**UNIVERSIDADE FEDERAL DE PERNAMBUCO**

**CENTRO DE TECNOLOGIA E GEOCIÊNCIAS**

**RELATÓRIO**

**Lista de Exercícios III**



**Disciplina:** Métodos Computacionais para Engenharia Civil.

**Professora:** Liliane Fonseca.

**Aluno:** Alexandre de Souza Junior.

**Recife, 2017.**

MÉTODOS COMPUTACIONAIS PARA ENGENHARIA CIVIL

LISTA DE EXERCÍCIOS – AJUSTE DE CURVAS E DIFERENCIAÇÃO NUMÉRICA

**QUESTÃO 1:**

>> dist = [2.4; 1.5; 2.4; 1.8; 1.8; 2.9; 1.2; 3; 1.2];

>> largf = [2.9; 2.1; 2.3; 2.1; 1.8; 2.7; 1.5; 2.9; 1.5];

>> plot(dist,largf,'o');

>> RegressLinear(dist,largf);

% Equacao da reta: y=0.71672+0.73349\*x

>> largmin = 0.71672 + 0.73349\*1.8

>> largmin =

2.0370



**QUESTÃO 2:**

>> x = [0.1; 0.2; 0.4; 0.6; 0.9; 1.3; 1.5; 1.7; 1.8];

>> y = [0.75; 1.25; 1.45; 1.25; 0.85; 055; 0.35; 0.28; 0.18];

Para linearização a equação dada no enunciado, aplicou-se o logaritmo natural e suas propriedades, a fim de se obter uma equação do tipo:

Que pode ser reescrita da forma:

E, portanto, tem-se a equação linearizada:

O código referente a este processo de linearização encontra-se a seguir:

function rpot = RegressLinearPot2(a,b)

%Modelo de potência (QUESTÃO 2)

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

n = length(a);

if n~=length(b)

error('os vetores x e y devem ter a mesma dimensao')

end

for i = 1:n

x(i) = a(i);

y(i) = log(b(i)) - log(a(i));

end

Sx = sum(x);

Sy = sum(y);

Sxy = sum(x.\*y);

Sx2 = sum(x.^2);

a1 = (n \* Sxy - Sx \* Sy) / (n \* Sx2 - (Sx).^2 );

a0 = (Sy / n) - (Sx / n) \* a1;

alfa = exp(a0);

beta = a1;

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

%Quantificacao do erro da regressao

%Calculo do erro global (Sr)

E = sum( (b - a0 - a1\*a).^2 );

%Calculo do desvio St em relacao à media

St = sum ( (b - (Sy/n)).^2 );

%Calculo do erro-padrao da estimativa

Sy\_x = sqrt(E/(n-2));

%Coeficiente de determinacao

r2 = (St-E) / St;

%Coeficiente de correlacao:

r=sqrt(r2);

rpot = r;

%Gráficos

xg = linspace(min(x),max(x),2);

yreg = a0 + a1 \* xg;

disp(['Equação da reta: ','y=',num2str(a0),'+',num2str(a1),'\*x']);

disp(['Equação da curva: ','y=(',num2str(alfa),')\*x\*exp(',num2str(beta),'\*x)']);

subplot(2,2,1)

plot(xg,yreg,'r-');

hold on

plot(x,y,'o');

grid on

title('Regressao Linear - Modelo de Potencia')

%Transformando para o modelo original (para fazer previsões):

xreal = linspace(min(a),max(a));

yreal = alfa\*xreal.\*exp(beta\*xreal);

subplot(2,2,2)

plot(xreal,yreal,'-m');

hold on

plot(a,b,'o');

grid on

title('Curva de previsão e dados originais');

Dados de saída:

>> RegressLinearPot2(x,y);

****

****

Equação da reta: y=2.2682-2.4733\*x

Equação da curva: y=(9.6618)\*x\*exp(-2.4733\*x)

**QUESTÃO 3:**

>> x = [0.10; 0.17; 0.27; 0.35; 0.39; 0.42; 0.43; 0.44];

>> y = [10; 20; 30; 40; 50; 60; 70; 80].\*10^4;

(a) Para a parte linear do sistema:

>> RegressLinear(x(1:4),y(1:4));



Equação da reta: y=-10682.2881+1171605.7891\*x

K = 117,16 \*104 N/m

Para a parte não-linear do sistema:

>> RegresPol(x(5:8),y(5:8));

Equação da curva: y = 19934806.6244-99392265.167\*x+127071823.1725\*x^2



(b) As equações para uma reta e para uma curva de potência obtidas seguem abaixo, incluindo o gráfico no qual as duas estão inseridas, junto com os pontos originais. Pode-se observar que há pouca diferença no modo como cada curva se ajusta aos dados, embora visualmente, seja o modelo de potência o que resulta em menor erro.

>> RegressLinear(x,y)

Equação da reta: y = -128978.9303 + 1802269.0438\*x

>> RegressLinearPot(x,y)

Equação da curva: y = 1854460.7656\*x^1.2876



**QUESTÃO 4:**

>> x = [5; 10; 15; 20; 25; 30; 35; 40; 45; 50];

>> y = [17; 24; 31; 33; 37; 37; 40; 40; 42; 41];

(a) Para o ajuste de uma reta, tem-se:

>> RegressLinear(x,y)

Equação da reta: y = 20.6 + 0.49455\*x



(b) Para o ajuste de equação de potência, tem-se:

>> RegressLinearPot(x,y)

Equação da curva: y=9.9529\*x^0.38508



(c) Para o ajuste de uma curva de função inversa, tem-se:

>> RegressLinearInv(x,y)

Equação da curva: y = (50.0921)\*x/(9.8914+x)



(d) Para o ajuste de uma curva de parábola, tem-se:

>> RegressPol(x,y)

Equação da curva: y = 11.7667 + 1.3779\*x + -0.016061\*x^2



Agrupando todas as curvas num só gráfico, nota-se que não há uma curva superior, embora pode-se observar que as curvas referentes à parábola e à taxa de crescimento de saturação são as que mais se aproximam dessa característica na maior parte do intervalo de domínio.



**QUESTÃO 5:**

>> x = [0.1; 0.4; 0.5; 0.7; 0.7; 0.9];

>> y = [0.61; 0.92; 0.99; 1.52; 1.47; 2.03];

Foi criado para este problema o seguinte código, referente ao método dos mínimos quadrados geral:

function rpol = MMQgeral(x,y)

%Método dos Mínimos Quadrados Geral

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

% Variaveis de entrada:

% x: variavel independente

% y: variavel dependente

% Variaveis de saida

% a: vetor com os coeficientes do polinomio

% r2: coef. de determinacao

% s\_xy: erro-padrão da estimativa

n = length(x);

if n~=length(y)

error('os vetores x e y devem ter a mesma dimensao')

end

m = 4; %numero de funcoes

fx = zeros(m,n);

for i = 1:n

fx(1,i) = 1;

fx(2,i) = x(i);

fx(3,i) = sin(x(i));

fx(4,i) = exp(x(i));

end

a = zeros(m,m);

b = zeros(m,1);

for k = 1:m

for j = 1:m

somaA = 0;

somaB = 0;

for i = 1:n

somaA = somaA + fx(k,i).\*fx(j,i);

somaB = somaB + y(i).\*fx(k,i);

end

a(k,j) = somaA;

end

b(k) = somaB;

end

Sy = sum(y);

% Chamando a rotina Gauss para resolver o sistema

c = Gauss(a,b);

a0=c(1); a1=c(2); a2=c(3); a3=c(4);

%Calculando valores do Yajustado (yfit) - (parábola)

xfit = x(1):0.01:x(end);

yfit = a0 + a1\*xfit + a2\*sin(xfit) + a3\*exp(xfit);

disp(['Equação da curva: ','y = ',num2str(a0),' + ',num2str(a1),'\*x + ', num2str(a2), '\*sen(x) + ', num2str(a3), '\*exp(x)']);

%Gráfico dos dados originais e traçado do polinômio ajustado

plot(x,y,'or',xfit,yfit)

hold on

grid on

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

%Quantificacao do erro da regressao

%Calculo do erro global (Sr)

E = sum( (y - a0 - a1\*x - a2\*x.^2).^2 );

%Calculo do desvio St em relacao à media

St = sum ( (y - (Sy/n)).^2 );

%Calculo do erro-padrao da estimativa

Sy\_x = sqrt(E/(n-3));

%Coeficiente de determinacao

r2 = (St-E) / St;

%Coeficiente de correlacao:

r=sqrt(r2);

rpol = r;

Equação da curva:

y = -5.5881 + -21.2452\*x + 14.6464\*sen(x) + 6.2095\*exp(x)



**QUESTÃO 6:**

>> x = [16; 40; 64; 88; 112];

>> y = [4.2; 9.2; 10; 10.7; 8.6];

(a) Para o valor da velocidade igual a 105km/h e usando o método de Lagrange:

>> economia150 = IntLagrange(x,y,105)

Coeficientes de Lagrange:

-0.1673 2.0068 -5.1873 8.9243 4.3549

economia150 =

9.9314

(b) Para o valor da velocidade igual a 48km/h e usando o método de Newton:

>> economia48 = IntNewton(x,y,48)

Coeficientes do polinomio de Newton:

4.2000 0.2083 -0.0036 0.0000 -0.0000

economia48 =

9.5909

**QUESTÃO 7:**

(a) Para definir o valor da função de forma analítica, tem-se que, ao plotar o gráfico da função no intervalo de domínio entre x = 2 e x = 3, pode-se observar que o valor de x em que a função é igual a 0.85 é aproximadamente x = 2.381.



(b) Usando três pontos a fim de ajustar um polinômio quadrático, e visto que se trata de um número pequeno de pontos, pode-se utilizar da forma padrão ou convencional para a obtenção do polinômio interpolador.

%INTERPOLACAO QUADRÁTICA

>> x2 = [1; 2; 3];

>> y2 = [0.5; 0.8; 0.9];

>> fx2 = intPoli(x2,y2);

f(x) = -0.1000\*x^2 + 0.6000\*x + -0.0000

>> xraiz2 = bissec('funcao',2,3,1e-06,100);

xraiz2 =

2.2929

(c) Procedendo-se da mesma maneira, mas usando desta vez quatro pontos a fim de ajustar um polinômio de ordem 3, tem-se que:

%INTERPOLACAO CUBICA

>> x3 = [1; 2; 3; 4];

>> y3 = [0.5; 0.8; 0.9; 0.941176];

>> fx3 = intPoli(x3,y3);

f(x) = 0.0235\*x^3 + -0.2412\*x^2 + 0.8588\*x + -0.1412

>> xraiz3 = bissec('funcao2',2,3,1e-6,100);

xraiz3 =

2.3506

**QUESTÃO 8:**

>> x = [0; 0.2; 0.4; 0.6; 0.8; 1.0];

>> y = 10^-2\*[0; 7.78; 10.68; 8.38; 3.97; 0];

>> EI = 1.2e7;

Para os dados acima, foi desenvolvido o algoritmo que segue abaixo, para as aproximações exigidas no enunciado da questão.

function dx2 = deriv8(x,y)

% calcula a segunda derivada de uma função para um conjunto de dados igualmente espaçados. (QUESTAO 8)

%Usando diferenças:

% progressivas na extremidade esquerda com quatro pontos

% centradas nos pontos interiores

% regressivas na extremidade direita com quatro pontos

% Input:

% x vetor com as coordenadas de x.

% y vetor com as coordenadas de y

% Output:

% dx2 vetor com o valor da segunda derivada em cada ponto

n = length(x);

dx2(1) = (2\*y(1)-5\*y(2)+4\*y(3)-y(4))/(x(2)-x(1))^2; %progressiva

for i=2:n-1

dx2(i) = (y(i-1)-2\*y(i)+y(i+1))/(x(i)-x(i-1))^2; %centrada

end

dx2(n) = (-y(n-3)+4\*y(n-2)-5\*y(n-1)+2\*y(n))/(x(n)-x(n-1))^2; %regressiva

%Graficos

subplot(2,1,1)

plot(x,y,'.-k')

xlabel('x'); ylabel('f(x)');

subplot(2,1,2)

plot(x,dx2,'.-r')

xlabel('x'); ylabel('df(x)');

Dados de saída:

>> Mx = -EI\*deriv8(x,y);

Mx =

1.0e+07 \*

1.3680 1.4640 1.5600 0.6330 -0.1320 -0.8970

**QUESTÃO 9:**

Para dados desigualmente espaçados, usa-se como alternativa o ajuste por interpolação de um polinômio de 2º grau para o conjunto de dados a seguir, nos quais calculam-se as derivadas, que consistem na vazão Q para cada instante, em barris/min.

>> t = [0; 10; 20; 30; 45; 60; 75];

>> v = [0.4; 0.7; 0.77; 0.88; 1.05; 1.17; 1.35].\*10^6;

>> Q = PrimDerivLag(t,v)

Q =

1.0e+04 \*

4.1500 1.8500 0.9000 1.1133 0.9667 1.0000 1.4000