UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE LORENA

LOM3260 – COMPUTAÇÃO CIENTÍFICA EM PYTHON – TURMA: F3

PROFESSOR: Luiz Tadeu Fernandes Eleno

GABRIEL DE TOLEDO PAULA -12609794

GABRIEL DOS SANTOS MELO -12776968

JORGE CASMAMIE NUNES -12548233

MATHEUS CRUVINEL DE SOUZA -12717395

OSWALDO HENRIQUE DE FREITAS -12716967

VINICIUS MATIAS FLORENTINO -12566421

PYMATH – DOCUMENTAÇÃO

VERSÃO 1.0

LORENA

2021

Sumário

1. INTRODUÇÃO – O QUE É O PYMATH?

O Pymath é um Software que realiza uma regressão linear simples utilizando o método dos mínimos quadrados de acordo com os dados fornecidos pelo usuário. O Software realiza cálculos através de diversas funções da estatística (método dos mínimos quadrados) e com esses valores o Pymath cria gráficos com os pontos dados e uma reta (função de ajuste) que ajuda a interpretar se os dados estão crescendo, decrescendo ou se estão estáveis.

2. REGRESSÃO LINEAR SIMPLES E MÉTODO DOS MÍNIMOS QUADRADOS

No Pymath, utiliza-se como base teórica o Ajuste Linear Simples, ou Regressão Linear Simples, uma vez que o software organiza os dados em um modelo linear com apenas duas variáveis.

Existem dois tipos de Ajuste Linear, o ajuste simples que utiliza apenas duas variáveis (uma variável dependente e um independente, onde a última é utilizada para calcular a dependente) e o ajuste múltiplo que utiliza mais de duas variáveis. Para ambos é necessário que exista uma relação linear entre os dados, caso não haja será preciso artifícios matemáticos para linearizar os dados e garantir melhor interpretação dos dados, como uma função exponencial transformada em uma função logarítmica.

As equações utilizadas para Regressão Linear Simples no Software são:

```
# FÓRMULAS #
Sx = sum(x) # Soma de todos os valores de X.
Sy = sum(y) # Soma de todos os valores de Y.
Sx2 = (x**2).sum() # Soma de todos os valores de X elevados a 2.
Sy2 = (y**2).sum() # Soma de todos os valores de Y elevados a 2.
Sxy = (x*y).sum() # Soma de todos os valores de X multiplicados aos valores de Y.

Mx = np.mean(x) # Valor médio de X.
My = np.mean(y) # Valor médio de Y.

A = Sx2 - (Sx**2)/n
B = Sxy - (Sx*Sy)/n
C = Sy2 - (Sy**2)/n

a = B/A
a = float("%.4f" % a)
b = My - a*Mx
b = float("%.4f" % b)

CORR = np.sqrt((B**2)/(A*C))
func = f'{b:+} {a:+}x' # Função na forma de y = b + ax
yi = [b + a*m for m in x]
```

Com as equações já definidas pelo método dos mínimos quadrados é possível determinar uma função afim, chamada também de função de ajuste, que determina a relação entre as variáveis a partir do coeficiente linear ***a*** e da intersecção ***b*** da reta com o eixo da variável dependente e por fim interpretar a partir do gráfico o que acontece com os dados calculados, se há decadência, ascensão ou estabilidade.

3. METODOLOGIA

3.1 BIBLIOTECAS UTILIZADAS:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gs
from tkinter import *
from tkinter import filedialog
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
import webbrowser
```

- A biblioteca *numpy* (apelidada de *np*) foi utilizada para o cálculo da médias dos dados de X e Y durante a Regressão Linear Simples.

```
Mx = np.mean(x) # Valor médio de X.
My = np.mean(y) # Valor médio de Y.
```

- A biblioteca *matplotlib.pyplot* (apelidada de *plt*) foi utilizada para criar os gráficos, além do seus títulos, pontos, retas, barras de erro, escalas.

```
fig = plt.figure() # Cria uma Figura para o Gráfico
plt.title('Gráfico Padrão X × Y') # Título do Gráfico
plt.plot(x, yi, '-') # Os pontos de X e Y serão marcados por pontos (o) e ligados por linhas contínuas (-)
plt.errorbar(x, y, yerr=erro, fmt='ro', ecolor='black', linestyle=None, capsize=2) # Cria as Barras de Erro, com pontos vermelhos (ro)
plt.xscale('linear') # Escala linear do Eixo X
plt.yscale('linear') # Escala linear do Eixo Y
plt.show() # Exibe o Gráfico
```

- A biblioteca *matplotlib.gridspec* (apelidada de *gs*) foi utilizada para criar a grade de exibição do gráfico, e em que posição dessa grade estará o gráfico.

```
grid = gs.GridSpec(7, 3) # Define o tamanho como uma grade para a Figura
grafico = fig.add_subplot(grid[:5, :]) # Define o espaço da Figura que será ocupada pelo Gráfico
```

- A biblioteca *tkinter* foi utilizada para criação da interface gráfica do Pymath.

```
root = Tk() # Janela da Interface
self.root = root
```

- As bibliotecas *reportlab.pdfgen* e *reportlab.lib.pagesizes* foram utilizadas para criar um arquivo PDF em folha A4, que contém os cálculos da Regressão Linear. Essa biblioteca foi baixada pelo Prompt de Comando, através do comando *"pip install reportlab"*.

```
pdf = canvas.Canvas("Regressao_Linear.pdf", pagesize = A4) #Gerando PDF
```

- A biblioteca *webbrowser* foi utilizada para que o Pymath possa abrir um link, direcionando o usuário para o site do Pymath.

```
url = "https://sites.google.com/usp.br/pymath/como-utilizar-o-pymath" # Link do Site do Pymath
webbrowser.open_new(url)
```

3.2 CÓDIGO:

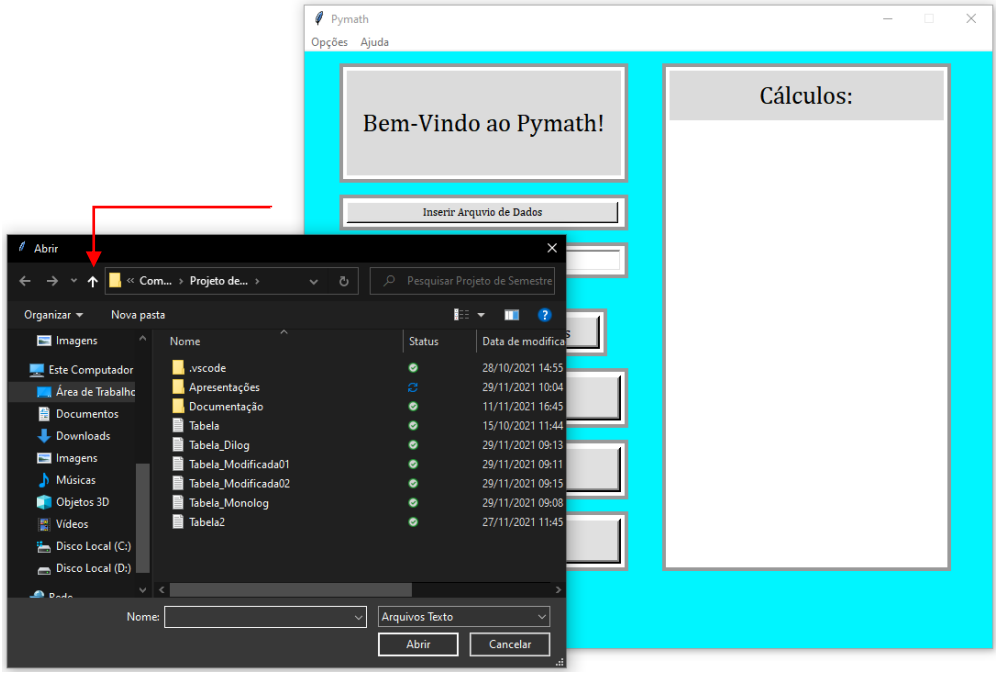
RECEBENDO DADOS:

Antes de realizar qualquer cálculo, o Pymath precisa saber com quais dados ele irá trabalhar.

Para isso, foram criadas as seguintes funções:

```
def upload_arq(self): # Função para abrir o seletro de arquvios
def read_arq(self): # Função que le os dados X e Y do Arquvio
def erro(self): # Função que receberá o valor para as Barras de Erro
```

- *upload_arq(self)* é responsável por abrir o Seletor de Arquivos. Com isso, o usuário poderá navegar pelos arquivos em seu computador e selecionar o arquivo com dados de X e Y que ele deseja usar no programa.




```
def upload_arq(self): # Função para abrir o seletro de arquvios
    if self.arquivo_salvo == False:
        self.file = filedialog.askopenfilename(filetypes=[('Arquivos Texto','*.txt'), ('Todos os Arquviso', '*.*')])
        self.arq = open(self.file,'r')
        self.arquivo_salvo = True
    else:
        self.arq = open(self.file,'r')
```

```
    x, y = np.genfromtxt(self.arq, unpack=True)
    return x, y
```

A variável *"self.arquivo_salvo"* foi definida previamente dentro da função "*__init__(self)*", já com o valor *"False"*, ou seja, sua função está inicialmente "desativada".
Com a função *upload_arq(self)*, se *self.arquivo_salvo* possuir valor *False*, vai abrir o Seletor de Arquivos, através do comando *filedialog.askopenfilename()*. Quando o usurário selecionar um Arquivo de Dados, este será salvo como a variável *"self.arq"*, que será responsável por chamar esse Arquivo de Dados no decorrer do código. Depois disso, a variável *self.arquivo_salvo* recebe o valor *True*. Desse modo, o código entende que o programa já possui o Arquivo de Dados salvo em *self.arq*, sem ser necessário abrir o Seletor de Arquivos novamente.

Com isso, a função cria as variáveis *"x"* e *"y"*, que serão, respectivamente, os valores de X e Y fornecidos pelo Arquivo de Dados do usuário. Essas variáveis devem ser retornadas pela função, para que possam ser chamadas para os cálculos da Regressão Linear Simples e para a construção dos gráficos.

- *read_arq(self)* é responsável apenas por ler os dados de X e Y que foram fornecidos pelo Arquivo de Dados, sem a necessidade de se abrir o Seletor de Arquivos novamente. Ela também retorna o valor das variáveis *"x"* e *"y"*, pelo menos motivo anterior.

```
def read_arq(self): # Função que le os dados X e Y do Arquvio
    x, y = np.genfromtxt(self.arq, unpack=True)
    return x, y
```

- *erro(self)* é responsável por pegar o valor do Desvio Padrão dos Dados que será fornecido pelo usuário. Esse valor será salvo dentro da variável *"erro"* e, posteriormente, esse valor será transformado em um número decimal, pelo comando *"float(erro)"*.

A função retorna o valor de *erro*, que será usado para criar as Barras de Erro no Gráfico Padrão, que será mostrado em breve.

```
def erro(self): # Função que receberá o valor para as Barras de Erro
    erro = self.erro_entry.get() # Caixa de entrada do Desvio Padrão
    erro = float(erro) # Transforma o Desvio Padrão em um número decimal
    return erro
```

REGRESSÃO LINEAR SIMPLES:

No Pymath, a Regressão Linear Simples é realizada pela função *self.reg_lin()*. Primeiramente, o código chama os valores dos de X e Y, que foram determinados na função *self.upload_arq()*.

```
x, y = self.upload_arq() # Chama os dados de X e Y
```

Além disso, ele também determina *n*, que é a quantidade de elementos no eixo X e Y. Esse valor de *n* será importante para alguns cálculos da Regressão Linear Simples, que serão vistos em breve.

```
n = x.size # Determina a quantidade de valor de X e, consequentemente, Y.
```

A seguir, é realizado os cálculos da Regressão Linear Simples:

```
Sx = sum(x) # Soma de todos os valores de X.
Sy = sum(y) # Soma de todos os valores de Y.
Sx2 = (x**2).sum() # Soma de todos os valores de X elevados a 2.
Sy2 = (y**2).sum() # Soma de todos os valores de Y elevados a 2.
Sxy = (x*y).sum() # Soma de todos os valores de X multiplicados aos valores de Y.

Mx = np.mean(x) # Valor médio de X.
My = np.mean(y) # Valor médio de Y.

A = Sx2 - (Sx**2)/n
B = Sxy - (Sx*Sy)/n
C = Sy2 - (Sy**2)/n

a = B/A
a = float("%.4f" % a)
b = My - a*Mx
b = float("%.4f" % b)

CORR = np.sqrt((B**2)/(A*C))
func = f'{b:+} {a:+}x' # Função na forma de y = b + ax
yi = [b + a*m for m in x
```

O principal objetivo da Regressão Linear Simples é determinar o Coeficiente de Correlação Linear, denominado como CORR, e a equação $y = b + a.x$ determinada pela variável *"yi"*.

O CORR é utilizado para. Já a equação $y = b + a.x$ será utilizada pelo Pymath para construir a reta da Regressão Linear Simples no Gráfico Padrão, que será visto em breve.

Após determinar os valores de todos os elementos da Regressão Linear Simples, a função *self.reg_lin()* organiza esse valores em um texto, denominado de "*calculos*", que posteriormente, com os comando de outra função, serão exibidos para o usuário na janela da interface.

```
# Texto com os símbolos e resultados da Regressão Linear #
calculos = f'''
∑x = {"%.4f" % Sx}
∑y = {"%.4f" % Sy}
∑(x²) = {"%.4f" % Sx2}
∑(y²) = {"%.4f" % Sy2}
∑(x.y) = {"%.4f" % Sxy}

μx = {"%.4f" % Mx}
μy = {"%.4f" % My}

A = {"%.4f" % A}
B = {"%.4f" % B}
C = {"%.4f" % C}

a = {a}
b = {b}

CORR = {"%.4f" % CORR}

y = {func}
'''
```

Por fim, a *self.reg_lin()* é responsável por criar o arquivo PDF, denominado de "*pdf*", que será baixado para o usuário pela mesma função que exibirá os cálculos da Regressão Linear Simples.

```
pdf = canvas.Canvas("Regressao_Linear.pdf", pagesize = A4) #Gerando PDF
```

Primeiramente, cria-se a varável *"pdf"*, que, com uso do comando *"canvas.Canvas"*, gera o arquivo PDF e determina o tamanho da página (A4).

```
pdf.setFont("Times-Roman", 14) #Definindo fonte e tamanho
```

O comando *"setFont"* associado a *pdf* define a fonte do PDF (Times-Roman) e o tamanho da fonte (14).

```
pdf.drawCentredString(297, 250, 'Obrigado por usar o Pymath!') # Mensagem de agradecimento
```

Esse comando cria uma mensagem de agradecimento ao usuário, que estará centralizada no PDF, nas coordenadas (297, 250), com a frase “Obrigado por usar o Pymath!”.

```
pdf.drawString(90, 675, 'Cálculos da Regressão Linear:') # Definido o conteúdo do PDF
pdf.drawString(90, 650, f'∑X = {Sx}')
pdf.drawString(90, 625, f'∑Y = {Sy}')
pdf.drawString(90, 600, f'∑(X²) = {Sx2}')
pdf.drawString(90, 575, f'∑(Y²) = {Sy2}')
pdf.drawString(90, 550, f'∑(X.Y) = {Sxy}')
pdf.drawString(90, 500, f'μx = {Mx}')
pdf.drawString(90, 525, f'μy = {My}')
pdf.drawString(90, 475, f'A = {A}')
pdf.drawString(90, 450, f'B = {B}')
pdf.drawString(90, 425, f'C = {C}')
pdf.drawString(90, 375, f'CORR = {CORR}')
```

Esses próximos comandos exibem os cálculos da Regressão Linear Simples no PDF, todas com a mesma coordenada em X (90), mas cada uma com uma altura (coordenada em Y) diferente (quanto maior o valor da coordenada em Y, mais em cima estará o texto).

```
pdf.drawCentredString(297, 325, 'Equação da reta:')
pdf.drawCentredString(297, 300, f'y = {func}')
```

Esses dois comandos criam textos centrais, com a mesma coordenada em X (297), que exibem a equação da Regressão Linear Simples, no formato “$y = b + a.x$”.

O PDF só será salvo para o usuário quando for executada a função *self.impres_reglin()*, com uso do comando *“pdf.save()”*.

Ao terminar todas as suas operações, a função *self.reg_lin()* retorna os comandos “*calculos”*, *“yi”* e *“pdf”*, que serão utilizadas em próximas função.

```
return calculos, yi, pdf
```

Agora, entra em ação a função *self.impres_reglin()*, que é responsável por exibir a regressão linear simples na interface e por fazer o download do arquivo PDF para o usuário.

Primeiro, ela chama as variáveis *“calculos”* e *“pdf”* da função *self.reg_lin()*.

Obs.1: a função *self.impres_reglin()* também chama a variável *“yi”*. Embora ela não use essa variável, isso é necessário porque *“yi”* também é retornada pela *self.reg_lin()*. Sempre que uma função chama uma variável retornada por uma outra função, ela precisa chamar a quantidade exata de variáveis retornadas pela outra função.

Posteriormente, *self.impres_reglin()* determina as configuração para a exibição da variável *“calculos”*: ela estará na posição *self.frame_10* da interface, e possuirá a fonte “Arial 14”.

Depois, é determinado a posição do texto em relação ao *self.frame_10*: sua origem será 16% para a direita e 15% para baixo; sua largura ocupa 70% do espaço, enquanto que seu comprimento ocupa 80%.

Por fim, a função chama o comando *"self.imp.pack"*, responsável por iniciar a exibição do texto, e o comando *"pdf.save()"*, responsável por baixar o arquivo PDF para o usuário.

```
def impres_reglin(self): # Função que imprime a Regressão Linear na Interface
    calculos, yi, pdf = self.reg_lin() # Chama o texto com os símbolos e resultados da Regressão Linear
    self.imp = Label(self.frame_10, text=f'{calculos}', font=('Arial', 14), bg='white')
    self.imp.place(relx = 0.16, rely = 0.15, relwidth = 0.70, relheight = 0.80)
    self.imp.pack
    pdf.save()
```

GRÁFICOS:

O Pymath permite que o usuário visualize três tipos de gráficos, que se diferenciam pela escala dos eixos X e Y. O primeiro gráfico, definido pela função *self.graf_padrao()*, possui ambos os eixos em escala linear, ou seja, duas graduações cuja diferença vale 10 estão a uma distância constante. O segundo gráfico, definido pela função *self.graf_mlog()*, possui o eixo X em escala linear e o eixo Y em escala logarítmica, ou seja, duas graduações cuja razão vale 10 estão a distância constante. O terceiro gráfico, definido pela função *self.graf_dlog()*, possui ambos os eixos em escala logarítmica.

```
def graf_padrao(self): # Função que criará o Gráfico em escala Padrão
    x, y = self.upload_arq() # Chama os dados de X e Y
    calculos, yi, pdf = self.reg_lin()
    erro = self.erro() # Chama o valor do Erro
    fig = plt.figure() # Cria uma Figura para o Gráfico
    grid = gs.GridSpec(7, 3) # Define o tamanho como uma grade para a Figura
     grafico = fig.add_subplot(grid[:5, :]) # Define o espaço da Figura que será ocupada pelo Gráfico
    grafico.grid(True) # Ativa as linhas de grade no Gráfico
    plt.title('Gráfico Padrão X × Y') # Título do Gráfico
     plt.plot(x, yi, '-') # Os pontos de X e Y serão marcados por pontos (o) e ligados por linhas contínuas (-)
     plt.errorbar(x, y, yerr=erro, fmt='ro', ecolor='black', linestyle=None, capsize=2) # Cria as Barras de Erro, com pontos vermelhos (ro)
    plt.xscale('linear') # Escala linear do Eixo X
    plt.yscale('linear') # Escala linear do Eixo Y
    plt.show() # Exibe o Gráfico
def graf_mlog(self): # Função que criará o Gráfico em escala Mono-Log
    x, y = self.upload_arq() # Chama os dados de X e Y
    fig = plt.figure() # Cria uma Figura para o Gráfico
```

```
    grid = gs.GridSpec(7, 3) # Define o tamanho como uma grade para a Figura
    grafico = fig.add_subplot(grid[:5, :]) # Define o espaço da Figura que será ocupada pelo Gráfico
    grafico.grid(True) # Ativa as linhas de grade no Gráfico
    plt.title('Gráfico Mono-Log X × Y') # Título do Gráfico
    plt.plot(x, y, '-o') # Os pontos de X e Y serão marcados por pontos (o) e ligados por linhas contínuas (-)
    plt.xscale('linear') # Escala linear do Eixo X
    plt.yscale('log') # Escala logarítmica do Eixo Y
    plt.show() # Exibe o Gráfico
def graf_dlog(self): # Função que criará o Gráfico em escala Di-Log
```

Primeiramente, as funções dos gráficos chamam novamente os valores de X e Y, assim como foi feito pela função da Regressão Linear Simples.

```
x, y = self.upload_arq() # Chama os dados de X e Y
```

Depois, a função *self.graf_padrao()* chama a variável *"yi"*, que foi retornada pela função *self.reg_lin()*. Essa variável será usada para formar a reta da Regressão Linear Simples.

```
calculos, yi, pdf = self.reg_lin()
```

Obs.2: a função *self.graf_padrao()* também chama as variável *"calculos"* e *"pdf"*, pelos mesmos motivos descritos na Obs.1.

Obs.3: as funções *self.graf_mlog()* e *self.graf_dlog()* não chamam as variáveis retornadas pela *self.reg_lin()*, pois seus gráficos não são formados pela Regressão Linear Simples.

Posteriormente, a função *self.graf_padrao()* chama a variável *"erro"*, que foi retornada pela função *self.erro()*. Essa variável será utilizada para determinar o tamanho das Barras de Erro no Gráfico Padrão.

```
erro = self.erro() # Chama o valor do Erro
```

Obs.4: as funções *self.graf_mlog()* e *self.graf_dlog()* também não chamam essa variável pois eles não necessitam de barras de erro.

Após chamar todas as variáveis necessárias, os gráficos criam sua primeira variável, denominada *"fig"*. Essa variável utiliza a biblioteca *plt* para criar a primeira 'figura' onde será formado o gráfico. Depois é tem-se a variável *"grid"*, que utiliza a biblioteca *gs* para determinar como será a grade da figura formada anteriormente.

```
grid = gs.GridSpec(7, 3) # Define o tamanho como uma grade para a Figura
```

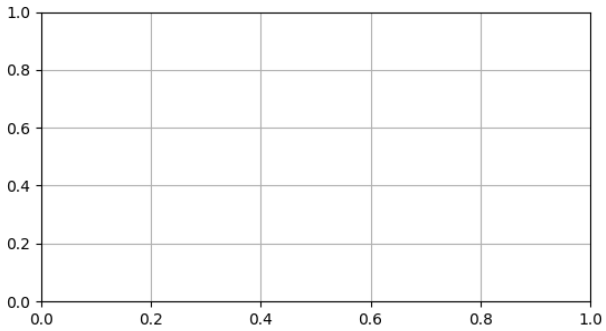
Com isso, a figura formada será composta por 7 linhas e 3 colunas.

Agora, as funções dos gráficos criam a variável *"grafico"* que determinará, junto com a *"fig"* e a *"grid"* em quais linhas e colunas da figura será construído o gráfico.

```
grafico = fig.add_subplot(grid[:5, :]) # Define o espaço da Figura que será ocupada pelo Gráfico
```

Ou seja, o gráfico ocupa as 5 primeiras linhas e todas as colunas da figura.

Além disso, com o uso da variável *"grafico"*, as funções dos gráficos realizam o comando *grafico.grid(True)*, que ativa as linhas de grade no gráfico, como por exemplo:




```
grafico.grid(True) # Ativa as linhas de grade no Gráfico
```

Depois, as funções dos gráficos utilizam a biblioteca *plt* para definir o resto das configurações dos gráficos:

- *plt.title* definirá o título dos gráficos. Os títulos são ***Gráfico Padrão X × Y*** para a *self.graf_padrao()*, ***Gráfico Mono-Log X × Y*** para a *self.graf_mlog()* e ***Gráfico Di-Log X × Y*** para a *self.graf_dlog()*.
- *plt.plot* definirá a linha dos gráficos. A *self.graf_padrao()* usará a variável *"yi"* para criar a reta da Regressão Linear Simples, formada pela Função de Ajuste.

```
plt.plot(x, yi) # Cria a Reta da Regressão Linear Simples
```

Já a *self.graf_mlog()* e a *self.graf_dlog()* fará uma linha que passa por todos os pontos de X e Y.

```
plt.plot(x, y, '-o') # Os pontos de X e Y serão marcados por pontos (o) e ligados por linhas contínuas (-)
```

- A função *self.graf_padrao()* usará o comando *plt.errorbar* para criar as barras de erro. Elas estarão nos pontos determinados por X e Y. O tamanho delas é o Desvio Padrão dos Dados, e será fornecido pelo usuário na função *self. erro()*. Os pontos estarão na cor vermelha, enquanto as barras estarão na cor preta.

```
plt.errorbar(x, y, yerr=erro, fmt='ro', ecolor='black', linestyle=None, capsize=2) # Cria as Barras de Erro, com pontos vermelhos (ro)
```

- *plt.xscale* e *plt.yscale* definirão as escalas dos eixos X e Y, respectivamente.

➢ Gráfico Padrão

```
plt.xscale('linear') # Escala linear do Eixo X
plt.yscale('linear') # Escala linear do Eixo Y
```

➢ Gráfico Mono-Log

```
plt.xscale('linear') # Escala linear do Eixo X
```

```
plt.yscale('log') # Escala logarítmica do Eixo Y
```

- Gráfico Di-Log

```
plt.xscale('log') # Escala logarítmica do Eixo X
plt.yscale('log') # Escala logarítmica do Eixo Y
```

- Por fim, o comando *plt.show()* permite que os gráficos sejam exibidos.

```
plt.show() # Exibe o Gráfico
```

INTERFACE GRÁFICA:

A Interface Gráfica do Pymath foi feita utilizando a biblioteca *tkinter*. Primeiramente, cria-se uma variável *"root = Tk()"* responsável pela criação da janela. Além disso, estabeleceu-se um *"loop"* para que o programa fique aberto e uma função *"self"* para facilitar a criação e desenvolvimento do código.

```
root = Tk() # Janela da Interface
root.mainloop() # Faz a Interface rodar em loop
def __init__(self): # Define os parâmetros da classe
```

Com a janela criada, define-se algumas configurações da janela para posteriormente serem criados "widgets" fundamentais para o funcionamento do programa, tais como botões, janelas, menus, entre outras funcionalidades.

Para isso, existe a classe denominada *"Aplication"*, para facilitar e organizar a elaboração do código, mostrando mais claramente as definições adotadas pelo programa. Dentro da classe, existe a função *__init__(self)*, que definindo os parâmetros da classe, ou seja, as funções que serão ativadas logo que o programa começar a rodar.

```
class Aplication():

    def __init__(self): # Define os parâmetros da classe
        root = Tk() # Janela da Interface
        self.root = root
        self.arquivo_salvo = False
        self.tela() # Função de configurações da Janela
        self.frames() # Função de divisões da Tela
        self.label() # Função de criação dos Textos
        self.botoes() # Função de criação dos Botões
        self.Menus() # Função de Barra de Menu
        root.mainloop() # Faz a Interface rodar em loop
```

Agora, a função *self.tela()*, irá definir as configurações da janela da Interface

- A função *"self.root.title"* é responsável pela criação de um título, ou seja, "Pymath".
- A função *"self.root.configure"* é responsável pela configuração visual da interface, que foi definido que o fundo – "background" ficasse da cor turquoise1.
- A função *"self.root.geometry('750x650')"* definiu o dimensionamento da interface.
- A função *"self.root.resizable(False, False)*" delimitou que, o usuário não pode – "False" arrastar ou comprometer de qualquer forma, aumentando ou diminuindo, o dimensionamento da interface, ou seja, a dimensão da interface (750x650) é fixa.

```
def tela(self): # Configurações da Tela
        self.root.title("Pymath") # Título
        self.root.configure(background = 'turquoise1') # Cor principal
        self.root.geometry("750x650") # Tamanho da Interface
        self.root.resizable(False, False) # Tamanho fixo
```

Posteriormente, a função *"self.frames()"* é responsável por armazenar e dimensionar a divisão e configuração da tela inicial do programa.

```
def frames(self): # Divisões da Tela
  # Caixa 1 - Introdução #
    self.frame_1 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60',
highlightthickness = 4)
  self.frame_1.place(relx = 0.05, rely = 0.02, relwidth = 0.42, relheight = 0.2)
  # Caixa 2 - Arquivo #
    self.frame_2 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60',
highlightthickness = 4)
  self.frame_2.place(relx = 0.05, rely = 0.24, relwidth = 0.42, relheight = 0.06)
  # Caixa 4 - Erro #
    self.frame_4 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60',
highlightthickness = 4)
  self.frame_4.place(relx = 0.05, rely = 0.32, relwidth = 0.42, relheight = 0.06)
  # Caixa 6 - Regressão Linear Simples #
    self.frame_6 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60',
highlightthickness = 4)
  self.frame_6.place(relx = 0.09, rely = 0.43, relwidth = 0.35, relheight = 0.08)
  # Caixa 7 - Padrão #
    self.frame_7 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60',
highlightthickness = 4)
  self.frame_7.place(relx = 0.05, rely = 0.53, relwidth = 0.42, relheight = 0.1)
  # Caixa 8 - MonoLog #
    self.frame_8 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60',
highlightthickness = 4)
  self.frame_8.place(relx = 0.05, rely = 0.65, relwidth = 0.42, relheight = 0.1)
  # Caixa 9 - DiLog #
    self.frame_9 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60',
highlightthickness = 4)
  self.frame_9.place(relx = 0.05, rely = 0.77, relwidth = 0.42, relheight = 0.1)
  # Caixa 10 - Cálculos #
```

```
    self.frame_10 = Frame(self.root, bd = 4, bg = 'white', highlightbackground = 'gray60', highlightthickness = 4)
    self.frame_10.place(relx = 0.52, rely = 0.02, relwidth = 0.42, relheight = 0.85)
```

Com a criação dos frames, a tela inicial possui diversas divisões. Foi feita essa divisão por dois motivos, o primeiro e principal foi para a facilitação da definição de botões que foram associados aos frames, facilitando no dimensionamento e na configuração, o segundo motivo foi puramente visual.

Inicialmente, cria-se uma variável denominada “self.frame_1”, na qual será definida pelo “Frame”, com as seguintes configurações dentro dos parênteses:

- *“self.root”* define que o Frame será criado na janela da Interface Gráfica.
- *“bd”* é responsável pela delimitação da borda.
- *“bg”* é responsável pela definição de uma cor ao “background”, ou seja, a cor de fundo do frame.
- *“highlightbackground”* define a cor da borda do frame.
- *“highlightthickness”* define a largura da borda no frame.

Na parte de baixo, o comando *“self.frame_1.place”* determina a limitação do frame, ou seja, demarca sobre toda a interface a posição do frame:

- *“relx”* delimita a origem do frame no eixo x.
- *“rely”* delimita a origem do frame no eixo y.
- *“relwidht”* delimita a largura relativa do frame na interface.
- *“relheight”* delimita a altura relativa do frame na interface.

O mesmo procedimento se repete para a criação dos outros Frames, adaptando seus tamanhos e posições de acordo com a organização da Interface

Agora, a função *"self.label()"* criará alguns textos (Label) e caixas de entrada (Entry) que ocuparão determinados frames da Interface.

```
def label(self): # Criação dos Textos
        # Texto 1 - Introdução
         self.lb_intro = Label(self.frame_1, text="Bem-Vindo ao Pymath!", bd=4, bg='gray86', fg='black', font=('Cambria Math',20))
        self.lb_intro.place(relx=0, rely=0, relwidth=1, relheight=1)
        # Texto 3 - Erro
         self.lb_erro = Label(self.frame_4, text="Desvio Padrão de Dados:", bd=1, bg='white', fg='black')
        self.lb_erro.place(relx=0, rely=0)
        # Entrada 2 - Erro
        self.erro_entry = Entry(self.frame_4, bd=2)
        self.erro_entry.place(relx=0.46, rely=0, relwidth=0.54, relheight=1)
        # Texto 4 - Cálculos
         self.lb_calculos = Label(self.frame_10, text="Cálculos:", bd=4, bg='gray86', fg='black', font=('Cambria Math',20))
        self.lb_calculos.place(relx=0, rely=0, relwidth=1, relheight=0.1)
```

- *"self.lb_intro"* é variável responsável pela criação da mensagem de introdução, com o texto "Bem-Vindo ao Pymath!"
- *"self.lb_erro"* é a variável que exibe o texto pedindo ao usuário informar o Desvio Padrão dos Dados, que será utilizado para a determinação do tamanho das barras de erro no Gráfico Padrão.
- *"self.erro_entry"* é a variável que cria a caixa de texto em que o usuário informará o Desvio Padrão dos Dados.
- *"self.lb_calculos"* é a variável que exibe o texto responsável por indicar onde ao cálculos da Regressão Linear Simples serão exibidos.

Todas essas variáveis foram determinadas como uma Label (Texto) ou uma Entry (Entrada) como em *"self.lb_intro = Label()"* e *"self.erro_entry = Entry()"*. Após isso, define-se as seguintes configurações dentro dos parênteses:

- *"self.frame_1"* é onde se estabelece em que região se localizará a caixa de texto. Nesse caso, *"self.frame_1"* é a localização da primeira Label.
    - *"text"* define o texto que deve ser gerado – determinado entre aspas.
    - *"bd"* é responsável pela delimitação da borda da caixa de texto.
    - *"bg"* é responsável pela definição de uma cor ao "background", ou seja, a cor de fundo da caixa de texto.
    - *"fg"* é responsável pela definição da cor das letras do texto.

- *"font"* determina a fonte usada pelas letras do texto.

Obs.5: A vírgula que acompanha a fonte usada delimita outro espaço que é responsável pela delimitação do tamanho das letras usadas. Ou seja, *"font = ('Arial', 20)"* determina que, a fonte da caixa de texto usada é a Arial, e o tamanho das letras corresponde ao tamanho 20.

Como visto anteriormente, o comando *"self.lb_intro.place()"* é responsável pela determinação da localização, dentro do frame. No exemplo da primeira Label:

- *"relx"* delimita a posição da origem da caixa texto, no frame, pelo eixo x.
- *"rely"* delimita a posição da origem da caixa texto, no frame, pelo eixo y.
- *"relwidht"* delimita a largura relativa da caixa de texto, no frame, na interface.
- *"relheight"* delimita a altura relativa da caixa de texto, no frame, na interface.

Agora, cria-se a função *"self.botoes()"*, responsável por criar os botões que realizarão diversos comando do Pymath.

```
def botoes(self): # Função de criação dos Botões
    # Botão 1 - Upload Arquivo
        self.bt_uparq = Button(self.frame_2, text = "Inserir Arquvio de Dados", command=self.upload_arq, bd = 2, bg = 'gray88', fg = 'black', font = ('Cambria Math', 9))
    self.bt_uparq.place(relx = 0, rely = 0, relwidth = 0.99, relheight = 1)
    # Botão 3 - Regressão
        self.bt_regress = Button(self.frame_6, text = "Regressão Linear Simples", command=self.impres_reglin, bd = 4, bg = 'gray88', fg = 'black', font = ('Cambria Math', 13))
    self.bt_regress.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
    # Botão 4 - Gráfico Padrão
    self.bt_pad = Button(self.frame_7, text = "Gráfico Padrão", command=self.graf_padrao, bd = 4, bg = 'gray88', fg = 'black', font = ('Cambria Math', 16))
    self.bt_pad.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
    # Botão 5 - Gráfico MonoLog
    self.bt_mono = Button(self.frame_8, text = "Gráfico Mono-Log", command=self.graf_mlog, bd = 4, bg = 'gray88', fg = 'black', font = ('Cambria Math', 16))
    self.bt_mono.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
    # Botão 6 - Gráfico DiLog
    self.bt_di = Button(self.frame_9, text = "Gráfico Di-Log", command=self.graf_dlog, bd = 4, bg = 'gray88', fg = 'black', font = ('Cambria Math', 16))

    self.bt_di.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
```

Primeiramente, define-se a variável *"self.bt_uparq = Button()"*, que é o primeiro botão da função. Após isso, define-se as seguintes configurações dentro dos parênteses:

- *"self.frame_2"* delimita a posição do botão, que neste caso, se encontra no segundo frame da interface.
- *"text"* cria um texto que se encontrará internamente no botão.

- *"command"* chama a função que possui o comando que será realizado pelo botão
- *"bd"* é responsável pela delimitação da borda do botão.
- *"bg"* é responsável pela definição de uma cor ao "background", ou seja, a cor de fundo do botão.
- *"fg"* é responsável pela definição da cor das letras do texto.
- *"font"* determina a fonte usada pelas letras do texto.

Analisado a parte da criação do botão, analisaremos agora a questão posicional do botão, representado pela variável "self.bt_save1.place"

Como visto anteriormente, o comando *"self.bt_uparq.place()"* é responsável pela determinação da localização, dentro do frame. No exemplo do primeiro Botão:

- *"relx"* delimita a posição da origem do botão, no frame, pelo eixo x.
- *"rely"* delimita a posição da origem do botão, no frame, pelo eixo y.
- *"relwidht"* delimita a largura relativa do botão, no frame, na interface.
- *"relheight"* delimita a altura relativa do botão, no frame, na interface.

Por fim, a última função da Interface Gráfica do Pymath é a *"self.Menus()"*, responsável pela criação da Barra de Menu.

```
def Menus(self): # Função que cria o Menu
   menubar = Menu(self.root) # Cria a barra de Menu
   self.root.config(menu=menubar)
   menu_opc = Menu(menubar, tearoff=0) # Cria 1ª Caixa na barra de Menu
   menu_ajd = Menu(menubar, tearoff=0) # Cria 2ª Caixa na barra de Menu
  menubar.add_cascade(label = "Opções", menu = menu_opc) # Nomeia de 'Opções' a 1ª Caixa
do Menu
   menubar.add_cascade(label = "Ajuda", menu = menu_ajd) # Nomeia de 'Ajuda' a 2ª Caixa
do Menu
   def reset(): # Função para Reiniciar o Pymath
      self.arquivo_salvo = False
         self.erro_entry.delete(0, END) # Limpa a entrada do Desvio Padrão
       # Limpa os cálculos da Regressão Linear
       self.imp = Label(self.frame_10, text="", font = ('Arial', 14), bg='white')
       self.imp.place(relx = 0.16, rely = 0.15, relwidth = 0.70, relheight = 0.80)
       self.imp.pack
   def openweb(): # Função com o link do Site do Pymath
       url = "https://sites.google.com/usp.br/pymath/como-utilizar-o-pymath" # Link do Site do
Pymath
       webbrowser.open_new(url)
     menu_opc.add_command(label="Reiniciar", command=reset) # Botão que executa a
Reinicialização
    menu_ajd.add_command(label = "Site Python", command=openweb) # Botão que abre o
Site do Pymath
```

- *"menubar=Menu(self.root)"* cria a barra de menu encontrada na parte superior esquerda da interface.
- *"self.root.config(menu=menubar)"* amplia a configuração da barra de menu como variável, facilitando o desenvolvimento de botões dentro da barra de menu.
- *"menu_opc"* define a primeira caixa na Barra de Menu.
- *"menu_ajd"* define a segunda caixa na Barra de Menu.

O comando *"menubar.add_cascade()"* é responsável pela configuração dos Menus de Cascata da Barra de Menu. Após os parênteses, inicia-se a configuração e determinação dos Menus de Cascata:

- *"label"* é texto que irá representar o botão na Barra de Menu.
- *"menu=menu_opc"* ou *"menu=menu_ajd"* determina em qual das caixas da Barra de Menu o botão estará localizado.

Obs.6: Menus de Cascata são botões da Barra de Menu que guardam outros botões dentro de si. São esses outros botões que realmente realizarão comandos dentro do Pymath.

A função *"reset()"* é responsável pela reinicialização do Pymath, eliminando o Arquivo de Dados e o Desvio Padrão de Dados que foram inicialmente inseridos, além de apagar a caixa de texto com a exibição dos cálculos da Regressão Linear Simplificada.

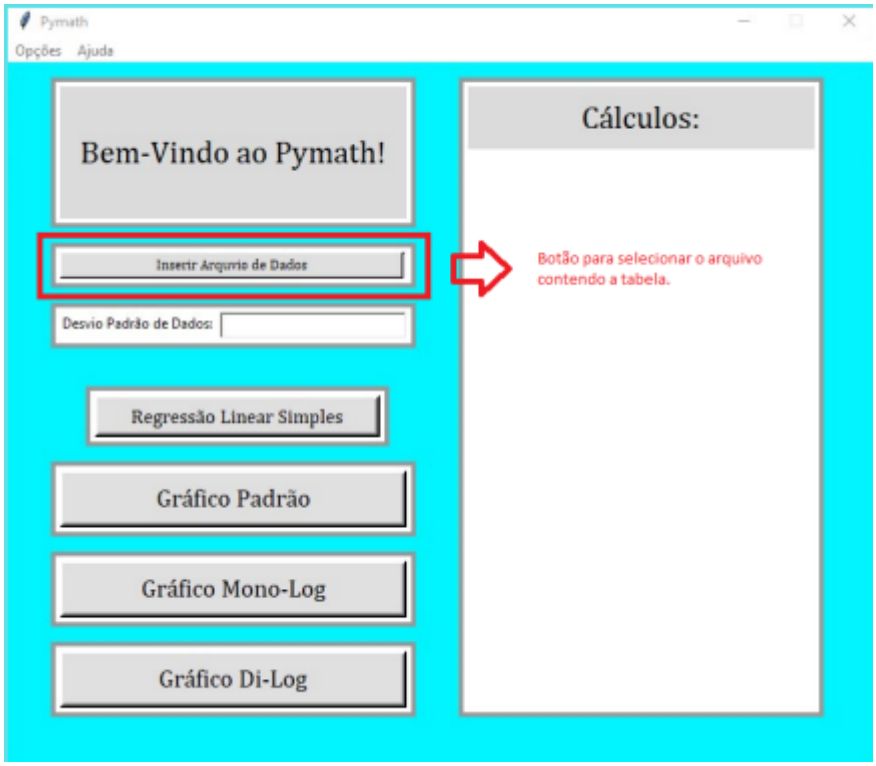
A função *"openweb()"* é responsável por adicionar o link do site do Pymath (https://sites.google.com/usp.br/pymath/como-utilizar-o-pymath) na Barra de Menu.

O comando *"menu_opc.add_command(label='Reiniciar', command=reset)"* cria um botão, dentro do Menu de Cascata *menu_opc*, com o título "Reiniciar" e que realiza a função *"reset()"*.

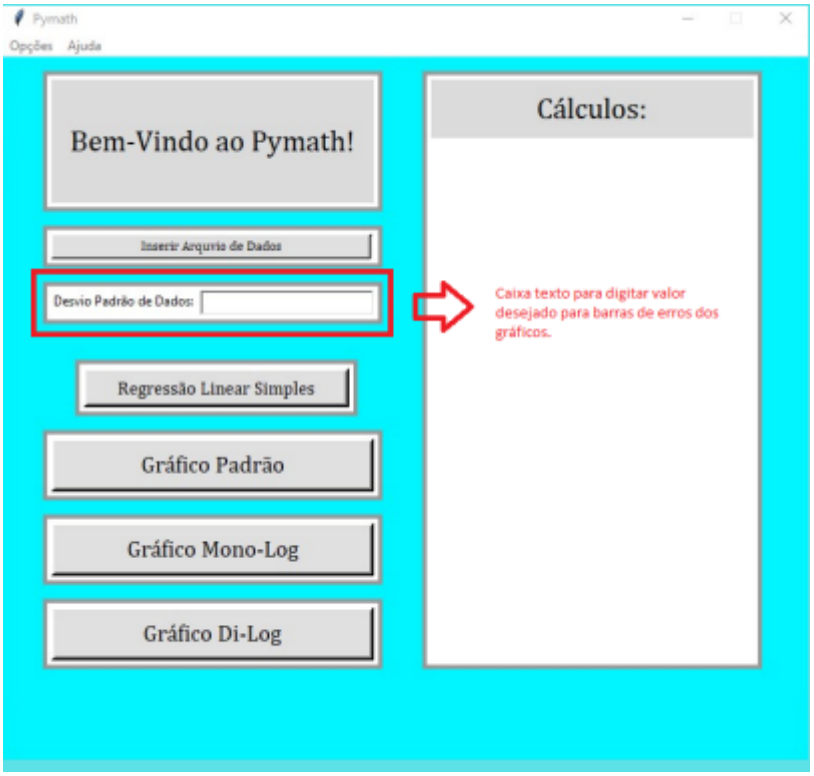
O comando *"menu_ajd.add_command (label='Site Python', command=openweb) "* cria um botão, dentro do Menu de Cascata "*menu_ajd"*, com o título "Site Python" e que realiza a função *"openweb()"*.

Encerrando o programa, é necessário chamar a classe *Aplication()*, para que ela realize o comando *root.mainloop()* e exiba a interface com todas as suas configurações.

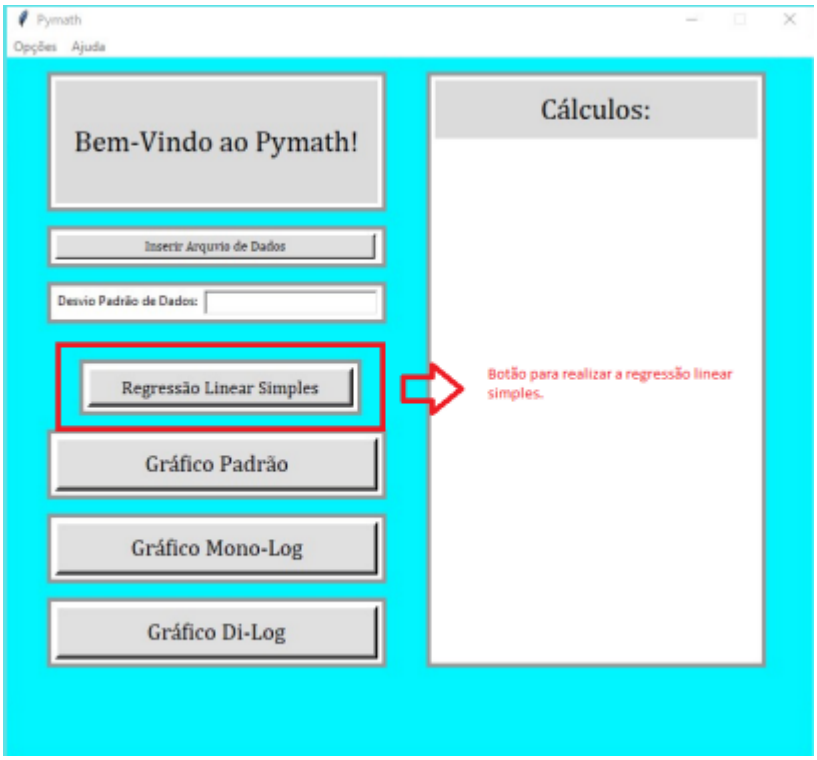
4. COMO USAR O PROGRAMA



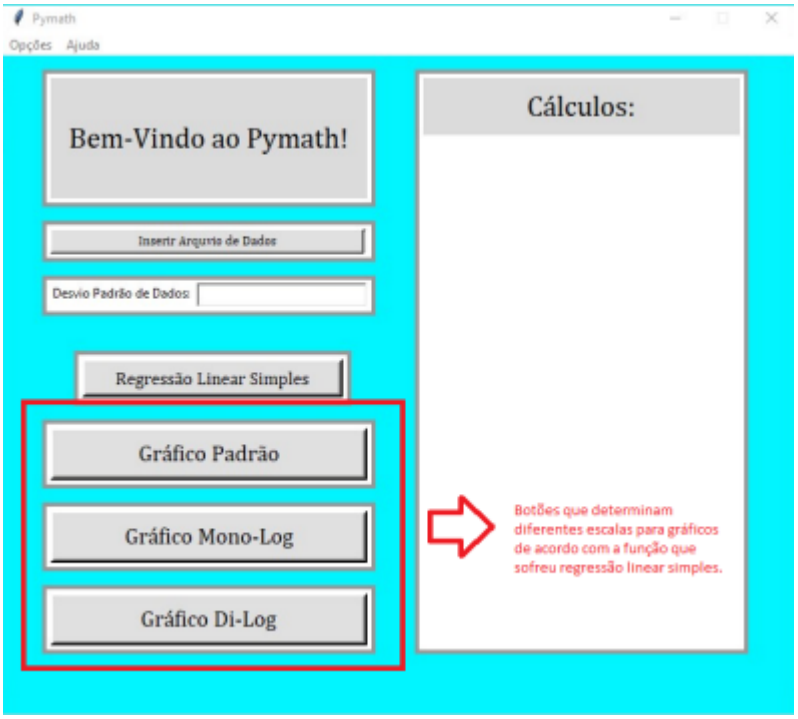
1. Selecione a tabela que contém os dados para realização da regressão linear simples, a partir do botão "Inserir Arquivo de Dados", como apresentado na imagem abaixo.



2. Após inserir a tabela, é necessário digitar na caixa de texto "Desvio Padrão de Dados" um valor para as barras de erro dos gráficos, como indicado na imagem abaixo.



3. Ao clicar no botão "Regressão Linear Simples", será mostrado para o usuário na parte "Cálculos" os cálculos referentes a regressão linear simples.



4. Também é possível escolher qual escala para o gráfico será utilizada, o mesmo será mostrado após o acionamento botão.

5. Por fim, a regressão linear simples aparecerá no quadro de cálculo, como mostrado na figura abaixo.

5. TERMOS E CONDIÇÕES

MIT License

CONTATOS:

Gabriel de Toledo Paula: gabrieldetoledopaula@usp.br
Gabriel dos Santos Melo: gabriel_melo@usp.br
Jorge Camasmie Nunes: jocamasmienunes@usp.br
Matheus Cruvinel de Souza: matheus_csouza@usp.br
Oswaldo Henrique de Freitas: ozhenriqueff@usp.br
Vinicius Matias Florentino: vinicius.matias@usp.com

REFERÊNCIAS:

https://lamfo-unb.github.io/2020/02/07/O-M%C3%A9todo-dos-M%C3%ADnimos-Quadrados-Ordin%C3%A1rios-e-Regress%C3%A3o-Linear-Simples/
https://pt.wikipedia.org/wiki/M%C3%A9todo_dos_m%C3%ADnimos_quadrados#Hist%C3%B3ria
https://psicometriaonline.com.br/category/post/tamanho-de-efeito/
https://numpy.org/
https://docs.python.org/3/library/tkinter.html
https://matplotlib.org/

AGRADECIMENTOS:

Deixamos aqui nosso agradecimento ao nosso professor doutor de Computação Científica em Python; Luiz Tadeu Fernandes Eleno, e aos nossos monitores; Lucas Campos Achcar e Leonardo Izaias que também nos ensinaram e ajudaram durante a programação do Software.