Projeto de Matemática Computacional:Partes I e II

Autores:
Inês Agostinho (ist192810)
João Belo (ist196994)
Mariana Nunes (ist194164)
Miguel Fernandes (ist192836)

Professores:
Professora Ana Leonor Silvestre
Professora Adélia da Costa Sequeira dos Ramos Silva
Professora Sonia Seyedallaei

Data: 23 de Dezembro de 2019

Conteúdo

1 I

1.1 Exercício 1.

1.1.1 Exercício 1. (a)

Considere-se o método de Steffensen um método um método do ponto fixo tal que x$_{n+1}$ = g(x$_n$) onde

$$g(x_n) = x_n - \frac{(f(x_n))^2}{f(x_n + f(x_n)) - f(x_n)}[1]$$

Seja f uma função de classe C^2 tal que $f(z) = 0$ e suficientemente regular num intervalo (I) $x_n, z \in I$. z é o ponto fixo da função g tal que $g(z) = z$. Queremos provar que z é um ponto fixo superatrator para garantir a convergência local do método.

$$g'(z) = \lim_{x\to z} \frac{g(x) - g(z)}{x - z} = \lim_{x\to z} \frac{x - \frac{(f(x))^2}{f(x+f(x))-f(x)} - z}{x - z}$$

$$= \lim_{x\to z} \left(\frac{x-z}{x-z} - \frac{f(x)^2}{(f(x+f(x)) - f(x))(x-z)} \right) = 1 - \lim_{x\to z} \frac{\frac{f(x)^2}{f(x+f(x))-f(x)}}{x-z}[2]$$

$$\lim_{x\to z} \frac{\frac{f(x)^2}{f(x+f(x))-f(x)}}{x-z} = \lim_{x\to z} \frac{\frac{2f(x)f'(x)[f(x+f(x))-f(x)]-f(x)^2[f'(x+f(x))(1+f'(x))-f'(x)]}{[f(x+f(x))-f(x)]^2}}{1}$$

$$= \lim_{x\to z} \frac{2f(x)f'(x)[f(x+f(x)) - f(x)] - (f(x))^2[f'(x+f(x))(1+f'(x)) - f'(x)]}{[f(x+f(x)) - f(x)]^2}$$

$$= \lim_{x\to z} 2f'(x)H - H^2[f'(x+f(x))(1+f'(x)) - f'(x)](*)$$

onde

$$H = \frac{f(x)}{f(x+f(x)) - f(x)}$$

$$\lim_{x\to z} H = \lim_{x\to z} \frac{f(x)}{f(x+f(x)) - f(x)} = \lim_{x\to z} \frac{f(x)}{f'(x+f(x))(1+f'(x)) - f'(x)}$$

$$= \frac{f'(z)}{f'(z+f(z)(1+f'(z))-f'(z)} = \frac{f'(z)}{f'(z)[1+f'(z)-1]} = \frac{1}{f'(z)} \wedge f'(z) \neq 0$$

$$(*) \lim_{x\to z} 2f'(x)\frac{1}{f'(x)} - \frac{1}{(f'(x))^2}[f'(x+f(x))(1+f'(x))-f'(x)] =$$

$$= 2 - \frac{1}{(f'(z))^2}[f'(z+f(z))(1+f'(z))-f'(z)] = 2 - \frac{1}{(f'(z))^2}[f'(z)+(f'(z))^2-f'(z)]$$

$$= 2 - 1 = 1[3]$$

Substituindo em [2] o limite dado por [3] obtemos:

$$= 1 - 1 = 0$$

Como z é ponto fixo superatrator então o método é localmente convergente par z.
Sabemos que f é uma função de classe C^2 tal que $f(z) = 0$ e $f'(z) \neq 0$.Para determinar a ordem de convergência do método façamos a expansão de Taylor de $f[x_n + f(x_n)]$ em torno de x_n:

$$f[x_n + f(x_n)] = f(x_n) + f'(x_n)f(x_n) + \frac{f''(\xi)}{2}f(x_n)^2$$

onde ξ é um ponto entre x_n e $x_n + f(x_n)$.

$$\frac{f(x_n)^2}{f[x_n+f(x_n)]} = \frac{f(x_n)}{f'(x_n) + \frac{f''(\xi)}{2}f(x_n)}$$

Sendo que o primeiro membro por [1] é igual a $x_n - x_{n+1}$ temos:

$$x_n - x_{n+1} = \frac{f(x_n)}{f'(x_n) + \frac{f''(\xi)}{2}f(x_n)} \Leftrightarrow z - z + x_n - x_{n+1} = \frac{f(x_n)}{f'(x_n) + \frac{f''(\xi)}{2}f(x_n)}$$

$$\Leftrightarrow z - x_{n+1} = z - x_n + \frac{f(x_n)}{f'(x_n) + \frac{f''(\xi)}{2}f(x_n)}$$

Reduzindo o segundo membro ao mesmo denominador ficamos com:

$$z - x_{n+1} = \frac{f(x_n) + f'(x_n)(z-x_n) + \frac{f''(\xi)}{2}f(x_n)(z-x_n)}{f'(x_n) + \frac{f''(\xi)}{2}f(x_n)}[4]$$

Desenvolvendo a série de Taylor de $f(z)$ em torno de x_n temos:

$$0 = f(z) = f(x_n) + f'(x_n)(z - x_n) + \frac{f''(\xi*)}{2}(z - x_n)^2[5]$$

onde $\xi*$ é um ponto entre z e x_n. Por outro lado a série de Taylor de $f(z)$ em torno de x_n pode ser feita apenas até à primeira ordem:

$$0 = f(z) = f(x_n) + \frac{f'(\xi')}{2}(z - x_n)[6]$$

onde ξ' é um ponto entre z e x_n.

Substituindo o $f(x_n)$ da expressão [5] e [6] em [4] obtemos a finalmente:

$$z - x_{n+1} = -\frac{\frac{f''(\xi*)}{2}(z - x_n)^2 + \frac{f''(\xi)}{2}f'(\xi')(z - x_n)^2}{f'(x_n) + \frac{f''(\xi)}{2}f(x_n)}$$

Fazendo o limite da expressão:

$$lim_{x \to z}\frac{|z - x_{n+1}|}{|z - x_n|^2} = lim_{x \to z}\left|\frac{\frac{f''\xi*}{2}(z - x_n)^2 + \frac{f''\xi}{2}f'(\xi')(z - x_n)^2}{f'(x_n) + \frac{f''(\xi)}{2}f(x_n)}\right| = \frac{1}{2}\left|\frac{f''(z)}{f'(z)}\right|\left|1 + f'(z)\right|$$

Desta forma encontra-se o factor assintótico do método e conclui-se que o método tem ordem de convergência 2. Como o fator assintótico tem de ser maior que 0, $k_\infty > 0$ então $f''(z) \neq 0$.

1.1.2 Exercício 1. (b)

Considerando um método de convergência supralinear sabemos que:

$$lim_{n \to \infty} = \frac{|z - x_{n+1}|}{|z - x_n|^p} = \frac{\left|g^{(p)}(z)\right|}{p!} \Leftrightarrow |z - x_{n+1}| \leq max\frac{\left|g^{(p)}(z)\right|}{p!}|z - x_n|^p$$

onde $x \in int(x_n, z)$. Se $k = max\frac{\left|g^{(p)}(z)\right|}{p!}|z - x_n|^p$ então:

$$|z - x_{n+1}| \leq k|z - x_n|^p\,(7)$$

$|e_{n+1}| = |z - x_{n+1}|$; $|e_n| = |z - x_n|$

$$\Leftrightarrow |e_{n+1}| \leq k|e_n|^p\,(8)$$

A sucessão dos erros tende mais rapidamente para 0 quanto maior a ordem de iteração:

$$lim_{n\to\infty} |e_n| = 0 \Rightarrow lim_{n\to\infty} |e_n|^{p-1} = 0$$

Assim, existe uma ordem a partir da qual temos que: $k\,|e_n|^{p-1} \leq C(9)$.
Multiplicado (9) de ambos os lados por$|e_n|$ obtemos:

$$k\,|e_n|^p \leq C\,|e_n|$$

Por (8) e aplicando a desigualdade triângular ficamos com:

$$|z - x_{n+1}| = |e_{n+1}| \leq k\,|e_n|^p \leq C\,|e_n| = C\,|z - x_n| \Rightarrow$$

$$\Rightarrow |z - x_{n+1}| \leq C\,|z - x_n| = C\,|z - x_{n+1} + x_{n+1} - x_n| \leq C\,|z - x_{n+1}| + C\,|x_{n+1} - x_n| \Rightarrow$$

$$\Rightarrow (1 - C)\,|z - x_{n+1}| \leq C\,|x_{n+1} - x_n| \Leftrightarrow |z - x_{n+1}| \leq \frac{C}{1 - C}\,|x_{n+1} - x_n|\,(10)$$

Se em (10) considerarmos $C \leq \frac{1}{2}$, o quociente $\frac{C}{1-C} \leq 1$ e obtemos a desigualdade pretendida:

$$|z - x_{n+1}| \leq |x_{n+1} - x_n|$$

1.1.3 Exercício 1. (c)

A implementação do Método de Steffensen foi feita usando a linguagem de programação Python. Os dados de entrada deste método são a função f, a aproximação inicial x0, um número máximo de iterações M e uma tolerância de erro e.
O programa correrá até se atingir um erro dentro da tolerância ou até atingir o número máximo de iterações estabelecido.
O critério de paragem que impõe a tolerância de erro ε é $\varepsilon\,|x_n - x_{n+1}|$.

```
def steffensen(f, x0, M, e):  # implementação do método de
   Steffensen
  x=[float(x0)] # conjunto das iteradas com a iterada inicial
  print ('Aproximação incial: %s' %(str(x[0])))

  def iteracao(x,f): # iteração do método
    return float(x - (f(x))**2.0 / (f(x+f(x)) - f(x)))

  for i in range(1, M+1): #ciclo iterativo
    x+=[iteracao(x[-1],f)] # nova iteração
    erro = abs(x[-1]-x[-2]) # cálculo do erro
```

```
    print ('Iterada n.  %s: %s, Erro: %s' %(str(i), str(x[-1]),
       str(erro)))
    if erro<e: # verifica se o erro é menor que a tolerância
      break # se sim pára o ciclo
  return x
```

```
steffensen(lambda x: 545.0*x+4+3333*x**3+7*x**3,0.1,20,.00000001)
Out: Aproximação incial: 0.1
Iterada n.  1: 0.09999518207036494, Erro: 4.817929635067553e-06
Iterada n.  2: 0.0999903638986046, Erro: 4.818171760331835e-06
Iterada n.  3: 0.09998554548468612, Erro: 4.818413918486475e-06
Iterada n.  4: 0.09998072682857656, Erro: 4.81865610955922 7e-06
Iterada n.  5: 0.09997590793024302, Erro: 4.8188983335362146e-06
Iterada n.  6: 0.09997108878965258, Erro: 4.819140590445192e-06
Iterada n.  7: 0.0999662694067723, Erro: 4.819382880272283e-06
Iterada n.  8: 0.09996144978156926, Erro: 4.819625203045241e-06
Iterada n.  9: 0.0999566299140105, Erro: 4.819867558764068e-06
Iterada n.  10: 0.09995180980406307, Erro: 4.820109947428763e-06
Iterada n.  11: 0.09994698945169402, Erro: 4.820352369053205e-06
Iterada n.  12: 0.09994216885687036, Erro: 4.82059482365127e-06
Iterada n.  13: 0.09993734801955916, Erro: 4.820837311209081e-06
Iterada n.  14: 0.0999325269397274, Erro: 4.8210798317543935e-06
Iterada n.  15: 0.0999277056173421, Erro: 4.821322385301086e-06
Iterada n.  16: 0.09992288405237026, Erro: 4.821564971835279e-06
Iterada n.  17: 0.0999180622447789, Erro: 4.8218075913708525e-06
Iterada n.  18: 0.09991324019453497, Erro: 4.822050243921683e-06
Iterada n.  19: 0.09990841790160547, Erro: 4.822292929501648e-06
Iterada n.  20: 0.09990359536595737, Erro: 4.822535648096871e-06
[0.1,
 0.09999518207036494,
 0.0999903638986046,
 0.09998554548468612,
 0.09998072682857656,
 0.09997590793024302,
 0.09997108878965258,
 0.0999662694067723,
 0.09996144978156926,
 0.0999566299140105,
 0.09995180980406307,
```

```
0.09994698945169402,
0.09994216885687036,
0.09993734801955916,
0.0999325269397274,
0.0999277056173421,
0.09992288405237026,
0.0999180622447789,
0.09991324019453497,
0.09990841790160547,
0.09990359536595737]
```

1.2 Exercício 2.

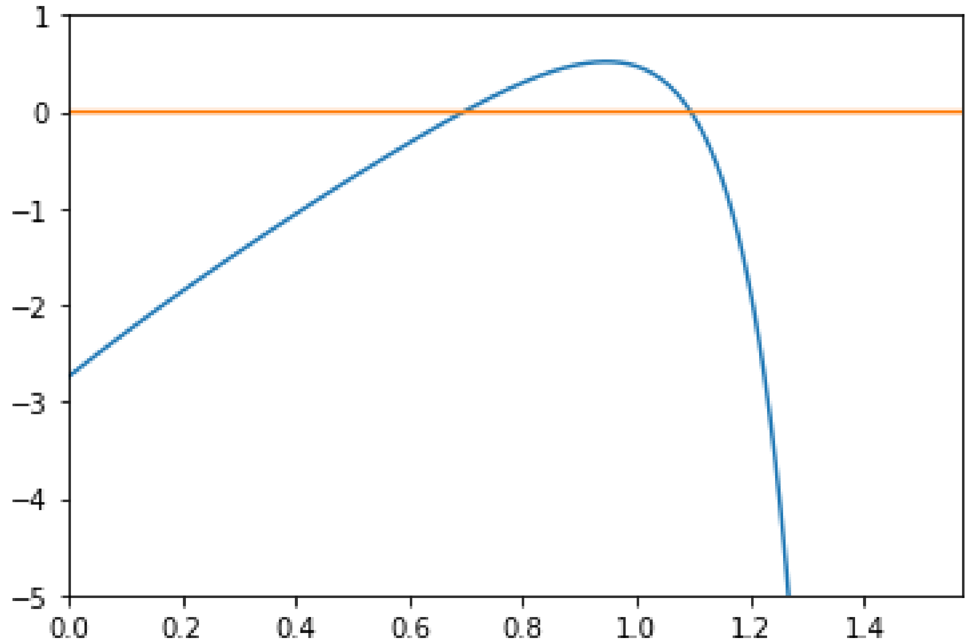
1.2.1 Exercício 2. (a)

```
In [56]: from pylab import *
         import matplotlib.pyplot as plt
         from numpy import *
```

Seja f(x) uma função que recebe o ângulo de lançamento da bola. Dado que x = 4.7 metros, esta função anula-se quando a altura da bola é igual a 3.05 metros (altura do cesto).

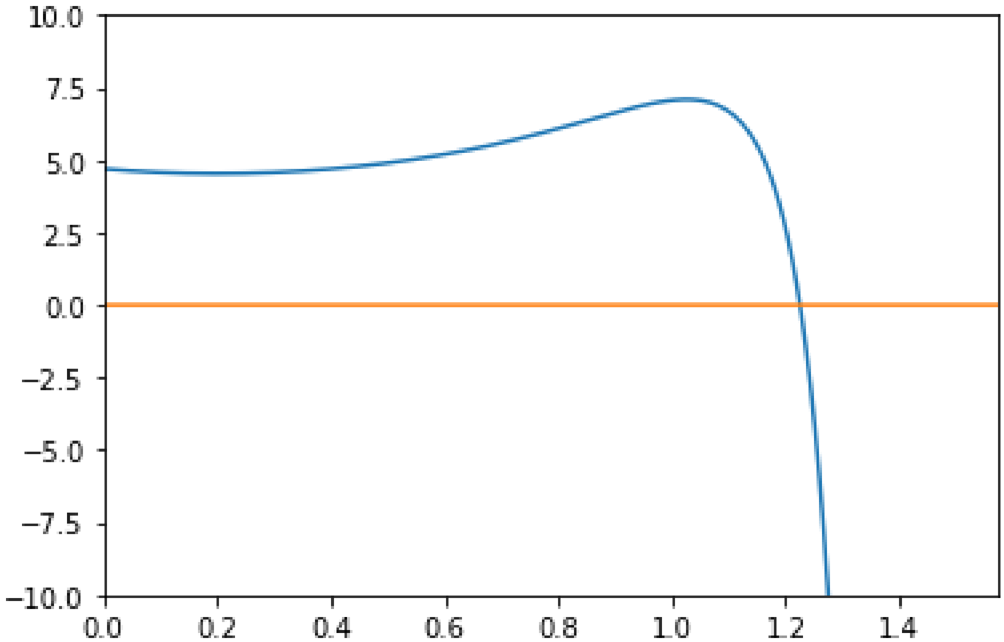
```
In [57]: def f(x):
             return -3.05+2+4.7*tan(x)-((9.81*4.7*4.7)/(2*8*8*cos(x)*cos(x)))
```

```
In [58]: plt.axis((0,pi/2,-5,1)) #Especificação dos limites do eixo das
abcissas e do eixo das coordenadas, respetivamente
         x=linspace(0,pi/2,2000)
         plt.plot(x,f(x))
         plt.plot(x,0*x)
         plt.show() #Exibir os dois gráficos
```



Considera-se que os ângulos de lançamento se encontram no intervalo [0, pi/2], visto que um ângulo superior a pi/2 significaria lançar a bola para trás, o que não é consistente com este problema. Com base no gráfico anterior, é possível observar que existem duas soluções para f(x)=0 no intervalo [0, pi/2], pois existem duas interseções desta função com y=0.

```
In [59]: def derivada_f(x):
             return (4.7/((cos(x))**2))-((9.81*4.7*4.7*tan(x))/
         (2*8*8*cos(x)*cos(x)))

         plt.axis((0,pi/2,-10,10))
         plt.plot(x,derivada_f(x))
         plt.plot(x,0*x)
         plt.show()
```




```
In [60]: f(0.6)
```

```
Out[60]: -0.31994147588786426
```

```
In [61]: f(0.8)
```

```
Out[61]: 0.30147556262560693
```

Como f(0.6)f(0.8)<0 e a derivada de f(x) não se anula neste intervalo, é possível concluir que uma das soluções de f(x)=0 se encontra no intervalo]0.6,0.8[.

```
In [62]: f(1.0)
```

```
Out[62]: 0.47044238029102026
```

```
In [63]: f(1.2)
```

```
Out[63]: -1.8546489227974998
```

Como f(1.0)f(1.2)<0 e a derivada de f(x) não se anula neste intervalo, é possível concluir que a outra solução de f(x)=0 se encontra no intervalo]1.0,1.2[.

Para encontrar as soluções de f(x)=0, utilizar-se-á o método de Steffensen sendo f a função f(x), x0 a iterada inicial, M o número máximo de iterações e "e"o erro definido.

Solução de f(x)=0 em]0.6;0.8[e respetivo gráfico da trajetória da bola:

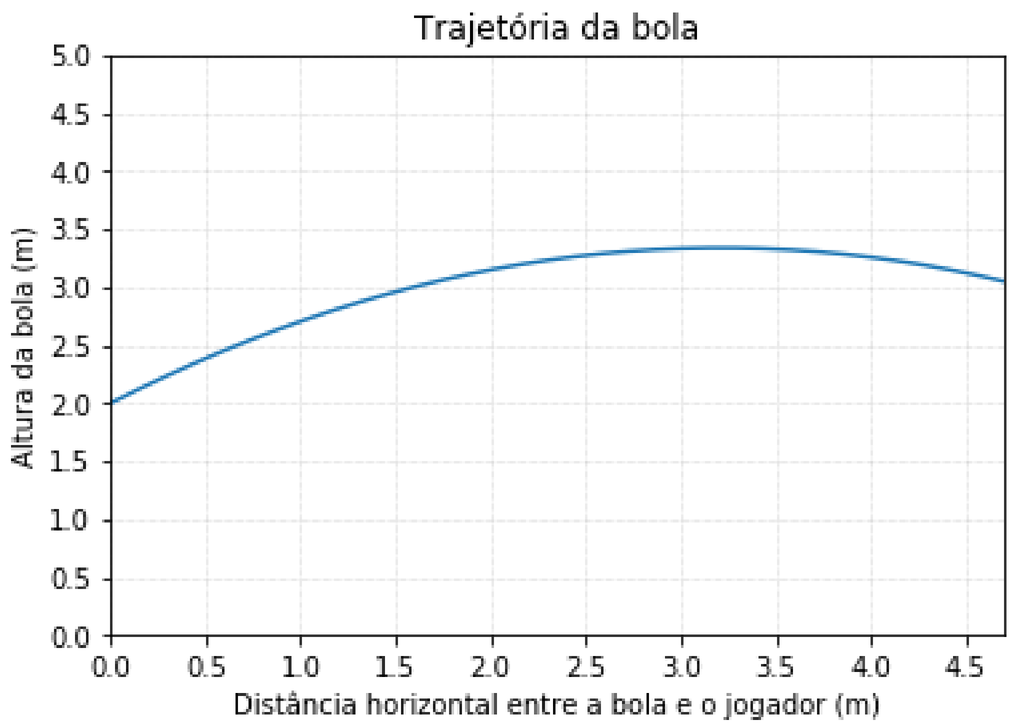
Para a função f(x), x0=0.3, M=20 e e=0.0000001

```
In [65]: steffensen(f,0.3,20,0.000000000001)

Iteração n.º 0: 0.40358194878839293, Erro: 0.10358194878839294
Iteração n.º 1: 0.5810822390843385, Erro: 0.1775002902959456
Iteração n.º 2: 0.6809592226052681, Erro: 0.09987698352092955
Iteração n.º 3: 0.6948725775000485, Erro: 0.013913354894780405
Iteração n.º 4: 0.6953703738657854, Erro: 0.00049779636573688
Iteração n.º 5: 0.6953710931462714, Erro: 7.192804860611801e-07
Iteração n.º 6: 0.6953710931477795, Erro: 1.5080159343483501e-12
Iteração n.º 7: 0.6953710931477798, Erro: 3.3306690738754696e-16


Out[65]: 0.6953710931477798

In [66]: x7=0.6953710931477798
```

Seja h(x) a função que relaciona a altura com a distância percorrida pela bola com as seguintes condições iniciais: a altura no instante do lançamento é igual a 2; o ângulo no instante do lançamento é igual à última iterada calculada pelo método de steffensen; e a magnitude da velocidade com que a bola é lançada é igual a 8 m/s.

```
In [68]: def h(x):
             return 2+x*tan(x7)-((9.81*x**2)/(2*8*8*cos(x7)*cos(x7)))
         x = linspace(0,4.7,1000)

         plt.title('Trajetória da bola') #Título do gráfico
         plt.xlabel('Distância horizontal entre a bola e o jogador (m)')
         #Nome do eixo das abcissas
         plt.ylabel('Altura da bola (m)') #Nome do eixo das ordenadas
         plt.axis((0,4.7,0,5)) #Especificação dos limites do eixo das
         abcissas e do eixo das coordenadas, respetivamente

         plt.yticks(arange(0,5.1,0.5)) #para controlar o número e o
         espaçamento de marcas nos eixos (step=0.5)
```

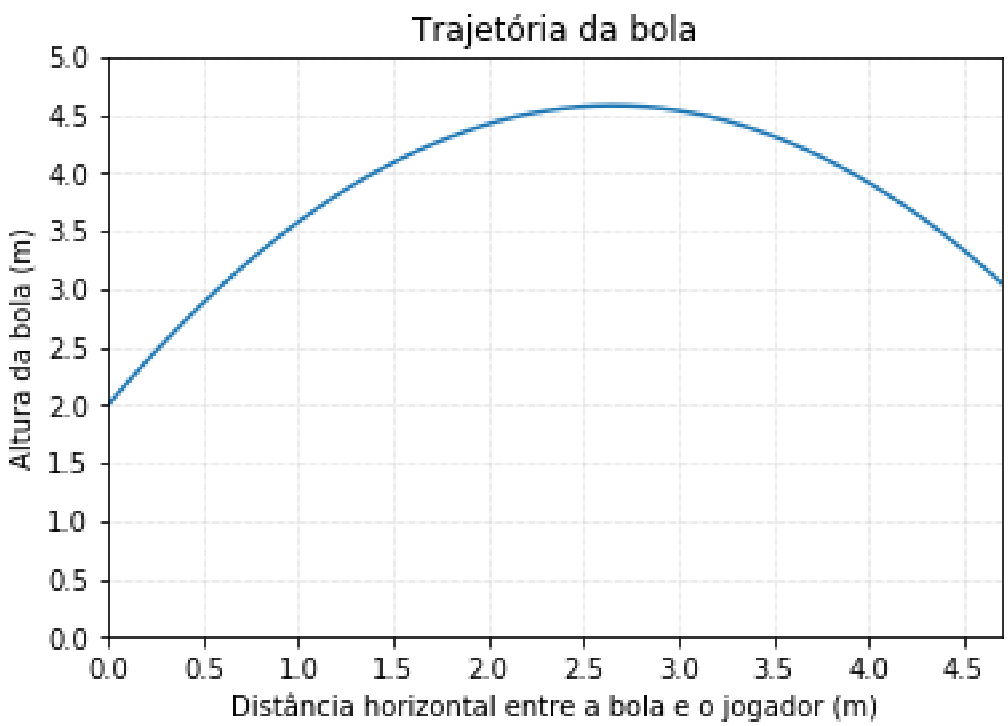
```
plt.xticks(arange(0,4.7,0.5))
plt.grid(linestyle=':', linewidth=0.5) #Grelha a linha pontilhada

plt.plot(x,h(x)) #Criar o gráfico
plt.show() #Exibir o gráfico
```



Solução de f(x)=0 em]1.0;1.2[e respetivo gráfico da trajetória da bola:

```
In [69]: steffensen(f,1.1,20,0.00000000001)

Iteração n.º 0: 1.093921779497973, Erro: 0.00607822050202711
Iteração n.º 1: 1.095139486280463, Erro: 0.0012177067824901133
Iteração n.º 2: 1.095219981638653, Erro: 8.04953581898804e-05
Iteração n.º 3: 1.095220305165953, Erro: 3.2352730006124375e-07
Iteração n.º 4: 1.095220305171149, Erro: 5.196065799850658e-12


Out[69]: 1.095220305171149
```

```
In [70]: x4=1.095220305171149

In [71]: def h(x):
             return 2+x*tan(x4)-((9.81*x**2)/(2*8*8*cos(x4)*cos(x4)))

         plt.title('Trajetória da bola')
         plt.xlabel('Distância horizontal entre a bola e o jogador (m)')
         plt.ylabel('Altura da bola (m)')
         plt.axis((0,4.7,0,5))

         plt.yticks(arange(0,5.1,0.5))
         plt.xticks(arange(0,4.7,0.5))
         plt.grid(linestyle=':', linewidth=0.5)

         plt.plot(x,h(x))
         plt.show()
```



1.2.2 Exercício 2. (b)

Para verificar que a solução no intervalo]0.6;0.8[está de acordo com a estimativa |z - xn+1| $\leq$ |xn+1 - xn|, foram tomados os valores z=x7 e n=5.

```
In [72]: x5 = 0.6953710931462714
         x6 = 0.6953710931477795
         x7 = 0.6953710931477798
```

```
In [73]: x7-x6
```

```
Out[73]: 3.3306690738754696e-16
```

```
In [74]: x6-x5
```

```
Out[74]: 1.5080159343483501e-12
```

Verifica-se então que |z - xn+1| $\leq$ |xn+1 - xn| no intervalo]0.6,0.8[para z=x7 e n=5, conforme pretendido.

Para verificar que a solução no intervalo]1.0;1.2[está de acordo com a estimativa |z - xn+1| $\leq$ |xn+1 - xn|, foram tomados os valores z=x4 e n=2.

```
In [75]: x2 = 1.095219981638653
         x3 = 1.095220305165953
         x4 = 1.095220305171149
```

```
In [76]: x4-x3
```

```
Out[76]: 5.196065799850658e-12
```

```
In [77]: x3-x2
```

```
Out[77]: 3.2352730006124375e-07
```

Verifica-se então que |z - xn+1| $\leq$ |xn+1 - xn| no intervalo]1.0,1.2[para z=x4 e n=2, conforme pretendido.

2 II

2.1 Exercício 1.

2.1.1 Exercício 1. (a)

Fazendo k=1 na expressão de $R_{j,k}$ obtemos:

$$R_{j,1} = \frac{4R_{j,0} - R_{j-1,0}}{3}[1]$$

Sabemos que $R_{j,0} = T_{m(j)}(f)$ para $j = 0, 1, ..., n$. Desta forma:

$$R_{m(j,0)}(f) = T_{m(j)}(f)[2]$$

e

$$R_{m(j-1,0)}(f) = T_{m(j-1)}(f)[3]$$

Aplicando a fórmula dos trapézios a [2]:

$$T_{m(j)}(f) = \frac{h}{2}[f(x_0) + f(x_j) + 2\sum_{i=1}^{m_{(j)}-1} f(x_i)]$$

onde $h = \frac{b-a}{m}$ e $m_{(j)} = 2^j$

$$\Leftrightarrow\ T_{m(j)}(f) = \frac{b-a}{2m}[f(x_0) + f(x_m) + 2\sum_{i=1}^{m-1} f(x_i)][4]$$

Aplicando a fórmula dos trapézios a [3]:

$$T_{m(j-1)}(f) = \frac{h}{2}[f(x_0') + f(x_{m(j-1)}) + 2\sum_{i=1}^{m_{(j-1)}-1} f(x_i)]$$

onde $h = \frac{b-a}{2m_{j-1}}$ e $m_{j-1} = 2^(j-1) = \frac{2^j}{2} = \frac{m}{2}$

$$\Leftrightarrow\ T_{m(j-1)}(f) = \frac{b-a}{m}[f(x_0') + f(x'_{\frac{m}{2}}) + 2\sum_{i=1}^{\frac{m}{2}-1} f(x_i)][5]$$

Procuramos agora relacionar os nós de integração de [2] e [3].

Os nós de integração de [2] são dados por: $x_i = a + ih$ onde $h = \frac{b-a}{m}$

Os nós de integração de [3] são dados por: $x'_i = a + ih'$ onde $h' = \frac{b-a}{\frac{m}{2}} \Leftrightarrow \mathrm{x}'_i = a + 2ih$

Desta forma podemos relacionar os nós de integração de [2] e [3] sendo que:$x'_i = x_{2i}$.Consequentemente obtemos que [5] pode ser dado por a expressão equivalente:

$$T_{m(j-1)}(f) = \frac{b-a}{m}[f(x_0) + f(x_m) + 2\sum_{i=1}^{\frac{m}{2}-1} f(x_{2i})][6]$$

Fazendo a substituição de [4] e [6] em [1] obtemos:

$$R_{j,1} = \frac{1}{3}\left[\frac{4(b-a)}{2m}[f(x_0) + f(x_m) + 2\sum_{i=1}^{m-1} f(x_i)] - \frac{b-a}{m}[f(x_0) + f(x_m) + 2\sum_{i=1}^{\frac{m}{2}-1} f(x_{2i})]\right] =$$

$$= \frac{1}{3}\left[2h[f(x_0) + f(x_m) + 2\sum_{i=1}^{m-1} f(x_i)] - h[f(x_0) + f(x_m) + 2\sum_{i=1}^{\frac{m}{2}-1} f(x_{2i})]\right] =$$

$$= \frac{h}{3}\left[f(x_0) + f(x_m) + 4\sum_{i=1}^{m-1} f(x_i)] - 2\sum_{i=1}^{\frac{m}{2}-1} f(x_{2i})]\right] =$$

$$= \frac{h}{3}[f(x_0) + f(x_m) + 4f(x_1) + 4f(x_2) + ... + 4f(x_{m-1}) - 2f(x_2) - 2f(x_4) - ... - 2f(x_{m-2})] =$$

$$= \frac{h}{3}[f(x_0) + f(x_m) + 4f(x_1) + ... + 4f(x_{m-1}) + 2f(x_2) + 2f(x_4) + ... + 2f(x_{m-2})] =$$

$$= \frac{h}{3}\left[f(x_0) + f(x_m) + 4\sum_{i=1}^{\frac{m}{2}} f(x_{2i-1}) + 2\sum_{i=1}^{\frac{m}{2}-1} f(x_{2i})\right] = S_m(f) = S_{2^j}(f)$$

Prova-se deste modo que $R_{j,1} = S_{2^j}$

2.1.2 Exercício 1. (b)

No âmbito deste exercício fez uma implementação da regra dos trapézios na linguagem Python, que recebe como argumentos os extremos de integração a e b, uma função f e o número de suintervalos, N.

```
def trap(f, a, b, N): # Implementação da Regra dos Trapézios
    h=(b-a)/N # Cálculo do comprimento h do subintervalo
    return ((h/2)*(f(a)+f(b)+2*sum([f(a+i*h) for i in range(1,N)])))
```

```
import math
trap(lambda x:math.e**(x) ,-1, 1, 4)
Out: 2.3991662826140026
```

2.1.3 Exercício 1. (c)

Usando a função criada na última alínea implementou-se o método de Romberg na mesma linguagem:

```
def romberg(f,a,b,N,e):
    r=[[' ' for j in range(0, N+1)] for k in range(0, N+1)] #
       cria uma lista com n+1 sublistas com n+1 elementos ' '
       para guardar os valores Rj,k na forma r[k][j]
    r[0]=[trap(f, a, b, 2**j) for j in range(0, N+1)] # calcula R
        para k = 0 e todos os j, estes ficam na primeira lista
    for k in range(1, N+1):
        for j in range(k, N+1):
            r[k][j]=((4**k) * r[k-1][j] - r[k-1][j-1])/(4**k-1) #
                iteração do Método de Romberg
        if abs(r[k][k] - r[k-1][k-1])<e: # cálculo do erro
            return r[k][k] # devolve o último valor calculado
            break # sai do ciclo for
    return r[k][k] # caso o primeiro critério de paragem
       referente ao erro não se tenha verificado o último valor
       calculado é devolvido
```

```
romberg(lambda x: math.e**(x**2),0,2,4,10**(-1))
Out: 16.452917565112486
```

2.2 Exercício 2.

2.2.1 Exercício 2. (a)

Nesta parte do projeto pretende-se obter uma aproximação de π usando o método de Romberg implementado na alínea anterior para o cálculo da integral $4\int_0^1 \frac{1}{1+x^2}dx = \pi$.

Assim, foram criadas duas funções, uma que devolvia os valores calculados para π e outra que devolvia os erros absolutos associados a estes cálculos:

```
import math
import numpy as np

from decimal import * # valores com mais precisão
ctx = Context(prec=20) #precisão de 20 dígitos

getcontext().prec = 28
def rombergpi(f, a, b, N): # dados para a tabela do pi calculado
  r=[[0 for j in range(0, N+1)] for k in range(0, N+1)] #r[k][j]
  r[0]=[Decimal(trap(f, a, b, 2**j)) for j in range(0, N+1)]
  for k in range(1, N+1):
    for j in range(k, N+1):
      r[k][j]=((Decimal(4**k)) * r[k-1][j] - r[k-1][j-1])/(
         Decimal(4**k-1)) # iteração do Método de Romberg
  return [[float(4*r[k][j]) if j>=k else 0 for j in range(0, N+1)
     ] for k in range(0, N+1)] # calcula um valor aproximado para
      pi aplicando o método de Romberg

np.savetxt('valorespi.txt', rombergpi(lambda x: 1/(1+x**2), 0, 1,
    16)) # guarda os valores num ficheiro de texto


def rombergpierros(f, a, b, N): # cálculo dos erros absolutos
  r=[[0 for j in range(0, N+1)] for k in range(0, N+1)] #r[k][j]
  r[0]=[Decimal(trap(f, a, b, 2**j)) for j in range(0,N+1)]
  for k in range(1, N+1):
    for j in range(k, N+1):
      r[k][j]=((Decimal(4**k)) * r[k-1][j] - r[k-1][j-1])/(
         Decimal(4**k-1))
  return [[float(abs(4*r[k][j] - Decimal(math.pi))) if j>=k else
     0 for j in range(0, N+1)] for k in range(0, N+1)] # calcula
     o desvio em relação ao pi

np.savetxt('erros.txt', rombergpierros(lambda x: 1/(1+x**2), 0,
   1, 16)) # guarda os valores num ficheiro de texto
```

Os valores obtidos são de seguida apresentados em tabelas.

j	valor calculado
0	3.0000000000000000
1	3.1333333333333304
2	3.1421176470588823682
3	3.1415857837618736 96
4	3.1415926652777175 22
5	3.141592653638244137
6	3.141592653589720285
7	3.141592653589794004
8	3.141592653589791340
9	3.141592653589788675
10	3.141592653589788675
11	3.141592653589788231
12	3.141592653589791784
13	3.141592653589803330
14	3.141592653589775797
15	3.141592653589791784
16	3.141592653589774020

Valores de pi calculados pelo programa

j	erro
0	1.415926535897931160e-01
1	8.259320256459664819e-03
2	5.249934690305953612e-04
3	6.869827919567549360e-06
4	1.168792435232289360e-08
5	4.845105665818562326e-11
6	7.296575861230854352e-14

7	8.188193446854557960e-16
8	1.837156994495544165e-15
9	9.181371965860456321e-15
10	4.624953540301544149e-15
11	4.824757557510543891e-15
12	1.187878036853544113e-15
13	1.012433065458945554e-14
14	1.712091139473454329e-14
15	1.370114232550544210e-15
16	1.930472659896454575e-14

Erros Absolutos

Através da análise dos dígitos corretos de cada aproximação e dos erros absolutos associados, considera-se que o método de Romberg é bastante eficiente na medida em que é necessário um número reduzido de subintervalos para obter boas aproximações.

2.2.2 Exercício 2. (b)

Com os dados obtidos na alínea anterior foi feita uma regressão linear de modo a estimar o grau de precisão p do Método dos Trapézios ($R_{j,0}$) e do Método de Simpson ($R_{j,1}$) que correspondem ao Método de Romberg tomando k = 0 e para k = 1, respetivamente.

Para a regra dos trapézios aplicada a uma função f, o erro é nos dado pela expressão:

$$E_n^T(f) = -\frac{(b-a)h^2}{12} f''(\xi_1) [7]$$

com $\xi_1 \in]a, b[$. Da fórmula do erro podemos concluir a ordem de precisão do método uma vez que esta é da ordem de h. Assim a regra dos trapézios tem grau de precisão 2 pois:

$$E_n^T(f) = O(h^2)$$

Para a regra de Simpson aplicada a uma função f, o erro é nos dado pela expressão:

$$E_n^S(f) = -\frac{(b-a)h^4}{180} f^4(\xi_2) [8]$$

com $\xi_2 \in]a,b[$. Assim a regra dos trapézios tem grau de precisão 4 uma vez que:

$$E_n^S(f) = O(h^4)$$

Aplicando o módulo em [7] e [8] obtem-se:

$$\left|E_n^T(f)\right| \leq \frac{(b-a)h^2}{12} max\left|f''(\xi_1)\right| [9]$$

e

$$\left|E_n^S(f)\right| \leq \frac{(b-a)h^4}{180} max\left|f''(\xi_2)\right| [10]$$

com $\xi_1, \xi_2 \in]a,b[$. Deste modo [9] e [10] podem ser vistos respetivamente como:

$$\left|E_n^T(f)\right| = Ch^2 [11]$$

e

$$\left|E_n^S(f)\right| = Ch^4 [12]$$

Com esse objetivo gerou-se uma sucessão do erros, $|E_h|$ dada por $|\pi - T_n(f)|$ e $|\pi - S_n(f)|$ consoante o método considerado.

Para confirmar a ordem de precisão prevista pela teoria, recorreu-se à realização de uma regressão linear para ambos os métodos. Assim [11] e [12] podem podem ser encarados de forma genérica como:

$$|E_h| = Ch^p [13]$$

onde p é a ordem de precisão do método. De forma a linearizar [13] aplicou-se o logaritmo à expressão obtida:

$$log|E_h| = log(C) + plog(h)$$

onde $h = \frac{1}{2^j}$, para o cálculo do integral considerado. Considerando a regressão linear como uma reta sob a forma $y = mx + b$, y será o $log|E_h|$ e x o $log(h)$, pelo que o declive da reta gerada será a ordem de precisão, p.

Desta forma aplicou-se a função:

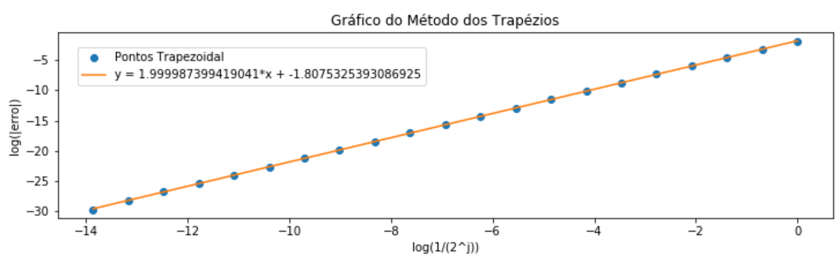
```
import math
from decimal import * # valores com mais precisão
ctx = Context(prec=20) #precisão de 20 dígitos


getcontext().prec = 28
```

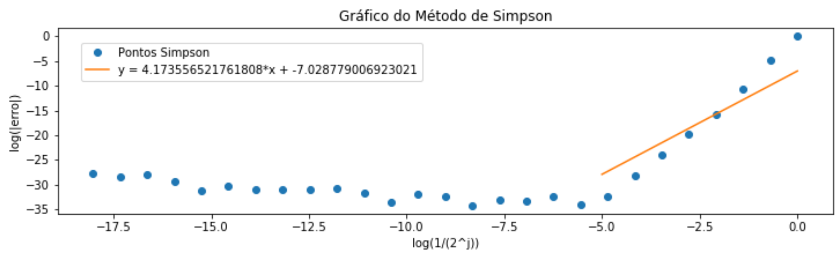
```
def rombergpir(f, a, b, N):
  r=[[0 for j in range(0,N+1)] for k in range(0,N+1)]
  r[0]=[Decimal(trap(f,a,b,2**j)) for j in range(0,N+1)]
  for k in range(1,N+1):
    for j in range(k,N+1):
      #r[0][j]=trap(f,a,b,2**j)
      r[k][j]=((Decimal(4**k)) * r[k-1][j] - r[k-1][j-1])/(
         Decimal(4**k-1))
  return [[float(abs(4*r[k][j] - Decimal(math.pi)).ln(ctx)) if j
     >=k else 0 for j in range(0,N+1)] for k in range(0,N+1)] #
     calcula o logaritmo do desvio em relação ao pi

import numpy as np
import matplotlib.pyplot as plt
from math import *

a=26

plt.subplots(2,2,figsize=(12,8))

plt.subplots_adjust(hspace=0.7)

x=[log(1/(2**j)) for j in range(a+1)] # cria as abcissas que
   correspondem ao tamanho do subintervalo usado
y=rombergpir(lambda x: 1/(1+x**2),0,1,a)[0] # calcula o logaritmo
    dos erros absolutos para k = 0 (corresponde à Regra dos Trapé
   zios)

fit = np.polyfit([log(1/(2**j)) for j in range(a+1)], y, 1)


plt.subplot(2,1,1)
plt.title('Gráfico da Regra dos Trapézios') # Dá um título ao grá
   fico
plt.xlabel('log(1/(2^j))') # Dá um nome aos eixos
plt.ylabel('log(|erro|)')

plt.plot(x,y, 'o', linewidth=1, label='Pontos Trapézios')
```

```
xr = np.linspace(log(1/(2**(a))),0,2) # cria as abcissas dos dois
    pontos que irão servir para traçar a reta de regressão linear
print("Declive da reta de regressão linear para a Regra dos trapé
   zios: " + str(fit[0]))  # Apresenta o declive da reta de
   regressão linear
yr=fit[0]*xr+fit[1] # completa a criação dos dois pontos que
   servirão para traçar a representação gráfica da reta
   determinando as suas ordenadas
plt.plot(xr,yr, label='y = ' + str(fit[0]) + "*x + " + str(fit
   [1])) # traça a reta de regressão linear

plt.legend(bbox_to_anchor=(0.25, 0.95), loc = 'upper center') #
   coloca a legenda na figura / configuração da localização da
   legenda




plt.subplot(2,1,2)
plt.title('Gráfico do Método de Simpson') # Dá um título ao grá
   fico
plt.xlabel('log(1/(2^j))') # Dá um nome aos eixos
plt.ylabel('log(|erro|)')

x=[log(1/(2**j)) for j in range(a+1)] # cria as abcissas que
   correspondem ao tamanho do subintervalo usado
y=rombergpir(lambda x: 1/(1+x**2),0,1,a)[1] # calcula o logaritmo
    dos erros absolutos para k = 1 (corresponde à Regra de
   Simpson)
fit = np.polyfit([log(1/(2**j)) for j in range(1,a+1)], y[1:], 1)

plt.plot(x,y, 'o', linewidth=1, label='Pontos Simpson')

xr = np.linspace(log(1/(2**(a))),0,2) # cria as abcissas dos dois
    pontos que irão servir para traçar a reta de regressão linear
print("Declive da reta de regressão linear para a Regra de
   Simpson: " + str(fit[0])) # Apresenta o declive da reta de
   regressão linear
yr=fit[0]*xr+fit[1] # completa a criação dos dois pontos que
   servirão para traçar a representação gráfica da reta
```

```
    determinando as suas ordenadas
plt.plot(xr,yr, label='y = ' + str(fit[0]) + "*x + " + str(fit
    [1])) # traça a reta de regressão linear


plt.legend(bbox_to_anchor=(0.25, 0.95), loc = 'upper center') #
    coloca a legenda na figura / configuração da localização da
    legenda

plt.savefig('Erros.png') # guarda a figura num formato de imagem
plt.show() # mostra a figura
```

Os valores obtidos para p foram: 1.995 para a regra dos trapézios e 4.174 para a regra de Simpson. A ordem de precisão para a regra dos trapézios é de aproximadamente 2 o que se encontra em consonância com o previsto teoricamente. A ordem de precisão para a regra de Simpson é de aproximadamente 4 o que também se encontra em consonância com o previsto teoricamente. No entanto quando aumentámos o número de intervalos verificamos que o declive da reta se começa a afastar deste valor.





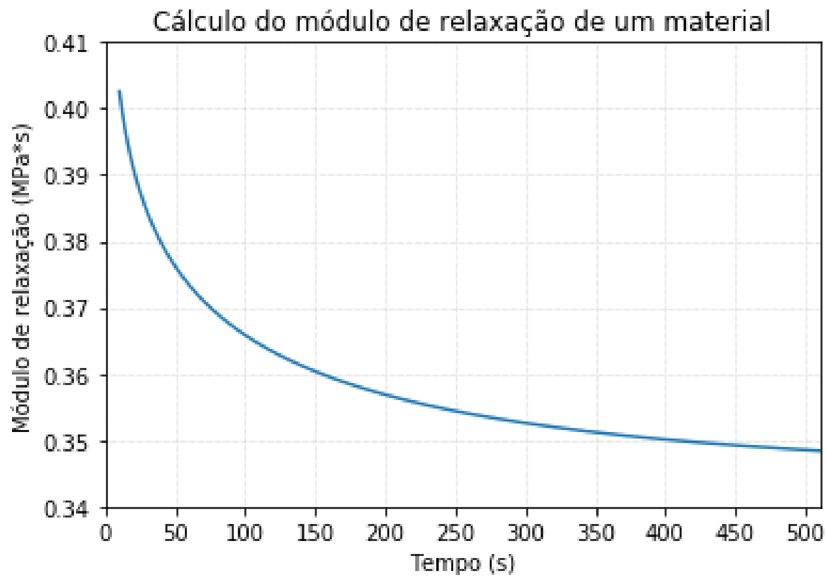
Gráficos do logarítmo do erro absoluto em função do logaritmo do número de intervalos e gráfico da regressão linear

2.3 Exercício 3.)

```
        def g(t): #função a ser integrada que depende de t e de x
            def f(x):
                return (e**(-t/x))*(1/x)
        return f
```

```
        def total(g,a,b,n,e):
            #t corresponde ao tempo
            t=10 #instante inicial
            w=[]
            while t<511: #instante do tempo final será igual a 510
                w=w+[0.34473795*(1+0.05*romberg1(g(t),a,b,n,e))]
                t=t+1
            #w é uma lista em que cada elemento corresponde à
aplicação da função Fung a um dado t, para 501 valores de t
            return w
```

```
from numpy import *
t = linspace(10,510,1000)

plt.title('Cálculo do módulo de relaxação de um material')
#Título do gráfico
plt.xlabel('Tempo (s)') # Nome do eixo das abcissas
plt.ylabel('Módulo de relaxação (MPa)') #Nome do eixo das ordenadas
```




```
plt.axis((10,510,0.34,0.41)) #Especificação dos limites do eixo das
#abcissas e do eixo das coordenadas, respetivamente
plt.yticks(arange(0.34,0.42,0.01))
 # para controlar o número e o espaçamento de marcas nos eixos
 #(step=0.5)
plt.xticks(arange(0,511,50))
plt.grid(linestyle=':', linewidth=0.5) #Grelha a linha pontilhada

plt.plot(list(range(10,511)),total(g,0.05,500,10,10**-14))
# criar gráfico que atribui uma lista de valores da função de Fung
# a uma lista de instantes
plt.show()
```

3 Parte III

3.1 Exercício 1

Neste exercício um método de Runge-Kutta de ordem 3 foi implementado na linguagem Python:

```
def rungekutta3(a, b, f, y_a, n):
  y = [y_a] # cria o conjunto das iteradas com a iterada inicial
  h = (b - a) / n # calcula a distância h entre cada ponto
  t = [a + i * h for i in range(n+1)] # cria o conjunto dos pontos t[i]

  def gerark(i): # calcula k para calcular a próxima iterada
    k_1 = f(t[i], y[i])
    k_2 = f(t[i] + h / 2, y[i] + (h / 2) * k_1)
    k_3 = f(t[i] + h, y[i] - h * k_1 + 2 * h * k_2)
    return [i, k_1, k_2, k_3]

  for i in range(n): # calcula todas as iterações
    k = gerark(i)
    y += [y[i] + (h / 6) * (k[1] + 4 * k[2] + k[3])]

  return y
```

3.2 Exercício 2