SciPy

Berian James
24 Sep 2012

Numpy for IDL or MATLAB users:
https://www.cfa.harvard.edu/~jbattat/computer/python/science/idl-numpy.html
http://www.scipy.org/NumPy_for_Matlab_Users

scipy

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
maxentropy	Maximum entropy methods
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions
weave	C/C++ integration

```
>>> import numpy as np
>>> import scipy as sp
```

Vectorizing for Speed (II)

This is silly:

```
>>> dumb_myfun = lambda a,b: numpy.array([a[i] if a[i] < b else 2*b \
                                          for i in xrange(len(a))])
>>> x = np.arange(1.e6)/1.e6
>>> print x
[  0.00000000e+00   1.00000000e-06   2.00000000e-06 ...,   9.99997000e-01
   9.99998000e-01   9.99999000e-01]
>>> %timeit dumb_myfun(x,0.5)
1 loops, best of 3: 2.56 s per loop
```

Use the scipy.vectorize, which is like `map` for numpy arrays:

```
>>> myfun = lambda a,b: a if a < b else 2*b
>>> vec_myfun = scipy.vectorize(myfun)
>>> %timeit  vec_myfun(x,0.5)
1 loops, best of 3: 599 ms per loop
```

recall also numexpr

```
>>> b = 0.5
>>> %timeit ne.evaluate("where(x < %f, x, %f)" % (b,b*2))
100 loops, best of 3: 9.36 ms per loop
```

scipy.constants

```
>>> from scipy import constants as cons
>>> print (cons.milli, cons.eV, cons.c)
(0.001, 1.602176565e-19, 299792458.0)
>>> print cons.physical_constants
{'joule-electron volt relationship': (6.24150934e+18, 'eV',
140000000000.0), 'conductance quantum': (7.7480917346e-05, 'S',
2.5e-14),....}
>>> len(cons.physical_constants.keys())
399
>>> cons.find("Newton")  ## search substrings
['Newtonian constant of gravitation', 'Newtonian constant of
gravitation over h-bar c']
>>> cons.value('Newtonian constant of gravitation')
6.67384e-11
>>> cons.unit('Newtonian constant of gravitation')
'm^3 kg^-1 s^-2'
>>> cons.precision('Newtonian constant of gravitation')
0.00011987101878378866
>>> cons.F2K(32.0)
273.14999999999998
```

conversions, & constants from 2010 CODATA

scipy.interpolate

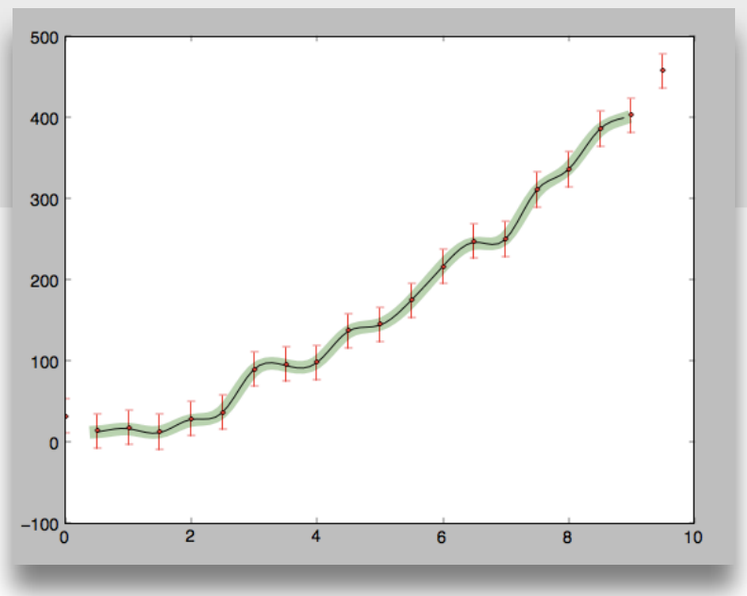
```
from scipy.interpolate import interp1d, UnivariateSpline
from scipy import constants as cons
%pylab inline


# set up some fake data, listing locations versus time
M = 5.98e24  # mass and radius of Earth
R = 6.38e6
accel = cons.G*M/R**2 # Earth's grav acceleration
times = np.arange(0,10,0.5) # seconds
locations = 0.5*accel*times**2 + 50*np.random.random(len(times))


# now interpolate onto a much finer grid, using both iterp1d and UnivariateSpline
i_times = np.arange(0.5,9,0.1)
i_locs = interp1d(times, locations, kind="cubic")  # list of interpolated values of location at i_times
s = UnivariateSpline(times, locations) # a function that will return interpolated values


# plot up several different views of this example
plt.errorbar(times, locations, yerr=50/2.35, linestyle='None', marker=".", label='data')
plt.plot(itimes, s(itimes), c="g", lw=10, alpha=0.3, label='UnivariateSpline')
plt.plot(itimes, iloc(itimes), c="black", lw=1, alpha=0.9, label='interp1d')
plt.ylabel('locations')
plt.xlabel('times (s)')
plt.legend(loc='best')
```

see notebook:
interpolation_example.ipynb



Numerical Integration

scipy.integrate

```
>>> from scipy.integrate import quad
>>> val, err =quad( lambda x: sin(x) , 0 , np.pi, full_output=False)
>>> print val, err
>>> from scipy.integrate import ode
>>> ode?
```

scipy.special

```
>>> from scipy import special
>>> special?
Airy Functions
Elliptic Functions and Integrals
Bessel Functions
Struve Functions
Gamma and Related Functions
Error Function and Fresnel Integrals
Legendre Functions
Orthogonal polynomials --- 15 types
HyperGeometric Functions

...
>>> from scipy.special import betainc
>>> betainc(10,10,0.2)

0.00157912054916710 57
```

Optimize

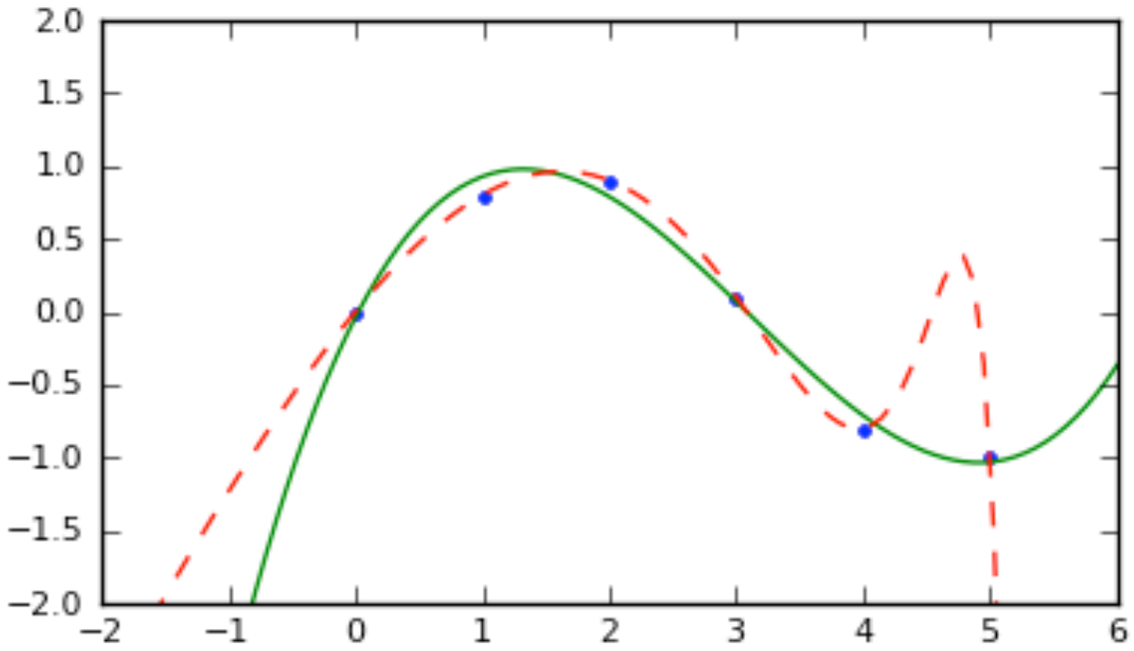
Least Squares Fitting

(usually use fmin...more control)

```
>>> from scipy.optimize import leastsq
>>> def model(par):
        return par[0] + par[1]*sin(2*pi*t/p) + par[2]*cos(2*pi*t/p)
>>> sys_err=0.02  # systematic error term
>>> dy0 = sqrt(dy**2+sys_err**2)
>>> def resid(par):
        return (model(par)-y)**2/dy0
>>> rez = leastsq(resid,[y.mean(),1,0.])
>>> mdl = model(rez[0])
```

Polynomial Fitting

```
>>> x = np.array([0.0, 1.0, 2.0, 3.0,  4.0,  5.0])
>>> y = np.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])
>>> z = np.polyfit(x, y, 3)
>>> p = np.poly1d(z)
>>> p30 = np.poly1d(np.polyfit(x, y, 30))
>>> xp = np.linspace(-2, 6, 100)
>>> plt.plot(x, y, '.', xp, p(xp), '-', xp,
p30(xp), '--')
>>> plt.ylim(-2,2)
>>> plt.show()
```



Basic (least squares) polynomial fitting can be performed using the *polyfit* routine. More complicated fitting tasks require *scipy*

Breakout: Amateur Astronomer

www.google.com/trends/?q=ice+cream,+full+moon

Google Trends

ice cream, full moon

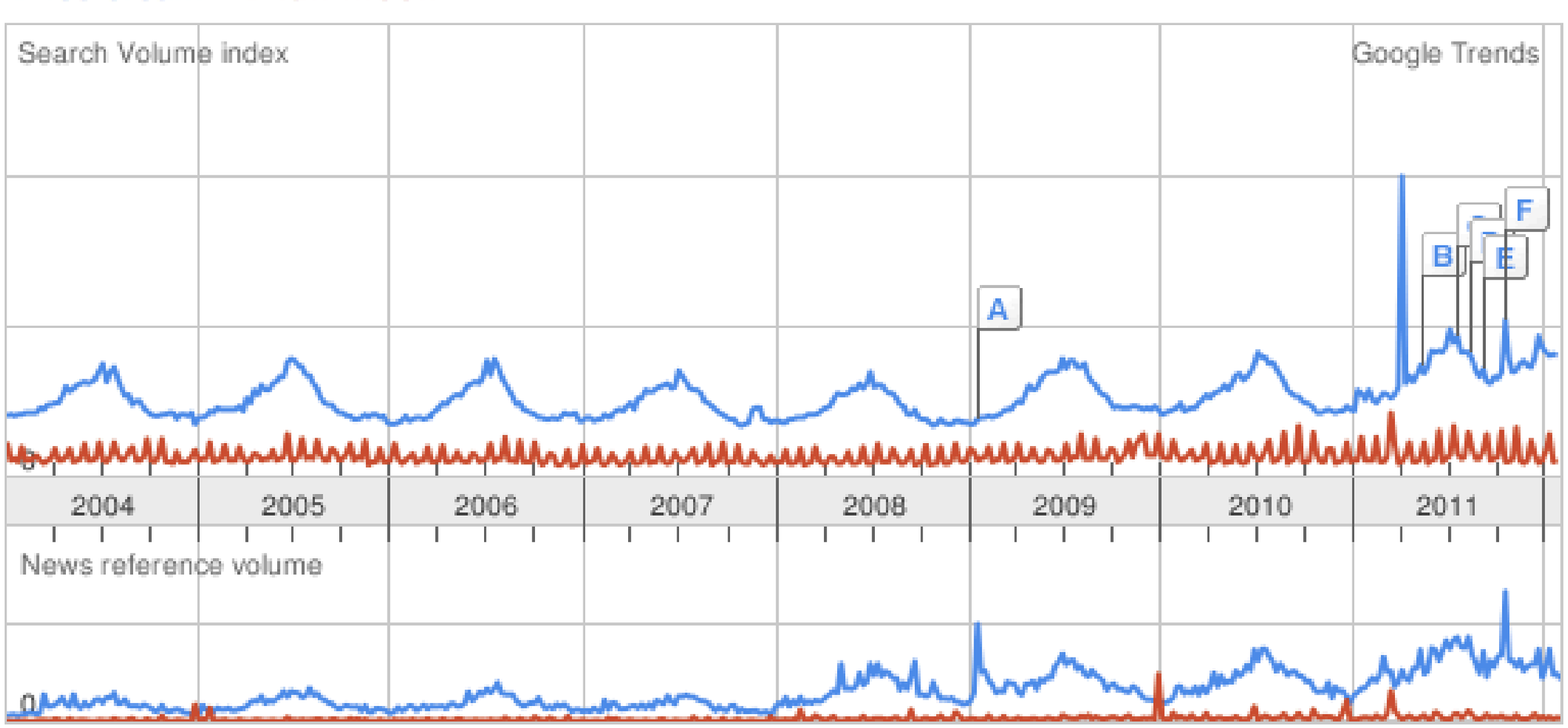
Search Trends

Tip: Use commas to compare multiple search terms.

Searches Websites

An improvement to our geographical assignment was applied retroactively from 1/1/2011. Learn more

● ice cream ● full moon



Rank by ice cream

Breakout: Amateur Astronomer

1) use `loadtxt` to load astro.csv

hint: use the `converters` to handle the date string and percentages

2) approximately what is the lunar period (in days)?

```
from scipy.signal.spectral import lombscargle
```