Introdução ao Python (x,y)

Bruna Santos

June 1, 2010

c0701003@alunos.fc.up.pt

Centro de Matemática da Universidade do Porto

Agradecimento

Gostaria de agradecer à Dr.ª Ana Paula Rocha pelas correcções e sugestões efectuadas neste tutorial.

Abstract

Este tutorial é uma breve introdução à extensão do software livre Python, o Python (x,y). Esta extensão pode ser utilizada como uma *(mega)* calculadora, dispondo de uma enorme variedade de constantes, operações e funções matemáticas predefinidas. Apresentar-se-á algumas funcionalidades dos módulos *Numpy, Scipy e Matplotlib*. Além disso, será feita uma introdução ao interface gráfico *Sypder*. A arquitectura do interface Spyder está organizado em diversas componentes possibilitando uma maior interatividade com o utilizador.

Contents

1 Introdução

1.1 O que é Python (x,y)?

O Python (x,y) é um software livre vocacionado para cálculos científicos e numéricos, análise de dados, visualização de gráficos em duas e três dimensões, entre outras funcionalidades. O download do programa pode ser feito através do link: `http://www.pythonxy.com/`

Possuí diversos ambientes integrados, nomeadamente o Spyder. Este ambiente caracteriza-se por ser intuitivo, interactivo e semelhante ao Matlab.

1.2 Conceitos Básicos

Antes de iniciarmos o estudo e a descrição dos módulos assim como dos pacotes, no Python (x,y), é importante clarificar os conceitos sobre estes.

Um módulo em Python, é um conjunto de programas que foram criados para serem aproveitados para um determinado objectivo. Por exemplo, o módulo *math* contém funções matemáticas como seno, cos-seno, tangente, etc. Contém, também, constantes matemáticas como o π e o número de euler.

Um pacote designa-se como hierarquias dos módulos. Por exemplo, o Numpy é a base para que o Scipy funcione, ou seja, trata-se de um dependência computacional.

1.2.1 Numpy

O NumPy é um módulo da linguagem Python que permite trabalhar com vectores e matrizes multidimensionais .Possui diversas ferramentas sofisticadas, onde se destacam:

- Ferramentas de álgebra linear;
- Transformadas de Fourier básicas;
- Ferramentas para geração de números aleatórios;.

1.2.2 SCIPY

O Scipy é outro módulo da linguagem Python. Implementa diversas algoritmos de cálculo científico e complementa o suporte de vectores multidimensionais do Numpy. Os sub-módulos do Scipy podem ser utilizados para diferentes objectivos, nomeadamente, implementação de algoritmos de integração numérica, processamento de sinal e imagem, optimização, entre outros. Destacam-se:

- *OPTIMIZE* - Implementa algoritmos de optimização e minimização;
- *SIGNAL* - Rotinas de Processamento de Sinal;
- *INTEGRATE* - Integração e resolução de equações diferenciais ordinárias;

- *LINSOLVE* - Resolução de sistemas de equações lineares
- *INTERPOLATE:* Funções de interpolação;
- *STATS*: Possui um conjunto de distribuições discretas e contínuas[1] assim como funções estatisticas usuais(media, desvio padrão, entre outras).

2 O Ambiente Spyder

O interface Spyder possui algumas potencialidades na apresentação dos resultados, gestão da variáveis, formatação numérica dos resultados, edição da linha de comandos, entre outros. Nesta secção, descrevemos algumas das suas potencialidades.

2.1 Ambiente Gráfico: *Spyder*

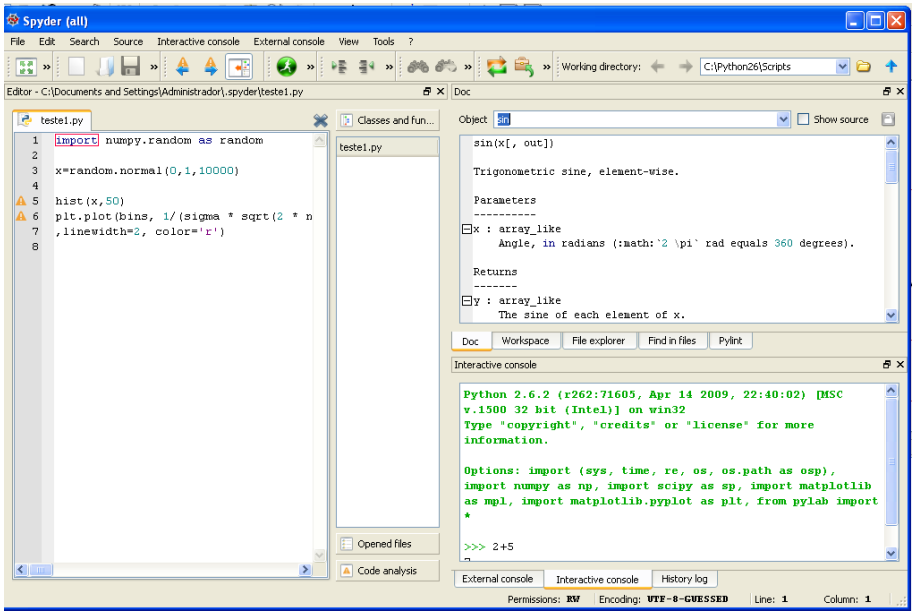


Figure 2.1: Ambiente Interativo Spyder

2.2 Help

Para obtermos informações sobre qualquer função disponivel no Python (x,y) recorremos ao comando *help*. Por exemplo, se pretendemos saber como utilzar a função ***sin***, basta fazer:

[1] Dez distribuições discretas e oitenta e uma distribuições conínuas

>>>help (sin)
obtendo-se uma descrição da função sin ()

Figure 2.2: Comando: *help*

Outa forma é através da opção *Object* do interface do Spyder:

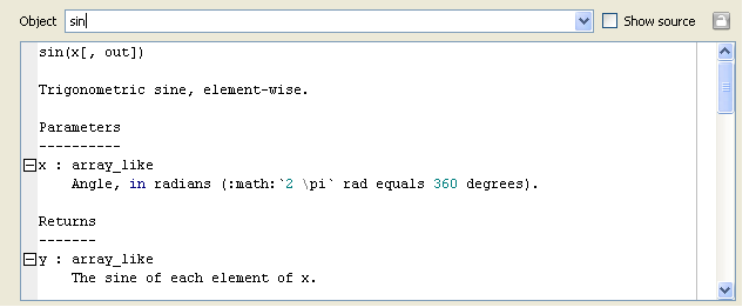


Figure 2.3: Descrição da opção *Object*

2.3 Consola Interativa

Uma expressão ou operação é digitada na linha de comandos ">>>". Para avaliar a expressão introduzida basta clicar na tecla Enter

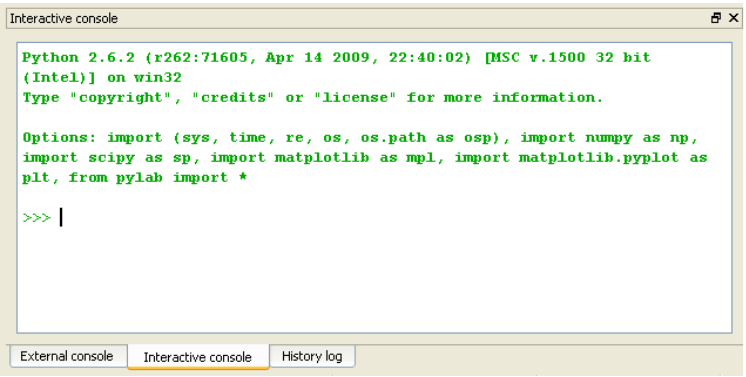
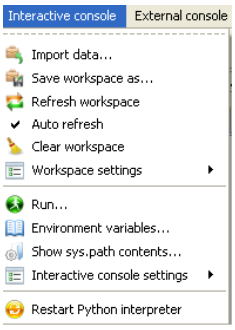


Figure 2.4: Interactive console

.

Como se pode visualizar os diferentes módulos podem ficar automaticamente disponíveis.

Na opção *Interactive Console* na barra de ferramentas do Spyder acedemos às seguintes opções:



2.4 Histórico

Para acedermos ao histórico podemos clicar na opção History log.

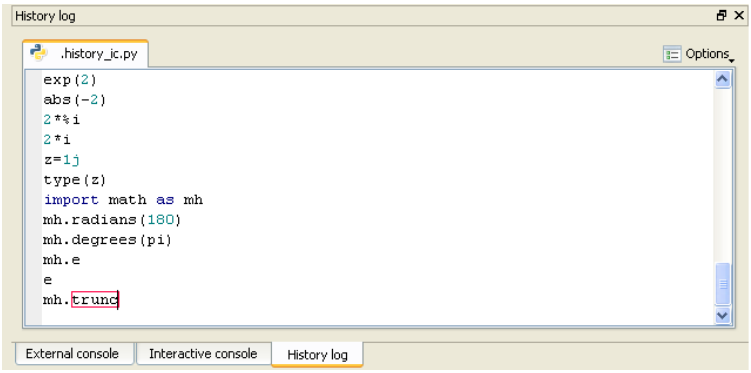


Figure 2.5: History log

2.5 Shell

Do lado esquerdo temos uma *Shell* onde é possuir desenvolver, por exemplo, implementações computacionais ou Classes.

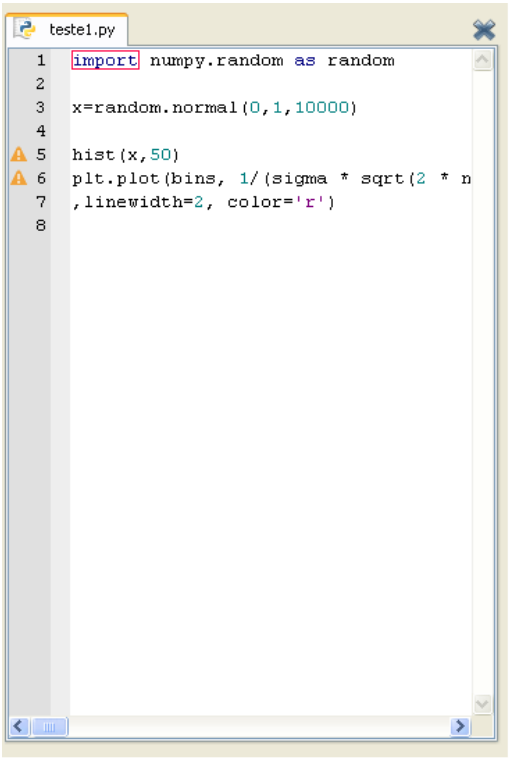


Figure 2.6: Shell no Python (x,y)

Para executarmos uma implementação computacional na Shell, temos duas opções:

1. F9
2. Na opção Source $\Rightarrow$*Run in interactive console*

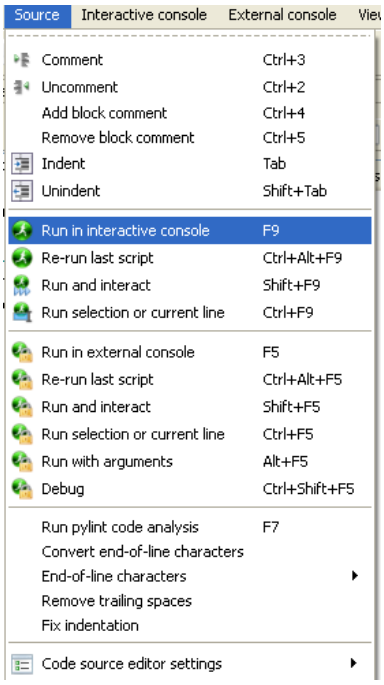
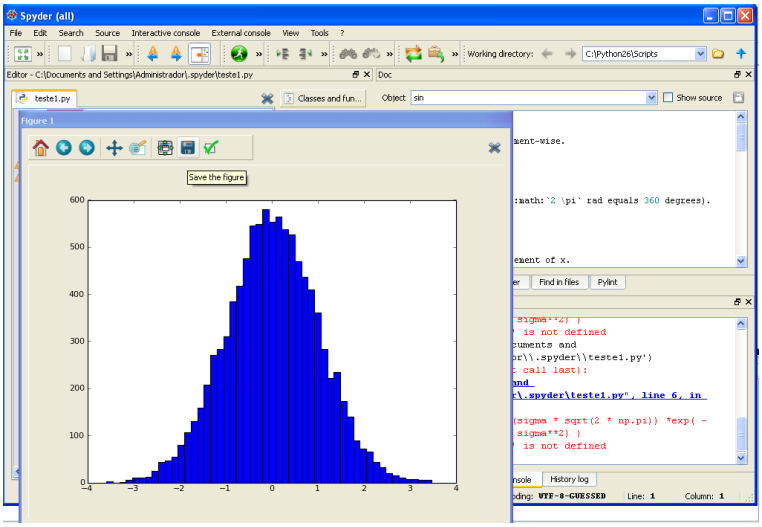


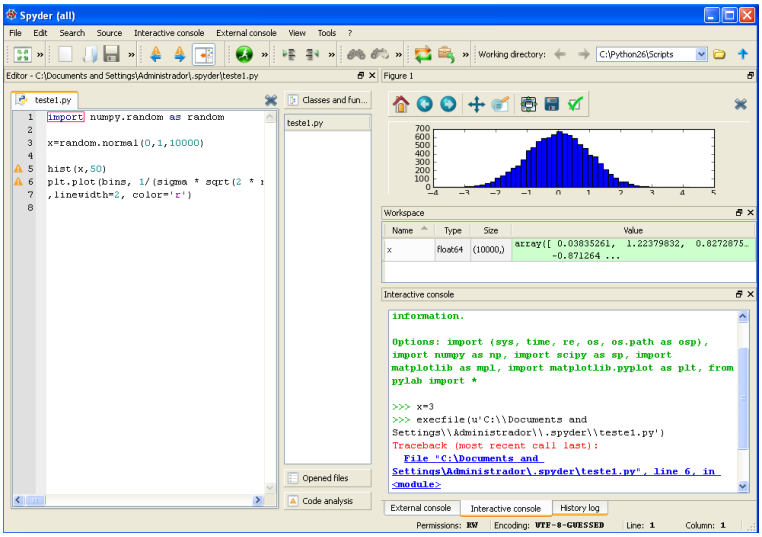
Figure 2.7: Executar um ficheiro com extensão *.py*

Após a execução, deste exemplo, surgirá um gráfico da seguinte forma:



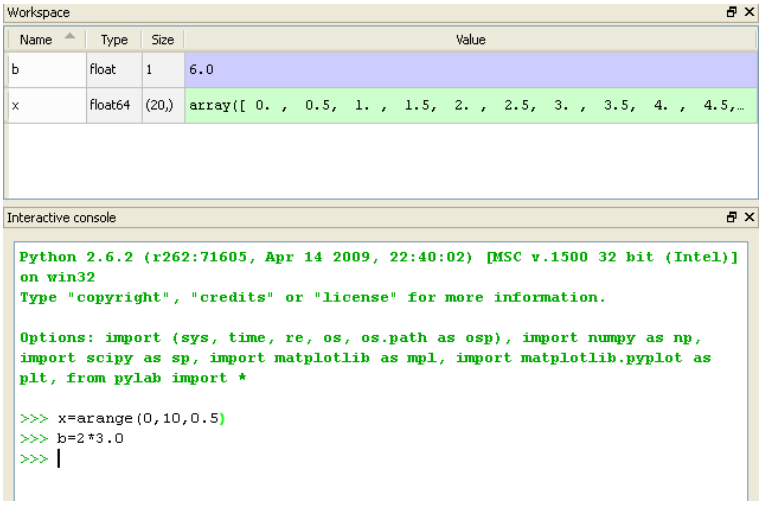
Dispomos de diversas opções quando surge uma janela que contém o gráfico semelhante à anteriormente apresentada. As opções são:

1. Eliminarmos através da opção ***X***
2. Salvamos através do icone 
3. Incorporamos o gráfico no interface do Spyder com podemos ver em seguida:



2.6 Workspace

A opção *Workspace* armazena e indica o tipo de variavél assim como o seu tamanho. Por exemplo:



3 Secção Rápida

A expressão intoduzida no prompt >>> designa-se por célula de entrada *(input)*. O resultado é avaliado e devolvido.

No Python (x,y) a distinção de um input para output faz-se na presença de ">>>", no caso input e na ausência de ">>>" no caso output.

Por exemplo, podemos avaliar a expressão:

>>> 2+5

7

>>>2*3+4;←Ao intoduzirmos ";" o resultado da expressão não é impresso no ecrã.

Se pretendemos aceder às últimas expressões introduzida, avaliadas, carregamos, por exemplo, prosseguidamente na seta do teclado △

3.1 Operações Básicas

Nesta secção, ilustra-se uma breve utilização do Python (x,y). As expressões do Python (x,y) assemalha-se à notação matemática usual. As operações básica estão descritas na tabela 1:

Operador	Descrição
+	Adição
-	Subtracção
*	Multiplicação
/	Divisão inteira
**	Exponenciação

Table 1: Operações aritméticas usuais

Na divisão de inteiros é preciso impregar (.) no divisor ou numerador para que seja impressa a parte decimal:

```
>>>1/6
0
>>>1./6
0.16666666666666666
```

Figure 3.1: Exemplo de divisão de inteiros

3.2 Constantes Matemática

No Python (x,y) as constantes π e número de Euler são referidas como *pi* e *e*, respectivamente:

```
>>>pi
3.1415926535897931
>>>e
2.7182818284590451
```

Figure 3.2: Número de Euler e a constante π

```
>>>log(10)
2.3025850929940459
>>>log10(10)
1
>>>log2(3)
1.5849625007211563
```

Figure 3.3: Operações com funções logaritmicas

No Python (x,y) a função logaritmica está definida para bases com três valores diferentes:

1. log() - logaritmo nepriano
2. log10() - logaritmo de base 10
3. log2()- logaritmo de base 2

Nas funções trignométricas o argumento introduzido terá que ser em radianos.

3.3 Funções matemáticas do módulo *math*

O módulo math possui um conjunto de funções matemáticas que nos permitem resolver diferentes problemas de cálculo. Na tabela 2 mostramos um conjunto de funções matemáticas comuns:

```
>>>sin(pi/4)
0.70710678118654757
>>>sqrt(2)/2
0.70710678118654757
```

Figure 3.4: Operações com funções do módulo *math*

```
>>>a=2 # define uma variável
>>>b=3
>>>c2=3
>>> (a+b)^2
7
```

Figure 3.5: Cálculos Simbólicos

Função	Descrição
sin(argumento)	seno
cos(argumento)	cosseno
tan(argumento)	tangente
sqrt(argumento)	raíz quadrada
exp(argumento)	exponencial
abs(argumentos)	valor absoluto
conj(argumento)	numero complexo conjugado
j	complexo
real(argumento)	parte real
imag(argumento)	unidade imaginária
maximum(argumentos)	máximo de um conjunto de valores
minimun(argumentos)	minimio de um conjunto de valores
reciprocal(x)	$\frac{1}{x}$
factorial(x)	x!
radians(ângulo)	converte ângulos em graus para radianos
degrees(ângulo)	converte ângulos em radianos para graus

Table 2: Funções Matemáticas Comuns

3.4 Variáveis

A atribuição de valores a variáveis é feito da seguinte forma:

3.5 Polinómios

Para criar polinómios de grau n utiliza-se o comando *poly1d ()*. Vejamos, através seguinte exemplo, como podemos:

1. Criar polinómios de grau *n;*
2. Calcular as raízes de um polinómio;
3. Avaliar um polinómio num dado valor;
4. Integrar e derivar um polinómio;

```
>>> coef=([2, -3, 4])
>>> p=poly1d(coef)
>>> print p
```
$2x^2 + 3x - 4$
```
>>>zeros=roots(p)
array([-2.35078106, 0.85078106])
>>> p(5)
```
61
```
>>> Derivada=p.deriv()
>>>print Derivada
```
$4x + 3$
```
>>>Integral=p.integ()
>>>print Integral
```
$0.6667x^3 + 1.5x^2 - 4x$

Figure 3.6: Cálculo Diferencial e Integral

3.6 Funções

A construção de uma função, por exemplo, a função real h definida por $h(x) = (x+2)^2$ é definida pela instrução *def* seguida pelo nome da função e por parêntesis onde se encontra a lista dos parâmetros, separados por virgulas. Por exemplo:

```
# Definindo uma função na Shell
def h(x):
  return (x+1)**2
```
Após, salvarmos a função com o nome h, por exemplo, pressionamos a tecla F9 para executar o ficheiro.

Na consola interativa podemos calcular o valor da função h em 0 e 3, por exemplo:
```
>>>h(0)
1
>>>h(3)
16
```

Figure 3.7: Definição de uma função

4 Numpy

Os recursos matemáticos, apresentados na seccção anterior, fazem parte da distribuição padrão do Python(x,y). O Numpy é um pacote que inclui diversas operações em *vectores* e *matrizes* assim como funções associadas a estes.

4.1 Array

Um *array* define-se como uma lista de valores do mesmo tipo e com um número arbitário de elementos.

```
# criando um array
>>>a=np.array([1, 2, 3, 4, 5])
>>> print a
array([1, 2, 3, 4, 5])
>>>b=np.arange(0.,4.5,.5)
>>> b
array([ 0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. ]) ←O valor final não é impresso não é impresso com o comando arange()
>>># Dados sobre array's
>>>Print " Formato do array:"
>>>b.shape
(9,)
>>>Print "Tipo de dados:"
>>>b.dtype
>>>dtype('float64')
```

Figure 4.1: Objecto: array

⇒Ao contrário das listas, que podem na mesma lista conter strings e valores numéricos, o objecto *array* são homogéneos, ou seja, todos os elementos são do mesmo tipo.

4.2 Matrizes

No **Numpy** podemos definir matrizes e efectuar operações sobre. Por exemplo:

```
>>>Print "Criar uma matriz a partir de uma lista:"
>>>l=[[3,4,5], [6,7,8], [9,0,1]];
>>>x=np.matrix(l);
>>> print transpose(M)
[[3 6 9] [4 7 0] [5 8 1]]
>>> # Criando outra matriz
>>> R=np.matrix([[3,2,1]])
>>> print R*M
[[30 26 32]]
>>>print "Resolvendo um sistema linear:"
Resolvendo um sistema linear:
>>> np.linalg.solve(M,np.array([0,1,2]))
array([ 0.33333333, 1. , -1. ])
```

Figure 4.2: Operações com matrizes

O módulo numpy.linalg também implementa funções de decomposição de matrizes.

```
>>> from numpy import *
>>> A=array([(9,4,2), (5,3,1), (2,0,7)]);
>>> # Decompondo usando QR
>>> Q,R = linalg.qr(A)
>>>Q
>>> array([[-0.85811633, 0.14841033, -0.49153915], [-0.47673129,
-0.58583024, 0.65538554], [-0.19069252, 0.79672913, 0.57346234]]);
>>>R
>>> array([[-10.48808848, -4.86265921, -3.52781158], [ 0. ,
-1.16384941, 5.28809431], [ 0. , 0. , 3.68654364]])
```

Figure 4.3: Módulo: *Linalg*

⇒O **Numpy** serve de base para diversos outros projectos de código aberto nomeadamente o **Matplotlib** e o **Scipy.**

5 Matplotlib

O Matplotlib é biblioteca disponivel com o Python(x,y). Além de produzir gráficos em duas dimensões, também permite criar histogramas, espectro de potência, gráfico de barras, entre outros.

Vejamos alguns exemplos:

Algorithm 1 Algoritmo: Representação do gráfico de uma função cos(x)

```
# exemplo 1
x=arange(0,20,0.1);
y=cos(x)
# Gráfico da função cos(x)
plot(x,y)
#Legenda para o eixo X
xlabel('x')
#Legenda para o eixo Y
ylabel('cos(x)')
# Legenda no topo da figura
title('f(x)=cos(x)')
```

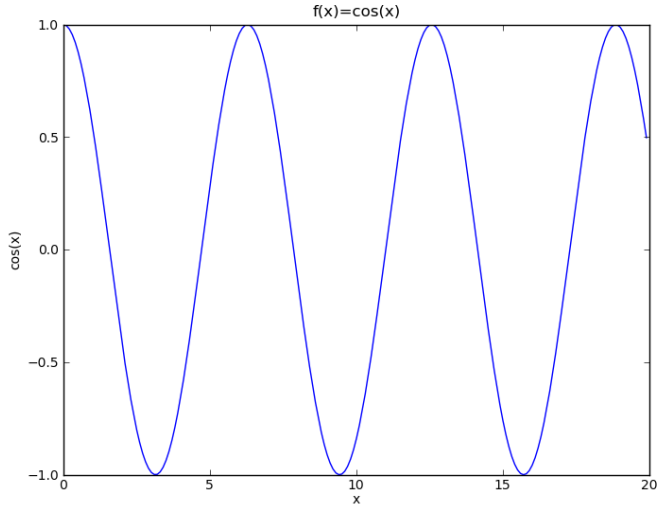


Figure 5.1: Gráfico da função cosseno

Algorithm 2 Representação Gráfica de diferentes tipos de gráficos

```
# exemplo 2
x = arange(0,7,0.01)
y1=np.cos(x)
y2=np.sin(x)
y=y2-y1
# Divide a figura em 2 linhas e uma coluna
subplot(2,1,1)
plot(x, y1, 'b', x, y2, 'g')
subplot(2,1,2)
#Desenha barras horizontais ao longo do gráfico
bar(arange(len(y))+0.5, y, 0.5)
```

Algorithm 3 Coeficiente de correlação de Pearson.

```
#exemplo 3 - Correlação de Pearson entre duas variáveis
# importa o módulo stats do scipy
from scipy import stats
# declaração das duas variáveis
a = [3,4,5,6,7,8];
b = [9,10,11,12,13,14];
c=stats.pearsonr(a,b);
print c
```

O resultado impresso na consola interativa é:
(1.0, 1.4999999999999995e-40)

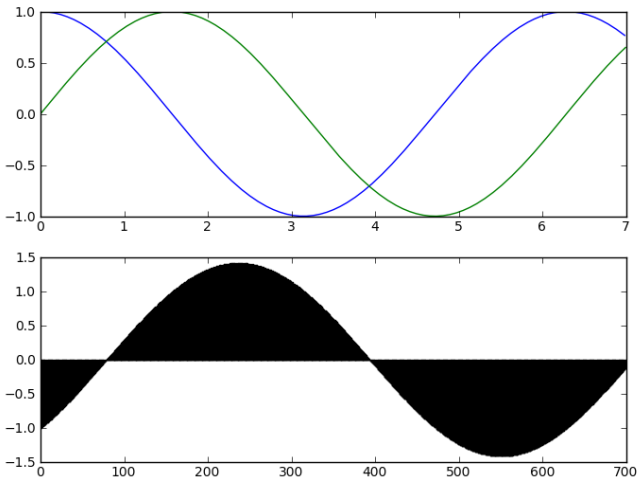


Figure 5.2: Diferentes tipos de gráficos

6 Scipy

O **Scipy** é um módulo que expande o **Numpy.** Seguem-se alguns exemplos de utilização:

Vamos testar a correlação linear entre duas variáveis:

Um dos procedimentos em estatística para a comparação entre duas amostras é o teste T-student.

Algorithm 4 teste T-student

```
from scipy import stats
a = [2,3,4,3,5,6] # amostra a
b = [5,7,8,8,5,6] # amostra b
T=stats.ttest_ind(a,b)
print T
```

O resultado impresso na consola interativa é:
(-3.2391053207156637, 0.0088828669913687541)
→*O primeiro valor que retornou corresponde ao valor T e o segundo corresponde ao valor p .*

7 Conclusões

Conclui-se que o Python (x,y) é uma excelente ferramenta nas ciências exactas revelando-se uma óptima alternativa aos demais softwares comerciais.

References

[1] http://docs.python.org/index.html

[2] http://python.pt/blog/tag/tutoriais/

[3] http://www.dcc.fc.up.pt/~acm/aulas/IP10/#ee

[4] http://www.dcc.fc.up.pt/~acm/aulas/IP10/how.pdf