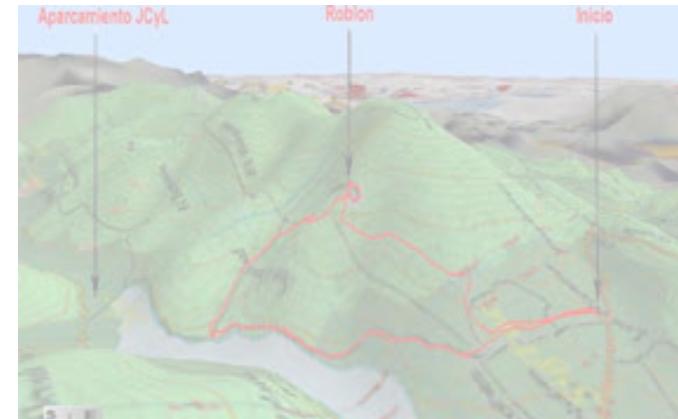
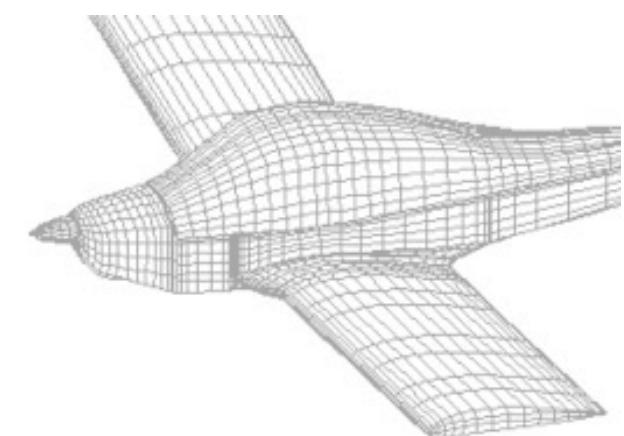


TEMA 4 OPTIMIZACIÓN

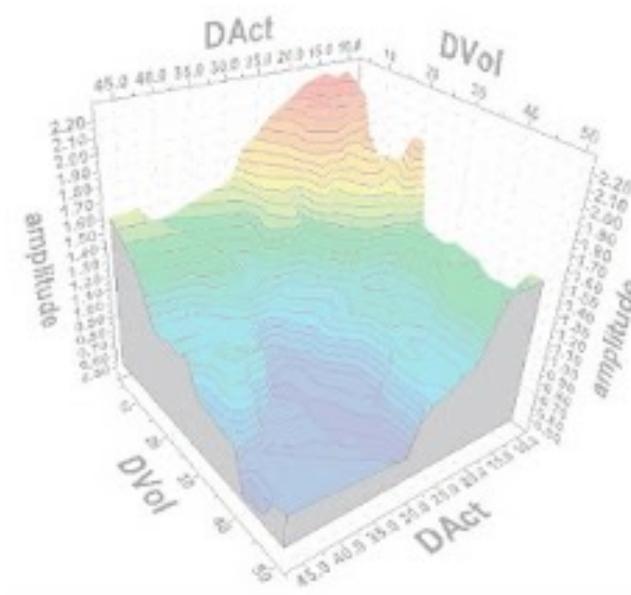


(1 sesión teórica + 2 sesiones prácticas)



Rocío del Río (rrio@us.es)

Dpto. Electrónica y Electromagnetismo





Contenidos

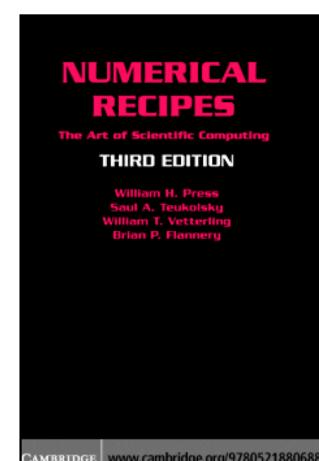
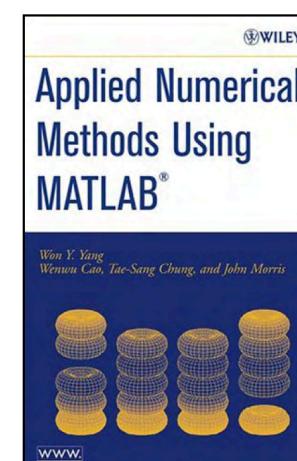
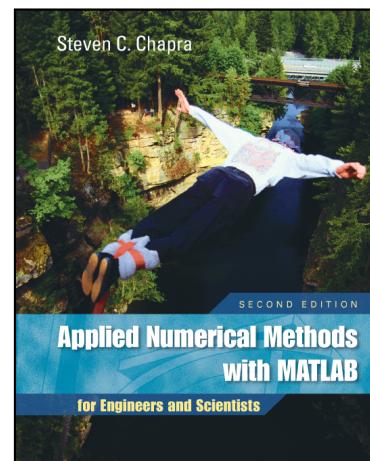
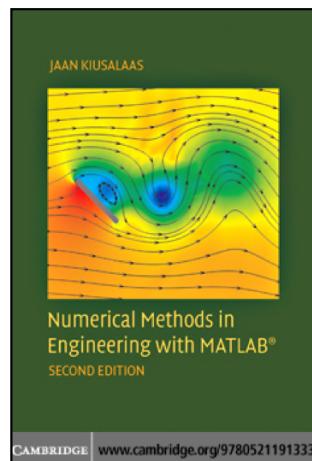
- Introducción
- Optimización local de funciones de una variable:
 - Bracketing
 - Golden-section search (GS), Interpolación parabólica (IP), Función fminbnd
- Optimización local de funciones multi-variable:
 - Downhill Simplex, Función fminsearch
- Optimización local de funciones con restricciones (fmincon)
- Introducción a la optimización global:
 - Multi-Start, Enfriamiento simulado (SA), Algoritmos genéticos (GA)
- Toolboxes y funciones de MATLAB

Objetivos

- Comprender la diferencia entre:
 - Optimización local y global
 - Optimización sin y con ligaduras
 - Optimización uni- y multi-dimensional
 - Métodos directos e indirectos
- Saber reformular un problema de búsqueda de máximos para resolverlo mediante un algoritmo de minimización.
- Minimización uni-dimensional:
 - Saber aplicar bracketing para acotar un intervalo unimodal
 - Saber aislar el mínimo con una tolerancia dada (GS, IP, `fminbnd`)
- Minimización multi-dimensional:
 - Saber representar superficies y curvas de contorno
 - Saber localizar el mínimo (`downhill simplex`, `fminsearch`)
- Saber aplicar la función `fmincon` de minimización con restricciones.

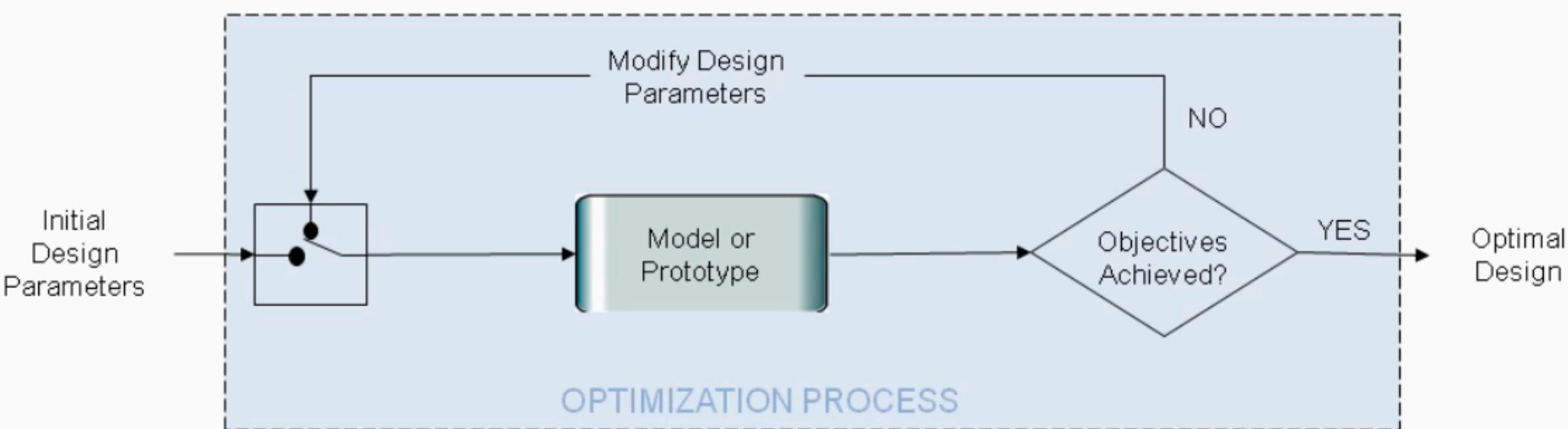
Bibliografía

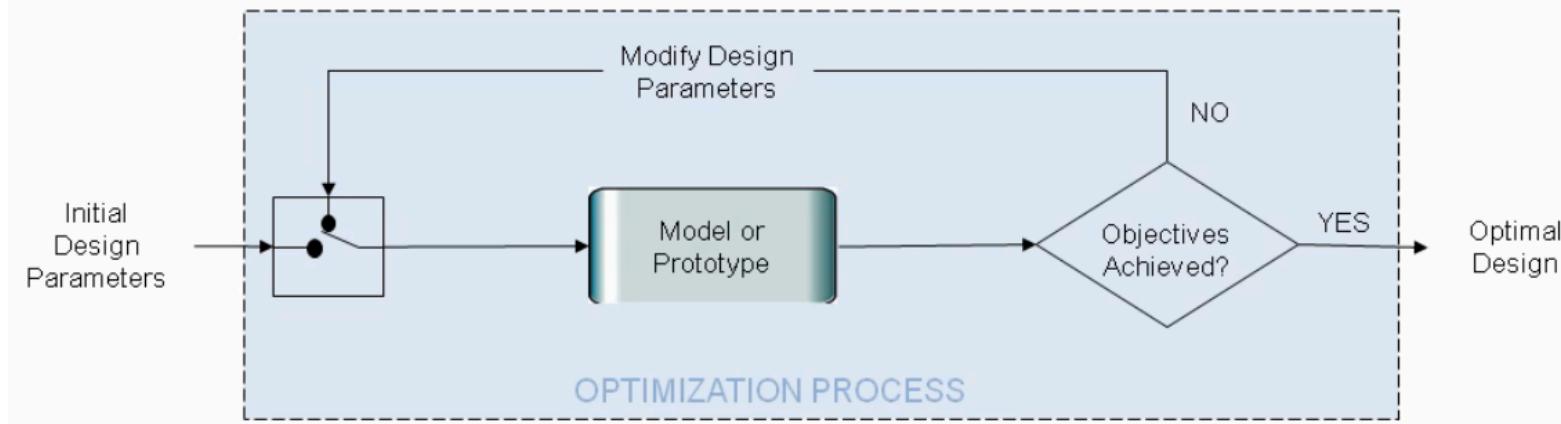
- J. Kiusalaas: *Numerical Methods in Engineering with MATLAB*, 2/E. Cambridge University Press, 2010.
- S.C. Chapra: *Applied Numerical Methods with MATLAB for Engineers and Scientists*, 2/E. McGraw-Hill, 2006.
- W.Y. Yang *et al.*: *Applied Numerical Methods Using MATLAB*. Wiley 2005.
- W.H. Press *et al.*: *Numerical Recipes. The Art of Scientific Computing*, 3/E. Cambridge University Press, 2007.



Optimización

- Encontrar la mejor solución a un **problema automáticamente**.
 - Problemas complejos → Compromiso prestaciones-restricciones
 - No es un proceso manual (prueba-error, cambio de parámetros uno a uno)
- **Métodos numéricos de optimización = Técnicas de búsqueda automática**

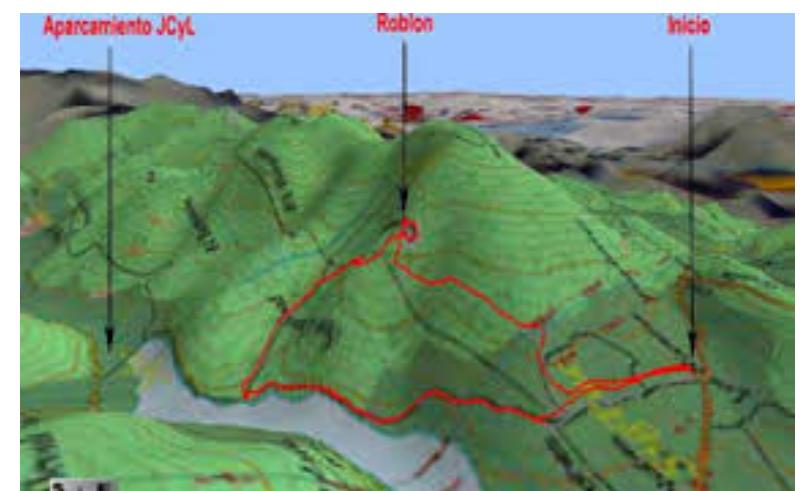
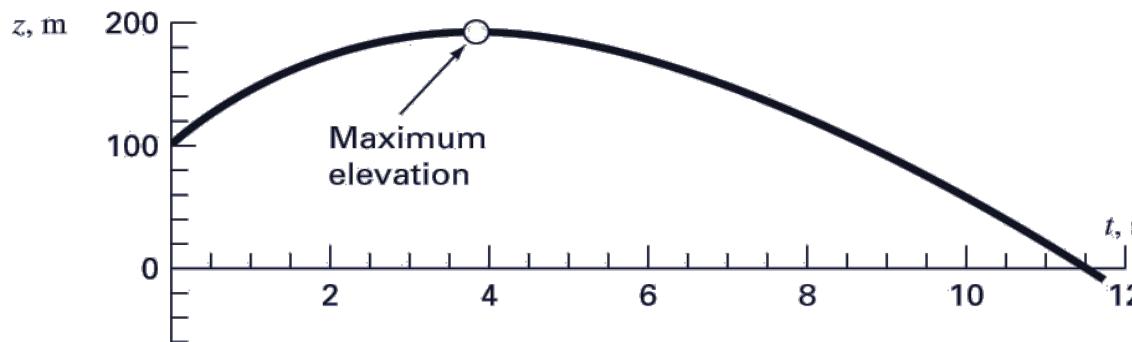


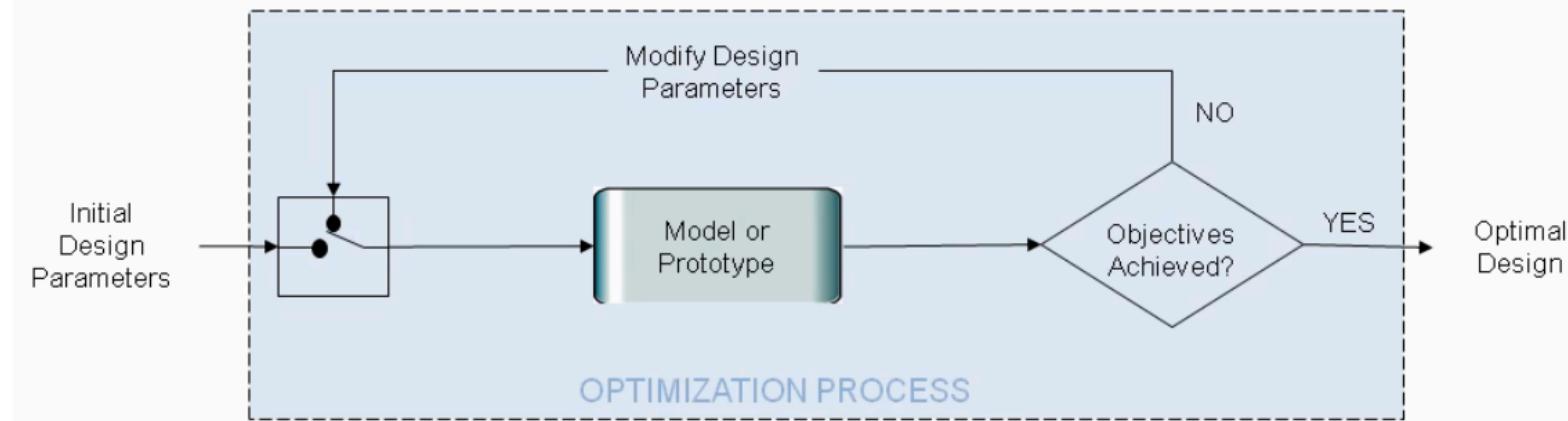


Min $f(x)$
 x
sujeto a **restricciones**

Problema de Optimización

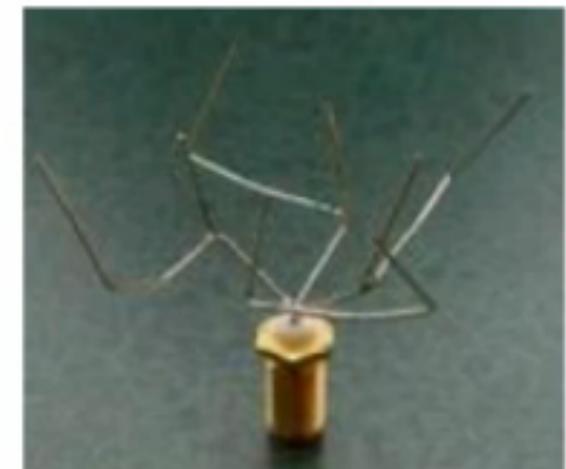
- $f(x)$ = función a minimizar, **función coste**
- x = variables independientes, **parámetros de diseño**





$$\underset{x}{\text{Min } f(x)}$$

sujeto a restricciones

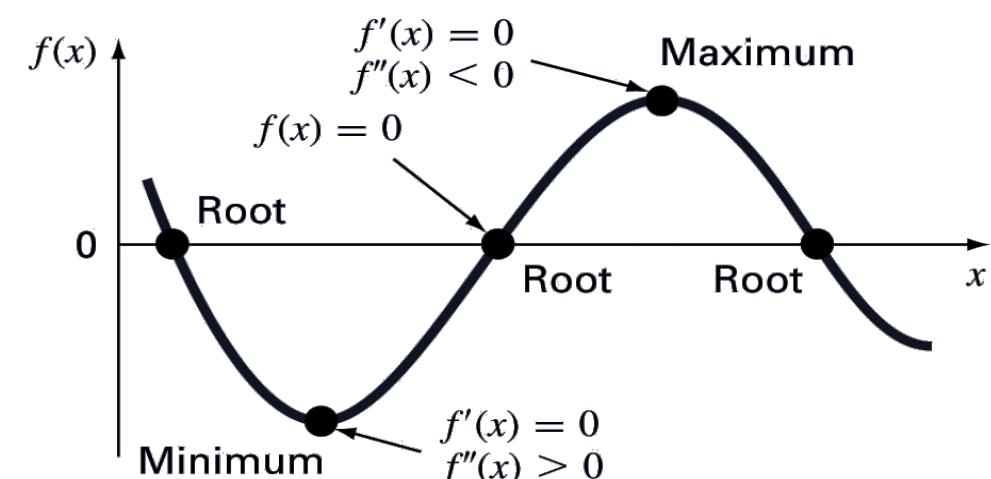
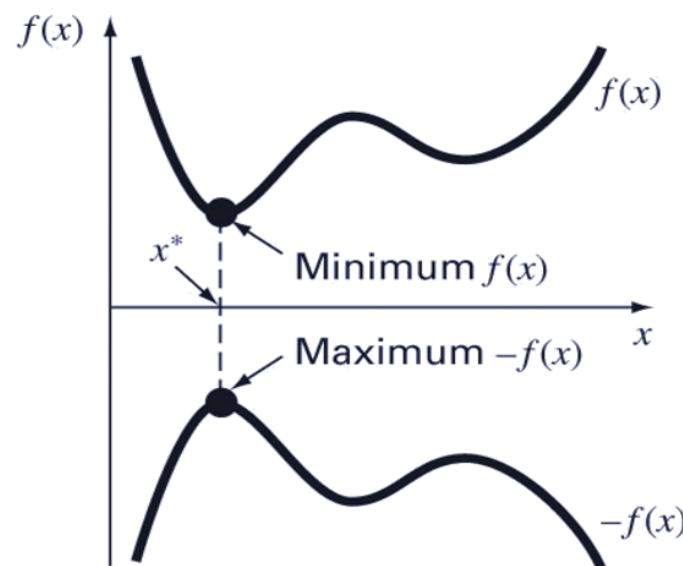


- **Beneficios de la optimización:**
 - Encontrar mejores soluciones
 - Evaluaciones más rápidas
 - Encontrar soluciones no intuitivas

Antenna Design Using Genetic Algorithm
<http://lc.arc.nasa.gov/projects/esg/research/antenna.htm>

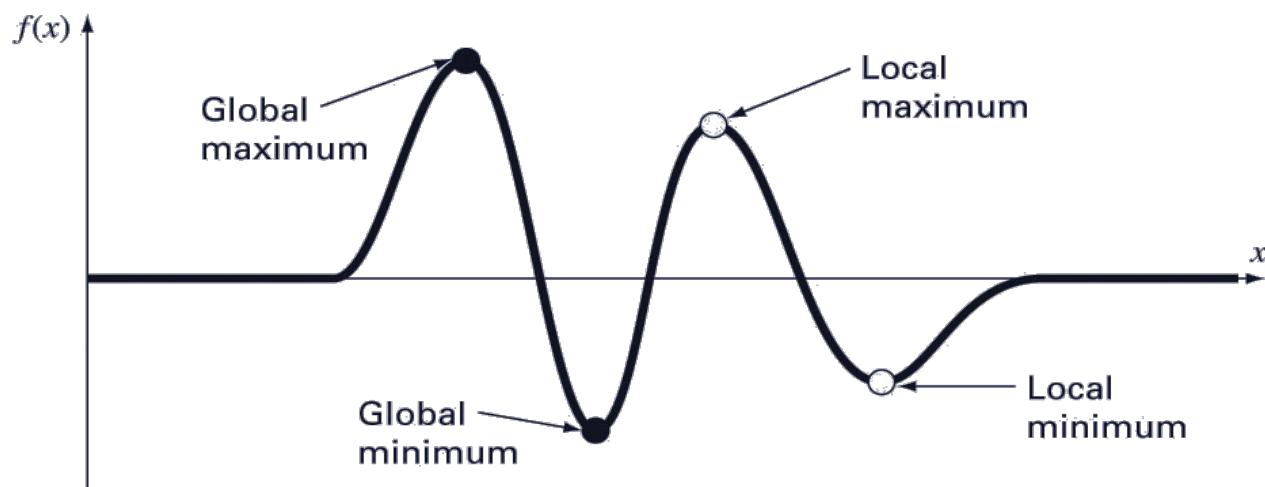
Optimización

- La optimización normalmente se asimila con la **minimización**.
maximización de $f(x)$ \Leftrightarrow minimización de $-f(x)$
- Presenta ciertas similitudes con los métodos de localización de raíces.
Possible estrategia → Diferenciar $f(x)$
Obtener las raíces de $f'(x)=0$

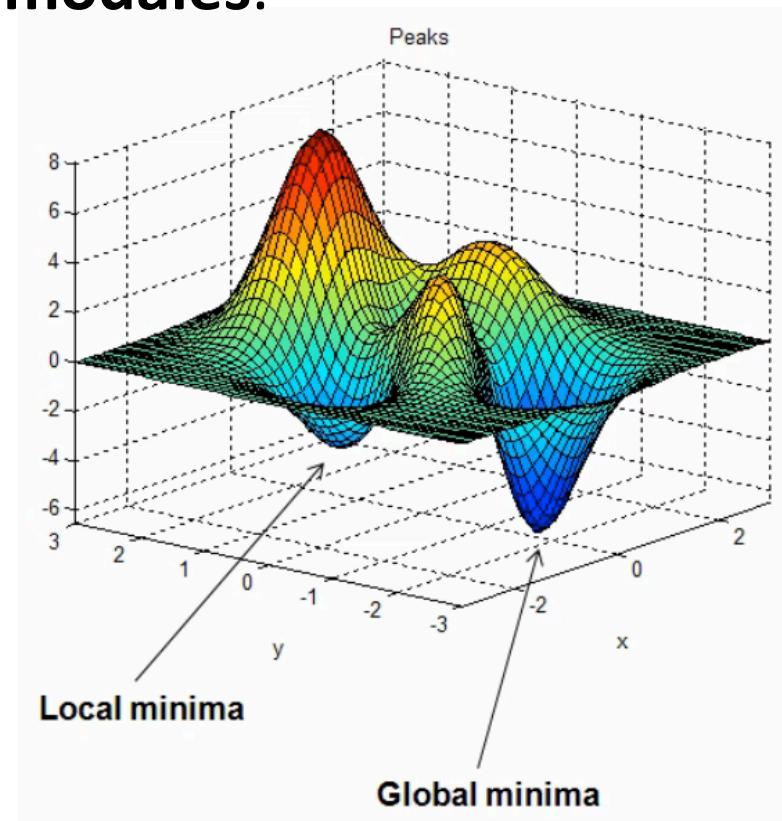


Mínimos Locales vs. Globales

- Un **mínimo global** representa la mejor de las soluciones, mientras que un **mínimo local** es mejor que sus vecinos inmediatos.
- **Intervalos** con varios mínimos locales → **multimodales**.



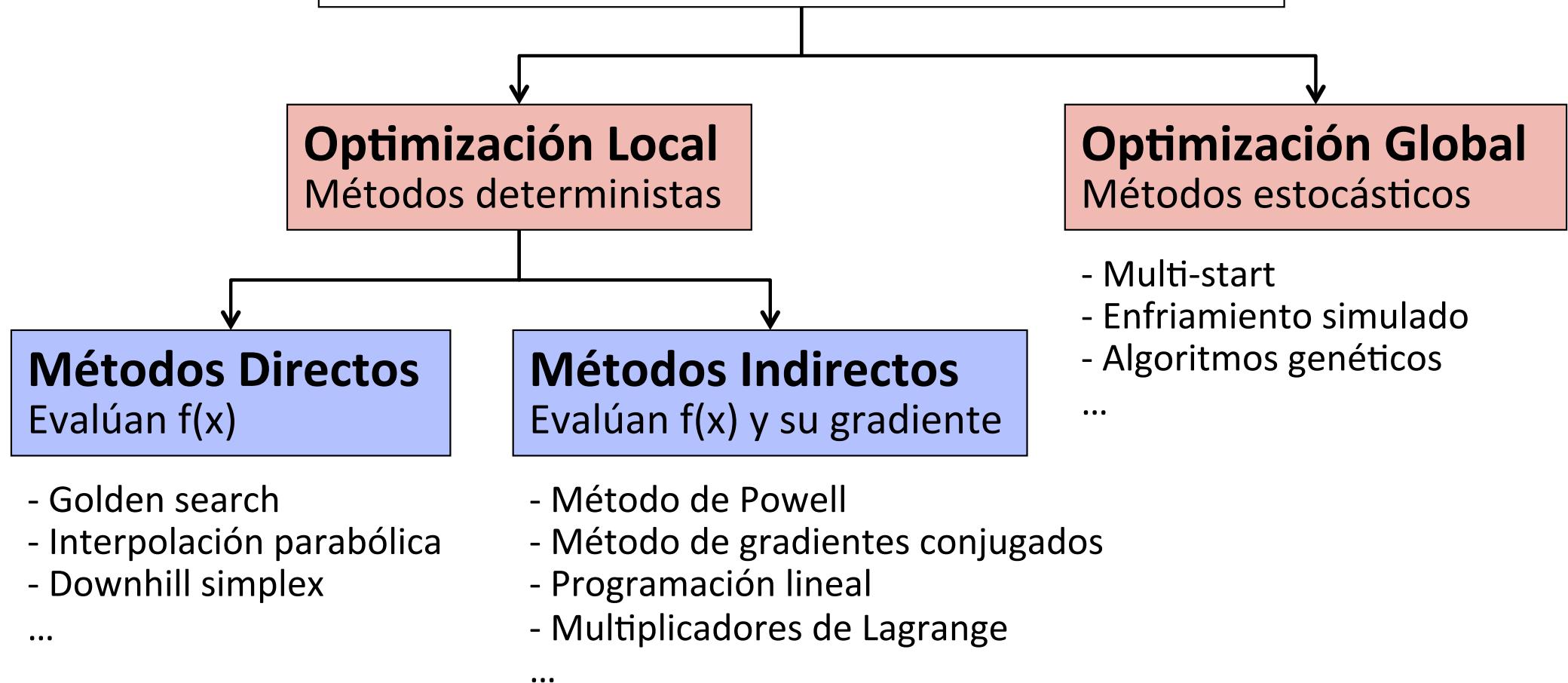
- ¿Cómo encontrar el mínimo global?
- ¿Cómo no confundir un mínimo local con uno global?



Aspectos Básicos

- La mayoría de los **problemas reales** no admiten solución analítica y requieren **soluciones numéricas por ordenador**.
- Los **algoritmos** de optimización son **procesos iterativos** que requieren un **valor inicial x_0** de las variables de diseño.
 - Si $f(x)$ presenta varios mínimos locales, x_0 determina cuál se alcanzará.
 - **No hay manera garantizada de encontrar el mínimo global.**
- Un buen algoritmo debe reducir el **esfuerzo computacional**
→ **número de veces que se evalúa $f(x)$** .
 - Optimización dentro de un método numérico más complejo.
 - Alto tiempo de cómputo para evaluar funciones complejas.

Algoritmos de Optimización



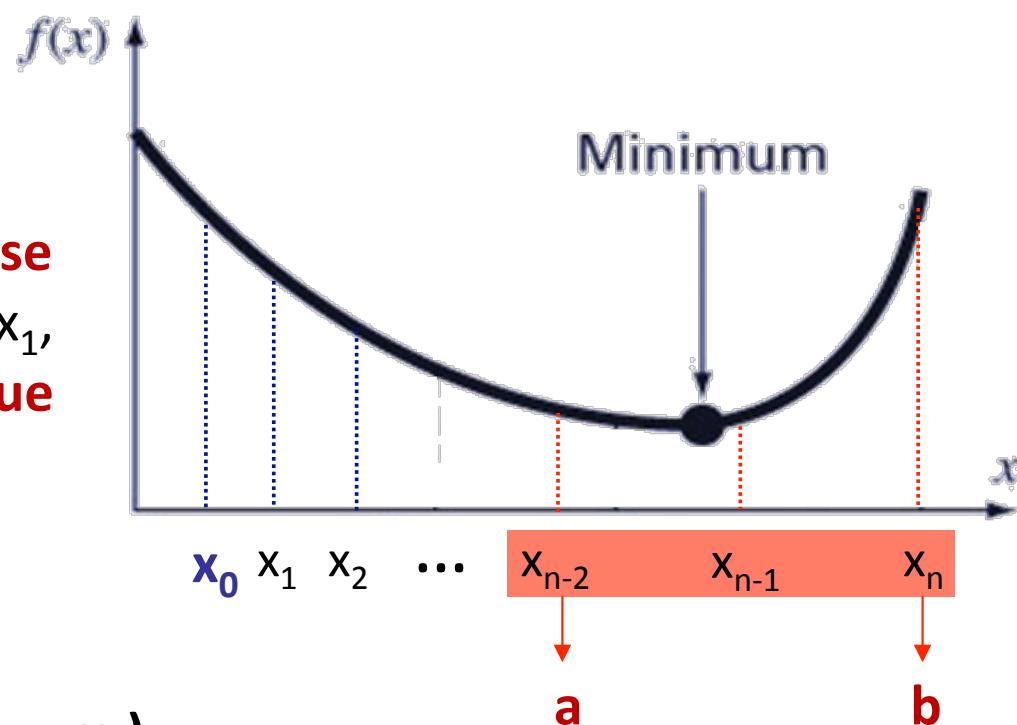
Bracketing

- Antes de aislar un mínimo mediante un algoritmo de optimización hay que acotar su posición $\rightarrow x_{\min} \in (a, b)$

- Procedimiento:

- Comenzar con un valor inicial **x_0 y moverse “cuesta abajo”** evaluando la función en x_1 , x_2 , ..., **hasta** llegar a un punto x_n en el que **f(x) crezca** por primera vez.

$$f(x_{n-1}) < f(x_{n-2}), f(x_{n-1}) < f(x_n)$$



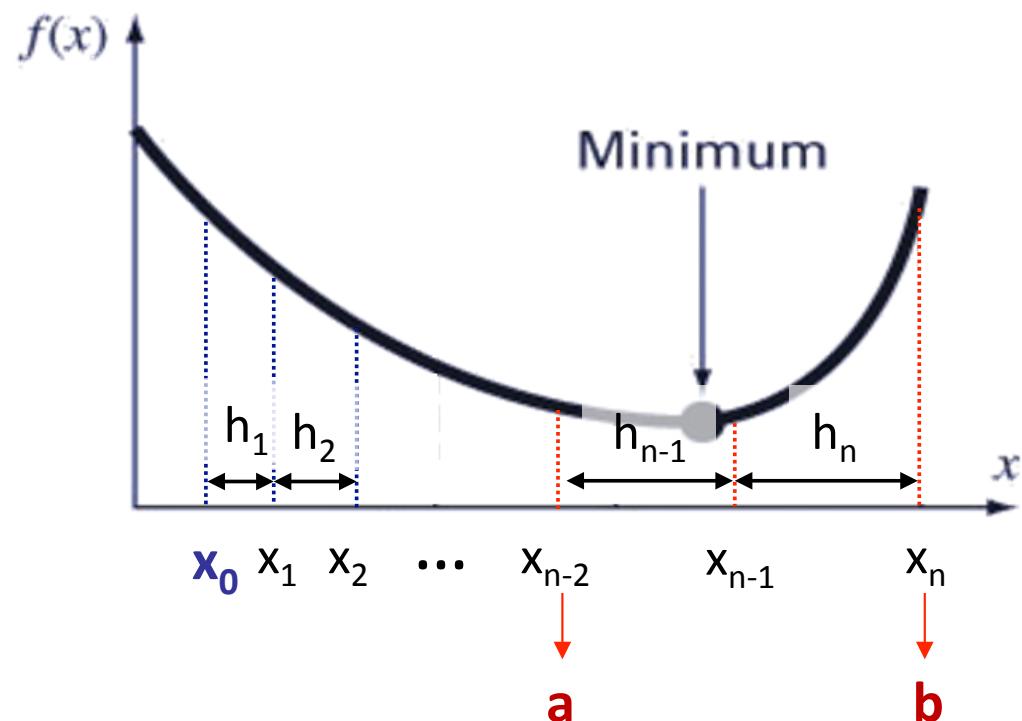
- El **mínimo** estará entonces **acotado en (x_{n-2}, x_n)** .
 - Son **necesarios 3 puntos** para hacer la acotación.

Bracketing

- La acotación **no debe ser precisa, sino eficiente.**

- Tamaño del paso $h_i = x_{i+1} - x_i$
 - Un paso constante suele llevar a demasiadas iteraciones.
 - Aumentando el paso en cada iteración se alcanza el mínimo más rápido
→ “**aprovechar la cuesta abajo**”

$$h_{i+1} = c \cdot h_i, c > 1$$

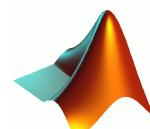




Bracketing

EJEMPLO_01.m

Acotar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$.



```
%> Bracketing
func = @(x) 1/10*x.^2-2*sin(x); % Función coste
x0 = 0.5; % Definimos el valor inicial de x para buscar
h0 = 0.1; % Definimos la zancada inicial (valor por defecto)
[a,b,data_out] = bracket(func,x0,h0); % Aplicamos la función
```



```
function [a,b,data_out] = bracket(func,x0,h)
% Brackets the minimum point of f(x).
% USAGE: [a,b,data_out] = bracket(func,xStart,h)
% INPUT:
%   func = handle of function that returns f(x).
%   x0 = starting value of x.
%   h = initial step size used in search (default = 0.1).
% OUTPUT:
%   a, b = limits on x at the minimum point.
%   data_out = Vector of [x1 x2 x3 f1 f2 f3] values at each
```

bracket.m

- Determina dónde está la “cuesta abajo” para el valor inicial x_0 .
- Se mueve “cuesta abajo” evaluando la función en x_1, x_2, \dots , hasta llegar al punto x_n en el que $f(x)$ crece.

$$f(x_{n-1}) < f(x_{n-2}), f(x_{n-1}) < f(x_n)$$

- El paso usado es $h_{i+1} = c \cdot h_i$, $c=1.61803\dots$

Bracketing

EJEMPLO_01.m

Acotar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$.

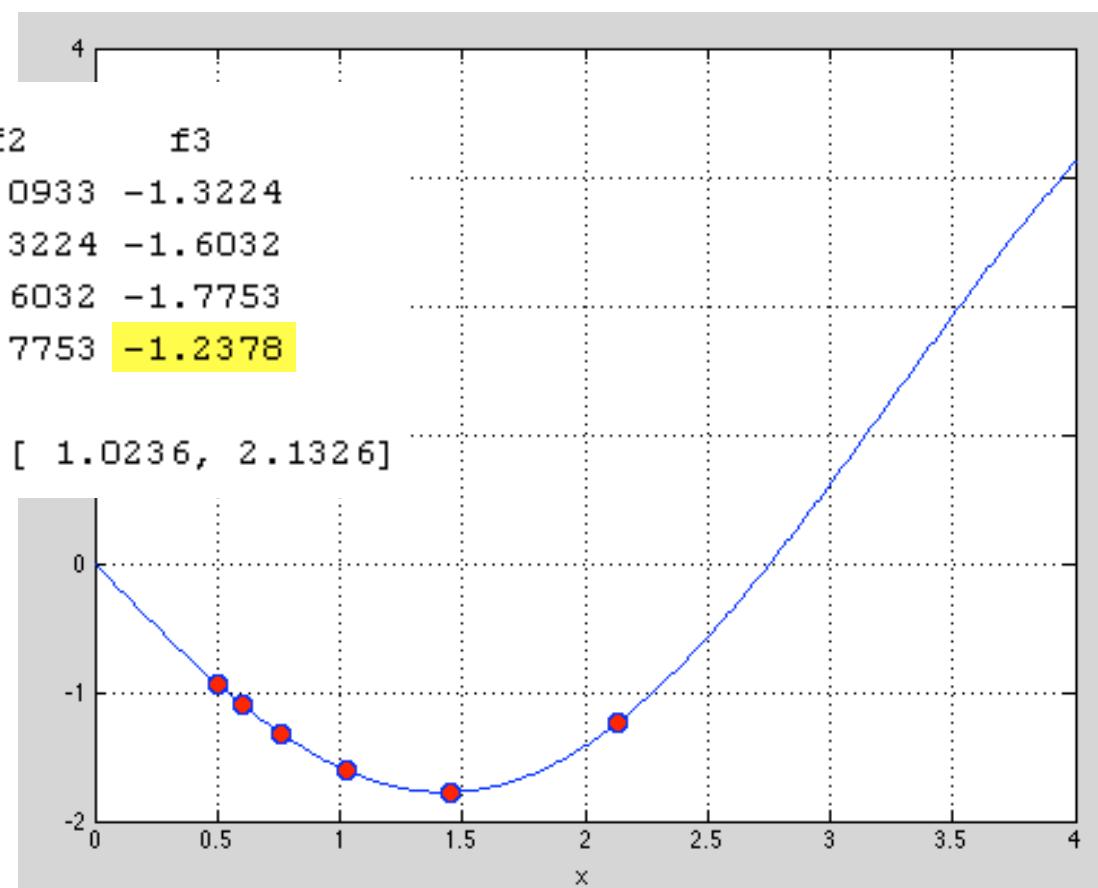
```
>> EJEMPLO_01
```

iter	x1	x2	x3	f1	f2	f3
1	0.5000	0.6000	0.7618	-0.9339	-1.0933	-1.3224
2	0.6000	0.7618	1.0236	-1.0933	-1.3224	-1.6032
3	0.7618	1.0236	1.4472	-1.3224	-1.6032	-1.7753
4	1.0236	1.4472	2.1326	-1.6032	-1.7753	-1.2378

Bracket found for a minimum --> [a,b] = [1.0236, 2.1326]



¿ x_0 distintos?
¿pasos distintos?
¿mismos resultados?

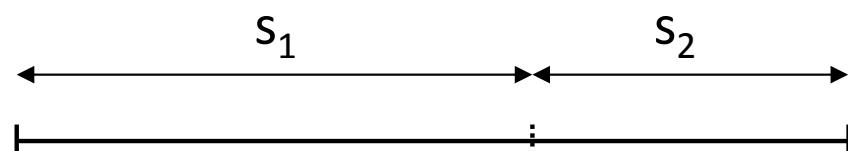


Golden-Section Search

- Algoritmo de acotación de un mínimo en el **intervalo unimodal (a,b)**
- Utiliza la **razón áurea (golden ratio) $\Phi=1.618\dots$** para determinar la posición de **dos puntos x_1 y x_2** internos a (a,b).
- Uno de los puntos internos se re-usa en la siguiente iteración para mejorar la acotación del mínimo.

“Una línea recta se dice dividida en el extremo y su proporcional cuando la línea entera es al segmento mayor como el mayor es al menor.”

Euclides (~300aC)



$$\frac{s_1 + s_2}{s_1} = \frac{s_1}{s_2} \Rightarrow \phi = 1 + \frac{1}{\phi} = \frac{s_1}{s_2} = \frac{1 + \sqrt{5}}{2} = 1.61803\dots$$

Golden-Section Search

$$\begin{aligned} h &= (\Phi-1)(b-a) \\ x_1 &= a+h \\ x_2 &= b-h \end{aligned}$$

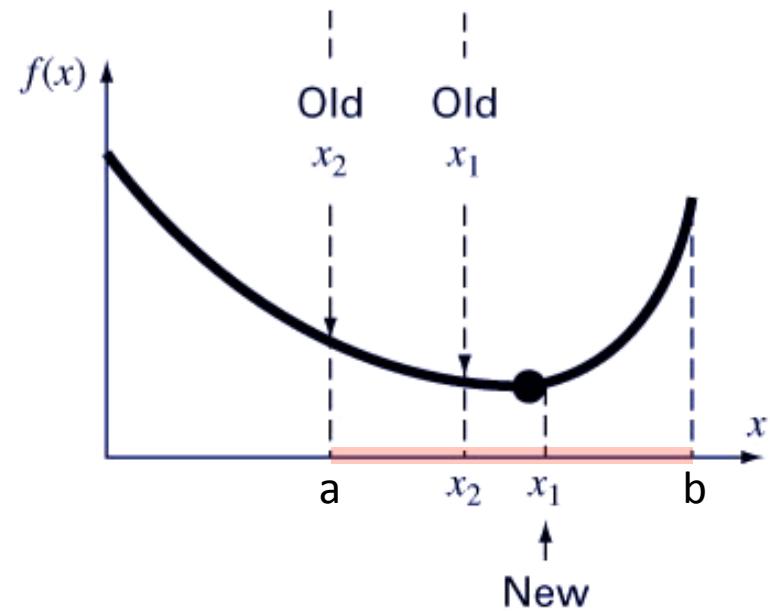
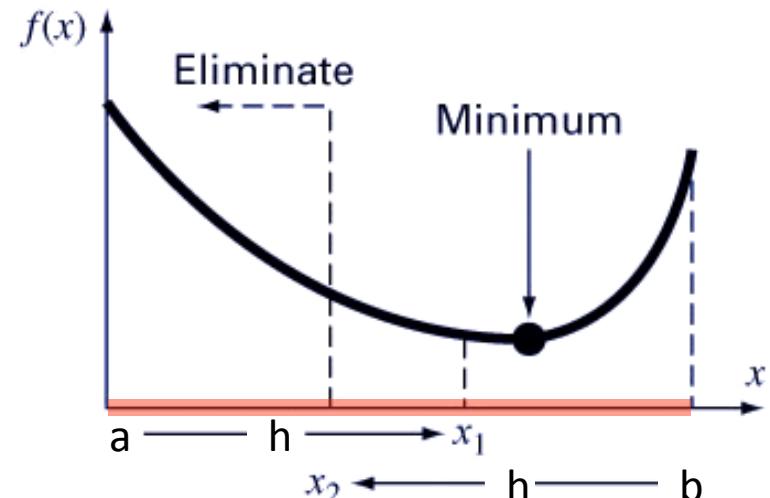
- GS**
- Si $f(x_1) < f(x_2)$:
 $a \leftarrow x_2, x_2 \leftarrow x_1$
 - Si $f(x_1) > f(x_2)$:
 $b \leftarrow x_1, x_1 \leftarrow x_2$

En cada iteración:

- Sólo se necesita un nuevo punto interior y $f(x)$ se evalúa sólo una vez más.
- Se reduce la longitud del intervalo un 38%.

Número de iteraciones n para una **tolerancia ε** dada:

$$|b-a|(\phi-1)^n = \varepsilon \Rightarrow n = -2.0781 \cdot \ln\left(\frac{\varepsilon}{|b-a|}\right)$$

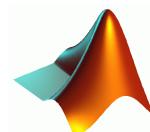




Golden-Section Search

EJEMPLO_02.m

Aislar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$ con una tolerancia de 10^{-2} .



```
%% Golden Search
func = @(x) 1/10*x.^2-2*sin(x); % Función coste
a=0.0; b=4.0; % Intervalo unimodal en el que buscar el mínimo
tol=1e-2; % Tolerancia para localizar el mínimo (valor por defecto)
[xMin,fMin,data_out] = goldSearch(func,a,b,tol);
```



goldSearch.m

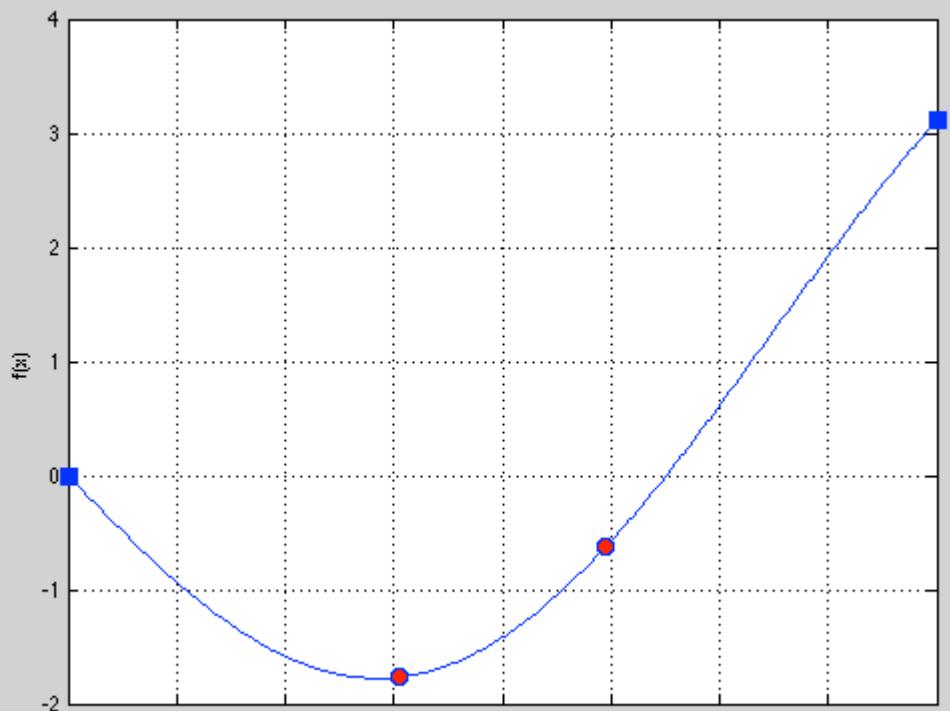
```
function [xMin,fMin,data_out] = goldSearch(func,a,b,tol)
% Golden section search for the minimum of f(x).
% The minimum point must be bracketed in a <= x <= b.
% USAGE: [xMin,fMin] = goldSearch(func,a,b,tol)
% INPUT:
% func = handle of function that returns f(x).
% a, b = limits of the interval containing the minimum.
% tol = error tolerance (default is 1.0e-4).
% OUTPUT:
% fMin = minimum value of f(x).
% xMin = value of x at the minimum point.
% data_out = Vector of [a b x1 x2 f1 f2] values at each iteration
```

- Calcula el número de iteraciones necesarias para una tolerancia dada.
- Aplica *GS* (transparencia anterior) ese número de veces.
- Determina el mínimo entre los $f(x_1)$ y $f(x_2)$ finales.



iter	a	b	x1	x2	f1	f2
1	0.0000	4.0000	2.4721	1.5279	-0.6300	-1.7647
2	0.0000	2.4721	1.5279	0.9443	-1.7647	-1.5310

Iter 1



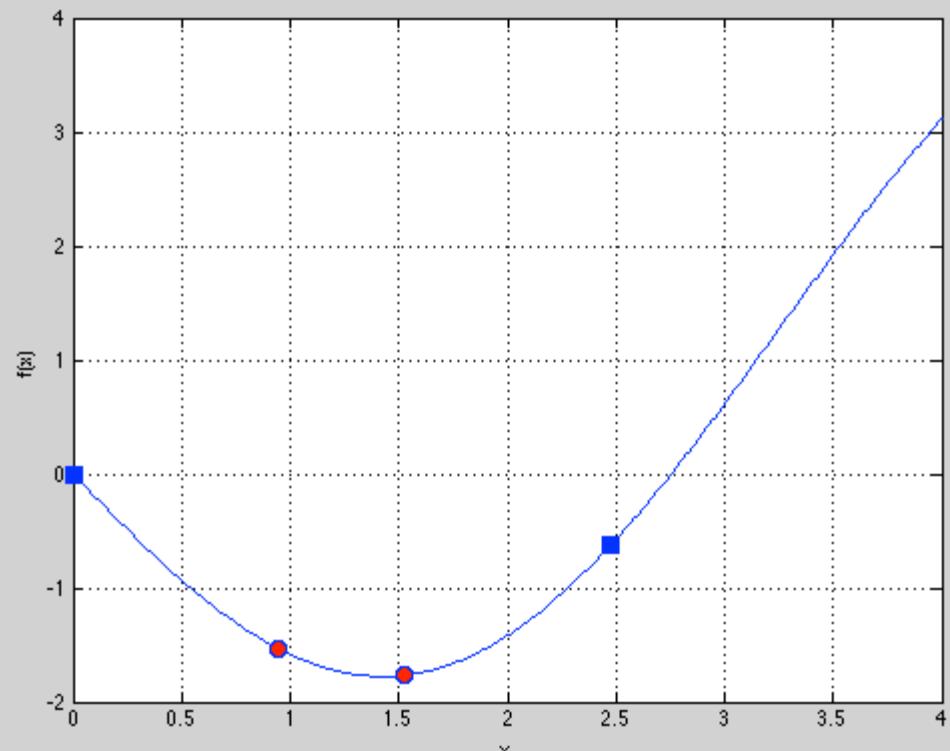
a

x_2

x_1

b

Iter 2



a

x_2

x_1

b

Golden-Section Search

EJEMPLO_02.m

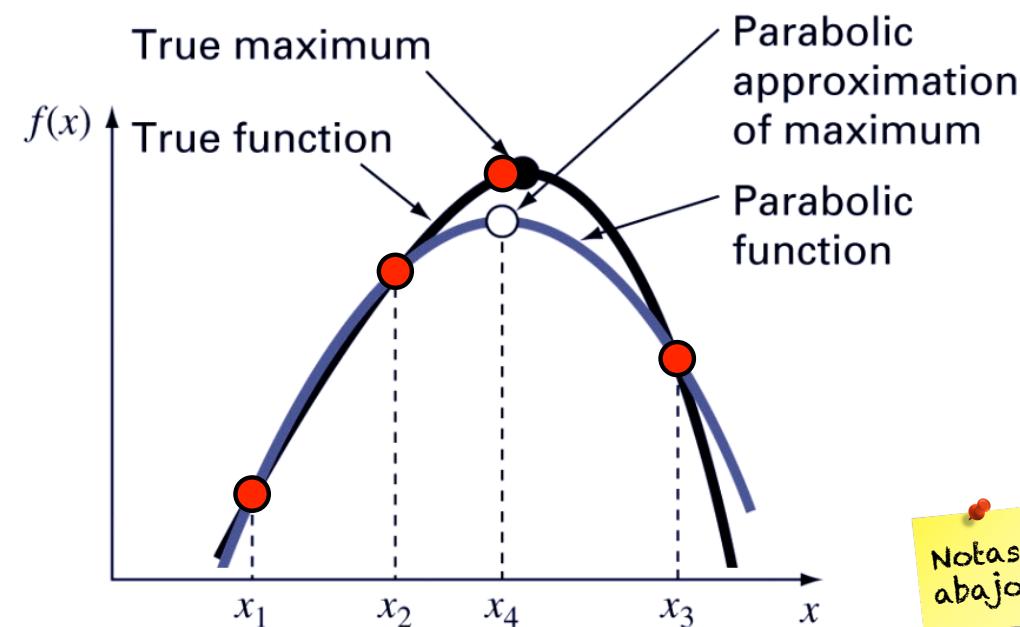
Aislar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$ con una tolerancia de 10^{-2} .

iter	a	b	x1	x2	f1	f2
1	0.0000	4.0000	2.4721	1.5279	-0.6300	-1.7647
2	0.0000	2.4721	1.5279	0.9443	-1.7647	-1.5310
3	0.9443	2.4721	1.8885	1.5279	-1.5432	-1.7647
4	0.9443	1.8885	1.5279	1.3050	-1.7647	-1.7595
5	1.3050	1.8885	1.6656	1.5279	-1.7136	-1.7647
6	1.3050	1.6656	1.5279	1.4427	-1.7647	-1.7755
7	1.3050	1.5279	1.4427	1.3901	-1.7755	-1.7742
8	1.3901	1.5279	1.4752	1.4427	-1.7732	-1.7755
9	1.3901	1.4752	1.4427	1.4226	-1.7755	-1.7757
10	1.3901	1.4427	1.4226	1.4102	-1.7757	-1.7754
11	1.4102	1.4427	1.4303	1.4226	-1.7757	-1.7757
12	1.4226	1.4427	1.4350	1.4303	-1.7757	-1.7757
13	1.4226	1.4350	1.4303	1.4274	-1.7757	-1.7757

xMin = 1.4274, fMin = -1.7757 obtained after 13 iterations

Interpolación Parabólica

- **Ajusta una parábola a 3 pares de puntos** – $[x_i, f(x_i)]$, $i=1,2,3$ – procedentes del bracketing de un óptimo de la función coste.
- Se sabe analíticamente dónde está el **máximo/mínimo de esa parábola**:
$$x_4 = x_2 - \frac{1}{2} \frac{(x_2 - x_1)^2 [f(x_2) - f(x_3)] - (x_2 - x_3)^2 [f(x_2) - f(x_1)]}{(x_2 - x_1)[f(x_2) - f(x_3)] - (x_2 - x_3)[f(x_2) - f(x_1)]}$$
- **Se evalúa $f(x_4)$ y se conserva éste punto y los 2 adyacentes** (1,2 ó 2,3) en la siguiente iteración
 - Siguiente iteración → nueva parábola pasando por (2,4,3) ...

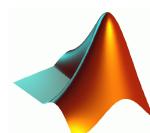




Interpolación Parabólica

EJEMPLO_03.m

Aislar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$ con una tolerancia de 10^{-2} .



```
% Interpolación Parabólica
func = @(x) 1/10*x.^2-2*sin(x);

a=0.0; b=4.0; % Intervalo unimodal en el que buscar el
tol=1e-2; % Tolerancia para localizar el mínimo (valor

[xMin,fMin,data_out] = parabInterp(func,a,b,tol);
```



parabInterp.m

```
function [xMin,fMin,data_out] = parabInterp(func,a,b,tol)
% Parabolic interpolation for the minimum of f(x).
% The minimum point must be bracketed in a <= x <= b.
% USAGE: [xMin,fMin] = parabInterp(func,a,b)
% INPUT:
%   func = handle of function that returns f(x).
%   a, b = limits of the interval containing the minimum.
%   tol = error tolerance (default is 1.0e-4).
% OUTPUT:
%   fMin = minimum value of f(x).
%   xMin = value of x at the minimum point.
%   data_out = Vector of [a b x1 x2 f1 f2] values at each
```

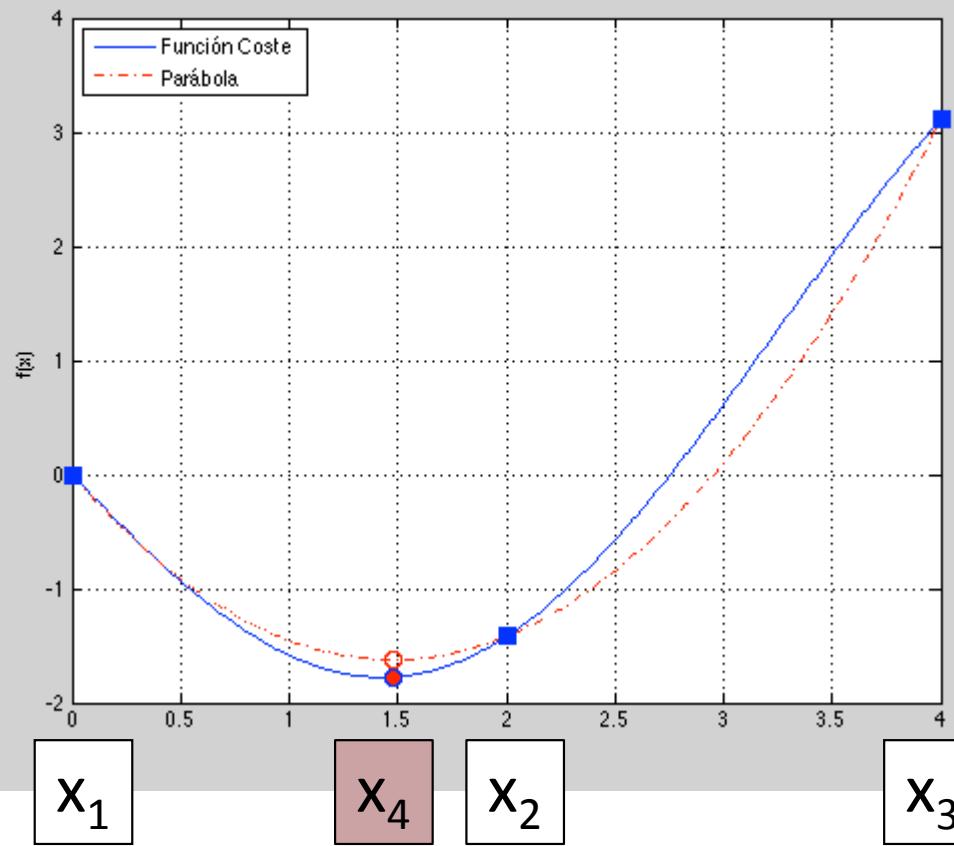
- Determina la parábola que pasa por $x_1, f(x_1); x_2, f(x_2); x_3, f(x_3)$.
- Computa el mínimo de la parábola x_4 y evalúa $f(x_4)$.
- Compara los valores de x_4 y $f(x_4)$ para eliminar una de las coordenadas anteriores.



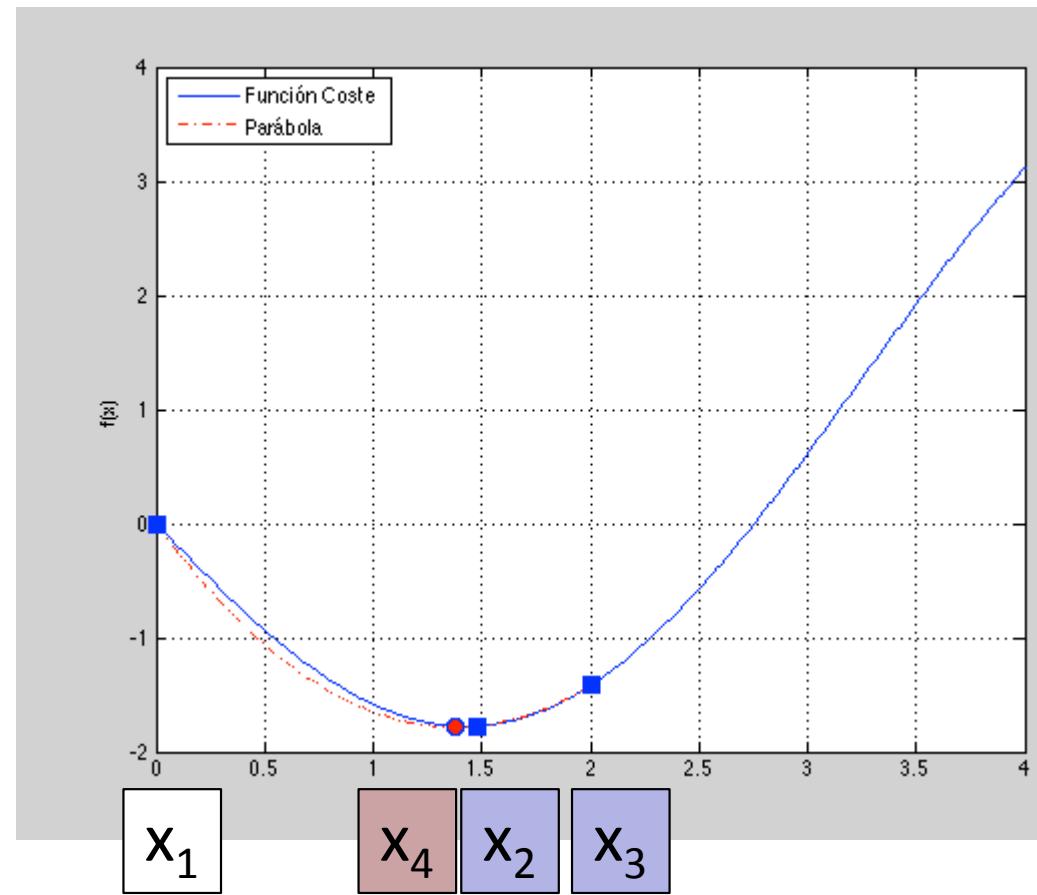
>> EJEMPLO _03

iter	x1	x2	x3	x4	f1	f2	f3	f4
1	0.0000	2.0000	4.0000	1.4768	0.0000	-1.4186	3.1136	-1.7731
2	0.0000	1.4768	2.0000	1.3777	0.0000	-1.7731	-1.4186	-1.7730

Iter 1



Iter 2



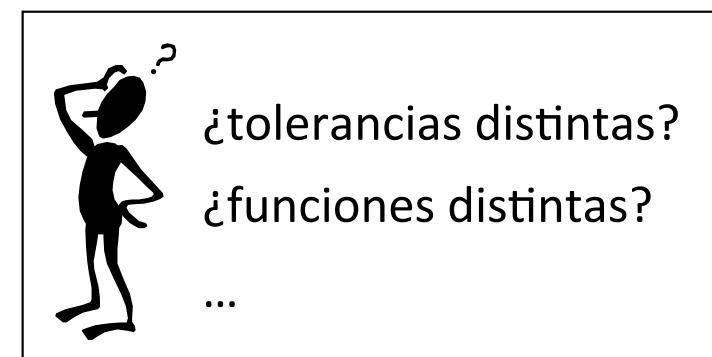
Interpolación Parabólica

EJEMPLO_03.m

Aislar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$ con una tolerancia de 10^{-2} .

iter	x1	x2	x3	x4	f1	f2	f3	f4
1	0.0000	2.0000	4.0000	1.4768	0.0000	-1.4186	3.1136	-1.7731
2	0.0000	1.4768	2.0000	1.3777	0.0000	-1.7731	-1.4186	-1.7730
3	1.3777	1.4768	2.0000	1.4275	-1.7730	-1.7731	-1.4186	-1.7757
4	1.3777	1.4275	1.4768	1.4275	-1.7730	-1.7757	-1.7731	-1.7757

xMin = 1.4275, fMin = -1.7757 obtained after 4 iterations



Función fminbnd

- Función propia de MATLAB que utiliza el **método de Brent** para minimización en 1D dentro de un intervalo.
- Combina **golden search** (robusto, lento) e **interpolación parabólica** (poco robusto, rápido).

Min $f(x)$
 x
para $a < x < b$

```
[xmin, fval] = fminbnd(fun, a, b, options)
```

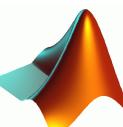
- **optimset** sirve para controlar las opciones de la optimización (Display, TolX, TolFun, MaxFunEvals, MaxIter, ...)

```
options = optimset('display', 'iter', 'TolX', 1e-2)
```

Función fminbnd

EJEMPLO_04.m

Aislar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$ con una tolerancia de 10^{-2} .



```
func = @(x) 1/10*x.^2-2*sin(x);
a=0.0; b=4.0; % Intervalo unimodal en el que buscar el mínimo

options = optimset('display','iter','TolX',1e-2);
% Define las opciones para la optimización: mostrar información de las
% iteraciones, fijar la tolerancia, ...

[xMin,fMin] = fminbnd(func,a,b,options)
```

fminbnd



```
>> type fminbnd

function [xf,fval,exitflag,output] = fminbnd(funfcn,ax,bx,options,varargin)
%FMINBND Single-variable bounded nonlinear function minimization.
%
% X = FMINBND(FUN,x1,x2) attempts to find a local minimizer X of the function
% FUN in the interval x1 < X < x2. FUN is a function handle. FUN accepts
% scalar input X and returns a scalar function value F evaluated at X.
%
% X = FMINBND(FUN,x1,x2,OPTIONS) minimizes with the default optimization
% parameters replaced by values in the structure OPTIONS, created with
% the OPTIMSET function. See OPTIMSET for details. FMINBND uses these
% options: Display, TolX, MaxFunEval, MaxIter, FunValCheck, PlotFcns,
% and OutputFcn.
```

Función fminbnd

EJEMPLO_04.m

Aislar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $0 \leq x \leq 4$ con una tolerancia de 10^{-2} .

Func-count	x	f(x)	Procedure
1	1.52786	-1.76472	initial
2	2.47214	-0.629974	golden
3	0.944272	-1.53098	golden
4	1.42704	-1.77573	parabolic
5	1.4237	-1.77571	parabolic
6	1.43037	-1.77572	parabolic

Optimization terminated:
the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-02

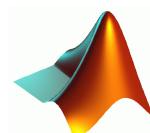
```
xMin =  
  
1.4270  
  
fMin =  
  
-1.7757
```



Ejemplo de Resumen

■ EJEMPLO_05.m

Aislar el mínimo de $f(x) = 0.1x^2 - 2\sin(x)$ en el intervalo $-10 \leq x \leq 10$.



```
% EJEMPLO 05: MÉTODOS DE MINIMIZACIÓN UNI-DIMENSIONALES
% TODOS LOS MÉTODOS SON UNI-MODALES
% Objetivo: Aislar el mínimo global de f(x)=1/10*x^2-2*sin(x)
% en el intervalo x=[-10,10]
clear; clc;

%% Input problem
func = @(x) 1/10*x.^2-2*sin(x); % Defines the function

x0=0.5; % Initial value to start bracketing of a uni-modal interval
h0=0.1; % Initial step of the bracketing

tol=1e-4; % Tolerance for the minimum isolation

%% Bracketing
fprintf('\n *** BRACKETING *** ')
[a,b] = bracket(func,x0,h0);

%% Golden-section search
fprintf('\n *** GOLDEN-SECTION SEARCH *** ')
[xMin_gs,fMin_gs] = goldSearch(func,a,b,tol);

%% Parabolic interpolation
fprintf('\n *** PARABOLIC INTERPOLATION *** ')
[xMin_pi,fMin_pi] = parabInterp(func,a,b,tol);

%% MATLAB built-in function fminbnd
fprintf('\n *** MATLAB built-in FMINBND *** ')
options = optimset('display','iter','TolX',tol);
[xMin_m,fMin_m] = fminbnd(func,a,b,options)
```

Probar x_0 distintos:
 $x_0 = -10, -6, 1, 4, 8$



- ¿Por qué se obtienen resultados distintos?
- ¿Cuál es el mínimo?
- ¿Cómo encontrarlo?



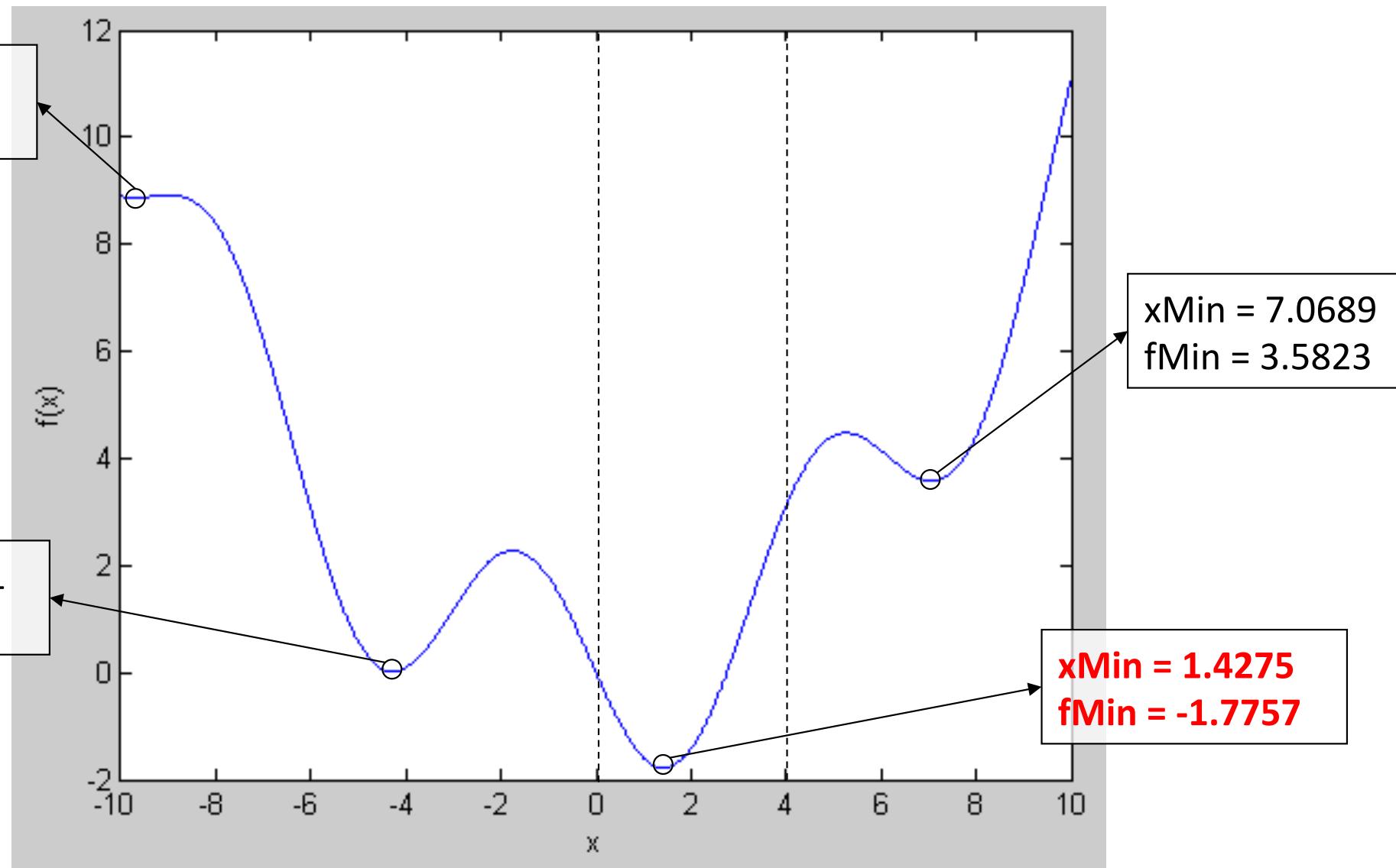
Ejemplo de Resumen

$x_{\text{Min}} = -9.6789$
 $f_{\text{Min}} = 8.8653$

$x_{\text{Min}} = -4.2711$
 $f_{\text{Min}} = 0.0158$

$x_{\text{Min}} = 7.0689$
 $f_{\text{Min}} = 3.5823$

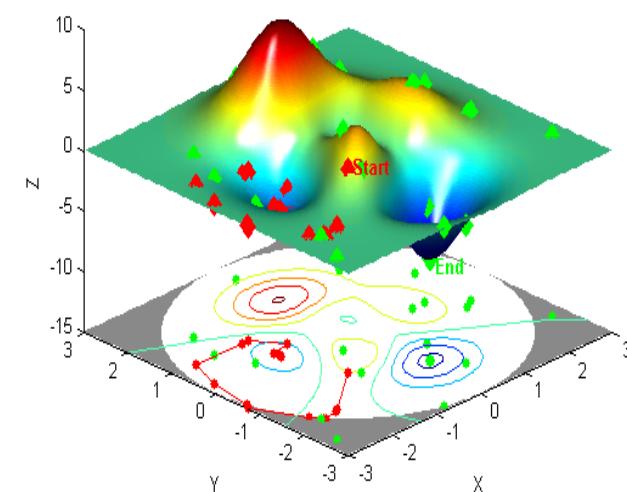
$x_{\text{Min}} = 1.4275$
 $f_{\text{Min}} = -1.7757$



Optimización Multi-Dimensional y Ligaduras

- La optimización multi-D trabaja con funciones de dos o más variables
→ $f(\mathbf{x})$, con $\mathbf{x}=(x_1, x_2, \dots, x_n)$
vector de variables de diseño
- Los problemas reales llevan normalmente la optimización con **restricciones**:

$$\begin{array}{lll} \text{Min } f(\mathbf{x}) & \text{sujeto a} & g_i(\mathbf{x})=0, i=1, 2, \dots, m \\ \mathbf{x} & & h_j(\mathbf{x}) \leq 0, j=1, 2, \dots, n \end{array}$$

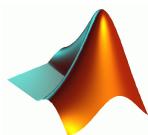


- Los algoritmos de optimización con restricciones suelen ser complejos y a veces difícilmente accesibles.



Gráficas Multi-Dimensionales

- Las funciones de dos variables se pueden visualizar mediante **superficies 3D (`surf`, `mesh`)** y **gráficas de contorno 2D (`contour`)**.

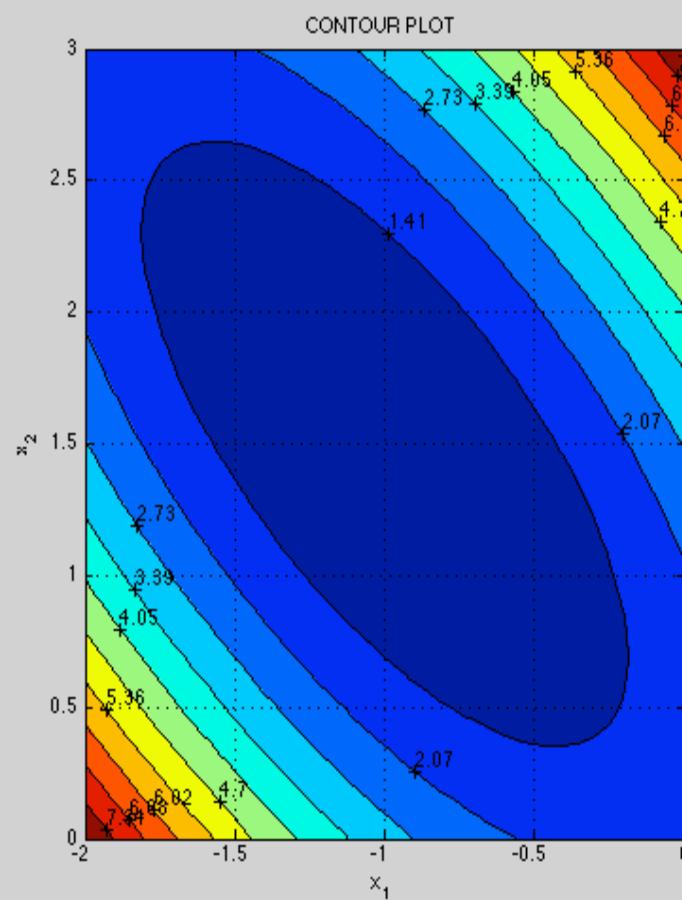
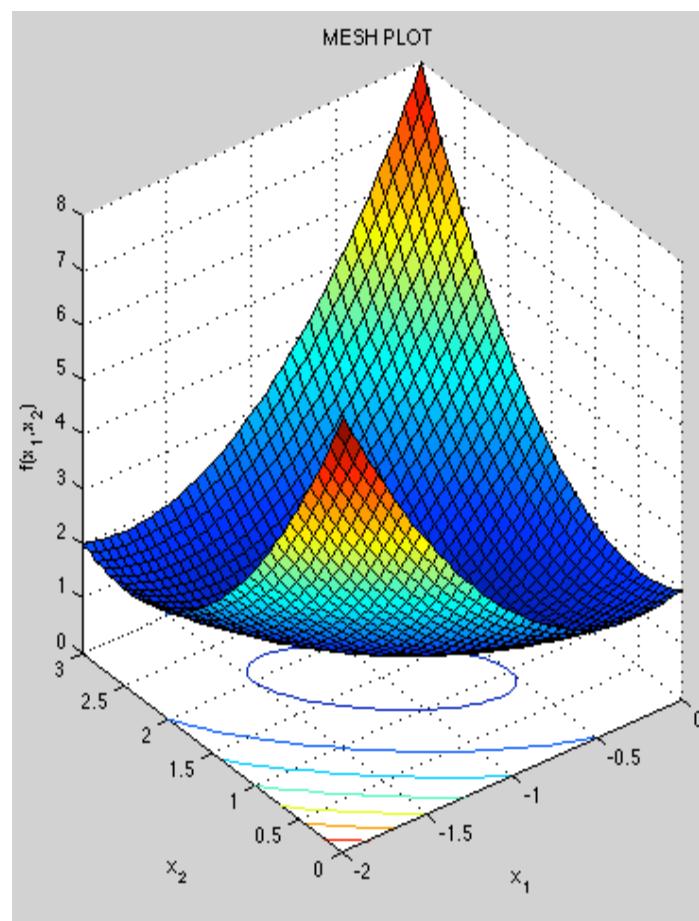


EJEMPLO_06.m

Representación de

$$f(x) = 2+x_1-x_2+2x_1^2+2x_1x_2+x_2^2$$

con $-2 \leq x_1 \leq 0$ y $0 \leq x_2 \leq 3$





Método Downhill Simplex

- Método downhill simplex (simplex “cuesta abajo”) o de **Nelder-Mead**.

n-simplex → Figura de n-dimensiones con $n+1$ vértices conectados mediante líneas rectas y limitada por caras poligonales.

0-simplex → punto

1-simplex → línea

2-simplex → triángulo

3-simplex → tetraedro

- Procedimiento: **Mover un simplex por el espacio de diseño hasta encerrar el mínimo y encoger su tamaño hasta una tolerancia dada.**

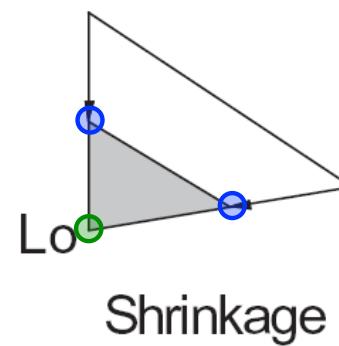
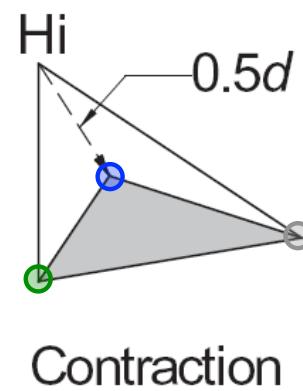
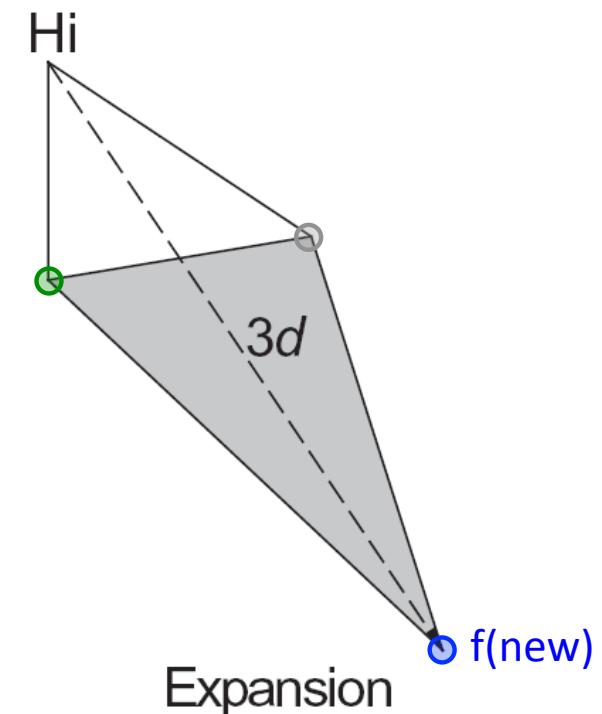
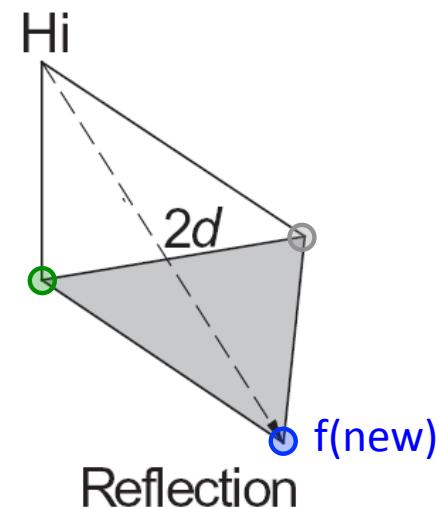
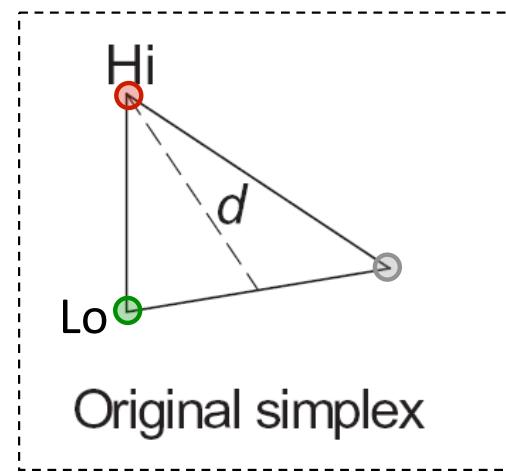
- Se permiten 4 tipos de movimientos del simplex.
- La dirección del movimiento depende del valor de f en los vértices:
 - Hi → vértice con mayor valor de f
 - Lo → vértice con menor valor de f

Notas
abajo



Método Downhill Simplex

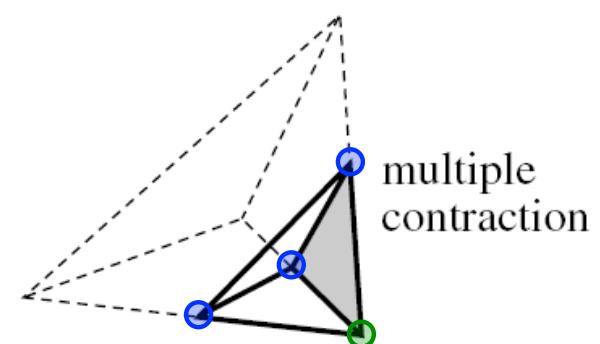
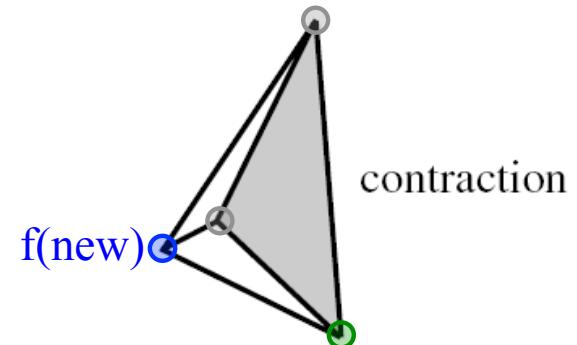
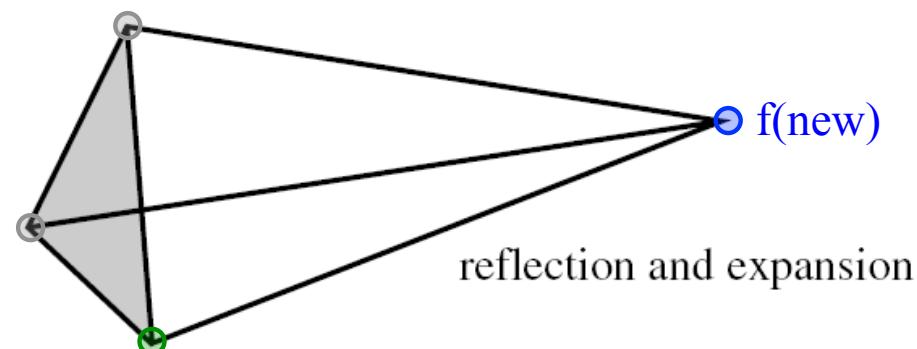
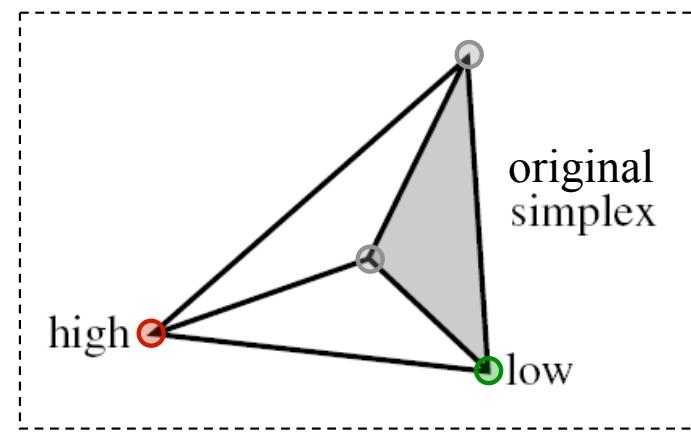
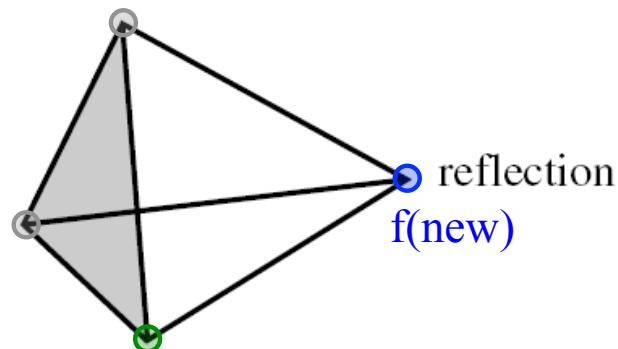
■ Movimientos (2D):





Método Downhill Simplex

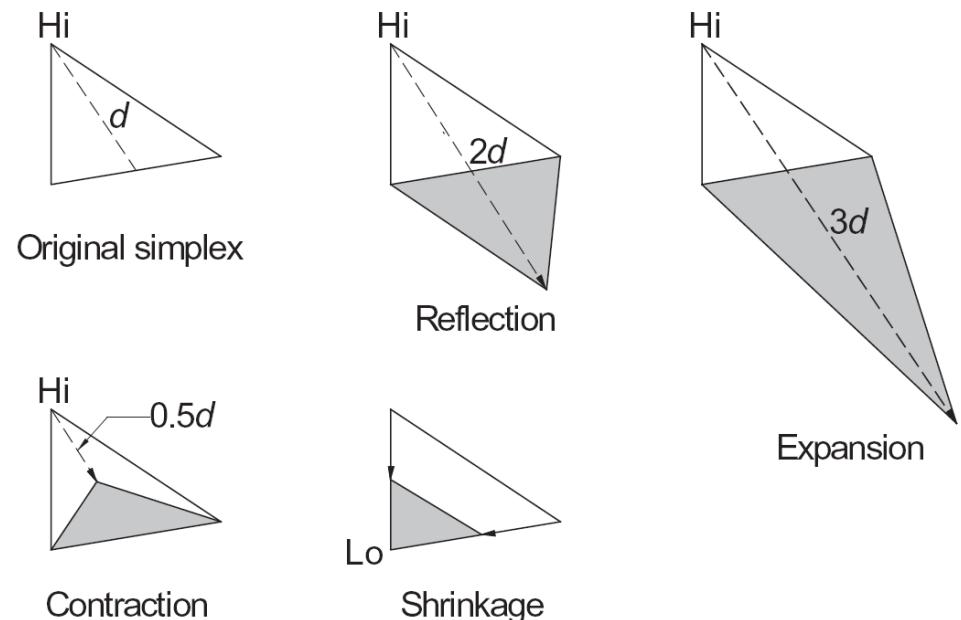
■ Movimientos (3D):



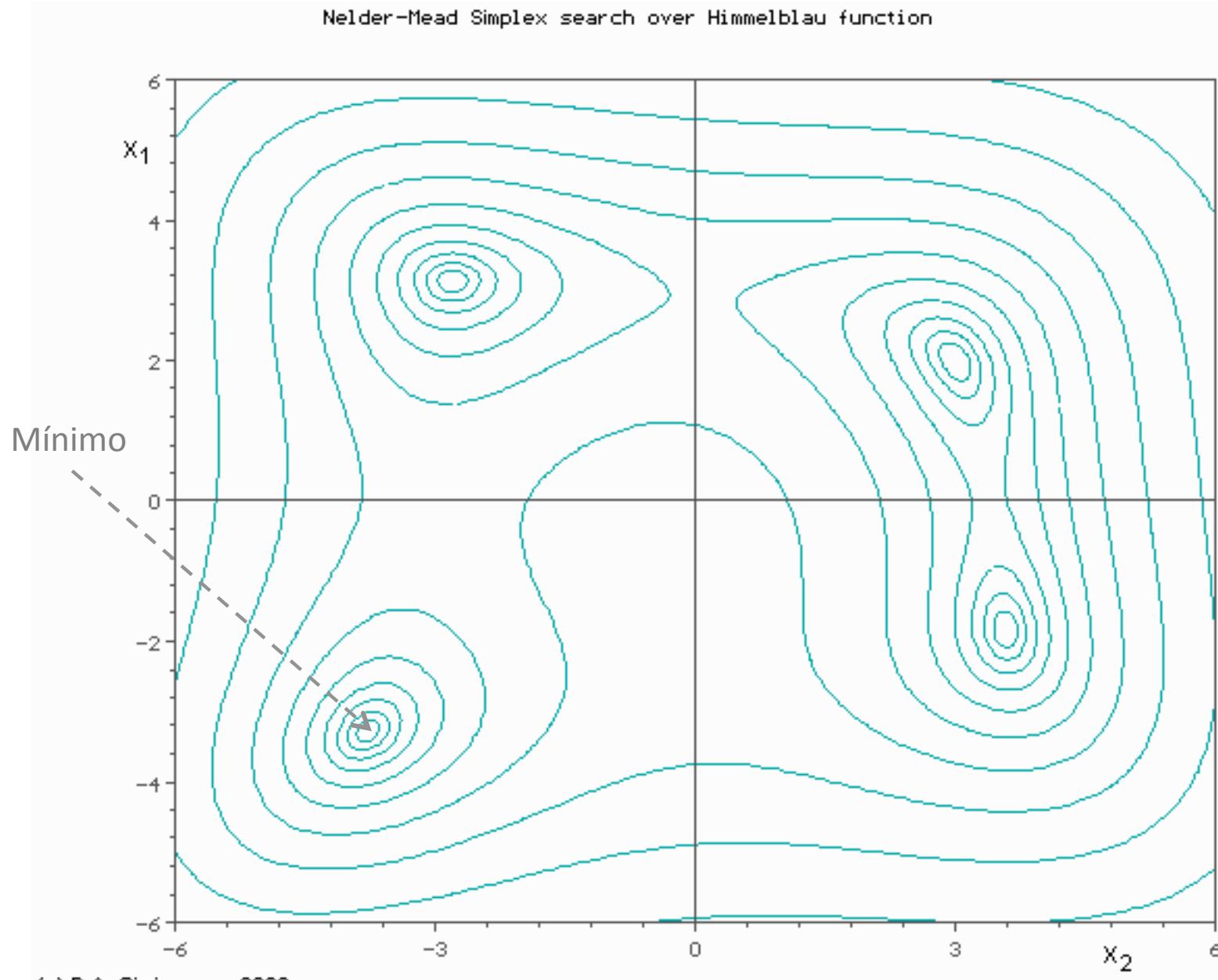
Método Downhill Simplex

Pseudo-código:

- Escoger un **simplex inicial**
- **Iterar hasta $d \leq \epsilon$** (ϵ = tolerancia)
 - **Intentar la reflexión:**
 - * Si $f(\text{new}) < \text{Hi}$, aceptar la reflexión
 - * Si $f(\text{new}) < \text{Lo}$, intentar la **expansión**
 - * Si $f(\text{new}) < \text{Lo}$, aceptar la expansión
 - * Si la reflexión es aceptada, comenzar un nuevo ciclo
 - **Intentar la contracción:**
 - * Si $f(\text{new}) < \text{Hi}$, aceptar la contracción y comenzar un nuevo ciclo
 - **Contracción múltiple** (encoger)
 - Fin de iteraciones



Por lo general, el algoritmo es lento pero robusto.



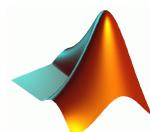
FUENTE: http://es.wikipedia.org/wiki/Archivo:Nelder_Mead2.gif

Notas
abajo

Método Downhill Simplex

EJEMPLO_07.m

$$\text{Minimizar } f(x_1, x_2) = x_1^2 - x_1x_2 - 4x_1 + x_2^2 - x_2$$



```
% MINIMIZACION EN N DIMENSIONES MEDIANTE EL ALGORITMO DE DOWNHILL SIMPLEX
clear; clf; close all;

%% Definición del problema
% Definimos una función vectorizada usando los nombres que deseemos para las
% variables independientes (x1, x2, ...; x, y, ...; a, b, ...)
F1 = @(x1,x2) x1.^2-x1.*x2-4*x1+x2.^2-x2;
% Re-definimos la función en base a un vector (x) de 1xN variables independientes
F2 = @(x) F1(x(1),x(2));

x0 = [0 0]; % vector de condiciones iniciales
b = 0.1; % initial edge length of the simplex (default = 0.1).
tol = 1e-4; % error tolerance (default = 1.0e-4)

[xmin,Fmin,niter] = downhill_simplex(F2,x0,b,tol);
% xMin = vector 1xN, donde cada índice representa el valor de una
% variable independiente que lleva al mínimo de la función
% Fmin = escalar con el mínimo de la función coste
% ncycl = escalar que representa el número de iteraciones realizadas

fprintf('\n\n *** DOWNHILL SIMPLEX *** ')
fprintf('\n x1_min = %.4f, x2_min = %.4f, F_min = %.4f ',xmin)
fprintf('\n obtained after %i iterations \n\n',niter)

%% Comprobación gráfica
x1=linspace(-5,5,30); x2=linspace(-5,5,30);
[x1,x2]=meshgrid(x1,x2);
Z=F1(x1,x2);

subplot(121);
surf(x1,x2,Z);
xlabel('x_1'); ylabel('x_2'); zlabel('f(x_1,x_2)');
title('MESH PLOT');

subplot(122);
cs=contourf(x1,x2,Z); clabel(cs);
xlabel('x_1'); ylabel('x_2');
title('CONTOUR PLOT'); grid;
shg;
```

downhill_simplex.m

```
function [xMin,fMin,nCycl] = downhill_simplex(func,x0,b,tol)
% Downhill simplex method for minimizing f(x1,x2,...,xn).
% USAGE: [xMin,fMin,nCycl] = downhill(func,xStart,b,tol)
% INPUT:
% func = handle of function to be minimized.
% x0 = coordinates of the starting point.
% b = initial edge length of the simplex (default = 0.1).
% tol = error tolerance (default = 1.0e-4).
% OUTPUT:
% xMin = coordinates of the minimum point.
% fMin = minimum value of f
% nCycl = number of cycles to convergence.
```

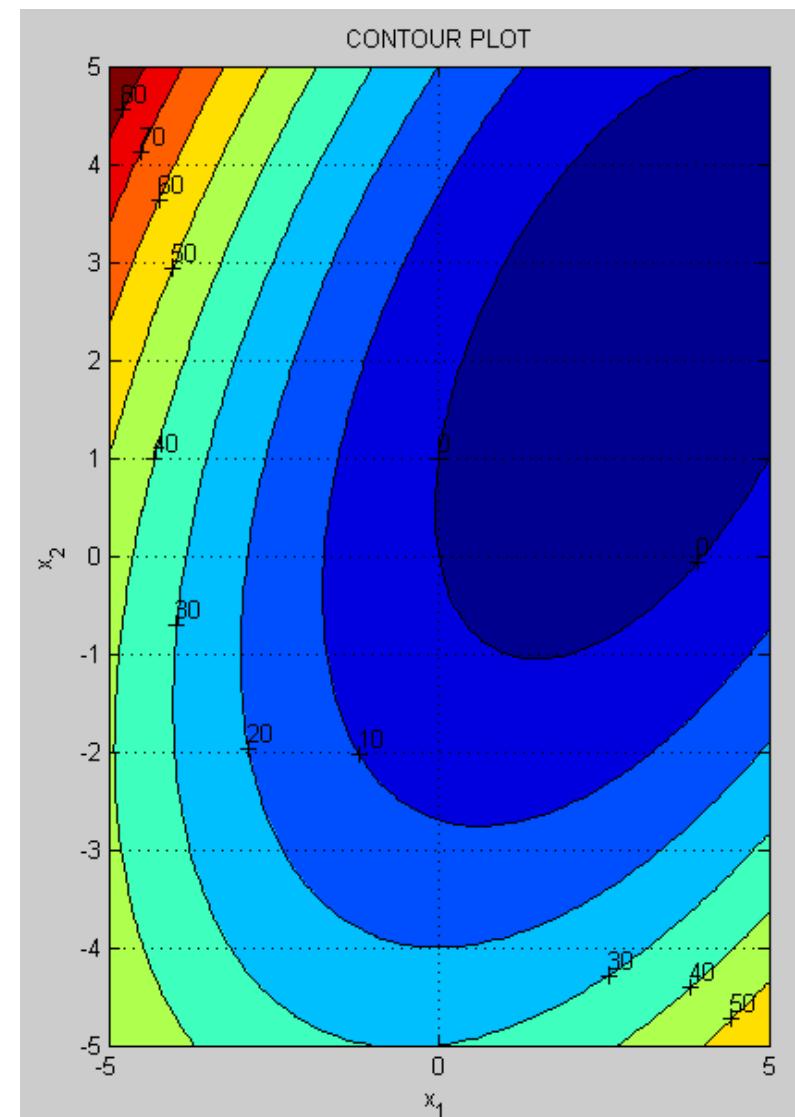
Método Downhill Simplex

EJEMPLO_07.m

$$\text{Minimizar } f(x_1, x_2) = x_1^2 - x_1x_2 - 4x_1 + x_2^2 - x_2$$

iter	fLo	fHi	MOVEMENT
1	-0.3900	-0.7275	reflection & expansion
2	-0.7275	-1.3931	reflection & expansion
3	-1.3931	-2.3527	reflection & expansion
4	-2.3527	-3.5310	reflection & expansion
5	-3.5310	-5.2234	reflection & expansion
6	-5.2234	-5.0631	shrinkage
7	-5.2234	-5.4679	reflection
8	-5.4679	-6.5439	reflection & expansion
9	-6.5439	-5.5263	shrinkage
10	-6.5439	-6.7477	reflection & expansion
11	-6.7477	-6.3502	shrinkage
12	-6.7477	-6.9027	contraction
13	-6.9027	-6.9409	contraction
14	-6.9409	-6.9756	contraction
⋮	⋮	⋮	⋮
44	-7.0000	-7.0000	contraction

```
*** DOWNHILL SIMPLEX ***
x1_min = 3.0000, x2_min = 2.0000, F_min = -7.0000
obtained after 45 cycles
```



Función fminsearch

- Función de MATLAB que utiliza el **método de Nelder-Mead** para minimización multi-D sin restricciones.

Min $f(x)$
 x

```
[xmin, fval] = fminsearch(fun, x0, options)
```

- **Las N variables independientes deben aglutinarse en un vector 1xN.**

```
fun = @(x) x(1).^2+3*x(2)
```

- x_0 y x_{\min} son vectores 1xN.
- **optimset** sirve para controlar las opciones de la optimización (Display, TolX, TolFun, MaxFunEvals, MaxIter, ...)

```
options = optimset('MaxIter', 1e3, 'TolX', 1e-4)
```

Función fminsearch

EJEMPLO_08.m

Minimizar $f(x,y) = (x-0.5)^2 (x+1)^2 + (y+1)^2 (y-1)^2$

```
% MINIMIZACION DE N VARIABLES USANDO
% LA FUNCION PROPIA DE MATLAB fminsearch
clear; clf; close all;

%% Definición del problema
F1 = @(x,y) (x-0.5).^2.* (x+1).^2+(y+1).^2.* (y-1).^2;
F2 = @(x) F1(x(1),x(2));

x0 = [0 0]; % Vector of initial values
% Check different initial values [0 0][-0.5 0.5][0 0.5]

options=optimset('display','iter','TolX',1e-4);
[xMin,FMin]=fminsearch(F2,x0,options)
```

fminsearch

```
%% Comprobación gráfica
npoints=50; % Number of points for mesh grid
nclines=15; % Number of contour lines

x=linspace(-1.5,1.0,npontos); y=
[x,y]=meshgrid(x,y);
Z=F1(x,y);

subplot(121); surf(x,y,Z);
xlabel('x'); ylabel('y'); zlabel('MESH PLOT');

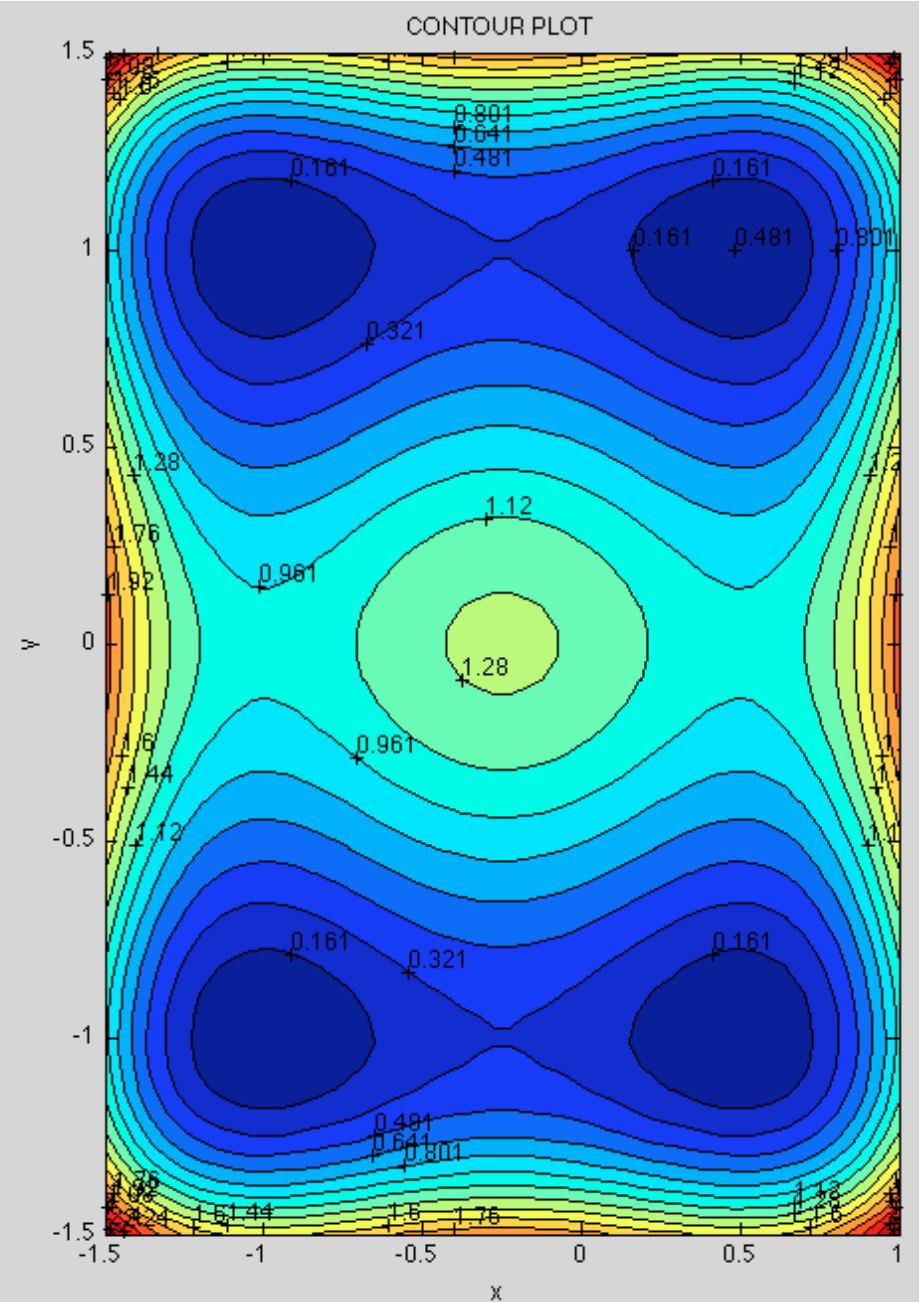
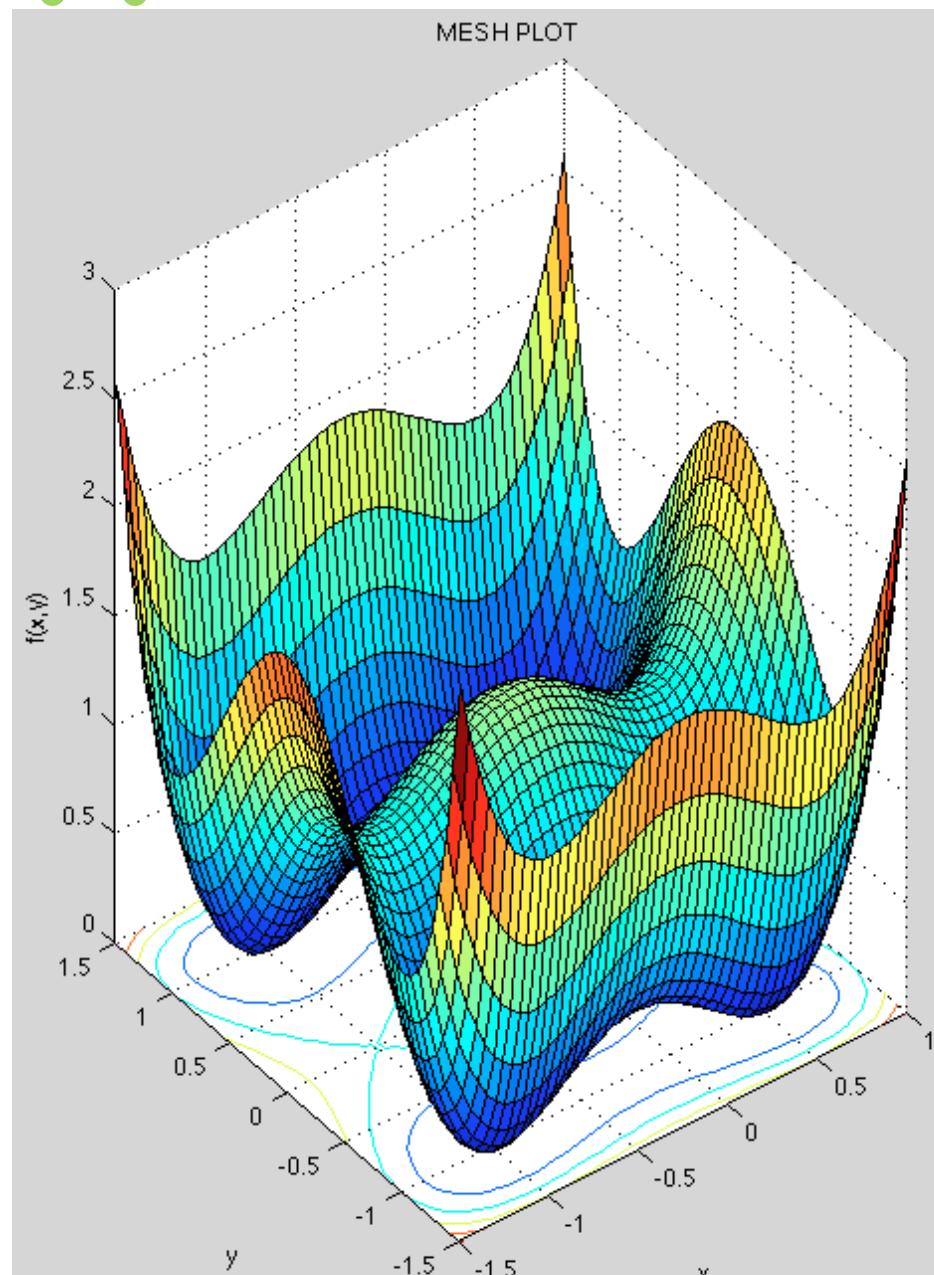
subplot(122); cs=contourf(x,y,Z)
xlabel('x'); ylabel('y');
title('CONTOUR PLOT');
shg;
```

```
>> type fminsearch

function [x,fval,exitflag,output] = fminsearch(funfcn,x,options,varargin)
%FMINSEARCH Multidimensional unconstrained nonlinear minimization (Nelder-Mead).
%
% X = FMINSEARCH(FUN,X0) starts at X0 and attempts to find a local minimizer
% X of the function FUN. FUN is a function handle. FUN accepts input X and
% returns a scalar function value F evaluated at X. X0 can be a scalar, vector
% or matrix.
```

Optimización

Minimización multi-D

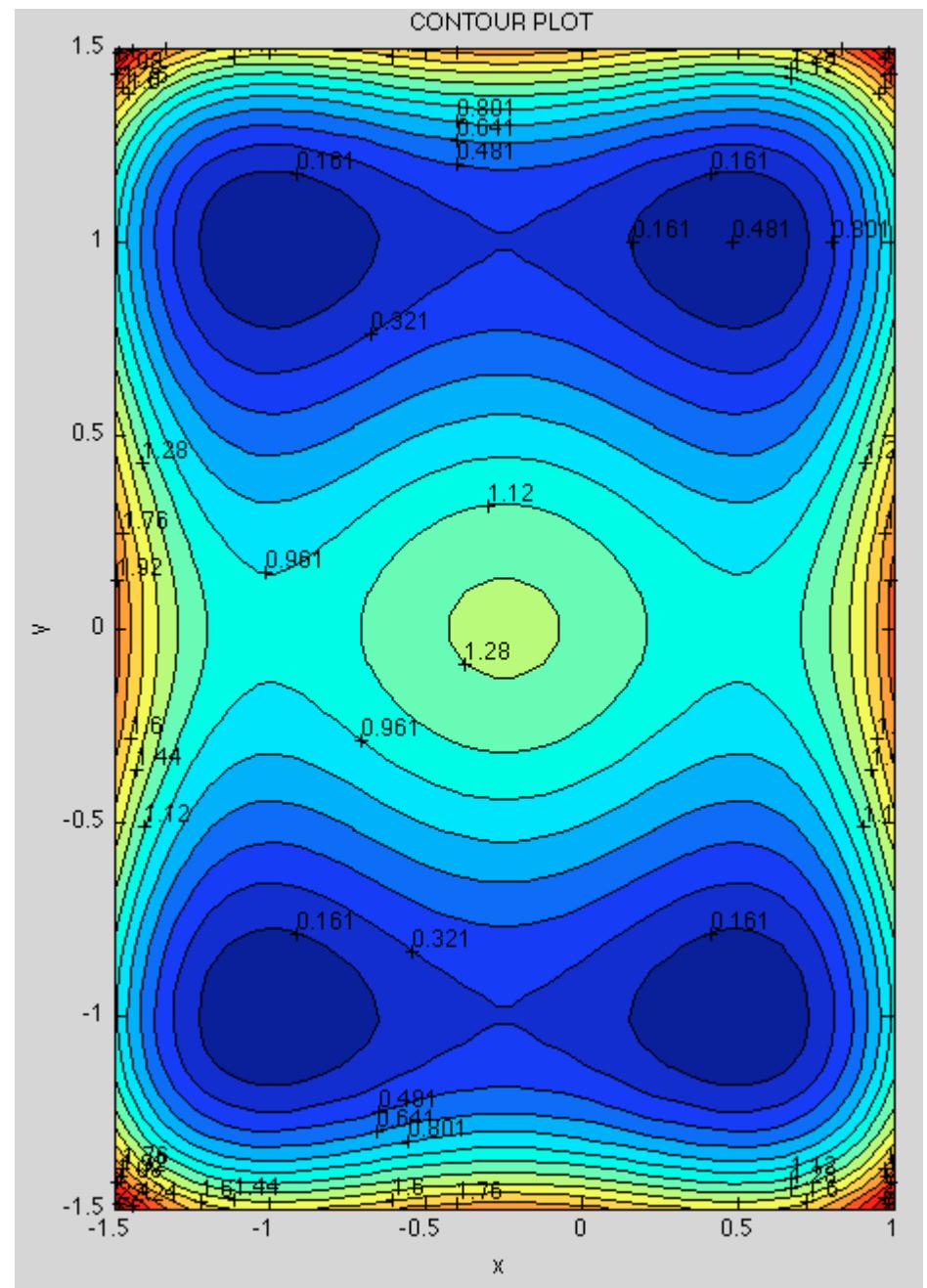


Comprobar distintas condiciones iniciales:

- $x_0 = [0 \ 0]$ → Alcanza un mínimo
 $x_{\min} = [0.50, 1.00]$, $F_{\min} = 0.00$
- $x_0 = [-0.5 \ 0.5]$ → Alcanza otro mínimo
 $x_{\min} = [-1.00, 1.00]$, $F_{\min} = 0.00$
- $x_0 = [0 \ 0.5]$ → Se pierde (max iter)
 $x_{\min} = [0.02, 1.01]$, $F_{\min} = 0.24$



- ¿Cómo saber si el mínimo que encontramos es uno de entre varios mínimos locales?
- ¿Podemos asegurar que el mínimo alcanzado sea global?



Función fmincon

- Función de MATLAB para minimización multi-D con restricciones:

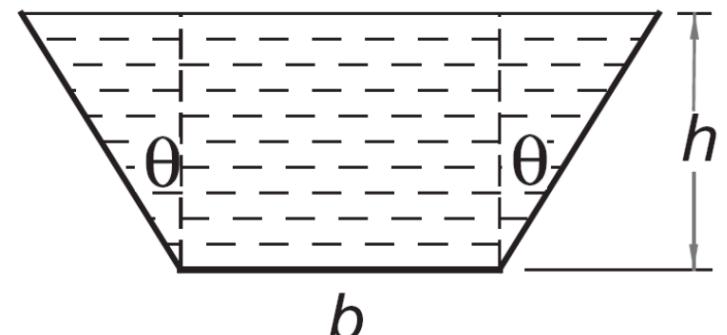
$$\begin{array}{ll} \text{Min } f(\mathbf{x}) & \text{sujeto a} \\ \mathbf{x} & \end{array} \quad \begin{array}{l} A^* \mathbf{x} \leq B, A_{eq}^* \mathbf{x} = B_{eq} \quad (\text{restricciones lineales}) \\ C(\mathbf{x}) \leq 0, C_{eq}(\mathbf{x}) = 0 \quad (\text{restricciones no lineales}) \\ LB \leq \mathbf{x} \leq UB \quad (\text{límites}) \end{array}$$

```
[xmin, fval] = fmincon(fun, x0, A, B, Aeq, Beq, ...
    LB, UB, nonlcon, options)
```

- $x, B, Beq, LB, UB \rightarrow$ vectores $1 \times N$
 - $A, Aeq \rightarrow$ matrices $N \times N$
 - $nonlcon \rightarrow$ función que acepta el vector x y devuelve los vectores C y Ceq

Función fmincon

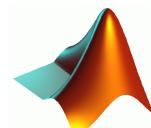
Determine los valores de b , h y θ que minimizan el perímetro mojado de un canal de agua cumpliendo que el área de la sección transversal sea de 8m^2 .



$$\text{Perim} = b + 2 * h * \sec(\theta)$$

$$\text{Area} = \frac{1}{2} * [b + (b + 2 * h * \tan(\theta))] * h = (b + h * \tan(\theta)) * h$$

EJEMPLO_09.m



$$\begin{array}{lll} \text{Min Perim}(x) & \text{sujeto a} & \text{Ceq}(x) = 0 \quad (\text{restricciones no lineales}) \\ x & & \end{array}$$

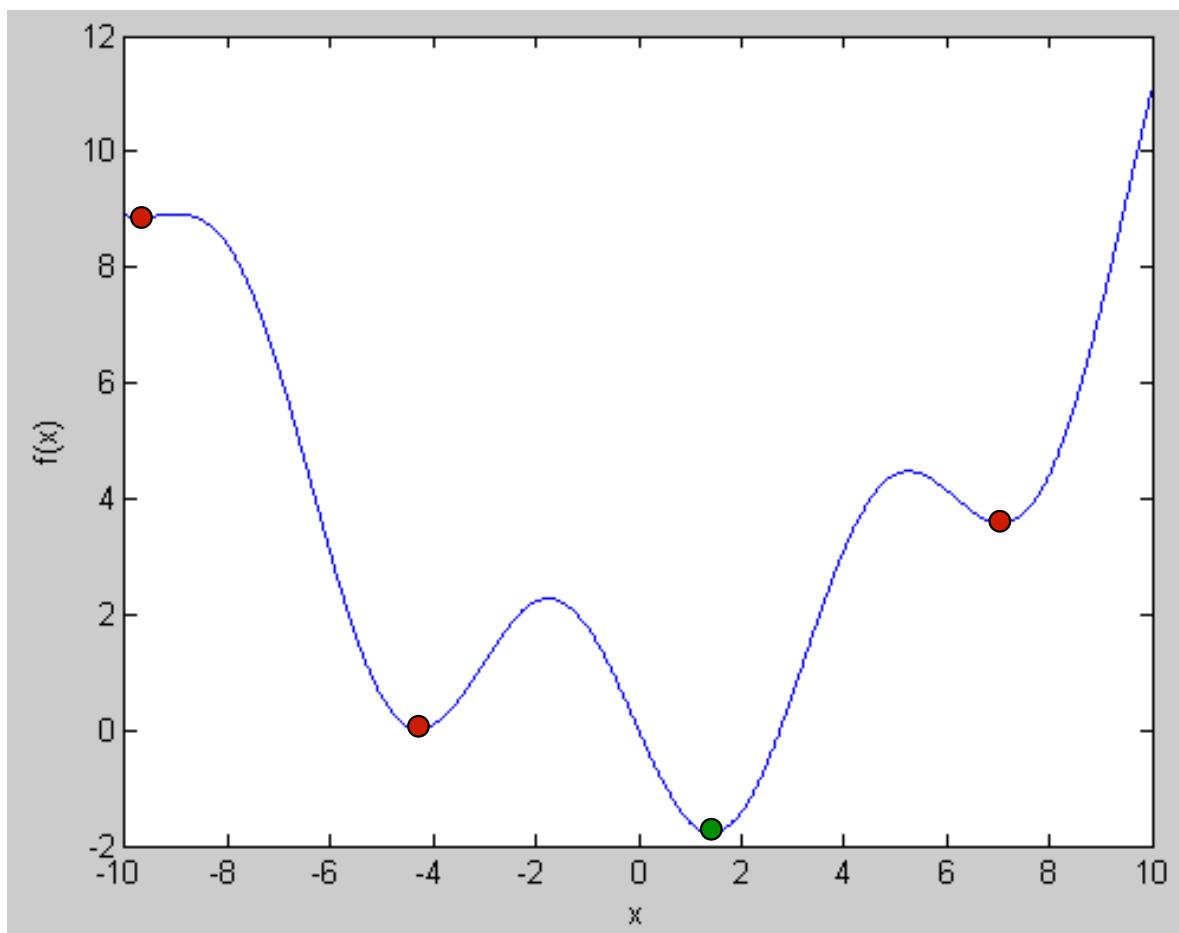
$$\begin{array}{ll} \text{con} & \text{Ceq}(x) = \text{Area}(x) - 8 \\ & x = [b \ h \ \theta] \end{array}$$

Notas
abajo



Optimización Global

- Hasta ahora los métodos vistos son **deterministas** → para las mismas condiciones de aplicación, siempre localización el mismo mínimo



Alternativas al determinismo:

- ¿Y si buscamos de la misma manera (localmente) pero desde muchos sitios distintos?
→ **Multi-start**

- ¿Y si no paramos cuando encontramos el primer mínimo?
- ¿Con qué criterio decidimos por dónde buscar?

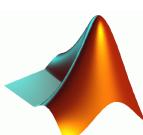
Métodos no-deterministas



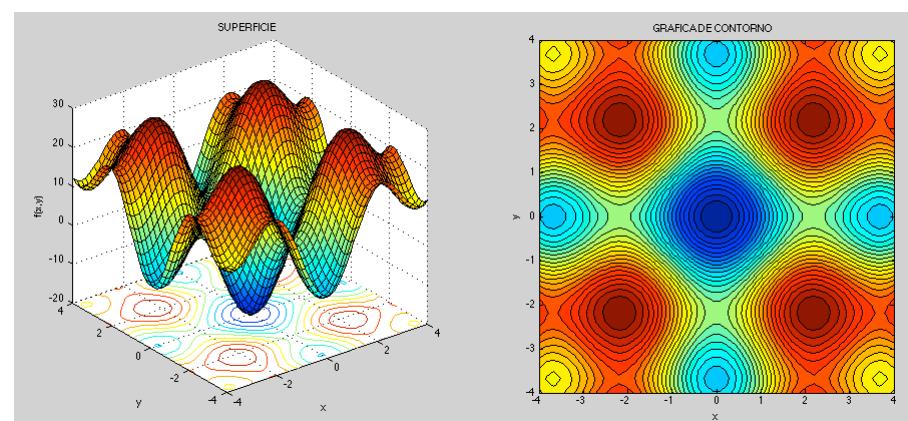
Métodos Multi-Start

- Lanzar un optimizador local un número repetido de veces **desde condiciones iniciales distintas e identificar el menor de los mínimos alcanzados.**

- EJEMPLO_10.m**



Minimizar $f(x,y) = x^2 + y^2 - 10 \cos(x \pi/2) y^2 - 10 \cos(y \pi/2)$ en $-4 \leq x, y \leq +4$ combinando fminsearch con una ejecución multi-start.

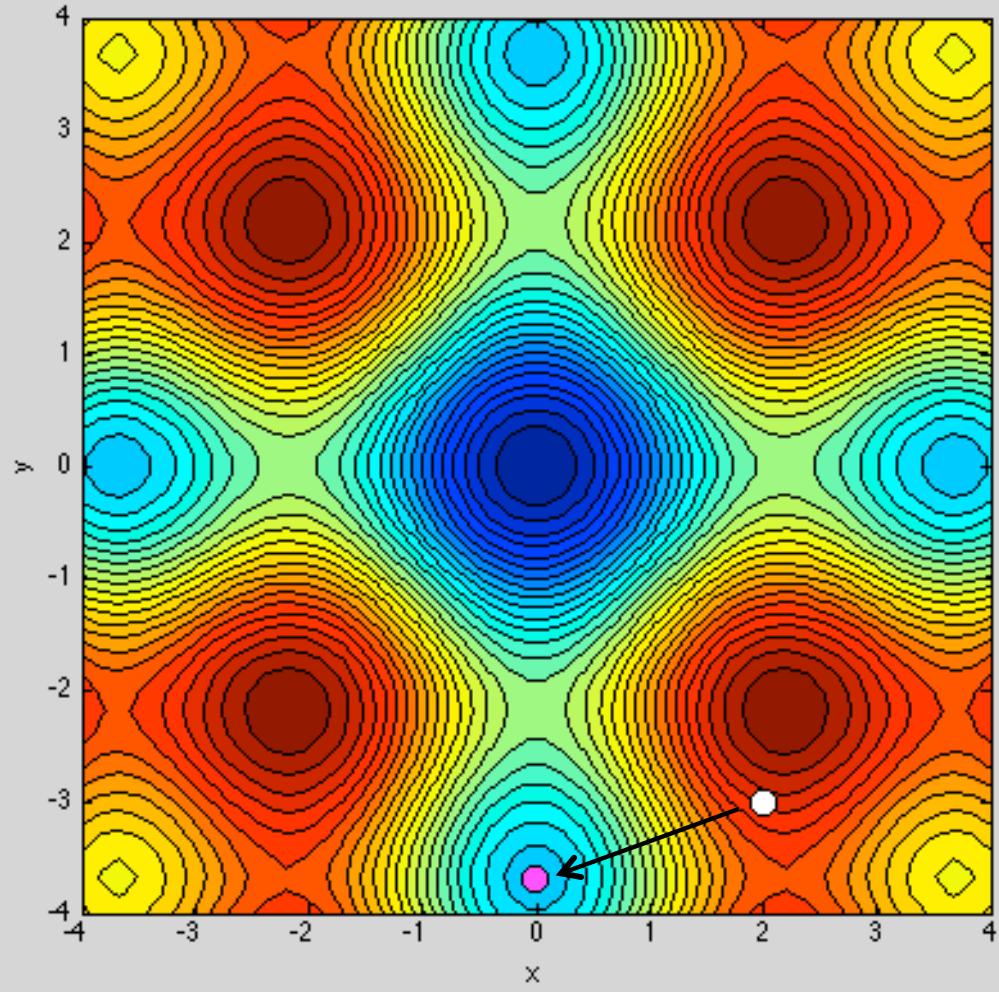


Optimización

Optimización global



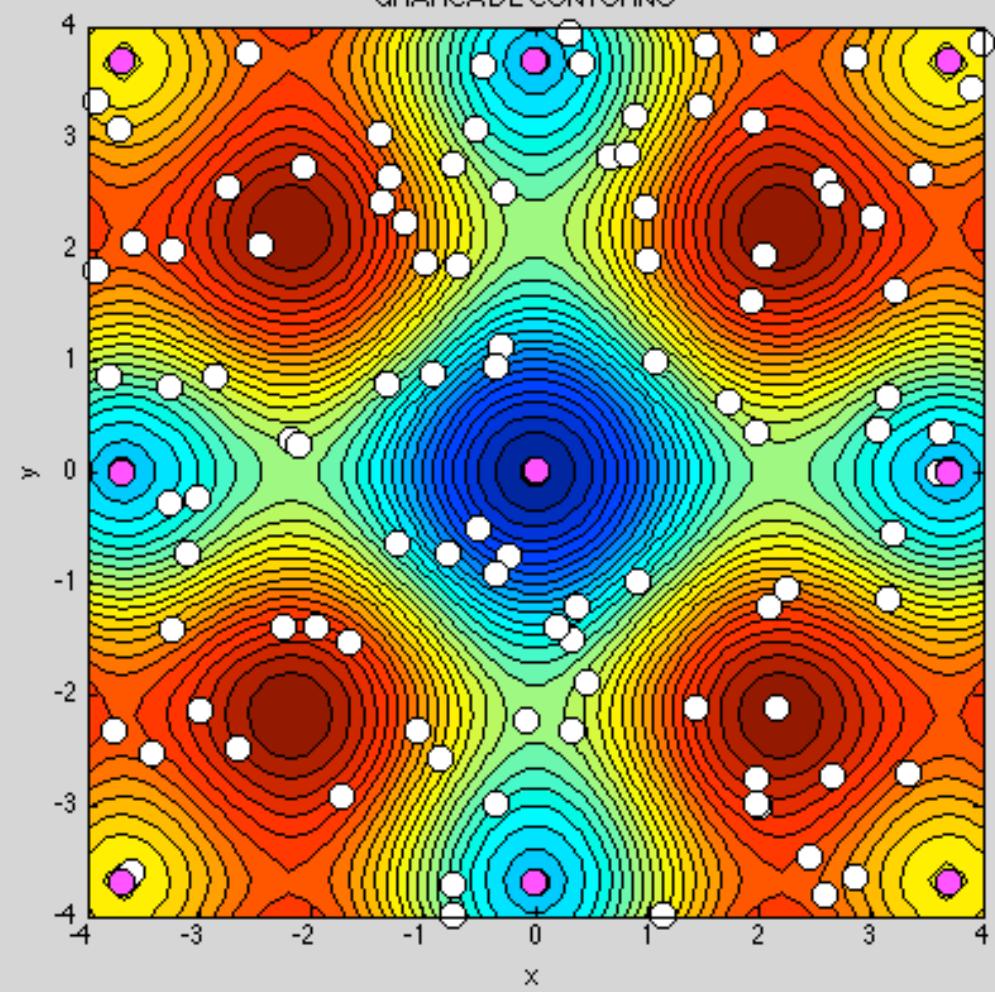
GRAFICA DE CONTORNO



*** OPTIMIZACION LOCAL ***

Mínimo encontrado --> $f_{min} = -5.2$ en $(-0.0, -3.7)$

GRAFICA DE CONTORNO



*** OPTIMIZACION LOCAL CON MULTI-START ***

Menor mínimo alcanzado --> $f(-0.0, -0.0) = -20.0$



Métodos Estocásticos

	Enfriamiento Simulado (Simulated Annealing, SA)	Algoritmo Genético (Genetic Algorithm, GA)
Idea	Metalurgia, Termodinámica	Biología
Función a optimizar	“energía”	“adaptación”
Selección del nuevo estado	Muestro estocástico (Boltzmann, Metropolis)	Mutación y reproducción de los mejor adaptados

- Annealing = **Temple** de metales
Enfriamiento controlado de un metal que permite que los átomos se re-alineen desde un estado de alta energía a un estado de menor energía con ordenación cristalina.
- Algoritmo Genético = **Selección natural** (supervivencia del mejor adaptado)
Los individuos de una población se representan mediante cadenas binarias que juegan el papel de cromosomas y se cruzan y mutan generando nuevos individuos.

Enfriamiento Simulado (SA)

Esquema del algoritmo:

1. Comenzar en un estado s_0 (punto del espacio de diseño). Computar el coste o energía $E(s_0)$. Fijar la temperatura a T .
2. Considerar (de algún modo) un estado diferente s_1 . Computar $E(s_1)$.
3. Si $E(s_1) < E(s_0)$, cambiar del estado s_0 al s_1 .

En otro caso, computar la probabilidad de transición:

$$\text{Prob}(s_0 \rightarrow s_1) = e^{-[E(s_1) - E(s_0)]/KT}$$

*Algoritmo de
Metropolis*

Computar un número aleatorio p uniformemente distribuido en $[0,1]$.

Si $p < \text{Prob}(s_0 \rightarrow s_1)$, aceptar el paso a s_1 . En otro caso, permanecer en s_0 .

4. Ir al paso 2. (Disminuir T después de algunos pasos → **Esquema de enfriamiento**).

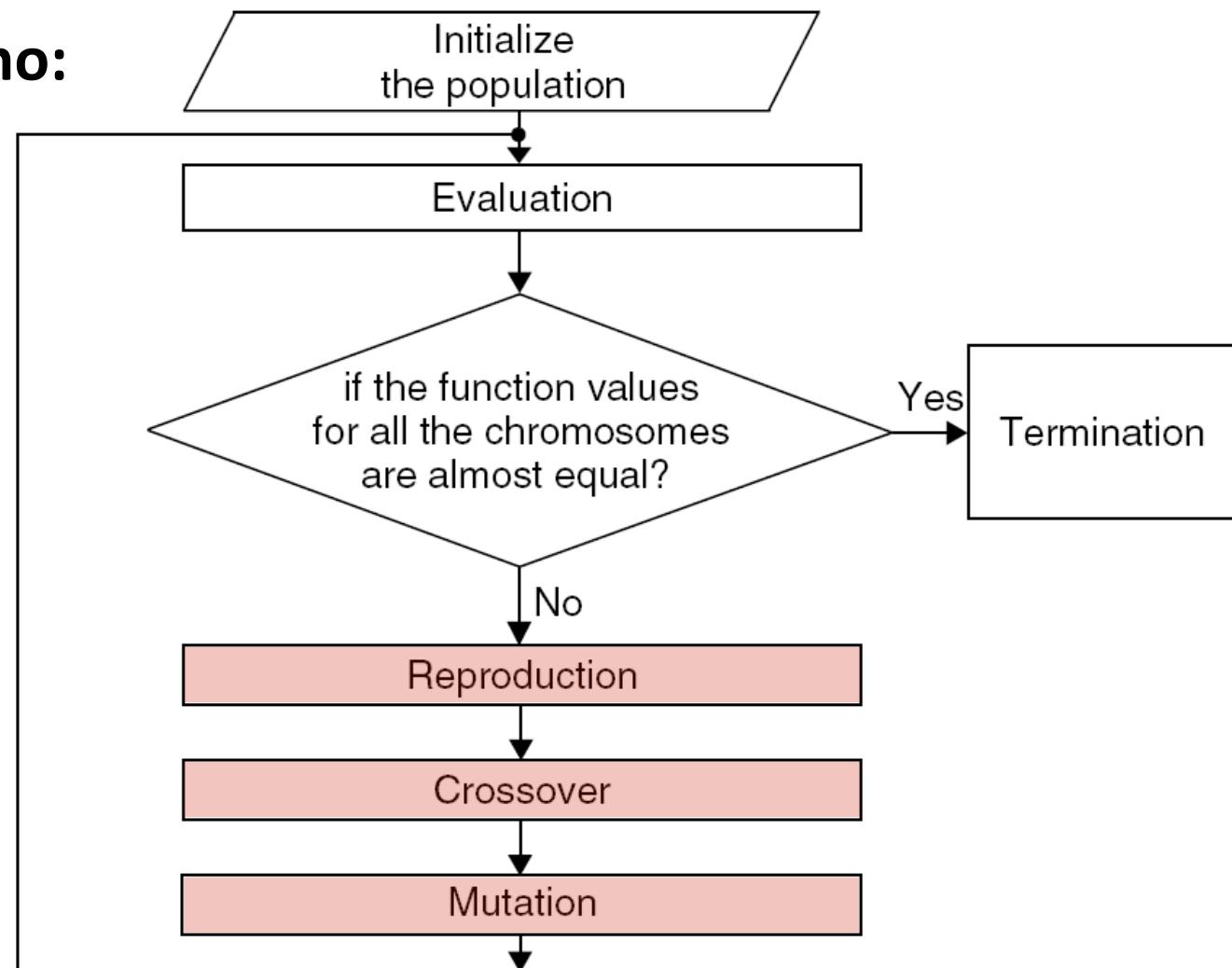
Si T varía demasiado rápido, podemos quedar atrapados en un mínimo local.

Notas
abajo



Algoritmo Genético (GA)

Esquema del algoritmo:



$N_p = 8, N = 2, N_b = [8 \ 8]$ pool P		X_1	X_2	$f(x_1, x_2)$
		population X		fx
01100110	01100110	-1.0000	-1.0000	-10.00
01001111	10101011	-1.9020	-1.7059	-40.95
11110110	01101000	4.6471	-0.9216	44.67
01110111	11101111	-0.3333	4.3725	18.84
10101101	10110011	decode 1.7843	2.0196	evaluate (-54.22)
11011011	11110110	3.5882	4.6471	19.71
11011000	00000001	3.4706	-4.9608	85.84
10011100	00011110	1.1176	-3.8235	-12.54
random pairing		↓ reproduction		
a1 01111100	01111110 a2	-0.1209	-0.0466	0.28
b1 010 10111	1010101 1 b2	-1.5527	1.7356	-36.40
c1 1100 0010	100111 00 c2	2.6259	1.1550	-50.62
d1 10010011	11001111 d2	encode 0.7713	3.1452	-44.58
e1 10101101	10110011 e2	1.7843	2.0196	-54.22
f1 110 00010	1101001 0 f2	2.6360	3.2601	(-74.73)
g1 1010 1101	101100 11 g2	1.7843	2.0196	-54.22
h1 10100001	01001010 h2	1.3160	-2.0846	-35.76
crossover/mutuation ↓				
a1 01111100	01111110 a2	-0.1373	-0.0588	0.31
b1' 01000010	10101010 b2'	-2.4118	1.6667	-46.12
c1' 11001101	100111 11 c2'	3.0392	1.2353	-52.96
d1 10010011	11001111 d2	decode 0.7647	3.1176	-44.73
e1 10101101	10110011 e2	→ 1.7843	2.0196	-54.22
f1' 11010111	11010011 f2'	3.4314	3.2745	(-69.94)
g1' 10100010	10110000 g2'	1.3529	1.9020	-43.50
h1 10100001	01001010 h2	1.3137	-2.0980	-35.88



Traveling Salesman Problem (TSP)

Es un famoso problema de **minimización combinatoria**.

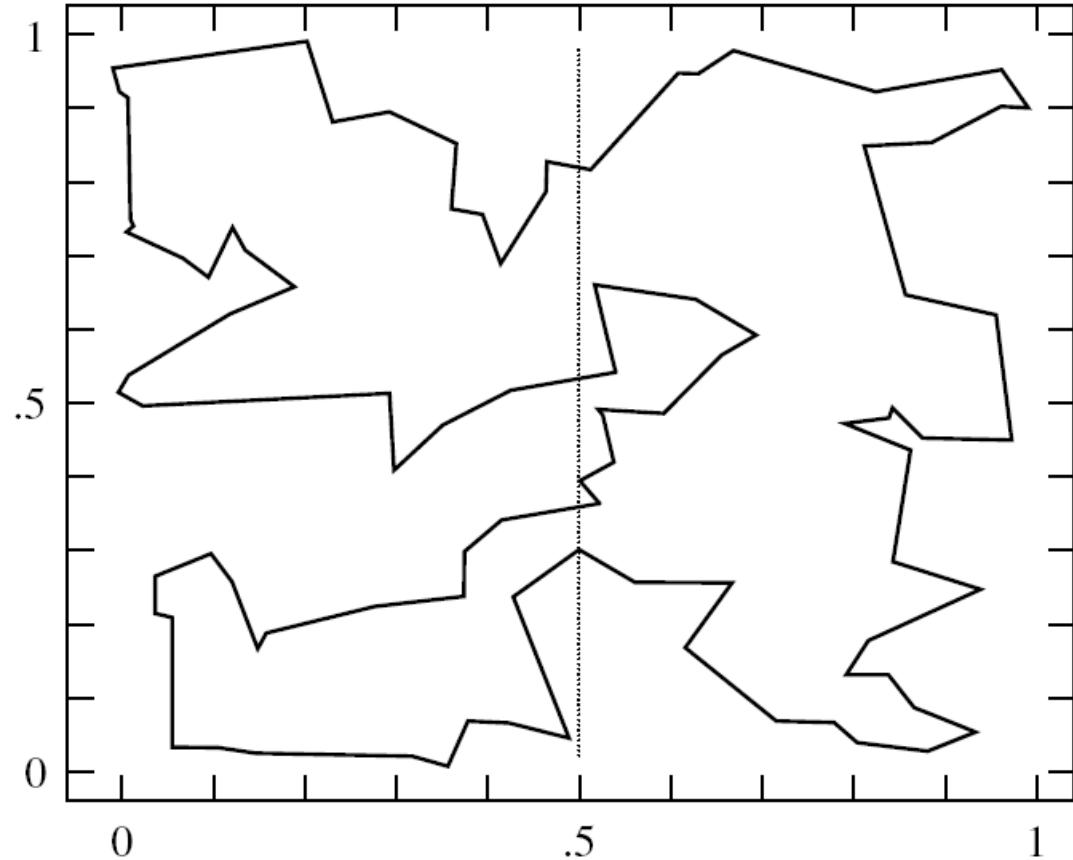
- Problema: Un viajante debe visitar N ciudades y volver a casa siguiendo la ruta más corta.
- Solución: Existen $N!$ caminos diferentes a tomar.
Si N es grande, es imposible probar todas las posibilidades.
(Ej: Si N = 30 existen 2.65×10^{32} rutas posibles)



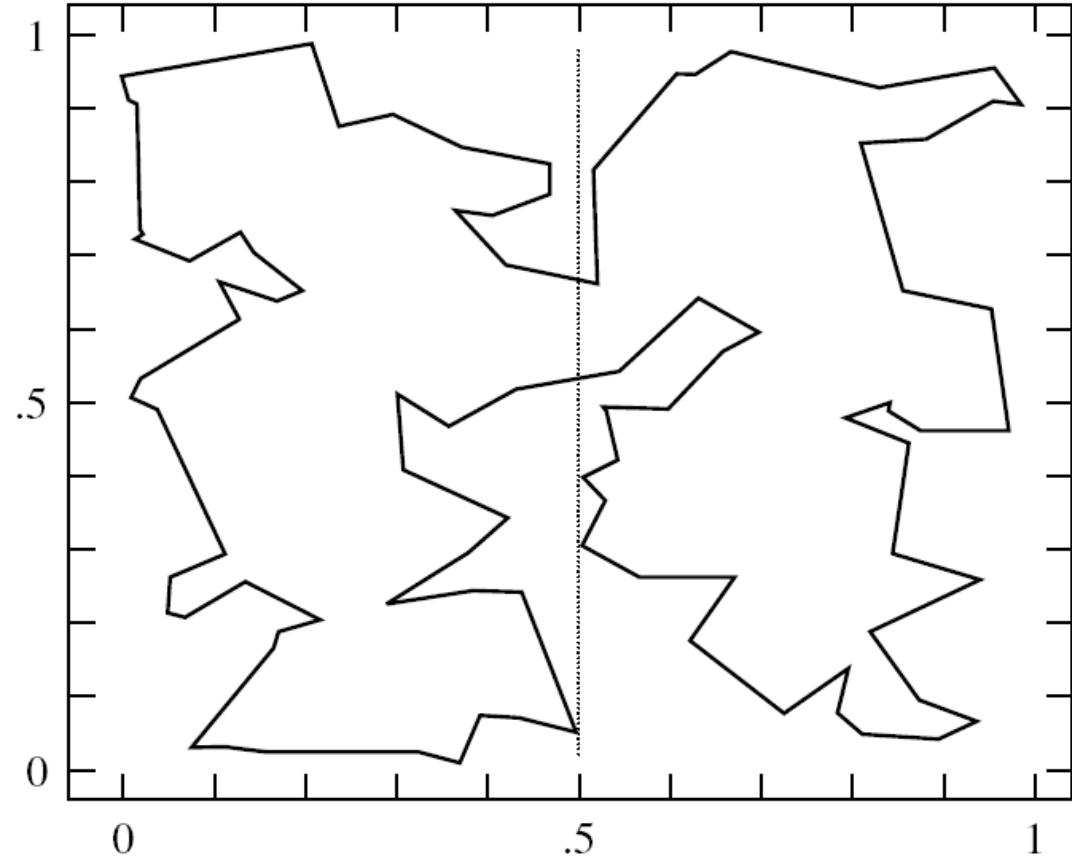
Traveling Salesman (TSP)

Rutas para $N = 100$ ciudades con un río que las divide.

Sin coste asociado al cruce del río

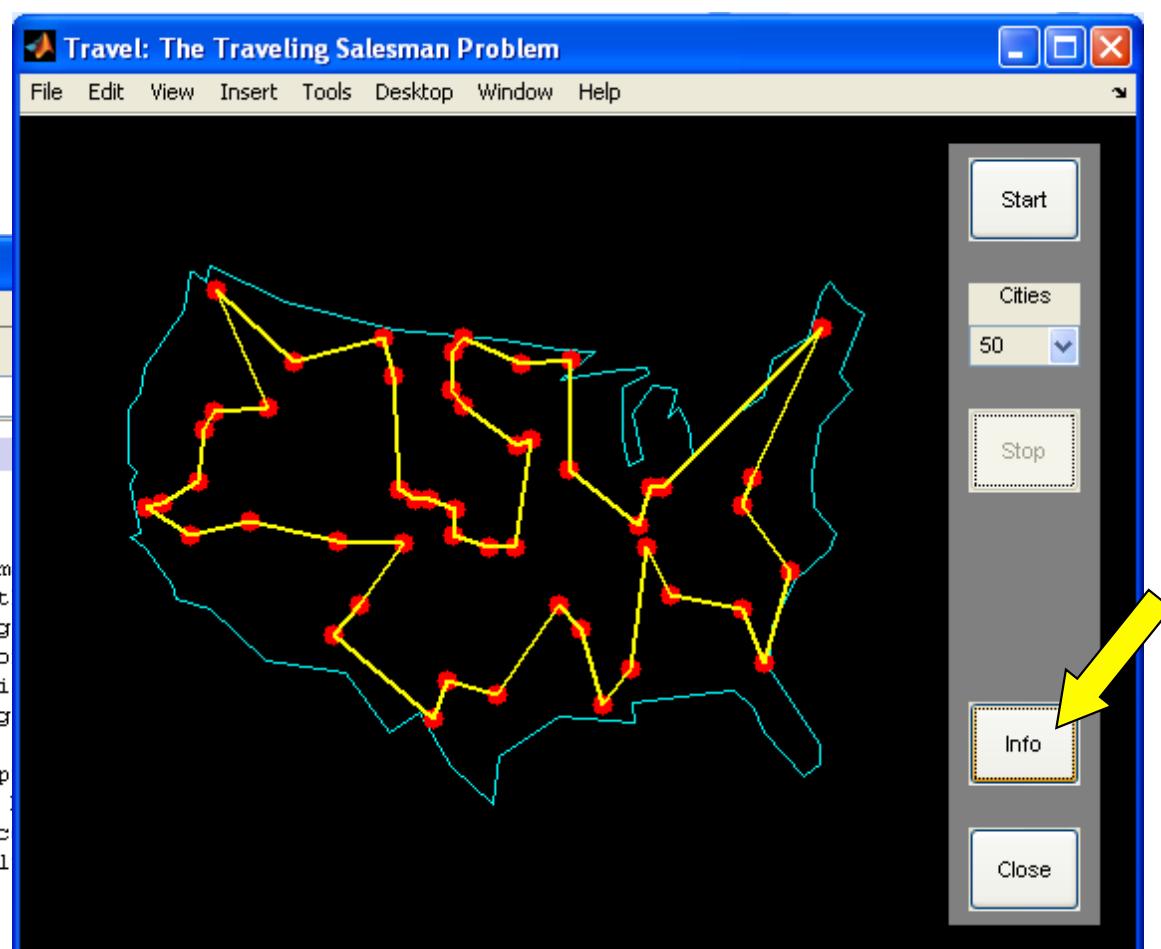
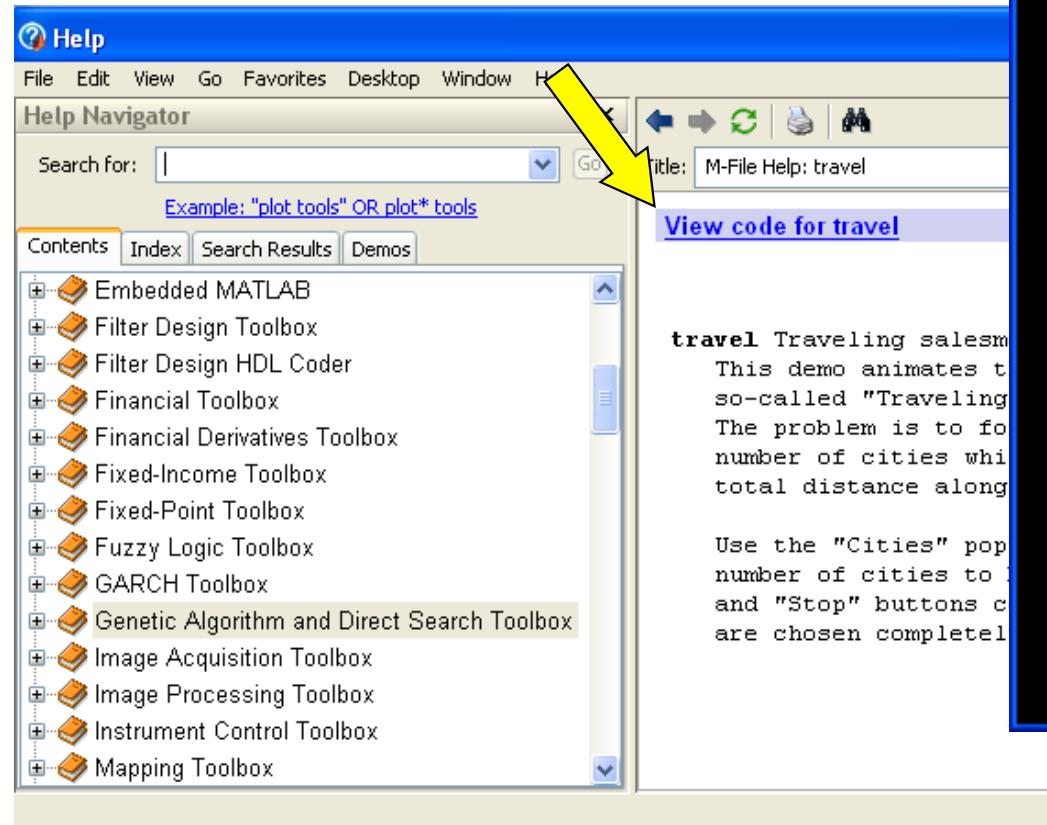


Cruzar el río es caro (peaje)



TSP – Demo de MATLAB

>> travel





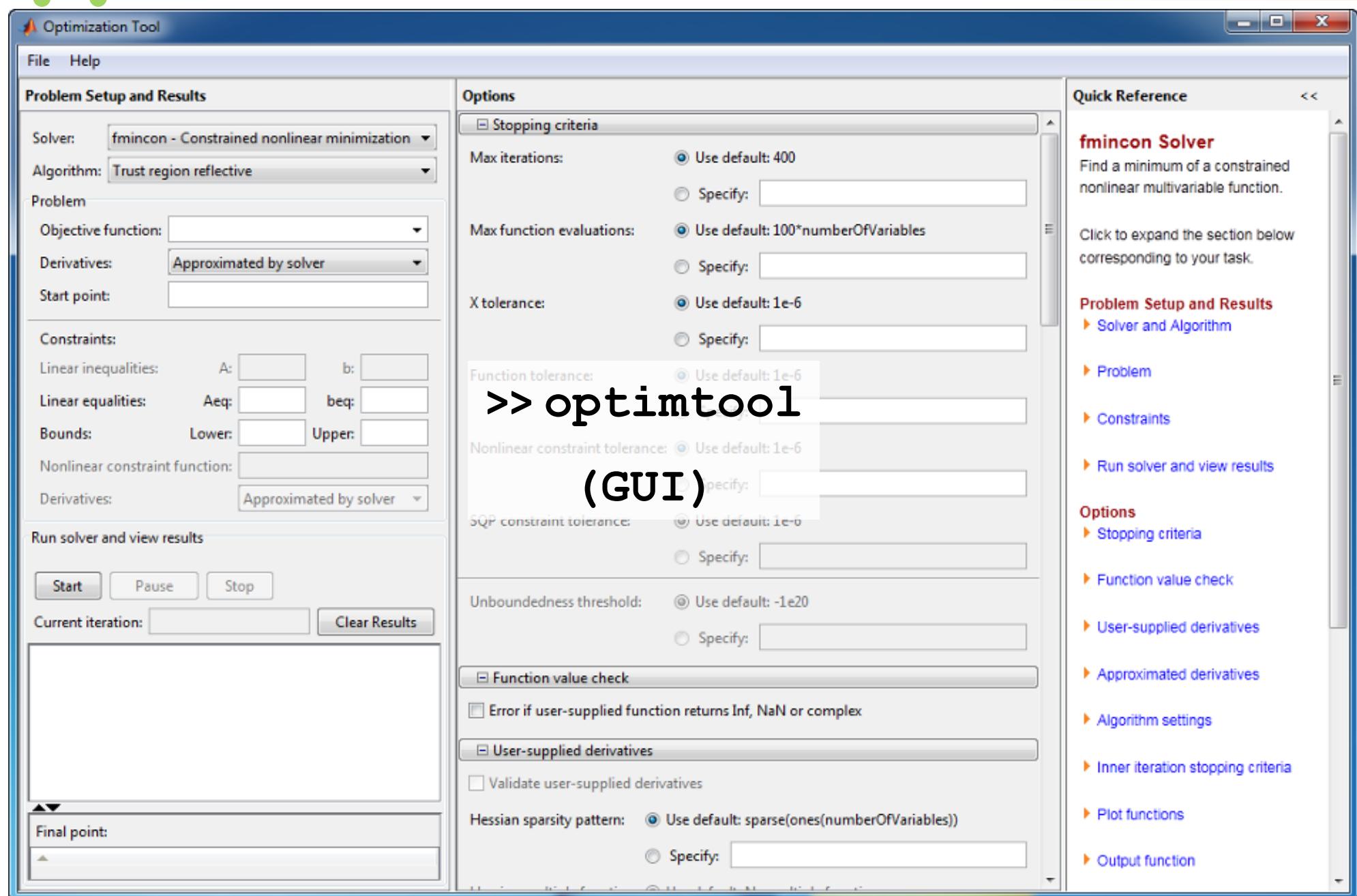
Optimización en MATLAB

■ Optimization Toolbox:

- Problemas de minimización (`fminbnd`, `fminsearch`, `fmincon`, ...)
- Problemas multi-objetivo
- Problemas de solución de ecuaciones (\, `fzero`, `fsolve`, ...)
- Problemas de mínimos cuadrados (\, `lsqnonlin`, ...)

■ Global Optimization Toolbox:

- Métodos deterministas → Multi Start, Global Search, Pattern Search
- Métodos estocásticos → Simulated Annealing, Genetic Algorithm





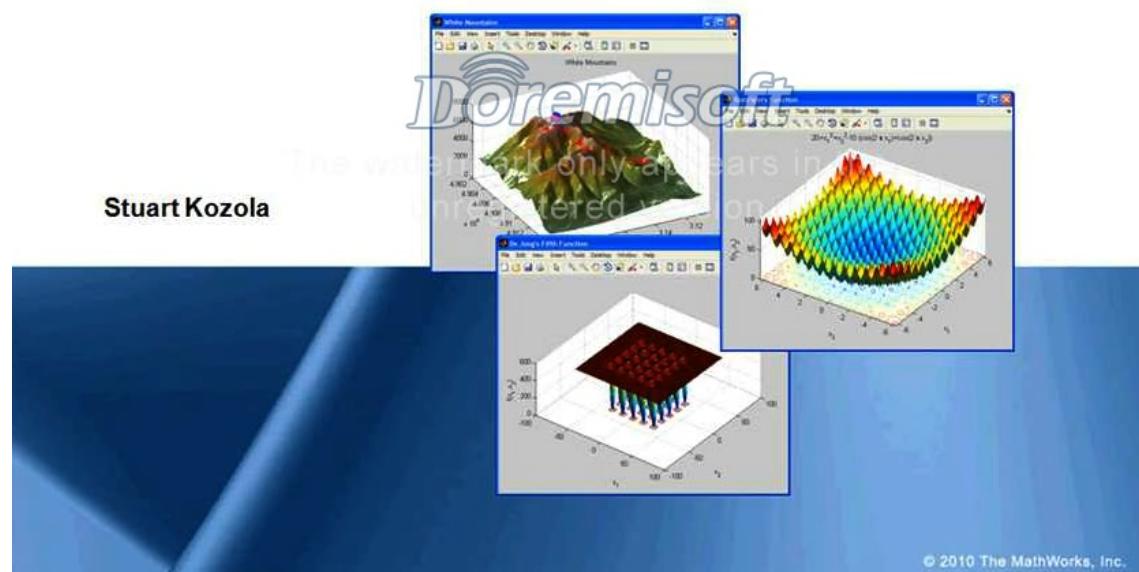
Webinar sobre Optimización Global



MATLAB & SIMULINK

Global Optimization with MATLAB® Products

Stuart Kozola



[Enlace al webinar \(requiere registro\)](#)

--> <http://es.mathworks.com/videos/global-optimization-with-matlab-products-81716.html>

[M-files del webinar](#) --> <http://www.mathworks.com/matlabcentral/fileexchange/27178>

Objetivos

- Comprender la diferencia entre:
 - Optimización local y global
 - Optimización sin y con ligaduras
 - Optimización uni- y multi-dimensional
 - Métodos directos e indirectos
- Saber reformular un problema de búsqueda de máximos para resolverlo mediante un algoritmo de minimización.
- Minimización uni-dimensional:
 - Saber aplicar bracketing para acotar un intervalo unimodal
 - Saber aislar el mínimo con una tolerancia dada (GS, IP, `fminbnd`)
- Minimización multi-dimensional:
 - Saber representar superficies y curvas de contorno
 - Saber localizar el mínimo (`downhill simplex`, `fminsearch`)
- Saber aplicar la función `fmincon` de minimización con restricciones.