Revista de la Facultad de Ingeniería Industrial
Vol. (8) 2: pp. 61-68 (2005) UNMSM
ISSN: 1560-9146 (impreso) / ISSN: 1810-9993 (electrónico)

OPTIMIZACIÓN POR COMPUTACIÓN EVOLUCIONARIA

[1] Eduardo Raffo Lecca
[2] Edgar Ruiz Lizama

Recepción: Noviembre de 2005 / Aceptación: Diciembre de 2005

RESUMEN

El artículo presenta la implementación de algoritmos de evolución diferencial en el software MATLAB®. El precio que se paga por una optimización numérica eficiente, significa hacer uso de matemáticas complejas. La evolución diferencial, es una excepción porque es fácil de aplicar y robusto en la optimización numérica. La evolución diferencial es una herramienta de diseño de gran utilidad en las aplicaciones prácticas. El programa que se presenta en este artículo, es una implementación mejorada y eficiente al programa que presentan Price y Storm.

Palabras Clave: Evolución diferencial, programación evolucionaria, algoritmos genéticos, optimización.

OPTIMIZATION FOR EVOLUTIONARY CALCULATION
ABSTRACT

The article presents the implementation of differential evolution algorithm in MATLAB® software. The price paid for an efficient numerical optimization, means to use of complex mathematics. The differential evolution is an exception because is easy to apply and robust in the numerical optimization. The differential evolution is a design tool of great utility in practical applications. The program presented in this article is an improved and efficient implementation of the program that present Price and Storm.

Key words: Diferential evolution, evolutionary programing, algorithms genetics, optimization.

(1) Ingeniero Industrial. Profesor del Departamento de Ingeniería de Sistemas e Informática, UNMSM. E-mail: eraffol@unmsm.edu.pe
(2) Ingeniero Industrial. Profesor del Departamento de Ingeniería de Sistemas e Informática, UNMSM. E-mail: eruizl@unmsm.edu.pe

INTRODUCCIÓN

Desde comienzo de 1950 el interés en algoritmos que se basan en analogías de procesos naturales ha ido en aumento [6]. Desde esos tiempos, los investigadores están usando los conceptos basados en la teoría de la evolución de Darwin, para la solución de problemas de optimización.

Numerosos algoritmos basados en el principio de Darwin, han empezado a desarrollarse en las últimas tres décadas. Ellos están usando el término de "métodos de computación evolucionaria". El término de algoritmos evolucionarios, es usado en forma indistinta para describir diferentes métodos de computación evolucionaria. Para las tareas de optimización de funciones de variables reales, la evolución estratégica ha emergido como un gran contendor a diversos métodos de solución tradicionales.

Los métodos de la computación evolucionaria, trabajan con el principio de Darwin de la selección natural; principio que le sirvió a Herbert Simon para acuñar el término "supervivencia de la especie".

Entre las técnicas mas notables de este grupo, destacan los algoritmos genéticos o GA (Genetic Algorithms), estrategias evolutivas o ES(Evolution Strategies), programación evolucionaria o EP(Evolutionary Programming), sistemas clasificados y programación genética GP(Genetic Programming).

Un algoritmo genético, es un tipo de sistema basado en la evolución, y corresponde a una clase de algoritmo de programación estocástica adaptivo, que envuelve búsqueda y optimización, siendo usado por primera vez por John Holland[5]. Holland, creó un organismo electrónico, como una cadena de binarios o string de bits(denominado cromosoma), y tomando los principios de la genética y la evolución de las especies, tales como la selección y la reproducción(incluyendo cruce aleatorio o crossover y mutación), los utilizó para buscar una solución eficiente contando con un espacio enorme de solución. Si ejecuta el M-file de MATLAB® gaGraph.m de la Figura 1, podrá apreciar la magnitud de un problema de optimización, cuyo grafico se presenta en la Figura 2.

OPTIMIZACIÓN NUMÉRICA

Tradicionalmente, los GA han sido representados usando números binarios: El problema aparecía con optimizaciones de muchas variables y amplio dominio en el rango de búsqueda. Pero para efectos de análisis teóricos, permitían una elegante operación genética.

```
function gaGraph()

X=-2:0.01:12.1;
Y=4.1:0.01:5.8;
[x,y]=meshgrid(X,Y);
z=21.5+x.*sin(4*pi*x)+y.*sin(20*pi*y);
surf(x,y,z,gradient(z))
colormap cool
colorbar
shading interp
xlabel('x')
ylabel('y')
zlabel('z')
```

Figura 1. M-file gaGraph.m
Fuente: Elaboración propia, 2005.

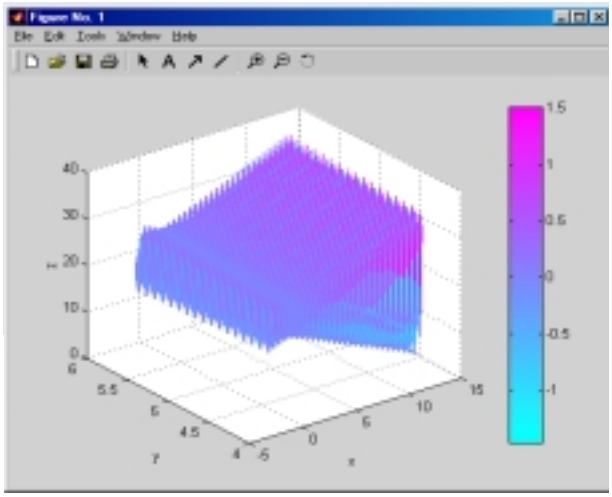

Figura 2. Gráfico de gaGraph.m
Fuente: Elaboración propia, 2005.

El dilema ha sido siempre binarios contra punto flotante o FP(de Floating Point). Una característica de la representación ha sido siempre el espacio de solución. Una propiedad importante, es que dos puntos cerrados al espacio de representación, también debe ser cerrado al espacio del problema y viceversa. En una representación en binario, la discrepancia de la distancia, se define por el número de diferentes posiciones de los bits. El código Gray, trajo la reducción de tales discrepancias.

El Cuadro 1, muestra algunos números binarios y su correspondiente código Gray. Note que esta clase de código, cumple con la propiedad anterior, porque el espacio difiere en un solo bit. El incremento de un valor, sólo significa el cambio de un solo bit.

El proceso de conversión de un string en binario $\mathbf{b} = (b_1, b_2, \cdots, b_n)$ a un código de Gray $\mathbf{g} = (g_1, g_2, \cdots, g_n)$, está definido por la operación XOR, como:

$$g_1 = b_1$$
$$g_k = b_{k-1}\ XOR\ b_k, \quad k = 2,3,$$

Así, para convertir el binario 0110 a código de Gray, se efectúan los cálculos del Cuadro 2.

Cuadro 1. Números binarios y su correspondiente código Gray

Entero	Binario	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101

La representación del punto flotante, es cerrada al espacio del problema y permite una fácil implementación de los operadores.

Las iniciales estrategias evolutivas, estaban basadas en poblaciones de un solo miembro. El único operador genético usado en los procesos de evolución es la mutación. La representación del individuo estaba basada en un vector; siendo $\mathbf{x}$ un punto en el espacio de solución y $\boldsymbol{\sigma}$ un vector de desviación estándar: El proceso de mutación reemplaza $\mathbf{x}$ por

$$\mathbf{x}^{t+1} = \mathbf{x}_t + N(0, \boldsymbol{\sigma})$$

siendo $N(0, \boldsymbol{\sigma})$ un vector de variable aleatoria Gaussiana.

La mutación individual, conocida como offspring, acepta como un nuevo miembro de la población (en reemplazo de su pariente); es decir $(\mathbf{x}^{t+1}, \boldsymbol{\sigma})$ reemplaza a $(\mathbf{x}^t, \boldsymbol{\sigma})$ si se cumple para caso de la minimización que $f(\mathbf{x}^{t+1}) < f(\mathbf{x}^t)$; en otro caso el offspring es eliminado, permaneciendo la población sin alteración.

El teorema de la Convergencia, establece que un óptimo global es hallado, con una probabilidad de

Cuadro 2. Conversión del binario 0110 a código Gray

K	b_k	Operación	Gray
1	0	0	0
2	1	0 XOR 1	1
3	1	1 XOR 1	0
4	0	1 XOR 0	1

1, en un tiempo de búsqueda suficientemente largo.

Teorema de la Convergencia
Para ó > 0 y optimización regular con $f_{\text{opt}} > -\infty$ (minimización) o $f_{\text{opt}} < \infty$ (maximización),

$$p\{\lim_{t\to\infty} f(\mathbf{x}^t) = f_{\text{opt}}\} = 1$$

La eficiencia de la mutación mejora con el tamaño del paso; es decir con desviación estándar. También el mejoramiento es función del tiempo.

La estructura de un programa evolutivo o EP, es presentada en la Figura 3, la cual es un algoritmo probabilístico que mantiene una población de individuos $P(t) = \{x^t_1, x^t_2, \cdots, x^t_n\}$ para la iteración *t*.

Zbigniew Michalewicz, ha acuñado la ecuación para los programas evolutivos que emula a la planteada por Nicklaus Wirth(el autor del lenguaje de programación Pascal):

$$Genetic\ A\lg orithms + Data\ Structures = Evolution\ \Pr ograms$$

para indicar que EP, son algoritmos que imitan los principios de la evolución natural para los problemas de optimización. Cada representación individual representa una potencial solución al problema; y es implementada en una estructura de datos, como la que se muestra en la Figura 4.

EVOLUCIÓN DIFERENCIAL

La evolución diferencial o DE, es una técnica de búsqueda paralela, directa y estocástica, de la optimización; la que es una simple y robusta optimización numérica.

La DE, maneja funciones objetivo no diferenciables, no lineales y multimodales. En una población con

```
Empezar en el tiempo t = 0
Inicializar población init P(t)
Evaluar P(t)
While no se termine
      Incrementar el tiempo t = t + 1
      Seleccionar subpoblación P'(t)
      Recombinar los genes
      Perturbar  mutar P'(t)
      Evaluar P'(t)
      Seleccionar sobrevivientes
End while
```

Figura 3. Pseudo código para GA
Fuente: Elaboración propia, 2005.

solución potencial en el espacio de búsqueda E - dimensional, un número fijo de vectores son inicializados en forma aleatoria; entonces en el tiempo se explora el espacio de búsqueda y se localiza el punto mínimo de la función objetivo.

En cada iteración denominada generación, nuevos vectores son generados por la combinación de vectores los que a su vez son generados en forma aleatoria desde la población. Esto es lo que se denomina mutación. La operación denominada de recombinación produce un vector de prueba o trial.

El vector trial, es aceptado para la siguiente generación, si y solo si existe una reducción en el valor de la función objetivo. Esta operación es conocida como selección.

La DE usa las mutaciones como mecanismos de búsqueda y selección, para llevar a cabo la búsqueda en las regiones del espacio de solución. GA se encarga de generar una secuencia de poblaciones usando mecanismos de selección. GA usa crossover y mutación como mecanismo de búsqueda.

La principal diferencia entre GA y DE es que GA toma un crossover, como un mecanismo de probabilidad y de útil intercambio de información entre las soluciones para localizar la mejor solución; mientras que ES usa la mutación como el principal mecanismo de búsqueda. La Evolución Diferencial o DE usa un crossover no uniforme. Usando componentes de los miembros de poblaciones existentes, construye un vector trial con información de combinaciones exitosas, con la finalidad focalizar la búsqueda del óptimo en las regiones mas promisorias del espacio de solución.

Una vez que la nueva solución trial ha sido generada y seleccionada, se determina cual sobrevive en la siguiente generación. DE mantiene dos arreglos. El arreglo primario, que almacena la población actual, mientras que el arreglo secundario que almacena la selección para la siguiente generación.

Debido a que la eficiencia de la mutación también es una función del tiempo, una fuente apropiada de ruido, corresponde a la población misma.

Una forma conveniente de perturbar a la población; es tomando cada par de vectores (x_a, x_b), y definir un vector diferencial $(x_a - x_b)$. Cuando ambos son generados en forma aleatoria, este vector de diferencia puede ser utilizado en reemplazo del ruido Gaussiano, de las tempranas ES de los vectores de mutación, que empleaban un ruido $N(0, \boldsymbol{\sigma})$ (véase Figura 5).

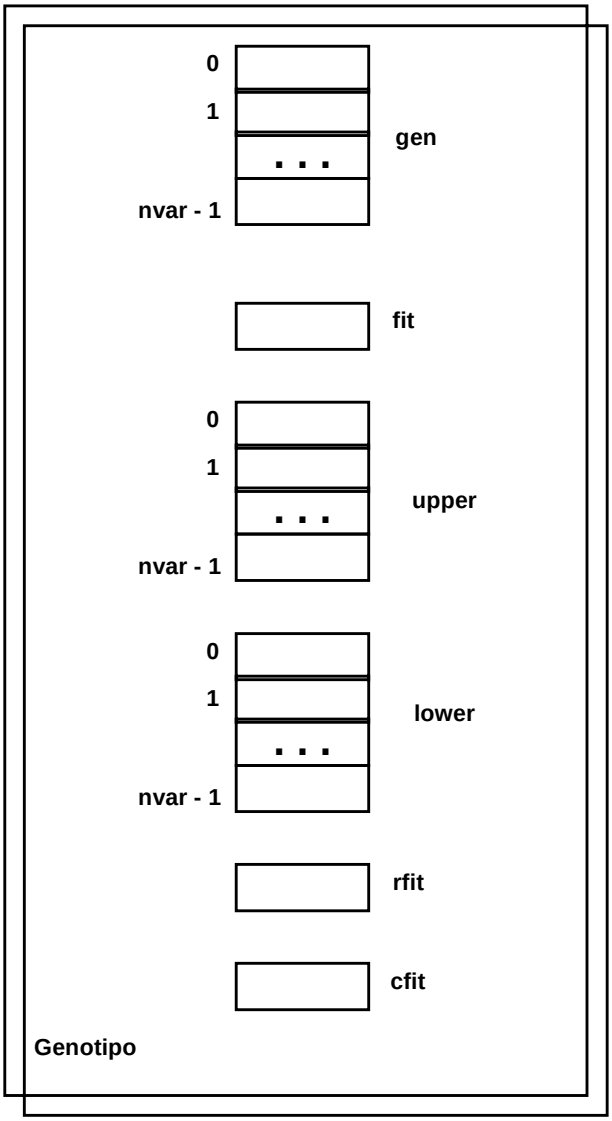

Figura 4. Estructura de datos para un genotipo

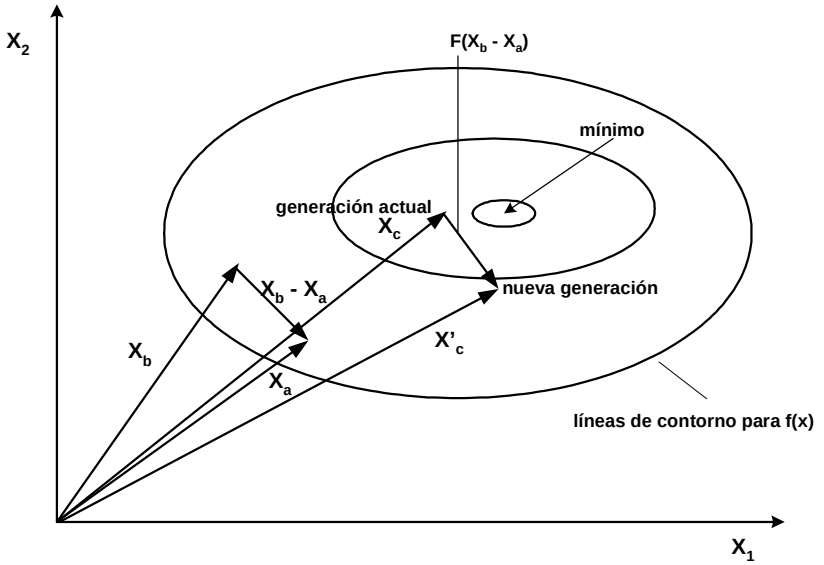

Figura 5. Esquema de mutación

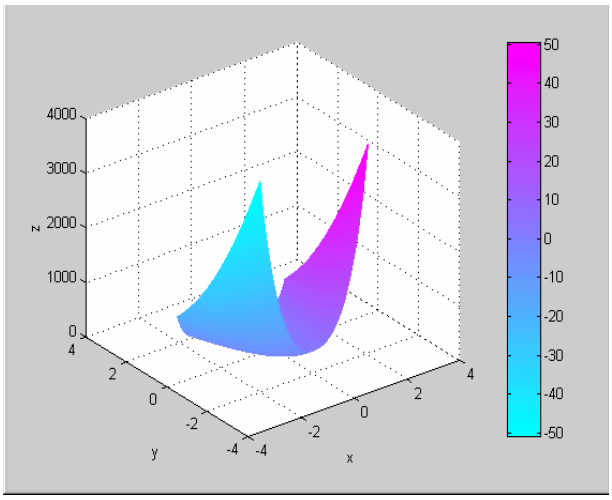

Figura 6. Salida para graphF2.m
Fuente: Elaboración propia, 2005.

```
function graphF2()

X=-2.048:0.01:2.048;
[x,y]=meshgrid(X);
z=100*(x.^2-y).^2+(1-x).^2;
surf(x,y,z,gradient(z))
colorbar
colormap cool
shading interp
xlabel('x')
ylabel('y')
zlabel('z')
```

Figura 7. M-file graphF2
Fuente: Elaboración propia, 2005.

Este proceso queda expresado matemáticamente como:

$$x_c' = x_c + F(x_a - x_b)$$

El factor *F*, es un factor de escala entre el rango de $0 \le F \le 1{,}2)$. Los autores aconsejan un valor en el rango de 0,4 a 1,0.

Diferentes estrategias, pueden ser adoptadas en algoritmos DE; dependiendo del tipo de problemas para los cuales DE es aplicado. Las estrategias pueden variar al vector a ser perturbado, en el número de vectores de diferencia a ser considerados en la perturbación y en el tipo de crossover utilizado. En la página http://www.ICSI.Berkeley.edu/~storn/code.html, se encuentran diez diferentes estrategias recomendadas:

1. DE/best/1/exp
2. DE/rand/1/exp
3. DE/rand to best/1/exp
4. DE/best/2/exp
5. DE/rand/2exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/rand to best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

La nomenclatura es ES/x/y/z; donde para ES le corresponde DE, x representa el string del vector perturbado, y el número de vectores de diferencia y z el tipo de crossover (sea exp = exponencial y bin = binomial).

La estrategia DE/rand/1/bin, es la de mayor éxito y la más utilizada. Los parámetros de control en DE son: NP, el tamaño de la población, CR, la constante de crossover y *F* el peso aplicado a la diferencia aleatoria.

Existen varias funciones de test o prueba, que pueden ser usadas en experimentos de Benchmark para algoritmos en ES. Son clásicos los DeJong function Fx, Schaffer y Bohachevsky. En la Figura 6, se presenta la function F2, conocida como de Rosembrock. En la Figura 7, el M-file grapF2.m, que ejecuta esta gráfica.

IMPLEMENTACIÓN DEL PROGRAMA DiffEvol

El programa que se presenta en la Figura 8, ha sido escrito en MATLAB©, y hace uso de la DE. Los parámetros son: la función f, los límites inferiores y superiores de los parámetros(o variables), CR, F y el número máximo de iteraciones.

El programa calcula los valores de NP y D, desde la longitud del vector low (o upp); y utilizando la regla: NP = 5*D, determina el tamaño de la población. Esta regla es una de las dos, que sugieren los autores: NP es 5 o 10 veces el valor de D.

El proceso de iniciación se resume en generar los valores de las variables, para toda la población. Esto se presenta en el código mostrado por la Figura 9.

El proceso de mutación, genera los valores aleatorios a, b y c y desarrolla pruebas binomiales, definiendo el vector trial (véase la Figura 10).

Finalmente el proceso de evaluación/selección, al determina el menor costo encuentra la nueva pobla-

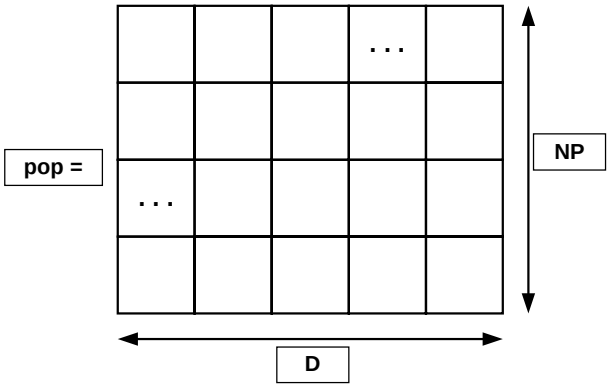

Figura 8. El proceso de mutación en DE
Fuente: Elaboración propia, 2005.

```
% inicio
for i=1:NP
   for j=1:D
      pop(i,j)=low(j)+rand*(upp(j)-
      low(j));
   end
end
```

Figura 9. Código para el proceso de iniciación
Fuente: Elaboración propia, 2005.

```
for i = 1 to NP
   Generar a, b, c entre [ 1, NP ]
   Generar j entre [ 1, D]
   for  k = 1 to D
      if rand < CR or k = D
         trialj = pop c,j + F( . . . )
      else
         trialj = pop i,j
         j = j + 1
   endfor
endfor
```

Figura 10. Código para el proceso de mutación en DE
Fuente: Elaboración propia, 2005.

```
value=feval(f,trial);
   if value<=val(i);
      newPop(i,:)=trial;
      val(i)=value;
   else
      newPop(i,:)=pop(i,:);
   end
```

Figura 11. Código para el proceso de evaluación/selección
Fuente: Elaboración propia, 2005.

```
function[min,popBest]=DiffEvol(f,low,upp,CR,F,maxIter)
% A simple evolution strategy for fast optimization
% E. Raffo Lecca
%
% Numerical Optimization
% Datos
% f      =function with D parameters
% NP     =population size
% D      =number of variable
% CR     =crosover constant
% F      =scaling factor
% low    =lower bound
% upp    =upper bound
% maxIter=maximun number of generations
% pop    =population
% newPop =new population
% popBest=best population
% val    =cost
% Resultados
% min =optimun

ERROR=0.000001;
D=length(low);
NP=5*D;
pop=zeros(NP,D);
newPop=zeros(NP,D);
popBest=zeros(1,D);
val=zeros(1,NP);
trial=zeros(1,D);
% inicio
for i=1:NP
  for j=1:D
```

Figura 12a. M-file diffEvol.m

```
            pop(i,j)=low(j)+rand*(upp(j)-low(j));
    end
end
ibest=1;
val(1)=feval(f,pop(ibest,:));
min=val(1);
for i=2:NP
  val(i)=feval(f,pop(i,:));
  if val(i)<min
     min=val(i);
     ibest=i;
    end
end
popBest=pop(ibest,:);
% loop
iter=0;
while (iter<maxIter & min >ERROR)
   for i=1:NP
     % mutacion/recombinacion
     while 1
        a=floor(rand*NP)+1;
        if a~=i
           break;
        end
      end
     while 1
        b=floor(rand*NP)+1;
        if b~=i & b~=a
           break;
        end
      end
     while 1
        c=floor(rand*NP)+1;
        if c~=i & c~=b & c~=a
           break;
```

```
        end
      end
     j=floor(rand*D)+1;
      for k=1:D
        % pruebas binomiales
        if rand < CR | k==D

           trial(j)=pop(c,j)+F*(pop(a,j)-pop(b,j));
        else
           trial(j)=pop(i,j);
        end
       j=rem(j,D)+1;
      end
      % Evaluate/select
           value=feval(f,trial);
     if value<=val(i);
       newPop(i,:)=trial;
       val(i)=value;
      else
       newPop(i,:)=pop(i,:);
     end
   end
  % new generation
 pop(:,:)=newPop(:,:);
 iter=iter+1;
 ibest=1;
 min=val(1);
 for i=2:NP
           if val(i)<min
                                  min=val(i);
                                  ibest=i;
           end
  end
  popBest=pop(ibest,:);
 end % End loop
```

Figura 12b. M-file diffEvol.m

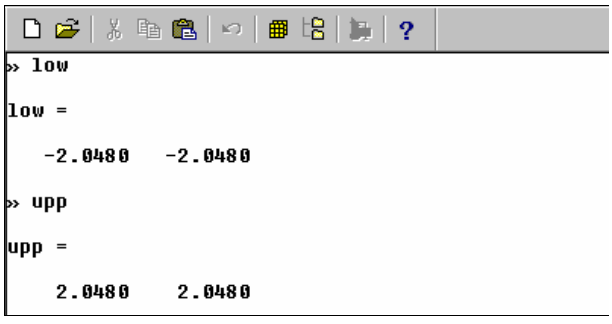

Figura 13. Entrada de los límites
Fuente: Elaboración propia, 2005.

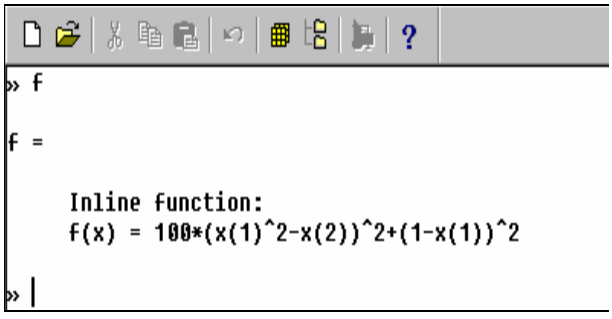

Figura 14. Definición de la función
Fuente: Elaboración propia, 2005.

```
» [min x]=DiffEvol(f,low,upp,0.1,0.7,314)
min =
  2.7211e-007
x =
    1.0000    1.0001
```

Figura 15. Salida de diffEvol.m
Fuente: Elaboración propia, 2005.

ción. El código de dicho proceso se muestra en la Figura 11.

El programa retorna el óptimo: tanto de la función objetivo, como del vector de variables. El código completo del M-File se presenta en la Figura 12 (a y b).

EJECUCIÓN DE DiffEvol

La ecuación del punto de silla de Rosembrock (Rosembrock´s saddle) más conocida como el test DeJong function F2, será la prueba para el programa DiffEvol.m. Esta ecuación se define como:

$$100(x^2{}_1 - x_2)^2 + (1 - x_1)^2, \quad donde \ -2.048 \le x_i \le 2.048$$

El mínimo se cumple en: $f_2(\mathbf{x}) = 0, \mathbf{x} = (1,1)$

En las figuras 13, 14 y 15, se presentan las entradas de los límites Inferiores y superiores para los parámetros de búsqueda, la función a optimizar; y la salida del programa. Se observa que concuerda el óptimo.

CONCLUSIONES

El algoritmo DE, es robusto porque reproduce los mismos resultados consistentemente sobre muchas pruebas, allí donde algoritmos como el PSO (Particle Swarm Optimization) fallan. DE es un algoritmo de búsqueda basado en poblaciones, el cual es una versión mejorada de GA. DE, ha resultado muy eficiente en la solución de optimización de muchas variables.

La implementación en MATLAB, permite la ejecución de funciones de optimización, de un modo simple y seguro; toda vez que los clásicos métodos de optimización son inconvenientes al resolver problemas de optimización con múltiples objetivos.

El programa que se presenta en este artículo, es una implementación mejorada y eficiente al programa que presentan Price [9] y Storm [11] y [12].

BIBLIOGRAFIA

1. Babu, B.V., Leenus Jehan, M. (2005). *Differential Evolution for Multi-Objective Optimization*. En: http://discovery.bits-pilani.ac.in/discipline/chemical/BVb/RevisedBabuLeenus%20CEC2003.pdf (visitado el 10 de Noviembre del 2005) .

2. Christos Dimopoulos, Ali M. S. Zalzala,.(2000). *Recent Development in Evolutionary Computation for Manufacturing Optimization: Problems, Solutions, and Comparisons.* IEEE Transactions On Evolucionary Computation, Vol. 4, No. 2, July. USA.

3. Corne, D.; Dorigo, M. and Glover, F. (1999). *New Ideas in Optimization.* McGraw-Hill. New York, USA.

4. Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley. USA.

5. Holland, H.J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press. USA.

6. Michalewicz, Z. (1996). *Genetic Algorithms+Data Structures=Evolution Programs.* Springer-Verlag. Berlin, Germany.

7. Osman, I.H. and Kelly, J.P. (1996). *Meta-Heuristics:Theory and Applications.* Kluwer. Dordrecht, Netherlands.

8. Plagianakos, V.P. and Vrahatis, M.N. (2002). *Parallel Evolutionary Training Algorithms for Hardware-Friendly Neural Networks.* Natural Comp. 1, 307-322.

9. Price, K. and Storn, R. (1997). *Differential Evolution.* Dr. Dobb's J., Issue 264, 18-24 and 78, April.

10. Raffo Lecca, E. (2005). *Métodos Numéricos para Ciencia e Ingeniería con MATLAB.* Lima, Perú.

11. Storn, R. (1999). *System Design by Constraint Adaptation and Differential Evolution.* IEEE Transactions On Evolucionary Computation, Vol. 3, No 1, April. USA.

12. Storn, R. and Price, K. (1997). *Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*. Journal of Global Optimization 11, 341-359. USA.