FUNDAÇÃO GETÚLIO VARGAS

ESCOLA DE MATEMÁTICA APLICADA

MESTRADO 2015.1

ÁLGEBRA LINEAR E APLICAÇÕES

Prof Moacyr

Resolução dos Exercícios da Lista 1

KIZZY TERRA

RIO DE JANEIRO

MARÇO DE 2015

1 Exercício 1

Para encontrar um polinômio de grau 3 da forma$P(x) = a.x^3 + b.x^2 + c.x + d$ que passe pelos pontos dados ($P_i = (x, y)$) devemos encontrar os valores de a, b, c, e d que garantem para cada x a respectiva imagem y. Decorre que:

1) Para o ponto $P_1 = (0, 1)$:
$P(0) = a.0^3 + b.0^2 + c.0 + d = 1$
2) Para o ponto $P_2 = (1, 0)$:
$P(1) = a.1^3 + b.1^2 + c.1 + d = 0$
3) Para o ponto $P_3 = (2, -1)$:
$P(2) = a.2^3 + b.2^2 + c.2 + d = -1$
1) Para o ponto $P_4 = (3, 2)$:
$P(3) = a.3^3 + b.3^2 + c.3 + d = 2$

Assim, para garantir que o polinômio passará por todos os pontos fornecidos devemos resolver o seguinte sistema:

$$0a + 0b + 0c + d = 2$$

$$a + b + c + d = 0$$

$$8a + 4b + 2c + d = -1$$

$$27a + 9b + 3c + d = 2$$

Que é equivalente a resolver o sistema $A.x = b$ onde:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 8 & 4 & 2 & 1 \\ 27 & 9 & 3 & 1 \end{bmatrix} \text{ e } b = \begin{bmatrix} 1 & 2 & 3 & 1 \end{bmatrix}^T$$

Resolvendo este sistema atilizando a linguagem GNU octave obtem-se a solução:

ans $=$ 0.66667 -2.00000 0.33333 1.00000

Após encontrar os coeficientes do polinômio foi possível traçar o gráfico pedido (figura a seguir).

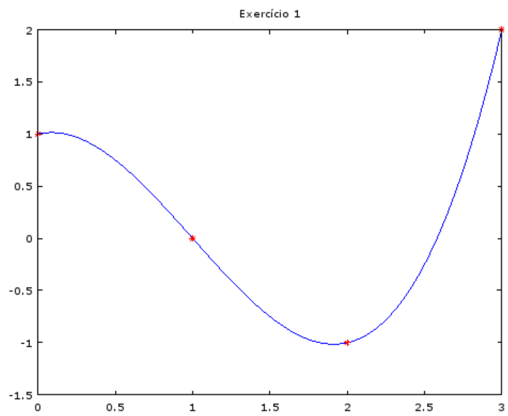

Figura 1: Gráfico com os pontos P_i(em vermelho) e o polinômio interpolador(em azul)

2 Exercício 2

Resolução do sistema $A.x = b$ por eliminação gaussiana:

$$\begin{array}{ccc|c} 1 & 0 & 1 & 1 \\ 2 & 1 & 1 & 2 \\ 3 & 1 & 2 & 3 \\ 3 & 2 & 1 & 1 \end{array}$$

$$L_2 \leftarrow L_2 - 2L_1 \Longrightarrow \begin{array}{ccc|c} (1) & 0 & 1 & 1 \\ 0 & 1 & -1 & 0 \\ 3 & 1 & 2 & 2 \\ 3 & 2 & 1 & 1 \end{array} \qquad L_3 \leftarrow L_3 - 3L_1 \Longrightarrow \begin{array}{ccc|c} (1) & 0 & 1 & 1 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 3 & 2 & 1 & 1 \end{array}$$

$$L_4 \leftarrow L_4 - 3L_1 \Longrightarrow \begin{array}{ccc|c} (1) & 0 & 1 & 1 \\ 0 & (1) & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 2 & -2 & -2 \end{array} \qquad L_3 \leftarrow L_3 - L_2 \Longrightarrow \begin{array}{ccc|c} (1) & 0 & 1 & 1 \\ 0 & (1) & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & -2 & -2 \end{array}$$

$$L_4 \leftarrow L_4 - 2L_2 \Longrightarrow \begin{array}{ccc|c} (1) & 0 & 1 & 1 \\ 0 & (1) & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \end{array}$$

Obs.: Os elementos da matriz assinalados entre ( ) são os pivôs que foram escolhidos durante a eliminação.

A partir da matriz escalonada encontrada podemos concluir que o sistema não possui solução, pois a última equação não pode ser verdadeira:

$$0.x + 0.y + 0.z = -2 \Rightarrow Absurdo$$

3 Exercício 3

Primeiramente iremos verificar se a matriz A possui inversa, através do cálculo do determinante:
$detA = (1.1.2) + (0.0.1) + (-1.1.1) - (1.1.1) - (0.1.1) - (0. - 1.2) = 2 + 0 + (-1) - 1 - 0 - 0 = 0$

Uma vez que verificamos que o determinante de A é nulo, podemos afirmar que a matriz A não possui inversa. Entretanto, iremos verificar este resultado utilizando o método de Gauss-Jordan como segue:

$$\begin{array}{ccc|ccc} 1 & -1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{array}$$

$$L_3 \leftarrow L_3 - L_1 \Longrightarrow \begin{array}{ccc|ccc} (1) & -1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & -1 & 0 & 1 \end{array}$$

$$L_3 \leftarrow L_3 - L_2 \Longrightarrow \begin{array}{ccc|ccc} (1) & -1 & 1 & 1 & 0 & 0 \\ 0 & (1) & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 \end{array}$$

Pode-se observar que não é possível colocar a matriz A na forma reduzida, visto que a última linha foi totalmente zerada e assim, não é possível calcular a inversa pelo método de Gauss-Jordan. Cabe reaaltar, que este resultado já era esperado, visto que a matriz A possui determinante nulo e por essa razão podemos afirmar que esta matriz não possui inversa.

4 Exercício 6

Para resolver este exercício implementou-se uma função em Python a qual recebe uma matriz L_{nxn} (NumPy.matrix) e um vetor b^T_{nx1} (NumPy.array) e retorna um vetor solução x^T_{nx1}.

A idéia do algortimo implementado está em fazer substituição de variáveis a partir da linha 1(que possui um elemento não nulo) até a linha n (com todos os elementos não nulos), considerando o sistema $L.x = b$ os elementos de x podem ser escritos como:

$$x_{i1} = \frac{b_{i1} - \sum_{k=1}^{i-1} L_{ik}.x_{k1}}{L_{ii}}$$

Analisando a complexidade:

1) Para calcular x_{11}:

$$x_{11} = \frac{b_{11} - \sum_{k=1}^{1-1} L_{ik}.x_{k1}}{L_{11}} = \frac{b_{11}}{L_{11}} \Longrightarrow 1 \text{ operação} = 1 \text{ multiplicação}$$

2) Para calcular x_{21}:

$$x_{21} = \frac{b_{21} - \sum_{k=1}^{2-1} L_{ik}.x_{k1}}{L_{22}} = \frac{b_{21} - a_{21}.x_{11}}{L_{22}} \Longrightarrow 3 \text{ operações} = 2 \text{ multiplicações} + 1 \text{ subtração}$$

3) Para calcular x_{31}:

$x_{31} = \dfrac{b_{31} - \sum_{k=1}^{3-1} L_{ik}.x_{k1}}{L_{33}} = \dfrac{b_{31} - a_{31}.x_{11} - a_{32}.x_{21}}{L_{33}} \Longrightarrow$ 5 operações = 3 multiplicações + 2 subtrações

Analisando os itens anteriores e a fórmula para x_{i1}:

$$multiplicações \Longrightarrow 1+2+3+4+5+\cdots+n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

$$subtrações \Longrightarrow \qquad 0+1+2+3+4+\cdots+(n-1) = \frac{(n-1)n}{2} = \frac{n^2-n}{2}$$

\- -

$$operações \Longrightarrow \frac{n^2+n}{2} + \frac{n^2-n}{2} = n^2 = O(n^2)$$

Algoritmo 1 resolverMatrizTriangular(L, b)

```
import numpy as np
import math

def resolverMatrizTriangular(L, b):
    x = []
    try:
        dimensao = b.size
        linhaAtual = 0 #indica linha
        primeiroElemento = b.item(linhaAtual)/L.item((linhaAtual, linhaAtual))
        x.insert(linhaAtual, primeiroElemento)
        linhaAtual += 1

        for linhaAtual in range(1, dimensao):
            colunaAtual = 0 #indica coluna
            elemento = b.item(linhaAtual)
            for colunaAtual in range(0, linhaAtual):
                elemento -= L.item((linhaAtual, colunaAtual))*x[colunaAtual]
                colunaAtual += 1
            elemento /= L.item((linhaAtual, linhaAtual))
            x.insert(linhaAtual, elemento)
            linhaAtual = linhaAtual + 1

    except Exception as e:
        print e
        return x
```

Execução:

```
L = np.matrix ('1 0 0; 2 3 0; 4 5 6')
b = np.array([2,7,19])
resolverMatrizTriangular(L, b)
```

Saída:

```
[2, 1, 1]
```

5 Exercício 7

Os tempos encontrados para a solução dos sistemas foram:

n = 16 ; t = 1.0372e-02

n = 32; t = 3.3808e-04

n = 64; t = 7.1692e-04

n = 128; t = 3.6421e-03

n = 256; t = 1.0221e-02

n = 512; t = 7.5802e-02

n = 1024; t = 5.2819e-01

n = 2048; t = 3.8269e+00

A partir dos resultados obtidos plotou-se o gráfico do log do número de colunas contra o log dos tempos registrados a fim de analisar a relação linear e verificar se o algoritmo possui complexidade inferior a $O(n^3)$. Além deste gráfico foram plotas outros dois para possibilitar a comparação da complexidade como segue:

1) A curva em vermelho evidencia a inclinação de uma curva cúbica quando linearizada

2) A curva em verde evidencia a inclinação de de uma curva quadrática quando linearizada

3) A curva em azul foi obtida como requisitado no exercício

Da análise das três curvas podemos concluir que a complexidade do algoritmo está entre $O(n^2)$ e $O(n^3)$, se considerarmos apenas expoentes inteiros para n então a curva pode ser dita $O(n^3)$, pois não cresce mais do que $k.n^3$ (para algum k).

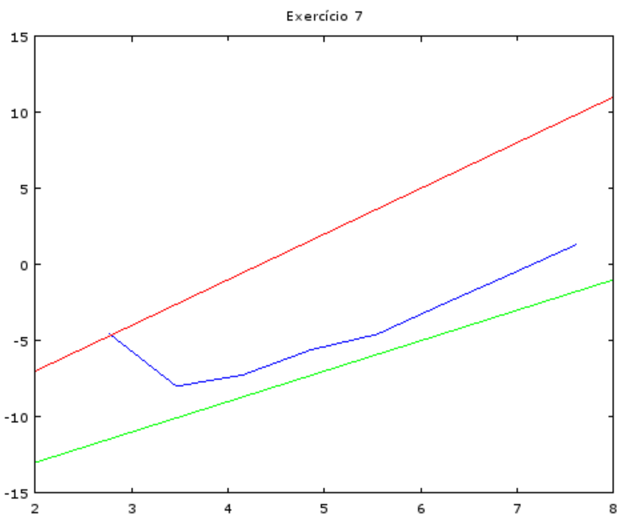

Figura 2: Gráfico pedido em azul, Gráfico de y=3x em vermelho e Gráfico de y=2x em verde