Métodos Numéricos em Otimização

EP Número 1

Victor Alberto Romero, NUSP: 8405274

12 de setembro de 2013

1 Problema

O problema é programar o método da maior decida usando busca linear, e com ele minimizar a função de Rosenbrock. É preciso usar como passo inicial $\alpha_0 = 1$ e imprimir a longitude do passo usado pelo método em cada iteração. Primeiro temos que tentar como um ponto inicial $x_0 = (1.2, 1.2)$ e depois como um ponto um pouco mais difícil, $x_0 = (-1.2, 1)$.

2 Função

A função de Rosenbrock está definida como:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Esta função é continuamente derivável, com um gradiente igual a:

$$\nabla f(x_1, x_2) \quad = \quad \begin{bmatrix} 400x_1(x_1^2 - x_2) + 2(x_1 - 1) \\ 200(x_2 - x_1^2) \end{bmatrix}$$

3 Considerações

3.1 Constantes

As constantes usadas são as seguintes:

Constante	**Valor**	**Razão**
γ	10^{-4}	Recomendado pelo professor
M	2	Recomendado pelo professor
ϵ	10^{-10}	ϵ é o valor usado para definir que algo é zero. Se toma como valor 10^{-10} porque esse valor é considerado suficientemente pequeno em comparação com as outras constantes.

3.2 Condições de parada

Como condições de parada se usaram as seguintes:

- Se evalua quando a norma dois do gradiente da função é zero (para evitar error numéricos, se usa ϵ, como pode-se ver no código no anexo). Quando esto ocorre, assumimos que o x^k está perto o suficiente do mínimo local.
- Também se tem uma condição sobre o número máximo de iterações permitidas para o algoritmo. Isto é com o fim de evitar que o programa fique calculando e não pare num tempo razoável quando exista um ponto que converge muito devagar à solução.

4 Resultados

4.1 Resultados exercicios propostos

O programa foi executado com os dois pontos pedidos. Com o ponto $x_1 = (1.2, 1.2)$ obtive-se o minimo com 24370 iterações, com o ponto $x_2 = (-1.2, 1)$ pode-se obter o mínimo depois de 25116 iterações. Os valores de de α válidos para algumas iterações podem-se ver nas tabelas a seguir.

Iter. x_1	α_1^k	Iter. x_1	α_1^k
0	0,000977	12500	0,001953
500	0,015625	13000	0,001953
1000	0,001953	13500	0,001953
1500	0,001953	14000	0,003906
2000	0,001953	14500	0,001953
2500	0,001953	15000	0,001953
3000	0,001953	15500	0,001953
3500	0,001953	16000	0,001953
4000	0,001953	16500	0,001953
4500	0,001953	17000	0,001953
5000	0,001953	17500	0,001953
5500	0,003906	18000	0,001953
6000	0,001953	18500	0,001953
6500	0,001953	19000	0,001953
7000	0,001953	19500	0,001953
7500	0,001953	20000	0,001953
8000	0,001953	20500	0,001953
8500	0,001953	21000	0,001953
9000	0,001953	21500	0,001953
9500	0,001953	22000	0,001953
10000	0,001953	22500	0,001953
10500	0,003906	23000	0,001953
11000	0,001953	23500	0,001953
11500	0,001953	24000	0,001953
12000	0,001953	**Média**	0,002181

Iter. x_2	α_2^k	Iter. x_2	α_2^k
0	0,000977	12500	0,001953
500	0,003906	13000	0,001953
1000	0,001953	13500	0,001953
1500	0,001953	14000	0,001953
2000	0,001953	14500	0,001953
2500	0,003906	15000	0,001953
3000	0,001953	15500	0,001953
3500	0,001953	16000	0,001953
4000	0,001953	16500	0,001953
4500	0,001953	17000	0,001953
5000	0,001953	17500	0,001953
5500	0,001953	18000	0,001953
6000	0,001953	18500	0,001953
6500	0,001953	19000	0,001953
7000	0,001953	19500	0,001953
7500	0,001953	20000	0,001953
8000	0,001953	20500	0,001953
8500	0,001953	21000	0,001953
9000	0,001953	21500	0,001953
9500	0,001953	22000	0,001953
10000	0,001953	22500	0,001953
10500	0,001953	23000	0,001953
11000	0,001953	23500	0,001953
11500	0,001953	24000	0,003906
12000	0,001953	**Média**	0,002066

Nota-se que o α válido para as iterações foi praticamente o mesmo, o que leva a pensar que a velocidade com que o método se aproxima à solução é lineal.

4.2 Outros Resultados

O programa também foi executado com um número mais amplio de provas (nos intervalos de $(-50, 50)$ com um passo de 0.5 para as duas variáveis). Nestes casos foi possível observar que a medida que o ponto inicial afastava-se do ótimo global, o algoritmo precisava mais iterações para chegar numa solução. Também foi possível ver que dita mudança no número de iterações podia ser muito diferente inclusive para nós próximos. Por exemplo temos:

Ponto	**Iterações**
$(-48, -40)$	25586
$(-47.5, -39)$	487839968
$(-46, -37)$	24862

5 Conclusões

- O método de otimização lineal, pegando como direção o menos gradiente (máxima decida) funciona na prática, entregando um mínimo local.
- O método de máxima decida é lento, precisando muitas iterações inclusive para pontos simples.
- Dadas as aproximações naturais da representação dos números decimais no computador, uma boa prática é usar um ϵ para o qual, qualquer valor menor do que ele é considerado como zero.

Anexo

O código implementado pode-se ver a continuação:

```
1  /*
2   * Victor Alberto Romero
3   * Métodos numéricos em otimização
4   * 12 de Setembro de 2013
5   * IME - USP
6   */
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10
11 /*
12  * Global variables
13  */
14 double M = 2.0;
15 double gama = 0.0001;
16 double epsilon = 0.0000000001; /* 10^(-10) its practicaly 0*/
17
18 /*
19  * Objective function: Rosenbrock function
20  * f(x1,x2) = 100(x2-x1^2)^2 + (1-x1)^2
21  * x* = (1,1)^T is the only local minimizator
22  */
23 void Rosenbrock(double* xk, double* output)
24 {
25     *output = 100*(xk[1]-xk[0]*xk[0])*(xk[1]-xk[0]*xk[0]) + (1-xk[0])*(1-xk[0]);
26 }
27
28 /*
```

```
29  * First derivate of the Rosenbrock function in R2
30  * f'(x1,x2) = [400*x1^3 -400*x2*x1 +2*x1 - 2 ; 200(x2 - x1^2) ]
31 */
32 void RosenbrockD(double* xk, double* output)
33 {
34     output[0] = 400*xk[0]*xk[0]*xk[0] - 400*xk[1]*xk[0] + 2*xk[0] -2;
35     output[1] = 200*(xk[1]-xk[0]*xk[0]);
36 }
37
38 /*
39  * Function that return the minimum valor with linear search
40 */
41 void returnMin(double* xk)
42 {
43     double* d      = (double*)malloc(2*sizeof(double));
44     double* fD     = (double*)malloc(2*sizeof(double));
45     double* xkNew = (double*)malloc(2*sizeof(double));
46     double f, fNew, alfa;
47     int iterations = 0;
48
49     while(iterations < 1000000000) /* Stop by iterations */
50     {
51         /* Basic parameters */
52         alfa = 1;
53         Rosenbrock(xk, &f);
54         RosenbrockD(xk, fD);
55
56         /* Calculate the gradient and multiply by -1 */
57         d[0]=(-1)*fD[0];
58         d[1]=(-1)*fD[1];
59
60         /* Check stop conditions: The two norm */
61         if( sqrt(d[0]*d[0] + d[1]*d[1]) < epsilon )
62             break;
63
64         /* Calculate a possible new xk */
65         xkNew[0]= xk[0]+alfa*d[0];
66         xkNew[1]= xk[1]+alfa*d[1];
67         Rosenbrock(xkNew, &fNew);
68
69         /* Calculate a new alfa */
70         while( fNew > ( f + gama*alfa*(fD[0]*d[0] + fD[1]*d[1])  ))
71         {
72             alfa = alfa / M;
73             xkNew[0]= xk[0]+alfa*d[0];
74             xkNew[1]= xk[1]+alfa*d[1];
75             Rosenbrock(xkNew, &fNew);
76         }
77
78         /* Update xk */
79         xk[0]=xkNew[0];
80         xk[1]=xkNew[1];
81
82         iterations++;
83     }
84
85     /* Print the result */
86     printf("Point (%f,%f) Iterations:%d\n",xk[0],xk[1],iterations);
87
88     /* Memory free */
89     free(fD);
```

```
90      free(d);
91      free(xkNew);
92  }
93
94  int main(void)
95  {
96      double i,j;
97      double limit = 50;
98      double step = 0.5;
99      double* x0 = (double*)malloc(2*sizeof(double));
100
101     for (i = -limit; i <= limit; i+=step)
102     {
103         for (j = -limit; j <= limit; j+=step)
104         {
105             x0[0]=i;
106             x0[1]=j;
107             printf("x0=(%f,%f) ",x0[0],x0[1]);
108             returnMin(x0);
109         }
110     }
111     return 0;
112 }
```