Scipy, Numpy and Biopython

What are modules and packages?

Essentially just python code in other files.

Module: One file
(which usually contains many functions)

Package: Collection of many files(modules)

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

```
from sound.effects import echo
```

```
echo.echofilter(input, output, delay=0.7, atten=4)
```

stdlib: packages included with python as default

Scipy Stack:

- Python
- NumPy
- SciPy library
- Matplotlib
- pandas
- SymPy

Easiest way to install:
Anaconda distribution

Download:
https://www.continuum.io/downloads

Packages included in Anaconda:
https://docs.continuum.io/anaconda/pkg-docs

- Array oriented computing
- 1D, 2D, 3D arrays
- Faster due to calculating in C (static array)
- Lots of data? Use this

>>> import numpy as np

```
>>> a = np.arange(6)                         # 1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4,3)           # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>>
>>> c = np.arange(24).reshape(2,3,4)         # 3d array
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

More NumPy examples

Modify arrays using vectors

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print y  # Prints "[[ 2  2  4]
         #          [ 5  5  7]
         #          [ 8  8 10]
         #          [11 11 13]]"
```

Modify arrays using indices

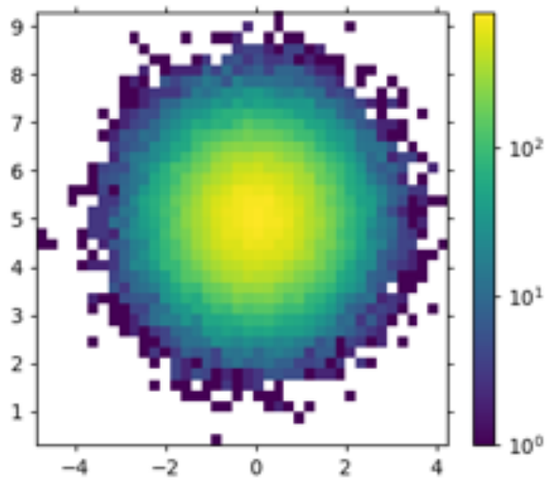
```
import numpy as np

# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print a  # prints "array([[ 1,  2,  3],
         #                [ 4,  5,  6],
         #                [ 7,  8,  9],
         #                [10, 11, 12]])"

# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print a[np.arange(4), b]  # Prints "[ 1  6  7 11]"

# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10

print a  # prints "array([[11,  2,  3],
         #                [ 4,  5, 16],
         #                [17,  8,  9],
         #                [10, 21, 12]])
```

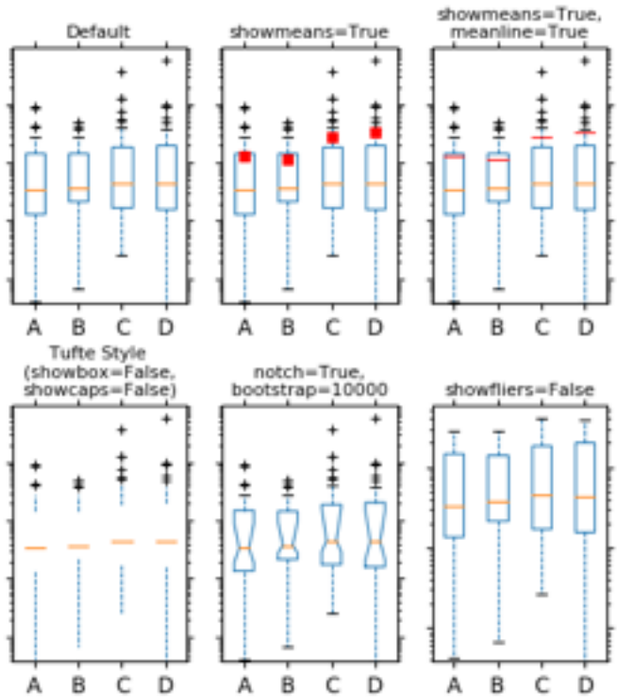
Array math

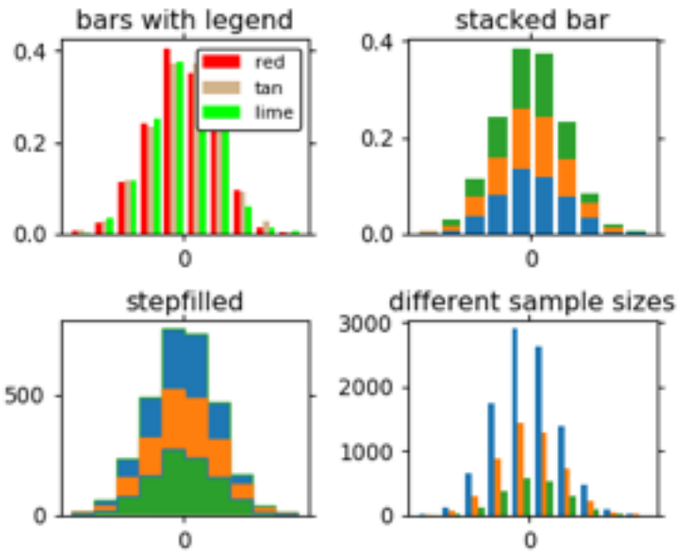
```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print x + y
print np.add(x, y)

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print x - y
print np.subtract(x, y)

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print x * y
print np.multiply(x, y)

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print x / y
print np.divide(x, y)

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print np.sqrt(x)
```

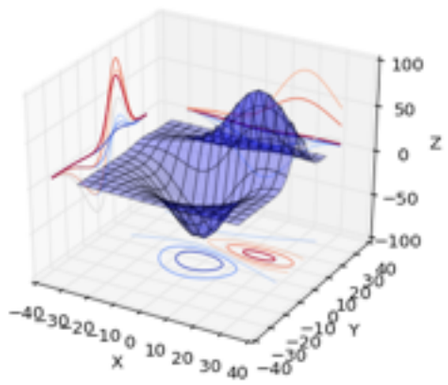
matplotlib

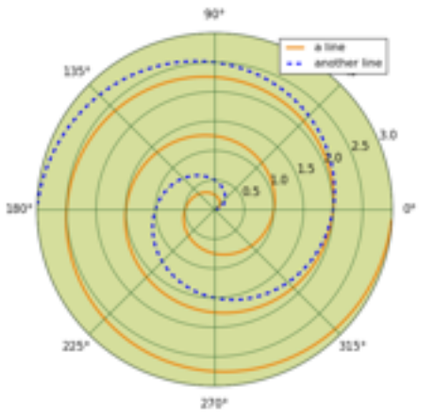
Turn your data into graphs!







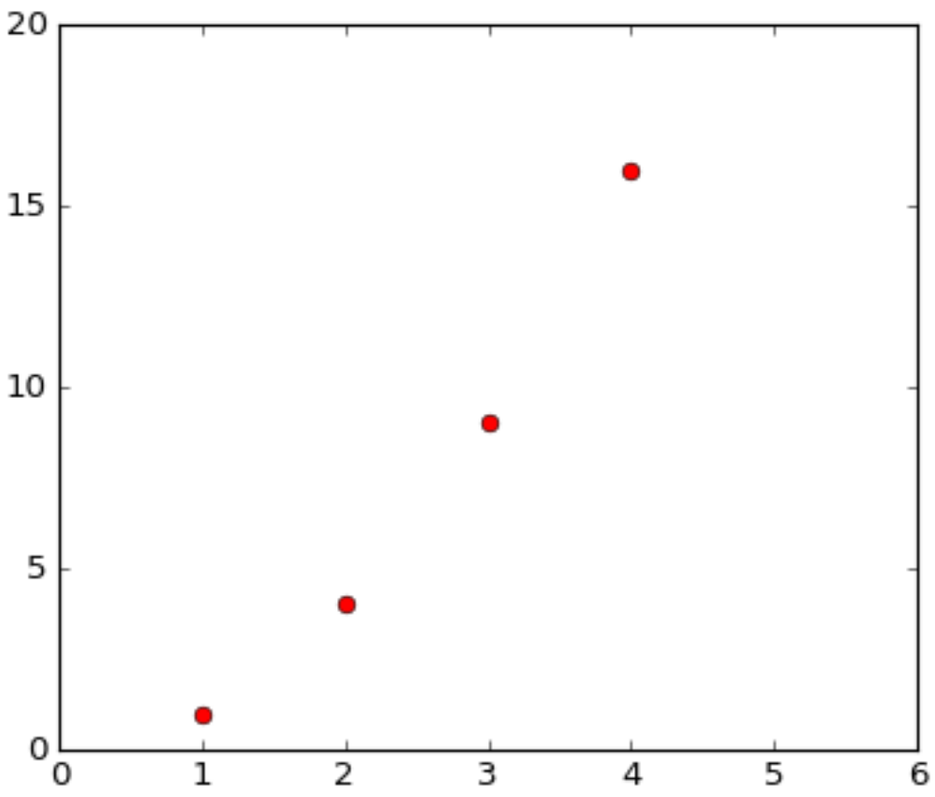




http://matplotlib.org/users/beginner.html

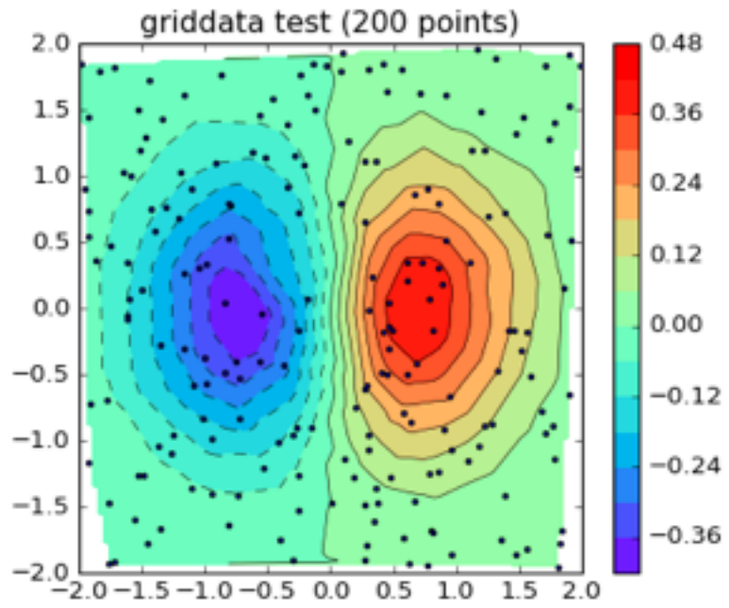
matplotlib

Simple




```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

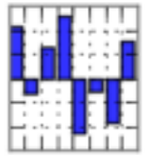
Less simple

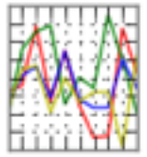


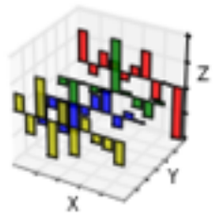

```
from numpy.random import uniform, seed
from matplotlib.mlab import griddata
import matplotlib.pyplot as plt
import numpy as np
# make up data.
#npts = int(raw_input('enter # of random points to plot:'))
seed(0)
npts = 200
x = uniform(-2, 2, npts)
y = uniform(-2, 2, npts)
z = x*np.exp(-x**2 - y**2)
# define grid.
xi = np.linspace(-2.1, 2.1, 100)
yi = np.linspace(-2.1, 2.1, 200)
# grid the data.
zi = griddata(x, y, z, xi, yi, interp='linear')
# contour the gridded data, plotting dots at the nonuniform data points.
CS = plt.contour(xi, yi, zi, 15, linewidths=0.5, colors='k')
CS = plt.contourf(xi, yi, zi, 15, cmap=plt.cm.rainbow,
                  vmax=abs(zi).max(), vmin=-abs(zi).max())
plt.colorbar()  # draw colorbar
# plot data points.
plt.scatter(x, y, marker='o', c='b', s=5, zorder=10)
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.title('griddata test (%d points)' % npts)
plt.show()
```

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$







Table/database treatment (similar to SQL)

```
In [6]: dates = pd.date_range('20130101', periods=6)

In [7]: dates
Out[7]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')

In [8]: df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

In [9]: df
Out[9]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

http://pandas.pydata.org/pandas-docs/stable/10min.html#min

- Symbolic mathematics
- Useful for algebraic equations

http://docs.sympy.org/latest/tutorial/

Modify equations

```
>>> import sympy
>>> x, y = sympy.symbols(‘x y')
>>> expr = x + 2*y
>>> expr
```

$$x+2y$$

```
>>> expr - x
```

$$2y$$

```
>>> expanded_expr = expand(x*expr)
>>> expanded_expr
```

$$x^2+2xy$$

```
>>> factor(expanded_expr)
```

$$x(x+2y)$$

IP[y]: IPython
Interactive Computing

An extended interactive shell.

Functionality includes:

- Tab auto-completion
- Magic commands (%)
- Shell commands (!)
  ex. “files = !ls”
- Configuration & aliases

command	description
?	Introduction and overview of IPython’s features.
%quickref	Quick reference.
help	Python’s own help system.
object?	Details about ‘object’, use ‘object??’ for extra details.

- Designed specifically with biology data in mind
- Lots of modules that work with sequences
- Parsing!

Seq object:

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq("AGTACACTGGT")
>>> my_seq
Seq('AGTACACTGGT', Alphabet())
>>> print(my_seq)
AGTACACTGGT
>>> my_seq.complement()
Seq('TCATGTGACCA', Alphabet())
>>> my_seq.reverse_complement()
Seq('ACCAGTGTACT', Alphabet())
```

Transcribing, translating, rev/comp

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> coding_dna
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
>>> messenger_rna = coding_dna.transcribe()
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
>>> coding_dna.reverse_complement().transcribe()
Seq('CUAUCGGGCACCCUUUCAGCGGCCCAUUACAAUGGCCAU', IUPACUnambiguousRNA())
>>> coding_dna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

More BIOPYTHON examples

Parsing a fasta file

```
from Bio import SeqIO
for seq_record in SeqIO.parse("/home/xalmal/SSQXT-19_S5_L001_R1_001.fasta", "fasta"):
    print(seq_record.id)
    print(repr(seq_record.seq))
    print(len(seq_record))
```

Web BLASTn:

```
>>> from Bio.Blast import NCBIWWW
>>> fasta_string = open(“your_sequences.fasta").read()
>>> result_handle = NCBIWWW.qblast("blastn", "nt", fasta_string)
>>> result_handle.read()
```

Local BLASTn:

```
>>> from Bio.Blast.Applications import NcbiblastxCommandline
>>> blastx_cline = NcbiblastxCommandline(query="opuntia.fasta", db="nr",
                   evalue=0.001, outfmt=5, out="opuntia.xml")
```

Installing python packages

Windows

OS X

GNU/Linux

It's Easy (On Mac):

1. Install easy_install
   ```
   curl https://bootstrap.pypa.io/ez_setup.py -o - | sudo python
   ```
2. Install pip
   ```
   sudo easy_install pip
   ```
3. Now, you could install external modules. For example
   ```
   pip install regex   # This is only an example for installing other modules
   ```

share improve this answer

edited Nov 4 '15 at 0:13 — Kevin Guan — 10.4k ● 9 ● 23 ● 47

answered Mar 21 '15 at 22:12 — Pavan — 870 ● 1 ● 6 ● 11

http://blog.troygrosfield.com/2010/12/18/installing-easy_install-and-pip-for-python/

```
sudo easy_install -f http://biopython.org/DIST/ biopython
```

From source (download tarball)
or via package handler
(apt-get/rpm/yum)

Anaconda: conda install <package>
(conda install pip -> pip install <package>)

Summary

Numpy:	Calculations on data
matplotlib:	Graphs
pandas:	Tables
SymPy:	Equations
IPython:	Useful coding tool
Biopython:	Sequence handling

These slides can be found at:
https://github.com/alvaralmstedt/Tutorials/blob/master/numpy_scipy_biopython.pdf