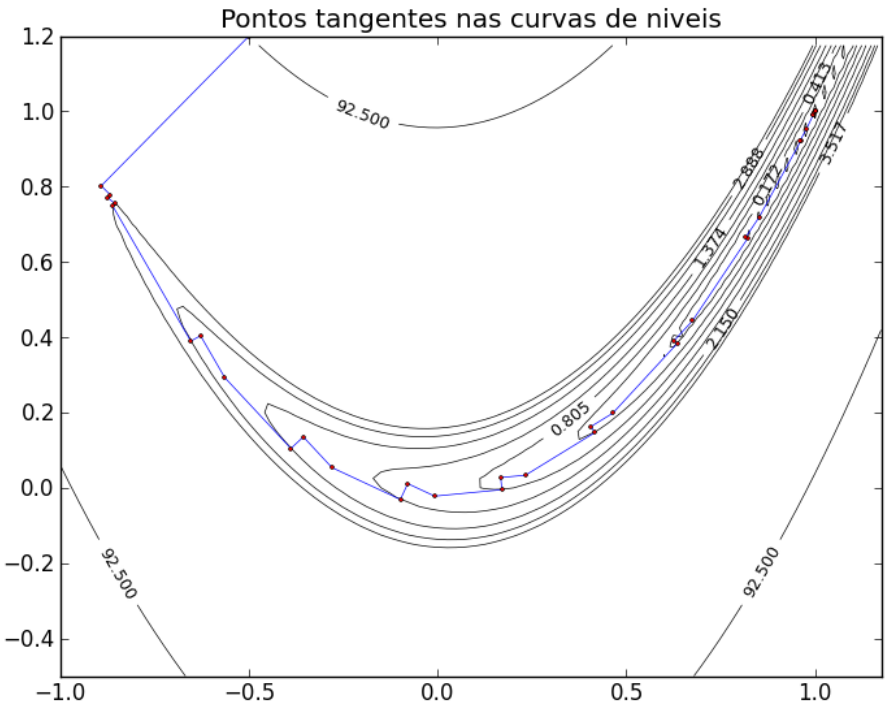
Laboratório de Computação e Simulação

Arthur Tsuyoshi Sakano Vetori - 7990450
Victor Shin Kinoshita - 5947051

23/05/2013



1 Objetivo

Criar um algoritmo que encontra o mínimo de uma função com uma restrição $h(x) = 0$ usando busca linear por ajuste quadrático, ParTan e método da penalidade.

2 Desenvolvimento

Por padrão, usamos orientação a objetos para completarmos o objetivo. Usamos também o GitHub para versionar nosso programa e para ajudar no desenvolvimento em dupla. Para acessar o projeto completo, entre na página `https://github.com/vkinoshita/ep5.git`

Como estrutura do projeto, temos:

1. ep5.py - programa principal. Ele que vai consumir todos os objetos para encontrar o mínimo da função.
2. entrada.py - os valores de entrada que implementa a dimensão, a função, o gradiente, a restrição, o ponto inicial e o erro.
3. Pasta lib:
   (a) grg.py - possui o algoritmo que encontra o gradiente reduzido generalizado
   (b) busca_linear.py - responsável por encontrar o mínimo de uma função de acordo com uma direção dada. Ele implementa o ajuste quadrático como recurso pra encontrar o mínimo.
   (c) partan.py - possui o algoritmo que encontra o mínimo da função.
   (d) penalidade.py - implementa as funções do método da penalidade.
   (e) saida.py - escreve o arquivo de saída.
4. Pasta testes - possui todos os testes realizados durante o desenvolvimento do programa. Para executá-los, basta jogar todos os arquivos na mesma pasta que o ep5.py.

Para realizarmos os testes, tomamos como referência os seguintes parâmetros com a seguinte função (Figura 1):

```
d = 3
m = 1
x0 = [1, 3, 10]
eps = 0.001

def f(x):
fx = x[0]**2 + 2*(x[1]**2) + 3*(x[2]**2)
return(fx)

def g(x):
gx = [2*x[0], 4*x[1], 6*x[2]]
return(gx)

def h(x):
hx = [x[0]**2 +x[1] -5*x[2]]
return(hx)

def j(x):
jx = [[2*x[0], 1, -5]]
return(jx)
```

3 Sobre o Algoritmo

O programa basicamente realiza as seguintes etapas:

1. Leitura do arquivo de entrada.
2. Busca do valor mínimo pelo ParTan usando a busca linear por ajuste quadrático.
3. Escrita do arquivo de saída

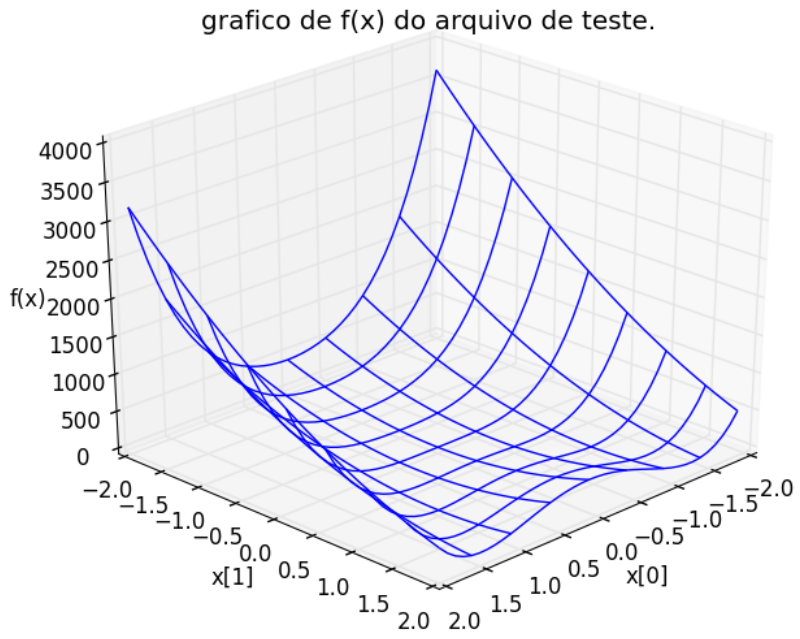


Figura 1: Função usada para o teste

4 Busca Linear

A busca linear consiste na influência de um elemento sobre o próximo. Para a etapa k, uma posição x_k e uma certa direção d, resumidamente temos que

$$x_{k+1} = x_k + \eta_k d \tag{1}$$

Assim, temos uma função para cada posição de η

$$g(\eta) = f(x_k + \eta d) \tag{2}$$

Entre os diversos tipos de busca linear, o que muda, é a forma em que o x_{k+1} é encontrado.

4.1 Ajuste Quadrático

O ajuste quadrático consiste em interpolar 3 pontos previamente escolhidos para uma função quadrática. Durante as etapas, o algoritmo realiza perturbações nesses 3 pontos, sempre tentando encontrar o menor valor.

Em outras palavras, inicialmente são "chutados"os pontos $\{\eta_1, \eta_2, \eta_3\}$, com seus respectivos valores

$$\begin{cases} \phi_1 = g(\eta_1) = f(x_0 + \eta_1 d) \\ \phi_2 = g(\eta_2) = f(x_0 + \eta_2 d) \\ \phi_3 = g(\eta_3) = f(x_0 + \eta_3 d) \end{cases}$$

Satisfazendo as seguintes condições

$$\eta_1 < \eta_2 < \eta_3 \tag{3}$$

$$\phi_1 \geq \phi_2 \leq \phi_3 \tag{4}$$

os pontos ficam prontos para serem usados na interpolação da função quadrática.

4.2 Interpolação

A função será dada por

$$\phi(\eta) = a\eta^2 + b\eta + c \tag{5}$$

Como já temos $\{\eta_1, \eta_2, \eta_3\}$ e $\{\phi_1, \phi_2, \phi_3\}$, manipulando algebricamente, encontramos que

$$a = \frac{(\eta_2 - \eta_3)\phi_1 + (\eta_3 - \eta_1)\phi_2 + (\eta_1 - \eta_2)\phi_1}{-(\eta_2 - \eta_1)(\eta_3 - \eta_2)(\eta_3 - \eta_1)} \tag{6}$$

$$b = \frac{(\eta_3^2 - \eta_2^2)\phi_1 + (\eta_1^2 - \eta_3^2)\phi_2 + (\eta_2^2 - \eta_1^2)\phi_1}{-(\eta_2 - \eta_1)(\eta_3 - \eta_2)(\eta_3 - \eta_1)} \tag{7}$$

$$c = \frac{(\eta_3\eta_2^2 - \eta_2\eta_3^2)\phi_1 + (\eta_1\eta_3^2 - \eta_3\eta_1^2)\phi_2 + (\eta_2\eta_1^2 - \eta_1\eta_2^2)\phi_1}{-(\eta_2 - \eta_1)(\eta_3 - \eta_2)(\eta_3 - \eta_1)} \tag{8}$$

Agora, basta encontrar o η mínimo, tal que $g'(\eta) = 0$. Como é uma função de segundo grau, facilmente, temos que o ponto mais baixo é dado por

$$\eta_{min} = -\frac{b}{2a} \tag{9}$$

4.3 Trinca inicial

Para que a busca funcione, é fundamental encontrar os ηs iniciais. Fizemos então, uma trinca inicial $\{-0.0005, 0, 0.0005\}$ e moldamos ele ao ponto de satisfazer $\eta_1 < \eta_2 < \eta_3$ e $\phi_1 \geq \phi_2 \leq \phi_3$.

Foi feito, para um $\alpha > 1$:

1. Enquanto $\phi_1 < \phi_2$, η_1 recebe $\eta_1\alpha$.
2. Enquanto $\phi_3 < \phi_2$, η_3 recebe $\eta_3\alpha$.

Assim, multiplicamos cada uma das pontas por uma constante até que os pontos fiquem bons.

4.4 Perturbações

As perturbações são sempre baseadas no valor encontrado por (9).

Como a meta do algoritmo é encontrar o mínimo de uma função, à partir do momento que temos η_{min} e $\phi_{min} = g(\eta_{min})$, geramos novas trincas $\{\eta_1, \eta_2, \eta_3\}$ e $\{\phi_1, \phi_2, \phi_3\}$, sempre satisfazendo as condições (3) e (4). Ou seja,

1. Se $\eta_{min} > \eta_2$:

(a) Se $\phi_{min} > \phi_2$, escolhemos as trincas novas $\{\eta_1, \eta_2, \eta_{min}\}$ e $\{\phi_1, \phi_2, \phi_{min}\}$.

(b) caso contrário, escolhemos as trincas novas $\{\eta_2, \eta_{min}, \eta_3\}$ e $\{\phi_2, \phi_{min}, \phi_3\}$.

2. Se $\eta_{min} \leq \eta_2$:

(a) Se $\phi_{min} > \phi_2$, escolhemos as trincas novas $\{\eta_{min}, \eta_2, \eta_3\}$ e $\{\phi_{min}, \phi_2, \phi_3\}$.

(b) caso contrário, escolhemos as trincas novas $\{\eta_1, \eta_{min}, \eta_2\}$ e $\{\phi_1, \phi_{min}, \phi_2\}$.

Observe que as trocas sempre priorizam as condições (3) e (4) e, ao mesmo tempo, adicionam um ponto menor sem denegrir a função original $g(\eta)$. Ou seja, ele sempre vai em direção ao ponto de menor valor.

4.5 Erro

Para garantir a qualidade do resultado, foi estipulado a seguinte condição de erro:

$$|\eta_{min} - \eta_2| < \epsilon \tag{10}$$

Ou seja, se as perturbações começarem a gerar pontos pouco distantes (menores que ϵ), garantimos então que η não irá variar mais que ϵ.

No programa, como foi observado comportamentos muito indesejáveis em pontos distantes, foi adotado um ϵ = 0.0000001 para que ele consiga resultados consistentes no partan. Não foi usado o ϵ de entrada do programa (entrada.py) com o propósito de que o algoritmo aceite o máximo de pontos possíveis, reservando esse erro apenas para o ParTan, que será descrito na próxima seção.

4.6 Algoritmo da Busca Linear por Ajuste Quadrático

Juntando todos os conceitos anteriores, temos o algoritmo:

1. Encontrar as trincas iniciais $\{\eta_1, \eta_2, \eta_3\}$ e $\{\phi_1, \phi_2, \phi_3\}$.

2. Gerar η_{min} e ϕ_{min} à partir da trinca inicial.

3. Se $|\eta_{min} - \eta_2| < \epsilon$, retornar η_{min}, caso contrário:

(a) Atualizar a trinca com η_{min} e ϕ_{min}. (Dado na seção "Perturbações").

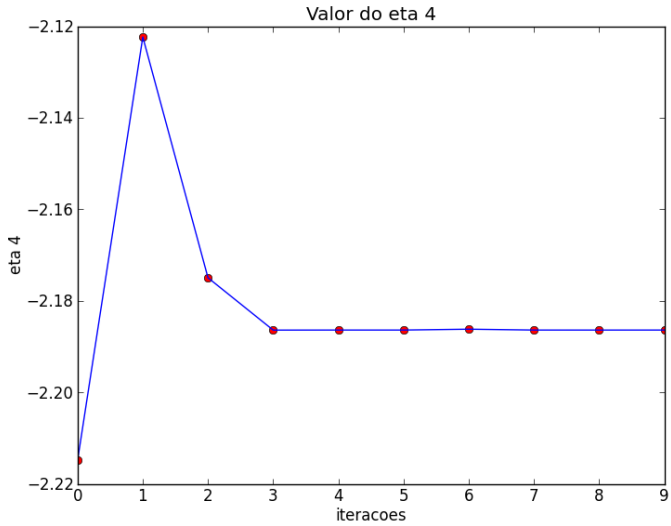
(b) Gerar η_{min} e ϕ_{min} à partir da trinca atual.

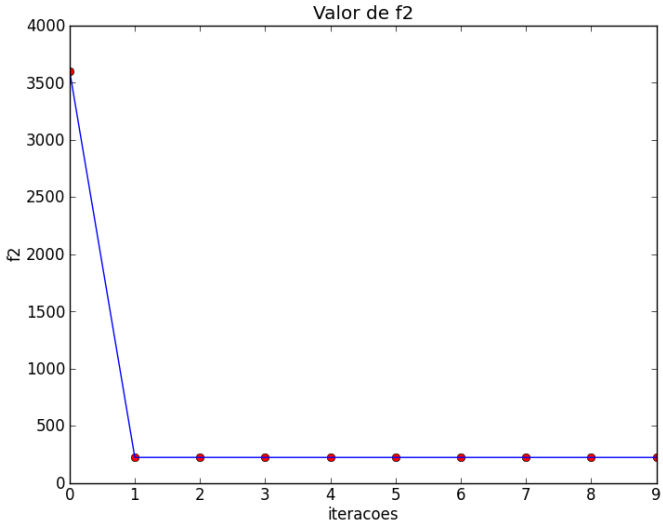
(c) Voltar para o item 3.

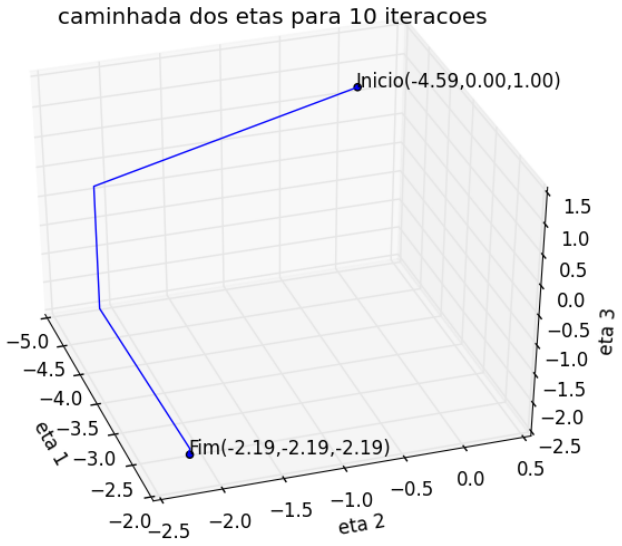
4.7 Resultados

Como esperado, o algoritmo encontra o mínimo de forma consistente. Sendo que, quanto mais o ponto está longe do mínimo, mais ele demora pra chegar. Como no caso do [2, -2] que precisou de menos de 10 iterações pra convergir e no caso do [100, 100] que precisou de muito mais pontos para estabilizar.

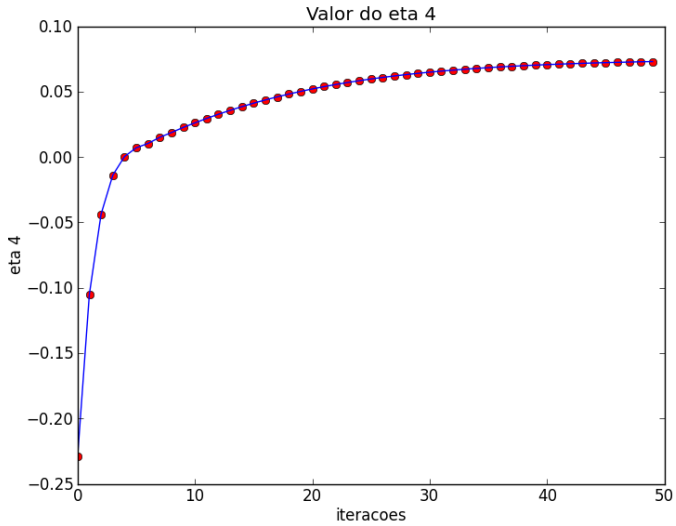
4.7.1 Início em [2, -2] para 10 iterações

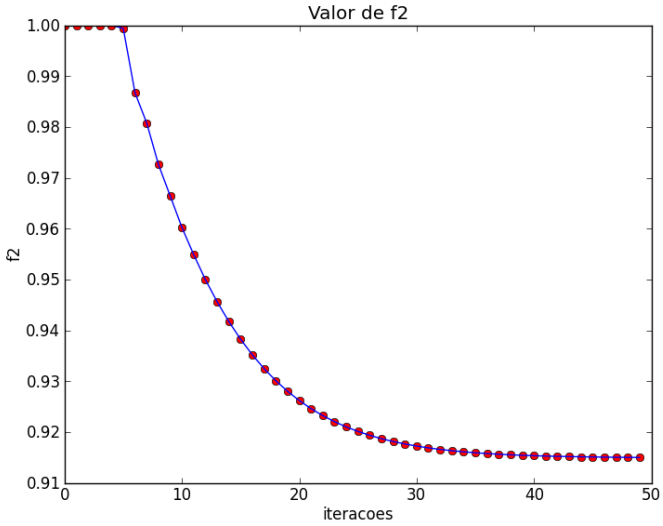


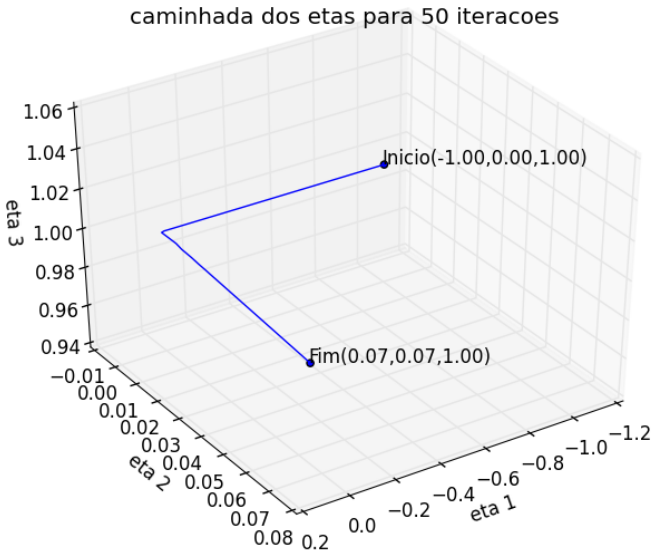



caminhada dos etas para 10 iteracoes
Inicio(-4.59,0.00,1.00)
Fim(-2.19,-2.19,-2.19)
eta 1
eta 2
eta 3
−5.0
−4.5
−4.0
−3.5
−3.0
−2.5
−2.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5

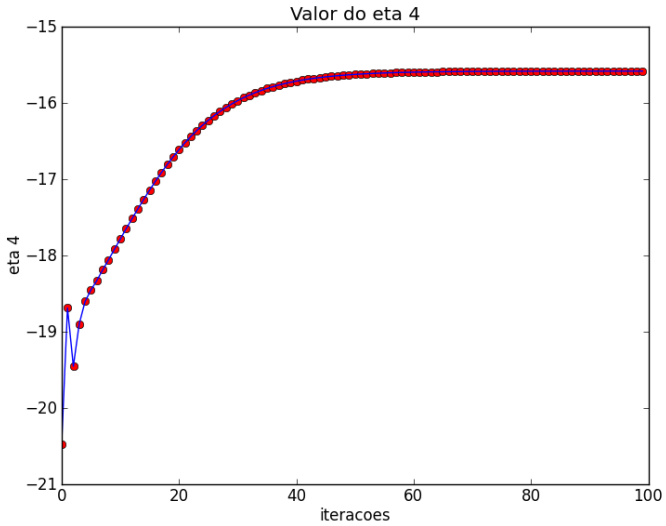
4.7.2 Início em [0, 0] para 50 iterações

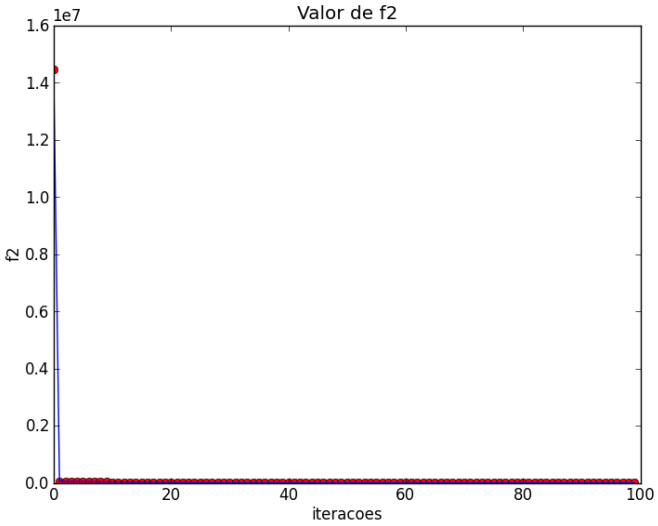


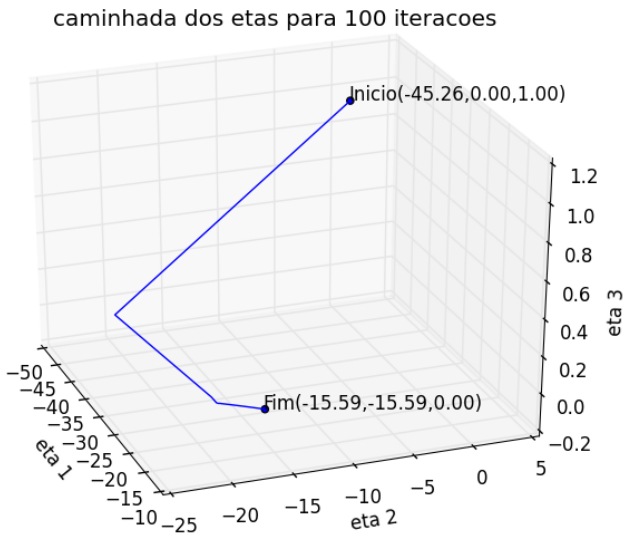



caminhada dos etas para 50 iteracoes
Inicio(-1.00,0.00,1.00)
Fim(0.07,0.07,1.00)
eta 3
eta 2
eta 1
1.06
1.04
1.02
1.00
0.98
0.96
0.94
−0.01
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2

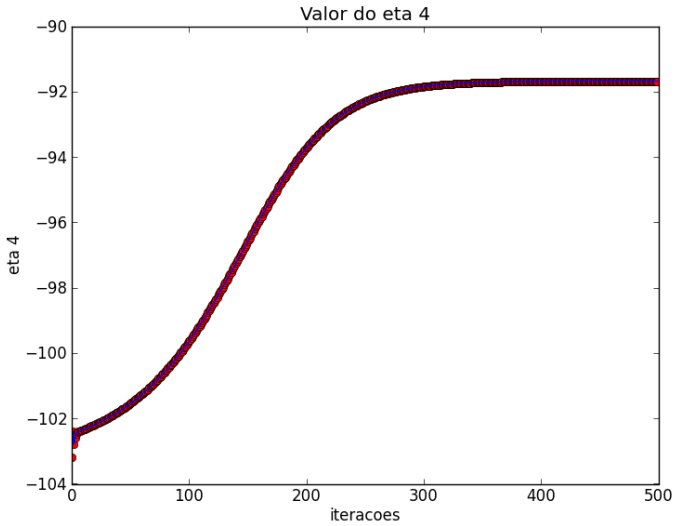
4.7.3 Início em [20, 20] para 100 iterações

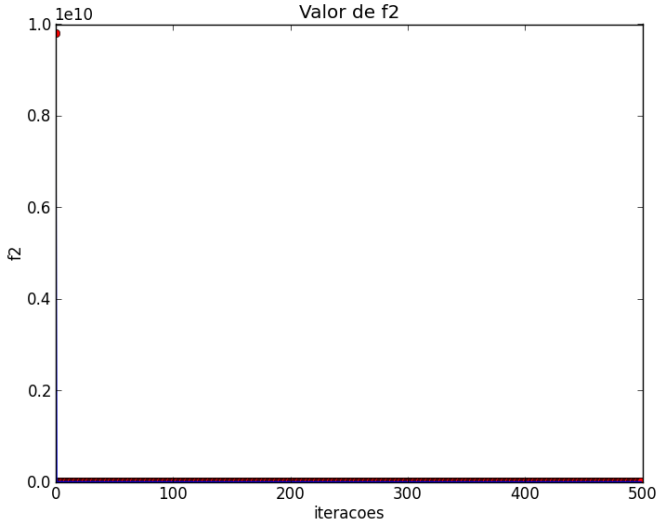


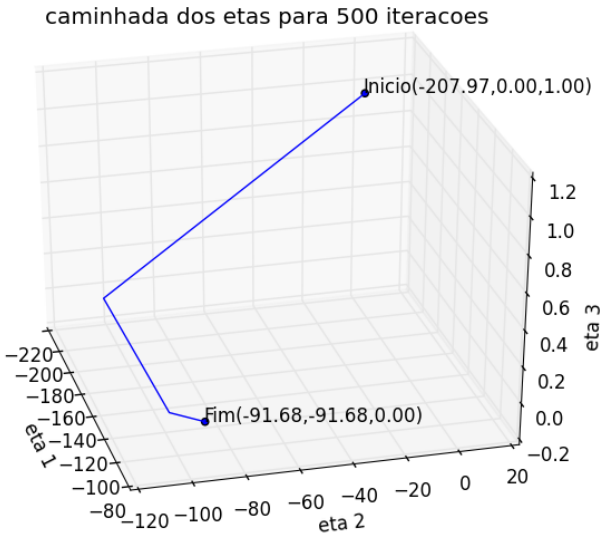



caminhada dos etas para 100 iteracoes
Inicio(-45.26,0.00,1.00)
Fim(-15.59,-15.59,0.00)
eta 1
eta 2
eta 3
−50
−45
−40
−35
−30
−25
−20
−15
−10
−25
−20
−15
−10
−5
0
5
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

4.7.4 Início em [100, 100] para 500 iterações






caminhada dos etas para 500 iteracoes
Inicio(-207.97,0.00,1.00)
Fim(-91.68,-91.68,0.00)
eta 1
eta 2
eta 3
−220
−200
−180
−160
−140
−120
−100
−80
−120
−100
−80
−60
−40
−20
0
20
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

5 ParTan

5.1 Minimização em R^2

O método das Tangentes Paralelas(ParTan) foi utilizado para a minimização de uma função $f : R^2 \to R$. O método consiste em, a cada iteração, a partir de um ponto x_0 dado, passar uma reta pelo ponto inicial e pelo ponto que minimiza a função ao longo da direção.

5.2 ParTan - descrição

Considerando um problema em minimizar uma função $f(x)$ sobre todo $x \epsilon E^n$ onde E^n é o espaço Euclidiano n-dimensional real. O procedimento ParTan pode ser descrito recursivamente da seguinte forma:

$k = 0$:

$$x^1 = x^0 - \mu^0 \nabla f(x^0)$$

onde $\mu^0 = argmin\{f(x^0 - \mu \nabla f(x^0))\}$

Para $k \geqslant 1$:

$$v^k = x^k - \mu^k \nabla f(x^k)$$

onde $\mu^k = argmin\{f(x^k - \mu \nabla f(x^k))\}$

$$x^{k+1} = x^{k+1} + \omega^k [v^k - x^{k-1}]$$

onde $\omega^k = argmin\{f(x^{k-1} + \omega(v^k - x^{k-1}))\}$

Desde que $x^1 - x^0$ esteja sobre o gradiente negativo, $x^1 - x^0$ é perpendicular à tangente em x^0. Desde que x^1 é o ponto mínimo sobre $x^1 - x^0$ e $v^1 - x^1$ esteja sobre o gradiente negativo, então $v^1 - x^1$ é perpendicular a $x^1 - x^0$. Consequentemente, $v^1 - x^1$ é paralela à tangente no ponto x^0. Desde que v^1 seja o mínimo sobre $v^1 - x^1$, a tangente em v^1 contêm a reta $v^1 - x^1$, na qual implica que a tangente no ponto x^0 é paralela à tangente em v^1. Portanto, x^2

é o ponto de mínimo sobre a direção $v^1 - x^0$, e também, o ponto de mínimo da função quadrática.

5.3 Condição de Parada

A condição de parada utilizada foi:

$$||\nabla f(x^0)|| < \epsilon$$

onde ϵ é o erro desejado definido inicialmente pela entrada do programa.

Ou seja, quando a norma do gradiente de x^0 for menor que o erro, significa que o ponto desejado foi encontrado.

5.4 Algoritmo de ParTan

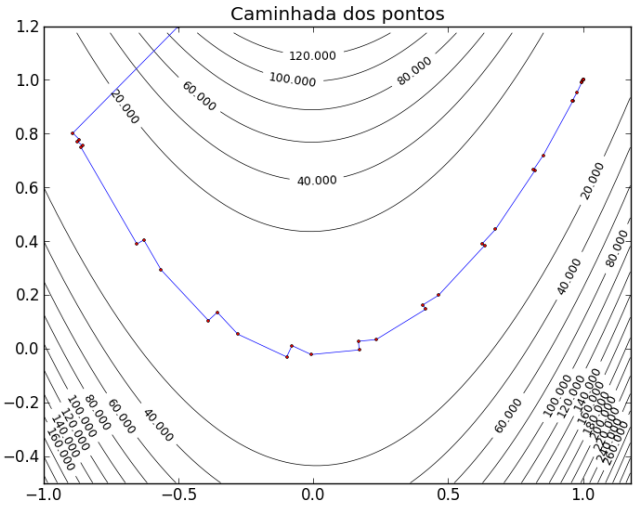
Procedimento:

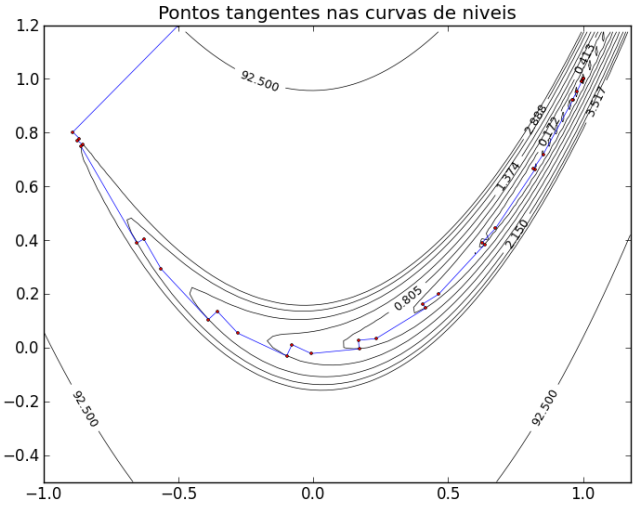
1. Definir um ponto x_0 inicial e fazer $x = x_0$
2. Enquanto $||\nabla f(x)|| > \epsilon$, onde ϵ é o erro, repetir:
   (a) Busca linear no ponto x na direção $\nabla f(x)$, encontrando o ponto p_0.
   (b) Busca linear no ponto p_0 na direção $\nabla f(p_0)$, encontrando o ponto p_1.
   (c) Fazer $d = p_1 - x$ depois busca linear no ponto x na direção $\nabla f(d)$, encontrando um novo ponto para x

5.5 Representação gráfica do comportamento do ParTan

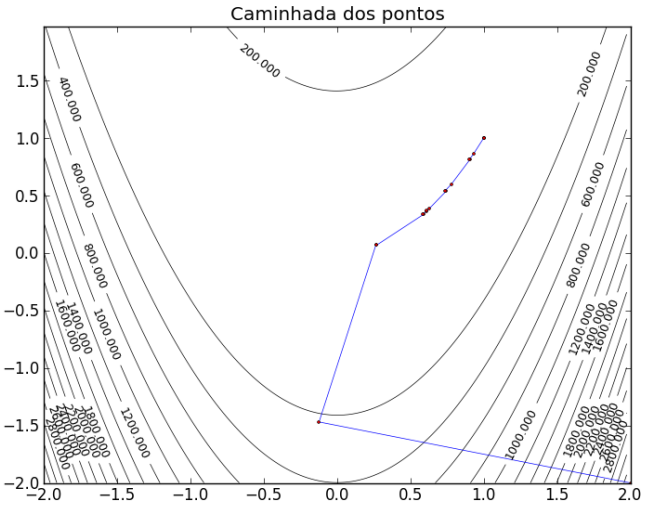
Realizando testes com a função de exemplo, ficou evidente que os pontos tangenciam cada curva de nível, sempre encontrando o menor ponto possível.

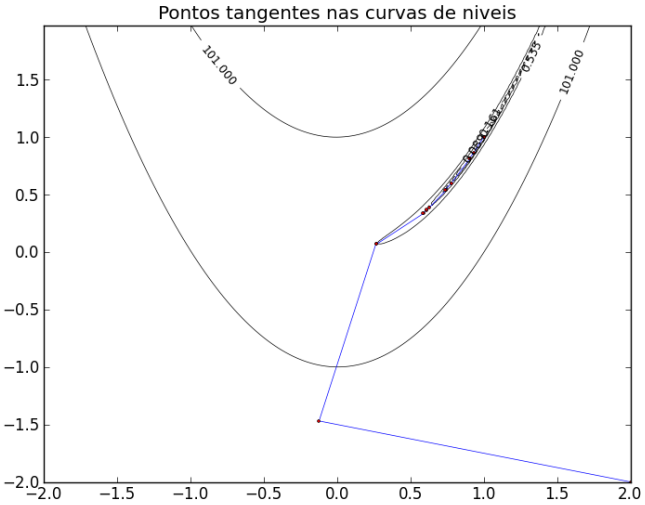
5.5.1 Início em [-0.5, 1.2]





5.5.2 Início em [2, -2]





6 Resultados do ParTan

Juntando o ParTan e a Busca Linear, tivemos resultados satisfatórios.

entrada.py

```
x = [2, -2]
argumento otimo = [1.0001283267248189, 1.000256585231958]
valor otimo = 1.64684654649e-08


x = [20, -3]
argumento otimo = [0.99999901590570672, 0.99999808130539891]
valor otimo = 1.21339745142e-12

x = [1, 1] # é o próprio mínimo
argumento otimo = [1, 1]
valor otimo = 0

x = [100, 100]
argumento otimo = [1.00000039736144, 1.0000008221948953]
valor otimo = 2.33366409184e-13

x = [-100, 100]
argumento otimo = [0.99999997463153278, 0.99999993370986051]
valor otimo = 2.48337799561e-14

x = [-200, 200]
argumento otimo = [0.99996538160948201, 0.99993193640225708]
valor otimo = 1.3357878133e-09

x = [-200, 300]
argumento otimo = [0.99999997979362665, 0.9999999599657462]
valor otimo = 4.2262317825e-16


#testes feitos com diferentes funções e diferentes dimensões
```

```
#entrada.py e entrada5.py são de dimensão 2
#entrada2.py e entrada3.py são de dimensao 3
#entrada4.py é de dimensão 4
#os arquivos estão na pasta teste

entrada.py
x = [2, -2]
argumento otimo = [1.0001283267248189, 1.000256585231958]
valor otimo = 1.64684654649e-08

entrada2.py
x = [3, -2, -10]
argumento otimo = [-4.2515391632884002e-05, -0.00025509234979765683,
    -1.8895729615115498e-06]
valor otimo = 1.80836188505e-08

entrada3.py
x = [20, -2, 7.43]
argumento otimo = [1.0000498288148181, 1.000099899787638,
    1.3994559549746369e-05]
valor otimo = 2.53761712535e-09

entrada4.py
x = [-2, 2, 2, 2]
argumento otimo = [0.99907342705390434, 0.99814390466666225,
    2.7409407719371738e-05, -0.0001747125669226792]
valor otimo = 9.28855395766e-07

entrada5.py
x = [10, -10]
argumento otimo = [3.5527136788005009e-15, 1.7763568394002505e-14]
valor otimo = 50.0
```

7 Método da Penalidade

Considere o seguinte problema:

$$Minimizar\, f(x)$$

$$Sujeita\, a\, h(x) = 0$$

Onde $f : R^n \rightarrow R$ e $h : R^n \rightarrow R^l$ são diferenciáveis. Seja $\mu > 0$ um parâmetro de penalidade, ou seja, um peso dado para a restrição, e considere o problema de penalidade para minimizar $q(x)$ sujeita a $x\, \epsilon\, R^n$, onde $q(x) = f(x) + \mu \sum_{i=1}^{l} h_i^2(x)$.

Essa equação motiva o próximo ponto a chegar perto do ponto mínimo, o qual $\sum_{i=1}^{l} h_i^2(x)$ fica perto de zero e ao mesmo tempo minimiza $f(x)$.

Como critério de parada, utilizamos o erro

$$\epsilon = \sqrt{\mu \sum_{i=1}^{l} h_i^2(x)} \tag{11}$$

Para implementar o método da penalidade, basta usar o partan com a função $q(x)$ e o gradiente $\nabla q(x)$ com o erro descrito acima.

7.1 Resultados

Para a entrada

```
d = 3
m = 1
x0 = [1.0, 3.0, 10.0]
eps = 0.00001

def f(x):
    fx = x[0]**2 + (2*x[1])**2 + (3*x[2])**2
    return(fx)

def g(x):
    gx =[2*x[0], 4*x[1], 6*x[2]]
    return(gx)

def h(x):
```

```
    hx = [x[0]**2 +x[1] -5*x[2]]
    return(hx)

def j(x):
    jx = [[2*x[0], 1, -5]]
    return(jx)
```

Tivemos os seguintes resultados:

```
argumento otimo = [5.9299676016700425e-06, 1.5182586370540642e-07
    , 1.2902465525097896e-07]
valor_otimo = 3.54065463834e-11
```

Para a entrada

```
d = 3
m = 1
x0 = [1.0, 3.0, 10.0]
eps = 0.001

def f(x):
fx = x[0]**2 + (2*x[1])**2 + (3*x[2])**2
return(fx)

def g(x):
gx =[2*x[0], 4*x[1], 6*x[2]]
return(gx)

def h(x):
hx = [x[0]**2 +x[1] -5*x[2] + 10]
return(hx)

def j(x):
jx = [[2*x[0], 1, -5]]
return(jx)
```

Tivemos os seguintes resultados:

```
argumento otimo = [0.33437263916786558, 0.8678659669925346, 2.1959342219464055]
valor_otimo = 46.5237143725
```

8 Conclusão

Os problemas propostos neste trabalho, assim como o ParTan, a Busca Linear e o GRG, foram uma ótima forma de introdução aos métodos de Programação Não Linear, onde o objetivo é encontrar a solução para a minimização de uma função. Com todas essas ferramentas foi possível desenvolver um algoritmo que encontra o mínimo de qualquer função unimodal com restrição.

Referências

[1] Andreas Antoniou. Wu-Sheng Lu - Practical Optimization Algorithms and Engineering Applications

[2] Stern, Julio Michael - Cognitive Constructivism and the Epistemic Significance of Sharp Statistical Hypotheses in Natural Sciences

[3] Stern, Julio Michael. Pereira, Carlos Alberto de Bragança. Ribeiro, Celma de Oliveira. Dunder, Cibele. Nakano, Fabio. Lauretto, Marcelo - Otimização e Processos Estocásticos Aplicados à Economia e Finanças

[4] Sun, W. Yuan, Y - Optimization Theory and Methods, Nonlinear Programming

[5] Wong, Peter J. Dressler, Robert M. Luenberger, David G. - A combined parallel-tangents/penalty-function approach to solving trajectory optimization problems

[6] `http://web.cecs.pdx.edu/~edam/Slides/LineSearchx4.pdf`

[7] Bauer, Jane - Implementação de Algoritmos de Programação Não Linear

[8] Ghorbani, Akk A. Bhavsar, Virendra C. - Parallel Tangent Learning Algorithm For Training Artificial Neural Networks

[9] S.S Rao - Engineering Optimization - Theory and Practice S.S.

[10] Bazaara, M. S. - Nonlinear programming: theory and algorithms