matemática: Una publicación de la FCNM – ESPOL
2013, Vol. 11, No. 1

SISTEMAS NO LINEALES Y LA METAHEURÍSTICA SCATTER SEARCH

Martín Carlos[1]

Resumen. Este artículo presenta un algoritmo metaheurístico que permite encontrar soluciones reales a sistemas de ecuaciones no lineales, en contraste a los métodos numéricos iterativos convencionales. Se demuestra que resolver un sistema no lineal de ecuaciones es equivalente a resolver un problema de optimización, particularmente un problema de minimización. Se utiliza la metaheurística "scatter search" para el problema de optimización. Finalmente, se incluyen algunos resultados sobre la ejecución del algoritmo en sistemas no lineales.

Palabras Clave: Sistema de ecuaciones no lineales, scatter search, metaheurística, problema de optimización.

Abstract. This article presents a metaheuristic algorithm that allows finding real solutions to systems of nonlinear equations, in contrast to the traditional iterative numerical methods. We prove that solving a nonlinear system of equations is equivalent to solving an optimization problem, particularly a minimization problem. We use the metaheuristic "scatter search" for the optimization problem. Finally, include some results of execution of the algorithm used in nonlinear systems as examples.

Key Words: System of nonlinear equations, scatter search, metaheuristic, optimization problem.

Recibido: Febrero 2013
Aceptado: Marzo 2013

1. INTRODUCCIÓN

Existen muchos métodos numéricos iterativos para resolver sistemas de ecuaciones no lineales. Los algoritmos que implementan estos métodos normalmente trabajan con matrices, y necesitan además las primeras y segundas derivadas parciales, o sus aproximaciones, de algunas funciones de varias variables. Es necesario también efectuar operaciones matriciales como sumas, productos, incluso en algunos métodos calcular la inversa de matrices, todo esto por cada iteración, lo que hace que el costo computacional sea alto. Son muy conocidos los métodos de Newton y sus variaciones como Newton Amortiguado, Quasi-Newton o Newton Mejorado. Existen también un sinnúmero de métodos de punto fijo. Otra característica de estos métodos iterativos clásicos es que son localmente convergentes, es decir, el punto inicial, con el cual se comienza a iterar, debe estar "algo cerca" de la solución para que el algoritmo converja en un número aceptable de iteraciones. Lo que se propone en este documento es un algoritmo distinto. Lo primero que haremos es expresar el sistema de ecuaciones no lineales en términos de un problema de optimización y, se va a demostrar formalmente, que resolver el problema de optimización es equivalente a resolver el sistema no lineal. Una vez hecho esto, nos enfocaremos en resolver el problema de optimización, que consiste en minimizar una función objetivo que se construye usando cada una de las ecuaciones que componen el sistema no lineal.

Se usará la metaheurística de "scatter search" para resolver el problema de optimización. Se dará una explicación de cada uno de los pasos del algoritmo. Un objetivo secundario es mostrar la metodología y la filosofía de trabajo del algoritmo scatter search cuando se resuelven problemas de optimización. Se podrá apreciar que la mezcla de calidad y diversidad que utiliza el algoritmo es fascinante. Luego, la solución real encontrada para el problema de optimización es también la solución real del sistema no lineal de ecuaciones. Finalmente, se procede a compartir los resultados obtenidos al ejecutar el algoritmo que proponemos sobre algunos sistemas de ecuaciones no lineales que usaremos como ejemplos.

2. SISTEMAS DE ECUACIONES NO LINEALES

Un sistema no lineal de ecuaciones con incógnitas es de la forma:

$$\begin{cases} f_1(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_2(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_3(x_1, x_2, x_3, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, x_3, \ldots, x_n) = 0 \end{cases}$$

O en presentación vectorial $F(X) = \vec{0}$, donde: $X = (x_1, x_2, x_3, \ldots, x_n)$,

[1] *Martín Carlos, M.Sc., Profesor de la Escuela Superior Politécnica del Litoral (ESPOL).*
(e_mail: cmmartin@espol.edu.ec).

$$F(X)=\begin{pmatrix} f_1(x_1,x_2,x_3,\ldots,x_n) \\ f_2(x_1,x_2,x_3,\ldots,x_n) \\ f_3(x_1,x_2,x_3,\ldots,x_n) \\ \vdots \\ f_n(x_1,x_2,x_3,\ldots,x_n) \end{pmatrix}$$ y $\vec{0}$ es el vector nulo de R^n

Se va a suponer que las n funciones $f_1, f_2, f_3, \ldots, f_n$ son, en general, funciones no lineales de n variables reales independientes y que tienen un dominio en común el cual se va a denotar como $\Omega \subseteq R^n$. Es decir, para cada $i = 1,2,3,\ldots,n$:

$$f_i: \quad \Omega \quad \mapsto \quad R$$
$$X=(x_1,x_2,x_3,\ldots,x_n) \rightarrow f_i(X)$$

Se procede ahora a definir la función objetivo g la cual tiene también obviamente su dominio en el conjunto Ω:

$$g(X)=\sum_{i=1}^{n}|f_i(X)|=$$
$$|f_1(X)|+|f_2(X)|+|f_3(X)|+\ldots+|f_n(X)|$$

3. EL PROBLEMA DE OPTIMIZACIÓN

A continuación se va a enunciar y a demostrar matemáticamente una proposición que justifica el plantear un sistema de ecuaciones no lineales en términos de un problema de minimización.

Proposición.- El vector $X^*=(x_1^*,x_2^*,x_3^*,\ldots,x_n^*)\in\Omega$ es una solución de $F(X)=\vec{0}$ (es decir: $F(X^*)=\vec{0}$), si y sólo si, la función g (definida anteriormente) toma como valor mínimo en X^* el valor 0

Demostración

Primera Parte: Supongamos que $F(X^*)=\vec{0}$. Entonces para cada $i=1,2,3,\ldots,n$ ocurre que $f_i(X^*)=0$. Luego:

$$g(X^*)=\sum_{i=1}^{n}|f_i(X^*)|=0$$

Además para toda $i=1,2,3,\ldots,n$ y para toda $X\in\Omega$ se tiene que $|f_i(X)|\geq 0$, por lo que $g(X)\geq 0$ para toda $X\in\Omega$.

Entonces la función g toma el valor mínimo cero en $X^*\in\Omega$

Segunda Parte: Ahora supongamos que la función g toma el valor mínimo cero en $X^*\in\Omega$. Se debe entonces demostrar que $F(X^*)=\vec{0}$.

Por hipótesis $g(X^*)=0$. Entonces

$$g(X^*)=\sum_{i=1}^{n}|f_i(X^*)|=0$$

Es decir

$$|f_1(X^*)|+|f_2(X^*)|+|f_3(X^*)|+\ldots+|f_n(X^*)|=0$$

Se debe probar que para toda $i=1,2,3,\ldots,n$ ocurre que $f_i(X^*)=0$. Hagamos la prueba por contradicción. Supongamos que lo anterior es falso, es decir, que existe una $1\leq i\leq n$ tal que $f_i(X^*)\neq 0$. Si esto es así, sucede que

$$g(X^*)=\sum_{i=1}^{n}|f_i(X^*)|$$
$$=|f_1(X^*)|+|f_2(X^*)|+|f_3(X^*)|$$
$$+\ldots+|f_n(X^*)|>0$$

lo que obviamente contradice la hipótesis que dice que $g(X^*)=0$, lo cual no puede ser. Por lo tanto, para toda $1\leq i\leq n$ se da que $f_i(X^*)=0$ y $F(X^*)=\vec{0}$. Es decir, X^* es una solución de $F(X)=\vec{0}$.

De aquí en más, para resolver un sistema no lineal en particular, lo primero que haremos es construir la función g utilizando las funciones $f_1, f_2, f_3, \ldots, f_n$ que componen el sistema. Luego, el conjunto Ω, para la ejecución del algoritmo, lo vamos a suponer de la forma:

$$\Omega=\left\{ \begin{array}{c} (x_1,x_2,x_3,\ldots,x_n)\in R^n / \alpha_i\leq x_i\leq\beta_i \\ para\ \ cada\ \ i=1,2,3,\ldots,n \end{array} \right\}$$

Es decir, para resolver el sistema de ecuaciones no lineales, se debe minimizar la función no lineal g y se deben fijar cotas inferiores y superiores para cada una de las variables del sistema. Se espera que exista una solución para el sistema no lineal en el conjunto Ω, para que la solución encontrada en el problema de optimización la podamos tomar también como solución del

sistema no lineal. Debido a que el sistema no lineal es el modelo matemático de alguna aplicación o problema del mundo real, se puede construir un conjunto Ω donde se sospecha o se considera se encuentra la solución del sistema no lineal. La experiencia del modelador, o de las personas involucradas en el problema que es representado por el sistema no lineal, es vital y sumamente importante para definir dicho conjunto Ω.

4. EL ALGORITMO METAHEURÍSTICO

El problema de optimización que se pretende resolver es un problema de minimización en el cual la función objetivo $g(x_1, x_2, x_3, \ldots, x_n)$ es una función no lineal y, cada una de las variables independientes de g, está acotada tanto inferior como superiormente. Es decir, para cada $i = 1,2,3,\ldots,n$ ocurre que $x_i \in [\alpha_i, \beta_i]$. En otras palabras:

$$\Omega = \prod_{i=1}^{n} [\alpha_i, \beta_i] = [\alpha_1, \beta_1] \text{ X } [\alpha_2, \beta_2] \text{ X } [\alpha_3, \beta_3] \text{ X } \ldots \text{ X } [\alpha_n, \beta_n] \subseteq R^n$$

Para ejecutar el algoritmo necesitamos ingresar entonces la función g y, los n intervalos para cada una de las n variables donde se cree que se encuentra la solución del sistema no lineal.

Algo importante de mencionar es que al usar cualquier algoritmo metaheurístico estamos concientes que no existe garantía de encontrar el óptimo del problema, pero podemos encontrar una "buena solución" en un tiempo prudencial y con un costo computacional aceptable, en comparación con cualquier método exacto que se pueda usar para el problema de optimización. Ahora, ¿por qué el algoritmo "scatter search"? Porque ha sido muy ampliamente usado y con mucho éxito para la optimización de funciones lineales y no lineales de varias variables sobre un espacio de soluciones que puede ser continuo y también discreto. A continuación se explica cada etapa del algoritmo.

Generación del Conjunto P

Lo primero que debe hacerse al resolver un problema de optimización con "scatter search" es construir un conjunto al que llamaremos P. Una entrada del algoritmo entonces es el tamaño del conjunto P, definida con la variable $PSize$. Los elementos del conjunto P serán vectores de R^n que pertenecen al conjunto Ω. Es decir, cada elemento del conjunto P es una solución factible del problema de minimización. Estos elementos se van a generar aleatoriamente pero de forma tal que podamos cubrir o abarcar todo el conjunto Ω, para así tener diversidad en esta primera generación de soluciones factibles. Sin embargo, el tema de la calidad es también importante en esta primera etapa del algoritmo. A continuación se explica cómo se van a generar los vectores.

Cada variable independiente x_i de la función g "vive" en un intervalo $[\alpha_i, \beta_i]$. Se va a dividir o a particionar cada intervalo $[\alpha_i, \beta_i]$ en m subintervalos. El número de subintervalos de igual tamaño en el que se va a dividir el intervalo $[\alpha_i, \beta_i]$ se va a representar entonces con la variable m, el cual es también un parámetro de entrada para el algoritmo. De esta manera, la longitud de cada subintervalo (todos los subintervalos para la variable x_i tienen la misma longitud) es: $\Delta x_i = (\beta_i - \alpha_i)/m$

Ahora se van a generar $PSize$ elementos con una mezcla de diversidad y calidad. Primero hacemos uso de la diversidad, ¿cómo trabaja? Así: Para cada variable x_i se selecciona aleatoriamente un subintervalo (se genera un número entero aleatorio entre 1 y m), luego se genera un número aleatorio de punto flotante dentro del subintervalo seleccionado anteriormente, lo que va a representar el valor de la *i*-ésima componente del vector. Esto se debe hacer entonces por cada una de las n componentes del vector de R^n. Finalmente, se aplica un método de mejora al vector que se acaba de construir, en otras palabras, a través de una "local search" se busca si existe "cerca" de la solución generada otro vector que mejora el comportamiento de la función objetivo (es decir que minimiza la función g). Posteriormente se explicará la forma de trabajar de la búsqueda local. Pero, para asegurar la diversificación de los elementos en el conjunto P, la probabilidad de elegir un subintervalo, para una variable cualquiera x_i, es inversamente proporcional a la cantidad de números de punto flotante generados en dicho subintervalo de esa variable x_i. De esta manera se trata de "balancear" las soluciones a lo largo de todo el conjunto Ω. Como podemos apreciar, se tiene ahora un conjunto P con una mezcla muy buena de calidad y diversidad.

Búsqueda Local

La "local search" o búsqueda local implementada recibe como entrada la solución a mejorar y un parámetro adicional que le dice al método el número de intentos que debe hacer para buscar un "mejor" vector en una vecindad del vector inicial, este número de intentos está en función del número de veces que se va a evaluar la función

g. ¿Cómo trabaja? A la *i*-ésima componente del vector inicial se le genera un número aleatorio que se le suma o se le resta a esa *i*-ésima componente (lo cual también es aleatorio) de manera tal que se obtiene un nuevo valor para esa *i*-ésima componente y, de esta forma, al efectuar esto n veces, se obtiene un nuevo vector. Aquí se debe tener todo el cuidado de que el valor de la *i*-ésima componente del nuevo vector respete los límites definidos del subintervalo al cual pertenece el valor de la *i*-ésima componente del vector original (esto es lo que hace que la búsqueda sea local, el nuevo valor de cada variable no se debe salir del subintervalo en el que se encuentra el valor original de esa variable). Cada vez que se construye un vector en la vecindad del vector original se ha realizado "un intento". Como resultado del intento se tiene un nuevo vector que es descartado si no mejora la función g o tomado como el nuevo vector original si la función g mejora con este nuevo vector. Después de una cantidad de intentos (lo que es un argumento de este procedimiento) se detiene la búsqueda. Si se ha llegado al número de intentos tope de búsqueda definido a través del argumento "número de evaluaciones" y no se ha encontrado una mejor solución, el método devuelve la primera solución que se pasó como argumento, es decir, se indica así que "nos quedamos igualitos", no se encontró nada mejor. Cabe indicar que otro método que se implementó para la búsqueda local es el conocido algoritmo heurístico "Nelder-Mead" propuesto por Jhon Nelder y Roger Mead en 1965 para la optimización continua de una función objetivo sin restricciones. Este algoritmo sólo utiliza la regla de correspondencia de la función a optimizar y no utiliza las derivadas parciales de ningún orden para dicha función, motivo por el cual fue escogido también. Además se trata de un algoritmo que ha tenido mucho éxito en diversos problemas presentando buenos resultados, pero hay que tener presente que se trata de un algoritmo local y no global. De esta manera, el algoritmo scatter search que se ha construido tiene la capacidad de ejecutar con cualquiera de los dos algoritmos de búsqueda local antes indicados.

Construcción del Conjunto R

En el método de "scatter search" existe un segundo conjunto importante, el conjunto R, el cual es construido a partir del conjunto P. La cardinalidad de este conjunto R es también un parámetro del algoritmo. Para construir el conjunto R nos preocupamos que la variable $RSize$, que define el tamaño de R, sea par, ya que la mitad de los elementos de R serán escogidos por calidad mientras que la otra mitad por diversificación, como ya se dijo, haciendo uso del conjunto P. Aquí es importante indicar que se pudo haber parametrizado esto, es decir, que b_1 elementos sean escogidos por calidad y b_2 por diversidad teniendo presente que $b_1 + b_2 = RSize$. Retomando, se toman entonces los $RSize/2$ "primeros en calidad" elementos del conjunto P (es decir, los mejores elementos de dicho conjunto) y se incorporan al conjunto R. ¿Cómo escoger los elementos restantes?

De los elementos que quedan de P se deben tomar $RSize/2$ elementos más para completar el conjunto R, ¿cómo se lo hace para garantizar diversidad? Se calcula la distancia de cada elemento de P (ya no se consideran aquellos que se llevan a R) al conjunto R con la fórmula: Sea $v \in P$, luego $d(v,R) = \underset{w \in R}{MIN} \| v - w \|$ y el elemento que se almacena en R es aquel elemento de P que dista más de R, es decir, el más lejano al conjunto R. La norma utilizada para calcular la distancia mencionada es la norma euclidiana.

En este momento se tiene ya construido totalmente el conjunto R con una mezcla muy buena y adecuada de calidad y diversidad.

Actualización y Mantenimiento del Conjunto R

Durante la búsqueda del óptimo el conjunto R debe actualizarse muchas veces y eso es lo que se va a explicar. Cabe indicar aquí que el número de actualizaciones del conjunto R es también un parámetro del algoritmo. Una vez construido el conjunto R se procede a crear subconjuntos de R. En esta implementación el tamaño de cada subconjunto es de dos elementos de R (pero en otras implementaciones con scatter search la cardinalidad de estos subconjuntos puede ser de más elementos), es decir, usando R se generan todas las parejas posibles de elementos sin repetir, porque se les va a aplicar un método de combinación. ¿Cómo trabaja el método de combinación? Eso se explica unas líneas más adelante. Al aplicar un método de combinación a cada subconjunto, se generan cuatro elementos y, a cada uno de estos nuevos elementos (llamados vectores prueba) se les aplica un método de mejora, es decir, se emplea la operación de búsqueda local explicada anteriormente para estudiar la vecindad de estas "soluciones prueba" en busca de unas mejores soluciones. Entonces, por cada subconjunto se tienen cuatro "soluciones mejoradas" que se añaden a un conjunto denominado $Pool$. Es decir que finalmente en $Pool$ se tiene el resultado de aplicar métodos de "combinación" y "mejora" a cada uno de los subconjuntos construidos a partir del gran

conjunto R. Una vez que el conjunto $Pool$ está lleno se construye un conjunto de "vida corta" $Temp$ de forma tal que ahí se almacenen todos los elementos de R y de $Pool$, es decir $Temp = R \cup Pool$. Este conjunto temporal, tiene una cardinalidad igual a la cardinalidad de R más la cardinalidad de $Pool$. Los $RSize$ mejores elementos son los que pasan finalmente al conjunto R. ¿Cuántas veces ocurre esto? Hay un parámetro llamado "Número de Iteraciones" que se debe ingresar antes de correr el algoritmo de scatter search, pues este parámetro es el que sirve para configurar el número de veces que el conjunto R se va a actualizar. La opción de reconstruir el conjunto R, aclaramos, no se ha incorporado en la implementación. Es decir, si el conjunto R permanece sin cambios después de una actualización (debido a que el "combinar y mejorar" no produce nada nuevo) se recomienda una reconstrucción donde participa nuevamente el conjunto P y el conjunto R se actualiza nuevamente por calidad y diversidad. No se implementó esta etapa debido a que no se lo consideró necesario ya que en las pruebas que se realizaron el algoritmo siempre encontraba soluciones muy cercanas al óptimo, es decir, el software encontraba correctamente valores muy próximos al óptimo del problema de minimización en todos los ejemplos de sistemas no lineales en los que se lo puso a prueba. Cabe indicar que se probó el algoritmo en sistemas no lineales de hasta cinco ecuaciones y cinco incógnitas, y con conjuntos Ω "pequeños" donde se sospechaba se encontraba la o las soluciones. Pero sería importante considerar la reconstrucción del conjunto R si se tienen sistemas no lineales de mayor tamaño.

Método de Combinación

El método de combinación implementado es muy sencillo y opera así: se generan cuatro números aleatorios de punto flotante en el intervalo $(0,1)$ con la idea de aplicar combinaciones lineales "convexas". Es decir, el método de combinación lo único que recibe como parámetro es el subconjunto de R que tiene dos soluciones $SolA$ y $SolB$, y devuelve como salida cuatro combinaciones lineales convexas distintas. Es importante decir que esta salida no pasa directamente a R ya que antes se aplica el método de mejora a cada una de las cuatro soluciones obtenidas por combinación, y ellas sí pasan ya al conjunto R.

Método Principal del Algoritmo

A continuación se va a mostrar el método principal llamado "EjecutarSS" que gobierna el comportamiento y la operación general del algoritmo de scatter search que se implementó. Todas las etapas o fases anteriores coordinan y se enlazan aquí. A continuación el método:

```
    public double[] EjecutarSS(out double
MejorValor)
    {
      ArrayList Pool = new ArrayList();
      ArrayList EvalPool = new ArrayList();
      Stopwatch Tiempo;
      double[,] MejoresValores =
      new double[this.NumIteraciones, 2];
      Tiempo = Stopwatch.StartNew();
      this.GenerarP();
      this.ConstruirR(this.P, this.EvalP, this.R,
this.EvalR);

      ArrayList SubSets = new ArrayList();
      int Iteraciones = 0;
      while (true)
      {
        this.GenerarSubSets(SubSets, R);
        while (SubSets.Count != 0)
        {
          double[][] SubSet =
(double[][])SubSets[0];
          double[][] TrialSolutions =
this.Combinar(SubSet);
          double Valor1 =
this.LS(TrialSolutions[0], 100);
          Pool.Add(TrialSolutions[0]);
          EvalPool.Add(ValorUno);
          double Valor2 =
this.LS(TrialSolutions[1], 100);
          Pool.Add(TrialSolutions[1]);
          EvalPool.Add(ValorDos);
          double Valor3 =
this.LS(TrialSolutions[2], 100);
          Pool.Add(TrialSolutions[2]);
          EvalPool.Add(ValorTres);
          double Valor4 =
this.LS(TrialSolutions[3], 100);
          Pool.Add(TrialSolutions[3]);
          EvalPool.Add(ValorCuatro);
          SubSets.RemoveAt(0);
        }
        this.ActualizarR(R, EvalR, Pool,
EvalPool);
        MejoresValores[Iteraciones, 0] =
EvalR[0];
        MejoresValores[Iteraciones, 1] =
        Tiempo.ElapsedMilliseconds;
        Iteraciones++;
        if (Iteraciones == this.NumIteraciones)
break;
      }
      MejorValor = EvalR[0];
      return R[0];
    }
```

5. SOLUCIÓN DE ALGUNOS SISTEMAS NO LINEALES

A continuación se presentan algunas corridas del algoritmo scatter search. Para empezar, y para poder dar una interpretación geométrica de los resultados, vamos a mostrar lo que ocurre con la ecuación $f(x)=x^3-e^x\ sen(x)$. Haciendo uso del teorema de Bolzano se puede probar que f tiene un CERO en el intervalo $\Omega=[1,2]$. Por la proposición enunciada y demostrada anteriormente se puede afirmar que x^* satisface $f(x)=x^3-e^x\ sen(x)=0$, si y sólo si, x^* minimiza la función $g(x)=\left| x^3-e^x\ sen(x)\right|$ en el intervalo $[1,2]$ con el valor cero. La figura # 1 muestra la gráfica de la función f en el intervalo $[1,2]$, mientras que la figura # 2 nos muestra la gráfica de la función a minimizar g en el mismo intervalo $[1,2]$.

FIGURA 1

Sistemas no lineales y la metaheurística scatter search

Gráfico de $f(x)=x^3-e^x\ sen(x)$

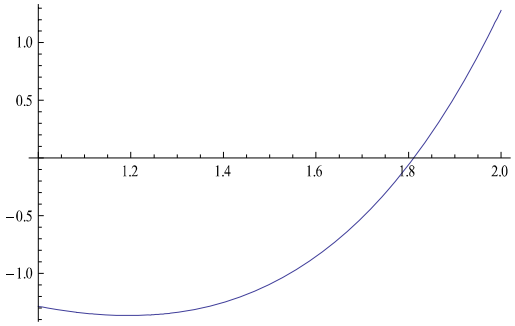
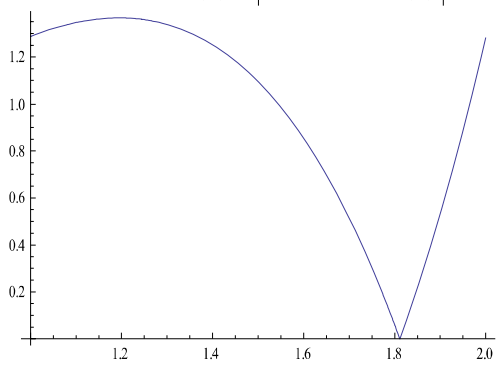


FIGURA 2

Sistemas no lineales y la metaheurística scatter search

Gráfico de $g(x)=\left| x^3-e^x\ sen(x)\right|$



Se puede observar que mayor a 1.8 es el valor de x^*. En una de las corridas del algoritmo metaheurístico de "scatter search" encontramos que $x^*=1.8112693646$ y que $g(x^*)=0.0000655681$. Para esta ejecución se dividió el intervalo $\Omega=[1,2]$ sólo en 10 subintervalos y se solicitaron 5 actualizaciones del conjunto R.

Un segundo ejemplo es el sistema:

$$\begin{cases} x\ sen(y)-1=0 \\ x^2+cos(2y)=0 \end{cases}$$

Se puede apreciar que el par ordenado $(1,\pi/2)$ es una solución del sistema en la región $\Omega=\{(x,y)\in R^2\ /\ 0\le x\le 2\ ,\ 0\le y\le 2\}$.

La gráfica de la función $g(x,y)=\left| x\ sen(y)-1\right|+\left| x^2+cos(2y)\right|$ sobre la región Ω del plano xy se muestra en la figura 3 y en la figura 4.

FIGURA 3

Sistemas no lineales y la metaheurística scatter search

Gráfico de

$g(x,y)=\left| x\ sen(y)-1\right|+\left| x^2+cos(2y)\right|$

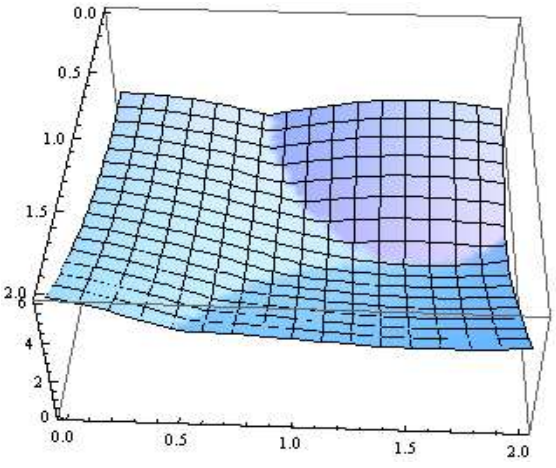
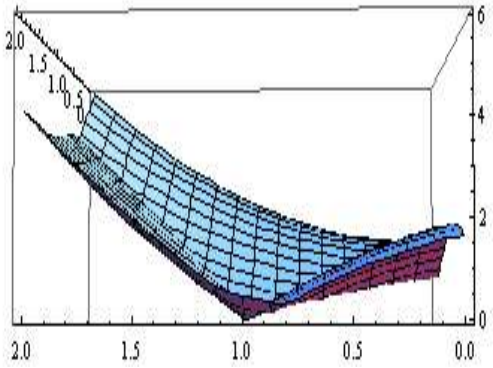


FIGURA 4
Sistemas no lineales y la metaheurística scatter search
Otra vista de
$g(x,y)=|\ x\,sen\,(y)-1\,|+|\ x^2+cos\,(2y)\,|$

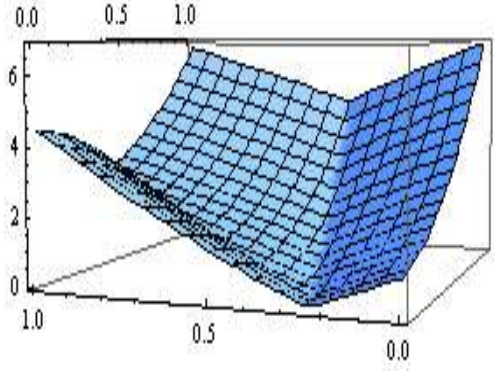


En una de las ejecuciones del algoritmo "scatter search", para este segundo ejemplo, se obtuvo que $(x^*,y^*)=(1.0,1.5707963290621922)$ y que $g(x^*,y^*)$ es cero. Para esta corrida se solicitaron 5 actualizaciones del conjunto R y, para cada variable, se ingresó el intervalo $[-10,10]$, el cual se particionó en 10 subintervalos.
Otro ejemplo con interpretación gráfica es el siguiente:

$$\begin{cases} 5x^2-y^2=0 \\ 4y-sen\,(x)-cos\,(y)=0 \end{cases}$$

Este sistema no lineal tiene solución en $\Omega=[0,1]\,X\,[0,1]$. La gráfica de la función $g(x,y)=|\ 5x^2-y^2\ |+|\ 4y-sen\,(x)-cos\,(y)\,|$ se muestra en la figura 5.

FIGURA 5
Sistemas no lineales y la metaheurística scatter search
Gráfico de



Con los mismos parámetros de entrada de la ejecución del segundo ejemplo se obtuvo que $(x^*,y^*)=(0.12124191148054805,\ 0.27110515579243255)$ y que $g(x^*,y^*)$ es del orden de 10^{-14}. Es importante indicar que se pueden solicitar más actualizaciones del conjunto R, aumentar el valor del parámetro m, aumentar la cardinalidad de los conjuntos P y R, todo esto con el objetivo de mejorar la precisión. Sin embargo, los costos computacionales serán mayores.
En otro ejemplo, considere el sistema no lineal que se muestra a continuación:

$$\begin{cases} (x-2)^2+(y-1)^2+x\,y-3=0 \\ x\,e^{x+y}+y\,z-3=0 \\ sen\,(x+z)+x-y-z+1=0 \end{cases}$$

Se ejecutó el algoritmo "scatter search" y se obtuvo como resultado la solución (x^*,y^*,z^*) $=(3.03924738054514,\ -1.6104828215273,\ 6.01336437928736)$ y además que $g(x^*,y^*,z^*)=0.0000000345641$ en la región sólida $\Omega=[-10,10]\ X\ [-10,10]\ X\ [-10,10]$. Se solicitaron nuevamente 5 actualizaciones del conjunto R y además $m=10$.
Finalmente, como un último ejemplo, se tiene el sistema no lineal:

$$\begin{cases} x^2+2y^2+cos\,(z)-w^2=0 \\ 3x^2+y^2+sen^2\,(z)-w^2=0 \\ -2x^2-y^2-cos\,(z)+w^2=0 \\ -x^2-y^2-2cos^2\,(z)+w^2=0 \end{cases}$$

Este sistema no lineal presenta muchas soluciones. Dos de ellas son: $(1,-1,0,2)$ y $(-1,1,0,-2)$. Se ingresaron como parámetros $m=10$, un total de 5 actualizaciones para el conjunto R y el espacio de búsqueda $\Omega=[-10,10]X[-10,10]X[-10,10]X[-10,10]$. El valor de la función g en el punto encontrado (x^*,y^*,z^*,w^*) $=(0.99896239,\ -0.99896202,\ 0.03718569,\ 1.99827054)$ es igual a 0.000002593939, es decir, del orden de 10^{-6}.
Se realizaron pruebas con sistemas de hasta cinco ecuaciones con cinco incógnitas y con otros sistemas no lineales de menor tamaño, que no se muestran en este documento, con excelentes resultados.

6. TABLA COMPARATIVA

A continuación se muestra una tabla que compara los algoritmos metaheurísticos con los algoritmos iterativos convencionales para la solución de sistemas no lineales. Se presentan ventajas y desventajas de cada uno.

Algoritmos Metaheurísticos		**Algoritmos Iterativos Convencionales**	
Ventajas	**Desventajas**	**Ventajas**	**Desventajas**
Dominio de convergencia ampliado debido a las estrategias de "diversificación"	La implementación del algoritmo es más complicada, existen muchas fases o etapas involucradas	La implementación del algoritmo es más sencilla, usualmente mediante fórmulas recursivas	Normalmente un "pequeño" dominio de convergencia
Se los puede utilizar para determinar el "dato inicial" que necesitan los algoritmos iterativos convencionales o para futuras corridas del mismo algoritmo en un espacio de búsqueda más reducido	No existe garantía de encontrar el óptimo del problema, son algoritmos que utilizan en su trabajo la aleatoriedad. Además, pueden quedarse atrapados en óptimos locales	Si el "dato inicial" se encuentra dentro del intervalo de convergencia se garantiza que el método converge a la solución exacta	Pueden tener dependencia de otros métodos que proporcionen el "dato inicial" para poder comenzar con las iteraciones
Las funciones de varias variables que se utilizan no necesitan cumplir con condiciones de continuidad y diferenciabilidad de ningún orden	Espacio de búsqueda considerable, en general convergencia lenta. La velocidad de convergencia depende del tamaño del espacio de búsqueda	En general se tiene convergencia al menos cuadrática, es decir, son métodos rápidamente convergentes	Dependiendo del método que se utilice, las funciones de varias variables empleadas deben ser de clase C^1 o de clase C^2 o deben satisfacer o cumplir la condición de Lipschitz
No se necesitan cálculos matriciales tediosos ni calcular derivadas parciales de ningún orden	No se puede estimar el número de iteraciones necesarias para estar tan cerca de la solución exacta como se desee.	Se puede tener control sobre el error. Se puede estimar el número de iteraciones para que la distancia entre la solución exacta y el valor estimado no supere la tolerancia deseada	Por cada iteración muchos cálculos matriciales (en algunos casos el cálculo de matrices inversas y producto de matrices) y cálculos de derivadas parciales
En una misma ejecución se pueden encontrar muchas soluciones, si es que existen	La técnica principal en que basan su trabajo es una computación evolutiva. Existe muy poca teoría matemática detrás del método que estudie o analice convergencia y error	Muchos teoremas y proposiciones que estudian convergencia y error de los métodos	Si el sistema no lineal tiene muchas soluciones, sólo una se puede encontrar por cada corrida

7. CONCLUSIONES Y RECOMENDACIONES

Los algoritmos metaheurísticos usados para resolver sistemas de ecuaciones no lineales tienen la ventaja de que las funciones de varias variables que componen el sistema no necesitan ser de clase C^1, por el uso del Jacobiano, ni de clase C^2, por el empleo de la matriz Hessiana, ni ser diferenciables, ni satisfacer condiciones de Lipschitz como en los métodos de punto fijo. Los métodos numéricos iterativos convencionales exigen estas condiciones en las funciones del sistema no lineal $f_1, f_2, f_3, \ldots, f_n$. En los algoritmos metaheurísticos podrían las funciones

anteriores incluso no ser funciones continuas sobre su dominio $\Omega \subseteq R^n$ y sin embargo algoritmos como scatter search podrían ejecutar sin inconvenientes y resolver los sistemas no lineales. Sólo necesitamos que las funciones se encuentren definidas sobre el conjunto Ω. Además, estos algoritmos no necesitan efectuar cálculos matriciales que puedan complicar la implementación, consumir mucha memoria y tiempo excesivo de procesador. A pesar de que en general los algoritmos metaheurísticos no garantizan entregar la respuesta exacta, para sistemas no lineales de pequeño o mediano tamaño han probado tener mucho éxito. La recomendación principal para futuros trabajos es la de implementar al menos un par de métodos iterativos clásicos y de comparar los resultados de ejecución, en eficacia y eficiencia, con el algoritmo metaheurístico que se presenta en este artículo. Finalmente se podría intentar construir un "algoritmo híbrido", es decir, un algoritmo donde algunas partes provengan por ejemplo del método de Newton o de una de sus variantes y, por otro lado, también tenga componentes que utilicen "inteligentemente" la aleatoriedad como lo hacen los algoritmos metaheurísticos.

Otro trabajo a futuro que podría efectuarse sería el de preparar el algoritmo de forma tal que no sólo se puedan encontrar soluciones reales sino también soluciones complejas para los sistemas no lineales que las posean. Se puede apreciar también que "scatter search", el algoritmo metaheurístico utilizado, no hace un uso intensivo e indiscriminado de la aleatoriedad, es decir, no abusa de ella como sí lo hacen, por ejemplo, en general los algoritmos genéticos. La mezcla de diversidad y calidad del algoritmo "scatter search" muestra lo inteligente de su forma de trabajar. Podríamos también considerar el empleo de otros algoritmos metaheurísticos para resolver sistemas no lineales. Finalmente, cabe mencionar que existe otro método iterativo no tradicional que permite resolver sistemas no lineales de ecuaciones y que usa "homotopía y continuación numérica". En este método se expresa el sistema no lineal en términos de un sistema de ecuaciones diferenciales ordinarias de primer orden donde se podría, por ejemplo, usar algún método de Runge-Kutta. Se trata de un problema equivalente, de forma tal que la solución del sistema de ecuaciones diferenciales es también la solución del sistema no lineal de ecuaciones.

REFERENCIAS BIBLIOGRÁFICAS Y ELECTRÓNICAS

[1]. **GROSAN Y ABRAHAM** (2008). "*A New Approach for Solving Nonlinear Equations Systems*", IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, Vol. 38, No. 3

[2]. **LAGUNA Y MARTÍ** (2003). "*Scatter Search: Methodology and Implementations in C*", Kluwer Academic Publishers, Norwell Massachusetts

[3]. **MARTÍ Y LAGUNA** (2003). "*Scatter Search: Diseño Básico y Estrategias Avanzadas*", Universidad de Valencia, España