Scientific Computing using Python

Srijith Rajamohan

Advanced Research Computing

Wednesday 15th April, 2015
Course Contents

This week:

- Introduction to Scientific Python
- Python Programming
- NumPy
- SciPy
- Plotting with Matplotlib
- Interoperability with C
- Conclusion
Section 1

1. Introduction to Scientific Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Interoperability with C
7. Conclusion
Why Python?

- Intuitive and minimalistic code
- Expressive language
- Dynamically typed
- Automatic memory management
- Interpreted
Python Features

Advantages

- Ease of programming
- Minimizes the time to develop and maintain code
- Modular and object-oriented
- Large community of users
- A large standard and user-contributed library

Disadvantages

- Interpreted and therefore slower than compiled languages
- Decentralized with packages
Where Is Python Used?

- PyClaw for wave propagation problems
- Petsc4py for linear solvers and Slepc4py for eigenvalue solvers
- SymPy
- Scikit-learn
- ParaView, Visit and VTK for visualization
- PyQt
- VisTrails
Code Performance vs Development Time

![Graph showing code performance vs development time with high-level language, low-level language 1, and low-level language 2. The graph has a title "Optimizing What?!".](image)
Versions of Python

- Two versions of Python in use now - Python 2 and Python 3
- Python 3 not backward-compatible with Python 2
- A lot of packages are available for Python 2
- Check version using the following command

Example

```bash
$ python --version
```
You can also use the interactive **Ipython** interpreter

- Command history
- Execute system commands
- Command auto-completion
- Great for plotting!
- [http://ipython.org](http://ipython.org)
Spyder GUI

- Spyder is an IDE for Python - coding, debugging, and execution in an integrated environment.
- Code editor with syntax highlighting
- Variable explorer
NOTE: Indentation is very important in Python. It defines the extent of a code block.

Example

```python
#!/usr/bin/env python  # Path to python interpreter on Unix systems

print("Hello World!")
```
To run a program named ‘hello.py’ on the command line

Example

```
$ python hello.py
```

You can do the same in the interpreter. Invoke the interpreter by typing ‘python’ on the command line and then use `execfile`

Example

```
>>> execfile("hello.py")
```
Python Modules

- Python functionality such as I/O, string manipulation, math routines etc. provided by modules
- Reference to Python standard library of modules at http://docs.python.org/2/library/

Example

```
import math  #This imports the whole module
x = math.sin(math.pi)
print x
```

Python Modules

- Python functionality such as I/O, string manipulation, math routines etc. provided by modules
- Reference to Python standard library at http://docs.python.org/2/library/

Example

```python
from math import *  # This imports all symbols to the current namespace
x = sin( pi )
print x
```
Python Modules - Documentation

- Your Python installation already comes with plenty of modules built-in
- Use the `dir` command to list the symbols (functions, classes and variables) in a module
- The `help` command can be used on each function to obtain documentation as long as they have ‘docstrings’, which is a string within triple quotes

Example

```python
def test_help():
    """Return the pathname of the root directory.""
    print "hello"
```
Example

```python
>>> print(dir(math))
```
Example

```python
>>> help(math.log)
Help on built-in function log in module math:

log(...)
    log(x[, base])

Return the logarithm of x to the given base.
If the base not specified, returns the natural logarithm (base e) of x.
```
Section 2

1. Introduction to Scientific Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Interoperability with C
7. Conclusion
Variables

- Variable names can contain alphanumerical characters and some special characters
- It is common to have variable names start with a lower-case letter and class names start with a capital letter
- Some keywords are reserved such as ‘and’, ‘assert’, ‘break’, ‘lambda’
- Python is dynamically typed, the type of the variable is derived from the value it is assigned.
- A variable is assigned using the ‘=’ operator
Variable types

- Variable types
  - Integer
  - Float
  - Boolean
  - Complex
  - String

- Use the `type` function to determine variable type, e.g. `type(x)`

- Variables can be cast to a different type, e.g. `complex(x)`
Operators

- Arithmetic operators +, -, *, /, // (integer division for floating point numbers), '**' power
- Boolean operators and, or and not
- Comparison operators >, <, >= (greater or equal), <= (less or equal), == equality
Strings

Example

```python
>>> s = "Hello world"
>>> dir(s)
```
Strings

Example

```python
>>> len(s)
11
>>> s
'Hello world'
>>> s[0]  # indexing starts at 0
'H'
>>> s2 = s.replace("world", "test")
>>> print(s2)
Hello test
```
Printing strings

Example

```python
# concatenates strings with a space
>>> print("str1", "str2", "str3")
str1 str2 str3
# concatenated without space
>>> print("str1" + "str2" + "str3")
str1str2str3
# C-style string formatting
>>> print("value = \%f" % 1.0)
value = 1.000000
# Creating a formatted string
>>> s2 = "value1 = \%.2f. value2 = \%d" % (3.1415, 1.5)
>>> print(s2)
value1 = 3.14. value2 = 1
```
Lists

Array of elements of arbitrary type

Example

```python
>>> a = [1,2,3]
>>> type(a)
<type 'list'>
>>> a = [1,a,"hello"]
>>> type(a)
<type 'list'>
```
Lists

Example

# create a new empty list
>>> l = []
# add elements using ‘append’
l.append("A")
l.append("d")
l.append("d")
print(l)
[‘A’, ‘d’, ‘d’]
Lists

Example

# Lists are mutable - its value can be changed

```python
>>> l[1] = 'p'
>>> l[2] = 'p'
>>> print(l)
['A', 'p', 'p']
```
Lists

Example

```python
>>> l.insert(0, "i")
>>> l.insert(1, "n")
>>> l.insert(2, "s")
>>> l.insert(3, "e")
>>> l.insert(4, "r")
>>> l.insert(5, "t")
>>> print(l)
['i', 'n', 's', 'e', 'r', 't', 'A', 'd', 'd']
```
# Remove first element

```python
>>> l.remove("A")
>>> print(l)
['i', 'n', 's', 'e', 'r', 't', 'd', 'd']
```

# Remove an element at a specific location

```python
>>> del l[7]
>>> del l[6]
>>> print(l)
['i', 'n', 's', 'e', 'r', 't']
```
Tuples

Tuples are like lists except they are immutable

Example

```python
>>> point = (10, 20)  # Note () for tuples instead of []
>>> type(point)
<type 'tuple'>
>>> point = 10, 20
>>> type(point)
<type 'tuple'>
>>> point[2] = 40  # Cannot do this!
```

Tuples are immutable
Dictionaries are also like lists, except that each element is a key-value pair

Example

```python
>>> params = {
"parameter1" : 1.0,
... "parameter2" : 2.0,
... "parameter3" : 3.0,
}

>>> type(params)
<type 'dict'>

>>> print params
{"parameter1": 1.0, "parameter3": 3.0, '
  parameter2": 2.0}

>>> params["parameter3"]
3.0
```
Conditional statements: if, elif, else

Example

```python
>>> statement1 = False
>>> statement2 = True
>>> if statement1 is True:
...     print "Statement 1 is true"
... elif statement2 is True:
...     print "Statement 2 is true"
... else:
...     print "None"
...
Statement 2 is true
```
Loops - For

Example

```python
>>> for x in [1,2,3):
    ...
    print(x)
1
2
3
```

```python
>>> for word in ["scientific", "computing", "with", "python"]:
    ...
    print(word)
scientific
computing
with
python
```
Loops - While

Example

```python
>>> i = 0
>>> while i < 5:
    ...
    print(i)
    ...
    i = i + 1
0
1
2
3
4
```
Functions

Example

```python
>>> def func1(s):
...     """
...     Print a string 's' and tell how many characters it has
...     """
...     print(s + " has " + str(len(s)) + " characters")

>>> func1("test")
```

35 / 96
Example

```python
>>> def powers(x):
...     return x ** 2, x ** 3, x ** 4

>>> x2, x3, x4 = powers(3)
```
Example

```python
>>> def myfunc(x, p=2, debug=False):
...     if debug:
...         print(str(x) + str(p))
...     return x**p
```
Classes

- Classes are one of the key features of object-oriented programming
- An instance of a class is an object
- A class contains attributes and methods that are associated with this object
Classes

Example

```python
>>> class Point:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...     def translate(self, dx, dy):
...         self.x += dx
...         self.y += dy
...     def __str__(self):
...         return "Point at [\(\%f, \%f\)] \(\%f, \%f\)" % (self.x, self.y)
```
# To create a new object

```python
>>> p1 = Point(0, 0)  # this will invoke the __init__ method in the Point class
```

```python
>>> print(p1)  # this will invoke the __str__ method
```

```
Point at [0.000000, 0.000000]
```
Section 3

1. Introduction to Scientific Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Interoperability with C
7. Conclusion
NumPy

Used in almost all numerical computations in Python
  • Used for high-performance vector and matrix computations
  • Written in C and Fortran
  • Vectorized computations
Arrays

Example

```python
# the argument to the array function is a Python list
>>> v = array([1,2,3,4])
# the argument to the array function is a nested Python list
>>> M = array([[1, 2], [3, 4]])
>>> type(v), type(M)
(numpy.ndarray, numpy.ndarray)
```
Arrays

Example

```python
>>> v.shape, M.shape
((4,), (2, 2))
>>> M.size
4
>>> M.dtype
dtype('int64')
# Explicitly define the type of the array
>>> M = array([[1, 2], [3, 4]], dtype=complex)
```
Arrays - Using array-generating functions

Example

```python
>>> x = arange(0, 10, 1) # arguments: start, stop, step
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> linspace(0,10,11) # arguments: start, end and number of points (start and end points are included)
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```
Mgrid

Example

```python
>>> x, y = mgrid[0:3, 0:2]
>>> x
array([[0, 0],
       [1, 1],
       [2, 2]])

>>> y
array([[0, 1],
       [0, 1],
       [0, 1]])
```
Diagonal and Zero matrix

Example

```python
>>> diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])

>>> zeros((3,3))
array([[ 0., 0., 0.],
       [ 0., 0., 0.],
       [ 0., 0., 0.]])
```
Array Access

Example

```python
>>> M = rand(3,3) # not a Numpy function
>>> M
array([[ 0.37389376,  0.64335721,  0.12435669],
       [ 0.01444674,  0.13963834,  0.36263224],
       [ 0.00661902,  0.14865659,  0.75066302]])
>>> M[1,1]
0.13963834214755588
```
Array Access

Example

# Access the first row

```python
>>> M[1]
array(
[ 0.01444674,  0.13963834,  0.36263224])
```

# The first row can be also be accessed using this notation

```python
>>> M[1,:]
array(
[ 0.01444674,  0.13963834,  0.36263224])
```

# Access the first column

```python
>>> M[:,1]
array(
[ 0.64335721,  0.13963834,  0.14865659])
```
Array Access

Example

```python
# You can also assign values to an entire row or column

>>> M[1, :] = 0
>>> M
array([[ 0.37389376,  0.64335721,  0.12435669],
       [ 0.    ,  0.    ,  0.    ],
       [ 0.00661902,  0.14865659,  0.75066302]])
```
Array Slicing

Example

```python
# Extract slices of an array
>>> M[1:3]
array([[ 0.,  0.,  0.],
       [ 0.00661902,  0.14865659,  0.75066302]])
>>> M[1:3,1:2]
array([[ 0.],
       [ 0.14865659]])
```
### Array Slicing - Negative Indexing

#### Example

```python
# Negative indices start counting from the end of the array

>>> M[-2]
array(
[ 0., 0., 0.])

>>> M[-1]
array(
[ 0.00661902, 0.14865659, 0.75066302])
```
Array Access - Strided Access

Example

```python
# Strided access

>>> M[::2,::2]
array([[ 0.37389376,  0.12435669],
       [ 0.00661902,  0.75066302]])
```
Array Operations - Scalar

These operation are applied to all the elements in the array

Example

```python
>>> M*2
array([[ 0.74778752,  1.28671443,  0.24871338],
       [ 0. ,     0. ,     0. ],
       [ 0.01323804,  0.29731317,  1.50132603]])
```

```python
>>> M + 2
array([[ 2.37389376,  2.64335721,  2.12435669],
       [ 2. ,     2. ,     2. ],
       [ 2.00661902,  2.14865659,  2.75066302]])
```
Matrix multiplication

```python
>>> M * M  # Element-wise multiplication
array([
    [1.397965e-01, 4.139085e-01, 1.546458e-02],
    [0.000000e+00, 0.000000e+00, 0.000000e+00],
    [4.381141e-05, 2.209878e-02, 5.634949e-01]])
>>> dot(M,M)  # Matrix multiplication
array([
    [0.14061966, 0.25903369, 0.13984616],
    [0. , 0. , 0. ],
    [0.00744346, 0.1158494 , 0.56431808]])
```
Iterating over Array Elements

- In general, avoid iteration over elements
- Iterating is slow compared to a vector operation
- If you must, use the `for` loop
- In order to enable vectorization, ensure that user-written functions can work with vector inputs.
  - Use the `vectorize` function
  - Use the `any` or `all` function with arrays
Vectorize

Example

```python
>>> def Theta(x):
...     """
...     Scalar implementation of the Heaviside step function.
...     """
...     if x >= 0:
...         return 1
...     else:
...         return 0

>>> Theta(1.0)
1
>>> Theta(-1.0)
0
```
Without vectorize we would not be able to pass v to the function

Example

```python
>>> v
array([1, 2, 3, 4])
>>> Tvec = vectorize(Theta)
>>> Tvec(v)
array([1, 1, 1, 1])
>>> Tvec(1.0)
array(1)
```
Arrays in conditions

Use the `any` or `all` functions associated with arrays

**Example**

```python
>>> v
array([1, 2, 3, 4])
>>> (v > 3).any()
True
>>> (v > 3).all()
False
```
Section 4

1. Introduction to Scientific Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Interoperability with C
7. Conclusion
SciPy

- SciPy framework built on top of the NumPy framework
- SciPy imports all the functions from the NumPy namespace
- Large number of scientific algorithms
  - Integration
  - Optimization
  - Linear Algebra
  - Sparse Eigenvalue Problems
  - Statistics
  - File I/O
  - Fourier Transforms
  - ... and many more
Let's look at some examples

Using any of these subpackages requires an explicit import

- Linear Algebra
- Dense and Sparse Matrices
- Optimization
Get system parameters

Example

```python
>>> import sys
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min
=2.2250738585072014e-308, min_exp
=-1021, min_10_exp=-307, dig=15,
mant_dig=53, epsilon=2.220446049250313e
-16, radix=2, rounds=1)
```
Linear Algebra

To solve an equation of the form $A\ x = b$

Example

```python
>>> from scipy import *
>>> from scipy import linalg
>>> A = array([[1,2,3], [4,5,6], [7,8,9]])
>>> b = array([1,2,3])
>>> x = linalg.solve(A, b)
array([-0.33333333, 0.66666667, 0. ])
>>> linalg.norm(dot(A, x) - b)
1.1102230246251565e-16
```
Linear Algebra - Inverse

Example

```python
>>> A = rand(3,3)
>>> A
array([[ 0.24514116,  0.52587023,  0.18396222],
       [ 0.90742329,  0.16622943,  0.13673048],
       [ 0.09218907,  0.51841822,  0.5672206 ]])
>>> linalg.inv(A)
array([[-0.13406351,  1.16228558, -0.23669318],
       [ 2.87602299, -0.69932327, -0.76418374],
       [-2.60678741,  0.45025145,  2.49988679]])
```
Example

```python
>>> evals, evecs = linalg.eig(A)
>>> evals
array([-0.46320383+0.j,  1.09877378+0.j,  0.34302124+0.j])
>>> evecs
array([[-0.49634545,  0.49550686, -0.20682981],
        [ 0.79252573,  0.57731361, -0.35713951],
        [-0.35432211,  0.64898532,  0.91086377]])
```
Dense and Sparse Matrices

Example

```python
>>> from scipy.sparse import *
>>> M = array([[1, 0, 0, 0], [0, 3, 0, 0], [0, 1, 1, 0], [1, 0, 0, 1]])
>>> A = csr_matrix(M)
<4x4 sparse matrix of type <type 'numpy.int64'>
with 6 stored elements in Compressed Sparse Row format>
```
Dense and Sparse Matrices

Example

```python
>>> A.todense()  # print A as dense matrix
matrix([[1, 0, 0, 0, 0],
        [0, 3, 0, 0, 0],
        [0, 1, 1, 0, 0],
        [1, 0, 0, 1, 0]])
```
Dense and Sparse Matrices

Example

```python
>>> A = csr_matrix(A); A
<4x4 sparse matrix of type 'numpy.float64'>
with 6 stored elements in Compressed Sparse Row format>
>>> A = csc_matrix(A); A
<4x4 sparse matrix of type 'numpy.float64'>
with 6 stored elements in Compressed Sparse Column format>
```
Optimization

Compute the minima of a single variable function

Example

```python
>>> from scipy import optimize
>>> def f(x):
    return 4*x**3 + (x-2)**2 + x**4
```
Function \( f(x) \)
Optimization

Example

```python
>>> x_min = optimize.fmin_bfgs(f, -2)
Optimization terminated successfully.
Current function value: -3.506641
Iterations: 6
Function evaluations: 30
Gradient evaluations: 10
array([-2.67298167])
```
Section 5

1. Introduction to Scientific Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Interoperability with C
7. Conclusion
Matplotlib

- Used for generating 2D and 3D scientific plots
- Support for LaTeX
- Fine-grained control over every aspect
- Many output file formats including PNG, PDF, SVG, EPS
Matplotlib - Customize matplotlibrc

- Configuration file ‘matplotlibrc’ used to customize almost every aspect of plots
- On Linux, it looks in .config/matplotlib/matplotlibrc
- On other platforms, it looks in .matplotlib/matplotlibrc
- Use ‘matplotlib.matplotlib_fname()’ to determine from where the current matplotlibrc is loaded
- Customization options can be found at http://matplotlib.org/users/customizing.html
Matplotlib

- Matplotlib is the entire library
- Pyplot - a module within Matplotlib that provides access to the underlying plotting library
- Pylab - a convenience module that combines the functionality of Pyplot with NumPy
- Pylab interface convenient for interactive plotting
Example

```python
>>> from pylab import *
>>> ion()
>>> isinteractive()
True
>>> x = [1, 3, 7]
>>> plot(x)  # if interactive mode is off use show() after the plot command
[<matplotlib.lines.Line2D object at 0x10437a190>]
>>> savefig('fig_test.pdf', dpi=600, format='pdf')
```
Pylab

Simple Pylab plot

```
0.0 0.5 1.0 1.5 2.0
1 2 3 4 5 6
```

```python
import matplotlib.pyplot as plt

x = [0, 0.5, 1, 1.5, 2]
y = [0, 0.5, 1, 1.5, 2]
plt.plot(x, y)
plt.show()
```
Example

```python
>>> import matplotlib.pyplot as plt
>>> plt.isinteractive()
False
>>> x = np.linspace(0, 3*np.pi, 500)
>>> plt.plot(x, np.sin(x**2))
[<matplotlib.lines.Line2D object at 0x104bf2b10>]
>>> plt.title('Pyplot plot')
<matplotlib.text.Text object at 0x104be4450>
>>> plt.show()
>>> savefig('fig_test_pyplot.pdf', dpi=600, format='pdf')
```
Pyplot
Pyplot - 3D plots

Surface plots

Visit http://matplotlib.org/gallery.html for a gallery of plots produced by Matplotlib
Section 6

1. Introduction to Scientific Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Interoperability with C
7. Conclusion
Mixing C with Python

Ctypes
- Ctypes is a foreign function library for Python
- Compatible with C data types
- Allows interfacing with shared libraries

Cython (Not covered here)
- Hybrid approach between C and Python
- Python code with type declarations
C types - Data types

C compatible data types in ctypes

- `c_bool`, `c_char`, `c_int`, `c_float`, `c_double` etc.
- These correspond to `bool`, `character string`, `int`, `float` and `float` respectively in Python
- List of all data types here
  https://docs.python.org/2/library/ctypes.html
Initialize a ctype data type

**Example**

```python
>>> from ctypes import *

>>> c_int()
c_int(0)

>>> c_wchar_p("Hello, World")
c_wchar_p('Hello, World')

>>> c_float(-3.4)
c_float(-3.4)
```
```c
#include <stdio.h>

void hello(int n);

void
hello(int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf("C says hello\n");
    }
}
```
Compiling the C code

Compile the C code containing the functions into a shared library

Example

```
gcc -c -Wall -O2 -Wall -ansi -pedantic -fPIC -o functions.o functions.c
gcc -o libfunctions.so -shared functions.o
```
Using numpy.ctypeslib

Example

```python
import numpy
import ctypes

_libfunctions = numpy.ctypeslib.load_library('libfunctions', '.

_libfunctions.hello.argtypes = [ctypes.c_int]
_libfunctions.hello.retype = ctypes.c_void_p

def hello(n):
    return _libfunctions.hello(int(n))
```
Call the C function from Python

Example

```python
>>> import functions
>>> functions.hello(3)  # This calls the Python wrapper function
C says hello
C says hello
C says hello
```
Passing a data structure to a C function

Example

def dprod(double x, int n):
    y = 1.0
    for i in range(n):
        y *= x[i]
    return y
Example

```python
_libfunctions.dprod.argtypes = [numpy.
crosseslib.ndpointer(dtype=numpy.
float),
crosses.c_int]
_libfunctions.dprod.restype = ctypes.
c_double

def dprod(x, n=None):
    # Takes a list ‘x’
    if n is None:
        n = len(x)
    x = numpy.asarray(x, dtype=numpy.
float)
    # Converts ‘x’ to a numpy array
    return _libfunctions.dprod(x, int(n))
```
Calling the C function from Python

Example

```python
>>> functions.dprod([1,2,3,4,5])
120.0
>>> functions.dprod([1,2,3,4,5],5)
120.0
>>> functions.dprod([1,2,3,4,5],4)
24.0
```
Using ctypes.CDLL

No Python wrapper needed here, you access the C function directly

Example

```python
>>> t = ctypes.CDLL('libfunctions.so')
>>> t.dprod.restype = c_double
>>> x = (ctypes.c_double * 3)()
>>> x[0] = 1.0
>>> x[1] = 2.0
>>> x[2] = 3.0
>>> t.dprod(x,c_int(3))
6
```
Section 7

1. Introduction to Scientific Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Interoperability with C
7. Conclusion
Conclusion

- Python used extensively by the educational and scientific community
- Used as both a scripting and prototyping tool
- Plenty of libraries out there
- Extensively documented!
Questions

Thank you for attending!