Nesta tarefa você vai estudar a aproximação numérica por polinômio de Taylor. Para tanto, comece estudando a exemplo feito no Colab, veja link no AVA.

A Tarefa desta unidade é fazer um relatório com algumas partes descritas abaixo. Você pode entregar em PDF ou em `.ipynb` feito no Colab ou no Jupyter (salve local em `.ipynb` e dê upload no AVA do arquivo `.ipynb`). O relatório deve conter todo o código utilizado. O código deve estar comentado, com nomes de variáveis adequados, e com as justificativas e fórmulas matemáticas utilizadas.

Siga o seguinte roteiro:

1. Escolha uma função $f(x)$ não polinomial "interessante" (não pode ser polinômio). Por interessante entende-se uma função que envolva a composição ou razão de duas ou mais funções envolvendo trigonométricas, ou exponenciais, ou raízes, ou suas inversas, logaritmos, etc...

2. Escolha um ponto x_0 para fazer a expansão em série de Taylor.

3. Construa funções em Python para calcular 1a, 2a, e 3a derivadas desta função. Se usar um pacote algébrico diga no relatório qual.

4. Construa uma função em Python que calcula o polinômio de Taylor de ordem 3.

5. Assim como no notebook `Colab_Taylor_v1.ipynb` deixado no AVA, construa dois gráficos com a função e o polinômio. Há dois enquadramentos: (1) o mais amplo que mostra uma visão geral da f junto como polinômio, (2) e outro onde mostra-se a função e o polinômio começando a separar.

6. Calcule o erro $Err(x_j) = |f(x_j) - P_3(x_j)|$ nos pontos
$$x_j = x_0 + h_j, \tag{1}$$
$$h_j = \frac{H}{2^j} \tag{2}$$
para $j = 1, 2, \ldots, 5$. Note que h_j começa valendo $H/2$ e vai sendo dividido por 2 e por 2 novamente... No caso do `Colab_Taylor_v1.ipynb` o valor é $H = 0.2$. Mas para a função que você escolheu pode ser que H tenha que ser escolhido maior ou menor. Tente um H onde o erro começa em x_1 da ordem de `1e-2` ou `1e-3`.

7. Calcule o $\log_2$ da razão dos erros consecutivos:
$$p_j = \log_2\left(\frac{Err(x_j)}{Err(x_{j+1})}\right) \tag{3}$$

8. Qual o valor aproximado dos p_j? Obtenha-os numericamente e explique matematicamente porque isto acontece. (Se tudo estiver certo, a aproximação de Taylor calculada deverá ser de ordem 4.)

Recomendações gerais para as tarefas

Originalidade na resolução e no relatório será fortemente premiada na nota final. Vocês podem discutir dificuldades comuns, diferentes abordagens adotadas e comparar resultados. Porém *cada um deve fazer seu próprio código e relatório* para não correr o risco de prejudicar a originalidade do trabalho. Os exercícios serão avaliados não apenas por alcançar a resposta mas também por sua elegância e eficiência na solução. Códigos e relatórios bem organizados serão valorizados.

Há duas opções de formato para entregar o trabalho. Primeira opção, ele pode ser entregue como um *único arquivo do Jupyter/Colab (`arquivo.ipynb`)*, isto é, um único arquivo contendo os programas e o relatório. Segunda opção, ele pode ser entregue *num único arquivo PDF* e uploaded no AVA/Moodle. Este PDF tem que conter tudo, relatórios, código do programa, figuras, etc... Peça ajuda caso você tenha dificuldade para criar um único arquivo em PDF.

O notebook do Colab para começar

No que segue, está uma transcrição do `Colab_Taylor_v1.ipynb` (versão de 2/6/2022) para consulta. Provavelmente haverão modificações, fique atendo e viste a versão "live" em caso de dúvida.

Este é um exemplo de código em iPython notebook (feito no Colab). Nele é feita a comparação entra a função

$$f(x) = \frac{x}{\sqrt{4x^2+1}} \tag{4}$$

e o polinômio de Taylor de ordem 2 expandido ao redor de $x_0 = 1$. O objetivo é observar como o erro decai a media que $x \to x_0$. Ao final é deixada uma pergunta para você concluir qual é ordem da aproximação.

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):  #a função
  y = x/np.sqrt(4*x**2+1)
  return y

def df(x): #sua derivada
  y = 1/np.power(4*x**2+1,3/2)
  return y

def d2f(x): # a segunda derivada
  y = x*((48*x**2)/(4*x**2 + 1)**(5/2) - 4/(4*x**2 + 1)**(3/2)) - (8*x)/(4*
```

```
    return y

x0 = 1   #ponto de expansão do polinômio de Taylor
def Poli_Taylor_ord_2(x):  #polinomio de Taylor de ordem 2
    y = f(x0)+ df(x0)*(x-x0)+(d2f(x0)/2)*(x-x0)**2
    return y
```

Este aqui é o gráfico que mosta a função e seu polinômio.

```
x=np.linspace(-2,4,100) #gera um vetor x com 100 valores igualmente espaçad
cm = 1/2.54  # centímetos em uma polegada (intch)
fig, ax = plt.subplots(figsize=(15*cm, 10*cm)) #tamanho
ax.plot(x, f(x), label = 'a função f')        #plota f e dá sua respectiva
ax.plot(x, Poli_Taylor_ord_2(x), label = 'Poli. Taylor ordem 2')
ax.scatter(x0, f(x0), c='tab:red' , s=30, label='ponto', marker = 'o') #pon
ax.set_xlabel('x',fontsize=14)   #legenda dos eixos
ax.set_ylabel('y',fontsize=14)
ax.set_title('f(x) e o poinomio de Taylor',fontsize=14) #titulo
ax.legend(loc='lower right')       #gera a legenda e dá a posição
ax.grid(True)                      #coloca o grid de linhas
plt.show()                         #mostra
```

Este aqui é um zoom mostrando onde o polinômio ajusta bem à função, (igual cima e troca só a 1a linha)

```
x=np.linspace(0.7,1.3,100)
```

Este gráfico mostra a diferença
$g(x) = f(x) - P_2(x)$.
Poderíamos mostrar o erro também simplesmente colocando o módulo:
$Err(x) = |f(x) - P_2(x)|$.
porém o erro não é tão bonito porque ele forma um "bico" em x_0. Note a forma de $g(x)$ como ela se parece com um polinômio cúbico.

```
x=np.linspace(0.7,1.3,100) #gera um vetor x com 100 valores igualmente espa
cm = 1/2.54  # centimeters in inches
fig, ax = plt.subplots(figsize=(15*cm, 10*cm))
ax.plot(x, f(x) - Poli_Taylor_ord_2(x), label = 'diferença f(x)-P2(x)')
ax.set_xlabel('x',fontsize=14)
ax.set_ylabel('y',fontsize=14)
ax.set_title('Diferença entre f(x) e o poinomio de Taylor',fontsize=14)
ax.legend(loc='lower right')
ax.grid(True)
plt.show()
```

Os valores abaixo mostram o erro em pontos da forma

$$x_j = x_0 + h_j, \tag{5}$$

$$h_j = \frac{0.2}{2^j} \tag{6}$$

para $j = 1, 2, \ldots, 5$. Note que h_j começa valendo 0.1 e vai sendo dividido por 2 e por 2 novamente...

```
x = x0 + np.array([0.1, 0.05, 0.025, 0.0125, 0.00625]) #np.array transforma
print('x = ', x)
print('erro = ', np.abs(f(x) - Poli_Taylor_ord_2(x)) )
```

Será que o erro é cúbico na variável $h = x - x_0$?

Isso pode ser quantificado numericamente.

Primeiramente vamos supor que o erro é aproximadamente uma função cúbica de h. Ou seja, nossa hipótese para o erro é que ele é aproximadamente da forma

$$Err(h) \approx Ch^3 \tag{7}$$

Isso é muito razoável porque sabemos, pelo Teorema de Taylor, que o erro é dado por

$$Err(h) = \left| \frac{f^{(3)}(\xi)}{3!}(x - x_0)^3 \right| \tag{8}$$

para algum ξ entre x e x_0. O que é muito similar ao próximo temo na série de Taylor que é de fato dado por

$$Err(h) \approx \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 = \frac{f^{(3)}(x_0)}{3!}h^3 \tag{9}$$

O seja, espera-se que o valor de C em

$$Err(h) \approx Ch^3 \tag{10}$$

seja aproximadamente $f^{(3)}(x_0)/3!$.

Você deve prosseguir esse raciocínio calculando o log da razão dos erros consecutivos:

$$p_j = \log_2 \left(\frac{Err(h_j)}{Err(h_{j+1})} \right) \tag{11}$$

Uma das perguntas da tarefa é:

Qual o valor aproximado dos p_j? Obtenha-os numericamente e explique matematicamente porque isto acontece. Isso verifica que a aproximação do polinômio de Taylor é realmente cúbica?