962	**Holland** desarrolla los *planes reproductivos* y *adaptativos* en un intento de hacer que las computadoras aprendan imitando el proceso de la evolución. Esta técnica después sería conocida mundialmente como el *Algoritmo Genético.* Aunque concebido originalmente en el contexto del aprendizaje de máquina, el algoritmo genético se ha utilizado mucho en optimización.
1963	**Biener, Rechenberg y Schwefel** desarrollaron en la Universidad Técnica de Berlín una técnica que después denominarían *Estrategias Evolutivas.* El énfasis de esta técnica fue el poder optimizar funciones que usan variables reales, y cuyo espacio de búsqueda es tan complejo que las técnicas convecionales de optimización no encuentran resultados satisfactorios.
1965	**L.Fogel** concebió el uso de la evolución simulada en una población de algoritmos contendientes para desarrollar inteligencia artificial en una disciplina que él mismo denominó *programación evolutiva.* El veía el comportamiento inteligente como algo que requería la capacidad de: -Predecir su propio ambiente. -Traducir estas predicciones en una respuesta adecuada con respecto a la meta específica que se persiguiera.

Tabla 5.2: Continuacion...

	Para fines de generalización, el ambiente se describió como una secuencia de símbolos tomados de un alfabeto finito. El problema evolutivo se definió entonces como el de evolucionar un algoritmo que operaría sobre la secuencia de símbolos que se hubiera observado hasta ese momento de tal manera que se produjera un símbolo de salida que tuviera una buena probabilidad de maximizar la eficiencia del algoritmo en términos de la predicción.
1967	**Box** y sus colegas desarrollan la *operación evolutiva* para simular el manejo de una planta de manufactura. Esta técnica se aplicó a un gran número de procesos de manufactura, aunque su dominio más común fué la industria química.
1967	**Reed, Toombs y Barricelli** realizaron una de las primeras simulaciones basadas en principios evolutivos, siguiendo los lineamientos de lo que hoy se conoce como *Vida Artificial.*
1974	**Conrad** propone el *modelo de circuitos de aprendizaje evolutivo* en el cual se especula sobre la posibilidad de que el cerebro use el mismo tipo de mecanismos que usa la evolución para aprender.
1991	**Ray** libera el simulador *Tierra* como una aplicación de la llamada *Dinámica Evolutiva.* El objetivo del trabajo de Ray ha sido el poder desarrollar simulaciones evolutivas que permitan estudiar las características fundamentales de todos los sistemas evolutivos.
1992	**Koza** propone el uso de expresiones-S en LISP para representar cromosomas en vez de cadenas binarias de Holland, en un intento por incrementar el poder del algoritmo genético. Su técnica sería posteriormente denominada *Programación Genética.*

una solución óptima del problema.

Una defición general de los AG podría ser como sigue:

Los Algoritmos Genéticos son métodos estocásticos de búsqueda ciega de soluciones cuasióptimas. En ellos se mantiene una población que representa a un conjunto de posibles soluciones la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos

candidatos y a un proceso de selección sesgado en favor de los mejores candidatos [10].

Como ya mencionamos en el párrafo anterior, en un Algoritmo Genético, una población de individuos, que representan a un conjunto de candidatos a soluciones de un poblema, es sometida a una serie de transformaciones con las que se actualiza la búsqueda y después en un proceso de selección se favorece a los mejores individuos. Cada ciclo de *selección+búsqueda* constituye una generación, donde el mejor individuo de dicha población en esa generación representa a un candidato lo suficientemente próximo a la solución buscada. La estructura genérica del bucle básico de un AG se sintetiza en la figura 5.1:

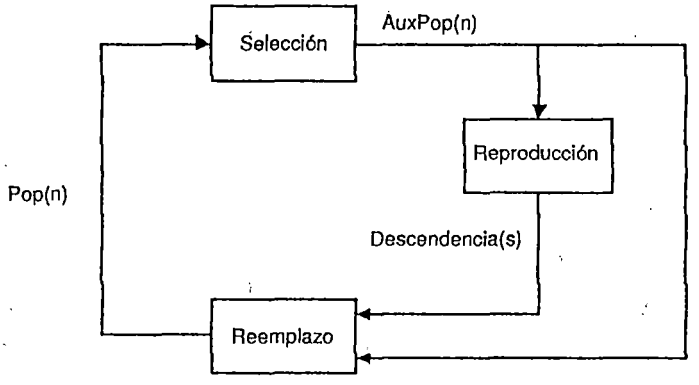

Figura 5.1: Bucle básico de un AG.

Como puede verse, una población *Pop*, que consta de n miembros se somete a un proceso de *selección* para constituir una población intermedia *AuxPop* de n *criadores*. De dicha población intermedia se extrae un grupo reducido de individuos llamados *progenitores* que son los que efectivamente se van a reproducir. Sirviéndose de los *operadores genéticos*, los progenitores son sometidos a ciertas transformaciones de *alteración y recombinación* en la fase de *reproducción* en virtud de las cuales se generan s nuevos individuos que constituyen la *Descendencia*. Para formar la nueva población *Pop[t+1]* se deben seleccionar n supervivientes entre los $n+s$ de la población auxiliar y la descendencia, eso se hace en la fase de *reemplazo*.

Cada individuo x consta de m posiciones que son ocupados por otros tantos atributos o *genes* los cuales se codifican en $l = L_1 + \ldots + L_m$ bits. Naturalmente,

si un atributo puede tomar más de dos valores o *alelos* se deberá representar mediante varios bits, de hecho es inmediato comprobar que si el *j-ésimo* gen consta de a_j alelos, entonces se deberá codificar mediante:

$$L_j = [\log_2 a_j] bits \tag{5.1}$$

En este punto conviene distinguir entre la estructura de los individuos y el contenido de cada uno: la estructura se deriva de la *representación*, es igual para todos y se expresa así "los 4 primeros bits representan al primer gen, los 4 siguientes al segundo, etc...", el contenido se deriva de la *codificación* y es simplemente el entero binario con que se codifica cierto individuo.

En la jerga al uso de la estructura de un individuo se le dice *genotipo* y a su contenido *fenotipo.*

Todo el procedimiento de búsqueda está guiado exclusivamente por una *función de aptitud u(x)*, la cual se obtiene directamente a partir de la *función de evaluación f(x)* del correspondiente problema. La función de evaluación no tiene por qué estar expresada de forma cerrada como una función objetivo clásica, basta con que proporcione un "índice de idoneidad" para cada uno de los candidatos a solución que se le presenten. De todos modos es conveniente que el procedimiento de obtención de dicho índice se pueda implantar con facilidad en una computadora, dado que la evolución de todo el algoritmo va a depender de él. Comúnmente, a los valores proporcionados por la función de evaluación se les dice *evaluaciones* o bien *aptitudes brutas* mientras que a los valores proporcionados por la función de aptitud se les dice aptitudes a secas, y también, para distinguirlas de las aptitudes brutas, *aptitudes netas.*

Considerando lo discutido en el párrafo anterior, vamos a mostrar un seudocódigo con el cual vamos a implementar en las secciones siguientes nuestro AG:

```
BEGIN /*Algoritmo Genético Simple*/
    Generar la población inicial.
    Calcular las funciones de evaluación de cada
    individuo de la población.
    WHILE NOT Terminado DO
    BEGIN /*Pasar a la siguiente generación*/
        FOR Tamaño de la Población
        BEGIN /*Operador Genéticos*/
            Selección dos individuos de la generación anterior,
                para el cruce.
            Cruzamiento con cierta probabilidad
                establecida los dos individuos se cruzan para obtener sus
                descendientes.
            Cálculo se calcula las aptitudes de los
                nuevos pobladores
        END
        IF la población converge THEN
            Terminado=TRUE
        END
    END
END
```

5.2.1. Criterios para Implementar un AG

Para implementar nuestro AG es necesario definir los siguientes criterios:

1. **Criterio de Codificación.** Dado que nuestro AG a implementar va a operar exclusivamente con cadenas binarias[4] debido a la fácil implementación de este tipo de cadenas debemos expecificar el modo de hacer corresponder cada punto del dominio del problema con una cadena determinada, o en términos de las definiciones anteriores, el mecanismo de paso del genotipo a los fenotipos.

   El modo más sencillo de realizar este procedimiento es discretizando el dominio del problema, $[x_{min}, x_{max}]$ en una cantidad de puntos iguales a 2^{Length} tal que la distancia entre dos puntos consecutivos sea menor que la tolerancia especificada:

[4] Cadenas binarias sin signo

$$\frac{x_{max} - x_{min}}{2^{length} - 1} < TOL \tag{5.2}$$

De este modo cada punto del espacio de búsqueda queda representado mediante un entero binario de longitud *length*, comenzando por la cadena $0 \ldots 0$ que representa a x_{min} hasta la cadena $1 \ldots 1$ que representa a x_{max}.

En definitiva la clave de la representación reside en calcular la longitud de los individuos. Si se dispone de la tolerancia TOL el cálculo es inmediato a partir de la ecuación (5.2):

$$length = log_2(1 + \frac{x_{max} - x_{min}}{TOL})\, bits \tag{5.3}$$

Por ejemplo para representar un conjunto de posibles soluciones dentro del intervalo $[-1, 1]$ con una toleracia de 0.01, aplicamos la ecuación (5.3) con lo cual obtenemos el siguiente resultado:

$$length = log_2(1 + \frac{1 - (-1)}{0{,}01})$$

$$length = 7{,}65 \sim 8bits$$

Esto quiere decir que para poder representar a los posibles candidatos con una tolerancia menor o igual a 0.01, necesitamos codificar a cada uno de ellos utilizando 8 bits.

Otro problema que se tiene que tener presente dentro del criterio de codificación es el de la codificación binaria, pero para tratar este punto con más detalle es preciso definir el concepto de *Distancia de Hamming*[6]:

Distancia de Hamming: Sea el $x = [x1, \ldots, xn]$ e $y = [y1, \ldots, yn]$ dos vectores que pertenecen al espacio euclidiano *n-dimensional*, donde $x_i, y_i \in [0, 1]$, entonces se define la distancia de Hamming como:

$$d_{hamming} = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2} \tag{5.4}$$

Por ejemplo en la figura 5.2, se muestra un cubo euclidiano[5] de tres dimensiones R^3, con todos los vectores que forman parte de este plano.

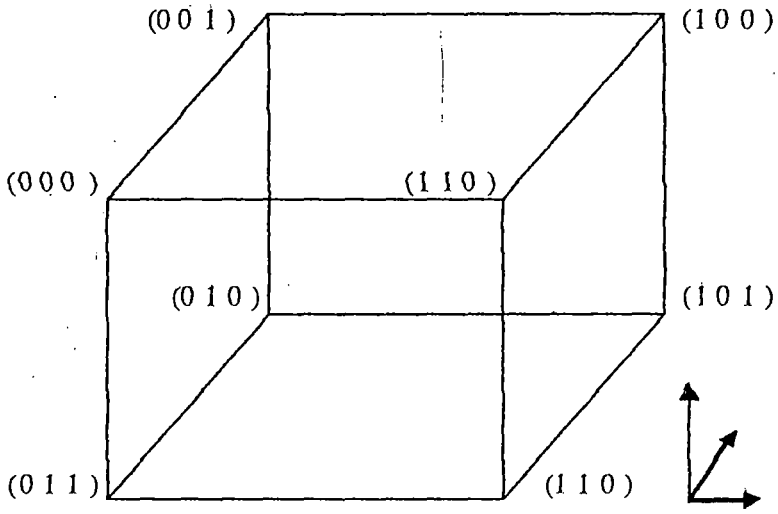

Figura 5.2: Cubo Euclidiano de tres dimensiones.

[5]El concepto de cubo euclidiano hace alusión a un plano formado por n-dimensiones

Ahora una vez que tenemos en claro el concepto de distancia Hammintoniana, vamos a resolver el problema de la representación binaria.

Cuando se representa dos enteros consecutivos a través del alfabeto binario, estas dos representaciones tienen codificaciones muy distintas, y esta carencia de coincidencias entre dos enteros consecutivos se conoce como *Acantilados de Hamming*[10]. Como consecuencia de esto, cuando el individuo se somete al operador genético de la mutación puede producir cambios drásticos en los puntos representados. Para solucionar este problema es necesario el uso de la Codificación Gray que reduce grandemente la frecuencia de aparición de los *Acantilados de Hamming*, para ver mejor este problema, lo vamos a graficarlo con un ejemplo:

Sean dos enteros consecutivos $X_1 = 115$ y $X_1 = 116$.
Sea el espacio euclidiano con 7 dimensiones $\longrightarrow R^7$, y sean dos vectores de ese espacio (en representación binaria):

$$x_{bin} = (1110011) \leftarrow 115$$

$$y_{bin} = (1110100) \leftarrow 116$$

Entonces, la distancia Hammintoniana segun la ecuación 5.2 es:

$$d_{hamming-binario} = \sqrt{0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 1^2 + 1^2}$$

$$d_{hamming-binario} = \sqrt{3} \sim 1{,}7321$$

Ahora esos mismos números enteros en código gray tendrían las siguientes representaciones:

$$x_{gray} = (1001010)$$

$$y_{gray} = (1001110)$$

Entonces la distancia hammintoniana para la representación con el código gray segun la ecuación 5.2 es:

$$d_{hamminggray} = \sqrt{0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2}$$

$$d_{hamming-gray} = 1$$

Con lo cual podemos observar que en la codificación binaria la distancia de dos números consecutivos tiene una $d_{hamming}$ mayor a la de la codificación gray, por lo tanto si realizamos la representación de los individuos en código gray los cambios producidos debido a la mutación proporcionarán ligeras modificaciones y unas cuantas pocas veces cambios drásticos los cuales darán una zona de exploración completamente nueva. En cambio, en la codificación binaria, la mutación produciría cambios drásticos generandos puntos de solución no factibles.

2. **Criterio de Tratamiento de los puntos no factibles.** No siempre es posible encontrar una correspondencia punto por punto entre el dominio del problema y el conjunto de las cadenas binarias usadas para resolverlos. Como consecuencia, no todas las cadenas codifican elementos válidos del espacio de búsqueda y se deben habilitar procedimientos útiles para distinguirlas. Pero como sabemos un AG es un método de búsqueda irrestricto y cualquier solución a este problema pasará por la incorporación, de manera directa o indirecta, de conocimiento al problema.

Tradicionalmente se han empleado varias técnicas (Golberg y Wolfgang han realizado numerosos estudios acerca de este punto y los han recopilado en sus respectivos libros [11],[5]), pero en la presente tesis analizaremos en detalle la técnica de penalización.

- **Técnicas de Penalización.** Consiste en resolver directamente el problema pero penalizando a los individuos no factibles. Es decir el AG va a generar individuos sin tener en cuenta su factibilidad pero reduciendo la aptitud de los individuos que resulten no factibles.

  La única exigencia de estas técnicas es que debe poderse medir de alguna manera *el grado de no-factibilidad*, para así dar mayor penalización a los puntos más alejados del dominio del problema. Ésto es fácil de hacer en problemas de optimización paramétrica, pues en ellos se suele definir el dominio de los puntos factibles X a través de un conjunto de p restricciones expresadas como inecuaciones:

$$g_i(x) \geq 0$$

  Entonces con esos datos definimos el *grado de violación de la i-ésima restricción* de la siguiente manera:

$$h_i(x) = \begin{cases} -g_i(x) & \text{si } g_i(x) < 0 \\ 0 & \text{si } g_i(x) \geq 0 \end{cases} \qquad (\forall i = 1, \ldots, p)$$

  Luego incluimos esos h_i, en la función de evaluación:

$$Eval(v) \longleftarrow Eval(v) - K \cdot Penalty(h_i, \ldots, h_p) \qquad (5.5)$$

  donde:

p es el número total de restricciones.
h_i es el grado de violación de la i-ésima restricción.
K es un coeficiente global de penalización.

La función $Penalty(\cdot, \ldots, \cdots)$ es una función positiva[6] y creciente que se define para cada problema.

5.3. Operadores Genéticos

Los Operadores Genéticos son operaciones que han sido modelados según el Principio Darwiniano de reproducción y supervivencia del más apto.

Exiten una gran variedad de operadores genéticos pero los más importantes (los necesarios para lograr para que el AG dé una buena aproximación), son la *Selección*, el *Cruzamiento* y la *Mutación*. A continuación mostraremos el concepto de cada una de ellos y sus técnicas respectivas, los cuales serán utilizados en nuestros programas.

5.3.1. Selección

Primero, debemos tener presente que todo AG debe verificar los siguiente:

"Para un buen funcionamiento de un AG es esencial tener controlado en todo momento la diversidad de la población. La diversidad de los indiviuos con lleva a una diversidad de aptitudes."

La necesidad de que exista una diversidad de individuos radica en que el *operador de cruce* va a poder intercambiar una mayor diversidad de carga genética (de lo contrario la búsqueda se estanca) y la necesidad de que exista una diversidad de aptitudes radica en que, en una población que tiene poca diversidad de aptitudes, el individuo más apto (superindividuo[7] para designar a los individuos

[6]En este caso estamos asumiendo que estamos maximizando la función, cuando minimicemos, será negativa.

[7]Término utilizado por [5].

de una población que tienen una aptitud muy superior a la del promedio) poblaría rápidamente, con lo cual conllevaría al AG a una *convergencia prematura*, habitualmente un óptimo local. En las fases iniciales del AG es deseable que el AG evalúe la mayor área de la región de búsqueda y que en las fases tardías, el AG converga al área de búsqueda donde se encuentra el óptimo global.

Nuestro operador de Selección debe cumplir con esos requisitos expuestos en el párrafo anterior, por lo tanto vamos a exponer un tipo de Selección que ha mostrado buenas características frente a la Selección Proporcional (De Ruleta) que es la Selección por Torneos.

- Selección Proporcional (De Ruleta)

El modo de funcionamiento de esta técnica es la siguiente: A cada individuo se le asigna una porción de la *ruleta circular*, que es proporcional a la aptitud de cada individuo. La rueda se gira N veces, donde N es el número de pobladores. En cada vuelta, el individuo que se encuentra bajo el marcador de la ruleta es seleccionado como uno de los padres para la siguiente generación.

El problema principal de este tipo de selección es que para un AG que trabaja con pocos individuos, el más apto poblaría rápidamente, haciendo que el AG converga prematuramente sin explorar la mayor parte de la región de búsqueda.

A continuación se muestra el seudocódigo de esta técnica de selección:

```
BEGIN /*Selección Proporcional-Ruleta*/
    Sumar las aptitudes de todos los individuos de la poblacion.
    Llamamos a esa suma S
    FOR N veces
        Escoger un número aleatorio r entre 0 y S.
        Ciclar a través de los individuos, sumando sus
        aptitudes hasta que la suma sea menor o igual que r
        El individuo cuya aptitud sea mayor a la suma r
        es seleccionado.
    END
END
```

- Selección por Torneo

Opera de la siguiente manera: se escogen aleatoriamente dos individuos de la población y se hacen competir en base a su aptitud. El individuo que resulte más apto será el ganador del tornero y se elige como padre para la siguiente generación.

Con está técnica se previene la convergencia prematura del AG, se ejerce una presión selectiva en momentos en que la varianza de la aptitud es baja, logrando una convergencia en las fases tardías del AG. Esta técnica computacionalmente consume muy poco tiempo y es ideal para su implementación en paralelo. A continuación se muestra el seudocódigo de esta técnica de selección:

```
BEGIN /*Selección por Torneo*/
    FOR N veces
        Selecciono dos individuos de la población.
        Evalúo su aptitud.
        El individuo ganador es seleccionado
    END
END
```

5.3.2. Cruzamiento

El operador de cruce se introduce dentro de un AG para proporcionar un mecanismo de intercambio estructurado de información útil (buenos *bloques constructores*) entre los individuos. Existen diversas técnicas de cruce, entre las más conocidas tenemos: *Cruce de Un Punto*, *Cruce de Dos Puntos*, y *Cruce Uniforme.*

- Cruce De Un Punto.

Teniendo los dos padres seleccionados para su cruce respectivo, se toma un número aleatorio k comprendido entre 0 y la longitud de la representación *length*, luego tenemos lo siguiente:

$$Padre_1 = \langle p_1, \ldots, p_{length} \rangle$$

$$Padre_2 = \langle q_1, \ldots, q_{length} \rangle$$

$$Hijo_1 = \langle p_1, \ldots, p_k, q_{k+1}, \ldots, q_{length} \rangle$$

$$Hijo_2 = \langle q_1, \ldots, q_k, p_{k+1}, \ldots, p_{length} \rangle$$

En la figura 5.3 se puede apreciar gráficamente como opera esta técnica.

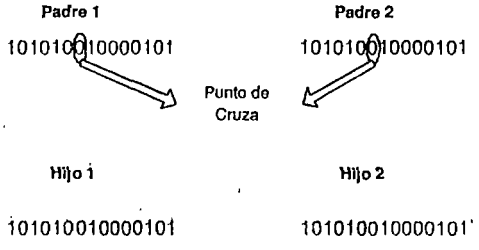

Figura 5.3: Cruze de un solo punto.

- Cruce De Dos Puntos.

Teniendo los dos padres seleccionados para su cruce respectivo, se toma dos números aleatorios k y m, donde cumplirse de que $k < m$ y k debe estar comprendido entre 0 y m, y m entre k y la longitud de la representación *length*, entonces tenemos lo siguiente:

$$Padre_1 = \langle p_1, \ldots, p_{length} \rangle$$
$$Padre_2 = \langle q_1, \ldots, q_{length} \rangle$$
$$Hijo_1 = \langle p_1, \ldots, p_k, q_{k+1}, \ldots, q_m, p_{m+1}, \ldots, p_{length} \rangle$$
$$Hijo_2 = \langle q_1, \ldots, q_k, p_{k+1}, \ldots, p_m, q_{m+1}, \ldots, q_{length} \rangle$$

En la figura 5.4 se puede apreciar gráficamente como opera esta técnica.

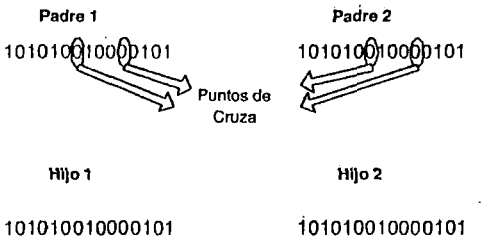

Figura 5.4: Cruce de dos puntos.

- **Cruce Uniforme.**

Teniendo los dos padres seleccionados, se genera una *máscara de cruce* aleatoria comprendida entre 0 y 2^{length} y luego se somete a los padres a la siguiente operación lógica:

$$Padre_1 = \langle p_1, \ldots, p_{length} \rangle$$
$$Padre_2 = \langle q_1, \ldots, q_{length} \rangle$$
$$M_{cruce} = \langle m_1, \ldots, m_{length} \rangle$$
$$Hijo_1 = (M_{cruce}\, AND\, Padre_1)\, OR\, (NOT\, (M_{cruce})\, AND\, Padre_2)$$
$$Hijo_2 = (M_{cruce}\, AND\, Padre_2)\, OR\, (NOT\, (M_{cruce})\, AND\, Padre_1)$$

Para visualizar mejor las operaciones anteriores, mostramos está técnica en la figura 5.5.

5.4. Ejemplo de Aplicación

En esta sección vamos a implementar un algoritmo genético simple (SGA), para optimizar una función. En la figura 5.7 mostramos el diagrama de flujo del AG que vamos a utilizar para su implementación.

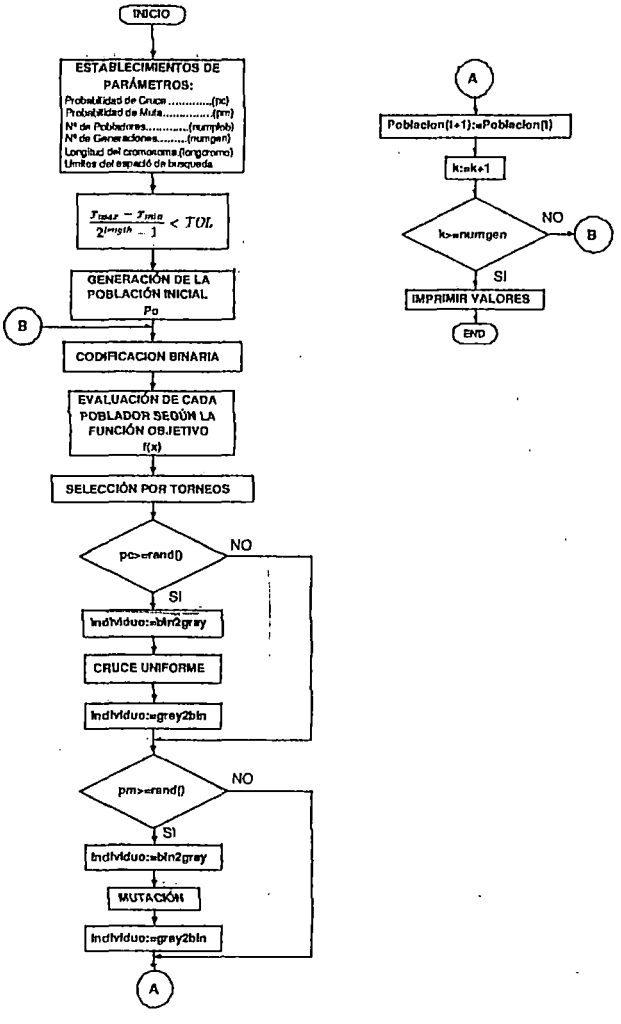

Figura 5.7: Diagrama de flujo del AG.

El objetivo que deberá cumplir nuestro AG será el de maximizar la siguiente función:

5.4. Ejemplo de Aplicación

En esta sección vamos a implementar un algoritmo genético simple (SGA), para optimizar una función. En la figura 5.7 mostramos el diagrama de flujo del AG que vamos a utilizar para su implementación.

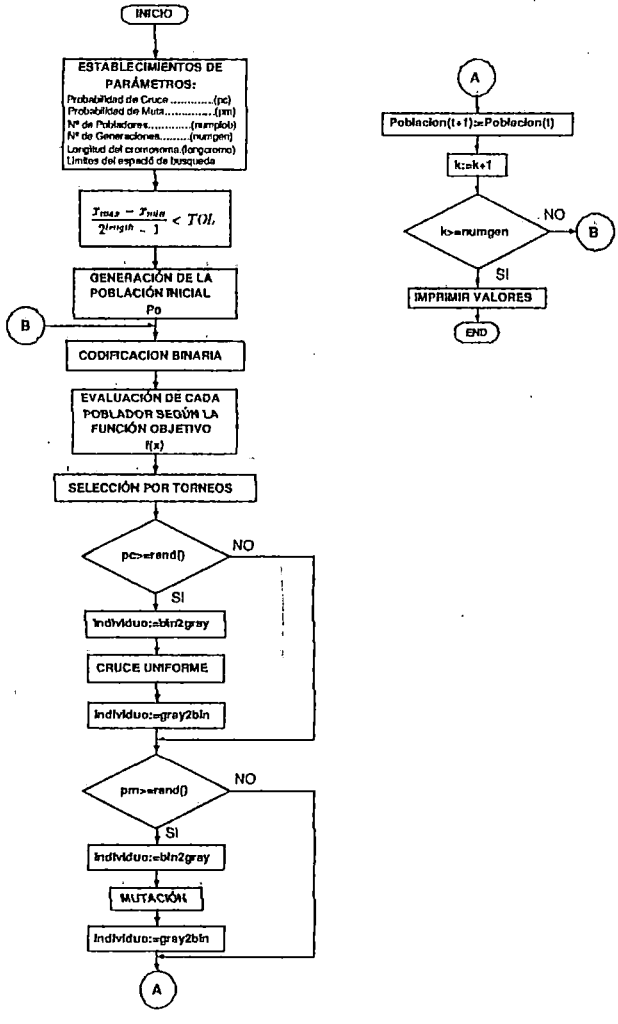

Figura 5.7: Diagrama de flujo del AG.

El objetivo que deberá cumplir nuestro AG será el de maximizar la siguiente función:

$$y = t \cdot \sin(10\pi t) \qquad \forall t \epsilon \langle -0,4; 1 \rangle$$

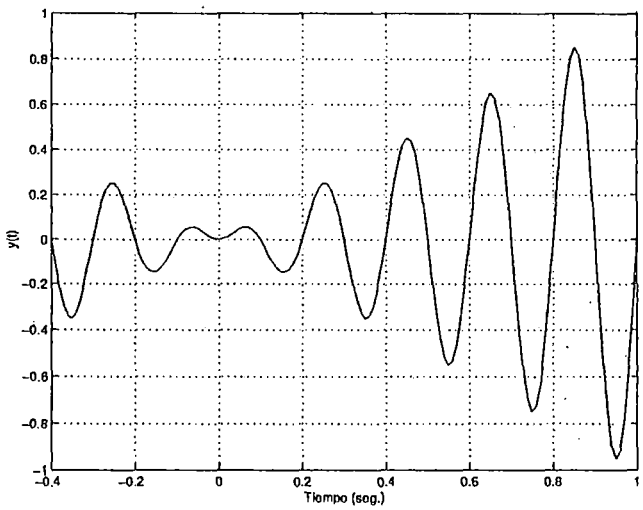

Figura 5.8: Gráfica de la función a maximizar.

Para lo cual nuestro AG será implementado con los siguientes datos:

Número de Pobladores *numplob*	100
Tolerancia de represetación *TOL*	0.01
Número de Generaciones *numgen*	100
Tipo de Selección	Por Torneo
Tipo de Cruce	Uniforme
Probabilidad de Cruce p_{cruce}	0.8
Probabilidad de Mutación p_{muta}	0.1
Límite Inferior de la región de búsqueda L_{min}	-0.4
Límite Superior de la región de búsqueda L_{max}	1.0

De la fórmula 5.3 obtenemos lo siguiente:

$$length = log_2(1 + \frac{1 - (-0{,}4)}{0{,}01})bits$$

$$length = 7{,}13 \sim 8bits$$

Con lo cual debemos generar pobladores de 8 bits (precisión de *0.0054*). Con el fin de ilustrar cómo se implanta un AG, se ha programado en MATLAB, el listado de dicho programa se encuentra en el APÉNDICE A con el nombre de *plantillagenN.m.*

En la figura 5.9 se puede observar que el AG converge hacia el valor de 1.8518, a partir de la 30^{ava} generación, los abruptos que aparecen son originados por la mutación que opera para que al AG "salga" de un óptimo local en caso de que halla "caído" en uno de ellos.

Pero como se puede observar esa inclusión de nuevo material genético debido a la mutación es rápidamente desechada debido a que el AG ya "encontró" el óptimo global.

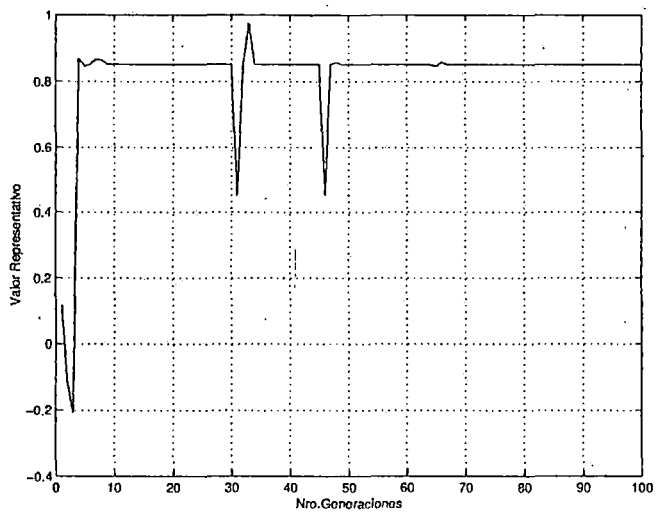

Figura 5.9: Valor a la que converge el AG.

En la figura 5.10 se muestra la evolución de las aptitudes del AG, se gráfica tanto su aptitud media como su aptitud máxima. El valor obtenido finalmente por el AG es 1.8518, el cual comparado con el máximo de la función analizada (de acuerdo a la figura 5.8) nos dá: 1.8518-1.85 = 0.0018 el cual es menor que la tolerancia requerida por nuestro problema, entonces podemos decir que el AG ha encontrado el máximo de la función con una precisión de 0.0018.

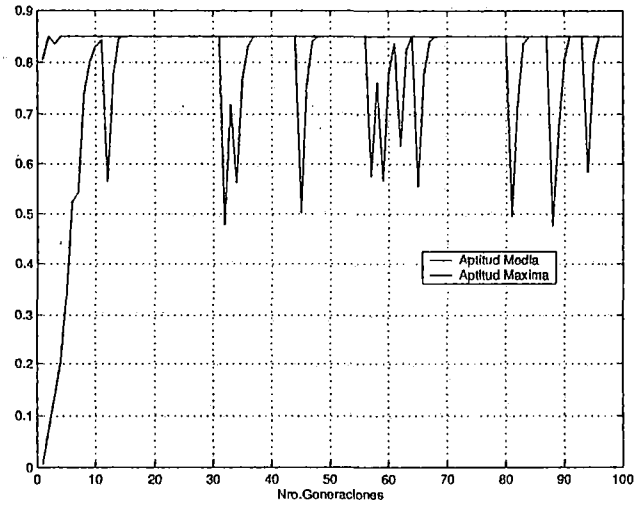

Figura 5.10: Aptitud media y aptitud máxima del algoritmo genético.

Como hemos visto, el AG es muy flexible y fácilmente implementable en una PC, proporcionando resultados aceptables (en cuanto a precisión), para una gran cantidad de problemas difícilemente resolubles por otros métodos. Ahora en la sección siguiente vamos a ver ¿porque es que funcionan los AG?.

5.5. Fundamento Matemático de los Algoritmos Genéticos

Aunque los algoritmos genéticos son simples de describir y programar, su comportamiento es muy complicado, y todavía existen muchas preguntas acerca de cómo funcionan y sobre el tipo de problemas en los que son más adecuados.

La teoría tradicional[8] de los AG asume que, en un nivel muy general de descripción, los AG trabajan descrubriendo, enfatizando y recombinado buenos *bloques constructores* de soluciones de una manera altamente paralelizada. Holland introdujo también la noción de *esquemas* para formalizar la noción de *bloques constructores.* Un *esquema* es un conjunto de cadena de bits que pueden describirse mediante una plantilla (patrón de similitud) formada por 1's , 0's y comodines (*don't care [5]*) los cuales pueden tomar cualquier valor dentro del alfabeto que

[8]Formulada originalmente por John Holland en 1975.

estemos utilizando.

Por ejemplo si trabajos sobre el alfabeto binario $V = \{0, 1\}$ y definimos el símbolo "*" como el comodín (el cual pueden tomar el valor de 1 ó de 0) entonces el esquema:

*S=(0*1100*101)*

Representa a cuatro cadenas binarias:

C1=(0011000101)

C2=(0011001101)

C3=(0111000101)

C4=(0111001101)

De esta manera podemos deducir las siguientes propiedades:

- Si un esquema contiene k símbolos *"don't care"* entonces dicho esquema representa a 2^k cadena binarias.

- Una cadena binaria de longitud *length* encaja en 2^{length} esquemas distintos.

- Si se tiene una cadena binaria de longitud *length* entonces se pueden formar 3^{length} esquemas.

Ahora definiremos dos conceptos importantes como son el *Orden de un Esquema o(S)*, y *Longitud Característica de un Esquema* $\delta(S)$.

Orden de un Esquema $o(S)$. Es el número de posiciones especificas, es decir la cantidad total de 0's y 1's, que posee dicho esquema. El concepto de

orden del esquema se utiliza para calcular la probabilidad de supervivencia del esquema con relación al operador mutación.

Longitud Característica de un Esquema $\delta(S)$. Es la distancia entre posiciones extremas fijas, vale decir entre la primera y la última posición fija. El concepto anterior se utiliza para calcular la probabilidad de supervivencia del esquema frente al operador de cruce.

Por ejemplo:

Sea el siguiente esquema:

$$S=(****00**1*)$$

Tiene 3 posiciones fijas (2 ceros y un 1) por lo tanto su orden es $o(S) = 3$ y su longitud característica es $\delta(S) = 9 - 5 = 4$.

Ahora definiremos algunos conceptos que nos ayudarán a comprender mejor el *Teorema de los Esquemas*. Dada una población de cadenas binarias $P[t]$ y un esquema S tenemos :

- **Presencia de S en P[t],** $\xi(S, P[t])$: Es el número de cadenas de $P[t]$ que encajan en el esquema S.

- **Aptitud del Esquema S en P[t],** $Fitness(S, P[t])$: Es el promedio de aptitudes de todas las cadenas de la población que encajan en el esquema S en el intante t. Esto es, si se numeran como $v_1, \ldots, v_p$, siendo $p = \xi(S, P[t])$, a las cadenas de $P[t]$ que encajan en el esquema S tenemos que:

$$Fitness(S, P[t]) = \frac{1}{p}\sum_{i=1}^{p} Fitness(v_i) \tag{5.6}$$

- **Aptitud Media de la Población en el instante t,** $AveFitn(P[t])$: Es el promedio de las aptitudes de todas las cadenas de la población en el instante t, o lo que es equivalente, la aptitud del esquema (*...*) en $P[t]$:

$$AveFitn(P[t]) = Fitness((*...*),P[t]) = \frac{1}{PopSize} \sum_{v_i \in P(t)} Fitness(v_i) \quad (5.7)$$

- **Aptitud Relativa de S en P[t],** $AveFitn(S, P[t])$: Es el cociente entre la aptitud del esquema en la población y la aptitud media de la población en cierto instante:

$$AveFitn(S, P[t]) = \frac{Fitness(S,P[t])}{AveFitn(P[t])} \quad (5.8)$$

5.5.1. Teorema de los Esquemas

¿Cómo procesa un AG los esquemas? Cualquier cadena de bits dada de longitud *length* es una instancia de 2^{length} esquemas diferentes, en otras palabras, cualquier población dada con *numplob* cadenas contiene entre 2^{length} y $numplob \cdot 2^{length}$ esquemas diferentes. Todo esto quiere decir que, en una generación, mientras el AG evalúa explícitamente *numplob* cadenas de la población, está también estimando implícitamente las aptitudes promedio de un número mayor de esquemas. Así como los esquemas no se representan o evalúan explícitamente por el AG, las aptitudes promedio de esos esquemas *Fitness(S,P[t])*, no se calculan ni almacenan explícitamente por el AG. Sin embargo, el comportaminto del AG, en términos del incremento o decremento de los esquemas dados en una población, puede describirse como si realmente estuviera calculando y almacenando estos promedios.

En una población $P[t]$, de *numplob* elementos de longitud *length* se tiene la presencia del esquema S el cual evoluciona en promedio según la siguiente fórmula:

$$\overline{\xi}(S, P[t+1]) = \overline{\xi}(S, P[t]) \cdot k_g \cdot k_s \tag{5.9}$$

donde k_g es el *factor de crecimiento* de S en la población y k_s es el *factor de supervivencia.*

El *factor de crecimiento* mide la tendencia del esquema a aumentar su presencia en la población, y el *factor de supervivencia* mide la probabilidad de que pase a la siguiente generación.

El operador genético que hace que un esquema pase a la siguiente generación es decir que hace que aumente el número de esquemas, es la selección. De hecho es fácil deducir que si se usa una selección de ruleta, la presencia de un esquema en la siguiente generación es:

$$\overline{\xi}(S, P[t+1]) = \overline{\xi}(S, P[t]) AveFitn(S, P[t]) = \overline{\xi}(S, P[t]) \frac{Fitness(S,P[t])}{AveFitn(P[t])} \tag{5.10}$$

Ahora bien existe la posibilidad que los esquemas sean destruídos por los operador de cruce y mutación antes de llegar a la siguiente generación, esta probabilidad es cuantificada por el factor de supervivencia.

Cuando se aplica el operador de cruce y mutación, estos dos operadores trabajan directamente sobre el orden y la longitud del esquema. Así tenemos:

$$k_s \geq (1 - \frac{p_{cruce} \cdot \delta(S)}{Length - 1}) \cdot (1 - p_{muta})^{o(S)} \tag{5.11}$$

El primer factor mide la probabilidad de que S sobreviva a un cruce, o más exactamente, es decir la probabilidad de que el cruce no ocurra sobre su longitud característica. El segundo factor mide la probabilidad de que no se vea afectado por una mutación. Entonces de la ecuación (5.10) y de la ecuación (5.11) resulta lo siguiente:

$$\overline{\xi}(S, P[t+1]) = \overline{\xi}(S, P[t]) AveFitn(S, P[t]) \cdot (1 - \frac{p_{cruce} \cdot \delta(S)}{Length - 1}) \cdot (1 - p_{muta})^{o(S)} \tag{5.12}$$

Pero como generalmente $p_{muta} \ll 1$, se puede aproximar la ecuación (5.12) a lo siguiente:

$$k_s = (1 - \frac{p_{cruce} \cdot \delta(S)}{Length - 1} - p_{muta} o(S)) \tag{5.13}$$

$$\overline{\xi}(S, P[t+1]) = \overline{\xi}(S, P[t]) AveFitn(S, P[t]) \cdot (1 - \frac{p_{cruce} \cdot \delta(S)}{Length - 1} - p_{muta} o(S)) \tag{5.14}$$

Tal y como se desprende de la ecuación (5.14), esquemas cortos, de bajo orden y con una adaptación superior a la adaptación media, se espera que a medida que evoluciona el Algoritmo Genético, obtengan un incremento exponencial en el número de individuos que se asocian con los mismos.

Desde esta perspectiva, se puede explicar la introducción de los tres operadores del AG de la siguiente manera:

- *Selección*: Se encarga de incrementar geométricamente la presencia de los esquemas aventajados y reducir la presencia de los retrasados. No obstante,

la selección no introduce nuevos esquemas.

- *Cruce*: Permite el intercambio estructurado de información útil (esquemas cortos, bajo orden y aventajados) entre individuos. El cruce es un operador esencial para el eficaz funcionamiento de un AG.

- *Mutación*: Introduce variedad en el juego de esquemas de la población y proporciona un mecanismo de seguridad frente a posibles pérdidas de información valiosa. Visto de esta manera, la mutación es un operador secundario, de ahí que se aplique con bastante menor frecuencia.

Asi respondemos a las dos preguntas formuladas al principio de esta sección, a saber:

- *¿Por qué funcionan los AG?* Porque dan oportunidades de una proliferación exponencialmente creciente a los esquemas, más aptos, y exponencialmente decreciente a los menos aptos.

- *¿Qué procesan los AGs?* En sentido amplio los AGs procesan esquemas y en particular los AGs procesan de modo útil bloques constructivos.

5.5.2. Propiedad del Paralelismo Implícito de los AGs

Como ya quedó dicho, en una generación de *numplob* individuos binarios de tamaño *length* existen entre 2^{length} y $numplob \cdot 2^{length}$ esquemas distintos. Algunos de ellos, mayormente los bloques constructivos, serán procesados de modo útil, esto es, reproducidos exponencialmente y no alterados por los operadores genéticos. Pues bien, el mecanismo que da eficacia a los AGs a pesar de su generalidad se basa en el hecho de que, aunque el AG sólo procesa *numplob* estructuras en cada generación, se puede demostrar que una AG procesa al menos $numplob^3$ esquemas.

Es decir los AG son más robustos que otros métodos de búsqueda ciega, ello se debe a que realizan implícitamente una búsqueda en paralelo. Dicho paralelismo implícito se obtiene sin ningún requerimiento adicional de memoria o de potencia de cálculo, y seguramente los AG son el más destacado ejemplo de explosión combinatoria que actúa de modo beneficioso.

5.6. Templado Simulado (*Simulated Annealing*)

Kirlpatrick, Gelatt y Vecchi proponen en 1983 un procedimiento para obtener soluciones aproximadas a problemas de optimización, llamado *Templado Simulado*. Este procedimiento se basa en una analogía con el comportamiento de un sistema físico al someterlo a un baño de agua caliente. El Templado Simulado ha sido probado con éxito en numerosos problemas de optimización, mostrando gran "habilidad" para evitar quedar atrapado en óptimos locales. Debido a su sencillez de implementación así como a los buenos resultados que iban apareciendo, experimentó un gran auge en la década de los 80's.

5.6.1. Teoría de la Información

En 1948 Claude Shannon publicó un artículo histórico donde propone una teoría para el estudio de la información que después sería adoptada y utilizada por las personas dedicadas al estudio de las ciencias de la computación.

Sabemos que un bit es un dígito binario, que toma dos valores [0,1]. La memoria de las computadoras digitales está construída de una serie de bits unidos entre sí lógicamente para formar *bytes* o *palabras*. En la rama de las matemáticas la teoría de la información, sin embargo, bit es otra cosa.

Supongamos que se produce un cierto suceso e, con cierta probabilidad $P(e)$. Si se observa que e ha tenido lugar, entonces según la teoría de la información, hemos recibido:

$$I(e) = \log_2 \frac{1}{P(e)} bits\ de\ información \tag{5.15}$$

Por ejemplo, supongamos que $P(e) = 1/2$, entonces aplicamos la ecuación (5.16), $I(e)$=$\log_2 2$=1 bit. Por tanto, se puede definir un bit como la cantidad de información que se recibe cuando se especifica una de entre dos alternativas igualmente probables. Ahora si se sabe con seguridad que un suceso va a tener lugar, entonces recibiremos $\log_2 1$=0 bits de información. Esto quiere decir que cuando tengamos la menor cantidad de pistas de que sí un suceso va a pasar o no, entonces recibiremos la mayor cantidad de información.

Ahora sea S una sucesión de acontecimientos $s_1, \ldots, s_n$ cada uno con probabilidad $P(s_1), \ldots, P(s_n)$. Definimos $I(S)$ como una variable aleatoria discreta y su esperanza $E[I(S)]$ representa la "información promedio" que aporta la observación de S. Entonces tenemos que:

La cantidad de información recibida viene dada por:

$$I(s_i) = \log_2 \frac{1}{P(s_i)} \tag{5.16}$$

y la cantidad media de información:

$$\langle I(s_i) \rangle = \sum_{i=1}^{n} P(s_i) I(s_i) = \sum_{i=1}^{n} P(s_i) \log_2 \frac{1}{P(s_i)} \tag{5.17}$$

La ecuación (5.17) se define como la entropía de S, $H(S)$ en analogía con el concepto físico que mide la cantidad de desorden de un sistema termodinámico (a mayor entropía, mayor desorden en el sistema).

5.6.2. Conceptos de Mecánica Estadística

La mecánica estadística es la rama de la física que estudia sistemas con gran número de partículas tales no podemos conocer las caracterísicas de dichos

sistemas estudiando cada partícula de una manera individual. En general, estas características las podemos conocer estudiando una colección finita de sistemas idénticos. A esta colección se le llama una agrupación o *ensamble* y está definido por el promedio de las partículas que constituyen al sistema, así como por las variaciones estadísticas sobre el promedio.

Así, podemos definir la energía promedio del sistema como sigue:

$$E = \sum_{r=1}^{n} E_r P_r \tag{5.18}$$

donde, n es el número de estados, P_r expresa la probabilidad de tener algún estado con energía E_r. Dada una temperatura T, se puede asegurar experimentalmente, que un *ensamble* con estas características, alcanza su equilibrio térmico (la energía no cambia significativamente), sólo cuando la distribución de las probabilidades de estar en estados con energía E_r es:

$$P_r = \frac{1}{Z} e^{-\frac{E_r}{k_b T}} \tag{5.19}$$

donde T es la temperatura en grados Kelvin, k_b es la constante de Boltzmann ($k_b = 1{,}39 \cdot 10^{-23} J/K$) y Z es la función de partición, que corresponde al factor de normalización de la distribución. Z está dada por:

$$Z = \sum_{r} e^{-\frac{E_r}{k_b T}} \tag{5.20}$$

En esta distribución, los estados con menor energía tienen una mayor probabilidad de ocurrir y al disminuir la temperatura T, la probabilidad se concentra en los estados de menor energía. La entropía del sistema se define esencialmente de la misma manera que en la teoría de información.

5.6.3. Funcionamiento del Templado Simulado

Nuestra intuición nos dice que un trozo de sustancia a alta temperatura posee un estado de energía más alto que el de otro trozo idéntico a una temperatura menor. Supongamos que se desea reducir la energía de la sustancia al menor valor posible . Bajar simplemente la temperatura al cero absoluto no asegurará necesariamente que la sustancia se encuentre en su configuración energética más baja posible. El *Templado Simulado* es un técnica de optimización para resolver problemas de optimización combinacional. Asumimos una función de costo $C(x)$ que nosotros queremos minimizar. Los algoritmos del *Templado Simulado* son similares a los algoritmos *Hill Climbing*, pero muchas veces ellos aceptan una configuración con un costo alto. La posibilidad de ir a una configuración de costo alto, depende de una parámetro T, llamado temperatura . Inicialmente esta temperatura es alta y la posibilidad de que un costo se incremente es alta. En las iteraciones siguientes la temperatura es decrementada de acuerdo a un procedimiento de actualización. El *Templado Simulado* clásico comienza en una configuración arbitraria x. Entonces éste genera aleatoriamente una configuración vecindad y. Si $C(y) \leq C(x)$ nosotros escogemos la configuración y. Si $C(y) > C(x)$ nosotros aceptamos la nueva configuración con la probabilidad $e^{C(y)-C(x)/t}$. Bajo un apropiado procedimiento de actualización de T este algoritmo converge al mínimo global.

5.7. Aproximación Funcional Usando Algoritmos Genéticos

Continuando con el mismo patrón de los capítulos anteriores vamos a proceder a probar primero nuestro Algoritmo Genético para hallar los parámetros del polinomio NARMAX, luego vamos a validar el aproximador funcional hallado con diferentes entradas que no han sido utilizadas para su entrenamiento.

El diagrama de bloques que se va a seguir para la identificación parama'trica del aproximador funcional es mostrada en la figura 5.11.

Los elementos del polinomio NARMAX que vamos a utilizar en nuestro AG va han ser:

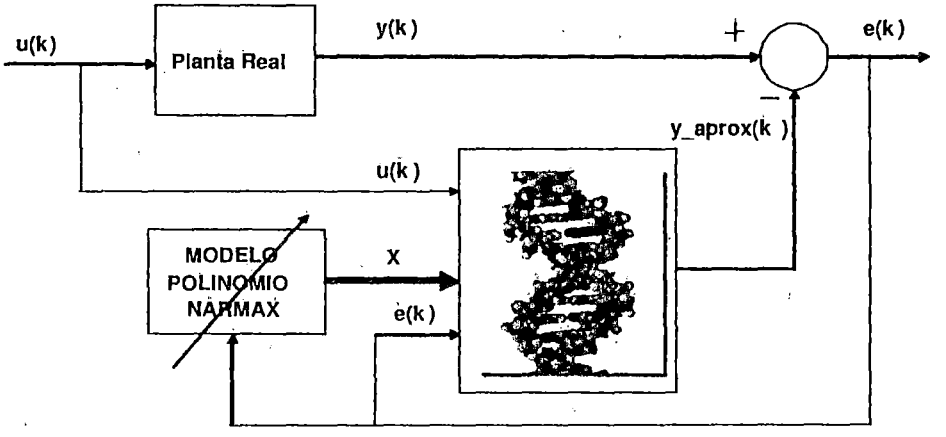

Figura 5.11: Diagraba de bloques de la estructura a utilizar para la identificación mediante aproximadores funcionales.

$$\begin{aligned}\widehat{y}_{k+1} = \; & a_0 + a_1\widehat{y}_k + a_2\widehat{y}_{k-1} + a_3\widehat{y}_k + a_3u_k + a_4u_{k-1} + \ldots \\ & + a_1^2\widehat{y}_k^2 + a_1a_2\widehat{y}_k\widehat{y}_{k-1} + a_1a_3\widehat{y}_ju_j + a_1a_4\widehat{y}_ku_{k-1} + \ldots \\ & + a_2^2\widehat{y}_{k-1}^2 + a_2a_3\widehat{y}_{k-1}u_k + a_2a_4\widehat{y}_{k-1}u_{k-1} + a_3^3u_k^2 + \ldots \\ & + a_3a_4u_ku_{k-1} + a_4^2u_{k-1} + e_k + e_{k-1} \end{aligned} \tag{5.21}$$

Como se puede apreciar en este caso hemos incrementado el número de términos del polinomio NARMAX, esto es porque si alguno de los elementos sobrase (o no es influyente) el AG le dará a su coeficiente respectivo un valor cercano a cero.

Como se ha estado procediendo en los capítulos anteriores vamos a encontrar primero el aproximador funcional para el caso de una planta lineal de segundo orden luego lo haremos para la planta no lineal de Narendra.

El diagrama de flujo que vamos a seguir es el que se muestra en la figura 5.12.

Los parámetros que vamos a utilizar para implementar nuestro AG serán los que se muestran en la tabla 5.3:

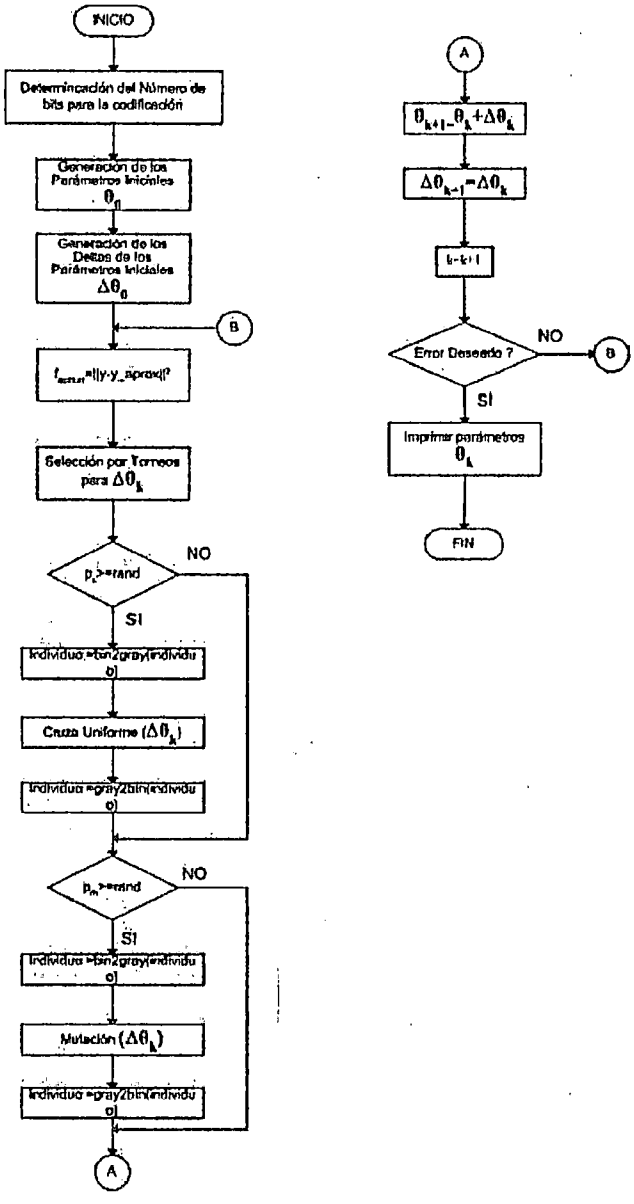

Figura 5.12: Diagrama de flujo para el aproximador funcional usando Algoritmos Genéticos.

Como se puede apreciar en la tabla anterior el número de pobladores con el cual se ha trabajado es 4, permitiéndonos una mayor velocidad de ejecución del AG. Los límites $\Delta\theta$ son los cuales van a variar la pequeñas modificaciones de los parámetros (ver figura 5.12). La función de costo va ser el error cuadrático entre la salida real y la salida aproximada:

Tabla 5.3: Parámetros del AG utilizado para identificación paramétrica del Modelo NARMAX

Número de Generaciones (NumGen)	21
Número de Pobladores (NumPlob)	4
Número de bits a codificar (Longcromo)	40
Probabilidad de Cruce (p_{cruce})	0.9
Probabilidad de Mutación (p_{muta})	0.3
Límite Inferior de los $\Delta\theta$	-0.0001
Límite Superior de los $\Delta\theta$	0.0001

$$\int_{costo} = (y_k - \widehat{y}_k)^2 \tag{5.22}$$

Ahora vamos a hacer evolucionar nuestros parámetros del modelo NARMAX propuesto, según la función de costo propuesta anteriormente. Para ello someteremos como ya mencionamos a nuestra planta de segundo orden a una entrada aleatoria (ver figura 5.13):

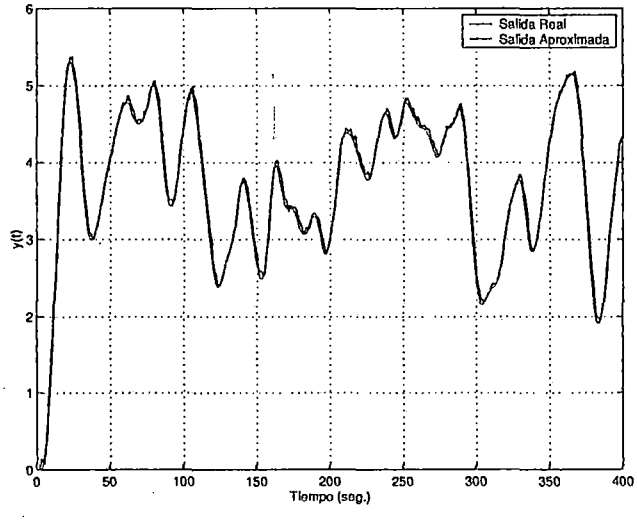

Figura 5.13: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada aleatoria (sistema lineal).

Una vez que hemos obtenido los parámetros de nuestro polinomio NARMAX a través de nuestro AG, vamos a validar el modelo sometiendo el sistema aproximado a entrada escalón, senoidal y cuadrada (ver figura):

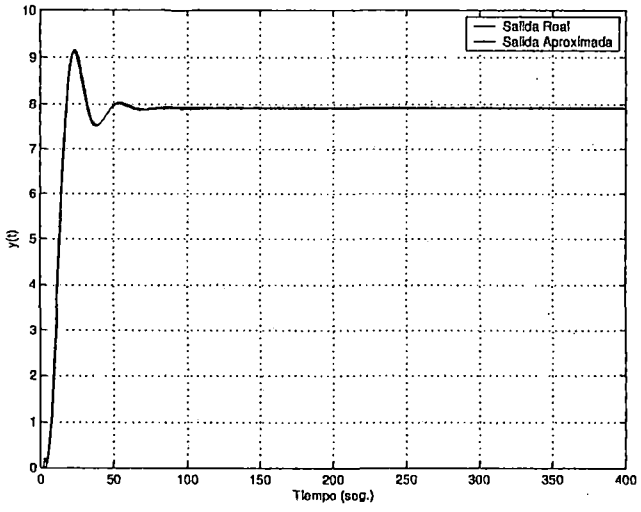

Figura 5.14: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada escalón (sistema lineal).

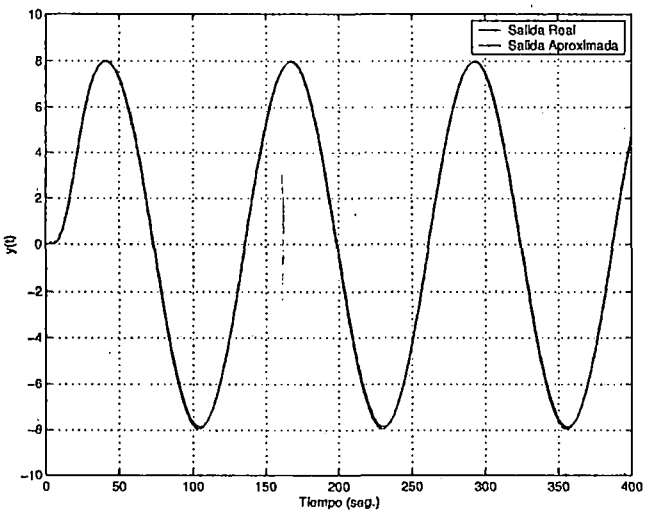

Figura 5.15: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada senoidal (sistema lineal).

De la figuras 5.14, 5.15, 5.16, deducimos la tabla 5.4 de errores:

Como se observa de la tabla 5.4 el algoritmo genético a encontrado los parámetros con un error aceptable (frente a los diferentes tipo de entrada). El programa de esta parte se encuentra en el *Apéndice A* con el nombre de *GA_lineal.m*.

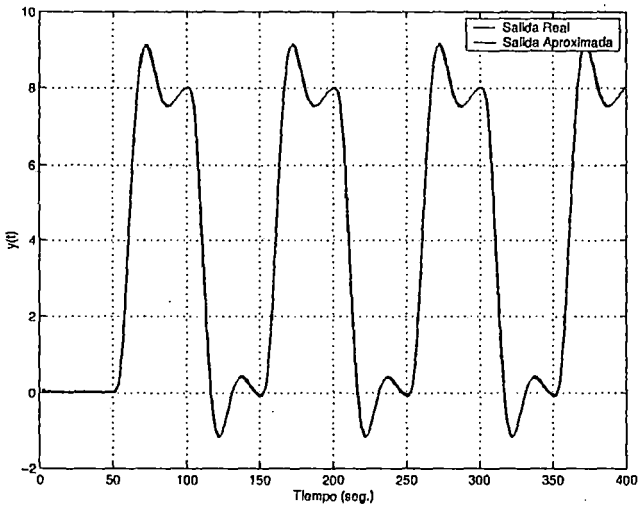

Figura 5.16: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada cuadrada (sistema lineal).

Tabla 5.4: Errores del aproximador funcional usando AG frente a diferentes tipos de entradas - sistema lineal

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N})\cdot 100\,\%$
Escalón	0.1615 %
Senoidal	1.6050 %
Cuadrada	1.8584 %

Ahora vamos a comprobar la eficiencia de nuestro AG a través de la planta no lineal de Narendra [8]:

$$y_{k+1} = \frac{y_k}{(1+y_k)^2} + u_k^3 \tag{5.23}$$

Al igual que en los casos anteriores someteremos para la evolución del AG a una entrada aleatoria tanto al sistema real como al sistema aproximado (ver figura 5.17).

Una vez que se logró la evolución del AG para encontrar los parámetros del polinomio NARMAX someteremos al sistema real y al sistema aproximado a diferentes tipos de entrada (escalón, senoidal y cuadrada) para comprobar si el

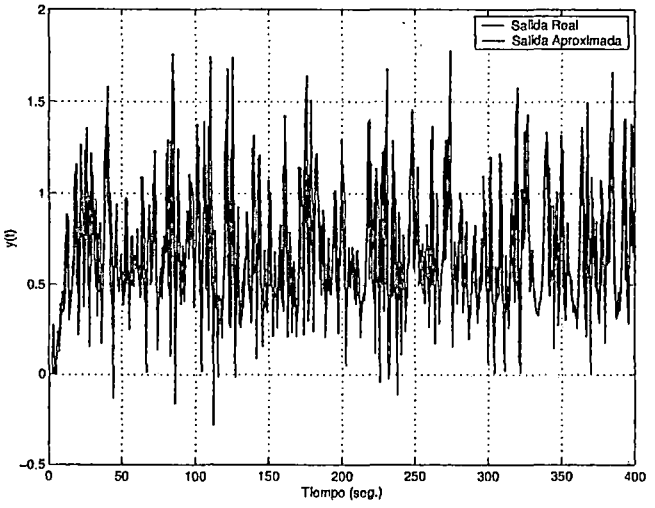

Figura 5.17: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada cuadrada (sistema no lineal).

sistema hallado cumple con seguir la dinámica completa del sistema (ver figuras 5.18, 5.19, 5.20).

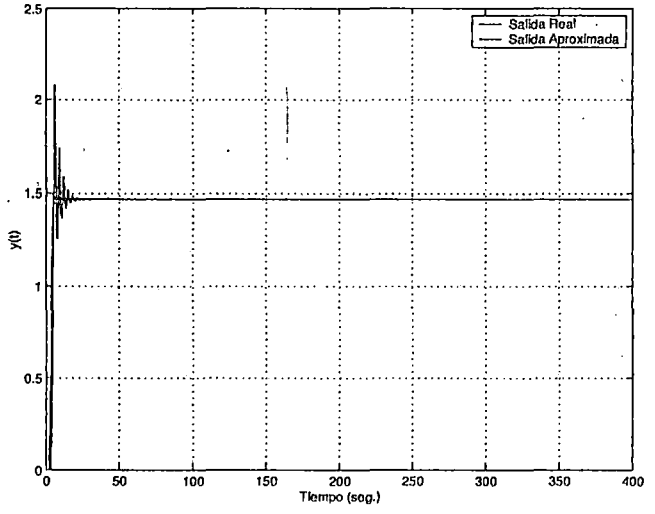

Figura 5.18: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada escalón unitario (sistema no lineal).

De la figuras 5.18, 5.19, 5.20, deducimos la tabla 5.5:

Como podemos apreciar de la tabla 5.5 el aproximador funcional encon-

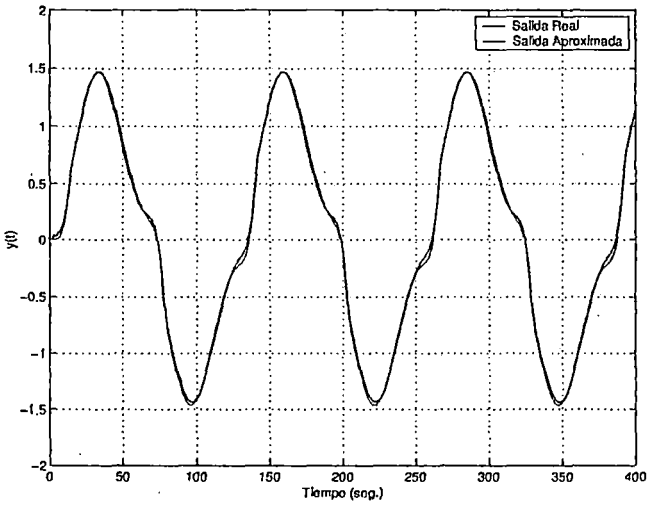

Figura 5.19: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada senoidal (sistema no lineal).

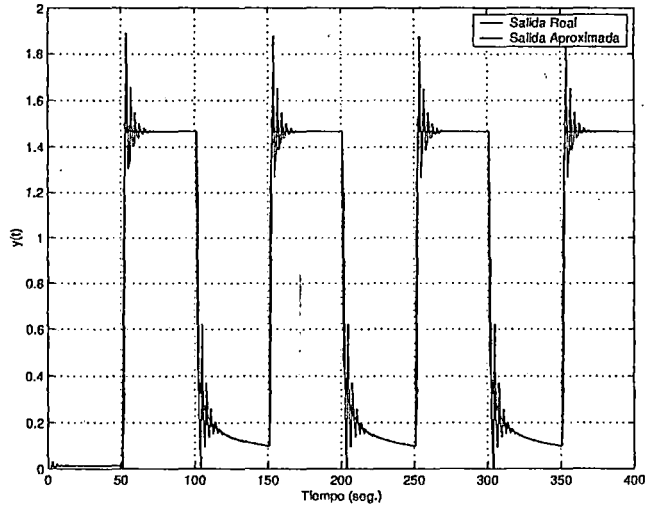

Figura 5.20: Respuesta del sistema real y del sistema aproximado usando AG frente a una entrada cuadrada (sistema no lineal).

trado, cumple con seguir las características dinámicas del sistema no lineal con un error aceptable, por lo tanto el AG ha cumplido su función. Pero debemos precisar que el tiempo de convergencia del AG fue de aproximadamente 12 minutos[9]. El programa de esta parte se encuentra en el *Apéndice A* con el nombre de *GA_no_lineal.m*.

[9]Con un procesador Pentium IV, 1.2GHZ, 128K.

Tabla 5.5: Errores del aproximador funcional usando AG frente a diferentes tipos de entradas - sistema no lineal

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N}) \cdot 100\,\%$
Escalón	1.2435 %
Senoidal	6.9051 %
Cuadrada	1.4952 %

5.8. Aproximación Funcional Usando el Algoritmo Híbrido *SA-GA*

Ahora vamos a implementar un híbrido entre un Algoritmo Genético y el Templado Simulado. Este último algoritmo lo introduciremos al AG para proporcionar una convergencia más rápida al AG.

El diagrama de flujo que vamos a utilizar es el que se muestra en la figura 5.21.

Los parámetros que vamos a utilizar para hacer evolucionar nuestro algoritmo SAGA son los mismos que hemos utilizado para evolucionar nuestro AG, pero con la única diferencia que vamos a necesitar una menor cantidad de generaciones para lograr que la SAGA llegue a un error aceptable.

Como se ha estado procediendo hasta el momento vamos a someter primero a la planta de segundo orden a una entrada aleatoria para el correspondiente entrenamiento del nuestro algoritmo híbrido SAGA. Sometiendo a una entrada aleatoria a nuestro sistema real y nuestro sistema aproximado (utilizaremos el mismo polinomio utilizado en el ejemplo anterior) obtenemos la figura 5.22.

Ahora para comprobar si el modelo hallado cumple con seguir la dinámica del sistema vamos a proceder a someter al sistema aproximado a diferentes entradas (escalón, senoidal, cuadrada).

Ahora con las figuras 5.23, 5.24, 5.25 creamos la tabla 5.6.

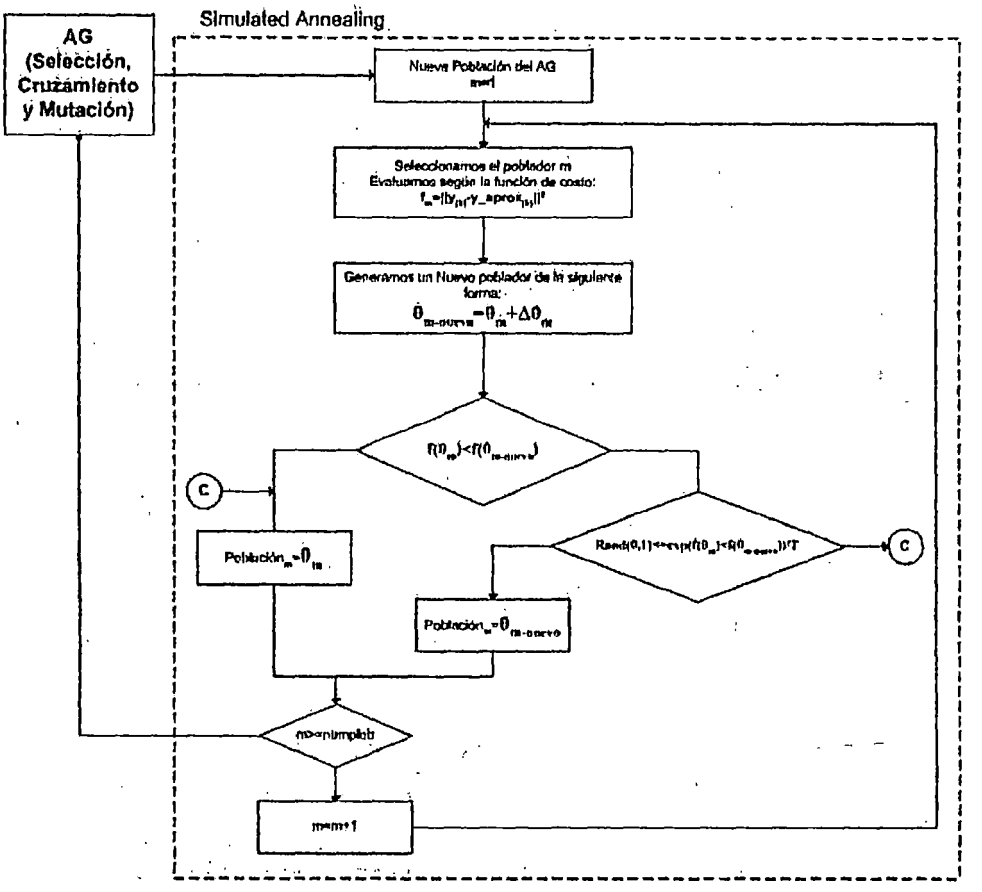

Figura 5.21: Diagrama de flujo para el aproximador funcional usando el híbrido algoritmos genéticos y templado simulado (SAGA).

Tabla 5.6: Errores del aproximador funcional usando el algoritmo SAGA frente a diferentes tipos de entradas - sistema lineal.

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N})\cdot 100\,\%$
Escalón	1.4863 %
Senoidal	1.3631 %
Cuadrada	0.0662 %

Como se observa de la tabla 5.7 el algoritmo SAGA a encontrado los parámetros óptimos del polinomio NARMAX y además el aproximador funcional cumple con seguir la dinámica del sistema con un error aceptable. El programa de esta parte se encuentra en el *Apéndice A* con el nombre de *SAGA_lineal.m*.

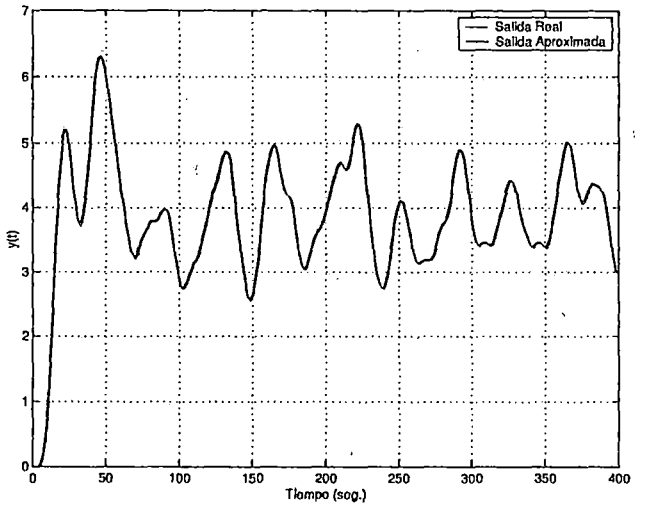

Figura 5.22: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada aleatoria (sistema lineal).

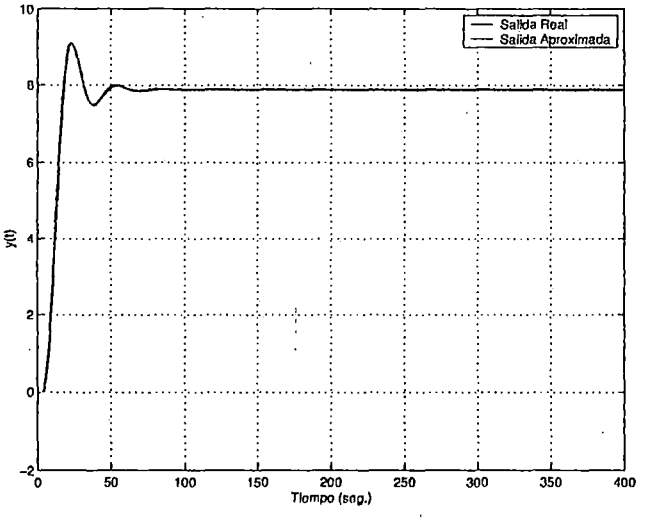

Figura 5.23: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada escalón (sistema lineal).

Ahora vamos a probar la eficacia del algoritmo SAGA con la planta no lineal que hemos estado trabajando durante la presente tesis[8]:

$$y_{k+1} = \frac{y_k}{(1+y_k)^2} + u_k^3 \tag{5.24}$$

Someteremos para la evolución de la SAGA, tanto al sistema real como al

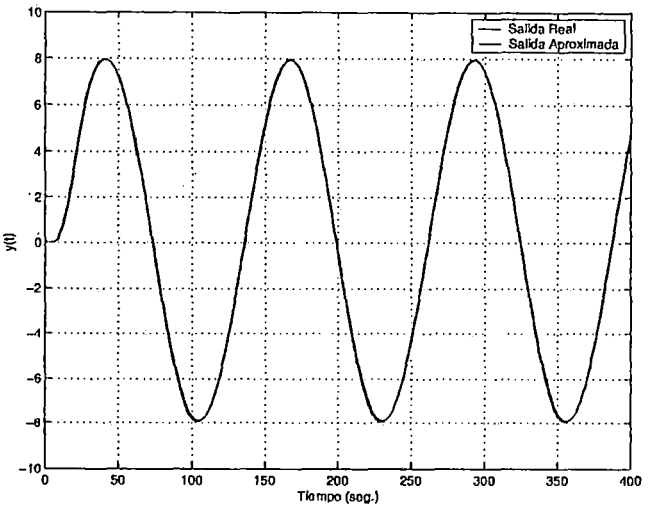

Figura 5.24: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada senoidal (sistema lineal).

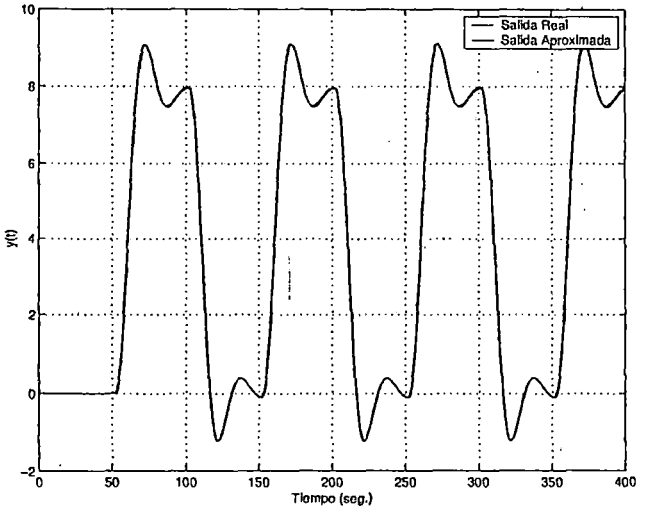

Figura 5.25: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada cuadrada (sistema lineal).

sistema NARMAX a una entrada aleatoria (ver figura 5.26).

Ahora que hemos hecho evolucionar nuestra SAGA someteremos a diferentes entradas (escalón, senoidal y cuadrada) para comprobar si el aproximador funcional cumple con seguir la dinámica del sistema (ver figuras 5.23, 5.24, 5.29).

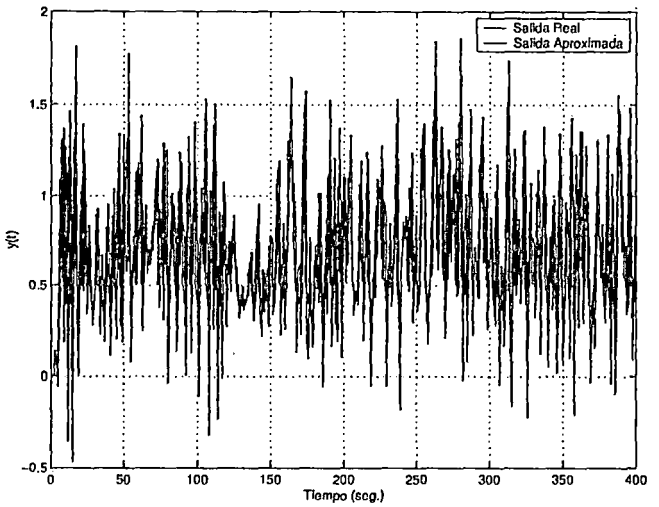

Figura 5.26: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada cuadrada (sistema no lineal).

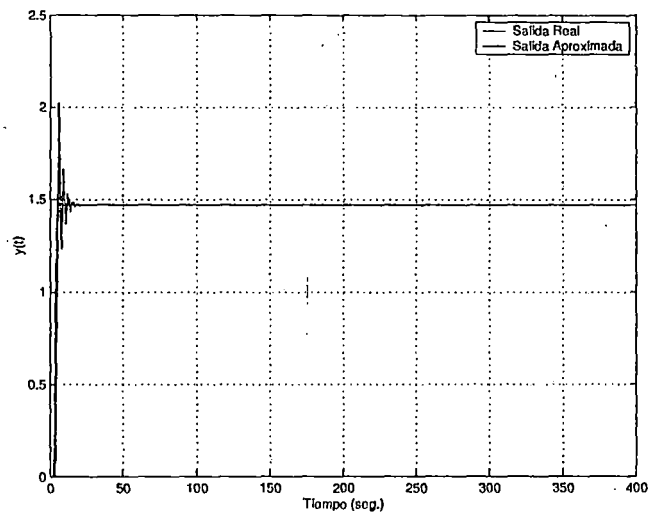

Figura 5.27: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada escalón (sistema no lineal).

Ahora con las figuras 5.27, 5.28, 5.29 creamos la tabla 5.7.

Como se observa en la tabla 5.7 el aproximador hallado por el algoritmo SAGA cumple con seguir la dinámica del sistema frente a diferentes tipos de entradas que no han sido utilziados para su entrenamiento por lo tanto el aproximador funcional hallado es aceptable. El programa de esta parte se encuentra en

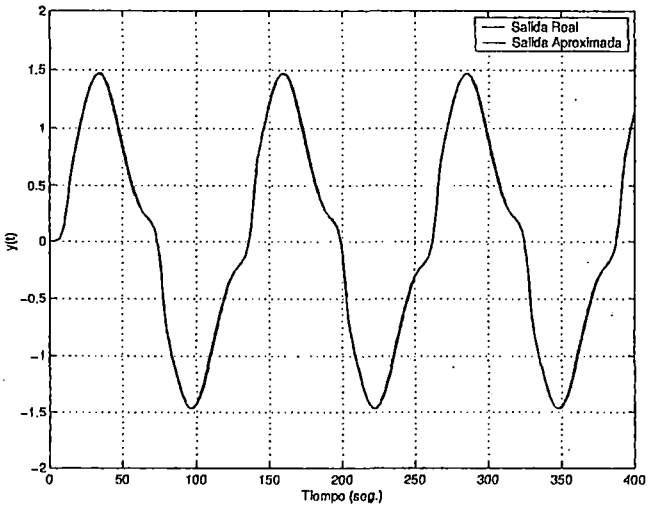

Figura 5.28: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada senoidal (sistema no lineal).

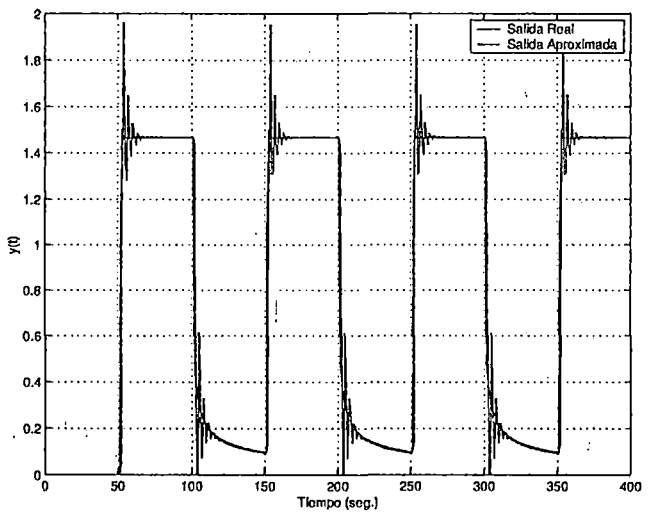

Figura 5.29: Respuesta del sistema real y del sistema aproximado usando el algoritmo SAGA frente a una entrada cuadrada (sistema no lineal).

el *Apéndice A* con el nombre de *SAGA_no_lineal.m*.

Tabla 5.7: Errores del aproximador funcional usando el algoritmo SAGA frente a diferentes tipos de entradas - sistema no lineal

Entrada	$Error(\%)=(\frac{y_N-\widehat{y}_N}{y_N}) \cdot 100\,\%$
Escalón	1.0289 %
Senoidal	2.3883 %
Cuadrada	1.9031 %

CAPÍTULO VI
IDENTIFICACION PARAMÉTRICA *"ON-LINE"* APLICADO A UN MOTOR DC

6.1. Principios y Funcionamiento del Motor DC

Los motores DC son pequeñas máquinas especialmente diseñadas para control de posicionamiento. Aunque el principio de funcionamiento es el de una máquina de corriente contínua convencional con excitación independiente, su forma constructiva está adaptada para obtener un comportamiento dinámico, rápido y estable y un par de arranque importante.

Por lo general, el inductor se encuentra en el estator y puede ser o bobinado o de imán permanente. El inducido, alojado en el rotor, se suele contruir de forma que presente una inercia mínima. Constructivamente se diferencia básicamente en la forma del rotor (ver figura 6.1). Las más habituales son:

- Rotor alargado.
- Rotor en forma de Cesta.
- Rotor de Disco.

Los dos primeros suelen tener un colector clásico de forma cilíndrica, mientras que en los de disco suele estar dispuesto en forma radial. El rotor de dicho motores de disco puede estar construído de cable rígido con soporte de resina, dando una inercia propia extremadamente baja.

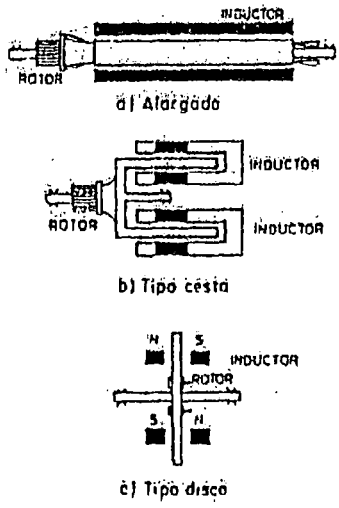

Figura 6.1: Formas constructivas del rotor en motores DC.

Los parámetros escenciales de un motor DC son los siguientes[7]:

η	velocidad *(r.p.m)*.
E_i	fuerza electromotriz del inducido *(voltios)*.
U_i	tensión del inducido *(voltios)*.
I_i	corriente del inducido *(amperios)*.
ϕ_i	flujo inductor o excitación, en caso de motores con bobinado de excitación es proporcional a la corriente de la bobina indcutora.
T_i	Constante de tiempo eléctrica L_i/R_i.
C_m	par motor $(metros \cdot newton)$.
P	potencial *(vatios)*.
K_e	constante eléctrica $(r.p.m./voltio)$. Su valor se puede obtener de la relación $\eta_i/E_{nominal}$.
K_m	constante mecánica, medida en $metros \cdot newton/amperio$.

Las relaciones de funcionamiento entre dichos parámetros para un motor DC con excitación por imanes permanentes o excitación independiente y constante son las siguientes:

$$\eta = \frac{K}{\phi_e} E_i = K_e(U_i - R_i I_i) \tag{6.1}$$

$$C_m = K_m I_i \tag{6.2}$$

$$P = E_i I_i = 0{,}1047 C_m \eta \tag{6.3}$$

A partir de estas relaciones [7] se deduce que el control de la velocidad del motor puede hacerse regulando la tensión del inducido y compensando la caída de tensión $R_i I_i$ y el control de par requiere regular la corriente de inducido. En ambos casos debe mantener constante el flujo de excitación.

Para caracterizar el comportamiento dinámico de un accionamiento debemos obtener el diagrama de bloques del motor más la carga, supuesta ésta con un par resistente C_m, una inercia J, y un rozamiento vizcoso f. La figura 6.2 muestra un esquema del accionamiento junto con el diagrama de bloques completo donde J_t representa la inercia total del rotor del propio motor más la de la carga.

Dado que el rozamiento viscoso suele ser pequeño frente a la inercia, el sistema mecánico se comporta prácticamente como un integrador puro. En tal caso, simplificando el diagrama en lazo cerrado, se obtiene la siguiente función de transferencia:

$$N(s) = [U_i - C_r \frac{R_i}{K_m}(1 + T_d s)] \frac{\frac{1}{K_e}}{1 + T_m s + T_m T_e s^2} \tag{6.4}$$

donde T_m es la denominada constante de tiempo mecánica, cuyo valor es:

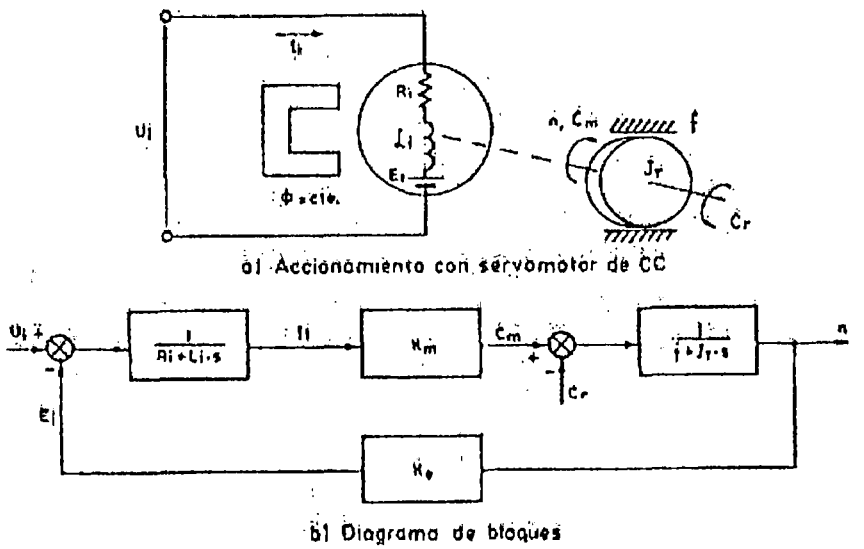

Figura 6.2: Modelo de bloques de un motor DC.

$$T_m = \frac{J_t R_i}{K_m K_e} \tag{6.5}$$

Este modelo de motor permite determinar el comportamiento dinámico del motor más carga y elegir el controlador más conveniente para sistemas de control de velocidad o posición.

6.2. Diagrama General para la Identificación *on line*

Para poder obtener la salida del motor y_k vamos a proceder de la siguiente manera. Primero realizaremos la etapa de sensado en la cual tomamos una salida del encoder (línea A o línea B), esta salida está conectada al PIC16F876 (Pin RC0). Este PIC se encargará de contar los pulsos del encoder y transmitirlos a través de la comunicación serial establecida entre la PC y el PIC. Luego la PC se encargará de procesar esa lectura y mandará la señal de entrada u_k (que vendría a ser la señal PWM) de ocho bits a través de su puerto paralelo hacia otro PIC 16F876, el cual se encargará de procesar esa señal y arrojar por su pin RC2 la

señal PWM que irá a la etapa de potencia. En la etapa de potencia la señal de PWM será amplificada a través del LMD18200, el cual mandará su señal de salida al motor DC. En la figura 6.3 se visualiza el esquema general del funcionamiento para la identificación de sistemas.

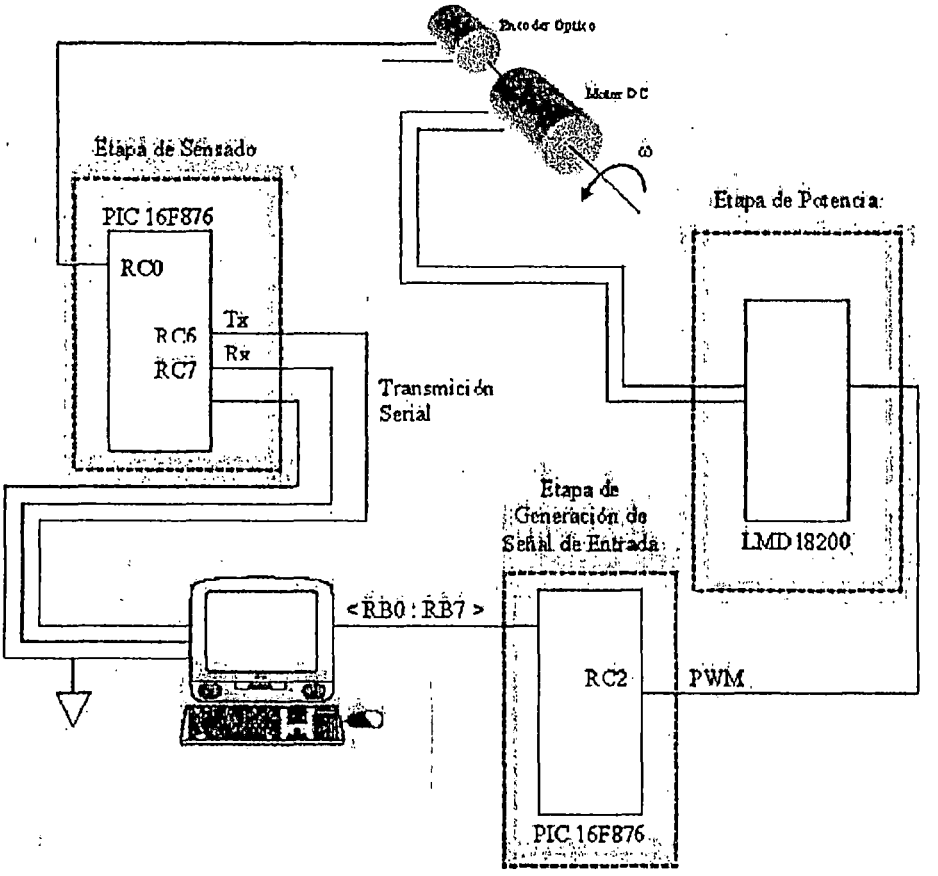

Figura 6.3: Diagrama general para la identificación *on line* de un motor DC.

6.3. Etapa de Sensado

Para poder tener la salida del motor y_k es necesario contar con un sensor que me permita leer cuanto ha girado el motor. Eso se logra a través de un sensor óptico llamado encoder. El encoder es un disco concéntrico al eje del motor y que gira junto con él. Este disco tiene una serie de agujeros alrededor de su perímetro, en nuestro caso el encoder tiene 504 agujeros. En este disco viene colocado un sistema optoacoplador, el cual consiste de un emisor de luz y un receptor, que

cada vez que le llega ese haz de luz, emite un pulso por su respectiva salida.

Así este sistema optoacoplador se coloca en el disco, de tal forma que los haces de luz pasen a través de los agujeros del disco, cuando esto ocurre el receptor mandará una pulso. Pero debemos aclarar que este sistema también da una señal que esta desfasada con respecto a la primera 90°, es decir las dos señales están en cuadratura. Para poder ayudarnos mejor mostramos la figura 6.4, en el cual observamos las ondas en cuadratura y una tercera. Esta última se implementa para contar el número de vueltas, es decir manda un pulso cada vez que el eje ha girado una vuelta completa.

Del párrafo anterior podemos deducir la siguiente ecuación:

$$\text{Presición} = \frac{N^{o}\ de\ pulsos\ en\ una\ vuelta}{360^{o}}$$

En nuestro caso la presición de nuestro encoder es de 1.5278 pulsos/°. Estos tipos de encoders que poseen sólo dos canales (o a veces el canal de Z el cual emite un pulso cada vez que se ha completado una vuelta) se llaman *Encoders Incrementales.*

Una vez que ya tenemos listo la señal del motor, ahora tenemos que contar esos pulsos y adecuarlos para enviarlos a la PC. Esto lo hacemos a través de un microcontralor PIC, este μC tiene la gran ventaja que su programación es fácil y su costo relativamente bajo (PIC de la gama media: $ 12.00). Además permiten programarlos y borrar lo que hemos grabado en ellos varias veces. El PIC que hemos elegido es el PIC16F876 (gama media) (ver figura 6.5) debido a que nos ofrece una serie de ventajas que mencionaremos a continuación.

La primera es que nos permite contar los pulsos que le llegan a la patita RC0 sin necesidad de programar un código determinado, sólo es necesario setear los parámteros adecuados del TMR1 (este microcontrolador posee tres relojes internos TMR0, TMR1, TMR2), en modo de un contador asíncrono. Además posee

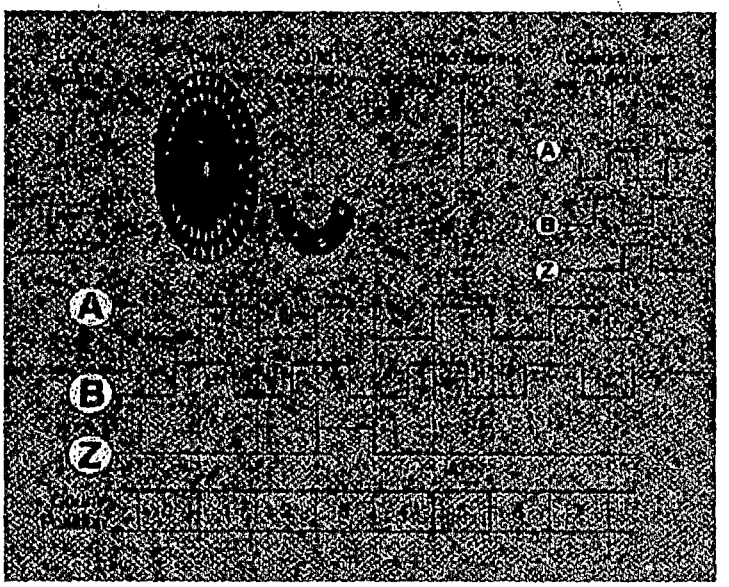

Figura 6.4: Esquema que muestra el funciomamiento del encoder y de las señales de salida.

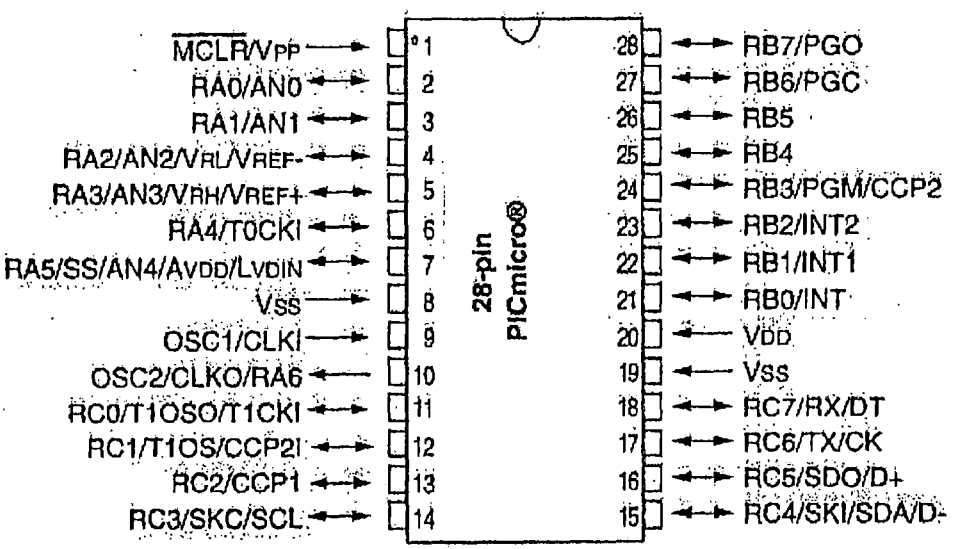

Figura 6.5: Familia del PIC16F876.

una salida de PWM (Modulación por Ancho de Pulso) la cual su programación es relativamente fácil. Por esos dos motivos es que se ha elejido este PIC.

Una vez que ya tenemos el PIC, lo seteamos para que cada vez que le llegue un pulso por su patita RC0 este aumente su contador interno. El sensado de los pulsos del encoder lo haremos cada 40 mseg, esto quiere decir que el contador del TMR1 va ir aumentando cada vez que llegue un pulso a la patita RC0 y cuando ya paso el periódo de los 40 mseg, entonces se procederá a descargar ese contador a un registro (cuando realicemos esta operación el contador del TMR1 se pondrá a cero para la siguiente cuenta) para luego transmitirlo serialmente (sobre la transimición hablaremos en mas detalle en las sección siguiente).

6.4. Etapa de Generación de la Señal de Entrada

En esta etapa la PC, a través del programa de interface (*identificación.exe*), el cual se explicará con más detalle en la sección correspondiente a la de programación, envía por el puerto paralelo LPT1 el valor correspondiente a la señal u_k codificada en ocho bits hacia otro PIC16F876, llegándole a las patitas del puerto B (RB0:RB7). Este PIC va estar constantemente leyendo este puerto y obteniendo el valor que hay en él, para luego procesarlo y colocarlo en los registros correspondientes al PWM. El valor de la señal de entrada u_k va a estar determinada por el tipo de entrada que se seleccione en el programa.

En la figura 6.8 se puede apreciar el diagrama de flujo que se ha seguido para la programación de esta parte.

6.5. Etapa de Amplificación de Potencia

En esta etapa, se realiza un amplificación de la señal generada por el etapa del PWM, donde el amplificador de potencia será el encargado de alimentar al motor DC con una señal que es proporcional a la generada por el PWM, generada por el PIC. El amplificador de potencia que hemos elegido es el LMD18200 (ver figura 6.10). Este amplificador cuenta con una configuración puente H tipo MOSFET's el cual tiene una lógica digital que permite una entrada PWM de tipo TTL y otra que indica la dirección del movimiento del motor. Esta lógica digital nos permite proteger a los transistores MOSFET's, ya que la configuración del

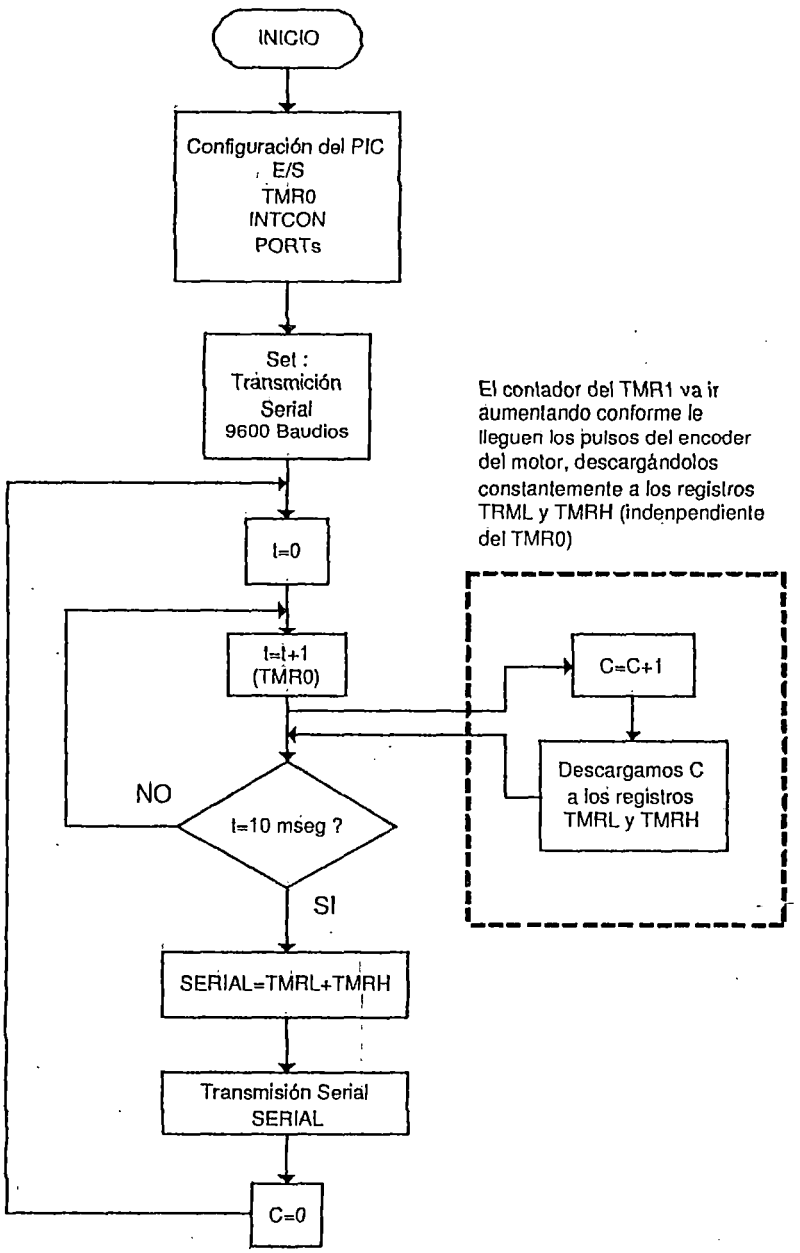

Figura 6.6: Diagrama de flujo de la lectura del encoder y del envío a la PC a través del puerto serial.

puente H no puede activar sus dos fases a la vez. También tenemos que destacar que este amplificador posee una protección de sobrecarga térmica, el cual se activa cuando se sobrepasa su limite de corriente (3 amperios), abriendo su circuito.

En las figuras 6.10 y 6.11 se puede observar la estructura física del am-

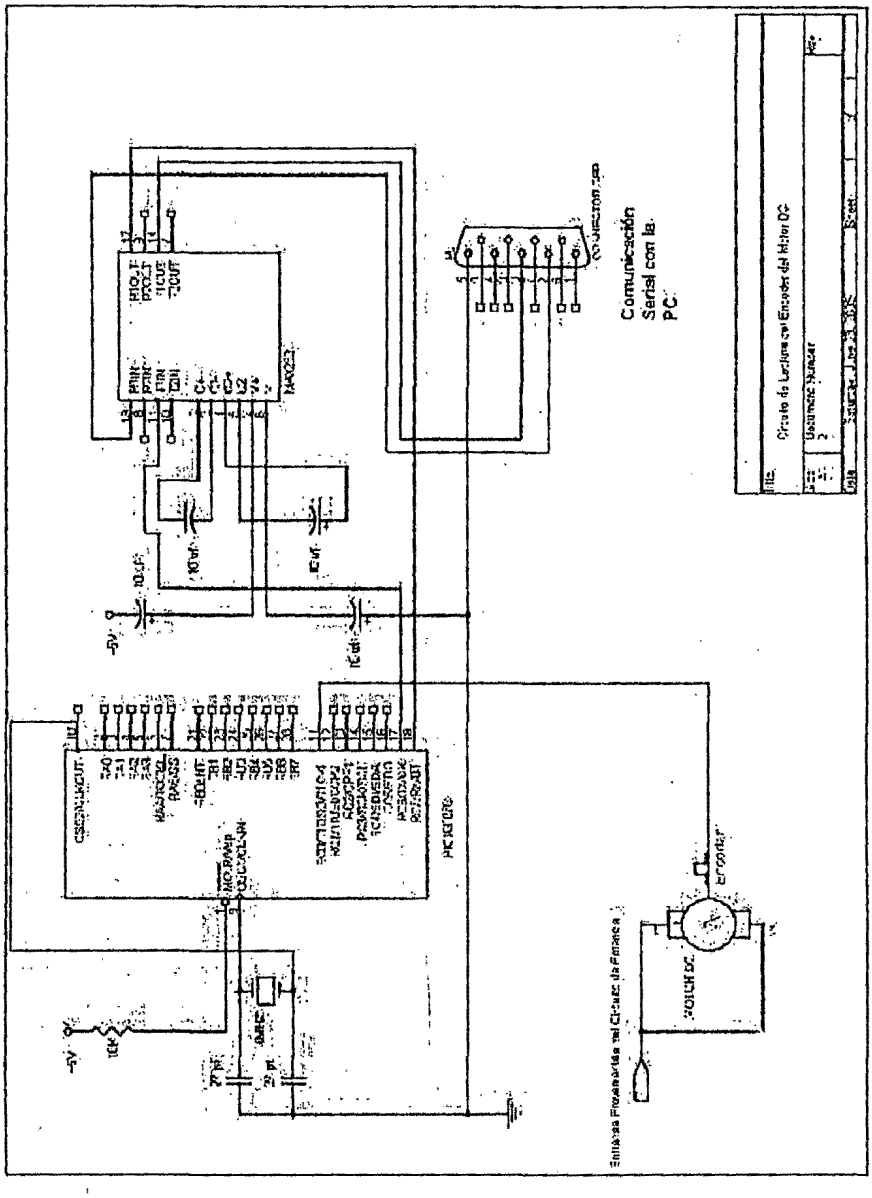

Figura 6.7: Circuito de la etapa de sensado del encoder del motor DC.

plificador LMD18200 y su funcionamiento a través de su diagrama de bloques respectivo.

6.6. Etapa de Comunicación

Esta parte corresponde a la comunicaciones establecidas entre el primer PIC16F876 y la PC (destinado para la transmisión de la lectura del encoder) que

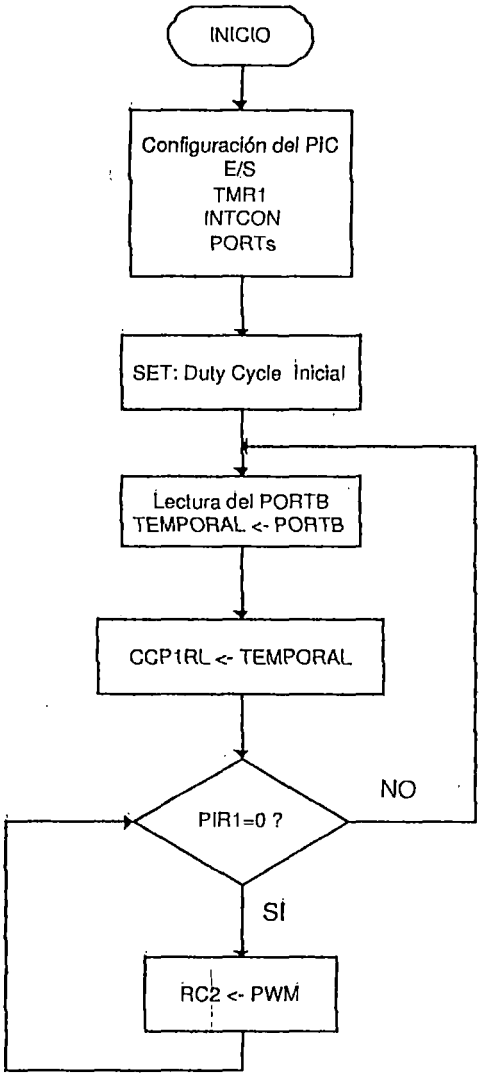

Figura 6.8: Diagrama de flujo de la generación de la señal de entrada.

es una transmisión serial y la establecida entre la PC y el segundo PIC16F876 (destinado a la transmisión de *duty cycle* correspondiente al PWM) que es una transmisión paralela.

6.6.1. Etapa de Comunicación Serial

Los PIC16F876 contienen un módulo MSSP con dos puertas para la comunicación serie síncrona, o sea con señal de reloj. Además, también disponen de un módulo USART capaz de soportar la comunicación serie síncrona y asíncrona.

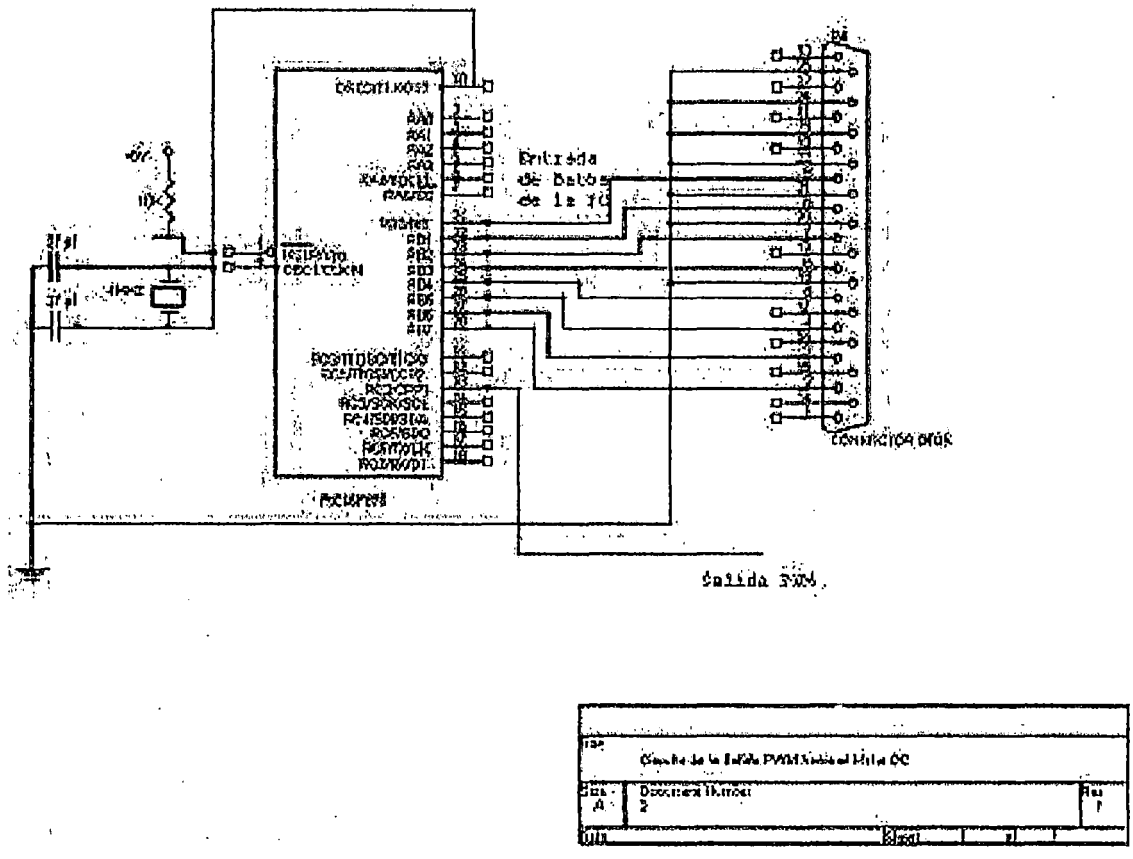

Figura 6.9: Circuito de acondicionamiento de la señal proveniente de la PC al circuito de potencia a través de un PIC.

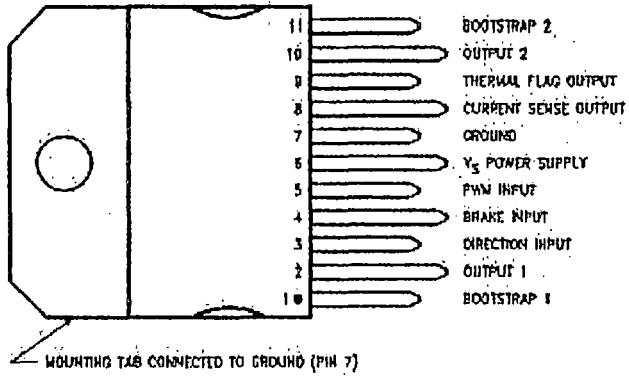

Figura 6.10: El Amplificador de potencia LMD18200.

De los dos modos de funcionamiento del USART, la comunicación serie asíncrona es la más utilizada. El PIC incorpora el hardware para comunicarse vía RS-232 con la PC.

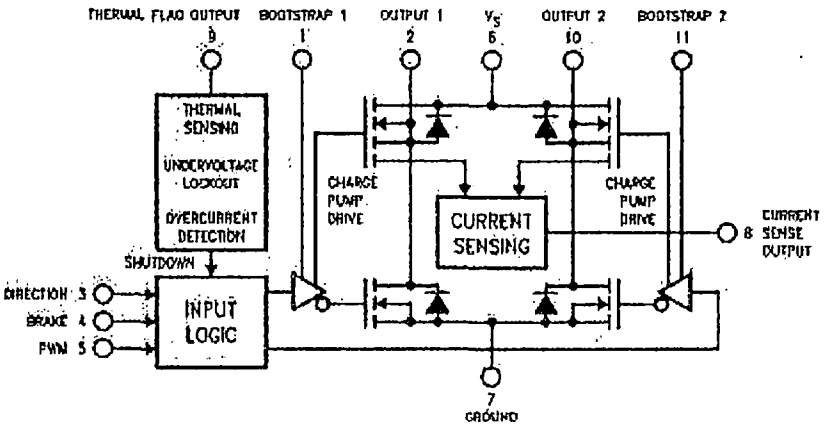

Figura 6.11: Diagrama de bloques del funcionamiento del LMD18200.

La transferencia de informaciones realiza sobre dos líneas TX (transmisión) y RX (recepción), saliendo y entrando los bits por dichas líneas al ritmo de una frecuencia controlada internamente por el USART. Las líneas de comunicación son las dos de más peso del puerto C: RC6/TX/CK y RC7/RX/DT.

En la forma de comunicación serie es común usa la norma RS-232-C, donde cada palabra de información o dato se envía independientemente de los demás. Suele constar de 8 o 9 bits y van precedidos por un bit de START (bit de inicio) y detrás de ellos se coloca un bit de STOP (bit de paro), de acuerdo con las normas del formato estándar NRZ (NonReturn-to-Zero). Los bits se transfieren a una frecuencia fija y normalizada.

Protocolo Estándar

En el protocolo estándar se transfieren 10 bits, un bit de inicio, 8 bits de datos y un bit de paro, formando una palabra de 10 bits. La eficiencia está dada por:

$$eficiencia = \frac{\text{Número de Bits de Datos}}{\text{Número Total de Bits}} \cdot 100 = x\,\%$$

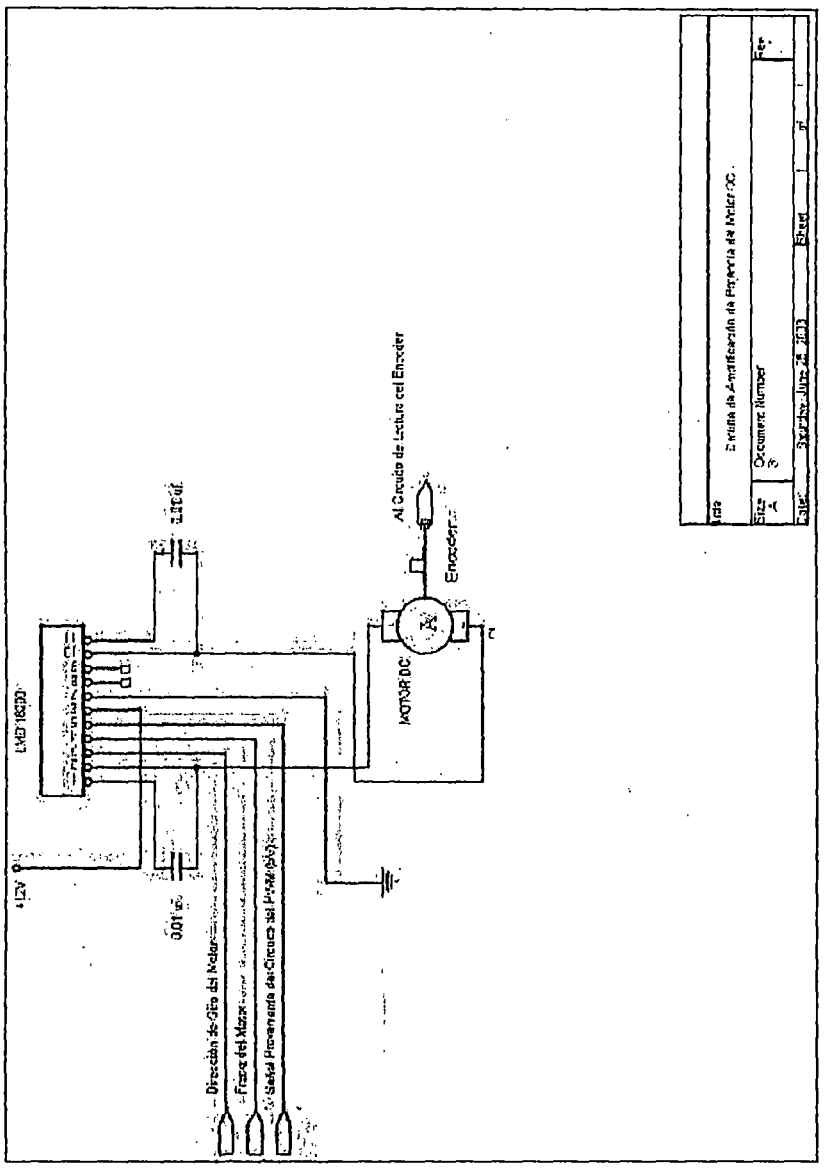

Figura 6.12: Circuito de la etapa de potencia.

Para saber si hubo errores en la transmisión se usa un bit de paridad el cual indica el error. Por ejemplo: en el byte 00111100 se tienen 4 unos por lo que al bit de paridad se le asigna un 0 y se dice que tiene paridad par y en el byte 10111100 se tienen 5 unos por lo que al bit de paridad se le asigna un 1 y se dice que tiene paridad impar.

Registro de Configuración del Hardware

TXSTA. Registro de configuración de la transmisión serial.
RCSPA. Registro deconfiguración de la recepción serial.
SPBRG. Contador Programable digitalmente que configura la velocidad de transmisión.

Generador de Baudios

En el protocolo asíncrono RS-232, la frecuencia en baudios (bits por segundo) a la que se realiza la transferencia se debe efectuar a un valor normalizado: 330, 600, 1200, 2400, 4800, 9600, 19200, 38400, etc. Para generar esta frecuencia, el USART dispone de un Generador de Frecuencia en Baudios, BRG, cuyo valor es controlado por el contenido grabado en el registro SPBRG.

Además del valor X cargado en el registro SPBRG, la frecuencia en baudios del generador depende del bit BRGH del registro TXSTA. En el caso de que BRGH $= 0$ se trabaje en baja velocidad y si BRGH $= 1$ se trabaja en alta velocidad. Según este bit se obtendrá el valor de una constante K necesaria en la determinación de la frecuencia de funcionamiento.

$$FrecuenciaenBaudios = \frac{F_{oscilacion}}{K(X+1)}$$

Donde X es el valor cargado en el registro SPBRG (valor expresado en decimal y que debe ser un número entero entre 0 y 255). Si BRGH $= 0$, baja velocidad y $K = 64$. Si BRGH $= 1$, baja velocidad y $K = 16$.

Por ejemplo, para una $F_{oscilacion} = 4MHZ$ y una $F_{baudios} = 9600baudios$, tenemos que a baja velocidad $X = 6$, lo que da un error de aproximadamente 7 % y para alta velocidad $X = 25$, lo que da un error de 0.16 %.

Transmisor Asíncrono

El dato que se debe transmitir por el USART transmisor se deposita en el registro TXREG y a continuación se traspasa al registro de desplazamiento TSR, que va sacando los bits secuencialmente y a la frecuencia establecida. Además, antes de los bits del dato de información incluye un bit de inicio y después de sacar todos los bits añade un bit de parada. El USART receptor recibe, uno a uno, los bits, elimina los dos de control y los de información una vez que han llenado el registro de desplazamiento TSR los traslada automáticamente al registro RCREG, donde quedan disponibles para su posterior procesamiento. En la figura 6.6 se muestra el diagrama de flujo de la comunicación serial utilizada en el programa.

6.6.2. Etapa de Comunicación Paralela

Esta etapa de comunicación es la más sencilla, puesto que el programa interface *interface.exe* sólo se limita a poner en el bus de datos de la PC (LPT1) la señal u_k y el segundo PIC16F876 es el encargado de leer constantemente esta señal a través de todo se puerto B. El diagrama de flujo de esta etapa se muestra en la figura 6.8.

6.7. Selección del Periódo de Muestreo y Diagrama de Tiempos

Primero debemos analizar el número de pulsos que da el encoder en cada milisegundo, para ver si esta cantidad de pulsos está dentro del intervalo que nos da ofrece los registros del PIC:

$$N°_{maximo\_que\_puede\_representar} = TMRH < 15:8 > + TMRL < 7:0 >= 65536$$

Ahora debemos calcular cuantos pulsos puede dar como máximo el encoder en 1 milisegundo. De la hoja técnica del motor tenemos que la velocidad máxima sin carga que puede desarrollar el motor es de 3000 rpm, ahora con esta información deducimos lo siguiente:

$$3000\frac{revoluciones}{minuto} = 50\frac{revoluciones}{segundo} = 50\frac{revoluciones}{segundo} \cdot \frac{504 pulsos}{revolucion} = \cdots$$
$$= 25200\frac{pulsos}{segundo} \cdot \frac{0{,}001 segundo}{1 mseg} = 25{,}2\frac{pulsos}{mseg}$$

El resultado anterior nos dice que en un milisegundo el PIC va a recibir aproximadamente 26 pulsos del encoder del motor, lo que a su vez nos dice que el periódo máximo hasta donde el PIC puede recibir los datos del encoder es $(65536/26) mseg = 2,52 segundos$, asegurando que se va a tomar todos los pulsos que se reciba del encoder:

$$1\, milisegundo \leq T_{muestreo} \leq 2{,}52\, segundos$$

Ahora debemos calcular el tiempo de la recepción de los datos del PIC (comunicación serial). Debemos tener presente que la cantidad de bits que vamos a trasmitir serialmente es 20 bits (8 del TMRL, 8 del TMRH, y cuatro de control). Como hemos elejido una velocidad de transmisión de 9600 baudios entonces tenemos que los 20 bits serán trasmtidos en 2.0833 mseg.

La trasmición paralela es mas rápida (depende en gran medida de la velocidad de la computadora), el tiempo que consume es de 0.012 mseg[1].

El tiempo de ejecución del algoritmo de identificación es de 12.5320 mseg y el tiempo de gráfico es de 8.012 mseg.

Ahora sumamos todos los tiempos que hemos calculado y obtenemos lo siguiente:

$$T_{\text{transmisión\_serial}} = 2{,}0833\, mseg$$

[1]Pentium IV, 1.2 GHZ, 128K.

$$
\begin{aligned}
T_{\text{algoritmo\_identificación}} &= 12{,}5320\,mseg \\
T_{\text{gráficas}} &= 8{,}0120\,mseg \\
T_{\text{transmisión\_paralela}} &= 0{,}0120\,mseg \\
T_{\text{ejecución}} &= 22{,}6393\,mseg \\
T_{\text{muestreo}} &= T_{\text{ejecución}} \cdot 1{,}5\,mseg = 33{,}9589\,mseg
\end{aligned}
\tag{6.6}
$$

En la figura 6.13 mostramos la distribución de los tiempos durante el procesamiento de los datos.

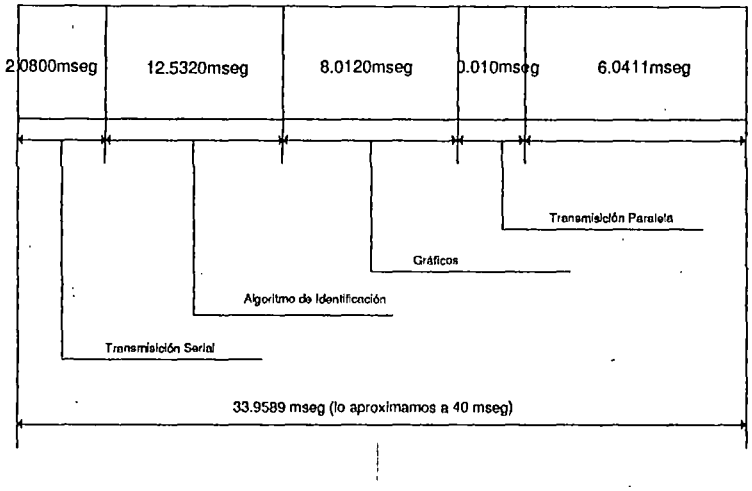

Figura 6.13: Diagrama de tiempos.

6.8. Programación

Para la programación, se dicidió utilizar el Visual Basic 6.0, debido a que ofrece mejores prestaciones visuales, un manejo y programación mucho más sencilla (uso de nemotécnicos fácilmente relacionables) frente otros lenguajes como el C, C^{++} ó incluso el $VisualC^{++}$, pero la gran desventaja que posee este lenguaje de programación es que por ser un lenguaje de alto nivel, consume muchos recursos de memoria lo cual nos limita a sintetizar al máximo el código de programación (evitar cualquier redundancia), para así lograr una mayor eficiencia del programa que realicemos.

Para poder realizar las pruebas de la identificación en línea, primero se tuvo que crear un pequeño *"Generador de Ondas"*, el cual me permite mandar por el puerto paralelo LPT1, un número binario que a la vez es interpretado por el PIC, convirtiendo ese número en una señal de PWM la cual alimenta al circuito de potencia del motor DC.

Este módulo que se presenta en la figura 6.14 permite la generación de tres tipos de ondas: escalón, senoidal y cuadrada. Para la señal senoidal debemos aclarar que es una senoide *"positiva"*. Esto quiere decir que esta señal sigue la siguiente ecuación matemática:

$$u(k) = A + A \cdot sin(k) \tag{6.7}$$

Donde A es una constante positiva, que va a depender del valor máximo que arroja el PWM del PIC (en nuestro caso $A = 4{,}231$).

Esto se ha hecho para simplificar la programación del algoritmo de la generación de ondas. El módulo que se muestra en la 6.14 contiene una serie de botones, los cuales poseen nombres con relación a la función que desempeñan.

A continuación, en la figura 6.15, presentamos la interfaz gráfica del programa principal de identificación.

Al igual que en el caso anterior esta interface posee una serie de botones, los cuales poseen etiquetas que me permiten fácilmente manipular el programa. El programa en funcionamiento es mostrado en el Capítulo VII, en donde también se analizan los resultados obtenidos.

Tanto el ejecutable de ambos programas así como sus instaladores están en el CD que viene adjunto con la presente tesis. Las rutas donde se ubican dentro del CD son mostradas en la siguiente tabla.

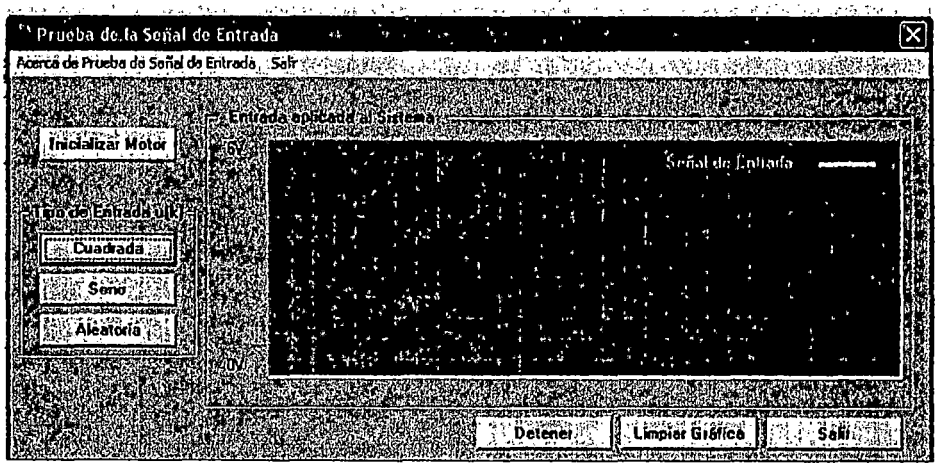

Figura 6.14: Interfaz gráfica del generador de ondas.

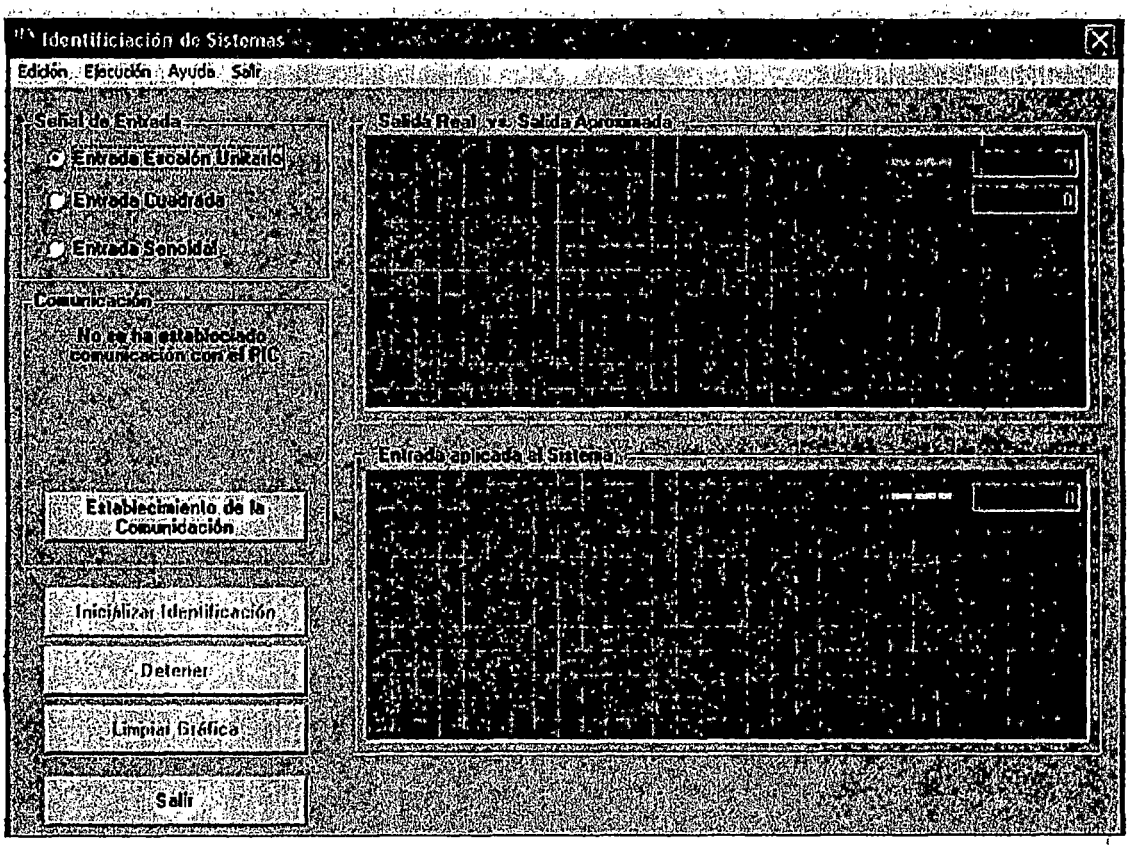

Figura 6.15: Interfaz gráfica del programa de identificación de sistemas.

*VisualBasic**Generador***señales.exe**	Generador de ondas
*VisualBasic**Identificacion***identificacion.exe**	Realiza la identificación de sistemas utilizando redes neuronales recurrentes (BPÉA)

6.9. Diagrama de Flujo de la Programación

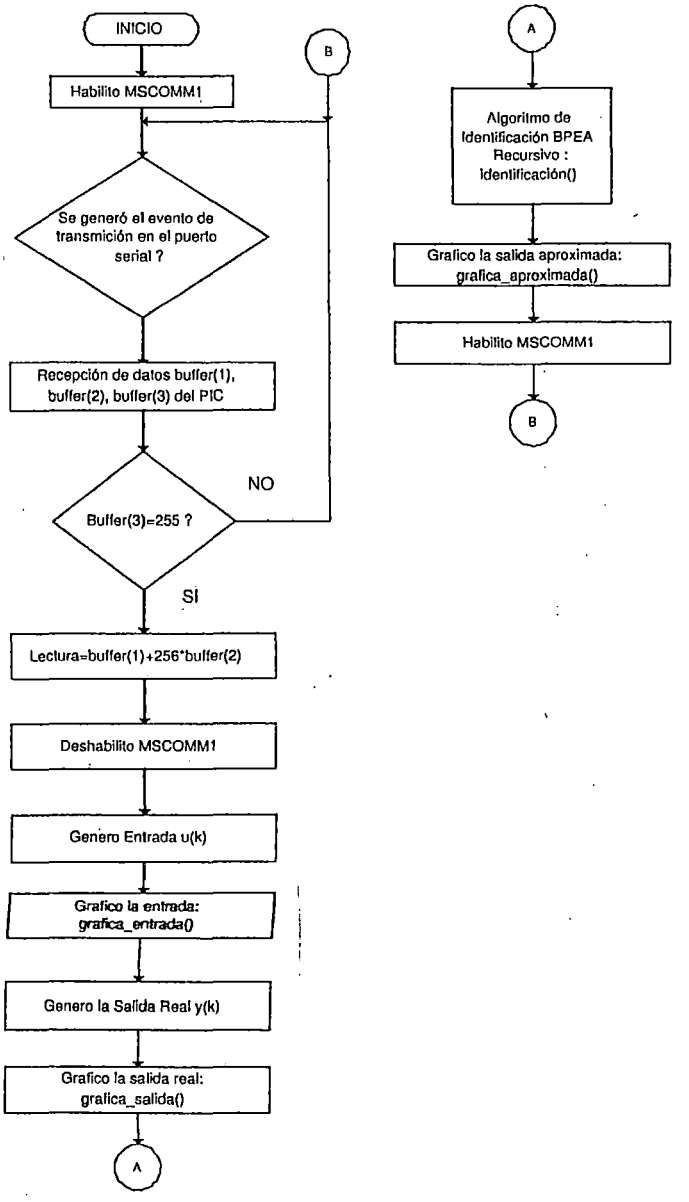

Figura 6.16: Diagrama de flujo de la programación para realizar la identificación de sistemas en línea (utilizando Redes Neuronales recurrentes - algoritmo BPEA).

CAPÍTULO VII
Análisis de Resultados

En este capítulo se analizará los resultados obtenidos de la identificación paramátrica de sistemas frente a diferentes tipos de entrada: escalón, cuadrada y escalón. Para realizar la identificación hemos decidido hacerlo mediante redes neuronales (BPEA). Hemos elegido este método y no el de Mínimos Cuadrados Recursivo o la identificación usando Algoritmos Genéticos, debido, a que en el primer caso, existe una amplia documentación de la identificación en línea utilizando este método, tanto en *papers, libros* y en la web, en el caso de la identificación usando Algoritmos Genéticos, este método consume demasiados recursos de memoria y tiene un tiempo de ejecución[1] alto en comparación con las redes neuronales o con el RLS. Por estos dos motivos se decidió a implementar el algoritmo BPEA para la identificación paramétrica del motor DC.

En las gráficas de la respuesta del motor DC que estamos utilizando veremos que frente a diferentes tipos de señales el motor no responde como la curva teórica del motor DC manda, las razones por las cuales ocurre esta situación se debe a lo siguiente:

- Error en la comunicación: es el que se genera debido a la transmisión serial[2] propiamente dicha, 0.16 %.
- Error en el Encoder: debido a la antiguedad del motor el sensor óptico que posee falla cada cierto tiempo, enviando una cantidad de pulsos que no correponde a lo que el motor genera (saturación del sensor).

Pero a pesar de estas condiciones la red neuronal recursiva deberá ser capaz de representar con un mínimo error al sistema real. A continuación mostraremos

[1]Pentium IV, 1.2GHZ, 128k Tiempo de Ejecución 12 minutos
[2]Dato obtenido de la Hoja téctina del PIC16F876, año 1999, pag. 98

la respuesta del sistema real y del sistema aproximado frente a las diferentes entradas a que son sometidas.

7.1. Aproximación Funcional frente a una entrada Escalón

Cuando sometemos nuestro sistema a una entrada escalón obtenemos la respuesta que se presenta en la figura 7.1, donde se aprecia que luego de algunos segundo, donde la red neuronal empieza ha entrenarse, esta logra aproximar en buena medida a la curva real del motor DC utlizado. A pesar de los sobrepicos que se observa en la figura la red neuronal trata de seguir incluso a esas no linealidades, cumpliendo en forma satisfactoria representar al sistema real.

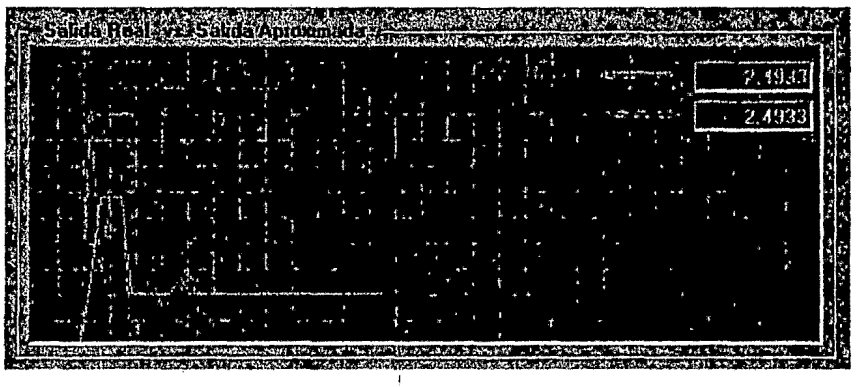

Figura 7.1: Respuesta del sistema real y del sistema aproximado frente a una entrada escalón unitario.

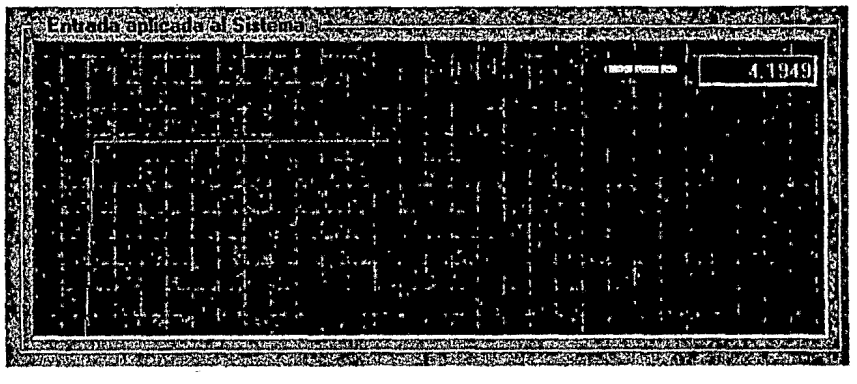

Figura 7.2: Gráfica de la señal de entrada: escalón unitario.

7.2. Aproximación Funcional frente a una entrada Senoidal

Ahora someteremos al sistema real y al sistema aproximado a la onda senoidal "positiva"[3]. Para analizar mejor los resultados obtenidos hemos tomado dos secuencias de la onda de entrada y las respectivas respuestas de los sistemas: la primera es cuando la onda de entrada se encuentra en la fase de -90° a 90° y la segunda cuando la onda de entrada se encuentra en la fase de 90° a 180°.

En el primer caso, como podemos apreciar de la figura 7.3, la red neuronal se adapta rápidamente al sistema real incluso a los cambios bruscos del sistema.

Ahora en el segundo caso, se repite el mismo caso anterior, la red responde con un error aceptable. Pero tenemos que aclarar en esta parte se ha observado que para valores iniciales altos del sistema real, la red rápidamente se adapta a ellas, pero para valores bajos, la red tarda un tiempo mayor en adaptarse.

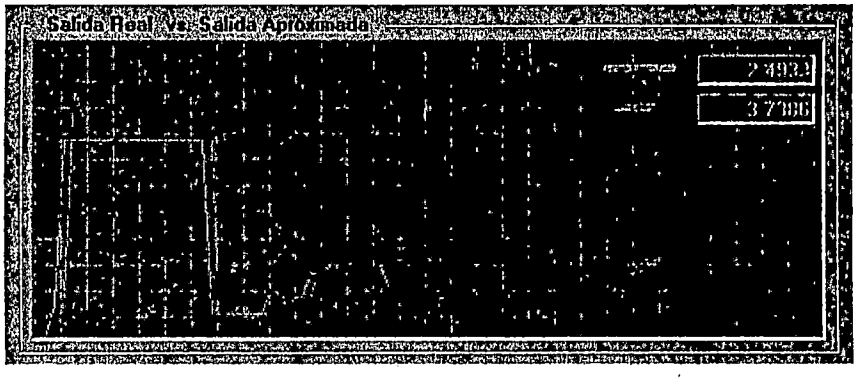

Figura 7.3: Respuesta del sistema real y del sistema aproximado frente a una entrada senoidal (-90° a 90°).

7.3. Aproximación Funcional frente a una entrada Cuadrada

Frente a esta entrada el aproximador funcional responde con un buen margen de error, acercándose casi exactamente a la curva real del sistema incluso cuando se presentan las no linealidades ya comentadas. Pero debemos hacer presente, que como en el caso anterior se observó que la red neuronal se adapta

[3] $u(k) = 4{,}24 + 4{,}24 \cdot sin(k)$

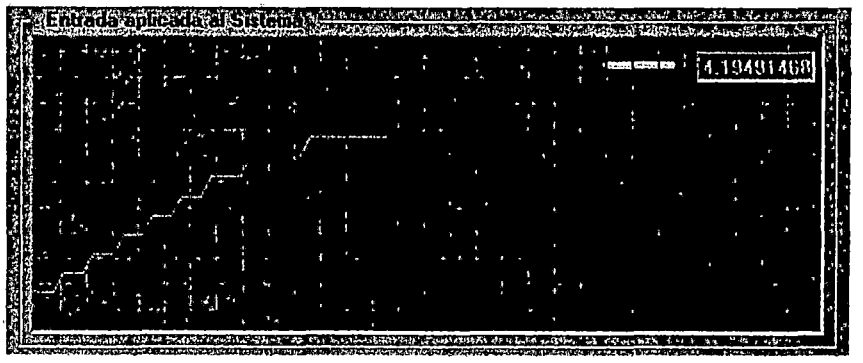

Figura 7.4: Gráfica de la señal de entrada: seno (-90° a 90°).

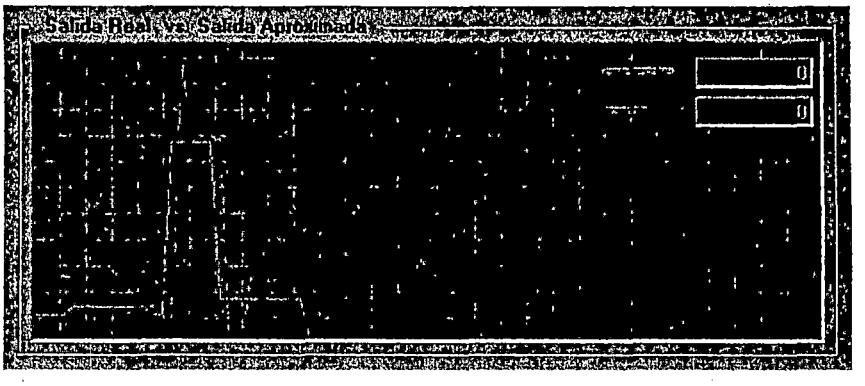

Figura 7.5: Respuesta del sistema real y del sistema aproximado frente a una entrada senoidal (90° a 180°).

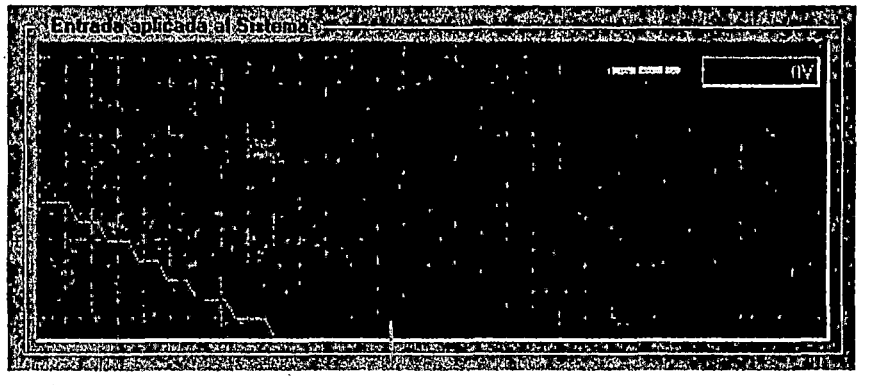

Figura 7.6: Gráfica de la señal de entrada: seno (90° a 180°).

rápidamente a valores altos de la entrada y lentamente para valores bajos.

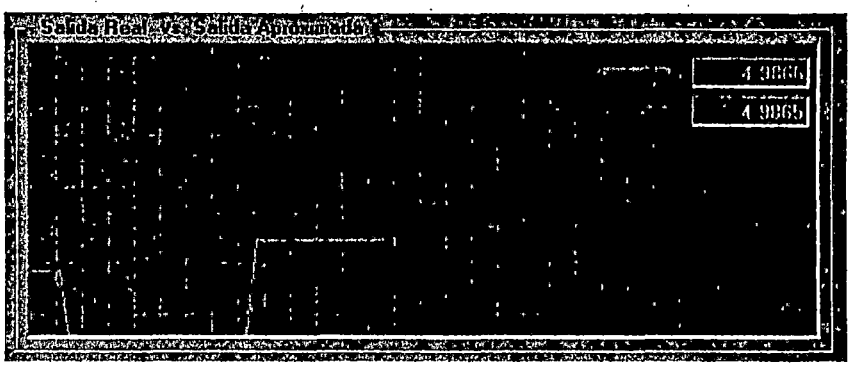

Figura 7.7: Respuesta del sistema real y del sistema aproximado frente a una entrada cuadrada.

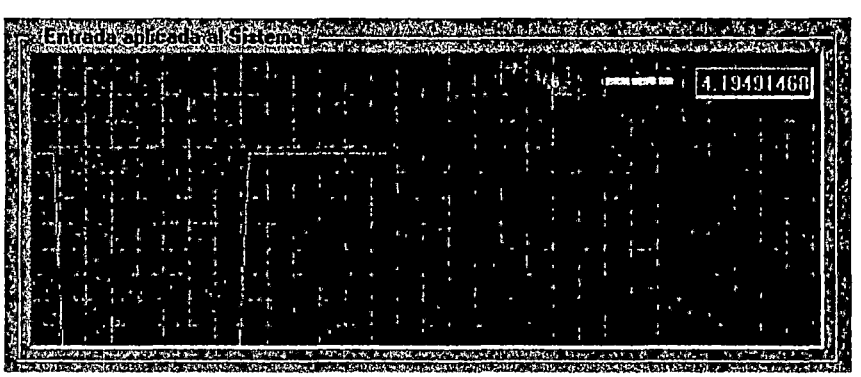

Figura 7.8: Gráfica de la señal de entrada: cuadrada.

CONCLUSIONES

La presente tesis ha girado en torno al problema de hallar un modelo matemático que me represente la dinámica completa de un sistema. El modelo matemático que se ha trabajado ha sido el modelo polinomial NARMAX. Se ha analizado diferentes métodos para hallar los parámetros de este modelo polinomial: mínimos cuadrados recursivos, redes neuronales recurrentes y los algoritmos genéticos. Se mostró la eficacia de cada uno de ellos frente a la identificación de sistemas tanto lineales como no lineales.

De acuerdo a los resultados obtenidos durante el desarrollo de la presente tesis, obtenemos las siguientes conclusiones:

- En primer lugar es importante destacar que todo el conocimiento a priori que se tenga del proceso deberá ser incluido directamente en el modelo, especialmente en la etapa de selección de las variables de entrada. En segundo lugar recordar siempre de *probar primero lo más sencillo.* Según este criterio se ensayarán primero los modelos lineales, haciendo uso de técnicas de identificación de sistemas bien establecidas. Si se detectasen características no lineales en el proceso o que el modelo hallado no puede seguir al proceso con exactitud, el siguiente modelo a ensayar sería el modelo polinomial NARMAX (modelo no lineal autoregresivo de media móvil con entradas exógenas).

- Una herramienta fundamental para la selección de las variables de entrada de los modelos no lineales es el Análisis Estadístico de Sensibilidades. Este análisis permite identificar aquellas variables de entrada que no tienen influencia en las salidas de los modelos ajustados. La eliminación de estas variables reduce la complejidad artificial del modelo, aumentando su ca-

pacidad de generalización.

- Con lo que respecta a la identificación paramétrica de sistema utilizando el *Método de los Mínimos Cuadrados Recursivos* podemos afirmar que este procedimiento nos asegura muy buenos resultados de aproximación, para modelos lineales y además el tiempo de procesamiento de este algoritmo es pequeño en comparación con las redes neuronales y los algoritmos genéticos. Pero este método muestra grandes deficiencias frente a la identificación de sistemas no lineales.

- Se logra una notable mejora al incluir, tanto para la identificación lineal como para la identificación no lineal, el factor de olvido. La inclusión de este factor dentro del algortimo de Mínimos Cuadrados Recursivos me permite reducir la memoria del identificador con el objeto de que éste pueda seguir las variaciones del sistema, ponderando las medidas de forma que tengan más peso las últimas sobre las antiguas.

- Al realizar la identificación de una planta mediante el polinomio NARMAX utilizando el *Método de los Mínimos Cuadrados Recursivos* se logra disminuir el procentaje de error de la aproximación.

- Ahora para aliviar el inconveniente anterior recurrimos a las *Redes Neuronales Recurrentes*. A través de ellas realizamos la identificación de procesos lineales mediante el uso del polinomio NARMAX obteniendo márgenes de errores muy bajos, demostrando la eficacia de la redes neuronales para identificación de sistemas lineales. Al realizar la identificación de un sistema no lineal, la red neuronal obtuvo un sistema aproximado con una margen de error bajo, lo que demostró también la eficacia de este método para la identificación de sistema no lineales.

- Una de las ventajas que ofrece al usar las redes neuronales es que me per-

mite hallar el modelo aproximado de cualquier planta sea lineal o no lineal. También las redes neuronales necesitan poca o casi nada de información acerca del sistema a modelar.

- La gran desventaja de las redes neuronales, es que para su entrenamiento a través de la algoritmo del BPEA (Back Propagation Error Algorithm) se consume mucho recursos de memoria debido a que la PC debe trajar con matrices de dimensiones muy grandes.

- La eficacia de la Redes Neuronales Recurrentes quedó demostrada en la identificación en línea de un motor DC. Pese a las no linealidades que presentaba el motor la red neuronal pudo identificar al sistema con un pequeño margen de error.

- Los Algoritmos Genéticos ofrecen un gran método para la identificación cuando no se conoce absolutamente nada acerca de la dinámica del sistema. Los AG realizarán una búsqueda sobre todos posibles patrones de soluciones hasta llegar a encontrar la solución óptima. Esta búsqueda puede tardar tiempos mayores a los necesitados por el RLS y las Redes Neuronales, pero tendremos siempre la certeza que el AG va ha encontrar la mejor solución. Para acelerar el tiempo de convegencia del AG se ha introducido el Templado Simulado, para que este pueda dirigir al AG evitando que pierda tiempo evaluando a soluciones no factibles.

- Los AG son buenos métodos estocásticos de búsqueda de soluciones cuasióptimas, pero cuando se encuentra un método heurístico que soluciona tal problema, entonces los AG se tornan lentos y obsoletas, dado que el método heurístico ofrecerá mejores resultados y en menos tiempo.

- La característica esencial de los AGs no se observa directamente, la cual es la capacidad que tiene de intercambio estructurado de información en

paralelo o lo que se llama *paralelo implícito*, es decir que los AG procesan externamente cadenas de códigos, sin embargo, lo que está procesando internamente son similitudes entre cadenas y de manera tal que al procesar cada una de las cadenas de la población se están procesando a la vez todos los patrones de similitud que contienen, que son muchos más. Es precisamente por esto que los AGs son mucho más eficaces que otros métodos de búsqueda ciega, y por tanto más robustos.

RECOMENDACIONES PARA TRABAJOS FUTUROS

Con la presente tesis ha quedado abierta la línea de investigación del modelamiento de sistemas dinámicos a través de la expanción polinomial del modelo NARMAX. De acuerdo a los diferentes puntos presentados durante el desarrollo de la presente tesis y los expuestos en las conclusiones podemos dar las siguientes recomendaciones:

- Implementar el algoritmo de identificación usando Algoritmos Genéticos y estudiar los resultados que se obtendrían. Se recomienda implementar el algoritmo en un DSP que es un dispositivo que me permite una fácil manejo de operaciones de punto flotante y posee una frecuencia de reloj que me permitiría trabajar fácilemente con las operaciones que demandan el AG y interactuar al mismo tiempo con la PC.

- Adicionar un filtro a la salida del encoder del motor DC, el cual me permitirá filtrar los datos del encoder y así obtener una secuencia de datos uniformemente distribuídos.

- Implementar un módulo para que se pueda realizar algún tipo de control, ya sea clásico, predictivo, adaptativo,etc., para comprobar la eficiencia del modelo hallado en línea.

- Implementar módulos para que se pueda tomar datos de otros sistemas o procesos tales como la lectura del nivel de un tanque, lectura de la temperatura de un horno, entre otros, para poder hallar así el modelo aproximado de esos sistemas y realizar posteriormente el control respectivo.

- Las últimas versiones del Matlab vienen con librerías que me permiten interactuar los programas que se hacen en este lenguaje con el Visual Basic. Entonces queda como punto para investigar más acerca de este tema, ya que los algoritmos de identificación son muchos más fáciles de programas en Matlab que en Visual Basic. Esto nos facilitaría la implementación de un gran número de algoritmos para su posterior análisis.

APÉNDICE A
PROGRAMAS EN MATLAB

Listado de Programas

Debido a que los programas realizados en Matlab son demasiados extensos para incluirlos en este apéndice, se ha optado por colocarlos en el CD que se adjunta a la presente tesis. A continuación se muestra una tabla en la que se detalla el nombre del programa, su ubicación dentro del CD y una breve descripción de lo que realiza el programa.

\ \ *Matlab* \ *Aproximadores* \ **AES_1.m**	Realiza el análisis estadístico de sensibilidades para una estructura lineal.
\ \ *Matlab* \ *Aproximadores* \ **AES_2.m**	Realiza el análisis estadístico de sensibilidades para una estructura no lineal.
\ \ *Matlab* \ Identificación \ *RLS* \ **lqs_mejorado_V.m**	Realiza la identificación de sistemas utilizando el algoritmo de RLS para una planta lineal.
\ \ *Matlab* \ Identificación \ *RLS* \ **lqs_mejorado_V_no.m**	Realiza la identificación de sistemas utilizando el algoritmo de RLS para una planta no lineal.

\ \ *Matlab* \ Identificación \ *RN* \ **bpea_V.m**	Realiza la identificación de sistemas utilizando RN (algoritmo BPEA) una planta lineal.
\ \ *Matlab* \ Identificación \ *RN* \ **bpea_V.m**	Realiza la identificación de sistemas utilizando RN (algoritmo BPEA) una planta no lineal.
\ \ *Matlab* \ Identificación \ *GA* \ **plantillagenN.m**	Realiza la demostración del funcionamiento de un AG simple
\ \ *Matlab* \ Identificación \ *GA* \ **GA_lineal.m**	Realiza la identificación de sistemas utilizando AG para una planta lineal.
\ \ *Matlab* \ Identificación \ *GA* \ **GA_no_lineal.m**	Realiza la identificación de sistemas utilizando AG para una planta no lineal.
\ \ *Matlab* \ Identificación \ *GA* \ **SAGA_lineal.m**	Realiza la identificación de sistemas utilizando SA-GA para una planta lineal.
\ \ *Matlab* \ Identificación \ *GA* \ **SAGA_no_lineal.m**	Realiza la identificación de sistemas utilizando SA-GA para una planta no lineal.

Rutinas Utilizadas por el Algoritmo Genético

\ \ *Matlab* \ Identificación \ *GA* \ *Rutinas* \ **binario1.m**	Convierte un número decimal a binario.
\ \ *Matlab* \ Identificación \ *GA* \ *Rutinas* \ **decimal1.m**	Convierte un número binario a decimal.
\ \ *Matlab* \ Identificación \ *GA* \ *Rutinas* \ **bin2gray.m**	Convierte un número binario a gray.
\ \ *Matlab* \ Identificación \ *GA* \ *Rutinas* \ **gray2bin.m**	Convierte un número gray a binario.

APÉNDICE B
PROGRAMAS CON LA INTERFASE GRÁFICA GUI DE MATLAB

En este apéndice se muestra una serie de programas implementados con la Interface Gráfica del Matlab (GUI). La creación de una Interfase Gráfica al Usuario basada en MATLAB (GUI), no importando qué tan grande o pequea, es *única*, con *cierta personalidad* para ayudar a un *usuario específico* a realizar una *tarea específica*. A lo que está enfocada este apéndice es moldear todos éstos aspectos, sabiendo de antemano que el usuario específico es cualquier persona interesada en el estudio del modelamiento de sistemas dinámicos a través de los métodos expuestos en la tesis, que cuente con una noción previa de la materia, para reafirmar sus conocimientos y desarrollar capacidades por medio de la repetida práctica sobre el *AMBIENTE GUI INTERACTIVO* creado en MATLAB, el usuario específico puede o no haber previamente tenido contacto con el programa MATLAB, pues más que llegar a desarrollar algún código en ése lenguaje trabajará directamente con un ambiente diseado de tal manera que no se le dificulte su utilización debido a que está compuesto por elementos obvios al manejo y sencillos de maniobrar.

Menú Principal

Si ejecutamos en la línea de comandos del Matlab, *menu*, se mostrará una interfaz gráfica en la cual se puede ver una serie de botones (ver figura B.1, que indican los diferentes métodos utilizados para la identificación de sistemas. Al ejecutar cualquiera de los botones estos llamarán automáticamente a la interfaz que le corresponde. También podemos observar un botón de *Ayuda* el cual me da una idea de como se ejecuta el programa y un botón de *Créditos y Valores* en el cual se muestran los créditos y agradecimientos a las personas que colaboraron en la creación de estas interfaces.

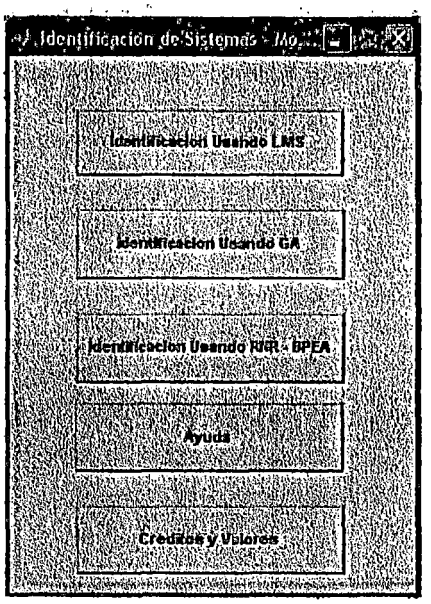

Figura B.1: Menú principal del GUI de identificación de sistemas.

Identificación de Sistemas Utilizando Mínimos Cuadrados Recursivos en el Modelo ARMAX

Mediante esta interface, el usuario podrá modelar sus sistema dinámico utilizando el método de lo mínimos cuadrados recursivo. El usuario puede seleccionar entre tres modelos existentes (Modelo de Un Motor DC, Modelo Lineal de Segundo Orden y un Modelo No Lineal) (ver figura B.2 Letra **D**). Luego que el usuario ha seleccionado el modelo con el que desea trabajar, deberá seleccionar entre las diferentes entradas que se ofrecen para entrenar el RLS (ver figura B.2 Letra **E**).

Un vez realizado los dos pasos anteriores, tiene la opción de variar los parmetros del RLS de acuerdo a su criterio o conveniencia (ver figura B.2 Letra **F**) para el posterior entrenamiento del RLS (ver figura B.2 Letra **G**). Una vez que ya hemos ejecutado el entrenamiento del RLS podemos ir viendo como se va adaptando la curva aproximada a la curva real (ver figura B.2 Letra **B**) y como van variando los coeficientes del modelo ARMAX (ver figura B.2 Letra **J**),

también podemos ver la evolución del error a lo largo de la simulación (ver figura B.2 Letra C).

Finalmente, una vez que ya terminó la simulación, el usuario deber proceder a validar el modelo hallado por el RLS sometiendo a diferentes tipos de entrada (ver figura B.2 Letra H) y podrá apreciar sus caractersticas en los gráficos de la simulación.

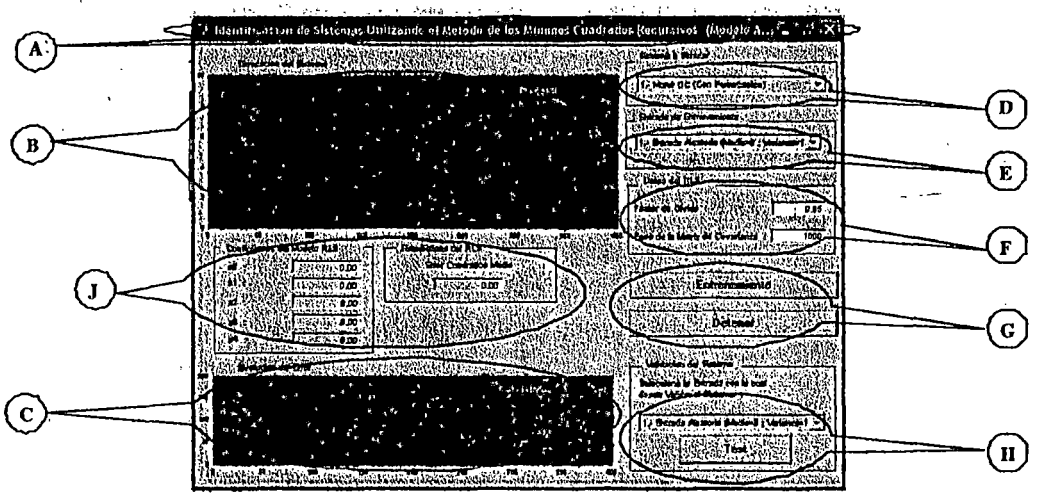

Figura B.2: Interface gráfica para la identificación de sistemas utilizando el método de los Mínimos Cuadrados recursivo (modelo ARMAX).

- A Título de la interfaz gráfica.

- B Espacio gráfico donde se realiza el entrenamiento del algoritmo de identificación y su respectiva validación.

- C Espacio gráfico donde se puede preciar la evolución del error a lo largo del tiempo.

- D Me permite seleccionar entre tres diferentes modelos.

- **E** Me permite seleccionar entre diferentes tipos de entrada para el entrenamiento del algoritmo de identificación.

- **F** Muestra los parámetros de trabajo del RLS, el cual puede ser cambiado por el usuario según su criterio.

- **G** Botones que me permiten el inicio del entrenamiento del algoritmo de identificación y la detención del mismo.

- **H** Esta parte me permite seleccionar entre diferentes tipos de entrada para la validación del sistema, y su correspondiente simulación.

- **J** Me muestra como van evolucionando los coeficientes del polinomio ARMAX que va encontrando el RLS, así como también la evolución del error cuadrático medio.

Identificación de sistemas utilizando Redes Neuronales recurrentes (algoritmo BPEA) en el modelo NARMAX

Mediante este interface, el usuario podrá modolear sus sistema dinámico utilizando una red neuronal recursiva que es entrenada mediante el algoritmo BPEA. El usuario puede seleccionar entre tres modelos existentes (Modelo de Un Motor DC, Modelo Lineal de Segundo Orden y un Modelo No Lineal) (ver figura B.3 Letra **D**). Luego que el usuario ha selecciondo el modelo con el que desea trabajar, deberá seleccionar entre las diferentes entradas que se ofrecen para entrenar la red neuronal (ver figura B.3 Letra **E**).

Un vez realizado los dos pasos anteriores, tiene la opción de variar los parmetros de la red neuronal recursiva de acuerdo a su criterio o conveniencia (ver figura B.3 Letra **F**) para el posterior entrenamiento de la red neuronal (ver figura B.3 Letra **G**). Una vez que ya hemos ejecutado el entrenamiento de la

red neuronal (en este punto cabe destacar la importancia de esta interface que me permite ver el modelo hallado por la red neuronal entrenado mediante diferentes tipos de entrada, ya que cada entrada ofrece un entrenamiento distinto a las demás, variando su presición) podemos ir viendo como se va adaptando la curva aproximada a la curva real (ver figura B.3 Letra B) y como van variando los coeficientes del modelo NARMAX (ver figura B.3 Letra J), también podemos ver la evolución del error a lo largo de la simulación (ver figura B.3 Letra C).

Finalmente, una vez que ya terminó la simulación, el usuario deber proceder a validar el modelo NARMAX hallado por la red neuronal recursiva sometiendo a diferentes tipos de entrada (ver figura B.3 Letra H) y podrá apreciar sus caracteristicas en los gráficos de la simulación.

Hay que tener presente que el tiempo de la simulación del ésta interface es mayor a la del RLS, debido a que la red neuronal trabaja con una mayor cantidad de matrices.

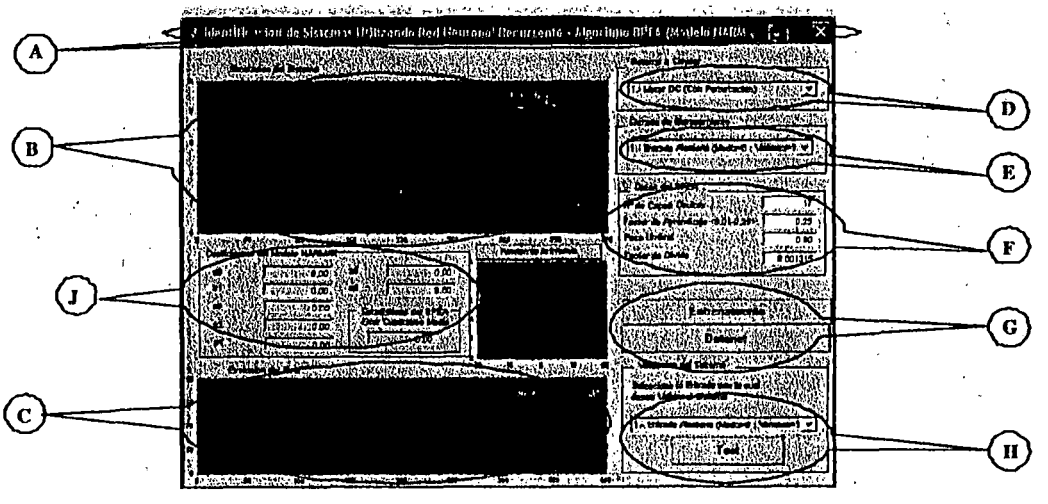

Figura B.3: Interface gráfica para la identificación de sistemas utilizando Redes Neuronales recurrentes (modelo NARMAX).

- A Título de la interfaz gráfica.

- B Espacio gráfico donde se realiza el entrenamiento del algoritmo de identificación y su respectiva validación.

- C Espacio gráfico donde se puede apreciar la evolución del error a lo largo del tiempo.

- D Me permite seleccionar entre tres diferentes modelos.

- E Me permite seleccionar entre diferentes tipos de entrada para el entrenamiento del algoritmo de identificación.

- F Muestra los parámetros de trabajo de la red neuronal recursiva, el cual puede ser cambiado por el usuario según su criterio.

- G Botones que me permiten el inicio del entrenamiento del algoritmo de identificación y la detención del mismo.

- H Esta parte me permite seleccionar entre diferentes tipos de entrada para la validación del sistema, y su correspondiente simulación.

- J Me muestra como van evolucionando los coeficientes del polinomio NARMAX que va encontrando la red neuronal recursiva, así como también la evolución del error cuadrático medio.

Identificación de sistemas utilizando algoritmos genéticos en el modelo NARMAX

Mediante este interface, el usuario podrá modolear sus sistema dinámico utilizando el codiciado algoritmo genético que evoluciona a lo largo de un número

de generaciones que el usuario puede colocar. El usuario puede seleccionar entre tres modelos existentes (Modelo de Un Motor DC, Modelo Lineal de Segundo Orden y un Modelo No Lineal) (ver figura B.4 Letra **D**). Luego que el usuario ha selecciondo el modelo con el que desea trabajar, deberá seleccionar entre las diferentes entradas que se ofrecen para lograr la evolución del algoritmo genético (ver figura B.4 Letra **E**).

Un vez realizado los dos pasos anteriores, tiene la opción de variar los parmetros del algoritmo genético (ver figura B.4 Letra **F**) para la posterior evolución de este (ver figura B.3 Letra **G**). Una vez que ya se ha realizado la evolución de los parámtros del polinomio NARMAX podemos ir viendo como se va adaptando la curva aproximada a la curva real (ver figura B.4 Letra **B**) y como van variando los coeficientes del modelo NARMAX (ver figura B.4 Letra **J**), también podemos ver la evolución del error a lo largo de la simulación (ver figura B.4 Letra **C**).

Finalmente, una vez que ya terminó la simulación, el usuario deber proceder a validar el modelo NARMAX hallado por el algoritmo genético sometiendo a diferentes tipos de entrada (ver figura B.4 Letra **H**) y podrá apreciar sus caractersticas en los gráficos de la simulación.

Hay que tener presente que el tiempo de la simulación del esta interface es mucho mayor a la del RLS y a la de la red neuronal, debido a que el algoritmo genético ocupa una mayor cantidad de recursos de memoria que en los dos casos anteriores.

- **A** Título de la interfaz gráfica.

- **B** Espacio gráfico donde se realiza el entrenamiento del algoritmo de identificación y su respectiva validación.

- **C** Espacio gráfico donde se puede apreciar la evolución del error a lo largo

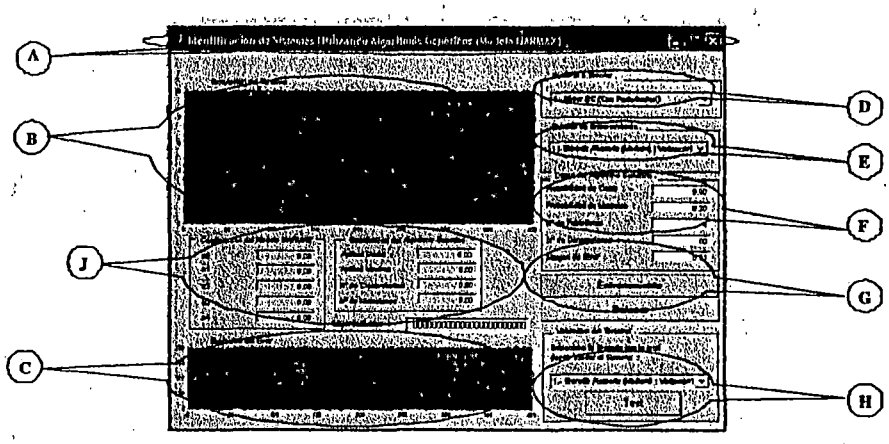

Figura B.4: Interface gráfica para la identificación de sistemas utilizando algoritmos genéticos (Modelo NARMAX).

del tiempo.

- D Me permite seleccionar entre tres diferentes modelos.

- E Me permite seleccionar entre diferentes tipos de entrada para el entrenamiento del algoritmo de identificación.

- F Muestra los parámetros de trabajo del algoritmo genético, el cual puede ser cambiado por el usuario según su criterio.

- G Botones que me permiten el inicio del entrenamiento del algoritmo de identificación y la detención del mismo.

- H Esta parte me permite seleccionar entre diferentes tipos de entrada para la validación del sistema, y su correspondiente simulación.

- J Me muestra como van evolucionando los coeficientes del polinomio NARMAX del AG, así como también la evolución del error cuadrático medio.

APÉNDICE C
CIRCUITOS IMPRESOS

Circuito impreso de la etapa de sensado

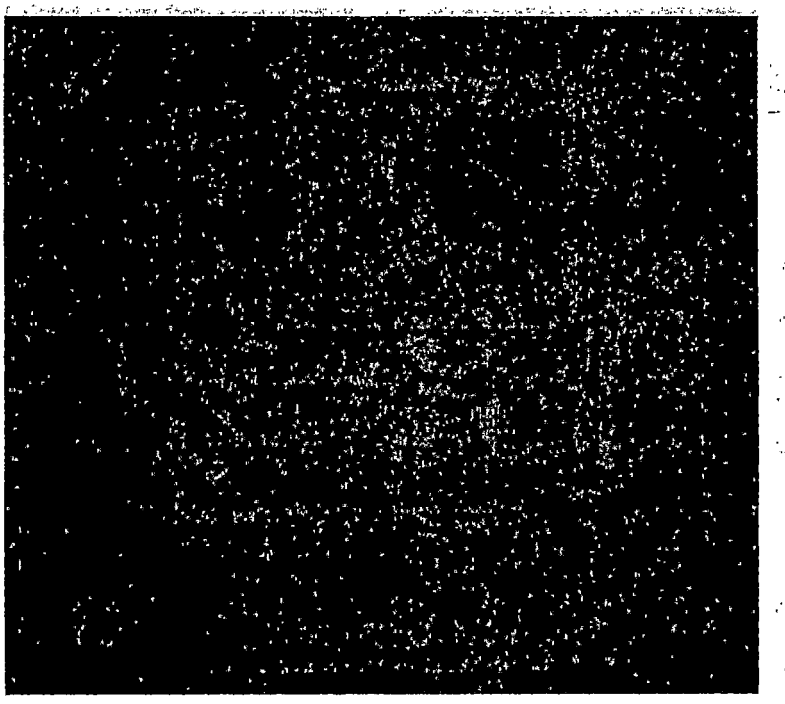

Figura C.1: Circuito impreso de la etapa de sensado.

Circuito impreso de la etapa de acondicionamiento de la señal del PWM proveniente de la PC

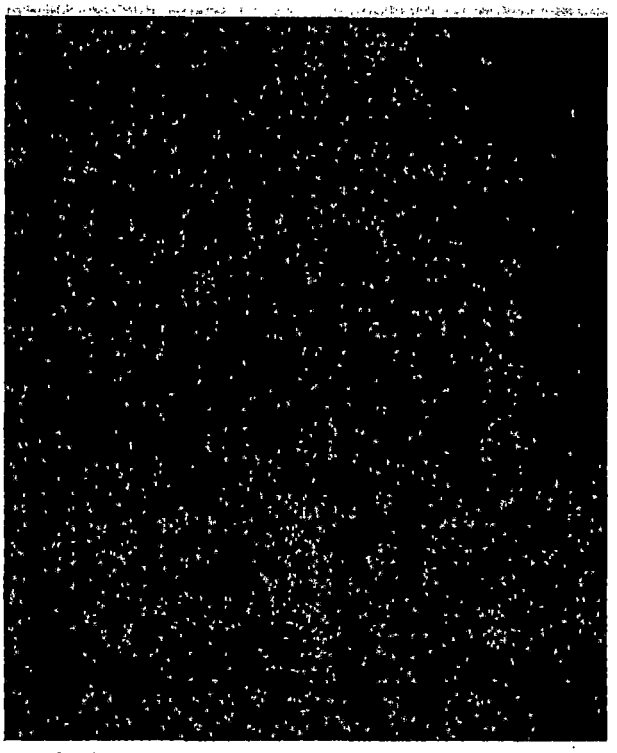

Figura C.2: Circuito impreso de la etapa de acondicionamiento de la señal del PWM proveniente de la PC.

Circuito impreso de la etapa de potencia

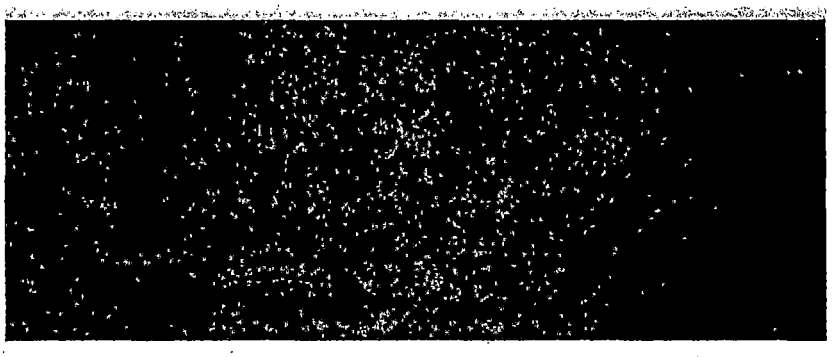

Figura C.3: Circuito impreso de la etapa de amplificación de potencia.

APÉNDICE D
CARACTERÍSTICAS FÍSICAS Y ELÉCTRICAS DEL MOTOR DC

Características físicas y eléctricas del motor DC

El motor DC[1] es un dispositivo electromotriz, esto quiere decir que convierte la energía eléctrica en energía motriz. Todos los motores disponen de un eje de salida para acoplar un engranaje, polea o mecanismo capaz de transmitir el movimiento creado por el motor.

Las características físicas y eléctricas del motor que se utilizar para la identificación en línea son mostradas en la figura D.1. El motor marca BUEHLER, modelo DC PM Motor 1.13.044 tiene acoplado un encoder óptico en la parte opuesta al eje. La resolución de este encoder es de $504ppr$ [2].

[1] DC Direct Current
[2] ppr pulsos por revolución

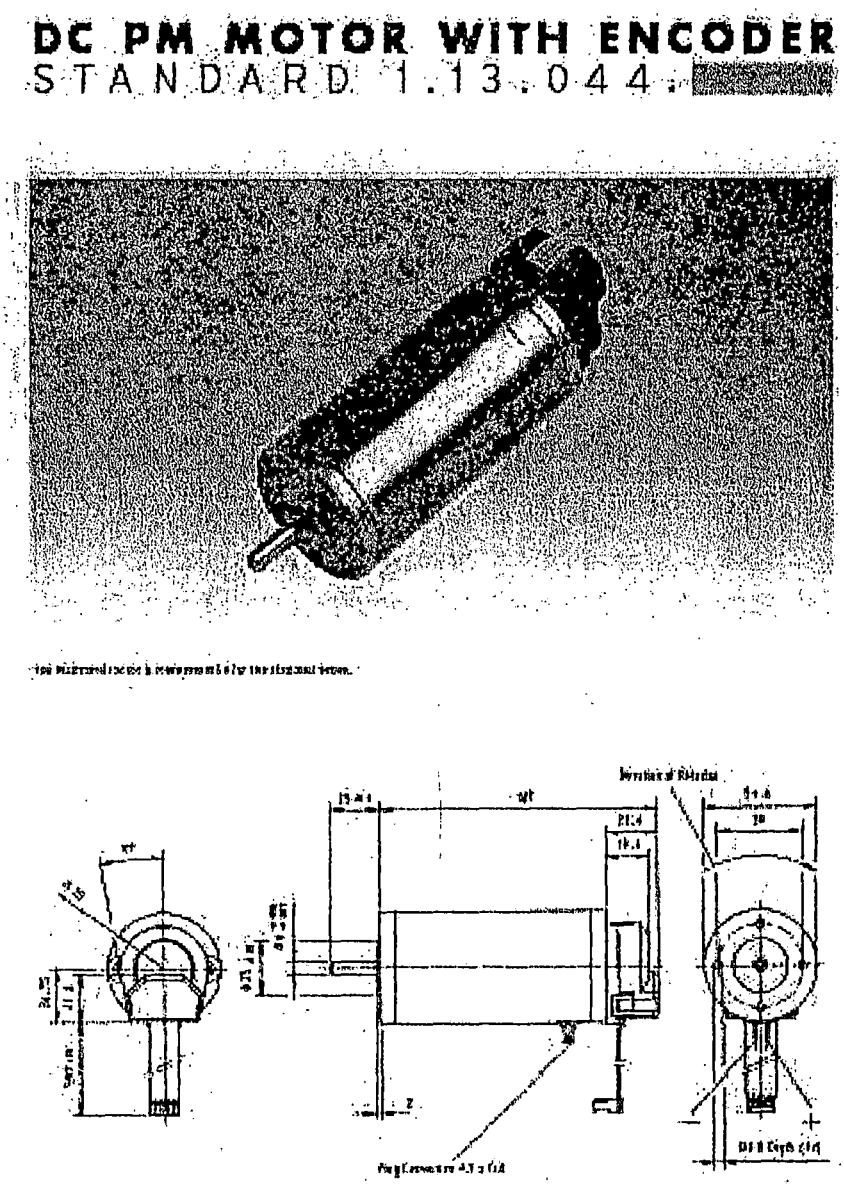

Figura D.1: Características físicas motor BUEHLER DC PM 1.13.044.

Technical Data *)

Type		413	414
Rated Voltage	V	12	24
Rated Torque	mNm	180	180
Rated Speed	rpm	3000	3000
Rated Output Power	W	57	57
Rated Current (**)	A	7.3	3.5
Max. Allow. Cont. Current (**)	A	4.9	2.9
No Load Speed	rpm	3900	3800
No Load Current	A	0.86	0.41
Stall Current	A	31	17
Stall Torque	mNm	840	940
Rotor Inertia	gcm²	250	250
Speed Regulation Constant	rpm/mNm	4.7	4.1
Torque Constant	mNm/A	27	57
Mech. Time Constant	ms	12	13
Elect. Time Constant	ms	1.0	1.2
Terminal Resistance	Ohm	0.38	1.4
Thermal Resistance	R_{th1} K/W	4.4	3.4
	R_{th2} K/W	4.3	4.3
Weight	g	940	940

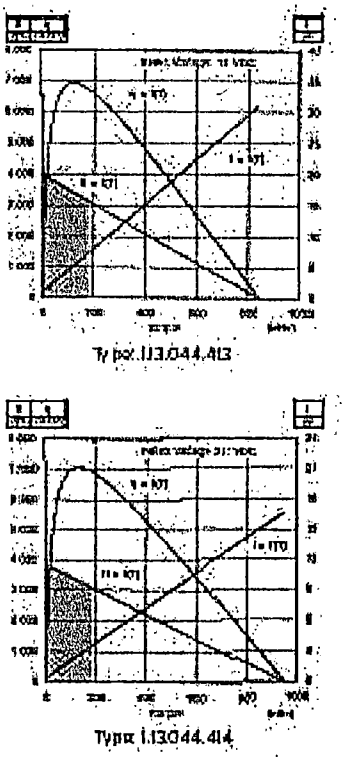

Figura D.2: Características eléctricas motor BUEHLER DC PM 1.13.044.

APÉNDICE E
CARACTERÍSTICAS FÍSICAS Y ELÉCTRICAS DEL ENCODER

Características físicas y eléctricas del encoder acoplado al motor BUEHLER

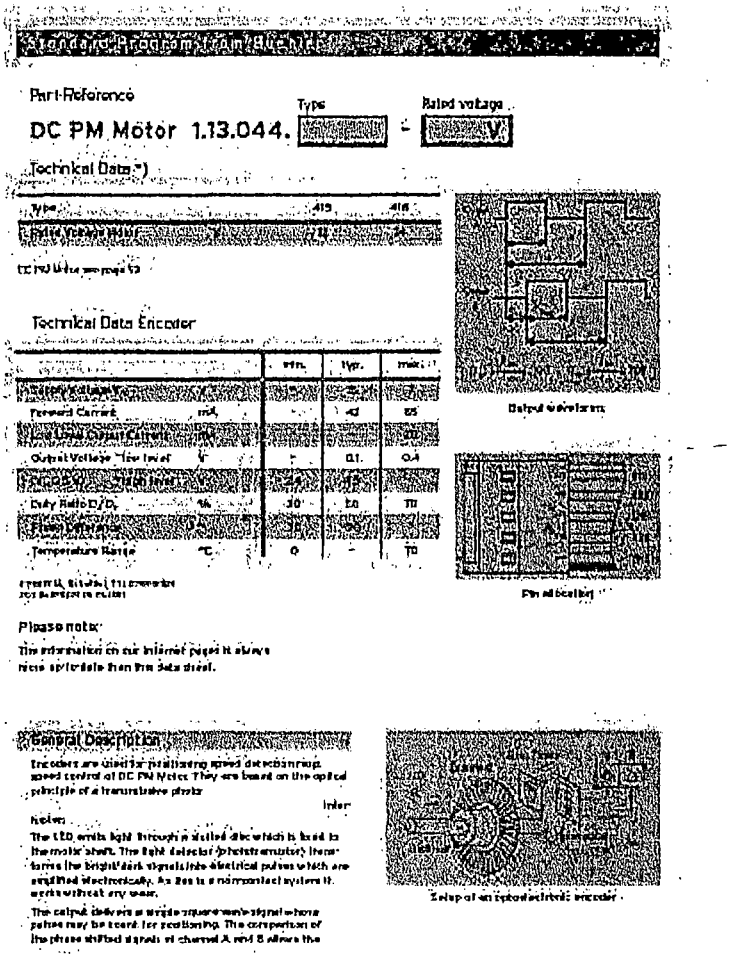

Figura E.1: Características físicas y eléctricas del encoder acoplado al motor BUEHLER.

BIBLIOGRAFÍA

[1] Duc Truong, Liu Xing. *Neural Networks and Identification, Prediction and Control.* Springer-Verlag London Limited, 1997.

[2] Edgar Campor Furtado, Eduardo M. A. M. Mendes, Erivelton G.Ñepomuceno. *IDENTIFICAO DE SISTEMAS DINAMICOS NAO-LINEARES CONTÍNUOS UTILIZANDO MODELOS NARMAX: ESTUDO DE CASO DE UM FORNO A ARCO ELÉTRICO.* XIV-Congreso Brasileiro de Automática, Setiembre 2002.

[3] Fonseca, Carlos Manuel Mira Da. *Multiobjetive Genetic Algorithms with Application to Control Engineering Problem.* Departament of Automatic Control an System Engineering, Unversity of Sheffield, September 1995.

[4] Francisco Rodríguez Rubio, Manuel Jesús López Sánchez. *Control Adaptivo y Robusto.* Secretariado de Publicaciones de la Universidad de Sevilla, 1996.

[5] Goldberg, David E. *GENETIC ALGORITHMS* in search, Optimization and Machine Learning. Adison Wesley, 1999.

[6] James A. Freeman, David M. Skapura. *Redes Neuronales, Algoritmos, aplicaciones y técnicas de programación.* Adison Wesley, 1993.

[7] Josep Balcells, José Luis Romeral. *AUTÓMATAS PROGRAMABLES.* Alfa-Omega Marcombo, 1997.

[8] Kumpati S.Ñarendra, Kannan Parthasarathy. *Identification and Control of Dynamical Systems Using Neural Networks.* IEEE transactionnson Neural Networks Vol1.Nro1, March 1990.

[9] Ogata, Katushito. *Ingeniería de Control Moderna.* Prentice Hall Hispanoamericana, 1993.

[10] Serrada, Anselmo Pérez. *Una Introducción a la Computación Evolutiva.* Publicación Independiente, 1996.

[11] Wolfgang Banzhaf, PeterÑordin, Robert E. Keller Frank D. Francone. *Genetic Programming, An Introduction.* Morgan Kaufmann Publishers, San Francisco, California, 1998.

[12] www.aircenter.net/gaia.