Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Introduction to Python

Srijith Rajamohan and Peter Radics

Advanced Research Computing, Virginia Tech

Monday 27th July, 2015

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Course Contents

This week:

- Introduction to Python
- Python Programming
- NumPy
- SciPy
- Plotting with Matplotlib
- Debugging
- Exception Handling
- Model problems
- Conclusion

Section 1

1. Introduction to Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Debugging
7. Exception Handling
8. Model problems
9. Conclusion

Python Features

Why Python ?

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

Python Features

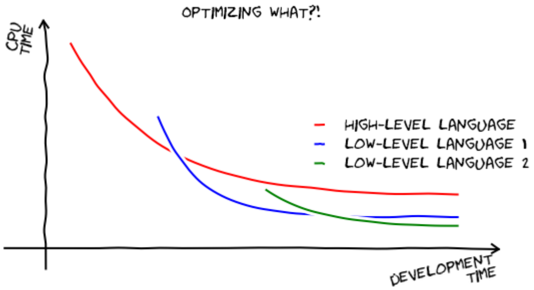
Advantages

- 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

Code Performance vs Development Time



Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Versions of Python

- Two versions of Python in use - 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

```
$ python --version
```

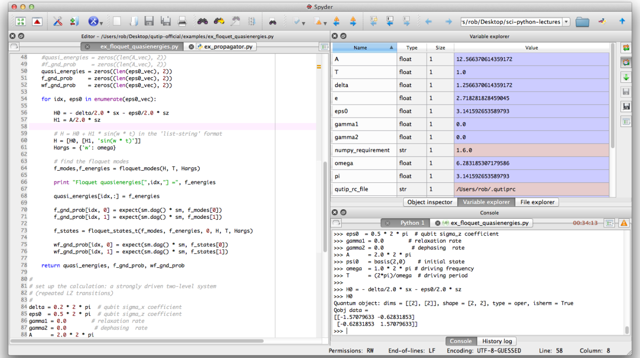
Ipython

You can also use the interactive **Ipython** interpreter

- Command history
- Execute system commands
- Command auto-completion
- Great for plotting!
- `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


Spyder
Editor - /Users/rob/Desktop/qutip-official/examples/ex_floquet_quasienergies.py
ex_floquet_quasienergies.py
ex_propagator.py
s/rob/Desktop/sci-python-lectures
Variable explorer
Object inspector
Variable explorer
File explorer
Console
Python 1
ex_floquet_quasienergies.py
00:34:13
Console
History log

```
    #quasi_energies = zeros((len(A_vec), 2))
    #f_gnd_prob     = zeros((len(A_vec), 2))
    quasi_energies = zeros((len(eps0_vec), 2))
    f_gnd_prob     = zeros((len(eps0_vec), 2))
    wf_gnd_prob    = zeros((len(eps0_vec), 2))

    for idx, eps0 in enumerate(eps0_vec):

        H0 = - delta/2.0 * sx - eps0/2.0 * sz
        H1 = A/2.0 * sz

        # H = H0 + H1 * sin(w * t) in the 'list-string' format
        H = [H0, [H1, 'sin(w * t)']]
        Hargs = {'w': omega}

        # find the floquet modes
        f_modes,f_energies = floquet_modes(H, T, Hargs)

        print "Floquet quasienergies[",idx,"] =", f_energies

        quasi_energies[idx,:] = f_energies

        f_gnd_prob[idx, 0] = expect(sm.dag() * sm, f_modes[0])
        f_gnd_prob[idx, 1] = expect(sm.dag() * sm, f_modes[1])

        f_states = floquet_states_t(f_modes, f_energies, 0, H, T, Hargs)

        wf_gnd_prob[idx, 0] = expect(sm.dag() * sm, f_states[0])
        wf_gnd_prob[idx, 1] = expect(sm.dag() * sm, f_states[1])

    return quasi_energies, f_gnd_prob, wf_gnd_prob

#
# set up the calculation: a strongly driven two-level system
# (repeated LZ transitions)
#
delta = 0.2 * 2 * pi  # qubit sigma_x coefficient
eps0  = 0.5 * 2 * pi  # qubit sigma_z coefficient
gamma1 = 0.0         # relaxation rate
gamma2 = 0.0          # dephasing  rate
A      = 2.0 * 2 * pi
```

Name	Type	Size	Value
A	float	1	12.566370614359172
T	float	1	1.0
delta	float	1	1.2566370614359172
e	float	1	2.718281828459045
eps0	float	1	3.141592653589793
gamma1	float	1	0.0
gamma2	float	1	0.0
numpy_requirement	str	1	1.6.0
omega	float	1	6.283185307179586
pi	float	1	3.141592653589793
qutip_rc_file	str	1	/Users/rob/.qutiprc

```
>>> eps0  = 0.5 * 2 * pi  # qubit sigma_z coefficient
>>> gamma1 = 0.0        # relaxation rate
>>> gamma2 = 0.0         # dephasing  rate
>>> A      = 2.0 * 2 * pi
>>> psi0   = basis(2,0)    # initial state
>>> omega  = 1.0 * 2 * pi # driving frequency
>>> T      = (2*pi)/omega  # driving period
>>>
>>> H0 = - delta/2.0 * sx - eps0/2.0 * sz
>>> H0
Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = True
Qobj data =
[[-1.57079633 -0.62831853]
 [-0.62831853  1.57079633]]
>>> |
```

Permissions: RW   End-of-lines: LF   Encoding: UTF-8-GUESSED   Line: 58   Column: 8

Anaconda Python

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

- Anaconda Python is a free Python distribution
- Used for data analytics, scientific computing
- Conda - an open-source package and environment manager
- Uses Python 2.7
- Launch the anaconda app and select the Ipython interface

Anaconda Python - conda

Example

```
To get help with the installation
$ conda -h
To install a package
$ conda install <pkg name>
You can also use the following
$ pip install <pkg name>
```

Anaconda Python - conda

Example

```
#To search for a package type
$ binstar search -t conda ggplot2
#Returns names of packages it can find, in
    this case asmeurer/r-ggplot2, r-old/r-
   ggplot2 and r/r-ggplot2 with name,
   version package type and platform
...
...
#Install package using the following
   command
$ conda install --channel https://conda.
   anaconda.org/r r-ggplot2
#More information here http://conda.pydata
   .org/docs/faq.html
```

Aside: Notation

We will use the following notation in these slides:

Example (Command Line)

```
$ python hello.py
```

Example (Python Interpreter)

```
>>> print ("Hello world!")
```

Example (.py File)

```
print("Hello World!")
```

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Hello World - hello.py !

NOTE: Indentation is very important in Python. It defines the extent of a code block.
Let us look at the file 'hello.py'

Example

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

print("Hello World!")
```

Python Interpreter

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 2 standard library of modules at `http://docs.python.org/2/library/`

Python Modules

Example

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

Example

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

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

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

```
def test_help():
        """Prints 'hello'."""
        print "hello"
```

Python Modules - Documentation

Introduction to Python
Srijith Rajamohan and Peter Radics
Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Example

```
>>> print(dir(math))
['__doc__', '__loader__', '__name__', '
    __package__', 'acos', 'acosh', 'asin',
    'asinh', 'atan', 'atan2', 'atanh', '
    ceil', 'copysign', 'cos', 'cosh', '
    degrees', 'e', 'erf', 'erfc', 'exp', '
    expm1', 'fabs', 'factorial', 'floor', '
    fmod', 'frexp', 'fsum', 'gamma', 'hypot
    ', 'isfinite', 'isinf', 'isnan', 'ldexp
    ', 'lgamma', 'log', 'log10', 'log1p', '
    log2', 'modf', 'pi', 'pow', 'radians',
    'sin', 'sinh', 'sqrt', 'tan', 'tanh', '
    trunc']
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Python Modules - Documentation

Example

```
>>> 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.
```

Python Modules

Python modules are cached. To reload a module, use the following

Example

```
>>> import imp
>>> imp.reload(os)
```

Python Modules

- When a module is imported, there are a list of directories that are searched
- Given by the sys.path

Example

```
>>> import sys
>>> sys.path
['',
 '/home/varoquau/.local/bin',
 '/usr/lib/python2.7',
 '/home/varoquau/.local/lib/python2.7/site
    -packages',
 '/usr/lib/python2.7/dist-packages',
 '/usr/local/lib/python2.7/dist-packages',
 ...]
```

Python Modules

- Modify the sys.path to add a directory

Example

```
>>> sys.path.append("/home/srijithr")
```

Directory and file operation

Get current directory

Example

```
>>> import os
>>> os.getcwd()
'/home/srijithr'
>>> os.curdir
'.'
>>> os.listdir(os.curdir)
['.index.rst.swo',
  'control_flow.rst',
 'debugging.rst',
 ...
```

Directory and file operation

- Make a directory and rename it

Example

```
>>> os.mkdir("junkdir")
>>> "junkdir" in os.listdir(os.curdir)
True
>>> os.rename("junkdir", "foodir")
>>> os.rmdir("foodir")
>>> open("junk.text",'w').close()
>>> a = os.path.abspath("junk.txt")
>>> os.remove("junk.txt")
```

Directory and file operation

Example

```
>>> os.path.split(a)
 ('/home/srijithr','junk.txt')
>>> os.path.dirname(a)
('/home/srijithr'
>>> os.path.basename(a)
 'junk.txt'
>>> os.path.splitext(os.path.basename(a))
('junk', '.txt')
>>> os.path.exists('junk.txt')
True
>>> os.path.isfile('junk.txt')
True
>>> os.path.isdir('junk.txt')
False
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Running external system commands

Example

```
# use os system commands
>>> os.system('ls')
# use the 'sh' module, may not be
   installed
>>> import sh
>>> com = sh.ls()
>>> print com.exit_code
0
```

Section 2

1 Introduction to Python

2 Python programming

3 NumPy

4 SciPy

5 Matplotlib

6 Debugging

7 Exception Handling

8 Model problems

9 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`’. A list of keywords are located at `https://docs.python.org/2.5/ref/keywords.html`
- Python is dynamically typed, the type of the variable is derived from the value it is assigned.
- A variable is assigned using the ‘=’ operator

Introduction to Python
Srijith Rajamohan and Peter Radics

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Variable naming

Variable names can make it easier (or harder) to understand a program!
Try to give variables clear, descriptive names!

Example

```
log_file = open("/var/log/syslog", "r")
userName = "pradics"
```

Avoid single-character names and abbreviations!

Example

```
f = open("/var/log/syslog", "r")
un = "pradics"
```

Variable types

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

- Variable types
  - Integer (`int`)
  - Float (`float`)
  - Boolean (`bool`)
  - Complex (`complex`)
  - String (`str`)
  - ...
  - User Defined! (classes)
- Documentation
  - https://docs.python.org/2/library/types.html
  - https://docs.python.org/2/library/datatypes.html

Variable types

- Use the `type` function to determine variable type

Example

```
>>> log_file = open("/var/log/syslog","r")
>>> type(log_file)
file
```

Variable types

- Variables can be *cast* to a different type

Example

```
>>> share_of_rent = 295.50 / 2.0
>>> type(share_of_rent)
float
>>> rounded_share = int(share_of_rent)
>>> type(rounded_share)
int
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

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

Operators

Example

```
>>> bar_tab = 35.28
>>> my_share = bar_tab / 3
>>> tip_amount = my_share * 0.2
>>> my_total = my_share + tip_amount
>>> enough_money = my_total < 15.00
>>> feeling_good = True
>>> good_night = enough_money and
   feeling_good
>>> print(my_total)
14.112
>>> print(enough_money)
True
>>> print(good_night)
True
```

Strings (str)

Example

```
>>> dir(str)
[..., 'capitalize', 'center', 'count', '
    decode', 'encode', 'endswith', '
    expandtabs', 'find', 'format', 'index',
     'isalnum', 'isalpha', 'isdigit', '
    islower', 'isspace', 'istitle', '
    isupper', 'join', 'ljust', 'lower', '
    lstrip', 'partition', 'replace', 'rfind
    ', 'rindex', 'rjust', 'rpartition', '
    rsplit', 'rstrip', 'split', 'splitlines
    ', 'startswith', 'strip', 'swapcase', '
    title', 'translate', 'upper', 'zfill']
```

Strings

Example

```
>>> greeting = "Hello world!"
>>> len(greeting)
12
>>> greeting
'Hello world'
>>> greeting[0] # indexing starts at 0
'H'
>>> greeting.replace("world", "test")
Hello test!
```

Printing strings

Example

```
# concatenates strings with a space
>>> print("Go", "Hokies")
Go Hokies
# concatenated without space
>>> print("Go" + "Tech" + "Go")
GoTechGo
# C-style string formatting
>>> print("Bar Tab = %f" %35.28)
Bar Tab = 35.280000
# Creating a formatted string
>>> total = "My Share = %.2f. Tip = %d" %
    (11.76, 2.352)
>>> print(total)
My Share = 11.76. Tip = 2
```

Lists

Array of elements of arbitrary type

Example

```
>>> numbers = [1,2,3]
>>> type(numbers)
list
>>> arbitrary_array = [1,numbers,"hello"]
>>> type(arbitrary_array)
list
```

Lists

Example

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

Lists

Lists are *mutable* - their values can be changed.

Example

```
>>> characters = ["A","d","d"]
# Changing second and third element
>>> characters[1] = "p"
>>> characters[2] = "p"
>>> print(characters)
['A', 'p', 'p']
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Lists

Example

```
>>> characters = ["A","d","d"]
# Inserting before "A","d","d"
>>> characters.insert(0, "i")
>>> characters.insert(1, "n")
>>> characters.insert(2, "s")
>>> characters.insert(3, "e")
>>> characters.insert(4, "r")
>>> characters.insert(5, "t")
>>>print(characters)
['i', 'n', 's', 'e', 'r', 't', 'A', 'd', '
   d']
```

Lists

Example

```
>>> characters = ['i', 'n', 's', 'e', 'r',
    't', 'A', 'd', 'd']
# Remove first occurrence of "A" from list
>>> characters.remove("A")
>>> print(characters)
['i', 'n', 's', 'e', 'r', 't', 'd', 'd']
# Remove an element at a specific location
>>> del characters[7]
>>> del characters[6]
>>> print(characters)
['i', 'n', 's', 'e', 'r', 't']
```

Tuples

Tuples are like lists except they are *immutable*. Difference is in performance

Example

```
>>> point = (10, 20)  # Note () for tuples
    instead of []
>>> type(point)
tuple
>>> point = 10,20
>>> type(point)
tuple
>>> point[2] = 40 # This will fail!
TypeError: 'tuple' object does not support
    item assignment
```

Dictionary

Dictionaries are lists of key-value pairs

Example

```
>>> prices = {"Eggs" : 2.30,
...           "Sausage" : 4.15,
...           "Spam" : 1.59,}
>>> type(prices)
dict
>>> print (prices)
{'Eggs': 2.3, 'Sausage': 4.15, 'Spam':
   1.59}
>>> prices["Spam"]
1.59
```

File I/O

- File modes denote how files are opened
- r for read-only mode
- w for write-only mode, this can overwrite existing files
- a for appending to a file
- r+ for read and write
- b for binary mode (in addition to one of the other modes)

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

File I/O

To write to a file use the following

Example

```
>>> work = open('workfile', 'w') # opens
   the workfile file
>>> type(work)
file
>>> work.write('Teach a python tutorial.')
>>> work.write('Be awesome.')
>>> work.close()
```

File I/O

To read from a file use the following

Example

```
>>> work = open('workfile', 'r')
>>> task = work.read()
>>> print(task)
Teach a python tutorial.
>>> task2 = work.read()
>>> print(task2)
Be awesome.
>>> work.close()
```

Conditional statements: if, elif, else

Example

```
>>> peter_is_tired = False
>>> peter_is_hungry = True
>>> if peter_is_tired is True:    # Note
   the colon for a code block
...     print ("You have to teach!")
... elif peter_is_hungry is True:
...     print ("No food for you!")
... else:
...     print "Go on...!"
...
No food for you!
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Loops - For

Example

```
>>> for i in [1,2,3]: # i is an arbitrary
    variable for use within the loop
    section
...     print(i)
1
2
3
>>> for word in ["scientific", "computing"
    , "with", "python"]:
...     print(word)
scientific
computing
with
python
```

Loops - While

Example

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

Functions

Example

```
>>> def print_word_length(word):
...     """
...     Print a word and how many
    characters it has
...     """
...     print(word + " has " + str(len(
    word)) + " characters.")
>>>print_word_length("Diversity")
Diversity has 9 characters.
```

Functions - arguments

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

- Passing immutable arguments like integers, strings or tuples acts like *call-by-value*
    - They cannot be modified!
- Passing mutable arguments like lists behaves like *call-by-reference*

Functions - arguments

Call-by-value

Example

```
>>> def make_me_rich(balance):
            balance = 1000000
account_balance = 500
>>> make_me_rich(account_balance)
>>> print(account_balance)
500
```

Functions - arguments

Call-by-reference

Example

```
>>> def talk_to_advisor(tasks):
            tasks.insert(0, "Publish")
            tasks.insert(1, "Publish")
            tasks.insert(2, "Publish")
>>> todos = ["Graduate","Get a job","...",
   "Profit!"]
>>> talk_to_advisor(todos)
>>> print(todos)
 ["Publish","Publish","Publish","Graduate"
    ,"Get a job","...","Profit!"]
```

Functions - arguments

- However, you cannot assign a new object to the argument
  - A new memory location is created for this list
  - This becomes a local variable

Example

```
>>> def switcheroo(favorite_teams):
...     print (favorite_teams)
...     favorite_teams = ["Cavaliers"]
...     print (favorite_teams)
>>> my_favorite_teams = ["Hokies", "German
    Soccer Team"]
>>> switcheroo(my_favorite_teams)
["Hokies", "German Soccer Team"]
["Cavaliers"]
>>> print (my_favorite_teams)
["Hokies", "German Soccer Team"]
```

Functions - Multiple Return Values

Example

```
>>> def powers(number):
...     return number ** 2, number ** 3
>>> squared, cubed = powers(3)
>>> print(squared)
9
>>> print(cubed)
27
```

Functions - Default Values

Example

```
>>> def likes_food(person, food="Broccoli"
    , likes=True):
...     if likes:
...         print(str(person) + " likes "
    + food)
...     else:
...         print(str(person) + " does not
    like " + food)
>>> likes_food("Peter", likes=False)
Peter does not like Broccoli
```

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

```
>>> 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]" % (
    self.x, self.y))
```

Classes

Example

```
# To create a new object
>>> origin = Point(0, 0) # this will
   invoke the __init__ method in the Point
    class
>>> print(origin)          # this will
   invoke the __str__ method
Point at [0.000000, 0.000000]
```

Section 3

1. Introduction to Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Debugging
7. Exception Handling
8. Model problems
9. Conclusion

NumPy

Used in almost all numerical computations in Python

- Used for high-performance vector and matrix computations
- Provides fast precompiled functions for numerical routines
- Written in C and Fortran
- Vectorized computations

Why NumPy?

Example

```
>>> from numpy import *
>>> import time
>>> def trad_version():
      t1 = time.time()
      X = range(10000000)
      Y = range(10000000)
      Z = []
      for i in range(len(X)):
        Z.append(X[i] + Y[i])
      return time.time() - t1

>>> trad_version()
1.9738149642944336
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Why NumPy?

Example

```
>>> def numpy_version():
        t1 = time.time()
        X = arange(10000000)
        Y = arange(10000000)
        Z = X + Y
        return time.time() - t1

>>> numpy_version()
 0.059307098388671875
```

Arrays

Example

```
>>> from numpy import *
#  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

```
>>> 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

```
>>> 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

```
>>> 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

```
>>> 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

```
>>> M = random.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

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Example

```
# Access the first row
>>> M[1]
array(
[ 0.01444674,  0.13963834,  0.36263224])
# The first row can be also be accessed
   using this notation
>>> M[1,:]
array(
[ 0.01444674,  0.13963834,  0.36263224])
# Access the first column
>>> M[:,1]
array(
[ 0.64335721,  0.13963834,  0.14865659])
```

Array Access

Example

```
# 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

Introduction to Python

Srijith Rajamohan and Peter Radics

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Example

```
# 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

```
# 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

```
# 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

```
>>> M*2
array([
[ 0.74778752,  1.28671443,  0.24871338],
[ 0.        ,  0.        ,  0.        ],
[ 0.01323804,  0.29731317,  1.50132603]])
>>> M + 2
array([
[ 2.37389376,  2.64335721,  2.12435669],
[ 2.        ,  2.        ,  2.        ],
[ 2.00661902,  2.14865659,  2.75066302]])
```

Matrix multiplication

Example

```
>>> M * M # Element-wise multiplication
array([
[1.397965e-01,4.139085e-01,1.546458e-02],
[0.000000e+00,0.000000e+00,0.00000e+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

```
>>> def Theta(x):
...     """
...     Scalar implemenation of the
    Heaviside step function.
...     """
...     if x >= 0:
...         return 1
...     else:
...         return 0
...
>>> Theta(1.0)
1
>>> Theta(-1.0)
0
```

Vectorize

Without `vectorize` we would not be able to pass v to the function

Example

```
>>> 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

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

Section 4

1. Introduction to Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Debugging
7. Exception Handling
8. Model problems
9. 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

Lets look at some examples

Using any of these subpackages requires an explicit import

- Linear Algebra
- Optimization

Get system parameters

Example

```
>>> 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 $\mathbf{A}\ \mathbf{x} = \mathbf{b}$

Example

```
>>> 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

```
>>> A = random.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]])
```

Linear Algebra - Eigenvalues and Eigenvector

Example

```
>>> 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]])
```

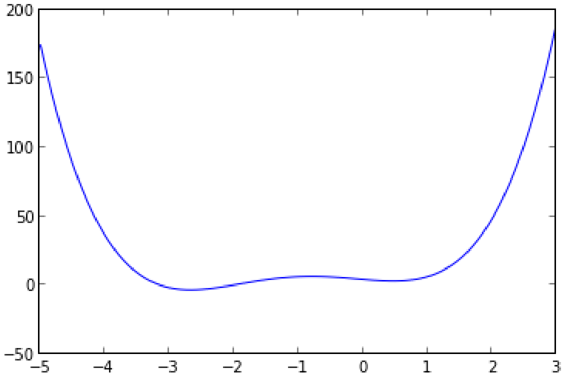
Optimization

Compute the minima of a single variable function

Example

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

Function f(x)



Optimization

Example

```
>>> 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 Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Debugging
7. Exception Handling
8. Model problems
9. 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 plotting
- 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`

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Matplotlib

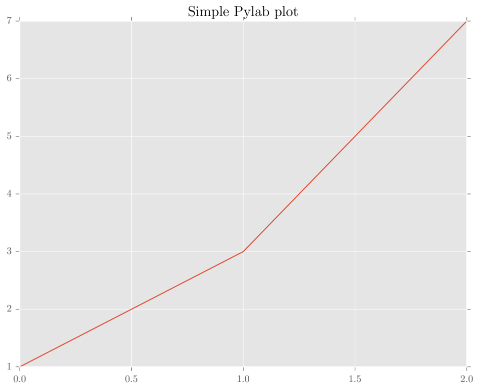
- Matplotlib is the entire library
- Pyplot - a module within Matplotlib that provides access to the underlying plotting library
- Pylab - a convenience module that combines the functionality of Pyplot with Numpy
- Pylab interface convenient for interactive plotting

Pylab

Example

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

Pylab

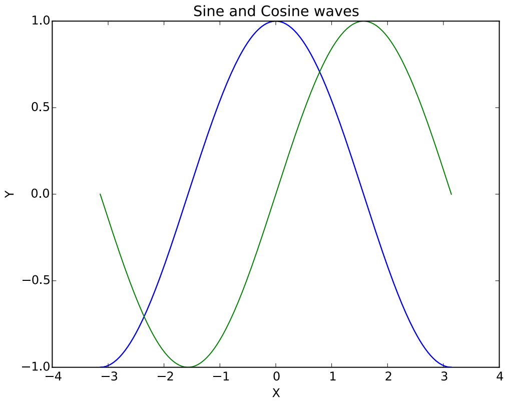


Pylab

Example

```
>>> X = np.linspace(-np.pi, np.pi, 256,
    endpoint=True)
>>> C, S = np.cos(X), np.sin(X)
# Plot cosine with a blue continuous line
    of width 1 (pixels)
>>> pl.plot(X, C, color="blue", linewidth
    =1.0, linestyle="-")
>>> pl.xlabel("X") ; pl.ylabel("Y")
>>> pl.title("Sine and Cosine waves")
# Plot sine with a green continuous line
    of width 1 (pixels)
>>> pl.plot(X, S, color="green", linewidth
    =1.0, linestyle="-")
>>> pl.show()
```

Pylab



Pylab - subplots

Introduction to Python

Srijith Rajamohan and Peter Radics

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Example

```
>>> pl.figure(figsize=(8, 6), dpi=80)
>>> pl.subplot(1, 2, 1)
>>> C, S = np.cos(X), np.sin(X)
>>> pl.plot(X, C, color="blue", linewidth
    =1.0, linestyle="-")
>>> pl.subplot(1, 2, 2)
>>> pl.plot(X, S, color="green", linewidth
    =1.0, linestyle="-")
>>> pl.show()
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

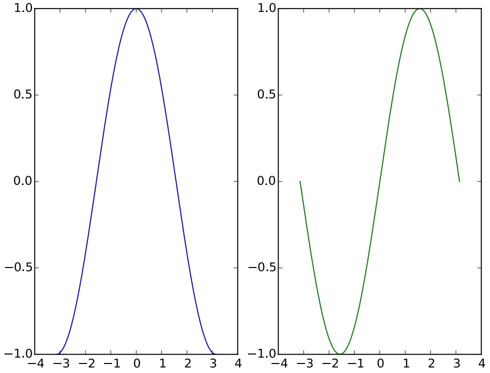
Debugging

Exception Handling

Model problems

Conclusion

Pylab - subplots



Pylab - xlim, ylim

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Example

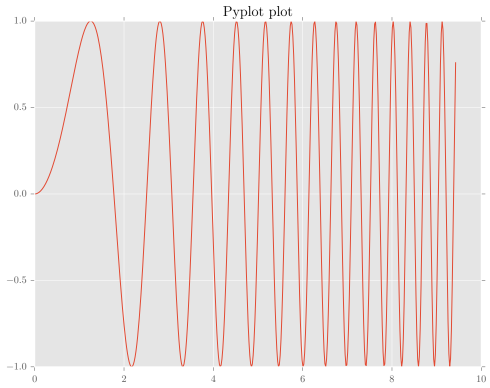
```
...
...
...
# Set x limits
>>> pl.xlim(-4.0, 4.0)
>>> pl.xticks(np.linspace(-4, 4, 9,
   endpoint=True))
# Set y limits
>>> pl.ylim(-1.0, 1.0)
# Set y ticks
>>> pl.yticks(np.linspace(-1, 1, 5,
   endpoint=True))
>>> pl.show()
```

Pyplot

Example

```
>>>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 0
   x104bf2b10>]
>>>plt.title('Pyplot plot')
<matplotlib.text.Text object at 0
   x104be4450>
>>>savefig('fig_test_pyplot.pdf',dpi=600,
   format='pdf')
>>>plt.show()
```

Pyplot

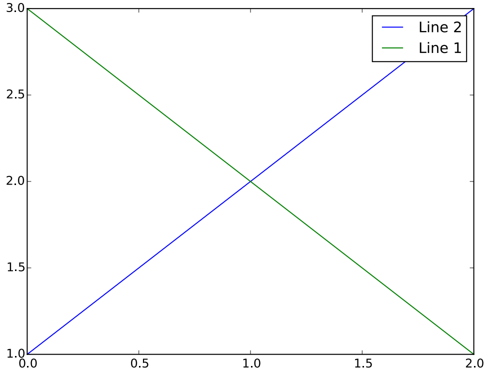


Pyplot - legend

Example

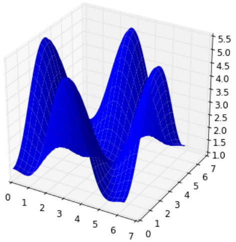
```
>>> import matplotlib.pyplot as plt
>>> line_up, = plt.plot([1,2,3], label='
    Line 2')
>>> line_down, = plt.plot([3,2,1], label='
    Line 1')
>>> plt.legend(handles=[line_up, line_down
    ])
<matplotlib.legend.Legend at 0x1084cc950>
>>> plt.show()
```

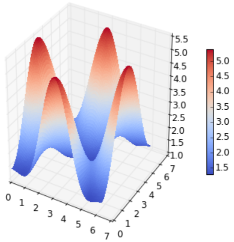
Pyplot - legend



Pyplot - 3D plots

Surface plots





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

Section 6

1. Introduction to Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Debugging
7. Exception Handling
8. Model problems
9. Conclusion

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Debugging

- Debugging - an essential tool for non-trivial code
- Make it fail reliably
- Attempt to isolate the offending section of code. Try to change only thing at a time when doing this !
- A systematic approach helps cut down on debugging time.

Pdb - Python debugger

Used to interactively step through the code and do the following:

- View the source code.
- Walk up and down the call stack.
- Inspect values of variables.
- Modify values of variables.
- Set breakpoints.

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Pdb - error.py

Example

```
def index_error():
    a = 5
    lst = list('foobar')
    print lst[len(lst)]

if __name__ == '__main__':
    index_error()
```

Pdb - Python debugger

Use it at the command line by invoking `pdb`

Example

```
$ python -m pdb error.py
```

or within IPython by using `run -d`

Example

```
>>> run -d error.py
```

Pdb - Python debugger

Type help within the debugger for interactive help

Example

```
ipdb> help
...
...
...
ipdb> help c
c(ont(inue))
Continue execution, only stop when a
   breakpoint is encountered.
```

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Some common pdb commands

l(list)	Lists the code at the current position
u(p)	Walk up the call stack
d(own)	Walk down the call stack
n(ext)	Execute the next line
s(tep)	Execute the next statement
bt	Print the call stack
a	Arguments to the current function

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Pdb - Python debugger

Step into error.py

Example

```
ipdb> n
> /Users/srijithrajamohan/error.py(4)
   index_error()
      3 def  index_error():
1---> 4       lst = list('foobar')
      5       print lst[len(lst)]
```

Section 7

1 Introduction to Python

2 Python programming

3 NumPy

4 SciPy

5 Matplotlib

6 Debugging

7 Exception Handling

8 Model problems

9 Conclusion

Exceptions

- Two different kinds of errors - syntax errors and exceptions
- Exceptions are detected using runtimes even when the program is syntactically correct
- Exceptions are raised by different kinds of errors when running your code
- Built-in `exceptions` module
- You can write your own exception-handling routines and error types

Exceptions - try/except

Use try/except to catch exceptions

Example

```
>>> while True:
      try:
          x = int(raw_input('Please
             enter a number: '))
          break
      except ValueError:
          print('That was no valid
             number. Try again...')
```

Exceptions - finally

Use finally to execute statements no matter what in a try statement

Example

```
>>> try:
        x = int(raw_input('Please enter a
            number: '))
    finally:
        print('Thank you for your input')
```

User-defined Exceptions

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Create a new class for user-defined exceptions derived from the `Exception` class

Example

```
class ValueTooSmallError(Error):
    """Raised when the input value is too
       small"""
    pass

class ValueTooLargeError(Error):
    """Raised when the input value is too
       large"""
    pass
```

User-defined Exceptions

Create a new class for user-defined exceptions

Example

```
while True:
   try:
    i_num = int(input("Enter a number: "))
    if i_num < number:
           raise ValueTooSmallError
    elif i_num > number:
           raise ValueTooLargeError
    break
   except ValueTooSmallError:
    print("This value is too small")
   except ValueTooLargeError:
    print("This value is too large")
print("You guessed it correctly.")
```

Section 8

1 Introduction to Python

2 Python programming

3 NumPy

4 SciPy

5 Matplotlib

6 Debugging

7 Exception Handling

8 Model problems

9 Conclusion

Model problems

We will cover 3 sample modeling problems

- Compute height and velocity of a thrown ball
- Compute the numerical derivative
- Solve a system of equations using the Newton's method

Compute height and velocity of a thrown ball

Equation for displacement as a function of time

$h(t) = 0.5 * g * t^2 + v_0 * t + h_0$

Equation for velocity as a function of time

$v(t) = v_0 + g * t$

$g =$ gravitational constant

$t =$ time

$h =$ height

$h_0 =$ initial height

$v =$ velocity

$v_0 =$ initial velocity

Compute numerical derivative

Compute the numerical derivative of analytical function
$y = sin(x)$
Compute the derivative using forward differences

$$[\frac{\partial y}{\partial x}]_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Compute the derivative using backward differences

$$[\frac{\partial y}{\partial x}]_i = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

Compute the derivative using central differences

$$[\frac{\partial y}{\partial x}]_i = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}$$

Solve a system of non-linear equations with the Newton's method

Introduction to Python
Python programming
NumPy
SciPy
Matplotlib
Debugging
Exception Handling
Model problems
Conclusion

Solve for 2 equations $f_0(a,b) = 0$ and $f_1(a,b) = 0$

$$f_0(a,b) = a^3 + b - 1 \tag{1}$$
$$f_1(a,b) = b - a + 1 \tag{2}$$

Define vectors **F** and **X**

$$\mathbf{F} = \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} \tag{3}$$

$$\mathbf{X} = \begin{bmatrix} a \\ b \end{bmatrix} \tag{4}$$

Solve this non-linear system of 2 equations using the Newton's method

Solve a system of non-linear equations with the Newton's method

$$Jacobian = \frac{\partial \mathbf{F}}{\partial \mathbf{X}} \tag{5}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial f_0}{\partial a} & \frac{\partial f_0}{\partial b} \\ \frac{\partial f_1}{\partial a} & \frac{\partial f_1}{\partial b} \end{bmatrix} \tag{6}$$

$$\Delta X = -\frac{\mathbf{F}}{Jacobian} \tag{7}$$

Solve a system of non-linear equations with the Newton's method

Algorithm

- Use an iterative process for finding the values of a and b
- Start with initial estimates of 0 for both a and b (**X**)
- Compute **F** and the $Jacobian$ and solve for ΔX. Loop until the norm of **F** becomes lower than a certain tolerance 1.0e-5

Section 9

1. Introduction to Python
2. Python programming
3. NumPy
4. SciPy
5. Matplotlib
6. Debugging
7. Exception Handling
8. Model problems
9. Conclusion

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

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

Introduction to Python

Python programming

NumPy

SciPy

Matplotlib

Debugging

Exception Handling

Model problems

Conclusion

Thank you for attending !