UC Berkeley Extension

Python for Data Analysis and Scientific Computing

X433.3 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

Course Content Outline

- **NumPy 2/3**
- Array operations
- Reductions
- Broadcasting
- Array: shaping, reshaping, flattening, resizing, dimension changing
- Data sorting
- Good coding practices

- **NumPy 3/3**
- Type casting
- Masking data
- Organizing arrays
- Loading data files
- Dealing with polynomials

Midterm, Project proposal due

- **Scipy 1/2**
- What is Scipy?
- Working with files
- Algebraic operations
- The Fast Fourier Transform
- Signal Processing

- **Scipy 2/2**
- Interpolation
- Statistics
- Optimization
- Multithreading and Multiprocessing

- **Project**
- Project Presentation

Final Project

Midterm – discussion

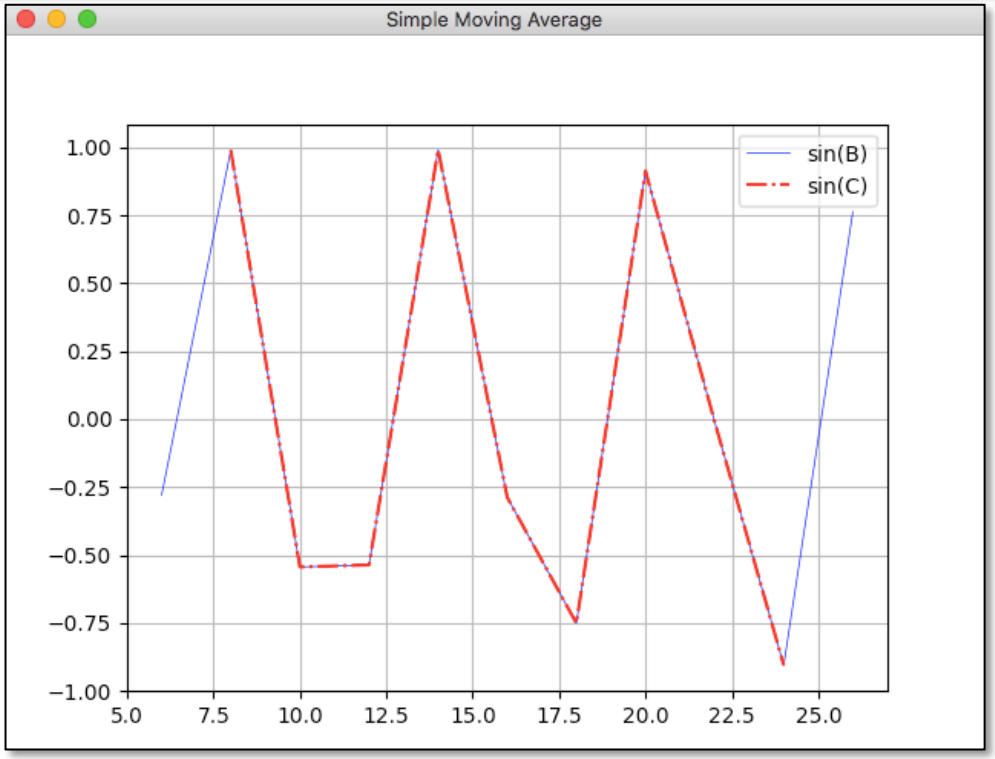
```
7  ## Import:
8  import pylab as plb
9
10 # An array A with 12 elements beginning from number 5, containing consecutive odd numbers:
11 A = plb.arange(5,28,2)
12
13 ## Compute the Simple Moving Average (SMA) of A, with an adjustable window using
14 # a function. The function calculating the SMA of the array must take two input arguments:
15 # array A and a window width with a default value = 2:
16 def sma(B, x = 2):
17     D = plb.zeros(len(B)-(x-1)); E = plb.zeros(len(B)-(x))
18     for index, i in enumerate(plb.arange(len(B)-x+1)):
19         D[int(i)] = sum(B[int(i):int(i)+x])/x
20         if i > 0:
21             E[int(i-1)] = round(cma(D[0:i+1]),2) # Call the CMA with min 2 numbers
22     print('## For window width = %s:' %x)
23     print_fun(D)
24     print('   The CMA of this SMA is: ', E)
25     print('')
26     return D, x
27
28 ## After every SMA is collected (as array) calculate its current cumulative moving average (CMA):
29 def cma(q):
30     E = plb.mean(q)
31     return E
```

Midterm – discussion

```
## Print the results using a short description of the data:
#   Moving average result is:
def print_fun(C):
    print('   The SMA result is: ', C)

## Plotting:
#   The sin functions for arrays B and C, specifying different: color, linewidth, linestyle,
#   label, and figure name passed as string to the plotting function. Show the: grid, legend:
def plot_fun(string):
    plb.figure(string)
    plb.plot(B, plb.sin(B), color="blue",  linewidth=0.5, linestyle="-", label="sin(B)")
    plb.plot(C, plb.sin(C), color="red",  linewidth=1.5, linestyle="-.", label="sin(C)")
    plb.legend(loc='upper right'), plb.grid(True)
    plb.pause(1)

## User input:
X = int(input('Please specify your window width: '))
print('')
print('## The original array is: ', A)
print('')

## Make two function calls to SMA:
#   One, with the array only. Save the result to B:
B, y = sma(A)
#   The other, specifying a window width = 4. Save the result to C:
C, x = sma(A, X)
plot_fun('Simple Moving Average')
```

Midterm – discussion

```
Please specify your window width: 4

## The original array is:  [ 5  7  9 11 13 15 17 19 21 23 25 27]

## For window width = 2:
   The SMA result is:  [ 6.  8. 10. 12. 14. 16. 18. 20. 22. 24. 26.]
   The CMA of this SMA is:  [ 7.  8.  9. 10. 11. 12. 13. 14. 15. 16.]

## For window width = 4:
   The SMA result is:  [ 8. 10. 12. 14. 16. 18. 20. 22. 24.]
   The CMA of this SMA is:  [ 9. 10. 11. 12. 13. 14. 15. 16.]
```



Midterm – discussion

```
Please specify your window width: 4

## The original array is:  [ 5  7  9 11 13 15 17 19 21 23 25 27]

[6. 8.]
[ 6.  8. 10.]
[ 6.  8. 10. 12.]
[ 6.  8. 10. 12. 14.]
[ 6.  8. 10. 12. 14. 16.]
[ 6.  8. 10. 12. 14. 16. 18.]
[ 6.  8. 10. 12. 14. 16. 18. 20.]
[ 6.  8. 10. 12. 14. 16. 18. 20. 22.]
[ 6.  8. 10. 12. 14. 16. 18. 20. 22. 24.]
[ 6.  8. 10. 12. 14. 16. 18. 20. 22. 24. 26.]
## For window width = 2:
   The SMA result is:  [ 6.  8. 10. 12. 14. 16. 18. 20. 22. 24. 26.]
   The CMA of this SMA is:  [ 7.  8.  9. 10. 11. 12. 13. 14. 15. 16.]

[ 8. 10.]
[ 8. 10. 12.]
[ 8. 10. 12. 14.]
[ 8. 10. 12. 14. 16.]
[ 8. 10. 12. 14. 16. 18.]
[ 8. 10. 12. 14. 16. 18. 20.]
[ 8. 10. 12. 14. 16. 18. 20. 22.]
[ 8. 10. 12. 14. 16. 18. 20. 22. 24.]
## For window width = 4:
   The SMA result is:  [ 8. 10. 12. 14. 16. 18. 20. 22. 24.]
   The CMA of this SMA is:  [ 9. 10. 11. 12. 13. 14. 15. 16.]
```

CMA of the current SMA

CMA of the current SMA

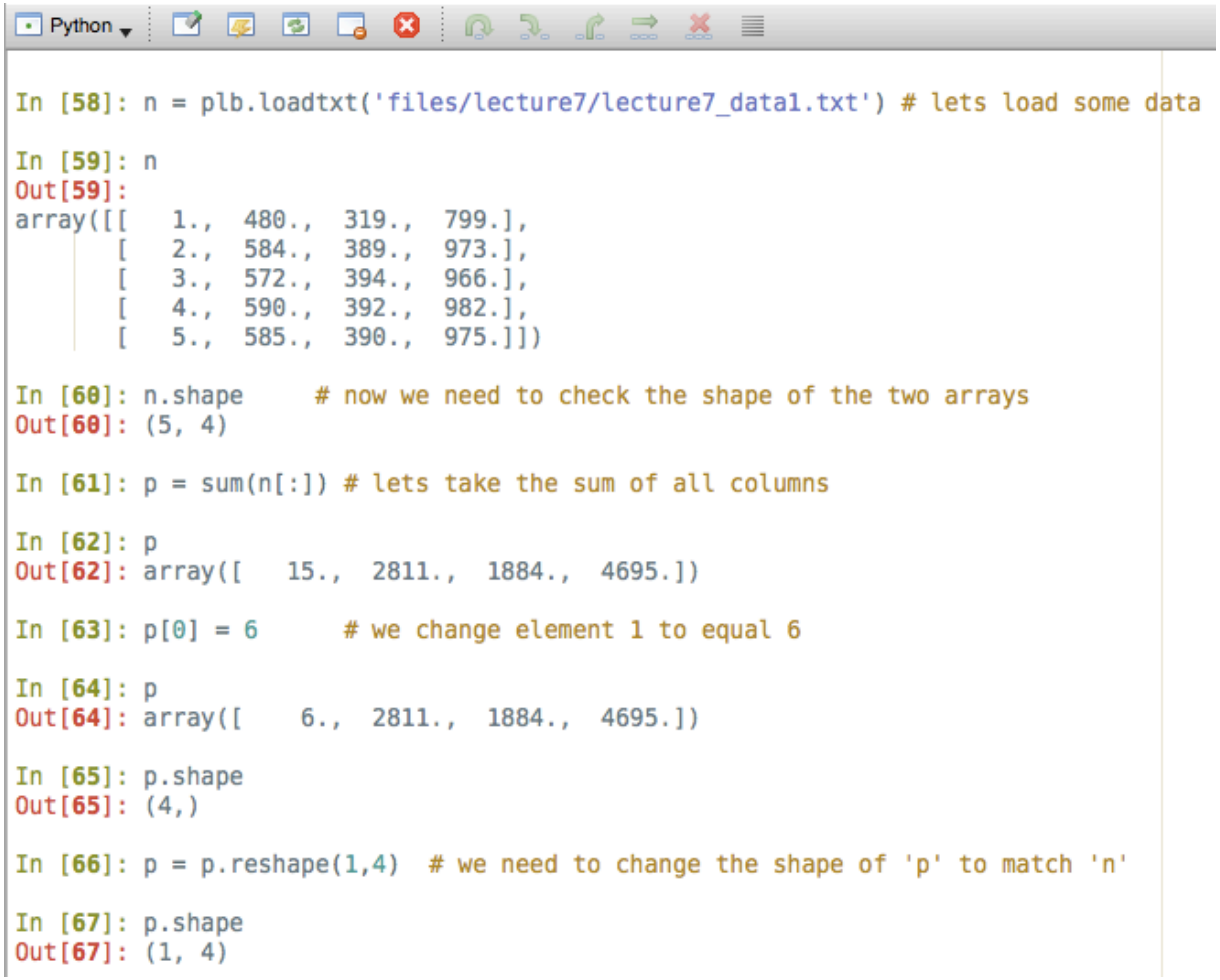
Loading data files

- Loading data files - text
  - most of the time it is necessary to load large sets of pre-calculated data for analysis
  - sometimes the data is provided through databases, but it is also common to provide data by using files
  - the most common data type that can be loaded is simple text data from large tables containing different calculations
  - here is a sample text file format we will load:

```
# sample  train  test  Total
1  480  319  799
2  584  389  973
3  572  394  966
4  590  392  982
5  585  390  975
```

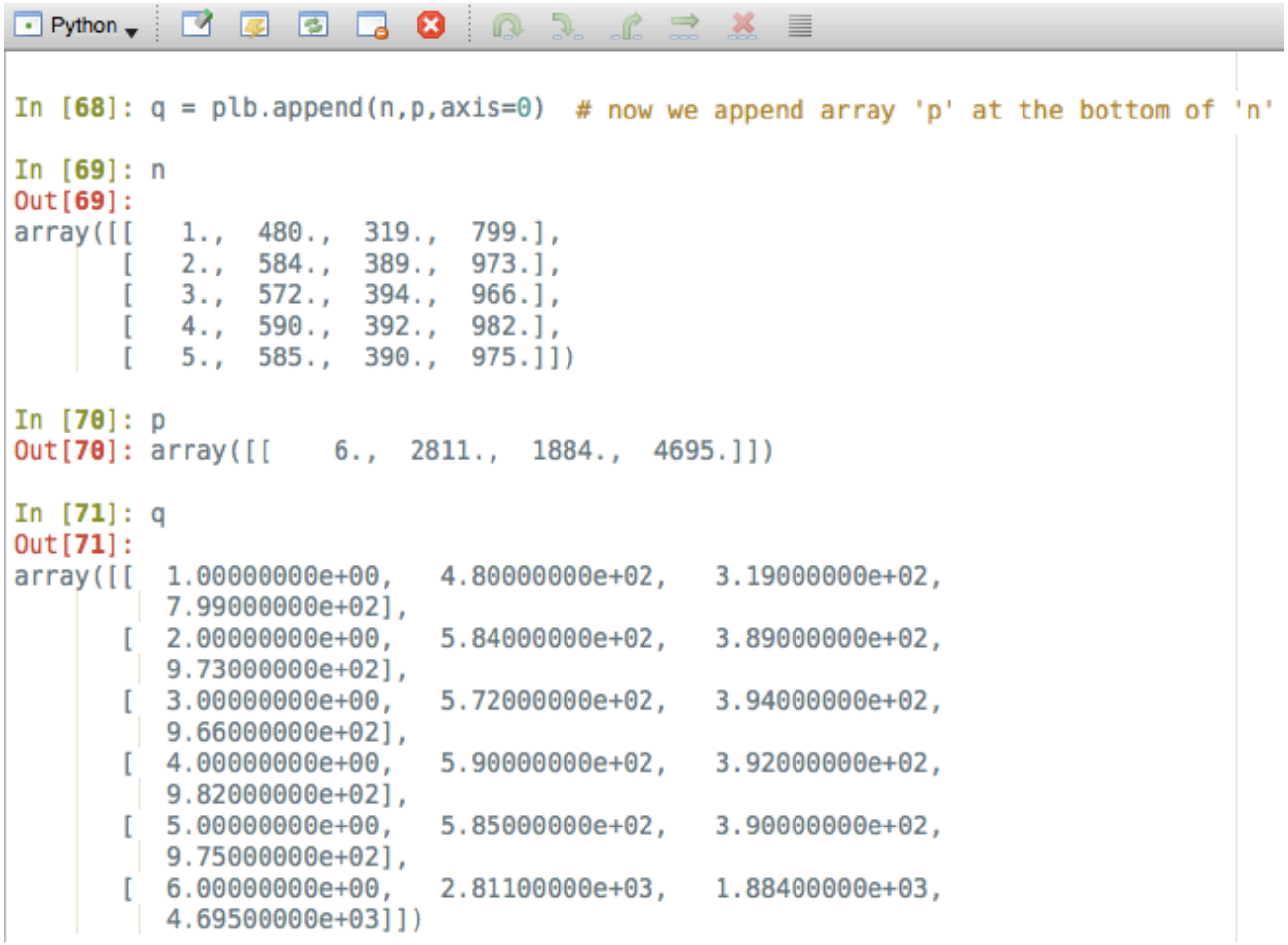
Loading data files

- Loading data files - text



Loading data files

- Loading data files - text



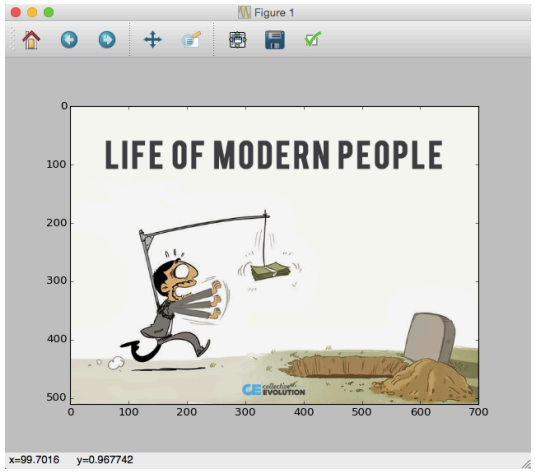
Loading data files

- Loading data files - text

```
Python

In [72]: q = q.astype(int)   # before we write to a file we type cast to 'int'

In [73]: q
Out[73]:
array([[   1,  480,  319,  799],
       [   2,  584,  389,  973],
       [   3,  572,  394,  966],
       [   4,  590,  392,  982],
       [   5,  585,  390,  975],
       [   6, 2811, 1884, 4695]])

In [74]: plb.savetxt('files/lecture7/lecture7_data2.txt',q)    # save the new data

In [75]: r = plb.loadtxt('files/lecture7/lecture7_data2.txt')  # load the new data

In [76]: r.astype(int)
Out[76]:
array([[   1,  480,  319,  799],
       [   2,  584,  389,  973],
       [   3,  572,  394,  966],
       [   4,  590,  392,  982],
       [   5,  585,  390,  975],
       [   6, 2811, 1884, 4695]])
```

Loading data files

- Loading data files – images
  - you can load images, show, manipulate and save them back to files



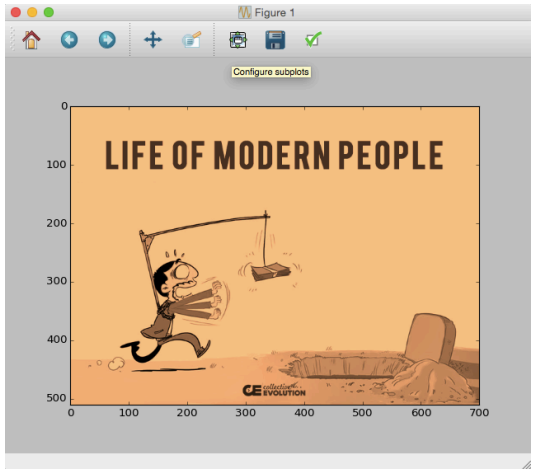
PNG - 100 KB

Python

```
In [77]: s = plb.imread('files/lecture7/test1.png') # lets read a '.png' image

In [78]: plb.imshow(s) # now show it
Out[78]: <matplotlib.image.AxesImage at 0x113718780>

In [79]: s.dtype
Out[79]: dtype('float32')

In [80]: s.shape
Out[80]: (511, 700, 3)

In [81]: plb.imsave('files/lecture7/test1_retro.png', s[:,:,0], cmap=plb.cm.copper)

In [82]: plb.savefig('files/lecture7/test1_cp.png') # saves with x,y scales

In [83]: t = plb.imread('files/lecture7/test1_retro.png') # load the changed image

In [84]: plb.imshow(t) # .. and show it
Out[84]: <matplotlib.image.AxesImage at 0x1137149e8>
```

Loading data files

- Loading data files – images
  - you can load images, show, manipulate and save them back to files



PNG - 135 KB

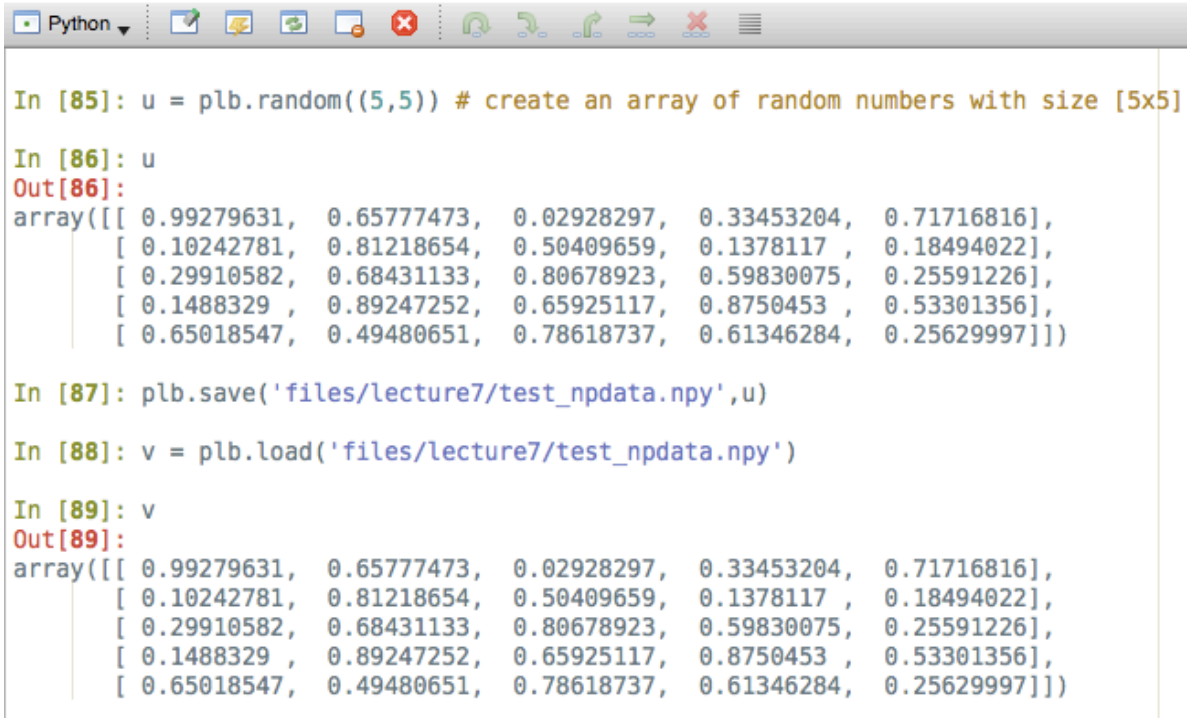
```
Python

In [77]: s = plb.imread('files/lecture7/test1.png') # lets read a '.png' image

In [78]: plb.imshow(s) # now show it
Out[78]: <matplotlib.image.AxesImage at 0x113718780>

In [79]: s.dtype
Out[79]: dtype('float32')

In [80]: s.shape
Out[80]: (511, 700, 3)

In [81]: plb.imsave('files/lecture7/test1_retro.png', s[:,:,0], cmap=plb.cm.copper)

In [82]: plb.savefig('files/lecture7/test1_cp.png') # saves with x,y scales

In [83]: t = plb.imread('files/lecture7/test1_retro.png') # load the changed image

In [84]: plb.imshow(t) # .. and show it
Out[84]: <matplotlib.image.AxesImage at 0x1137149e8>
```

observe the change in file size and number of channels

```
In [2]: s.shape
Out[2]: (511, 700, 3)

In [3]: t.shape
Out[3]: (511, 700, 4)
```

Loading data files

- Loading data files – NumPy data format
  - NumPy has its own data file writing format that efficiently stores and executed large data sets
  - … we will load sounds with SciPy later on



Loading data files

- Loading data – from databases
  - Python allows you to obtain data from an SQL Server
  - to install on your terminal do:
    >>> brew update
    >>> brew install unixodbc
  - to use type:
    >>> ipython # -> on your terminal or use any IDE (Pyzo) directly:

```
import pyodbc
import pandas.io.sql as psql

cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER=server_name;
            DATABASE=database_name;UID=USER;trusted_connection=true')
cursor = cnxn.cursor()
sql = "SELECT * from dbo.test_in_class"
df = psql.read_sql_query(sql, cnxn)
cnxn.close()
print(df)          -> it will contain the data read from the database 'test_in_class'
```

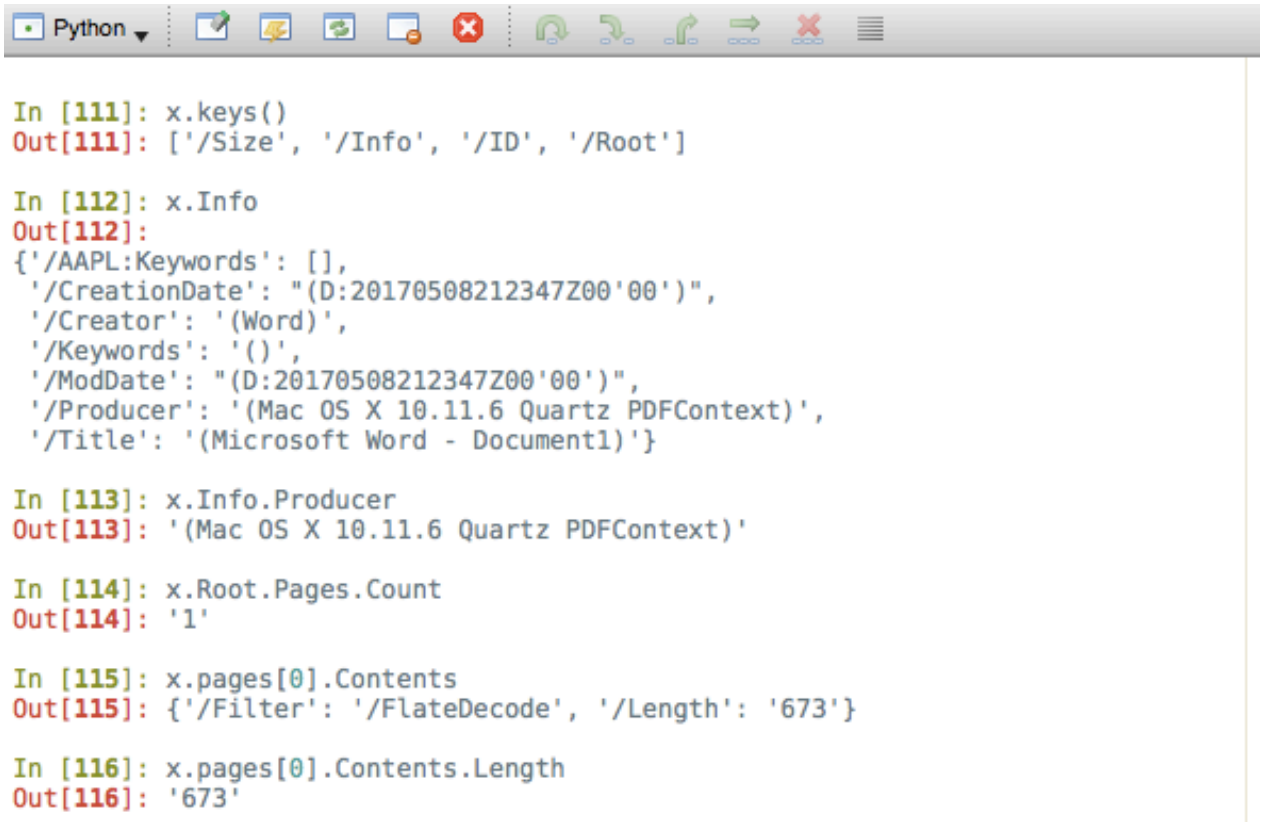
Loading data files

- Loading data – from PDF files
  - Try using: pdfrw

```
131  ## 4. Reading and writing PDFs:
132  from pdfrw import PdfReader, PdfWriter
133  # Reading pdfs:
134  x = PdfReader('files/lecture7/sample.pdf')
135  x.keys()
136  x.Info
137  x.Info.Producer
138  x.Root.Pages.Count
139  x.pages[0].Contents # returns a dictionary
140  x.pages[0].Contents.Length
141
142  # Writing pdfs:
143  y = PdfWriter()
144  y.addpage(x.pages[0])
145  y.write('files/lecture7/result1.pdf')
```

Loading data files

- Loading data – from PDF files
  - Try using: pdfrw



Loading data files

- Loading data – from PDF files
  - Try using: pdfrw

Example: merge / append files

```
147 ## 4.1 Write and append pages to pdfs:
148 import os
149 from pdfrw import PdfReader, PdfWriter
150
151 # Get a handle to the .pdf writer:
152 writer = PdfWriter()
153
154 # Check for all .pdf files in the current directory:
155 files = [x for x in os.listdir('files/lecture7/') if x.endswith('.pdf')]
156
157 # Combine all .pdf files in the current directory:
158 for fname in sorted(files):
159     writer.addpages(PdfReader(os.path.join('files/lecture7/', fname)).pages)
160
161 # Create the new .pdf file:
162 writer.write("files/lecture7/result2.pdf")
```

Loading data files

- Loading data – from Excel
  - Try using: pandas

```
159 ## Reading .csv files:
160 import pandas as pd
161
162 test = pd.read_csv('files/lecture7/Titanic.csv')
163 test.head(5)                # reads the first 5 lines
164 rows = len(test)            # counts the number of rows in the file
165 shape = test.shape          # shows the shape
166 columns = (test.columns)    # shows the column titles
167 survived = test.loc[test["Survived"] == 1]      # selection based on criteria
168 non_survived = test.loc[test["Survived"] == 0]
169
170 print(survived.count.__call__()[1])      # read information based on certain criteria
171 print(non_survived.count.__call__()[1])  # read information based on certain criteria
172 print(non_survived.count.__call__()[1] + survived.count.__call__()[1]) # total number
173
174 female = test.loc[test["Sex"] == "female"]
175 female_survived = female.loc[female["Survived"] == 1]
176
177 print(female.count.__call__()[0])          # access information
178 print(female_survived.count.__call__()[0]) # access information
```

Loading data files

- Loading data – from Excel
  - Try using: xlrd, xlwt, xlsxwriter

Xlwt - generate spreadsheet files compatible with Microsoft Excel versions 95 to 2003
Xlrd - This package is for reading data and formatting information from older Excel
Openpyxl - for reading and writing Excel 2010 xlsx/xlsm/xltx/xltm files

```
181  # begin reading '.xlsx' file
182  import openpyxl
183  import os
184
185  # begin reading '.xls' file
186  import xlrd     # for reading
187  import xlwt     # for writing
188
189  # read all fields of interest specified in file named 'list of indicators.xlsx':
190  wb = openpyxl.load_workbook('list of indicators.xlsx')
191  all_sheets = (wb.sheetnames)
192  sheet = wb.get_sheet_by_name(all_sheets[0]) # all_sheets[0] = "Sheet 1"
```

Dealing with polynomials

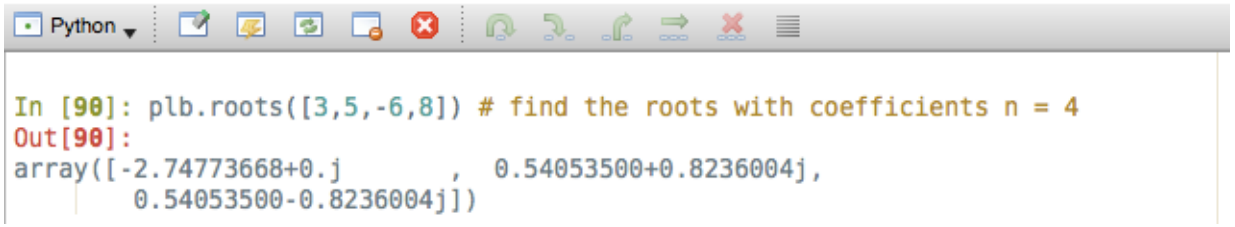
- Dealing with polynomials
  - NumPy has several ways of solving polynomials
  - using roots:

will find the roots of a polynomial with coefficients [n] represented in p[n]:
p[1] * x**(n-1) + ... + p[n-1]*x + p[n]

Example: find the roots of a polynomial with n = 4 coefficients [3, 5, -6, 8] :

$$3 * x^3 + 5 * x^2 - 6 * x + 8$$

solution:



Dealing with polynomials

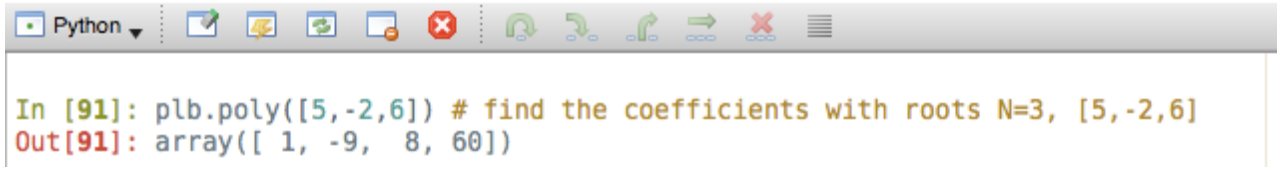
- Dealing with polynomials
  - NumPy has several ways of solving polynomials
  - using poly:

will find the coefficients of a polynomial given the roots:

c[0] * x**(n) + c[1] * x**(n-1) + ... + c[n-1] * x + c[n] where c[0] = 1 always

Example: find the coefficients of polynomial with n = 3 roots [5,-2,6] :

solution:



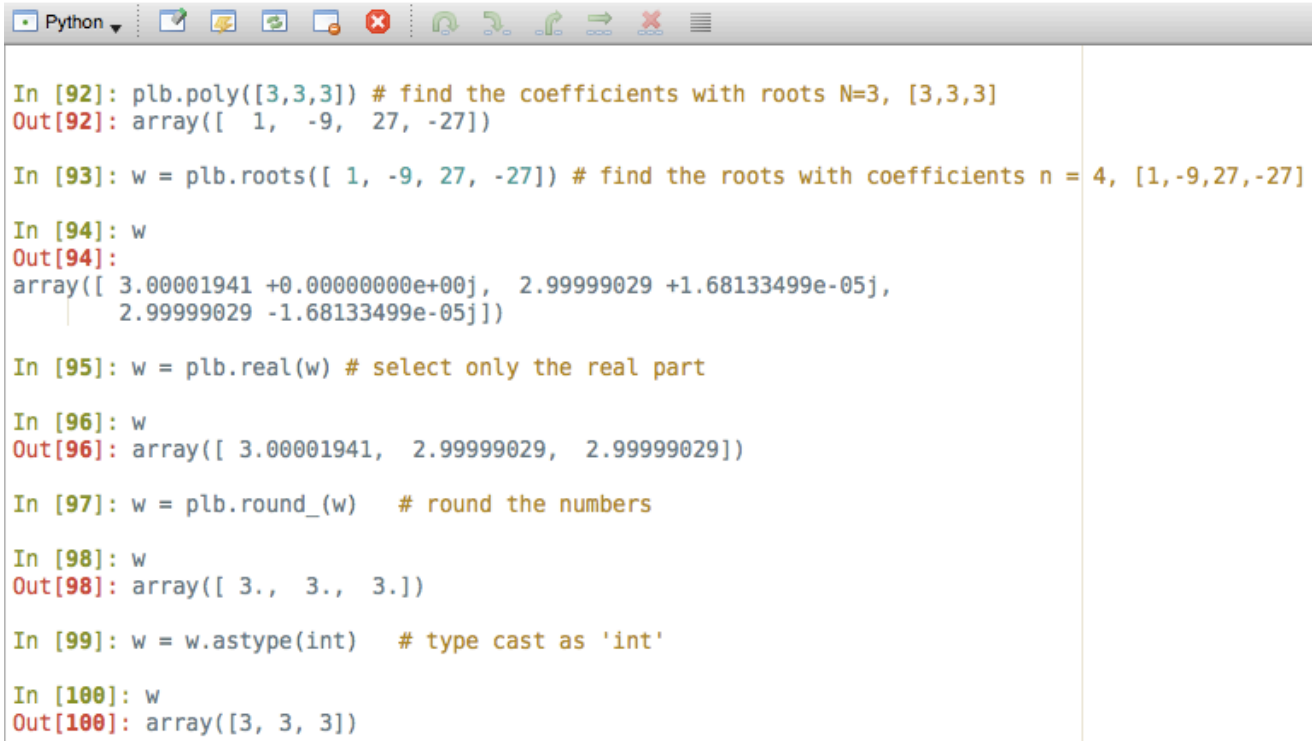
the polynomial looks like this:

$$1 * z^3 - 9 * z^2 + 8 * z + 60$$

Dealing with polynomials

- Dealing with polynomials
  - NumPy has several ways of solving polynomials
  - using roots and poly:

Example:



Dealing with polynomials

- Dealing with polynomials
  - NumPy has several ways of solving polynomials
  - using polyfit: to find the least squares polynomial fit:

$$p(x) = p[0] * x^{**}deg + ... + p[deg]$$

where *deg* is the degree of the fitting polynomial to points (x, y)
the solution is a vector with coefficients p, which minimizes the squared error (Least Square Error)

  - The minimization of the square error in equations:

x[0]**n * p[0] + ... + x[0] * p[n-1] + p[n] = y[0]
x[1]**n * p[0] + ... + x[1] * p[n-1] + p[n] = y[1] -> more equations than unknowns
...
x[k]**n * p[0] + ... + x[k] * p[n-1] + p[n] = y[k]

is given by:

$$E_{LS} = \sum_{i=0}^{m} |p(x_i) - y_i|^2$$

where E_{LS} is the Least Square Error and it minimizes the sum of all square residuals

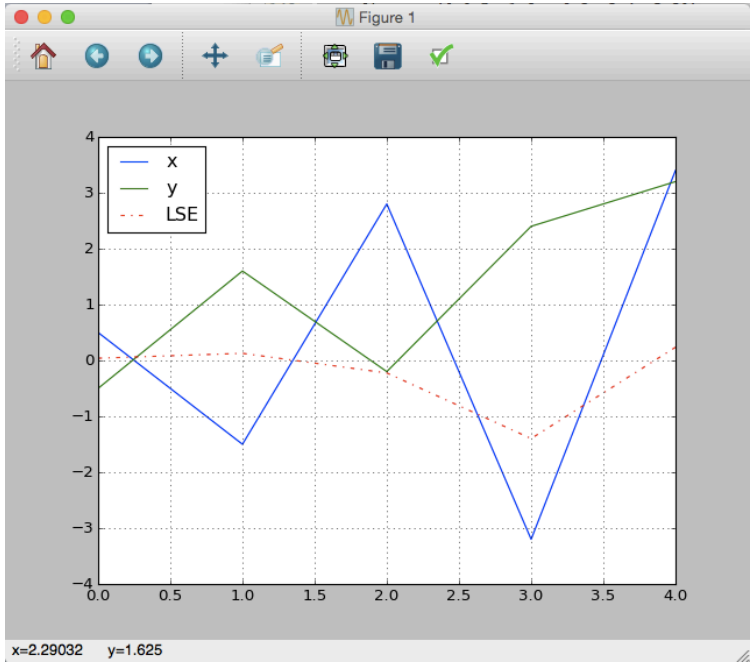
Dealing with polynomials

- Dealing with polynomials
  - NumPy has several ways of solving polynomials
  - using polyfit for least squares polynomial fit:

p(x) = p[0] * x**n+ ... + p[n] where n is the degree of the fitting polynomial to points *(x, y)*
The solution is a vector of coefficients p that minimizes the squared error

Example:

```
144  # Using polyfit - least squares polynomial fit:
145  x = plb.array([0.5, -1.5, 2.8, -3.2,  3.4])
146  y = plb.array([-0.5, 1.6, -0.2, 2.4, 3.2])
147  z = plb.polyfit(x, y, 4)
148  z
149  plb.plot(x, color='b', label='x')
150  plb.grid(True)
151  plb.hold(True)
152  plb.plot(y, color='g', label='y')
153  plb.plot(z, linestyle='-.', color='r', label='LSE')
154  plb.legend(loc='upper left')
```



Dealing with polynomials

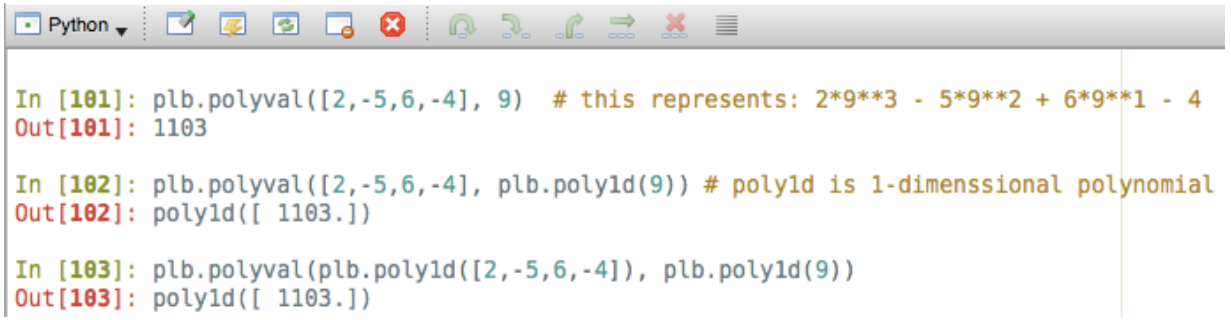
- Dealing with polynomials
  - NumPy has several ways of solving polynomials
  - using polyval to evaluate polynomial at a specific point:

p[0]*x**(n-1) + p[1]*x**(n-2) + ... + p[n-2]*x + p[n-1]

where: - p is a 1D array of polynomial coefficients,
- p is of length n
- when x is a sequence, p(x) is returned for each element of x

Example:

$$2 * 9^3 - 5 * 9^2 + 6 * 9 - 4$$



Dealing with polynomials

- Dealing with polynomials
  - NumPy has several ways of solving polynomials
  - using poly1d is a one dimensional polynomial class. Two main usages:

    1/ poly1d([3,-4,7]) represents $3 * x^2 - 4 * x + 7$

    2/ poly1d([3,-4,7], True) represents $(x-3)(x+4)(x-7) = x^3 - 6x^2 - 19x + 84$

  - we can perform the following operations on polynomials:
    - » addition
    - » subtraction
    - » multiplication
    - » division
    - » gradation

Dealing with polynomials

- Dealing with polynomials

1/ poly1d([3,-4,7]) represents $3 * x^2 - 4 * x + 7$

2/ poly1d([3,-4,7], True) represents $(x-3)(x+4)(x-7) = x^3 - 6x^2 - 19x + 84$

Example:

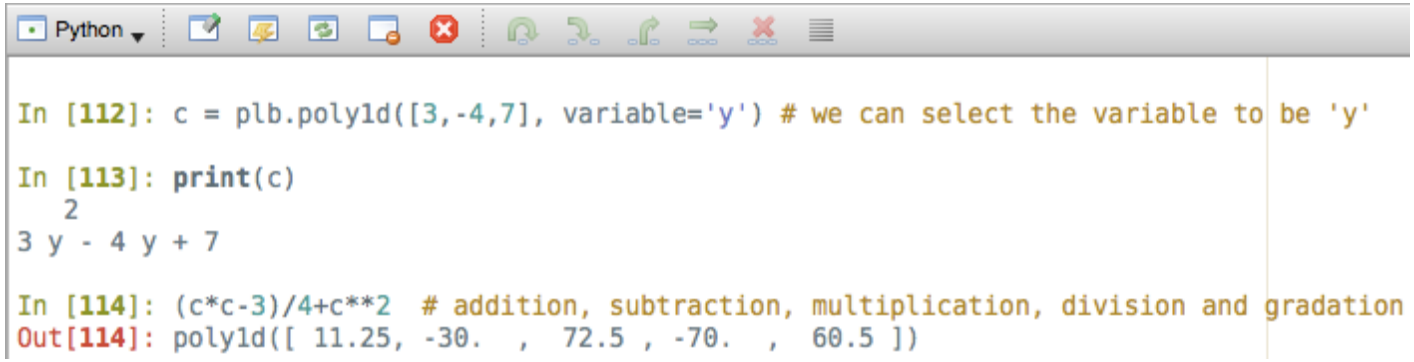
Python

```
In [103]: a = plb.poly1d([3,-4,7]) # represents: 3*x**2 - 4*x + 7

In [104]: print(a)    # construct the polynomial
   2
3 x - 4 x + 7

In [105]: b = plb.poly1d([3,-4,7], True)   # represents: (x-3)(x+4)(x-7)

In [106]: print(b)    # construct the polynomial
   3     2
1 x - 6 x - 19 x + 84

In [107]: b.r         # show the roots of the polynomial
Out[107]: array([-4.,  7.,  3.])

In [108]: b.c         # show the coefficients of the polynomial
Out[108]: array([  1,  -6, -19,  84])

In [109]: b.order     # show the order of the polynomial
Out[109]: 3

In [110]: a(8)        # evaluate the polynomial at point x=8
Out[110]: 167

In [111]: b(8)        # evaluate the polynomial at point x=8
Out[111]: 60
```

Dealing with polynomials

- Dealing with polynomials
  - we can change the variable of the polynomial
  - we can perform the following operations on polynomials:
    - » addition
    - » subtraction
    - » multiplication
    - » division
    - » gradation

Example:

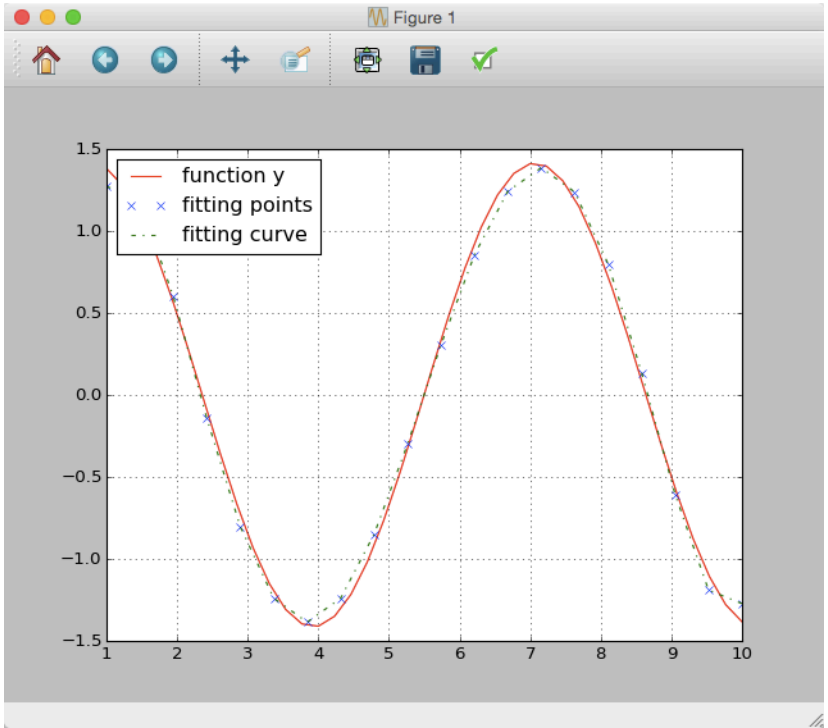


Dealing with polynomials

- Dealing with polynomials

Example:

```
179  x = plb.linspace(1, 10, 40) # create vector 'x' with 40 numbers between [1:10]
180  y = plb.cos(x) + plb.sin(x) # create and calculate function 'y'
181  p = plb.poly1d(plb.polyfit(x, y, 5))    # find the LSE with 5 degrees fitting
182  t = plb.linspace(1, 10, 20)
183  plb.plot(x, y, color='r', label='function y')   # plot with some specifics
184  plb.plot(t, p(t), 'x', label='fitting points')
185  plb.plot(t, p(t), '-.', label='fitting curve')
186  plb.grid(True)
187  plb.legend(loc='upper left')    # add the legend will labels
```



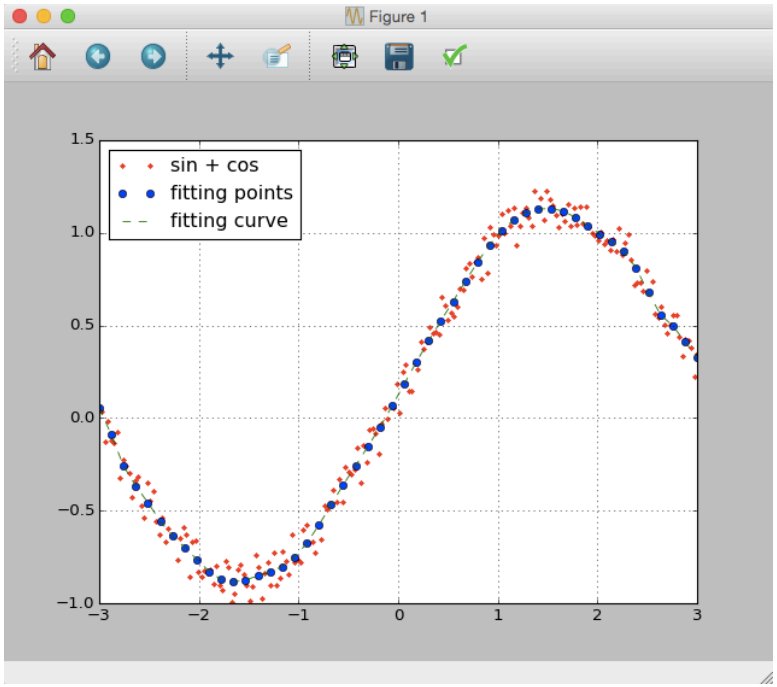
Dealing with polynomials

- Dealing with polynomials

Example:

```
190 import numpy as npy # we import numpy to use the 'polynomial' method
191 x = npy.linspace(-3, 3, 200) # create vector 'x' with 200 numbers between [-3:3]
192 y = npy.sin(x) + 0.25*plb.rand(200) # create and calculate function 'y'
193 p = npy.polynomial.Legendre.fit(x, y, 24)  # use a Legendre series class object
194 t = npy.linspace(-3, 3, 50)
195 plb.plot(x, y, 'r.', lw=1.75, label='sin + cos') # plot with some specifics
196 plb.plot(t, p(t), 'o', label='fitting points')
197 plb.plot(t, p(t), '--', linewidth=.75, label='fitting curve')
198 plb.grid(True)
199 plb.legend(loc='upper left')    # add the legend will labels
```

Note:
In order to take advantage of the 'numpy.polynomial' method, which contains different series such as Chebyshev, Hermite, Legendre and Laguerre we import numpy



Course Content Outline

- **NumPy 2/3**
- Array operations
- Reductions
- Broadcasting
- Array: shaping, reshaping, flattening, resizing, dimension changing
- Data sorting
- Good coding practices

- **NumPy 3/3**
- Type casting
- Masking data
- Organizing arrays
- Loading data files
- Dealing with polynomials — Midterm, Project proposal due

- **Scipy 1/2**
- What is Scipy?
- Working with files
- Algebraic operations
- The Fast Fourier Transform
- Signal Processing

- **Scipy 2/2**
- Interpolation
- Statistics
- Optimization
- Multithreading and Multiprocessing

- **Project**
- Project Presentation — Final Project

What is Scipy?

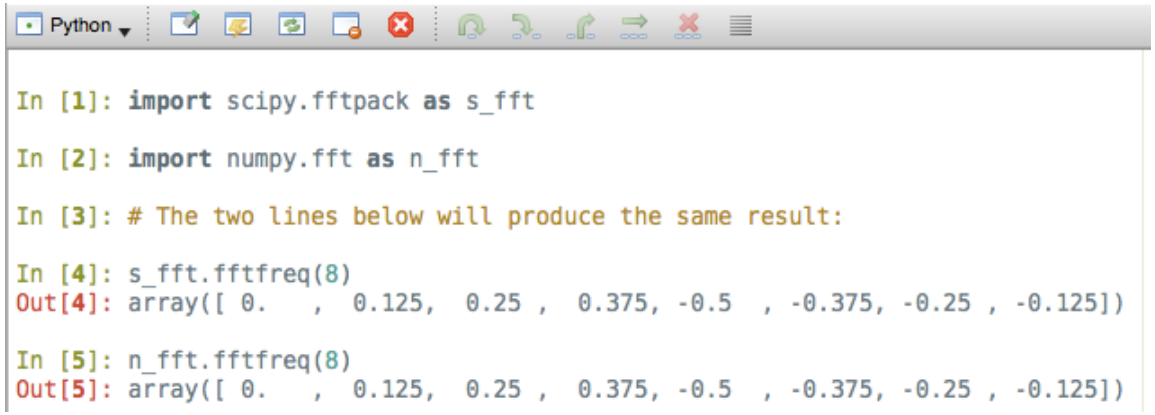
- What is Scipy?
  - Scipy is Python's scientific core package designed for high level scientific computing
  - it contains many prebuild common mathematical functions, series, transforms that are easy to use
  - its toolboxes are well set to solve common scientific problems
  - good knowledge of the package allows for building complicated algorithms and proof-of concept solutions
  - Scipy has modules that can work on problems involving integration, optimization, interpolation, image and sound processing, Fourier transform, statistics, and many other special functions
  - Scipy's scientific libraries resemble the ones available in Matlab
  - Scipy integrates very well with NumPy and can operate on NumPy arrays very efficiently

What is Scipy?

- What is Scipy?
  - here is a list of some of the most common Scipy modules:
    - Scipy.fftpack – contains Fast Fourier Transforms (FFTs)
    - Scipy.integrate – contains a variety of integrating functions
    - Scipy.interpolate – contains a variety of interpolation classes
    - Scipy.io – functions for reading and writing data from/to a variety of file formats
    - Scipy.io.wavfile – read/write data from/to a variety of file formats 'wav','arff', etc.
    - Scipy.linalg – contains linear algebra routines
    - Scipy.ndimage – contains many functions for multi-dimensional image processing
    - Scipy.signal – rich filtering capabilities, wavlets, spectral analysis, and much more
    - Scipy.optimize – local optimization package and root finding
    - Scipy.spatia – nearest neighbor queries and distance functions
    - Scipy.stats – large number of probability distributions and statistical functions
    - Scipy.special – large variety of functions such as: elliptic, bessel, legendre, etc.
    - Scipy.misc – variety of other functions

What is Scipy?

- What is Scipy?
  - scipy.fftpack vs numpy.fft:
    - Some of the NumPy code is exported through Scipy, hence there are similarities between the two packages:



    - scipy.sin = numpy.sin, etc. –

```
cpy.sin(90)
0.89399666360055785

npy.sin(90)
0.89399666360055785
```

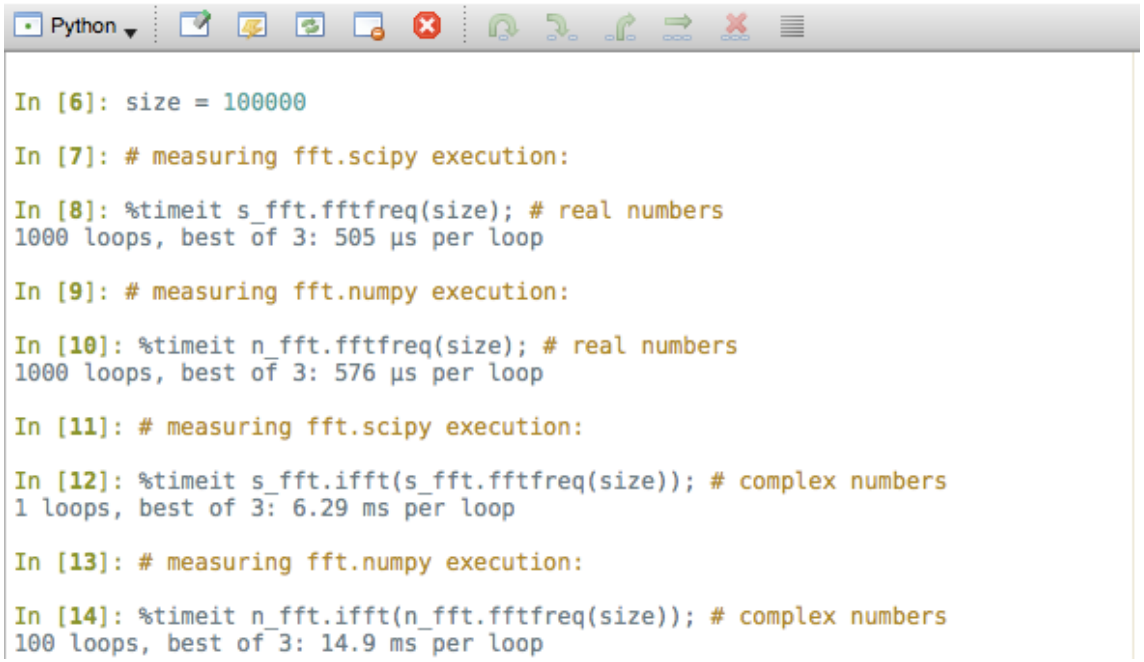
What is Scipy?

- What is Scipy?
  - scipy.fftpack vs numpy.fft:
    - the scipy.fftpack does much more on top of what numpy.fft offers:
      - » fft and ifft - the Discrete Fourier Transform and its inverse of real or complex sequence of numbers
      - » fft2 and ifft2 - 2D discrete Fourier transform and its inverse
      - » fftn and ifftn - multidimensional discrete Fourier transform and its inverse
      - » dct and idct - Discrete Cosine Transform of arbitrary type sequence
      - » dst and idst - Discrete Sine Transform of arbitrary type sequence
      - » tilbert and itilbert - the h-Tilbert transform of a periodic sequence and its inverse
      - » hilbert and ihilbert - Hilbert Transform of a periodic sequence and its inverse
      - » fftfreq - the Discrete Fourier Transform sample frequencies
      - » convolve - performs convolution on a given signal
      - » … and more

What is Scipy?

- What is Scipy?
  - scipy.fftpack vs numpy.fft:
    - Scipy runs faster

Scipy is much faster when arrays increases in size



What is Scipy?

- What is Scipy?
  - we usually import Scipy like this:

```
[15] import numpy as np
[16] from scipy import signal      # or choose any other Scipy module
```

  - since there are many modules in Scipy, we usually import only specific Scipy functionality and never need to import the complete Scipay package
  - Scipy depends on the NumPy library, while it provides convenient and fast N-dimensional array operation and manipulation
  - just like Python, NumPy and Matplotlib, Scipy is open-source

Working with files

- Working with files
  - most of the times it is necessary to load and save data files of different types such as:
    - text, sound or image
  - we already saw that we can import text files and images with NumPy
  - using Scipy, we can create and import more sophisticated file structures such as .csv, .mat, etc.
  - since Matlab .mat files are widely used in the scientific computing community, using Scipy we can read and write .mat files
  - using Python we can write directly to databases using a specifically designed non-SQL storage ways such as: PyTables and HDF5, but also in regular SQL tables

Working with files

- Working with files – text / .mat – 1/2

saving and loading .mat files

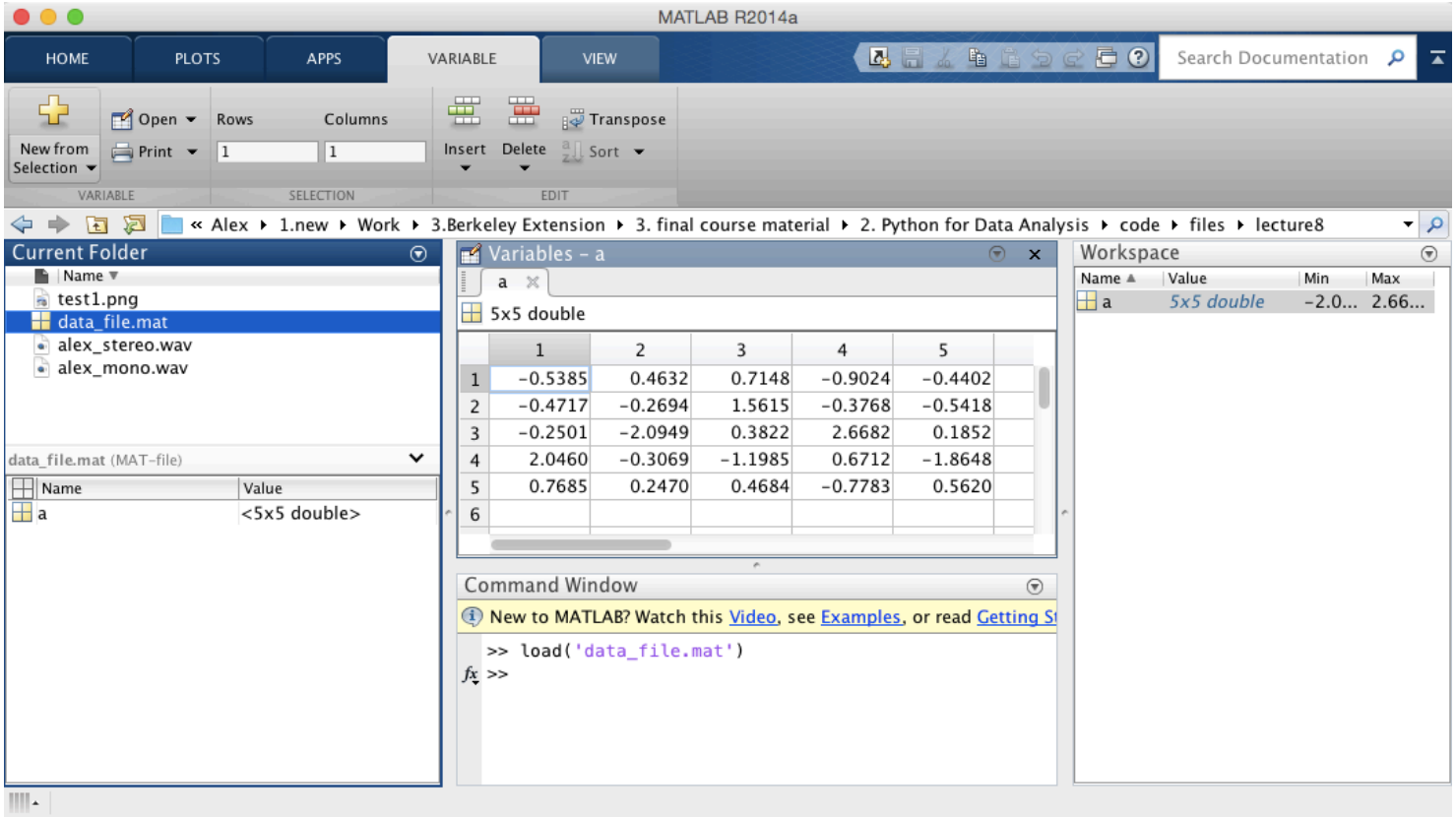
Python

```
In [17]: from scipy import randn as rnd

In [18]: from scipy import io as sio

In [19]: a = rnd(5, 5) # create an array using "standard normal" distribution

In [20]: a
Out[20]:
array([[-0.53850611,  0.46316393,  0.71478529, -0.90239455, -0.44023213],
       [-0.47173271, -0.26944028,  1.56151908, -0.37684898, -0.5418205 ],
       [-0.2501384 , -2.0949024 ,  0.38220796,  2.66823633,  0.18517809],
       [ 2.04596859, -0.30689392, -1.19849106,  0.67119412, -1.86481571],
       [ 0.7685391 ,  0.24704472,  0.46837002, -0.77831832,  0.56196278]])

In [21]: sio.savemat('files/lecture8/data_file.mat', {'a':a}) # savemat expects a dictionary

In [22]: a_data = sio.loadmat('files/lecture8/data_file.mat', struct_as_record=True)

In [23]: a_data      # this call will bring the heather containing Matlab details
Out[23]:
{'__version__': '1.0',
 '__globals__': [],
 '__header__': b'MATLAB 5.0 MAT-file Platform: posix, Created on: Mon Jul 13 21:58:13 2015',
 'a': array([[-0.53850611,  0.46316393,  0.71478529, -0.90239455, -0.44023213],
        [-0.47173271, -0.26944028,  1.56151908, -0.37684898, -0.5418205 ],
        [-0.2501384 , -2.0949024 ,  0.38220796,  2.66823633,  0.18517809],
        [ 2.04596859, -0.30689392, -1.19849106,  0.67119412, -1.86481571],
        [ 0.7685391 ,  0.24704472,  0.46837002, -0.77831832,  0.56196278]])}

In [24]: a_data['a'] # this call will simply show the data - only
Out[24]:
array([[-0.53850611,  0.46316393,  0.71478529, -0.90239455, -0.44023213],
       [-0.47173271, -0.26944028,  1.56151908, -0.37684898, -0.5418205 ],
       [-0.2501384 , -2.0949024 ,  0.38220796,  2.66823633,  0.18517809],
       [ 2.04596859, -0.30689392, -1.19849106,  0.67119412, -1.86481571],
       [ 0.7685391 ,  0.24704472,  0.46837002, -0.77831832,  0.56196278]])
```

Working with files

- Working with files – text / .mat – 2/2

loading .mat files in Matlab, created in Scipy



Working with files

- Working with files – image 1/3
  - to be able to read images using Scipy, we may import Python Imaging Library (PIL)
  - PIL is for Python 2
  - PIL is not a dependency of SciPy
  - PIL needs to be installed separately, but it sometimes needs a relaxed installation to avoid rejection of package installation:
    - pip install pil --allow-external pil --allow-unverified pil
      --allow-external pil means to allow installation from an external source
      --allow-unverified pil means not to validate package using checksum

Working with files

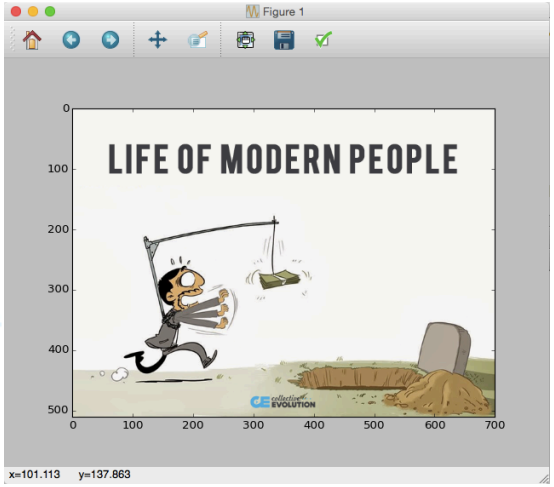
- Working with files – image 2/3
  - Pillow is the “friendly PIL” package (easier to remember)
  - Pillow is based on PIL, but evolved to be better, friendlier and more modern than PIL
  - Pillow is for Python 2 and 3
  - the Pillow image module needs to be installed separately:

```
[25] pip install pillow
Collecting pillow
  Downloading Pillow-2.9.0.tar.gz (9.3MB)
Installing collected packages: pillow
  Running setup.py install for pillow
Successfully installed pillow-2.9.0
```

Working with files

- Working with files – image 3/3
  - ndimage can take advantage of the newly installed PIL (Pillow) functionality and is part of the Scipy main package

```
Python

In [26]: # Scipy has this image functionality:

In [27]: from scipy import ndimage as simg

In [28]: img1 = simg.imread('files/lecture8/test1.png')

In [29]: img1.shape
Out[29]: (511, 700, 3)

In [30]: # Matplotlib has a similar functionality:

In [31]: import matplotlib.pyplot as mpt

In [32]: img2 = mpt.imread('files/lecture8/test1.png')

In [33]: img2.shape
Out[33]: (511, 700, 3)

 47  mpt.imshow(img2) # unlike with Scipy, we can plot with Matplotlib
 48  plt.pause(2)
 49  mpt.imshow(img1) # we can also plot 'img1' since they are both NumPy arrays
 50  plt.pause(1)
```



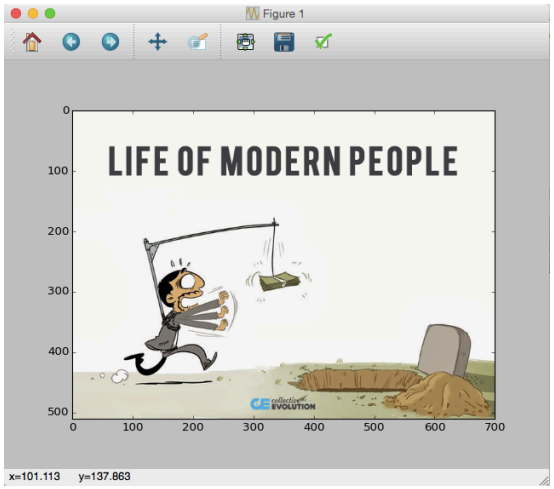
Working with files

- Working with files – image 3/3
  - ndimage using Pillow, can read images in different mode with different pixel resolutions

```
57  img2 = simg.imread('files/lecture8/test1.png', flatten=False, mode='RGB')
```

  - same can be said for pyplot from Matplotlib, since it also uses Pillow to plot
  - Here are the options:

```
`mode` can be one of the following strings:

* 'L' (8-bit pixels, black and white)
* 'P' (8-bit pixels, mapped to any other mode using a color palette)
* 'RGB' (3x8-bit pixels, true color)
* 'RGBA' (4x8-bit pixels, true color with transparency mask)
* 'CMYK' (4x8-bit pixels, color separation)
* 'YCbCr' (3x8-bit pixels, color video format)
* 'I' (32-bit signed integer pixels)
* 'F' (32-bit floating point pixels)

PIL also provides limited support for a few special modes, including
'LA' ('L' with alpha), 'RGBX' (true color with padding) and 'RGBa'
(true color with premultiplied alpha).
```

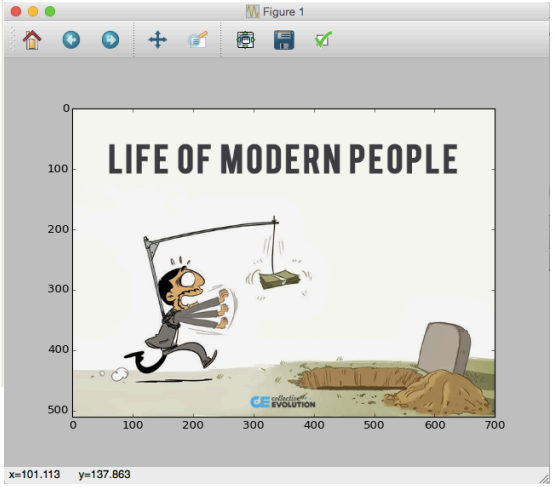


Working with files

- Working with files – image 3/3
  - be careful how you use the options:

```
In [300]: img1 = simg.imread('files/lecture8/test1.png')

In [301]: img1.shape
img1.nbytes
img1.itemsize
Out[301]: (511, 700, 3)
Out[302]: 1073100
Out[303]: 1

In [304]: img2 = simg.imread('files/lecture8/test1.png', flatten=False, mode='RGB')

In [305]: img2.shape
img2.nbytes
img2.itemsize
Out[305]: (511, 700, 3)
Out[306]: 1073100
Out[307]: 1

In [308]: img3 = simg.imread('files/lecture8/test1.png', 'RGB')

In [309]: img3.shape
img3.nbytes
img3.itemsize
Out[309]: (511, 700)
Out[310]: 1430800
Out[311]: 4
```

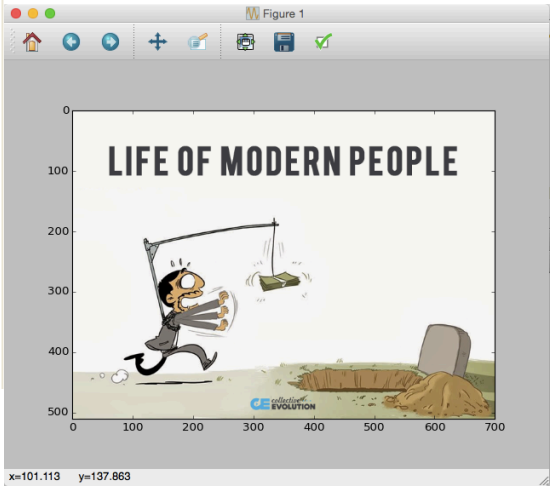
When `mode` is not None and `flatten` is True, the image is first converted according to `mode`, and the result is then flattened using mode 'F'



Working with files

- Working with files – image 3/3
  - be careful how you use the options:

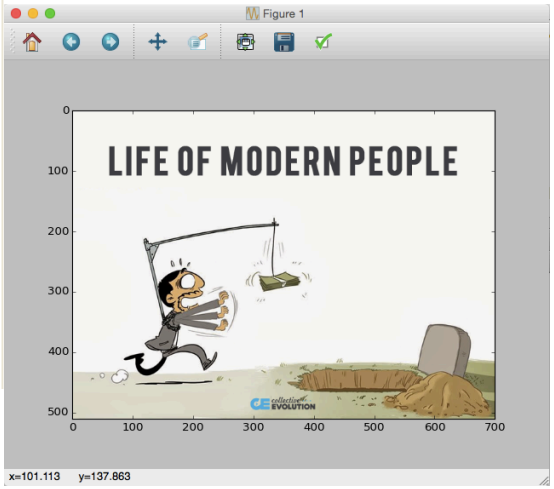
```
In [300]: img1 = simg.imread('files/lecture8/test1.png')

In [301]: img1.shape
img1.nbytes
img1.itemsize
Out[301]: (511, 700, 3)
Out[302]: 1073100
Out[303]: 1

In [304]: img2 = simg.imread('files/lecture8/test1.png', flatten=False, mode='RGB')

In [305]: img2.shape
img2.nbytes
img2.itemsize
Out[305]: (511, 700, 3)
Out[306]: 1073100
Out[307]: 1

In [308]: img3 = simg.imread('files/lecture8/test1.png', 'RGB')

In [309]: img3.shape
img3.nbytes
img3.itemsize
Out[309]: (511, 700)
Out[310]: 1430800
Out[311]: 4
```



Working with files

- Working with files – image 3/3
  - be careful how you use the options:

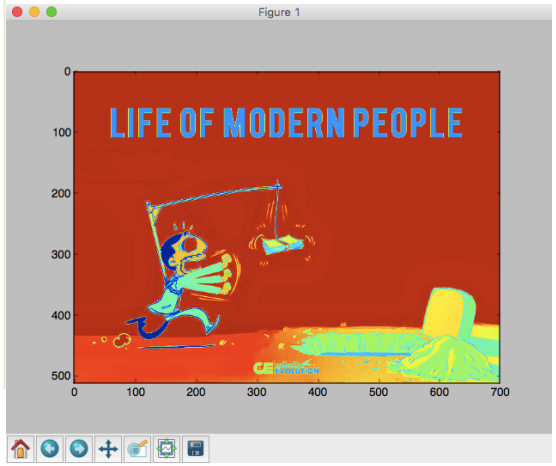
```
Python

In [300]: img1 = simg.imread('files/lecture8/test1.png')

In [301]: img1.shape
img1.nbytes
img1.itemsize
Out[301]: (511, 700, 3)
Out[302]: 1073100
Out[303]: 1

In [304]: img2 = simg.imread('files/lecture8/test1.png', flatten=False, mode='RGB')

In [305]: img2.shape
img2.nbytes
img2.itemsize
Out[305]: (511, 700, 3)
Out[306]: 1073100
Out[307]: 1

In [308]: img3 = simg.imread('files/lecture8/test1.png', 'RGB')

In [309]: img3.shape
img3.nbytes
img3.itemsize
Out[309]: (511, 700)
Out[310]: 1430800
Out[311]: 4
```



Working with files

- Working with files – image 3/3
  - be careful how you use the options:

```
In [300]: img1 = simg.imread('files/lecture8/test1.png')

In [301]: img1.shape
img1.nbytes
img1.itemsize
Out[301]: (511, 700, 3)
Out[302]: 1073100
Out[303]: 1

In [304]: img2 = simg.imread('files/lecture8/test1.png', flatten=False, mode='RGB')

In [305]: img2.shape
img2.nbytes
img2.itemsize
Out[305]: (511, 700, 3)
Out[306]: 1073100
Out[307]: 1

In [308]: img3 = simg.imread('files/lecture8/test1.png', 'RGB')

In [309]: img3.shape
img3.nbytes
img3.itemsize
Out[309]: (511, 700)
Out[310]: 1430800
Out[311]: 4
```



Working with files

- Working with files – sound 1/5

reading .wav files

Python

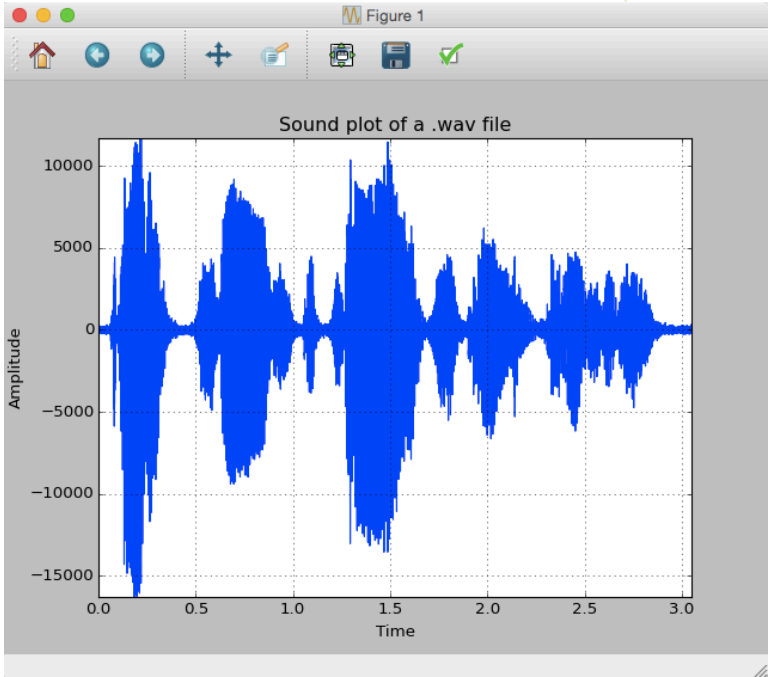
```
In [35]: from scipy.io.wavfile import read

In [36]: (fsx, x) = read('files/lecture8/alex_mono.wav')

In [37]: print(len(x.shape))   # '1' is mono
1

In [38]: print(x[:,])   # to access the channel no digit is used after ','
[-4 43 23 ..., 27 42 -3]

In [39]: x
Out[39]: array([-4, 43, 23, ..., 27, 42, -3], dtype=int16)

In [40]: (fsy, y) = read('files/lecture8/alex_stereo.wav')

In [41]: print(len(y.shape))   # '2' is stereo 2-dimensional array
2

In [42]: y
Out[42]:
array([[-4, -4],
       [44, 43],
       [20, 23],
       ...,
       [26, 29],
       [43, 41],
       [-4, -3]], dtype=int16)

In [43]: print(y[:,0])   # to access each channel separately use '0' or '1'
[-4 44 20 ..., 26 43 -4]

In [44]: print(y[:,1])
[-4 43 23 ..., 29 41 -3]
```

Working with files

- Working with files – sound 2/5

plotting .wav files

```
67  # More on sound:
68  # Example 1:
69  from pylab import linspace, plot, title, xlabel, ylabel, grid, axis
70  from scipy.io.wavfile import read
71
72  (Fs, x) = read('files/lecture8/alex_mono.wav') # Fs - sampling frequency, x - signal
73  length = len(x)  # number of smaples in 'x'
74  time = length/Fs # calculate the length of the .wav file in secs
75  t = linspace(0,time,length) # create evenly spaced numbers between [0:time]
76  plot(t,x) # plot signal 'x'
77  title('Sound plot of a .wav file')
78  xlabel('Time')
79  ylabel('Amplitude')
80  axis('tight')
81  grid(True)
```



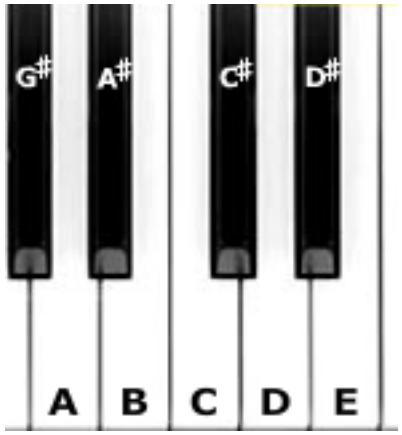
Working with files

- Working with files – sound 3/5
  - lets create a our own music by using the sin function alone

generating
writing
saving .wav files

Tone	Freq
A	440
B flat	466
B	494
C	523
C sharp	554
D	587
D sharp	622
E	659
F	698
F sharp	740
G	784
A flat	831
A	880

+



=

E C# C# D B B A B C# D E E E
E C# C# D B B A C# E E A A A

our melody

Working with files

- Working with files – sound 3/5
  - lets create our own music file by using the sin function alone

generating
writing
saving .wav files

```
# Example 2:
from pylab import plot, title, xlabel, ylabel, grid, axis
from numpy import linspace, int16, zeros, asarray, concatenate
from scipy import arange, sin, pi
from scipy.io.wavfile import read,write

Fs=8000 # Sampling frequency Fs

# Synthesis of the note:
def note(freq, Fs, length, amplitude=5):
 t = linspace(0,length,length*Fs)
 sig = sin(2*pi*freq*t)*amplitude
 return sig.astype(int16) # returns 2 byte integers

# Creating each tone with Fs samples per second and length 0.5 or 0.8 seconds:
Es = note(659,Fs,0.5,amplitude=5000)
El = note(659,Fs,0.8,amplitude=5000)
Css = note(554,Fs,0.5,amplitude=5000)
Csl = note(554,Fs,0.8,amplitude=5000)
D = note(587,Fs,0.5,amplitude=5000)
Bs = note(494,Fs,0.5,amplitude=5000)
Bl = note(494,Fs,0.8,amplitude=5000)
As = note(440,Fs,0.5,amplitude=5000)
Al = note(440,Fs,0.8,amplitude=5000)
Pau = zeros([200], dtype=int16) # this is the pause between the tones

# Create the melody:
All = concatenate((Es,Pau,Css,Pau,Csl,Pau,D,Pau,Bs,Pau,Bl,Pau,As,Pau,Bs,Pau,
                   Css,Pau,D,Pau,Es,Pau,Es,Pau,El,Pau,Pau,
                   Es,Pau,Css,Pau,Csl,Pau,D,Pau,Bs,Pau,Bl,Pau,As,Pau,Css,Pau,
                   Es,Pau,Es,Pau,As,Pau,As,Pau,Al))

write('files/lecture8/melody.wav',Fs,All) # writing our melody to a file
```

=

E C# C# D B B A B C# D E E E
E C# C# D B B A C# E E A A A

our melody

… lets hear the melody

Working with files

- Working with files – sound 4/5
  - lets create a (pseudo) stereo music from a single (mono) channel

manipulating .wav files

Note: this is not a true stereo Signal

```
130 ## Example 3 - manipulating sounds - creating pseudo-stereo from mono:
131 from numpy import zeros, concatenate
132 from scipy import fft, arange, ifft, sin, pi
133 from scipy.io.wavfile import read,write
134
135 (Fs, x) = read('files/lecture8/melody.wav')
136 x      # contains all the samples
137 y = x  # we create the second channel
138 z=zeros([200])  # create an array of zeros
139 # 1. Time/Phase shift the two channel:
140 L=concatenate((x,z)) # we add zeros after the 'x' signal to create Left channel
141 R=concatenate((z,y)) # we add zeros before the 'y' signal to create Right channel
142 A=zeros([len(L),2])  # we now create the array to store 'x' and 'y' as L and R
143 A[:,0]=L  # we assign the Left channel
144 A[:,1]=R  # we assign the Right channel
145 # 2. Amplitude change:
146 A[:,0]=A[:,0]*1.2e-4 # we decrease the amplitude on the Left to avoid clipping
147 A[:,1]=A[:,1]*1.5e-4 # ampl. decrease on Right channel is more since it is delayed
148
149 # we write the file:
150 write('files/lecture8/melody_stereo.wav',Fs,A)
```

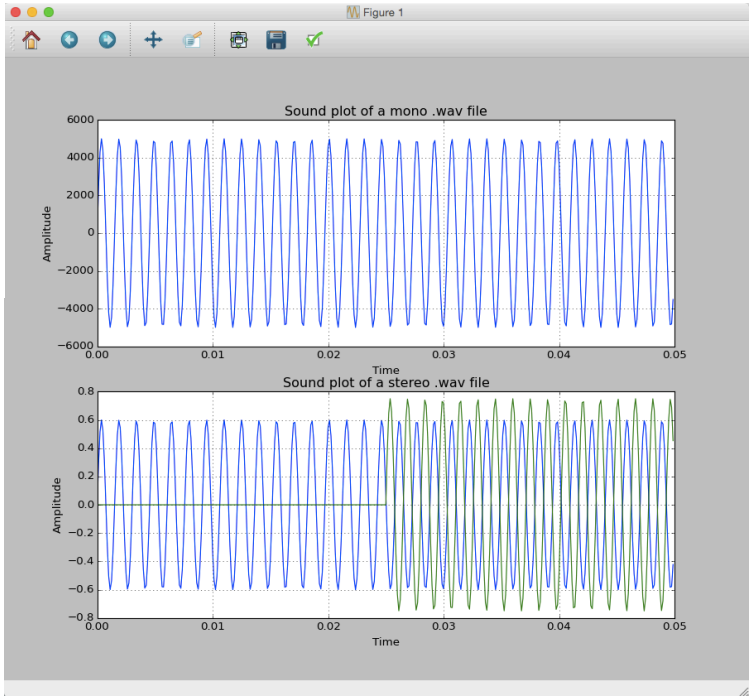
… lets hear the melody

Working with files

- Working with files – sound 5/5
  - we plot the mono and stereo music we just created

```
152 ## Lets plot the mono and pseudo-stereo sounds:
153 from pylab import linspace, plot, subplot, title, xlabel, ylabel, grid, pause
154
155 length = len(L)  # number of smaples in either channel 'L' (they are equal)
156 time = length/Fs # calculate the length of the .wav file in seconds
157 t = linspace(0,time,length) # create evenly spaced numbers between [0:time]
158
159 subplot(2,1,1)
160 plot(t[0:400],L[0:400]) # plot the first 400 samples from the mono signal 'L'
161 title('Sound plot of a mono .wav file')
162 xlabel('Time'); ylabel('Amplitude'); grid(True)
163
164 subplot(2,1,2)
165 plot(t[0:400],A[0:400]) # plot the first 400 samples from the stereo signal 'A'
166 title('Sound plot of a stereo .wav file')
167 xlabel('Time'); ylabel('Amplitude'); grid(True)
168 pause(1)
```

manipulating
.wav files



Working with files

- Working with files – playing sounds with PyAudio:

PyAudio: for playing and recording sounds with Python

Installation steps for PyAudio on Mac OS X:

Step 1: execute this line in your terminal:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Step 2: execute this line in your terminal:

```
brew install portaudio
```

Step 3: execute this line in your Python setup (PyZo):

```
pip install pyaudio
```

Step 4: write your routine to play the audio and save it as a .py module

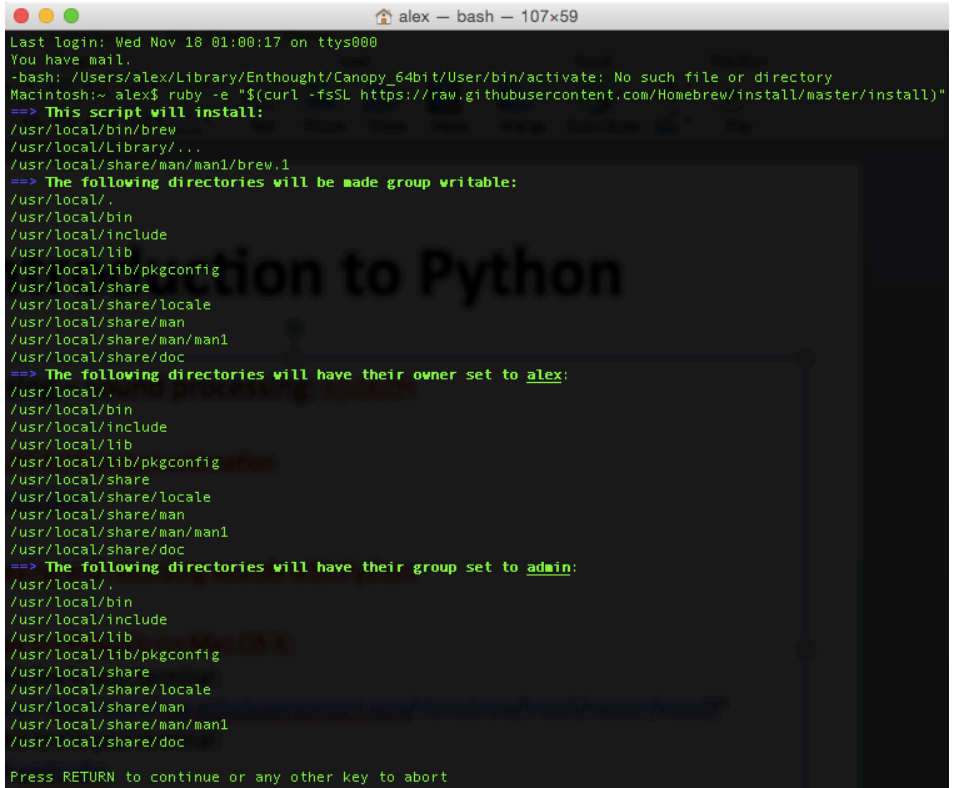
Working with files

- Working with files – playing sounds with PyAudio:

Installation steps for PyAudio on Mac OS X:

Step 1: execute this line in your terminal:

ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"



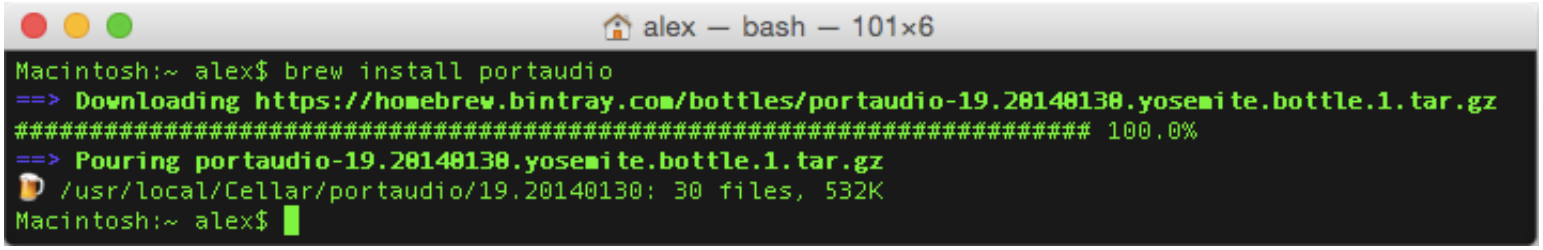
Working with files

- Working with files – playing sounds with PyAudio:

Installation steps for PyAudio on Mac OS X:

Step 2: execute this line in your terminal:

brew install portaudio



Step 3: execute this line in your Python setup (Pyzo):

pip install pyaudio

Step 4: write your routine to play the audio and save it as a .py module

More info for other systems: http://people.csail.mit.edu/hubert/pyaudio/#binaries

Working with files

- Working with files – playing sounds with PyAudio:

play_wave.py

```
1  import pyaudio
2  import wave
3  import sys
4
5  class AudioFile:
6      window = 1024    # create a window width for the sound samples (data chunk)
7
8      def __init__(self, file):
9          """ Init audio stream """
10         self.wf = wave.open(file, 'rb') # open a sound file for reading
11         self.p = pyaudio.PyAudio()      # create a PyAudio object
12         self.stream = self.p.open(      # opent the stream
13             format = self.p.get_format_from_width(self.wf.getsampwidth()),
14             channels = self.wf.getnchannels(),  # read the number of channels
15             rate = self.wf.getframerate(),      # get the Fs rate
16             output = True
17         )
18
19     def play(self):
20         """ Play entire file """
21         data = self.wf.readframes(self.window)      # read 1024 sample window 1st time
22         while data != '':
23             self.stream.write(data)
24             data = self.wf.readframes(self.window)  # read all windows 1-by-1
25
26     def close(self):
27         """ Graceful shutdown """
28         self.stream.close()             # just close the stream and release the object
29         self.p.terminate()
30
31 # Usage example for pyaudio
32 a = AudioFile(input("Please enter 'name.wav' here: > ")) # input like this: > file.wav
33 a.play()
34 a.close()
```

Working with files

- Working with files – playing sounds with PyAudio:

play_wave.py

```
import pyaudio
import wave
import sys

class AudioFile:
    window = 1024    # create a window width for the sound samples (data chunk)
```

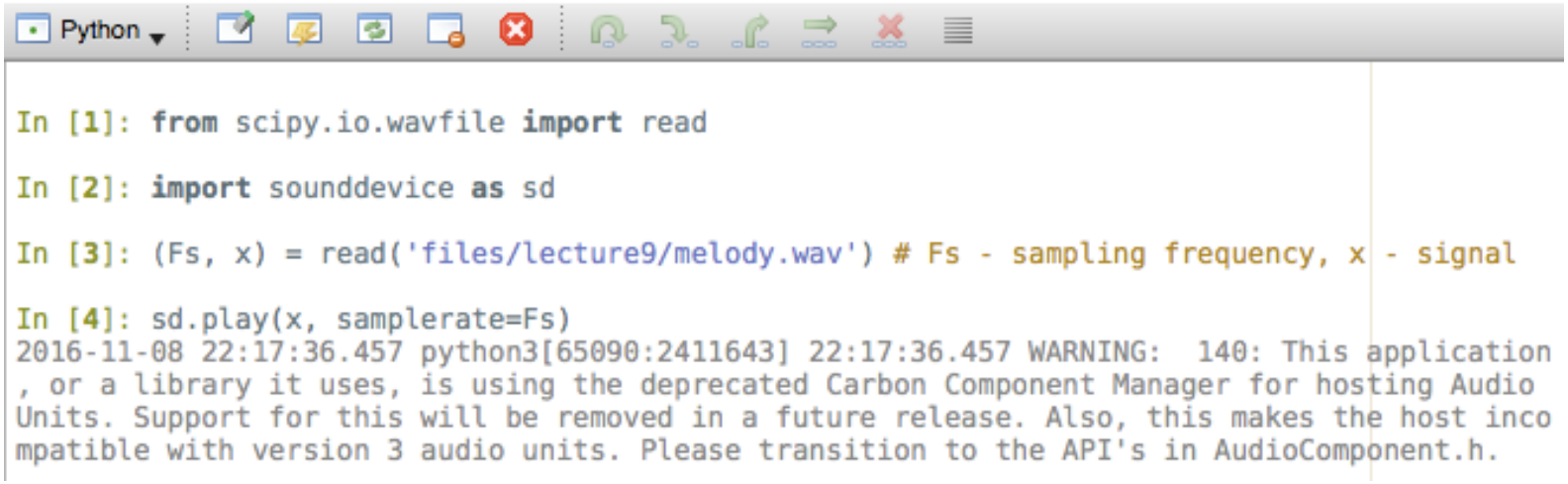
```
(<module>)>>> a.wf.getframerate()
8000

(<module>)>>> a.wf.getnchannels()
1

(<module>)>>> a.wf.getcompname()
'not compressed'

(<module>)>>> a.wf.getparams()
_wave_params(nchannels=1, sampwidth=2, framerate=8000, nframes=16000, comptype='NONE', compname='not compressed')
```

```
        while data != '':
            self.stream.write(data)
            data = self.wf.readframes(self.window)  # read all windows 1-by-1

    def close(self):
        """ Graceful shutdown """
        self.stream.close()             # just close the stream and release the object
        self.p.terminate()

# Usage example for pyaudio
a = AudioFile(input("Please enter 'name.wav' here: > ")) # input like this: > file.wav
a.play()
a.close()
```

Working with files

- Working with files – playing sounds with sounddevice:

  sounddevice:
  - is much easier to install and use as there is no need for you to create your play module
  - … however depending on your setup it might give you a warning message of depricated components so it may not run



Working with files

- Working with files – recording sounds with sounddevice:

sounddevice:

```
177 ## Working with files – recording sounds with sounddevice 1/2:
178 from scipy.io.wavfile import read
179 import sounddevice as sd
180
181 # Set the sampling frequency:
182 Fs = 8000 # kHz
183
184 # Duration of our recoring in 'sec' will be:
185 duration = 3
186
187 print('Begin talking to the microphone... for %s sec.' %duration)
188 # To record audio data from your sound device into a NumPy array, use:
189 recorded_audio_1 = sd.rec(int(duration * Fs), samplerate=Fs, channels=2, blocking=True)
190
191 # We can now listen to our recording:
192 sd.play(recorded_audio_1, Fs)
193 print('Playing your recording back...')
```

Note: make sure you use 'blocking=True', so that recording has finished and avoid unpleasant sounds

Working with files

- Working with files – recording sounds with sounddevice:

sounddevice:

```
195 ## Working with files – recording sounds with sounddevice 2/2:
196 from scipy.io.wavfile import read
197 import sounddevice as sd
198
199 # Set the sampling frequency:
200 Fs = 8000 # kHz
201
202 # Duration of our recoring in 'sec' will be:
203 duration = 3
204
205 # Let set our default parameters:
206 sd.default.samplerate = Fs
207 sd.default.channels = 2     # define default channels = stereo signal
208
209 print('Begin talking to the microphone... for %s sec.' %duration)
210 # By default, the recorded array has the data type 'float32', but we can change that:
211 recorded_audio_2 = sd.rec(duration * Fs, dtype='float64', blocking=True)
212
213 # We can now listen to our recording:
214 sd.play(recorded_audio_2, Fs)
215 print('Playing your recording back...')
```

Linear Systems

- Linear systems
  - Solving linear systems
    - to compute the solution vector of a linear system with any number of equations and unknowns, all we have to do is construct the right hand-side vector and left hand-side matrix with coefficients
    - lets consider the system:

$$7x + 5y - 4z = 15$$
$$4x - 9y + 6z = 12$$
$$3x + 8y + 2z = 10$$

    - the solution vector can be found by using the matrix inverse:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 & 5 & -4 \\ 4 & -9 & 6 \\ 3 & 8 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 15 \\ 12 \\ 10 \end{bmatrix} = \begin{bmatrix} 2.39 \\ 0.18 \\ 0.67 \end{bmatrix}$$

    - now lets find the solution using Python
    - we will use scipy.linalg as it is faster than numpy.ndarray

Linear Systems

- Linear systems
  - Solving linear systems

dot product:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

we will use linalg.solve as it is faster than inv.dot

```
Python
In [85]: # Solving linear systems:

In [86]: import numpy as npy

In [87]: from scipy import linalg

In [88]: # Solving the linear system:

In [89]: # 7x + 5y - 4z = 15

In [90]: # 4x - 9y + 6z = 12

In [91]: # 3x + 8y + 2z = 10

In [92]: e = npy.array([[7,5,-4],[4,-9,6],[3,8,2]]) # left hand-side matrix 'e':

In [93]: e
Out[93]:
array([[ 7,  5, -4],
       [ 4, -9,  6],
       [ 3,  8,  2]])

In [94]: f = npy.array([[15],[12],[10]]) # we create the right hand-side vector 'f'

In [95]: f
Out[95]:
array([[15],
       [12],
       [10]])

In [96]: linalg.inv(e).dot(f) # we solve the system using 'linalg.inv'
Out[96]:
array([[ 2.39814815],
       [ 0.18209877],
       [ 0.67438272]])

In [97]: %timeit linalg.inv(e).dot(f) # we calculate how fast it is (it is slower)
10000 loops, best of 3: 72 µs per loop

In [98]: npy.linalg.solve(e,f) # we solve the system using 'linalg.solve'
Out[98]:
array([[ 2.39814815],
       [ 0.18209877],
       [ 0.67438272]])

In [99]: %timeit npy.linalg.solve(e,f) # we calculate how fast it is (it is faster)
10000 loops, best of 3: 38.5 µs per loop
```

Singular Value Decomposition

- SVD
  - Singular Value Decomposition svd
  - It is a great tool for dimensionality reduction

Can be used in:
- Image compression
- Computer vision (using PCA - SVD)
- Signal processing
- Fitting solutions
- Statistics

```
Python
In [100]: # More advanced features:

In [101]: # Using svd to obtain a Singular Value Decomposition:

In [102]: e
Out[102]:
array([[ 7,  5, -4],
       [ 4, -9,  6],
       [ 3,  8,  2]])

In [103]: a = e + npy.diag([1, 1, 1]) # an array of singular real, non-negative values

In [104]: a
Out[104]:
array([[ 8,  5, -4],
       [ 4, -8,  6],
       [ 3,  8,  3]])

In [105]: u, spectrum, v = linalg.svd(a) # Singular Value Decomposition (a == u*spectrum*v)

In [106]: spectrum # is the resulting spectrum array
Out[106]: array([ 13.42424973,   9.1261769 ,   6.28509461])

In [107]: s = npy.diag(spectrum)  # 's' is a matrix with non-zeros in its main diagonal

In [108]: s
Out[108]:
array([[ 13.42424973,   0.        ,   0.        ],
       [  0.        ,   9.1261769 ,   0.        ],
       [  0.        ,   0.        ,   6.28509461]])

In [109]: u   # ndarray: unitary matrix having left singular vectors as columns
Out[109]:
array([[-0.61451975, -0.5313394 , -0.58313285],
       [ 0.58595351, -0.80234418,  0.1135883 ],
       [-0.52822719, -0.27188648,  0.80439653]])

In [110]: v  # ndarray: unitary matrix having right singular vectors as rows
Out[110]:
array([[-0.30966539, -0.89286512,  0.32695447],
       [-0.9068147 ,  0.1738915 , -0.38399069],
       [-0.28599729,  0.41539575,  0.86351139]])

In [111]: svd_matrix = u.dot(s).dot(v) # to recompose the original matrix

In [112]: svd_matrix
Out[112]:
array([[ 8.,  5., -4.],
       [ 4., -8.,  6.],
       [ 3.,  8.,  3.]])
```

Linear algebra operations

- Linear algebra operations
  - there is so much more you can do:
    - determinant of a matrix can be found by using linalg.det()
    - computing norms
    - solving linear least-squares problems
    - finding eigenvalues and eigenvectors
    - singular value decomposition SVD (among other decompositions)
    - exponential and logarithm functions
    - trigonometric functions
    - matrix functions and many other special functions

more info: http://docs.scipy.org/

The Fast Fourier Transform

- The Fast Fourier Transform – quick intro
  - FFT is the faster implementation of the DFT
  - FFT is the basis for frequency analysis that converts any signal in time to frequency domain
  - fft is the Fast Fourier Transform (FFT) coverts time-domain signals to frequency-domain
  - ifft is the Inverse Fast Fourier Transform (IFFT) and coverts frequency to time-domain
  - the Fourier transform is represented like this (ex: an audio signal):

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}}$$

where: $[x_0, \ldots, x_{N-1}]$ are complex conjugate numbers and $[k=0, \ldots, N-1]$

  - the most commonly used FFT is the Cooley–Tukey algorithm

The Fast Fourier Transform

- The Fast Fourier Transform – quick intro
  - FFT is the faster implementation of the DFT
  - FFT is the basis for frequency analysis that converts any signal in time to frequency domain
  - fft is the Fast Fourier Transform (FFT) coverts time-domain signals to frequency-domain
  - ifft is the Inverse Fast Fourier Transform (IFFT) and coverts frequency to time-domain
  - the Fourier transform is represented like this (ex: an image):

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\imath 2\pi (\frac{ki}{N} + \frac{lj}{N})}$$

where: where *f(a,b)* is the image in the spatial domain and *F(k,l)* corresponds to each pixel

  - the most commonly used FFT is the Cooley–Tukey algorithm

The Fast Fourier Transform

- The Fast Fourier Transform

Example:

- notice that we only import what we need

- we add three simple tones to create one complex tone

- to go from time domain to frequency domain we use fft in two ways

- each frequency bin represents frequency in [Hz] ->

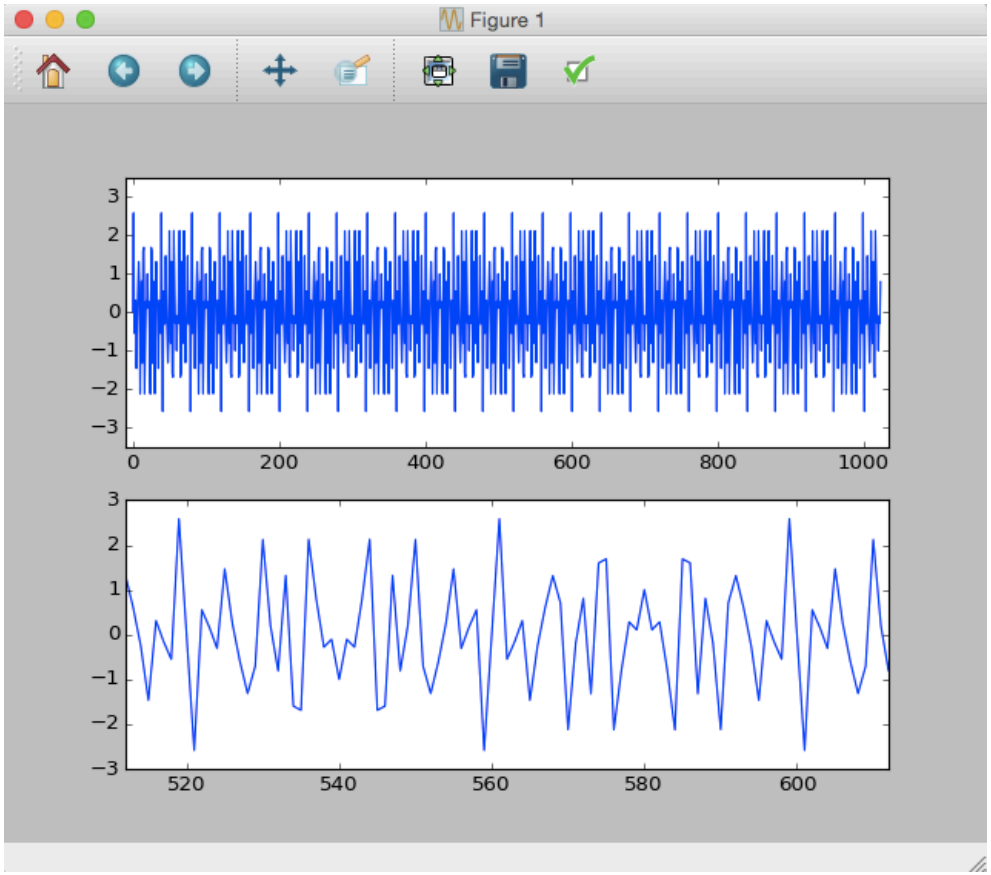
```
243  # FFT example:
244  from numpy.fft import rfft
245  from scipy import arange, sin, pi, fft, real ,imag, log10
246  from scipy.io.wavfile import write
247  from matplotlib.pyplot import figure, plot, subplot, axis, grid
248  from pylab import xticks, yticks, xlim, ylim, xlabel, ylabel, title
249
250  Fs=4000 # sampling frequency
251  a = arange(1024) # create a vector holding the number of bins
252  signal1 = sin(2*pi*a*(650/Fs))   # create a tone with frequency = 650Hz
253  signal2 = sin(2*pi*a*(1150/Fs))   # create a tone with frequency = 1.15kHz
254  signal3 = sin(2*pi*a*(1450/Fs))   # create a tone with frequency = 1.425kHz
255  signal4 = sin(2*pi*a*(1250/Fs))   # create a tone with frequency = 1.25kHz
256
257  # Create a complex tone:
258  signal = signal1 + signal2 + signal3
259
260  # Take the FFT of the complex signal:
261  freq_domain_npy = rfft(signal)  # using npy
262  freq_domain_cpy = fft(signal)   # using cpy
263  bin_val = Fs/len(a)    # calculate the value of each frequeny bin in [Hz]
264  bin_val # each bin in [Hz]
265  t = arange(1,Fs/2+2,bin_val)  # create a vector of frequency bins in [Hz]
266
267  # Save into a wav file:
268  write('files/lecture8/fft_file_example.wav',Fs,signal) # save to file
```

The Fast Fourier Transform

- The Fast Fourier Transform

Example:

```
270  # Plot the raw Time domain signal:
271  figure(1), subplot(2,1,1), plot(signal), xlim(-10, len(a)+10), ylim(-3.5,3.5)
272  subplot(2,1,2), plot(signal), xlim(len(a)/2,len(a)/2+100) # just a snipped of the signal
```

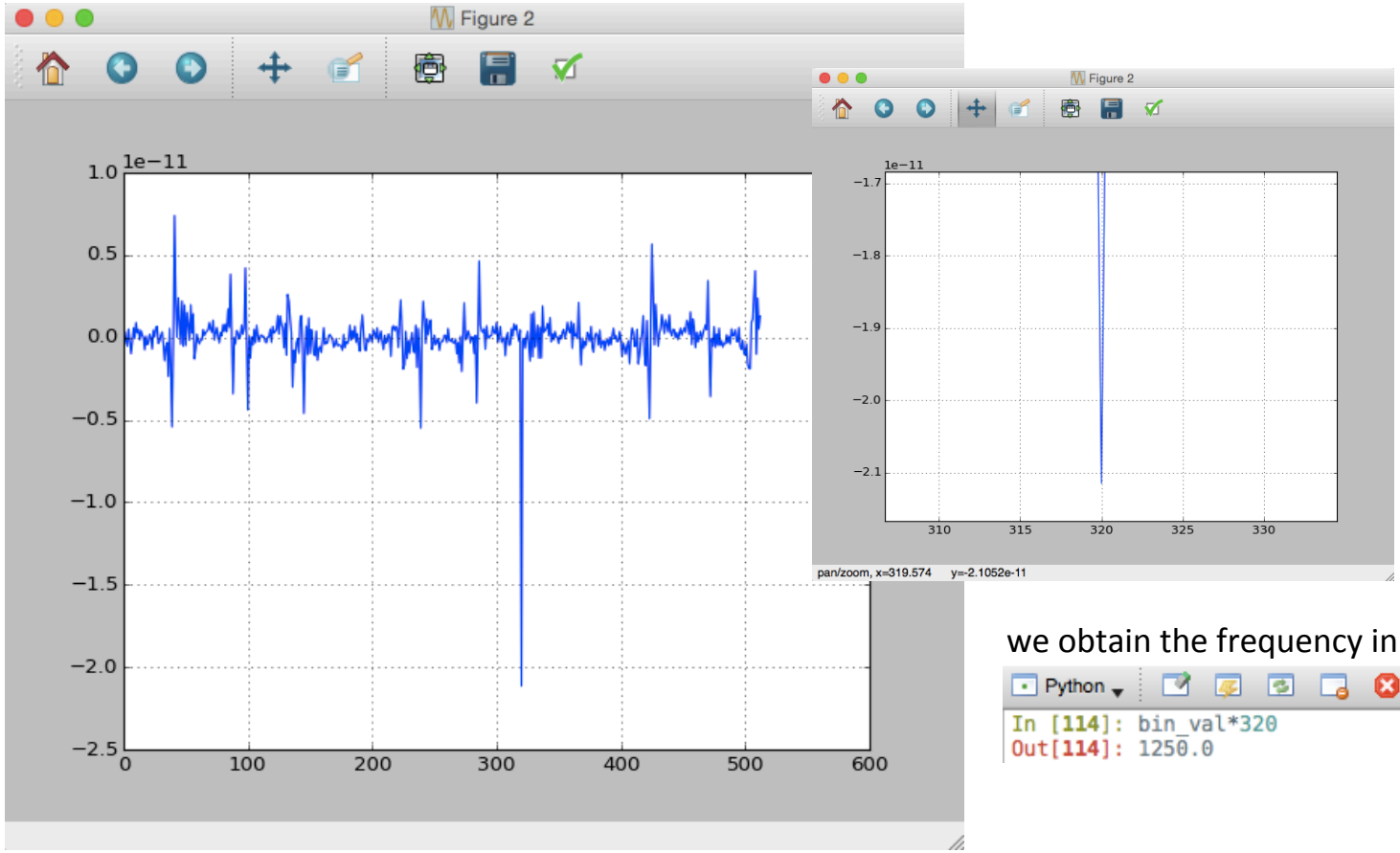


The Fast Fourier Transform

- The Fast Fourier Transform

Example:

```
274  # Plot the Frequency domain of 'signal4':
275  figure(2), plot(rfft(signal4)) # observe the quantization noise due to rounding errors
276  grid(True)
```



- observe the quantization noise due to rounding errors

we obtain the frequency in [Hz]:

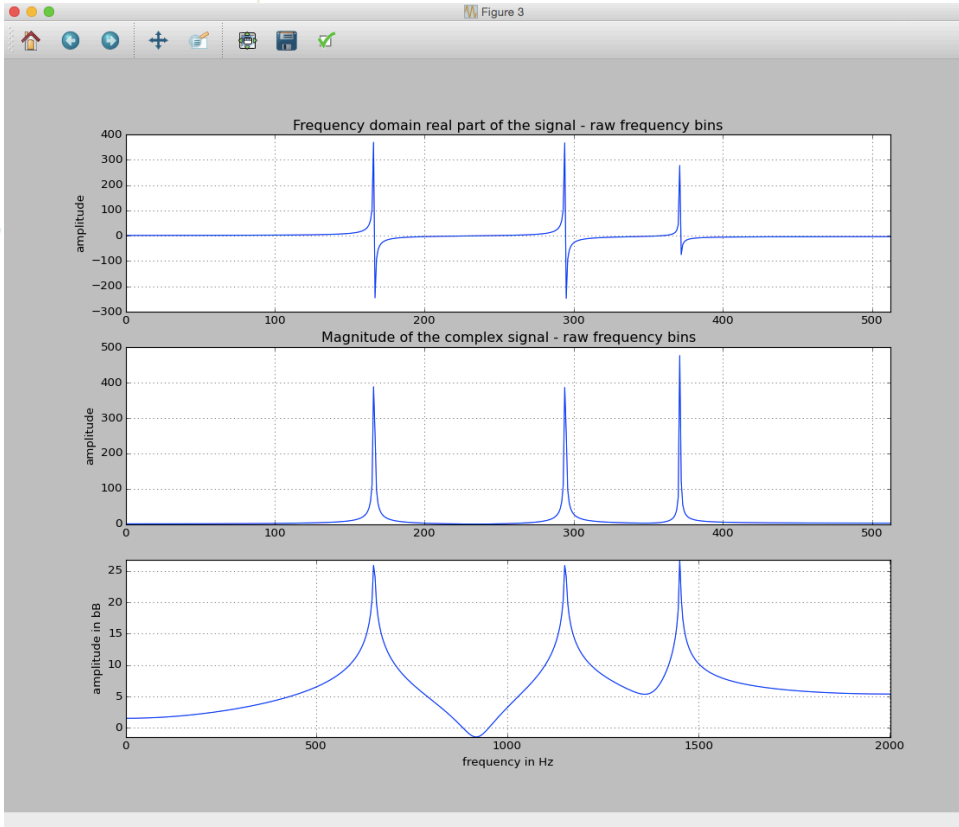
```
Python
In [114]: bin_val*320
Out[114]: 1250.0
```

The Fast Fourier Transform

- The Fast Fourier Transform

Example:

```
# Plot the Frequency domain signal using numpy:
figure(3)
subplot(3,1,1),
title('Frequency domain real part of the signal - raw frequency bins'),
plot(freq_domain_npy),
xlim(0,len(a)/2),
ylabel('amplitude'),
grid(True)

# Plot the magnitude of the complex signal output:
subplot(3,1,2),
plot(abs(freq_domain_cpy)),
title('Magnitude of the complex signal - raw frequency bins'),
xlim(0,len(a)/2),
ylabel('amplitude'),
grid(True)

# Plot in dB scale:
subplot(3,1,3),
plot(t,10*log10(freq_domain_npy)),
xlabel('frequency in Hz'),
ylabel('amplitude in bB'),
axis('tight'),
grid(True)
```



- notice the difference in the x scales

The Fast Fourier Transform

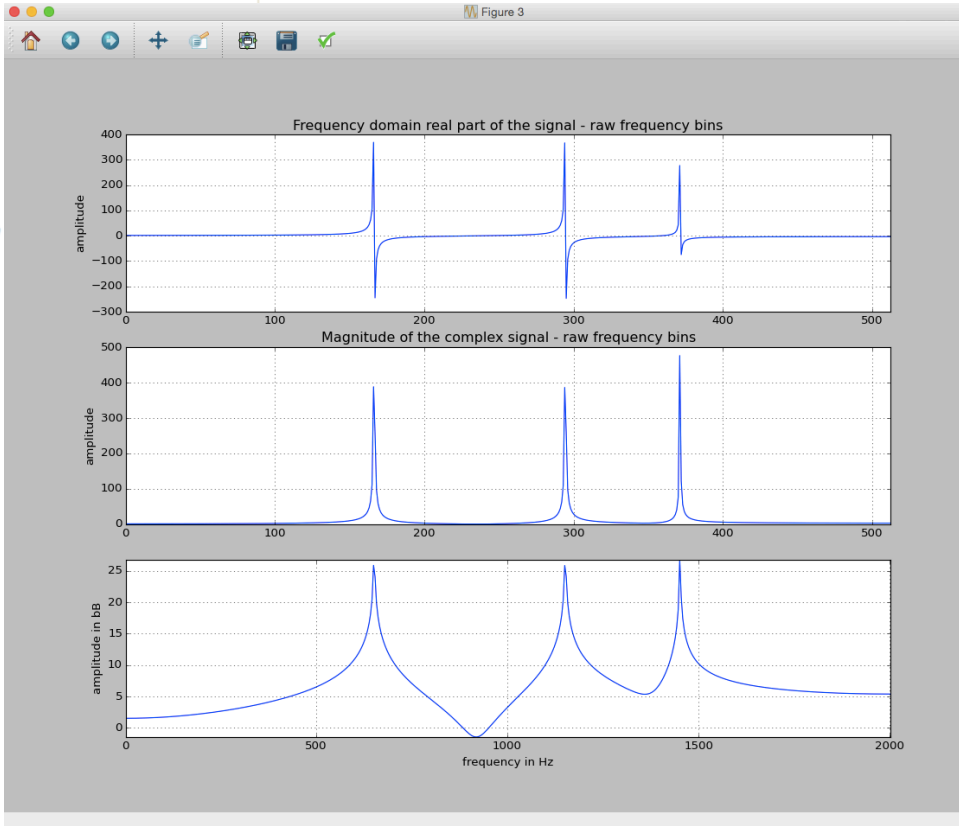
- The Fast Fourier Transform

Example:

```
278  # Plot the Frequency domain signal using numpy:
279  figure(3)
280  subplot(3,1,1),
281  title('Frequency domain real part of the signal - raw frequency bins'),
     plot(freq_domain_npy),
282  xlim(0,len(a)/2),
283  ylabel('amplitude'),
284  grid(True)
285
286  # Plot the magnitude of the complex signal output:
287  subplot(3,1,2),
288  plot(abs(freq_domain_cpy)),
289  title('Magnitude of the complex signal - raw frequency bins'),
290  xlim(0,len(a)/2),
291  ylabel('amplitude'),
292  grid(True)
293
294  # Plot in dB scale:
295  subplot(3,1,3),
296  plot(t,10*log10(freq_domain_npy)),
297  xlabel('frequency in Hz'),
298  ylabel('amplitude in bB'),
299  axis('tight'),
300  grid(True)
```

```
In [11]: bin_val*166
Out[11]: 648.4375

In [12]: bin_val*294
Out[12]: 1148.4375

In [13]: bin_val*371
Out[13]: 1449.21875
```

```
reate a vec
*(650/Fs))
*(1150/Fs))
*(1450/Fs))
*(1250/Fs))
```



- notice the difference in the x scales

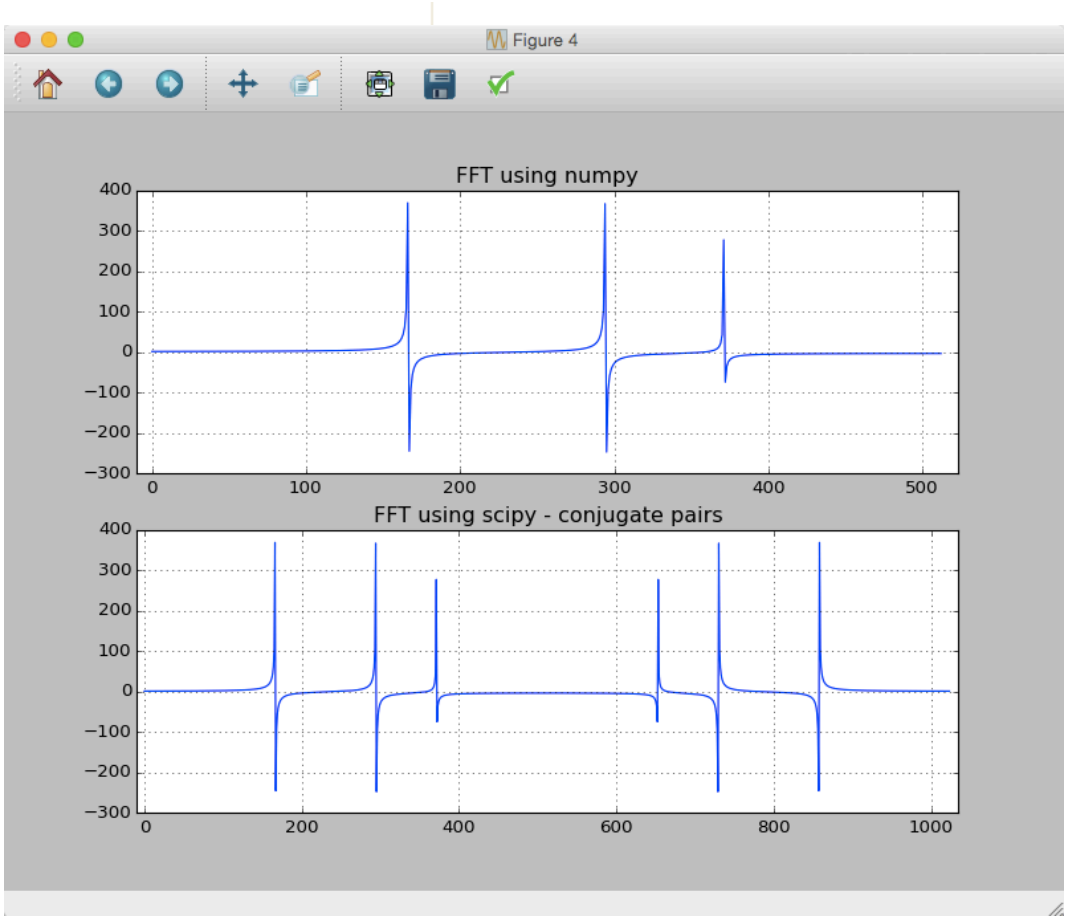
The Fast Fourier Transform

- The Fast Fourier Transform

Example:

```
302 # Plot the Frequency domain signal using numpy:
303 figure(4)
304 subplot(2,1,1),
305 plot(freq_domain_npy),
306 xlim(-10,len(t)+10),
307 title('FFT using numpy'),
308 grid(True)
309
310 # Plot the Frequency domain signal using scipy:
311 subplot(2,1,2),
312 plot(freq_domain_cpy),
313 xlim(-10,len(freq_domain_cpy)+10),
314 title('FFT using scipy - conjugate pairs'),
315 grid(True)
384 pause(1)
```

- notice the difference between:
  - rfft from NumPy and
  - fft from Scipy



The Fast Fourier Transform

- The Fast Fourier Transform

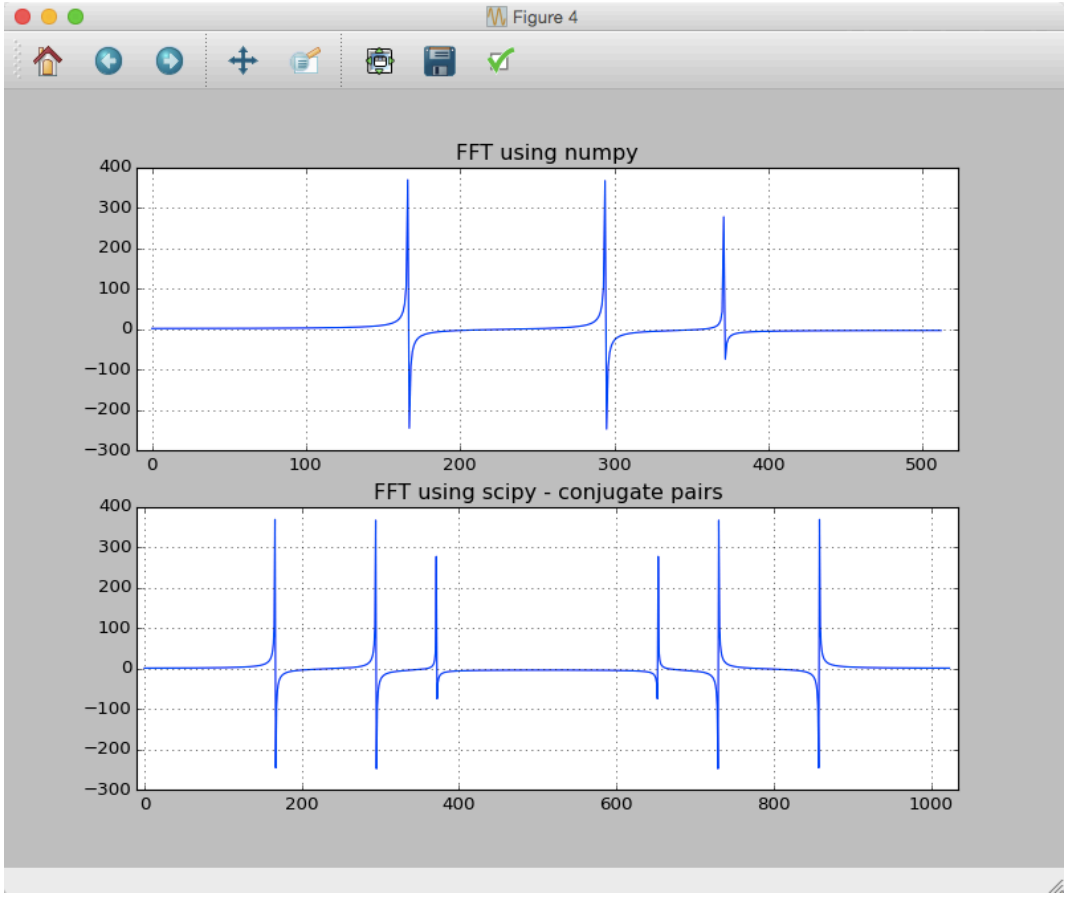
Example:

- the FFT of a real-valued input signal produces a conjugate symmetric result with positive frequencies and their negative counterparts

- so the negative frequencies are just mirrored duplicates of the positive frequencies

- therefore they can just be ignored when analyzing the result

- they are needed when we want to reconstruct the signal back to time domain using the inverse FFT the IFFT



The Fast Fourier Transform

- The Fast Fourier Transform

Example:

- the FFT of a real-valued input signal produces a conjugate symmetric result with positive frequencies and their negative counterparts

- so the negative frequencies are just mirrored duplicates of the positive frequencies

- therefore they can just be ignored when analyzing the result

- they are needed when we want to reconstruct the signal back to time domain using the inverse FFT the IFFT