**Trabalho Computacional 4**

**Entrega:** 29/10/2019

**PARTE 1**

*Resolva os exercícios do TC 3 parte 3* ***utilizando a forma de Newton****. Em cada exercício explique se você encontrou vantagens ou desvantagens em relação ao Método de Lagrange.*

**Exercício 1 -** *A tabela abaixo dá a viscosidade V do etanol como uma função do percentual w de álcool anidro. Determine a viscosidade para w=12, w=25 e w=38. Use a FORMA DE NEWTON para obter um polinômio de grau menor ou igual a 5. Faça o gráfico do polinômio encontrado e destaque os pontos dados.*

| **w** | 5 | 10 | 15 | 20 | 30 | 40 |
| --- | --- | --- | --- | --- | --- | --- |
| **V** | 1226 | 1498 | 1822 | 2138 | 2662 | 2840 |

**R:** A vantagem da utilização do método de Newton é que ele utiliza todas as iterações correspondente com o número de graus, o tornando mais completo, enquanto o método de Lagrange se responsabiliza por uma única forma com todos os dados. A grande desvantagem do método é a quantidade de iterações responsáveis.

**Código implementado:**

| *import numpy as np*  *import math as mt*  *import matplotlib.pyplot as plt*  *from numpy.polynomial import polynomial as poly*  *#x = np.array([-1,0,1,3], dtype="double")*  *#y = np.array([3,1,3,43], dtype="double")*  *y = np.array([1226, 1498, 1822, 2138, 2662, 2840], dtype = "double")*  *x = np.array([5,10,15,20,30,40], dtype = "double")*  *#inicializando a tabela*  *T = np.zeros((y.size, y.size));*  *#primeira coluna*  *T[:,0] = y;*  *#segintes colunas*  *for i in range(1, y.size):*  *for j in range(i, y.size):*  *T[j, i] = (T[j, (i-1)] - T[(j-1), (i-1)]) / (x[j] - x[j-i])*    *#polinomio interpolador*  *p = np.array([T[0,0]], dtype="double")*  *paux = np.array([-x[0],1], dtype="double")*  *for i in range(1, T[0].size):*  *p.resize(i+1)*  *p += T[i,i]\*paux*  *paux = poly.polymul(paux,[-x[i],1])*  *print("\nPolinômio interpolador do conjunto de pontos dados: ", p)*  *v = 0*  *w = np.copy(x)*  *w = np.append(w, 12)*  *w = np.append(w, 25)*  *w = np.append(w, 38)*  *w = np.sort(w)*  *v = np.zeros(w.size)*  *for i in range(w.size):*  *for j in range(p.size):*  *v[i] += (w[i]\*\*j)\*p[j]*  *print("\nW: ", w)*  *print("\nV: ", v)*  *plt.plot(w, v, 'b', label='3.1', color='r')*  *plt.plot(w, v, 'p', label='3.1', color='b')*  *plt.savefig("ex1.png")*  ***RESULTADOS:***  *Polinômio interpolador do conjunto de pontos dados: [ 1.12925714e+03 -1.06257143e+01 7.57523810e+00 -3.53190476e-01 7.77904762e-03 -7.12380952e-05]*  *W: [ 5. 10. 12. 15. 20. 25. 30. 38. 40.]*  *V: [1226. 1498. 1625.849728 1822. 2138. 2422.54285714 2662. 2859.665792 2840.]* |
| --- |

**Exercício 2 -** *Determine um polinômio de grau <=5, usando a FORMA DE NEWTON, que interpola a tabela a seguir:*

| **x** | 1 | 2 | 3 | 4 | 5 | 6 |
| --- | --- | --- | --- | --- | --- | --- |
| **f(x)** | 14.5 | 19.5 | 30.5 | 53.5 | 94.5 | 159.5 |

*Use esse polinômio para encontrar o valor de f(4.5). Compare com o valor real f(4.5)=71.375. Faça o gráfico do polinômio encontrado e destaque os pontos dados.*

**R:** A aproximação suficiente do método de Newton é precisa com relação à casa decimal, mas imprecisa na casa das unidades. Essa divergência se desenvolve devido à imprecisão do método com relação às regressões das iterações.

**Código implementado:**

| *import numpy as np*  *import math as mt*  *import matplotlib.pyplot as plt*  *from numpy.polynomial import polynomial as poly*  *x = np.array([1, 2, 3, 4, 5, 6], dtype = "double")*  *y = np.array([14.5, 19.5, 30.5, 56.5, 94.5, 159.5], dtype = "double")*  *#inicializando a tabela*  *T = np.zeros((y.size, y.size));*  *#primeira coluna*  *T[:,0] = y;*  *#segintes colunas*  *for i in range(1, y.size):*  *for j in range(i, y.size):*  *T[j, i] = (T[j, (i-1)] - T[(j-1), (i-1)]) / (x[j] - x[j-i])*    *#polinomio interpolador*  *p = np.array([T[0,0]], dtype="double")*  *paux = np.array([-x[0],1], dtype="double")*  *for i in range(1, T[0].size):*  *p.resize(i+1)*  *p += T[i,i]\*paux*  *paux = poly.polymul(paux,[-x[i],1])*  *print("\nPolinômio interpolador do conjunto de pontos dados: ", p)*  *v = 0*  *w = np.copy(x)*  *w = np.append(w, 4.5)*  *w = np.sort(w)*  *v = np.zeros(w.size)*  *for i in range(w.size):*  *for j in range(p.size):*  *v[i] += (w[i]\*\*j)\*p[j]*  *print("\nW: ", w)*  *print("\nV: ", v)*  *plt.plot(w, v, 'b', label='3.1', color='r')*  *plt.plot(w, v, 'p', label='3.1', color='b')*  *plt.savefig("ex2.png")*  ***RESULTADOS:***  *Polinômio interpolador do conjunto de pontos dados: [-35.5 106. -79.75 27.75 -4.25 0.25]*  *W: [1. 2. 3. 4. 4.5 5. 6. ]*  *V: [ 14.5 19.5 30.5 56.5 73.8359375 94.5 159.5 ]* |
| --- |

**Exercício 3 -** *Um braço robótico com um scanner a laser está fazendo uma verificação rápida da qualidade dos furos feitos em uma placa retangular. Os centros dos furos na placa descrevem o caminho que o braço precisa tomar. Os centros dos furos estão localizados em um sistema de coordenadas cartesianas (com a origem no canto inferior esquerdo da placa) dado pelas especificações na tabela a seguir..*

| **x (cm)** | **y(cm)** |
| --- | --- |
| 2.00 | 7.2 |
| 4.25 | 7.1 |
| 5.25 | 6.0 |
| 7.81 | 5.0 |
| 9.20 | 3.5 |
| 10.60 | 5.0 |

*Se o laser estiver percorrendo um caminho linear de x=2 a x=4.25, qual é o valor de y quando x=4?*

**R:** O valor de y seria 7,4749, de acordo com o método de Newton implementado.

**Código implementado:**

| *import numpy as np*  *import math as mt*  *import matplotlib.pyplot as plt*  *from numpy.polynomial import polynomial as poly*  *x = np.array([2, 4.25, 5.25, 7.81, 9.2, 10.6], dtype = "double")*  *y = np.array([7.2, 7.1, 6, 5, 3.5, 5], dtype = "double")*  *print(y)*  *#inicializando a tabela*  *T = np.zeros((2, 2));*  *#primeira coluna*  *T[:,0] = y[:2];*  *#segintes colunas*  *for i in range(1, len(y[:2])):*  *for j in range(i, len(y[:2])):*  *T[j, i] = (T[j, (i-1)] - T[(j-1), (i-1)]) / (x[j] - x[j-i])*    *print(T)*  *#polinomio interpolador*  *p = np.array([T[0,0]], dtype="double")*  *paux = np.array([-x[0],1], dtype="double")*  *for i in range(1, T[0].size):*  *p.resize(i+1)*  *p += T[i,i]\*paux*  *paux = poly.polymul(paux,[-x[i],1])*  *print("\nPolinômio interpolador do conjunto de pontos dados: ", p)*  *v = 0*  *w = np.copy(x)*  *w = np.append(w, 4)*  *w = np.sort(w)*  *v = np.zeros(w.size)*  *for i in range(w.size):*  *if i <= 2:*  *for j in range(p.size):*  *v[i] += (w[i]\*\*j)\*p[j]*  *else:*  *v[i] = y[i-1]*  *print("\nW: ", w)*  *print("\nV: ", v)*  *plt.plot(w, v, 'b', label='3.1', color='r')*  *plt.plot(w, v, 'p', label='3.1', color='b')*  *plt.savefig("ex3.png")*  ***RESULTADOS:***  *[7.2 7.1 6. 5. 3.5 5. ]*  *[[ 7.2 0. ]*  *[ 7.1 -0.04444444]]*  *Polinômio interpolador do conjunto de pontos dados: [ 7.28888889 -0.04444444]*  *W: [ 2. 4. 4.25 5.25 7.81 9.2 10.6 ]*  *V: [7.2 7.11111111 7.1 6. 5. 3.5 5.]* |
| --- |

**PARTE 2**

**Exercício 1 -** *Ajuste os dados apresentados na tabela abaixo, utilizando o Método dos Mínimos Quadrados, por:*

*a) Uma reta.*

*b) Uma parábola y=a+bx+cx²*

*c) Como você compararia as duas curvas com relação aos dados?* ***Calcule a soma do erro quadrático para cada ponto.***

| **x** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **f(x)** | 0,5 | 0,6 | 0,9 | 0,8 | 1,2 | 1,5 | 1,7 | 2,0 |

**Código implementado:**

| *import numpy as np*  *import matplotlib.pyplot as plt*  *def seidel(A, b, n):*  *x0 = np.zeros(n)*  *x1 = np.copy(x0)*  *for cs in range(1000):*  *for i in range(n):*  *x1[i] = b[i]*  *for j in range(n):*  *if i != j:*  *x1[i] -= A[i,j]\*x1[j]*  *x1[i] /= A[i,i]*  *if (np.linalg.norm(x1-x0,np.inf)) > 1000:*  *return False*  *if (np.linalg.norm(x1-x0,np.inf) < 10\*\*(-6)):*  *return x1*  *else:*  *x0 = np.copy(x1)*  *return False*  *def ajuste(x, y, n):*  *X = np.zeros((n,n))*  *Y = np.zeros(n)*  *for i in range(n):*  *Y[i] = np.sum(y\*(x\*\*i))*  *for j in range(n):*  *X[i,j] = np.sum(1\*(x\*\*(j+i)))*  *return seidel(X, Y, n)*    *x = np.array([1, 2, 3, 4, 5, 6, 7, 8], dtype = "double")*  *y = np.array([0.5, 0.6, 0.9, 0.8, 1.2, 1.5, 1.7, 2.0], dtype = "double")*  *g = 0 #Grau do polinômio*  *g = 1*  *reta = ajuste(x, y, g+1)*  *g = 2*  *parabola = ajuste(x, y, g+1)*  *print("Equação da reta: ", reta)*  *print("Equação da parábola: ", parabola)*  *yP = []*  *for i in x:*  *yP.append(reta[0]+reta[1]\*i)*  *rEQ = (np.sum((y - yP)\*\*2))\*\*0.5*  *print("Erro quadrático médio da reta: ", rEQ)*  *plt.plot(x, yP, 'b', label='reta', color='r')*  *plt.plot(x, y, 'p', label='pontos', color='b')*  *plt.show("reta-ex201.png")*  *plt.clf()*  *yP = []*  *for i in x:*  *yP.append(parabola[0]+parabola[1]\*i+parabola[2]\*(i\*\*2))*  *pEQ = (np.sum((y - yP)\*\*2))\*\*0.5*  *print("Erro quadrático médio da parábola: ", pEQ)*  *plt.plot(x, yP, 'b', label='reta', color='r')*  *plt.plot(x, y, 'p', label='pontos', color='b')*  *plt.show("parabola-ex201.png")*  ***RESULTADOS:***  *Equação da reta: [0.17500383 0.21666599]*  *Equação da parábola: [0.40706145 0.07742484 0.0154716 ]*  *Erro quadrático médio da reta: 0.2972092417093519*    *Erro quadrático médio da parábola: 0.21930627423982446* |
| --- |

**Exercício 2 -** *Use o método dos quadrados mínimos para ajustar uma curva aos dados da população brasileira entre os anos de 1872 e 1996. Estime a população no ano 2000 e compare com os dados reais do IBGE.*

*Ajuste uma curva na forma de um polinômio de segundo grau y=a+bx+cx² onde y denota a população e x o ano.*

*Tabela da população brasileira (em milhões)*

| **ano** | 1872 | 1890 | 1900 | 1920 | 1940 | 1950 | 1960 | 1970 | 1980 | 1991 | 1996 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **população** | 9.9 | 14.3 | 17.4 | 30.6 | 41.2 | 51.9 | 70.2 | 93.1 | 119.0 | 146.2 | 157.1 |

**R:** A população no Brasil, pelo Método dos Quadrados Mínimos, no ano de 2000, seguindo o parâmetro dos demais valores, é dada por 167,8911. No censo feito pelo IBGE em 2000 no Brasil apontou 175,3 milhões de pessoas. O erro matemático sobre o número de pessoas se dá pelo crescimento exponencial de pessoas, enquanto método contempla apenas funções polinomiais.

**Código implementado:**

| *import numpy as np*  *import matplotlib.pyplot as plt*  *def elimGauss(matriz, vetor, n):*  *m = n*  *x = np.zeros(n)*  *for k in range(len(matriz)):*  *for r in range(k+1,len(matriz)):*  *f=(matriz[r,k]/matriz[k,k])*  *vetor[r]=vetor[r]-(f\*vetor[k])*  *for c in range(0,n):*  *matriz[r,c]=matriz[r,c]-(f\*matriz[k,c])*  *#substituição para trás*  *x[m-1]=vetor[m-1]/matriz[m-1,m-1]*  *for r in range(m-2,-1,-1):*  *suma=0*  *for c in range(0,n):*  *suma=suma+matriz[r,c]\*x[c]*  *x[r]=(vetor[r]-suma)/matriz[r,r]*  *return x*  *def ajuste(x, y, n):*  *X = np.zeros((n,n))*  *Y = np.zeros(n)*  *for i in range(n):*  *Y[i] = np.sum(y\*(x\*\*i))*  *for j in range(n):*  *X[i,j] = np.sum(1\*(x\*\*(j+i)))*  *return elimGauss(X, Y, n)*    *x = np.array([1872, 1890, 1900, 1920, 1940, 1950, 1960, 1970, 1980, 1991, 1996], dtype = "double")*  *y = np.array([9.9, 14.3, 17.4, 30.6, 41.2, 51.9, 70.2, 93.1, 119.0, 146.2, 157.1], dtype = "double")*  *g = 0 #Grau do polinômio*  *g = 2*  *parabola = ajuste(x, y, g+1)*  *print("Equação da parábola: ", parabola)*  *yP = []*  *for i in x:*  *yP.append(parabola[0]+(parabola[1]\*i)+(parabola[2]\*(i\*\*2)))*  *pEQ = (np.sum((y - yP)\*\*2))\*\*0.5*  *print("Erro quadrático médio da parábola: ", pEQ)*  *estimativa = parabola[0]+(parabola[1]\*2000)+(parabola[2]\*(2000\*\*2))*  *print("Estimativa para o ano 2000: ", estimativa)*  *plt.plot(x, yP, 'b', label='reta', color='r')*  *plt.plot(x, y, 'p', label='pontos', color='b')*  *plt.show("parabola-ex202.png")*  ***RESULTADOS:***  *Equação da parábola: [ 4.60447745e+04 -4.87162803e+01 1.28889193e-02]*  *Erro quadrático médio da parábola: 14.19634983677444*  *Estimativa para o ano 2000: 167.89115312960348* |
| --- |

**Exercício 3 -** *Seja os valores da função apresentados na tabela abaixo. Use o Método de Mínimos Quadrados para determinar a equação da curva que melhor ajusta os pontos dados. Antes, faça um gráfico com os pontos para escolha da melhor curva a se ajustar aos pontos.*

**x** -1,0 -0,75 -0,6 -0,5 -0,3 0,0 0,2 0,4 0,5 0,7 1,0

**f(x)** 2,0 1,153 0,45 0,4 0,5 0,0 0,2 0,6 0,512 1,2 2,05

**Código implementado:**

| *PRELIMINAR:*  *import numpy as np*  *import matplotlib.pyplot as plt*    *x = np.array([-1, -0.75, -0.6, -0.5, -0.3, 0, 0.2, 0.4, 0.5, 0.7, 1], dtype = "double")*  *y = np.array([2.0, 1.153, 0.45, 0.4, 0.5, 0, 0.2, 0.6, 0.512, 1.2, 2.05], dtype = "double")*  *plt.plot(x, y, 'p', label='pontos', color='b')*  *plt.show("preliminar-ex203.png")*    Observando a distribuição dos pontos conclui-se que uma parábola se adequaria melhor a distribuição:  *REGRESSÃO:*  *import numpy as np*  *import matplotlib.pyplot as plt*  *from random import randint*  *def elimGauss(matriz, vetor, n):*  *m = n*  *x = np.zeros(n)*  *for k in range(len(matriz)):*  *for r in range(k+1,len(matriz)):*  *f=(matriz[r,k]/matriz[k,k])*  *vetor[r]=vetor[r]-(f\*vetor[k])*  *for c in range(0,n):*  *matriz[r,c]=matriz[r,c]-(f\*matriz[k,c])*  *#substituição para trás*  *x[m-1]=vetor[m-1]/matriz[m-1,m-1]*  *for r in range(m-2,-1,-1):*  *suma=0*  *for c in range(0,n):*  *suma=suma+matriz[r,c]\*x[c]*  *x[r]=(vetor[r]-suma)/matriz[r,r]*  *return x*  *def ajuste(x, y, n):*  *X = np.zeros((n,n))*  *Y = np.zeros(n)*  *for i in range(n):*  *Y[i] = np.sum(y\*(x\*\*i))*  *for j in range(n):*  *X[i,j] = np.sum(1\*(x\*\*(j+i)))*  *return elimGauss(X, Y, n)*    *x = np.array([-1, -0.75, -0.6, -0.5, -0.3, 0, 0.2, 0.4, 0.5, 0.7, 1], dtype = "double")*  *y = np.array([2.0, 1.153, 0.45, 0.4, 0.5, 0, 0.2, 0.6, 0.512, 1.2, 2.05], dtype = "double")*  *g = 0 #Grau do polinômio*  *g = 2*  *parabola = ajuste(x, y, g+1)*  *print("Equação da parábola: ", parabola)*  *yP = []*  *for i in x:*  *yP.append(parabola[0]+(parabola[1]\*i)+(parabola[2]\*(i\*\*2)))*  *pEQ = (np.sum((y - yP)\*\*2))\*\*0.5*  *print("Erro quadrático médio da parábola: ", pEQ)*  *plt.plot(x, yP, 'b', label='reta', color='r')*  *plt.plot(x, y, 'p', label='pontos', color='b')*  *plt.show("parabola-ex203.png")*  ***RESULTADOS:***  *Equação da parábola: [0.09633627 0.1078397 1.91387149]*  *Erro quadrático médio da parábola: 0.479814876341552* |
| --- |