

Lecture-3-Scipy

June 19, 2020

1 SciPy - Library of scientific algorithms for Python

J.R. Johansson (jrjohansson at gmail.com)

The latest version of this [IPython notebook](http://github.com/jrjohansson/scientific-python-lectures) lecture is available at <http://github.com/jrjohansson/scientific-python-lectures>.

The other notebooks in this lecture series are indexed at <http://jrjohansson.github.io>.

```
[1]: # what is this line all about? Answer in lecture 4
      %matplotlib inline
      import matplotlib.pyplot as plt
      from IPython.display import Image
```

1.1 Introduction

The SciPy framework builds on top of the low-level NumPy framework for multidimensional arrays, and provides a large number of higher-level scientific algorithms. Some of the topics that SciPy covers are:

- Special functions ([scipy.special](#))
- Integration ([scipy.integrate](#))
- Optimization ([scipy.optimize](#))
- Interpolation ([scipy.interpolate](#))
- Fourier Transforms ([scipy.fftpack](#))
- Signal Processing ([scipy.signal](#))
- Linear Algebra ([scipy.linalg](#))
- Sparse Eigenvalue Problems ([scipy.sparse](#))
- Statistics ([scipy.stats](#))
- Multi-dimensional image processing ([scipy.ndimage](#))
- File IO ([scipy.io](#))

Each of these submodules provides a number of functions and classes that can be used to solve problems in their respective topics.

In this lecture we will look at how to use some of these subpackages.

To access the SciPy package in a Python program, we start by importing everything from the `scipy` module.

```
[2]: from scipy import *
```

If we only need to use part of the SciPy framework we can selectively include only those modules we are interested in. For example, to include the linear algebra package under the name `la`, we can do:

```
[3]: import scipy.linalg as la
```

1.2 Special functions

A large number of mathematical special functions are important for many computational physics problems. SciPy provides implementations of a very extensive set of special functions. For details, see the list of functions in the reference documentation at <http://docs.scipy.org/doc/scipy/reference/special.html#module-sciPy.special>.

To demonstrate the typical usage of special functions we will look in more detail at the Bessel functions:

```
[4]: #
# The scipy.special module includes a large number of Bessel-functions
# Here we will use the functions jn and yn, which are the Bessel functions
# of the first and second kind and real-valued order. We also include the
# function jn_zeros and yn_zeros that gives the zeroes of the functions jn
# and yn.
#
from scipy.special import jn, yn, jn_zeros, yn_zeros
```

```
[5]: n = 0      # order
x = 0.0

# Bessel function of first kind
print "J_%d(%f) = %f" % (n, x, jn(n, x))

x = 1.0
# Bessel function of second kind
print "Y_%d(%f) = %f" % (n, x, yn(n, x))
```

```
File "<ipython-input-5-4a9e29886bb1>", line 5
print "J_%d(%f) = %f" % (n, x, jn(n, x))
      ^
```

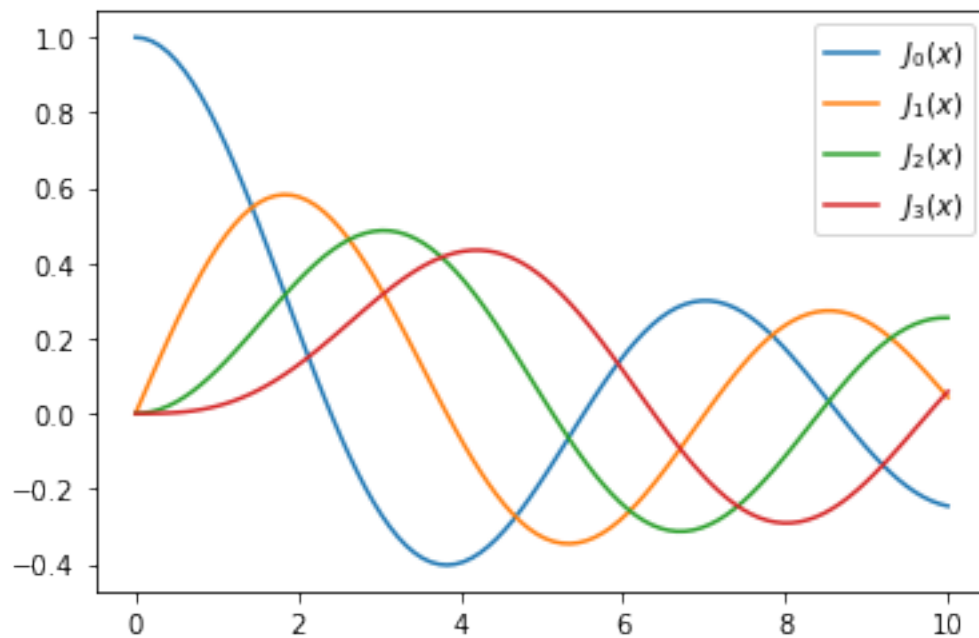
SyntaxError: invalid syntax

```
[6]: x = linspace(0, 10, 100)

fig, ax = plt.subplots()
```

```
for n in range(4):
    ax.plot(x, jn(n, x), label=r"$J_{%d}(x)$" % n)
ax.legend();
```

<ipython-input-6-2df7696e85df>:1: DeprecationWarning: scipy.linspace is deprecated and will be removed in SciPy 2.0.0, use numpy.linspace instead
x = linspace(0, 10, 100)



```
[7]: # zeros of Bessel functions
n = 0 # order
m = 4 # number of roots to compute
jn_zeros(n, m)
```

```
[7]: array([ 2.40482556,  5.52007811,  8.65372791, 11.79153444])
```

1.3 Integration

1.3.1 Numerical integration: quadrature

Numerical evaluation of a function of the type

$$\int_a^b f(x) dx$$

is called *numerical quadrature*, or simply *quadrature*. SciPy provides a series of functions for different kind of quadrature, for example the `quad`, `dblquad` and `tplquad` for single, double and triple integrals, respectively.

```
[8]: from scipy.integrate import quad, dblquad, tplquad
```

The quad function takes a large number of optional arguments, which can be used to fine-tune the behaviour of the function (try `help(quad)` for details).

The basic usage is as follows:

```
[9]: # define a simple function for the integrand
def f(x):
    return x
```

```
[10]: x_lower = 0 # the lower limit of x
x_upper = 1 # the upper limit of x

val, abserr = quad(f, x_lower, x_upper)

print "integral value =", val, ", absolute error =", abserr
```

```
File "<ipython-input-10-ce8a12fe5a6f>", line 6
print "integral value =", val, ", absolute error =", abserr
    ^
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean
↳ print("integral value =", val, ", absolute error =", abserr)?
```

If we need to pass extra arguments to integrand function we can use the `args` keyword argument:

```
[11]: def integrand(x, n):
    """
    Bessel function of first kind and order n.
    """
    return jn(n, x)

x_lower = 0 # the lower limit of x
x_upper = 10 # the upper limit of x

val, abserr = quad(integrand, x_lower, x_upper, args=(3,))

print val, abserr
```

```
File "<ipython-input-11-ec663057c1c9>", line 13
print val, abserr
    ^
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(val,
↳ abserr)?
```

For simple functions we can use a lambda function (name-less function) instead of explicitly defining a function for the integrand:

```
[12]: val, abserr = quad(lambda x: exp(-x ** 2), -Inf, Inf)

print "numerical =", val, abserr

analytical = sqrt(pi)
print "analytical =", analytical
```

```
File "<ipython-input-12-701c84ebcf41>", line 3
print "numerical =", val, abserr
    ^
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean
↳ print("numerical =", val, abserr)?
```

As show in the example above, we can also use 'Inf' or '-Inf' as integral limits.

Higher-dimensional integration works in the same way:

```
[13]: def integrand(x, y):
        return exp(-x**2-y**2)

x_lower = 0
x_upper = 10
y_lower = 0
y_upper = 10

val, abserr = dblquad(integrand, x_lower, x_upper, lambda x : y_lower, lambda x:
↳ y_upper)

print val, abserr
```

```
File "<ipython-input-13-0aba04f9ec4f>", line 11
print val, abserr
    ^
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(val,
↳ abserr)?
```

Note how we had to pass lambda functions for the limits for the y integration, since these in general can be functions of x.

1.4 Ordinary differential equations (ODEs)

SciPy provides two different ways to solve ODEs: An API based on the function `odeint`, and object-oriented API based on the class `ode`. Usually `odeint` is easier to get started with, but the `ode` class offers some finer level of control.

Here we will use the `odeint` functions. For more information about the class `ode`, try `help(ode)`. It does pretty much the same thing as `odeint`, but in an object-oriented fashion.

To use `odeint`, first import it from the `scipy.integrate` module

```
[14]: from scipy.integrate import odeint, ode
```

A system of ODEs are usually formulated on standard form before it is attacked numerically. The standard form is:

$$y' = f(y, t)$$

where

$$y = [y_1(t), y_2(t), \dots, y_n(t)]$$

and f is some function that gives the derivatives of the function $y_i(t)$. To solve an ODE we need to know the function f and an initial condition, $y(0)$.

Note that higher-order ODEs can always be written in this form by introducing new variables for the intermediate derivatives.

Once we have defined the Python function f and array y_0 (that is f and $y(0)$ in the mathematical formulation), we can use the `odeint` function as:

$$y_t = \text{odeint}(f, y_0, t)$$

where t is an array with time-coordinates for which to solve the ODE problem. y_t is an array with one row for each point in time in t , where each column corresponds to a solution $y_i(t)$ at that point in time.

We will see how we can implement f and y_0 in Python code in the examples below.

Example: double pendulum Let's consider a physical example: The double compound pendulum, described in some detail here: http://en.wikipedia.org/wiki/Double_pendulum

```
[15]: Image(url='http://upload.wikimedia.org/wikipedia/commons/c/c9/
      ↪Double-compound-pendulum-dimensioned.svg')
```

```
[15]: <IPython.core.display.Image object>
```

The equations of motion of the pendulum are given on the wiki page:

$$\dot{\theta}_1 = \frac{6}{m\ell^2} \frac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)}$$

$$\dot{\theta}_2 = \frac{6}{m\ell^2} \frac{8p_{\theta_2} - 3\cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)}.$$

$$\dot{p}_{\theta_1} = -\frac{1}{2}m\ell^2 [\dot{\theta}_1\dot{\theta}_2 \sin(\theta_1 - \theta_2) + 3\frac{g}{\ell} \sin \theta_1]$$

$$\dot{p}_{\theta_2} = -\frac{1}{2}m\ell^2 [-\dot{\theta}_1\dot{\theta}_2 \sin(\theta_1 - \theta_2) + \frac{g}{\ell} \sin \theta_2]$$

To make the Python code simpler to follow, let's introduce new variable names and the vector notation: $x = [\theta_1, \theta_2, p_{\theta_1}, p_{\theta_2}]$

$$\dot{x}_1 = \frac{6}{m\ell^2} \frac{2x_3 - 3\cos(x_1 - x_2)x_4}{16 - 9\cos^2(x_1 - x_2)}$$

$$\dot{x}_2 = \frac{6}{m\ell^2} \frac{8x_4 - 3\cos(x_1 - x_2)x_3}{16 - 9\cos^2(x_1 - x_2)}$$

$$\dot{x}_3 = -\frac{1}{2}m\ell^2 [\dot{x}_1\dot{x}_2 \sin(x_1 - x_2) + 3\frac{g}{\ell} \sin x_1]$$

$$\dot{x}_4 = -\frac{1}{2}m\ell^2 [-\dot{x}_1\dot{x}_2 \sin(x_1 - x_2) + \frac{g}{\ell} \sin x_2]$$

```
[16]: g = 9.82
      L = 0.5
      m = 0.1

      def dx(x, t):
          """
          The right-hand side of the pendulum ODE
          """
          x1, x2, x3, x4 = x[0], x[1], x[2], x[3]

          dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * cos(x1-x2) * x4)/(16 - 9 * cos(x1-x2)**2)
          dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * cos(x1-x2) * x3)/(16 - 9 * cos(x1-x2)**2)
          dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * sin(x1-x2) + 3 * (g/L) * sin(x1))
          dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * sin(x1-x2) + (g/L) * sin(x2))

          return [dx1, dx2, dx3, dx4]
```

```
[17]: # choose an initial state
      x0 = [pi/4, pi/2, 0, 0]
```

```
[18]: # time coordinate to solve the ODE for: from 0 to 10 seconds
      t = linspace(0, 10, 250)
```

```
<ipython-input-18-02dd0324cced>:2: DeprecationWarning: scipy.linspace is
deprecated and will be removed in SciPy 2.0.0, use numpy.linspace instead
      t = linspace(0, 10, 250)
```

```
[19]: # solve the ODE problem
      x = odeint(dx, x0, t)
```

```
<ipython-input-16-82fff5157982>:11: DeprecationWarning: scipy.cos is deprecated
and will be removed in SciPy 2.0.0, use numpy.cos instead
```

```
      dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * cos(x1-x2) * x4)/(16 - 9 * cos(x1-x2)**2)
```

```
<ipython-input-16-82fff5157982>:12: DeprecationWarning: scipy.cos is deprecated
and will be removed in SciPy 2.0.0, use numpy.cos instead
```

```
      dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * cos(x1-x2) * x3)/(16 - 9 * cos(x1-x2)**2)
```

<ipython-input-16-82fff5157982>:13: DeprecationWarning: scipy.sin is deprecated and will be removed in SciPy 2.0.0, use numpy.sin instead

```
dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * sin(x1-x2) + 3 * (g/L) * sin(x1))
```

<ipython-input-16-82fff5157982>:14: DeprecationWarning: scipy.sin is deprecated and will be removed in SciPy 2.0.0, use numpy.sin instead

```
dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * sin(x1-x2) + (g/L) * sin(x2))
```

[20]: *# plot the angles as a function of time*

```
fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].plot(t, x[:, 0], 'r', label="theta1")
axes[0].plot(t, x[:, 1], 'b', label="theta2")
```

```
x1 = + L * sin(x[:, 0])
y1 = - L * cos(x[:, 0])
```

```
x2 = x1 + L * sin(x[:, 1])
y2 = y1 - L * cos(x[:, 1])
```

```
axes[1].plot(x1, y1, 'r', label="pendulum1")
axes[1].plot(x2, y2, 'b', label="pendulum2")
axes[1].set_ylim([-1, 0])
axes[1].set_xlim([1, -1]);
```

<ipython-input-20-c258a0156c14>:8: DeprecationWarning: scipy.sin is deprecated and will be removed in SciPy 2.0.0, use numpy.sin instead

```
x1 = + L * sin(x[:, 0])
```

<ipython-input-20-c258a0156c14>:9: DeprecationWarning: scipy.cos is deprecated and will be removed in SciPy 2.0.0, use numpy.cos instead

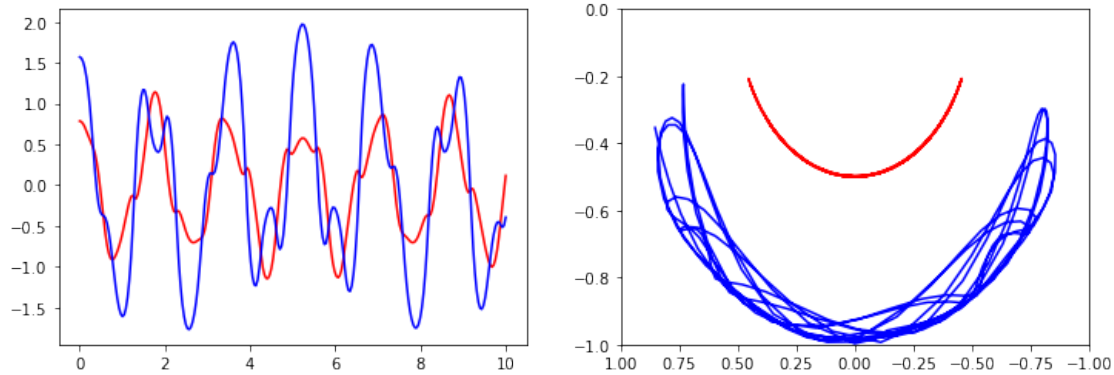
```
y1 = - L * cos(x[:, 0])
```

<ipython-input-20-c258a0156c14>:11: DeprecationWarning: scipy.sin is deprecated and will be removed in SciPy 2.0.0, use numpy.sin instead

```
x2 = x1 + L * sin(x[:, 1])
```

<ipython-input-20-c258a0156c14>:12: DeprecationWarning: scipy.cos is deprecated and will be removed in SciPy 2.0.0, use numpy.cos instead

```
y2 = y1 - L * cos(x[:, 1])
```

Simple animation of the pendulum motion. We will see how to make better animation in Lecture 4.

```
[21]: from IPython.display import display, clear_output
import time
```

```
[22]: fig, ax = plt.subplots(figsize=(4,4))

for t_idx, tt in enumerate(t[:200]):

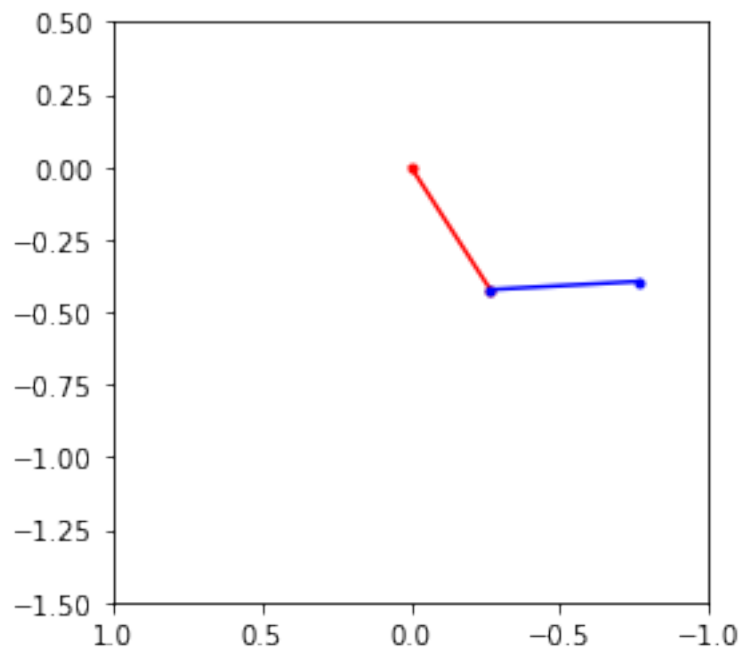
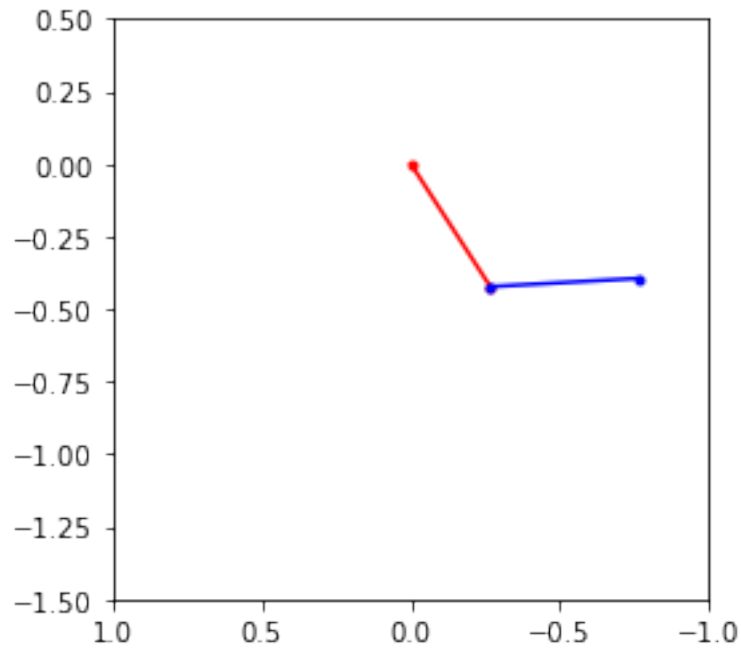
    x1 = + L * sin(x[t_idx, 0])
    y1 = - L * cos(x[t_idx, 0])

    x2 = x1 + L * sin(x[t_idx, 1])
    y2 = y1 - L * cos(x[t_idx, 1])

    ax.cla()
    ax.plot([0, x1], [0, y1], 'r.-')
    ax.plot([x1, x2], [y1, y2], 'b.-')
    ax.set_ylim([-1.5, 0.5])
    ax.set_xlim([1, -1])

    clear_output()
    display(fig)

    time.sleep(0.1)
```



Example: Damped harmonic oscillator ODE problems are important in computational physics, so we will look at one more example: the damped harmonic oscillation. This problem is well described on the wiki page: <http://en.wikipedia.org/wiki/Damping>

The equation of motion for the damped oscillator is:

$$\frac{d^2x}{dt^2} + 2\zeta\omega_0 \frac{dx}{dt} + \omega_0^2 x = 0$$

where x is the position of the oscillator, ω_0 is the frequency, and ζ is the damping ratio. To write this second-order ODE on standard form we introduce $p = \frac{dx}{dt}$:

$$\frac{dp}{dt} = -2\zeta\omega_0 p - \omega_0^2 x$$

$$\frac{dx}{dt} = p$$

In the implementation of this example we will add extra arguments to the RHS function for the ODE, rather than using global variables as we did in the previous example. As a consequence of the extra arguments to the RHS, we need to pass an keyword argument `args` to the `odeint` function:

```
[23]: def dy(y, t, zeta, w0):  
      """  
      The right-hand side of the damped oscillator ODE  
      """  
      x, p = y[0], y[1]  
  
      dx = p  
      dp = -2 * zeta * w0 * p - w0**2 * x  
  
      return [dx, dp]
```

```
[24]: # initial state:  
y0 = [1.0, 0.0]
```

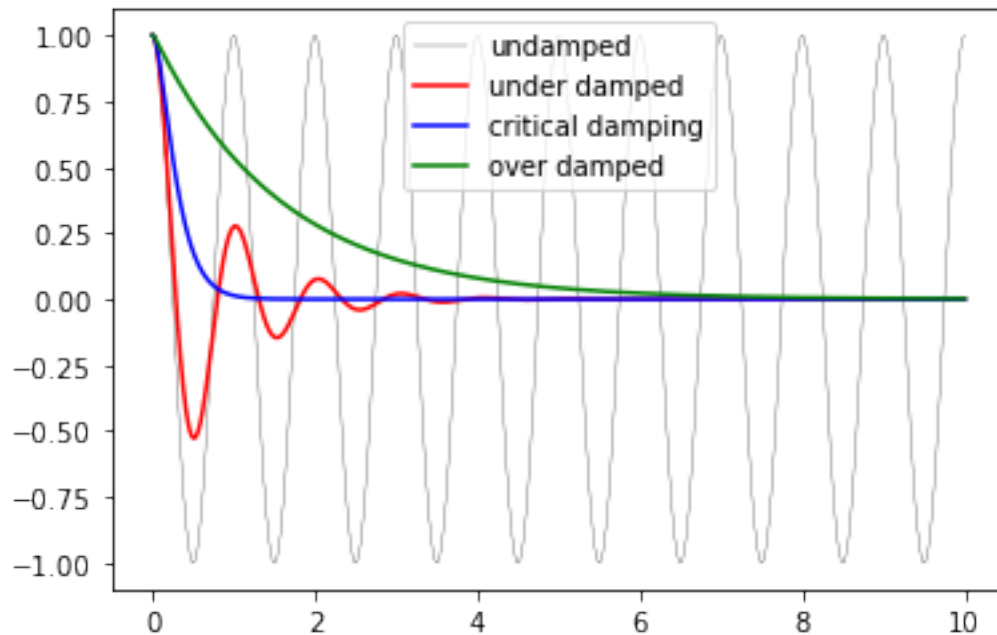
```
[25]: # time coordinate to solve the ODE for  
t = linspace(0, 10, 1000)  
w0 = 2*pi*1.0
```

```
<ipython-input-25-c9ca301b6aa9>:2: DeprecationWarning: scipy.linspace is  
deprecated and will be removed in SciPy 2.0.0, use numpy.linspace instead  
t = linspace(0, 10, 1000)
```

```
[26]: # solve the ODE problem for three different values of the damping ratio  
  
y1 = odeint(dy, y0, t, args=(0.0, w0)) # undamped  
y2 = odeint(dy, y0, t, args=(0.2, w0)) # under damped  
y3 = odeint(dy, y0, t, args=(1.0, w0)) # critial damping  
y4 = odeint(dy, y0, t, args=(5.0, w0)) # over damped
```

```
[27]: fig, ax = plt.subplots()  
ax.plot(t, y1[:,0], 'k', label="undamped", linewidth=0.25)  
ax.plot(t, y2[:,0], 'r', label="under damped")
```

```
ax.plot(t, y3[:,0], 'b', label=r"critical damping")
ax.plot(t, y4[:,0], 'g', label="over damped")
ax.legend();
```



1.5 Fourier transform

Fourier transforms are one of the universal tools in computational physics, which appear over and over again in different contexts. SciPy provides functions for accessing the classic [FFTPACK](#) library from NetLib, which is an efficient and well tested FFT library written in FORTRAN. The SciPy API has a few additional convenience functions, but overall the API is closely related to the original FORTRAN library.

To use the `fftpack` module in a python program, include it using:

```
[28]: from numpy.fft import fftfreq
      from scipy.fftpack import *
```

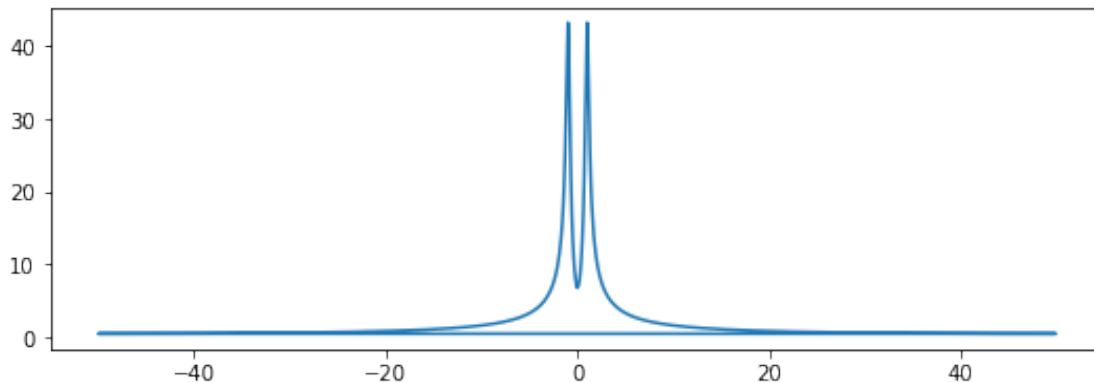
To demonstrate how to do a fast Fourier transform with SciPy, let's look at the FFT of the solution to the damped oscillator from the previous section:

```
[29]: N = len(t)
      dt = t[1]-t[0]

      # calculate the fast fourier transform
      # y2 is the solution to the under-damped oscillator from the previous section
      F = fft(y2[:,0])
```

```
# calculate the frequencies for the components in F
w = fftfreq(N, dt)
```

```
[30]: fig, ax = plt.subplots(figsize=(9,3))
      ax.plot(w, abs(F));
```

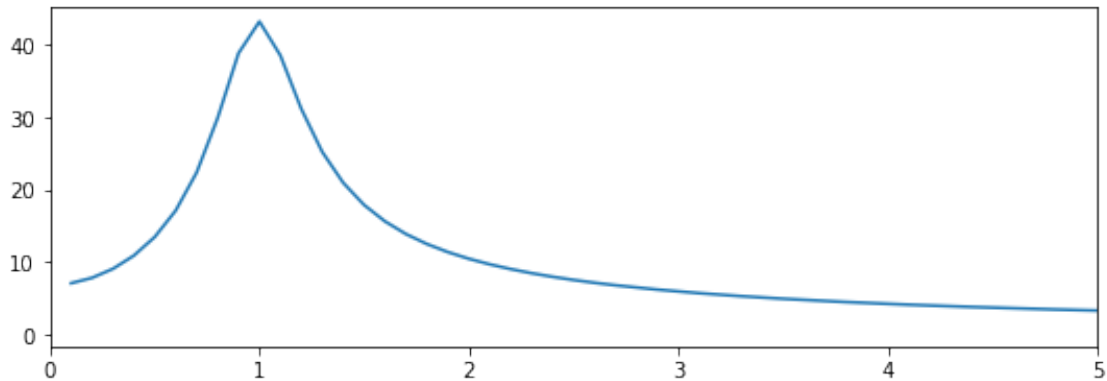


Since the signal is real, the spectrum is symmetric. We therefore only need to plot the part that corresponds to the positive frequencies. To extract that part of the `w` and `F` we can use some of the indexing tricks for NumPy arrays that we saw in Lecture 2:

```
[31]: indices = where(w > 0) # select only indices for elements that corresponds to
      →positive frequencies
      w_pos = w[indices]
      F_pos = F[indices]
```

```
<ipython-input-31-9b01025eb9eb>:1: DeprecationWarning: scipy.where is deprecated
and will be removed in SciPy 2.0.0, use numpy.where instead
      indices = where(w > 0) # select only indices for elements that corresponds to
      positive frequencies
```

```
[32]: fig, ax = plt.subplots(figsize=(9,3))
      ax.plot(w_pos, abs(F_pos))
      ax.set_xlim(0, 5);
```



As expected, we now see a peak in the spectrum that is centered around 1, which is the frequency we used in the damped oscillator example.

1.6 Linear algebra

The linear algebra module contains a lot of matrix related functions, including linear equation solving, eigenvalue solvers, matrix functions (for example matrix-exponentiation), a number of different decompositions (SVD, LU, cholesky), etc.

Detailed documetation is available at: <http://docs.scipy.org/doc/scipy/reference/linalg.html>

Here we will look at how to use some of these functions:

1.6.1 Linear equation systems

Linear equation systems on the matrix form

$$Ax = b$$

where A is a matrix and x, b are vectors can be solved like:

```
[33]: from scipy.linalg import *
```

```
[34]: A = array([[1,2,3], [4,5,6], [7,8,9]])
      b = array([1,2,3])
```

```
<ipython-input-34-d72a989a316d>:1: DeprecationWarning: scipy.array is deprecated
and will be removed in SciPy 2.0.0, use numpy.array instead
```

```
A = array([[1,2,3], [4,5,6], [7,8,9]])
```

```
<ipython-input-34-d72a989a316d>:2: DeprecationWarning: scipy.array is deprecated
and will be removed in SciPy 2.0.0, use numpy.array instead
```

```
b = array([1,2,3])
```

```
[35]: x = solve(A, b)
```

```
x
```

LinAlgError Traceback (most recent call last)

```
<ipython-input-35-1be648719d5f> in <module>
----> 1 x = solve(A, b)
      2
      3 x

c:\program files\python38\lib\site-packages\scipy\linalg\basic.py in
↳ solve(a, b, sym_pos, lower, overwrite_a, overwrite_b, debug, check_finite,
↳ assume_a, transposed)
    214                                     (a1, b1))
    215     lu, ipvt, info = getrf(a1, overwrite_a=overwrite_a)
--> 216     _solve_check(n, info)
    217     x, info = getrs(lu, ipvt, b1,
    218                     trans=trans, overwrite_b=overwrite_b)

c:\program files\python38\lib\site-packages\scipy\linalg\basic.py in
↳ _solve_check(n, info, lamch, rcond)
    29         '.'.format(-info))
    30     elif 0 < info:
---> 31         raise LinAlgError('Matrix is singular.')
    32
    33     if lamch is None:

LinAlgError: Matrix is singular.
```

```
[36]: # check
      dot(A, x) - b
```

```
<ipython-input-36-65dbf7dd8f1d>:2: DeprecationWarning: scipy.dot is deprecated
and will be removed in SciPy 2.0.0, use numpy.dot instead
      dot(A, x) - b
```

ValueError Traceback (most recent call last)

```
<ipython-input-36-65dbf7dd8f1d> in <module>
      1 # check
----> 2 dot(A, x) - b
```

```

c:\program files\python38\lib\site-packages\scipy\_lib\deprecation.py in
call(*args, **kwargs)
    18         warnings.warn(msg, category=DeprecationWarning,
    19                         stacklevel=stacklevel)
--> 20         return fun(*args, **kwargs)
    21     call.__doc__ = msg
    22     return call

```

```
<__array_function__ internals> in dot(*args, **kwargs)
```

```
ValueError: shapes (3,3) and (250,4) not aligned: 3 (dim 1) != 250 (dim 0)
```

We can also do the same with

$$AX = B$$

where A, B, X are matrices:

```
[37]: A = rand(3,3)
      B = rand(3,3)
```

```
<ipython-input-37-fb78b91fd340>:1: DeprecationWarning: scipy.rand is deprecated
and will be removed in SciPy 2.0.0, use numpy.random.rand instead
```

```
A = rand(3,3)
```

```
<ipython-input-37-fb78b91fd340>:2: DeprecationWarning: scipy.rand is deprecated
and will be removed in SciPy 2.0.0, use numpy.random.rand instead
```

```
B = rand(3,3)
```

```
[38]: X = solve(A, B)
```

```
[39]: X
```

```
[39]: array([[ 0.09848085,  0.27559363,  0.79789233],
             [-0.03249301,  0.75583669,  0.07135543],
             [ 0.17460686,  0.44178777, -0.08412888]])
```

```
[40]: # check
      norm(dot(A, X) - B)
```

```
<ipython-input-40-03be49e89a6e>:2: DeprecationWarning: scipy.dot is deprecated
and will be removed in SciPy 2.0.0, use numpy.dot instead
```

```
norm(dot(A, X) - B)
```

```
[40]: 2.797157557069881e-17
```


1.6.2 Eigenvalues and eigenvectors

The eigenvalue problem for a matrix A :

$$Av_n = \lambda_n v_n$$

where v_n is the n th eigenvector and λ_n is the n th eigenvalue.

To calculate eigenvalues of a matrix, use the `eigvals` and for calculating both eigenvalues and eigenvectors, use the function `eig`:

```
[41]: evals = eigvals(A)
```

```
[42]: evals
```

```
[42]: array([ 1.48259965+0.j          , -0.26360912+0.54975957j,
          -0.26360912-0.54975957j])
```

```
[43]: evals, vecs = eig(A)
```

```
[44]: evals
```

```
[44]: array([ 1.48259965+0.j          , -0.26360912+0.54975957j,
          -0.26360912-0.54975957j])
```

```
[45]: vecs
```

```
[45]: array([[ -0.59578591+0.j          ,  0.42773997+0.41602475j,
           0.42773997-0.41602475j],
          [ -0.61777776+0.j          ,  0.32500586-0.39447121j,
           0.32500586+0.39447121j],
          [ -0.51321515+0.j          , -0.6186482 +0.j          ,
           -0.6186482 -0.j          ]])
```

The eigenvectors corresponding to the n th eigenvalue (stored in `evals[n]`) is the n th *column* in `vecs`, i.e., `vecs[:,n]`. To verify this, let's try multiplying eigenvectors with the matrix and compare to the product of the eigenvector and the eigenvalue:

```
[46]: n = 1

norm(dot(A, vecs[:,n]) - evals[n] * vecs[:,n])
```

```
<ipython-input-46-289ff0b508b5>:3: DeprecationWarning: scipy.dot is deprecated
and will be removed in SciPy 2.0.0, use numpy.dot instead
norm(dot(A, vecs[:,n]) - evals[n] * vecs[:,n])
```

```
[46]: 2.435541875787129e-16
```

There are also more specialized eigensolvers, like the `eigh` for Hermitian matrices.

1.6.3 Matrix operations

```
[47]: # the matrix inverse
      inv(A)
```

```
[47]: array([[ -0.75596532,  1.55429748, -0.21036934],
            [-0.37661276,  0.11216194,  1.11410373],
            [ 1.46744288, -0.7717956 , -0.10000618]])
```

```
[48]: # determinant
      det(A)
```

```
[48]: 0.5511198761464301
```

```
[49]: # norms of various orders
      norm(A, ord=2), norm(A, ord=Inf)
```

```
[49]: (1.5304704006299419, 1.6102008947137665)
```

1.6.4 Sparse matrices

Sparse matrices are often useful in numerical simulations dealing with large systems, if the problem can be described in matrix form where the matrices or vectors mostly contains zeros. Scipy has a good support for sparse matrices, with basic linear algebra operations (such as equation solving, eigenvalue calculations, etc).

There are many possible strategies for storing sparse matrices in an efficient way. Some of the most common are the so-called coordinate form (COO), list of list (LIL) form, and compressed-sparse column CSC (and row, CSR). Each format has some advantages and disadvantages. Most computational algorithms (equation solving, matrix-matrix multiplication, etc) can be efficiently implemented using CSR or CSC formats, but they are not so intuitive and not so easy to initialize. So often a sparse matrix is initially created in COO or LIL format (where we can efficiently add elements to the sparse matrix data), and then converted to CSC or CSR before used in real calculations.

For more information about these sparse formats, see e.g. http://en.wikipedia.org/wiki/Sparse_matrix

When we create a sparse matrix we have to choose which format it should be stored in. For example,

```
[50]: from scipy.sparse import *
```

```
[51]: # dense matrix
      M = array([[1,0,0,0], [0,3,0,0], [0,1,1,0], [1,0,0,1]]); M
```

```
<ipython-input-51-156055dd1015>:2: DeprecationWarning: scipy.array is deprecated
and will be removed in SciPy 2.0.0, use numpy.array instead
```

```
      M = array([[1,0,0,0], [0,3,0,0], [0,1,1,0], [1,0,0,1]]); M
```

```
[51]: array([[1, 0, 0, 0],
           [0, 3, 0, 0],
           [0, 1, 1, 0],
           [1, 0, 0, 1]])
```

```
[52]: # convert from dense to sparse
A = csr_matrix(M); A
```

```
[52]: <4x4 sparse matrix of type '<class 'numpy.intc'>'
      with 6 stored elements in Compressed Sparse Row format>
```

```
[53]: # convert from sparse to dense
A.todense()
```

```
[53]: matrix([[1, 0, 0, 0],
             [0, 3, 0, 0],
             [0, 1, 1, 0],
             [1, 0, 0, 1]], dtype=int32)
```

More efficient way to create sparse matrices: create an empty matrix and populate with using matrix indexing (avoids creating a potentially large dense matrix)

```
[54]: A = lil_matrix((4,4)) # empty 4x4 sparse matrix
A[0,0] = 1
A[1,1] = 3
A[2,2] = A[2,1] = 1
A[3,3] = A[3,0] = 1
A
```

```
[54]: <4x4 sparse matrix of type '<class 'numpy.float64'>'
      with 6 stored elements in List of Lists format>
```

```
[55]: A.todense()
```

```
[55]: matrix([[1., 0., 0., 0.],
             [0., 3., 0., 0.],
             [0., 1., 1., 0.],
             [1., 0., 0., 1.]])
```

Converting between different sparse matrix formats:

```
[56]: A
```

```
[56]: <4x4 sparse matrix of type '<class 'numpy.float64'>'
      with 6 stored elements in List of Lists format>
```

```
[57]: A = csr_matrix(A); A
```

```
[57]: <4x4 sparse matrix of type '<class 'numpy.float64'>'
      with 6 stored elements in Compressed Sparse Row format>
```

```
[58]: A = csc_matrix(A); A
```

```
[58]: <4x4 sparse matrix of type '<class 'numpy.float64'>'
      with 6 stored elements in Compressed Sparse Column format>
```

We can compute with sparse matrices like with dense matrices:

```
[59]: A.todense()
```

```
[59]: matrix([[1., 0., 0., 0.],
             [0., 3., 0., 0.],
             [0., 1., 1., 0.],
             [1., 0., 0., 1.]])
```

```
[60]: (A * A).todense()
```

```
[60]: matrix([[1., 0., 0., 0.],
             [0., 9., 0., 0.],
             [0., 4., 1., 0.],
             [2., 0., 0., 1.]])
```

```
[61]: A.todense()
```

```
[61]: matrix([[1., 0., 0., 0.],
             [0., 3., 0., 0.],
             [0., 1., 1., 0.],
             [1., 0., 0., 1.]])
```

```
[62]: A.dot(A).todense()
```

```
[62]: matrix([[1., 0., 0., 0.],
             [0., 9., 0., 0.],
             [0., 4., 1., 0.],
             [2., 0., 0., 1.]])
```

```
[63]: v = array([1,2,3,4])[:,newaxis]; v
```

```
<ipython-input-63-c06991884813>:1: DeprecationWarning: scipy.array is deprecated
and will be removed in SciPy 2.0.0, use numpy.array instead
```

```
v = array([1,2,3,4])[:,newaxis]; v
```

```
[63]: array([[1],
            [2],
            [3],
            [4]])
```

```
[64]: # sparse matrix - dense vector multiplication
A * v
```

```
[64]: array([[1.],
          [6.],
          [5.],
          [5.]])
```

```
[65]: # same result with dense matrix - dense vector multiplication
A.todense() * v
```

```
[65]: matrix([[1.],
             [6.],
             [5.],
             [5.]])
```

1.7 Optimization

Optimization (finding minima or maxima of a function) is a large field in mathematics, and optimization of complicated functions or in many variables can be rather involved. Here we will only look at a few very simple cases. For a more detailed introduction to optimization with SciPy see: http://scipy-lectures.github.com/advanced/mathematical_optimization/index.html

To use the optimization module in scipy first include the optimize module:

```
[66]: from scipy import optimize
```

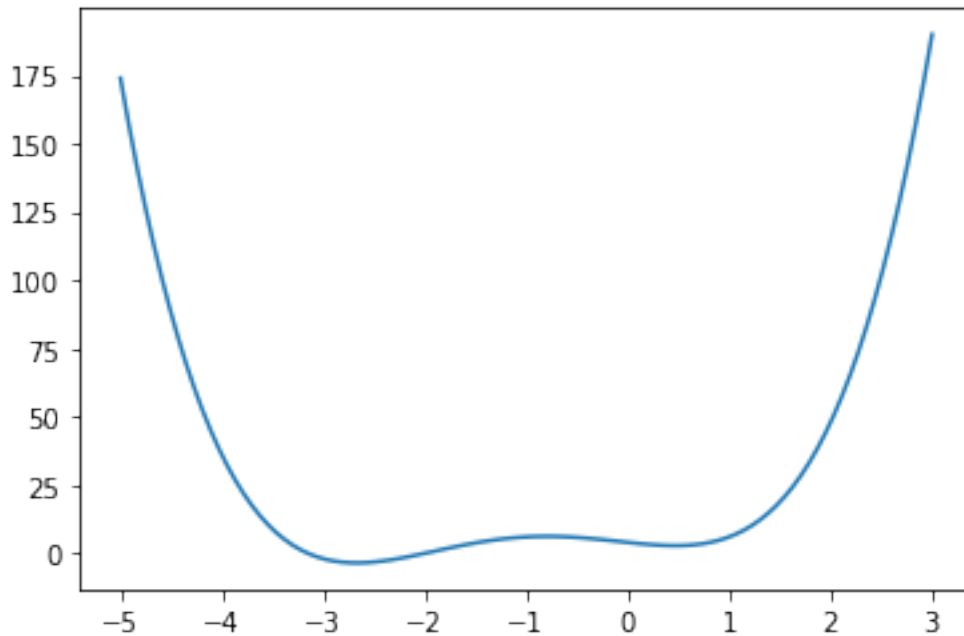
1.7.1 Finding a minima

Let's first look at how to find the minima of a simple function of a single variable:

```
[67]: def f(x):
      return 4*x**3 + (x-2)**2 + x**4
```

```
[68]: fig, ax = plt.subplots()
      x = linspace(-5, 3, 100)
      ax.plot(x, f(x));
```

```
<ipython-input-68-b4dec7d143f8>:2: DeprecationWarning: scipy.linspace is
deprecated and will be removed in SciPy 2.0.0, use numpy.linspace instead
      x = linspace(-5, 3, 100)
```



We can use the `fmin_bfgs` function to find the minima of a function:

```
[69]: x_min = optimize.fmin_bfgs(f, -2)
      x_min
```

```
Optimization terminated successfully.
      Current function value: -3.506641
      Iterations: 5
      Function evaluations: 24
      Gradient evaluations: 8
```

```
[69]: array([-2.67298155])
```

```
[70]: optimize.fmin_bfgs(f, 0.5)
```

```
Optimization terminated successfully.
      Current function value: 2.804988
      Iterations: 3
      Function evaluations: 15
      Gradient evaluations: 5
```

```
[70]: array([0.46961745])
```

We can also use the `brent` or `fminbound` functions. They have a bit different syntax and use different algorithms.

```
[71]: optimize.brent(f)
```

```
[71]: 0.46961743402759754
```

```
[72]: optimize.fminbound(f, -4, 2)
```

```
[72]: -2.6729822917513886
```

1.7.2 Finding a solution to a function

To find the root for a function of the form $f(x) = 0$ we can use the `fsolve` function. It requires an initial guess:

```
[73]: omega_c = 3.0
def f(omega):
    # a transcendental equation: resonance frequencies of a low-Q SQUID
    # terminated microwave resonator
    return tan(2*pi*omega) - omega_c/omega
```

```
[74]: fig, ax = plt.subplots(figsize=(10,4))
x = linspace(0, 3, 1000)
y = f(x)
mask = where(abs(y) > 50)
x[mask] = y[mask] = NaN # get rid of vertical line when the function flip sign
ax.plot(x, y)
ax.plot([0, 3], [0, 0], 'k')
ax.set_ylim(-5,5);
```

```
<ipython-input-74-8fa973be786d>:2: DeprecationWarning: scipy.linspace is
deprecated and will be removed in SciPy 2.0.0, use numpy.linspace instead
```

```
    x = linspace(0, 3, 1000)
```

```
<ipython-input-73-35ba93b217b0>:4: DeprecationWarning: scipy.tan is deprecated
and will be removed in SciPy 2.0.0, use numpy.tan instead
```

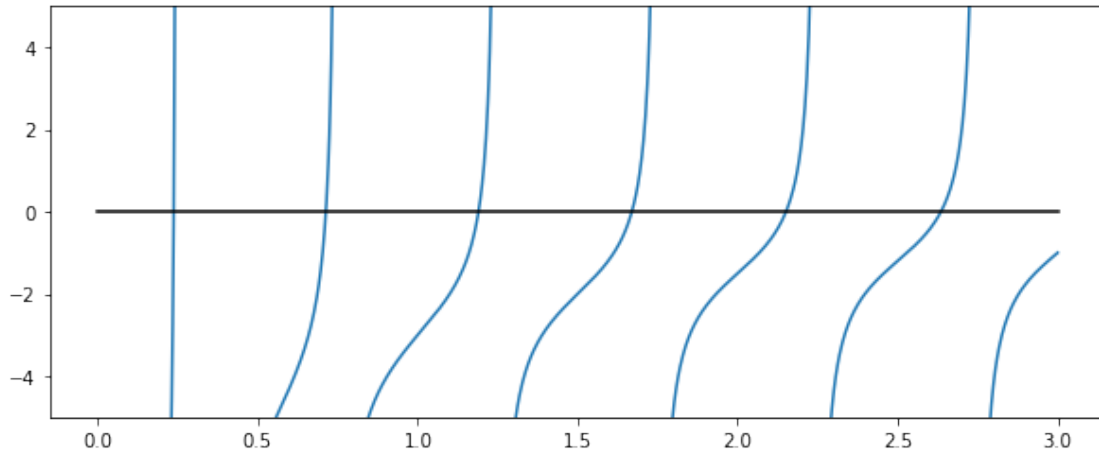
```
    return tan(2*pi*omega) - omega_c/omega
```

```
<ipython-input-73-35ba93b217b0>:4: RuntimeWarning: divide by zero encountered in
true_divide
```

```
    return tan(2*pi*omega) - omega_c/omega
```

```
<ipython-input-74-8fa973be786d>:4: DeprecationWarning: scipy.where is deprecated
and will be removed in SciPy 2.0.0, use numpy.where instead
```

```
    mask = where(abs(y) > 50)
```



```
[75]: optimize.fsolve(f, 0.1)
```

```
<ipython-input-73-35ba93b217b0>:4: DeprecationWarning: scipy.tan is deprecated
and will be removed in SciPy 2.0.0, use numpy.tan instead
    return tan(2*pi*omega) - omega_c/omega
```

```
[75]: array([0.23743014])
```

```
[76]: optimize.fsolve(f, 0.6)
```

```
<ipython-input-73-35ba93b217b0>:4: DeprecationWarning: scipy.tan is deprecated
and will be removed in SciPy 2.0.0, use numpy.tan instead
    return tan(2*pi*omega) - omega_c/omega
```

```
[76]: array([0.71286972])
```

```
[77]: optimize.fsolve(f, 1.1)
```

```
<ipython-input-73-35ba93b217b0>:4: DeprecationWarning: scipy.tan is deprecated
and will be removed in SciPy 2.0.0, use numpy.tan instead
    return tan(2*pi*omega) - omega_c/omega
```

```
[77]: array([1.18990285])
```

1.8 Interpolation

Interpolation is simple and convenient in scipy: The `interp1d` function, when given arrays describing X and Y data, returns an object that behaves like a function that can be called for an arbitrary value of x (in the range covered by X), and it returns the corresponding interpolated y value:

```
[78]: from scipy.interpolate import *
```



```
[79]: def f(x):  
      return sin(x)
```

```
[80]: n = arange(0, 10)  
      x = linspace(0, 9, 100)  
  
      y_meas = f(n) + 0.1 * randn(len(n)) # simulate measurement with noise  
      y_real = f(x)  
  
      linear_interpolation = interp1d(n, y_meas)  
      y_interp1 = linear_interpolation(x)  
  
      cubic_interpolation = interp1d(n, y_meas, kind='cubic')  
      y_interp2 = cubic_interpolation(x)
```

<ipython-input-80-b19110fe243b>:1: DeprecationWarning: scipy.arange is deprecated and will be removed in SciPy 2.0.0, use numpy.arange instead

```
n = arange(0, 10)
```

<ipython-input-80-b19110fe243b>:2: DeprecationWarning: scipy.linspace is deprecated and will be removed in SciPy 2.0.0, use numpy.linspace instead

```
x = linspace(0, 9, 100)
```

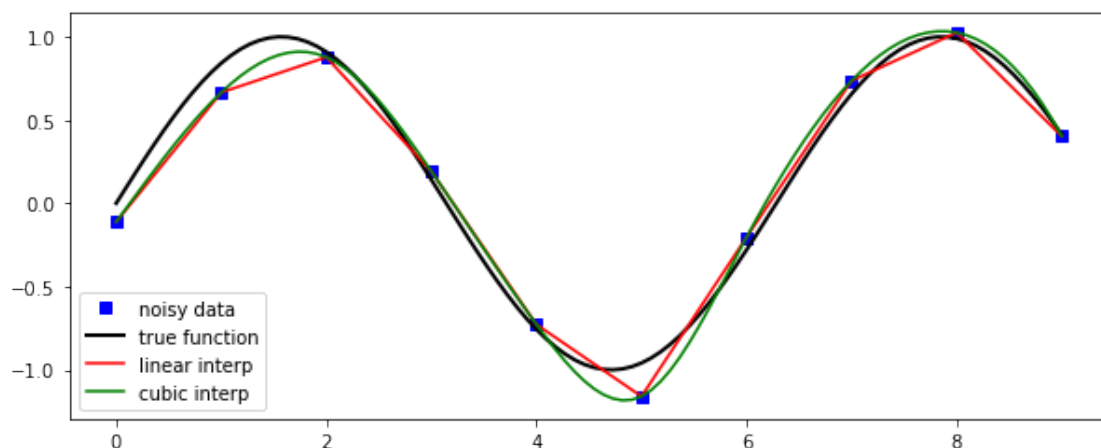
<ipython-input-79-2ef07bb30186>:2: DeprecationWarning: scipy.sin is deprecated and will be removed in SciPy 2.0.0, use numpy.sin instead

```
return sin(x)
```

<ipython-input-80-b19110fe243b>:4: DeprecationWarning: scipy.randn is deprecated and will be removed in SciPy 2.0.0, use numpy.random.randn instead

```
y_meas = f(n) + 0.1 * randn(len(n)) # simulate measurement with noise
```

```
[81]: fig, ax = plt.subplots(figsize=(10,4))  
      ax.plot(n, y_meas, 'bs', label='noisy data')  
      ax.plot(x, y_real, 'k', lw=2, label='true function')  
      ax.plot(x, y_interp1, 'r', label='linear interp')  
      ax.plot(x, y_interp2, 'g', label='cubic interp')  
      ax.legend(loc=3);
```



1.9 Statistics

The `scipy.stats` module contains a large number of statistical distributions, statistical functions and tests. For a complete documentation of its features, see <http://docs.scipy.org/doc/scipy/reference/stats.html>.

There is also a very powerful python package for statistical modelling called `statsmodels`. See <http://statsmodels.sourceforge.net> for more details.

```
[82]: from scipy import stats
```

```
[83]: # create a (discrete) random variable with poissonian distribution

X = stats.poisson(3.5) # photon distribution for a coherent state with n=3.5
    ↪ photons
```

```
[84]: n = arange(0,15)

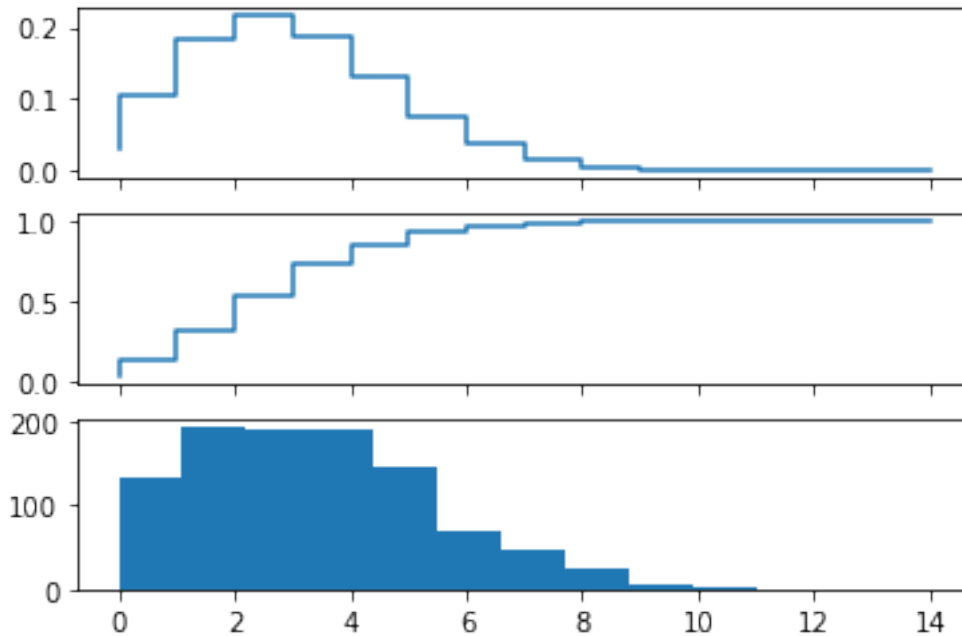
fig, axes = plt.subplots(3,1, sharex=True)

# plot the probability mass function (PMF)
axes[0].step(n, X.pmf(n))

# plot the cumulative distribution function (CDF)
axes[1].step(n, X.cdf(n))

# plot histogram of 1000 random realizations of the stochastic variable X
axes[2].hist(X.rvs(size=1000));
```

```
<ipython-input-84-078f0313a228>:1: DeprecationWarning: scipy.arange is
deprecated and will be removed in SciPy 2.0.0, use numpy.arange instead
  n = arange(0,15)
```



```
[85]: # create a (continous) random variable with normal distribution
Y = stats.norm()
```

```
[86]: x = linspace(-5,5,100)

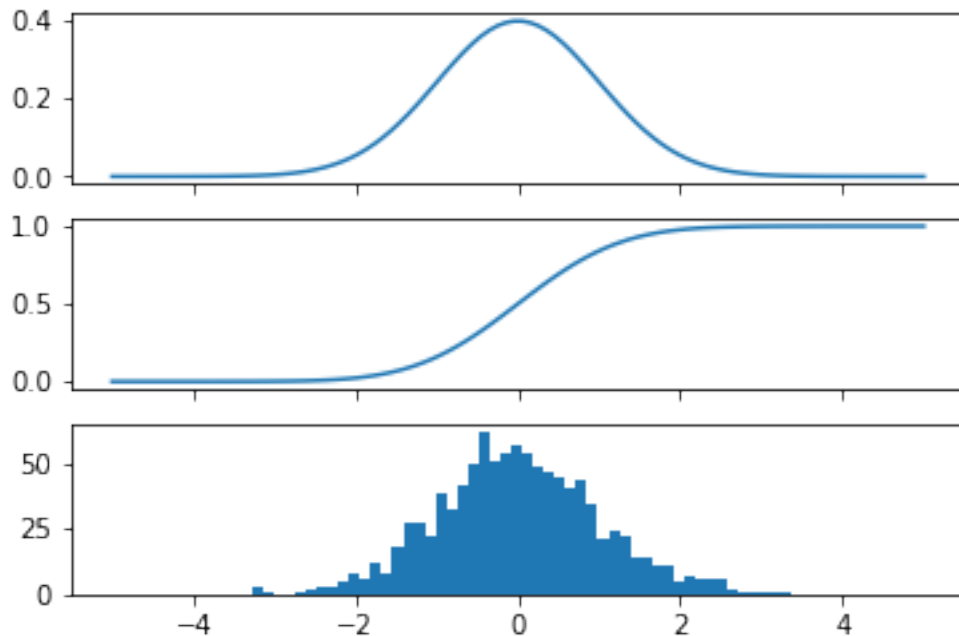
fig, axes = plt.subplots(3,1, sharex=True)

# plot the probability distribution function (PDF)
axes[0].plot(x, Y.pdf(x))

# plot the commulative distributin function (CDF)
axes[1].plot(x, Y.cdf(x));

# plot histogram of 1000 random realizations of the stochastic variable Y
axes[2].hist(Y.rvs(size=1000), bins=50);
```

<ipython-input-86-c7b37fdaada3>:1: DeprecationWarning: scipy.linspace is deprecated and will be removed in SciPy 2.0.0, use numpy.linspace instead
 x = linspace(-5,5,100)



Statistics:

```
[87]: X.mean(), X.std(), X.var() # poisson distribution
```

```
[87]: (3.5, 1.8708286933869707, 3.5)
```

```
[88]: Y.mean(), Y.std(), Y.var() # normal distribution
```

```
[88]: (0.0, 1.0, 1.0)
```

1.9.1 Statistical tests

Test if two sets of (independent) random data comes from the same distribution:

```
[89]: t_statistic, p_value = stats.ttest_ind(X.rvs(size=1000), X.rvs(size=1000))

print "t-statistic =", t_statistic
print "p-value =", p_value
```

```
File "<ipython-input-89-0db034d77d0f>", line 3
print "t-statistic =", t_statistic
^
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean
↳ print("t-statistic =", t_statistic)?
```

Since the p value is very large we cannot reject the hypothesis that the two sets of random data have *different* means.

To test if the mean of a single sample of data has mean 0.1 (the true mean is 0.0):

```
[90]: stats.ttest_1samp(Y.rvs(size=1000), 0.1)
```

```
[90]: Ttest_1sampResult(statistic=-3.6509309802271876, pvalue=0.00027483761569202465)
```

Low p-value means that we can reject the hypothesis that the mean of Y is 0.1.

```
[91]: Y.mean()
```

```
[91]: 0.0
```

```
[92]: stats.ttest_1samp(Y.rvs(size=1000), Y.mean())
```

```
[92]: Ttest_1sampResult(statistic=-1.9098489555750828, pvalue=0.05643882242266383)
```

1.10 Further reading

- <http://www.scipy.org> - The official web page for the SciPy project.
- <http://docs.scipy.org/doc/scipy/reference/tutorial/index.html> - A tutorial on how to get started using SciPy.
- <https://github.com/scipy/scipy/> - The SciPy source code.

1.11 Versions

```
[93]: %reload_ext version_information  
  
%version_information numpy, matplotlib, scipy
```

```
[93]:
```

Software	Version
Python	3.8.2 64bit [MSC v.1916 64 bit (AMD64)]
IPython	7.15.0
OS	Windows 10 10.0.19041 SP0
numpy	1.18.5
matplotlib	3.2.1
scipy	1.4.1
Fri Jun 19 11:25:42 2020 India Standard Time	