

**ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO**



**MAP3121 - Métodos Numéricos e Aplicações**  
**Exercício Programa 1- GPS e o Método de Newton**

Gabriel de Amorim Auler - 9834671  
Arnaldo Lucas Barbarelli Ferreira Sima - 9834959

## **Sumário**

1. Introdução.....	3
2. Funções.....	4
3. Resultados .....	7
4. Considerações sobre o Programa.....	10

## 1. Introdução

A formação de engenheiro pode conter muitas conexões com outras áreas do conhecimento teórico, incluindo programação. Conseguir modelar e programar um problema, torna sua resolução mais eficiente e a engenharia consegue alcançar seu objetivo.

Assim, o curso de Métodos Numéricos tem como objetivo agregar essa eficiência no raciocínio e trabalho dos futuros engenheiros da Escola Politécnica. Essas características são trabalhadas no exercício programa proposto pelo curso, uma vez que o problema é muito prático e presente no mundo real.

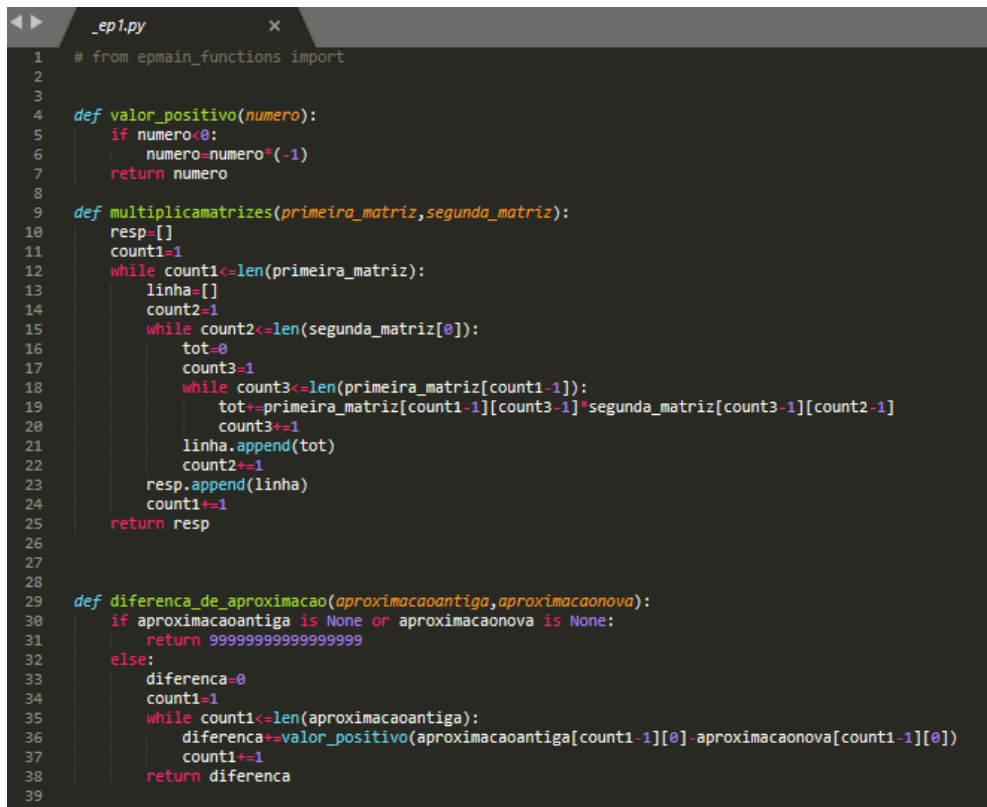
A situação abordada é um GPS(Sistema de Posicionamento Global), algo recorrente no cotidiano de todos em muitos aplicativos e ações simples como, por exemplo, achar o caminho para ir de carro a um lugar desconhecido.

O problema principal é achar uma localização a partir da resolução do sistema não linear  $r_i(x, y, z, w) - r_n(x, y, z, w) = 0, 1 \leq i \leq n - 1$ . Além disso, existem 3 testes que devem ser executados Para isso, o enunciado cita métodos de que podem ser utilizados para essa resolução:

- Mínimos Quadrados Lineares: Quando é suposto que a equação  $r_i(x, y, z, w) = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 - (w_i - w)^2 = 0$  é exata e é resolvida pela multiplicação  $AT Ax = AT b$ .
- Mínimos Quadrados não lineares: quando é necessário diminuir a soma dos quadrados dos resíduos  $r$ , dado que, para isso, o gradiente deve ser nulo.
- Sistema LU: método utilizado para resolver sistemas, apresentado durante as aulas do curso.

## 2. Funções

Todas as funções podem ser vistas na figura 1 e 2.



```
1 # from epmain_functions import
2
3
4 def valor_positivo(numero):
5     if numero<0:
6         numero=numero*(-1)
7     return numero
8
9
10 def multiplicarmatrizes(primeira_matriz,segunda_matriz):
11     resp=[]
12     count1=1
13     while count1<=len(primeira_matriz):
14         linha=[]
15         count2=1
16         while count2<=len(segunda_matriz[0]):
17             tot=0
18             count3=1
19             while count3<=len(primeira_matriz[count1-1]):
20                 tot+=primeira_matriz[count1-1][count3-1]*segunda_matriz[count3-1][count2-1]
21                 count3+=1
22             linha.append(tot)
23             count2+=1
24         resp.append(linha)
25         count1+=1
26     return resp
27
28
29 def diferenca_de_aproximacao(aproximacaoantiga,aproximacaonova):
30     if aproximacaoantiga is None or aproximacaonova is None:
31         return 9999999999999999
32     else:
33         diferenca=0
34         count1=1
35         while count1<=len(aproximacaoantiga):
36             diferenca+=valor_positivo(aproximacaoantiga[count1-1][0]-aproximacaonova[count1-1][0])
37             count1+=1
38     return diferenca
39
```

Figura 1 - Funções valor\_positivo, multiplicarmatrizes e diferenca\_de\_aproximacao

### 2.1. valor\_positivo

Trata-se de uma função cujo objetivo é tirar o módulo de um número, tornando-o sempre positivo.

### 2.2. multiplicarmatrizes

Essa função tem como objetivo multiplicar quaisquer matrizes que necessitem sofrer alguma multiplicação. Logo, a função recebe a primeira matriz, a segunda matriz devolve o resultado da multiplicação das mesmas.

### 2.3. diferenca\_de\_aproximacao

É uma função que recebe as aproximações antiga e nova e retorna a diferença entre as duas.

### 2.4. resolver\_equacao

Trata-se de uma função que visa resolver a equação pelo método LU.

É uma função que recebe as aproximações antiga e nova e retorna a diferença entre as duas.

```

41 # Resolver por LU
42 def resolver_equacao(matriz_esquerda, matriz_direita):
43     matriz_esquerda_copia=[]
44     for i in range(len(matriz_esquerda)):
45         linha=[]
46         for j in range(len(matriz_esquerda[i])):
47             linha.append(matriz_esquerda[i][j])
48         matriz_esquerda_copia.append(linha)
49     matriz_direita_copia=[]
50     for i in range(len(matriz_direita)):
51         linha=[]
52         for j in range(len(matriz_direita[i])):
53             linha.append(matriz_direita[i][j])
54         matriz_direita_copia.append(linha)
55
56
57     equacoes=len(matriz_esquerda_copia)
58     k=1
59     while k<=equacoes:
60         i=k # LU-1
61         while i<=equacoes:
62             j=1
63             soma=0
64             while j<=k-1:
65                 soma+=matriz_esquerda_copia[i-1][j-1]*matriz_esquerda_copia[j-1][k-1]
66                 j+=1
67             matriz_esquerda_copia[i-1][k-1]=matriz_esquerda_copia[i-1][k-1]-soma
68             i+=1
69
70         i=k # LU-2
71         maior=0
72         while i<=equacoes:
73             if valor_positivo(matriz_esquerda_copia[i-1][k-1])>maior:
74                 maior=valor_positivo(matriz_esquerda_copia[i-1][k-1])
75             i+=1
76
77         l=k # LU-3
78         pk=None
79         while l<=equacoes:
80             if valor_positivo(matriz_esquerda_copia[l-1][k-1])==maior:
81                 pk=l
82                 l+=1
83         if pk:
84             if k!=pk: # LU-4
85                 matriz_direita_copia[k-1],matriz_direita_copia[pk-1]=matriz_direita_copia[pk-1],matriz_direita_copia[k-1]
86                 matriz_esquerda_copia[k-1],matriz_esquerda_copia[pk-1]=matriz_esquerda_copia[pk-1],matriz_esquerda_copia[k-1]
87

```

```

87
88     j=k+1 # LU-5
89     while j<=equacoes:
90         i=1
91         soma=0
92         while i<=k-1:
93             soma+=matriz_esquerda_copia[k-1][i-1]*matriz_esquerda_copia[i-1][j-1]
94             i+=1
95         matriz_esquerda_copia[k-1][j-1]=matriz_esquerda_copia[k-1][j-1]-soma
96         matriz_esquerda_copia[j-1][k-1]=matriz_esquerda_copia[j-1][k-1]+matriz_esquerda_copia[k-1][j-1]
97         j+=1
98     k+=1
99
100
101     matriz_l=[] # Obtemos L
102     matriz_u=[] # Obtemos U
103     for i in range(len(matriz_esquerda_copia)):
104         linha_l=[]
105         linha_u=[]
106         for j in range(len(matriz_esquerda_copia[i])):
107             if i==j:
108                 linha_u.append(matriz_esquerda_copia[i][j])
109                 linha_l.append(0)
110             if i>j:
111                 linha_u.append(matriz_esquerda_copia[i][j])
112                 linha_l.append(1)
113             if i<j:
114                 linha_u.append(0)
115                 linha_l.append(matriz_esquerda_copia[i][j])
116         matriz_l.append(linha_l)
117         matriz_u.append(linha_u)
118
119
120     resol=[] # Resolve L
121     count1=1
122     matriz_direita_copia_copia=[]
123     for i in range(len(matriz_direita_copia)):
124         linha_direita_copia_copia=[]
125         for j in range(len(matriz_direita_copia[i])):
126             linha_direita_copia_copia.append(matriz_direita_copia[i][j])
127         matriz_direita_copia_copia.append(linha_direita_copia_copia)
128     while count1<len(matriz_l):
129         count2=count1-1
130         val=matriz_direita_copia_copia[count1-1][0]-matriz_l[count1-1][count1-1]
131         resol.append(val)
132         while count2<len(matriz_l):
133             matriz_direita_copia_copia[count2][0]=matriz_direita_copia_copia[count2][0]-matriz_l[count2][count1-1]*val

```

Figura 2 - Função resolver\_equacao parte 1

```

113         if i>j:
114             linhau.append(0)
115             linha1.append(matriz_esquerda_copia[i][j])
116         matriz1.append(linha1)
117         matrizu.append(linhau)
118
119
120     respl = [] # Resolve L
121     count1=1
122     matriz_direita_copia_copia = []
123     for i in range(len(matriz_direita_copia)):
124         linha_direita_copia_copia = []
125         for j in range(len(matriz_direita_copia[i])):
126             linha_direita_copia_copia.append(matriz_direita_copia[i][j])
127         matriz_direita_copia_copia.append(linha_direita_copia_copia)
128     while count1<len(matriz1):
129         count2=count1-1
130         val = matriz_direita_copia_copia[count1-1][0] * matriz1[count1-1][count1-1]
131         respl.append([val])
132         while count2<len(matriz1):
133             matriz_direita_copia_copia[count2][0] = matriz_direita_copia_copia[count2][0] * matriz1[count2][count1-1] * val
134             count2+=1
135         count1+=1
136
137     respl = []
138     for i in range(len(matriz1)):
139         y_val = matriz_direita_copia[i][0] * matriz1[i][1]
140         respl.append([y_val])
141         count = i + 1
142         while count < len(matriz1):
143             matriz_direita_copia[count][0] = matriz_direita_copia[count][0] * matriz1[count][1] * y_val
144             count += 1
145
146     respu = [] # Resolve U
147     count1 = len(matrizu)-1
148     while count1>=0:
149         val = matriz_direita_copia[count1][0] / matrizu[count1][count1]
150         respu.append([val])
151         count2 = count1-1
152         while count2 >= 0:
153             matriz_direita_copia[count2][0] = matriz_direita_copia[count2][0] - matrizu[count2][count1] * val
154             count2 -= 1
155         count1 -= 1
156     respu.reverse()
157
158     return respu
159
160

```

Figura 3 - Função resolver\_equacao parte 2

### 3. Resultados

#### 3.1. Programa Principal

A resposta do programa principal pode ser visualizada na figura 4, onde são dados os valores de x,y,z e w, bem como a sua latitude e longitude. Essa localização é dada em metros , e posteriormente convertida para graus na Latitude e Longitude.

```

Digite a string da opção: ep

O resultado da primeira aproximação foi:
X=3508022.4191286447
Y=780520.7047600267
Z=5251965.932186775
W=24693.132805298275

O resultado da aproximação foi:
X=3508022.4191286447
Y=780520.7047600267
Z=5251965.932186775
W=24693.132805298275
Houve 2 interações

Raio da Terra=6363849.465262262
Latitude=56.25933587668038
Longitude=12.543765649651911
Erro do GPS: 8.236742501807125e-05 segundos

```

Figura 4 - Resultado do Programa Principal

#### 3.2. Testes

##### 3.2.1. Teste 1

O resultado obtido no teste 1 pode ser visto na figura 5. Nela, é possível ver que o resultado do teste foi  $x = 2.0$  e  $y = 3.0$ . Com isso, pode-se concluir que programa conseguiu receber a função  $F(x,y)$  e achar o ponto onde o gradiente adquire valor nulo.

```

Olá!
O que você deseja que o programa rode...

ep
teste1
teste2
teste3

Digite a string da opção: teste1

Temos  $F(x,y)$  definido por  $(x-2)^2 + (y-3)^2$ 
Logo o gradiente de  $F$  é  $(2x-4, 2y-6)$ 

O resultado do teste1 foi:
X=2.0
Y=3.0

```

Figura 5 - Resultado do teste 1

### 3.2.2. Teste 2

O resultado do teste 2 pode ser observado na figura 6, onde é demonstrado o valor de X, Y, Z e W, bem como o número de iterações. Logo, o programa também foi capaz de conseguir receber a  $F(x_1, x_2, x_3, x_4)$  e determinar a sua raiz.

```

Olá!
O que você deseja que o programa rode...

ep
teste1
teste2
teste3

Digite a string da opção: teste2

Temos  $F(x_1, x_2, x_3, x_4)$  e queremos encontrar o zero do sistema de equações
A aproximação inicial é de  $[1, 1, 1, 1]$ 

O resultado do teste2 foi:
X=0.0
Y=0.7071067811873449
Z=0.7071067811873449
W=1.0000000000000002
O número de iterações foi:5

```

Figura 6 - Resultado do teste 2



### 3.2.3. Teste 3

Podemos ver na figura 7 o resultado obtido pelo teste 3, cujo  $n=20$

```
Digite a string da opção: teste3

Temos um sistema de equação  $(n-1) \times (n-1)$  e queremos encontrar o zero do sistema de equações
A aproximação inicial é nula, de  $[0,0,0,0]$ 
Digite o valor de n desejado: 20

A resposta do teste3 foi:
[-0.0012343937021573896]
[-0.0024969140157446073]
[-0.001259364012803249]
[6.242577661509408e-08]
[-0.0012406361432611927]
[-0.002496960835418462]
[-0.0012531214546502613]
[9.363905508758844e-08]
[-0.0012468787794473765]
[-0.002496976442106469]
[-0.0012468787794473574]
[9.363905508737965e-08]
[-0.0012531214546502804]
[-0.0024969608354184617]
[-0.0012406361432611736]
[6.242577661494289e-08]
[-0.001259364012803268]
[-0.002496914015744607]
[-0.0012343937021573705]
```

Figura 7 - Resultado do teste 3

Para nos assegurar que obtivemos um resultado suficientemente preciso, colocamos estes valores no sistema de equação em questão, resultando nos seguintes dados abaixo (cada linha representa a distância para zero).

```
-0.002468789308191085
-0.0049938358182314265
-0.002518730007271677
1.2485155322528724e-07
-0.002481274209699532
-0.0049939294578710365
-0.0025062448713725903
1.8727811016425525e-07
-0.002493759501470647
```

-0.004993960671344353  
-0.002493759501470609  
1.8727811016425525e-07  
-0.002506244871372629  
-0.004993929457871036  
-0.0024812742096994938  
1.2485155322485356e-07  
-0.002518730007271715  
-0.004993835818231426  
-0.002468789308191047

Os testes então foram realizados para  $n=40$  e  $n=80$ .

#### **4. Considerações sobre o programa**

O programa inteiro foi feito utilizando uma linha lógica de raciocínio de acordo com o que cada teste requisitava. Dessa forma, não houve uma divisão clara de cada etapa, porém o exercício programa contém comentários que podem ajudar a entender como os alunos elaboraram a ordem e as escolhas que resultaram na programação entregue.

Durante todo o exercício programa, o conteúdo ensinado em aula e consultado em materiais fornecidos no site da matéria foi extremamente presente, mostrando que os métodos numéricos podem ser aplicados a situações práticas do cotidiano.