EESC • USP	**Universidade de São Paulo** **Escola de Engenharia de São Carlos** **Laboratório de Análise Computacional em Sistemas Elétricos de Potência**	**LACOSEP**

Programação Básica em Python

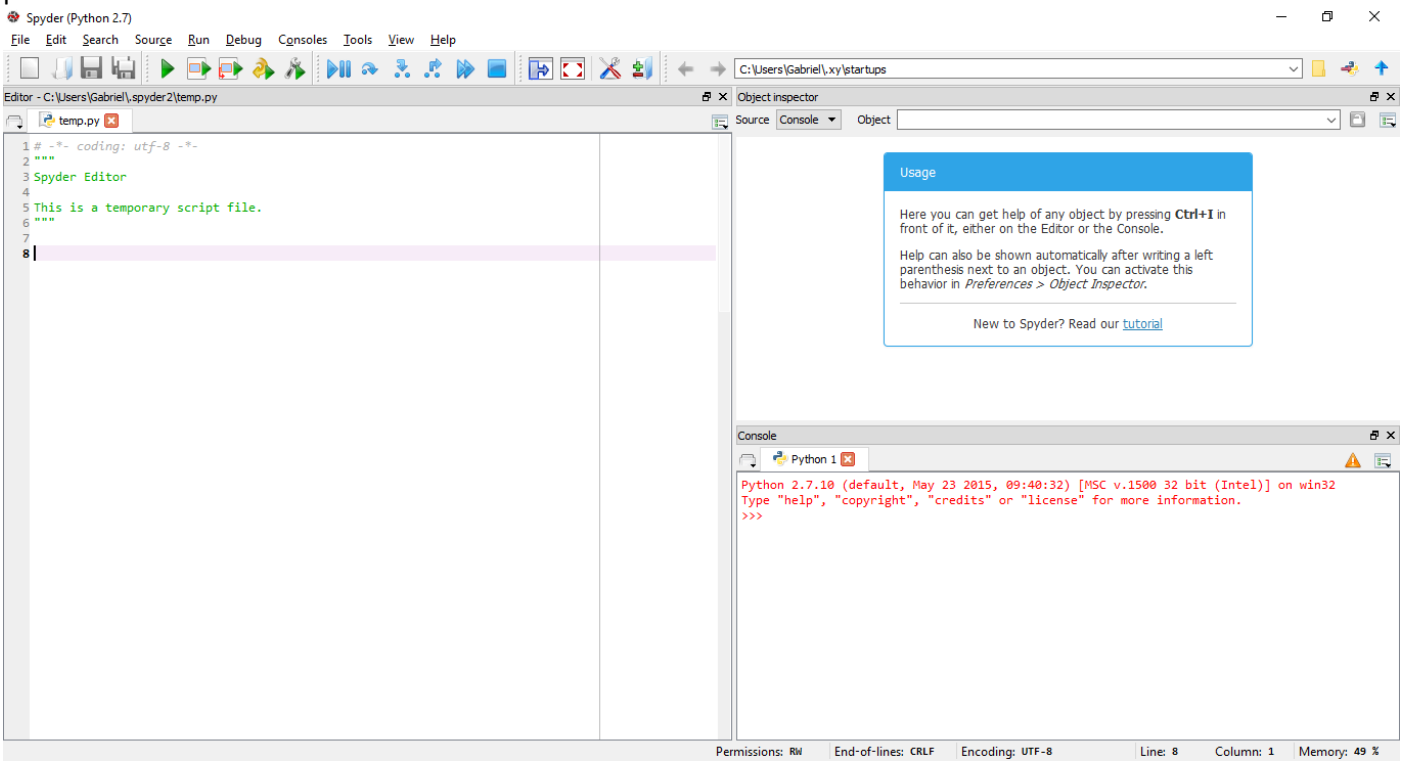
Gabriel Negrelli – gabriel.jose.gomes@usp.br

I) Introdução

Os arquivos do matlab tem extensão '.py'.
Existem dois tipos de arquivos:
- Script: não aceita parâmetros de entrada e saída
- Função: aceita parâmetros de entrada e saída. As variáveis dentro da função são variáveis locais.

II) Editor de Programas

Existem diversos editores para programação em Python. Neste tutorial será utilizado o software Spyder, um editor instalado juntamente com o Python(x,y), um software de desenvolvimento voltado para engenharia. O Python(x,y) pode ser encontrado no site http://python-xy.github.io/ e a versão utilizada é a 2.7.10.0.
A vantagem de se utilizar o Python(x,y) está no fato de ele vir com as bibliotecas mais utilizadas nas aplicações de engenharia já instaladas. O Spyder tem uma interface simples, como o editor de código do lado esquerdo e o console, alem de outras ferramentas, do lado direito. O layout das ferramentas pode ser alterado da forma que melhor agradar o usuário. Pode ser acionado presionando o simbolo do botão "new file". Uma vez criado o arquivo .py, coloque nome dele "a_teste1" (que a partir de agora será denominada arquivo principal) em uma pasta da sua preferência.

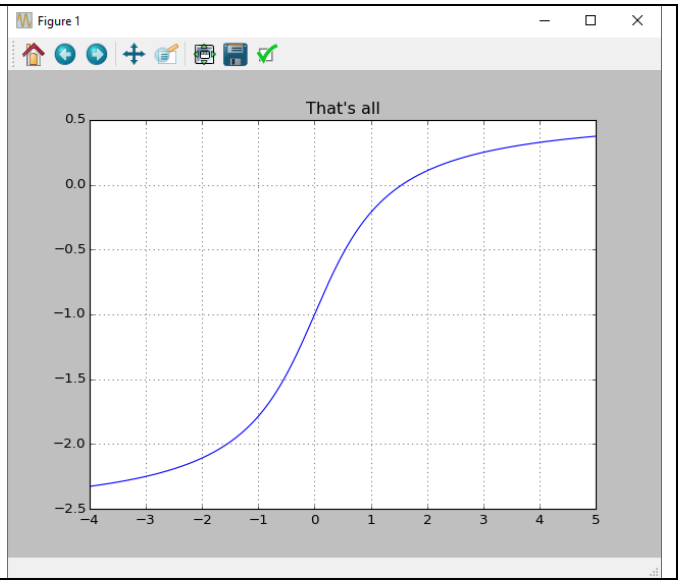


A partir desta tela, fazendo click em cada item do menu do editor de programas, pode-se obter muitas opções. Fica a critério de cada estudante aprender a maior parte destas opções. Por exemplo fazendo click no menu "Edit", pode-se transformar um trecho selecionado em comentário [Comment] ou desfazer o comentário [Uncoment]. [Indent] e [Unindent] ajustam os recuos de forma organizada o trecho selecionado, etc.

Exemplo 1: Plotando uma equação algébrica dentro de um arquivo Script.

Crie o código abaixo dentro do arquivo "a_teste1.py". Após para executá-lo pressione a tecla F5 que pode ser encontrado dentro do menu "debug".

```
#Importando bibliotecas necessárias
import matplotlib.pyplot as plt
import numpy as np

#Criando vetores para serem plotados
t = np.arange(-4.0, 5.0, 0.01)
s = np.arctan(t) - 1

#Plotando valores
plt.plot(t, s)
plt.title('That\'s all')
plt.grid(True)
plt.show()
```

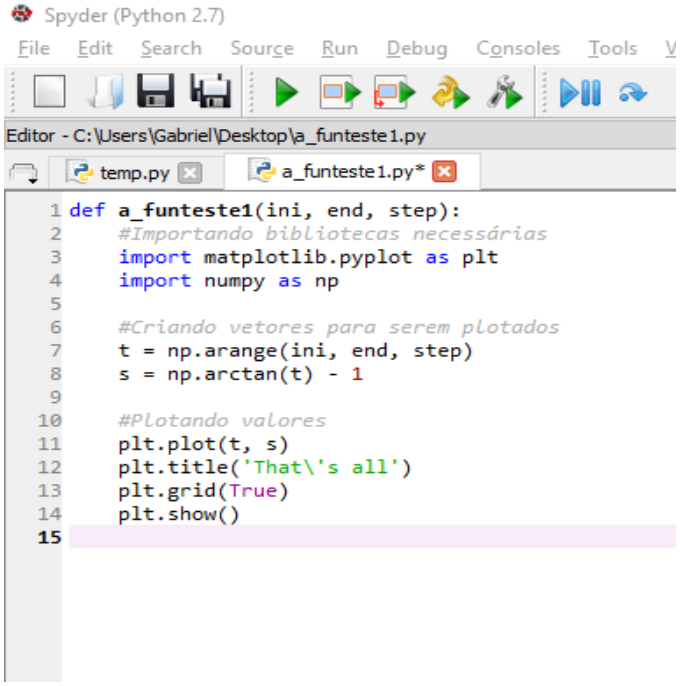


II) Arquivos de função

Parte de um programa que já funciona corretamente pode ser tratado como uma sub-rotina ou função para a qual é necessária alguns parâmetros de entrada e saída.

Exemplo 2: Plotando uma equação algébrica usando uma função.

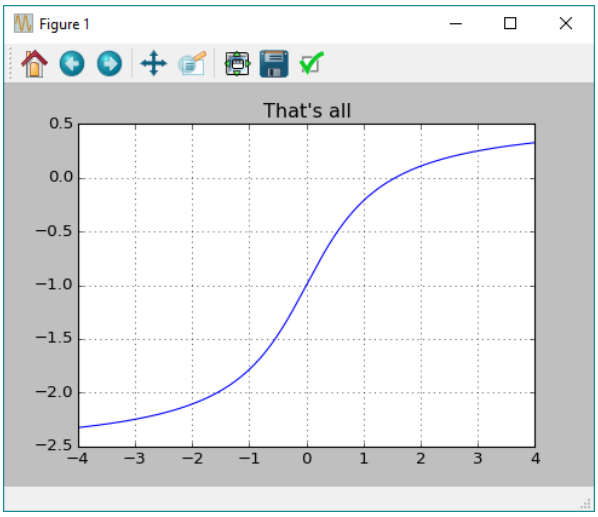
Crie uma função com nome “a_funteste1.m” e coloque o código como segue na figura.



Execução de uma função:

Para executar a função basta chamá-la pelo console com os atributos de ini, end e step.

```
y=a_funteste1(-4,4,0.001)
```



III Depuração de um programa.

O sucesso de um programador baseia-se em quão rápido pode encontrar detectar erros na execução do programa. Estes erros podem estar associados à digitação errada do código, erro da lógica de programação, etc. A depuração de um programa visa encontrar estes erros e eliminá-los a fim de que o programa funcione. No Spyder estes comandos podem ser encontrados no menu “Debug”

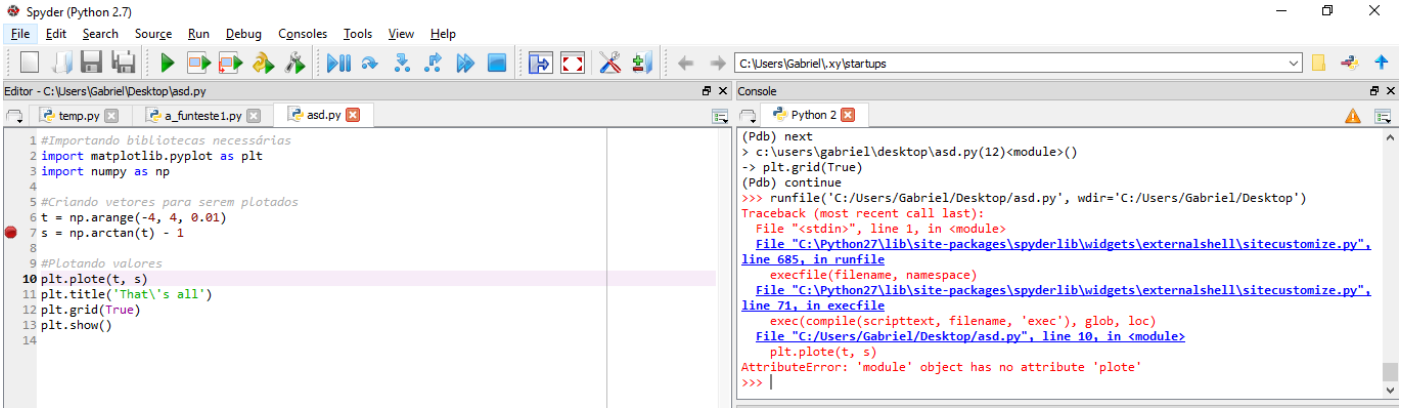
- **[step]:** executa a próxima linha
- **[step into]:** executa a próxima linha e caso uma função seja chamada, entra na execução dela.
- **[step return]:** Sai da subfunção e executa a próxima linha de comando
- **[Run/Continue]:** Inicia a execução em modo depuração ou continua até o final da rotina.
- **[Exit]:** Sai do modo de depuração
-**[Set/clear Breakpoint]:** Controla configuração de pontos de parada

Estes recursos também estão no menu com ícones.

Um breakpoint também pode ser configurado com dois cliques na área esquerda do número de linha. O breakpoint será exibido com um ponto vermelho.

Exemplo3: No código anterior pratique os comandos de depuração.

Caso o programa não funcione, uma mensagem de erro é editada no console. Lendo esta mensagem pode-se identificar facilmente qual é o tipo de erro e onde é localizado. Para isto, basta ler atentamente o que aparecer escrito em vermelho no console.



Como pode observar-se o erro que indicou que a função “plt.plote” não existe. Como é de esperar a palavra reservada era plot e não plote.

IV Comandos de Programação

a) if, elif, else:

```
if < Condição 1 >:
    <Comandos>
elif < Condição 2 >:
    <Comandos>
else:
    <Comandos>
```

Exemplo 4:
Crie uma função que permita calcular os valores da função

$$f(x) = \begin{cases} 1, se\ x < -1 \\ x^2, se\ -1 \leq x \leq 1 \\ -x + 2, se\ x > 1 \end{cases}$$

`def a_fun_avalia(x):`
`  if x < -1:`
`  return 1`
`  elif x >= -1 and x < 1:`
`  return x**2`
`  else:`
`  return -x + 2`
	Na tela de console:
	>>Y = a_fun_avalia(4)
	Y = -2
	>>Y = a_fun_avalia(1)
	Y = 1
	>>Y = a_fun_avalia(-4)
	Y = 1

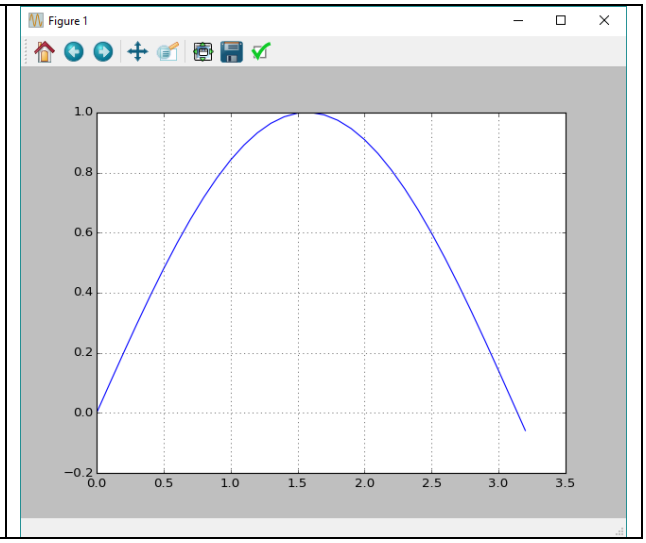
b) while: Repete a execução enquanto a condição dentro do while é satisfeita.

while<condição>:

<comandos>

Exemplo 5: Crie um programa que repita armazene em uma tabela os valores de x e sen(x) partindo de x=0 até pi com passos de 0.1.

```
from math import pi, sin
import matplotlib.pyplot as plt
import numpy as np


x = 0
tabela = np.array([x, sin(x)])
while x < pi:
    x += 0.1
    tabela = np.vstack((tabela,[x, sin(x)]))

plt.plot(tabela[:,0], tabela[:,1])
plt.grid(True)
```



c) for-else: Repete a execução dos comandos um número específico de vezes.

for<Valor>=inicio:Incremento:fim

<comandos>

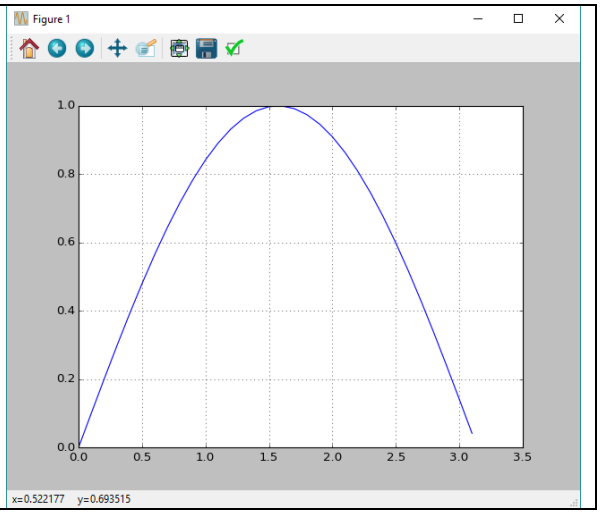
end;

Obs: Se o incremento não for informado, o Matlab o considera como 1.

Exemplo 6:

Refaça o programa anterior usando **for-end.**

```
from math import pi, sin
import matplotlib.pyplot as plt
import numpy as np

tabela = np.empty(shape=[0, 2])
for x in np.arange(0,pi,0.1):
    tabela = np.vstack((tabela,[x, sin(x)]))
else:
    plt.plot(tabela[:,0], tabela[:,1])
    plt.grid(True)
```

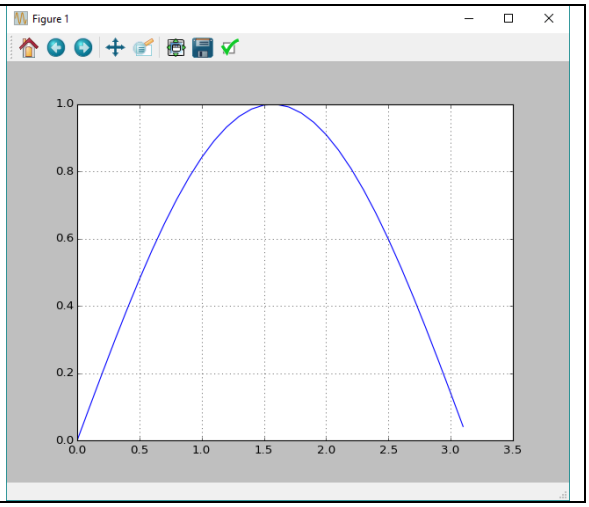


d) break: Interrompe a execução de um programa dentro de um comando while-end ou for-end e transfere para o seguinte end. Pode ser utilizada junto ao comando while para fazer a verificação de parada de um programa dentro do while da seguinte forma.

Exemplo 7:

Refaça o programa anterior usando **while e break**

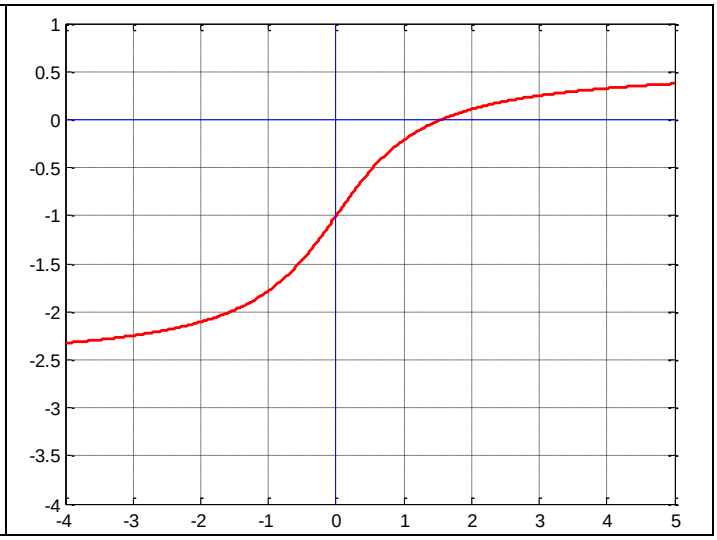
```
from math import pi, sin
import matplotlib.pyplot as plt
import numpy as np

x = 0
tabela = np.empty(shape=[0, 2])
while 1:
    tabela = np.vstack((tabela,[x, sin(x)]))
    x += 0.1
    if x > pi:
        print("You've got it!")
        break

plt.plot(tabela[:,0], tabela[:,1])
plt.grid(True)
```



V Métodos Numéricos:

Encontre a solução da raiz da seguinte equação:

```
f(x) = atan(x) + x -1=0
```



V.1) Pelo método de Gauss
X = 1- atan(x);
$x^{K+1} = 1 - \text{atan}(x^K)$
Ponto inicial $x^o=0$;
Critério de parada: $|x^{K+1} - x^K| <$ tolerancia; tolerância=0.001;
Além disso pode se limitar as iterações a 100;

Algoritmo:

1) Inicialize as variáveis xo=0; tol=0.001;maxite=100;cont=0;
2) avalie: x1=1-atan(xo);
   cont=cont+1;

3) Verifique a condição de parada foi atingido

   Se cont > maxite pare;
   Se $|x1 - x0| <$ tolerância, pare
   Caso contrario faça
   Xo=x1 e volte ao passo 2.

```
#Parte 1 Pelo metodo de Gauss
#1)Importando a biblioteca numPy
import numpy as np
#2)Inicializando as variáveis
x0, tol, maxite, cont = 0, .001, 100, 0
while 1:
    #3) Avalie
    x1 = 1 - np.arctan(x0)
    cont += 1
    #Verifique a condição de parada
    if cont > maxite:
        print("Passou do limite de iteracoes!")
        break
    if abs(x1 - x0) < tol:
        print("Convergiu para %f em %d iteracoes" %(x1,cont))
        break
    else:
        x0 = x1
```

Presionando F5 pode ver-se o resultado

Convergiu para 0.519889 em 30 iteracoes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V.1) Pelo método de Newton

f(x)= atan(x) + x - 1 ;

df(x) = 1/(1+x^2) +1;

x^{K+1} = x^K –inv(df(x^K))*f(x^K)

Ponto inicial x^o=0;

Critério de parada: | x^{K+1} - x^K|<tolerancia; tolerância=0.001;

Além disso pode se limitar as iterações em 100;

Algoritmo:

1) Inicialize as variáveis x0=0; tol=0.001;maxite=100;cont=0;
2) avalie:
   f(x1) = atan(x0) + x0 – 1;
   df(x1) = 1/(1+$x0^2$) +1;
   x1 = x0 –inv(df(x0))*f(x0)
   cont=cont+1;

3) Verifique a condição de parada foi atingido
   Se cont > maxite pare;
   Se| x1 – x0|<tolerância, pare
   Caso contrario faça
   xo=x1 e volte ao passo 2.

```
#Parte 2 Pelo metodo de Newton
#1)Importando a biblioteca numPy
import numpy as np
#2)Inicializando as variáveis
x0, tol, maxite, cont = 0.5, .001, 100, 0
tabela = np.empty(shape=[0, 5])
while 1:
  #3) Avalie
  fx = np.arctan(x0) + x0 - 1
  dfx = 1/(1 + x0**2) + 1
  x1 = x0 -fx*(dfx**-1)
```

```
    cont += 1
    tabela = np.vstack((tabela, [cont, x0, fx, dfx, x1]))
    #Verifique a condição de parada
    if cont > maxite:
        print("Passou do limite de iteracoes!")
        break
    if abs(x1 - x0) < tol:
        print("Pelo método de Newton")
        print("Convergiu para %f em %d iteracoes" %(x1,cont))
        print("iter    x0     fx     dfx     x1")
        print(tabela)
        break
    else:
        x0 = x1
```

Resultado:

```
Pelo método de Newton
Convergiu para 0.520269 em 2 iteracoes
          iter                x0                 fx                  dfx                 x1
[[  1.00000000e+00   5.00000000e-01  -3.63523910e-02   1.80000000e+00    5.20195773e-01]
 [  2.00000000e+00   5.20195773e-01  -1.30844322e-04   1.78702749e+00    5.20268992e-01]]
```

Referências:

[1] Fluent Python LUCIANO RAMALHO, Editora O'Reilly, 2015, 1a Edição