An introduction to Python programming with NumPy, SciPy and Matplotlib/Pylab

Antoine Lefebvre

Centre for Interdisciplinary Research in Music Media and Technology

Sound Modeling, Acoustics and Signal Processing Research Axis

Introduction

- Python is a simple, powerful and efficient interpreted language.
- Together with the NumPy, SciPy and Matplotlib/Pylab, it provides a nice environment for scientific works.
- It is a good alternative to Matlab. It does not provide as much functionality, but the language is way more powerful.
- Some programming experience (C/C++ and/or Matlab) would help to follow the workshop.

Goals of the presentation

- Introduce the Python programming language and standard libraries.
- Introduce the Numpy, Scipy and Matplotlib/Pylab packages.
- Discuss how it can be useful for CIRMMT members.

content of the presentation

- Python - description of the language
  - Language
  - Syntax
  - Types
  - Conditionals and loops
  - Errors
  - Functions
  - Modules
  - Classes
- Python - overview of the standard library
- NumPy
- SciPy
- Matplotlib/Pylab

What is Python?

- Python is an **interpreted**, **object-oriented**, **high-level** programming language with **dynamic semantics**.
- Python is **simple** and **easy to learn**.
- Python is **open source**, **free** and **cross-platform**.
- Python provides **high-level built in data structures**.
- Python is useful for **rapid application development**.
- Python can be used as a **scripting or glue language**.
- Python emphasizes **readability**.
- Python supports **modules and packages**.
- Python bugs or bad inputs will **never cause a segmentation fault**.

Features

- Python programs are compiled to bytecode before interpretation (provide fast execution).
- Python supports OS tools: environment variables, files, sockets, pipes, processes, threads, regular expressions, and so on.
- Python comes with an interface to the Tk GUI called Tkinter.
- Python runs as fast as Matlab (and sometimes faster).

More features (extract from the book "Learning Python")

- Dynamic typing
- Built-in object types
- Built-in tools
- Library utilities
- Third-party utilities
- Automatic memory management

Syntax

- comments are indicated with "#"
- two statements on the same line are separated with a semicolon ";"
- no semicolon at the end of lines
- a long line continue on next with "\" (it is not always needed)
- grouping is obtained through indentation
- one Python script is considered a *module* that can be run or imported by other modules
- assignment uses the equal sign "="

Assignment

- Assignment creates references, not values:
  `tmp = "hello"; tmp = 10`
  `# the first string will be deallocated`
- Contrary to C, assignment do not have value:
  `y = (x = x + 1)` is invalid
- As in C: `x += 1` is valid
- Note that pre/post increment/decrement: `x++; ++x; x--; --x` are invalid
- Multiple assignment (references to a unique object): `x=y=z=1`
- Multiple assignments: `(x,y,z)=(3.5,5.5,'string')`
- Example of swaping variables value: `(x,y)=(y,x)`

Identifiers

- First character: a letter or the symbol "_"
- Additional characters: alphanumeric or the symbol "_"
- They are case-sensitive

Documentation

The first unassigned string in a module, class declaration, or function declaration can be accessed through by using

```
obj.__doc__
```

where obj is the module, class, or function name.
`__doc__` is one of Python special variables.

Special variables

- Python relies on many special variables that can be accessed by your code.
- One is the "__name__" variables.
- When a module is run, it contains the string "__main__".
- When the module is imported, it contains the modules name.
- You can add code that runs only when a module is called directly:
  `if __name__ == '__main__': test()`
- The use of special variables is an advanced topic. We won't talk about that too much.

Built-in object types

Numbers	`3.1415, 1234, 999L, 3+4j`
Strings	`'spam', "guido's"`
Lists	`[1, [2, 'three'], 4]`
Dictionaries	`{'food':'spam', 'taste':'yum'}`
Tuples	`(1,'spam', 4, 'U')`
Files	`text = open('eggs', 'r').read()`

numbers

integers	`1234, -24, 0`
unlimited precision integers	`999999999999L`
floating	`1.23, 3.14e-10, 4E210, 4.0e+210`
oct and hex	`0177, 0x9ff`
complex	`3+4j, 3.0+4.0j, 3J`

strings (immutable sequences)

single quote	`s1 = 'egg'`
double quotes	`s2 = "spam's"`
triple quotes	`block = """..."""`
concatenate	`s1 + s2`
repeat	`s2 * 3`
index,slice	`s2[i], s2[i:j]`
length	`len(s2)`
formatting	`"a %s parrot" % 'dead'`
iteration	`for x in s2` # x loop through each character of s2
membership	`'m' in s2,` # return True if the 'm' is in the string s2

Lists

- Ordered collections of arbitrary objects
- Accessed by offset
- Variable length, heterogeneous, arbitrarily nestable
- Mutable sequence
- Arrays of object references

Lists operations

empty list	`L = []`
four items	`L2 = [0, 1, 2, 3]`
nested	`L3 = ['abc', ['def', 'ghi']]`
index	`L2[i], L3[i][j]`
slice, length	`L2[i:j], len(L2)`
concatenate, repeat	`L1 + L2, L2 * 3`
iteration, membership	`for x in L2, 3 in L2`
methods	`L2.append(4), L2.sort(), L2.index(1), L2.reverse()`
shrinking	`del L2[k], L2[i:j] = []`
assignment	`L2[i] = 1, L2[i:j] = [4,5,6]`
create list	`range(4), xrange(0, 4)` # useful to loop

Dictionaries

- Accessed by key, not offset
- Unordered collections of arbitrary objects
- Variable length, heterogeneous, arbitrarily nestable
- Of the category mutable mapping
- Tables of object references (hash tables)

Dictionaries operations

empty	`d1 = {}`
two-item	`d2 = {'spam': 2, 'eggs': 3}`
nesting	`d3 = {'food': {'ham': 1, 'egg': 2}}`
indexing	`d2['eggs'], d3['food']['ham']`
methods	`d2.has_key('eggs'), d2.keys(), d2.values()`
length	`len(d1)`
add/change	`d2[key] = new`
deleting	`del d2[key]`

tuples

- They are like lists but immutable. Why Lists and Tuples?
- When you want to make sure the content won't change.

Files

input	input = open('data', 'r')
read all	S = input.read()
read N bytes	S = input.read(N)
read next	S = input.readline()
read in lists	L = input.readlines()
output	output = open('/tmp/spam', 'w')
write	output.write(S)
write strings	output.writelines(L)
close	output.close()

Unsupported Types

- No Boolean type, use integers.
- no char or single byte, use strings of length one or integers
- no pointer
- int vs. short vs. long, there is only one integer type in Python (its a C long)
- float vs. double, there is only one floating point type in Python (its a C double)

Comparisons vs. Equality

- `L1 = [1, ('a', 3)]`
- `L2 = [1, ('a', 3)]`
- `L1 == L2, L1 is L2`
  (1, 0)

- The == operator tests value equivalence
- The `is` operator tests object identity

if, elif, else

```
>>> if not done and (x > 1):
>>>     doit()
>>> elif done and (x <= 1):
>>>     dothis()
>>> else:
>>>     dothat()
```

while, break

```
>>> while True:
>>>     line = ReadLine()
>>>     if len(line) == 0:
>>>         break

>>> def showMaxFactor(num):
>>>     cnt = num / 2
>>>     while cnt > 1:
>>>         if (num % cnt == 0):
>>>             print 'larg. fact. of %d is %d'%(num, cnt)
>>>             break
>>>         count = cnt - 1
>>>     else:
>>>         print num, "is prime"
```

for

```
>>> for letter in 'hello world':
>>>     print letter

>>> for item in [12, 'test', 0.1+1.2J]:
>>>     print item

>>> for i in range(2,10,2):
>>>     print i
```

Equivalent to the C loop:

```
for (i = 2; i < 10; i+=2){
    printf("%d\n",i);
}
```

pass

```
>>> def stub():
>>>     pass
```

switch/case

There is no such statement in Python. It can be implementd efficiently with a dictionary of functions:

```
>>> result = {
>>>  'a': lambda x: x * 5,
>>>  'b': lambda x: x + 7,
>>>  'c': lambda x: x - 2
>>>}
>>> result['b'](10)
```

Note: anonymous function need be defined with the lambda construct. The following functions f and g do the same thing:

```
>>> def f(x): return x**2
>>> g = lambda x: x**2
```

lambda functions can be place anywhere a function is expected without formal definition.

errors and exceptions

NameError	attempt to access an undeclared variable
ZeroDivisionError	division by any numeric zero
SyntaxError	Python interpreter syntax error
IndexError	request for an out-of-range index for sequence
KeyError	request for a non-existent dictionary key
IOError	input/output error
AttributeError	attempt to access an unknown object attribute

```
>>> try:
>>>     f = open('blah')
>>> except IOError:
>>>     print 'could not open file'
```

assertion

```
>>> assert 0 < val < 100, 'Value out of range'
```

Functions

- Functions can return any type of object
- When nothing is return the `None` object is returned by default
- There is two ways to specify function parameters: standard and keyworded
- Parameters can have default arguments
- Variable-length arguments are supported

function example

```
>>> def typical_function(a,b=2,d=func):
>>>     """ Function showing how to define
>>>         arguments with and w/o default values
>>>         Expect a function object for the third
>>>     """
>>>
>>>     return d(a,b)
>>>
>>> test(3); test(b=4,a=3);
>>> test(1,2,lambda x,y: x*y); test(1,2,g)
```

Functions with variable-length arguments

```
>>> def vargtest(a, b, *nkw, **kw):
>>>     'display regular args and all variable args'
>>>     print 'a is:', a
>>>     print 'b is:', b
>>>     for eachNKW in nkw:
>>>         print 'additional non-keyword arg:', \
>>>                 eachNKW
>>>     for eachKW in kw.keys():
>>>         print "additional keyword arg '%s': %s" %\
>>>                 (eachKW, kw[eachKW])
>>>
>>> vargtest(1,2,3,4,x=2,y=3)
>>> vargtest(1,2,*(4,5,6),**{'x':1,'y':3})
```

Modules, namespaces and packages

- A file is a module. Suppose we have a file called 'myio.py', implementing a function called 'load'
- If we want to use that function from another module we do

```
>>> import myio
>>> myio.load()
```

- All the code present in 'myio.py' will be in the 'myio' namespace
- You can import specific parts of a module

```
>>> from myio import load
>>> load()
```

- Packages are bundle of modules. We won't cover that.

Classes

```
>>> class Cone(WaveguideProfile):
>>>     def __init__(self,d0,de,L):
>>>         "Create a cone"
>>>         self.a0 = d0/2
>>>         self.ae = de/2
>>>         self.L = L
>>>     def __del__(self):
>>>         pass
>>>     def radius(self,z):
>>>         return self.ae + (self.a0-self.ae)*z/self.L
>>>     def radiusp(self,z):
>>>         "derivative of the radius at z"
>>>         return (self.a0-self.ae)/self.L
>>> c = Cone(0.1,0.2,1.5); c.radius(0.5)
```

overloading

- Python does not support method or function overloading
- You have to rely on the type() built-in function

```
>>> def testtype(a):
>>>     if type(a) == types.FloatType:
>>>         print "I got a float"
>>>     elif type(A) == types.ComplexType:
>>>         print "A complex number"
>>>     else:
>>>         print "something else"
```

standard library core modules

os	file and process operations
os.path	platform-independant path and filename utilities
time	datEs and times related functions
string	commonly used string operations
math,cmath	math operations and constants, complex version
re	regular expressions
sys	access to interpreter variables
gc	control over garbage collector
copy	allow to copy object

other standard library modules

- Support for threads, pipes, signals, etc.
- Support for common file formats: XML, SGML and HTML; zip and gzip; a lexer
- Support for network protocols
- Support for reading and writing images and sound
- Support for databases
- Support for debugging, profiling and analysing code

some GUI packages

Tkinter	standard Python interface to the Tk GUI toolkit (cross-platform)
wxPython	toolkit for Python built around the popular wxWidgets C++ toolkit (cross-platform)
PyQt	binding to the Qt toolkit (cross-platform)
PyGTK	bindings for the GTK widget set

NumPy - fundamental package for scientific computing with Python

- powerful N-dimensional array object
- sophisticated functions
- basic linear algebra functions
- basic Fourier transforms
- sophisticated random number capabilities
- tools for integrating Fortran code.
- tools for integrating C/C++ code.

Comparison with Matlab

- In NumPy, operation are elementwise by default
- There is a matrix type for linear algebra (subclass of array)
- Indexing start at 0 in NumPy
- Using Python with NumPy gives more programming power
- Function definition in Matlab have many restriction
- NumPy/SciPy is free but still widely used
- Matlab have lots of 'toolboxes' for specific task (lot less in Numpy/SciPy)
- There are many packages for ploting in Python that are as good as Matlab

Some Matlbab/NumPy equivalence

Matlab	NumPy
a = [1 2 3; 4 5 6]	a = array([[1.,2.,3.],[4.,5.,6.]])
a(end)	a[-1]
a(2,5)	a[1,4]
a(2,:)	a[1] or a[1,:]
a(1:5,:)	a[0:5] or a[:5] or a[0:5,:]
a(end-4:end,:)	a[-5:]
a(1:3,5:9)	a[0:3][:,4:9]
a(1:2:end,:)	a[::2,:]
a(end:-1:1,:) or flipud(a)	a[::-1,:]
a.'	a.transpose() or a.T
a'	a.conj().transpose() or a.conj().T
a * b	dot(a,b)
a .* b	a * b
a./b	a/b

Some Matlbab/NumPy equivalence cont.

Matlab	NumPy
a.∧3	a**3
find(a>0.5)	where(a>0.5)
a(a<0.5)=0	a[a<0.5]=0
a(:) = 3	a[:] = 3
y=x	y = x.copy()
y=x(2,:)	y = x[2,:].copy()
y=x(:)	y = x.flatten(1)
1:10	arange(1.,11.) or r_[1.:11.]
0:9	arange(10.) or r_[:10.]
zeros(3,4)	zeros((3,4))
zeros(3,4,5)	zeros((3,4,5))
ones(3,4)	ones((3,4))
eye(3)	eye(3)

Some Matlbab/NumPy equivalence cont.

Matlab	NumPy
diag(a)	diag(a) or a.diagonal()
diag(a,0)	diag(a,0) or a.diagonal(0)
rand(3,4)	random.rand(3,4)
linspace(1,3,4)	linspace(1,3,4)
[x,y]=meshgrid(0:8,0:5)	mgrid[0:9.,0:6.]
repmat(a, m, n)	tile(a, (m, n))
[a b]	concatenate((a,b),1) or hstack((a,b)) or c_[a,b
[a; b]	concatenate((a,b)) or vstack((a,b)) or r_[a,b]
max(max(a))	a.max()
max(a)	a.max(0)
max(a,[],2)	a.max(1)
max(a,b)	where(a>b, a, b)
norm(v)	sqrt(dot(v,v)) or linalg.norm(v)

Some Matlbab/NumPy equivalence cont.

Matlab	NumPy
inv(a)	linalg.inv(a)
pinv(a)	linalg.pinv(a)
a\b	linalg.solve(a,b)
b/a	Solve a.T x.T = b.T instead
[U,S,V]=svd(a)	(U, S, V) = linalg.svd(a)
chol(a)	linalg.cholesky(a)
[V,D]=eig(a)	linalg.eig(a)
[Q,R,P]=qr(a,0)	(Q,R)=Sci.linalg.qr(a)
[L,U,P]=lu(a)	(L,U)=linalg.lu(a) or (LU,P)=linalg.lu_factor(a)
conjgrad	Sci.linalg.cg
fft(a)	fft(a)
ifft(a)	ifft(a)
sort(a)	sort(a) or a.sort()
sortrows(a,i)	a[argsort(a[:,0],i)]

Scipy

- The SciPy library depends on NumPy
- gathers a variety of high level science and engineering modules together:

Fftpack	discrete fourier transform algorithms
Integrate	integration routines
Interpolate	interpolation tools
Linalg	linear algebra routines
Optimize	optimization tools
Signal	signal processing tools
Sparse	sparse matrices
Stats	statistical functions
Io	data input and output
Special	definitions of many usual math functions
Weave	C/C++ integration

scipy.signal - signal processing tools

- Convolution
- B-splines
- Filtering
- Filter design
- Matlab-style IIR filter design
- Linear Systems
- LTI Reresentations
- Waveforms
- Window functions
- Wavelets

scipy.io - data input and output

- It contains routines to read and write data
- Important for us, it supports MATLAB mat files
  - loadmat()
  - savemat()

Matplotlib/Pylab

- Matplotlib is a object oriented plotting library.
- Pylab is the interface on top of Matplotlib emulating MATLAB functions.
- You can use latex expressions to add math to your plot.

Examples and exercices

- We continue the presentation by studying code examples and experimenting with the Python interpreter.