CALCULO NUMÉRICO

Gabriela Cavalcante da silva 12/03/2020

Tarefa 1

1.1 Considere a função $f(x) = x^3 - 2x^2x + 2$. Plote em um mesmo gráfico:

- com a legenda **função cúbica** no intervalo de $x[1.5, 2.5]$
- a reta tangente no ponto $(1, f(1))$, com a legenda **reta tangente em** $x = 1$
- Os pontos:
  - $(1, f(1))$
  - a interseção entre a reta tangente e o eixo x
  - os dois pontos críticos (e as respectivas retas tangentes)
- Adicione um grid, eixo x e eixo y.

Calculando a deriva da função $f(x)$, obtemos $f'(x) = 3x^2 - 4x - 1$. Para achar os pontos críticos, basta achar os pontos x onde a função derivada é 0. Usando baskara:

$$x = \frac{-(-4) \pm \sqrt{(-4)^2 - 4 * 3 * (-1)}}{2 * 3} = \frac{4 \pm \sqrt{16 + 12}}{6} = \frac{4 \pm \sqrt{28}}{6}$$
$$x_1 = \frac{4 + \sqrt{28}}{6} \tag{1}$$
$$x_2 = \frac{4 - \sqrt{28}}{6}$$

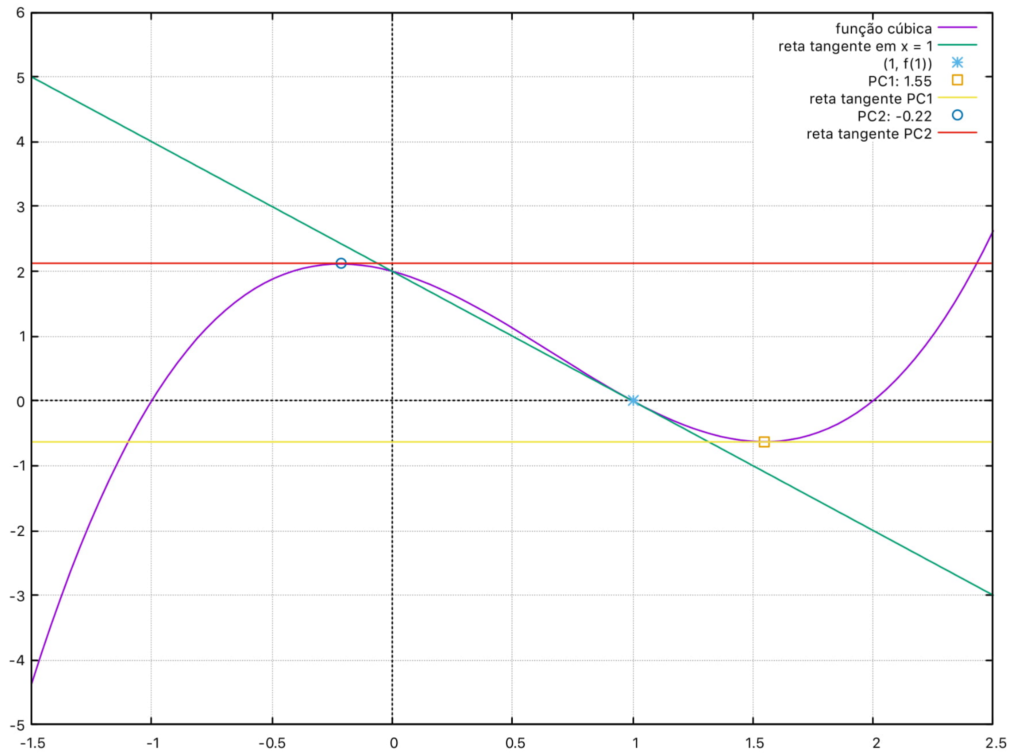

Figure 1: Plot final para a Tarefa 1.1

Listing 1: Código Python usado para a resolução da questão 1.

```python
import sys
import PyGnuplot as gp
from math import *
import numpy as np

def f(x):
    return (x**3) - (2* (x**2)) - x + 2

derive = lambda f, x, h: (f(x + h) - f(x))/h

def tangent_line_gp(f, a, b):
    x = np.linspace(a, b, 200)

    x0 = 1
    y_0 = f(x0)
    y_tan = derive(f, x0, 0.000000001) * (x - x0) + y_0
    gp.s([x, y_tan], filename="tangente.dat")

    x1 = (4 + sqrt(28))/6
    x2 = (4 - sqrt(28))/6

    gp.c('set grid')
    gp.c('set xrange[-1.5:2.5]')
    gp.c('set xzeroaxis')
    gp.c('set yzeroaxis')

    # Func Cubica
    gp.c('plot (x**3) - (2* (x**2)) - x + 2 title "fun  o c bica"')

    # Tangente x = 1
    gp.c('replot "tangente.dat" with lines title "reta tangente em x = 1"')

    # 1, f(1))
    gp.c(f'replot "<echo 1 {f(1)}" title "(1, f(1))"')

    # PC1
    gp.c(f'replot "<echo {x1} {f(x1)}" title "PC1: {x1:.2f}"')
    # PC1 tangente
    y_0 = f(x1)
    y_tan = derive(f, x1, 0.000000001) * (x - x1) + y_0
    gp.s([x, y_tan], filename="tangente-pc1.dat")
    gp.c('replot "tangente-pc1.dat" with lines title "reta tangente PC1"')

    # PC2
    gp.c(f'replot "<echo {x2} {f(x2)}" title "PC2: {x2:.2f}"')
    # PC2 tangente
    y_0 = f(x2)
    y_tan = derive(f, x2, 0.000000001) * (x - x2) + y_0
    gp.s([x, y_tan], filename="tangente-pc2.dat")
    gp.c('replot "tangente-pc2.dat" with lines title "reta tangente PC2"')


if __name__ == '__main__':
    tangent_line_gp(f, -1.5, 2.5)
```

Para plotar o gráfico usei o gnuplot com a biblioteca PyGnuplot[1], que permite usar comandos do gnuplot dentro do código Python. O código acima está disponível no github[2].

[1] https://github.com/benschneider/PyGnuplot

[2] https://github.com/gabicavalcante/numerical$_{analysis/blob/master/l1/l1e1.py}$

Tarefa 2

2.1 Seja uma função $f(x)$:

- Partindo de um ponto conhecido da função, estime outros pontos de $f(x)$ no intervalo $[6:6]$ sabendo que:
  - $f(0) = 1$ (o ponto conhecido)
  - $f'(x) = cos(x) - x \cdot sen(x)$
- Grave os pontos estimados em um arquivo e plote-os com **plot arquivos.pts with lp,** $x \cdot cos(x) + 1$
  - $f(x) = x \cdot cos(x) + 1$, mas utilizaremos essa informação apenas para comparar as estimativas de f(x) com f(x)

Como vimos em sala, $f'(x) = lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$, definindo h com valores fixos, temos $f'(x) = \frac{f(x+h)-f(x)}{h}$, logo $f(x+h) = f(x) + h \cdot f'(x)$. Como temos o valor de h e de $f'(x)$, podemos agora calcular os pontos. Vamos tomar h como -0.5 e 0.5, o gráfico gerado é:

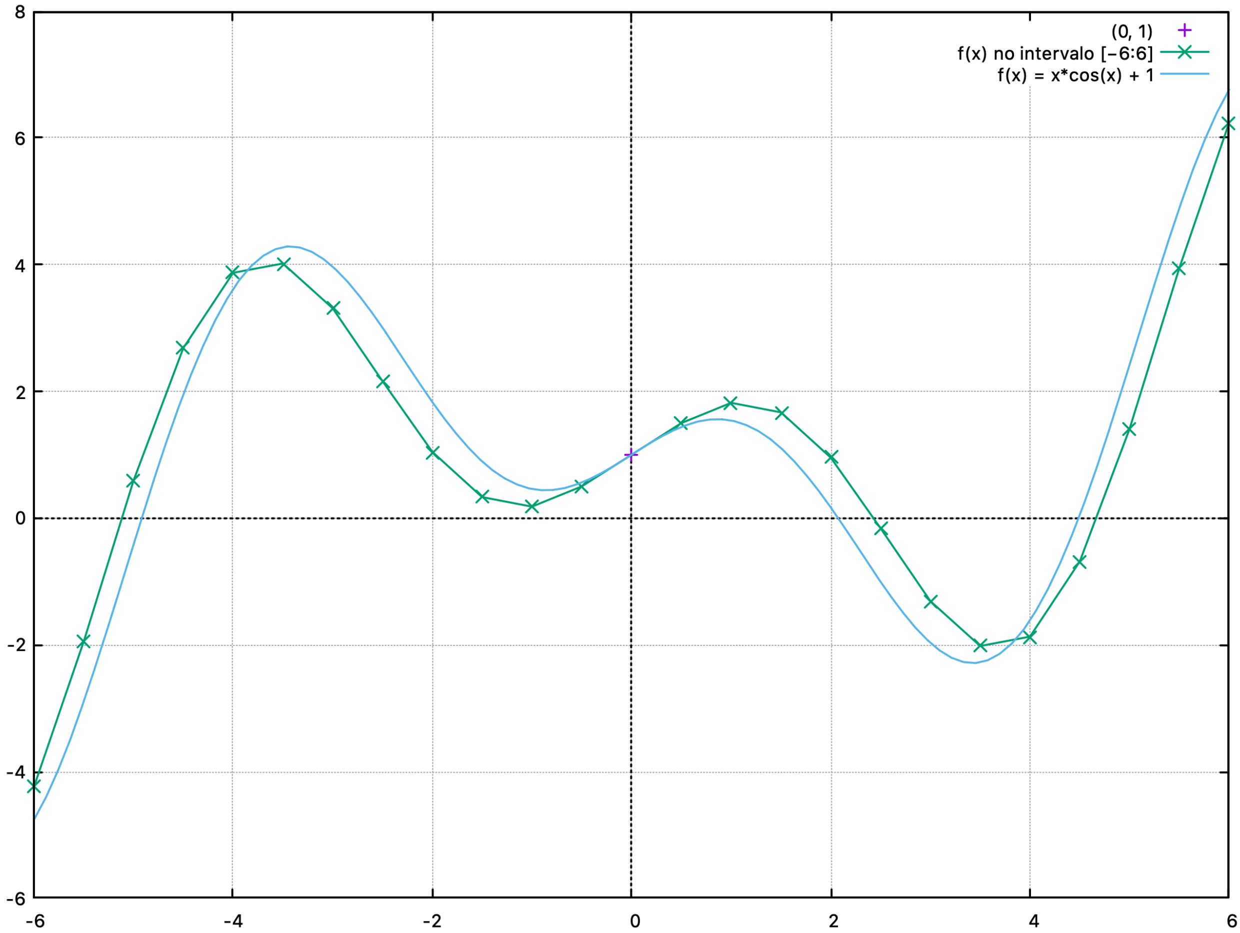

Figure 2: Plot final para a Tarefa 1.2 para valores de h sendo -0.5 e 0.5

Testando o valor de h como -0.1 e 0.1, teremos uma estimativa ainda mais precisa, como podemos ver na Figura 3. O código responsável por plotar esses gráficos pode ser encontrados no repositório do github [3], ou em anexo a esse relatório.

[3]https://github.com/gabicavalcante/numerical$_{a}nalysis/blob/master/l1/l1e2.py$

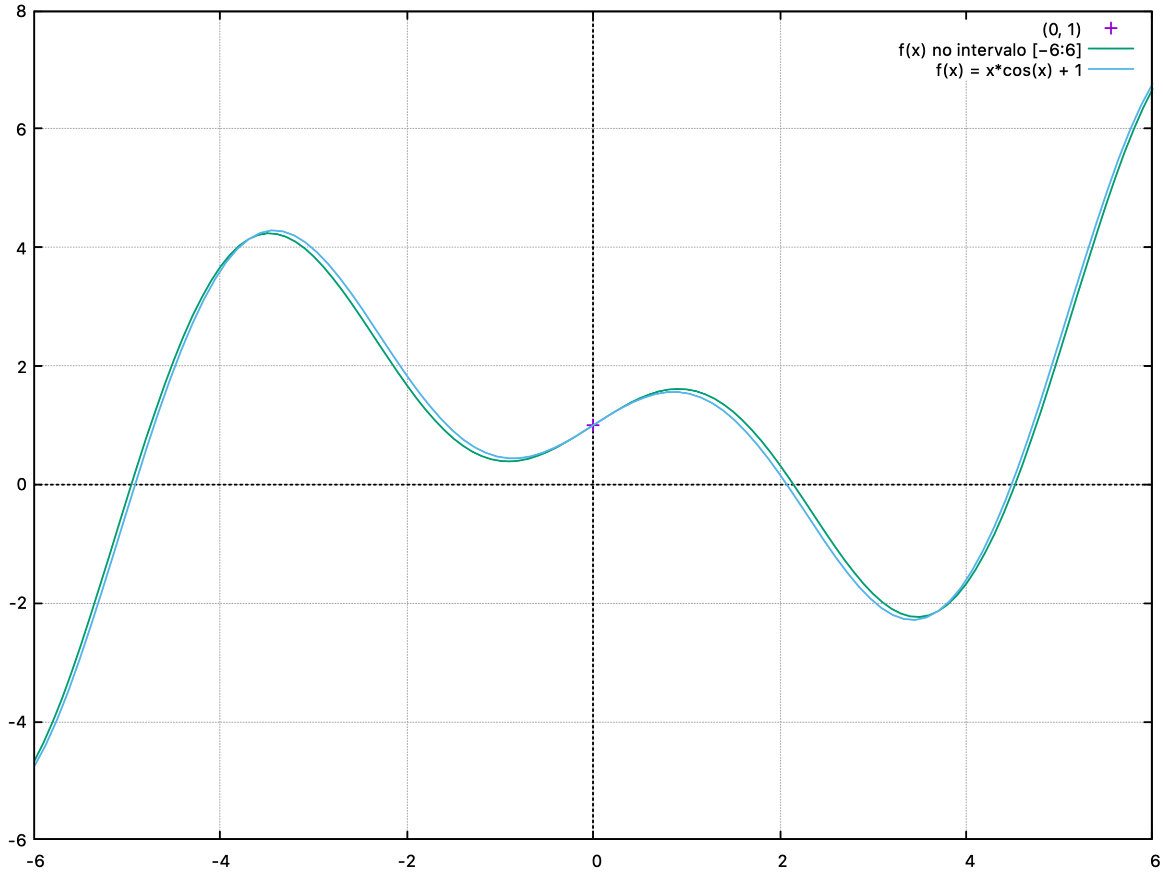

Figure 3: Plot final para a Tarefa 1.2 para valores de h sendo -0.1 e 0.1

Na segunda parte da questão, é necessário plotar em um gráfico a função $f(x) = x \cdot cos(x) + 1$ e sua aproximação pela série de taylor em torno de $a = 0$. O seguinte trecho de código foi usado para plotar o gráfico da Figura 4.

Listing 2: Código Python usado para a resolução da questão 1.

```
import PyGnuplot as gp
from math import *
import numpy as np

# impar
f_odd = lambda x, k: ((-1)**((k-1)/2)) * k * cos(x) + ((-1)**((k-1)/2)) * x * sin(x)
# par
f_even = lambda x, k: ((-1)**(k/2)) * k * sin(x) + ((-1)**(k/2)) * x * cos(x)


def taylor(x):
    sum_ = 0
    a = 0
    for n in range(0, 25):
        if n % 2 == 0:
            sum_ += (f_even(a, n)/factorial(n)) * ((x - a)**n)
        else:
            sum_ += (f_odd(a, n)/factorial(n)) * ((x - a)**n)

    return sum_

def plot_graph_taylo(a, b):
    gp.c('set grid')
```

```
24     gp.c(f'set xrange[{a}:{b}]')
25     gp.c(f'set yrange[-10:10]')
26     gp.c('set xzeroaxis')
27     gp.c('set yzeroaxis')
28
29     points_x = np.linspace(a, b, 100)
30     points_y = [taylor(x) for x in points_x]
31     gp.s([points_x, points_y], filename='taylor.pts')
32     gp.c('plot "taylor.pts"')
33     gp.c('replot x*cos(x) + 1 title "f(x) = x*cos(x) + 1"')
```

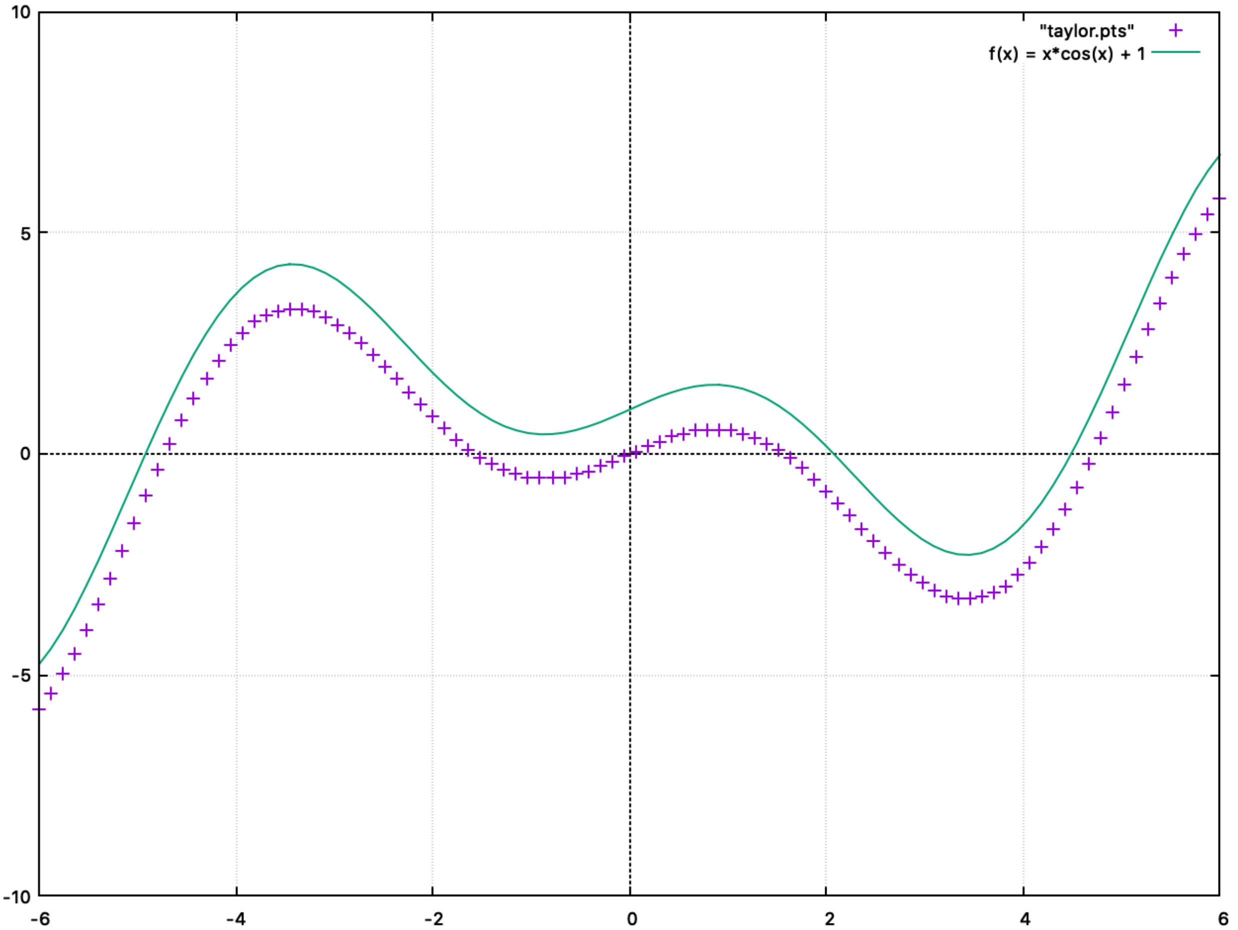

Figure 4: Plot da segunda parte da Tarefa 1.2

Observando os valores calculados na função da série de Taylor, vamos verificar que sempre no passo em que $n\%2 == 0$, o resultado é 0. Então podemos substituir a linha de código 32 do código acima, por:

```
gp.c(f'plot 1+(x**1)/{factorial(1)} + -3*x**3/{factorial(3)} + 5*x**5/{↩
    factorial(5)} + -7*x**7/{factorial(7)} title "taylor"')
```

Vamos obter o gráfico a seguir:

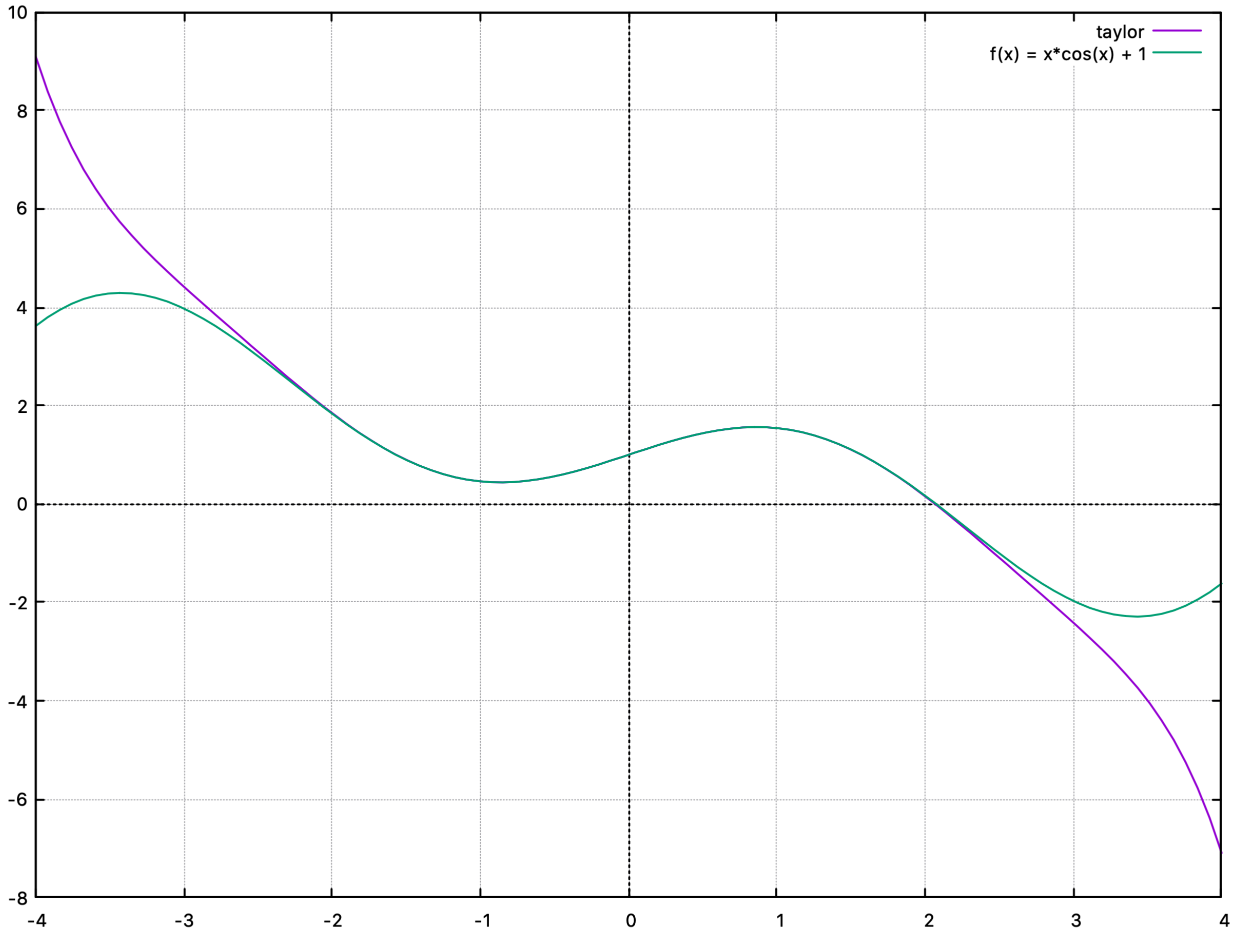

Figure 5: Plot da segunda parte da Tarefa 1.2