UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACION

ESTIMACION ADAPTATIVA DE PARAMETROS EN UN SISTEMA DINAMICO NO LINEAL

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN CIENCIAS

(COMPUTACION)

P R E S E N T A:

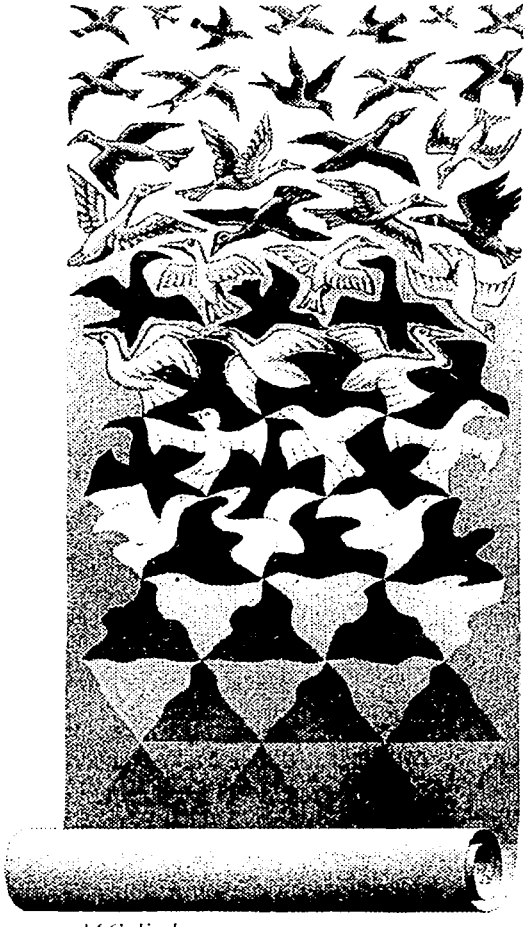
YURIRIA CORTES POZA

DIRECTOR DE TESIS: DR. JOSE LUIS MATEOS TRIGOS

MEXICO, D. F. 2003

Estimación adaptativa de parámetros en un sistema dinámico no lineal

Yuriria Cortés Poza



M.C. Escher

A mi familia
y amigos.

Agradezco a:

José Luis Mateos Trigos
por la dirección y asesoría para
la elaboración de esta tesis.

Fernando Arámbula Cosio,
Germinal Cocho Gil
Katya Rodríguez Vázquez,
y Yu Tang Xu,
por la revisión de este trabajo.

Ma. Lourdes González
por el apoyo recibido a
lo largo de mis estudios.

CONACyT y DGAPA
por la beca otorgada.

Índice

Introducción

En un sistema dinámico existen parámetros que caracterizan su comportamiento. Frecuentemente estos parámetros son desconocidos y no pueden ser medidos directamente, por lo cual, el monitoreo del sistema debe llevarse a cabo mediante observaciones. Estas observaciones son cantidades medibles que dependen de los parámetros del sistema. El mapeo entre los parámetros y las observaciones medidas puede ser llamado el operador de observación. El objetivo entonces es determinar el valor de los parámetros, generalmente utilizando algún criterio de mínimos cuadrados o métodos probabilísticos para determinar la convergencia. Este problema se llama *problema de estimación de parámetros*, y en la mayoria de los casos requiere la solución de un problema de optimización.

La estimación de parámetros es un problema común en diversas áreas de modelado de procesos, tanto en aplicaciones que funcionan en tiempo real (e.g. en robótica), como en aplicaciones que funcionan fuera de línea (e.g. el modelado de sistemas biológicos). Cuando el sistema dinámico es no lineal, el problema de estimación de parámetros se vuelve complicado, y el desempeño de métodos tradicionales, en estos casos, es bastante pobre. La estimación de parámetros en sistemas no lineales es un problema computacional importante con aplicaciones en diversas áreas, tanto científicas como tecnológicas.

En este trabajo estimamos los parámetros de un sistema dinámico no lineal utilizando un método híbrido que combina las redes neuronales con la teoria de los algoritmos genéticos. Desarrollamos una red neuronal que será entrenada con algoritmos genéticos, comúnmente llamada red *neuro-genética* que genera una estimación de los parámetros buscados. Determinamos la convergencia utilizando un criterio de mínimos cuadrados. Este método fue elegido

porque resultó más eficiente para nuestra aplicación que otros (e.g. redes neuronales con retropropagación, o algoritmos genéticos puros).

El método que proponemos se examina bajo diferentes condiciones aplicándolo a un sistema dinámico no lineal, caótico, que en los últimos años ha despertado un gran interés. Trabajamos con una partícula en un potencial periódico asimétrico en el que actúa una fuerza externa armónica dependiente del tiempo. Este sistema es comúnmente llamado Motor Molecular o *Ratchet* en inglés y lo hemos elegido porque exhibe una dinámica caótica y tiene muy variadas aplicaciones en la biología, la química y la ingeniería. Aplicar nuestro método a los motores moleculares, puede ser de utilidad para encontrar los parámetros de alguna trayectoria especial (e.g. la trayectoria que tiene una velocidad de transporte óptimo), lo cual resultaría excesivamente laborioso si se tuviera que hacer una búsqueda exhaustiva en el espacio de parámetros.

La red neuro-genética que hemos desarrollado encuentra el valor de los parámetros del sistema dinámico de forma eficiente y confiable. La información que se requiere para la estimación son únicamente las mediciones tomadas del comportamiento del sistema, es decir, una serie de tiempo de las variables importantes (trayectoria y velocidad de la partícula en el potencial). No se asume conocimiento alguno del operador de estimación como lo hacen métodos tradicionales, y siempre se converge a un óptimo global.

Capítulo I

Antecedentes

En este capítulo se establecen las bases teóricas y los antecedentes históricos de cada uno de los temas que se tratarán en este trabajo.

1.1 Estimación de parámetros

Modelar y predecir el comportamiento de muchos sistemas técnicos es complicado, ya que, por un lado, generalmente están caracterizados por un gran número de variables, parámetros e interacciones, y por otro, la información que se tiene del sistema es limitada. Las técnicas de identificación del sistema, se utilizan en muchos campos de la ciencia y la ingeniería para predecir el comportamiento de sistemas desconocidos. La mayor parte de estas técnicas de identificación de sistemas están basadas en la estimación de parámetros y funciones. Desafortunadamente, estos métodos tienen una explosión combinatoria cuando el número de parámetros aumenta.

La estimación de parámetros es un problema importante dentro del modelado de sistemas dinámicos. En modelos complejos es muy difícil hacerlo dado la gran cantidad de combinaciones posibles de los valores de los parámetros. El procedimiento habitual, utiliza métodos probabilísticos, y podemos encontrar una basta cantidad de literatura introductoria [Bard 74, Beck 77, Gallant 87] y numerosos artículos escritos sobre el tema, aplicando la estimación de parámetros a problemas de diversos campos, como la física, la

biología, la ingeniería, la química, etc. [Jorgensen 01, Park 98, Whigham, 01].

El problema de estimación de parámetros tradicionalmente se resuelve siguiendo los siguientes pasos: se encuentran los posibles rangos donde pueden estar los valores de los parámetros, se determina la sensibilidad de los parámetros y se calibran los más sensibles para obtener la mayor concordancia posible entre los valores observados de las variables más importantes del sistema y las predicciones de las mismas dadas por el modelo.

Cuando el sistema que esta siendo modelado es no lineal, el problema de estimación de parámetros se deteriora conforme aumenta la desviación entre la estimación inicial y el valor real, por lo que en métodos convencionales, es necesario tener una estimación inicial cercana al valor real, lo que generalmente es muy difícil. Diferentes técnicas han sido propuestas por muchos autores para resolver el problema de estimación de parámetros en sistemas dinámicos, sin embargo muchos de ellos tienen grandes limitaciones, sobre todo cuando se trabaja con sistemas no lineales.

1.2 Motores moleculares

En las células vivas existe un movimiento dirigido en ausencia de gradientes espaciales, de potencial, temperatura o concentración, efectuado por las moléculas motoras.

En los últimos años, ha habido un interés muy grande por estudiar y modelar este fenómeno, y en general las propiedades de transporte de los sistemas no lineales que pueden extraer trabajo útil a partir de fluctuaciones fuera del equilibrio. Esto puede ser modelado considerando una partícula Browniana en un potencial periódico, asimétrico, sobre el cual actúa una fuerza externa dependiente del tiempo, cuyo promedio es cero. Como ya se mencionó, este sistema es denominado motor molecular o *ratchet* en inglés.

Los motores moleculares han generado un gran interés en la comunidad científica, principalmente por su conexión con problemas biológicos. Sin embargo mientras tanto, un nuevo conjunto de paradigmas y posibles aplicaciones han sido identificados. Incluyendo en áreas como la cromatografía, efectos cuánticos específicos en gases, en sistemas óptico-eléctricos, en teoría de juegos, etc.

La mayoría de los *ratchets* pueden dividirse en dos categorías: los sistemas en los que las fluctuaciones fuera del equilibrio modulan el potencial en el que se encuentran llamados *flashing ratchets*. En estos sistemas el promedio espacial de las fuerzas impuestas por la estructura es cero en cada instante. En otros sistemas, llamados *rocking ratchets*, la estructura del potencial no cambia, pero una fuerza externa fluctuante de promedio cero actúa en las partículas.

Además del interés por las aplicaciones que tienen los ratchets, entre las cuales están el intentar explicar el transporte unidireccional de los motores moleculares en la biología y la búsqueda de nuevos métodos de separación de partículas, son sistemas de gran interés por sus propiedades dinámicas ya que la fuerza externa aplicada, en combinación con la asimetría del potencial, van a producir una dinámica caótica en el sistema.

Para poder entender la generación de movimiento unidireccional a partir de fluctuaciones fuera del equilibrio, se han construido diversos modelos [Cortés 2000, Magnasco 93, Sarmiento, Larralde 99].

Una revisión reciente del tema se puede consultar en el artículo publicado por Reimann [Reimann 02]. Este sistema se estudiará en detalle en el capítulo II.

1.3 Sistemas dinámicos caóticos

Muchos de los fenómenos que ocurren en la naturaleza pueden describirse utilizando modelos matemáticos lineales, donde lineales significa que el resultado de una acción siempre es proporcional a su causa. Sin embargo, la

mayor parte de los fenómenos en la naturaleza son no lineales. En las últimas tres décadas se han desarrollado métodos y tecnologías que han logrado un progreso significativo en los sistemas no lineales. Una clase entera de fenómenos que no existen en el marco de la teoría lineal se ha vuelto desarrollado, y se denomina *teoría del caos*. El comportamiento caótico describe estructuras irregulares y sumamente complejas en el tiempo y en el espacio que siguen leyes y ecuaciones deterministicas.

1.3.1 Antecedentes Históricos

Aunque la dinámica caótica se conoce desde hace mucho tiempo, su importancia para una amplia variedad de aplicaciones empezó a ser apreciada ampliamente a partir de últimas dos décadas, aproximadamente. En estos años, ha habido un creciente interés tanto en la teoría como en aplicaciones en la ingeniería y en la ciencia. El campo continúa creciendo rápidamente en muchas direcciones. Existe una amplia cantidad de bibliografía introductoria sobre el tema [Devaney 92, Ott 92, Peitgen 92].
El estudio de la dinámica caótica empezó con el trabajo del matemático francés Henri Poincaré en 1890. La motivación de Poincaré era poder entender el problema de las órbitas de 3 cuerpos celestiales que experimentan una atracción gravitacional mutua, considerando el comportamiento de las órbitas que surgen a partir de conjuntos de puntos iniciales. Este problema es comúnmente llamado el problema de los tres cuerpos.

Con las ecuaciones desarrolladas por Newton se puede resolver el problema para 1 y 2 cuerpos explícitamente, pero para más de 2 cuerpos, el problema continúa sin ser totalmente resuelto. Poincaré abordó el problema de una forma cualitativa, utilizando técnicas topológicas y geométricas para tratar de entender la estructura global del conjunto entero de soluciones, en vez de tratar de buscar soluciones individuales. De esta forma, Poincaré demostró que eran posibles órbitas muy complicadas (ahora llamadas caóticas).

Entre los trabajos importantes en la teoría del caos, después del trabajo de Poincaré, están el trabajo en dinámica caótica de G. Birkhoff en 1920, ML Cartwright y J.E. Littlewood en 1940, el trabajo de Smale en 1960 y el de Kolmogorov y sus colaboradores.

A pesar de todo el trabajo desarrollado, la teoría del caos y concretamente, la posibilidad de encontrar comportamiento caótico en sistemas físicos reales, no fue ampliamente aceptado hasta años recientes, donde gracias al desarrollo tecnológico que ha habido en las últimas dos décadas, se han podido encontrar soluciones numéricas de sistemas dinámicos no lineales en computadoras digitales, produciendo esto un gran interés en esta disciplina, en miembros de la comunidad científica de diversas áreas.

Utilizando estas soluciones, el carácter caótico de la evolución en el tiempo, en situaciones de importancia práctica se ha vuelto muy claro. Por otro lado, utilizando métodos numéricos, la complejidad de la dinámica no puede adjudicarse a errores experimentales, como puede ser el caso cuando se trabaja con sistemas físicos reales.

1.3.2 Sensibilidad ante condiciones iniciales

Un atributo que define un sistema cuya dinámica es caótica, es que tiene una dependencia exponencialmente sensible a las condiciones iniciales. Esto significa que conforme pasa el tiempo, pequeños errores en la solución pueden crecer exponencialmente en el tiempo, por lo tanto, después de un cierto periodo, efectos como el ruido y el redondeo de la computadora, pueden cambiar radicalmente la solución. Así, dado el estado de un sistema caótico, su futuro es difícil de predecir después de un cierto tiempo.

El hecho de que en un sistema caótico sea difícil la predicción del estado del sistema, después de un cierto tiempo, tiene consecuencias muy importantes. Dada la dificultad de obtener datos numéricos exactos, surge la pregunta de la validez de estudios numéricos de este tipo de sistemas. Para responder esta pregunta, existe la demostración rigurosa de la propiedad de sombreo (*shadowing*) para ciertos sistemas caóticos. Aunque una trayectoria numérica diverja exponencialmente de la trayectoria real, con la misma condición inicial, existe una trayectoria cierta (i.e. sin error) con una condición inicial ligeramente diferente, que se queda cerca de la trayectoria numérica. Por lo tanto la trayectoria numérica es real [Rasband 91].

Se dice que la dinámica de un atractor es caótica si existe sensibilidad exponencial a condiciones iniciales. Se dice que un atractor es extraño si es fractal. Por lo tanto, caos, describe la dinámica del atractor, y extraño se refiere a la geometría del atractor [Ruelle 89]. Es posible que atractores caóticos no sean extraños (algunos casos de los mapeos unidimensionales) y es también posible que los atractores sean extraños pero no caóticos. Para la mayoría de los casos que involucran ecuaciones diferenciales, el caos y los atractores extraños ocurren juntos [Grebogi 87].

1.4 Redes neuronales

Las redes neuronales artificiales son modelos que imitan la estructura y el funcionamiento del cerebro humano mediante modelos matemáticos. Las redes neuronales resuelven problemas que serían muy difíciles para métodos tradicionales y ya que imitan en forma sencilla el funcionamiento de las redes neuronales biológicas, en muchos casos son mejores para resolver problemas del *mundo real* [Freeman 92, Fausett 94].

1.4.1 Antecedentes históricos

Las redes neuronales se originaron pretendiendo ser un modelo del funcionamiento del cerebro. McCulloch y Pitts en 1943 formularon el primer modelo de red neuronal. Esta red contaba con neuronas digitales, pero no tenía la capacidad de aprender. A partir de entonces ha habido un gran desarrollo en este campo, los trabajos principales se describen a continuación. En [Hertz 91] y [Rojas 96] podemos encontrar una introducción histórica.

El trabajo del psicólogo Donald Hebb: *Organization of Behavior*, afirma que los cambios en los pesos sinápticos (la conexión entre las neuronas) son proporcionales a la activación de las neuronas y describe una regla para actualizar los pesos sinápticos en redes de dos capas, permitiendo así el aprendizaje. Esta fue la base formal de la creación de redes neuronales con capacidad de aprendizaje. Frank Rosenblatt incorporó esta idea de aprendizaje en una red de dos capas y le llamó a esta red *perceptrón*. Rosenblatt formu-

ló una regla de aprendizaje basada en ajustar los pesos en proporción con el error entre la salida de las neuronas de la última capa y la salida deseada, y demostró que de esta manera se obtendría el conjunto deseado de pesos (*Teorema de Convergencia del Perceptrón*). Rosenblatt además formuló un perceptrón de tres capas, pero fracasó en el intento de construir un método de entrenamiento.

Muchos problemas no pueden resolverse con redes de dos capas. La falta de un procedimiento matemático riguroso para permitir el aprendizaje en redes multicapa es un campo abierto en el desarrollo de las redes neuronales.

En 1969, Minsky, y Papert en su trabajo *Perceptrons* indicaban las limitaciones del perceptrón para muchos problemas y planteaban que aunque quizá una red multicapa podría superar estas limitaciones no veían que existiera la forma de encontrar un método para entrenarla, así que concluyeron que el trabajo en el área posiblemente no tendría futuro.

Aunque se escribieron trabajos de interés sobre redes de dos capas, el artículo de Minsky provocó que disminuyera el interés en las redes neuronales. A continuación se citan algunos de los trabajos hechos con redes de dos capas:

- Kohonen en 1984 [Kohonen 84] usó una red de dos capas para construir una memoria asociativa, en la que no es necesario tener un índice para recuperar datos de la memoria. La memoria asociativa está basada en un algoritmo de entrenamiento no supervisado, donde los pesos se ajustan mediante los patrones presentados, sin tomar en cuenta la salida deseada.

- Desde mediados de los años 60, Stephen Grossberg, había estado desarrollando modelos matemáticos del funcionamiento del cerebro [Grossberg 82]. Aunque el trabajo de Grossberg estaba principalmente orientado hacia la investigación en psicología y biología, mucho de su trabajo dio como resultado diversos modelos útiles para redes neuronales artificiales. Los principales avances que produjeron los modelos de Grossberg fueron: el permitir construir redes en las que el entrenamiento fuera en línea, redes con la capacidad de autoorganización y redes con las cuales se pudieran modelar de forma

compacta fenómenos complejos. Uno de los modelos más importantes de Grossberg, en el que el entrenamiento es en línea, es el ART (Adaptative Resonance Theory).

Paralelamente a todo el trabajo realizado para redes con dos capas, Werbos en 1974 y Parker en 1982, encontraron de manera independiente un nuevo método llamado retropropagación, el cual permitía el entrenamiento en redes multicapa. Esto fue un gran avance en el área y abrió de nuevo el campo que se creía muerto.

Rumelhart, Hinton y Williams trabajaron con la retropropagación, simulando procesos cognoscitivos en 1982. Su trabajo se dio a conocer mejor con el trabajo de McClelland y Rumelhart, llamado *Procesamiento Paralelo Distribuido* [McClelland 86].
La retropropagación ha sido utilizada para entrenar redes neuronales aplicadas a diferentes campos. Es una herramienta poderosa y práctica para resolver problemas, difíciles de resolver utilizando métodos computacionales tradicionales. Algunos de los problemas que pueden resolver este tipo de redes son el procesamiento de imágenes, reconocimiento de voz, predicción de series de tiempo, y optimización entre otros. A partir de entonces, han surgido diferentes métodos de entrenamiento, muchos más adecuados que la retropropagación para cierto tipo de aplicaciones.

El progreso que ha habido en utilizar las redes neuronales como una herramienta para resolver problemas ha sido muy grande, sin embargo esta herramienta aun tiene mucho que ofrecer en aplicaciones de muy diversas áreas.

1.4.2 Bases teóricas

Los modelos de redes neuronales pretenden emular el funcionamiento del cerebro humano en sus aspectos mas fundamentales [Amit 89]. Principalmente existen tres características en una red neuronal real que se mantienen en las redes neuronales artificiales. Estas son:

- Las neuronas se pueden comunicar localmente entre sí

- El conocimiento está distribuido sobre muchas neuronas dentro del cerebro
- El cerebro es adaptable

De esta forma, cuando construimos una red neuronal artificial, tendremos:

- Conexiones entre las neuronas (o nodos) que transmitirán la información.

- El aprendizaje que vaya adquiriendo la red, se almacenará de manera distribuida en estas conexiones.

- La red artificial será adaptable. Ajustando las conexiones entre los nodos, irá mejorando su desempeño, es decir, se irá adaptando hasta lograr resolver el problema propuesto.

1.4.2.1 Características

Podemos describir una red neuronal artificial a partir de las siguientes características [Fausett 94]:

A. Unidades neuronales, o unidades de procesamiento
B. La topología de la red
C. El método de entrenamiento

A. Unidad Neuronal

Las neuronas, o unidades de procesamiento son el procesador básico en una red neuronal. Son elementos sencillos que consisten principalmente de cuatro componentes:

- Pesos de Conexión
- Una variable de entrada
- Una función de activación o transferencia
- Una variable de salida

En el siguiente diagrama (Fig. 1) se muestra una unidad neuronal, con sus cuatro componentes básicos:

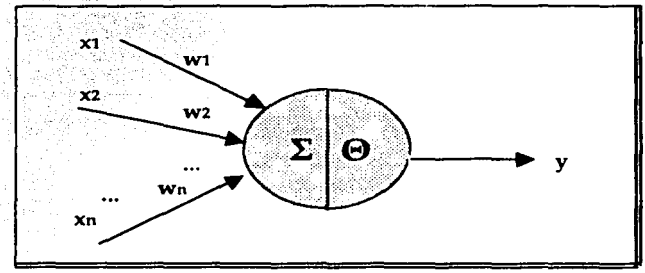


Fig 1. Unidad Neuronal. Donde Θ es la función de activación, **X** el vector de entrada, **W** los pesos de conexión, y la salida, y Σ el operador suma e indica que la salida, se calcula tomando en cuenta las contribuciones de cada entrada, como se verá más adelante.

Una neurona recibe señales mediante diversas conexiones de entrada. A cada una de estas le es asignado un cierto peso. Los pesos utilizados definen la fuerza de acoplamiento (sinapsis) de las conexiones respectivas y se establecen durante el proceso de aprendizaje, donde son modificados de acuerdo con el método de entrenamiento utilizado.

Existen diversos tipos de funciones de activación. La función de activación es generalmente una función simple, no lineal. Ejemplos de esta son:

- La función escalón:

$$f(x) = \{1 \text{ si } x >= 0,\ 0 \text{ si } x < 0\}$$

- La función sigmoide:

$$f(x) = 1/(1+e^{-kx}),\quad k \text{ constante}$$

- La función tangente hiperbólica:

$$f(x) = \tanh(x)$$

B. Topología de la red

Una unidad de procesamiento consiste de elementos homogéneos (neuronas). Estos elementos están interconectados para formar una estructura de red rígida, la cual tiene varias capas: Una capa de entrada, donde actúan las señales de entrada, una capa de salida y las capas intermedias u ocultas que pueden ser una o más.

Las neuronas generalmente están interconectadas completamente entre capa y capa direccionalmente. El número de capas se define al diseñar la red, generalmente la red funciona de una manera óptima con solo tres capas: las capas de entrada y salida y una capa intermedia.

El número de capas ocultas, el número de neuronas en cada capa, y algunos otros factores (e.g. la función de activación) determinan el desempeño de la red. Dependiendo del problema que se quiera resolver, habrá arquitecturas más eficientes que otras.
A continuación (Fig. 2) se muestra la estructura de una red neuronal multicapa.

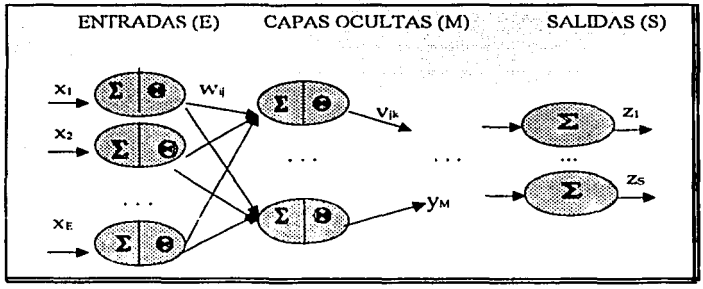


Fig 2. Topología de la red. Donde E, M y S son el número de neuronas en la capa de entrada, ocultas y salida respectivamente, x_i son las entradas de la capa inicial, y z_k son las salidas de l red, w_{ij} y v_{jk} son los pesos de las conexiones, Θ es la función de activación, y $0<i<E$, $0<j<M$, $0<k<S$.

C. Entrenamiento

El *conocimiento* que la red adquiere lo almacena en los pesos de las conexiones entre las neuronas (sinapsis), los cuales pueden representarse en un arreglo matricial. Las matrices de peso se inicializan aleatoriamente y se van ajustando durante el proceso de entrenamiento de la red, de acuerdo con una regla de aprendizaje. Cuando el resultado que proporciona la red (la salida de la última capa) para el problema en cuestión es satisfactorio finaliza el entrenamiento.

Existen diferentes métodos para entrenar una red. Dependiendo del problema que se quiere resolver, algunos métodos resultan más adecuados y eficientes que otros (e.g. [Tenorio 90]).

En general el proceso de entrenamiento es el método mediante el cual se van a ir ajustando los valores de la matriz de pesos de la red, de forma que en cada iteración, la salida producida vaya mejorando.

Algunos de los procesos de entrenamiento que existen son:

i. Supervisados

ii. No supervisados (auto-organizado)

iii. Estrategias estocásticas.

iv. Algoritmos genéticos

A continuación se describe brevemente como funciona cada uno:

i. Aprendizaje supervisado

Además de los datos de entrada, los datos de salida deseados son presentados a la red en la fase de entrenamiento. La red calcula una salida a partir del dato de entrada que compara con el valor deseado. Una señal de error se obtiene de la diferencia entre la salida generada y la deseada, la cual es

usada para modificar los pesos de acuerdo con la regla de aprendizaje. Como resultado la señal de error es reducida.

La red con aprendizaje supervisado, mejor conocida y más empleada es el *Perceptrón Multicapa con Retropropagación.*

ii. Aprendizaje no supervisado

En el aprendizaje no supervisado, la red encuentra un criterio de clasificación para los datos de entrada por sí misma. Intenta descubrir características comunes entre los datos de entrada presentados mediante una comparación de similitudes, y adaptar su estructura de pesos de acuerdo a esto.

iii. Estrategias estocásticas

Los métodos estocásticos de aprendizaje, utilizan procesos aleatorios y distribuciones de probabilidad para minimizar una función definida de energía en la red (e.g. Máquina de Boltzman).

iv. Algoritmos genéticos

Una forma alternativa de entrenar una red neuronal, es utilizando algoritmos genéticos. A este tipo de redes comúnmente se les denomina redes *Neuro-Genéticas.*

Las redes neuro-genéticas se han utilizado en diversas aplicaciones, y se ha visto que son muy eficientes. En este trabajo entrenaremos la red neuronal con un algoritmo genético.

La siguiente sección (1.5) explica las bases de los algoritmos genéticos y más adelante se trata con detalle la forma en la que construimos nuestra red neuro-genética, y su funcionamiento.

1.4.2.2 Algoritmo

Como hemos visto, una red neuronal es una gráfica dirigida formada por varias capas, donde los nodos de cada capa están conectados densamente con los de la capa siguiente, las conexiones (aristas) tienen un peso. Los nodos de la primera capa recibirán los datos de entrenamiento como entrada y la salida de la última capa será el resultado presentado por la red, para el problema que está resolviendo. Mediante el entrenamiento, se espera que la salida de la red sea cada vez mejor hasta llegar a un resultado satisfactorio.

A continuación se describen los pasos mas importantes que se siguen en una red neuronal de tipo perceptrón multicapa.

A. Inicialización

Los pesos de las aristas se pueden almacenar en una matriz (matriz de pesos). Como primer paso es necesario asignarle un valor inicial a cada elemento de la matriz de pesos. Esto se hace eligiendo un número aleatorio dentro de un cierto rango, generalmente [0, 1]. Posteriormente estos pesos se irán ajustando de manera que la red tenga un desempeño óptimo.

B. Entrada de la red

A la red se le presenta una entrada de entrenamiento, dada por datos que le proporcionen información útil del sistema que se esta tratando.La entrada será la salida de la capa inicial. Habrá un dato por cada nodo en la capa inicial.

C. Propagación de la señal

La propagación de la señal consiste en transmitir la información que va generando cada capa a la siguiente, hasta producir la salida de la red (salida de la última capa). La salida de la capa inicial está dada por la entrada de la red (ver inciso anterior), ésta, junto con los pesos de conexión entre la capa inicial y la primera capa oculta, producirá una señal, utilizando la función de activación elegida, se decidirá si ésta señal es lo suficientemente fuerte, como para pasar a la siguiente capa. Así se repite el proceso para cada capa

hasta llegar a la última, produciendo así una salida de la red, que será una aproximacion al resultado buscado.

D. Evaluación

La evaluación consiste en dar una medida de desempeño a una red neuronal, definida por sus matrices de pesos. Esta medida se calcula tomando en cuenta la salida producida por la red. La forma de evaluar la salida de la red depende del tipo de red que estemos utilizando y de la aplicación. A partir de esta evaluación se entrena a la red (ajuste de los valores de las matrices de pesos) hasta que la evaluación sea satisfactoria, y entonces, el algoritmo termina. Como ya mencionamos existen muchos métodos para entrenar una red, dependiendo del problema que estamos resolviendo, unos tienen mejor desempeño que otros.

1.4.2.3 Pseudocódigo

A continuación se enlista el pseudocódigo del proceso general que implementa una red neuronal:

```
COMIENZA RED NEURONAL

  E → valor fijo      //número de nodos de entrada
  M → valor fijo      // número de nodos en la capa media
  S → valor fijo      // número de nodos en la capa de salida

  inicializa matriz de pesos
  asigna valores de entrada a los nodos de la capa inicial

  REPITE
    se propaga la señal y se produce una salida
    la salida de la red será una aproximación al resultado
    se evalúa el resultado obtenido
    se entrena la red mediante el método elegido
  HASTA que el resultado sea satisfactorio

TERMINA
```

1.5 Algoritmos genéticos

Los algoritmos genéticos son algoritmos de búsqueda basados en la mecánica de la selección natural y la genética. Combinan la supervivencia del más apto, con un intercambio de información estocástico, para formar un algoritmo de búsqueda. En cada generación un nuevo conjunto de individuos artificiales es creado sin perder información de los individuos más aptos de la generación anterior [Michell 98].

1.5.1 Antecedentes históricos

Los algoritmos genéticos fueron desarrollados por John Holland y sus colaboradores en la Universidad de Michigan [Holland 75, 81]. Las metas de su investigación eran por un lado abstraer e intentar explicar rigurosamente los procesos adaptativos de sistemas biológicos y por otro lado diseñar sistemas computacionales artificiales, tomando como base los principios de los procesos adaptativos de los sistemas naturales. Esta investigación ha producido avances importantes tanto en los mecanismos de la biología como en los sistemas artificiales.

Se ha demostrado teóricamente y empíricamente que los algoritmos genéticos son un método de búsqueda robusto en espacios complejos. Cada vez hay más aplicaciones en diversos campos como en la ciencia, en las finanzas, en la ingeniería, etc. [Poli, Langdon 02]

1.5.2 Bases teóricas

Aunque parte del funcionamiento de los algoritmos genéticos es aleatorio, no son una simple caminata al azar. Los algoritmos genéticos explotan la información históricamente para especular en nuevos puntos de búsqueda con un mejor desempeño esperado.

A partir de una población inicial generada aleatoriamente, cada individuo es evaluado, y se le asigna una medida de desempeño. De acuerdo a esto, se hace una selección de los mejores individuos, con los cuales se realizan operaciones de cruza y mutación para crear la siguiente generación. Este proceso se repite hasta encontrar un individuo con un desempeño satisfactorio.

Existen muchas formas de implementar un algoritmo genético [Brarnlette 91]. En una gran cantidad de problemas, un algoritmo genético simple da muy buenos resultados [Goldberg 89].

1.5.2.1 Operadores

Los operadores de un algoritmo genético simple son:

A. Inicialización
B. Selección (con elitismo)
C. Cruza
D. Mutación
E. Evaluación

Se describen a continuación:

A. Inicialización

La población se inicializa de manera aleatoria. Cada individuo va a representar una posible solución del problema. Generalmente el individuo es representado codificando su valor en binario. De esta forma el individuo será una serie de bits (unos y ceros) (Fig. 3).

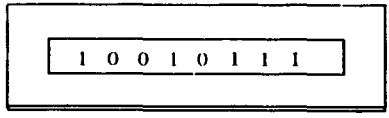


Fig 3. Individuo

B. Selección

La selección que llevaremos a cabo será mediante el *Método de la Ruleta* (Fig. 4). Cada individuo de la población tiene una cierta probabilidad de ser elegido (proporcional a su medida de desempeño), la suma de las probabilidades de todos los individuos es 100%. De esta forma los individuos con mejor desempeño tendrán una mayor probabilidad de ser elegidos que los individuos con un desempeño menor. Puede haber individuos que sean seleccionados más de una vez, e individuos que no sean seleccionados para la siguiente generación.

Para hacer el algoritmo más eficiente se utiliza elitismo en la selección, lo cual significa que el mejor individuo de la generación siempre va a pasar a la siguiente, al menos una vez.

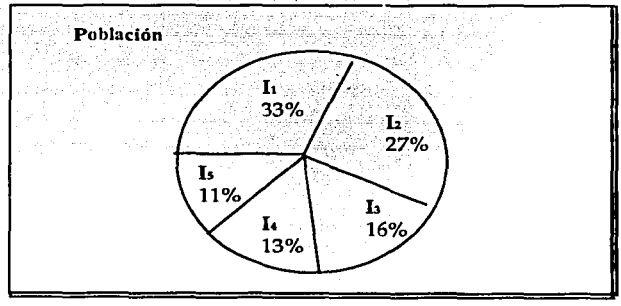


Fig 4. Selección: método de la ruleta

C. Cruza

Se seleccionan dos individuos, y se elige un punto de cruza *pc* aleatoriamente en el primer individuo. se parten ambos individuos en pc, y se concatena el inicio del primero con el final del segundo y viceversa (Fig. 5). Este proceso se repite hasta que un porcentaje PCruza, de la población haya sido cruzado.

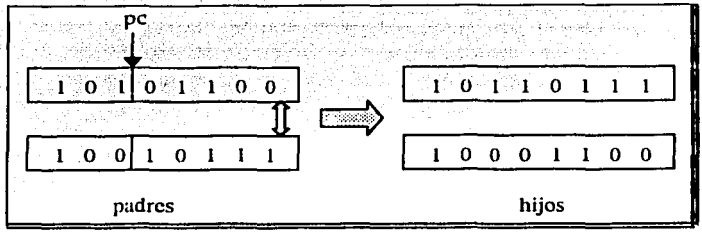


Fig 5. Cruzamiento

D. Mutación

Para llevar a cabo la mutación, se va a seleccionar un individuo de la población. El individuo será modificado en un punto *pm*. Este punto es elegido aleatoriamente y se cambia por una constante proporcional a su valor original, o si el individuo está codificado en bits, se le aplica el operador lógico de negación al (los) bit(s) elegidos (Fig. 6). Este proceso se repite hasta que un porcentaje PMuta haya sido mutado.

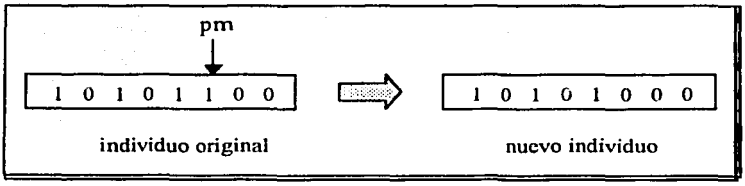


Fig. 6 Mutación

E. Evaluación

Cada individuo de una generación debe ser evaluado (que tan bueno es como solución al problema). Es decir, se calcula el error que produce el individuo, como solución al problema que está siendo tratado, y en base a esto se le asignará una medida de desempeño, comúnmente llamada *fitness*.

En la primera iteración, los individuos generados aleatoriamente deberán de ser evaluados. Una vez evaluada toda la población se procede a aplicar los operadores de los algoritmos genéticos: selección, cruza y mutación. Esto, como ya se dijo, generará una nueva población. Este nuevo conjunto de individuos será nuevamente evaluado y se vuelve a repetir el proceso. El algoritmo se detiene cuando se encuentra un individuo cuyo desempeño es satisfactorio (produce un error mínimo).

1.5.2.2 Pseudocódigo

El pseudocódigo de un algoritmo genético simple es el siguiente:

```
COMIENZA ALGORITMO GENETICO

  N   ← valor fijo      //número de individuos por generación
  PCruza ← valor fijo  //porcentaje de individuos a cruzar
  PMuta ← valor fijo   //porcentaje de individuos a mutar

  inicializa N individuos aleatoriamente

  REPITE

    evalúa cada individuo de la población
      selecciona N individuos

      REPITE
            elige dos individuos para cruzar
            cruza individuos
      HASTA alcanzar un PCruza de individuos cruzados

      REPITE
            elige un individuo para mutar
            muta individuo
      HASTA alcanzar un PMuta de individuos mutados

    HASTA que se encuentre un individuo satisfactorio

  TERMINA
```

II. Motores moleculares deterministas

Las fluctuaciones fuera de equilibrio, pueden generar movimiento dirigido en una estructura sin gradientes de temperatura o fuerzas macroscópicas, simplemente mediante movimiento Browniano. Los sistemas que operan bajo este principio se llaman comúnmente Ratchets Brownianos.

Los ratchets Brownianos se han estudiado como una posible explicación del funcionamiento de motores moleculares, y pueden tener aplicaciones en otros campos como en la separación de partículas, o en el diseño de motores a nano-escalas.

Aunque en algunos trabajos se considera la presencia de ruido [Riemann 2002], muy recientemente, ha habido motivaciones para entender con detalle las propiedades de transporte de ratchets deterministicos determinadas por lo general por una dinámica caótica clásica.

2.1 Modelo

Trabajaremos con el problema uni-dimensional de una partícula a la cual se le aplica una fuerza externa dependiente del tiempo, bajo la influencia de un potencial periódico asimétrico. El promedio en el tiempo de la fuerza externa es cero. En este trabajo no se toma en cuenta ningún tipo de ruido, por lo que la dinámica es determinista [Jung 1996]. A continuación se analizará el modelo introducido por Mateos (2000) y seguiremos la notación de este artículo.

La ecuación de movimiento está dada por:

$$m(d^2x/dt^2) + \mu(dx/dt) + dV(x)/dx = F_0 \cos(\omega_D t) \quad (1)$$

donde m es la masa de la partícula, μ el coeficiente de fricción, V(x) el potencial periódico asimétrico externo, F_0 la amplitud de la fuerza externa aplicada, y ω_D la frecuencia de la fuerza.

El potencial del motor molecular está dado por:

$$V(x) = V_1 - V_0 \text{sen}[2\pi(x-x_0)/L] - (V_0/4) \text{ sen } [4\pi(x-x_0)/L] \quad (2)$$

donde L es la periodicidad del potencial, V_0 la amplitud y V_1 una constante arbitraria (Fig. 7). El potencial se desplaza una cantidad x_0 de tal forma que el mínimo del potencial se encuentre en el origen.

Fig. 7 Potencial asimétrico dado por la ecuación 2

Definamos las siguientes cantidades adimensionales:

$$\begin{aligned} x' &= x/L \,, \\ x_0' &= x_0/L \,, \\ t' &= \omega_0 t \,, \\ w &= \omega_D/\omega_0 \,, \\ b &= \mu/m\omega_0 \,, \\ a &= F_0/mL\omega_0^2 \end{aligned} \quad (3)$$

La frecuencia ω_0 está dada por :

$$\omega_0^2 = 4\pi V_0 \lambda / mL^2 \quad (4)$$

λ es una constante definida por:

$$\lambda = \text{sen}(2\pi\,|x_0'|) + \text{sen}(4\pi\,|x_0'|) \qquad (5)$$

ω_0 es la frecuencia del movimiento linearizado alrededor del mínimo del potencial, por lo que estamos escalando el tiempo con el periodo natural del movimiento: $\tau_0 = 2\pi/\omega_0$.

La ecuación adimensional de movimiento, después de renombrar las variables, se vuelve:

$$d^2x/dt^2 + b(dx/dt) + dV(x)/dx = a\cos(wt)\,, \qquad (6)$$

y el potencial adimensional es:

$$V(x) = C - 1/(4\pi^2\lambda)\,(\text{sen}[2\pi(x-x0)] + \tfrac{1}{4}\,\text{sen}[4\pi(x-x0)]) \qquad (7)$$

C es una constante que se calcula de tal forma que V(0)=0, y está dada por:

$$C = -[\text{sen}(2\pi x0) + \tfrac{1}{4}\text{sen}(4\pi x0)]/4\pi^2\lambda \qquad (8)$$

En este caso $x \approx -0.19$, $C \approx 0.0173$

De esta forma obtenemos tres parámetros adimensionales: a, b, y w.

- El parámetro **a** es la razón entre la amplitud de la fuerza externa aplicada y la fuerza producida por el potencial V(x).
- El parámetro **b** es el coeficiente adimensional de fricción y
- **w** es la razón entre la frecuencia de la fuerza externa aplicada y $\omega 0$.

Despejando m de la ecuación [4] y sustituyendo en $a = F0/mL\omega0^2$ obtenemos que:

$$a = F_0 L/(V_0 4\pi^2\lambda) \qquad (9)$$

Por lo que se puede observar que excepto por un factor constante, a es la razón de F_0 y la fuerza V_0/L, donde V_0 es la amplitud y L la periodicidad del potencial.

Ya que la ecuación de movimiento:

$$d^2x/dt^2 + b(dx/dt) + dV(x)/dx = a\cos(wt)$$

es no lineal, obtendremos trayectorias periódicas y caóticas.

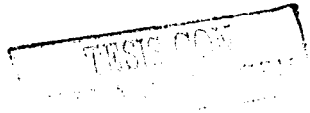
2.2 Trayectorias periódicas y caóticas

Para diferentes valores de los parámetros del sistema (a, w y b) tendremos trayectorias con estructura periódica o caótica.

Las trayectorias con estructura periódica tendrán corriente positiva (el movimiento de la partícula en el potencial tiende a ser hacia la derecha) o corriente negativa (el movimiento tiene hacia la izquierda).

En la figura 8 se muestra un ejemplo donde la partícula tiene una trayectoria con estructura periódica con corriente positiva, como podemos observar, después de un cierto tiempo t, la partícula repite su comportamiento. En este caso podríamos predecir la posición de la partícula. Además se muestra la velocidad de la partícula en el potencial, también periódica.

En la figura 9 observamos un ejemplo donde la estructura de la trayectoria de la partícula es caótica. Se observa como la particula inicia con movimiento irregular con corriente positiva, luego negativa, y así continúa, sin ningún tipo de periodicidad. Igual que en el caso anterior, se grafica la velocidad de la partícula, también caótica.



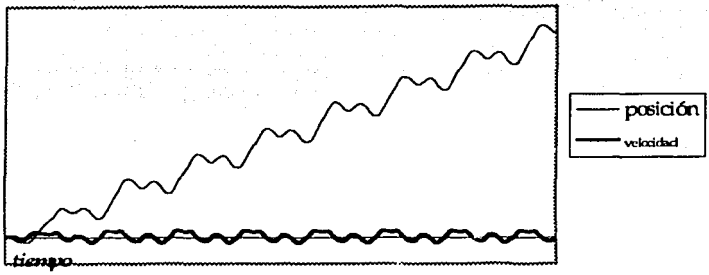


Fig. 8 Trayectoria periódica

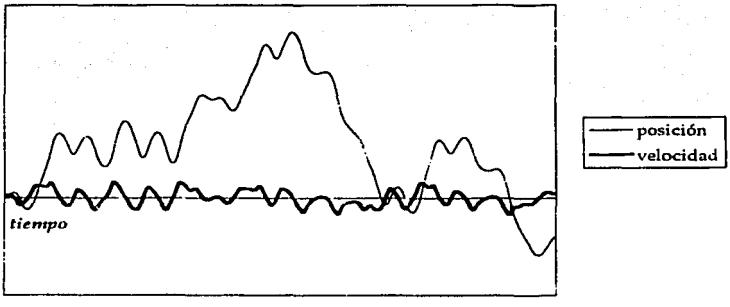


Fig. 9 Trayectoria caótica

El espacio fase extendido en el cual se lleva a cabo la dinámica del motor molecular es de tres dimensiones, ya que estamos trabajando con una ecuación diferencial no homogénea con una dependencia explícita en el tiempo.

Graficando la velocidad de la partícula como función de un parámetro de control (Fig. 10, tomada de [Mateos 02]), se obtiene el diagrama de bifurcación asociado con la dinámica caótica, mediante el cual podemos conocer las regiones de parámetros donde el sistema es periódico o caótico.

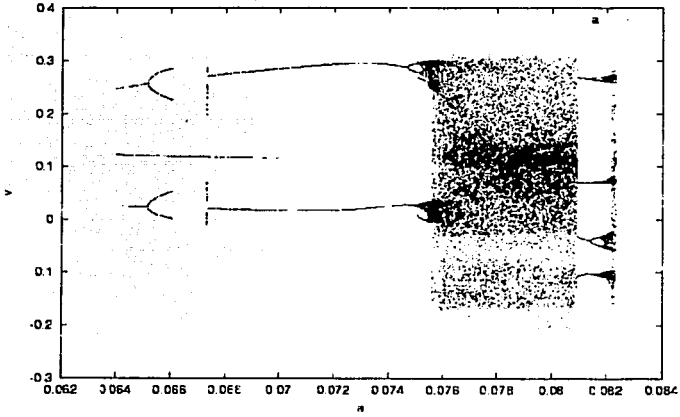


Fig. 10 Diagrama de bifurcaciones

2.3 Pseudocódigo

La ecuación de movimiento (6), se puede escribir como un sistema dinámico de tres dimensiones que se resuelve numéricamente usando Runge-Kutta de cuarto orden. El pseudocódigo es el siguiente:

COMIENZA RUNGE-KUTTA

b → valor fijo //parámetro fijo
x → valor inicial // posición inicial
v → valor inicial // velocidad inicial
Δt → valor fijo // intervalo de tiempo
δ → 1.6 // constante
N → 1000 // número de iteraciones

REPITE

$-dV/dx = 1/(4\pi\delta)\,[2\cos(2\pi\,(x-x0))+\cos(4\pi\,(x-x_0))]$
$f_0 = a\cos(wt)$ //amplitud
$k_{11} = v\ \Delta t$
$k_{12} = \Delta t\ [dV/dx + f_0-(bv)]$
$-dV/dx = 1/(4\pi\delta)[2\cos(2\pi(x+\frac{1}{2}k_{11}-x_0))+\cos(4\pi(x+\frac{1}{2}k_{11}-x_0))]$
$f_0 = a\cos[w(t+\frac{1}{2}\Delta t)]$
$k_{21} = (v+\frac{1}{2}\,k_{12})\,\Delta t$
$k_{22} = [dV/dx+f_0-(b\,(v+\frac{1}{2}\,k_{12}))]\,\Delta t$
$-dV/dx = 1/(4\pi\delta)[2\cos(2\pi(x+\frac{1}{2}k_{21}-x_0))+\cos(4\pi(x+\frac{1}{2}k_{21}-x_0))]$
$k_{31} = (v+\frac{1}{2}\,k_{22})\,\Delta t$
$k_{32} = [dV/dx+f_0-(b\,(v+\frac{1}{2}\,k_{22}))]\,\Delta t$
$-dV/dx = 1/(4\pi\delta)[2\cos(2\pi(x+\frac{1}{2}k_{31}-x_0))+\cos(4\pi(x+\frac{1}{2}k_{31}-x_0))]$
$f_0 = a\cos[w(t+\Delta t)]$
$k_{41} = (v+\frac{1}{2}\,k_{32})\,\Delta t$
$k_{42} = [dV/dx+f_0-(b\,(v+k_{32}))]\,\Delta t$
$x = x+(k_{11} + 2k_{21} + 2k_{31} + k_{41})/6$
$v = v+(k_{12} + 2k_{22} + 2k_{32} + k_{42})/6$
$t = t+\Delta t$

HASTA que se completen N iteraciones

TERMINA

Capítulo III.

Sistema Computacional

En este capítulo se explicara la forma en la que fue construida la red neuro-genética y la aplicación de ésta para resolver el problema de estimación de parámetros en el sistema dinámico no lineal con el que estamos trabajando.

3.1 Planteamiento del problema

El problema que resolveremos consiste en encontrar los parámetros a y w (ecuación 6) de un motor molecular determinista, a partir de una serie de tiempo de la velocidad y la posición de la partícula. Resolveremos esto utilizando una red neuronal entrenada mediante un algoritmo genético.

El problema de estimación de parámetros en términos generales consiste en lo siguiente. Tenemos un modelo de la forma:

$$y_i = f_i(\mathbf{x}, \boldsymbol{\beta}),$$

donde $\mathbf{x}=(x_1, x_2, \ldots, x_n)$ son las variables independientes y $\boldsymbol{\beta}=(\beta_1, \beta_2, \ldots, \beta_m)$, son los parámetros del sistema.
Dada una serie de tiempo y_i, $i=1,2,\ldots,p$, obtenida ya sea experimentalmente o numéricamente, el objetivo es determinar el valor óptimo para los parámetros, es decir, aquellos que se ajusten mejor a las ecuaciones

del sistema. Para esto, generalmente se utiliza un criterio de mínimos cuadrados, que requiere minimizar la función:

$$\zeta(\lambda)= \Sigma\ (y_i - f_i(\mathbf{x}, \lambda))/y_i)^2$$

Esto no es una tarea fácil si el sistema es no lineal. En este trabajo lo haremos utilizando una red neuronal entrenada mediante un algoritmo genético. La red neuro-genética que hemos diseñado, estima los parámetros a y w del sistema, dada una serie de tiempo de la posición y otra de la velocidad de una partícula en este tipo de potencial, ambas series de tiempo tienen 1000 datos. El parámetro b queda fijo en b=0.1.

Como podemos ver en el diagrama de bifurcación (Fig. 10), dependiendo del valor de los parámetros a y w, podemos tener trayectorias con una estructura periódica o caótica, en el espacio fase.

En este trabajo, resolvemos el problema de estimación, utilizando una red neuro-genética, es decir, una red neuronal entrenada mediante un algoritmo genético. En el siguiente diagrama (Fig. 11) se muestra la interacción entre la red neuronal, el sistema dinámico y el algoritmo genético, lo cual se explicará detalladamente en las siguientes secciones.

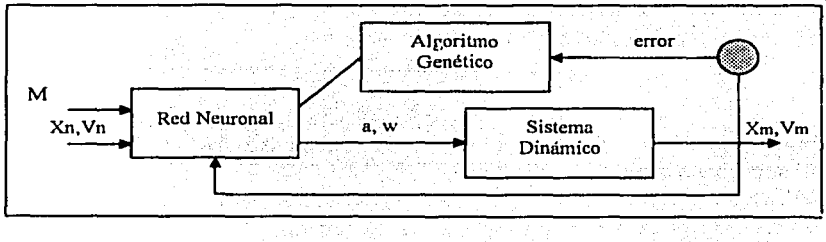


Fig. 11. Funcionamiento del Sistema. Donde $Xn=(x1,x2,...,xn)$ es la posición de la partícula y $Vn=(v1,v2,...,vn)$ es la velocidad. M es la matriz de pesos de la red, la salida de la red Fo y m son los parámetros estimados.

3.2 Red neuronal

La red que utilizaremos es un perceptrón de tres capas (capa inicial, media y final). Una red esta definida por los pesos de conexión entre las capas, así, tendremos una matriz de pesos entre la capa inicial y la capa media, y una matriz de pesos entre la capa media y la final. La capa inicial recibirá una entrada, que le proporcionará información útil del sistema dinámico con el que esté trabajando, esta información, junto con los pesos de conexiones, producirá una salida, que será evaluada. Para mejorar la salida, es necesario ajustar los valores de las matrices de pesos, esto se lleva a cabo en el entrenamiento, el cual se hará en este caso mediante un algoritmo genético. A continuación se describe con detalle cada uno de estos procesos.

3.2.1 Estructura de la red

- Capa inicial: Compuesta por 10 neuronas (nodos). Cada nodo de esta capa recibirá como entrada un valor de la posición de la partícula en el tiempo t. Elegimos 10 neuronas para esta capa, por ser un número mediante el cual obtenemos suficiente información del sistema sin que disminuya la eficiencia del sistema.
- Capa media: Compuesta por 4 neuronas. El encontrar el número óptimo de neuronas para la capa media, en general se hace de forma muy empírica. Si son demasiadas neuronas, la velocidad del proceso disminuye, ya que el número de operaciones que se deben de llevar a cabo aumenta. Si el número de neuronas es muy poco el sistema tardará más en converger.

- Capa final: Compuesta por 2 neuronas. Cada neurona de esta capa producirá una salida, que eventualmente debe de ser la solución buscada, i.e., los dos parámetros estimados.

La red que construiremos tiene la siguiente arquitectura (Fig. 13) :

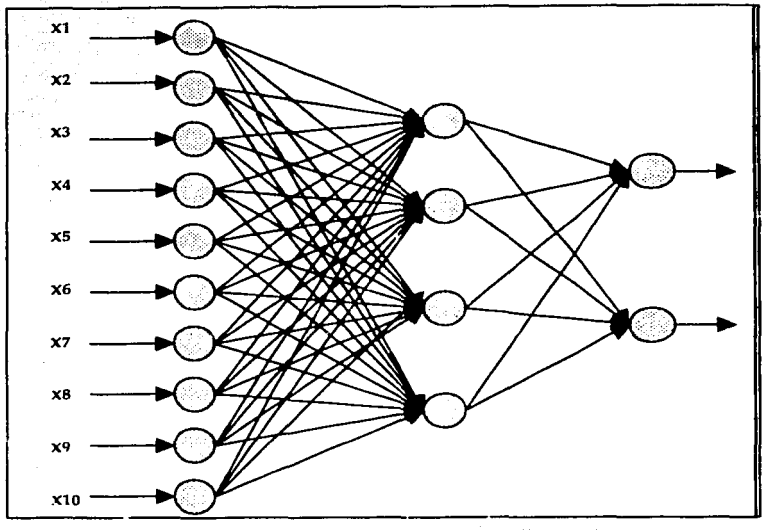


Fig. 12. Estructura de la red

Así, tendremos una matriz de pesos U de 10x4 entre la capa inicial y la capa media y una V de 4x2 entre la capa media y la capa final (Fig 12).

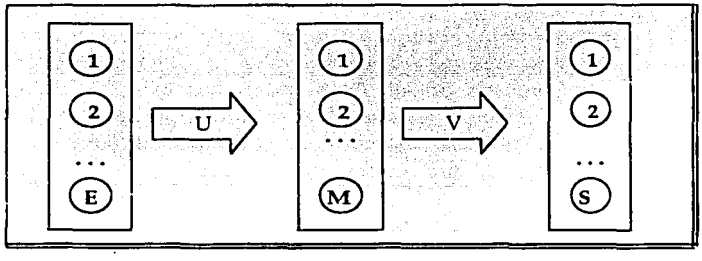


Fig 13. Matrices de pesos

donde:

$$\mathbf{U}=\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1M} \\ u_{21} & u_{22} & \cdots & u_{2M} \\ \cdots & \cdots & \cdots & \cdots \\ u_{E1} & u_{E2} & \cdots & u_{EM} \end{bmatrix}_{ExM}, \qquad \mathbf{V}=\begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1M} \\ v_{21} & v_{22} & \cdots & v_{2M} \\ \cdots & \cdots & \cdots & \cdots \\ v_{M1} & v_{M2} & \cdots & v_{MS} \end{bmatrix}_{MxS}$$

son las dos matrices de pesos (Fig. 13) y en este caso E=10, M=4 y S=2.

La función de activación θ utilizada es la función sigmoide:

$$\theta(x)=1/(1+e^{-kx}), \qquad k=3$$

3.2.2 Algoritmo

El algoritmo de una red neuronal está dado por los siguientes pasos:

A. Inicializar los pesos de las aristas
B. Fijar la entrada de la red
C. Propagar la señal
D. Evaluación del resultado
E. Entrenamiento de la red

A continuación se da una breve explicación de cada uno.

A. Inicialización

Inicialmente los pesos se generan aleatoriamente y son un número real pe[0,1]. Estas dos matrices, tendrán que ajustarse en el entrenamiento para producir la salida deseada.

B. Entrada de la red

La entrada de la red será una serie de tiempo del sistema dinámico con E (10) datos. En esta caso, la serie de tiempo será de la posición de la partícula en el potencial $\mathbf{X}=[x_1,x_2,...,x_{10}]$. Cada dato será introducido en una neurona de la capa inicial

C. Propagación de la señal

A partir de una matriz de pesos y un vector de entrada, la red neuronal debe de calcular un vector de salida.

Esto se hace de la siguiente forma:

Sea $\mathbf{X}=[x_1,x_2,...,x_{10}]$ el vector de entrada y U_{10x4} y V_{4x2} las matrices de peso entre la capa inicial y la media y la media y la final respectivamente.

La salida de las neuronas de la capa inicial es la entrada de la capa media. Como primer paso debemos calcular la salida de las neuronas de la capa media, $\mathbf{Y}=[y_1,...,y_4]$ que está dada por:

$$Y_j = \theta\,(\Sigma_i\,(x_i\,u_{ij}))$$

donde i=1,2,...,10, y j=1.2,...,4, uij es el elemento i, j de la matriz **U** y $\theta(x)=1/(1+e^{-kx})$, k constante.

La salida de la capa final $Z=[z_1,z_2]$ se calcula de manera similar, en esta ocasión, sin utilizar la función de activación:

$$Z_j = \Sigma_i\,(y_i\,v_{ij})$$

donde i=1,...,4, y j=1,2 y v_{ij} es el elemento i,j de la matriz **V**.

La salida **Z** son los parámetros estimados. En este caso solo tendremos dos neuronas en la capa final y por lo tanto dos datos de salida:

$a = z_1$, $w = z_2$

Esta salida son los parámetros estimados del sistema dinámico. De esta manera, con cada red (matrices U y V) obtendremos una salida diferente. La entrada será la misma para todas.

Para mejorar la salida debemos de ir ajustando los valores de las matrices de pesos. Esto se hace en el entrenamiento.

D. Evaluación

Como ya vimos, la salida de la red, será una estimación de los parámetros buscados. La evaluación consistirá en ver que tan buena es esta aproximación.

E. Entrenamiento

Entrenaremos nuestra red utilizando algoritmos genéticos. En la siguiente sección se explica detalladamente la forma en la que hacemos esto.

3.3 Red neuro-genética

La forma en que entrenaremos la red neuronal será mediante algoritmos genéticos. A este tipo de red se le suele llamar Red Neuro-Genética.

Las redes neuro-genéticas son un método híbrido que combina las redes neuronales con la teoría de los algoritmos genéticos. La tarea del algoritmo genético será entrenar la red, es decir, ajustar los valores de la matriz de pesos hasta que se produzca una salida satisfactoria.

Como ya dijimos, la red utilizada, es un perceptrón de tres capas: la capa inicial, una capa oculta o intermedia y la capa final. El algoritmo

genético utilizado en este trabajo, es un algoritmo genético simple, que cuenta con los operadores: selección con elitismo, cruza y mutación.
Como se vio en el capítulo anterior, los algoritmos genéticos trabajan con un conjunto de individuos, o población. En este caso, cada individuo será una red neuronal, i.e., las matrices de pesos que la definen. No se codifica en este trabajo a los individuos en binario, trabajaremos con datos reales. Así, cada individuo es un conjunto de (E·M)+(M·S) datos, dados por los pesos de ambas matrices (ver Fig. 13). Colocaremos primero los pesos de cada renglón de la matriz **U**, seguidos por los de cada renglón de la matriz **V**.

3.3.1 Algoritmo

Los pasos que se sigue el algoritmo de una red neuro-genética son los siguientes.

A. Inicialización de la población

B. Asignación entrada de la red

C. Propagación de la señal en la red
D. Evaluación del desempeño cada individuo de la población

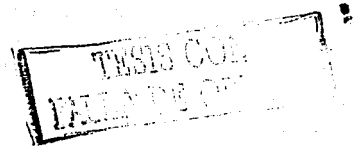
E. Selección

F. Cruza

G. Mutación

H. Elitismo

A continuación se describe brevemente cada paso:



A. Inicialización de la población

Se genera una población inicial, i.e., un conjunto de un número predeterminado de individuos o redes. Cada peso de las matrices de la red será generado aleatoriamente, con un valor real en el intervalo [0,1].

B. Asignación de la entrada de la red

Cada red se alimentará con una entrada, que de alguna forma le de información útil del sistema. En la capa inicial de la red habrá tantas neuronas como datos de entrada tengamos (ver sección 3.3.1). Como se mencionó en la sección anterior, en este trabajo, la entrada que se le proporciona a la red, será una serie de tiempo la trayectoria de la partícula, con diez datos.

C. Propagación de la señal en la red

Cada uno de los individuos, junto con la entrada que se le proporcionará a la red, producirá una salida, al propagar la señal por la red (ver sección 3.3.1). De esta manera, a cada individuo le corresponderá una salida. La salida de la red serán los dos parámetros estimados.

D. Evaluación del desempeño de cada individuo de la población

Evaluaremos a cada individuo, calculando el error que produce en el problema que está siendo trabajado, la salida producida. En base a este error se le asignará una medida de desempeño a cada individuo.

En este caso, para evaluar al individuo en cuestión, se obtiene una serie de tiempo de la posición y la velocidad (100 datos) de la partícula utilizando los nuevos parámetros estimados en la ecuación de movimiento del sistema dinámico. De esta manera, tendremos dos series de tiempo: la original y la calculada con los parámetros estimados.

Calculamos el error entre las dos series utilizando mínimos cuadrados. En base a este error le asignaremos al individuo o matriz de pesos correspondiente una medida de desempeño (fitness).

Este procedimiento se repite para cada individuo de la población.

E. Selección

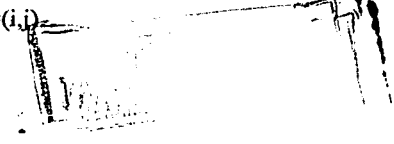
En este momento tenemos una población, donde cada individuo tiene una medida de desempeño. Procedemos ahora a hacer una selección de los individuos que pasarán a la siguiente generación, en este trabajo utilizamos un método de selección estándar proporcional (*método de la ruleta*, ver sección 1.5.2.1-B). Mediante este procedimiento elegiremos una nueva población con el mismo número de individuos que la población anterior.

Cada individuo tendrá una probabilidad de ser elegido proporcional a su medida de desempeño. De esta manera, los mejores individuos tendrán más probabilidad de ser elegidos. Sin embargo, los peores individuos también pueden llegar a ser seleccionados, evitando de esta manera caer en un óptimo local y permitiendo una mayor diversidad en la población.

F. Cruza

Aplicaremos el operador cruza a la nueva población. Como se vio en la sección 1.5.2.1-C , se le aplicará este operador a un porcentaje PCruza de la población. Como ya se mencionó en este caso no estamos trabajando con cadenas de bits, como se hace regularmente en los algoritmos genéticos. Aquí tenemos un conjunto de (E·M)+(M·S) números reales.

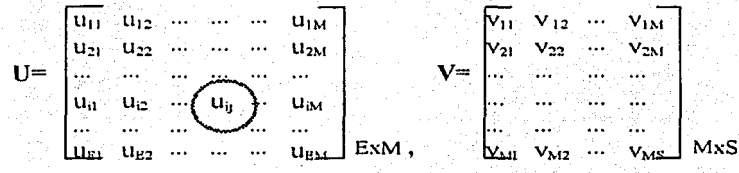
Una vez elegidos los dos individuos I_1, I_2, que serán cruzados, se elige (aleatoriamente) una de las dos matrices **U** ó **V**, donde se encontrará el punto de cruza pc. Posteriormente se seleccionará aleatoriamente un renglón *i* y una columna *j*, por lo que pc=(i,j).



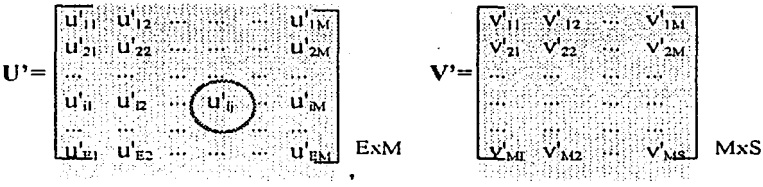
Si se eligió la matriz **U**, entonces i ϵ[1,E] y j ϵ[1,M], donde i, j ϵ**N**, y en otro caso, i ϵ[1,M] y j ϵ[1,S], donde i, j ϵ**N**.

Esto se ilustra con un ejemplo a continuación, donde la matriz seleccionada fue U y pc=(i,j).

Individuo 1:

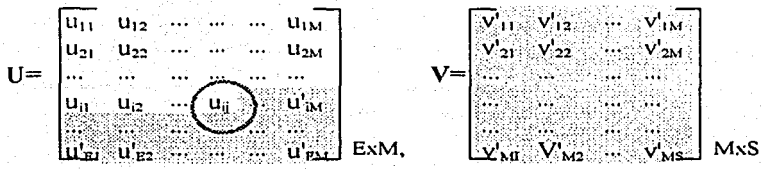


Individuo 2:

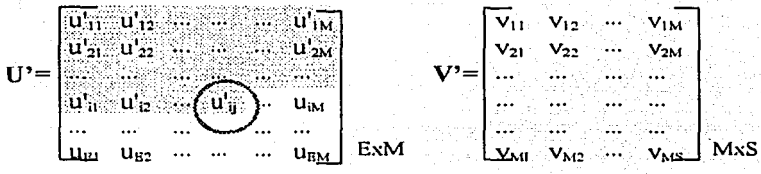


El punto de cruza pc, divide al individuo (compuesto por los datos de ambas matrices) en dos partes. Una vez elegido pc, se procede a hacer la combinación de ambos individuos (I_1 y I_2), concatenando la primera parte de I_1 con la segunda de I_2 y viceversa, formando así, dos nuevos individuos, como se puede ver a continuación (continuando con el ejemplo anterior).

Nuevo individuo 1:



Nuevo individuo 2:



G. Mutación

A continuación se le aplica el operador de mutación a un porcentaje PMuta de la población. Para esto se elegirá aleatoriamente una de las dos matrices que forman el individuo (**U** ó **V**) y posteriormente se selecciona un punto pm=(i,j), donde i es un renglón y j es una columna de la matriz seleccionada. El dato aij será sustituido por una constante proporcional a su valor original, ka_{ij}, donde k es un valor elegido aleatoriamente dentro de un rango generalmente (0, 1).

Esto se ilustra con un ejemplo a continuación. En este caso suponemos que la matriz seleccionada fue **V** y pm=(i,j).

Individuo original:

$$U=\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1M} \\ u_{21} & u_{22} & \cdots & u_{2M} \\ \cdots & \cdots & \cdots & \cdots \\ u_{i1} & u_{i2} & \cdots & u_{iM} \\ \cdots & \cdots & \cdots & \cdots \\ u_{E1} & u_{E2} & \cdots & u_{EM} \end{bmatrix}_{ExM}, \quad V=\begin{bmatrix} v_{11} & v_{12} & \cdots & \cdots & \cdots & v_{1M} \\ v_{21} & v_{22} & \cdots & \cdots & \cdots & v_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & v_{ij} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ v_{M1} & v_{M2} & \cdots & \cdots & \cdots & v_{MS} \end{bmatrix}_{MxS}$$

Nuevo individuo:

$$U=\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1M} \\ u_{21} & u_{22} & \cdots & u_{2M} \\ \cdots & \cdots & \cdots & \cdots \\ u_{i1} & u_{i2} & \cdots & u_{iM} \\ \cdots & \cdots & \cdots & \cdots \\ u_{E1} & u_{E2} & \cdots & u_{EM} \end{bmatrix}_{ExM}, \quad V=\begin{bmatrix} v_{11} & v_{12} & \cdots & \cdots & \cdots & v_{1M} \\ v_{21} & v_{22} & \cdots & \cdots & \cdots & v_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & kv_{ij} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ v_{M1} & v_{M2} & \cdots & \cdots & \cdots & v_{MS} \end{bmatrix}_{MxS}$$

donde k es una constante elegida aleatoriamente dentro de un cierto rango.

Elitismo

Al llevar a cabo la selección vamos a conservar siempre el mejor individuo de la población. De esta manera el individuo de la nueva población será por lo menos tan bueno como el mejor de la anterior.

Iteraciones

Después de aplicar los operadores cruza y mutación a la población, obtendremos una nueva población. Los individuos que no hayan sido modificados, pasarán a la nueva generación íntegramente. Con estos nuevos individuos repetiremos los incisos B, C y D. Si alguno de los individuos tiene un desempeño satisfactorio, el algoritmo se detiene, si no, continúa el proceso (incisos E, F y G).

PCruza y PMuta son constantes que se eligen al construir la aplicación. Si el porcentaje de mutación PMuta, es muy grande, puede provocar que se pierda información histórica sobre los individuos, si es muy pequeño, puede caerse en un óptimo local, ya que los individuos producidos por cruzamiento, no producen información nueva.

En esta implementación elegimos PCruza=80% y PMuta=20%, ya que fueron los valores que produjeron los mejores resultados.

Ya que cada individuo esta compuesto por 48 números reales (elementos de matriz), la probabilidad de mutación para cada dato es de 0.4% (0.2(1/48)≈0.004), que coincide con el porcentaje de mutación que generalmente se utiliza.

A continuación (Fig. 14) se muestra un esquema del funcionamiento del algoritmo genético en conjunción con la red neuronal y el sistema dinámico a tratar.

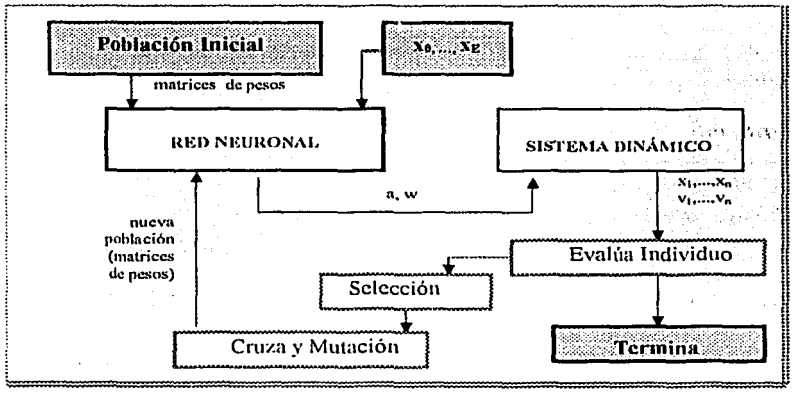


Fig. 14. Entrenamiento de la red

Como primer paso la red neuronal recibirá un conjunto de matrices de pesos (individuos), y una entrada (serie de tiempo de la posición de la partícula en el potencial). Con cada individuo se calculará una salida (parámetros estimados a y w), que pasará al sistema dinámico. El sistema producirá una serie de tiempo de la posición y la velocidad con los parámetros a y w estimados. Esta serie de tiempo se comparará con la *real* y en base a esto se evalúa al individuo. Se aplican los operadores selección, cruza y mutación a todo individuo de la población, esto produce una nueva población, con la que trabajara la red neuronal nuevamente. Este proceso se repetirá hasta encontrar un individuo cuyo desempeño sea el deseado.

3.3.2 Pseudocódigo

El pseudocódigo de una red neuro-genética es el siguiente:

```
COMIENZA RED NEURO-GENETICA

  E → valor fijo    //número de nodos de entrada
  M → valor fijo    // número de nodos en la capa media
  S → valor fijo    // número de nodos en la capa de salida
  N ← valor fijo    //número de individuos por generación
  PCruza ← valor fijo  //porcentaje de individuos a cruzar
  PMuta ← valor fijo   //porcentaje de individuos a mutar

  PARA cada individuo de la población
    inicializa los pesos de ambas matrices (U y V)

  se asigna la entrada a los nodos de la capa inicial

  REPITE

    PARA cada individuo de la población
      se propaga la señal en la red obteniendo así una salida
      se evalúa la salida asignándole una medida de desempeño

    se hace una selección de individuos (método de la ruleta)

        REPITE
              elige dos individuos para cruzar
              cruza individuos
        HASTA alcanzar un PCruza de individuos cruzados

        REPITE
              elige un individuo para mutar
              muta individuo
        HASTA alcanzar un PMuta de individuos mutados

  HASTA que se encuentre un individuo satisfactorio

TERMINA
```

3.4 Algoritmo

En este momento hemos explicado como se construye cada una de las partes que conforma nuestro sistema computacional: la red neuronal, el entrenamiento mediante un algoritmo genético y el sistema dinámico (motor molecular).

En esta sección, explicaremos como funciona el algoritmo que une estas tres partes, formando así, el sistema que hemos desarrollado.

Los pasos más generales que sigue el algoritmo se enlistan a continuación:

i. Para poder estimar los parámetros deseados (a y w) necesitamos tener una serie de tiempo de la posición y de la velocidad de la partícula. En este caso, estamos trabajando con series de tiempo de 1000 datos, obtenidas numéricamente. Guardamos los parámetros reales (con los que obtuvimos nuestras series de tiempo), únicamente para demostrar que los valores de los parámetros obtenidos por la red tienen un error tan pequeño como nosotros queramos (evidentemente los parámetros reales no se utilizan en el algoritmo). Para futuras aplicaciones puede trabajarse con series obtenidas experimentalmente.

ii. Se genera una población inicial, en este caso de 10 individuos, cada uno de los cuales es una matriz de pesos

iii. Se obtiene la entrada de la red neuronal, que está dada por 10 valores de mediciones sucesivas de la posición y la velocidad de la partícula (los 10 primeros valores de ambas series de tiempo).

iv. A partir de la entrada se calcula una salida, utilizando cada uno de los individuos (matrices de pesos) generados en (i).

v. La red genera dos salidas: los parámetros a y w. Con estos parámetros calculamos la posición y la velocidad de la partícula, iterando 1000 veces el método Runge-Kutta.

vi. Comparamos las series de tiempo originales con las obtenidas utilizando los parámetros estimados y para cada individuo obtenemos un error ei calculado mediante mínimos cuadrados:

$$e_i = \Sigma ((x'_i - x_i)/1000)^2$$

A cada individuo le asignamos una medida de desempeño f (fitness), utilizando el error:

$$f_i = 1/(e_i + 1)$$

de forma que $f_i \in [0,1]$, por lo que un individuo i que produzca una serie de tiempo con un error e i =0, tendrá una medida de desempeño $f_i = 1$

vii. Una vez que le hemos asignado una medida de desempeño a cada individuo de la población, procedemos a aplicarle los operadores de los algoritmos genéticos, como indicado en la sección anterior. Esto generará una nueva población.

viii. Con la nueva generación repetimos del inciso (iv) al (vii). El ciclo se termina cuando se encuentra un individuo con el desempeño deseado, o si se exceden el número de generaciones (o iteraciones) permitidas, en caso de que el problema no llegara a converger.

3.5 Pseudocódigo

El pseudocódigo del sistema se enlista a continuación.

COMIENZA

- Abre gráficos
- Abre archivos para guardar resultados
- Inicializa Motor Molecular (real)
- Datos Motor Molecular
- Ejecuta Sistema (datos reales)
- Inicializa Red
- Inicializa Motor Molecular (estimado)
- Inicializa Red (último individuo de la población)

PARA cada individuo de la población:
- Inicializa Red
- Inicializa Pesos
- Fija Entrada
- Propaga Red
- Evalúa Individuo

- Encuentra mejor Individuo

REPITE:
- Selecciona nueva población
- Selección cruza
- Muta Individuo

Para cada individuo de la población:
- Propaga red
- Evalúa Individuo

- Elitismo generación
- Reporta resultados

HASTA que se cumpla el criterio de paro o se exceda el número de generaciones permitidas.

- Cierra gráficos
- Cierra archivos

TERMINA

3.5.1 Datos del sistema

La implementación del sistema está hecha en el lenguaje C. El código está incluido en el apéndice D. A continuación se enlistan algunos de los datos importantes que fueron utilizados:

Red Neuronal:

- Número de capas en la red 3
- Número de neuronas en la capa inicial: E=3
- Número de neuronas en la capa media: M=10
- Número de neuronas en la capa final: E=2
- Función de activación: Sigmoide, $\theta(x)=1/(1+e^{-kx})$, k=3
- Valores de los pesos de conexión en el intervalo [0,1]

Algoritmo Genético:

- Número de individuos en la población: POB=10
- Genotipo formado por 48 números reales.
- Probabilidad de cruza: PCruza=80%
- Probabilidad de mutación: PMuta=20%
- Máximo número de generaciones: 5000
- Criterio de paro: desempeño>0.99999

Runge-Kutta:

- Orden: 4to
- Intervalo de tiempo: dt=0.05
- Parámetro b fijo: b=0.1
- Número de iteraciones: N=1000

Los datos de la red neuronal fueron elegidos experimentalmente. Se hicieron pruebas ejecutando el sistema, con diferentes datos y se llegó a la conclusión que los elegidos eran los que producían un comportamiento óptimo. El criterio siempre es, disminuir número de iteraciones, y disminuir el tiempo de cada iteración. Muchas veces estos dos criterios se contraponen, por ejemplo, si aumentamos el número de neuronas en

la capa intermedia, quizá haya una convergencia en un menor número de iteraciones, pero cada iteración tomará mas tiempo. Para elegir los datos de los algoritmos genéticos: número de individuos por población, porcentaje de cruza, y de mutación ocurre algo similar. Si aumentamos el número de individuos, se requieren menos iteraciones para llegar a una solución satisfactoria, sin embargo cada iteración es mas lenta. En los ejemplos tratados en este trabajo (capítulo 4), elegimos como criterio de paro el encontrar un individuo con un desempeño mayor que 0.999999. Esto se puede modificar, dependiendo de las necesidades de la aplicación que le estemos dando al sistema.

3.6 Otros métodos

En este trabajo elegimos para resolver el problema utilizar un método híbrido, que utiliza redes neuronales y algoritmos genéticos para el entrenamiento. Este método resultó sumamente adecuado para la estimación de parámetros que llevamos a cabo.

Ya que nuestro sistema es no lineal y tiene comportamiento caótico para ciertos valores de los parámetros, no fue conveniente utilizar métodos tradicionales (e.g. métodos estadísticos o probabilísticos).

Las alternativas que consideramos para resolver el problema, fueron: redes neuronales entrenadas mediante retropropagación, y algoritmos genéticos puros.

La desventaja que encontramos en utilizar una red neuronal entrenada con retropropagación, es que a diferencia de la red neuro-genética, inicialmente se parte de una sola red (inicializada aleatoriamente), y de ahí debe de llegar al resultado adecuado. Nuestro método parte de 10 redes diferentes, y hace una búsqueda a partir de cada una de ellas. Esto por un lado, hace que cada iteracion sea mas lenta, pues tenemos dos métodos anidados, pero el hecho de llevar a cabo una búsqueda *paralela* hace que el sistema converge mas rápidamente. Así con redes neuro-genéticas tenemos que cada iteración es un poco mas lenta, con la ventaja de que se requieren muchas menos iteraciones. Tomando en

taja de que se requieren muchas menos iteraciones. Tomando en cuenta estos dos factores, pensamos que la red neuro-genética resultaría mas eficiente.

Para decidir entre utilizar un algoritmo genético puro y una red neuro-genética, construimos ambos y los probamos con un problema de estimación de parámetros mas sencillo que el tratado en este trabajo. En el algoritmo genético, los individuos estaban dados por el conjunto de parámetros a estimar. La red neuro-genética funcionaba de la misma manera a la utilizada en este trabajo. Observamos que la red neuro-genética requería muchas menos iteraciones que el algoritmo genético. En el algoritmo genético encontramos que lo óptimo era trabajar con generaciones de 100 individuos, en contraste con la red neuro-genética donde utilizamos solo 10.

La implementación fue hecha en el lenguaje C. No utilizamos bibliotecas o *toolkits* como las de *Mathlab* para los algoritmos genéticos o las redes neuronales, porque por un lado, queríamos implementar funciónes que cubrieran específicamente las necesidades de nuestro sistema, y por otro sabemos que cualquier implementación en C es mucho mas eficiente y tiene un tiempo de ejecución mucho menor.

Capítulo IV.

Resultados

Hicimos mas de 100 pruebas de nuestro sistema para diferentes valores de los parámetros. Se observó que el sistema es confiable y eficiente tanto para trayectorias periódicas como para trayectorias caóticas.

En cada ejecución, el programa utiliza una serie de tiempo de la posición y una de la velocidad de la partícula en el potencial, generadas numéricamente. En base a esto la red estimará los valores del conjunto de parámetros que produjeron estas series. Mostraremos tres ejemplos significativos de diferentes corridas. En el primero se obtiene una trayectoria periódica con corriente positiva, en el segundo una trayectoria periódica con corriente inversa y en el último se produce una trayectoria caótica. Las iteraciones del programa se detienen cuando se encuentra un individuo con un desempeño $f \geq 0.9999990$, lo cual significa que, el error e será $e \leq 10^{-7}$. Si es necesario, se puede obtener un error menor, para lo cual se requerirá un mayor número de iteraciones

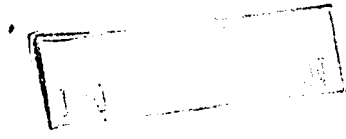
Para cada caso (Figs. 17, 23, 29) graficamos la evolución del desempeño del mejor individuo de cada población a través de las generaciones. Como es característico en los algoritmos genéticos, en cada una de estas gráficas se observa una mejora dramática en las primeras generaciones, posteriormente se aproxima lentamente al 1, siguiendo un comportamiento de saturación. Se observa en los tres casos que se alcanza un desempeño bastante bueno en las primeras 100 generaciones.

Las figuras 18, 24 y 30, nos muestran para cada ejecución, el promedio del desempeño de todos los individuos de la población, en cada genera-

Las figuras 18, 24 y 30, nos muestran para cada ejecución, el promedio del desempeño de todos los individuos de la población, en cada generación. En los tres casos se observa que el promedio es mayor que 0.6. En general a través de las generaciones el promedio fluctúa en un rango no muy amplio, sin tener una tendencia marcada a ni disminuir ni a aumentar. Se considera que se ha alcanzado la meta y se detiene la ejecución del programa, cuando se encuentra un individuo con el desempeño deseado, por lo que no necesariamente todos los individuos de la generación tienen que ser igual de buenos. Sin embargo, sí es necesario que el promedio de cada generación se mantenga dentro de un cierto nivel, ya que de ahí deben de salir los individuos de la siguiente generación (mediante cruzas y mutaciones) y se espera que cada vez el mejor individuo cada generación vaya teniendo un nivel de desempeño más alto.

Finalmente graficamos la desviación estándar del desempeño de los individuos de cada generación (Figs. 19, 25, 31). Observamos que en los tres ejemplos la fluctuación es menor que 0.4, lo cual es bueno, ya que por un lado una fluctuación demasiado pequeña, implicaría una falta de diversidad en los individuos, lo que podría provocar que se cayera en un óptimo local, y por otro lado fluctuaciones demasiado grandes implicarían que nuestra búsqueda es equivalente a una búsqueda estocástica.

4.1 Trayectoria periódica



En este ejemplo se muestra el comportamiento de la red cuando trabaja con una trayectoria periódica. En la figura 15 se grafica la posición como función del tiempo, se observa que la corriente es positiva. En la figura 16 se grafica la velocidad de la partícula como función del tiempo, la cual, en este caso, tiene un comportamiento periódico de periodo 2.

Ya que la trayectoria que sigue la partícula es periódica, la trayectoria producida con los parámetros reales es idéntica a la que se obtiene con

los parámetros estimados (Fig. 15). De igual manera, la gráfica de la velocidad es idéntica en ambos casos (Fig. 16).

En la figura 17, graficamos el mejor desempeño de cada generación, durante las primeras generaciones, que es donde realmente hay un cambio sustancial. Un desempeño mayor que 0.999 se alcanza con 59 iteraciones (generaciones) y uno mayor que 0.9999 se alcanza en la iteración número 110 y para obtener el desempeño deseado son necesarias 134 iteraciones. Estos valores varían en cada ejecución del programa, ya que los pesos de la red son inicializados aleatoriamente.

Los datos obtenidos son:

	a	b	w
Parámetros Reales	0.071000	FIJO (0.100)	0.670000
Parámetros Estimados	0.071003	FIJO (0.100)	0.669997
Error aproximado	$3x10^{-6}$	-	$7x10^{-6}$

Numero de generaciones: 134
Desempeño: 0.9999992
Error: $\leq 10^{-7}$

El error obtenido es calculado comparando ambas series de tiempo (la real y la producida por los parámetros estimados) utilizando un criterio de mínimos cuadrados, y es diferente al error que tienen los parámetros.

En la figura 18 se observa que el valor promedio del desempeño de cada generación está aproximadamente entre 0.5 y 0.7

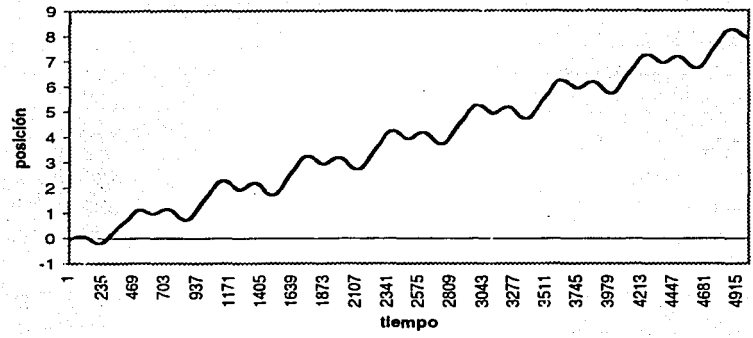


Fig 15. Trayectoria periódica: posición contra tiempo

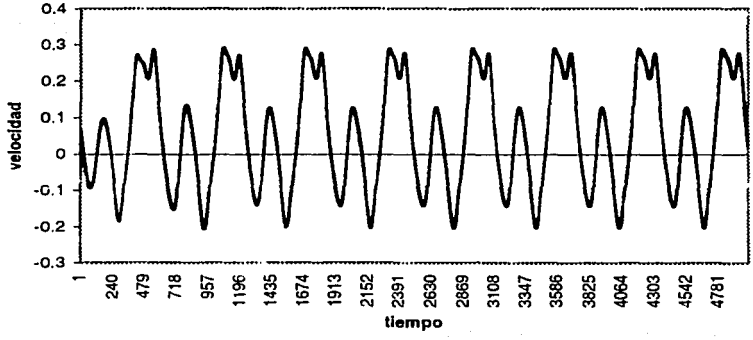


Fig 16. Trayectoria periódica: velocidad contra tiempo

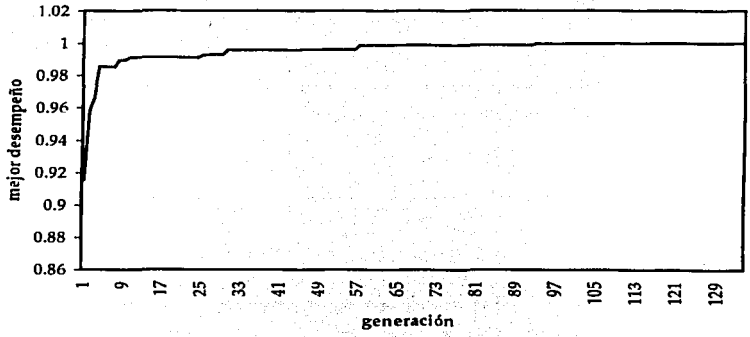


Fig 17. Trayectoria periódica: mejor desempeño de la población

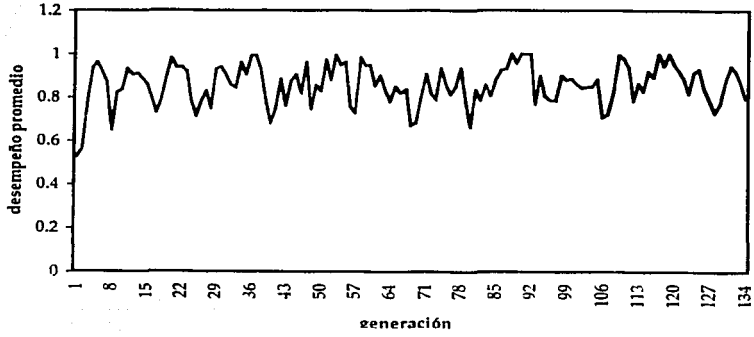


Fig 18. Trayectoria periódica: desempeño promedio de la población

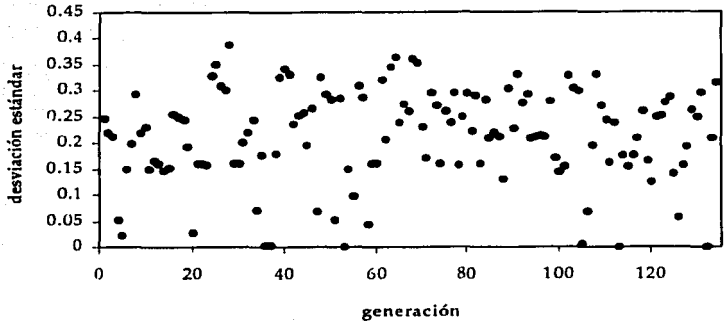


Fig 19. Trayectoria periódica: desviación estándar

Para probar que nuestro algoritmo es confiable llevamos a cabo mas de cien ejecuciones del programa con los mismos datos, en la siguiente grafica (Fig. 20) mostramos los resultados de 20 corridas diferentes elegidas al azar, obteniendo en promedio 409.05 generaciones necesarias para obtener el resultado deseado (un desempeño mayor que 0.999999), donde el mejor resultado es de 134 generaciones y el peor es 639. El algoritmo tiene un comportamiento estable, ya que en ninguna ejecución se requirió un número excesivamente grande de iteraciones.

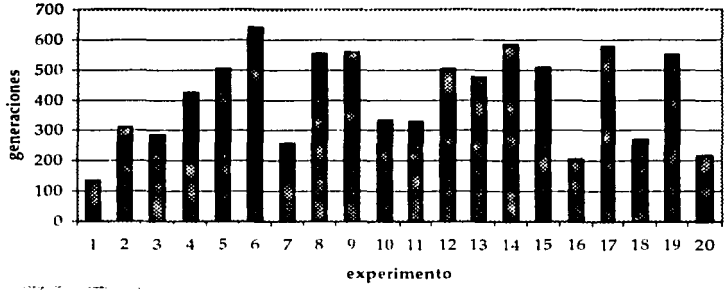


Fig 20. Trayectoria periódica: número de generaciones por experimento

4.2 Trayectoria periódica con corriente inversa

En este caso elegimos una trayectoria periódica, cuya corriente es negativa, se grafica su trayectoria como función del tiempo (Fig. 21) y la velocidad como función del tiempo (Fig. 22). En este caso tiene un comportamiento periódico de periodo 4.

Al igual que en el ejemplo anterior, la trayectoria producida con los parámetros reales es idéntica a la que se obtiene con los parámetros estimados (Fig. 21) y la gráfica de la velocidad también es idéntica en ambos casos (Fig. 22).

El comportamiento de la gráfica del mejor desempeño (Fig. 23) es similar al del ejemplo anterior. En este caso, el desempeño deseado se obtiene después de 206 generaciones. Un desempeño mayor que 0.999 se da después de 94 iteraciones, uno mayor que 0.9999 se alcanza en la iteración número 185, y un desempeño de 0.99999 se alcanza en 198. En este caso se mantiene en un desempeño cercano a 0.96 hasta la generación 66, a partir de la cual se inicia una mejora más dramática.

En las figuras 24 y 25 se muestra el promedio del desempeño de cada generación y la desviación estándar del desempeño respectivamente. Observamos que el desempeño promedio se encuentra aproximadamente entre 0.6 y 1 a través de las distintas iteraciones.

Los datos obtenidos son:

	A	B	w
Parámetros Reales	0.082000	FIJO (0.100)	0.670000
Parámetros Estimados	0.081992	FIJO (0.100)	0.670012
Error aproximado	8×10^{-6}	-	1×10^{-5}

Numero de generaciones: 206
Desempeño: 0.9999999
Error: $\leq 10^{-7}$

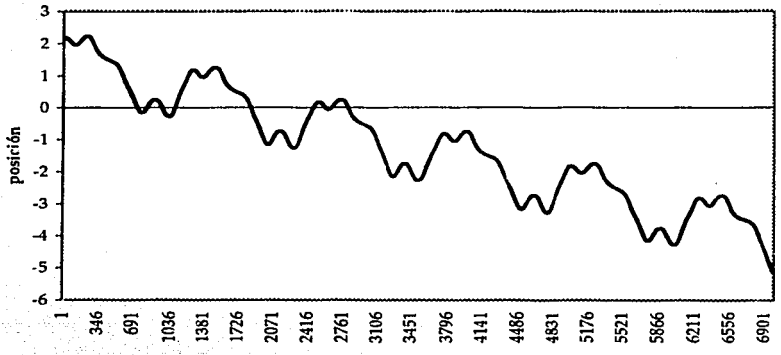


Fig 21. Trayectoria periódica con corriente inversa: posición contra tiempo

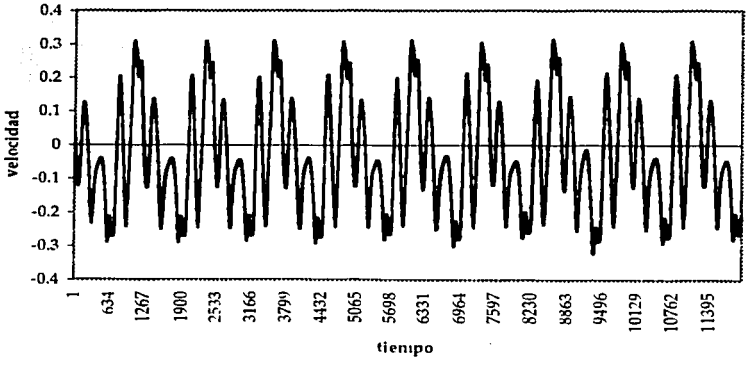


Fig 22. Trayectoria periódica con corriente inversa: velocidad contra tiempo

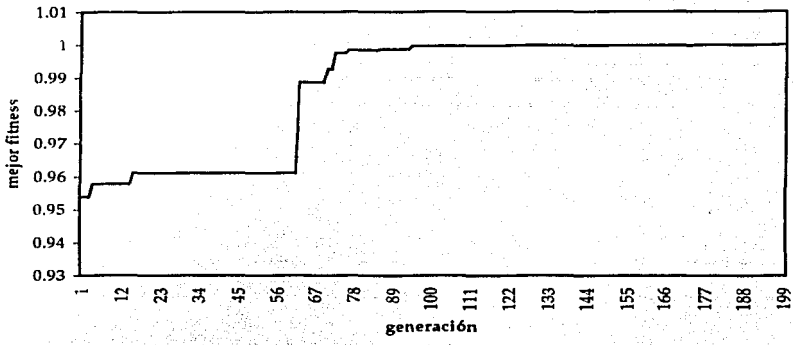


Fig 23. Trayectoria periódica con corriente inversa: mejor desempeño de la población

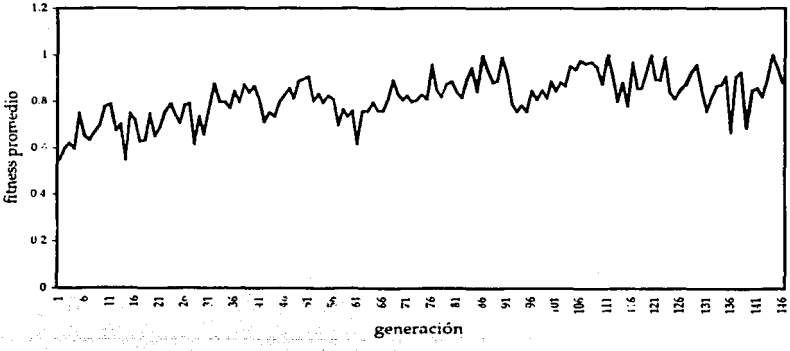


Fig 24. Trayectoria periódica con corriente inversa: desempeño promedio de la población

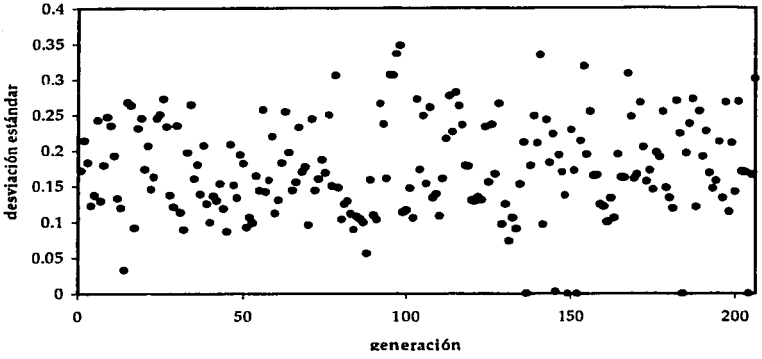


Fig 25. Trayectoria periódica con corriente inversa: desviación estándar de la población

Para este conjunto de datos también realizamos varias ejecuciones. De nuevo tomamos 20 al azar, y mostramos los resultados en la siguiente gráfica (Fig. 21). En este caso la mejor corrida se hizo en 206 generaciones y la peor en 693. El promedio de generaciones (o iteraciones) necesarias para alcanzar la meta propuesta es 369.10.

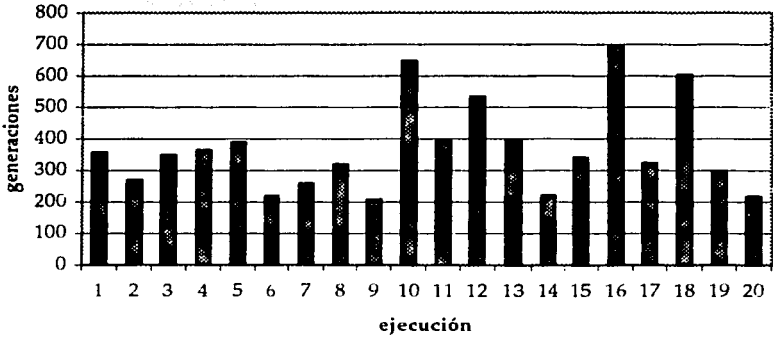


Fig 26. Trayectoria periódica con corriente inversa: número de generaciones por ejecución

4.3 Trayectorias caóticas

A continuación trabajaremos con una trayectoria caótica. En este caso, observamos una divergencia, después de un cierto tiempo, entre la trayectoria real y la trayectoria obtenida con los parámetros estimados (fig. 27), lo mismo ocurre cuando graficamos la velocidad en función del tiempo (fig. 28).

Como ya mencionamos, los ratchets son un sistema dinámico no lineal, por lo que para ciertos valores de los parámetros tendremos trayectorias caóticas, como ocurre en este caso. Un sistema caótico tiene una sensibilidad muy alta a ligeros cambios en las condiciones iniciales o en los parámetros. Podemos observar que el error de los parámetros obtenidos es muy pequeño (parecido al del caso periódico). Sin embargo, después de un cierto tiempo, la trayectoria obtenida con los parámetros estimados diverge de la trayectoria *real* (fig. 27). El tiempo en que permanecen juntas las dos trayectorias aumenta conforme disminuye el error, y esto se logra aumentando la precisión del criterio de paro, lo que aumentara el número de iteraciones (generaciones) y por lo tanto el tiempo de ejecución. De manera análoga, en la figura 28 observamos la gráfica de la velocidad *real* en función del tiempo y la obtenida con los parámetros estimados. Ambas gráficas divergen aproximadamente en el mismo tiempo en el que se separan las dos trayectorias (fig. 27).

En la figura 29 podemos observar la evolución del desempeño del mejor individuo. En este caso se llega a un individuo con el fitness deseado después de 145 generaciones. Un fitness de 0.99999 se alcanza en 130 generaciones y uno de 0.9999 en 130.

Las figuras 30 y 31 donde se observa el promedio del fitness por generación y la desviación estándar, muestran un comportamiento semejante al ocurrido en el caso de las trayectorias periódicas.

Los datos obtenidos son:

	a	b	w
Parámetros Reales	0.080800	FIJO (0.100)	0.670000
Parámetros Estimados	0.080801	FIJO (0.100)	0.670032
Error aproximado	10^{-6}	-	3×10^{-5}

Numero de Generaciones: 145
Fitness: 0.9999992
Error: $\leq 10^{-7}$

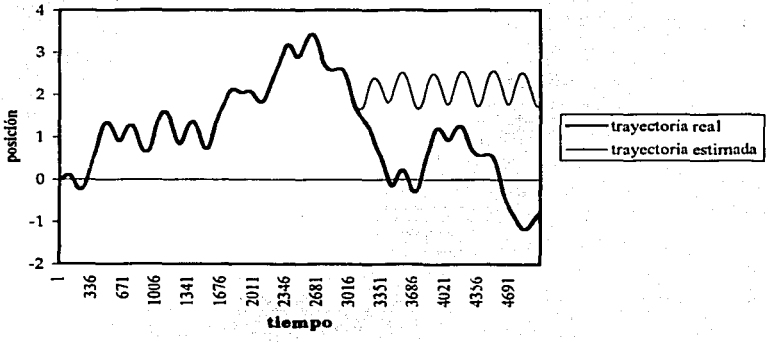


Fig 27. Trayectoria caótica: posición contra tiempo

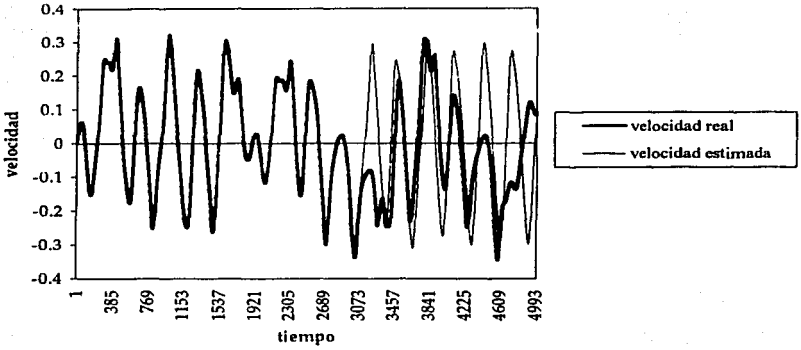


Fig 28. Trayectoria caótica: velocidad contra tiempo

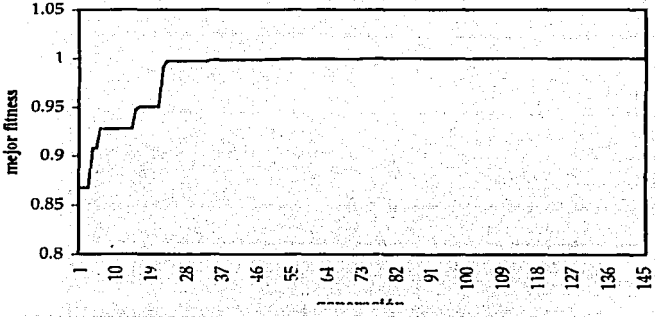


Fig 29. Trayectoria caótica: mejor fitness de la población

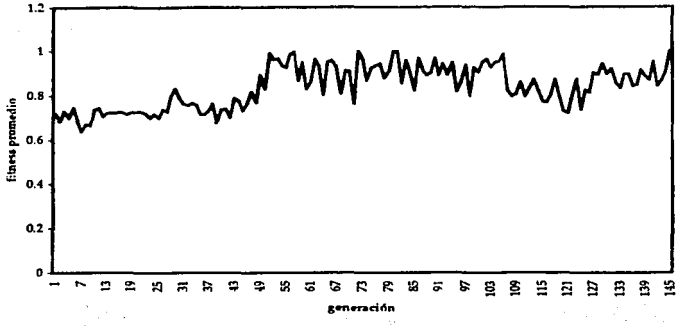


Fig 30. Trayectoria caótica: fitness promedio de la población

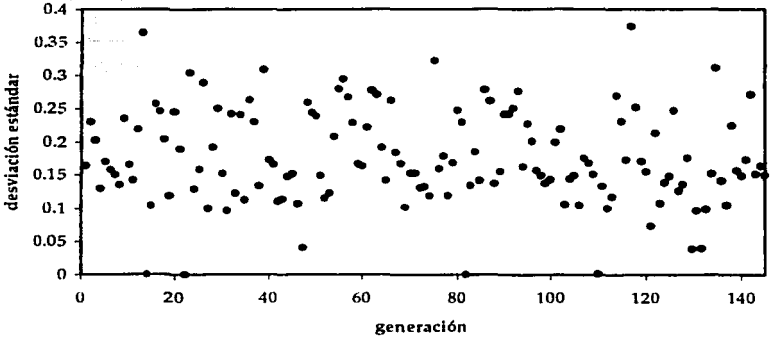
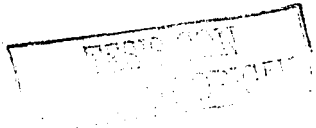


Fig 31. Trayectoria caótica: desviación estándar



Igual que en los dos casos anteriores, probamos nuestro programa con estos datos llevando a cabo múltiples ejecuciones. En la siguiente gráfica (fig. 32) observamos el desempeño en 20 casos diferentes. El promedio de iteraciones necesarias para alcanzar la meta es 398.05, la mejor ejecución se da en 145 iteraciones y la peor en 717. El desempeño del programa es parecido con trayectorias periódicas que con trayectorias caóticas.

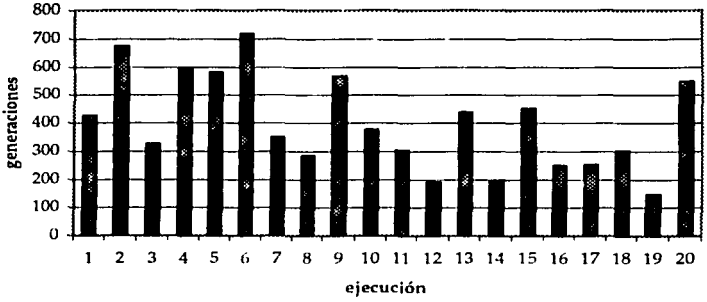
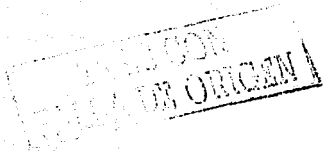


Fig 32. Trayectoria caótica: número de iteraciones por generación


ORIGIN

Capítulo V

Discusión y conclusiones

Conforme ha avanzando el desarrollo de las teorías y arquitecturas de las redes neuronales y los algoritmos genéticos, el alcance de sus aplicaciones ha crecido rápidamente.

Existen muchas aplicaciones computacionales donde se trabaja con procesos dinámicos que no se pueden resolver de manera eficiente mediante algoritmos secuenciales y por lo tanto son especialmente difíciles de abordar con métodos tradicionales. Herramientas como las redes neuronales y los algoritmos genéticos, utilizan un mecanismo *inteligente* que no requiere modelos matemáticos precisos o parámetros accesibles de sistemas físicos.

El lograr una estimación confiable de los parámetros de un sistema, puede ser necesario en muchos problemas de la ciencia, la ingeniería y las ciencias sociales. La estimación de parámetros de un sistema dinámico no lineal es un problema computacional complejo, para el cual las redes neuronales y los algoritmos genéticos resultan especialmente útiles.

En este trabajo se construyó una red neuronal entrenada mediante algoritmos genéticos capaz de estimar los parámetros de un sistema dinámico no lineal, con comportamiento caótico. Existen muchos problemas en diversas disciplinas, modelados mediante un sistema dinámico no lineal donde es de gran utilidad estimar los parámetros a partir de una serie de tiempo de alguna variable del sistema. La red que implementamos en

este trabajo se puede, en principio, adecuar a cualquier problema de este tipo y estimará los parámetros eficientemente.

Consideramos utilizar algún otro método (redes neuronales con retro-propagación o algoritmos genéticos puros) pero llegamos a la conclusión de que el método mas eficiente para resolver nuestro problema en particular, era utilizar una red neuro-genética.

Como sabemos, tanto las redes neuronales como los algoritmos genéticos muchas veces se construyen de forma *artesanal*, por lo que la elección de ciertos parámetros como número de capas, número de neuronas por capas, número de individuos por población, porcentaje de individuos que serán cruzados y mutados, etc., se eligen experimentalmente. El criterio siempre es, disminuir número de iteraciones, y disminuir el tiempo de cada iteración estos dos criterios frecuentemente se contraponen. En este trabajo hicimos pruebas variando estos datos, hasta encontrar un conjunto que datos que hicieran que nuestro sistema trabajara de manera óptima

Para probar el funcionamiento de la red neuro-genética que construimos, se tomó como caso de estudio un *ratchet* determinista, un sistema de una partícula en un potencial periódico asimétrico, sobre la cual actúa una fuerza externa armónica. Este sistema complejo, altamente no lineal y no integrable, es de sumo interés tanto por sus aplicaciones biológicas como por la dinámica que presenta.

La red estima de manera eficiente y confiable los parámetros en este sistema, tanto en los casos en los que la partícula en el potencial exhibe un comportamiento con cierta periodicidad, como en los casos en donde el comportamiento es caótico.

Como era de esperarse, las trayectorias caóticas reales se separan de las trayectorias obtenidas utilizando los parámetros estimados, que generalmente tienen un error del orden de 10^{-6}. Esto es debido a la alta sensibilidad que tienen los sistemas caóticos a ligeras perturbaciones en las condiciones iniciales o los parámetros en este caso. El tiempo que las trayectorias permanecen juntas puede aumentar si disminuimos el error en la estimación, lo que se logrará con un número mayor de iteraciones

del programa. Dependiendo de la aplicación que se le quiera dar al problema, se decide la precisión que se necesita, tomando en cuenta que mientras más preciso sea el resultado el tiempo de ejecución será más largo.

Consideramos que la contribución de este trabajo, es: por un lado, la creación de un sistema computacional útil para resolver el problema de estimación de parámetros, mostrando de esta manera la gran utilidad de las redes neuronales y los algoritmos genéticos, como una herramienta computacional, capaz de superar otras técnicas computacionales con mucha ventaja, y por otro lado, abrir un camino en la difícil tarea de caracterizar la dinámica de los ratchets, donde, a pesar del gran auge que ha tenido la investigación en el tema en los últimos años, aún queda mucho por hacer.

Dadas las múltiples aplicaciones que tienen los *ratchets*, consideramos que las contribuciones que se puedan hacer en este campo, por medio de nuestro enfoque pueden resultar de suma importancia.

SL

Apéndice A. Estructuras de datos

Para hacer más eficiente y ordenado el sistema, se utilizan las siguientes estructuras en las que se definen nuevos tipos de datos.

i) Estructura de una capa

En esta estructura se define una capa k de una red neuronal. La estructura contiene los siguientes campos:

- Un campo donde se almacenarán los pesos de la capa k, con la siguiente capa (la capa inicial k, con la capa media ó la capa media k con la capa final). Los pesos están dados por números reales, y el campo se define como un doble apuntador a un dato de tipo real, i.e. un arreglo bidimensional de números reales.

- Un campo donde se almacenan las salidas de cada neurona de la capa k. Las salidas están dadas por números reales y el campo se define como un apuntador a un dato de tipo real, i.e. un arreglo de números reales

- Un campo entero donde se indica el número de nodos de la capa.

```
Nombre de la estructura:  capa
Campos:        real    **w     (matriz de pesos)
               real    *s      (salida de la capa)
                       entero  nodos  (número de nodos)
```

ii) Estructura de una red

En esta estructura se define una red neuronal r. La estructura contiene los siguientes campos:

- Un campo donde se almacena la capa inicial de la red. Éste está definido por un apuntador a un dato de tipo capa (definido en (i)), i.e. un arreglo unidimensional de capas.

- Un campo donde se almacena la capa media de la red. También definido por un apuntador a un dato de tipo capa.

- Un campo donde se almacena la salida de la red (i.e. las salidas de cada nodo de la capa final). En este caso serán los parámetros estimados (a y w). Las salidas son números reales y el campo está definido por un apuntador a un dato de tipo real, i.e. un arreglo unidimensional de números reales.

- Un campo real donde se almacena el desempeño (fitness) de la red r.

Nombre de la estructura: red

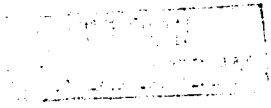
Campos:			
	capa	*inicial	(capa inicial)
	capa	*media	(capa media)
	real	*resultado	(salida de la red)
	real	fitness	(desempeño de la red)

iii) Estructura de un ratchet

En esta estructura se define un sistema de tipo ratchet. La estructura contiene los siguientes campos:

- Un campo donde se almacena la serie de tiempo de la posición de la partícula en el potencial. Se define como un apuntador a datos de tipo real, i.e. un arreglo unidimensional de números reales.

- Un campo donde se almacena la serie de tiempo de la velocidad de la partícula en el potencial, definido igual que en el caso anterior.



- Un campo de tipo real donde se almacena la masa de la partícula.

- Un campo de tipo real donde se almacena la frecuencia de la fuerza externa aplicada a la partícula.

Nombre de la estructura: ratchet

Campos:			
	double	*X	(serie de tiempo de la posición)
	double	*V	(serie de tiempo de la velocidad)
	double	m	(masa de la partícula)
	double	w	(frecuencia de la fuerza externa)

Apéndice B. Descripción de los módulos del sistema

Módulo Principal

En este módulo se encuentra el ciclo principal del programa. Cada iteración de este ciclo es una generación con una nueva población cada vez. Hace las llamadas a los módulos importantes del sistema.

Módulos de la Red Neuronal

1) Inicializa red
   Reserva espacio en memoria para los campos de un dato de tipo red e inicializa los parámetros fitness y resultado en ceros, y fija el número de nodos de las capas inicial y media.

2) Fija entrada
   Le asigna los valores de entrada a la red.

3) Propaga Red
   Propaga la señal de la red, llamado a las funciones (4) y (5).

4) Propaga Media
   Propaga la señal de la capa inicial a la capa media, es decir, calcula la salida de la capa media.

5) Propaga Final
   Propaga la señal de la capa media a la capa final, es decir, calcula la salida de la capa final.

Módulos del Algoritmo Genético

1) Inicializa individuos
Inicializa a cada individuo (matriz de pesos) de la población, i.e., le asigna un valor aleatorios a cada peso dentro de la matriz.

2) Evalúa individuo
Evalúa a un individuo, asignándole una medida de desempeño (*fitness)* según su desempeño. Esto lo lleva a cabo comparando la serie de tiempo que se obtiene con los parámetros estimados, con la serie de tiempo buscada y calcula el error utilizando mínimos cuadrados. A partir de este error se calcula el fitness.

3) Selección
Selecciona por el método de la ruleta a los individuos que pasan a formar parte de la nueva generación.

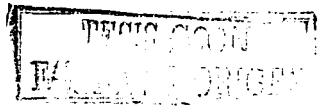
4) Selección Cruza
Selecciona aleatoriamente a dos individuos que serán cruzados.

5) Cruza Individuos
Cruza dos individuos, dividiéndolos en un punto elegido aleatoriamente y concatenando el principio del primero con el final del segundo y viceversa.

6) Muta Individuos
Muta un individuo modificando un de los pesos de su matriz, multiplicándolo por una constante aleatoria.

7) Elitismo Generación
Sustituye al peor individuo de la generación por el mejor individuo de la generación anterior, si el nuevo mejor individuo es mejor que el anterior, lo sustituye.

8) Mejor Individuo



Guarda el mejor individuo de la población en el ultimo lugar del arreglo de individuos.

Módulos del Sistema Dinámico (Ratchet)

1) Inicializa Ratchet
Reserva espacio en memoria para todos los campos de un dato de tipo ratchet y fija los parámetros m y w.

2) Datos Ratchet
Asigna los valores al ratchet *original.*

3) Ejecuta Sistema
Aplica el método Runge-Kutta de cuarto orden a la ecuación de movimiento del sistema dinámico *ratchet.*

Módulos adicionales

1) Ordenamiento
Ordena los individuos (matrices de pesos) según su fitness, utilizando el método de la burbuja (bubble sort)

2) Búsqueda
Dada la medida de desempeño relativa de un individuo, lo busca en la población utilizando una búsqueda binaria.

3) Inicializa Aleatorio
Inicializa la semilla para la generación de números aleatorios.

4) Genera Aleatorio
Genera un número aleatorio entre una cota inferior y superior.

5) Reporte

Escribe a archivos los datos que utilizaremos después para analizar los resultados.

6) Dibuja ejes

7) Inicializa gráficos

Apéndice C. Código del programa


```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#include <graphics.h>


/****** MACROS *****/

#define N 1000  //Numero de muestras tomadas para la
                  evaluación
#define POB 10  //Numero de individuos en la población
#define MAXGEN 5000   //Máximo numero de generaciones
#define PMUT .8 //Probabilidad de mutación
#define PCRU .6 //Probabilidad de cruza
#define E 10            //Numero de nodos capa inicial
#define M 4             //Numero de nodos capa media
#define datos 2 //Numero de datos a encontrar
#define PI (2*asin(1))
#define e 2.71

/***** ESTRUCTURAS *****/

    /* Estructura de una capa */
    typedef struct {
          double **W;         //Matriz de pesos
          double *S;          //Salida de la capa
          int nodos;          //Numero de nodos
    }capa;

    /* Estructura de una red */
    typedef struct {
          capa *inicial;      //Capa inicial
          capa *media;                //Capa media
          double *result;     //Salidas de la red
          double fitness;     //Desempeño de la red
    }red;

    /* Estructura del ratchet */
    typedef struct {
          double *X;          //Serie de tiempo posición
          double *V;          //Serie de tiempo velocidad
          double m;           //Masa
          double w;           //Frecuencia
    }ratchet;
```

```
/***** PROTOTIPOS *****/

    void ejecutaSistema(ratchet *RAT, int S, int c);
    void cruzaIndividuos(red *R1, red *R2);
    void inicializaRatchet(ratchet *RAT);
    void igualaRedes(red *R1, red *R2);

/***** DEFINICION DE FUNCIONES *****/

    /* GENERACION DE NUMEROS ALEATORIOS */

    /**
     Inicializa Semilla
     **/
    void inicializaAleatorio()
    {
    srand(4715);
    }

    /**
     Genera numero real aleatorio entre inf y sup
     **/
    double numeroAleatorio(double inf, double sup)
    {
    return ((double)rand()/RAND_MAX)*(sup-inf)+inf;
    }


    /* INICIALIZACION DE LA RED */

    /**
      Reserva espacio en memoria e inicializa los pará-
    metros    de la red
     **/
    void inicializaRed(red *R)
    {
    int i;

    R->inicial=(capa*)malloc(sizeof(capa));
    R->media  =(capa*)malloc(sizeof(capa));

    R->inicial->nodos=E;
    R->media->nodos  =M;

    R->inicial->S=(double*)calloc(E,sizeof(double));
    R->media->S   =(double*)calloc(M,sizeof(double));

    R->inicial->W=(double**)calloc(E,sizeof(double*));
    R->media->W   =(double**)calloc(M,sizeof(double*));

    R->result=(double*)calloc(datos,sizeof(double));
```

```
        for (i=0; i<E; i++)
            R->inicial-
>W[i]=(double*)calloc(M,sizeof(double));
        for (i=0; i<M; i++)
            R->media-
>W[i]=(double*)calloc(datos,sizeof(double));
        for (i=0; i<datos; i++)
              R->result[i]=0;
        R->fitness=0;
        }

        /**
         Inicializa los pesos de las matrices de cada capa
         aleatoriamente
         **/
        void  inicializaPesos(red  *R,  double  inf,  double
sup)
        {
        int i, j;

        for(i=0; i<E; i++)
              for(j=0; j<M; j++)
                    R->inicial-
>W[i][j]=numeroAleatorio(inf,sup);
        for(i=0; i<M; i++)
              for(j=0; j<datos; j++)
                    R->media-
>W[i][j]=numeroAleatorio(inf,sup);
        }


        /**
         Inicializa la entrada de la red
         **/
        void fijaEntrada(red *R, double *X)
        {
        int i;

        for (i=0; i<E; i++)
              R->inicial->S[i]=X[i];

        }


        /* EJECUCION DE LA RED NEURONAL */

        /**
         Propaga la señal de la capa inicial a la capa me-
dia
         **/
        void propagaMedia(red *R)
        {
```

```
        int i, j;
        double suma;

        for(i=0; i<M; i++) {
            suma=0;
            for(j=0; j<E; j++)
                suma+=(R->inicial->S[j])*(R->inicial-
>W[j][i]);
                R->media->S[i]=1/(1+pow(e,3*suma));
//sigmoide
            }
        }


        /**
         Propaga la señal de la capa media a la capa final
         **/
        void propagaFinal(red *R)
        {
        int i, j;
        double suma;

        for(i=0; i<datos; i++) {
                suma=0;
                for(j=0; j<M; j++)
                        suma+=(R->media->S[j])*(R->media-
>W[j][i]);
                R->result[i]=suma;
                }
        R->result[0]/=10;
        }


        /**
          Manda llamar a las funciones que propagan la se-
        ñal por  la red
         **/
        void propagaRed(red *R)
        {
        prcpagaMedia(R);
        propagaFinal(R);
        }

        /* OPERADORES DEL ALGORITMO GENETICO */

        /**
          Evalúa a un individuo, asignándole un fitness se-
        gún su desempeño
         **/
          void  evaluaIndividuo(red  *R,  double  *X,  double
        *V, ratchet *RAT)
        {
```

```
        double *Y, temp, errorX=0, errorV=0, error;
        int i;

        RAT->m=R->result[0];
        RAT->w=R->result[1];
        ejecutaSistema(RAT,0,0);
        for(i=0; i<N; i++) {
              temp=X[i] - RAT->X[i];
              temp=pow(temp,2);
              errorX+=temp;
              temp=V[i] - RAT->V[i];
              temp=pow(temp,2);
              errorV+=temp;
              }
        error=(errorX+errorV)/(2*N);
        error=errorV/N;
        R->fitness=1/(error+1);
        }


        /**
         Guarda el mejor individuo de la población en el
último
         lugar del arreglo
         **/
        void mejorIndividuo(red R[POB+1])
        {
        int mejor=0, i, j;

        for (i=1; i<POB; i++)
              if(R[i].fitness > R[POB].fitness) {
                    mejor=i;
                    R[POB].fitness=R[i].fitness;
                    }
        igualaRedes(&R[POB],&R[mejor]);
        }


        /**
          Sustituye al peor individuo de la generación por
        el mejor individuo de la generación anterior, si el
        nuevo mejor individuo es mejor que el anterior, lo
        sustituye
         **/
         void elitismoGeneracion(red R[POB+1])
        {
        int mejor=0, peor=0, i;

        //guarda el índice del mejor y peor individuo
        for(i=0; i<POB-1; i++) {
              if(R[i].fitness > R[i+1].fitness) {
                    if(R[i].fitness>=R[mejor].fitness)
                          mejor=i;
```

```
                    if(R[i+1].fitness<=R[peor].fitness)
                        peor=i+1;
                    }
                else {
                    if(R[i].fitness<=R[peor].fitness)
                        peor=i;
                    if(R[i+1].fitness>=R[mejor].fitness)
                        mejor=i+1;
                    }
                }
                if(R[mejor].fitness >= R[POB].fitness)
                    igualaRedes(&R[POB],&R[mejor]);
                else
                    igualaRedes(&R[peor],&R[POB]);
        }

  /**
     Selecciona por el metodo de la ruleta a la nueva
poblacion.
     (seleccion estandard proporcional)
     El mejor individuo siempre sobrevive.
     **/
  void seleccion(red R[POB], red antR[POB])
  {
       red temp[POB];
       int num=0,i,ind;
       double sum=0,p;


       //Pasamos la poblacion anterior a antR para llenar
R
       for (i=0; i<=POB; i++)
            igualaRedes(&antR[i],&R[i]);

       for (i=0; i<=POB; i++)
            sum += R[i].fitness;

       //Asigna temporalmente el fitness relativo a cada
       // individuo
       for (i=0; i<=POB; i++)
            antR[i].fitness/=sum;
       ordena(antR);
       igualaRedes(&antR[POB],&R[POB]);
       antR[POB].fitness/=sum;
       sum=antR[0].fitness;
        for (i=1; i<=POB; i++){
            antR[i].fitness+=sum;
            sum=antR[i].fitness;
            }
       }
  //Seleccionamos la siguiente generacion dejando el me-
jor individuo
       do {
```

```
            p=numeroAleatorio(0,1);
            ind=busquedaBin(antR,p);
            //Pasa el individuo seleccionado a la si-
guiente generacion
            igualaRedes(&R[num],&antR[ind]);
            num++;
            } while (num<POB);
  }


      /**
        Selecciona a dos individuos de acuerdo con la
      probabilidad de cruza para cruzarlos
       **/
      void seleccionCruza(red R[POB])
      {
      int i, num=0, ind=0;
      double pc;

      for(i=0; i<POB; i++) {
            pc=rand()%100;
            pc/=100;
            if(pc < PCRU) {
                  num++;
                  if(num%2 == 0)
                        cruzaIndividuos(&R[ind],&R[i]);
                  else ind=i;
                  }
            }
      }


      /**
       Cruza dos individuos
       **/
      void cruzaIndividuos(red *R1, red *R2)
      {
      int p1, p2, K, i, j;
      double w[E][M], v[M][datos];

      K=rand()%2; //Elige una capa
      if(K==0) {
            p1=rand()%E;
            p2=rand()%M;
            for(i=p2; i<M; i++) {
                  w[p1][i]=                R1->inicial-
>W[p1][i];
                  R1->inicial->W[p1][i]=R2->inicial-
>W[p1][i];
                  R2->inicial->W[p1][i]=w[p1][i];
                  }
            for(i=p1+1; i<E; i++)
               for(j=0; j<M; j++) {
```

```
                    w[i][j]= R1->inicial->W[i][j];
                    R1->inicial->W[i][j]=R2->inicial-
>W[i][j];
                    R2->inicial->W[i][j]=w[i][j];
                    }
              for(i=0; i<M; i++)
                 for(j=0; j<datos; j++) {
                    v[i][j]=              R1->media->W[i][j];
                    R1->media->W[i][j]=R2->media->W[i][j];
                    R2->media->W[i][j]=v[i][j];
                    }
              }
       else {
              p1=rand()%M;
              p2=rand()%datos;

              for(i=p2; i<datos; i++) {
                    v[p1][i]=                          R1->media-
>W[p1][i];
                    R1->media->W[p1][i]=R2->media-
>W[p1][i];
                    R2->media->W[p1][i]=v[p1][i];
                    }
              for(i=p1+1; i<M; i++)
                    for(j=0; j<datos; j++) {
                          v[i][j]=                     R1->media-
>W[i][j];
                          R1->media->W[i][j]=R2->media-
>W[i][j];
                          R2->media->W[i][j]=v[i][j];
                    }
              }

       }



       /**
         Elige a los individuos que serán mutados de
       acuerdo a la
       probabilidad de mutación y modifica uno de los pe-
sos de
       la matriz elegido aleatoriamente
        **/
       void mutaIndividuo(red R[POB+1], double inf, dou-
blesup)
       {
       double pm;
       int K, p1, p2, i, j, num;

       for(i=0; i<POB; i++) {
              pm=rand()%100;
              pm/=100;
```

```
            if(pm < PMUT) {
               num=rand()%20; //numero de pesos a mutar
               for(j=0;j<num;j++) {
                  K=rand()%2;       //capa que se va a mu-
tar
                  if(K==0) {
                     p1=rand()%E;
                     p2=rand()%M;
                     R[i].inicial->W[p1][p2]=
                  numeroAleatorio(inf,sup);
                              }
                        else {
                              p1=rand()%M;
                              p2=rand()%datos;
                              R[i].media->W[p1][p2]= nume-
                              roAleatorio(inf,sup);
                              }
                        }
                  }
            }

      }
      /**
       Recibe dos redes y copia los datos de la
       segunda en la primera
       **/
      void igualaRedes(red *R1, red *R2)
      {
      int i,j;

      for(i=0; i<E; i++)
            for(j=0; j<M; j++)
            R1->inicial->W[i][j]=R2->inicial->W[i][j];
      for(i=0; i<M;i++)
            for(j=0;j<datos;j++)
                  R1->media->W[i][j]=R2->media->W[i][j];
      R1->fitness=R2->fitness;
      for(i=0; i<datos; i++)
            R1->result[i]=R2->result[i];
      }

      /* FUNCIONES DEL SISTEMA */


      /**
       Inicializa los parámetros del ratchet
       **/
      void inicializaRatchet(ratchet *RAT)
      {
      RAT->X=(double*)calloc(N,sizeof(double));
      RAT->V=(double*)calloc(N,sizeof(double));
      RAT->m=0;
      RAT->w=0;
```

```
}

/**
 Asigna los valores al ratchet original
 **/
void datosRatchet(ratchet *RAT)
{
RAT->m=.072;
RAT->w=.67;
ejecutaSistema(RAT,1,2);
gotoxy(10,1); printf("a:  %f",RAT->m);
gotoxy(10,2); printf("w:  %f",RAT->w);
}

/**
  Runge Kutta de cuarto orden para calcular la tra-
yectoria del ratchet
**/
void ejecutaSistema(ratchet *RAT, int S, int color)
{
double a, w, dt=.05, px, py, temp[2], x=0, p=0,
t=0;  double f, fos, k[4][2], b, delta=1.6, x0=-.19;
int i, j=0, pasos=0, cont=0;
int h0=30,v0=250;
float sh=.5, sv=3.0;

b=.1;
a= RAT->m;
w= RAT->w;

do {
      f=(1/(4*PI*delta))*(2*cos(2*PI*(x-x0))+
      cos(4*PI*(x-x0)));

      fos=a*cos(w*t);


      k[0][0]=dt* p;
      k[0][1]=dt*(f+fos-(b*p));

        f=(1/(4*PI*delta))*(2*cos(2*PI*((x+k[0][0]/2
      )-x0))+cos(4*PI*((x+k[0][0]/2)-x0)));

      fos=a* cos(w* (t+ (dt/2)));

      for (i=0; i<2; i++) {
          k[i+1][0]=dt* (p+(k[i][1]/2));
          k[i+1][1]=dt*                    (f+fos-
(b*(p+(k[i][1]/2))));

            f=(1/(4*PI*delta))*(2*cos(2*PI*((x+k[i
          +1][0]/2)-
          x0))+cos(4*PI*((x+k[i+1][0]/2)-x0)));
```

```
            }

        fos=a*cos(w*(t+dt));

        k[3][0]=dt* (p+ k[2][1]);
        k[3][1]=dt* (f+ fos -(b* (p+k[2][1])) );

        for (i=0;i<2;i++)
              temp[i]=k[0][i]+                    (2*k[1][i])+
              (2*k[2][i])+ k[3][i];

        x+=temp[0]/6;
        p+=temp[1]/6;


        if (S!=0) {
              px=h0+(sh*cont*dt);
              py=v0-(sv*p);
              putpixel(px,py, LIGHTRED);
              px=h0+(sh*cont*dt);
              py=v0-(sv*x);
              putpixel(px,py, color+9);
              if (cont>10*N)    j=N;
              }
        else {
              RAT->X[j]=x;
              RAT->V[j]=p;
              j++;
              }
        t+=dt;
        cont++;
        pasos++;
        }while(j<N);
}


/**
 Metodo de Ordenamiento: Burbuja
 Ordena los individuos de la poblacion
 segun su fitness relativo
 **/
 void ordena(red R[POB+1])
 {
 int i,j, cambio=1;
 double sum;

 for (j=0;j<POB && cambio!=0;j++) {
      cambio=0;
      for (i=0;i<POB-1-j;i++)
      if (R[i].fitness>R[i+1].fitness) {
         cambio=1;
         igualaRedes(&R[POB],&R[i]);
```

```
                igualaRedes(&R[i],&R[i+1]);
                igualaRedes(&R[i+1],&R[POB]);
                }
            }
}


    /**
     Busqueda Binaria
     **/
    int busquedaBin(red R[POB+1], double p)
    {
    int ini=0, fin=POB, med;
    int i;

    do {
          med=(int)(fin+ini)/2;
          if(R[med].fitness>=p){
            if (R[med-1].fitness<p)
                return med;
            fin=med-1;
          }
          else if(R[med].fitness<p){
            if (R[med+1].fitness>p)
                return med+1;
            ini=med+1; }
          }while(ini<fin);
    return POB;
}


    /* GRAFICOS */

    /**
     Inicializa gráficos
     **/
    void graficos(void)
    {
    int gdriver = DETECT, gmode, errorcode;
    initgraph(&gdriver, &gmode, "d:\\tc\\bgi\\");
    errorcode = graphresult();
    if (errorcode != grOk) {
            printf("Error:%s\n", grapherrormsg
            (errorcode));
            getch();
            exit(1);
    }

    /**
     Dibuja ejes
     **/
    void ejes()
    {
```

```
            int h0=30, v0=250, ny=40,i;     //z, tt=1000;
            double sv=3.0,d,s;      //sh=.5,
            d=2*3.14159;
            s=sv*d;
            setcolor(WHITE);

            for (i=1; i<ny;i++) {
                  line (h0-3, v0-s*i, h0, v0-s*i);
                  line (h0-3, v0+s*i, h0, v0+s*i);
                  }
            for (i=5; i<ny; i+=5) {
                  line (h0-6, v0-s*i, h0, v0-s*i);
                  line (h0-6, v0+s*i, h0, v0+s*i);
                  }
            for (i=0; i<500; i+=20)
                  line(h0+i,v0,h0+1+i,v0);
      }


  /***
  Reporta el progreso de la simulación. La información
desplegada en el archivo de salida, es separada por co-
mas.
   ***/
  void reporte(red R[POB+1], FILE *fp, int g)
  {
      int i;                  /* Mejor fitness de la po-
blación */
      double mejor_val;      /* Fitness promedio población
*/
      double avg;       /*  Desviación estándar del fit-
ness*/      double stddev;          /*  Suma de cuadrados
*/
      double sum_square;       /* Cuadrado de la suma */
      double square_sum;       /* Suma total del fitness */
      double sum;

      sum=0.0;
      sum_square=0.0;

      for(i=0;i<POB;i++){
            sum+=R[i].fitness;
            sum_square+=R[i].fitness*R[i].fitness;
      }

      avg=sum/(double)POB;
      square_sum=avg*avg*(double)POB;
      stddev=sqrt((sum_square-square_sum)/(POB-1));
      mejor_val=R[POB].fitness;
      fprintf(fp,  "\n%5d,          %6.3f,  %6.3f,  %6.3f
\n\n",g,
                              mejor_val,avg,stddev);
  }
```

```
/***** MODULO PRINCIPAL *****/

      void main()
      {
      red R[POB+1];
      ratchet ratC,ratE;
      double inf=0, sup=1, mf;
      int i,g=0;
      FILE *fit,*fp;

      graficos();
      ejes();
      randomize();
      inicializaRatchet(&ratC);
      datosRatchet(&ratC);
      ejecutaSistema(&ratC,0,0);
      inicializaRatchet(&ratE);
      inicializaRed(&R[POB]);
      fp=fopen("res.txt","w");
      fit=fopen("fit.txt","w");
        fprintf(fp, "\n Mejor Fitness   Fitness Promedio
      Desviacion Etandard\n\n");
      for(i=0; i<POB; i++) {
            inicializaRed(&R[i]);
            inicializaPesos(&R[i],inf,sup);
            fijaEntrada(&R[i],ratC.V);
            propagaRed(&R[i]);
            evaluaIndividuo(&R[i],ratC.X,ratC.V,&ratE);
            }
      mejorIndividuo(R);
      do {
            seleccionCruza(R);
            mutaIndividuo(R,inf,sup);
            for(i=0; i<POB; i++) {
                  propagaRed(&R[i]);
                  evaluaIndi-
viduo(&R[i],ratC.X,ratC.V,&ratE);
                  }
            elitismoGeneracion(R);
            gotoxy(40,1);                    printf("a:
%f",R[POB].result[0]);
            gotoxy(40,2);                    printf("w:
%f",R[POB].result[1]);
            reporte(R,fp,g);
            g++;
            gotoxy(10,5);
            printf("%d",g);
            gotoxy(15,5);
            printf("%f",R[POB].fitness);
            fprintf(fit,"\n%f",R[POB].fitness);
```

```
            }       while       (g       <       MAXGEN       &&
R[POB].fitness<=.999999);
       ratC.m=R[POB].result[0];
       ratC.w=R[POB].result[1];
       ejecutaSistema(&ratC,1,1);
       getch();
       fclose(fp);
       fclose(fit);
       closegraph();
       }

  /*** TERMINA PROGRAMA *****/
```

Bibliografía

Algoritmos genéticos

1. [Brarnlette, 91] Brarnlette, M.F. "Initialization, mutation and selection methods in genetic algorithms for function optimization", *Proc Int. Conf. Genetic Algorithms*, **4**, 1991

2. [Goldberg 89] Goldberg, D. E. Genetic algorithms in search, optimization and machine learning. *Addison-Wesley Publishing Company*. NY 1989

3. [Holland 75] Holland, J.H, "Adaptation in natural and artificial systems", *University of Michigan Press*, Ann Arbor, MI. 1975

4. [Holland 81] Holland, J.H. "Genetic algorithms and adaptation". *Technical Reoprt No. 34*, Ann Arbor: University of Michigan, 1981

5. [Michell 98] Michell, M. An introduction to genetic algoritms: complex adaptive systems. *MIT Press*. EEUU 1998

6. [Poli 02] Poli, R., Langdon, W. Foundations of genetic programming, *Springer Verlag*, NY 2002

Estimación de parámetros

7. [Bard 74] Bard, Y. Nonlinear Parameter Estimation. *Academic Press*, NY, 1974

8. [Beck 77] Beck, J.V. , Arnold, K.J. Parameter estimation in engineering and science. *Wiley series in probability and mathematical statistics*. J. Wiley, NY. 1977

9. [Gallant 87] Gallant, A.R., Nonliner statistical models. *Wiley*, NY, 1987

10. [Jorgesen 94] Jorgensen, S.E., Fundamentals of Ecological Modelling 2da ed. *Elsevier*, Amsterdam 1994

11. [Jorgensen 01] Jorgensen, S.E., "Parameter estimation and calibration by use of exergy", *Ecological Modeling*, **146**, 2001

12. [Park 98] Park,T., Froment, G. "A hybrid genetic algorithm for the estimation of parameters un detailed kinetic models", *Computers chem. Engng.* **22**, 1998

13. [Whigham 01] Whigham, P.A., Rechnagel, F. "Predicting chlorophyll-a in fresh water lakes by hyridising process/based models and genetic algorithms. *Ecological Modeling*, **146**, 2001

Motores moleculares (*Ratchets*)

14. [Astumian 02] Astumian, R.D., Hänggi, P., "Brownian Motors", *Physics Today*, Nov. 2002.

15. [Cortés 00] Cortés. E., "Ratchet motion induced by a correlated stochastic force". *Physica A* **275**, 2000

16. [Jung 96] Jung, P., Kissner, J.G., Hänggi, P., "Regular and chaotic transport in asymetric periodic potentials: intertia ratchets", *Phys Rev Lett*, **76**, 1996

17. [Hänggi 96] Hänggi, P., Bartussek, P., Nonlinear Physics of Complex Systems, *Lecture Notes in Physics*, **476**, Berlin 1996.

18. [Magnasco 93] Magnasco M.M., Forced Termal Ratchets, *Physical Review Letters*, **71-10**, 1993

19. [Mateos 02] Mateos, J.L., "Current reversals in deterministic ratchets: points and dimers", *Physica D*, **168**, 2002

20. [Mateos 00] Mateos, J.L., "Chaotic Transport and Current Reversal in Deterministic Ratchets", *Physical Review Letters*, **84-2**, 2000

21. [Reimann 02] Reimann, P., "Brownian motors: noisy transport far from equilibrium", *Phys. Reports*, **361**, 2002.

22. [Sarmiento 99] Sarmiento A., Hernán Larralde, "Deterministic transport in ratchets", *Physical Review E*, **59-5**, 1999

Redes Neuronales

23. [Amit 89] Amit, D., Modeling Brain Function. Springer-Verlag, Berlin 1989

24. [Freeman 92] Freeman, J. A., Skapura, D. M. Neural Networks, Algorithms, Applications and Programming Techniques. *Addison-Wesley Publishing Company*. NY 1992

25. [Fausett 94] Fausett L., Fundamentals of Neural Networks: Architectures, Algorithms and Applications, *Prentice Hall*, NJ 1994

26. [Grossber 82] Grossber, S., The Adaptative Brain, *Reidel Press*, NY 1982

27. [Hertz 91] Hertz, J., Krogh, A, Palmer, R.,Introduction to the theory on neural computation. *Addison Wesley*, NY, 1991

28. [Kohonen 84] Kohonen, T., Self-Organization and Associative Memory, 2da. ed., *Springer-Verlag*, Berlin, 1984

29. [Rojas 96] Rojas, R. Neural Networks. A Systematic Introduction. *Springer Verlag*. Berlin 1996

30. [Rumelhart 86] Rumelhart, D.E., McClelland, J.L., Parallel Distributed Processing, *MIT Press*, Cambridge, 1986

31. [Tenorio 90] Tenorio, M.F., Lee, W.T. "Self organizing network for optimum supervised learning". *IEEE Transactions on Neural Networks*, **1**, 1990

Sistemas dinámicos caóticos

32. [Devaney 92] Devaney, R.L. Chaotic dynamical systems, *Addison Wesley*, NY, 1992

33. [Greborgi 87] Greborgi, C., Ott, E., Yorke, J.A., "Chaos, strange attractors, and fractal basin boundaries in nonlinear dynamics", *Science*, **238**, 1987

34. [Ott 93] Ott, E. Chaos in Dinámical Sistems, *Cambridge University Press*, NY, 1993

35. [Peitgen 92] Peitgen, H.O., Jürgens, H., Saupe, D. Chaos and Fractals: New Frontiers of Science, *Springer-Verlag*, NY, 1992

36. [Rasband 91] Rasband, S.N., Chaotic dynamics of nonlinear systems, *John Whikey & Sons*, NY, 1991.

37. [Ruelle 89] Ruelle, D. Chaotic Evolution and Strange Attractors, *Cambridge University Press*, Cambridge, 1989