METODOS UNIDIMENSIONAIS

A busca unidimensional e a base de muitos algoritmos que tentam encontrar um valor mínimo ou máximo de uma função que pode ser não linear e de múltiplas variáveis. O processo de encontrar um novo ponto de usando uma forma analítica eh extremamente complexo quando este existe. Devido a esta complexidade, utilizam-se métodos computacionais iterativos para se determinar o próximo ponto de uma função a partir de um ponto inicial, sintetizado conforme a seguinte expressão:

Onde:

: próximo ponto a ser determinado pertencente a

: largura do passo a ser dado

: direção admissível para o próximo passo

e também: , e .

O processo de encontrar um valor mínimo global desta função irrestrita consiste em resolver o seguinte problema:+

A melhor forma de resolver este problema eh encontrar um valor de de tal maneira que o problema a ser resolvido se torna

Fazendo a seguinte substituição:

Desta forma a função passa a ser uma sessão unidimensional da função .

Com esta transformação, podemos simplificar o problema através de uma simples busca de uma única variável que eh unidimensional. A partir deste ponto diversos métodos foram desenvolvidos a fim de encontrar o melhor valor de dentro de um intervalo fechado .

Serão apresentados quatro métodos de busca unidimensionais, Fibonacci, Newton, Falsa-Posição e Armijo. Sendo estes implementados computacionalmente através do software Matlab®.

Considerando a função

foi solicitado que se desenvolvessem funções para o calculo de Os códigos implementados seguem abaixo

**Calculo de**

function y = calc\_fx(x)

%-----------------------------------------------------------

% DESCRIÇÃO: calcula o valor da função em um ponto

% INPUT:

% f = (2\*x1\*x2-2)^2 + (x2-x3)^2 + (x2+2\*x3-5)^4;

% [ x1 ]

% x= | x2 | = vetor que contem os valores do vetor x

% [ x3 ]

% OUTPUT:

% y = valor da função no ponto x

%-----------------------------------------------------------

y = (2\*x(1,1)\*x(2,1)-2)^2 + (x(2,1)-x(3,1))^2 + (x(2,1)+2\*x(3,1)-5)^4;

**Calculo de**

function [y] = calc\_grad(f,var,pto)

%---------------------------------------------------------------------

% DESCRIÇÃO: calcula o gradiente de uma funçao

% INPUT:

% f = funçao simbolica

% f = (2\*x1\*x2-2)^2 + (x2-x3)^2 + (x2+2\*x3-5)^4;

% [var1] => primeira variavel de f(x)

% var = |var2| => segunda variavel de f(x)

% [var3 ] => terceira variavel de f(x)

%

% [a] = valor numerico relativo à variavel simbolica var1 do gradiente de f(x)

%pto =|b| = valor numerico relativo à variavel simbolica var2 do gradiente de f(x)

% [c] = valor numerico relativo à variavel simbolica var3 do gradiente de f(x)

% OUTPUT:

% y = vetor na forma [x y z]

%---------------------------------------------------------------------------

%funcao que determina simbolicamente o gradiente de f(x)

grad\_f = gradient(f,[var(1,1) var(2,1) var(3,1)]);

% Verifica se o numero de argumentos passado para a funcao calc\_grad for

%igual a 2(f,var) retorna a funcao em forma simbolica

if nargin == 2

y= grad\_f;

%caso contrario retorna o valor da funão no ponto a b c

else

%calculo do gradiente no ponto [a,b,c]

y = double(subs(grad\_f,[var(1,1) var(2,1) var(3,1)],[pto(1,1) pto(2,1) pto(3,1)]));

end

**Calculo de**

function [y] = calc\_hess(f,var,pto)

%---------------------------------------------------------------------------

% DESCRIÇÃO DA FUNCAO: Calcula a matriz hessiana de uma função

% ENTRADA:

% f = funçao simbolica

% f = (2\*x1\*x2-2)^2 + (x2-x3)^2 + (x2+2\*x3-5)^4;

% [var1] = primeira variavel de f(x)

%var = |var2| = segunda variavel de f(x)

% [var3] = terceira variavel de f(x)

% [a] = valor numerico relativo à variavel simbolica var1 da hessiana de f(x)

% pto =|b| = valor numerico relativo à variavel simbolica var2 da hessiana de f(x)

% [c] = valor numerico relativo à variavel simbolica var3 da hessiana de f(x)

% SAIDA:

% y = vetor coluna contendo os valores calculados

%---------------------------------------------------------------------------

if nargin ==2

%funcao que determina simbolicamente a hessiana de f(x)

hess\_f = hessian(f,[var(1,1) var(2,1) var(3,1)]);

y = hess\_f;

else

%funcao que determina simbolicamente a hessiana de f(x)

hess\_f = hessian(f,[var(1,1) var(2,1) var(3,1)]);

%calculo do gradiente no ponto [a,b,c]

y = double(subs(hess\_f,[var(1,1) var(2,1) var(3,1)],[pto(1,1) pto(2,1) pto(3,1)]));

end

Calculo da direção

function y = calc\_norma(var1)

% ---------------------------------------------------------------------------

% DESCRIÇÃO: FUNÇÃO QUE CALCULA A NORMA-2 DE UM VETOR

% INPUT:

% var1 = Vetor na forma [x; y; z]

% OUTPUT:

% Norma-2 de um vetor https://pt.wikipedia.org/wiki/Norma\_(matem%C3%A1tica)

% ---------------------------------------------------------------------------

y = norm(var1);

--

x0 = [0.5;0.5;0.5];

syms x1 x2 x3;

f = (2\*x1\*x2-2)^2 + (x2-x3)^2 + (x2+2\*x3-5)^4;

aux1 = -(calc\_grad(f,[x1;x2;x3],x0));

aux2 = calc\_norma(calc\_grad(f,[x1;x2;x3],x0));

d0 = aux1/aux2;

**d0 =**

**0.0078**

**0.4534**

**0.8913**

Por seguinte, foi solicitado que usando as funções acima e também os pontos fosse implementado os métodos de busca unidimensionais Fibonacci, Newton, Falsa Posiçao e Armijo.

Método Fibonacci

O método de Fibonacci eh um procedimento de minimização unidimensional de uma função quasi-convexa sobre um intervalo fechado. O método Fibonacci eh baseado sobre a sequencia numérica Fibonacci :

Durante uma iteraço supondo um intervalo de incerteza e tambem considere dois pontos  e o numero de avaliações planejadas.

O próximo intervalo de incerteza eh dado por se  ou o intervalo será se .

Caso seja dado uma tolerância de erro , a partir deste, deve-se determinar o numero de iterações necessárias usando a condição:

Abaixo esta o código implementado

function alpha = buscaFibonaci(alpha\_bar,d,tol,x)

%------------------------------------------------------

% DESCRICAO: Metodo de busca unidimensional Fibonaci

% INPUT:

% alpha\_bar: valor do limite superior do intervalo[0,alpha\_bar)

%

% [ dx1 ]

% d = | dx2 | = vetor que contem a direçao de descida

% [ dx3 ]

%

% tol = valor final do intervalo de incerteza

%

% [ x1 ]

% x= | x2 | = vetor que contem os pontos do vetor x

% [ x3 ]

%OUTPUT:

% alpha:

%------------------------------------------------------

tic

%funcao f(x+alpha\*d)

%f\_alpha = [2\*(x1+alpha\*dx1)\*(x2+alpha\*dx2)]^2 + [(x2+alpha\*dx2)+(x3+alpha\*dx3))]^2 + [(x2+alpha\*dx2)+(x3+alpha\*dx3)-5]^4;

%valor da sequencia fibonacci

Fn = alpha\_bar/tol;

%construção do vetor fibonacci

%OBS1: o vetor se inicia com inidce 1(um)

%-------------|------------------------------------------------------------

% n =|1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ...

%fibonacci(n)=|1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987...

%-------------|------------------------------------------------------------

%% Busca pelo valor Fn

i = 2;

fib(1) = 1;

fib(2) = 1;

while(fib(i) <= Fn)

i=i+1;

fib(i) = fib(i-1)+fib(i-2);

end

%indice relativo ao numero fibonacci Fn

% OBS: o metodo de busca começa com indice 0 por isso foi subtraido uma uni

% dade do indice

index\_fib = length(fib);

%% Determinação dos intervalos

%intervalo inicial: [0,alpha\_bar]

a=0;

b=alpha\_bar;

%regulador de iteraçoes

k = 1;

%iteracao de busca dos intervalos

while(k<index\_fib-1)

%verifica os proximos intervalos

lambida\_k = a + (fib(index\_fib-k-1)/fib(index\_fib-k+1))\*(b-a);

u\_k = a + (fib(index\_fib-k)/fib(index\_fib-k+1))\*(b-a);

%calcula o valor da função para os novos limites calculados

f\_lambidak = calc\_f\_alpha(x,d,lambida\_k);

f\_uk = calc\_f\_alpha(x,d,u\_k);

%testa e atribui os novos pontos

if(f\_lambidak > f\_uk)

a = lambida\_k;

b = b;

else

a = a;

b = u\_k;

end

k = k+1;

end

alpha = (lambida\_k+u\_k)/2;

% Caso tenha alguma duvida verificar BAZARAA,et. al. Nonlinear Programmim

% Theory and Algorithms,3rd Edition p.351

toc

function [y] = calc\_f\_alpha(x,dir,val)

%---------------------------------------------------------------------------

% DESCRIÇÃO DA FUNCAO: Calcula o valor da função unidimensional f(alpha)

% ENTRADA:

%

% [ x1 ]

% x = | x2 | = vetor que contem os pontos do vetor x

% [ x3 ]

%

% [ dx1 ]

% d = | dx2 | = vetor que contem a direçao de descida

% [ dx3 ]

%

% val = valor a ser avaliado

%

% SAIDA:

% y = valor de f(val)

%---------------------------------------------------------------------------

%Funcao de teste, vide slides prof Luciana

%y = 10 + 2\*val^4 - 4\*val^2;

%chamar no prompt buscaFibonaci(1.5,d0,0.3,x0)

%Funcao f(alpha)

y = ((2\*(x(1,1)+val\*dir(1,1)))\*(x(2,1)+val\*dir(2,1)))^2 + ...

((x(2,1)+val\*dir(2,1))+(x(3,1)+val\*dir(3,1)))^2 +...

((x(2,1)+val\*dir(2,1))+(x(3,1)+val\*dir(3,1))-5)^4;

Para os valores de e intervalo temos:

>> buscaFibonacci(1,d0,0.001,x0)

Numero de Iteraçoes:16

Elapsed time is 0.003733 seconds.

ans =

0.9994

Metodo Newton

O método de Newton consiste em realizar uma aproximação quadrática da função unimodal

Por meio de uma aproximação quadrática tal que:

A condição de primeira ordem para minimização determina que , logo realizando manipulações teremos que

Fazendo

O processo iterativo se mantem ate que a condição abaixo seja satisfeita

Sendo que pode ser entendido como um erro entre os pontos calculados

Abaixo esta o código implementado

function y = buscaNewton(d,x,tol,alpha0,simb)

%------------------------------------------------------

% DESCRICAO: Metodo de busca unidimensional de Newton

% INPUT:

%

%

% [ dx1 ]

% d = | dx2 | = vetor que contem a direçao de descida

% [ dx3 ]

%

%

% [ x1 ]

% x= | x2 | = vetor que contem os pontos do vetor x

% [ x3 ]

%

% tol = valor final do intervalo de incerteza

%

% alpha0 = valor inicial qualquer para iniciar a busca

% sim = variavel da funçao

% OUTPUT:

% alpha: = valor minimo da função f(alpha)

%------------------------------------------------------

tic

syms alpha;

alpha = simb;

%% Função theta(alpha)

f\_alpha = ((2\*(x(1,1)+alpha\*d(1,1)))\*(x(2,1)+alpha\*d(2,1)))^2 + ...

((x(2,1)+alpha\*d(2,1))+(x(3,1)+alpha\*d(3,1)))^2 + ...

((x(2,1)+alpha\*d(2,1))+(x(3,1)+alpha\*d(3,1)-5))^4;

%% Derivada primeira de thetha(alpha)

f\_1\_alpha = gradient(f\_alpha,alpha);

%theta\_1\_alpha = 2\*(dx2 + dx3)\*(x2 + x3 + alpha\*dx2 + alpha\*dx3) + 4\*(dx2 + dx3)\*(x2 + x3 + alpha\*dx2 + alpha\*dx3 - 5)^3 + 4\*dx1\*(2\*x1 + 2\*alpha\*dx1)\*(x2 + alpha\*dx2)^2 + 2\*dx2\*(2\*x1 + 2\*alpha\*dx1)^2\*(x2 + alpha\*dx2)

%% Derivada segunda de theta(alpha)

f\_2\_alpha = gradient(f\_1\_alpha,alpha);

%theta\_2\_alpha = 8\*dx1^2\*(x2 + alpha\*dx2)^2 + 2\*(dx2 + dx3)^2 + 2\*dx2^2\*(2\*x1 + 2\*alpha\*dx1)^2 + 12\*(dx2 + dx3)^2\*(x2 + x3 + alpha\*dx2 + alpha\*dx3 - 5)^2 + 16\*dx1\*dx2\*(2\*x1 + 2\*alpha\*dx1)\*(x2 + alpha\*dx2)

%% Loop

%este valor acumula o numero de iteraçoes

k=1;

%o primeiro valor de alpha é o inicial que foi passado

alpha\_k = alpha0;

%variaveis temporarias que armazenam o valor calculadas para a primeira e

%segunda derivadas

temp1 = double(subs(f\_1\_alpha,alpha,alpha\_k));

temp2 = double(subs(f\_2\_alpha,alpha,alpha\_k));

%proximo alpha

alpha\_k1 = alpha\_k - temp1/temp2;

while(abs(alpha\_k1-alpha\_k)>tol)

k=k+1;

%atualiza o valor do alpha atual

alpha\_k = alpha\_k1;

%variaveis temporarias

temp1 = double(subs(f\_1\_alpha,alpha,alpha\_k));

temp2 = double(subs(f\_2\_alpha,alpha,alpha\_k));

%atualiza proximo alpha

alpha\_k1 = alpha\_k - temp1/temp2;

end

%rotorno da função

fprintf('Numero de Iteraçoes:%d \n',k)

y= alpha\_k1;

toc

>> buscaNewton(d0,x0,0.001,1,alpha)

Numero de Iteraçoes:5

Elapsed time is 0.220514 seconds.

ans =

2.0188

Método da Falsa-Posição

O método da Falsa-Posicao se assemelha muito ao método de Newton, entretanto neste modelo, apenas derivadas de primeira ordem serão consideradas, ou seja, . Neste critério deve-se saber de antemão os valores de , desta forma a função pode ser aproximada por uma função quadrática contendo apenas primeiras derivadas de

O ponto pode ser determinado avaliando onde a derivada de deixam de existir. Assim realizando manipulações algébricas teremos a seguinte forma de determinar o próximo ponto da aproximação quadrática.

O processo iterativo se mantem ate que a condição abaixo seja satisfeita

Sendo que pode ser entendido como um erro entre os pontos calculados

Abaixo esta a implementação do método

function y = buscaFalsaPosicao(d,x,tol,a,b,simb)

%------------------------------------------------------

% DESCRICAO: Metodo de busca unidimensional de Falsa Posição

% OBS: Neste método é necessário passar dois pontos onde se acredita que o

% mínimo se encontra. A partir destes pontos o minimo eh determinado

%

% INPUT:

%

%

% [ dx1 ]

% d = | dx2 | = vetor que contem a direçao de descida

% [ dx3 ]

%

%

% [ x1 ]

% x = | x2 | = vetor que contem os pontos do vetor x

% [ x3 ]

%

% tol = valor final do intervalo de incerteza

%

% alpha0 = valor inicial qualquer para iniciar a busca

%

% a = valor mínimo do intervalo

%

% b = valor máximo do intervalo

%

% sim = variavel da funçao

%

% OUTPUT:

%

% alpha: = valor minimo da função f(alpha)

%------------------------------------------------------

tic

%variavel simbolica para derivaçao

syms alpha;

% atribuiçao da variavel simbolica para uma variavel local

alpha = simb;

%% Função theta(alpha)

f\_alpha = ((2\*(x(1,1)+alpha\*d(1,1)))\*(x(2,1)+alpha\*d(2,1)))^2 + ...

((x(2,1)+alpha\*d(2,1))+(x(3,1)+alpha\*d(3,1)))^2 + ...

((x(2,1)+alpha\*d(2,1))+(x(3,1)+alpha\*d(3,1)-5))^4;

%% Derivada primeira de theta(alpha)

f\_1\_alpha = gradient(f\_alpha,alpha);

%theta\_1\_alpha = 2\*(dx2 + dx3)\*(x2 + x3 + alpha\*dx2 + alpha\*dx3) + 4\*(dx2 + dx3)\*(x2 + x3 + alpha\*dx2 + alpha\*dx3 - 5)^3 + 4\*dx1\*(2\*x1 + 2\*alpha\*dx1)\*(x2 + alpha\*dx2)^2 + 2\*dx2\*(2\*x1 + 2\*alpha\*dx1)^2\*(x2 + alpha\*dx2)

%% Loop iterativo

%menor valor de alpha inicial

alpha\_ant=a;

%maior valor de alpha inicial

alpha\_atual=b;

%proximo alpha

alpha\_prox = alpha\_atual - (double(subs(f\_1\_alpha,alpha,alpha\_atual)))\*((alpha\_ant - alpha\_atual)/((double(subs(f\_1\_alpha,alpha,alpha\_ant)))-(double(subs(f\_1\_alpha,alpha,alpha\_atual)))));

%erro entre o proximo valor de alpha com o valor atual de alpha

erro = abs(alpha\_prox-alpha\_atual);

%variavel para contagem das iteracoes

k=1;

%loop

while (erro>tol)

%incrementa k

k=k+1;

%troca dos valores de alpha

alpha\_ant = alpha\_atual;

alpha\_atual = alpha\_prox;

alpha\_prox = alpha\_atual - (double(subs(f\_1\_alpha,alpha,alpha\_atual)))\*((alpha\_ant - alpha\_atual)/((double(subs(f\_1\_alpha,alpha,alpha\_ant)))-(double(subs(f\_1\_alpha,alpha,alpha\_atual)))));

%calculo do erro

erro = abs(alpha\_prox-alpha\_atual);

end

%impressao do numero de iteraçoes, valor de alpha e tempo

fprintf('Numero de Iteraçoes:%d \n',k)

y = alpha\_prox;

toc

>> buscaFalsaPosicao(d0,x0,0.001,0,1,alpha)

Numero de Iteraçoes:7

Elapsed time is 0.213286 seconds.

**ans =**

**2.0188**

Metodo Armijo

Ate o momento os métodos apresentavam soluções com alto nível de exatidão, uma vez que possamos ser mais flexíveis com estas condições, podemos