NumPy

Software Application Layers

Domain Specific GUI Applications
Semiconductor, Fluid Dynamics, Seismic Modeling, Financial, etc.

ETS (App construction)
Traits, Chaco, Envisage, Mayavi, etc.

SciPy (Scientific Algorithms)

NumPy (Array Mathematics)

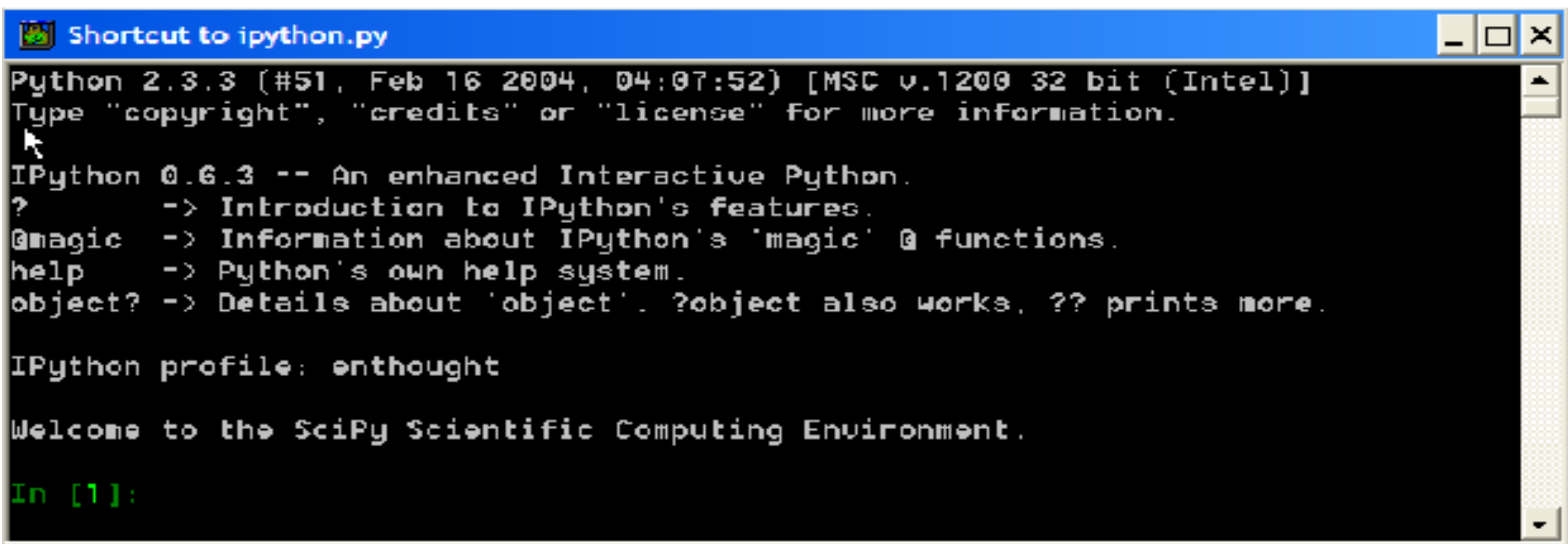
3rd Party Libraries
wxPython
VTK, etc.

Python

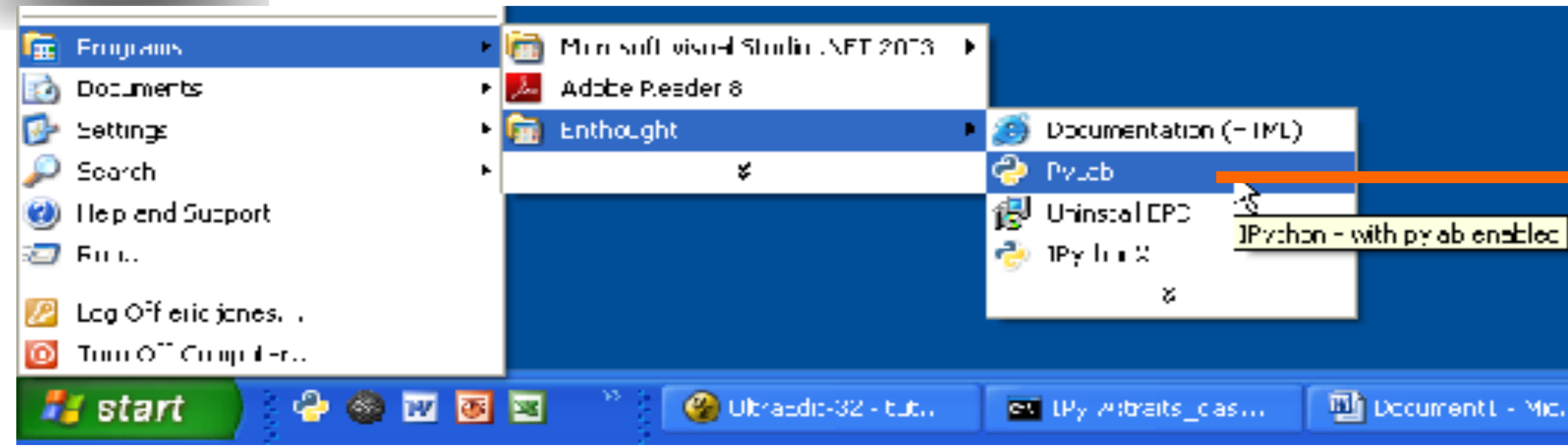
IPython

IPython command prompt

- Available at http://ipython.scipy.org/
- Fernando Perez, Brian Granger, and others
- Provides a nice environment for scientific computing with Python

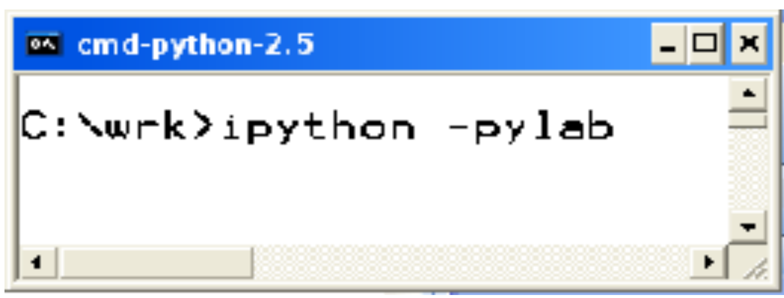


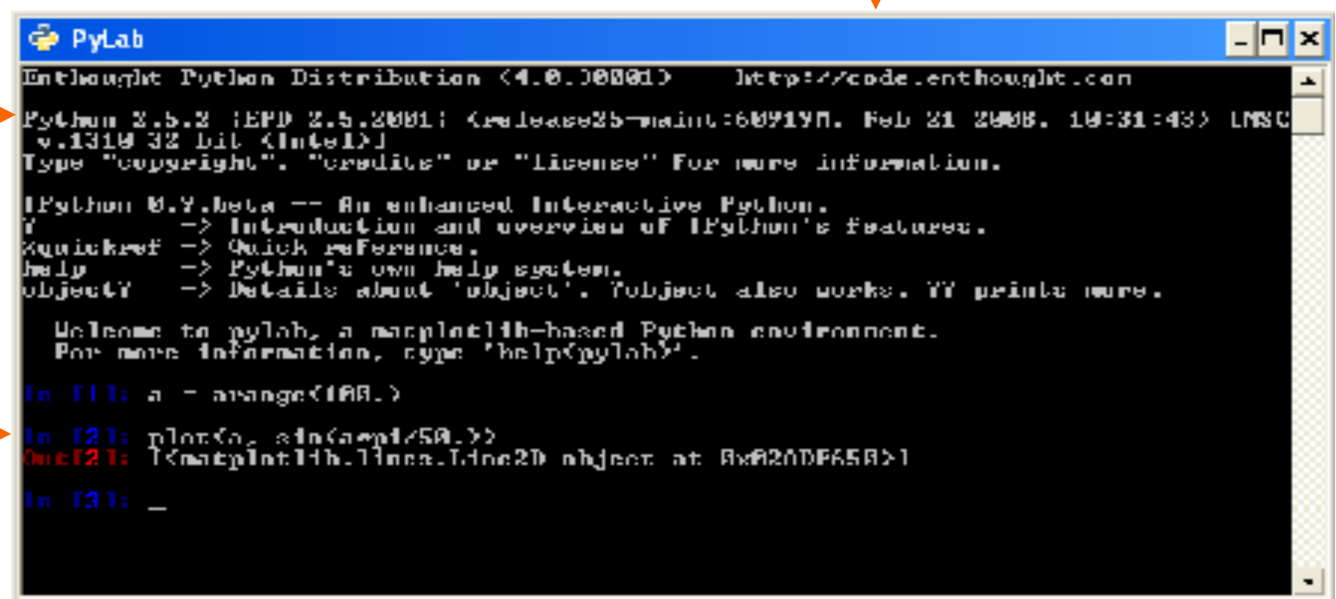
Starting PyLab



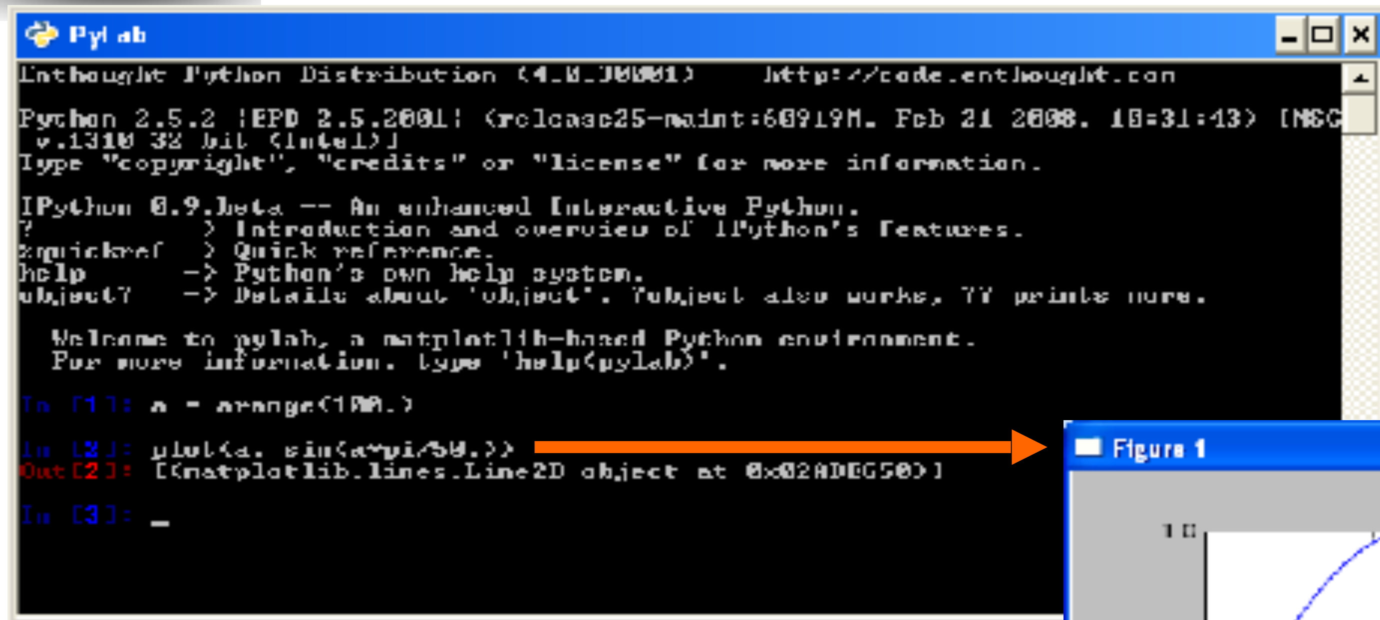
or...

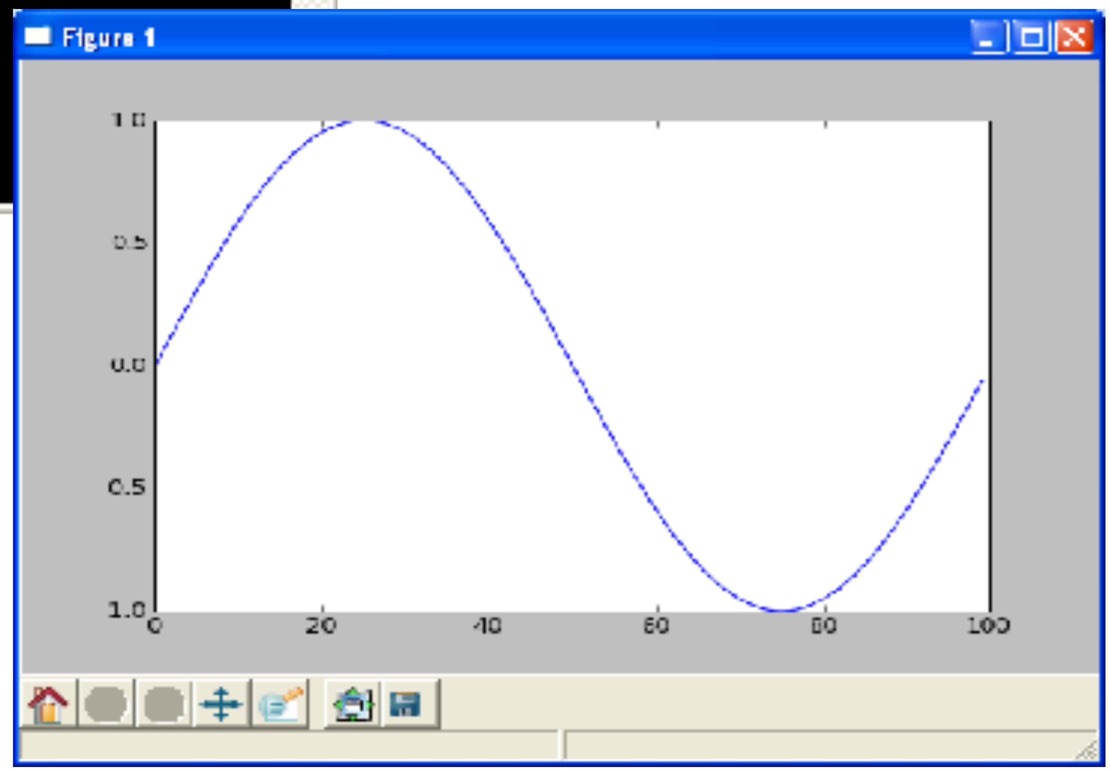
or...





PyLab: Interactive Python Environment





The PyLab mode in IPython handles some gory details behind the scenes. It allows both the Python command interpreter (above) and the GUI plot window (right) to coexist. This involves a bit of multi-threaded magic.

PyLab also imports some handy functions into the command interpreter for user convenience

IPython

STANDARD PYTHON

```
In [1]: a=1

In [2]: a
Out[2]: 1
```

HISTORY COMMAND

```
# List previous commands. Use
# 'magic' % because 'hist' is
# histogram function in pylab.
In [3]: %hist
1: a=1
2: a
```

INPUT HISTORY

```
# list string from prompt[2]
In [4]: _i2
Out[4]: 'a\n'
```

OUTPUT HISTORY

```
# grab result from prompt[2]
In [5]: _2
Out[5]: 1
```

AVAILABLE VARIABLES

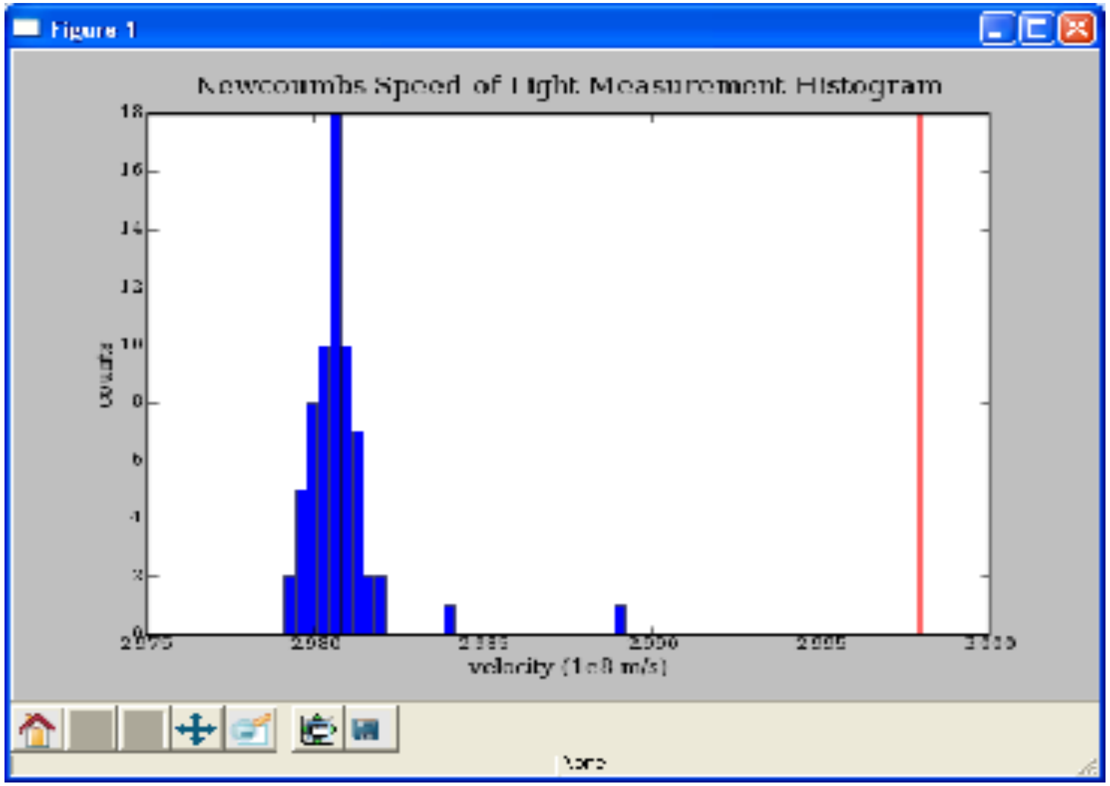
```
In [6]: b = [1,2,3]

# list available variables
In [7]: whos
Variable    Type    Data/Length
-----------------------------
a           int      1
b           list     [1, 2, 3]
```

Directory Navigation

```
# change directory (note Unix style forward slashes!)
In [9]: cd c:/demo/speed_of_light
c:\demo\speed_of_light

# list directory contents
In [10]: ls
 Volume in drive C has no label.
 Volume Serial Number is 5417-593D
 Directory of c:\demo\speed_of_light
09/01/2008  02:53 PM  <DIR>          .
09/01/2008  02:53 PM  <DIR>          ..
09/01/2008  02:48 PM           1,188 exercise_speed_of_light.txt
09/01/2008  02:48 PM       2,682,023 measurement_description.pdf
09/01/2008  02:48 PM         187,087 newcomb_experiment.pdf
09/01/2008  02:48 PM           1,312 speed_of_light.dat
09/01/2008  02:48 PM           1,436 speed_of_light.py
09/01/2008  02:48 PM           1,232 speed_of_light2.py
               6 File(s)      2,874,278 bytes
               2 Dir(s) 11,997,437,952 bytes free
```

Running Scripts

```
# tab completion
In [11]: run speed_of_li
speed_of_light.dat  speed_of_light.py

# execute a python file
In [11]: run speed_of_light.py
```



Function Info -- Magic Commands

HELP USING ?

```
# Follow a command with '?' to print its documentation.

In [19]: len?
Type:           builtin_function_or_method
Base Class:     <type 'builtin_function_or_method'>
String Form:    <built-in function len>
Namespace:      Python builtin
Docstring:
    len(object) -> integer

Return the number of items of a sequence or mapping.
```

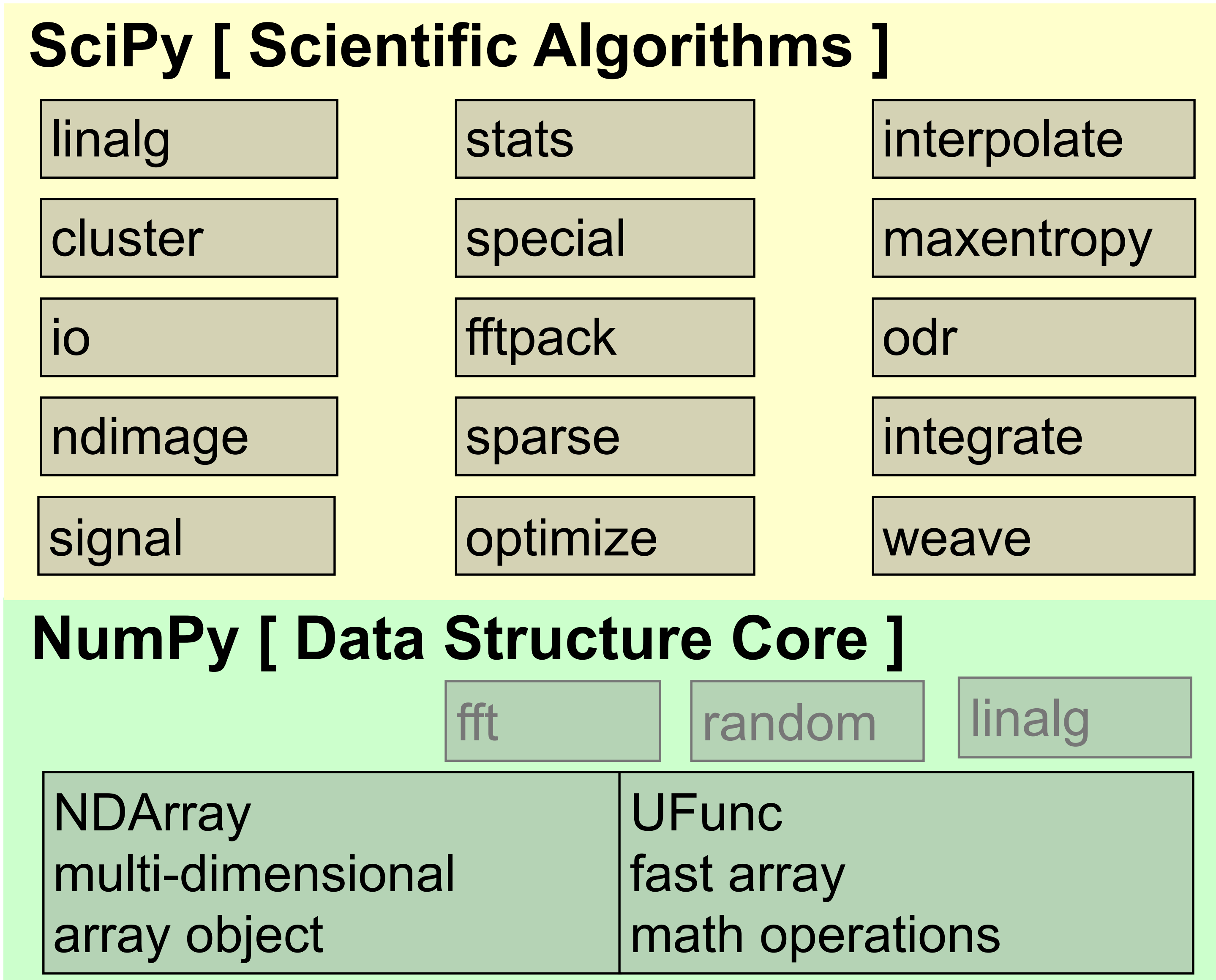
Function Info -- Magic Commands

SHOW SOURCE CODE USING ??

```
# Follow a command with '??' to print its source code.
In [43]: squeeze??
def squeeze(a):
    """Remove single-dimensional entries from the shape of a.
    Examples
    --------
    >>> x = array([[[1,1,1],[2,2,2],[3,3,3]]])
    >>> x.shape
    (1, 3, 3)
    >>> squeeze(x).shape
    (3, 3)
    """
    try:
        squeeze = a.squeeze
    except AttributeError:
        return _wrapit(a, 'squeeze')
    return squeeze()
```

`??` can't show the source code for "extension" functions that are implemented in C.

NumPy and SciPy



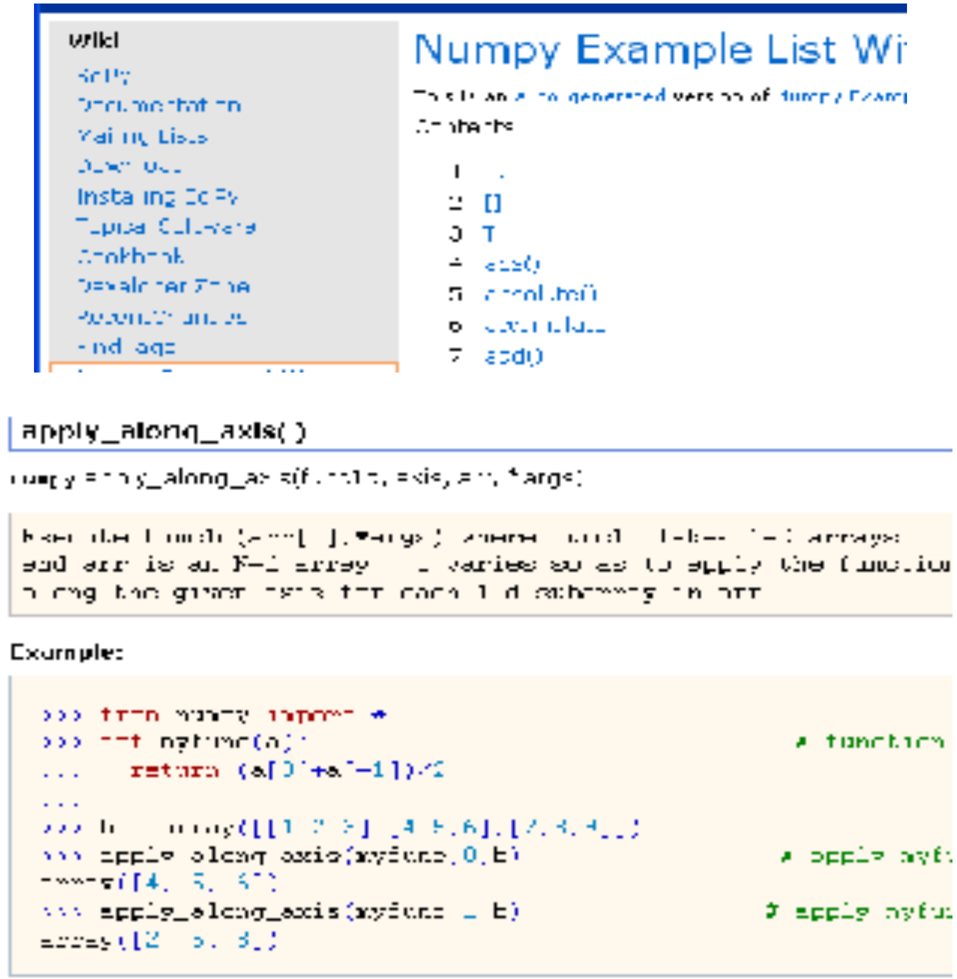
Helpful Sites

SCIPY DOCUMENTATION PAGE

http://www.scipy.org/Documentation

NUMPY EXAMPLES

http://www.scipy.org/Numpy_Example_List_With_Doc



Getting Started

IMPORT NUMPY

```
>>> from numpy import *
>>> __version__
1.0.2.dev3487
          or
>>> from numpy import array, ...
```

Often at the command line, it is handy to import everything from NumPy into the command shell.

However, if you are writing scripts, it is easier for others to read and debug in the future if you use explicit imports.

USING IPYTHON -PYLAB

```
C:\> ipython –pylab
In [1]: array((1,2,3))
Out[1]: array([1, 2, 3])
```

IPython has a 'pylab' mode where it imports all of NumPy, Matplotlib, and SciPy into the namespace for you as a convenience.

While IPython is used for all the demos, '>>>' is used on future slides instead of '`In [1]:`' because it takes up less room.

Array Operations

SIMPLE ARRAY MATH

```
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
```

NumPy defines the following constants:

pi = 3.14159265359

e = 2.71828182846

MATH FUNCTIONS

```
# create array from 0 to 10
>>> x = arange(11.)

# multiply entire array by
# scalar value
>>> a = (2*pi)/10.
>>> a
0.62831853071795862
>>> a*x
array([ 0.,0.628,…,6.283])

# in-place operations
>>> x *= a
>>> x
array([ 0.,0.628,…,6.283])
# apply functions to array
>>> y = sin(x)
```

Introducing NumPy Arrays

SIMPLE ARRAY CREATION

```
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

CHECKING THE TYPE

```
>>> type(a)
<type 'numpy.ndarray'>
```

NUMERIC ‘TYPE’ OF ELEMENTS

```
>>> a.dtype
dtype('int32')
```

BYTES PER ELEMENT

```
>>> a.itemsize
4
```

ARRAY SHAPE

```
# Shape returns a tuple
# listing the length of the
# array along each dimension.
>>> a.shape
(4,)
>>> shape(a)
(4,)
```

ARRAY SIZE

```
# Size reports the entire
# number of elements in an
# array.
>>> a.size
4
>>> size(a)
4
```

Introducing NumPy Arrays

BYTES OF MEMORY USED

```
# Return the number of bytes
# used by the data portion of
# the array.
 >>> a.nbytes
16
```

NUMBER OF DIMENSIONS

```
>>> a.ndim
1
```

ARRAY COPY

```
# Create a copy of the array.
>>> b = a.copy()
>>> b
array([0, 1, 2, 3])
```

CONVERSION TO LIST

```
# Convert a NumPy array to a
# Python list.
>>> a.tolist()
[0, 1, 2, 3]

# For 1-D arrays, list
# works equivalently, but
# is slower.
>>> list(a)
[0, 1, 2, 3]
```

Setting Array Elements

ARRAY INDEXING

```
>>> a[0]
0
>>> a[0] = 10
>>> a
[10, 1, 2, 3]
```

FILL

```
# set all values in an array
>>> a.fill(0)
>>> a
[0, 0, 0, 0]

# this also works, but may
# be slower
>>> a[:] = 1
>>> a
[1, 1, 1, 1]
```

BEWARE OF TYPE COERSION

```
>>> a.dtype
dtype('int32')

# assigning a float into
# an int32 array truncates
# the decimal part
>>> a[0] = 10.6
>>> a
[10, 1, 2, 3]

# fill has the same behavior
>>> a.fill(-4.8)
>>> a
[-4, -4, -4, -4]
```

Arrays from ASCII Data

BASIC PATTERN

```
# Read data into a list of lists,
# and then convert to an array.
file = open('myfile.txt')

# Create a list for all the data.
data = []

for line in file:
  # Read each row of data into a
  # list of floats.
  row_data = [float(x) for x in
               line.split()]
  # And add this row to the
  # entire data set.
  data.append(row_data)
# Finally, convert the "list of
# lists" into a 2D array.
data = array(data)
f.close()
```

COMMA SEPARATED FILES

```
# The csv module is also handy.
import csv
f = open('myfile.txt')
reader = csv.reader(f)

# Create a list for all the data.
data = []

# Note that the reader has
# already done the “split”
for line in reader:
  row_data = [float(x) for x in
               line]
  data.append(row_data)

# Finally, convert the “list of
# lists” into a 2D array.
data = array(data)
f.close()
```

Slicing

var[lower:upper:step]

Extracts a portion of a sequence by specifying a lower and upper bound.
The lower-bound element is included, but the upper-bound element is not included.
Mathematically: [lower,upper). The step value specifies the stride between elements.

SLICING LISTS

```
# indices:     0   1   2   3   4
>>> l = array([10,11,12,13,14])
# [10,11,12,13,14]
>>> l[1:3]
[11, 12]

# negative indices work also
>>> l[1:-2]
[11, 12]
>>> l[-4:3]
[11, 12]
```

OMITTING INDICES

```
# omitted boundaries are
# assumed to be the beginning
# (or end) of the list

# grab first three elements
>>> l[:3]
[10, 11, 12]
# grab last two elements
>>> l[-2:]
[13, 14]
# every other element
>>> l[::2]
[10, 12, 14]
```

Multi-Dimensional Arrays

MULTI-DIMENSIONAL ARRAYS

```
>>> a = array([[ 0, 1, 2, 3],
               [10,11,12,13]])
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,13]])
```

(ROWS,COLUMNS)

```
>>> a.shape
(2, 4)
>>> shape(a)
(2, 4)
```

ELEMENT COUNT

```
>>> a.size
8
>>> size(a)
8
```

NUMBER OF DIMENSIONS

```
>>> a.ndims
2
```

GET/SET ELEMENTS

```
>>> a[1,3]
13
```

column
row

```
>>> a[1,3] = -1
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

ADDRESS FIRST ROW USING SINGLE INDEX

```
>>> a[1]
array([10, 11, 12, -1])
```

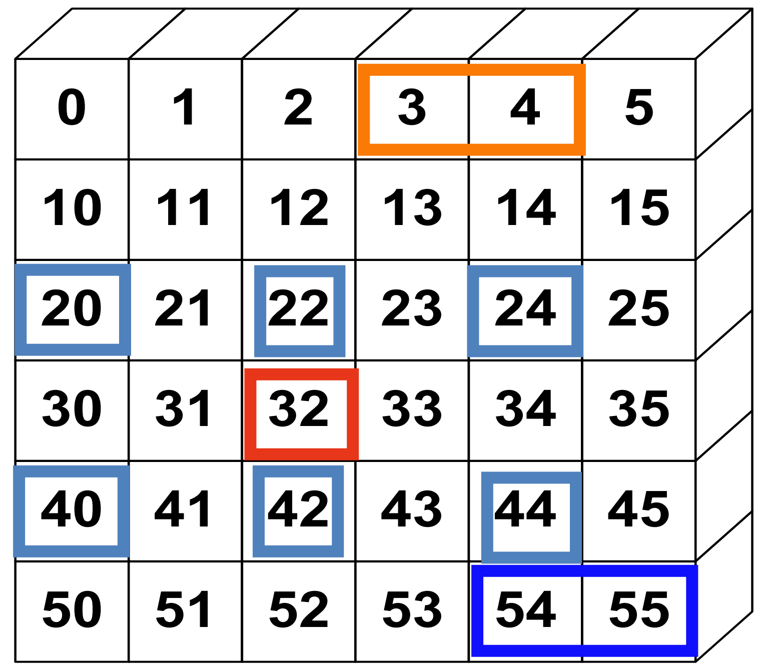
Array Slicing

SLICING WORKS MUCH LIKE STANDARD PYTHON SLICING

```
>>> a[0,3:5]
array([3, 4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])
```

STRIDES ARE ALSO POSSIBLE

```
>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Slices Are References

Slices are references to memory in the original array.

Changing values in a slice also changes the original array.

```
>>> a = array((0,1,2,3,4))

# create a slice containing only the
# last element of a
>>> b = a[2:4]
>>> b
array([2, 3])
>>> b[0] = 10

# changing b changed a!
>>> a
array([ 1,  2, 10, 3, 4])
```

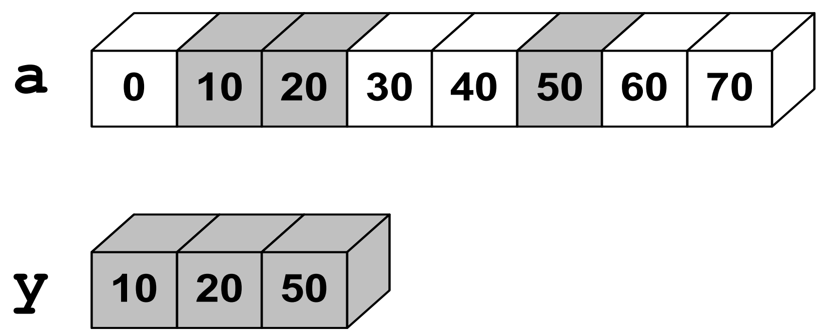
Fancy Indexing

INDEXING BY POSITION

```
>>> a = arange(0,80,10)


# fancy indexing
>>> y = a[[1, 2, -3]]
>>> print y
[10 20 50]

# using take
>>> y = take(a,[1,2,-3])
>>> print y
[10 20 50]
```

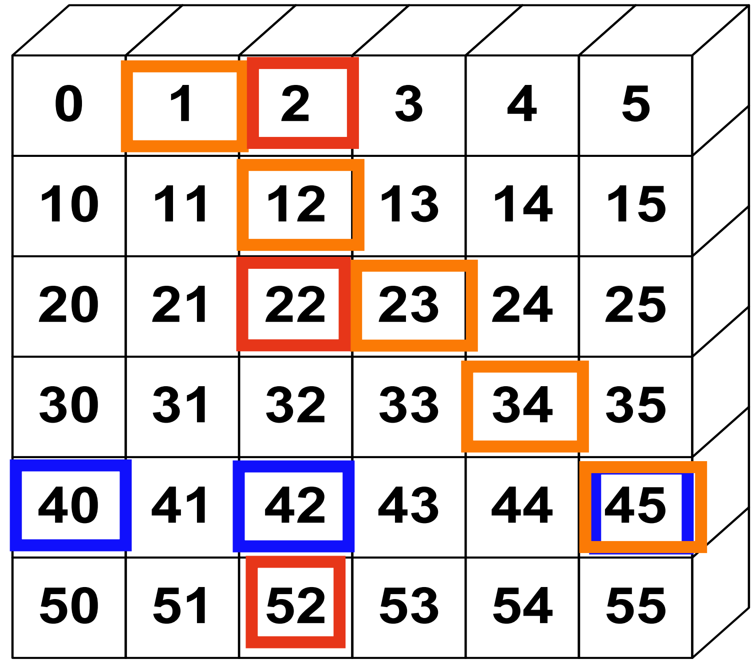
INDEXING WITH BOOLEANS

```
>>> mask = array([0,1,1,0,0,1,0,0],
...                     dtype=bool)

# fancy indexing
>>> y = a[mask]
>>> print y
[10,20,50]

# using compress
>>> y = compress(mask, a)
>>> print y
[10,20,50]
```



Fancy Indexing in 2-D

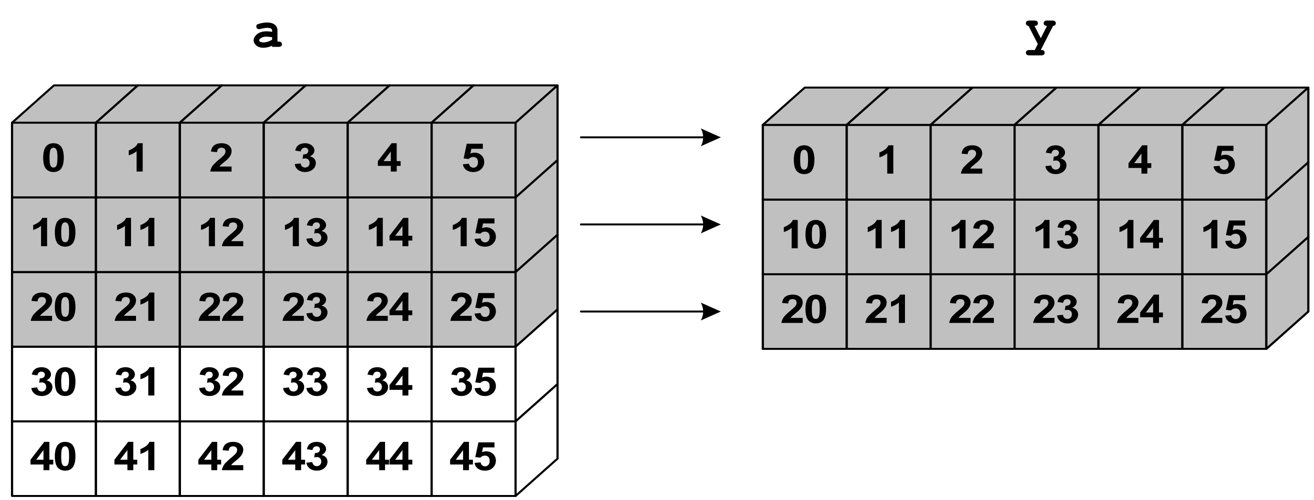
```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```



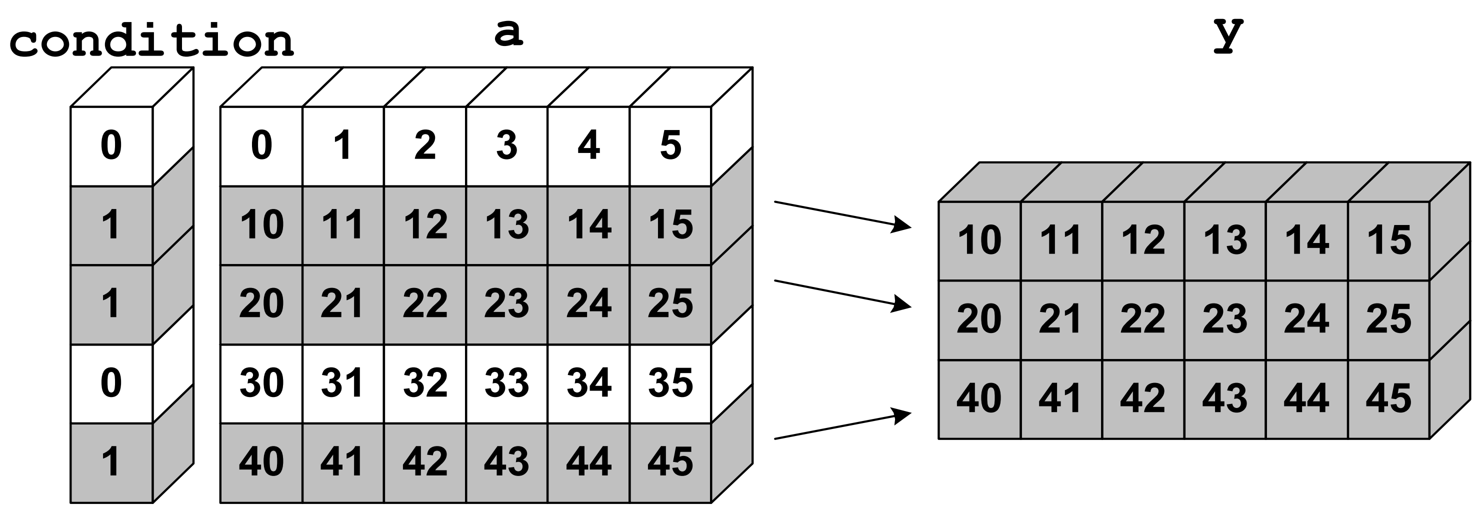
Unlike slicing, fancy indexing creates copies instead of views into original arrays.

“Incomplete” Indexing

```
>>> y = a[:3]
```



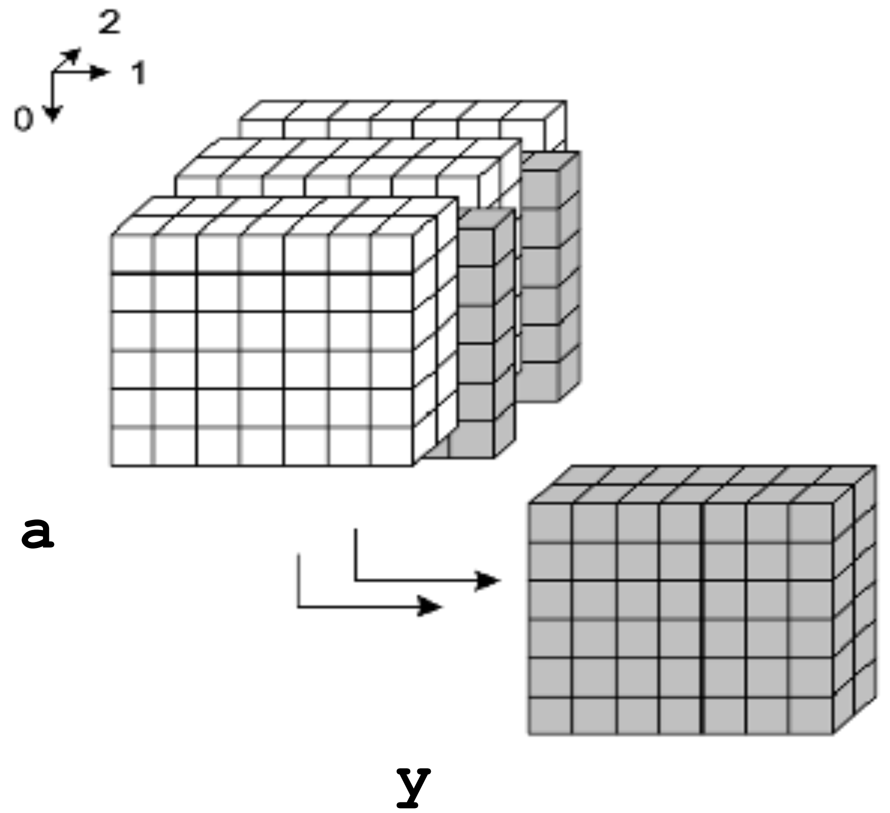

```
>>> y = a[condition]
```



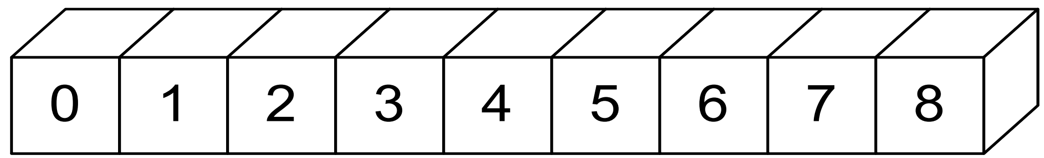
3-D Example

MULTIDIMENSIONAL

```
# retrieve two slices from a
# 3-D cube via indexing
>>> y = a[:,:,[2,-2]]

# the take() function also works
>>> y = take(a,[2,-2], axis=2)
```

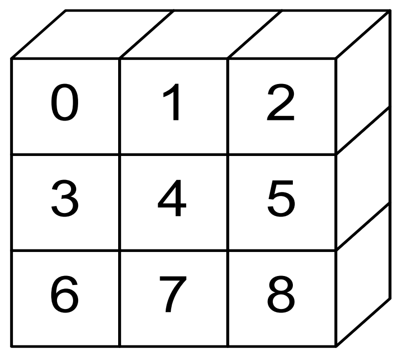


Array Data Structure



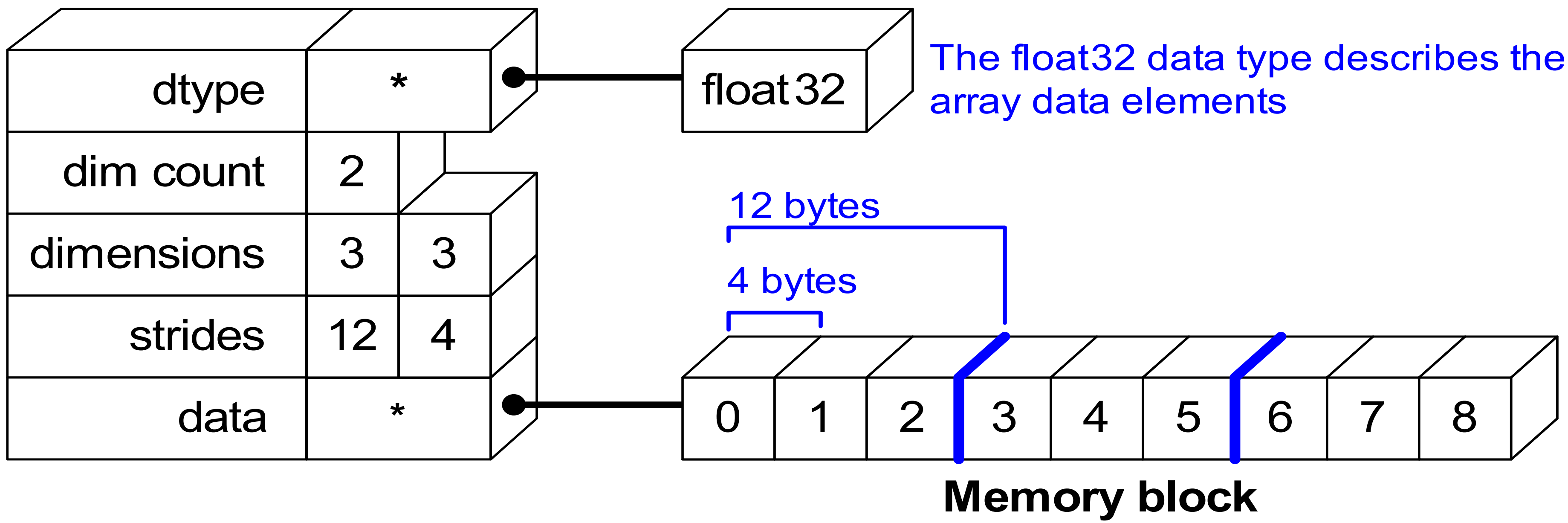
Memory block

Python View :

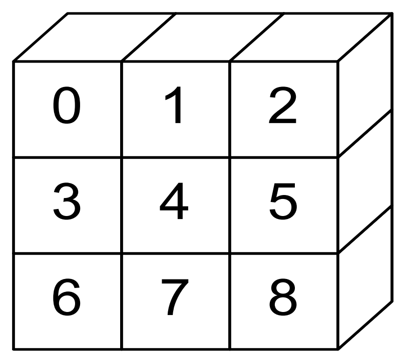


Array Data Structure

NDArray Data Structure



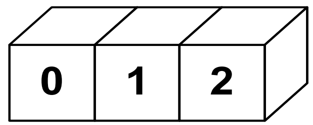
Python View :



Indexing with None

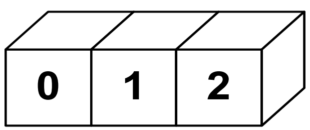
`None` is a special index that inserts a new axis in the array at the specified location.

Each `None` increases the array's dimensionality by 1.

a 

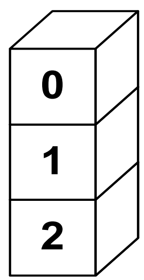
1 X 3

```
>>> y = a[None,:]
>>> shape(y)
(1, 3)
```



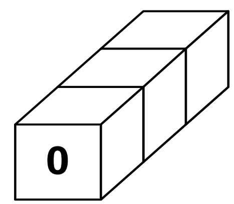
3 X 1

```
>>> y = a[:,None]
>>> shape(y)
(3, 1)
```



3 X 1 X 1

```
>>> y = a[:,None, None]
>>> shape(y)
(3, 1, 1)
```



NumPy dtypes

Basic Type	Available NumPy types	Comments
Boolean	`bool`	Elements are 1 byte in size.
Integer	`int8, int16, int32, int64, int128, int`	int defaults to the size of int in C for the platform.
Unsigned Integer	`uint8, uint16, uint32, uint64, uint128, uint`	uint defaults to the size of unsigned int in C for the platform.
Float	`float32, float64, float, longfloat,`	Float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform-dependent.
Complex	`complex64, complex128, complex`	The real and complex elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	`str, unicode`	
Object	`object`	Represent items in array as Python objects.
Records	`void`	Used for arbitrary data structures.

Summary of (most) array attributes/ methods

BASIC ATTRIBUTES

a.dtype – Numerical type of array elements: float32, uint8, etc.

a.shape – Shape of the array. (m,n,o,...)

a.size – Number of elements in entire array

a.itemsize – Number of bytes used by a single element in the array

a.nbytes – Number of bytes used by entire array (data only)

a.ndim – Number of dimensions in the array

Summary of (most) array attributes/methods

SHAPE OPERATIONS

a.flat – An iterator to step through array as if it is 1-D

a.flatten() – Returns a 1-D copy of a multi-dimensional array

a.ravel() – Same as flatten(), but returns a 'view' if possible

a.resize(new_size) – Changes the size/shape of an array in place

a.swapaxes(axis1, axis2) – Swaps the order of two axes in an array

a.transpose(*axes) – Swaps the order of any number of array axes a.T – Shorthand for a.transpose()

a.squeeze() – Removes any length==1 dimensions from an array

Summary of (most) array attributes/ methods

FILL AND COPY

```
a.copy() - Returns a copy of the array
a.fill(value) - Fills array with a scalar value
```

CONVERSION / COERCION

```
a.tolist() - Converts array into nested lists of values
a.tostring() - Raw copy of array memory into a Python string
a.astype(dtype) - Returns array coerced to the given dtype
a.byteswap(False) - Converts byte order (big <-> little endian)
a.view(type_or_dtype) - Creates new ndarray that sees the
                        the same memory but interprets it as
                        a new data-type (or subclass of ndarray)
```

Summary of (most) array attributes/methods

COMPLEX NUMBERS

```
a.real - Returns the real part of the array
a.imag - Returns the imaginary part of the array
a.conjugate() - Returns the complex conjugate of the array
a.conj()- Returns the complex conjugate of an array.
          (same as conjugate)
```

Summary of (most) array attributes/ methods

SAVING

```
a.dump(file) – Stores a binary array data out to the given file
a.dumps() – Returns the binary pickle of the array as a string
a.tofile(fid, sep="", format="%s") – Formatted ASCII output to file
```

SEARCH / SORT

```
a.nonzero() – Returns indices for all non-zero elements in a
a.sort(axis=-1) – In-place sort of array elements along axis
a.argsort(axis=-1) – Returns indices for element sort order along axis
a.searchsorted(b) – Returns index where elements from b would go in a
```

ELEMENT MATH OPERATIONS

```
a.clip(low, high) – Limits values in array to the specified range
a.round(decimals=0) – Rounds to the specified number of digits
a.cumsum(axis=None) – Cumulative sum of elements along axis
a.cumprod(axis=None) – Cumulative product of elements along axis
```

Summary of (most) array attributes/ methods

REDUCTION METHODS

All the following methods “reduce” the size of the array by 1 dimension by carrying out an operation along the specified axis. If axis is None, the operation is carried out across the entire array.

```
a.sum(axis=None) - Sums up values along axis
a.prod(axis=None) - Finds the product of all values along axis
a.min(axis=None)- Finds the minimum value along axis
a.max(axis=None) - Finds the maximum value along axis
a.argmin(axis=None) - Finds the index of the minimum value along axis
a.argmax(axis=None) - Finds the index of the maximum value along axis
a.ptp(axis=None) - Calculates a.max(axis) - a.min(axis)
a.mean(axis=None) - Finds the mean (average) value along axis
a.std(axis=None) - Finds the standard deviation along axis
a.var(axis=None) - Finds the variance along axis

a.any(axis=None) - True if any value along axis is non-zero   (or)
a.all(axis=None) - True if all values along axis are non-zero (and)
```

Matrix Objects

MATRIX CREATION

```
# Matlab-like creation from string
>>> A = mat('1,2,4;2,5,3;7,8,9')
>>> print A
Matrix([[1, 2, 4],
        [2, 5, 3],
        [7, 8, 9]])

# matrix exponents
>>> print A**4
Matrix([[ 6497,  9580,  9836],
        [ 7138, 10561, 10818],
        [18434, 27220, 27945]])

# matrix multiplication
>>> print A*A.I
Matrix([[ 1.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  0.,  1.]])
```

BMAT

```
# create a matrix from
# sub-matrices
>>> a = array([[1,2],
               [3,4]])
>>> b = array([[10,20],
               [30,40]])

>>> bmat('a,b;b,a')
matrix([[ 1,  2, 10, 20],
        [ 3,  4, 30, 40],
        [10, 20,  1,  2],
        [30, 40,  3,  4]])
```

Vectorizing Functions

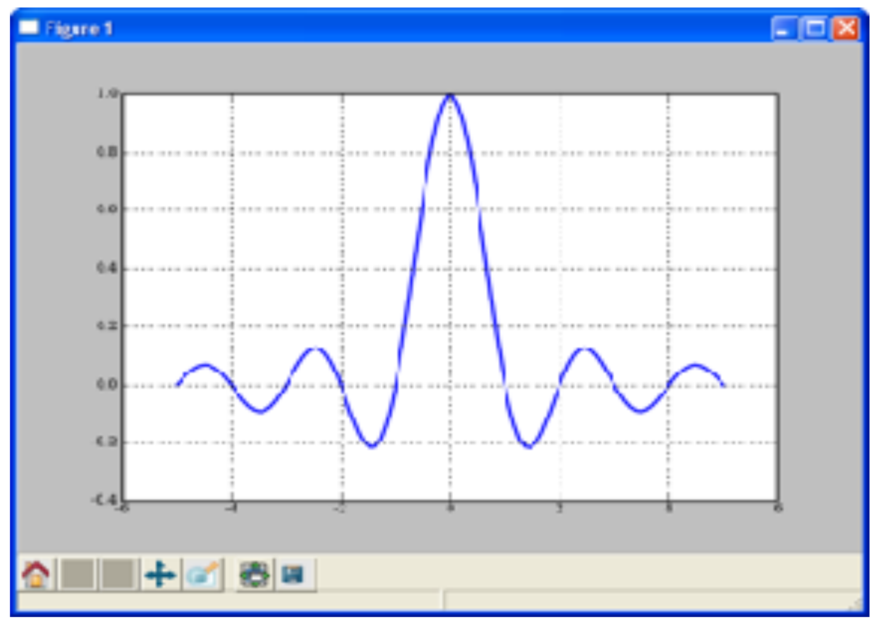
SCALAR SINC FUNCTION

```
# special.sinc already available
# This is just for show.
def sinc(x):
    if x == 0.0:
        return 1.0
    else:
        w = pi*x
        return sin(w) / w
```

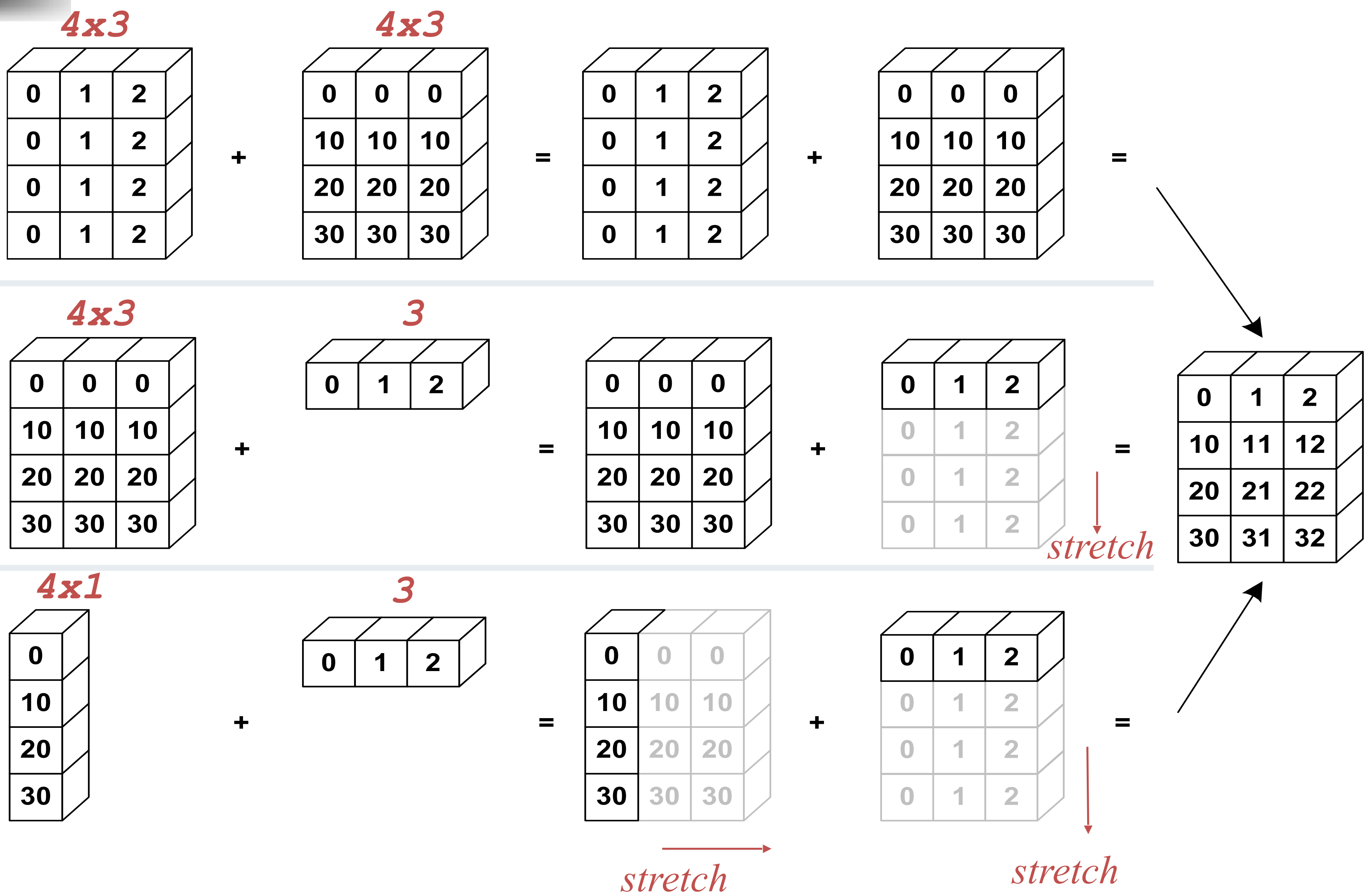
```
# attempt
>>> x = array((1.3, 1.5))
>>> sinc(x)
ValueError: The truth value of an
array with more than one element is
ambiguous. Use a.any() or a.all()
>>> x = r_[-5:5:100j]
>>> y = vsinc(x)
>>> plot(x, y)
```

SOLUTION

```
>>> from numpy import vectorize
>>> vsinc = vectorize(sinc)
>>> vsinc(x)
array([-0.1981, -0.2122])

>>> x2 = linspace(-5, 5, 101)
>>> plot(x2, sinc(x2))
```

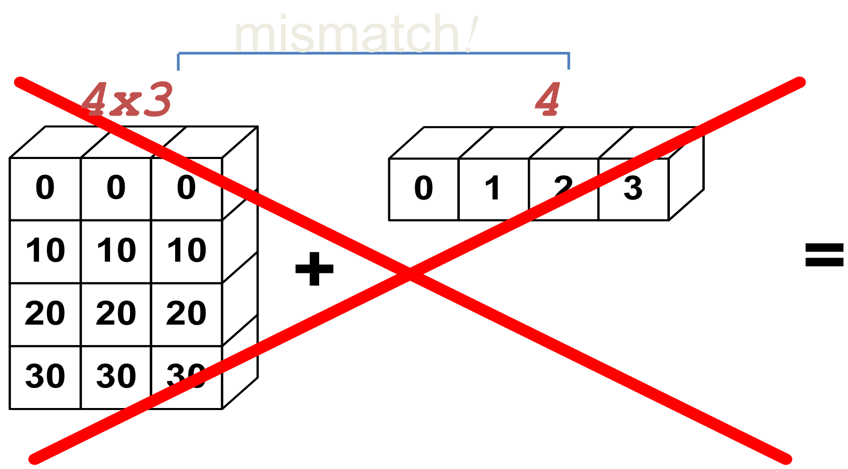


Array Broadcasting



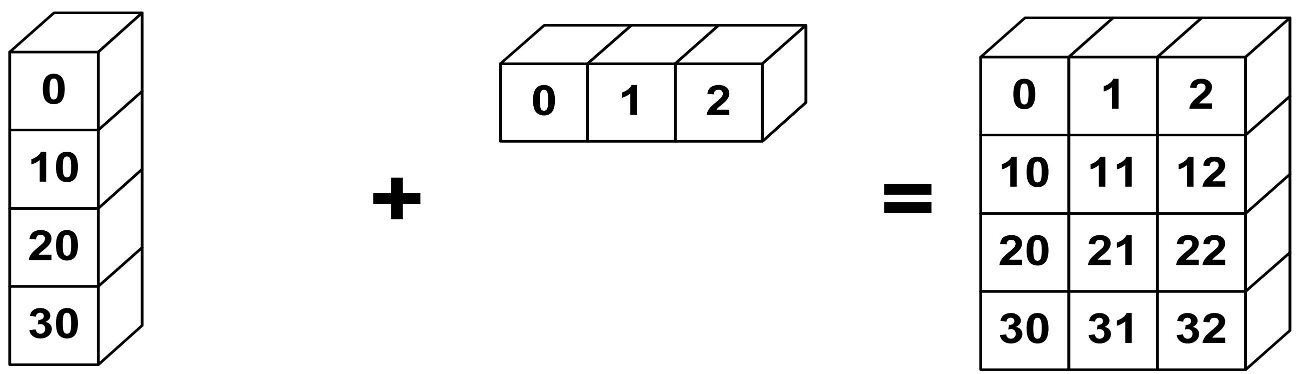
Broadcasting Rules

The trailing axes of both arrays must be either 1 or have the same size for broadcasting to occur. Otherwise, a `"ValueError: frames are not aligned"` exception is thrown.



Broadcasting in Action

```
>>> a = array((0,10,20,30))
>>> b = array((0,1,2))
>>> y = a[:, None] + b
```



Broadcasting Indices

Broadcasting can also be used to slice elements from different “depths” in a 3-D (or any other shape) array. This is a *very* powerful feature of indexing.

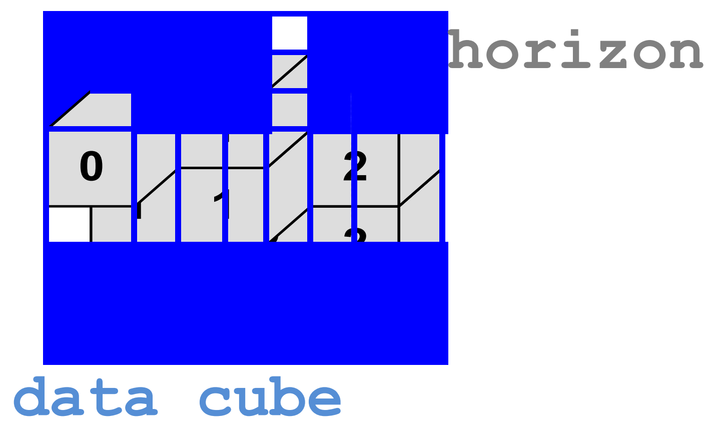
```
>>> xi,yi = ogrid[:3,:3]
>>> zi = array([[0, 1, 2],
                [1, 1, 2],
                [2, 2, 2]])
>>> horizon = data_cube[xi,yi,zi]
```

Indices

xi \ yi	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

zi

Selected Data



Controlling Output Format

```
set_printoptions(precision=None,threshold=None,
                 edgeitems=None, linewidth=None,
                 suppress=None)
```

precision	The number of digits of precision to use for floating point output. The default is 8.
threshold	Array length where NumPy starts truncating the output and prints only the beginning and end of the array. The default is 1000.
edgeitems	Number of array elements to print at beginning and end of array when threshold is exceeded. The default is 3.
linewidth	Characters to print per line of output. The default is 75.
suppress	Indicates whether NumPy suppresses printing small floating point values in scientific notation. The default is **False**.

Controlling Output Formats

PRECISION

```
>>> a = arange(1e6)
>>> a
array([  0.00000000e+00, 1.00000000e+00, 2.00000000e+00, ...,
         9.99997000e+05, 9.99998000e+05, 9.99999000e+05])
>>> set_printoptions(precision=3)
array([  0.000e+00,   1.000e+00,   2.000e+00, ...,
         1.000e+06,   1.000e+06,   1.000e+06])
```

Controlling Output Formats

SUPPRESSING SMALL NUMBERS

```
>>> set_printoptions(precision=8)
>>> a = array((1, 2, 3, 1e-15))
>>> a
array([  1.00000000e+00,   2.00000000e+00,   3.00000000e+00,
         1.00000000e-15])
>>> set_printoptions(suppress=True)
>>> a
array([ 1.,  2.,  3.,  0.])
```

Controlling Error Handling

```
seterr(all=None, divide=None, over=None,
       under=None, invalid=None)
```

Set the error handling flags in ufunc operations on a per thread basis. Each of the keyword arguments can be set to 'ignore', 'warn', 'print', 'log', 'raise', or 'call'.

all	All error types to the specified value
divide	'Divide-by-zero' errors
over	'Overflow' errors
under	'Underflow' errors
invalid	'Invalid' floating point errors

Controlling Error Handling

```
>>> a = array((1,2,3))
>>> a/0.
Warning: divide by zero encountered in divide
array([ 1.#INF0000e+000,  1.#INF0000e+000,  1.#INF0000e+000])

# Ignore division-by-zero.  Also, save old values so that
# we can restore them.
>>> old_err = seterr(divide='ignore')
>>> a/0.
array([ 1.#INF0000e+000,  1.#INF0000e+000,  1.#INF0000e+000])
```

Controlling Error Handling

```
# Restore original error handling mode.
>>> old_err
{'divide': 'print', 'invalid': 'print', 'over':
  'print', 'under': 'ignore'}
>>> seterr(**old_err)
>>> a/0.
Warning: divide by zero encountered in divide
array([ 1.#INF0000e+000,  1.#INF0000e+000,
  1.#INF0000e+000])
```

“Structured” Arrays

Elements of an array can be any fixed-size data structure!

```
name char[10]
age  int
weight double
```

Brad	Jane	John	Fred
33	25	47	54
135.0	105.0	225.0	140.0
Henry	George	Brian	Amy
29	61	32	27
154.0	202.0	137.0	187.0
Ron	Susan	Jennifer	Jill
19	33	18	54
188.0	135.0	88.0	145.0

EXAMPLE

```
>>> from numpy import dtype, empty
# structured data format
>>> fmt = dtype([('name', 'S10'),
                 ('age', int),
                 ('weight', float)
                ])
>>> a = empty((3,4), dtype=fmt)
>>> a.itemsize
22
>>> a['name'] = [['Brad', … ,'Jill']]
>>> a['age'] = [[33, … , 54]]
>>> a['weight'] = [[135, … , 145]]
>>> print a
[[('Brad', 33, 135.0)
  …
  ('Jill', 54, 145.0)]]
```

Structured Arrays

```
import numpy as np
format = np.dtype([('symbol', 'S4'),
                   ('date', 'O'),
                   ('open', 'f8'),
                   ('close', 'f8'),
                   ('low', 'f8'),
                   ('high', 'f8'),
                   ('adj_close',
'f8'),
                ('volume', 'i4')])
r = np.array(<data>, dtype=format)
r = r.view(np.recarray)
query1 = r[r.volume > 1e8]
print query1.symbol
print query1.date
query2 = r[r.close > 1.05*r.open]
print query2.symbol
print query2.date
mask = (r.close - r.open) >
0.10*r.open
query3 = r[mask]
```

Symbol	Date	Open	Close	...	Volume
GOOG	7/19/07	553.46	548.59	...	11127200
QQQQ	7/19/07	50.41	50.32	...	116563800
GE	7/19/07	40.58	40.71	...	29766400
AAPL	7/19/07	140.3	140.0	...	26174700
YHOO	7/19/07	26.32	26.03	...	29537900
MSFT	7/19/07	31.05	31.51	...	121159300
...	...	...	...	...	...

“Structured” Arrays

```
# "Data structure" (dtype) that describes the fields and
# type of the items in each array element.
>>> particle_dtype = dtype([('mass','f4'), ('velocity', 'f4')])
# This must be a list of tuples. NumPy doesn't like
# a list of arrays or a tuple of tuples.
>>> particles = array([(1,1), (1,2), (2,1), (1,3)],
                      dtype=particle_dtype)
>>> particles
[(1.0, 1.0) (1.0, 2.0) (2.0, 1.0) (1.0, 3.0)]
# Retrieve the mass for all particles through indexing.
>>> particles['mass']
[ 1.  1.  2.  1.]
```

“Structured” Arrays

```
# Retrieve particle 0 through indexing.
>>> particles[0]
(1, 1)
# Sort particles in place, with velocity as the primary field and
# mass as the secondary field.
>>> particles.sort(order=('velocity','mass'))
>>> particles
[(1.0, 1.0) (2.0, 1.0) (1.0, 2.0) (1.0, 3.0)]

# See demo/mutlitype_array/particle.py.
```

Overview

- Available at www.scipy.org
- Open Source BSD Style License
- 34 svn “committers” to the project

CURRENT PACKAGES

- Special Functions (scipy.special)
- Signal Processing (scipy.signal)
- Image Processing (scipy.ndimage)
- Fourier Transforms (scipy.fftpack)
- Optimization (scipy.optimize)
- Numerical Integration (scipy.integrate)
- Linear Algebra (scipy.linalg)
- Input/Output (scipy.io)
- Statistics (scipy.stats)
- Fast Execution (scipy.weave)
- Clustering Algorithms (scipy.cluster)
- Sparse Matrices (scipy.sparse)
- Interpolation (scipy.interpolate)
- More (e.g. scipy.odr, scipy.maxentropy)

Polynomials

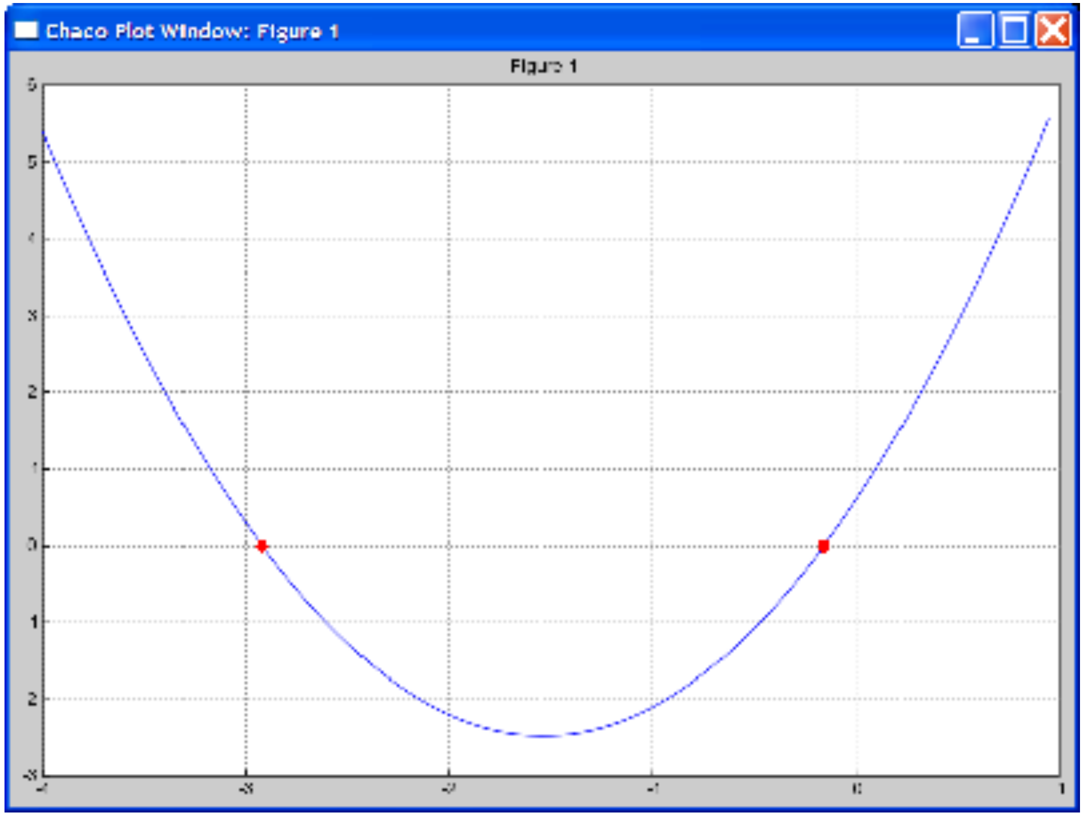
- p = poly1d(<coefficient array>)
- p.roots (p.r) are the roots
- p.coefficients (p.c) are the coefficients
- p.order is the order
- p[n] is the coefficient of x^n
- p(val) evaulates the polynomial at val
- p.integ() integrates the polynomial
- p.deriv() differentiates the polynomial
- Basic numeric operations (+,-,/,*) work
- Acts like p.c when used as an array
- Fancy printing

```
>>> p = poly1d([1,-2,4])
>>> print p
 2
x - 2 x + 4

>>> g = p**3 + p*(3-2*p)
>>> print g
 6     5      4      3      2
x - 6 x + 25 x - 51 x + 81 x - 58 x +
44

>>> print g.deriv(m=2)
    4       3       2
30 x - 120 x + 300 x - 306 x + 162

>>> print p.integ(m=2,k=[2,1])
         4          3     2
0.08333 x - 0.3333 x + 2 x + 2 x + 1

>>> print p.roots
[ 1.+1.7321j  1.-1.7321j]

>>> print p.coeffs
[ 1 -2  4]
```

Polynomials

FINDING THE ROOTS OF A POLYNOMIAL

```
>>> p = poly1d([1.3, 4.0, 0.6])
>>> print p
     2
1.3 x + 4 x + 0.6
>>> x = linspace(-4, 1.0, 101)
>>> y = p(x)
>>> plot(x,y,'-')
>>> hold(True)
>>> r = p.roots
>>> s = p(r)
>>> r
array([-0.15812627, -2.9187968 ])
>>> plot(r.real,s.real,'ro')
```



Special Functions

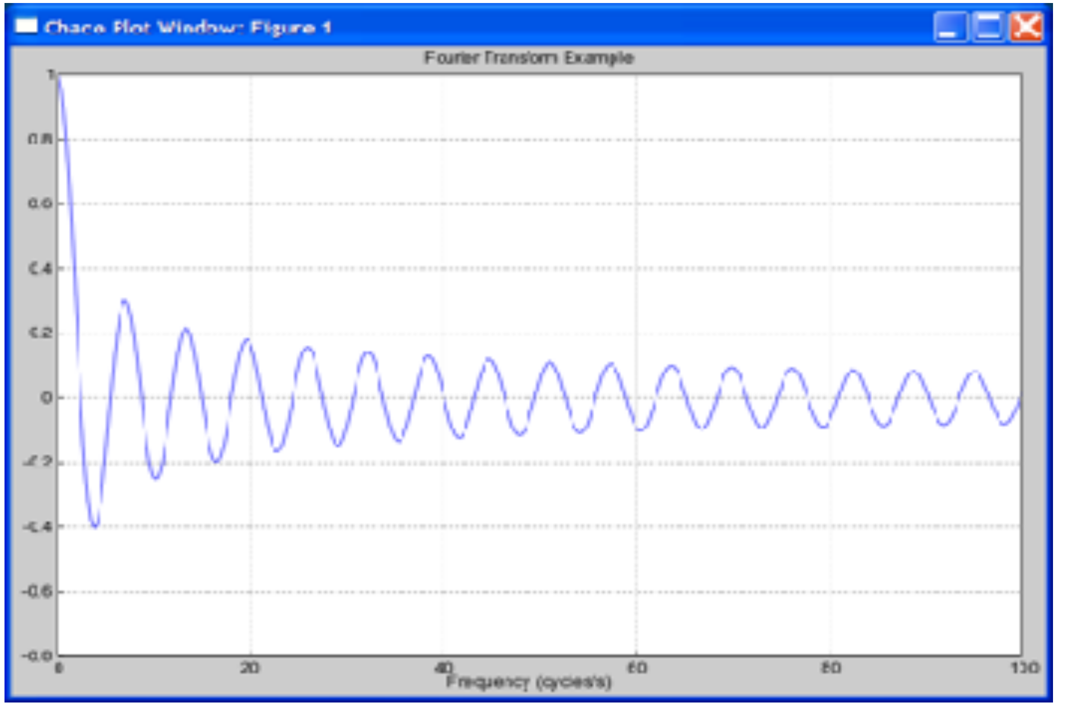
scipy.special

Includes over 200 functions:

Airy, Elliptic, Bessel, Gamma, HyperGeometric, Struve, Error, Orthogonal Polynomials, Parabolic Cylinder, Mathieu, Spheroidal Wave, Kelvin

FIRST ORDER BESSEL EXAMPLE

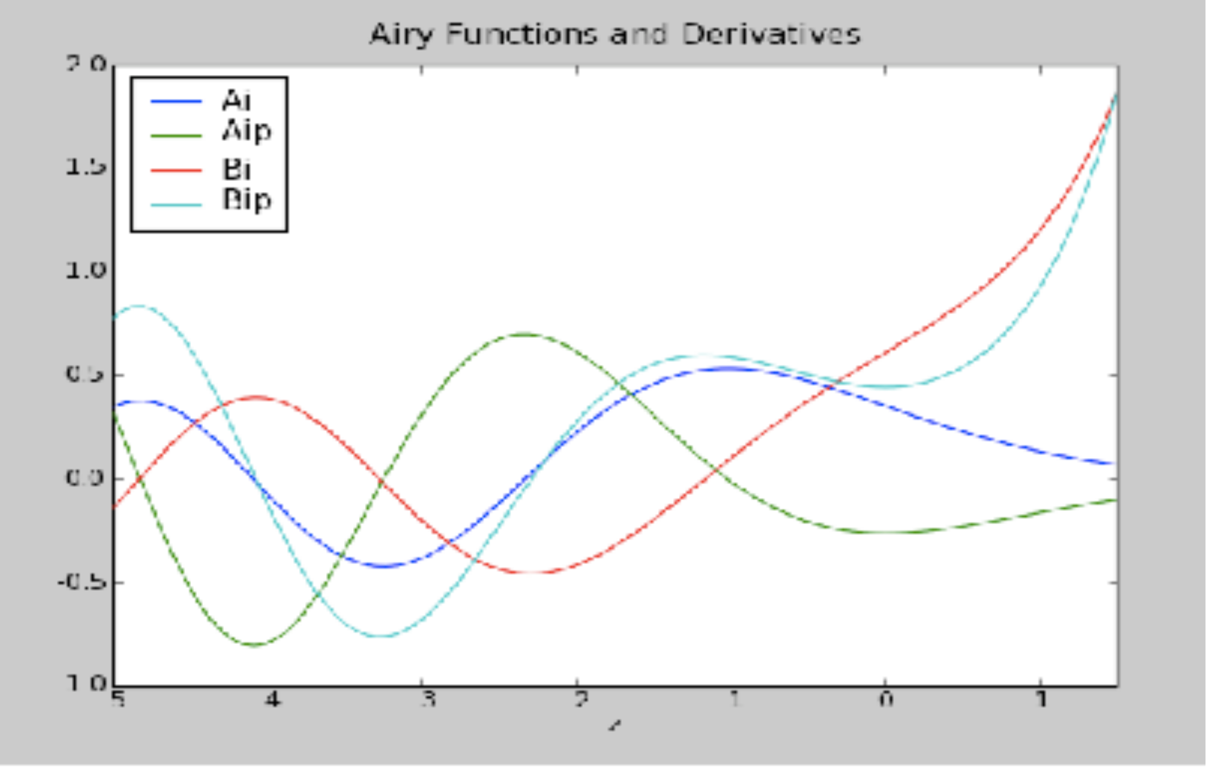
```
>>> from scipy import special
>>> x = linspace(0, 100, 1001)
>>> j0x = special.j0(x)
>>> plot(x,j0x)
```



Special Functions

AIRY FUNCTIONS

```
>>> z = linspace(-5, 1.5, 100)
>>> Ai, Aip, Bi, Bip = special.airy(z)
>>> plot(z, array(vals).T)
```



Interpolation

scipy.interpolate --- General purpose Interpolation

- **1-d Interpolating Class**
  - Constructs callable function from data points and desired spline interpolation order.
  - Function takes vector of inputs and returns interpolated value using the spline.
- **1-d and 2-d spline interpolation (FITPACK)**
  - Smoothing splines up to order 5
  - Parametric splines

1D Spline Interpolation

```
>>> from scipy.interpolate import interp1d
```

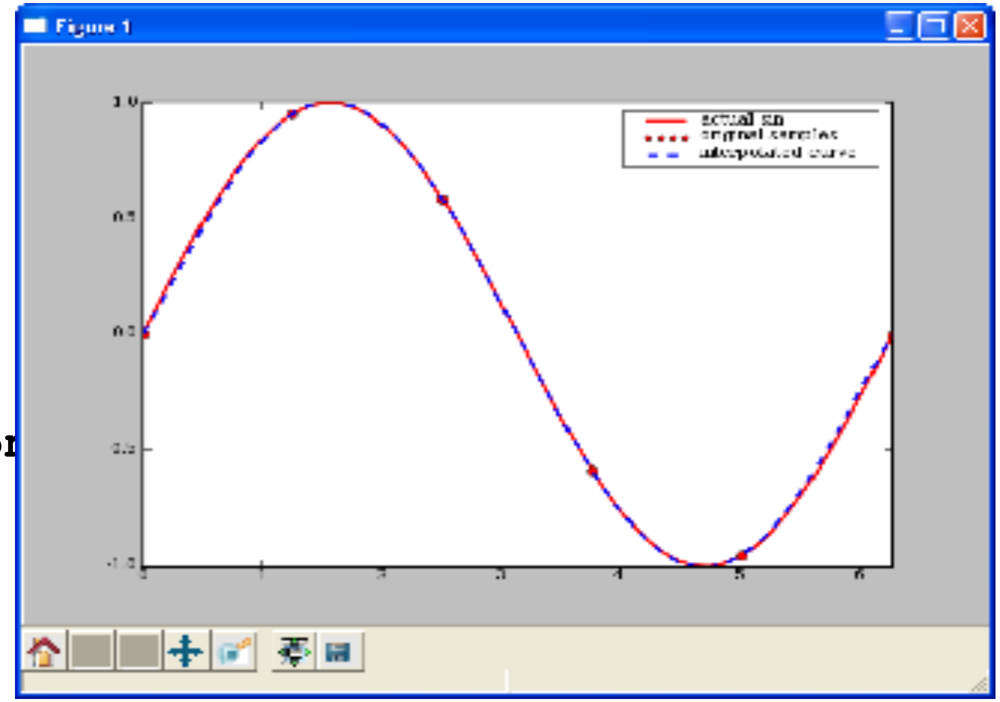
interp1d(x, y, kind='linear', axis=-1, copy=True, bounds_error=True, fill_value=numpy.nan)

Returns a function that uses interpolation to find the value of new points.

- **x** – 1d array of increasing real values which cannot contain duplicates
- **y** – Nd array of real values whose length along the interpolation axis must be len(x)
- **kind** – kind of interpolation (*e.g.* 'linear', 'nearest', 'quadratic', 'cubic'). Can also be an integer n>1 which returns interpolating spline (with minimum sum-of-squares discontinuity in nth derivative).
- **axis** – axis of y along which to interpolate
- **copy** – make internal copies of x and y
- **bounds_error** – raise error for out-of-bounds
- **fill_value** – if bounds_error is False, then use this value to fill in out-of-bounds.

1D Spline Interpolation

```
# demo/interpolate/spline.py
from scipy.interpolate import interp1d
from pylab import plot, axis, legend
from numpy import linspace
# sample values
x = linspace(0,2*pi,6)
y = sin(x)
# Create a spline class for interpolatio
# kind=5 sets to 5th degree spline.
# kind=‘nearest’ -> zeroth older hold.
# kind=‘linear’ -> linear interpolation
# kind=n -> use an nth order spline
```



1D Spline Interpolation

```
spline_fit = interp1d(x,y,kind=5)
xx = linspace(0,2*pi, 50)
yy = spline_fit(xx)
# display the results.
plot(xx, sin(xx), 'r-', x, y, 'ro', xx, yy, 'b--',
  linewidth=2)
axis('tight')
legend(['actual sin', 'original samples',
  'interpolated curve'])
```

2D Spline Interpolation

```
>>> from scipy.interpolate import interp2d

interp2d(x, y, z, kind='linear')
```

Returns a function, f, that uses interpolation to find the value of new points: z_new = f(x_new, y_new)

- x – 1d or 2d array
- y – 1d or 2d array
- z – 1d or 2d array representing function evaluated at x and y
- kind – kind of interpolation: 'linear', 'quadratic', or 'cubic'

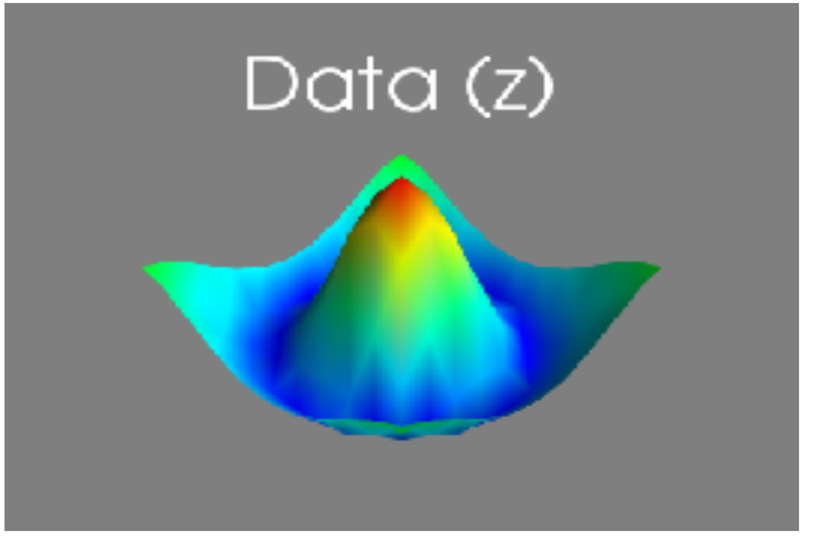
The shape of x, y, and z must be the same.

Resulting function is evaluated at cross product of new inputs.

2D Spline Interpolation

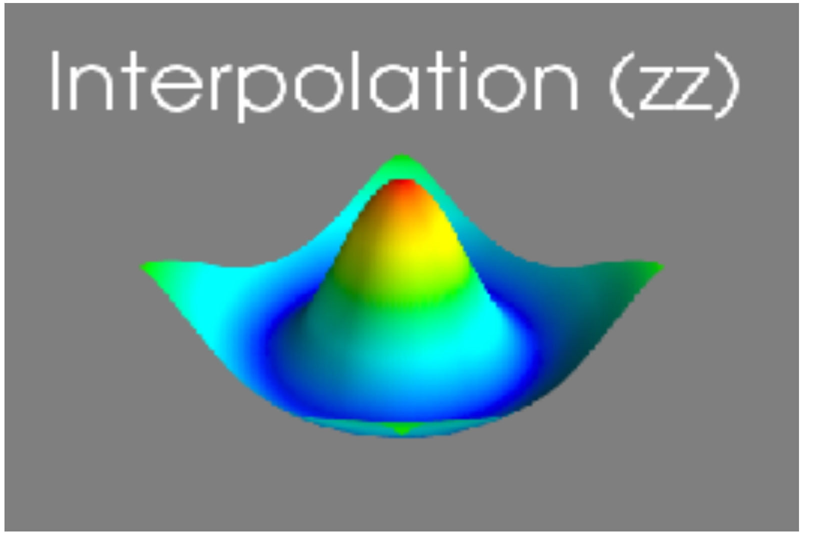
EXAMPLE

```
>>> from scipy.interpolate import \
...       interp2d
>>> from numpy import hypot, mgrid, \
...                      linspace
>>> from scipy.special import j0
>>> x,y = mgrid[-5:6,-5:6]
>>> z = j0(hypot(x,y))
>>> newfunc = interp2d(x, y, z,
...                       kind='cubic')
>>> xx = linspace(-5,5,100)
>>> yy = xx
# xx and yy are 1-d
# result is evaluated on the
# cross product
```



2D Spline Interpolation

```
>>> zz = newfunc(xx,yy)
>>> from enthought.mayavi import mlab
>>> mlab.surf(x,y,z)
>>> x2, y2 = mgrid[-5:5:100j,
...                -5:5:100j]
>>> mlab.surf(x2,y2,zz)
```



Statistics

scipy.stats --- CONTINUOUS DISTRIBUTIONS

over 80 continuous distributions!

METHODS

`pdf` `entropy`

`cdf` `nnlf`

`rvs` `moment`

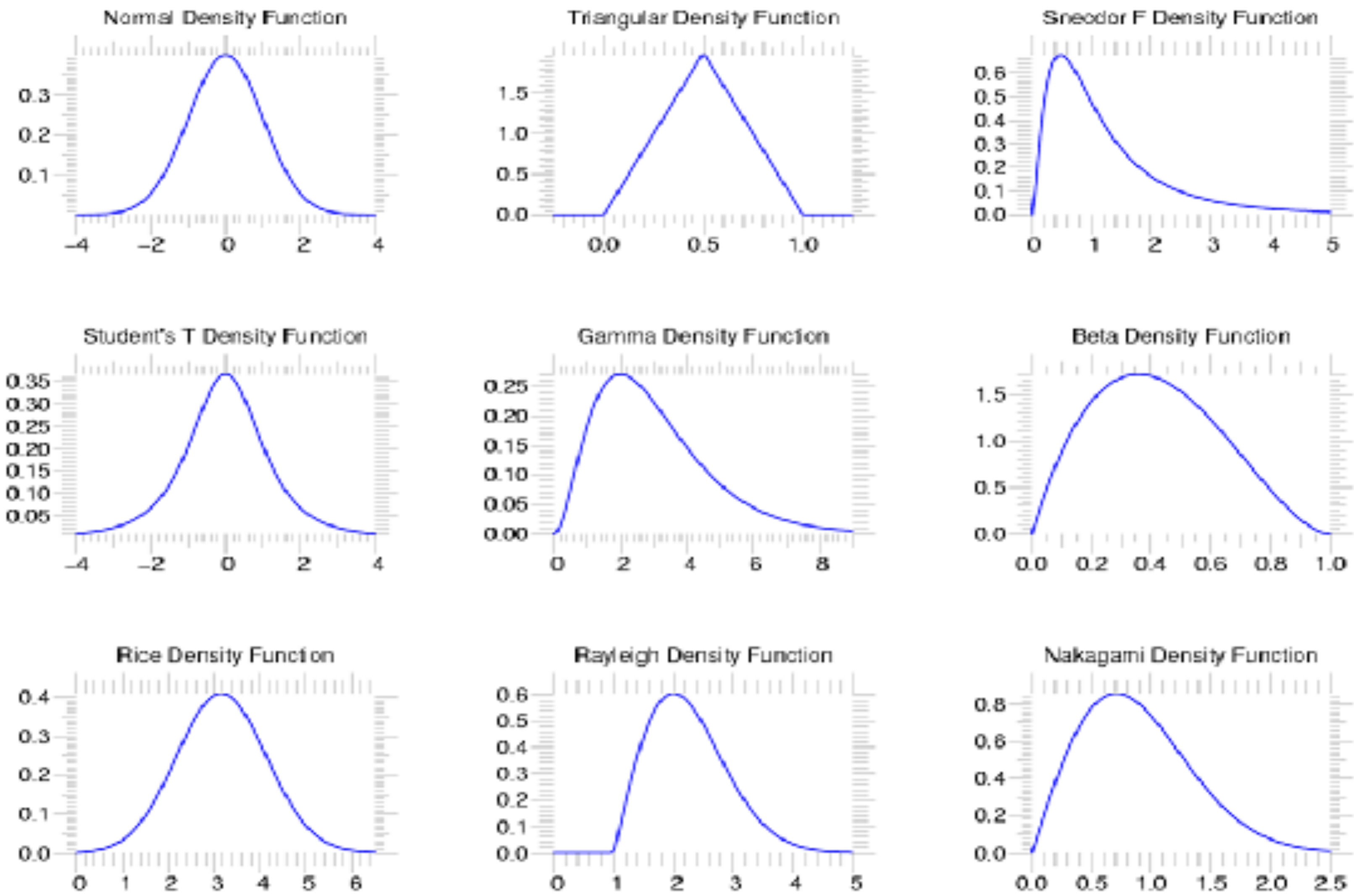
`ppf` `freeze`

`stats`

`fit`

`sf`

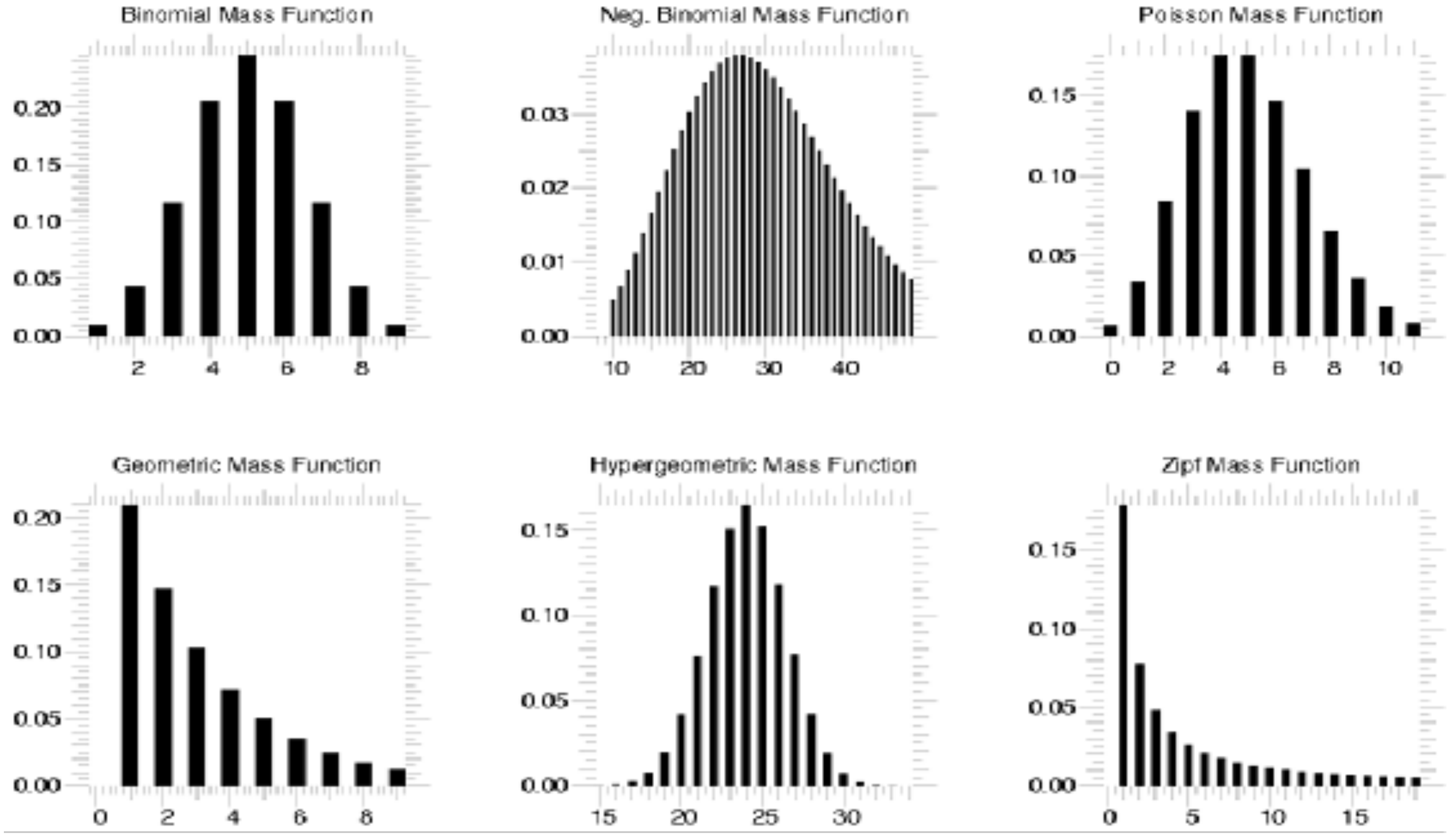
`isf`



Statistics

scipy.stats --- Discrete Distributions

10 standard discrete distributions (plus any finite RV)



METHODS

`pmf` `moment`

`cdf` `entropy`

`rvs` `freeze`

`ppf`

`stats`

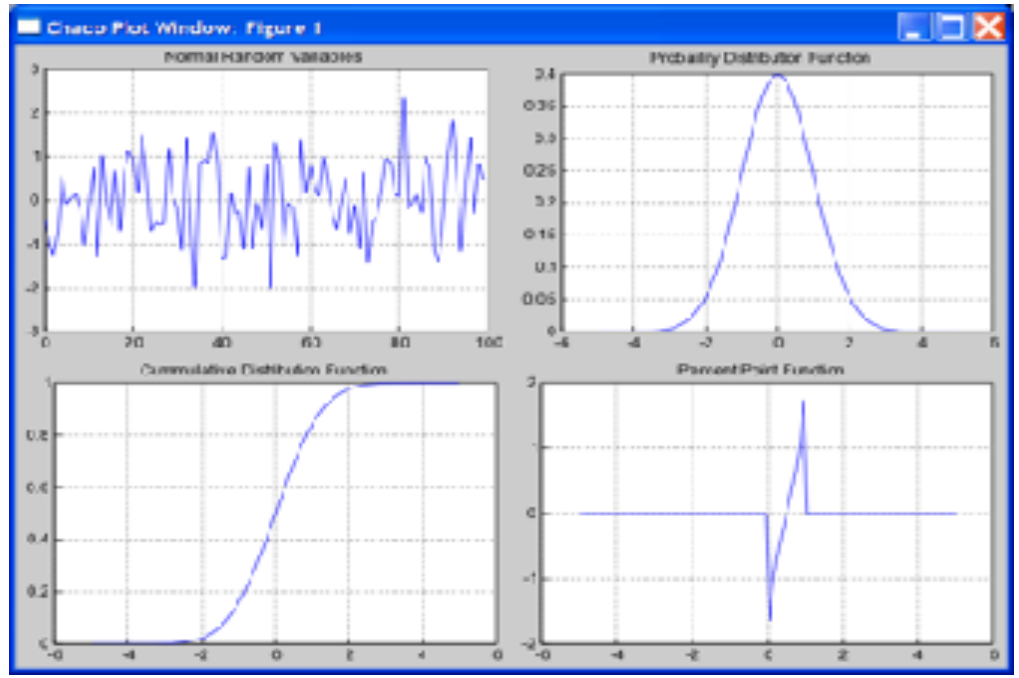
`sf`

`isf`

Using stats objects

DISTRIBUTIONS

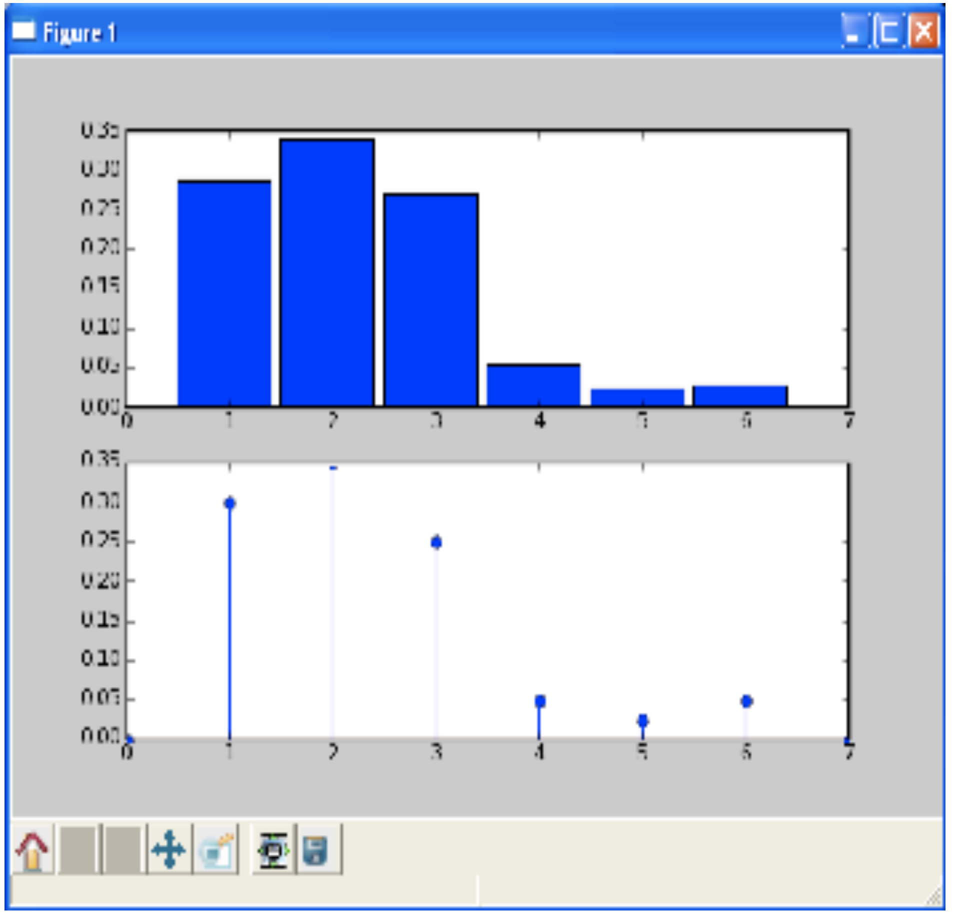
```
>>> from scipy.stats import norm
# Sample normal dist. 100 times.
>>> samp = norm.rvs(size=100)

>>> x = linspace(-5, 5, 100)
# Calculate probability dist.
>>> pdf = norm.pdf(x)
# Calculate cummulative dist.
>>> cdf = norm.cdf(x)
# Calculate Percent Point Function
>>> ppf = norm.ppf(x)

# Estimate parameters from data
>>> mu, sigma = norm.fit(samp)
>>> print "%4.2f, %4.2f" % (mu, sigma)
-0.14, 1.01
```



Using stats objects

CREATING NEW DISCRETE DISTRIBUTIONS

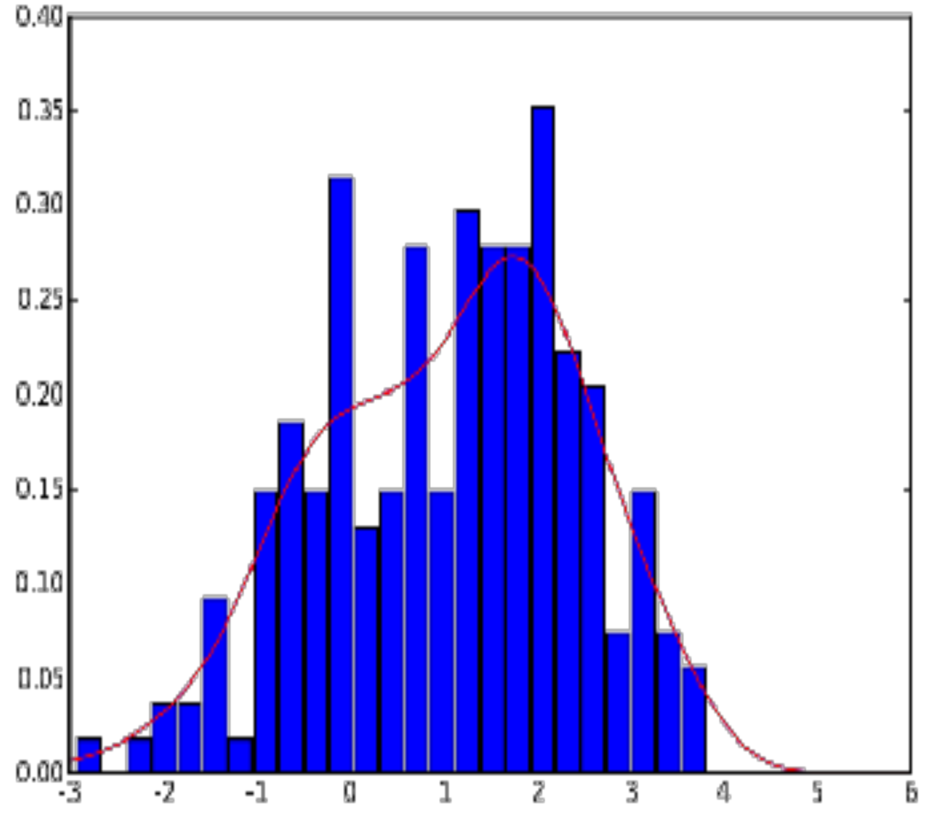
```
# Create loaded dice.
>>> from scipy.stats import rv_discrete
>>> xk = [1,2,3,4,5,6]
>>> pk = [0.3,0.35,0.25,0.05,
         0.025,0.025]
>>> new = rv_discrete(name='loaded',
                      values=(xk,pk))
# Calculate histogram
>>> samples = new.rvs(size=1000)
>>> bins=linspace(0.5,5.5,6)
>>> subplot(211)
>>> hist(samples,bins=bins,normed=True)
# Calculate pmf
>>> x = range(0,8)
>>> subplot(212)
>>> stem(x,new.pmf(x))
```



Statistics

CONTINOUS DISTRIBUTION ESTIMATION USING GAUSSIAN KERNELS

```
# Sample two normal distributions
# and create a bi-modal distribution
>>> rv1 = stats.norm()
>>> rv2 = stats.norm(2.0,0.8)
>>> samples = hstack([rv1.rvs(size=100),
                      rv2.rvs(size=100)])
# Use a gaussian kernel density to
# estimate the pdf for the samples.
>>> from scipy.stats.kde import
   gaussian_kde
>>> approximate_pdf =
   gaussian_kde(samples)
>>> x = linspace(-3,6,200)
# Compare the histogram of the samples to
# the pdf approximation.
>>> hist(samples, bins=25, normed=True)
>>> plot(x, approximate_pdf(x),'r')
```



Linear Algebra

scipy.linalg --- FAST LINEAR ALGEBRA

- Uses ATLAS if available --- very fast
- Low-level access to BLAS and LAPACK routines in modules `linalg.fblas`, and `linalg.flapack` (FORTRAN order)
- High level matrix routines
  - Linear Algebra Basics: `inv`, `solve`, `det`, `norm`, `lstsq`, `pinv`
  - Decompositions: `eig`, `lu`, `svd`, `orth`, `cholesky`, `qr`, `schur`
  - Matrix Functions: `expm`, `logm`, `sqrtm`, `cosm`, `coshm`, `funm` (general matrix functions)

Linear Algebra

LU FACTORIZATION

```
>>> from scipy import linalg
>>> a = array([[1,3,5],
...            [2,5,1],
...            [2,3,6]])
# time consuming factorization
>>> lu, piv = linalg.lu_factor(a)

# fast solve for 1 or more
# right hand sides.
>>> b = array([10,8,3])
>>> linalg.lu_solve((lu, piv), b)
array([-7.82608696,  4.56521739,
        0.82608696])
```

EIGEN VALUES AND VECTORS

```
>>> from scipy import linalg
>>> a = array([[1,3,5],
...            [2,5,1],
...            [2,3,6]])
# compute eigen values/vectors
>>> vals, vecs = linalg.eig(a)
# print eigen values
>>> vals
array([ 9.39895873+0.j,
       -0.73379338+0.j,
        3.33483465+0.j])
# eigen vectors are in columns
# print first eigen vector
>>> vecs[:,0]
array([-0.57028326,
       -0.41979215,
       -0.70608183])
# norm of vector should be 1.0
>>> linalg.norm(vecs[:,0])
1.0
```

Matrix Objects

STRING CONSTRUCTION

```
>>> from numpy import mat
>>> a = mat('[1,3,5;2,5,1;2,3,6]')
>>> a
matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 6]])
```

TRANSPOSE ATTRIBUTE

```
>>> a.T
matrix([[1, 2, 2],
        [3, 5, 3],
        [5, 1, 6]])
```

INVERTED ATTRIBUTE

```
>>> a.I
matrix([[-1.1739, 0.1304,  0.956],
        [ 0.4347, 0.1739, -0.391],
        [ 0.1739, -0.130,  0.0434]
       ])
# note: reformatted to fit slide
```

DIAGONAL

```
>>> a.diagonal()
matrix([[1, 5, 6]])
>>> a.diagonal(-1)
matrix([[3, 1]])
```

SOLVE

```
>>> b = mat('10;8;3')
>>> a.I*b
matrix([[-7.82608696],
        [ 4.56521739],
        [ 0.82608696]])

>>> from scipy import linalg
>>> linalg.solve(a,b)
matrix([[-7.82608696],
        [ 4.56521739],
        [ 0.82608696]])
```

Optimization

scipy.optimize --- unconstrained minimization and root finding

- **Unconstrained Optimization**

  `fmin` (Nelder-Mead simplex), `fmin_powell` (Powell's method), `fmin_bfgs` (BFGS quasi-Newton method), `fmin_ncg` (Newton conjugate gradient), `leastsq` (Levenberg-Marquardt), `anneal` (simulated annealing global minimizer), `brute` (brute force global minimizer), **brent** (excellent 1-D minimizer), **golden**, **bracket**

- **Constrained Optimization**

  `fmin_l_bfgs_b`, `fmin_tnc` (truncated newton code), `fmin_cobyla` (constrained optimization by linear approximation), `fminbound` (interval constrained 1-d minimizer)

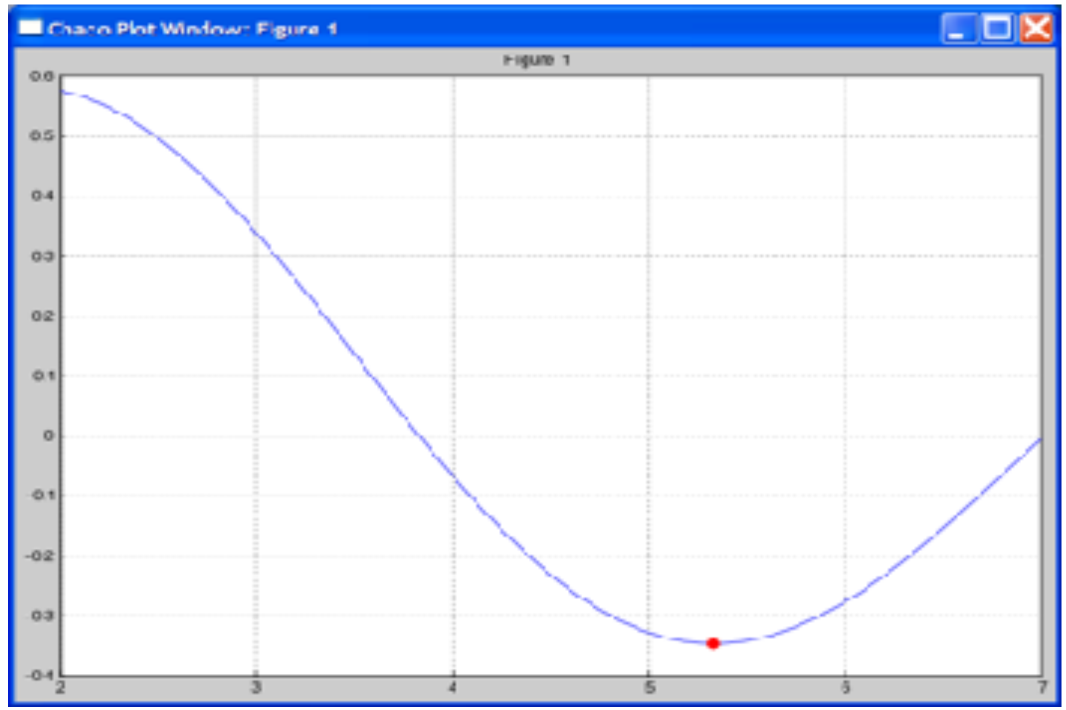
- **Root finding**

  `fsolve` (using MINPACK), `brentq`, `brenth`, `ridder`, `newton`, `bisect`, `fixed_point` (fixed point equation solver)

Optimization: 1D Minimization

EXAMPLE: MINIMIZE BESSEL FUNCTION

```
# minimize 1st order bessel
# function between 4 and 7
>>> from scipy.special import j1
>>> from scipy.optimize import \
...        fminbound

>>> x = r_[2:7.1:.1]
>>> j1x = j1(x)
>>> plot(x,j1x,'-')
>>> hold(True)
>>> x_min = fminbound(j1,4,7)
>>> j1_min = j1(x_min)
>>> plot([x_min],[j1_min],'ro')
```



Optimization: Solving Nonlinear Equations

FSOLVE

SYSTEM OF EQUATIONS

$$3x_0 - \cos(x_1 x_2) + a = 0$$
$$x_0^2 - 81(x_1 + 0.1)^2 + \sin(x_2) + b = 0$$
$$e^{-x_0 x_1} + 20x_2 + c = 0$$

```
>>> def equations(x,a,b,c):
...     x0, x1, x2 = x
...     eqs = \
...     [3 * x0 - cos(x1*x2) + a,
...     x0**2 - 81*(x1+0.1)**2 + sin(x2) + b,
...      exp(-x0*x1) + 20*x2 + c]
...     return eqs
# coefficients
>>> a = -0.5
>>> b = 1.06
>>> c = (10 * pi - 3.0) / 3
# Optimization start location.
>>> initial_guess = [0.1, 0.1, -0.1]
# Solve the system of non-linear equations.
>>> root = optimize.fsolve(equations, initial_guess, args=(a, b, c))
>>> print root
root: [  5.00e-01   1.38e-13  -5.24e-01]
>>> print nonlin(root, a, b, c)
solution at root: [0.0, -2.2311...e-012, 7.46069...e-014]
```

Optimization: Using Derivatives

FMIN: WITHOUT DERIVATIVE

```
>>> from scipy.optimize import rosen, fmin
>>> initial_guess = [1.3, 0.7, 0.8, 1.9, 1.2]
>>> optimal = optimize.fmin(rosen, x0, xtol=1e-7)
Optimization terminated successfully.
    Current function value: 0.000000
    Iterations: 234
    Function evaluations: 387
>>> print optimal
optimal: [ 1.  1.  1.  1.  1.]
```

Rosenbrock function

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} 100\left(x_i - x_{i-1}^2\right)^2 + (1 - x_{i-1})^2 .$$

Optimization: Using Derivatives

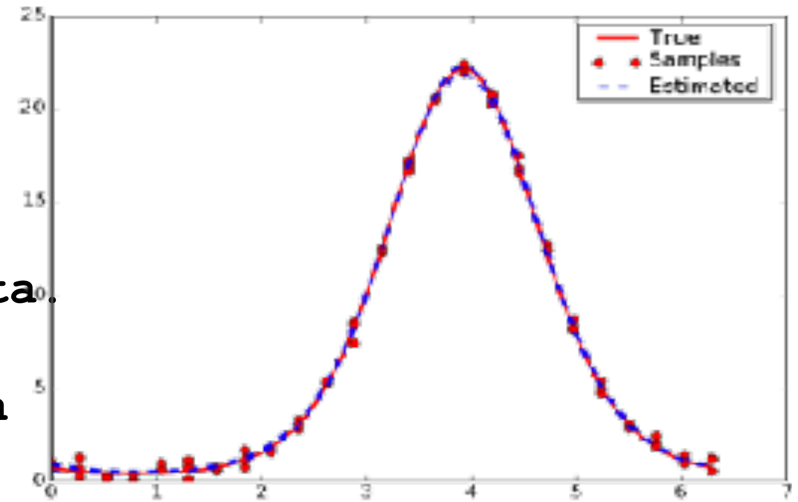
FMIN_BFGS: USING DERIVATIVE

```
>>> from scipy.optimize import rosen_der, fmin_bfgs
>>> optimal = fmin_bfgs(rosen, initial_guess, fprime=rosen_der)
Optimization terminated successfully.
    Current function value: 0.000000
    Iterations: 51
    Function evaluations: 63
    Gradient evaluations: 63
>>> print optimal
optimal: [ 1.  1.  1.  1.  1.]
```

Optimization: Data Fitting

NONLINEAR LEAST SQUARES

```
>>> from scipy.optimize import leastsq
# Define the function to fit.
>>> def function(x, a , b, f, phi):
...     result = a * exp(-b * sin(f * x + phi))
...     return result
# And an error function that compares it to data.
>>> def error_function(params, x_data, y_data):
...     result = func(x_data, *params) - y_data
...     return result
# Create a noisy data set.
>>> actual_params = [3, 2, 1, pi/4]
>>> x = linspace(0,2*pi,25)
>>> exact = function(x, *actual_params)
>>> noisy = exact + 0.3 * randn(len(x))
```



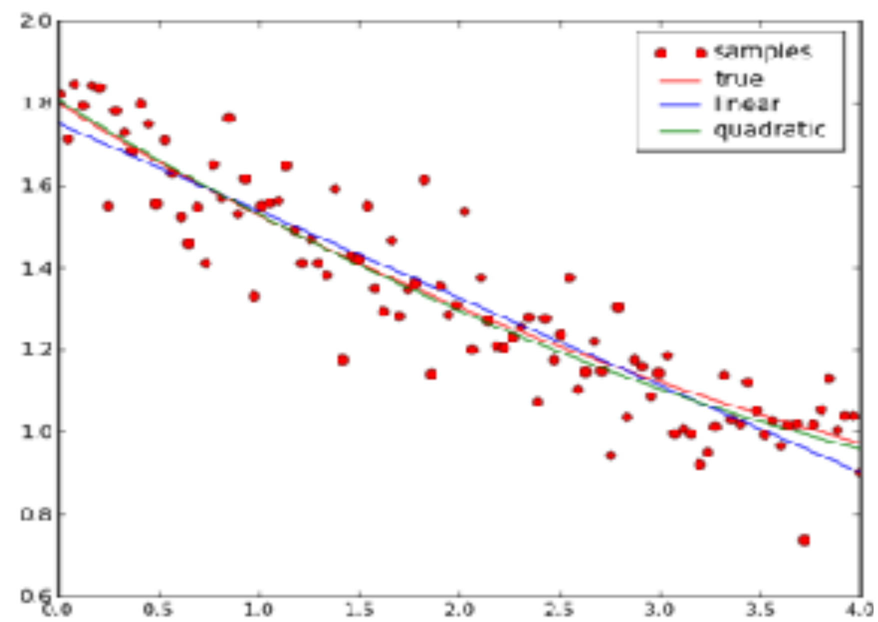
Optimization: Data Fitting

```
# Use least squares to to estimate the function parameters from the
   noisy data.
>>> initial_guess = [1,1,1,1]
>>> estimated_params, d = leastsq(error_function, initial_guess,
   args=(x, noisy))
>>> estimated_params
array([3.1705, 1.9501, 1.0206, 0.7034])
```

Fitting Polynomials (NumPy)

POLYFIT(X, Y, DEGREE)

```
>>> from numpy import polyfit, poly1d
>>> from scipy.stats import norm
# Create clean data.
>>> x = linspace(0, 4.0, 100)
>>> y = 1.5 * exp(-0.2 * x) + 0.3
# Add a bit of noise.
>>> noise = 0.1 * norm.rvs(size=100)
>>> noisy_y = y + noise

# Fit noisy data with a linear model.
>>> linear_coef = polyfit(x, noisy_y, 1)
>>> linear_poly = poly1d(linear_coef)
>>> linear_y = linear_poly(x),

# Fit noisy data with a quadratic model.
>>> quad_coef = polyfit(x, noisy_y, 2)
>>> quad_poly = poly1d(quad_coef)
>>> quad_y = quad_poly(x))
```



Signal Processing

scipy.signal --- Signal and Image Processing

What's Available?

Filtering

General 2-D Convolution (more boundary conditions)

N-D convolution

B-spline filtering

N-D Order filter, N-D median filter, faster 2d version,

IIR and FIR filtering and filter design

LTI systems

System simulation

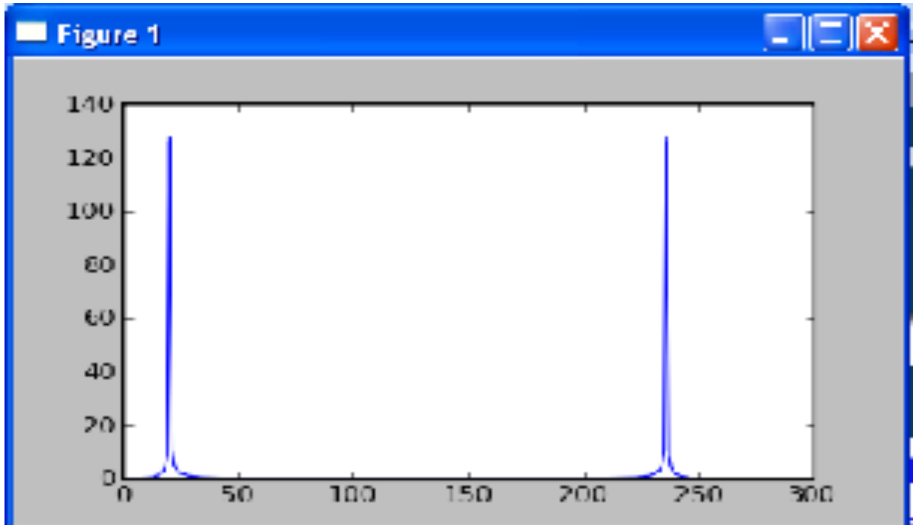
Impulse and step responses

Partial fraction expansion

FFT Functions

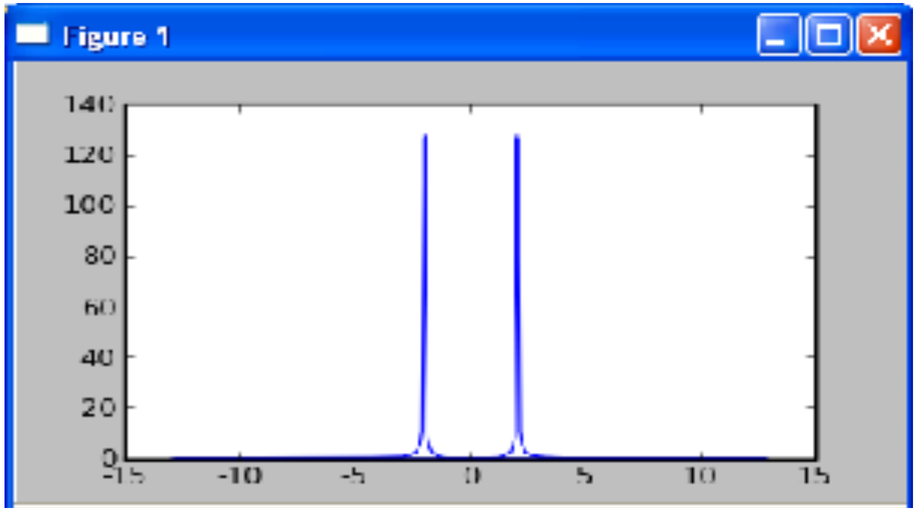
FFT

```
# Fast Fourier Transform
>>> from scipy.fftpack import *
>>> f = 2 # Hz
>>> w = 2*pi * f
>>> t = linspace(0,10,256)
>>> s = sin(w*t)
>>> S = fft(s)
>>> plot(abs(S))
```



FFTFREQ

```
# Frequencies in Hz for
# fft samples.

# Sampling rate
>>> rate = 10/256.
>>> f = fftfreq(256, rate)
>>> plot(f,abs(S))
```

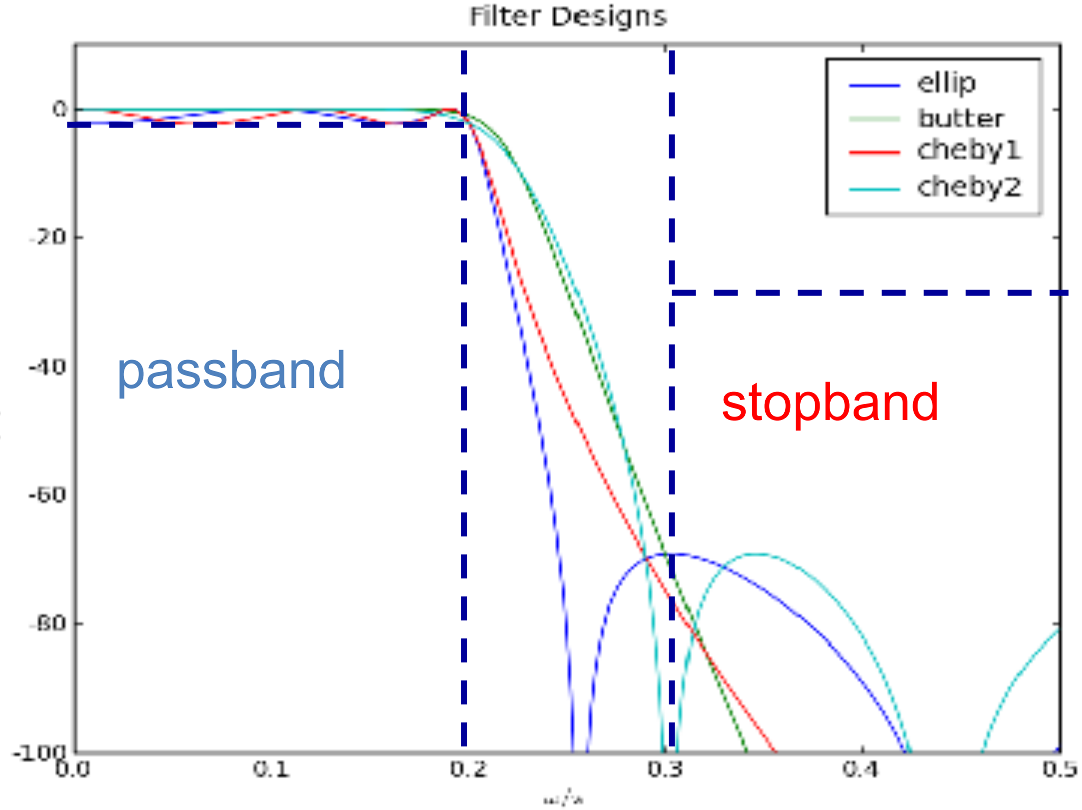


Filter Design

```
iirdesign(pass_edges, stop_edges, maximum_loss_in_passband,
          minimum_attenuation_in_stopband, ftype=type_of_filter)
```

IIR FILTER DESIGN

```
# Infinite Impulse Response
#  filter design
>>> import scipy.signal as ss
>>> for ftype in ['ellip', 'butter',
...                  'cheby1', 'cheby2']:
...      b,a = ss.iirdesign(0.2, 0.3,
...                          1, 30,
...                          ftype=ftype)
...      w,h = ss.freqz(b, a)
...      plot(w/pi, 20*log(abs(h)),
...          label=ftype)
>>> xlabel(r'$\omega/\pi}$')
>>> ylabel('|H| (dB)')
>>> title('Filter Designs')
>>> legend()
```



LTI Systems

```
>>> b,a = [1],[1,6,25]
>>> ltisys = signal.lti(b,a)
>>> t,h = ltisys.impulse()
>>> ts,s = ltisys.step()
>>> plot(t,h,ts,s)
>>> legend(['Impulse response','Step response'])
```

$$H(s) = \frac{1}{s^2 + 6s + 25}$$

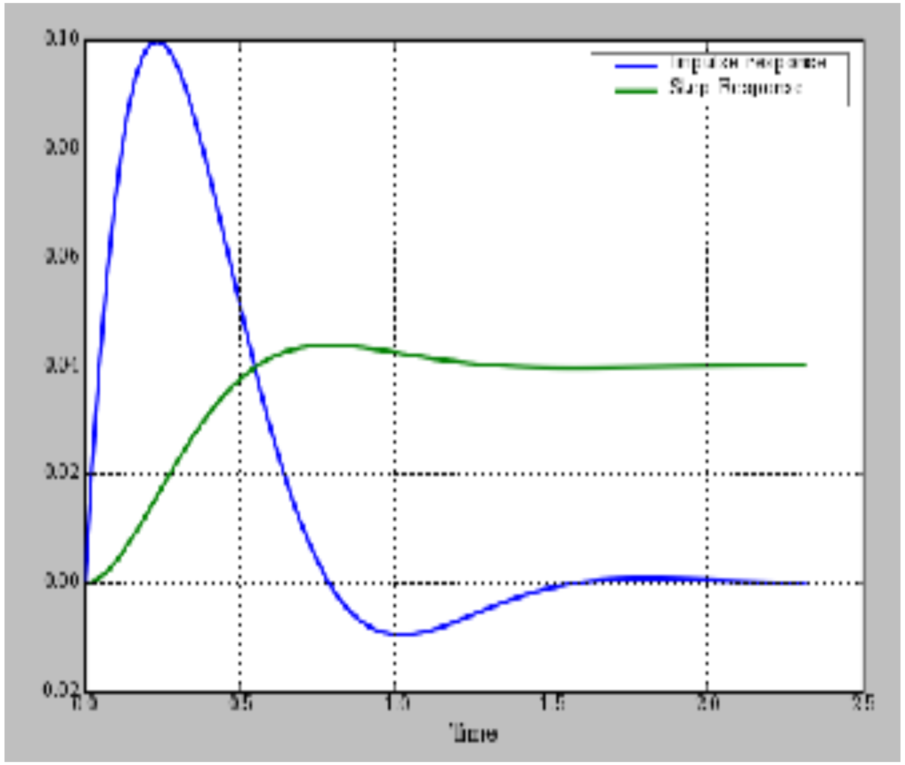
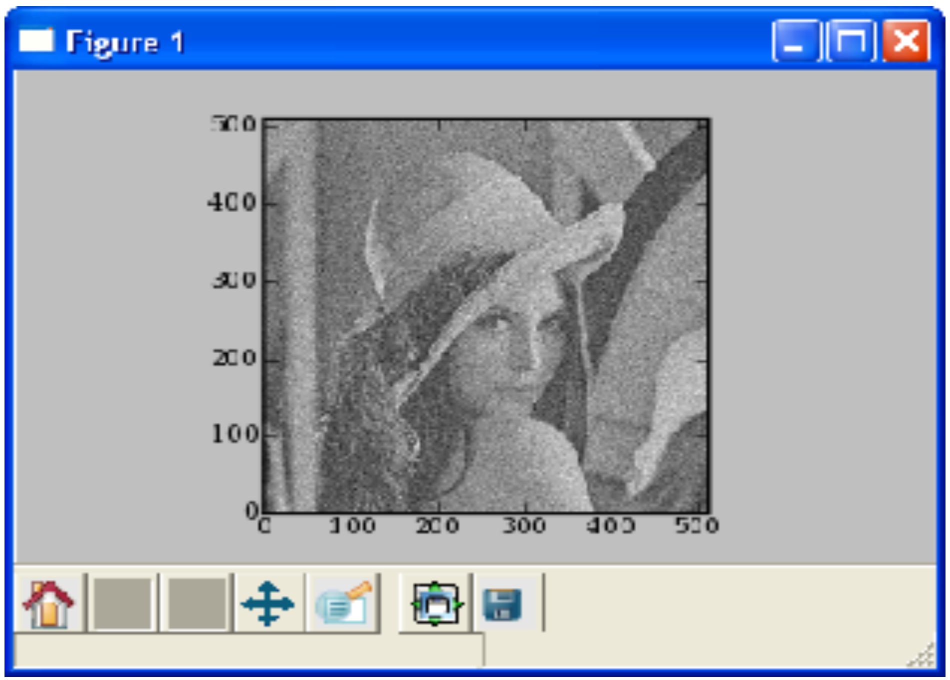


Image Processing

```
# Noise removal using wiener filter
>>> from scipy.stats import norm
>>> ln = lena + norm(0,32).rvs(lena.shape)
>>> imshow(ln)
>>> cleaned = signal.wiener(ln)
>>> imshow(cleaned)
```

NOISY IMAGE



FILTERED IMAGE

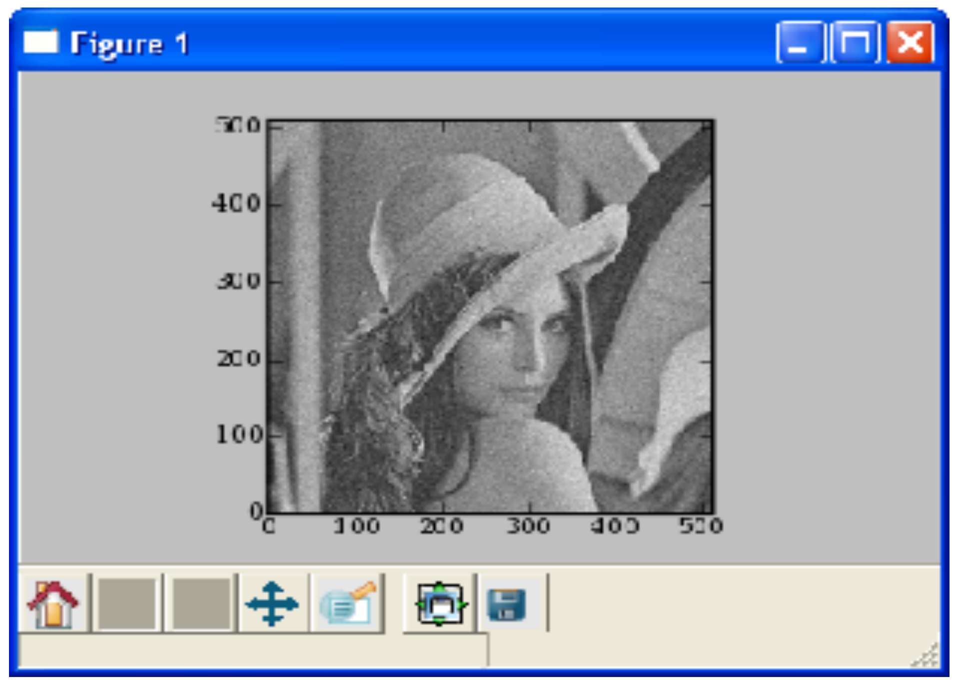
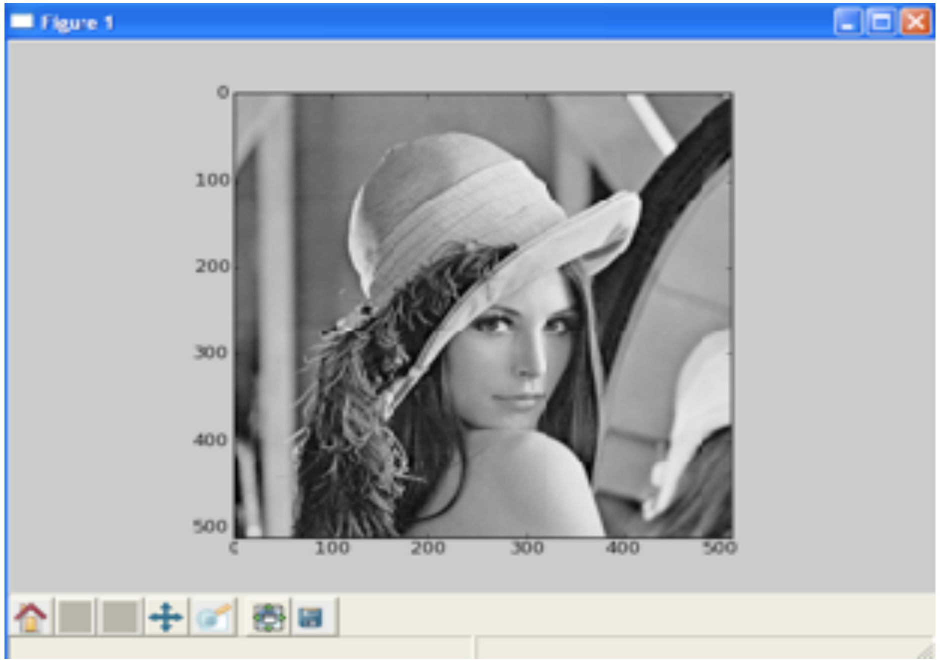


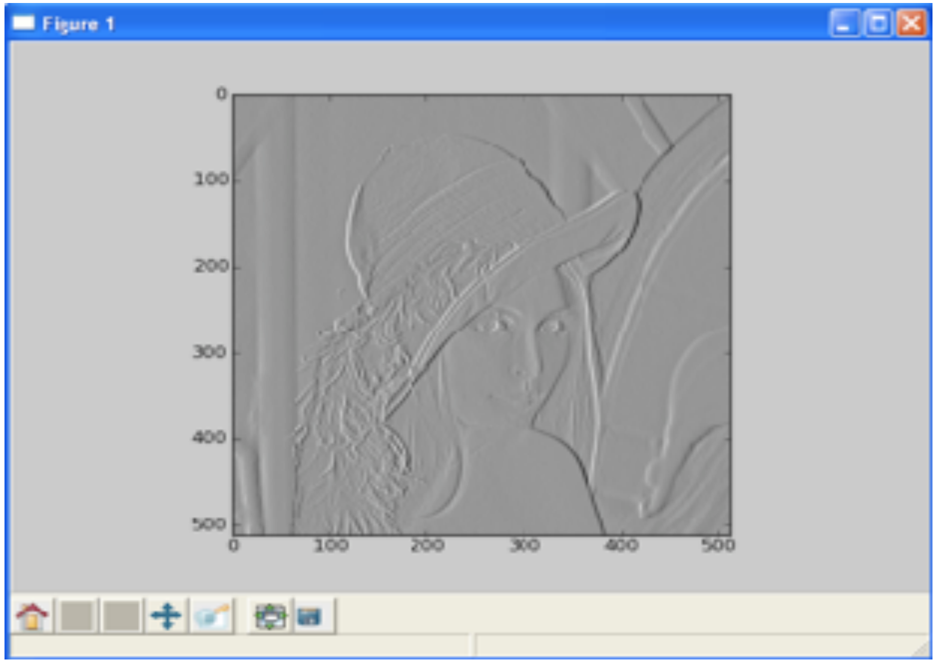
Image Processing

```
# Edge detection using Sobel filter
>>> from scipy.ndimage.filters import sobel
>>> imshow(lena)
>>> edges = sobel(lena)
>>> imshow(edges)
```

NOISY IMAGE



FILTERED IMAGE



Integration

scipy.integrate --- General purpose Integration

- **Ordinary Differential Equations (ODE)**

  `integrate.odeint`, `integrate.ode`

- **Samples of a 1-d function**

  `integrate.trapz` (trapezoidal Method), `integrate.simps` (Simpson Method), `integrate.romb` (Romberg Method)

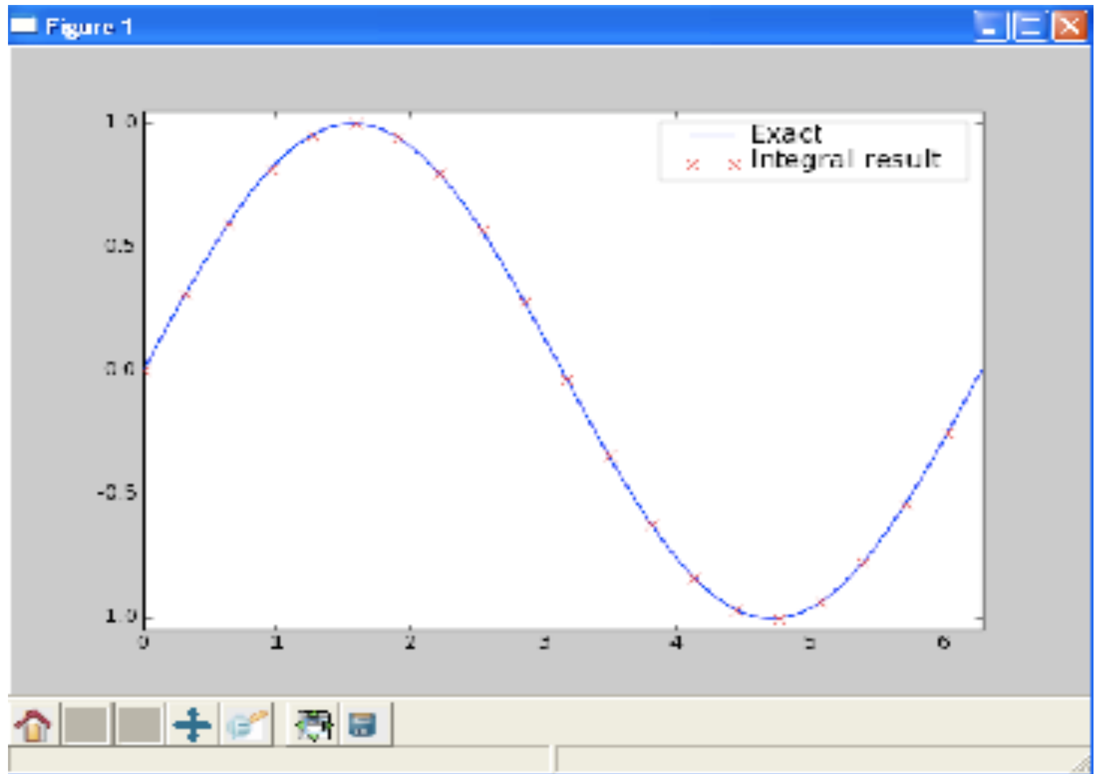
- **Arbitrary callable function**

  `integrate.quad` (general purpose), `integrate.dblquad` (double integration), `integrate.tplquad` (triple integration), `integrate.fixed_quad` (fixed order Gaussian integration), `integrate.quadrature` (Gaussian quadrature to tolerance), `integrate.romberg` (Romberg)

Integration

EXAMPLE (INTEGRATE A CALLABLE)

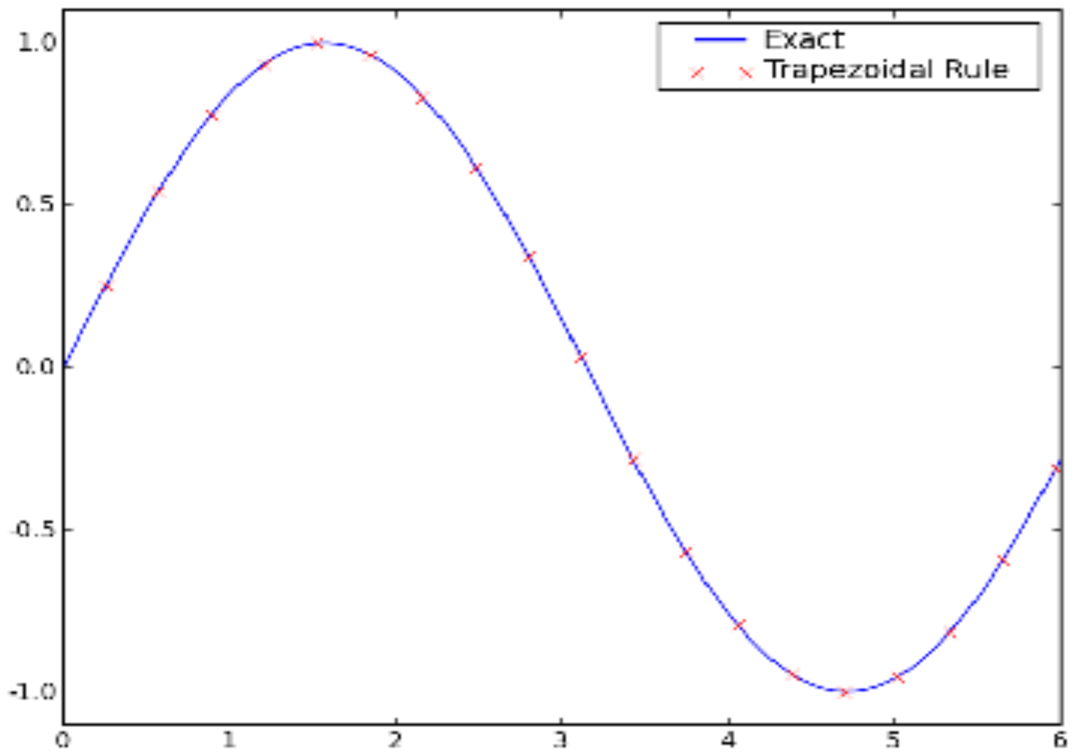
```
# Compare sin to integral(cos)
>>> def func(x):
     return integrate.quad(cos,0,x)
[0]
>>> vecfunc = vectorize(func)

>>> x = r_[0:2*pi:100j]
>>> x2 = x[::5]
>>> y = sin(x)
>>> y2 = vecfunc(x2)
>>> plot(x,y,x2,y2,'rx')
>>> legend([‘Exact‘,
...           ‘Integral Result’])
```



Integration

EXAMPLE (INTEGRATE FROM SAMPLES)

```
# Compare sin to integral(cos)
>>> x = r_[0:2*pi:100j]
>>> y = sin(x)
>>> fx = cos(x)
>>> N = len(x)+1
>>> y2 = [trapz(fx[:i],x[:i])
...          for i in range(5,N,5)]
>>> x2 = x[4::5]
>>> plot(x,y,x2,y2,'rx')
>>> legend(['Exact',
...             'Trapezoidal Rule'])
```



Ordinary Differential Equation

```
>>> from scipy.integrate import odeint
```

Find $y(t)$ given $y(t_0)$ and

$$\frac{dy}{dt} = f(y,t)$$

```
odeint(func, y0, t, args=())
```

```
func - func(y,tn,...) calculates dy/dt at tn, y can
       be a vector; func should then return the
       same length vector
y0   - initial value of y (can be a vector)
t    - a sequence of time points at which to solve for
       y, the initial value, t0, should be the first
       element of the sequence
args - any extra arguments needed by func
```

MATLAB FILES

SAVE FILE

```
>>> from scipy.io import mio
>>> mio.savemat('tst.mat',
                {'x': 1,
                 'y': 2})
```

LOAD MATLAB 7.1 OR BEFORE

```
>>> mf = mio.loadmat('tst.mat')
>>> print mf['x']
1
>>> print mf['y']
2
```

LOAD - MATLAB 7.3 OR AFTER

```
# Matlab 7.3 uses HDF files
# by default.  The 'tables'
# library handles hdf files
# well.
>>> import tables
>>> f = tables.openFile('tst.mat')
>>> x = file.root.x[:]
>>> print x
1
>>> y = file.root.y[:]
>>> print y
2
```

GA and Clustering

scipy.sandbox.ga --- Basic Genetic Algorithm Optimization

Routines and classes to simplify setting up a genome and running a genetic algorithm evolution

scipy.cluster --- Basic Clustering Algorithms

- **Observation whitening** `cluster.vq.whiten`
- **Vector quantization** `cluster.vq.vq`
- **K-means algorithm** `cluster.vq.kmeans`

Enthought Tool Suite

TRAITS

Initialization, Validation, Observation, and Visualization of Python class attributes

KIVA

2D primitives supporting path based rendering, affine transforms, alpha blending and more.

ENABLE

Object based 2D drawing canvas

CHACO

Plotting toolkit for building complex, interactive 2D plots

MAYAVI

3D Visualization of Scientific Data based on VTK

ENVISAGE

Application plugin framework for building scriptable and extensible applications

What are traits?

Traits provide additional characteristics for Python object attributes:

- Notification
- Visualization
- Others
  - Validation
  - Initialization
  - Delegation

Defining Simple Traits

```
from enthought.traits.api import HasTraits, Float

class Rectangle(HasTraits): # <----- Derive from HasTraits
    """ Simple rectangle class with two traits.
    """
    # Width of the rectangle
    width = Float   # <----- Declare Traits

    # Height of the rectangle
    height = Float  # <----- Declare Traits
```

Note: Run `main()` for this example to execute similar commands to those below.

```
In[1]: run rect_1.py
In[2]: main()
```

```
# Demo Code
>>> rect = Rectangle()
>>> rect.width
0.0

# Set rect width to 1.0
>>> rect.width = 1.0
1.0
```

```
# Float traits convert integers
>>> rect.width = 2
>>> rect.width
2.0

# THIS WILL THROW EXCEPTION
>>> rect.width = "1.0"
TraitError: The 'width' trait of a
Rectangle instance must be a value of
type 'float', but a value of 1.0 was
specified.
```

Traits for Basic Python Types

Coercing Trait	Casting Trait	Python Type	Default Value
Bool	CBool	bool	False
Complex	CComplex	complex	0+0j
Float	CFloat	float	0.0
Int	CInt	int	0
Long	CLong	long	0L
Str	CStr	str or unicode (whichever assigned)	''
Unicode	CUnicode	unicode	u''

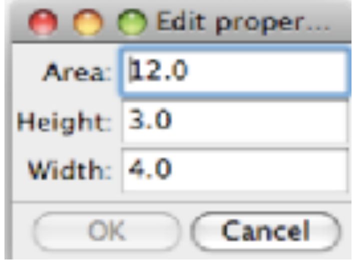
Derived properties

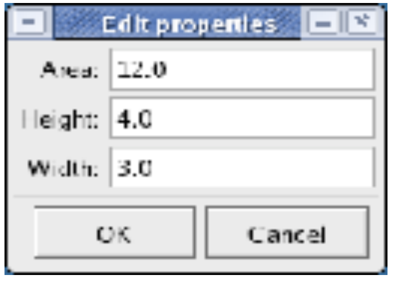
```
from enthought.traits.api import \
    HasTraits, Float, Property

class Rectangle(HasTraits):
    """ Rectangle class with
        read-only area property.
    """
    # Width of the rectangle
    width = Float(1.0)
    # Height of the rectangle
    height = Float(2.0)

    # The area of the rectangle
    # Defined as a property.
    area = Property
    # specially named method
    # automatically associated
    # with area.
    def _get_area(self):
        return self.width * self.height
```

```
# Demo Code

>>> rect = Rectangle(width=2.0,
                     height=3.0)
>>> rect.area
6.0
>>> rect.width = 4.0
>>> rect.area
8.0
```

Traits UI - Default Views

```
>>> rect = Rectangle(width=3.0, height = 4.0)
# Create a UI to edit the traits of the object.
>>> rect.edit_traits()
```





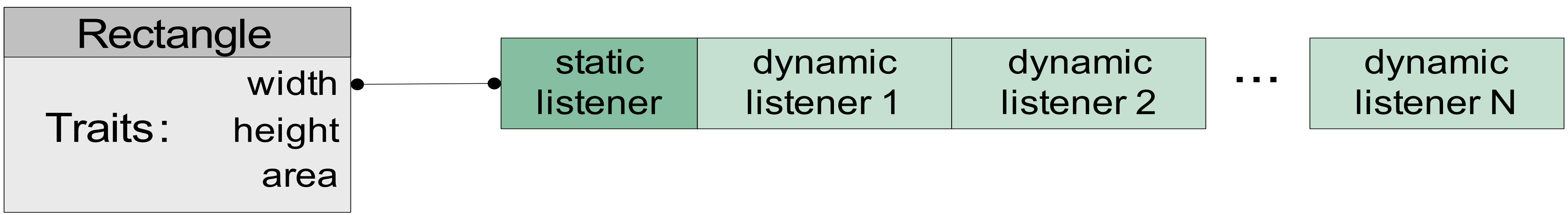
Trait Listeners

HasTraits Object

All traits automatically support the Listener Pattern

Listener Functions and Methods

Listeners are called in order whenever the ‘width’ trait changes



Static Trait Notification

```
class Amplifier(HasTraits):
      """ Guitar Amplifier Model
      """

      # Volume setting for the amplifier.
      volume = Range(0.0, 11.0, default=5.0)

      # Static observer method called whenever volume is set.
      def _volume_changed(self, old, new):
            if new == 11.0:
                  print "This one goes to eleven"
```

```
# Demo Code
>>> spinal_tap = Amplifier()
>>> spinal_tap.volume = 11.0
This one goes to eleven
>>> spinal_tap.volume = 11.0 # nothing is printed because
                             # the value didn't change.
```

Dynamic Trait Notification

```
class Amplifier(HasTraits):
      """ Guitar Amplifier Model
      """

      # Volume setting for the amplifier.
      volume = Range(0.0, 11.0, default=5.0)


def printer(value):
      print "new value:", value
```

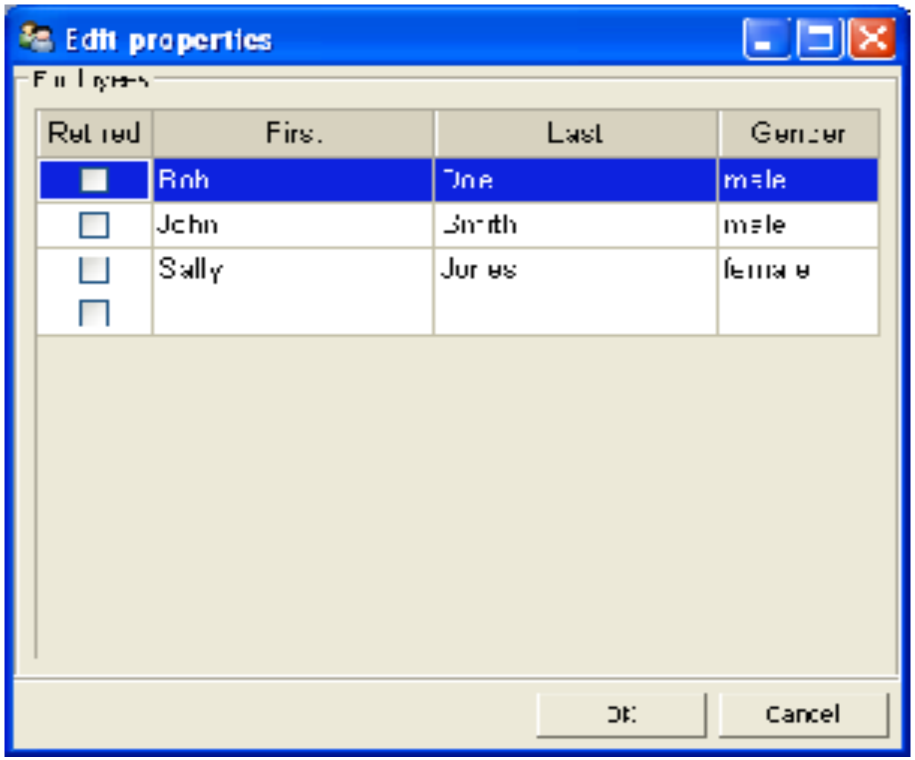
```
# Demo Code
>>> spinal_tap = Amplifier()
# In the following, name can also be a list of trait names
>>> spinal_tap.on_trait_change(printer, name='volume')
>>> spinal_tap.volume = 11.0
new value: 11.0
```

@on_trait_change decorator

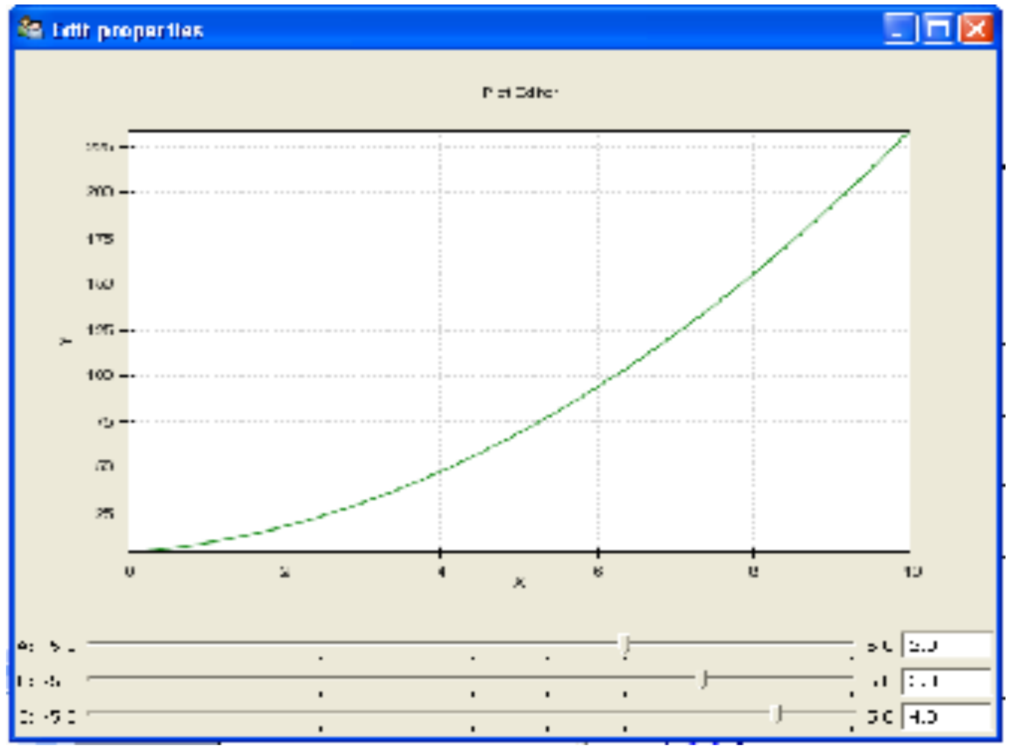
```
from enthought.traits.api import HasTraits, Range, on_trait_change

class Amplifier(HasTraits):
    """ Guitar Amplifier Model
    """

    volume = Range(0.0, 11.0, value=5.0)
    reverb = Range(0, 10.0, value=5.0)

    # The on_trait_change decorator can listen to multiple traits
    # Note the "list" of traits is specified as a string.
    @on_trait_change('reverb, volume')
    def update(self, name, value):
        print 'trait %s updated to %s' % (name, value)

# Demo Code
>>> spinal_tap = Amplifier()
>>> spinal_tap.volume = 11.0
trait volume updated to 11.0
>>> spinal_tap.reverb = 2.0
trait reverb updated to 2.0
```

UI Demos

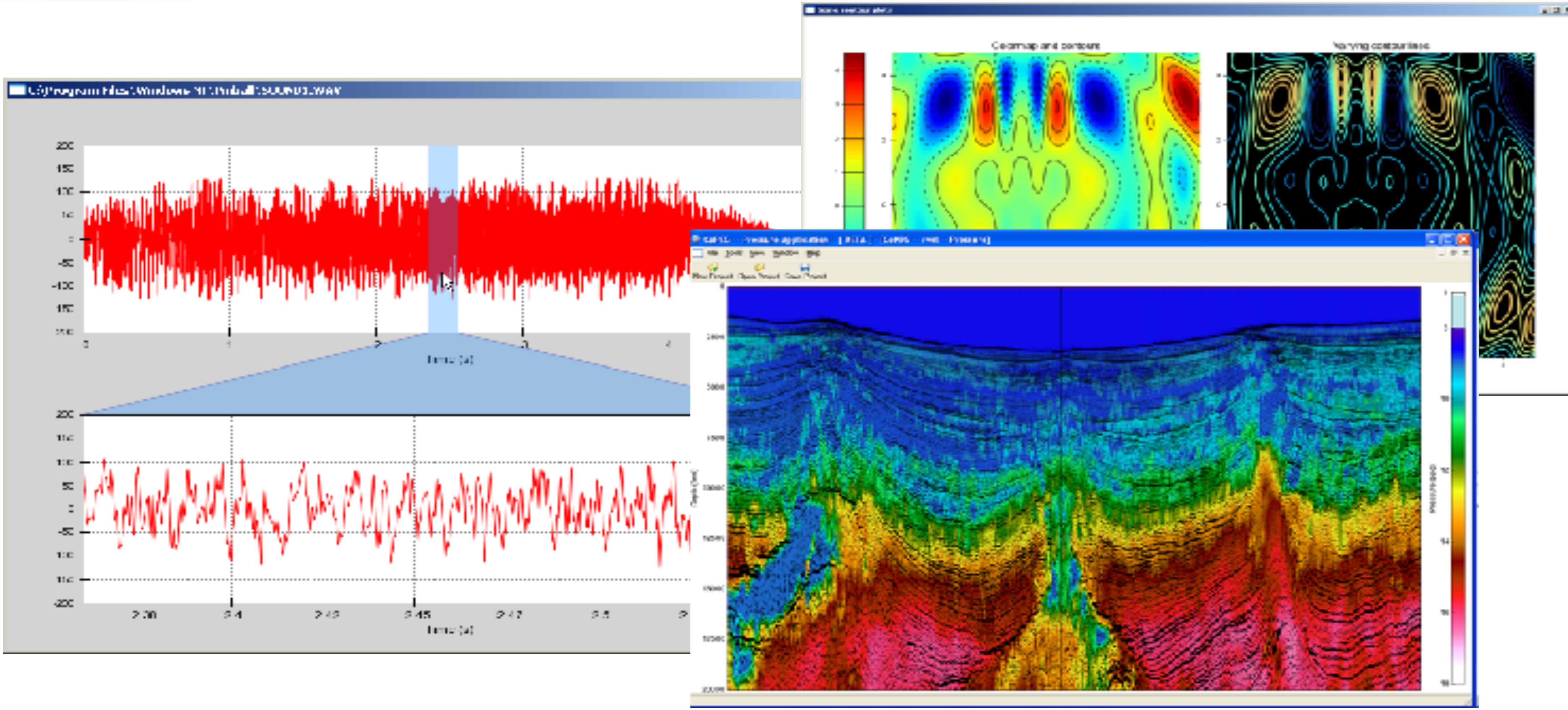
Table Demo



Polynomial Demo



Chaco: Interactive Graphics



Contexts with Events

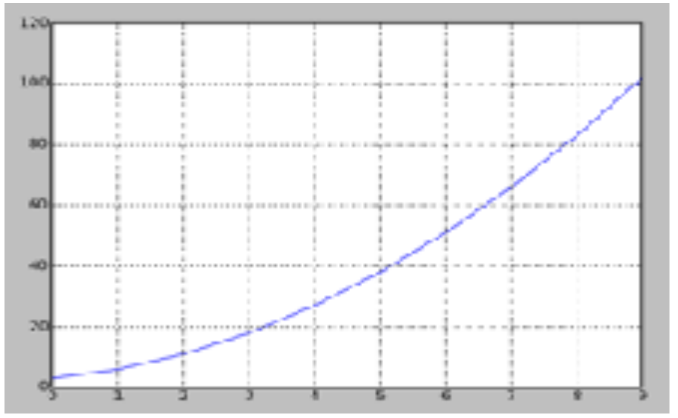
Code Block

```
a=1
b=2
c=3
y = a*x**2+b*x+c
```

Context

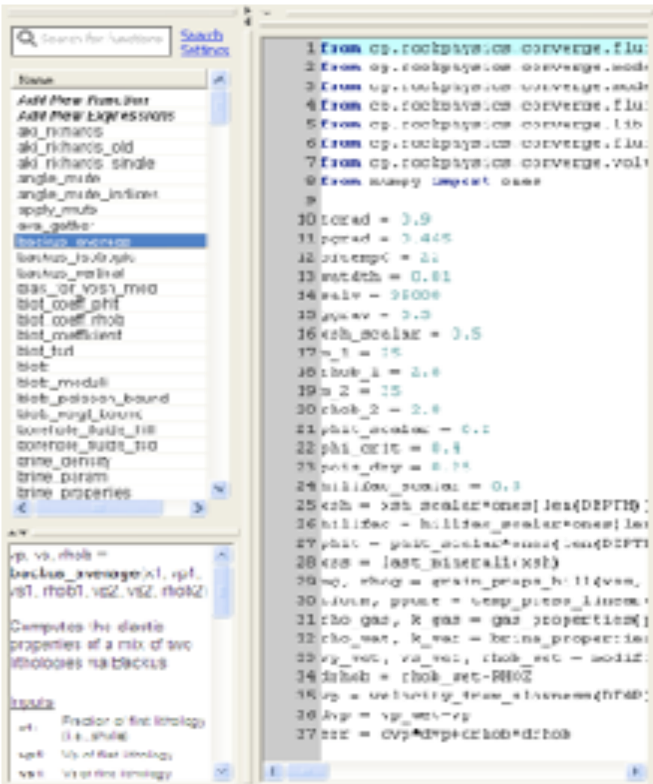
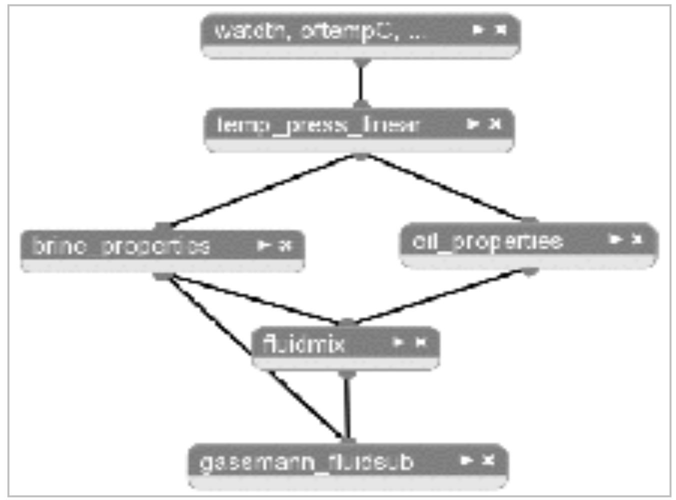
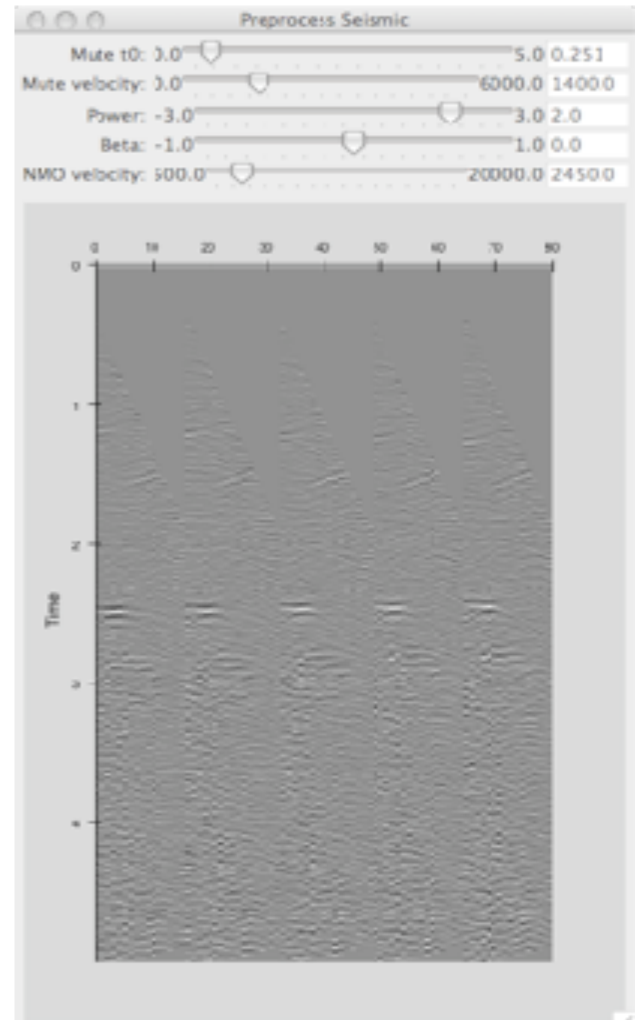
```
x: array( 0…9 )
a: 1
b: 2
c: 3
y: array( 3…102 )
```

Events Fire when data changes



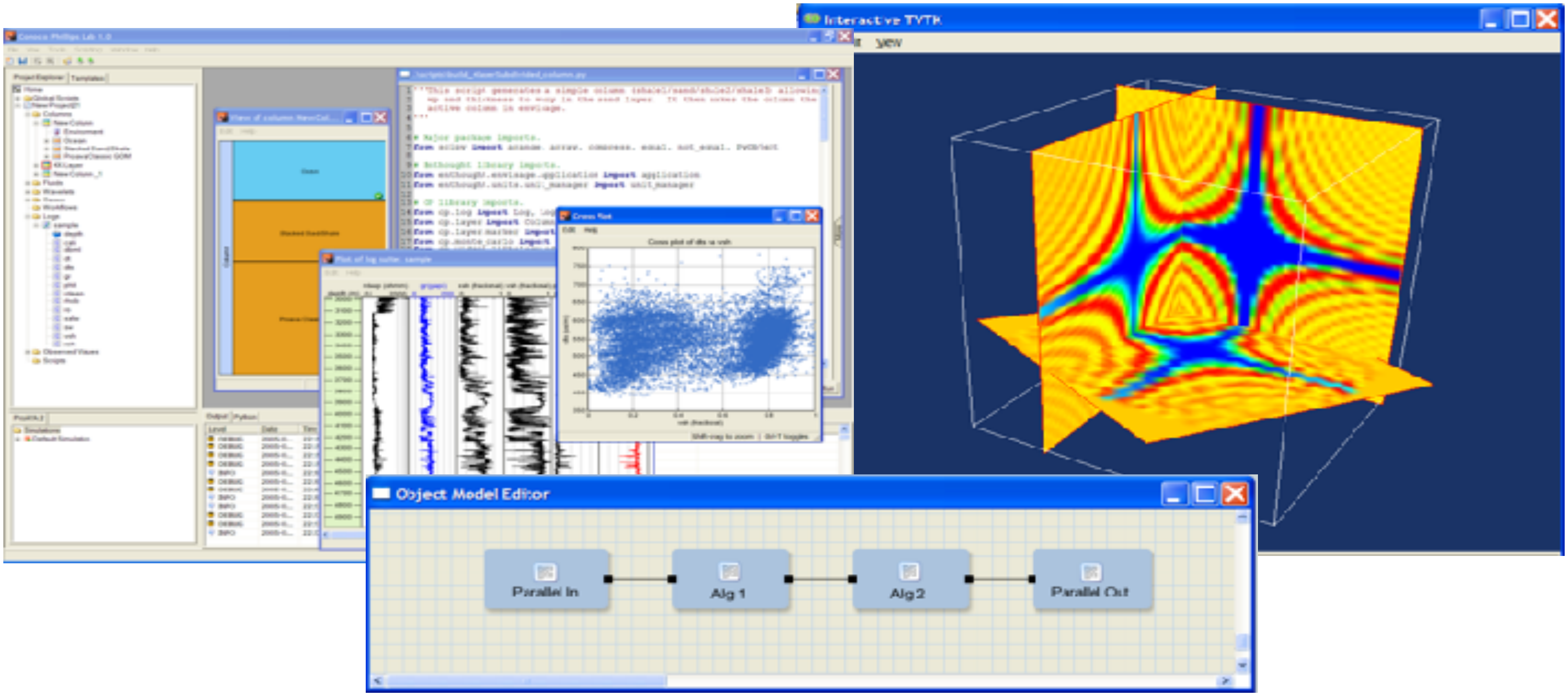
Updating Data View

Three Classes of Users

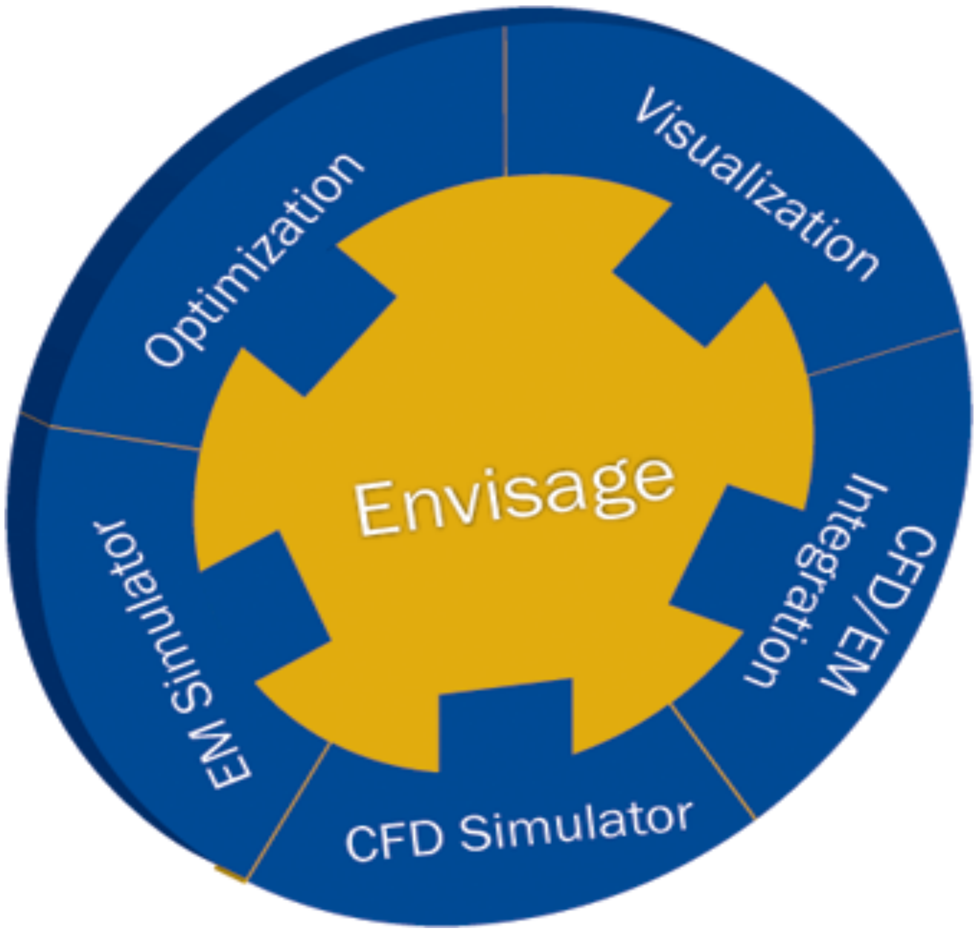
EXPERT	EXPERIENCED	NOVICE
		

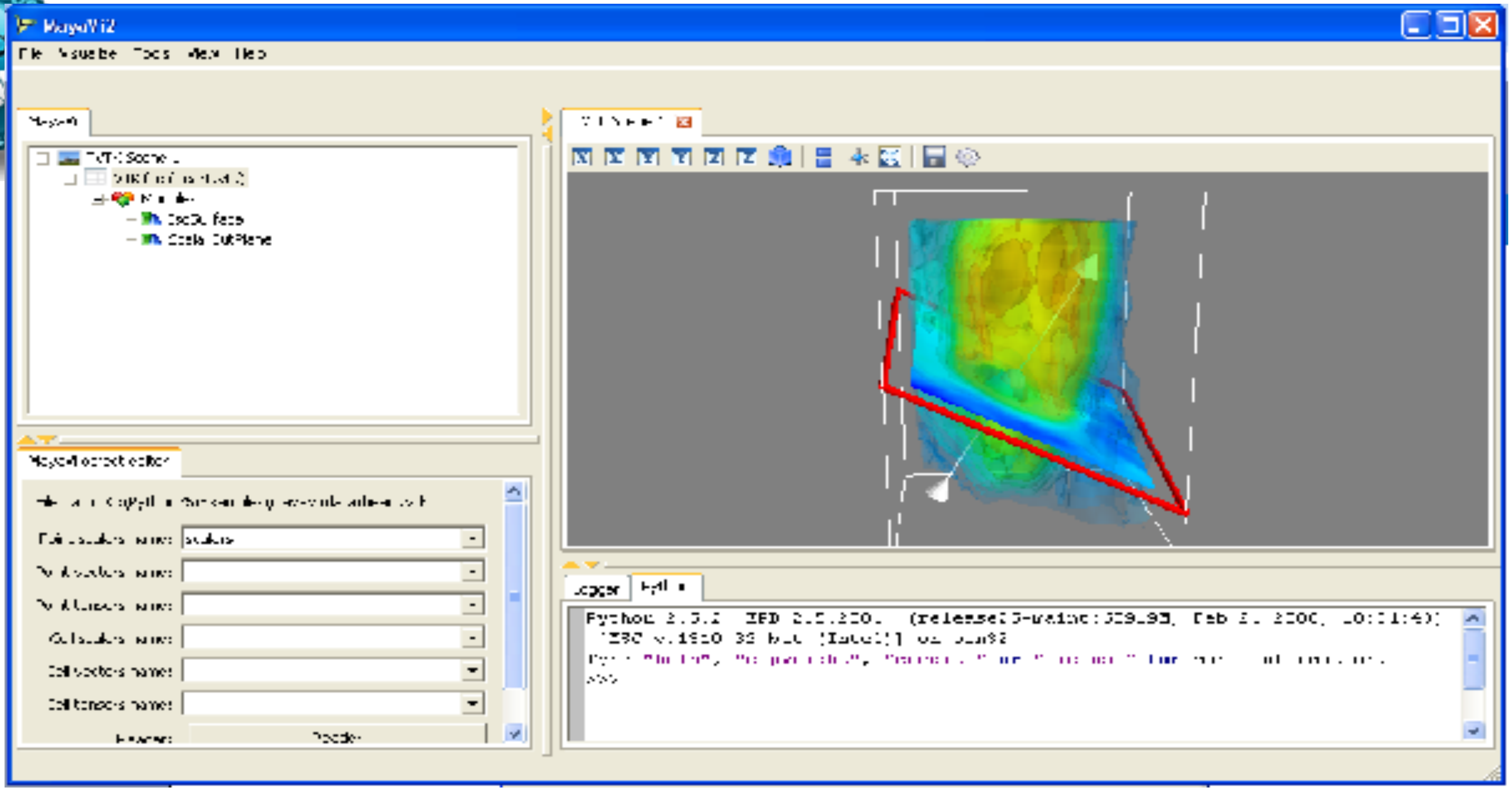
Envisage Application Framework

- Extensible
- Scriptable
- “Humane” Interface Tools built-in
- Easy Deployment



Multiple Plug-ins. One Application





Rich Client App (Geophysics, Finance, Etc)

Testing Framework Scripting Interface

Data Display

Equipment Interface

Compliance Tools

Scientific Algorithms

Database Access

Chaco Plotting

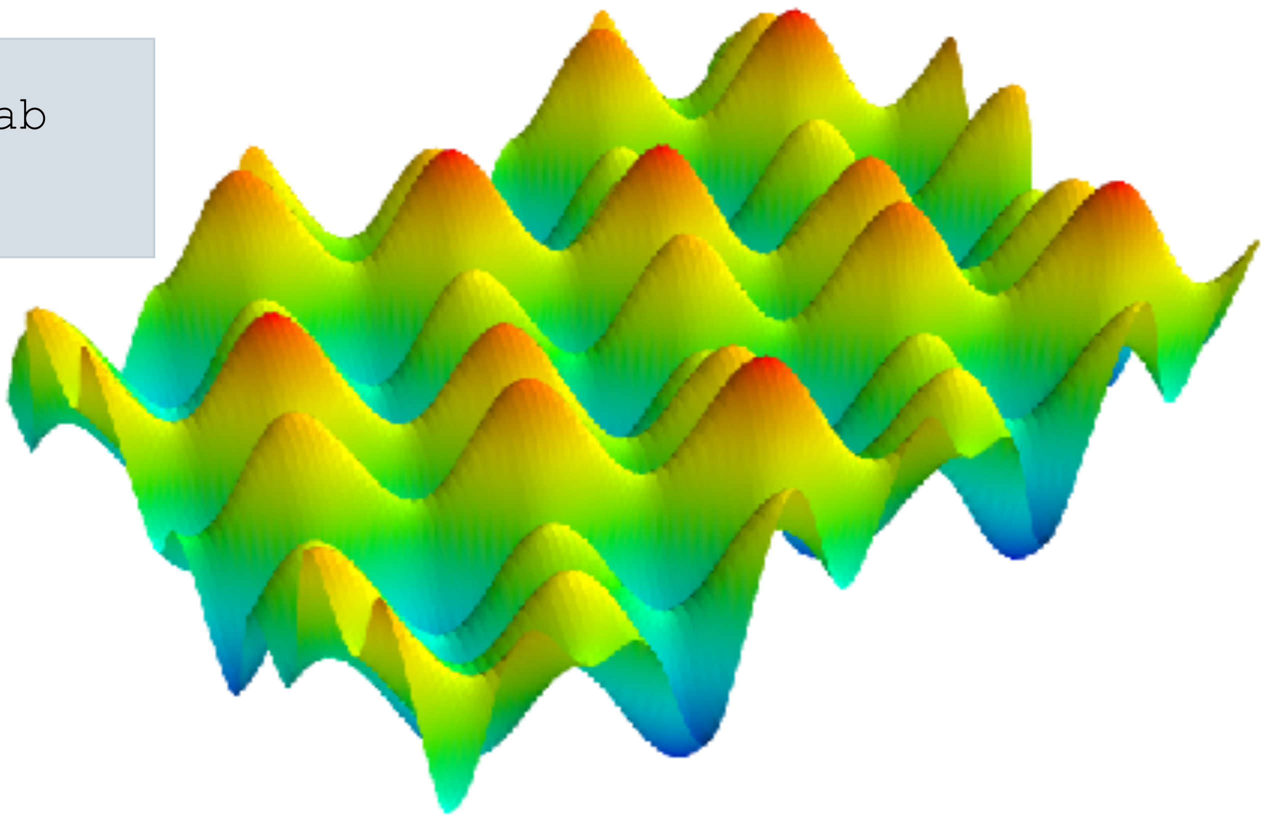
UI Elements

3D Visualization

```
C:\ ipython –wthread
```

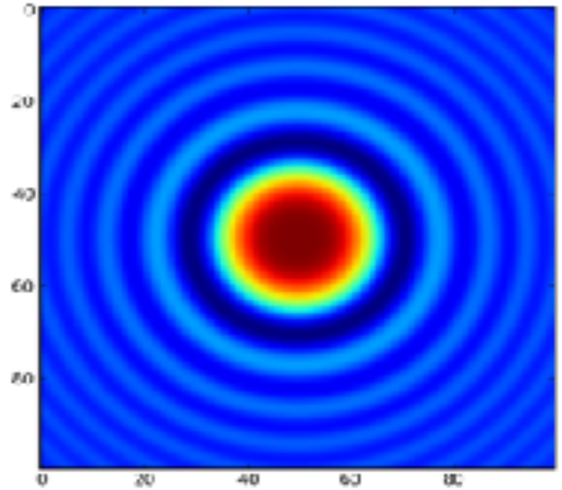
```
>>> from enthought.mayavi import mlab
```

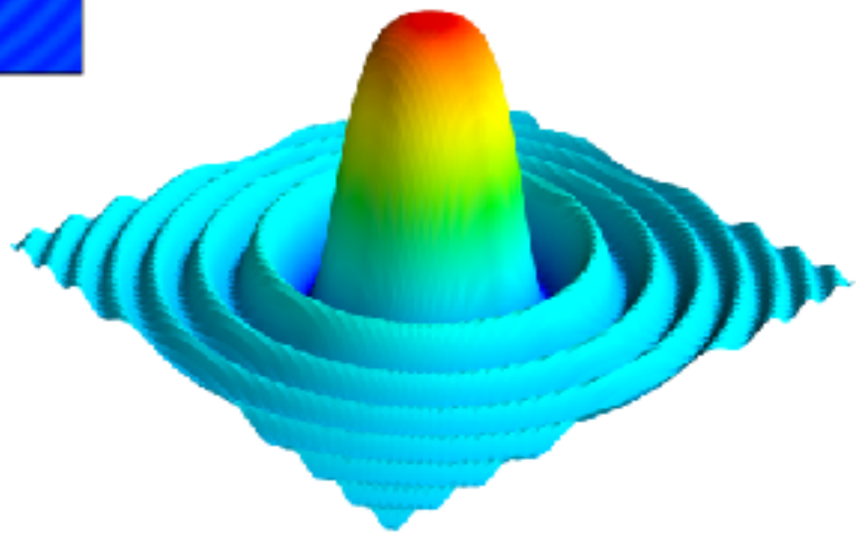
> `matplotlib` also has an `mlab` namespace. Be sure you are using the one from **`enthought.mayavi`**



One example

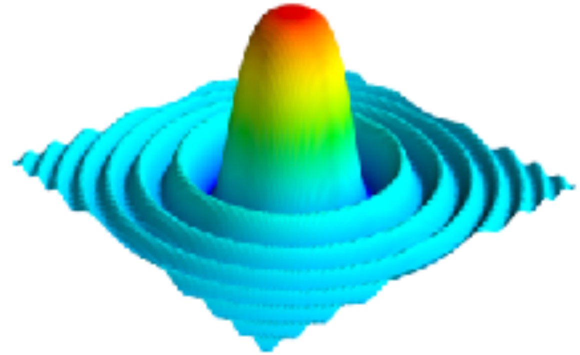
```
# create arrays
x, y = mgrid[-5:5:100j,-5:5:100j]
r = x**2 + y**2
z = sin(r)/r

# plot with pylab
imshow(z)

# plot with mayavi
from enthought.mayavi import mlab
mlab.surf(50*z)
```





Plotting commands

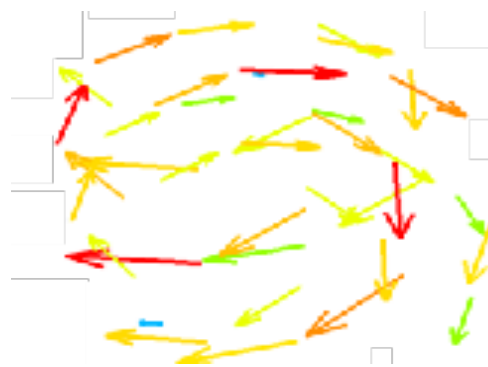
0D data
`mlab.points3d(x, y, z)`



2D data
`mlab.surf(x, y, z)`

3D data
`mlab.contour3d(x, y, z)`

1D data
`mlab.plot3d(x, y, z)`

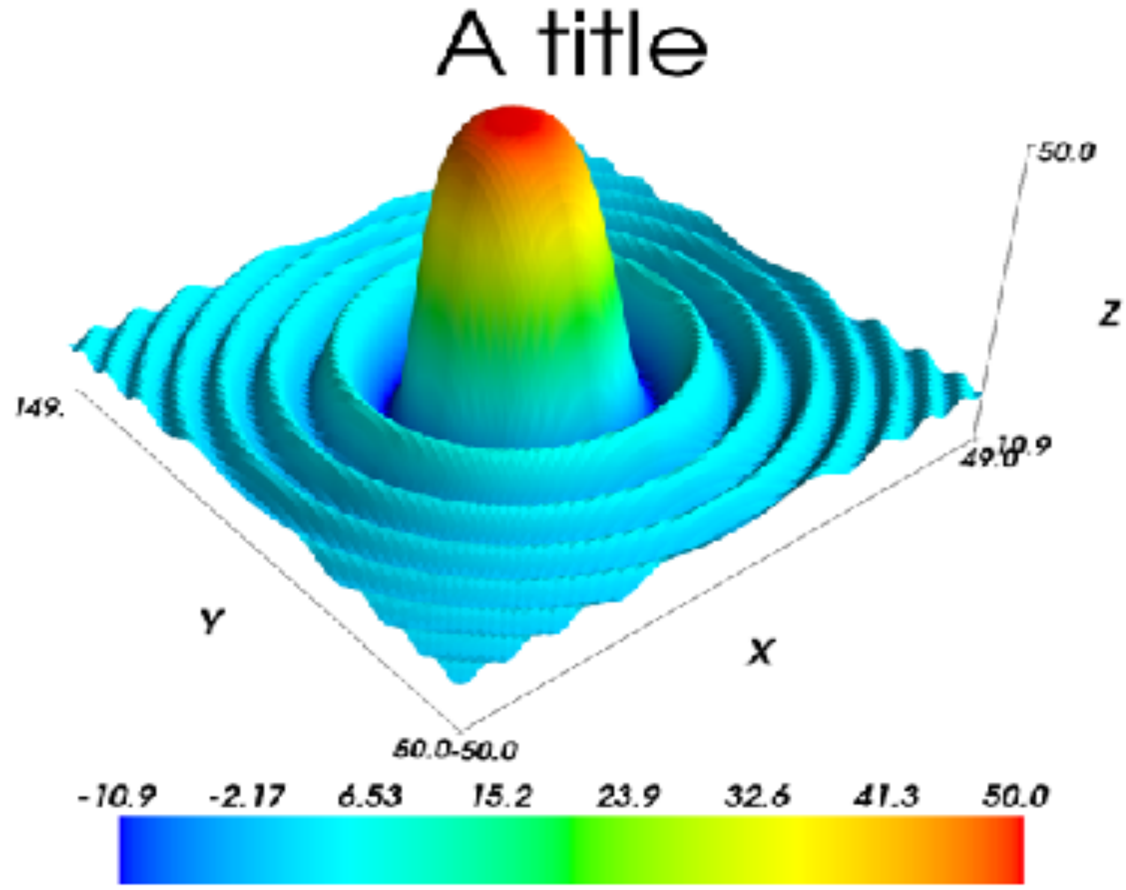


Vector field
`mlab.quiver(x, y, z, u, v, w)`

Figure decorations

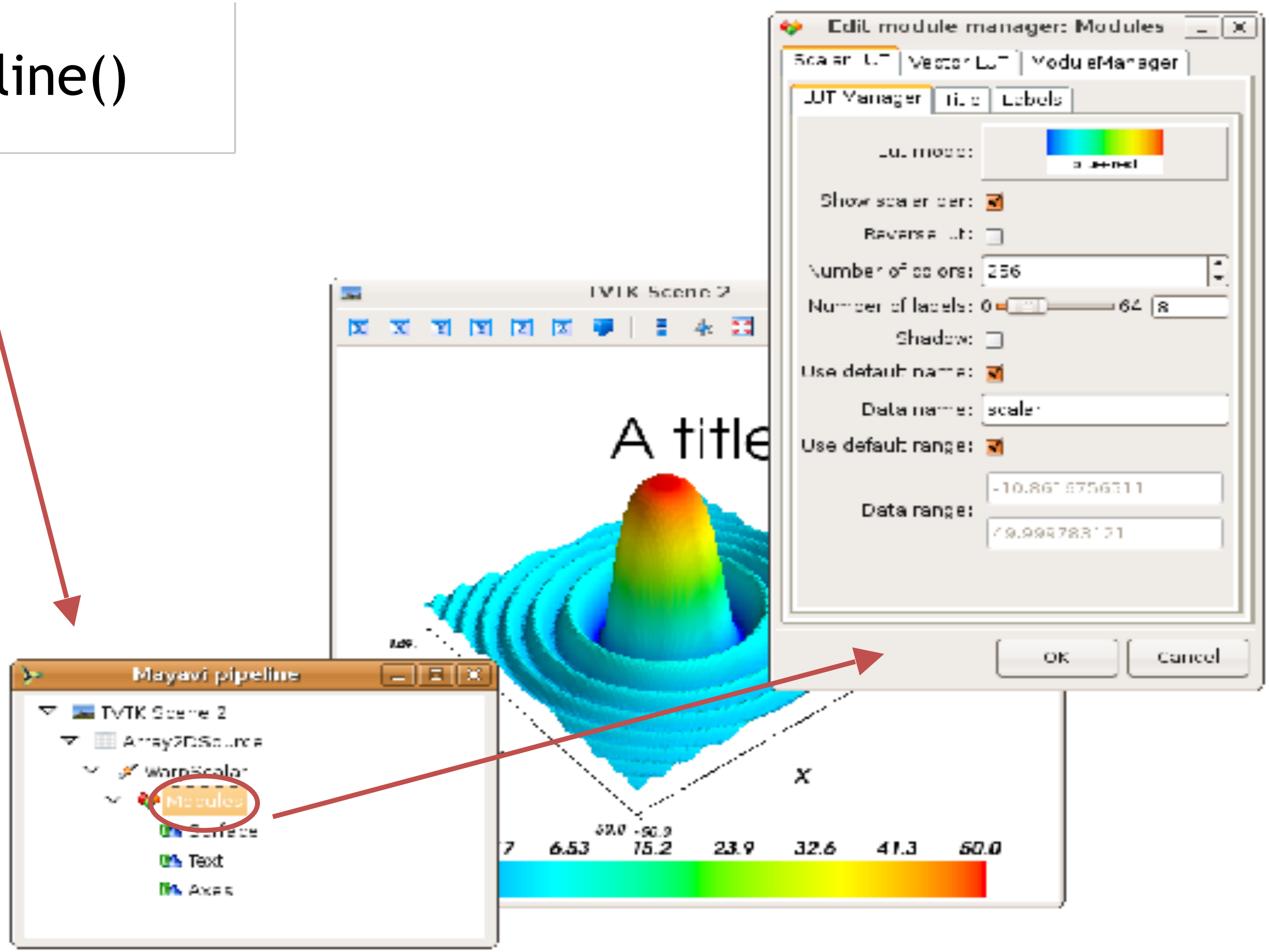
- `mlab.title('A title')`
- `mlab.axes()`
- `mlab.colorbar()`

- `mlab.clf()`
- `mlab.figure()`
- `mlab.gcf()`



Graphical User Interface

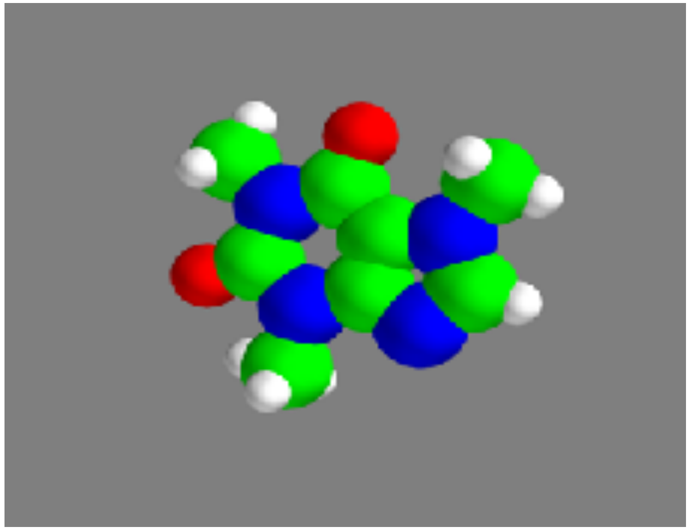
mlab.show_pipeline()



Examples and Demos

mlab.test_points3d()
mlab.test_plot3d()
mlab.test_surf()
mlab.test_contour3d()
mlab.test_quiver3d()

mlab.test_molecule()
mlab.test_flow()
mlab.test_mesh()



Use ?? in IPython to look at the source code of these examples.

Thank You