Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN-Fakultät
Fachbereich Informatik
Arbeitsbereich SAV/BV (KOGS)

MMS-Übungen

Einführung in die Signalanalyse mit Python

Wintersemester 2015/16

Benjamin Seppke

Outline

- Introduction
- Presenting the Python programming language
- Signal analysis using NumPy and SciPy
- Visualization with matplotlib and the spyder IDE
- Summary

Outline

- Introduction
- Presenting the Python programming language
- Signal analysis using NumPy and SciPy
- Visualization with matplotlib and the spyder IDE
- Summary

Prerequisites (Software)

- Python (we use version 2.X *with X>5*)
  - http://www.python.org
- NumPy and SciPy (with PIL: http://www.pythonware.com/products/pil)
  - http://www.scipy.org
- matplotlib
  - http://matplotlib.org
- spyder IDE
  - http://code.google.com/p/spyderlib

Installing Python and packages

- **Linux**
  - All of the prerequisites should be installable by means of the package manager of the distribution of your choice.
- **Mac OS X**
  - Install the MacPorts package manager (http://www.macports.org) and use this to get all necessary packages.
- **Windows**
  - Python-(x,y) (http://code.google.com/p/pythonxy) contains all necessary packages in binary form and an installer.

Goals for today...

- Draw interest to another programming language, namely: ***Python***
- Motivation of an interactive Workflow („Spielwiese“)
- „Easy access” into practical image processing tasks using ***NumPy, SciPy, matplotlib and spyder***
- Finally: Give you the ability to solve the exercises of this course

Outline

- Introduction
- Presenting the Python programming language
- Signal analysis using NumPy and SciPy
- Visualization with matplotlib and the spyder IDE
- Summary

Introducing Python

The following introduction is based on the official „Python-Tutorial“

http://docs.python.org/tutorial/index.html

Python

„Python is an easy to learn, powerful programming language. [...] Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.“

„By the way, the language is named after the BBC show "Monty Python's Flying Circus" and has nothing to do with reptiles.“

The Python Tutorial, Sep. 2010

Why another language?
Why Python?

- Interactive: no code/compile/test-cycle!
- A lot of currently needed and easy accessible functionality compared with traditional scripting languages!
- Platform independent and freely available!
- Large user base and good documentation!
- Forces compactness and readability of programs by syntax!
- Some say: can be learned in 10 minutes...

Getting in touch with Python (2.X)

- All of this tutorial will use the interactive mode:
  - Start the interpreter: **`python`**
  - Or, an advanced interpreter: **`ipython`**

1. Example:

```
> python
Python 2.7 (#1, Feb 28 2010, 00:02:06)
Type "help", "copyright", "credits" or "license" for more information.
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
...
Be careful not to fall off!
```

Data types – numbers (1)

- Python supports integer, floating point and complex valued numbers by default:

```
>>> 2+2
4
>>> # This is a comment
... 2+2
4
>>> # Integer division returns the floor:
... 7/3
2
>>> 7.0 / 2  # but this works...
3.5
>>> 1.0j * 1.0j
(-1+0j)
```

Data types – numbers (2)

- Assignments and conversions:

```
>>> a=3.0+4.0j
>>> float(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can't convert complex to float; use abs(z)
>>> a.real
3.0
>>> a.imag
4.0
>>> abs(a)  # sqrt(a.real**2 + a.imag**2)
5.0
```

Special variables

- Example: last result „_“ (only in interactive mode):

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

- Many more in ipython!

Data types – strings

- Sequences of chars (like e.g. in C), but immutable!

```
>>> word = 'Help' + 'A'
>>> word
'HelpA'
>>> '<' + word*5 + '>'
'<HelpAHelpAHelpAHelpAHelpA>'
>>> 'str' 'ing'                    #  <-  This is ok
'string'
>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[2:]    # Everything except the first two characters
'lpA'
```

Data types – lists

- Lists may contain different types of entries at once!
- First element has index: 0, last element: length-1.

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
>>> a[0]
'spam'
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
```

The first program (1)

- Counting Fibonacci series

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while b < 10:
...     print b
...     a, b = b, a+b
...
1
1
2
3
5
8
```

The first program (2)

- Counting Fibonacci series
  (with a colon after the print)

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while b < 10:
...     print b,
...     a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Conditionals – if

- Divide cases in if/then/else manner:

```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...      x = 0
...      print 'Negative changed to zero'
... elif x == 0:
...      print 'Zero'
... elif x == 1:
...      print 'Single'
... else:
...      print 'More'
...
More
```

Control flow – for (1)

- Python‘s for-loop:

```
>>> # Measure the length of some strings:
... a = ['two', 'three', 'four']
>>> for x in a:
...     print x, len(x)
...
two 3
three 5
four 4
```

- is actually a for-each-loop!

Control flow – for (2)

- What about a counting for loop?
- Quite easy to get:

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i, val in enumerate(a):
...     print i, val
...
0 Mary
1 had
2 a
3 little
4 lamb
```

Defining functions (1)

- Functions are one of the most important way to abstract from problems and to design programs:

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Defining functions (2)

- Functions are (themselves) just Python symbols!

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

- No explicit return value needed (default: “None”)

```
>>> fib(0)
>>> print fib(0)
None
```

Defining functions (3)

- Fibonacci series with a list of numbers as return value:

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Function argument definitions (1)

- Named default arguments:

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError('refuse user')
        print complaint
```

Function argument definitions (2)

- Calling strategy in more detail:

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print "-- This parrot wouldn't", action,
    print "if you put", voltage, "volts through it."
    print "-- Lovely plumage, the", type
    print "-- It's", state, "!"


parrot(1000)
parrot(action = 'VOOOOOM', voltage = 1000000)
parrot('a thousand', state = 'pushing up the daisies')
parrot('a million', 'bereft of life', 'jump')
```

Modules

- If you have saved this as „fibo.py“:

```
# Fibonacci numbers module
def fib(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

…you have already written your first Python module.

Call it using:

```
>>> import fibo
>>> fibo.fib(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Outline

- Introduction
- Presenting the Python programming language
- Signal analysis using NumPy and SciPy
- Visualization with matplotlib and the spyder IDE
- Summary

Signal analysis using NumPy and SciPy

Unfortunately, it is not possible to give a complete introduction in either NumPy or SciPy.
The image processing introduction is based on:
http://scipy-lectures.github.io/advanced/image_processing

More material regarding NumPy can e.g. be found at:
http://numpy.scipy.org
A good beginner's tutorial is provided at:
http://www.scipy.org/Tentative_NumPy_Tutorial

NumPy

SciPy.org

Discrete Signals as *efficient* arrays?!

- In many programming environments, like e.g. MatLab, signals are represented as (random access) arrays of different data types.
- Unfortunately, Python's built-in array is often neither flexible nor powerful enough for signal analysis
- Thus: Use NumPy arrays for signal representation.
- Idea of a first (very basic) workflow:
  - Load signals using csv- or NumPy binary formats
  - Analyze the signals using NumPy (and maybe SciPy)
  - Save the results using csv- or numpy binary formats.

NumPy at a glance

- *„NumPy is the fundamental package needed for scientific computing with Python. It contains among other things: a powerful N-dimensional array object […]“*
    - NumPy Homepage, 2010

- May have required a whole course on its own...
- Still growing scientific user community
- Reliable algorithms
- Quite fast, compared to commercial software implementations, like MatLab

Loading and storing signals

- Generate a noisy rect signal and save it:

```
>>> import numpy as np
>>> sig = np.zeros(3000)
>>> sig[1000:2000]=1
>>> sig = sig + np.random.normal(0, 0.01, 3000)
>>> np.save(„FILENAME.npy‟, sig)    # numpy binary format
>>> np.savetxt(„FILENAME.csv‟, sig) # comma-separated-values
```

- Read the signal from file system

```
>>> import numpy as np
>>> sig_npy = np.load(„FILENAME.npy‟)
>>> sig_csv = np.loadtxt(„FILENAME.csv‟)
```

- Attention: CSV export may reduce (float) accurancy!

NumPy signal representation (1)

- Note that numpy arrays are fixed to one unique data type, which is **`float64`** a.k.a. **`double`** by default

```
...
>>> def_sig = np.zeros(1000)
>>> def_sig.dtype
dtype(‚float64‘)
>>> int_sig = np.zeros(1000, dtype=np.uint8) # values from 0..255
```

- Pay attention to the datatype, especially when performing mathematical operations!

NumPy signal representation (2)

- Single vs. multidimensional signals:

```
...
>>> sig1D = np.zeros(1000)
>>> sig2D = np.zeros( (1000,2000) )
```

- Note: The shape is always passed as one parameter
  - Either a number (1D) or
  - A list for n-dimensional arrays
- Access on multidimensional arrays:

```
>>> sig2D[800, 1800]
0
```

```
>>> sig2D[1800, 800]
IndexError: index 1800 is out of bounds ...
```

NumPy slicing and index tricks

- Extract dimensions using slicing

```
>>> sig2D[:,0]          # first signal (fixing second dimension)

>>> sig2D[...,-1]       # last signal  (fixing second dimension)
```

- Extract sub-signals using index ranges:

```
>>> sig = np.zeros(3000)
>>> sig[1000:2000]=1
```

- **Attention**: NumPy often creates views and does not copy your data, when using index tricks!
→ Compare to Call-By-Reference Semantics

Basic signal analysis (1)

- Example: Adding energy / signals:

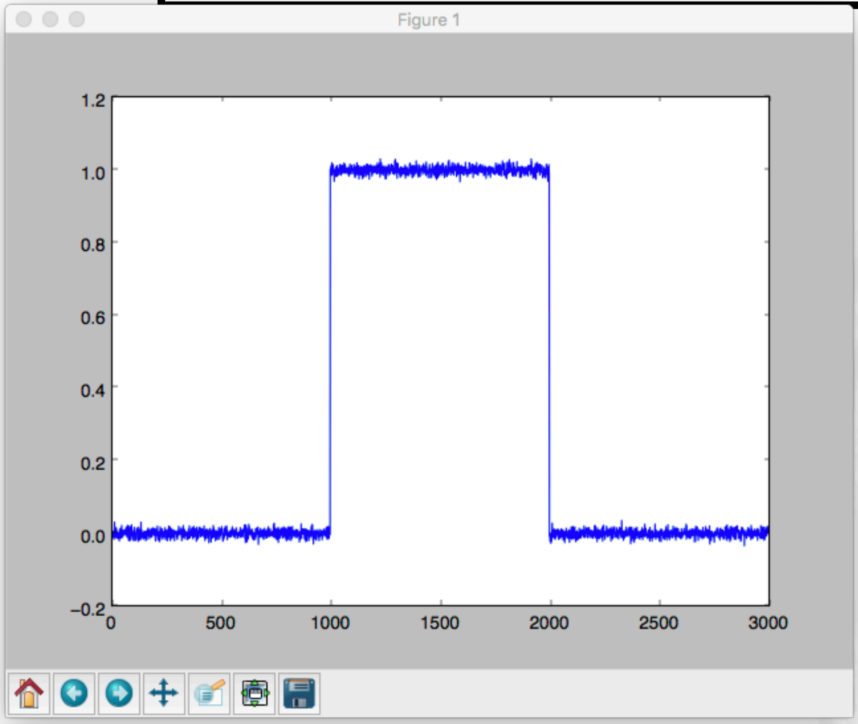
```
...
>>> res1  = 2 + sig
>>> res2 = sig + sig
>>> res2 = sig * sig
```

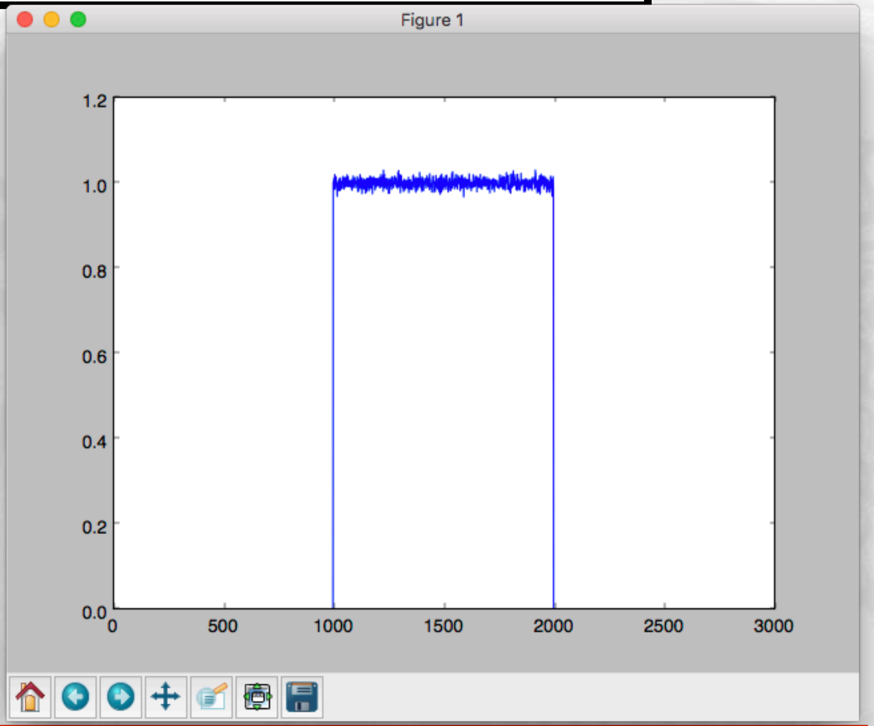
- Note:
  - Scalars and arrays (of compatible shape) can be combined
  - Basic arithmetic functions
  - Many more (advanced functions)
  - 2D-Arrays may also be interpreted as matrices, but compare to np.matrix class!

Basic signal analysis (2)

- Example: Threshold a signal (at a given amplitude):

```
...
>>> mask = sig < 0.5
>>> masked_sig = sig.copy()
>>> masked_sig[mask] = 0
```





Outline

- Introduction
- Presenting the Python programming language
- Signal analysis using NumPy and SciPy
- Visualization with matplotlib and the spyder IDE
- Summary

Visualization with matplotlib

"matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell...“

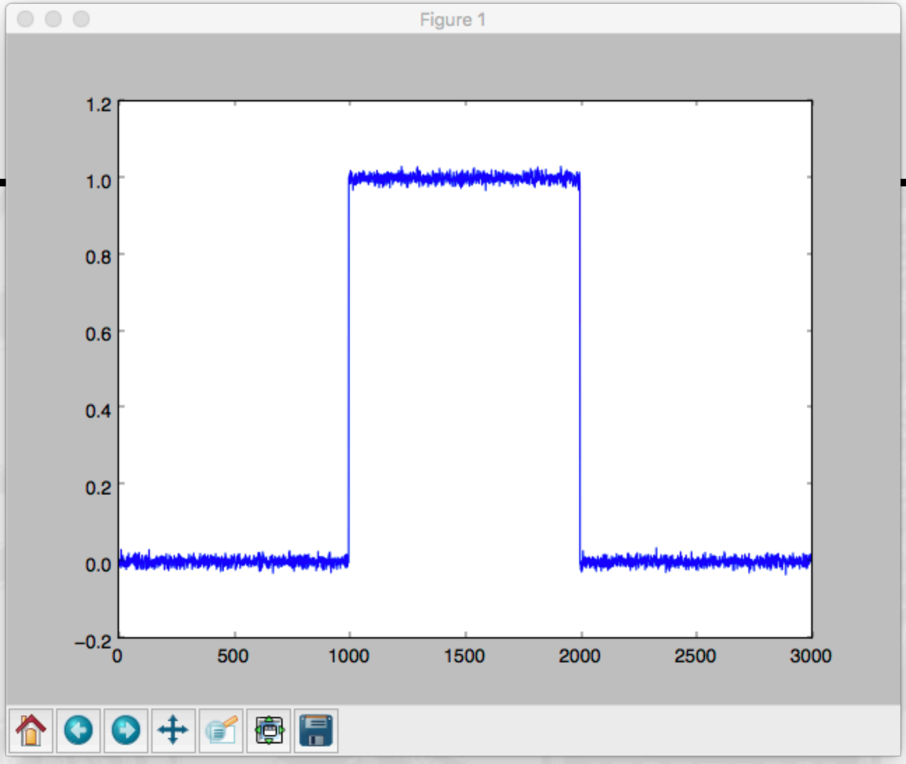
http://matplotlib.org, October 2013

This introduction is based on the matplotlib image tutorial:
http://matplotlib.org/users/image_tutorial.html

Showing signals interactively (1)

- Use matplotlib to show a signal plot:

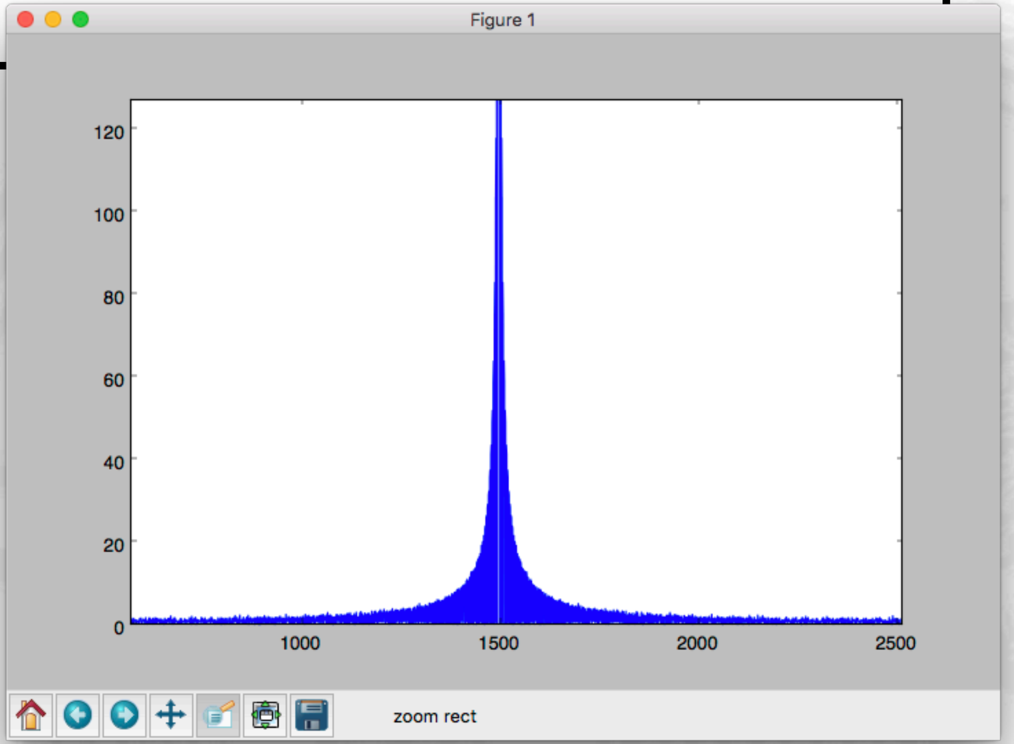
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> sig = np.load(„FILENAME.npy“)       # Load the signal

>>> sig_plot = plt.plot(sig)
>>> sig_plot.show()
```



Showing signals interactively (2)

- Show signal's (centered) magnitude spectrum:

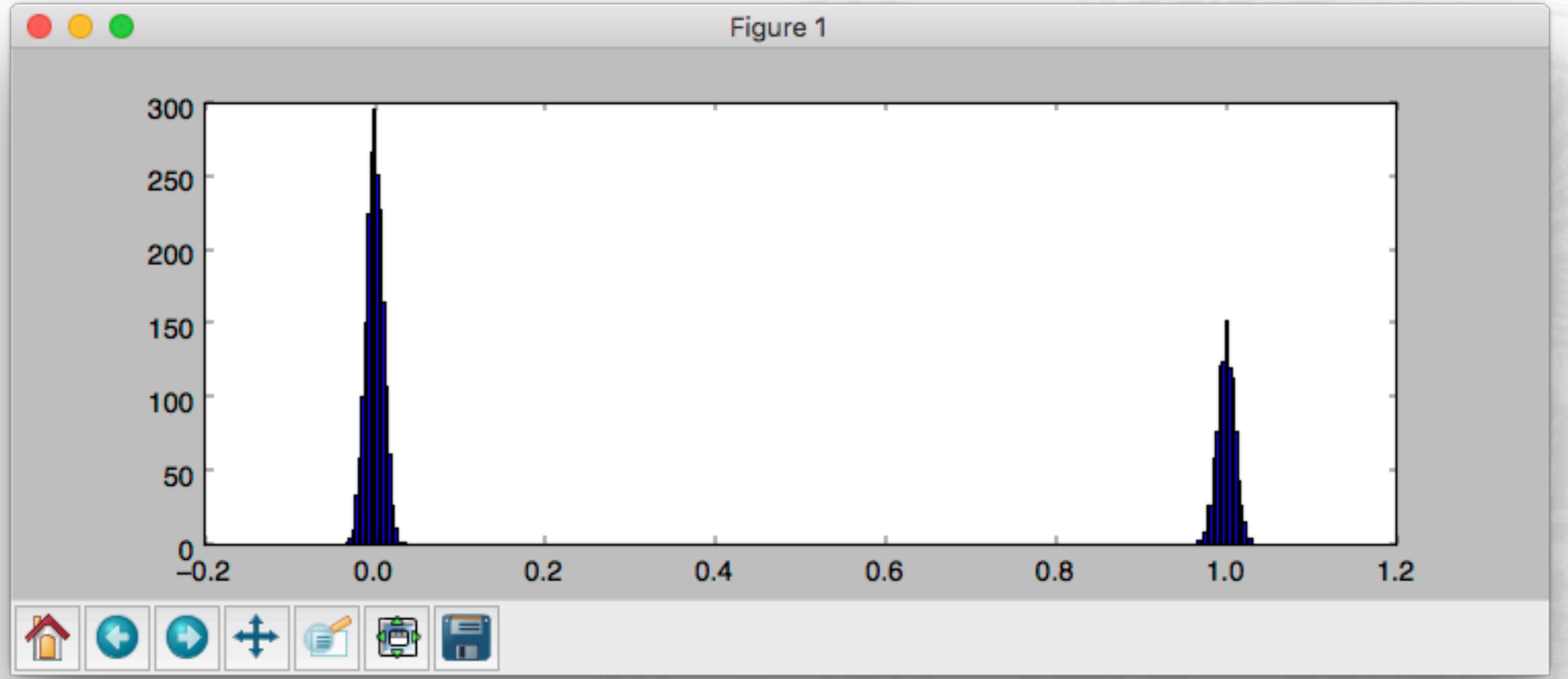
```
...
>>> centered_spectrum = np.fft.fftshift(np.fft.fft(sig))
>>> spec_plot = plt.plot(abs(centered_spectrum))
>>> spec_plot.show()
```



Histograms

- Use matplotlib to inspect the histogram:

```
...
>>> plt.hist(sig, 300) # collect values in 300 bins
>>> plt.show()
```



Working with the spyder IDE

„spyder (previously known as Pydee) is a powerful interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features.[...]

spyder lets you easily work with the best tools of the Python scientific stack in a simple yet powerful environment.[...]“

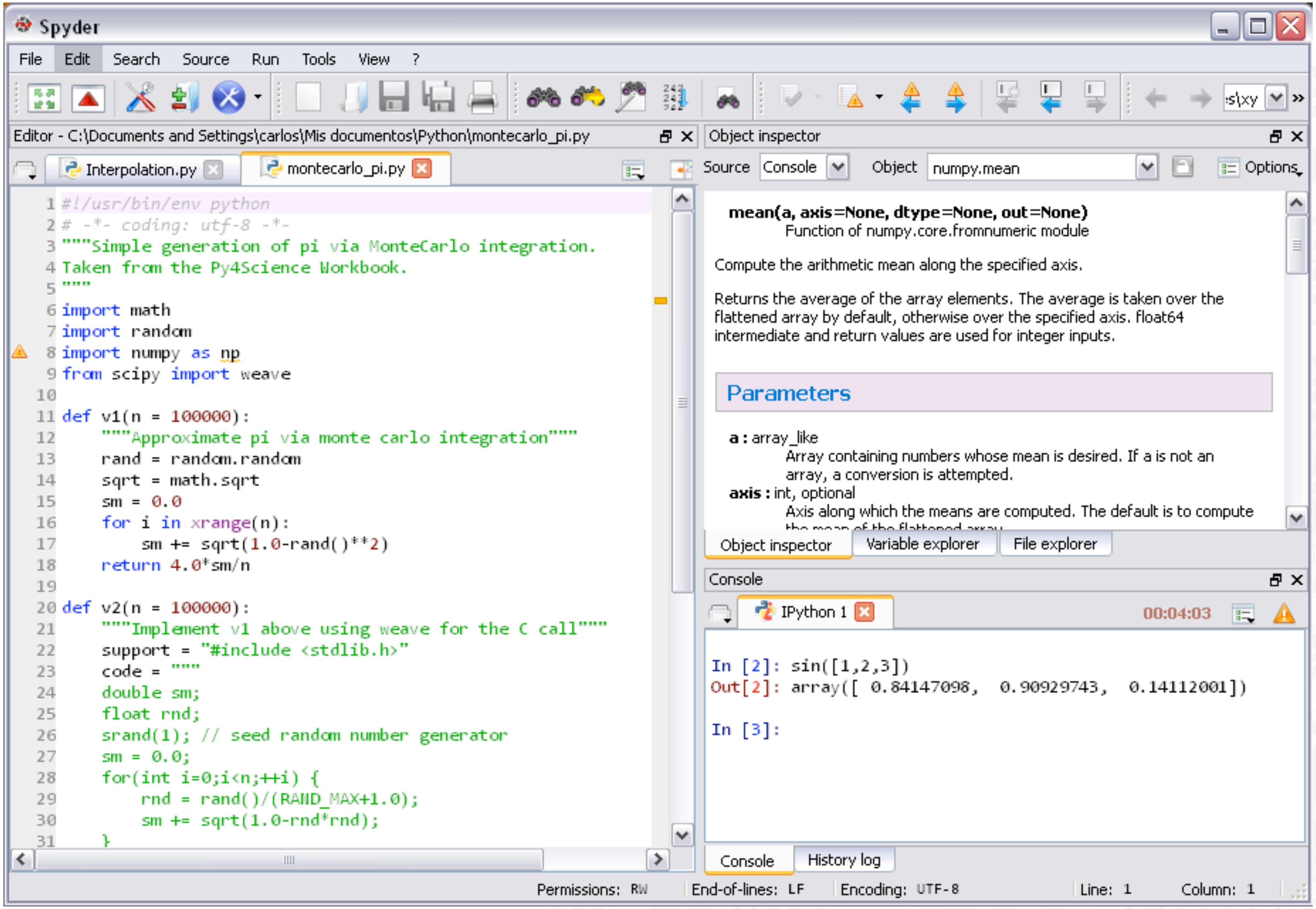
http://code.google.com/p/spyderlib, October 2013

The screenshots of this introduction have been taken from the spyder homepage.

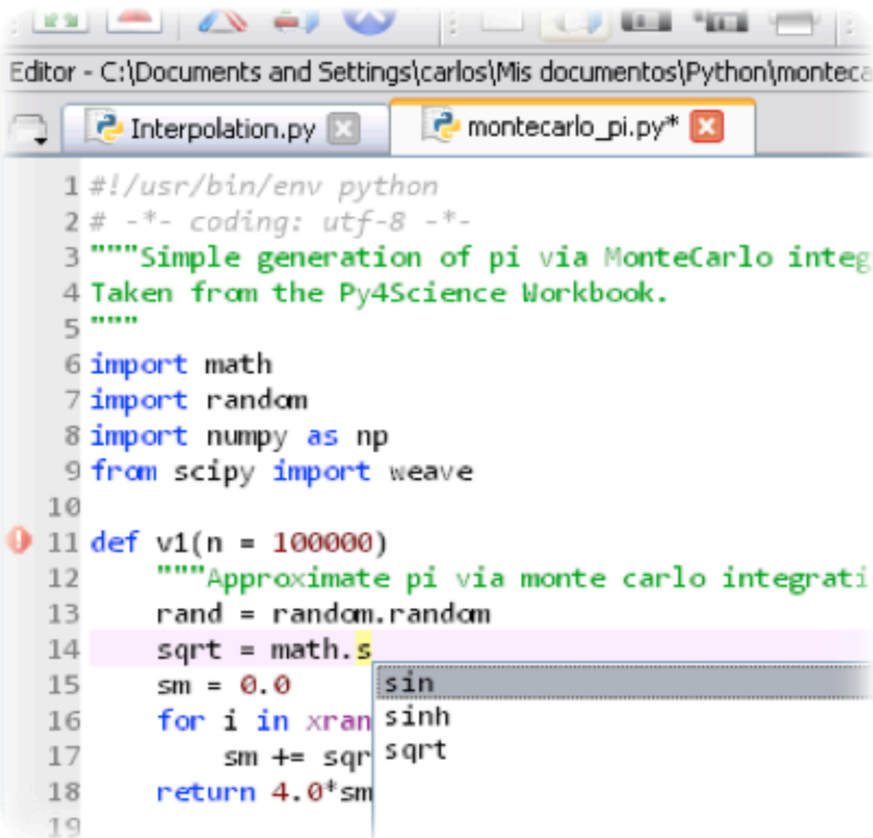
spyder2

Light, intuitive, simple, powerful

The spyder IDE



spyder - The editor



A powerful editor is a central piece of any good IDE. Spyder's editor has:

- Syntax coloring for Python, C/C++ and Fortran files
- Powerful dynamic code introspection features (powered by rope):
    - Code completion and calltips
    - Go to an object definition with a mouse click
- Class and function browser.
- Occurrence highlighting.
- To-do lists (TODO, FIXME, XXX).
- Get errors and warnings on the fly (provided by pyflakes)
- Breakpoints and conditional breakpoints to use with the python debugger (pdb).

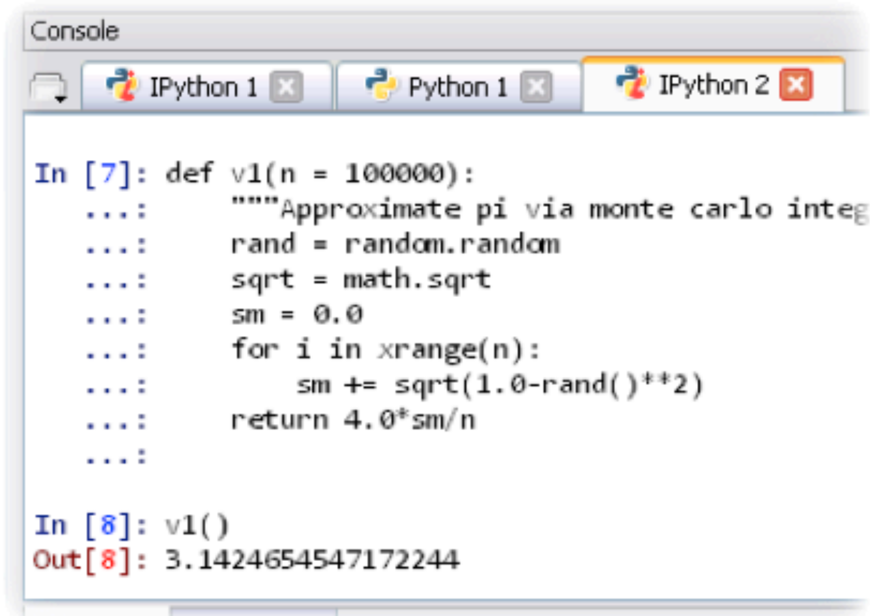
Learn More

spyder - The console

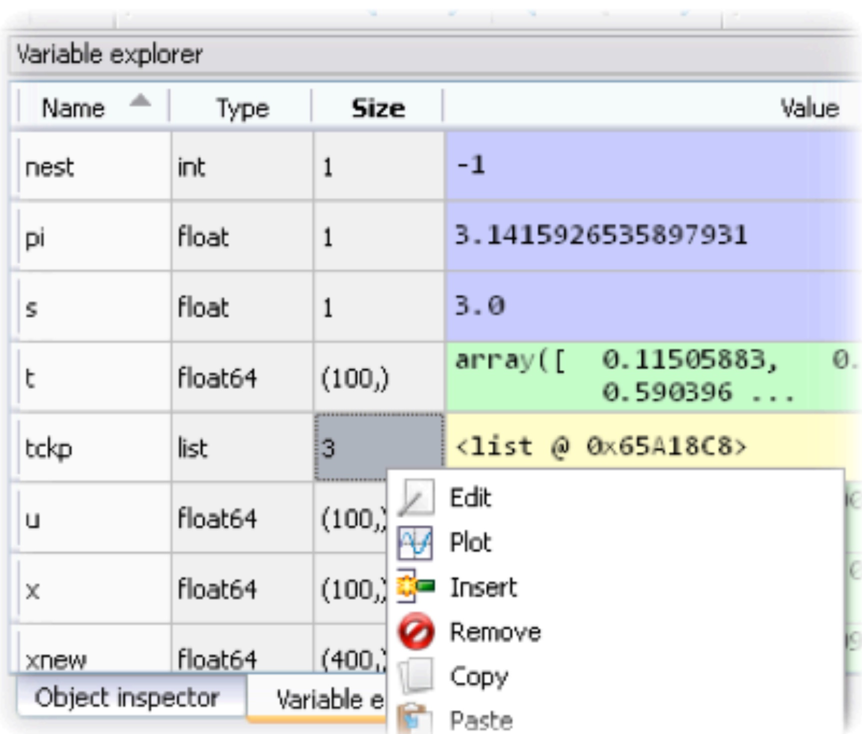
To easily interact with your code as you progress, Spyder lets you

- Open as many Python and Ipython consoles as you want
- Run a whole script or any portion of it from the Editor
- Have code completion and automatic link to documentation through the Object Inspector
- Execute all consoles in a separate process so they don't block the application

Learn More



spyder - The variable explorer



With the Variable Explorer you can browse and analyze all the results your code is producing, and also

- Edit variables with Spyder's Array Editor, which has support for a lot of data types (numbers, strings, lists, arrays, dictionaries)
- Have multiple Array Editors open at once, thus allowing to compare variable contents
- Import/Export data from/to a lot of file types (text files, numpy files, Matlab files)
- Generate 2D plots of list and arrays
- View local variables while you're debugging

Learn More

Outline

- Introduction
- Presenting the Python programming language
- Signal analysis using NumPy and SciPy
- Visualization with matplotlib and the spyder IDE
- Summary

Summary I

- The Python programming language
  - Readable, meaningful syntax (remember the tabs!)
  - Highly functional, full of functionality
  - Steep learning experience and fast results
  - Perfectly practicable for interactive work
  - Can be extended easily
  - Large global community

Summary II

- NumPy and SciPy
  - Efficient Array implementation
  - Loading and saving of multidimensional signals
  - Adds scientific stuff to Python
  - Contains basic signal processing functionality
  - Highly active and widely recommended packages

Summary III

- matplotlib
  - Plots everything...
  - Works well with NumPy arrays
- spyder
  - Nice IDE
  - Integrates scientific work flow (a bit like MatLab)
- Everything is there and freely available:
  Time to start with the exercises!