Introduction to scipy

- common scientific programming routine
- built on top if Numpy, functions available through scipy
- many specialized subpackages
  - fftpack,optimize,integrate,interpolate,special,stats,signal,...
  - additional experimental scikits

1D interpolation

```
import pylab as pl
import numpy as np
from scipy import interpolate

def function(x):
    return 2.*x**4 - 5.*x

x=np.random.uniform(-10,10,10)
x.sort()
y=function(x)

fit=interpolate.UnivariateSpline(x,y,k=3)

xx=np.array([.1*i for i in range(-100,101)])
y1=function(xx)
y2=fit(xx)
pl.plot(x,y,ls='None',marker='o')
pl.plot(xx,y1)
pl.plot(xx,y2)
pl.show()
```

Interpolation, gridded data

```
import pylab as pl
import numpy as np
from scipy import interpolate
from scipy.special import erf

def surface(x,y):
    return erf(x)*erf(y)

data=[[surface(i*.1,j*.1) for j in range(-20,21)] for i in range(-20,21)]
data=np.array(data)
pl.figure(1)
pl.imshow(data,interpolation='nearest')
pl.colorbar()
pl.show()

xdata,ydata=np.mgrid[-2:2:10j,-2:2:10j]
zdata=surface(xdata,ydata)

interpSurf = interpolate.bisplrep(xdata,ydata,zdata,s=0)
x2=np.linspace(-2,2,41)
y2=np.linspace(-2,2,41)
znew = interpolate.bisplev(x2,y2,interpSurf)
pl.figure(2)
#pl.imshow(znew,interpolation='nearest')
pl.imshow(znew)
pl.colorbar()
pl.show()
```

Interpolation, gridded data

```
import pylab as pl
import numpy as np
from scipy import interpolate
from scipy.special import erf

def surface(x,y):
    return erf(x)*erf(y)

data=[[surface(i*.1,j*.1) for j in range(-20,21)] for i in range(-20,21)]
data=np.array(data)
data=np.flipud(data) #flip to properly order data on plot
pl.figure(1)
pl.imshow(data,interpolation='nearest')
pl.colorbar()
pl.show()

xdata=np.random.uniform(-2,2,10)
ydata=np.random.uniform(-2,2,10)
zdata=surface(xdata,ydata)

interp=interpolate.Rbf(xdata,ydata,zdata)
xd1,yd1=np.mgrid[-2:2:41j,-2:2:41j]
xd2=.5*(xd1[1:,1:]+xd1[:-1,:-1])
yd2=.5*(yd1[1:,1:]+yd1[:-1,:-1])
znew=interp(xd2,yd2)

pl.figure(2)
pl.pcolor(xd1,yd1,znew)#pcolor zvalues for center of cells
```

```
pl.colorbar()
pl.scatter(xdata,ydata,s=2)
pl.xlim(-2,2)
pl.ylim(-2,2)
pl.show()
```

Statistical Distributions

```
from scipy import stats
import pylab as pl

x=[.1*i for i in range(-20,20)]
y=[stats.norm.cdf(i,loc=0,scale=.5) for i in x]

xx=[1*1.5**i for i in range(10)]
xx.extend([100.-i for i in xx])
xx.append(50.)
xx.sort()

yy=[stats.norm.ppf(i/100.) for i in xx]
yy2=[stats.norm.ppf(i) for i in y]

pl.figure(1)
pl.plot(x,y)

pl.figure(2)
pl.plot(xx,yy)
pl.plot(x,yy2)#prob paper-like plot
pl.show()
```

Stats: Linear Regression

```
import numpy as np
import pylab as pl

from scipy import stats

##make some data
x=np.random.uniform(0,20,10)
x.sort()
err=np.random.normal(0,1.,10)
y=[.1*x[i]**2 +1. + err[i] for i in range(len(x))]
##calc regression coef
m,b,r,p,stderr=stats.linregress(x,y)

pl.plot(x,y,ls='None',marker='s')
linfitX=[0,max(x)]
linfitY=[b,m*max(x)+b]
pl.plot(linfitX,linfitY)
pl.show()
```

Stats: Linear Regression (power law)

```
import numpy as np
import pylab as pl

from scipy import stats

##make some data
xx=np.random.uniform(0,20,10)
xx.sort()
err=np.random.normal(0,1.,10)
yy=[.1*2.**(xx[i] + err[i]) for i in range(len(xx))]
##transform data
x=np.log(xx)
y=np.log(yy)

##calc regression coef
m,b,r,p,stderr=stats.linregress(x,y)

pl.plot(x,y,ls='None',marker='s')
linfitX=[0,max(x)]
linfitY=[b,m*max(x)+b]
pl.plot(linfitX,linfitY)
pl.show()
```

Simultaneous Equations

```
import numpy as np
from scipy import linalg

#make matrix problem
A=5*np.identity(5)
L=np.tri(5,5,-1)- np.tri(5,5,-2)
U=np.tri(5,5,1)- np.tri(5,5,0)

A=A+2*L+2*U
b=np.arange(5)

x=linalg.solve(A,b)#solve mtrx eq
bb=np.dot(A,x)#mtrx mult
Ai=linalg.inv(A)#inverse of A
AiA=np.dot(Ai,A)
```