SciPy

Many of the examples can be found here: http://www.scipy.org/Cookbook

scipy.optimize

```
import numpy, pylab
import scipy.optimize as optimize
```

```
print dir(optimize)
```

```
['ScipyTest', '__all__', '__builtins__', '__doc__', '__file__', '__name__', '__path__', '__warningregistry__', '
    _cobyla', '_lbfgsb', '_minpack', '_zeros', 'anneal', 'approx_fprime', 'bisect', 'bisection', 'bracket', 'brent
    ', 'brenth', 'brentq', 'brute', 'check_grad', 'cobyla', 'fixed_point', 'fmin', 'fmin_bfgs', 'fmin_cg', '
    fmin_cobyla', 'fmin_l_bfgs_b', 'fmin_ncg', 'fmin_powell', 'fmin_tnc', 'fminbound', 'fsolve', 'golden', 'lbfgsb
    ', 'leastsq', 'line_search', 'linesearch', 'minpack', 'minpack2', 'moduleTNC', 'newton', 'optimize', 'ridder',
     'rosen', 'rosen_der', 'rosen_hess', 'rosen_hess_prod', 'test', 'tnc', 'zeros']
```

```
print
```

Curve-fitting with leastsq

```
from scipy.optimize import leastsq
```

```
# Generate data
from numpy import random,histogram,arange,sqrt,exp,nonzero

n = 1000; isi = random.exponential(0.1,size=n)
db = 0.01; bins = arange(0,1.0,db)
h = histogram(isi,bins)[0]
p = h.astype(float)/n/db

# Function to be fit
# x - independent variable
# p - tuple of parameters
fitfunc = lambda p, x: exp(-x/p[0])/p[0]

# Standard form, here err is absolute error
errfunc = lambda p, x, y, err: (y - fitfunc(p, x)) / err

```

```
# Initial values for fit parameters
pinit = numpy.array([0.2])

# Hist count less than 4 has poor estimate of the weight
# don't use in the fitting process
idx = numpy.nonzero(h>4)

out = leastsq(errfunc, pinit,args=(bins[idx]+0.01/2, p[idx],p[idx]/sqrt(h[idx])),full_output = 1)
```

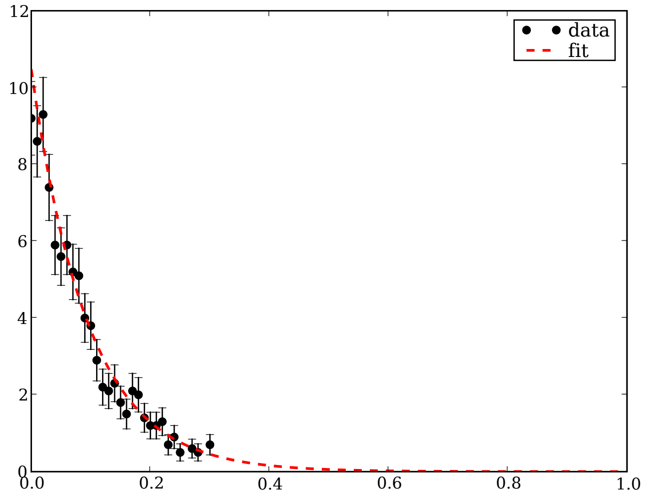
```

l1 = 'data'

pylab.errorbar(bins[idx],p[idx],yerr=p[idx]/sqrt(h[idx]),fmt='ko',label=l1)

l2 = 'fit'
pylab.plot(bins,fitfunc((out[0],),bins),'r--',lw=2,label=l2)

pylab.legend()
pylab.show()
```



```
print
```

scipy.stats

```
import scipy.stats as stats
```

```
a = numpy.random.normal(loc=10.,scale=5.0,size=1000)
b = numpy.random.normal(loc=10.,scale=5.0,size=1000)
c = numpy.random.normal(loc=20.,scale=5.0,size=1000)
print stats.ttest_rel(a,b)
```

```
(array(-0.97634636151146859), 0.329129212477)
```

```
print stats.ttest_rel(a,c)
```

```
(array(-44.677560208633174), 2.02572832058e-240)
```

```
print
```

scipy.linalg

Finding Inverse

```
a = numpy.mat('[1 3 5; 2 5 1; 2 3 8]')
```

```
print a
```

```
[[1 3 5]
 [2 5 1]
 [2 3 8]]
```

```
print a.I
```

```
[[-1.48  0.36  0.88]
 [ 0.56  0.08 -0.36]
 [ 0.16 -0.12  0.04]]
```

```
from scipy import linalg
print linalg.inv(a)
```

```
[[-1.48  0.36  0.88]
 [ 0.56  0.08 -0.36]
 [ 0.16 -0.12  0.04]]
```

```
print
```

Solving linear systems of equations

```
# x + 3y + 5z = 10
# 2x + 5y + z = 8
# 2x + 3y +8z = 3
a = numpy.mat('[1 3 5; 2 5 1; 2 3 8]')
b = numpy.mat('[10;8;3]')
print linalg.solve(a,b)
```

```
[[-9.28]
 [ 5.16]
 [ 0.76]]
```

```
print
```

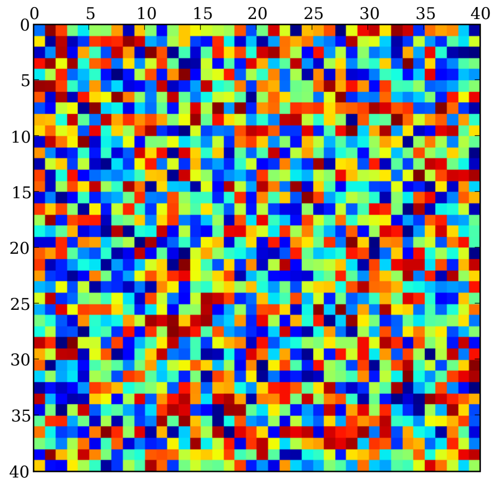
scipy.ndimage

```
import scipy.ndimage as ndimage
```

Generate a noise image

```
image = numpy.random.uniform(low=0.,high=1.,size=(40,40))
```

```
pylab.matshow(image)
pylab.show()
```

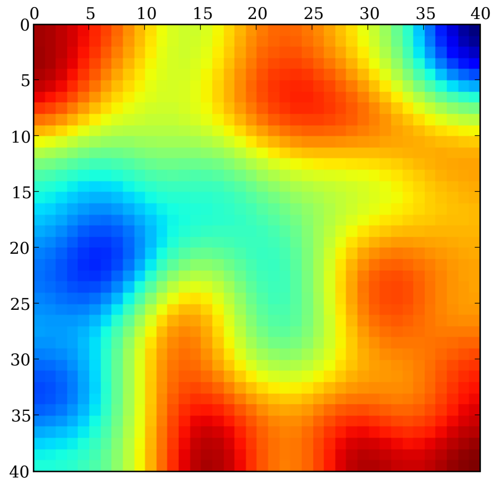


```
print
```

Apply Gaussian filter

```
sigma = 4.
```

```
image2 = ndimage.gaussian_filter(image,sigma)
pylab.matshow(image2)
pylab.show()
```

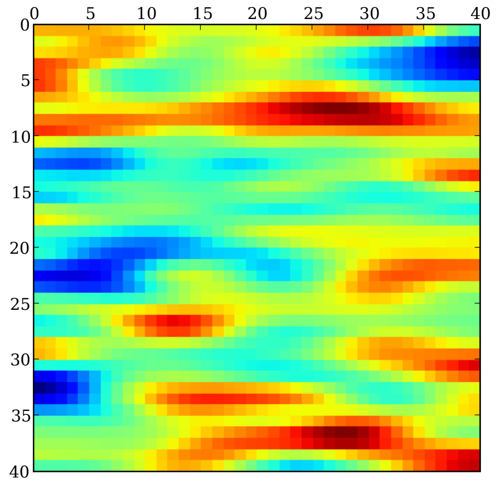


```
print
```

Apply Gaussian filter only in one dimension

```
sigma = (1.,4.)
```

```
image3 = ndimage.gaussian_filter(image,sigma)
pylab.matshow(image3)
pylab.show()
```

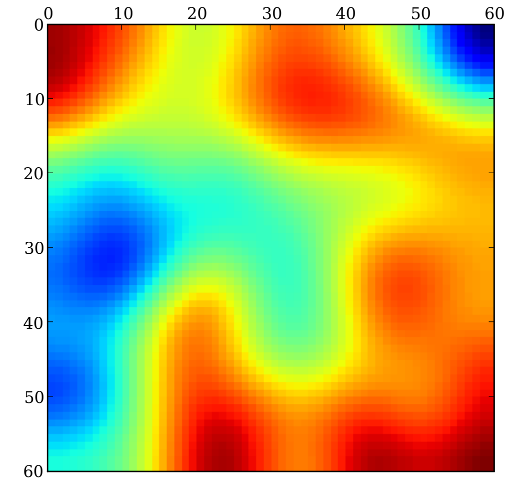


```
print
```

Zoom

```
zoom = (1.5,1.5)
```

```
image4 = ndimage.zoom(image2,zoom)
pylab.matshow(image4)
pylab.show()
```



```
print
```

scipy.weave

```
import scipy.weave as weave
```

```

def f_blitz(a,b,c):

    code = r"""

    for(int i=0;i<Na[0];i++) {
      c(i) = a(i)*b(i);
    }

    """

    weave.inline(code,['a','b','c'], type_converters=weave.converters.blitz)

a = 2*numpy.ones(10)
```

```
b = numpy.arange(0,10)
c = numpy.zeros(10)

f_blitz(a,b,c)
print c
```

```
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.]
```