Python in a Nutshell

Part III: Introduction to SciPy and SimPy

Manel Velasco,[1] PhD and Alexandre Perera,[1,2] PhD

[1]Departament d'Enginyeria de Sistemes, Automatica i Informatica Industrial (ESAII)
Universitat Politecnica de Catalunya

[2]Centro de Investigacion Biomedica en Red en Bioingenieria, Biomateriales y Nanomedicina (CIBER-BBN)
Alexandre.Perera@upc.edu Manel.Velasco@upc.edu

Introduction to Python for Engineering and Statistics
Febraury, 2013

Contents I

1. Introduction
   - Input/Output
2. Statistics
   - First statistics
   - Probability Distributions
   - Density Estimation
   - Statistical Testing
3. Some Calculus
   - Linear Algebra
   - Fast Fourier Transforms
   - Optimization
   - Interpolation
   - Numerical Integration
   - Signal and Image Processing
4. Storage Schemes and code profiling

Contents II

- Introduction to storage of large data
- Storage Schemes
- Linear System Solvers
- Others
- Notes On code optimization and profiling
- Profiling your code
- Speeding your code

5. SymPy
   - First Steps with SimPy
   - Algebraic manipulations
   - Calculus
   - Equation solving
   - Linear Algebra

SciPy

Scipy

Up to now:

python A general purpose programming language. It is interpreted and dynamically typed and is very suited for interactive work and quick prototyping, while being powerful enough to write large applications in.

Scipy

Up to now:

python A general purpose programming language. It is interpreted and dynamically typed and is very suited for interactive work and quick prototyping, while being powerful enough to write large applications in.

Numpy A language extension that defines the numerical array and matrix type and basic operations on them.

Scipy

Up to now:

- **python** A general purpose programming language. It is interpreted and dynamically typed and is very suited for interactive work and quick prototyping, while being powerful enough to write large applications in.
- **Numpy** A language extension that defines the numerical array and matrix type and basic operations on them.
- **matplotlib** A language extension to facilitate plotting.

Scipy

Up to now:

python A general purpose programming language. It is interpreted and dynamically typed and is very suited for interactive work and quick prototyping, while being powerful enough to write large applications in.

Numpy A language extension that defines the numerical array and matrix type and basic operations on them.

matplotlib A language extension to facilitate plotting.

Scipy Scipy is another language extension that uses numpy to do advanced math, signal processing, optimization, statistics and much more.

History of SciPy

- **1995** first, there was *Numeric*, developed by Jim Hugunin
- **2001** Several people used Numeric for writing sientific code. Travis Oliphant, Eric Jones and Pearu Peterson merged their modules in one scientific super package: *SciPy* was born.
- **2001-2004** *numarray* was created by Perry Greenfield, Todd Miller and Rick White at the Space Science Telescope Institute as a replacement for Numeric.
- **2005** Travis Oliphant took *Numeric* and assambled a multi dimensional array project *SciPy core*. *Numerix* was born but as there was a DSPs company with the same name, *NumPy* was reborn.
- **2006** Guido and Travis discussed which parts of *NumPy* should go into Python standard libraries.

http://scipy.org

Entrada

SciPy.org Sponsored By ENTHOUGHT

Cerca | Títols | Text

Wiki
- Documentation
- Mailing Lists
- Download
- Installing SciPy
- Topical Software
- Cookbook
- Developer Zone
- Blogs
- Conference
- SciPy

Pàgina
- Pàgina no modificable
- Informació
- Adjuncions
- Més accions:

SciPy

Download | Getting Started | Documentation | Report Bugs | Read the Blog

John Hunter (1968-2012)

All,

We have achingly sad news today. Our friend, colleague, and author of matplotlib, John Hunter, passed away on August 28, 2012 at 10am. He was 44. It was little more than a month ago when John was our keynote speaker at SciPy 2012. He returned home to a diagnosis of cancer followed by a brutally short battle with this terrible illness. Fernando Perez's fitting tribute to his close friend highlights just how much impact John has had on all of our work and lives.

John is survived by his wife Miriam, his three daughters Rahel, Ava and Clara, his sisters Layne and Mary, and his mother Sarah. If you have benefited from John's many contributions, please say thanks in the way that would matter

News

EuroSciPy 2013. EuroSciPy is the European gathering for scientists using Python. The 2013 edition will place in Brussels, Aug. 21-24.

SciPy 0.11.0 released. (2012-09-25) See the Download page.

SIAM CSE '13. The SIAM Conference on Computational Science and Engineering will take place in Boston, February 25-March 1, 2013, and for this version there will be a track focused on the topic of Big Data.

AMS Annual Meeting. The annual meeting of the American Meteorological Society takes place January 6-10, 2013, and includes the Third Symposium on Advances in Modeling and Analysis Using Python.

scipi.io

Maybe the most common IO in SciPy is to import and export Matlab files using *loadmat/savemat*. In SciPy is easy to write/read them.

```
>>> import numpy as np
>>> from scipy import io as spio
>>> py_a = np.ones((2,2))
>>> spio.savemat('ex.mat',{'mat_a': py_a})
>>> py_mat = spio.loadmat('ex.mat')
>>> py_mat['mat_a']
array([[ 1.,  1.],
       [ 1.,  1.]])
```

scipy.io

scipy.io contains modules, classes and functions to read and write data to a variety of formats:

Matlab `loadmat(file_name[, mdict, appendmat])`

`savemat(file_name[, mdict, appendmat])`

Matrix Market http://math.nist.gov/MatrixMarket/

`mminfo()`, `mmread()` and `mmwrite()`

Wav Through `scipy.io.wavfile`.

`read()`, `write(file, rate, data)`

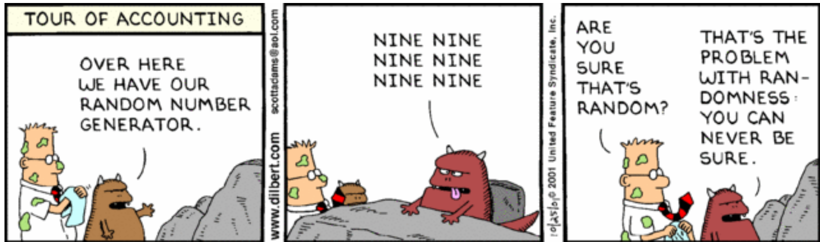
WEKA ARFF is a text file format which support numerical, string and data values, with support of missing and sparse data.

`loadarff()` from `arff` module

Netcdf Through `scipy.io.netcdf`

`netcdf_file(filename[, mode, mmap, version])`

Statistics



Statistics

scipy.stats

Contains statistical tools and probabilistic description of random processes.

Statistics

scipy.stats

Contains statistical tools and probabilistic description of random processes.

numpy.random

Contains random number generators for various random process.

Main Stats functions

- scipy.mean()
- scipy.var()
- scipy.std()
- scipy.median()
- scipy.scoreatpercentile()
- stats.describe()
- stats.mode()
- stats.moment()

Probability Distributions

- Scipy has functions that deal with several common probability distributions.
- Currently there are 81 continuous probability distributions and 10 discrete distributions.
- These are defined in the scipy.stats sub-package.
- This package also defines several statistical functions.

Prob Distributions

Continuous PDFs

norm	Normal or Gaussian
chi2	Chi-squared
t	Student's T
uniform	Uniform

Discrete PDFs

binom	Binomial
poisson	Poisson

Working with PDFs I

There are two ways of using probability distribution functions:

- Generate a frozen distribution object and then work with the methods of this object.

```
>>> from scipy import stats
>>> N = stats.norm(loc=1, scale=0.5)
```

We can then draw random numbers that follow the distribution we just defined:

```
>>> N.rvs(10)
array([ 1.26041313,  2.05286423,  0.50953812,  0.83991445,
        0.59828645,  0.90758433,  0.94395294,  1.13686641,
```

Working with PDFs II

Alternatively:

- Use functions in the appropriate class by always passing the parameters that define the distribution, when calling functions associated with the distribution.
- For example, to draw a random number from a Gaussian or Normal distribution with mean = 2 and standard deviation = 0.2 we can write:

```
>>> from scipy import stats
>>> stats.norm.rvs(loc=2, scale=0.2, size=3)
array([ 1.7615373 ,  1.91174333,  2.18555173])
```

stats.describe()

```
>>> from scipy import stats
>>> R = stats.norm.rvs(loc = 1, scale=0.5, size=1000)
>>> n, min_max, mean, var, skew, kurt = stats.describe(R)
>>> print("Number of elements: {0:d}".format(n))
Number of elements: 1000
>>> print("Minimum: {0:8.6f} Maximum: {1:8.6f}".format(min_max[0], min_
Minimum: -0.302407 Maximum: 2.508948
>>> print("Mean: {0:8.6f}".format(mean))
Mean: 0.985060
>>> print("Variance: {0:8.6f}".format(var))
Variance: 0.247292
>>> print("Skew : {0:8.6f}".format(skew))
Skew : 0.058949
>>> print("Kurtosis: {0:8.6f}".format(kurt))
Kurtosis: -0.237082
```

Working with PDFs III

Similarly, the value of the PDF at any value of the variate can be obtained using the function pdf of the concerned distribution,

```
>>> stats.norm(1,loc=2,scale=0.2)
<scipy.stats.distributions.rv_frozen object at 0x30f2ad0>
```

We can also pass an array of values to this function, to get the PDF at the specified values of the variate:

```
>>> N = stats.norm(loc=2, scale=0.2)
>>> N.pdf([-1,2])
array([  2.76535477e-49,   1.99471140e+00])
```

Multivariate random processes

Multivariate Random Processes

Are provided by the `np.random.multivariate` family.

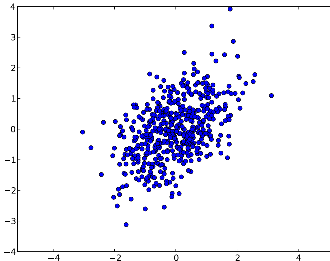
Could you create and plot a multivariate normal with:

$$\vec{\mu} = (0, 0) \tag{1}$$

$$\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \tag{2}$$

Solution

```
mean = [0,0]
cov = [[1,0.5],[0.5,1]]
import matplotlib.pyplot as plt
x,y = \
    np.random.multivariate_normal(\
        mean,cov,500).T
plt.plot(x,y,'bo')
plt.axis('equal')
```



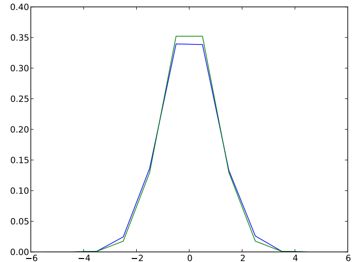
Density Estimation

Let's generate a random process and estimate its probability density function (PDF):

```
>>> x = np.random.normal(size=2000)
>>> cuts = np.arange(-6,6)
>>> cuts
array([-6, -5, -4, -3, -2, -1,  0,  1,  2,  3,  4,  5])
>>> hist = np.histogram(x, bins=cuts, normed=True)[0]
>>> bins = (cuts[1:] + cuts[:-1])/2.
>>> bins
array([-5.5, -4.5, -3.5, -2.5, -1.5, -0.5,  0.5,  1.5,  2.5,  3
```

Density Estimation

```
from scipy import stats
import matplotlib.pyplot as pl
x_pdf = stats.norm.pdf(bins)
pl.plot(bins,hist)
pl.plot(bins,x_pdf)
```



Exercise

Exercise

Generate a realization of 1000 samples following a Poisson distribution with a parameter of your choice.

- Search for the Poisson methods documentation. Can you estimate the parameter of your own distribution?

Solution

Poisson Probability Mass Distribution

$$f_{Poiss}(\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \quad (3)$$

$$\lambda = E(X) \quad (4)$$

```
>>> from pylab import plot,show,hist,figure,title
>>> from scipy.stats import poisson
>>> mu = 2.4
>>> R = poisson.rvs(mu, loc=0, size=1000)
>>> print("Mean: {0:1.2f}".format( R.mean() ) )
Mean: 2.45
```

Note that there also SciPy variants, `scipy.mean()`, `scipy.std()`.

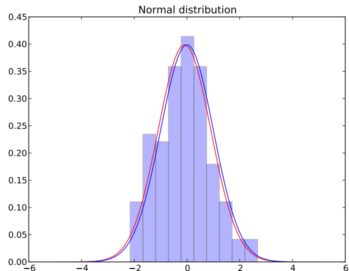
Fitting

Distribution fitting is the procedure of selecting a statistical distribution that best fits to a dataset generated by some random process. In this post we will see how to fit a distribution using the techniques implemented in the Scipy library.

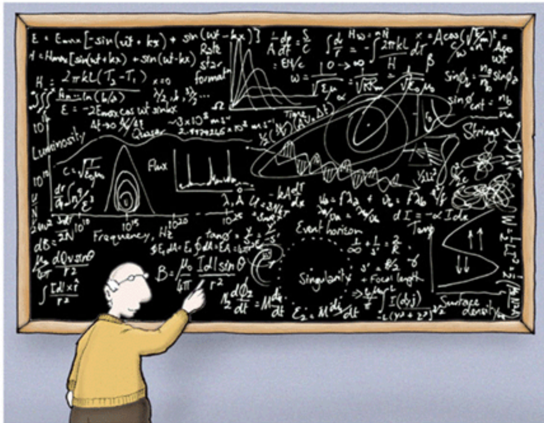
```
>>> from scipy.stats import norm
>>> from numpy import linspace
>>> from pylab import plot,show,hist,figure,title
>>>
>>> data = norm.rvs(loc=0, scale=1, size=150)
>>> param = norm.fit(data)
>>> param
(-0.09517336694399691\u200b8, 1.0003376318003636)
>>> x = linspace(-5,5,100)
>>> pdf_fitted = norm.pdf(x, loc=param[0], scale=param[1])
>>> pdf = norm.pdf(x)
```

Fitting

```
title('Normal distribution')
plot(x,
    pdf_fitted,'r-',
    x,pdf,'b-')
hist(data,normed=1,alpha=.3)
```



Statistics made easy



STATISTICS MADE EASY

Contingency tables

```
>>> from scipy.stats.contingency import expected_freq
>>> obs = np.array([[10,20,10],[20,20,10]])
>>> obs
array([[10, 20, 10],
       [20, 20, 10]])
```

χ^2-test

This function computes the chi-square statistic and p-value for the hypothesis test of independence of the observed frequencies in the contingency table observed.

```
>>> from scipy import stats
>>> stats.chi2_contingency(obs)
(2.2499999999999991, 0.32465246735834991, 2, array([[ 13.3333333
       [ 16.66666667,  22.22222222,  11.11111111]]))
```

Fisher exact test

Example

Say we spend a few days counting whales and sharks in the Atlantic and Indian oceans. In the Atlantic ocean we find 8 whales and 0 shark, in the Indian ocean 2 whales and 5 sharks. Then our contingency table is:

	Atlantic	Indian
whales	8	2
sharks	0	5

```
>>> oddsratio, pvalue = stats.fisher_exact([[8, 2], [1, 5]])
>>> pvalue
0.034965034965034919
```

t-test

ttest_ind()

Calculates the T-test for the means of TWO INDEPENDENT samples of scores.

```
>>> rvs1 = stats.norm.rvs(loc=5,scale=10,size=500)
>>> rvs2 = stats.norm.rvs(loc=5,scale=10,size=500)
>>> stats.ttest_ind(rvs1,rvs2)
(-0.24783971064054422, 0.80430941028957727)
>>> rvs3 = stats.norm.rvs(loc=7,scale=10,size=500)
>>> stats.ttest_ind(rvs1,rvs3)
(-2.4139498667262616, 0.015959786513326326)
```

t-test (matched)

ttest_ind()

Calculates the T-test for the means of TWO MATCHED samples.

```
>>> rvs1 = stats.norm.rvs(loc=5,scale=10,size=500)
>>> rvs2 = (stats.norm.rvs(loc=5,scale=10,size=500) +
...     stats.norm.rvs(scale=0.2,size=500))
...
>>> stats.ttest_rel(rvs1,rvs2)
(0.24126776434211311, 0.745977870811666)
>>> rvs3 = (stats.norm.rvs(loc=8,scale=10,size=500) +
...      stats.norm.rvs(scale=0.2,size=500))
...
>>> stats.ttest_rel(rvs1,rvs3)
(-2.9250238643536557, 0.0036010776543372864)
```

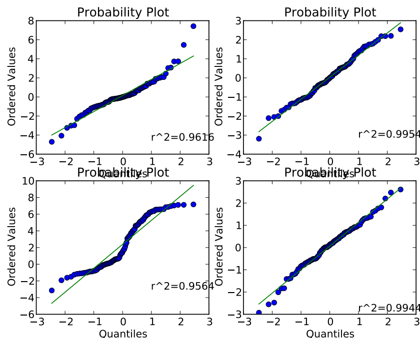
More tests

- **mannwhitneyu()** Computes the Mann-Whitney rank test on samples x and y.
- **spearmanr** Calculates a Spearman rank-order correlation coefficient and the p-value.
- **pearsonr** Calculates a Pearson correlation coefficient and the p-value for testing.
- **f_oneway** Performs a 1-way ANOVA.
- **oneway** Test for equal means in two or more samples from the normal distribution.
- **normaltest** Tests whether a sample differs from a normal distribution.
- **kruskal** Compute the Kruskal-Wallis H-test for independent samples.

Plot tests

probplot Calculate quantiles for a probability plot against a theoretical distribution.

```
import scipy.stats as stats
import matplotlib.pyplot as plt
nsample = 100
np.random.seed(7654321)
ax1 = plt.subplot(221)
x = stats.t.rvs(3, size=nsample)
res = stats.probplot(x, plot=plt)
ax2 = plt.subplot(222)
x = stats.t.rvs(25, size=nsample)
res = stats.probplot(x, plot=plt)
ax3 = plt.subplot(223)
x = stats.norm.rvs(loc=[0,5], scale=[1,1.5],
     size=(nsample/2.,2)).ravel()
res = stats.probplot(x, plot=plt)
ax4 = plt.subplot(224)
x = stats.norm.rvs(loc=0, scale=1, size=nsample)
res = stats.probplot(x, plot=plt)
```



scipy.linalg

The `scipy.linalg` module provides standard linear algebra operations, relying on an underlying efficient implementation (BLAS, LAPACK). Some of the main functions are:

- **inv** Compute the inverse of a matrix.
- **pinv** Compute the (Moore-Penrose) pseudo-inverse of a matrix.
- **solve** Solve the equation $A \cdot X = B$ for X.
- **det** Compute the determinant of a matrix.
- **norm** Matrix or vector norm.

scipy.linalg

- **eig** Solve an ordinary or generalized eigenvalue problem of a square matrix (`eigh()` for complex).
- **svd** Singular Value Decomposition.
- **orth** Construct an orthonormal basis for the range of A using SVD.
- **qr** Compute QR decomposition of a matrix.
- **expm** Compute the matrix exponential using Pade approximation.
- **logm** Compute matrix logarithm.
- **sinm** Matrix sin/cos/tan.
- **cosm** e.g. $cos(A) = I - \frac{1}{2!}A^2 + \frac{1}{4!}A^4 - \cdots$
- **tan**
- **funm** Evaluate a matrix function specified by a callable.

solve()

$A \cdot x = b$

Given a and b, solve for x.

```
>>> from scipy import linalg
>>> from numpy import dot
>>> a = np.array([[3,2,0],[1,-1,0],[0,5,1]])
>>> b = np.array([2,4,-1])
>>> x = linalg.solve(a,b)
>>> x
array([ 2., -2.,  9.])
>>> np.dot(a, x)
array([ 2.,  4., -1.])
```

pinv()

pinv()

Calculate a generalized inverse of a matrix using a least-squares solver.

```
>>> a = np.random.randn(5, 3)
>>> B = linalg.pinv(a)
>>> np.allclose(a, dot(a, dot(B, a)))
True
>>> np.allclose(B, dot(B, dot(a, B)))
True
```

numpy's allclose()

numpy.allclose()

Returns True if two arrays are element-wise equal within a tolerance.

$$abs(a - b) \leq atol + rtol \cdot abs(b) \tag{5}$$

Relative difference: $rtol \cdot abs(b)$.

rtol Defaults to $1e^{-5}$.

atol Defaults to $1e^{-8}$.

Singular Value Decomposition

Matrix Decompositions

SVD is commonly used in statistics and signal processing. Many other standard decompositions (QR, LU, Cholesky, Schur), as well as solvers for linear systems, are available in scipy.linalg.

```
>>> Mat = np.arange(9).reshape((3, 3)) + np.diag([1, 0, 1])
>>> uMat, S, vMat = linalg.svd(Mat)
>>> S
array([ 14.88982544,   0.45294236,   0.29654967])
```

Singular Value Decomposition

Matrix Decompositions

SVD is commonly used in statistics and signal processing. Many other standard decompositions (QR, LU, Cholesky, Schur), as well as solvers for linear systems, are available in scipy.linalg.

```
>>> Mat = np.arange(9).reshape((3, 3)) + np.diag([1, 0, 1])
>>> uMat, S, vMat = linalg.svd(Mat)
>>> S
array([ 14.88982544,   0.45294236,   0.29654967])
```

```
>>> sMat = np.diag(S)
>>> recMat =  uMat.dot(sMat).dot(vMat)
>>> np.allclose(recMat,Mat)
True
```

Fast Fourier Transforms

Ok, up to this, boring algebra, I need some action!!

Fast Fourier Transforms

Ok, up to this, boring algebra, I need some action!!

FFT through scipy.fftpack

The scipy.fftpack module allows to compute fast Fourier transforms. In this example we will:

- Generate a noisy signal.
- Detect a high frequency component (noise).
- Filter this noise in Fourier.
- Plot the filtered signal.

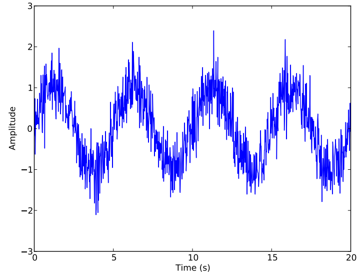
Signal Generation

A typical noisy input may look like the following:

```
time_step = 0.02
period = 5.0
time_vector = np.arange(0, 20, time_step)
signal = np.sin(2.0 * np.pi * time_vector / period) + \
 0.4 * np.random.randn(time_vector.size)
```

Signal Generation

```
plt.plot(time_vector, signal)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
```



FFT

For convenience, we need to first define a vector with the discrete Fourier Transform sample frequencies:

```
>>> from scipy import fftpack
>>> freqs = fftpack.fftfreq(signal.size, d=time_step)
>>> freqs[0:5]
array([ 0.  ,  0.05,  0.1 ,  0.15,  0.2 ])
>>> freqs[-5:-1]
array([-0.25, -0.2 , -0.15, -0.1 ])
```

And the transformation itself:

```
>>> sig_fft = fftpack.fft(signal)
```

FFT

We can find the peak on the signal as follows. First select positive frequencies.

```
>>> ind = np.where(freqs > 0 )
>>> freqs_p = freqs[ind]
>>> signal_abs = np.abs(sig_fft)[ind]
```

Then, where do we find the maximum amplitude ?

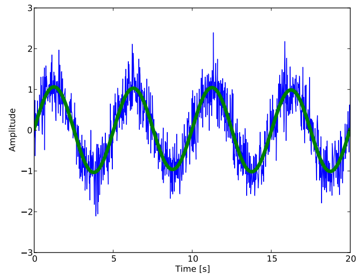
```
>>> fpeak = freqs_p[signal_abs.argmax()]
>>> print fpeak, 1./period
0.2 0.2
```

FFT

Let's filter the noise above the signal and compute the inverse transform:

```
sig_fft[np.abs(freqs) > fpeak] = 0
main_signal = fftpack.ifft(sig_fft)
plt.figure()
plt.plot(time_vector, signal)
plt.plot(time_vector,
    main_signal, linewidth=5)
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```

```
plt.show()
```



Challenge!

Let's consider this noisy image. https://www.dropbox.com/s/a73c0lla7qdjqiy/orionnebulaN.jpg Download link

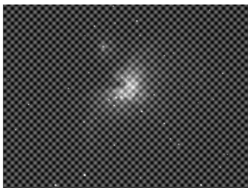


Figure: Picture with periodic noise.

Use the 2D FFT implementation in *scipy.fftpack* to remove the noise int the picture:

1. Import the image into ipython through `plt.imread()`
2. Compute the power spectrum of the Fourier Transform and plot it.
3. Cut the high-frequency part zeroing the 2D Fourier Transform matrix.
4. Apply the inverse Fourier transform to retrieve the original image.

Optimization

Optimization

Optimization is the problem of finding a numerical solution to a minimization or equality.

Optimization

Optimization

Optimization is the problem of finding a numerical solution to a minimization or equality.

scipy.optimize

The scipy.optimize module provides useful algorithms for:

1. Function minimization (scalar or multi-dimensional)
2. Curve fitting.
3. Root finding.

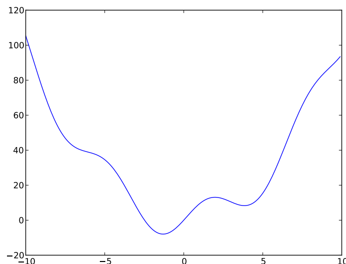
Finding a Minimum for a Scalar function

Let's define the following function:

$$f(x) = x^2 + 10sin(x) \tag{6}$$

```
def f(x):
    return x**2 + 10*np.sin(x)

x = np.arange(-10, 10, 0.1)
plt.plot(x, f(x))
```



Finding a Minimum for a Scalar Function

The general and efficient way to find a minimum for this function is to conduct a gradient descent starting from a given initial point. The BFGS algorithm is a good way of doing this:

```
>>> from scipy import optimize
>>>
>>> optimize.fmin_bfgs(f, 0)
Optimization terminated successfully.
         Current function value: -7.945823
         Iterations: 5
         Function evaluations: 24
         Gradient evaluations: 8
array([-1.30644003])
```

Brute Force Optimization

But, watch out!

```
optimize.fmin_tnc(f, 5, disp=0)
array([4.60643939])
```

Brute Force Optimization

But, watch out!

```
optimize.fmin_tnc(f, 5, disp=0)
array([4.60643939])
```

In case there is no information on the neighborhood - and therefore we have no clues on where to set up the initialization - we might need to search for a global minimum through brute force.

```
>>> grid = (-10, 10, 0.1)
>>> xmin_global = optimize.brute(f, (grid,))
>>> xmin_global
array([-1.30641113])
```

Optimization

In practical use `scipy.optimize.brute()` is not usable. There are more advanced alternatives in other functions and packages.

fminboun(f,a,b) contrained to the (a, b) interval.

anneal() `scipy.optimize.anneal()` offers an alternative using simulated annealing.

fmin_cg() Conjugate gradient methods.

fmin_ncg() Newton Methods (Nelder-Mead).

fmin Gradient-less methods

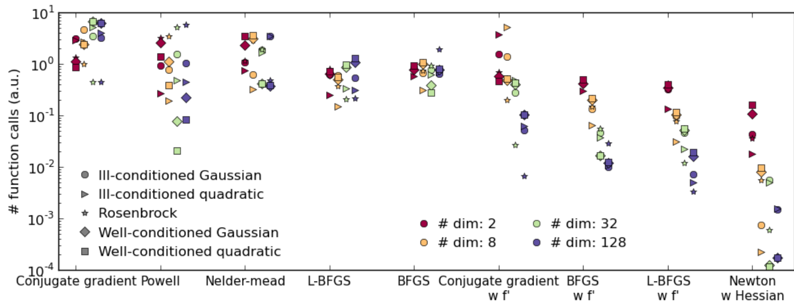
Packages OpenOpt

IPOPT

PyGMO

PyEvolve

Optimization



Source

Finding roots

Defined as the $x|_{f(x)=0}$. Roots are found with help of `fsolve`. For the case of $f(x) = x^2 + 10sin(x)$:

```
>>> optimize.fsolve(f, 1)
array([ 0.])
```

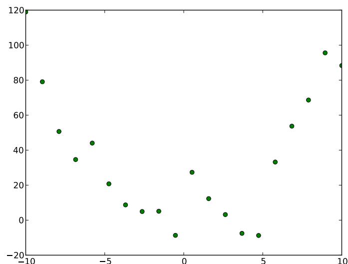
As seen from the plot of the function in the previous slides, there may exist more than one root. The root we find depends solely on the intial guess.

```
>>> optimize.fsolve(f, -2.5)
array([-2.47948183])
```

Curve Fitting

Assume we observe our process that follows the function $f(x)$, but we have some noise to our measurements.

```
x = np.linspace(-10, 10, num=20)
obs = f(x) + \
    10* np.random.randn(x.size)
```



Curve Fitting

Aha! We suspect that our process follows :

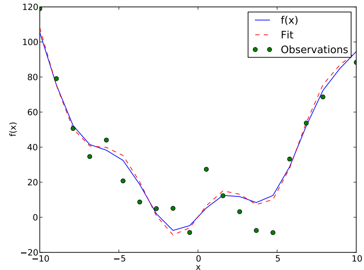
```
>>> def fguess(x, a, b):
...     return a*x**2 + b*np.sin(x)
...
```

So we can try to fit a and b parameters.

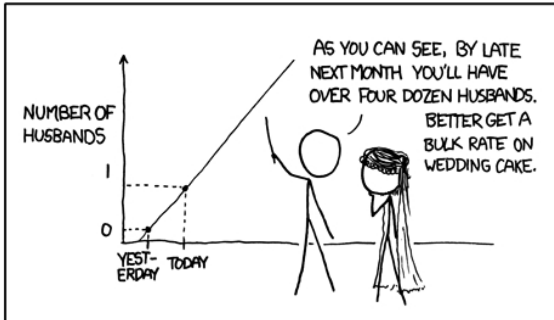
```
>>> init_pars=[2,2]
>>> params, params_covariance = optimize.curve_fit(fguess,
...     x, obs, init_pars)
...
>>> params
array([  1.01163503,  12.56131525])
>>>
```

Curve Fitting

```
plt.plot(x, f(x), 'b-',
    label="f(x)")
plt.plot(x, fguess(x, *params),
     'r--', label="Fit")
plt.plot(x,obs,'go',
    label='Observations')
plt.legend()
plt.xlabel('x')
plt.ylabel('f(x)')
```



Curve Fiting



Interpolation

The `scipy.interpolate` is useful for fitting a function from experimental data and thus evaluating points where no measure exists. Let's observe a process with oscillatory origin.

```
>>> t = np.linspace(0, 1, 10)
>>> N = (np.random.random(10)*2 - 1) * 1e-1
>>> obs = np.sin(2 * np.pi * t) + N
```

We can use the interpolate classes for building a linear "*interpolator*".

```
>>> from scipy.interpolate import interp1d
>>> interpolator = interp1d(t, obs)
```

Then we can use this object to evaluate our new data.

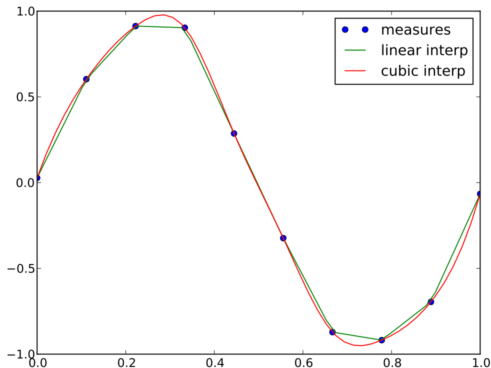
```
>>> int_t = np.linspace(0, 1, 50)
>>> int_obs = interpolator(int_t)
```

Challenge

Challenge

Try to interpolate the same function with a cubic interpolation. Plot the original function, the result of the linear and cubic interpolation and the original observations.

Interpolation



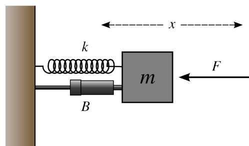
Integration

There exists a generic integration routune: `scipy.integrate.quad()`.

```
>>> from scipy.integrate import quad
>>> res, err = quad(np.sin, 0, np.pi/2)
>>> res,err
(0.9999999999999999, 1.1102230246251564e-14)
```

odeint() General-purpose integrator using LSODA (Livermore Solver for Ordinary Differential equations (from ODEPACK Library).

Integrator example



A damped spring-mass oscillator (2nd order oscillator). The dampig effect is linearly related to the velocity of the oscillations.

$$\frac{d^2x}{dt^2} + 2\zeta\omega_0\frac{dx}{dt} + \omega_0^2 x = 0 \tag{7}$$

where $\omega_0^2 = k/m$, k the spring constant, m the mass and $\zeta = \frac{c}{2m\omega_0}$ with c as damping coefficient.

Integrator example

The damping ratio $\zeta = \frac{c}{2m\omega_0} = \frac{c}{2m\sqrt{\frac{k}{m}}}$ determines:

Overdamped ($\zeta > 1$) The system returns to equilibrium without oscillating (exponencially decaying). Larger values of the damping ratio ζ return to equilibrium more slowly.

Critical damp ($\zeta = 1$) The system returns to equilibrium as quickly as possible without oscillating.

Underdamped ($0 < \zeta < 1$) The system oscillates (at reduced frequency compared to the undamped case) with the amplitude gradually decreasing to zero.

Undamped ($\zeta = 0$) The system oscillates at ω_0.

The values

```
>>> m, k, c = 0.5, 4, 0.4  # In kg, N/m, Ns/M
>>> c / (2 * m * np.sqrt(k/m))
0.1414213562373095
```

Integrator example

We will use scipy's `integrate.odeint()`. We need to transform the second order system into two first order equations for $Y = (y, \dot{y})$. Let's define $\nu = 2\zeta\omega_0 = \frac{c}{m}$ and $o = \omega_0^2 = \frac{k}{m}$:

```
>>> nu, o  = c / m , k / m
```

Then, we can express $Y = (y, \dot{y})$:

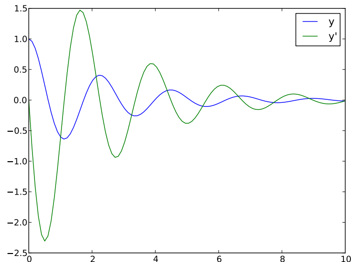
$$y = \dot{x} \tag{8}$$

$$\dot{y} = -\nu\dot{x} - ox \tag{9}$$

```
>>> from scipy.integrate import odeint
>>> def dy(y, t, nu, o):
...     return (y[1], -nu * y[1] - o  * y [0])
...
>>> time_vec = np.linspace(0, 10, 100)
>>> yarr = odeint(dy, (1, 0), time_vec, args=(nu, o))
```

Integrator Example

```
pl.plot(time_vec,
    yarr[:, 0], label='y')
pl.plot(time_vec,
    yarr[:, 1], label="y'")
pl.legend()
```



Signal Processing

scipy.signal

This module includes a large number of functions for signal processing. Covering the following areas:

Convolution

- convolve()
- fftconvolve()
- correlate()

Waveforms

- chirp()
- square()
- sawtooth()

Wavelets

- daub()
- cwt()

b-Splines

- bspline()
- spline_filter()

Peak Finding

- find_peaks_cwt()

Signal Processing at scipy.signal

Filter Design

- firwin()
- freqz()
- freqs()
- iirdesign()
- iirfilter()
- kaiserord()
- remez()
- butter()
- buttord()
- cheb1()
- cheb1ord()

Filtering

- order_filter()
- medfilt()
- wiener()
- decimate()
- resample()
- detrend()
- get_window()
- lfilter()

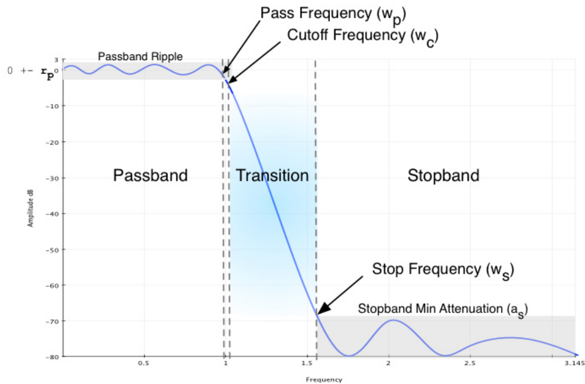
Challenge

15 minutes

Use scipy.signal for:

1. Create a linear signal.
2. Add a random noise process.
3. *detrend* the signal.

Low Pass Filters



Low Pass Filters

Parameters

ω_p Passband. This is the frequency range which we desire to let the signal through with minimal attenuation. In the scipy functions this is in normalized frequency, $1 > \omega_p > 0$, where 1 is the Nyquist frequency.

ω_s Stopband. This is the frequency range which the signal should be attenuated $1 > \omega_s > 0$.

$R_p, gpass$ The max variation in the passband, in decibels.

$A_s, gstop$ The min attenuation in the stopband, in decibels.

Notes: The cutoff frequency is the -3dB point. If the cutoff frequency is required the algorithm will work to meet the -3dB point at the ω_c frequency.

ω_p is the pass frequency, this is the last point were -gpass (R_p)

IIR filter design

A number of filters are available through the `iirdesign()` function:

Filter	Transition	Passband	Stopband	Phase	Comments
Bessel	Knee? What knee?	Monotonic	Monotonic	Near-linear	s-to-z mappings distort phase. FIR usually more efficient for linear phase
Butterworth	Rounded	Maximally flat, mono- tonic	Monotonic	nonlinear near cutoff	Easy to design by had Maple syrup is better on waffles
Chebychev I	Sharp	Ripples	Monotonic	Worse	Easy to design by hand
Chebychev II	Sharp	Monotonic	Ripples	Worse	Somewhat more complicated design than Chebychev I
Elliptic	Maximally sharp	Ripples	Ripples	Drunk fly on cross- country skies in a tor- nado	Not viable for de- sign by hand

Verbatim from *Grover and Deller, Digital Signal Processing and the Microcontroller*

```
iirdesign(Wp, Ws, Rpl, Asl, ftype='butter')
```

Example FIR design

Let's define two convenience plots.

```
def mfreqz(b,a=1):
    w,h=signal.freqz(b,a)
    h_dB=20*log10(abs(h))
    subplot(211)
    plot(w/max(w),h_dB)
    ylim(-150, 5)
    ylabel('Magnitude (db)')
    xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
    title(r'Frequency response')
    subplot(212)
    h_Phase = unwrap(arctan2(imag(h),real(h)))
    plot(w/max(w),h_Phase)
    ylabel('Phase (radians)')
    xlabel(r'Normalized Frequency (x$\pi$rad/sample)')
    title(r'Phase response')
    subplots_adjust(hspace=0.5)
```

Example FIR design

```
def impz(b,a=1):
    impulse = repeat(0.,50); impulse[0] =1.
    x = arange(0,50)
    response = signal.lfilter(b,a,impulse)
    subplot(211)
    stem(x, response)
    ylabel('Amplitude')
    xlabel(r'n (samples)')
    title(r'Impulse response')
    subplot(212)
    step = cumsum(response)
    stem(x, step)
    ylabel('Amplitude')
    xlabel(r'n (samples)')
    title(r'Step response')
    subplots_adjust(hspace=0.5)
```

Example Low Pass FIR design

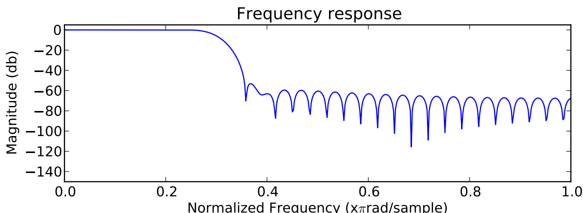
Low Pass Filter

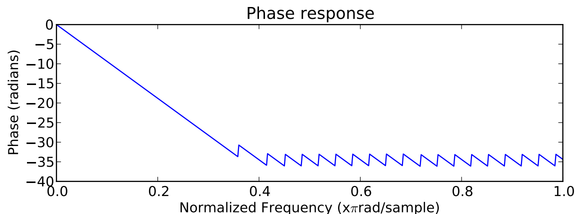
For designing lowpass FIR filters you can use the function `signal.firwin`. Define the window length, cut off frequency and the window:

```
>>> from scipy import signal
>>> from numpy import log10
>>> from pylab import *
>>> n = 61
>>> a = signal.firwin(n, cutoff = 0.3, window = "hamming")
>>> mfreqz(a)
```

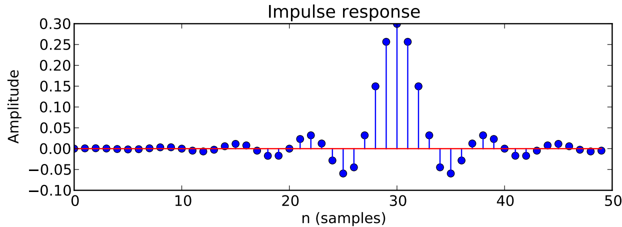
Example FIR design

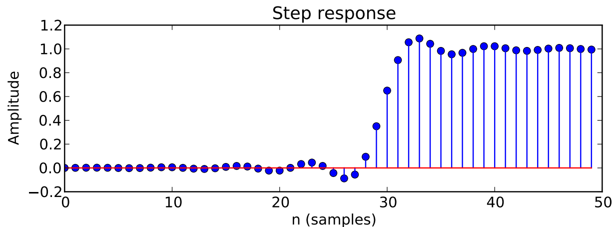
a = signal.firwin(n, cutoff = 0.3, window = "hamming")





Example FIR design





scipy.ndimage

scipy.ndimage

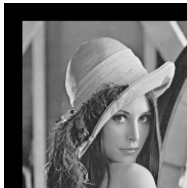
This package contains various functions for multi-dimensional image processing, mainly organized in four function groups:

- Filters (convolve, correlate, ...)
- Fourier Filters (Gaussian fourier filters, ...)
- Interpolation (affine_transform, rotate, ...)
- Measurements (histogram, extrema, ...)
- Morphology (closings, openings, ...)
- Utility (imread)

Geometrical transformations on images

```
>>> from scipy import ndimage
>>> from scipy import misc
>>> lena = misc.lena()
>>> shifted_lena = ndimage.shift(lena, (50, 50))
>>> shifted_lena2 = ndimage.shift(lena, (50, 50), mode='nearest
>>> rotated_lena = ndimage.rotate(lena, 30)
>>> cropped_lena = lena[50:-50, 50:-50]
>>> zoomed_lena = ndimage.zoom(lena, 2)
>>> zoomed_lena.shape
(1024, 1024)
```

Geometrical transformations on images



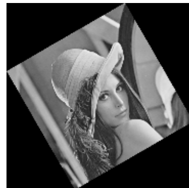
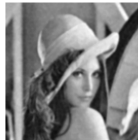
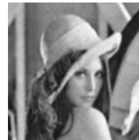


Image Filtering

```
from scipy import misc
import numpy as np
from scipy import signal
lena = misc.lena()
noisy_lena = np.copy(lena).astype(np.float)
noisy_lena += lena.std()*0.5*\
    np.random.standard_normal(lena.shape)
blurred_lena = ndimage.gaussian_filter(noisy_lena, sigma=3)
median_lena = ndimage.median_filter(blurred_lena, size=5)
wiener_lena = signal.wiener(blurred_lena, (5,5))
```

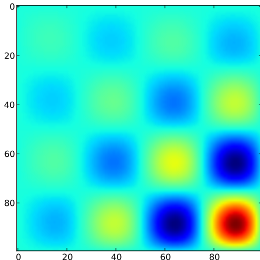
Image Filtering

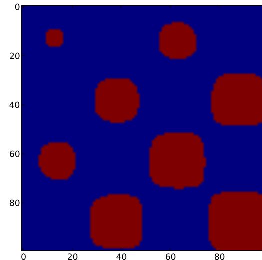
 

Measurements

Let us first generate a nice synthetic binary image.

```
x, y = np.indices((100, 100))
sig = np.sin(2*np.pi*x/50.)*\
    np.sin(2*np.pi*y/50.)*(1+x*y/50.**2)**2
mask = sig > 1
```



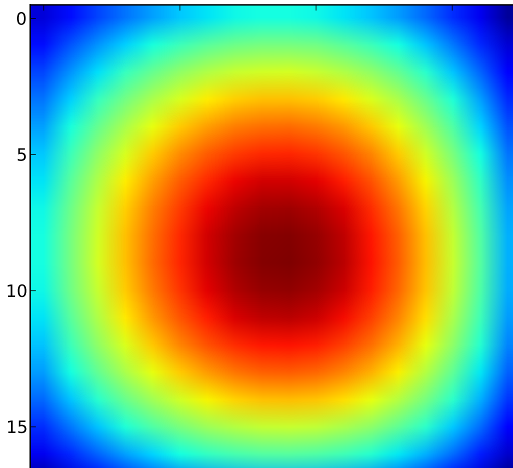


Measurements

Play with the following code, plot sig, and the labels:

```
>>> labels, nb = ndimage.label(mask)
>>> areas = ndimage.sum(mask, labels, xrange(1, labels.max()+1))
>>> areas
array([ 190.,   45.,  424.,  278.,  459.,  190.,  549.,  424.])
>>> maxima = ndimage.maximum(sig, labels, xrange(1, labels.max()
>>> maxima
array([  1.80238238,   1.13527605,   5.51954079,   2.49611818,
         6.71673619,   1.80238238,  16.76547217,   5.51954079])
>>> ndimage.find_objects(labels==4)
[(slice(30L, 48L, None), slice(30L, 48L, None))]
>>> sl = ndimage.find_objects(labels==4)
>>> import pylab as pl
>>> pl.imshow(sig[sl[0]])
<matplotlib.image.AxesImage object at 0x3ad5290>
```

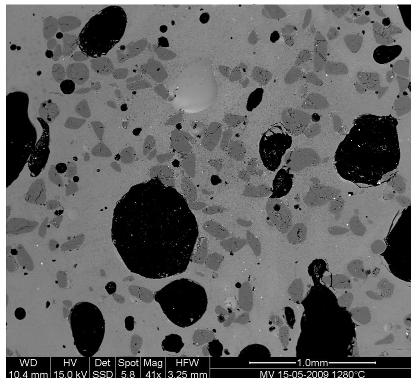
Measurements



Challenge

https://www.dropbox.com/s/cgc14z3rafshjho/MV_HFV_012.jpg
Download Image.

Download the following image Download Image.
This file shows a Scanning Element Microscopy image of glass sample (light gray matrix) with some bubbles (on black) and unmolten sand grains (dark gray). Our goal is to determine the fraction of the sample covered by these three phases, and to estimate the typical size of sand grains and bubbles, their sizes, etc.



Challenge

1. Open the image.
2. Crop unwanted segments.
3. Filter the image with a median filter.
4. Check the effect of the filter on the histogram.
5. From the histogram. Set thresholds for:
   - Sand pixels.
   - Glass pixels.
   - Bubble Pixels.
6. Display an image with the colored elements.
7. Could you estimate the mean size of bubbles?

Special Functions

There is a rich library for computing Special Functions, brought by `scipy.special`, main functions are:

- Elliptic Funs `sllipj()`, `ellipj()`, ...
- Bessel Funs `jn()`, `jv()`,`jve()`, ...
- Statistical Funs `btdtr()`, . It is better to use `scipy.stats` functions.
- Gamma Funs `gamma()`, `multigamma()`, ...
- Error Funs `erf()`, `erfc`, ...
- Legendre Funs `lpmv`, `legendre()`, ...
- Hypergeom. `hypf1()`, ...
- ++ http://docs.scipy.org/doc/scipy/reference/special.htmlAnd more...

A *dense matrix* is a matematical object for data storage of a 2D array ofvalues.

- Memory is allocated once for all items
- Storage in a contiguous chunk (aka NumPy ndarray)
- The access to individual items is fast.

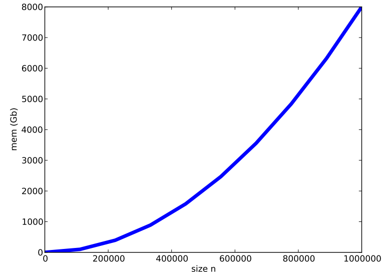
Why Space Matrices?

It's this memory thing... Imagine adjacency matrices for:

- All Graph theory.
- 40000 proteins in a typical PPI dataset.
- $10e^7$ Facebook users in a typical country.
- Partial Differential Equations (PDEs), Finite Elements and others.

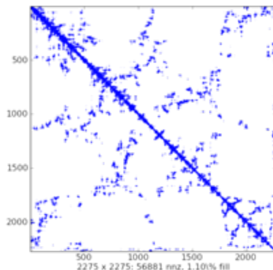
Grow, my son, grow...

```
import numpy as np
import matplotlib.pyplot as plt
e = np.linspace(0,1e6,10)
v = 8 * (e**2)/1e9
plt.plot(e,v,lw=5)
plt.xlabel('size n')
plt.ylabel('mem (Gb)')
```



Sparse Matrix

- Sparse Matrix is an almost empty matrix.
- If a zero means nothing, let's store nothing.
- It's a form of compression, huge memory savings.
- *Enables* applications.



Storage Schemes

There are seven storage schemes offered by `scipy.sparse`:

- **csc_matrix** Compressed Sparse Column format.
- **csr_matrix** Compressed Sparse Row format.
- **bsr_matrix** Block Sparse Row format.
- **lil_matrix** Lists of Lists format.
- **dok_matrix** Dictionary of Keys format.
- **coo_matrix** COOrdiante format (X_{ijk}).
- **dia_matrix** DIAgonal Matrix format.

spmatrix

object All `scipy.sparse` classes are subclasses of `spmatrix`.

- Default implementation of arithmetic ops.
- matrix/NumPy: toarray(),todense()

Attributes

mtx.A	toarray().
mtx.T	Transpose.
mtx.H	Hermitian transpose.
mtx.real	Real part of complex matrix.
mtx.imag	Imaginary part of complex matrix.
mtx.size	Non-zero size.
mtx.shape	The number of rows/columns.

storage In form of NumPuy arrays.

COO matrix challenge

Check the documentation of sparse.coo_matrix. Create a sparse matrix M so that it looks like the following

```
>>> M.todense()
matrix([[4, 0, 9, 0],
        [0, 7, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 5]])
```

Can you slice this type of matrix?

Solvers

SuperLU 4.0

- Included in SciPy.
- Real and Complex domains.
- Single and double precision.

umfpack

UMFPACK is a set of routines for solving unsymmetric sparse linear systems

- Real and Complex domains.
- Double precision.
- Fast.
- See `scikits.umfpack` and `scikits.suitesparse`.

Example

```
>>> import numpy as np
>>> from scipy import sparse
>>> mtx = sparse.spdiags([[1, 2, 3, 4, 5], [6, 5, 8, 9, 10]], [0
>>> mtx.todense()
matrix([[ 1,  5,  0,  0,  0],
        [ 0,  2,  8,  0,  0],
        [ 0,  0,  3,  9,  0],
        [ 0,  0,  0,  4, 10],
        [ 0,  0,  0,  0,  5]])
>>> rhs = np.array([1, 2, 3, 4, 5])
```

Example

```
>>> from scipy.sparse.linalg import dsolve
>>> mtx1 = mtx.astype(np.float32)
>>> x = dsolve.spsolve(mtx1, rhs, use_umfpack=False)
>>> x
array([ 106. ,  -21. ,    5.5,   -1.5,    1. ], dtype=float32)
>>> mtx1*x
array([ 1.,  2.,  3.,  4.,  5.], dtype=float32)
```

Eigensolvers

eigen module

arpack A colection of Fortran77 subroutines to solver large scale eigenvalue problems.

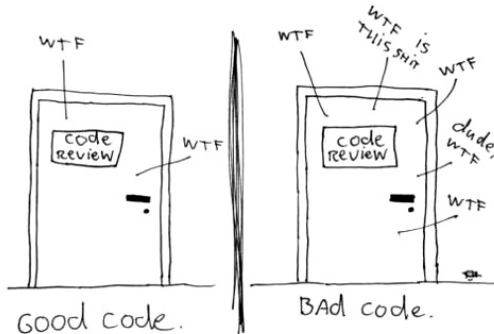
lobpcg Locally Optimal Block Preconditioned Conjugate Gradient Method. See also the PyAMG module.

PyAMG PyAMG is a library of Algebraic Multigrid (AMG) solvers. Check http://code.google.com/p/pyamg/

Pysparse

Pysparse is a fast sparse matrix library for Python. It provides several sparse matrix storage formats and conversion methods. It also implements a number of iterative solvers, preconditioners, and interfaces to efficient factorization packages.

Writing code



Coding in Science

- Things should be clever, but not too clever.
- Algorithms are optimal, both in speed as well as in readability.
- Classes, variables and functions are well named and make sense without having to think too much.
- You come back to it after a weekend off, and you can jump straight in.
- Things that will be reused are reusable.
- Write automated test cases
- Unit tests are easy to write.

Coding in Science

```
def p(n):
    """print 3.1415   """
    return n**2
exec(p.__doc__) # hidd
```

```
3.1415
```

Code profiling

timeit (only in ipython)

```
In [1]: import numpy as np

In [2]: a = np.arange(1000)

In [3]: %timeit a ** 2
100000 loops, best of 3: 5.73 us per loop

In [4]: %timeit a ** 2.1
1000 loops, best of 3: 154 us per loop

In [5]: %timeit a * a
100000 loops, best of 3: 5.56 us per loop
```

Profiler

This is very useful for large programs.

```
# For this example to run, you also need the 'ica.py' file

import numpy as np
from scipy import linalg
from ica import fastica

def test():
    data = np.random.random((5000, 100))
    u, s, v = linalg.svd(data)
    pca = np.dot(u[:10, :], data)
    results = fastica(pca.T, whiten=False)
test()
```

Profiler %run -t

```
In [1]: %run -t demo.py

IPython CPU timings (estimated):
    User  :     14.3929 s.
    System:    0.256016 s.
```

Profiler %run -p

```
In [2]: %run -p demo.py

      916 function calls in 14.551 CPU seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1   14.457   14.457   14.479   14.479 decomp.py:849(svd)
     1    0.054    0.054    0.054    0.054 {method 'random_sample' of '
     1    0.017    0.017    0.021    0.021 function_base.py:645(asarray
    54    0.011    0.000    0.011    0.000 {numpy.core._dotblas.dot}
     2    0.005    0.002    0.005    0.002 {method 'any' of 'numpy.ndar
```

Line profiler

The Line Profiler tells us from where our code is called! Save this program as demo.py

```
@profile
def test():
    data = np.random.random((5000, 100))
    u, s, v = linalg.svd(data)
    pca = np.dot(u[:10, :], data)
    results = fastica(pca.T, whiten=False)
```

Line Profiler

```
~ $ kernprof.py -l -v demo.py

Wrote profile results to demo.py.lprof
Timer unit: 1e-06 s

File: demo.py
Function: test at line 5
Total time: 14.2793 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     5                                           @profile
     6                                           def test():
     7         1        19015  19015.0      0.1      data = np.random.random((5000, 100))
     8         1     14242163 14242163.0   99.7      u, s, v = linalg.svd(data)
     9         1        10282  10282.0      0.1      pca = np.dot(u[:10, :], data)
    10         1         7799   7799.0      0.1      results = fastica(pca.T, whiten=False)
```

Use this Link to retrieve Kernprof.py (http://packages.python.org/line_profiler/kernprof.py)

Speed up you code

Some commonly encountered tricks to make code faster.

- Vectorizing for loops
  Avoid for loops using numpy arrays. For this, masks and indices arrays can be useful.
- Broadcasting
  Use broadcasting to do operations on arrays as small as possible before combining them
- In place operations:

```
In [1]: a = np.zeros(1e7)

In [2]: %timeit global a ; a = 0*a
10 loops, best of 3: 111 ms per loop

In [3]: %timeit global a ; a *= 0
10 loops, best of 3: 48.4 ms per loop
```

Speed up your code

- Use views, and not copies.
  Copying big arrays is as costly as making simple numerical operations on them:

```
In [1]: a = np.zeros(1e7)

In [2]: %timeit a.copy()
10 loops, best of 3: 124 ms per loop

In [3]: %timeit a + 1
10 loops, best of 3: 112 ms per loop
```

Speed up your code

Memory access is cheaper when it is grouped: accessing a big array in a continuous way is much faster than random acces. C or Fortran ordering has a strong effect on matrix access.
This example is really nice:

```
In [1]: c = np.zeros((1e4, 1e4), order='C')

In [2]: %timeit c.sum(axis=0)
1 loops, best of 3: 3.89 s per loop

In [3]: %timeit c.sum(axis=1)
1 loops, best of 3: 188 ms per loop

In [4]: c.strides
Out[4]: (80000, 8)
```

Challenge JUN

https://www.dropbox.com/s/0bvyrc1zf5x1hhk/jun.txt

```
GACATCATGGGCTATTTTTAGGGGTTGACTGGTAGCAGATAAGTGTTGAGCTCGGGCTGGATAAGGGCTC
AGAGTTGCACTGAGTGTGGCTGAAGCAGCGAGGCGGGAGTGGAGGTGCGCGGAGTCAGGCAGACAGACAG
ACACAGCCAGCCAGCCAGGTCGGCAGTATAGTCCGAACTGCAAATCTTATTTTCTTTTCACCTTCTCTCT
AACTGCCCAGAGCTAGCGCCTGTGGCTCCCGGGCTGGTGTTTCGGGAGTGTCCAGAGAGCCTGGTCTCCA
GCCGCCCCCGGGAGGAGAGCCCTGCTGCCCAGGCGCTGTTGACAGCGGCGGAAAGCAGCGGTACCCACGC
GCCCGCCGGGGGAAGTCGGCGAGCGGCTGCAGCAGCAAAGAACTTTCCCGGCTGGGAGGACCGGAGACAA
GTGGCAGAGTCCCGGAGCCAACTTTTGCAAGCCTTTCCTGCGTCTTAGGCTTCTCCACGGCGGTAAAGAC
CAGAAGGCGGCGGAGAGCCACGCAAGAGAAGAAGGACGTGCGCTCAGCTTCGCTCGCACCGGTTGTTGAA
CTTGGGCGAGCGCGAGCCGCGGCTGCCGGGCGCCCCCTCCCCCTAGCAGCGGAGGAGGGGACAAGTCGTC
GGAGTCCGGGCGGCCAAGACCCGCCGCCGGCCGGCCACTGCAGGGTCCGCACTGATCCGCTCCGCGGGGA
GAGCCGCTGCTCTGGGAAGTGAGTTCGCCTGCGGACTCCGAGGAACCGCTGCGCACGAAGAGCGCTCAGT
GAGTGACCGCGACTTTTCAAAGCCGGGTAGCGCGCGCGAGTCGACAAGTAAGAGTGCGGGAGGCATCTTA
ATTAACCCTGCGCTCCCTGGAGCGAGCTGGTGAGGAGGGCGCAGCGGGGACGACAGCCAGCGGGTGCGTG
CGCTCTTAGAGAAACTTTCCCTGTCAAAGGCTCCGGGGGGCGCGGGTGTCCCCCGCTTGCCACAGCCCTG
TTGCGGCCCCGAAACTTGTGCGCGCAGCCCAAACTAACCTCACGTGAAGTGACGGACTGTTCTATGACTG
CAAAGATGGAAACGACCTTCTATGACGATGCCCTCAACGCCTCGTTCCTCCCGTCCGAGAGCGGACCTTA
TGGCTACAGTAACCCCAAGATCCTGAAACAGAGCATGACCCTGAACCTGGCCGACCCAGTGGGGAGCCTG
AAGCCGCACCTCCGCGCCAAGAACTCGGACCTCCTCACCTCGCCCGACGTGGGGCTGCTCAAGCTGGCGT
CGCCCGAGCTGGAGCGCCTGATAATCCAGTCCAGCAACGGGCACATCACCACCACGCCGACCCCCACCCA
GTTCCTGTGCCCCAAGAACGTGACAGATGAGCAGGAGGGCTTCGCCGAGGGCTTCGTGCGCGCCCTGGCC
GAACTGCACAGCCAGAACACGCTGCCCAGCGTCACGTCGGCGGCGCAGCCGGTCAACGGGGCAGGCATGG
...
```

Challenge JUN (continued)

```
...
TGGCTCCCGCGGTAGCCTCGGTGGCAGGGGGCAGCGGCAGCGGCGGCTTCAGCGCCAGCCTGCACAGCGA
GCCGCCGGTCTACGCAAACCTCAGCAACTTCAACCCAGGCGCGCTGAGCAGCGGCGGCGGGGCGCCCTCC
TACGGCGCGGCCGGCCTGGCCTTTCCCGCGCAACCCCAGCAGCAGCAGCAGCCGCCGCACCACCTGCCCC
AGCAGATGCCCGTGCAGCACCCGCGGCTGCAGGCCCTGAAGGAGGAGCCTCAGACAGTGCCCGAGATGCC
CGGCGAGACACCGCCCCTGTCCCCCATCGACATGGAGTCCCAGGAGCGGATCAAGGCGGAGAGGAAGCGC
ATGAGGAACCGCATCGCTGCCTCCAAGTGCCGAAAAAGGAAGCTGGAGAGAATCGCCCGGCTGGAGGAAA
AAGTGAAAACCTTGAAAGCTCAGAACTCGGAGCTGGCGTCCACGGCCAACATGCTCAGGGAACAGGTGGC
ACAGCTTAAACAGAAAGTCATGAACCACGTTAACAGTGGGTGCCAACTCATGCTAACGCAGCAGTTGCAA
ACATTTTGAAGAGAGACCGTCGGGGGCTGAGGGGCAACGAAGAAAAAAAATAACACAGAGAGACAGACTT
GAGAACTTGACAAGTTGCGACGGAGAGAAAAAAGAAGTGTCCGAGAACTAAAGCCAAGGGTATCCAAGTT
GGACTGGGTTGCGTCCTGACGGCGCCCCCAGTGTGCACGAGTGGGAAGGACTTGGCGCGCCCTCCCTTGG
CGTGGAGCCAGGGAGCGGCCGCCTGCGGGCTGCCCCGCTTTGCGGACGGGCTGTCCCCGCGCGAACGGAA
CGTTGGACTTTTCGTTAACATTGACCAAGAACTGCATGGACCTAACATTCGATCTCATTCAGTATTAAAG
GGGGGAGGGGGAGGGGGTTACAAACTGCAATAGAGACTGTAGATTGCTTCTGTAGTACTCCTTAAGAACA
CAAAGCGGGGGGAGGGTTGGGGAGGGGCGGCAGGAGGGAGGTTTGTGAGAGCGAGGCTGAGCCTACAGAT
GAACTCTTTCTGGCCTGCCTTCGTTAACTGTGTATGTACATATATATATTTTTTAATTTGATGAAAGCTG
ATTACTGTCAATAAACAGCTTCATGCCTTTGTAAGTTATTTCTTGTTTGTTTGTTTGGGTATCCTGCCCA
GTGTTGTTTGTAAATAAGAGATTTGGAGCACTCTGAGTTTACCATTTGTAATAAAGTATATAATTTTTTT
ATGTTTTGTTTCTGAAAATTCCAGAAAGGATATTTAAGAAAATACAATAAACTATTGGAAAGTACTCCCC
TAACCTCTTTTCTGCATCATCTGTAGATACTAGCTATCTAGGTGGAGTTGAAAGAGTTAAGAATGTCGAT
TAAAATCACTCTCAGTGCTTCTTACTATTAAGCAGTAAAAACTGTTCTCTATTAGACTTTAGAAATAAAT
GTACCTGATGTACCTGATGCTATGGTCAGGTTATACTCCTCCTCCCCCAGCTATCTATATGGAATTGCTT
ACCAAAGGATAGTGCGATGTTTCAGGAGGCTGGAGGAAGGGGGGTTGCAGTGGAGAGGGACAGCCCACTG
AGAAGTCAAACATTTCAAAGTTTGGATTGTATCAAGTGGCATGTGCTGTGACCATTTATAATGTTAGTAG
AAATTTTACAATAGGTGCTTATTCTCAAAGCAGGAATTGGTGGCAGATTTTACAAAAGATGTATCCTTCC
AATTTGGAATCTTCTCTTTGACAATTCCTAGATAAAAAGATGGCCTTTGCTTATGAATATTTATAACAGC
ATTCTTGTCACAATAAATGTATTCAAATACCAA
```

Challenge

JUN TF gene

From the above gene, write a code that counts:

- Nuclotide frequency table.
- BiNucleotide frequency table.
- TriNucleotide frequency table.

Split your code in several functions. Use the profiler and line profiler to measure the performance of your code.
How much your code would spend to process the complete human genome ($3.5 \cdot 10^9$ Nucleotides)?

Symbolic Mathematics in Python

Sympy

SymPy is a Python library for symbolic mathematics. Implements a computer algebra system comparable with Mathematica or Maple. It has a separate website at http://sympy.org

Capabilities:

- Evaluate expressions with arbitrary precision.
- Perform algebraic manipulations on symbolic expressions.
- Perform basic calculus with symbolic expressions including limits, differentiation and integration.
- Solve polynomial and transcendental equations.
- Solve some differential equations.

Introduction

Rational class

SymPy offers the representation of a Rational class as a pair of two integers, so:

```
>>> from sympy import *
>>> r = Rational(1,2)
>>> r
1/2
>>> r.evalf()
0.50000000000000
```

Introduction

mpmath

SymPy uses mpmath.
Mpmath is a pure-Python library for multiprecision floating-point arithmetic. It provides an extensive set of transcendental functions ($f(x) = x^{\pi}$), unlimited exponent sizes, complex numbers, interval arithmetic, numerical integration and differentiation, root-finding, linear algebra, and much more.
This allows to perform computations using arbitraty precision atithmetic, including the evaluation of e, π, and the inclusion of ∞ as a symbol itself through the *oo*.

Introduction

```
>>> pi**2
pi**2
>>> pi.evalf()
3.14159265358979
>>> pi.evalf(60)
3.14159265358979323846264338327950288419716939937510582097494
>>> oo > 10000
True
>>> oo+10000
oo
```

Challenge

2 mins challenge

- Create two rationals corresponding to 1/2 and 2/3 and sum them.
- Compute the value of $\sqrt{\frac{2\pi}{3}}$ with 30 decimals.

Symbols

Symbolic variables must be declared explicitly.

```
>>> from sympy import *
>>> x = Symbol('x')
>>> y = Symbol('y')
```

And then comes the magic:

```
>>> x-y+x-y-x+y+y
x
```

Expand

Expand expands powers and multiplications:

```
>>> (x+y)**2
(x + y)**2
>>> expand((x+y)**2)
x**2 + 2*x*y + y**2
>>> expand((x)**2, complex=True)
-im(x)**2 + 2*I*im(x)*re(x) + re(x)**2
>>> expand(cos(x+y), trig=True)
-sin(x)*sin(y) + cos(x)*cos(y)
```

Simplify

simplify()

```
>>> simplify((x+x*y)/x)
y + 1
```

There are specific targeted simplify functions:

- **powsimp** Simplification of exponents.
- **trigsimp** Trigonometric expressions.
- **logcombine** $\log(x) + \log(y) = \log(xy)$ and $a\log(x) = \log(x^a)$
- **radsimp** Rationalize the denominator.
- **together** Rational expressions (*No heroic measures are taken to minimize degree of the resulting numerator and denominator.*, sic.)

challenge

2 min challenge

- Compute the expanded form of $(x + xy)^3$
- Simplify the expression $\frac{\sin(x)}{\cos(x)}$

Limits

Interestingly, limits are very easy to use in SymPy, just use the sintax:

limits

limit(function, variable, point)

Limits

```
>>> x = Symbol('x')
>>> limit(sin(x)/x, x, 0)
1
>>> limit(x, x, oo)
oo
>>> limit(1/x, x, oo)
0
>>> limit(x**x, x, 0)
1
```

Differentiation

You can differentiate any SymPy expression using:

differentiation

diff(func,var,n)

Differentiation

```
>>> x = Symbol('x')
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
2*cos(2*x)
>>> diff(tan(x), x)
tan(x)**2 + 1
>>> diff(sin(2*x), x, 3)
-8*cos(2*x)
```

Series Expansion

SymPy also knows how to compute the Taylor series of an expression at a point. Use :

series

series(expr,var)

```
>>> x = Symbol('x')
>>> series(cos(x), x)
1 - x**2/2 + x**4/24 + O(x**6)
```

Integration

Integration

SymPy has support for indefinite and definite integration of transcendental elementary and special functions via integrate() facility, which uses powerful extended Risch-Norman algorithm and some heuristics and pattern matching.

```
>>> x = Symbol('x')
>>> integrate(2*x**4,x)
2*x**5/5
>>> integrate(cos(x),x)
sin(x)
>>> integrate(log(x)*x,x)
x**2*log(x)/2 - x**2/4
>>> integrate(log(x)*x,(x,0,1))
-1/4
```

Solver

SymPy is able to solve algebraic equations, in one and several variables:

```
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> solve(x**4 - 1, x)
[1, -1, -I, I]
>>> solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
{x: -3, y: 1}
>>> solve(exp(x) + 1, x)
[I*pi]
```

Factor

For solving polinomials, also consider the use of `factor`:

```
>>> x = Symbol('x')
>>> f = x**4 - 3*x**2 + 1
>>> factor(f)
(x**2 - x - 1)*(x**2 + x - 1)
```

Challenge

2 mins challenge

Could you solve the system of equations:

$$x + y = 2 \quad (10)$$

$$2x + y = 0 \quad (11)$$

Matrix support

Matrices can be created as instances from the Matrix class:

```
>>> from sympy import Matrix
>>> Matrix([[1,0],[0,1]])
[1, 0]
[0, 1]
```

But...

```
>>> A = Matrix([[1,x],[y,1]])
>>> A.det()
-x*y + 1
>>> A**2
[x*y + 1,     2*x]
[    2*y, x*y + 1]
```

Challenge

Differential Equations

Create a Generic function f

```
>>> f = Function("f")
>>> f(x)
f(x)
>>> f(x).diff()
Derivative(f(x), x)
>>> e = Eq( f(x).diff(x,x) + 9*f(x) , 0)
>>> e
9*f(x) + Derivative(f(x), x, x) == 0
>>> dsolve(e,f(x))
f(x) == C1*cos(3*x) + C2*sin(3*x)
```

Questions