Python & SciPy Modules

Science and Technology
Support Group

20,21 February 2007

SciPy

Open Source Scientific Tools for Python

Developed By Enthought
Author: Eric Jones, Travis Oliphant,
Pearu Peterson and others

www.scipy.org

Contacts

- Peter Carswell
  - pete@osc.edu
  - (614) 292-1091
- OSC Help
  - oschelp@osc.edu
  - (locally) 292-1800
  - 1-800-686-6472

Download Example Code

- Download the sample code for day 2 :

(Type entire command on one line, then hit return)

```
$ wget
    http://www.osc.edu/~pete/python/examples/day02/Day2Examples.tgz
```

- Un-tar the archive using the following command

```
$ tar –xzvf Day2Examples.tgz
```

- All class materials will be available here :

  http://www.osc.edu/~pete/python

What is SciPy ?

- Python module for scientific computing
- Python style license
- Developed by Enthought. Authors : Author: Eric Jones, Travis Oliphant, Pearu Peterson and others
- Core based on NumPy
- Provides many numerical routines for statistics, optimization, signal/image processing, etc.

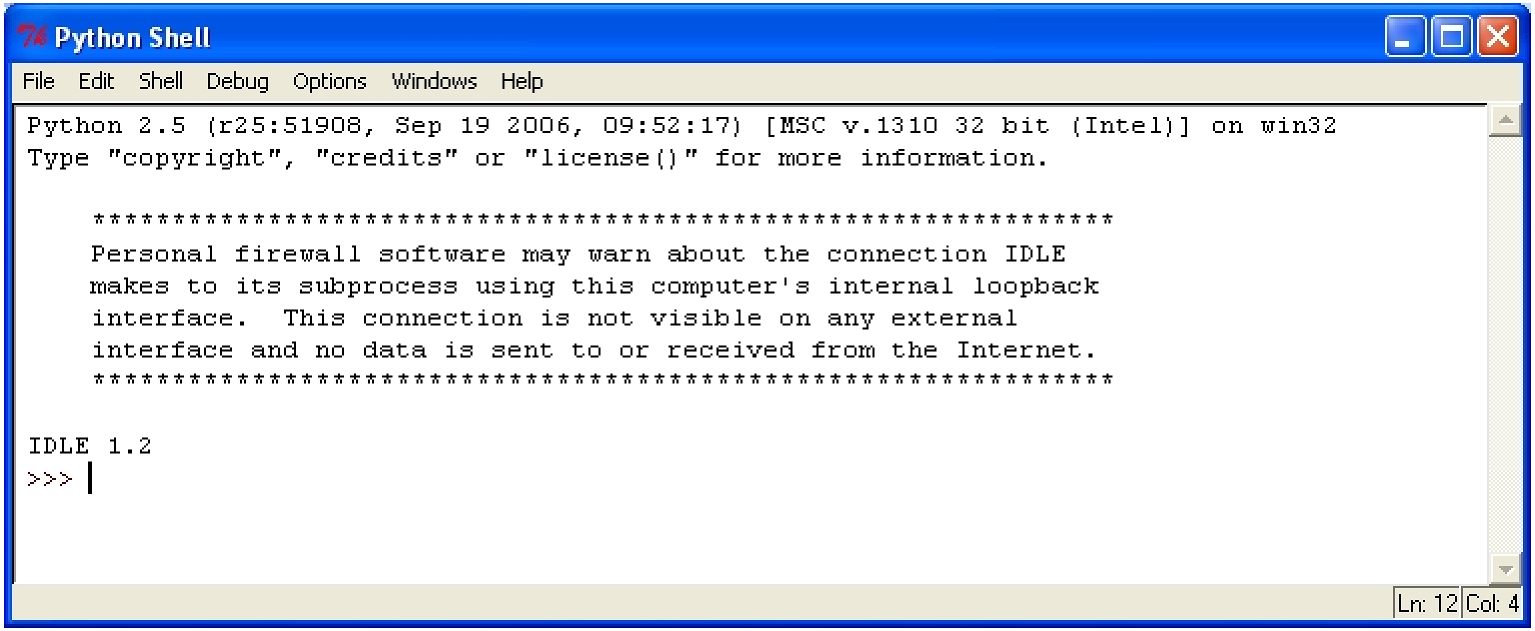
Python Editor

- Using emacs as the editor
  - “>> emacs &”
- Edit save and run
  - “>> python filename.py”
- Invoke python in a shell
  - “>> python”
  - In the interactive shell

```
Put image of screen dump here
```

Python Editor

- Start the python editor shell
  - **idle &**



```
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2
>>> 
```

Python Editor

- Other Python editors
  - IPython: an Enhanced Python Shell
    - Multiplatform, Free Software project (BSD licensed)
    - http://ipython.scipy.org/moin/
  - PyPE: limited portability
    - http://pype.sourceforge.net/index.shtml

Import SciPy

- Import names defined in the SciPy namespace
  - `>>> import scipy`
- Python provides facility for documentation strings
- Command ‘help’ in the pydoc module
  - Launches and interactive session
  - Allows for searching of keywords and modules
  - Running the command ‘help’ with an object as argument displays the documentation string of the object

```
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()

Welcome to Python 2.4!  This is the online help utility.
....
>>> help> modules
Please wait a moment while I gather a list of all available modules...
```

SciPy Functions

```
>>> dir(scipy)
        ['ALLOW_THREADS', 'BUFSIZE', 'CLIP', 'ERR_CALL', 'ERR_DEFAULT', 'ERR_DEFAULT2', 'ERR_IGNORE', 'ERR_LOG',
        'ERR_PRINT', 'ERR_RAISE', 'ERR_WARN', 'FLOATING_POINT_SUPPORT', 'FPE_DIVIDEBYZERO', 'FPE_INVALID',
        'FPE_OVERFLOW', 'FPE_UNDERFLOW', 'False_', 'Inf', 'Infinity', 'MAXDIMS', 'MachAr', 'NAN', 'NINF', 'NZERO', 'NaN', 'NumpyTest',
        'PINF', 'PZERO', 'PackageLoader', 'RAISE', 'RankWarning', 'SCIPY_IMPORT_VERBOSE', 'SHIFT_DIVIDEBYZERO', 'SHIFT_INVALID',
        'SHIFT_OVERFLOW', 'SHIFT_UNDERFLOW', 'ScalarType', 'ScipyTest', 'True_', 'UFUNC_BUFSIZE_DEFAULT',
        'UFUNC_PYVALS_NAME', 'WRAP', '__all__', '__builtins__', '__config__', '__doc__', '__file__', '__name__', '__numpy_version__',
        '__path__', '__version__', 'absolute', 'add', 'add_docstring', 'add_newdoc', 'add_newdocs', 'alen', 'all', 'allclose', 'alltrue', 'alterdot', 'amax',
        'amin', 'angle', 'any', 'append', 'apply_along_axis', 'apply_over_axes', 'arange', 'arccos', 'arccosh', 'arcsin', 'arcsinh', 'arctan', 'arctan2',
        'arctanh', 'argmax', 'argmin', 'argsort', 'argwhere', 'around', 'array', 'array2string', 'array_equal', 'array_equiv', 'array_repr', 'array_split',
        'array_str', 'asanyarray', 'asarray', 'asarray_chkfinite', 'ascontiguousarray', 'asfarray', 'asfortranarray', 'asmatrix', 'asscalar', 'atleast_1d',
        'atleast_2d', 'atleast_3d', 'average', 'bartlett', 'base_repr', 'binary_repr', 'bincount', 'bitwise_and', 'bitwise_not', 'bitwise_or', 'bitwise_xor',
        'blackman', 'bmat', 'bool8', 'bool_', 'broadcast', 'byte', 'c_', 'can_cast', 'cast', 'cdouble', 'ceil', 'central_diff_weights', 'cfloat', 'char', 'character',
        'chararray', 'choose', 'clip', 'clongdouble', 'clongfloat', 'cluster', 'column_stack', 'comb', 'common_type', 'compare_chararrays', 'complex128',
        'complex192', 'complex64', 'complex_', 'complexfloating', 'compress', 'concatenate', 'conj', 'conjugate', 'convolve', 'copy', 'corrcoef',
        'correlate', 'cos', 'cosh', 'cov', 'cross', 'csingle', 'ctypeslib', 'cumprod', 'cumproduct', 'cumsum', 'delete', 'deprecate', 'derivative', 'diag',
        'diagflat', 'diagonal', 'diff', 'digitize', 'disp', 'divide', 'dot', 'double', 'dsplit', 'dstack', 'dtype', 'e', 'ediff1d', 'emath', 'empty', 'empty_like', 'equal',
        'errstate', 'exp', 'expand_dims', 'expm1', 'extract', 'eye', 'fabs', 'factorial', 'factorial2', 'factorialk', 'fastCopyAndTranspose', 'fft', 'fftpack', 'finfo',
        'fix', 'flatiter', 'flatnonzero', 'flexible', 'fliplr', 'flipud', 'float32', 'float64', 'float96', 'float_', 'floating', 'floor', 'floor_divide', 'fmod', 'format_parser',
        'frexp', 'frombuffer', 'fromfile', 'fromfunction', 'fromiter', 'frompyfunc', 'fromstring', 'generic', 'get_array_wrap', 'get_include',
        'get_numarray_include', 'get_numpy_include', 'get_printoptions', 'getbuffer', 'getbufsize', 'geterr', 'geterrcall', 'geterrobj', 'gradient', 'greater',
        'greater_equal', 'hamming', 'hanning', 'histogram', 'histogram2d', 'histogramdd', 'hsplit', 'hstack', 'hypot', 'i0', 'identity', 'ifft', 'imag',
        'index_exp', 'indices', 'inexact', 'inf', 'info', 'infty', 'inner', 'insert', 'int0', 'int16', 'int32', 'int64', 'int8', 'int_', 'int_asbuffer', 'intc', 'integer',
        'integrate', 'interpolate', 'intersect1d', 'intersect1d_nu', 'intp', 'invert', 'io', 'iscomplex', 'iscomplexobj', 'isfinite', 'isfortran', 'isinf', 'isnan',
        'isneginf', 'isposinf', 'isreal', 'isrealobj', 'isscalar', 'issctype', 'issubclass_', 'issubdtype', 'issubsctype', 'iterable', 'ix_', 'kaiser', 'kron', 'ldexp',
        'left_shift', 'lena', 'less', 'less_equal', 'lexsort', 'lib', 'linalg', 'linsolve', 'linspace', 'little_endian', 'load', 'loads', 'log', 'log10', 'log1p', 'log2',
        'logical_and', 'logical_not', 'logical_or', 'logical_xor', 'logn', 'logspace', 'longdouble', 'longfloat', 'longlong', 'ma', 'mat', 'math', 'matrix',
        'maxentropy', 'maximum', 'maximum_sctype', 'mean', 'median', 'memmap', 'meshgrid', 'mgrid', 'minimum', 'mintypecode', 'misc', 'mod',
        'modf', 'msort', 'multiply', 'nan', 'nan_to_num', 'nanargmax', 'nanargmin', 'nanmax', 'nanmin', 'nansum', 'nbytes', 'ndarray', 'ndenumerate',
        'ndim', 'ndimage', 'ndindex', 'negative', 'newaxis', 'newbuffer', 'nonzero', 'not_equal', 'number', 'obj2sctype', 'object0', 'object_', 'ogrid',
        'oldnumeric', 'ones', 'ones_like', 'optimize', 'outer', 'pade', 'pi', 'piecewise', 'pkgload', 'place', 'poly', 'poly1d', 'polyadd', 'polyder', 'polydiv',
        'polyfit', 'polyint', 'polymul', 'polysub', 'polyval', 'power', 'prod', 'product', 'ptp', 'put', 'putmask', 'r_', 'rand', 'randn', 'random', 'rank', 'ravel',
        'real', 'real_if_close', 'rec', 'recarray', 'reciprocal', 'record', 'remainder', 'repeat', 'require', 'reshape', 'resize', 'restoredot', 'right_shift', 'rint',
        'roll', 'rollaxis', 'roots', 'rot90', 'round_', 'row_stack', 's_', 'sctype2char', 'sctypeDict', 'sctypeNA', 'sctypes', 'searchsorted', 'select',
        'set_numeric_ops', 'set_printoptions', 'set_string_function', 'setbufsize', 'setdiff1d', 'seterr', 'seterrcall', 'seterrobj', 'setmember1d', 'setxor1d',
        'shape', 'short', 'show_config', 'show_numpy_config', 'sign', 'signal', 'signbit', 'signedinteger', 'sin', 'sinc', 'single', 'sinh', 'size', 'sometrue',
        'sort', 'sort_complex', 'source', 'sparse', 'special', 'split', 'sqrt', 'square', 'squeeze', 'stats', 'std', 'str_', 'string0', 'string_', 'subtract', 'sum',
        'swapaxes', 'take', 'tan', 'tanh', 'tensordot', 'test', 'tile', 'trace', 'transpose', 'trapz', 'tri', 'tril', 'trim_zeros', 'triu', 'true_divide', 'typeDict', 'typeNA',
        'typecodes', 'typename', 'ubyte', 'ufunc', 'uint', 'uint0', 'uint16', 'uint32', 'uint64', 'uint8', 'uintc', 'uintp', 'ulonglong', 'unicode0', 'unicode_',
        'union1d', 'unique', 'unique1d', 'unravel_index', 'unsignedinteger', 'unwrap', 'ushort', 'vander', 'var', 'vdot', 'vectorize', 'version', 'void', 'void0',
        'vsplit', 'vstack', 'where', 'who', 'zeros', 'zeros_like']
```

SciPy Modules

- SciPy packages
  - SciPy is composed of several modules under the scipy namespace
- Tools
  - Cluster : vector quantization / kmeans
  - Fftpack : discrete fourier transform algorithms
  - Integrate : integration routines
  - Interpolate : interpolation tools
  - Linalg : linear algebra routines
  - Misc : various utilities (including Python Imaging Library)
  - Ndimage : n-dimensional image tools
  - Optimize : optimization tools
  - Signal : signal processing tools
  - Sparse : sparse matrices
  - Stats : statistical functions

SciPy Modules

- Other packages
  - Io : data input and output
  - Lib : wrappers to external libraries (BLAS, LAPACK)
  - Sandbox : incomplete, poorly-tested, or experimental code
  - Special : definitions of many usual math functions
  - Weave : C/C++ integration

Documentation

- Command ‘help’ (cont’d)

```
....
Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose descriptions contain the word "spam".

help> regex

Warning (from warnings module):
  File "C:\Program Files\Python24\lib\pydoc.py", line 262
    module = __import__(path)
DeprecationWarning: the regex module is deprecated; please use the re module
Help on built-in module regex:

NAME
    regex
........
```

- Can interfere with the terminal i/o where you are running the session
- Help system also available under the command scipy.info

Documentation

- Python provides facility for documentation strings

```
>>> info(scipy)
SciPy --- A scientific computing package for Python
================================================
You can support the development of SciPy by purchasing documentation at

  http://www.trelgol.com

It is being distributed for a fee for a limited time to try and raise
money for development.

Documentation is also available in the docstrings.

Contents
--------
  numpy name space
.....
```

Documentation

- Function or classes defined for a module may passed to ‘info’

```
>>> info(scipy.optimize)
Optimization Tools
==================

 A collection of general-purpose optimization routines.

   fmin        --  Nelder-Mead Simplex algorithm
                   (uses only function calls)
   fmin_powell --  Powell's (modified) level set method (uses only
                   function calls)
   fmin_cg     --  Non-linear (Polak-Ribiere) conjugate gradient algorithm
                   (can use function and gradient).
   fmin_bfgs   --  Quasi-Newton method (Broydon-Fletcher-Goldfarb-Shanno);
                   (can use function and gradient)
   fmin_ncg    --  Line-search Newton Conjugate Gradient (can use
                   function, gradient and Hessian).
   leastsq     --  Minimize the sum of squares of M equations in
                   N unknowns given a starting estimate.
......
```

Documentation

- ‘source’ : prints out listing of Python source code

```
>>> source(sqrt)
In file: C:\Program Files\Python24\Lib\site-packages\numpy\lib\scimath.py

def sqrt(x):
    x = _fix_real_lt_zero(x)
    return nx.sqrt(x)

>>>
```

- ‘dir’ can be used to look at the namespace of a package

```
>>> dir(sqrt)
['__call__', '__class__', '__delattr__', '__dict__', '__doc__', '__get__',
     '__getattribute__', '__hash__', '__init__', '__module__', '__name__', '__new__',
     '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__str__',
     'func_closure', 'func_code', 'func_defaults', 'func_dict', 'func_doc',
     'func_globals', 'func_name']
>>>
```

Base functions :: Interact with *NumPy*

- No need to import Numeric module
  - SciPy has subsumed Numeric’s functions
- Universal functions (addition, subtraction, etc.) have been altered
  - Floating point exceptions not raised
  - NaN and Inf returned
  - New universal functions
    - isnan()
    - isfinite()
    - isinf()
- Comparison and logical operation of complex numbers
  - Only on the real part
- Some basic functions modified to return complex numbers
  - For example: log(), sqrt(), inverse trig()
  - sqrt(-1) returns ‘1j’

Loading SciPy

- Load the SciPY module using the following command :

```
>> import scipy
>> scipy.info(scipy)
   # Display info about the scipy package
```

- The following command will import all SciPy functions :

```
>> from scipy import *
```

Help on SciPy

- To get information about the SciPy module run

```
>> scipy.info(scipy)
```

- For detailed help on the package :

```
>> help(scipy)
```

Some Important Available Packages

- stats – Statistical Functions
- signal – Signal Processing Tools
- linalg – Linear Algebra Tools
- linsolve – Linear Solvers
- sparse – Sparse Matrix
- fftpack – Discrete Fourier Transform algorithms
- ndimage – n-dimensional image package
- io – Data input and output
- integrate – Integration routines
- interpolate – Interpolation tools

For a comprehensive list run `help(‘scipy’)`

Variables and Basic Functions

Creating Arrays

- *arrary() :*

```
>>> a = array([0,1,2,3])  # Single dimensional array
>>> b = array([[1,2,3],[4,5,6]]) # Multidimensional array
```

- *arange() :*

```
>> c = arange(4) # Creates array of length 4 (from 0 to 3)
```

- *linspace() :*

```
>>> d = linspace(0,9,5) # Returns array of length 5, with
   contents from 0 to 9
>>> r_[0:10] # Shorthand for linspace
array([0,1,2,3,4,5,6,7,8,9])
>>> r_[0:9:10j]
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

- *ones() & zeros() :*

```
>>> a1 = ones(4) # Create 1 x 4 dimensional array of ones
>>> a2 = ones((4,3)) # Create 4 x 3 dimensional array
# Same syntax for ‘zeros’
```

- *identity() :*

```
>>> idt = identity(4) # Create n x n identity matrix
```

Creating grids

- *mgrid()*

```
>>>x,y = mgrid[0:3,4:7]
>>>x
array([[0,0,0],
         [1,1,1],
         [2,2,2]])
>>>y
array([[4,5,6],
         [4,5,6],
         [4,5,6]])
>>>x+y
array([[4,5,6],
        [5,6,7],
        [6,7,8]])
```

- *ogrid()*

```
>>>x,y = ogrid[0:3,4:7]
>>>x
array([[0],
        [1],
        [2]])
>>>y
array([[4,5,6]])


>>>x+y
array([[4,5,6],
        [5,6,7],
        [6,7,8]])
```

Creating Matrices

```
>>> A = mat(‘1,2;3,4’)
>>> print A
matrix([[1,2],
         [3,4]])
>>> 2*A  # Scalar multiply
matrix([[2,4],
         [6,8]])
>>> A.T  # Transpose
matrix([[1,3],
         [2,4]])
>>> A*A  # Matrix multiply
matrix([[7,10],
         [15,22]])
```

```
>>> A**2 # Matrix power
matrix([[7,10],
         [15,22]])
>>> B = 5*diag([1.,3,5])
# Diagonal array
>>> B
array([[5., 0., 0.],
        [0., 15., 0.],
        [0., 0., 25.]])
>>> mat(B) # To matrix
>>> mat(B).I # Matrix inverse
matrix([[0.2, -0.,      -0.],
        [0., 0.06666667, -0.],
        [0., 0.,       0.04]])
```

Basic Functions

- Math Functions –
  - sin, sinh, cos, cosh, tan, tanh, exp
  - log, log10
  - floor, ceil, fmod
  - bitwise_and, bitwise_or, bitwise_xor, invert
  - logical_and, logical_not, logical_or, logical_xor
  - conjugate

** A complete list can be found at the end of the notes*

Basic Functions

- Data/Object Type Handling
  - iscomplex, iscomplexobj, isreal, isrealobj
  - real, imag
  - isnan
  - type

```
>>> a=array([1+1j, 2, 5j])

>>> iscomplex(a)
array([True, False, True], dtype=bool)

>>> iscomplexobj(a)
True

>>> isreal(a)
array([False, True, False], dtype=bool)

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

Basic Functions

- Data/Object Shape Manipulation
  - squeeze
  - vstack
  - hstack
  - dstack
  - concatenate

```
>>> a=array([1,2,3])
>>> b=array([4,5,6])
>>>
>>> vstack((a,b))
array([[1, 2, 3],
       [4, 5, 6]])
>>> hstack((a,b))
array([1, 2, 3, 4, 5, 6])
>>>
>>> dstack((a,b))
array([[[1, 4],
        [2, 5],
        [3, 6]]]
```

Basic Functions

- Other useful functions
  - sum, cumsum, prod, cumprod
  - amax, amin
  - comb, factorial
  - any, all
  - fliplr, flipud
  - diag, eye
  - where
  - Who

** A more comprehensive list of functions can be found at the end of the notes*

```
>>> a=array([1+1j, 2, 5j])
>>> x,y = mgrid[0:41,0:31]
>>> who
<function who at 0x01327CF0>
>>> who()
Name          Shape          Bytes          Type
======================================


a             3              48             complex128
x             41 x 31        5084           int32
y             41 x 31        5084           int32

Upper bound on total bytes  =      10216
```

Lab 1

1. Write a function that generates a sine wave. The input arguments to this function are the frequency of the sine wave (in Hz), sampling frequency and the sampling duration. Use the equation : **`y = sin(ωt)`**, where **`ω=2*pi*f`**
2. The function should return a tuple which consists of the sine wave and the time steps at which the data was calculated. (In the above example, the function should return *y* and *t*)

Plotting and Graphics

- *matplotlib* used for plotting and graphics
- Can generate plots, histograms, power spectra, bar graphs, etc
- Website : http://matplotlib.sourceforge.net/
- Other graphing/plotting libraries available

1. gplt
2. xplt
3. wxpyplot

Basic Functions

- Plotting functions –
  - plot, stem
  - subplot
  - semilogx, semilogy, loglog
  - contour, contourf
  - polar
  - specgram, pie, hist
  - imshow
  - show
- Additional functions –
  - text, xlabel, ylabel, title, legend
  - hold, colormap
  - load

Loading `matplotlib`

- From python

```
>>> import pylab
```

OR

```
>>> from pylab import *
```

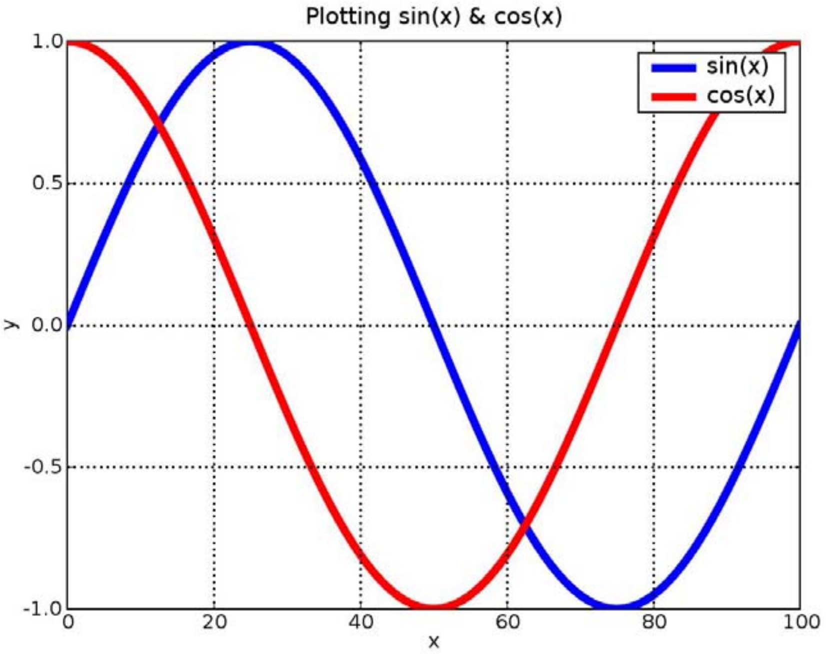
- A comprehensive listing can be found on the website :

  http://matplotlib.sourceforge.net/

2D Plotting

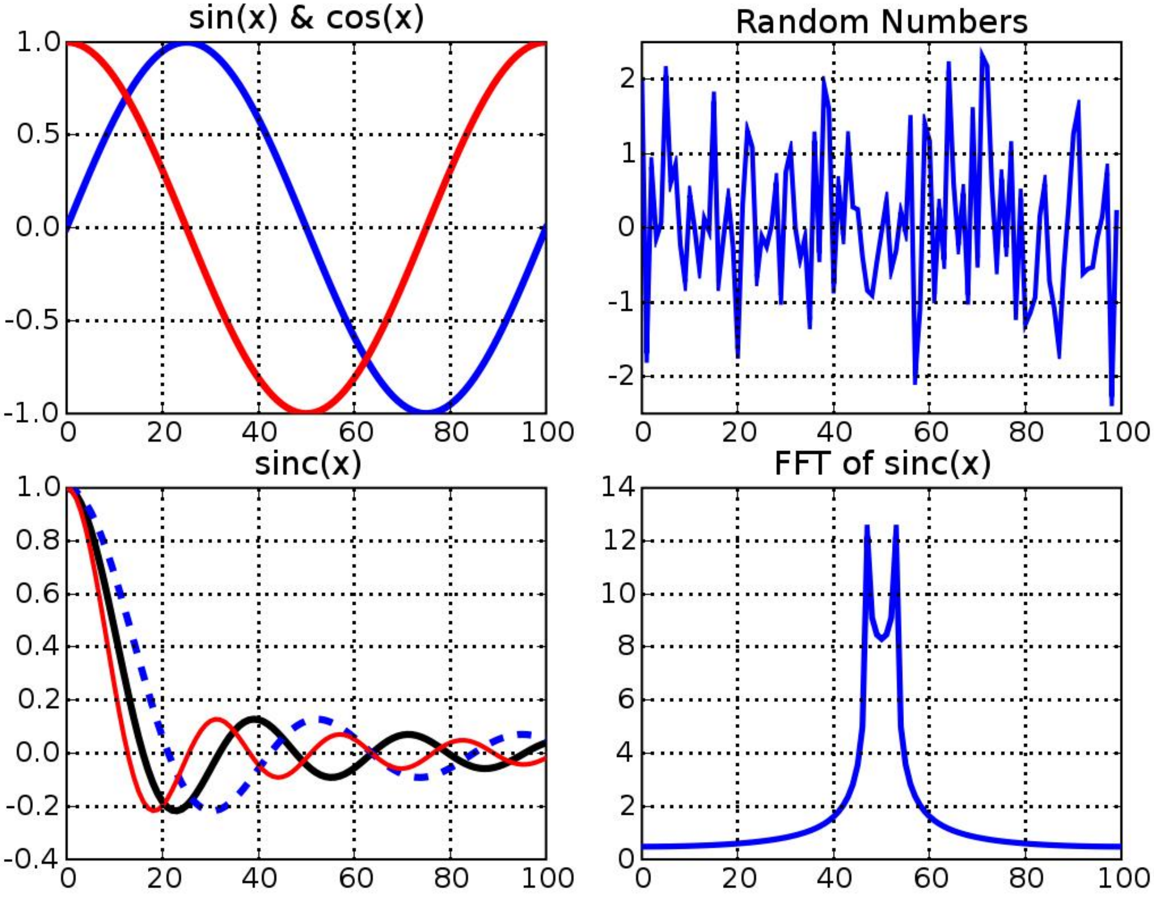
- File : example1.py

```
>>> from scipy import *
>>> from pylab import *
>>> x = r_[0:101]
>>> y01 = sin(2*pi*x/100)
>>> y02 = cos(2*pi*x/100)
>>> plot(x,y01,linewidth=5.0)
>>> hold(True)
>>> plot(x,y02,linewidth=5.0)
>>> xlabel(‘x’);ylabel(‘y)
>>> title(‘Plotting sin(x) & cos(x)’);
>>> legend((‘sin(x)’,’cos(x)’));
>>> grid(True)
```



2D Plotting : Examples

File : Example 2 (source on the next slide)



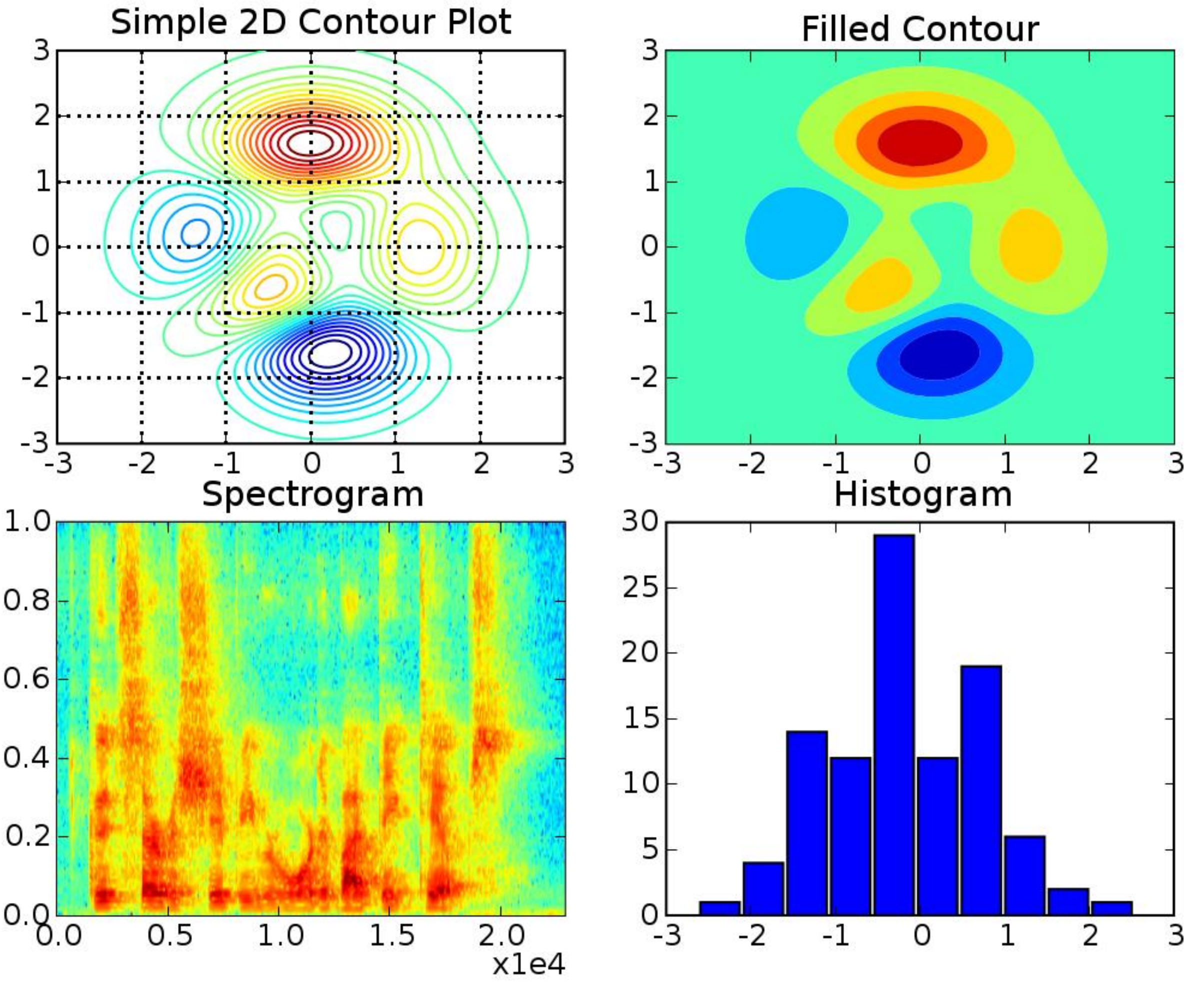
2D Plotting : Examples

- **File : Example 2**

```
# Example 2
# Multiple subplots in one figure

from pylab import *
from scipy import *
from scipy.fftpack import fftshift

x=r_[0:101]
y01=sin(2*pi*x/100)
y02=cos(2*pi*x/100)

y03 = randn(100);
y04 = sinc(2*pi*x/100);
Y04 = abs(fftshift(fft(y04)))
y05 = sinc(1.5*pi*x/100);
y06 = sinc(2.5*pi*x/100);
Y06 = abs(fftshift(fft(y06)))

subplot(2,2,1);plot(x,y01,linewidth=3);hold(True);
plot(x,y02,'r',linewidth=3)
grid(True);title('sin(x) & cos(x)');
subplot(2,2,2);plot(y03,linewidth=2);grid(True);title('Random Numbers');
subplot(2,2,3);plot(x,y04,'k',linewidth=3);grid(True);title('sinc(x)');
hold(True);
subplot(2,2,3);plot(x,y05,'--',linewidth=3);
subplot(2,2,3);plot(x,y06,'r',linewidth=2);
subplot(2,2,4);plot(Y04,linewidth=2.5);grid(True);title('FFT of sinc(x)');

show()
```

2D Plotting : Examples

File : Example 3 (next slide)



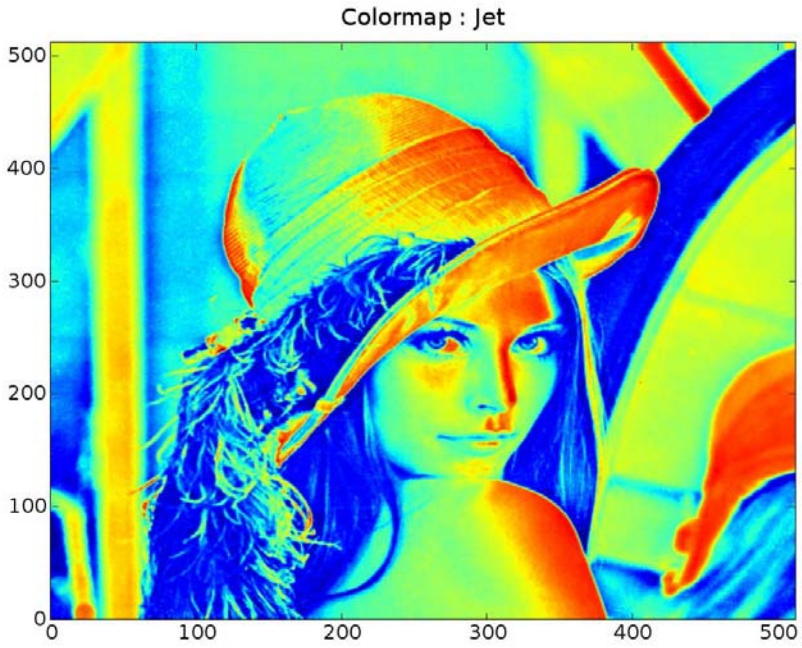
2D Plotting : Examples

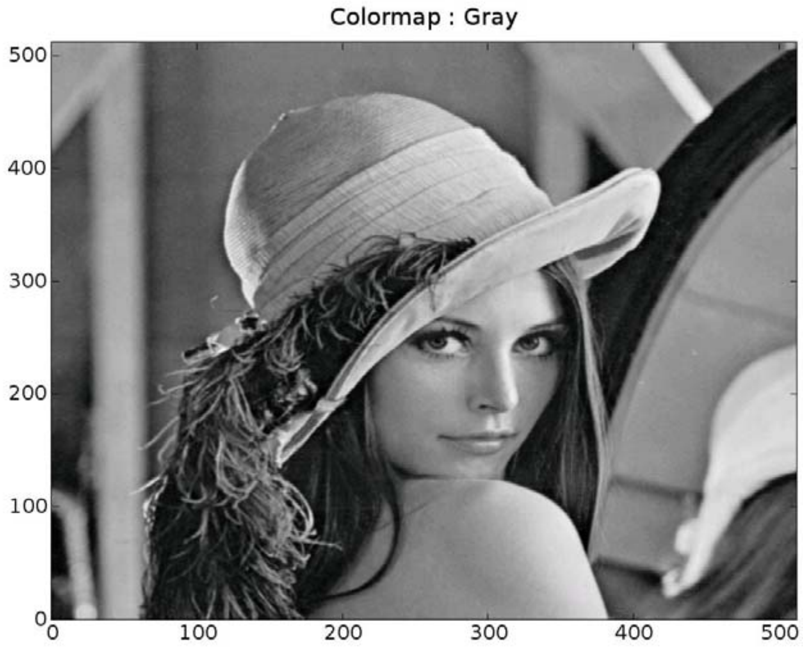
- **File : Example 3**

```
from scipy import *
from scipy.fftpack import fftshift
from pylab import *

x,y = meshgrid(r_[-3:3:100j], r_[-3:3:100j]);

z = 3*(1-x)**2*exp(-x**2-(y+1)**2) - 10*(x/5-x**3-y**5)*exp(-x**2-y**2) -
(1/3)*exp(-(x+1)**2-y**2);

wav = io.read_array('wavdata');
x1 = r_[1:wav.size+1]

subplot(2,2,1);contour(x,y,z,25);grid(True);title('Simple 2D Contour Plot');
subplot(2,2,2);contourf(x,y,z);title('Filled Contour');
subplot(2,2,3);h=specgram(wav);title('Spectrogram');
subplot(2,2,4);hist(randn(100));title('Histogram');

show();
```

2D Plotting : Images

```
>>> imshow(lena())
```





```
>>> gray()
```

Lab 2

1. Plot two (or more) sine waves of different frequencies on a single plot. Use the function you wrote in Lab 1 to generate the wave to be plotted
2. Generate two (or more) sine waves, add them together and display the spectrogram of the resulting signal
3. Use the “load” function in matplotlib to load an image from the “images” directory and display it

Linear Algebra : scipy.linalg

- Provides linear algebra routines
- Basic functions
  - inv, solve, det, norm, lstsq, pinv
- Eigenvalue decomposition
  - eig, eigvals, lu, svd, cholesky
- Matrix functions
  - sinm, cosm, tanm, expm

Linear Algebra Examples

```
>>> from scipy import *
>>> from scipy import linalg
>>> a=mat('1 1 1 ;1 -2 2; 0 1 -2')
>>> b=mat(‘0;4;2’)
>>> x = linalg.solve(a,b)

>>> linalg.inv(a)

>>> evals, vec = linalg.eig(a)
```

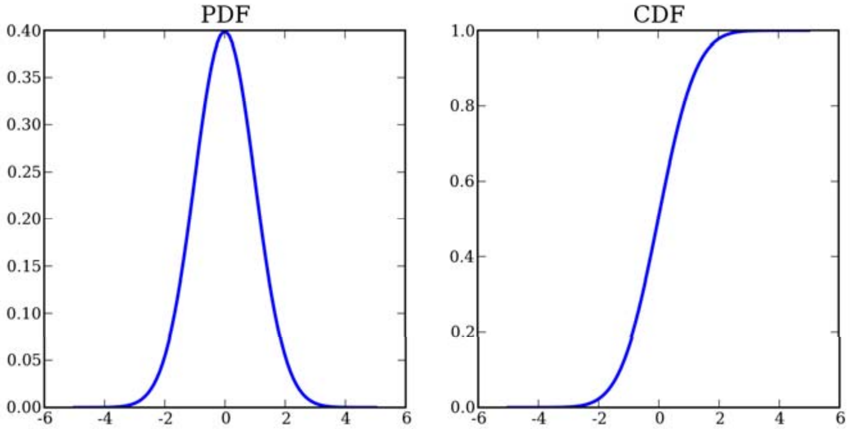
Project Step 2

- Use project step 1 to read in all the training images. Form the covariance matrix and then get the eigenvectors.

Statistics : scipy.stats

- Over 80 continuous distributions
- 10 Standard discrete distributions
- Mehtods
  - pdf, cdf, rvs, ppf, stats
- Statistical functions such as
  - mean, std, var, moment, skew, kurtosis

```
>>> from scipy import *
>>> from scipy import linalg
>>> from pylab import *
>>> x = r_[-5:5:100j]
>>> y = stats.norm.pdf(x)
>>> z = stats.norm.cdf(x)
>>> plot(x,y); hold(True);plot(x,z)
```



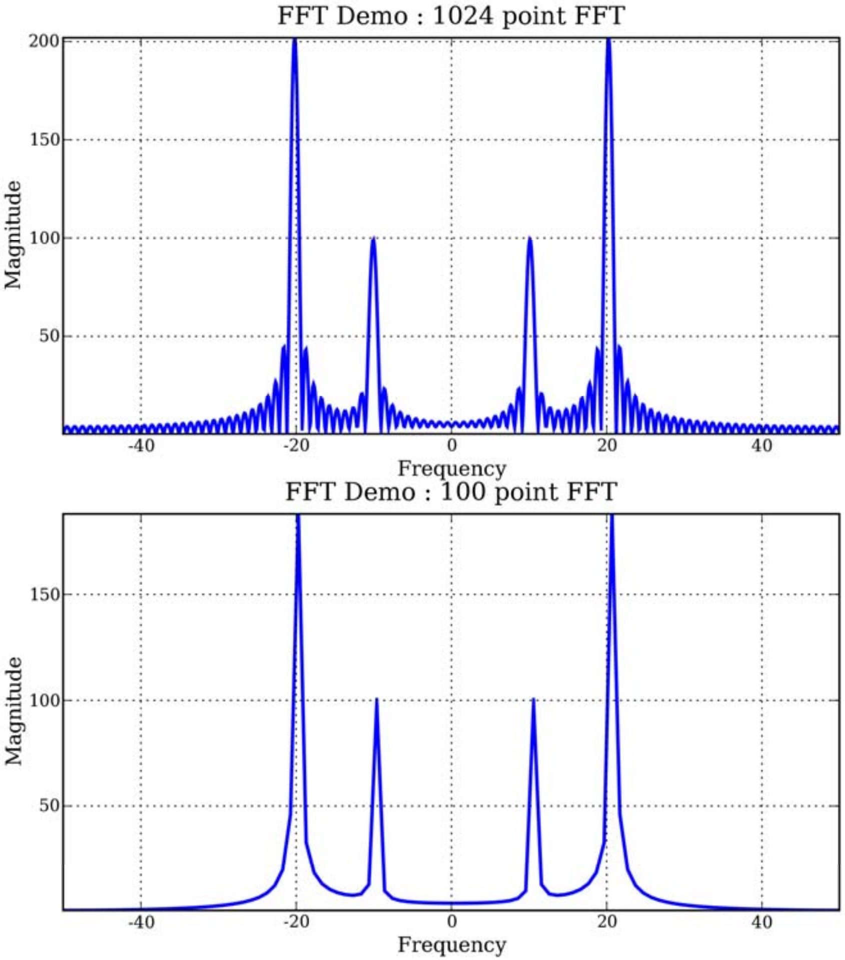
Signal Processing : scipy.fftpack, scipy.signal

- Discrete Fourier Transform Algorithms
  - fft, ifft, fft2, ifft2, fftn, ifftn
  - fftshift, ifftshift, fftfreq
- Signal Processing Tools
  - Convolution
  - B-Splines
  - Filtering & Filter Design
  - Linear Systems
  - Window Functions
  - Wavelets

scipy.fftpack : FFT Example

- FFT Example

```
>>> from scipy import *
>>> from scipy.fftpack import
   fftshift, fftfreq
>>> from pylab import *
>>> x = r_[0:1:100j]
>>> y = 2*sin(2*pi*10*x) +
   3*cos(2*pi*20*x)
>>> Y01 = fft(y)
>>> Y02 = fft(y, 1024)
>>> w = fftfreq(100)
>>> Y01 = fftshift(Y01)
>>> w = fftshift(w)
>>> plot(w, abs(Y01))
```



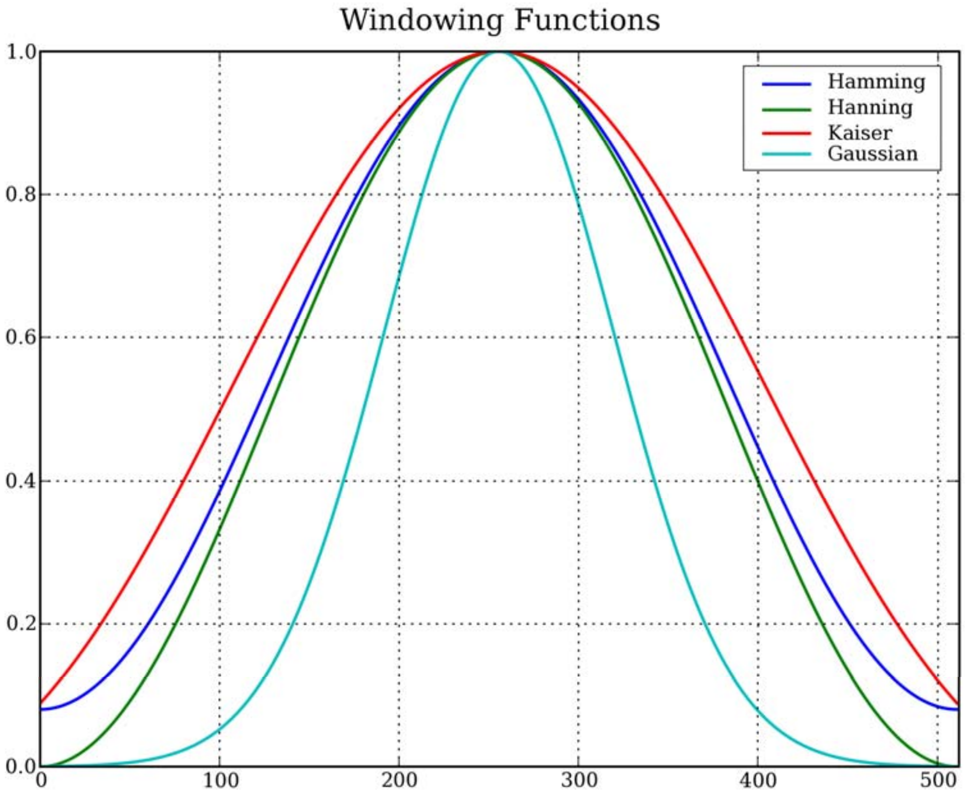
scipy.signal : Convolution

- Convolution
  - convolve : N-dimensional convolution
  - correlate : N-dimensional correlation
  - fftconvolve : convolution using FFT

```
>>> from scipy import *
>>> from scipy import signal
>>> x = r_[0:1:100j
>>> y01 = hamming(32)
>>> y02 = hanning(32)
>>> z01 = convolve(y01, y02)
>>> z02 = signal.convolve(y01, y02)
```

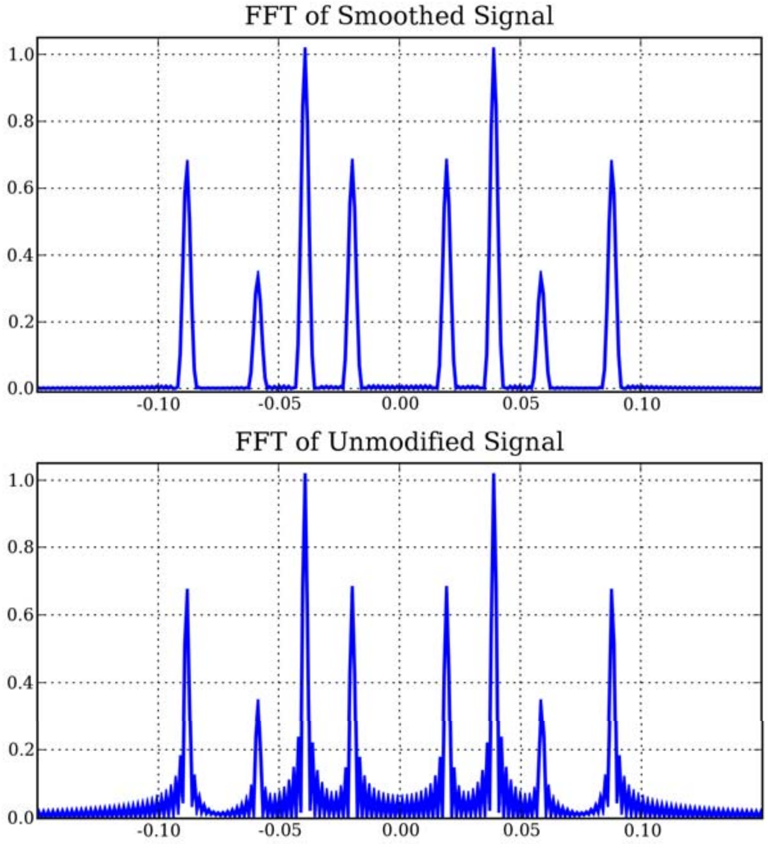
scipy.signal : Windowing Functions

- Functions to generate the following types of windows
  - boxcar, triang, blackman, hamming, kaiser, gaussian



scipy.signal : Effect of Windowing

```
>>> x = r_[0:1:512j]; h01 = hamming(512)
>>> theta = 2*pi*x; F=fftfreq(1024)
>>> y01 = 2*sin(10*theta) + 3*cos(20*theta) +
    sin(30*theta) + 2*cos(45*theta);
>>> ys = y01*h01;
>>> Ys = fft(ys, 1024); Y01 = fft(y01, 1024)
>>> Ysn = fftshift(abs(Ys)/max(abs(Ys)))
>>> Y01n = fftshift(abs(Y01)/max(abs(Y01)))
>>> subplot(2,1,1);plot(F,Ysn,
    linewidth=2);title('FFT of Smoothed
    Signal');axis([400, 620, -0.01, 1.01])
>>>
    subplot(2,1,2);plot(F,Y01n,linewidth=2);ti
    tle('FFT of Unmodified Signal');axis([400,
    620, -0.001, 1.01])
```

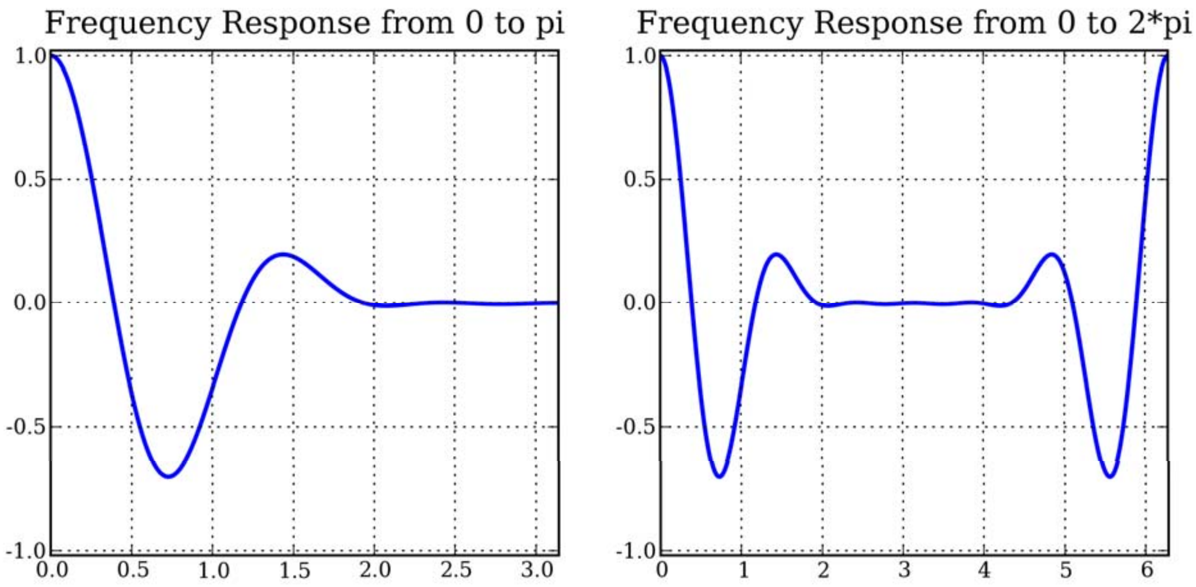


scipy.signal : Filtering & Filter Design

- Filter design
  - remez, firwin, iirdesign, iirfilter
- MATLAB – style IIR filter design
  - butter, cheby1, cheby2, ellip, bessel
- Filtering
  - lfilter, medfilt, medfilt2, wiener
- Utility functions
  - freqz : Digital filter frequency response

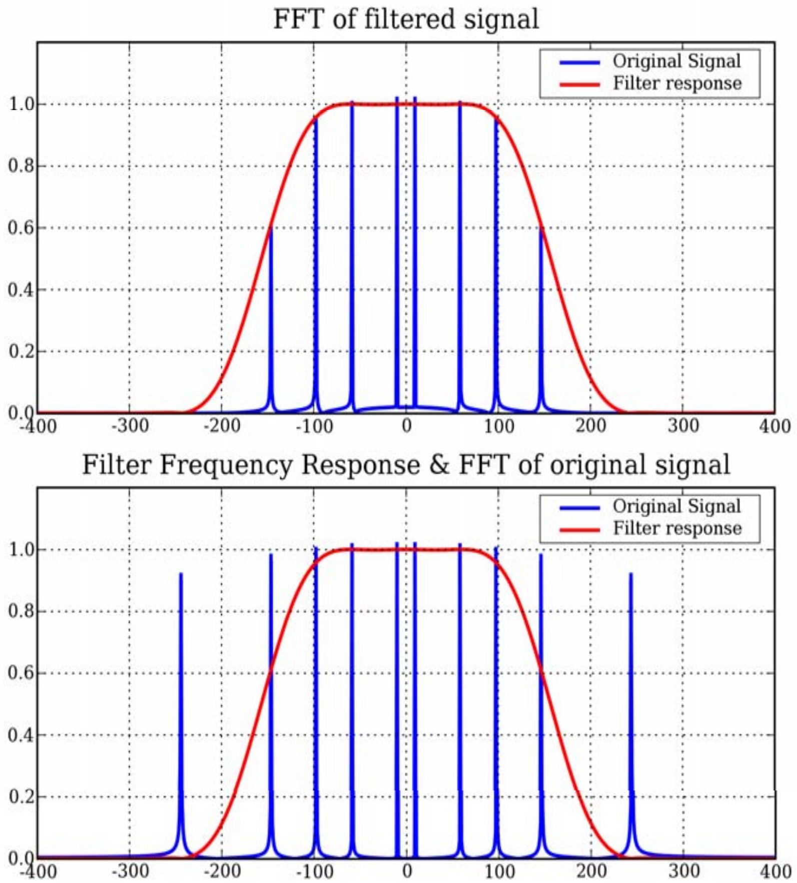
scipy.signal : Filter Design Example

```
>>> fc = 100; wc = fc*pi/fsampl; order = 20
>>> b01 = signal.firwin(order, wc)
>>> w01, h01 = signal.freqz(b01,1, worN=N)
>>> w02, h02 = signal.freqz(b01, 1, worN=N, whole=1)
>>> subplot(1,2,1);plot(w01,h01)
>>> subplot(1,2,2);plot(w02, h02)
```



scipy.signal : Filter Design Example

```
>>> t = r_[0:1:1024j]
>>> fsampl = 1e3; N = 1024
>>> F = fftshift(fsampl*fftfreq(N))
>>> y = sin(2*pi*10*t) +
   sin(2*pi*60*t) + sin(2*pi*100*t)
   + sin(2*pi*150*t) +
   sin(2*pi*250*t)
>>> fc = 100; wc = fc*pi/fsampl;
   order = 20
>>> b01 = signal.firwin(order, wc)
>>> w01, h01 = signal.freqz(b01,1,
   worN=N)
>>> y01 = signal.lfilter(b01, 1, y)
```



scipy.signal : Filter Design Example

- Remez Filter

```
>>>
```

- IIR Filter

```
>>> b,a=signal.iirfilter(8, [0.5, 0.6], rp=0.1,
   rs=0.01, output='ba')
```

- IIR Design

```
>>> b,a=signal.iirdesign(8
```

Lab

- Use the file ‘wavdata2’ as the input data. Design a band pass filter which will filter out the frequency range ----
- Design a band stop filter with the passband ----
- In each case, change the filter order. What effect does the filter order have on the output ?

Basic Functions

TYPE

iscomplexobj	real_if_close	isnan
iscomplex	isscalar	nan_to_num
isrealobj	isneginf	common_type
isreal	isposinf	cast
imag	isinf	type
real	isfinite	

SHAPE

squeeze	vstack	split
atleast_1d	hstack	hsplit
atleast_2d	column_stack	vsplit
atleast_3d	dstack	dsplit
apply_over_axes	expand_dims	apply_along_axis

OTHER

select	unwrap	roots
extract	sort_complex	poly
insert	trim_zeros	any
fix	fliplr	all
mod	flipud	disp
amax	rot90	unique
amin	eye	extract
ptp	diag	insert
sum	factorial	nansum
cumsum	factorial2	nanmax
prod	comb	nanargmax
cumprod	pade	nanargmin
diff	derivative	nanmin
angle	limits.XXXX	

- absolute
- add
- arccos
- arccosh
- arcsin
- arcsinh
- arctan
- arctan2
- arctanh
- around
- bitwise_and
- bitwise_or
- bitwise_xor
- ceil
- conjugate
- cos
- cosh
- divide
- divide_safe
- equal
- exp

- fabs
- floor
- fmod
- greater
- greater_equal
- hypot
- invert
- left_shift
- less
- less_equal
- log
- log10
- logical_and
- logical_not
- logical_or
- logical_xor
- maximum
- minimum
- multiply
- negative
- not_equal

- power
- right_shift
- sign
- sin
- sinh
- sqrt
- subtract
- tan
- tanh