Introdução ao Numpy, Scipy e Matplotlib
Visualizações com matplotlib
Leitura e escrita de ficheiros ".wav"
Ouvir sinais com PyAudio

Processamento Digital de Sinais

Sumário

1. Python
   a) História e Versões
   b) Instruções Básicas
   c) Tipos
   d) Estruturas de Controle e Funções
2. Numpy
3. Scipy
4. Matplotlib
5. Bibliografia

História da Linguagem

- Python começou a ser desenvolvido na decada de 80.
- Pretendia ser fácil uma linguagem aceite na academia e na industria
- Linguagem Open-Source
- Tornou-se uma das linguagens de programação mais utilizadas (top10)*

*TIOBE programming community index

Python e Versões

- Duas Versões principais:
  - 2.7
  - 3.2
- Vamos usar a 2.7

“PYTHON IS EXECUTABLE PSEUDO-CODE.”
—PYTHON LORE
(OFTEN ATTRIBUTED TO BRUCE ECKEL)

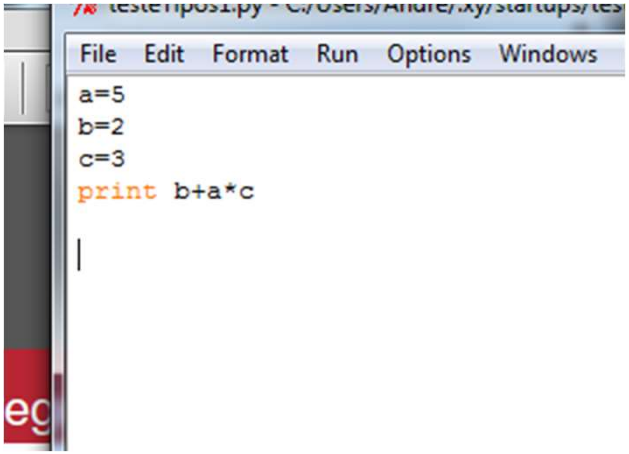
Instruções Básicas

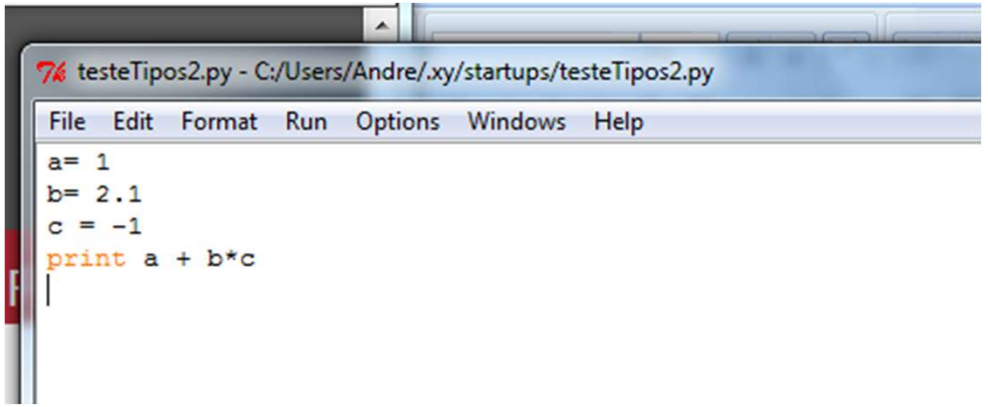
```
a=[1, 3, 5, -1, .3]

soma = 0
n = 0
for v in a:
    soma = soma + v
    n = n + 1

print "Resultado da Soma:", soma, "; media:", soma/n
```

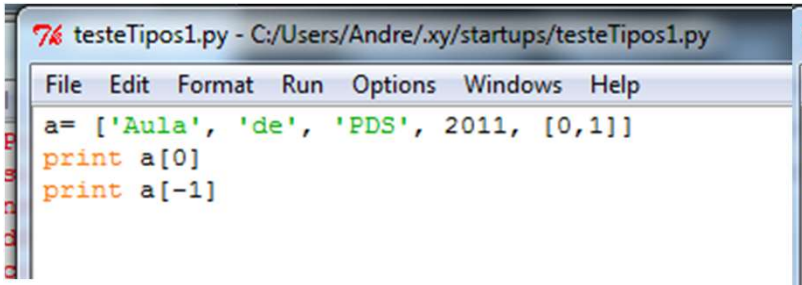
Tipos Básicos

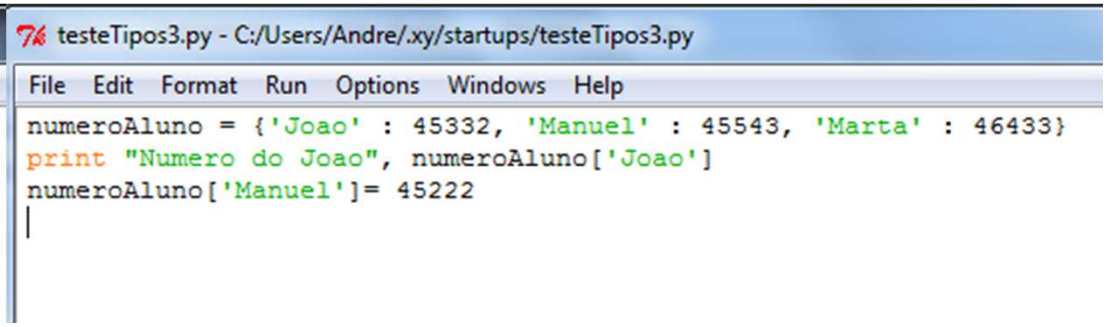
- Números (inteiros e floating point)
- Strings
- Listas e tuplos
- Dicionários

Exemplos Tipos









Números e Operações

- Inteiros e Floating-point
- Operações comuns:
  - x + y, x-y, x*y, x ** y (exponenciação)
  - x/y e x // y (divisão inteira)

Estruturas de Controle

```
vector=[0,1,2,3,4,5]
for v in vector:
    if v > 3:
        print "v>3"
    elif v> 1:
        print "2>v>3"
    else:
        print "v<2"
```

- Condições:
  - x == y
  - x != y (diferente)
  - x < y
  - x < y < z
  - x in lista
  - x not in lista

- Blocos de código definidos pela indentação
- Indentação normalmente dada por 4 espaços

Funções

```
def test(vector):
    counter=0
    for v in vector:
        if v > 3:
            print "v>3"
        elif v> 1:
            print "2>v>3"
            counter = counter+1
        else:
            print "v<2"
    return counter


vector=[0,1,2,3,4,5]
c= test(vector)
print c
```

Numpy is a language extension that defines the numerical array and matrix type and basic operations on them.

NUMPY

Tipo Base

- numpy.array ou numpy.matrix
- Diferenças para os arrays de Python:
  - Menos memória ocupada
  - Mais rápidos

Tipo Base

- Numpy's array class (which is used to implement the matrix class) is implemented with speed in mind, so accessing numpy arrays is faster than accessing Python lists. Further, numpy implements an ***array language***, so that most loops are not needed. For example, Plain Python (and similarly for C, etc.):

```
a = range(10000000)
b = range(10000000)
c = []
for i in range(len(a)):
  c.append(a[i] + b[i])
```

This loop can take 5-10 seconds on a few-GHz processor. With numpy:

```
import numpy as np
a = np.arange(10000000)
b = np.arange(10000000)
c = a + b
```

Importar bibliotecas

- Estruturado em packages que têm suporte em directorias

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

When importing the package, Python searches through the directories on `sys.path` looking for the package subdirectory.

Importar bibliotecas

- Importação de package individual:
  - import sound.effects.echo
  - sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
  - from sound.effects import echo
  - echo.echofilter(input, output, delay=0.7, atten=4)
  - from sound.effects.echo import echofilter
  - echofilter(input, output, delay=0.7, atten=4)
- Todo o package:
  - from sound.effects import *

Importação bibliotecas

- Uso de acronimos

- Convenções

File Edit Format Run Options Windows Help

```
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Propriedades Básicas

File Edit Format Run Options Windows Help

```
import numpy as np
vector = np.array([1,2,3])
matriz = np.array([[1,2,3],[4,5,6],[7,8,9]])
print vector.shape
print vector.size
print matriz.shape
print matriz.size
```

Operações Aritméticas

```
File  Edit  Format  Run  Options  Win
import numpy as np

a = np.array([1,2,3])
b = np.array([3,5,7])
print a
print b
print a+b
print a*b
print a/b
```

```
>>> ========== RESTART =====
>>>
[1 2 3]
[ 3.  5.  7.]
[  4.   7.  10.]
[  3.  10.  21.]
[ 0.33333333  0.4         0.42857143]
>>> ========== RESTART =====
>>>
[1 2 3]
[3 5 7]
[ 4  7 10]
[ 3 10 21]
[0 0 0]
>>>
```

Tipos

and how to modify an array's data-type.

Data type	Description
bool	Boolean (True or False) stored as a byte
int	Platform integer (normally either `int32` or `int64`)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float	Shorthand for `float64`.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex	Shorthand for `complex128`.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Criação de Arrays/ Indexação

```
import numpy as np

a = np.array([[1,2,3],[3,5,7]])
b = np.array([3,5,7])
print a
print b
print a[:,0]
print a[:,-1]
print a[1,1]
print a[1]
print b[2]
```

- A indexação pode ser realizada usando:
  - Listas
  - Slicing (notação :)

Slicing

- Python uses the term *slice* for the same operations as a real world slice. When you want to extract part of a string, or some part of a list, you use a slice. Other languages use the term *substring* for a string slice, and may have no mechanism for extract a list slice.

- Python's slice syntax is easy and elegant:

  slicedString = aString[beginIndex:endIndex]

  slicedList = aList[beginIndex:endIndex]

Based on: http://techearth.net/python/index.php5?title=Python:Basics:Slices

Slicing (sumário)

- General:

a[start:end:step] # start through not past end, by step

*The key point to remember is that the :end value represents the first value that is **not** in the selected slice.*

Default step is 1, so when start:end (step is assumed to be 1)

- Examples

a[start:end] # items start through end-1

a[start:] # items start through the rest of the array

a[:end] # items from the beginning through end-1

a[:] # a copy of the whole array

Baseado em: http://stackoverflow.com/questions/509211/the-python-slice-notation

Scipy is another language extension that uses numpy to do advanced math, signal processing, optimization, statistics and much more.

SCIPY

Overview

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Statistics (scipy.stats)
- Multi-dimensional image processing (scipy.ndimage)
- File IO (scipy.io)
- Weave (scipy.weave)

Scipy.io

```
from scipy.io import wavfile
import matplotlib.pyplot as plt
fs, data = wavfile.read('fala1.wav')
plt.plot(data)
plt.show()
```

SciPy.signal

- bsplines
- filter_design - Filter design.
- info - Convolution:
- ltisys
- setup
- signaltools
- sigtools
- spline
- waveforms
- wavelets

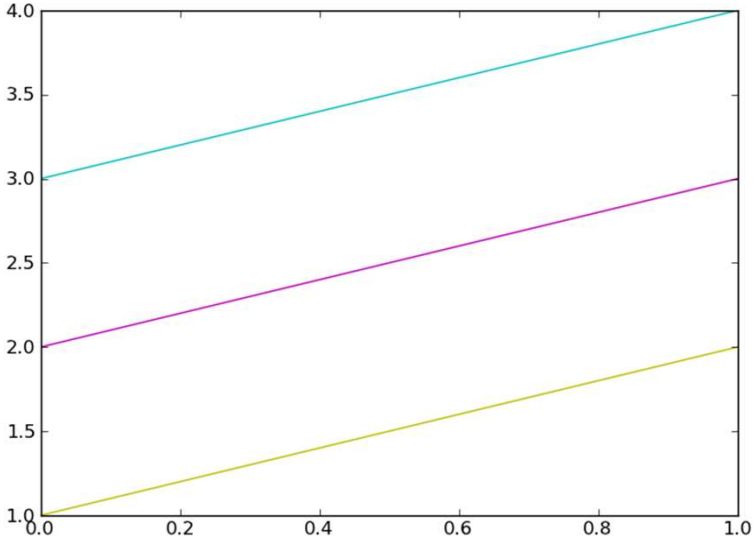
Scipy.fftpack

- fft(x[, n, axis, overwrite_x]) Return discrete Fourier transform of real or complex sequence.
- ifft(x[, n, axis, overwrite_x]) Return discrete inverse Fourier transform of real or complex sequence.


matplotlib

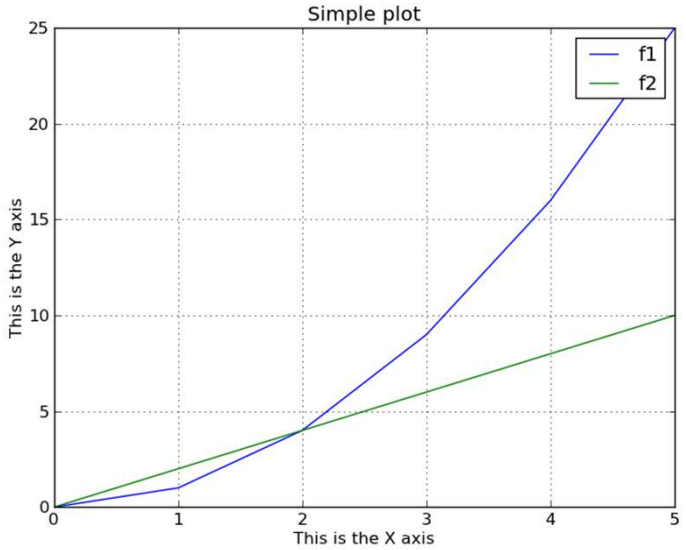
Gráficos (1)

```
import matplotlib.pyplot as plt
import numpy as np
y = np.arange(1, 3)
plt.plot(y, 'y');
plt.plot(y+1, 'm');
plt.plot(y+2, 'c');
plt.show()
```

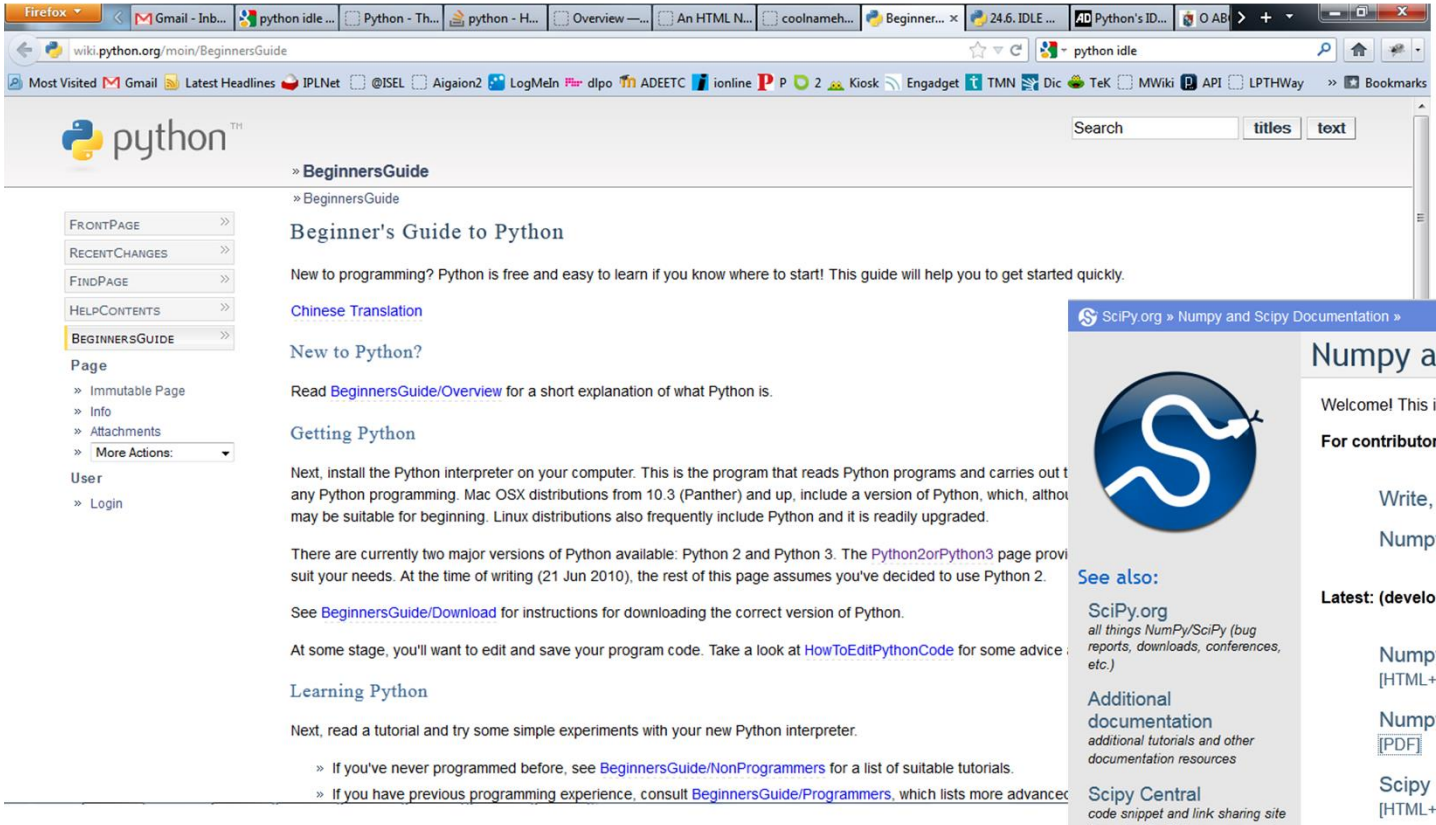


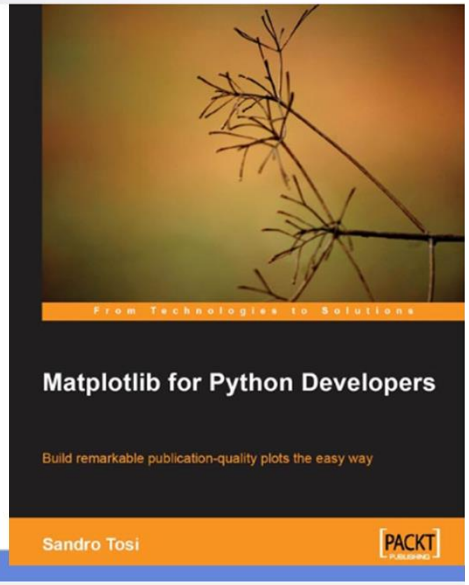
Gráficos (2)

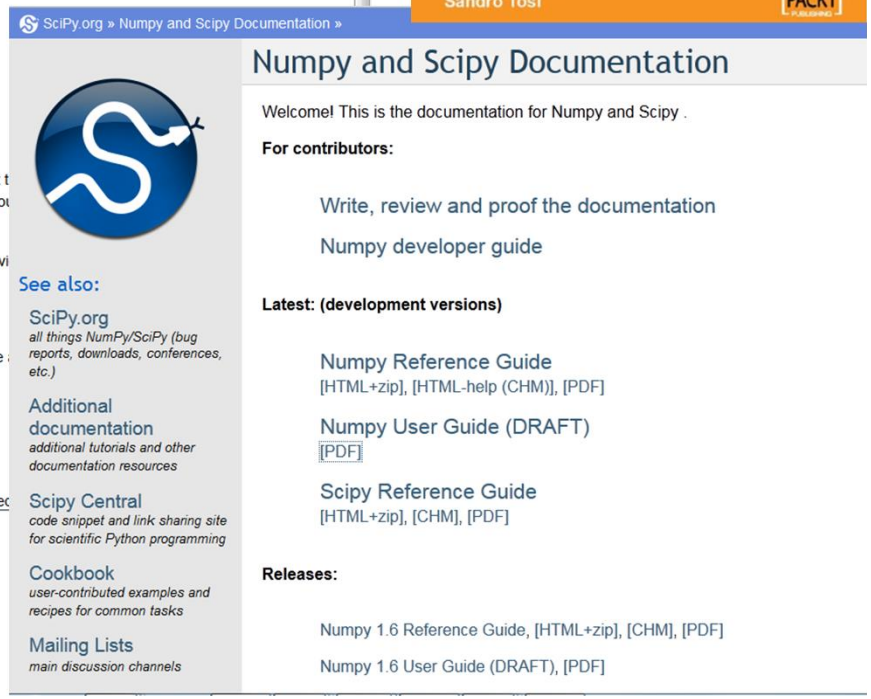
```
import matplotlib.pyplot as plt
x = range(6)
plt.plot(x, [xi**2 for xi in x], label='f1 )
plt.plot(x, [xi*2 for xi in x], label='f2')
plt.grid(True)
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
plt.title('Simple plot')
plt.legend()  ou plt.legend(['f1', 'f2'])
plt.show()
plt.savefig('plot123.png')
plt.xlim() , ou plt.ylim()
```



Documentação







Visualizações com matplotlib

Leitura e escrita de ficheiros “.wav”

Ouvir sinais com PyAudio

matplotlib

matplotlib é um módulo de Python para visualizações de dados.
Matplotlib permite fazer gráficos (2D e 3D) de funções, dados temporais espaciais (ex: ficheiros de ´audio e imagem), histogramas, gráficos de contornos, barras, pontos individuais, entre muitas outras funcionalidades.

Documentação:

- http://matplotlib.org/
- matplotlib User Guide (release 1.4.2)
- Anatomy of Matplotlib por WeatherGod do GitHub.

Leitura obrigatória para matplotlib “newbies”:

- Capítulo 3 do matplotlib User Guide (release 1.4.2)
  “Pyplot Tutorial” pp-23–36.

Leitura adicional:

- André Lourenço, Introdução ao Python e às bibliotecas Numpy, Scipy e Matplotlib, Slides PDS, ISEL-DEETC, 2012

- Introduction to NumPy por M. Scott Shell (ler, 24 p. total)

Gráficos 2D

Visualização de sinais

Para sinais contínuos é necessário criar a ilusão de continuidade

- Selecionar um intervalo de visualização
  (ex: para x(t) escolher t entre [tmin; tmax])

- Criar um array com valores de t no intervalo escolhido, com incrementos idênticos e suficientemente pequenos para obter uma boa definição.

Comandos: plot() e stem()

comandos relacionados: figure(), show(), subplot(), axis(), grid(), xticks(),xlegend(),savefig()

Para detalhes de configuração ler:

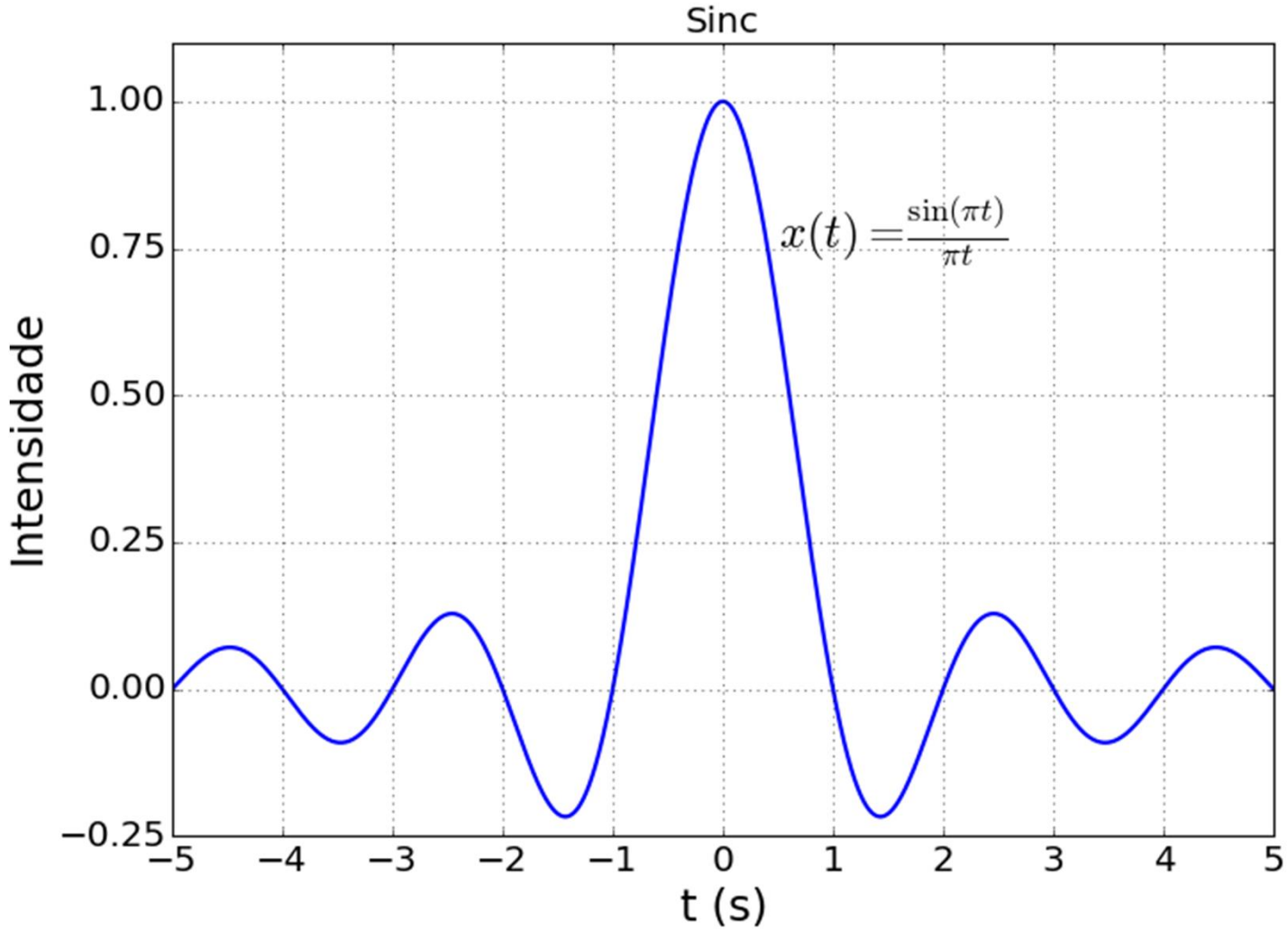
matplotlib User Guide (1.4.2) Chap. 11, pp-372-381

LAB1_1_16_17.py Faz gráfico de: $x(t) = \mathrm{sinc}(t) = \frac{\sin(\pi t)}{\pi t}$

```
@author: Isabel
"""
# -*- coding: latin-1 -*-
# 1a linha para poder usar acentos (utf-8 tb dá)
#PSD - script para desenhar a funcção x(t)=sinc(t)
import numpy as np
import matplotlib.pyplot as plt
#1000pts equi-espaçados entre [-5,5]
tmin=-5.0
tmax=5.0
t=np.arange(tmin,tmax,1e-3) #arange(início,fim,intervalo)
#pode-se igualmente usar linspace(início,fim,no de pts)
x=np.sinc(t)
#x1=np.sin(np.pi*t)/(np.pi*t) #x(t)=sinc(t)
plt.close('all')
plt.figure(facecolor='w',figsize=(10,7))
plt.plot(t,x,lw=2)
#plt.plot(t,x1,'r',lw=2)
plt.axis([tmin,tmax,-0.25,1.1]) #eixos
plt.grid() #grelha
plt.xticks(np.arange(-5,6),fontsize=18) #intervalos no x = 1
plt.yticks(np.arange(-.25,1.25,.25),fontsize=18) #intervalos no y = 1/4
plt.xlabel('t (s)',fontsize=22),
plt.ylabel('Intensidade',fontsize=22)
plt.title('Sinc',fontsize=18)
plt.text(.5,.75,r'$x(t)=\frac{\sin(\pi t)}{\pi t}$',fontsize=24)
plt.savefig('LAB1_1.png') #guardar em ficheiro ".png" na directoria corrente
plt.show()
```

Gráficos 2D

LAB1_1_16_17 `.py`: visualizar $x(t) = \frac{\sin(\pi t)}{\pi t}$

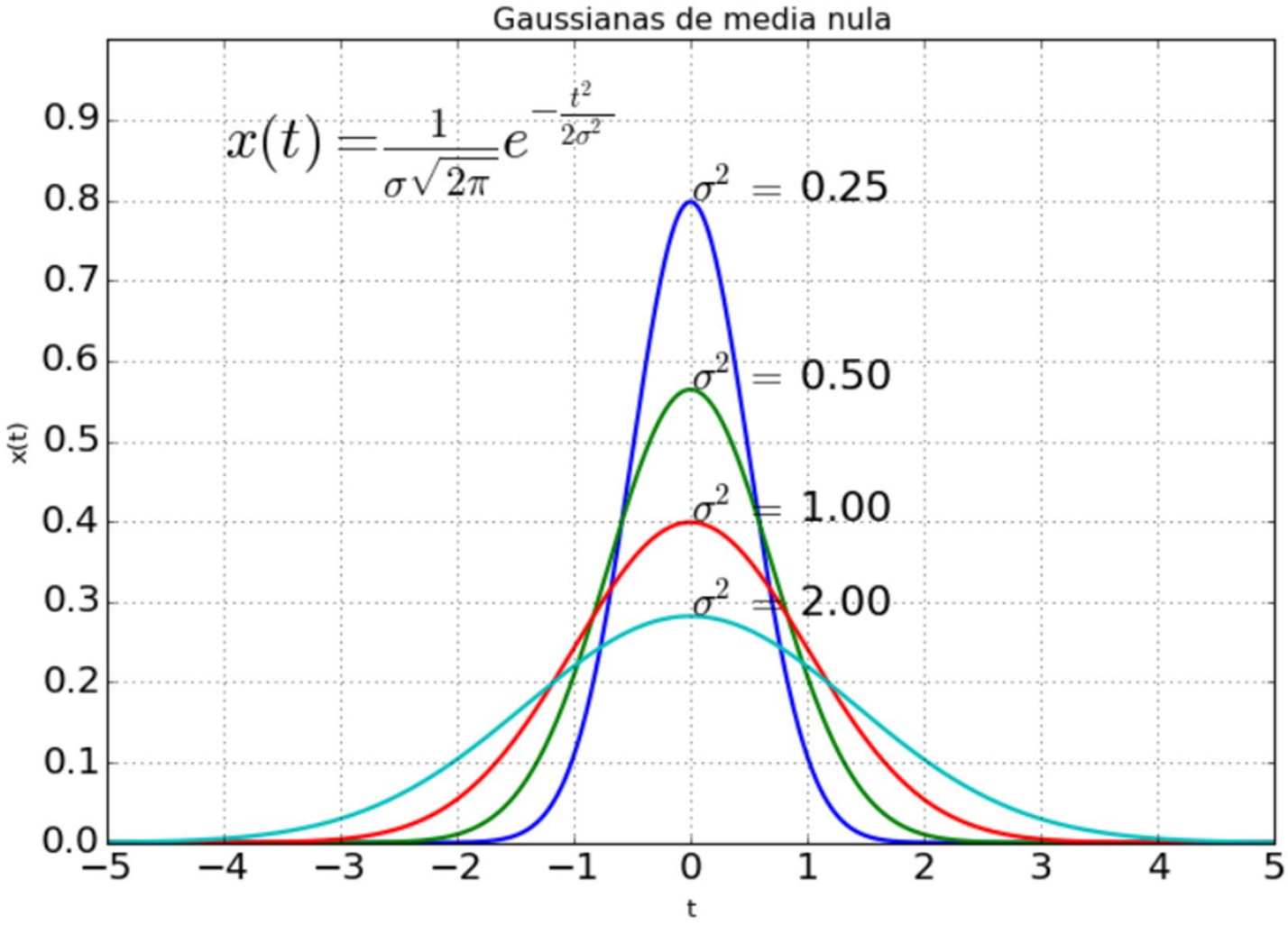


LAB1_2_16_17

```
 9 import numpy as np
10 from matplotlib import pyplot as plt
11 t=np.linspace(-5,5,1000)
12 sigma2=np.array([1./4,1./2,1.,2.,])
13 plt.close('all')
14 plt.figure(facecolor='w',figsize=(10,7))
15 for s in sigma2:
16     x=1/np.sqrt(2*np.pi*s)*np.exp(-1./(2*s)*t**2)
17     plt.plot(t,x,lw=2)
18     strTmp=r'$\sigma^2=$ %0.2f'%s
19     plt.text(0,x[500],strTmp,fontsize=20)
20 plt.axis([-5,5,0,1])
21 plt.xlabel('t')
22 plt.ylabel('x(t)')
23 plt.xticks(np.arange(-5,6),fontsize=18) #intervalos no t = 1
24 plt.yticks(np.arange(0,1,.1),fontsize=18) #intervalos no x = 0.1
25 plt.title('Gaussianas de media nula')
26 plt.text(-4.0,0.85,r'$x(t)=\frac{1}{\sigma \sqrt{2 \pi}} e^{-
   \frac{t^2}{2 \sigma^2}}$', fontsize=28)
27 plt.grid()
28 plt.savefig('LAB1_2.png') #guardar em ficheiro ".png"
   plt.show()
```

Gráficos 2D

LAB1_2_16_17 `.py`: visualizar $x(t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}t^2\right)$



Gráficos 2D

Sinais Discretos

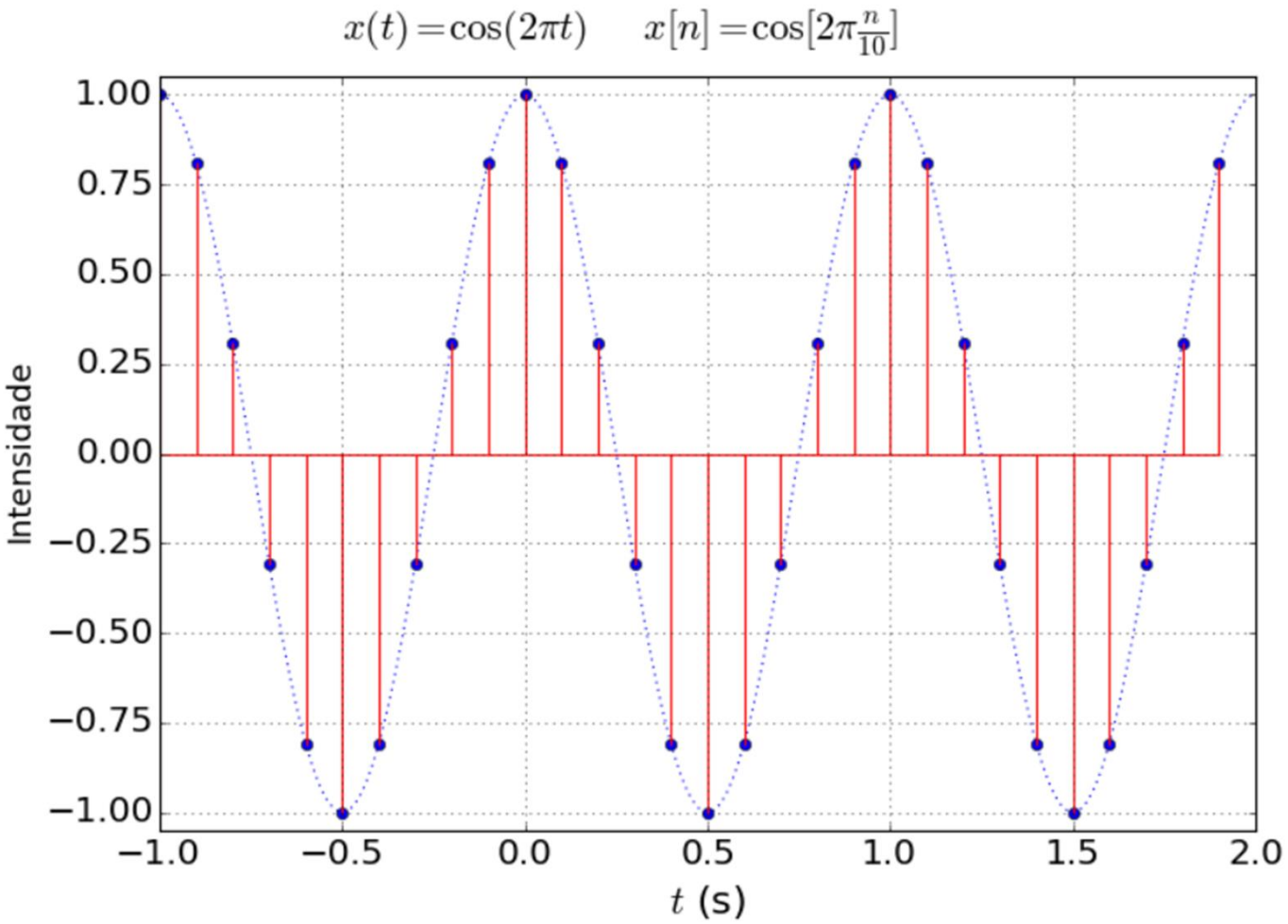
LAB1_3_16_17 .py: visualizar $x(t) = \cos(2\pi t)$ amostrado a $F_s = 10\text{Hz}$

```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  from matplotlib import pyplot as plt
4  t=np.linspace(-1,2,1000) #1000pts equi-espaçados entre [-1,2]
5  fc=1
6  x_t=np.cos(2*np.pi*fc*t) #fc Hz
7  N=10
8  n=np.arange(-N,2*N)
9  x_n=np.cos(2*np.pi*fc*n/N)
10 plt.close('all')
11 plt.figure(facecolor='w',figsize=(10,7))
12 plt.plot(t,x_t,':b',lw=1)
13 plt.stem(n*1./N,x_n,'r')
14 plt.axis([-1,2,-1.05,1.05])  #eixos
15 plt.grid('on')    #grelha
16 plt.xlabel(r'$t$ (s)',fontsize=20)
17 plt.ylabel('Intensidade',fontsize=16)
18 plt.text(-0.5,1.15,r'$x(t)=\cos(2\pi t)\quadx[n]=\cos[2\pi\frac{n}
   {10}]$',fontsize=20)
19 plt.xticks(np.arange(-1,2.1,.5),fontsize=18)
20 plt.yticks(np.arange(-1,1.1,.25),fontsize=18)
   plt.show()
```

Gráficos 2D

Sinais Discretos

LAB1_3_16_17 .py: visualizar $x(t) = \cos(2\pi t)$ amostrado a $F_s = 10\text{Hz}$

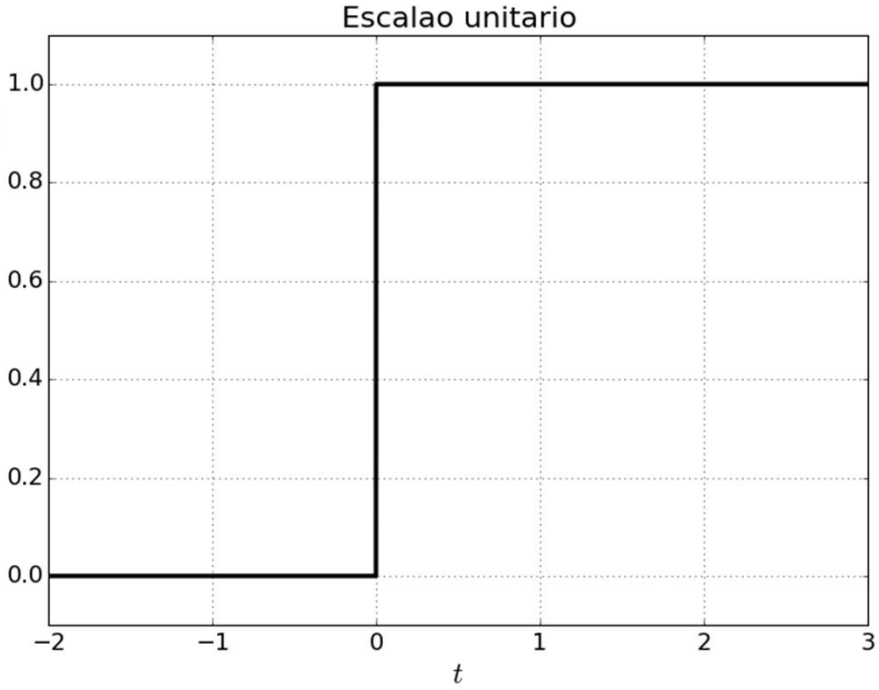


Gráficos 2D

Função escalão

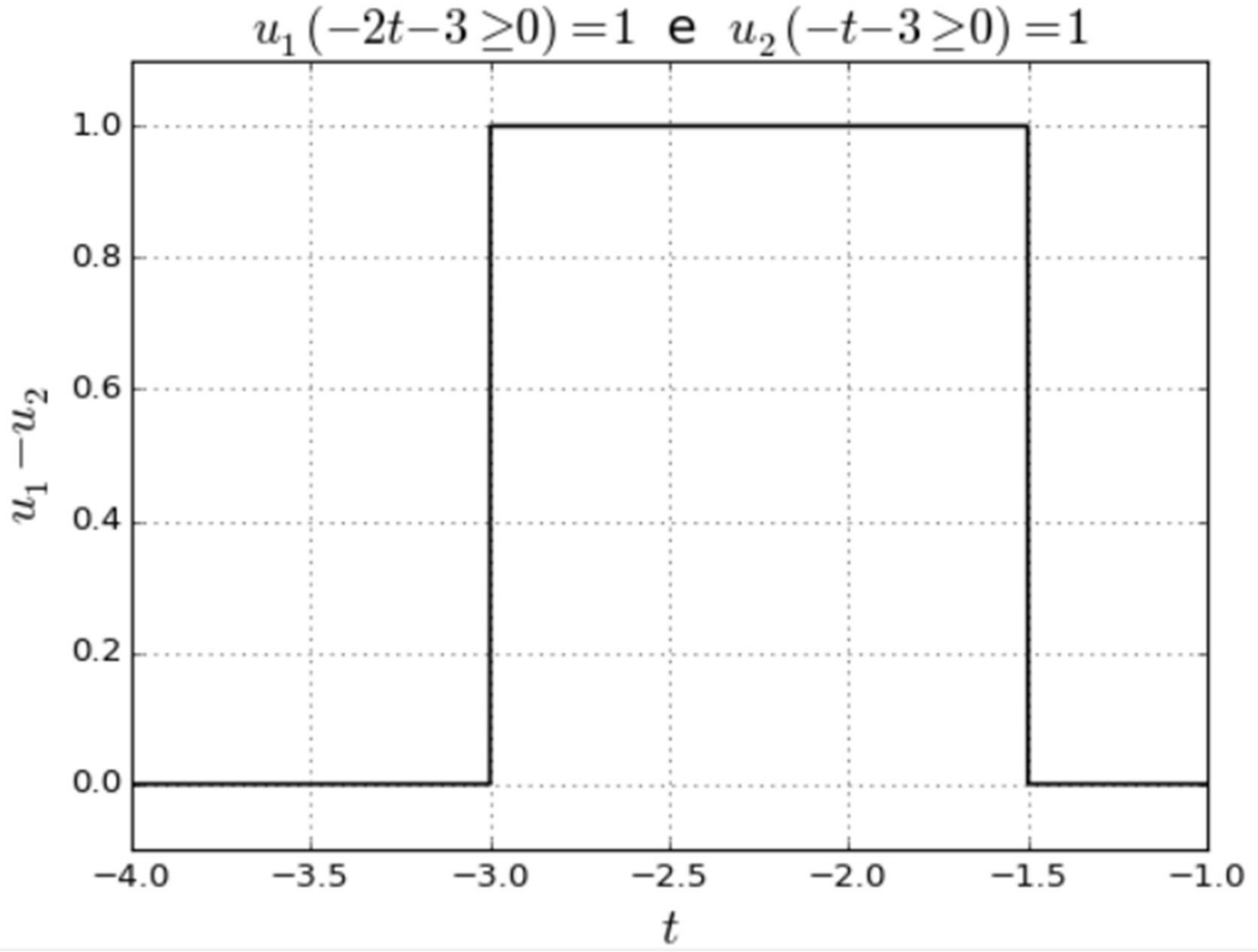
$$u[n] = \begin{cases} 0 \ , & n < 0 \\ 1 \ , & n \geq 0 \end{cases}$$

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 09 16:23:56 2017

@author: Isabel Rodrigues
"""

import matplotlib.pyplot as plt
import numpy as np
# x(t)=u(-3t-6)-u(-2t-8)
tI=[-2.0,3.0]
t=np.linspace(tI[0],tI[1],1e4)

u=np.zeros(t.shape)

u[(t>=0)]=1

plt.close('all')
plt.figure(facecolor='w',figsize=(10,7))
plt.plot(t,u,'k',linewidth=3)
plt.axis([tI[0],tI[1],-0.1,1.1])
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.title('Escalao unitario',fontsize=20)
plt.xlabel(r'$t$',fontsize=20)
plt.grid()
```



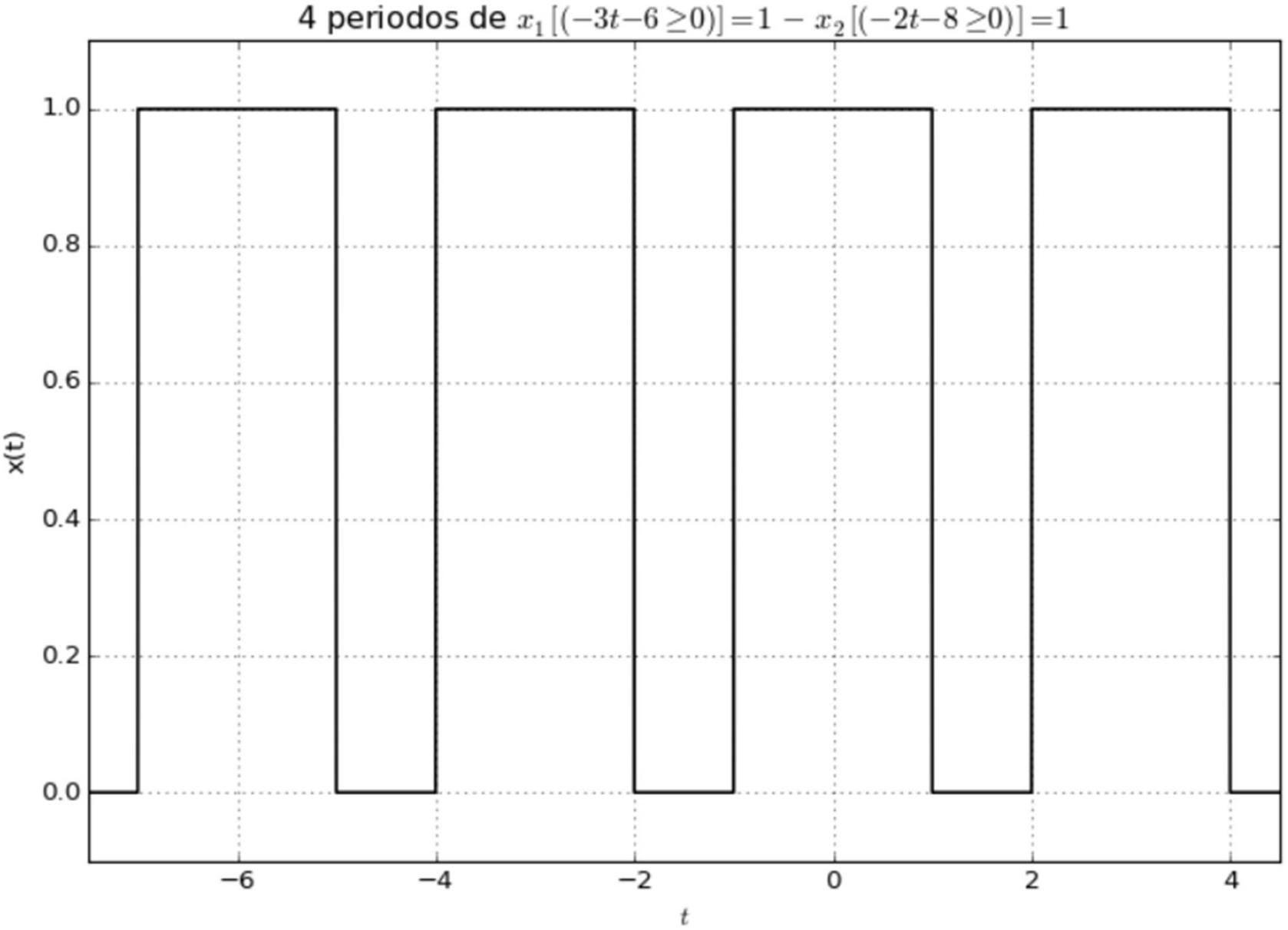
Diferença de escalões - onda quadrada

```
 8 # -*- coding: latin-1 -*-
 9 import matplotlib.pyplot as plt
10 import numpy as np
11
12 tI=[-4,-1]
13 t=np.linspace(tI[0],tI[1],1e4)
14 u1=np.zeros(t.shape)
15 u2=np.zeros(t.shape)
16 u1[(-2*t-3>=0)]=1
17 u2[(-t-3>=0)]=1.
18 u=u1-u2
19
20 plt.close('all')
21 plt.figure(facecolor='w',figsize=(7,5))
22 plt.plot(t,u,'k',linewidth=1.5)
23 plt.axis([tI[0],tI[1],-0.1,1.1])
24 plt.xlabel(r'$t$',fontsize=18)
25 plt.ylabel(r'$u_1-u_2$',fontsize=18)
26 plt.title('$u_1(-2t-3 \geq 0)=1$ e $u_2(-t-3 \geq 0)=1$',
   fontsize=18)
27 plt.grid()
   plt.show()
```


$u_1(-2t-3 \geq 0) = 1$ e $u_2(-t-3 \geq 0) = 1$
1.0
0.8
0.6
0.4
0.2
0.0
$u_1 - u_2$
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
t

Ex: 4 períodos do sinal $x_1[(-3t-6 \geq 0)] = 1 - x_2[(-2t-8 \geq 0)] = 1$

```
 9 import matplotlib.pyplot as plt
10 import numpy as np
11 tI=[-4.5,-1.5]
12 tt=np.linspace(tI[0],tI[1],1e3)
13 x1=np.zeros(tt.shape)
14 x2=np.zeros(tt.shape)
15 x1[(-3*tt-6>=0)]=1.
16 x2[(-2*tt-8>=0)]=1.
17 x=x1-x2
18
19 ttI=[-7.5, 4.5]
20 t=np.linspace(ttI[0],ttI[1],4e3)
21 x=np.concatenate((x,x,x,x),axis=0)
22
23 plt.close('all')
24 plt.figure(facecolor='w',figsize=(7,3.5))
25 plt.plot(t,x,'k',linewidth=1.5)
26 plt.axis([ttI[0],ttI[1],-0.1,1.1])
27 plt.xlabel('$t$'), plt.ylabel('x(t)')
28 plt.title('4 periodos de $x_1[(-3t-6 \geq 0)]=1$ $-$ $x_2[(-2t-8
   \geq 0)]=1$')
   plt.grid()
```


4 periodos de $x_1[(-3t-6 \geq 0)] = 1 - x_2[(-2t-8 \geq 0)] = 1$
1.0
0.8
0.6
0.4
0.2
0.0
x(t)
−6
−4
−2
0
2
4
t

Leitura e Escrita de Ficheiros `.wav`

Fazer `import SciPy.io.wavfile as wav`
Usar comandos: `wav.read()` e `wav.write()`

Pontos a ter em atenção

- São somente considerados ficheiros `.wav`, mono, PCM a 16 bits por amostras
- De notar que existem também outros formatos (ex: stereo, diferentes nº bits/amostra)
- O comando de leitura retorna `int16` (inteiros de 16 bits com valores entre $[-2^{15}, +2^{15}-1]$). Necessário converter para `float` para evitar problemas.
- Normalizar o sinal pela sua amplitude máxima (em termos absolutos) Ou dividir por 2^{15} para preservar a amplitude do sinal `.wav`

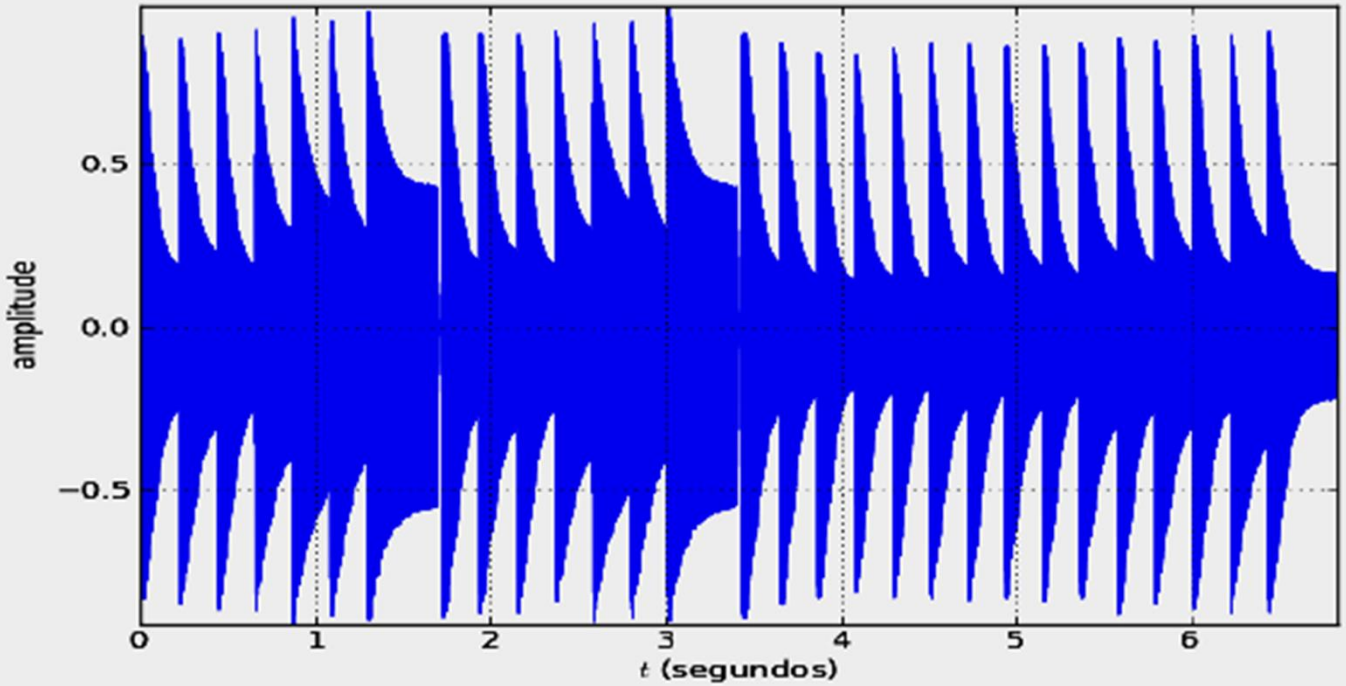
Leitura de Ficheiros .wav

LAB1_4_16_17 .py: leitura e visualização do ficheiro pipocas11k.wav

```
# -*- coding: latin-1 -*-
import numpy as np
import scipy.io.wavfile as wav #módulo de leitura/escrita de .wav
from matplotlib import pyplot as plt
#PDS-ler ficheiro wav (necessário estar na mesma dir. que código)
Fs,xInt=wav.read('pipocas11k.wav')
#x em int16 converter para float
x=np.array(xInt,'float32')
#normalizar amplitude
x=x/2.0**15 # ou por "np.abs(x).max()"
t=np.arange(0,len(x))/np.float(Fs)
plt.plot(t,x,'b')
plt.grid('on')     #grelha
plt.axis('tight')
plt.xlabel(r'$t$ (segundos)')
plt.ylabel('amplitude')
plt.savefig('../figs/L0PDS004.png') #guardar em ficheiro ".png"
plt.show()                          #(na directoria "../figs/")
```

Leitura de Ficheiros `.wav`

LAB1_4_16_17 `.py`: leitura e visualização do ficheiro `pipocas11k.wav`



Escrita de Ficheiros .wav

LAB1_5_16_17 .py:

multiplicar sinal de pipocas11k.wav por um co-seno a 3 kHz e gravar

```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  import scipy.io.wavfile as wav #módulo de leitura/escrita de .wav
4  #PDS- escrever ficheiro wav
5  Fs,xInt=wav.read('pipocas11k.wav')
6  #x em int16 converter para float
7  x=np.array(xInt,'float32')
8  #normalizar amplitude
9  x=x/2.0**15 # ou por "np.abs(x).max()"
10 t=np.arange(0,len(x))/np.float(Fs)
11 #gerar sinusóide de 3kHz
12 fc=3000.0
13 s=np.cos(2.0*np.pi*fc*t)
14 #multiplicar s com x
15 y=x*s
16 #reconverter para int16
17 y=np.int16(y*2.0**15)
18 #guardar no ficheiro 'pipocasCosseno.wav'
19 wav.write('pipocasCosseno.wav',Fs,y)
```

Ouvir Sinais

- Necessário ter instalado o módulo `PyAudio`
  Para detalhes de instalação, consultar página Web:
  http://people.csail.mit.edu/hubert/pyaudio/
- Para ouvir usar programa `soundPlay.py`

LAB1_6_16_17 .py

```
1  # -*- coding: latin-1 -*-
2  import numpy as np
3  import scipy.io.wavfile as wav
4  from soundPlay  import soundPlay #módulo para aceder
5  #PDS- ouvir sinais                #ao porto de áudio
6  Fs,xInt=wav.read('pipocas11k.wav')
7  x=np.array(xInt,'float32')/2.0**15
8  #array com instantes de tempo
9  t=np.arange(0,len(x))/np.float(Fs)
10 #gerar sinusóide de 3kHz e multiplicar por x
11 y=x*np.cos(2.0*np.pi*3000.*t)
12 #ouvir x
13 soundPlay(x,Fs)
14 #ouvir y
15 soundPlay(y,Fs)
16 #outros ex: soundPlay(x,Fs/2) ou soundPlay(x,Fs*2)
```

Exercícios em Python

1. Comando `plot()`
   Represente os seguintes sinais:
   - $x(t) = \cos\left(2\pi t + \frac{\pi}{4}\right) + \frac{1}{5}\sin\left(10\pi t - \frac{\pi}{3}\right)$ para $t \in [0, 2]$
   - $x(t) = \cos\left(2\pi t^2\right) + \frac{1}{3}\sin\left(25\pi t\right)$ para $t \in [-2, 2]$

2. Comando `stem()`
   Represente os seguintes sinais:
   - $x[n] = \begin{cases} \exp\left(-\frac{n}{5}\right) & , n \geq 0 \\ 0 & , n < 0 \end{cases}$ para $n = -2, -1, \ldots, 10$
   - Considere o sinal $u[n]$ (função escalão[1]). Represente os seguintes sinais (use o comando `subplot()`):

★ $x_1[n] = u[-n+3]$
para $n = -2, \ldots, 8$

★ $x_2[n] = u[n+3] - u[n-3]$
para $n = -5, \ldots, 5$

★ $x_3[n] = u[n+3] + u[-n-1]$
para $n = -5, \ldots, 5$

★ $x_4[n] = \cos\left(2\pi\frac{n}{10}\right)(u[n] - u(n-20))$
para $n = -5, \ldots, 25$

[1] $u[n] = 0$, para $n < 0$ e $u[n] = 1$ para $n \geq 0$

Exercícios em Python

3. Ficheiros .wav e audição de sinais
   - Leia o sinal do ficheiro `pipocas11k.wav`. Guarde uma em cada duas amostras deste sinal e oiça (tome em atenção de reduzir a *Fs* para metade). Repita este processo guardando uma em cada três, e uma em cada quatro amostras. Oiça os sinais resultantes.
   - Gere as seguinte sinusóides com duração de 3 segundos a um ritmo de 1000 amostras por segundo ($F_s = 10$ kHz)
     - ★ $x_1(t) = cos(2\pi 440t)$
     - ★ $x_2(t) = cos(2\pi 880t)$
     - ★ $x_3(t) = cos(2\pi 1760t)$
     - ★ $x_4(t) = cos(2\pi 3520t)$
   - Oiça os sinais gerados na alínea anterior, e oiça o sinal resultante da seguinte soma pesada dos mesmos:
     $y(t) = 0.5x_1(t) + 0.75x_2(t) + 0.15x_3(t) + 0.05x_4(t)$
   - Oiça os seguinte sinal (também com $F_s = 10$ kHz).
     $x(t) = \cos(2\pi 438t) + \cos(2\pi 442t)$