MÉTODOS MODERNOS

CESÁREO RAIMÚNDEZ

1. Métodos que Prescinden de las Derivadas

Entre los métodos que prescinden de la regularidad del modelo a optimizar se encuentran las técnicas más modernas basadas en paradigmas naturales y comportamentales. Estos métodos son viables gracias a la potencia de computación (*number crunching capacity*) disponible hoy día. Dentro de un enfoque todavía clásico, se pueden citar técnicas como método el simplex secuencial adaptativo (Nelder & Mead) y el método de la búsqueda aleatoria (*random search*). Ya el enfoque moderno se apoya en paradigmas evolutivos, basados en el modelo darwiniano de la evolución natural, paradigmas comportamentales, basados en el comportamiento de poblaciones en la búsqueda de alimentos, (enjambres, colonias de hormigas), modelos de energía y su importancia en la cristalización de los metales (recocido simulado), etc.

2. Método del Simplex Secuencial Adaptativo

Definición 2.1. Simplex La figura geométrica formada por un conjunto de $n+1$ puntos en un espacio de n dimensiones se llama simplex. Cuando los puntos son equidistantes el simplex se dice regular. Así en dos dimensiones, el simplex es un triángulo y en tres dimensiones es un tetraedro. La idea básica en este método es la de comparar el valor de la función objetivo en los $n+1$ vértices y mover el simplex gradualmente rumbo al punto de óptimo a través de un proceso iterativo, efectuando operaciones sobre el simplex original como contracción, expansión y reflexión. Las expresiones para generar un simplex n dimensional son:

$$\begin{aligned} X_i &= X_0 + pu_i + \sum_{j\neq i}^{n} qu_j \\ p &= \frac{a}{n\sqrt{2}}(\sqrt{n+1}+n-1) \\ q &= \frac{a}{n\sqrt{2}}(\sqrt{n+1}-1) \end{aligned} \tag{2.1}$$

con X_0 punto inicial, a el *tamaño* del simplex y los u_i, $\{i=1,\ldots,n\}$ son los versores de los ejes independientes en $\mathbb{R}^n$

2.1. Reflexión. Si X_h es el vértice correspondiendo al valor más elevado de la función objetivo entre los vértices del simplex, se puede esperar que el punto X_r obtenido a través de la reflexión del punto X_h en hiperplano representado por los demás n puntos asuma un valor más pequeño. Si este es el caso, se puede obtener un nuevo simplex a partir del anterior, rechazando el punto X_h e incorporando el nuevo punto X_r. Este proceso se puede observar en la figura 1 para los casos $n=2$

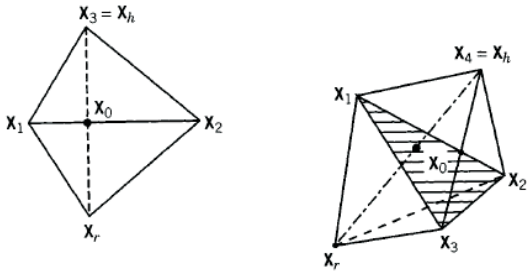

FIGURA 1. Reflexión

y $n = 3$. Si la función objetivo en convexa, el procedimiento descrito conlleva a un camino conforme ilustrado en la figura 2

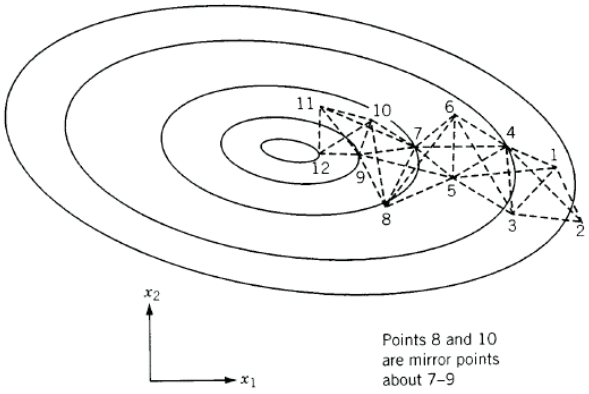

FIGURA 2. Progreso de la operación de reflexión

Le expresión matemática de la reflexión es

$$X_r = (1 + \alpha)X_0 - \alpha X_h \tag{2.2}$$

siendo que

$$\alpha = \frac{\|X_r - X_0\|}{\|X_h - X_0\|}$$

$$X_0 = \frac{1}{n} \sum_{i \neq h}^{n+1} X_i$$

donde

$$X_h = \arg \max_{i=1}^{i=n+1} f(X_i), \; X_l = \arg \min_{i=1}^{i=n+1} f(X_i)$$

si ahora $f(X_l) < f(X_r) < f(X_h)$ entonces X_h es substituido por X_r y se pasa al nuevo simplex. La utilización de apenas el proceso de reflexión no es suficiente para aproximarnos lo suficiente al mínimo.

2.2. Expansión. Si el proceso de reflexión proporciona un punto tal que $f(X_r) < f(X_l)$ se calcula

$$X_e = \gamma X_r + (1-\gamma)X_0$$

donde γ es un coeficiente de expansión tal que

$$\gamma = \frac{\|X_e - X_0\|}{\|X_r - X_0\|} > 1$$

si en el proceso de expansión ocurre que $f(X_e) < f(X_l)$ entonces X_h se substituye por X_e y se prosigue con el nuevo simplex. Si por otro lado $f(X_e) > f(X_l)$ entonces el proceso de expansión no llevó a buen puerto, y se substituye X_h por X_r continuando el proceso.

2.3. Contracción. Si el proceso de reflexión proporciona un punto X_r para el que se verifica $f(X_r) > f(X_i)$ para todo $i \neq h$ siendo $f(X_r) < f(X_h)$, substituimos el punto X_h por el punto X_r. Así el nuevo punto será X_r. En este caso contraeremos el simplex como sigue:

$$X_c = \beta X_h + (1-\beta)X_0$$

donde $0 \leq \beta \leq 1$ es el coeficiente de contracción y se define como

$$\beta = \frac{\|X_e - X_0\|}{\|X_h - X_0\|}$$

Si $f(X_r) > f(X_h)$ utilizaremos nuevamente la ecuación de contracción sin cambiar el punto X_h. Si la contracción produce un punto X_c para el que se verifica $f(X_c) < \text{mín}\,(f(X_h), f(X_r))$, sustituimos el punto X_h por X_c y proseguimos con las operaciones de reflexión. Si de otro modo ocurre que $f(X_c) \geq \text{mín}\,(f(X_h), f(X_r))$ el proceso de contracción fue un fracaso substituyendo X_i por $(X_i + X_l)/2$ volviendo al proceso de reflexiones. Este proceso converge para un mínimo local siempre y cuando se verifique

$$Q = \left\{ \sum_i^{n+1} \frac{(f(X_i) - f(X_0))^2}{n+1} \right\} \leq \epsilon$$

2.4. Algoritmo MATLAB para el Simplex Secuencial Adaptativo.

```
%---------------------------
% Programa Principal
%---------------------------
f = inline('x(1)*(x(1)-4-x(2)) +x(2)*(x(2)-1)','x');
x0        = [0 0];
TolX      = 1e-4;
TolFun    = 1e-9;
MaxIter   = 100;
[xon,fon] = opt_Nelder(f,x0,TolX,TolFun,MaxIter)



function [xo,fo] = opt_Nelder(f,x0,TolX,TolFun,MaxIter)
N = length(x0);

if N == 1 % para la búsqueda en dimensión 1
```

```
    [xo,fo] = opt_quad(f,x0,TolX,TolFun);
    return
end

S = eye(N);

for i = 1:N
    i1 = i + 1;

    if i1 > N
        i1 = 1;
    end

    abc = [x0; x0 + S(i,:);
    x0 + S(i1,:)];
    fabc = [feval(f,abc(1,:));
    feval(f,abc(2,:));
    feval(f,abc(3,:))];
    [x0,fo] = Nelder0(f,abc,fabc,TolX,TolFun,MaxIter);

    if N < 3
        break;
    end
end

xo = x0;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------
% Las funciones opt_quad, opt_quad0 son requeridas apenas para el caso
% de dimension 1
%-------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [xo,fo] = opt_quad(f,x0,TolX,TolFun,MaxIter)
%-------------------------------------------------------------------
% búsqueda del mínimo de f(x) a través de la aproximación cuadrática
%-------------------------------------------------------------------
if length(x0) > 2
    x012 = x0(1:3);
else
    if length(x0) == 2
         a = x0(1);
         b = x0(2);
    else
         a = x0 - 10;
         b = x0 + 10;
```

```
    end
    x012 = [a (a + b)/2 b];
end

f012 = f(x012);
[xo,fo] = opt_quad0(f,x012,f012,TolX,TolFun,MaxIter);



function [xo,fo] = opt_quad0(f,x012,f012,TolX,TolFun,k)
x0 = x012(1);
x1 = x012(2);
x2 = x012(3);
f0 = f012(1);
f1 = f012(2);
f2 = f012(3);
nd = [f0 - f2 f1 - f0 f2 - f1]*[x1*x1 x2*x2 x0*x0; x1 x2 x0]';
x3 = nd(1)/2/nd(2);
f3 = feval(f,x3);

if k <= 0 | abs(x3 - x1) < TolX | abs(f3 - f1) < TolFun
    xo = x3;
    fo = f3;
if k == 0
    fprintf('El mejor en # iteraciones')
    end
else
    if x3 < x1
        if f3 < f1
            x012 = [x0 x3 x1];
            f012 = [f0 f3 f1];
        else x012 = [x3 x1 x2];
            f012 = [f3 f1 f2];
        end
     else
        if f3 <= f1
            x012 = [x1 x3 x2];
            012 = [f1 f3 f2];
        else x012 = [x0 x1 x3];
            f012 = [f0 f1 f3];
        end
     end
     [xo,fo] = opt_quad0(f,x012,f012,TolX,TolFun,k - 1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [xo,fo] = Nelder0(f,abc,fabc,TolX,TolFun,k)
[fabc,I] = sort(fabc);
a   = abc(I(1),:);
b   = abc(I(2),:);
c   = abc(I(3),:);
fa  = fabc(1);
fb  = fabc(2);
fc  = fabc(3);
fba = fb - fa;
fcb = fc - fb;

if k <= 0 | abs(fba) + abs(fcb) < TolFun | abs(b - a) + abs(c - b) < TolX
    xo = a;
    fo = fa;

    if k == 0
        fprintf('el mejor en # de iteraciones')
    end
else
    m = (a + b)/2;
    e = 3*m - 2*c;
    fe = feval(f,e);

    if fe < fb
        c = e;
        fc = fe;
    else
        r = (m+e)/2;
        fr = feval(f,r);

        if fr < fc
            c = r;
            fc = fr;
        end

        if fr >= fb
            s = (c + m)/2;
            fs = feval(f,s);

            if fs < fc
                c = s;
                fc = fs;
            else
                b = m;
                c = (a + c)/2;
                fb = feval(f,b);
                fc = feval(f,c);
            end
```

```
        end
    end

    [xo,fo] = Nelder0(f,[a;b;c],[fa fb fc],TolX,TolFun,k - 1);
end
```

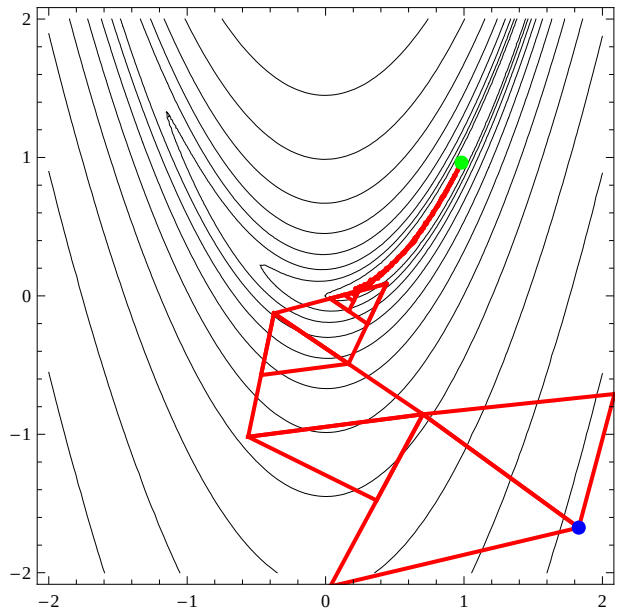

FIGURA 3. Traza del proceso Nelder-Mead de minimización, en la función $f(x, y) = (x - 1)^2 + 100(y - x^2)^2$

3. BÚSQUEDA ALEATORIA - *(Random Search)*

Los procedimientos de búsqueda aleatoria se basan en la utilización de generadores de números aleatorios en la búsqueda del punto de mínimo. Sea el problema

$$\min_{x \in \Omega(X)} f(x)$$

donde

$$\Omega(X) = \begin{cases} x_1 &= l_1 + r_1(u_1 - l_1) \\ x_2 &= l_2 + r_2(u_2 - l_2) \\ \vdots & \vdots \\ x_n &= l_n + r_n(u_n - l_n) \end{cases}$$

con $\{l_i \leq x_i \leq u_i\}$ intervalos del espacio viable y con $\{0 \leq r_i \leq 1\}$ números aleatorios. Efectuando los cálculos de la función sobre un reticulado de puntos generados aleatoriamente y guardando el de valor mínimo, se puede estimar en algunos casos, la región donde se sitúa el mínimo absoluto, pudiendo proseguir la búsqueda con un método más regular.

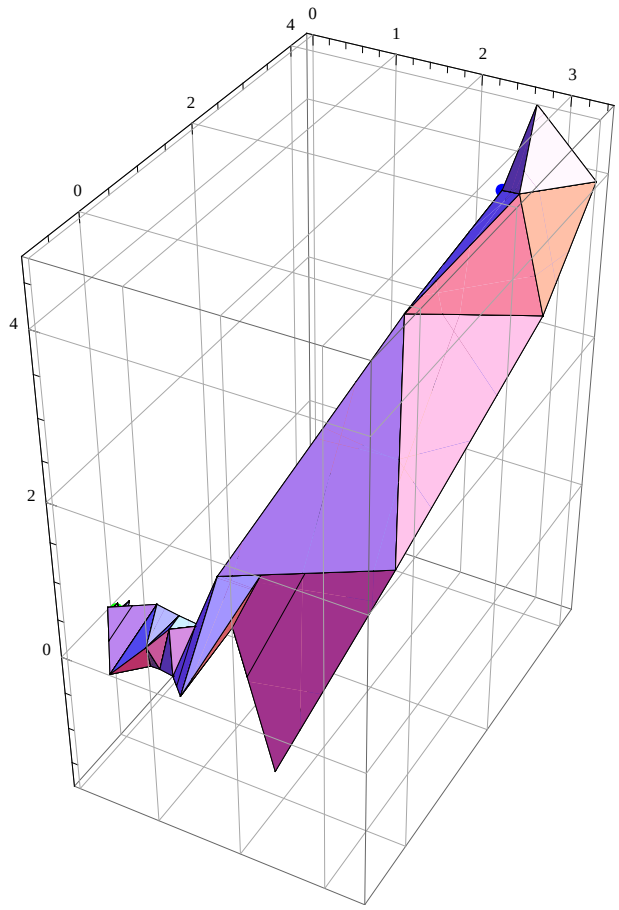

FIGURA 4. Traza del proceso Nelder-Mead de minimización, en la función $f(x, y, z) = x^2 + y^2 + z^2$

4. PASEO ALEATORIO - *(Random Walk)*

En este método se progresa a partir de un punto generado aleatoriamente, según una dirección generada aleatoriamente. Así empezando en X_1 se genera una dirección aleatoria u_1 y se camina a lo largo de esta dirección, asta el punto $X_2 = X_0 + \lambda u_1$. λ representa la largura del paso dado y puede modificarse a lo largo del camino. Resumiendo, el algoritmo podría representarse como:

1. Inicio en el punto X_1, con un paso λ lo suficiente grande, un paso mínimo estipulado ϵ y un número máximo de intentos N.
2. Determinar el valor de $f_1 = f(X_1)$
3. Establecer $i = 1$.
4. Generar un conjunto de n números aleatorios $\{r_1, r_2, \ldots, r_n\}$ en el intervalo $[-1, 1]$ calculando $R = r_1^2 + r_2^2 + \cdots + r_n^2$. Si $R \leq 1$ se acepta como bueno el conjunto de n valores aleatorios generado (si $R > 1$ se genera un nuevo

conjunto de valores rechazando el anterior) y definiendo

$$u = \frac{1}{\sqrt{R}} \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix}$$

5. Calcular $X = X_1 + \lambda u$ y $f = f(X)$
6. Comparar los valores de f y f_1. Si $f < f_1$ establece $X_1 = X$ y $f_1 = f$ y prosigue al paso 3. Si $f \geq f_1$ se sigue al paso 7.
7. Si $i \leq N$ se hace $i = i + 1$ y se sigue en el paso 4. Si $i > N$ se sigue al paso 8.
8. Computar el nuevo paso como $\lambda = \lambda/2$. Si ahora $\lambda \leq \epsilon$ se sigue en 9. Caso contrario se sigue al 4.
9. Parar el proceso tomando como solución $X_{min} \approx X_1$ y $f_{min} \approx f_1$

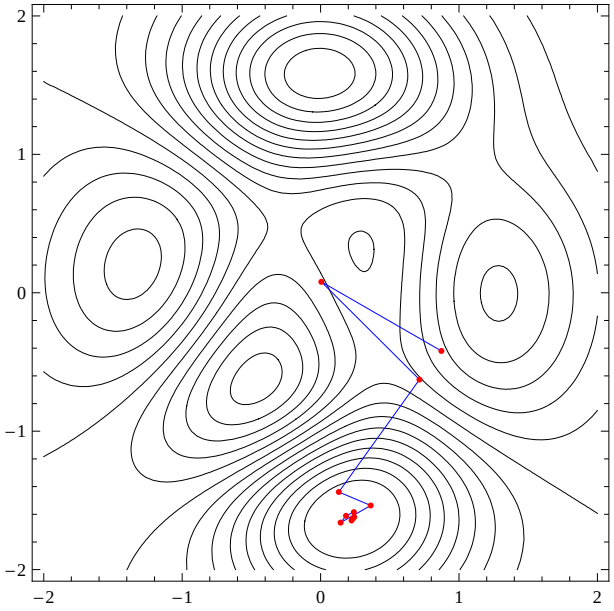

FIGURA 5. Evolución del proceso *random walk*, en la función $f(x,y) = 3e^{-x^2-(y+1)^2}(1-x)^2 - \frac{1}{3}e^{-(x+1)^2-y^2} - 10e^{-x^2-y^2}\left(-y^5 - x^3 + \frac{x}{5}\right)$

5. RECOCIDO SIMULADO - *(Simulated Annealing)*

La obtención del mínimo global de una función que presenta varios mínimos locales puede ser un verdadero problema. una de las formas de sortear este tipo de problema sería efectuar inicialmente un barrido inicial utilizando un algoritmo del tipo búsqueda aleatoria y proseguir a partir del mejor punto, con un procedimiento de refinamiento de soluciones. Un método alternativo elegante y potente es el método del recocido simulado que propone una búsqueda del mínimo global a partir de cualquier punto inicial, sin dejarse atrapar por los eventuales mínimos locales

presentes. La técnica utiliza la analogía entre el recocido y la búsqueda del punto de mínimo. El recocido el procedimiento por el cual se calienta un metal sólido por encima de su temperatura de fusión, sometiéndolo a un proceso de enfriamiento lento lo suficiente para que los átomos con más energía puedan llegar a formar una estructura cristalina regular, lo que representa una configuración de mínima energía potencial inter-atómica. Un enfriamiento rápido llevaría a cristalizados imperfectos (mínimos locales).

El procedimiento del recocido simulado se implementa utilizando la fórmula de distribución de probabilidades de Boltzmann que describe la probabilidad de ocurrencia de niveles de energía $E \geq 0$ a una temperatura T determinada.

$$p(E) = \frac{1}{kT} e^{-\frac{E}{kT}}$$

donde k es la constante de Boltzmann. El algoritmo para búsqueda en $\mathbb{R}^n$ posee la estructura:

1. Se toma el punto inicial x_0, el límite inferior l_b y el superior u_b, el número máximo de iteraciones k_{max}, el factor de recocido $q > 0$ y la tolerancia en la determinación del mínimo ϵ.
2. Se hace $x = x_0,\ x^o = x,\ f^o = f(x)$.
3. Para $k = 1$ hasta k_{max} se hace:
   a) Generar Δx por un proceso aleatorio conveniente tomándose $x_1 = x + \Delta x$ tal que $l_b \leq x_1 \leq u_b$.
   b) Si $\Delta f = f(x_1) - f(x) < 0$ {entonces $x = x_1$. Si también $f(x) < f^o$ {entonces $x^o = x$ y $f^o = f(x^o)$}}.
   c) Si $\Delta f = f(x_1) - f(x) \geq 0$ entonces generar aleatoriamente $z \in [0, 1]$. Si ahora
   $$z < e^{-(\frac{k}{k_{max}})^q \frac{\Delta f}{|f(x)|\,\epsilon}}$$
   entonces $x = x_1$.

Al final del proceso, el punto x^o estará convenientemente ubicado cercano al punto de mínimo global. A seguir, se presenta código MATLAB para determinar el mínimo de una función $f(x)$ para $l_b \leq x \leq u_b$, por el método del recocido simulado.

```
%-----------------------------------------------------------
% minimizar f(x) por recocido simulado - programa principal
%-----------------------------------------------------------
f       = inline('x(1)^4 - 16*x(1)^2 - 5*x(1) + x(2)^4 - 16*x(2)^2 - 5*x(2)','x');
l       = [-5 -5]; %limite inferior
u       = [ 5  5]; %limite superior
x0      = [0 0]
f0      = feval(f,x0)
TolX    = 1e-4;
TolFun  = 1e-9;
kmax    = 500;
q       = 1;
[xo_sa,fo_sa] = sim_anl(f,x0,l,u,kmax,q,TolFun)
```

```
function [xo,fo] = sim_anl(f,x0,l,u,kmax,q,TolFun)
%---------------------------------------------------------------------
% método del recocido simulado para minimizar f(x) para l <= x <= u
%---------------------------------------------------------------------
N  = length(x0);
x  = x0;
fx = feval(f,x);
xo = x;
fo = fx;

if nargin < 7
    TolFun = 1e-8;
end
if nargin < 6
    q = 1; % factor de recocido
end
if nargin < 5
    kmax = 100; % número máximo de iteraciones
end

for k = 0:kmax
    Ti = (k/kmax)^q; % inverso de la temperatura entre 0 y 1
    mu = 10^(Ti*100);
    dx = mu_inv(2*rand(size(x))- 1,mu).*(u - l); % incremento
    x1 = x + dx;                                 % próximo intento
    % confinar x1 en la región limitada por l_b and u_b.
    x1 = (x1 < l).*l +(l <= x1).*(x1 <= u).*x1 +(u < x1).*u;

    fx1 = feval(f,x1);
    df  = fx1 - fx;

    if df < 0|rand < exp(-Ti*df/(abs(fx) + eps)/TolFun)
       x  = x1;
       fx = fx1;
    end
    if fx < fo
       xo = x;
       fo = fx1;
    end

end


function x = mu_inv(y,mu)
% obtención del incremento en x
```

```
x = (((1+mu).^abs(y)- 1)/mu).*sign(y);
```

El método del recocido simulado puede aplicarse con éxito en la resolución de problemas combinatorios como por ejemplo, el problema del viajero que debe visitar una serie de ciudades pasando una sola vez por cada una, de modo a minimizar la distancia recorrida.

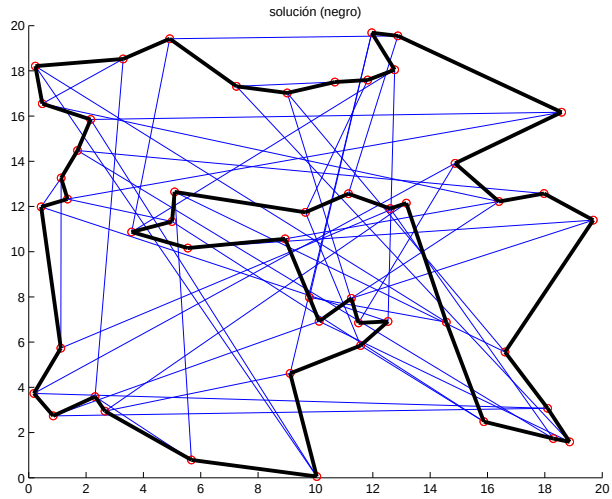

FIGURA 6. Solución al problema del viajante en 50 ciudades

UNIVERSIDAD DE VIGO - ETSII - INGENIERÍA DE SISTEMAS Y AUTOMÁTICA
E-mail address: cesareo@uvigo.es