Trabajo de Fin de Máster:

OPTIMIZACIÓN GLOBAL DE FUNCIONES CONTÍNUAS

ALUMNO: María Sánchez de Andrés
PROFESOR TFM: Miguel Delgado Pineda

Septiembre 2011

Índice general

Capítulo 1

INTRODUCCIÓN

Nos encontramos con el siguiente problema de optimización:

$$\text{min f(x) o max f(x)}, x \in \overline{X}$$

donde $f : \overline{X} \to \mathbb{R}$ es una función contínua, y $\overline{X}$ es un conjunto compacto, robusto[1] y no vacío.

Dado que max f(x) = -min(-f(x)), los problemas de maximización se pueden convertir en problemas de minimización, y podemos hablar genéricamente de "optimización". En este documento consideraremos únicamente problemas de minimización.

1.1. Tres problemas diferentes de optimización

Podemos distinguir tres problemas de optimización, dependiendo del significado exacto de la notación: min f(x), $x \in \overline{X}$

Problema A. Optimización local tradicional, o programación matemática:

El problema consiste en encontrar un elemento (un punto) $x^\circ \in \overline{X}$ tal que $f(x^\circ) \leq f(x), \forall x \in N_\delta(x^\circ)$, donde $N_\delta(x^\circ) = \{x \in \overline{X} / \|x - x^\circ\| < \delta, \delta > 0\}$, en algún entorno δ de $\overline{X}$, siendo $\|.\|$ una norma.

El problema A es muy antiguo y hay diferentes métodos para su solución, basados en (sub)gradientes, series de Taylor, etc.

No vamos a considerar el problema A en nuestro estudio.

Problema B. Optimización global limitada:

[1] Un conjunto Y es robusto si cumple cl int Y = cl Y.

El problema consiste en encontrar un elemento (un punto) $x^\circ \in \overline{X}$ tal que $f(x^\circ) \leq f(x), \forall x \in \overline{X}$.

Durante años se ha trabajado en este problema y se han encontrado varios métodos para llegar a una solución, como las técnicas enumerativas, separación, métodos cluster, etc.

Tampoco vamos a considerar el problema B en nuestro estudio.

Problema C. Optimización global completa:

El problema consiste en encontrar el conjunto $\overline{X}^\circ \subset \overline{X}$ tal que $f(x^\circ) = p^\circ < f(x), \forall x^\circ \in \overline{X}^\circ$, y $\forall x \in \overline{X} - \overline{X}^\circ$, siendo p° una constante. El conjunto $\overline{X}^\circ$ puede ser un punto o varios, una sucesión o un conjunto contínuo.

Podemos utilizar una notación más abreviada para el problema anterior:

$$\boxed{\begin{array}{r} \text{Encontrar } p^\circ = \min \text{ f(x)}, x \in \overline{X} \\ \text{y } \overline{X}^\circ = \{x \in \overline{X} / f(x) = p^\circ\} \end{array}}$$

teniendo en cuenta que la minimizacion de este problema se lleva a cabo sobre todo el conjunto $\overline{X}$, luego p° representa el *único* mínimo global de f(x) en $\overline{X}$.

Para resolver este problema solo se han encontrado hasta la fecha tres maneras de resolverlo: el método integral de optimización global, la aproximación por intervalos y el algoritmo Cubic y sus extensiones.

En este documento vamos a presentar la solución al problema C mediante el Algoritmo Cubic.

1.2. Discusión

La compacidad del conjunto $\overline{X}$ garantiza la existencia de mínimo y máximo de f(x), y los elementos de $\overline{X}$ o conjuntos $\overline{X}^\circ$ donde la función es óptima. En algoritmos iterativos, la compacidad es un requerimiento técnico y el problema se puede resolver sin esta condición. Aquí asumimos la compacidad de $\overline{X}$ por simplicidad.

La robustez de $\overline{X}$ es un requerimiento más importante. Si el conjunto de $\overline{X}$ es compacto pero no inf-robusto, entonces $\overline{X}$ y $\overline{X}^\circ$ existen pero no se pueden obtener por ningún método conocido.

Si $\overline{X}$ y f son convexos, los problemas A y B coinciden, pero C es distinto. Si $\overline{X}$ es convexo y f es estríctamente convexo, entonces A, B y C coinciden

en tener un único mínimo. Sin embargo, los tres difieren en la solución y en los métodos de cálculo de dicha solución.

Capítulo 2

EL ALGORITMO CUBIC

2.1. Introducción

La idea del algoritmo Cubic es dividir el conjunto en subconjuntos más pequeños y detectar y descartar aquellos que no contengan ningún minimizador global.

Consideramos el cubo cerrado $\overline{C} \in \mathbb{R}$, con el borde superior $c > 0$ y el problema de encontrar el valor mínimo global

$$s^{\circ} = min\ f(x), \forall x \in \overline{C},$$

y el conjunto de todos los minimizadores globales

$$\overline{K}^{\circ} = \{x/f(x) = s^{\circ}, x \in \overline{C}\}.$$

Asumimos aquí que la función f:$\mathbb{R}^n \to \mathbb{R}$ es una función Lipschitz, es decir que f(x) satisface la siguiente condición:

$$|f(x) - f(\widetilde{x})| \leq L\|x - \widetilde{x}\|, L = const > 0, x, \widetilde{x} \in \overline{C}$$

donde $\|.\|$ denota la norma Euclídea:

$$\|x - \widetilde{x}\| = \left[\sum_{i=1}^{n}(x_i - \widetilde{x}_i)^2\right]^{1/2}$$

Los puntos $x^{\circ} \in \overline{K}^{\circ}$ se llaman minimizadores globales[1].

2.2. Generador de la partición de igual diámetro

Definición 2.2.1 *Un **generador de partición** es un método para dividir $X \in \mathbb{R}^n$ en subconjuntos disjuntos $X_i \subset X$, con $X_i \cap X_j = \emptyset, i \neq j$ y tal que $\bigcup \overline{X}_i = \overline{X}$.*

[1] $\overline{K}$ denota la clausura de K.

Para particionar el cubo $\overline{C} \in \mathbb{R}^n$ tomamos un entero $N \geq 2$ y dividimos cada borde de $\overline{C}$ adyacente a uno de los vértices que elijamos en N partes. Hacemos después líneas paralelas a los lados del cubo para obtener asi una partición de $\overline{C}$ en N^n subcubos iguales $\overline{C}_i$, tales que $\overline{C}_i \cap \overline{C}_j = \emptyset$ si $i \neq j$, y $\bigcup \overline{C}_i = \overline{C}$. Todos los subcubos $\overline{C}_i$ tienen el mismo diámetro (longitud de la diagonal), lo cual es esencial para el algoritmo Cubic.

2.3. Generador de la rejilla por traslación

Definición 2.3.1 *Un **generador de rejilla** es una regla para elegir un punto $x \in X_i$.*

Sea el punto $x_0 \in \overline{C}$ y un entero $N \geq 2$ y una partición de $\overline{C}$ en N^n subcubos iguales $\overline{C}_i^1$ con las propiedades del apartado anterior.

El volumen de cada $\overline{C}_i^1$ es c^n/N^n y su diagonal es $c\sqrt{n}/N$, es decir:

$$d(\overline{C}_i^1) = sup\|x - x'\| = c\sqrt{n}/N, \forall i, x, x' \in \overline{C}_i^1.$$

Después de la partición, el punto $x_0 \in \overline{C}$ estará en uno de los conjuntos (o más, si x_0 está en una frontera común) de $\overline{C}_i^1$. Si ese es el caso, asignamos x_0 a uno solo de los $\overline{C}_i^1$, digamos $\overline{C}_{i_0}^1$, y decimos que x_0 es el representante de $\overline{C}_{i_0}^1$. Aplicamos traslación paralela de $\overline{C}_{i_0}^1$ para hacerlo coincidir sucesivamente con todos los $\overline{C}_i^1$, $i = 1...N^n, i \neq i_0$. De esta forma, $x_0 \in \overline{C}_{i_0}$ define el representante $x_i \in \overline{C}$ de cada $\overline{C}_{i_0}^1$. Esta regla define el **generador de rejilla por traslación**, y la colección de $x_i^1, i = 1...N^n, x_{i_0}^1 = x_0$ nos da la rejilla para cualquier elección de $x_0 \in \overline{C}$.

Determinados subcubos $\overline{C}_i^1 \subset \overline{C}$ se particionarán después de la misma forma. El índice superior m de $\overline{C}_i^m, m = 1, 2...$ representa el número de partición (iteración).

2.4. Cálculo de la constante

Denotamos al conjunto de índices de la primera partición como $I_0 = \{1, 2, ..., N^n\}$. Dados los puntos $x_i^1 \in \overline{C}_i^1$, calculamos todos los $f(x_i^1)$ y la primera constante de comparación:

$$s_1 = minf(x_i^1), i \in I_0.$$

Las siguientes constantes $s_m, m = 2, 3...$ se calculan de la misma manera en el proceso de iteración.

2.5. Operador de descarte de subconjuntos basado en la condición de Lipschitz

Definición 2.5.1 *Una regla para descartar un subconjunto que con seguridad no contiene un minimizador global x^0 se llama* ***operador de descarte****.*

Para construir un operador de descarte, definimos las sucesivas constantes de descarte r_m de la siguiente forma:

$$r_m = Ad(\overline{C}_i^m) = \frac{Ac\sqrt{n}}{N^m}, A \geq L > 0, m = 1, 2..., \forall i$$

Se cumple que $r_m = r_{m-1}/N$ y $r_m \to 0$ si $m \to \infty$.

Ahora, si tenemos una constante de comparación s_m, un operador de descarte simple viene dado por la siguiente desigualdad:

$$f(x_i^m) - s_m > r_m, \text{ para m=1,2...}$$

que utilizamos para descartar todos los subcubos $\overline{C}_i^m$ tal que la desigualdad anterior se satisface para uno de sus puntos, digamos x_i^m, que pertenece a la rejilla. La prueba de que esta regla define efectivamente un operador de descarte la veremos en el apartado siguiente.

2.6. El algoritmo Cubic

Dado un cubo $\overline{C} \in \mathbb{R}^n$, elegir su representante $x_0 \in \overline{C}$ y una constante entera de partición $N \geq 2$.

Iteración 1. Particionar $\overline{C}$ en N^n subcubos $\overline{C}_i^1$ como se describe en la sección 2.2. Aplicar el generador de rejilla por traslación explicado en la sección 2.3 para obtener los primeros puntos de rejilla $\{x_i^1\}$ como representantes de los subcubos $\overline{C}_i^1$.

Calcular f(x_i^1) y la primera constante de comparación:

$$s_1 = min\, f(x_i^1), i \in I_0,\ I_0 = \{1, 2, ..., N^n\}$$

como se explica en la sección 2.4.

Descartamos ahora todos los $\overline{C}_i^1$ que cumplen $f(x_i^1) - s_1 > r_1$, $i \in I_0$. Llamamos I_1 al conjunto de índices que no descartamos. La clausura de los subcubos que no hemos descartado define el siguiente conjunto cuasi-Cubic:

$$\overline{K}_1 = \{x | x \in \overline{C}_i^1, i \in I_1\}.$$

Si llamamos $\overline{K}_0 = \overline{C}$, es trivial ver que $I_1 \subseteq I_0$, $\overline{K}_1 \subseteq \overline{K}_0$.

Iteraciones sucesivas. Particionar cada $\overline{C}_i^1 \subset \overline{K}_1$ del mismo modo que $\overline{C}$ y generar una nueva rejilla x_i^2 con la misma regla de traslación que antes. Repetir la regla de iteración 1 para definir $I_2, \overline{K}_2$. Volver a particionar y a generar otra rejilla para definir $I_3, \overline{K}_3$ y así sucesivamente.

Este proceso nos da dos secuencias monótonas:

$$s_1 \geq s_2 \geq ... \geq s_m \geq ...$$

$$\overline{C} \supseteq \overline{K}_1 \supseteq \overline{K}_2 \supseteq ... \supseteq \overline{K}_m \supseteq ...$$

La primera sucesión tiende a un límite ya que está acotada inferiormente por la existencia de un mínimo global s^0, y la segunda tiene una intersección no vacía que define su límite, por la existencia de un conjunto no vacío de minimizadores globales $\overline{K}_0$.

Lo que tenemos que ver es si estos son efectivamente los límites, es decir si

$$\lim_{m\to\infty} s_m = s^0, \lim_{m\to\infty} \overline{K}_m = \overline{K}_0$$

o solo

$$\lim_{m\to\infty} s_m = \overline{s} > s^0, \lim_{m\to\infty} \overline{K}_m = \bigcap_{m=1}^{\infty} \overline{K}_m = \overline{K} \supset \overline{K}_0$$

2.7. Teorema de convergencia

Teorema 2.7.1

$$\lim_{m\to\infty} s_m = s^0 = \text{mín}\, f(x), x \in \overline{C},$$

$$\lim_{m\to\infty} \overline{K}_m = \bigcap_{m=1}^{\infty} \overline{K}_m = \overline{K}_0 = \{x | f(x) = s^0, x \in \overline{C}\}$$

Demostración:

a) Existencia y naturaleza del límite en $\mathbb{R}$:

Dado que $\overline{C}$ es compacto, $\exists x^0$ tal que $f(x^0) = s^0 = min f(x), x \in \overline{C}$. Ya que s_m es monótona y acotada inferiormente por s_0, tiende a un límite $\overline{s} \geq s_0$. $\exists \overline{x}$ tal que $f(\overline{x}) = \overline{s}$. Como $r_m = \frac{Ac\sqrt{n}}{N^m} \to 0$ cuando $m \to \infty$, si $\overline{s} > s_0$, los puntos $\overline{x}$ se descartarán en alguna iteración, si por lo menos un minimizador global x^0 permanece en el proceso indefinidamente, es decir si $\exists x^0 \in \overline{K}_m, \forall m$. En este caso, tendríamos $\lim_{m\to\infty} s_m < \overline{s}$, lo que contradice su definición: $\overline{s} = \lim_{m\to\infty} s_m$. Por tanto, para demostrar que $\overline{s} = s_0$, tenemos que demostrar que $\overline{K}_0 \cap \overline{K}_m \neq \emptyset$, para todo m = 1,2...

b) No eliminación de un minimizador global x^0:

El generador de rejilla garantiza que cada x_i^m se mantendrá en el proceso de iteración hasta ser descartado con su correspondiente subcubo C_i^m por el operador de descarte r_m. Por otra parte, por la condición de Lipschitz, tenemos que

$$\max_{x\in\overline{C}_i^m} f(x) - \min_{x\in\overline{C}_i^m} f(x) \leq L \parallel arg \max_{x\in\overline{C}_i^m} f(x) - arg \min_{x\in\overline{C}_i^m} f(x) \parallel$$

$$\leq L \max_{x,x'\in\overline{C}_i^m} \parallel x - x' \parallel \leq Ad(\overline{C}_i^m) = r_m, A \geq L > 0,$$

lo que implica

$$f(x_i^m) - s_m > r_m \geq \max_{x\in\overline{C}_i^m} f(x) - \min_{x\in\overline{C}_i^m} f(x) \geq f(x_i^m) - \min_{x\in\overline{C}_i^m} f(x),$$

luego

$$s_m < \min_{x\in\overline{C}_i^m} f(x),$$

para todos los subcubos $\overline{C}_i^m$ descartados. Por lo tanto, el operador de descarte no puede eliminar un punto $x^0 \in \overline{K}^0$ que sea un mínimo global. Esto demuestra que

$$\lim_{m\to\infty} s_m = s^0$$

y también la inclusión $\overline{K}^0 \subseteq \overline{K}_m$ para todo m=1,2..., de donde

$$\overline{K}^0 \subseteq \bigcap_{m=1}^{\infty} \overline{K}_m.$$

Falta probar la inclusión inversa.

c) Existencia y naturaleza del límite en $\mathbb{R}^n$:

Sea $\overline{K} = \bigcap_{m=1}^{\infty} \overline{K}_m$. Como $\overline{K}_0 \neq \emptyset$ y $\overline{K}_0 \subseteq \overline{K}$, tenemos que $\overline{K} \neq \emptyset$. Tomamos un punto $\widetilde{x} \in \overline{K}$. Como $\overline{K} = \bigcap_{m=1}^{\infty} \overline{K}_m$, si $\widetilde{x} \in \overline{K}$ entonces $\widetilde{x} \in \overline{K}_m$ para todo m=1,2..., y dado que $\widetilde{x} \in \overline{C}_i^m \subset \overline{K}$ tenemos que:

$$\|f(\widetilde{x}) - s_m\| = \|f(\widetilde{x}) - f(x_i^m) + f(x_i^m) - s_m\| \leq \|f(\widetilde{x}) - f(x_i^m)\| + \|f(x_i^m) - s_m\| \leq 2r_m$$

para todo m=1,2..., y por tanto,

$$s^0 = \min_{x\in\overline{C}} f(x) \leq f(\widetilde{x}) \leq s_m + 2r_m \to s^0 \; cuando \; m \to \infty,$$

ya que $r_m \to 0, s_m \to s^0$ cuando $m \to \infty$. Con esto tenemos que $f(\widetilde{x}) = s^0$ para todo $\widetilde{x} \in \overline{K}$, lo que significa que $\overline{K} \subseteq \overline{K}^0$. Con la inclusión $\overline{K}^0 \subseteq \overline{K}$

que hemos obtenido en el apartado b), concluimos que $\overline{K} = \overline{K}^0$, y por tanto el conjunto $\overline{K} = \bigcap_{m=1}^{\infty} \overline{K}_m$ es el conjunto que contiene todos los puntos que son mínimo global y sólo esos puntos.

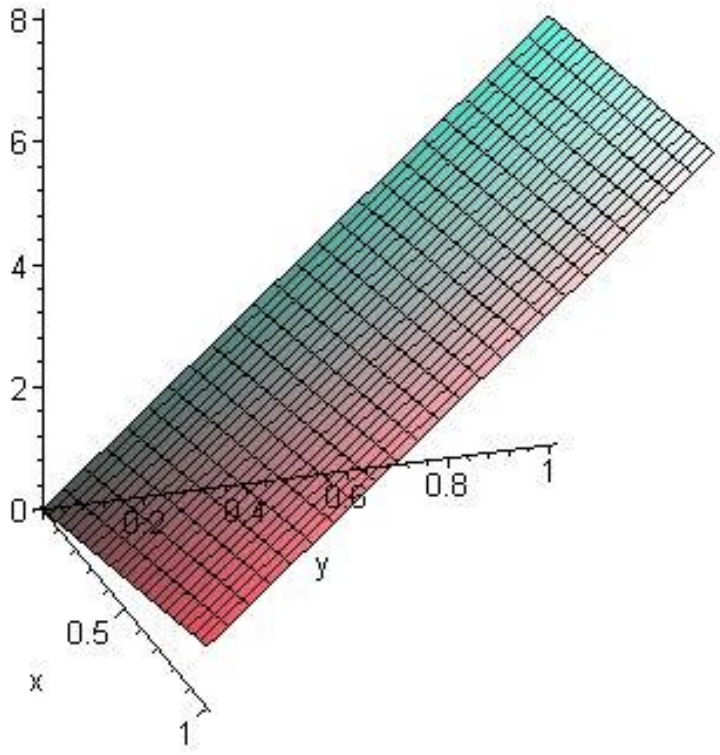
Con esto se completa la demostración.

2.8. Ejemplo

Vamos a ilustrar los conceptos anteriores con un ejemplo sencillo.

Ejemplo: determinar
$s^0 = \text{mín}(x_1 + 7x_2)$, con $x_1, x_2 \in \overline{C}_0$ y
$\overline{K}_0 = \{x_1, x_2 | x_1 + 7x_2 = s^0, x_1, x_2 \in \overline{C}_0\}$
siendo $\overline{C}_0$ el cuadrado unidad de vértices (0,0), (0,1), (1,0), (1,1).

Esta es una función sencilla ya que se trata de un plano, y el mínimo por tanto ha de estar en uno de los cuatro vértices del cuadrado. La función, en tres dimensiones, es de la siguiente forma:



Solución: Para una función lineal $z = a_1 x1 + ... + a_n x_n$, la constante de Lipschitz es $L = (a_1^2 + ... + a_n^2)^{1/2}$, por lo tanto para nuestra función la cons-

tante L=$\sqrt{50}$. Tomamos N=2 (número de particiones por lado), A=L=$\sqrt{50}$ y $x_0 = (0, 0)$. El operador de descarte es

$$r_m = \frac{Ac\sqrt{n}}{N^m} = \frac{\sqrt{50} \cdot 1 \cdot \sqrt{2}}{2^m} = \frac{5}{2^{m-1}}, m = 1, 2...$$

Iteración 1: Partimos el cubo inicial en 4 subcubos, lo que nos da:

$$\{x_i^1\} = \{(0,0), (0, \frac{1}{2}), (\frac{1}{2}, 0), (\frac{1}{2}, \frac{1}{2})\}$$

$$\{f(x_i^1)\} = \{0, \frac{7}{2}, \frac{1}{2}, 4\}.$$

Por tanto $s_1 = minf(x_i^1) = 0$, $r_1 = 5$, y la inecuación $f(x_i^1) - s_1 \geq 5$ no nos da ninguna solución, luego en la primera iteración no descartamos ningún punto.

Iteración 2: Partimos cada uno de los subcubos en 4 subcubos, por lo que tenemos 16 subcubos en esta iteración. Generamos la rejilla mediante el procedimiento visto anteriormente.

En la siguiente tabla se indican los valores de $f(x_i^2)$ de los 16 puntos de rejilla, ordenados como correspondería geométricamente (eje x_1 en la horizontal, y eje x_2 en la vertical):

5,25	5,5	5,75	6
3,5	3,75	4	4,25
1,75	2	2,25	2,5
0	0,25	0,5	0,75

El mínimo vuelve a ser $s_2 = minf(x_i^2) = 0$; $r_2 = 5/2$, y la inecuación $f(x_i^2) - s_2 \geq 5/2$ nos permite descartar 8 de los 16 subcubos. Nos quedamos entonces con los subcubos señalados con X.

X	X	X	X
X	X	X	X

Iteración 3:

Volvemos a partir cada subcubo en 4. Tenemos ahora 32 subcubos para los que generamos los puntos de rejilla. Calculamos los valores de $f(x_i^3)$, que aparecen en la siguiente tabla también como antes ordenados geométricamente:

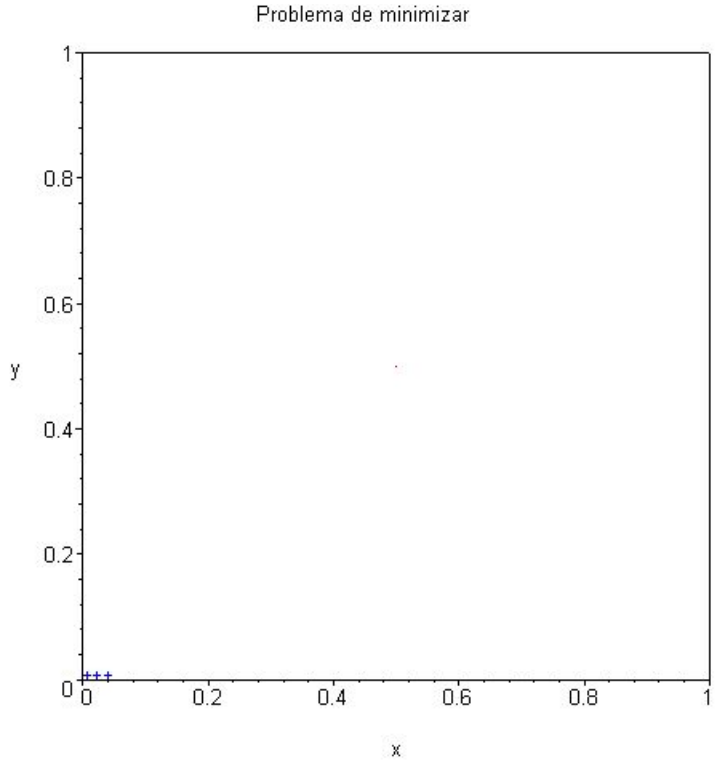
2,625	2,75	2,875	3	3,125	3,25	3,375	3,5
1,75	1,875	2	2,125	2,25	2,375	2,5	2,625
0,875	1	1,125	1,25	1,375	1,5	1,625	1,75
0	0,125	0,25	0,375	0,5	0,625	0,75	0,875

El mínimo vuelve a ser $s_3 = minf(x_i^3) = 0$; $r_3 = 5/4$, y la inecuación $f(x_i^3) - s_3 \geq 5/4$ nos permite descartar 20 de los 32 subcubos. Nos quedamos entonces con los subcubos señalados con X.

X	X	X	X				
X	X	X	X	X	X	X	X

Para ver la convergencia más rápidamente, en lugar de hacerlo de forma manual podemos utilizar la programación del algoritmo Cubic en Maple (se verá en el apartado siguiente). Éste nos da el siguiente resultado:

Nivel de subdivisiones = 7
NdePuntosNoEliminados = 3
LadoDelSubcubo = 0.007812500000
NDePuntosEpsilonSeparados = 1
Valor extremo = 0.09375000000



Si continuamos con iteraciones sucesivas, obtenemos $s^0 = \lim_{m\to\infty} s_m = 0$, y $\overline{K}^0 = \{(0,0)\}$. Aunque sabemos que el proceso de iteración no nos da s^0 y $\overline{K}^0$ de forma exacta, podemos aproximarlos tanto como queramos.

Capítulo 3

Programación del algoritmo Cubic

Este es un ejemplo de programación del algoritmo Cubic, para el que se ha utilizado el programa matemático Maple®. Como se ha visto en el apartado anterior, para generar la rejilla podemos escoger cualquier punto del subcubo como representante. En la documentación aportada para realizar el trabajo (Maple Code in $\mathbb{R}^n$ of the Cubic Algorithm for Global Optimizatio of Continuous Functions with Box Constraints, de M. Delgado Pineda) se utiliza el vértice inferior izquierdo de cada subcubo; para implementar aquí el algoritmo he escogido el punto medio del subcubo como representante. Con cualquier elección del representante se obtiene el mismo resultado.

Los principales procedimientos utilizados para la implementación del algoritmo son:

PuntoR: Pasa del hipercubo unitario $[0,1]^n$ al hiperrectángulo.

ListaPuntosR: Pasa los puntos del hipercubo al hiperrectángulo.

PuntoOrigen: Crea el punto origen en el hipercubo unitario, que será el punto medio del hipercubo.

verticesCuboU: calcula los puntos medios de los subcubos después de hacer la primera partición del hipercubo unidad, utilizando la numeración binaria (utiliza el procedimiento ListaBinarios).

NCubo: genera los puntos medios de un subcubo, basándose en el procedimiento anterior.

DepuraPuntos: Algoritmo de descarte. Evalúa la función correspondiente

en los puntos dados, calcula los mínimos de esa lista, y descarta los puntos que no cumplen con la condición del algoritmo.

TroceaCubos: A partir de una lista de puntos, saca los puntos medios de todos los subcubos generados utilizando el procedimiento NCubo.

LimpiaPuntos: Elimina los puntos muy próximos, según la precisión epsilon que definamos.

El código en Maple queda como sigue:

Maple code in $\mathbb{R}^n$ of the Cubic Algorithm for Global Optimizatio of Continous Functions with Box Constraints

>restart:with(plots):with(LinearAlgebra):

Procedimientos usados:

Procedimiento puntosR para determinar las coordenadas en el hiper-rectángulo

```
> puntoR:=proc(Punto,Rectangulo) local i, a, b, NuevoPunto:
> NuevoPunto:=[]:
> for i from 1 to nops(Punto) do
> a:=op(1,Rectangulo[i]): b:=op(2,Rectangulo[i]):
> NuevoPunto:=[op(NuevoPunto),(b-a)*Punto[i]+a]:
> od:
> RETURN(NuevoPunto) end:
```

ListaPuntosR es el procedimiento que genera, a partir de una lista de puntos del cubo unitario, la lista con los puntos en el hipe-rectángulo

```
> ListaPuntosR:=proc(listaPuntos,Rectangulo) local NuevaLis-
ta, i:
> NuevaLista:=[]:
> for i from 1 to nops(listaPuntos) do
> NuevaLista:=[op(NuevaLista),puntoR(listaPuntos[i],Rectangulo)]:
> od:
> RETURN(NuevaLista):
> end:
```

Coordenada del punto medio del cubo unidad en R^n

> PuntoOrigen:=proc(Dimension) local Origen, i:
> Origen:=[]:for i from 1 to Dimension do
> Origen:=[op(Origen),0.5]:
> od:
> RETURN(Origen) end:

Nsaltos obtiene los números binarios del 0 al 2^{n-1}

> listaBinarios:=proc(Dimension) local nbinarios, i:
> nbinarios:=[]:
> for i from 0 to $2^{Dimension-1}$ **do**
> nbinarios:=[op(nbinarios),convert(i,binary)]:
> od:
> RETURN(nbinarios):
> end:

LPuntos obtiene todos los vertices de un cubo unidad en el espacio R^n *con vertice en el origen*

> verticesCuboU:=proc(Dimension) local lpuntos, nnbb, numero, vectnumero, i, j, long, aux:
> lpuntos:=[]:nnbb:=listaBinarios(Dimension):
> for j from 1 to nops(nnbb) do
> numero:=nnbb[j]; long:=length(numero); vectnumero:=[];
> for i from Dimension to 1 by -1 do
> if long<i then vectnumero:=[op(vectnumero),-0.5] else
> if floor(numero/10^{i-1}**)=0 then aux:=-0.5 else aux:=0.5 fi:**
> vectnumero:=[op(vectnumero),aux]:
> numero:=numero-floor(numero/10^{i-1}**)***10^{i-1}**:**
> fi: od:
> lpuntos:=[op(lpuntos),vectnumero]: od:
> RETURN(lpuntos): end:

Ncubo es el procedimiento que genera, a partir de un punto, la lista con todos los vertices del subcubo correspondiente al parámentro m (valor h)

> Ncubo:= proc(Punto,Valorh,Dimension) local listavertices, j, lpuntos;
> listavertices:=[]: lpuntos:=verticesCuboU(Dimension):
> for j from 1 to nops(lpuntos) do
> listavertices:=[op(listavertices),Punto+salto(Valorh)*lpuntos[j]]: od:
> ;RETURN(listavertices); end:

DepuraPuntos es el procedimiento que genera, a partir de una lista de puntos, una lista con los correspondientes valores de la función, determina el valor mínimo de esa lista, y descarta los puntos que no cumplen la condición adecuada

> DepuraPuntos:= proc(ListaPuntos,IVE,Valorextermo,ValorA,Valorh,Dimension)
local Tablavalores, mTablaValores, NuevaLista, decide, aproxdecide, valor, i:
> Tablavalores:=[]:
> for i from 1 to nops(ListaPuntos) do
> Tablavalores:=[op(Tablavalores),g(ListaPuntos[i])]
> od:
> if (IVE=1) then mTablaValores:=Valorextermo;
> else mTablaValores:=min(op(Tablavalores)); end if:
> NuevaLista:=[]:
> for i from 1 to nops(ListaPuntos) do
> decide:=Tablavalores[i]-mTablaValores-ValorA*diametro(Valorh,Dimension):
>aproxdecide:=evalf(decide):
> if aproxdecide<=0 then NuevaLista:=[op(NuevaLista),ListaPuntos[i]]:
> fi: od;
> RETURN(NuevaLista): end:

TroceCubos es el procedimiento que genera, a partir de una lista de puntos, la lista con todos los vertices de los subcubos correspondientes todos los puntos según el parámentro m

> TroceaCubos:=proc(ListaPuntos,Valorh,Dimension)
> local NuevaLista,i:
> NuevaLista:=[]:
> for i from 1 to nops(ListaPuntos) do
> NuevaLista:=[op(NuevaLista),op(Ncubo(ListaPuntos[i],Valorh,Dimension))]:
> od:
> RETURN(NuevaLista):end:

LimpiaPuntos es el procedimiento que elimina los puntos muy próximos, según la precisión epsilon, de una lista de puntos dejando uno de cada agrupación

> LimpiaPuntos:= proc(ListaPuntos) local LDiferencia, MDiferencia, NuevaLista, i, k, valor, tope :
> NuevaLista:=[ListaPuntos[1]];tope:=nops(ListaPuntos):
> for i from 1 to tope do
> valor:=0:

```
> for k from 1 to nops(NuevaLista) do
> LDiferencia:=ListaPuntos[i]-NuevaLista[k]:
> MDiferencia:=max(abs(max(op(LDiferencia))),abs(max(op(LDiferencia)))):
> if MDiferencia>=10*epsilon then valor:=valor+1: fi:
> od:
> if valor=nops(NuevaLista) then NuevaLista:=[op(NuevaLista),ListaPuntos[i]];
> fi; od:
> RETURN(NuevaLista): end:
```

Datos aportados por el usuario

Minimizar (d=1) o Maximizar (d=-1): (puede cambiar su valor)

```
> d:=-1:
```

SI(ive=1) o NO (ive=0)se impone valor extremo: (puede cambiar su valor)

```
> ive:=0: ValorExtremo:=0:
```

Error de aproximación:

```
> epsilon:=1/500:
```

Dimensión del espacio R^n: (puede cambiar su valor)

```
> n:=2:
```

Hiper-rectágulo donde se estudia el problema:([a1,b1]X[a2,b2]X...X[anbn] se expresa [[a1,b1],[a2,b2],...,[anbn]])

```
> HRectangulo:=[[0,1],[0,1]]:
```

Constante natural N de división (≥ 2) cada arista del cubo unitario

```
> N:=2:
```

Función de R^n a R que estudia el problema: (un punto x= (x[1],x[2],...,x[n]) en R^n)

```
> f:=proc(x)
```

>RETURN(min(((40000*x[1]+10000)*0.04* $1 + 0{,}04^{8*x[2]+1}$ **)/(**$1 + 0{,}04^{8*x[2]+1}$ **-1),9015.15)*0.15*(8*x[2]+1)-**
>(40000*x[1]+10000)*((8*x[2]+1)*0.04-1+(8*x[2]+1)*0.04/($1 + 0{,}04^{8*x[2]+1}$ **-1))); end;**

Una constante A >= Constante de Lipschitz para la función f:

> **A:=1:**

Constante que multiplica a la función (utíl en el algoritmo):(1, 0.1,... 0.01)

> **CB:=0.001:**

DATOS DE SALIDA

Los puntos no eliminados (pne=1) o los puntos epsilon-separados (pne=0): (puede cambiar su valor)

> **pne:=1:**

MODO GRÁFICO:Si (sg=1) o No (sg=0)se representa gráficamente: (puede cambiar su valor)

> **sg:=1:**

MODO NUMÉRICO:Si (sn=1) o No (sn=0)se muestra numéricamente, con coordenadas EXACTAS(ve=1) o APROXIMADAS (ve=0): (puede cambiar su valor)

> **sn:=1: ve:=0:**

SI (smM=1) o NO (smM=0) se muestra el valor extremo considerado: (puede cambiar su valor)

> **smM:=1:**

Control básico de los datos del usuario

Control para que el tipo de problema esté bien definido

> if (d=1) then d:=1: mensaje:="Problema de minimizar": else d:=-1: mensaje:="Problema de Maximizar": end if:

Función que se valora en esta pilas de instrucciones:

> g:=proc(x) RETURN(d*CB*f(puntoR(x,HRectangulo))) end:

Control para que la dimensión esté bien definida

> if (abs(floor(n))=0) then n:=1 else n:=abs(floor(n)) end if:

Control del hiper-rectángulo de R^n

> if (n $\neq$ nops(HRectangulo)) then RectanguloMalDefinido end if;

Control para asegurar que se puede representar gráficamente según la dimensión

> if ($n < 3$)and(sg=1) then sg:=1 else sg:=0 end if:

Control sobre la salida numérica de datos para que esté bien definida

> if (sn=1) then sn:=1 else sn:=0 end if:

Control sobre la salida comprobando que el tipo de problema está bien definido

> if (smM=1) then smM:=1 else smM:=0 end if:

Longitud de la arista del cubo unitario

> lado:=1:

Búsqueda de los mínimos globales por iteración en m

La variable m que indica el nivel de subdivisión del cubo

> h:=1: m:=0: Vertice:=PuntoOrigen(n): LPuntosVertices:=verticesCuboU(n):

Primer cubo

> h:=N*h:m:=m+1: ListaPuntos:=Ncubo(Vertice,h,n):

Listado con valores de los puntos minimos (cubic algorithm) hasta lado-Subcubo < epsilon

> while salto(h) > epsilon do
> ListaPuntos:=DepuraPuntos(ListaPuntos,ive,d*ValorExtremo,A,h,n):
PuntosMinimos:=ListaPuntos:
> h:=N*h: m:=m+1: ListaPuntos:=TroceaCubos(ListaPuntos,h,n):
od:
> PuntosMinimos:

Listado con valores aproximados de los puntos minimos globales epsilon separados

> PMinimosGlobal:=LimpiaPuntos(PuntosMinimos):

Salida de Datos

Selección de los puntos de salida

> if (pne=1)then PP2:=ListaPuntosR(PuntosMinimos,HRectangulo):
> else PP2:=ListaPuntosR(PMinimosGlobal,HRectangulo): end
if:
> PP1:=ListaPuntosR(LPuntosVertices,HRectangulo):
> if (n=1) then PP1Dos:=comoDos(PP1): PP2Dos:=comoDos(PP2):
> PP:=[PP1Dos,PP2Dos]:
> x1:=op(1,HRectangulo[1]): x2:=op(2,HRectangulo[1]):
> y1:=op(1,HRectangulo[1]): y2:=op(2,HRectangulo[1]): end
if:
> if (n=2) then PP:=[PP1,PP2]: x1:=op(1,HRectangulo[1]):
> x2:=op(2,HRectangulo[1]): y1:=op(1,HRectangulo[2]):
> y2:=op(2,HRectangulo[2]): end if:

Salida de los datos del proceso

> NivelDeSubdivisiones=m; LadoDelSubcubo=evalf(salto(h));
> NDePuntosNoEliminados=nops(PuntosMinimos);
> NDePuntosEpsilonSeparados=nops(PMinimosGlobal);

salida del valor mínimo o Máximo de la función

> if (smM=1)then ElValorExtremo=evalf(d*f(op(1,PMinimosGlobal)));
fi;

salida numérica de la solución

> if (sn=1) and (ve=1) then PP2;fi;

> if (sn=1) and (ve $\neq$ 1) then evalf(PP2);fi;

salida gráfica de la solución

> if (sg=1) and (n<3) then
> plot(PP, x=x1..x2, y=y1..y2,axes=BOXED, color=[red,blue],
style=point, symbol=[POINT,CROSS], title=mensaje); fi;

En los caso de R o R^2 se representa gráficamente la función.

Capítulo 4

Caso práctico I

Como primer ejemplo de utilización del algoritmo, vamos a aplicarlo a una función de interés matemático. La función elegida es

$$f(x,y) =\mid sen(0{,}5 + 9{,}5 \cdot \sqrt{x^2 + y^2}) \mid$$

En este caso, los datos aportados por el usuario son los siguientes:

Datos aportados por el usuario

Minimizar (d=1) o Maximizar (d=-1): (puede cambiar su valor)

> **d:=1:**

SI(ive=1) o NO (ive=0)se impone valor extremo: (puede cambiar su valor)

> **ive:=1: ValorExtremo:=0:**

Error de aproximación:

> **epsilon:=1/100:**

Dimensión del espacio R^n: (puede cambiar su valor)

> **n:=2:**

Hiper-rectágulo donde se estudia el problema:([a1,b1]X[a2,b2]X...X[anbn] se expresa [[a1,b1],[a2,b2],...,[anbn]])

> **HRectangulo:=[[0,2],[0,2]]:**

Constante natural N de división (≥ 2) cada arista del cubo unitario

> **N:=2:**

Función de R^n a R que estudia el problema: (un punto x= (x[1],x[2],...,x[n]) en R^n)

> **f:=proc(x)**
>**RETURN(0.1*abs(sin(0.5+9.5*(**$(x[1])^2 + (x[2])^2)^{1/2}$**))) end:**

Una constante A >= Constante de Lipschitz para la función f:

> **A:=2:**

Constante que multiplica a la función (útil en el algoritmo):(1, 0.1,... 0.01)

> **CB:=1:**

DATOS DE SALIDA

Los puntos no eliminados (pne=1) o los puntos epsilon-separados (pne=0): (puede cambiar su valor)

> **pne:=1:**

MODO GRÁFICO:Si (sg=1) o No (sg=0) se representa gráficamente: (puede cambiar su valor)

> **sg:=1:**

MODO NUMÉRICO:Si (sn=1) o No (sn=0)se muestra numéricamente, con coordenadas EXACTAS(ve=1) o APROXIMADAS (ve=0): (puede cambiar su valor)

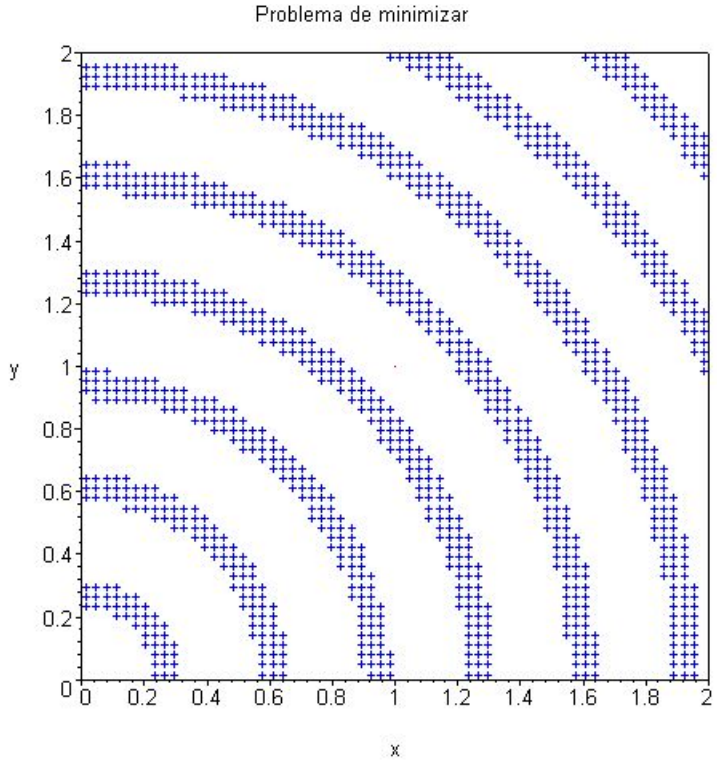
> **sn:=0: ve:=0:**

SI (smM=1) o NO (smM=0) se muestra el valor extremo considerado: (puede cambiar su valor)

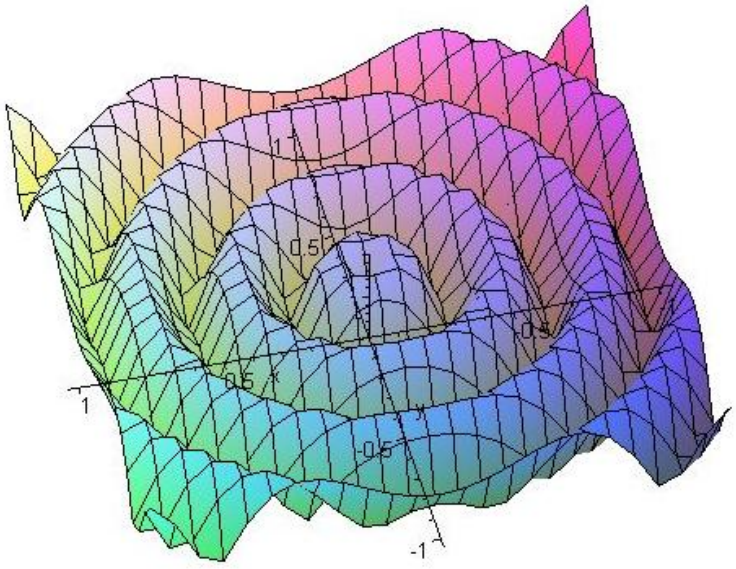
> **smM:=1:**

El resultado del algoritmo es el siguiente:

Nivel de subdivisiones = 7
NdePuntosNoEliminados = 1204
LadoDelSubcubo = 0.007812500000
NDePuntosEpsilonSeparados = 45
Valor extremo = 0.09989896430



Si dibujamos esta función en el cuadrado [-1,1]x[-1,1], podemos observar su comportamiento y verificar efectivamente que se corresponde con los máximos encontrados por el algoritmo anterior:

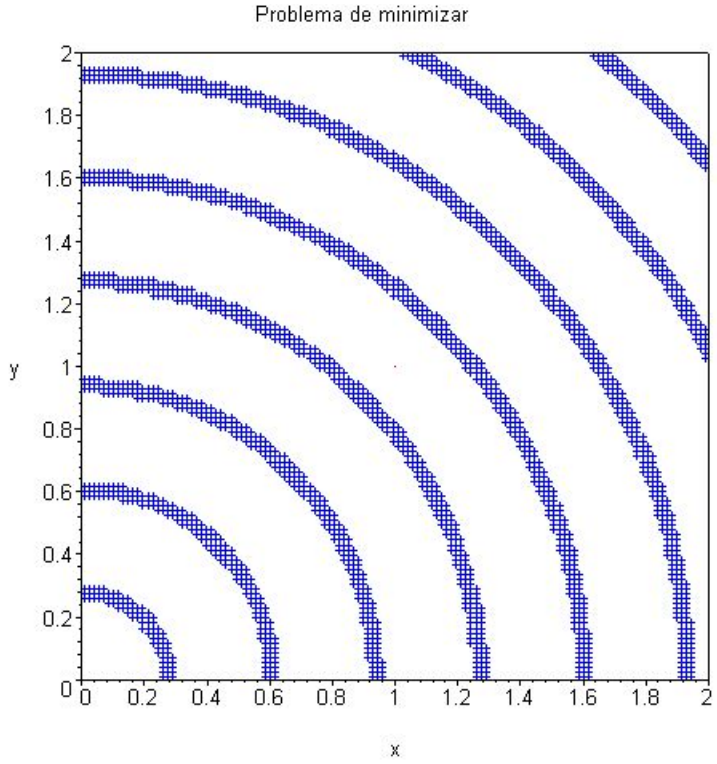


Dependiendo de la elección de la variable epsilon, obtendremos distintos gráficos con distintos tiempos de ejecución. En el ejemplo anterior, para epsilon = 1/100, el tiempo de ejecución es 2.1sg. Veamos diferentes resoluciones para otras elecciones de epsilon.

epsilon = 1/200

El tiempo de ejecución en este caso aumenta a 8.4sg, y el resultado del algoritmo es el siguiente:

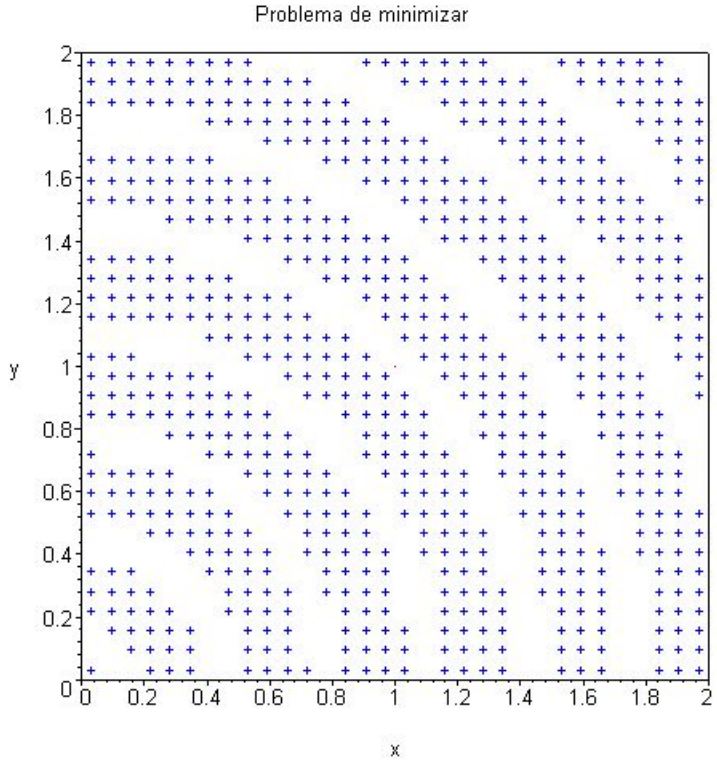
Nivel de subdivisiones = 8
NdePuntosNoEliminados = 2344
LadoDelSubcubo = 0.003906250000
NDePuntosEpsilonSeparados = 91
Valor extremo = 0.09887903680



epsilon = 1/50

Si ahora tomamos un epsilon mayor, tenemos un menor tiempo de ejecución (1.4sg), pero también se pierde precisión. El resultado del algoritmo queda así:

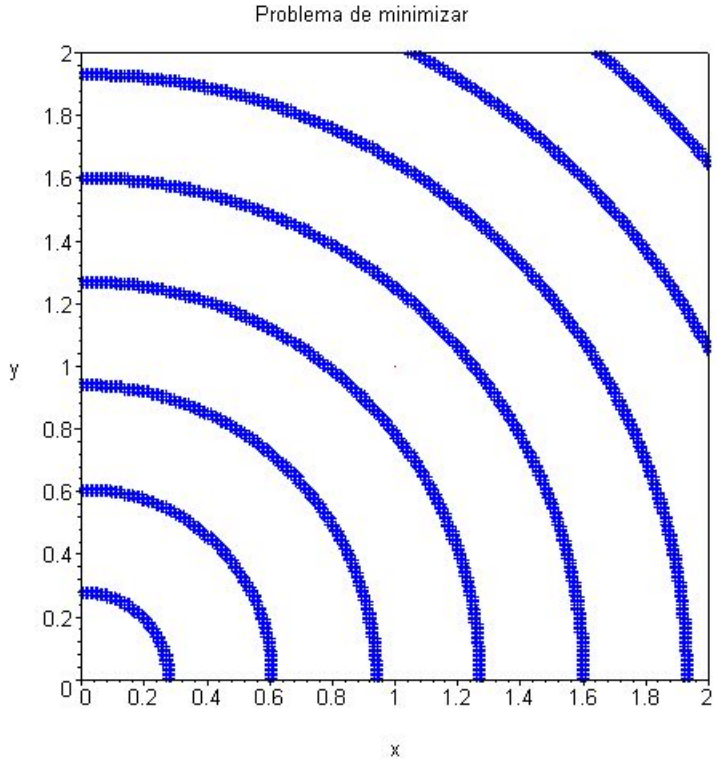
Nivel de subdivisiones = 6
NdePuntosNoEliminados = 707
LadoDelSubcubo = 0.01562500000
NDePuntosEpsilonSeparados = 23
Valor extremo = 0.06517748638



epsilon = 1/500

Finalmente, para un epsilon mucho menor (1/500), el tiempo de ejecución es 23sg pero la precisión es mucho mayor. Este es el resultado del algoritmo en este caso:

Nivel de subdivisiones = 9
NdePuntosNoEliminados = 4692
LadoDelSubcubo = 0.001953125000
NDePuntosEpsilonSeparados = 200
Valor extremo = 0.09798717540


Problema de minimizar
y
x
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

Capítulo 5

Caso práctico II

En este apartado vamos a aplicar el algoritmo implementado anteriormente a una función de interés financiero, relacionada con el interés prestatario y la devolución fiscal de Hacienda.

Explicación y desarrollo de la función

Supongamos que queremos pedir un préstamo financiero de cuota fija a n años por un capital C. Para la función que vamos a desarrollar se ha tomado un préstamo a interés fijo del 4 % y con vencimiento anual (igualmente se podría tomar una función con vencimiento mensual, o con un interés diferente al 4 %). Pongamos también que nuestros límites en cuanto a plazo e importe son C $\in$ [10.000, 50.000] euros y n $\in$ [1, 9] años.

En este tipo de préstamos, Hacienda devuelve el 15 % del importe pagado anualmente, hasta un máximo de importe pagado de 9.015,15 euros (si se paga más de este importe, Hacienda devuelve el 15 % de 9.015,15 euros).

Se quiere estudiar cuál sería el importe y plazos óptimos para pedir este préstamo minimizando las pérdidas (o maximizando las ganancias). Intuitivamente, si sólo se tuviera en cuenta el préstamo, sin las devoluciones de Hacienda, parece normal que el valor óptimo es pedirlo a los menos años posibles y el menor capital posible, para pagar los menos intereses posibles. Sin embargo, al contar con las devoluciones de Hacienda, se convierte en un problema más complejo.

Lo primero de todo es modelizar la función matemáticamente.

En los préstamos de cuota fija, la cuota viene dada por

$$a = \frac{C}{\frac{1-(1+i)^{-n}}{i}} = \frac{C \cdot i \cdot (1+i)^n}{(1+i)^n - 1},$$

el capital en el año s es

$$C_s = a \cdot \frac{1-(1+i)^{-n+s}}{i}$$

y el interés pagado en el año s es

$$I_s = C_{s-1} \cdot i = a \cdot \frac{1-(1+i)^{-n+s+1}}{i} \cdot i.$$

La suma total de los intereses pagados durante los n años del préstamo se calcula de la siguiente forma:

$$\sum_{s=1}^{n} I_s = \sum_{s=1}^{n} C_{s-1} \cdot i = i \cdot \sum_{s=1}^{n} C_{s-1} = i \cdot \sum_{s=1}^{n} a \cdot \frac{1-(1+i)^{-n+s+1}}{i} = a \cdot \sum_{s=1}^{n} 1-(1+i)^{-n+s+1} =$$

$$= a \cdot n - \frac{a}{(1+i)^{n+1}} \cdot \sum_{s=1}^{n} (1+i)^s = a \cdot n - \frac{a}{i} \cdot \Big(1 - \frac{1}{(1+i)^n}\Big) = a \cdot \Big(n - \frac{1}{i} + \frac{1}{i(1+i)^n}\Big)$$

Si sustituimos a por su valor y simplificamos, nos queda:

$$I_{total} = C \cdot \Big(n \cdot i - 1 + \frac{n \cdot i}{(1+i)^n - 1}\Big)$$

Por otra parte, la devolución total de Hacienda en n años será la siguiente:

$$D_{total} = min(a, 9015, 15) \cdot 0,15 \cdot n = min\Big(\frac{C \cdot i \cdot (1+i)^n}{(1+i)^n - 1}, 9015, 15\Big) \cdot 0,15 \cdot n$$

Para calcular la función a optimizar unimos las dos partes: la función que hay que maximizar es la devolución total de Hacienda menos los intereses abonados, es decir:

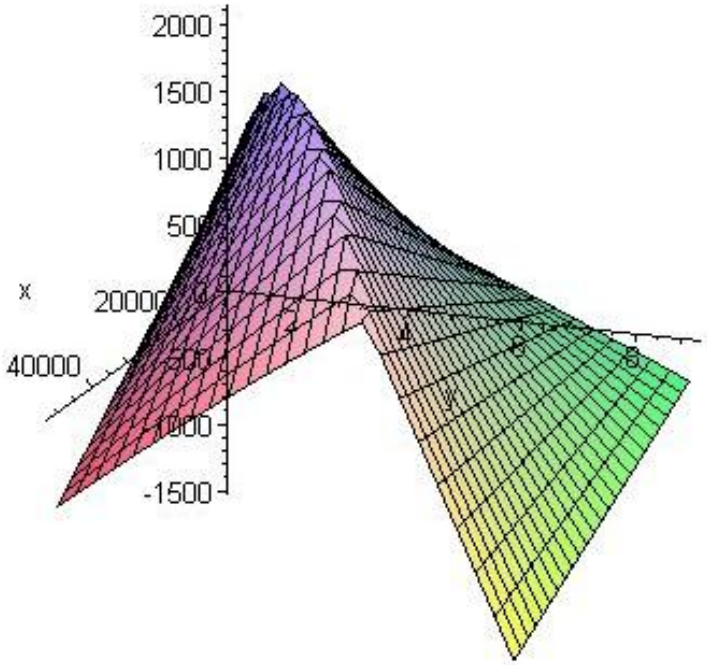
$$Z \equiv max\left(min\Big(\frac{C \cdot i \cdot (1+i)^n}{(1+i)^n - 1}, 9015, 15\Big) \cdot 0,15 \cdot n - C \cdot \Big(n \cdot i - 1 + \frac{n \cdot i}{(1+i)^n - 1}\Big)\right)$$

Y en nuestro caso concreto en el que $i = 4\,\%$ queda así:

$$Z \equiv max\left(min\Big(\frac{C \cdot 0,04 \cdot 1,04^n}{1,04^n - 1}, 9015, 15\Big) \cdot 0,15 \cdot n - C \cdot \Big(n \cdot 0,04 - 1 + \frac{n \cdot 0,04}{1,04^n - 1}\Big)\right),$$

con las variables C $\in$ [10.000, 50.000] y n $\in$ [1, 9].

La función tridimensional queda como sigue:



Aunque las variables de nuestro problema son discretas (podemos tomar el capital hasta con 2 decimales, y el número de años en fracciones de 12 meses), podemos utlizar el algoritmo Cubic, y con los puntos que nos salgan tomar las mejores aproximaciones. Este estudio se realiza para poder tomar la opción correcta o más favorable, aunque se trate como una función de variables contínuas y no discretas, a la hora de elegir se toma la opción discreta más cercana.

Para la función que nos ocupa, los datos introducidos en el problema quedarían así:

Datos aportados por el usuario

Minimizar (d=1) o Maximizar (d=-1): (puede cambiar su valor)

> **d:=-1:**

SI(ive=1) o NO (ive=0)se impone valor extremo: (puede cambiar su valor)

> **ive:=0: ValorExtremo:=0:**

Error de aproximación:

> **epsilon:=1/100:**

Dimensión del espacio R^n: (puede cambiar su valor)

> **n:=2:**

Hiper-rectágulo donde se estudia el problema:([a1,b1]X[a2,b2]X...X[anbn] se expresa [[a1,b1],[a2,b2],...,[anbn]])

> **HRectangulo:=[[10000,50000],[1,9]]:**

Constante natural N de división (≥ 2) cada arista del cubo unitario

> **N:=2:**

Función de R^n a R que estudia el problema: (un punto x= (x[1],x[2],...,x[n]) en R^n)

> **f:=proc(x)**
>**RETURN(min((x[1]*0.04*$1 + 0{,}04^{x[2]}$)/($1 + 0{,}04^{x[2]}$-1),9015.15)*0.15*x[2]-**

>**x[1]*(x[2]*0.04-1+x[2]*0.04/($1 + 0{,}04^{x[2]}$-1))); end;**

Una constante A >= Constante de Lipschitz para la función f:

> **A:=1:**

Constante que multiplica a la función (utíl en el algoritmo):(1, 0.1,... 0.01)

> **CB:=1/2100:**

DATOS DE SALIDA

Los puntos no eliminados (pne=1) o los puntos epsilon-separados (pne=0): (puede cambiar su valor)

> **pne:=1:**

MODO GRÁFICO:Si (sg=1) o No (sg=0)se representa gráficamente:

(puede cambiar su valor)

> **sg:=1:**

MODO NUMÉRICO:Si (sn=1) o No (sn=0)se muestra numéricamente, con coordenadas EXACTAS(ve=1) o APROXIMADAS (ve=0): (puede cambiar su valor)

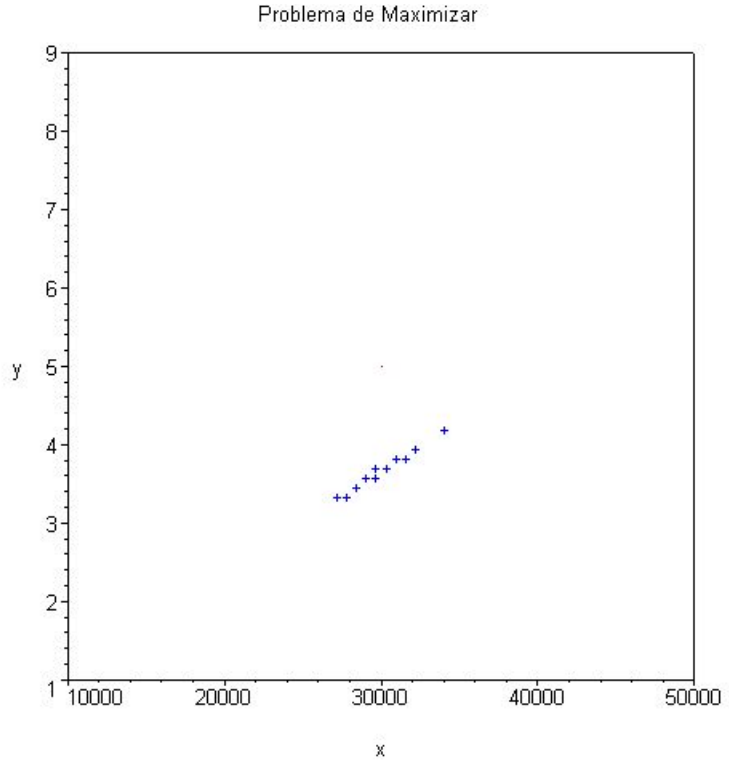
> **sn:=1: ve:=1:**

SI (smM=1) o NO (smM=0) se muestra el valor extremo considerado: (puede cambiar su valor)

> **smM:=1:**

El resultado del algoritmo es el siguiente:

Nivel de subdivisiones = 7
NDePuntosNoEliminados = 11
LadoDelSubcubo = 0.007812500000
NDePuntosEpsilonSeparados = 2
Valor extremo = -2.088,928462
[[27187.50000, 3.312500000], [27812.50000, 3.312500000], [28437.50000, 3.437500000], [29062.50000, 3.562500000], [29687.50000, 3.562500000], [29687.50000, 3.687500000], [30312.50000, 3.687500000], [30937.50000, 3.812500000], [31562.50000, 3.812500000], [32187.50000, 3.937500000], [34062.50000, 4.187500000]]



A la vista de estos resultados, se ve que el ahorro que podemos llegar a tener es de alrededor de 2.100 euros (en la totalidad de años que dura el préstamo). Como se ha dicho antes, al ser estos valores contínuos, hay que pasarlos a valores con los que podamos trabajar: el capital con dos decimales, y el periodo a años y meses. El primer valor, correspondiente al capital, ya tiene el formato correcto, ya que todos los valores del resultado tienen como mucho un decimal. Para formatear el segundo valor se ha implementado el siguiente programa en Maple, que toma el valor año-mes inmediatamente anterior y siguiente a los puntos escogidos:

```
> Meses:=[]:for aux from 1 to nops(PP2) do
> aux2:=op(aux,PP2):
> x2:=op(2,aux2):
> anyo:=floor(x2):
> mes:=floor((x2-anyo)*12);anyoAprox:=anyo+mes/12:
> Meses:=[op(Meses),[op(1,aux2),anyoAprox],[op(1,aux2),anyoAprox+1]]:
> od: Meses;
```

[[27187.50000, 13/4], [27187.50000, 17/4], [27812.50000, 13/4],
[27812.50000, 17/4], [28437.50000, 41/12], [28437.50000, 53/12],

[29062.50000, 7/2], [29062.50000, 9/2], [29687.50000, 7/2], [29687.50000, 9/2], [29687.50000, 11/3], [29687.50000, 14/3], [30312.50000, 11/3], [30312.50000, 14/3], [30937.50000, 15/4], [30937.50000, 19/4], [31562.50000, 15/4], [31562.50000, 19/4], [32187.50000, 47/12], [32187.50000, 59/12], [34062.50000, 25/6], [34062.50000, 31/6]]

```
> ValorOptimo:=evalf(f(op(1,Meses))):
> PuntoOptimo:=op(1,Meses):
> for i from 1 to nops(Meses) do
> fun:=evalf(f(op(i,Meses))):
> if fun>ValorOptimo then ValorOptimo:=fun:
> PuntoOptimo:=op(i,Meses): fi:
> od: ElValorOptimo=ValorOptimo; CapitalOptimo=op(1,PuntoOptimo);
> AnyosOptimos=floor(op(2,PuntoOptimo));
> MesesOptimos=12*(op(2,PuntoOptimo)-floor(op(2,PuntoOptimo)));
```

$$ElValorOptimo = 2079.867694$$
$$CapitalOptimo = 30312.50000$$
$$AnyosOptimos = 3$$
$$MesesOptimos = 8$$

La aplicación del algoritmo Cubic a nuestro problema lleva al resultado siguiente: si pedimos un capital de 30312.50 euros a un plazo de 3 años y 8 meses, el ahorro total es de 2079.87 euros.

Al igual que en el ejemplo anterior, se puede tomar un epsilon mayor o menor para aumentar o disminuir la precisión del algoritmo (asumiendo un mayor tiempo de ejecución cuanta más precisión se busque).