Introduction to NumPy & SciPy

James Snyder

Research is what I'm doing when I don't know what I'm doing.

—Werner von Braun

Q: Why do scientific computing with Python?

A: Large number of science-related modules:

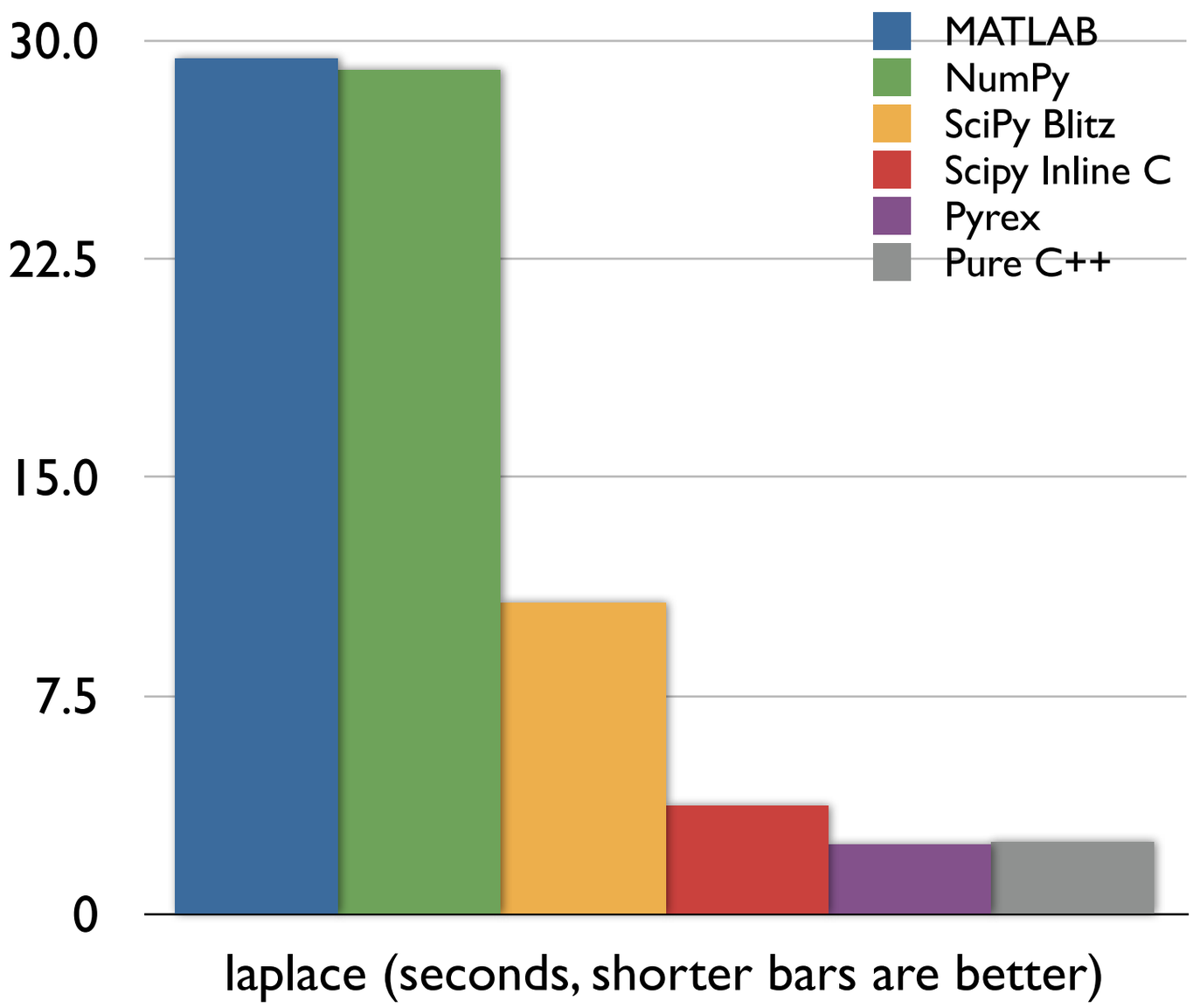
NumPy
SciPy
Matplotlib
ScientificPython
Chaco
MayaVi
OpenOpt
SfePy
AstroLib
PySolar
ffnet
pyem
pymorph
Monte
hcluster
PyMC
Brian
PySAT
PyDSTool
NIPY
Simpy
PyFemax
CDAT
ClimPy
PyClimate
GIS Python
PyMOL
SloppyCell
Biskit
pyaudio
GNU Radio
SymPy

For a more complete list: http://www.scipy.org/Topical_Software

Q: Why use NumPy & SciPy?

A: Performs well compared with commercial options.

Laplace Transform (1000 iterations, 500x500 grid)



Plain Python: 2110 seconds
Python + Psyco: 1760 seconds

Q: Why use NumPy & SciPy?

A:

- Batteries included + lots of other modules
  - Build tools and applications faster
- Good balance between ease of use and speed
  - Similar performance to expensive commercial solutions
  - Plenty of options to optimize critical parts
  - Only spend time on speed if you need it
- Most tools are open source & free
- No license management

NumPy

- N-dimensional arrays (ndarray)
- Universal functions (ufunc)
- linear algebra, fourier transforms, PRNGs
- Tools to integrate C/C++/Fortran
- Heavy lifting done by C/Fortran code
  - ATLAS or MKL, UMFPACK, FFTW, etc..

Making Arrays:

```
# Initialize with lists: array with 2 rows, 4 columns
>>> np.array([[1,2,3,4],[8,7,6,5]])
```

```
# Make Array of evenly spaced numbers over and interval
>>> np.linspace(1,100,10)
array([   1.,   12.,   23.,   34.,   45.,   56.,   67.,
78.,   89.,  100.])
```

```
# Fill an array with zeros
>>> np.zeros((2,5))
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

Indexing Arrays:

```
>>> a = np.array([[1,2,3,4],[9,8,7,6],[1,6,5,4]])
>>> a[0,:] # get row zero
array([1, 2, 3, 4])
```

```
>>> a[:,[0,2]] # get columns 0 and 2
array([[ 1,  3],
       [ 9,  7],
       [ 1,  5]])
```


```
>>> a[:,[0,2]] # get columns 0 and 2
array([[ 1,  3],
       [ 9,  7],
       [ 1,  5]])
```

Referencing With Slices:

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

# point b to columns 0 and 2, like: range(0,3,2)
>>> b = a[:,0:3:2]
>>> b
array([[10,  3],
       [ 9,  7],
       [ 1,  5]])

>>> b[0,0] = 5 # assignment shows up in b & a
>>> b
array([[5, 3],
       [9, 7],
       [1, 5]])

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

Broadcasting with ufuncs:

apply operations to many elements with one call

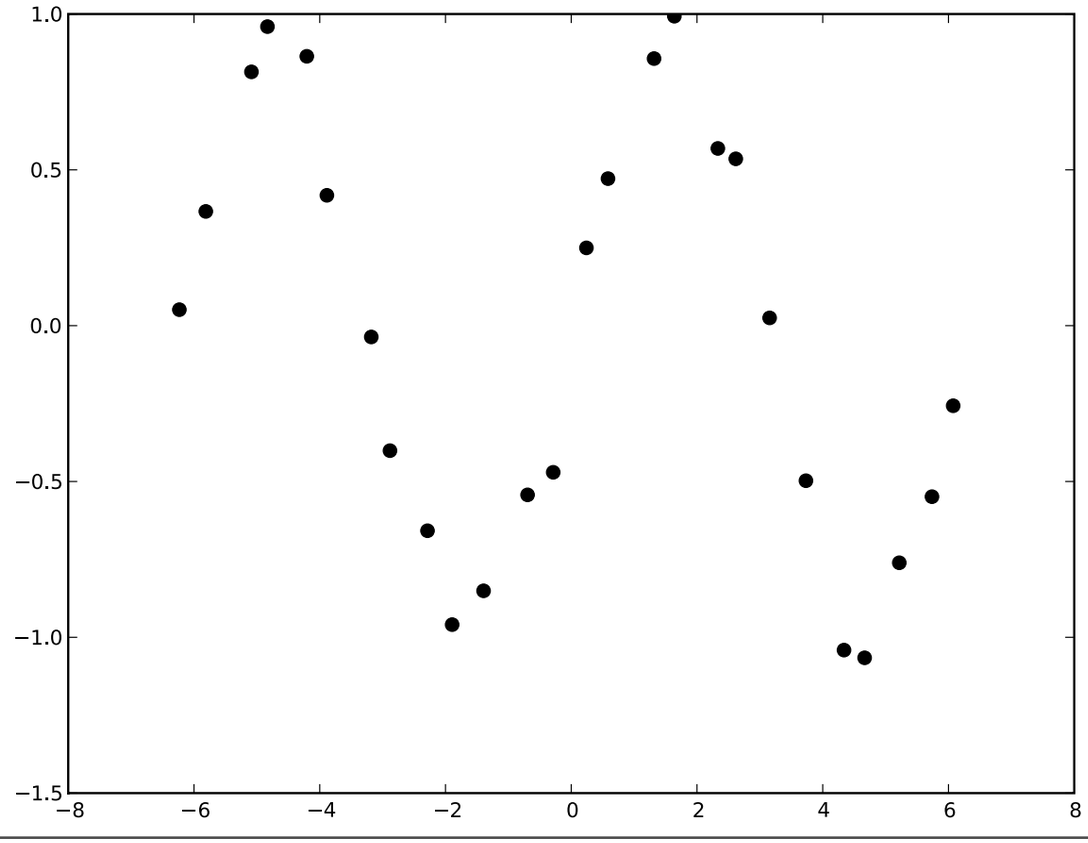
```
>>> a = np.array([[1,2,3,4],[8,7,6,5]])
>>> a
array([[1, 2, 3, 4],
       [8, 7, 6, 5]])

# Rule 1: Dimensions of one may be prepended to either array
>>> a + 1 # add 1 to each element in array
array([[2, 3, 4, 5],
       [9, 8, 7, 6]])


# Rule 2: Arrays may be repeated along dimensions of length 1
>>> a + array([[1],[10]]) # add 1 to 1st row, 10 to 2nd row
>>> a + ([1],[10]) # same as above
>>> a + [[1],[10]] # same as above
array([[ 2,  3,  4,  5],
       [18, 17, 16, 15]])


>>> a**([2],[3]) # raise 1st row to power 2, 2nd to 3
array([[  1,   4,   9,  16],
       [512, 343, 216, 125]])
```

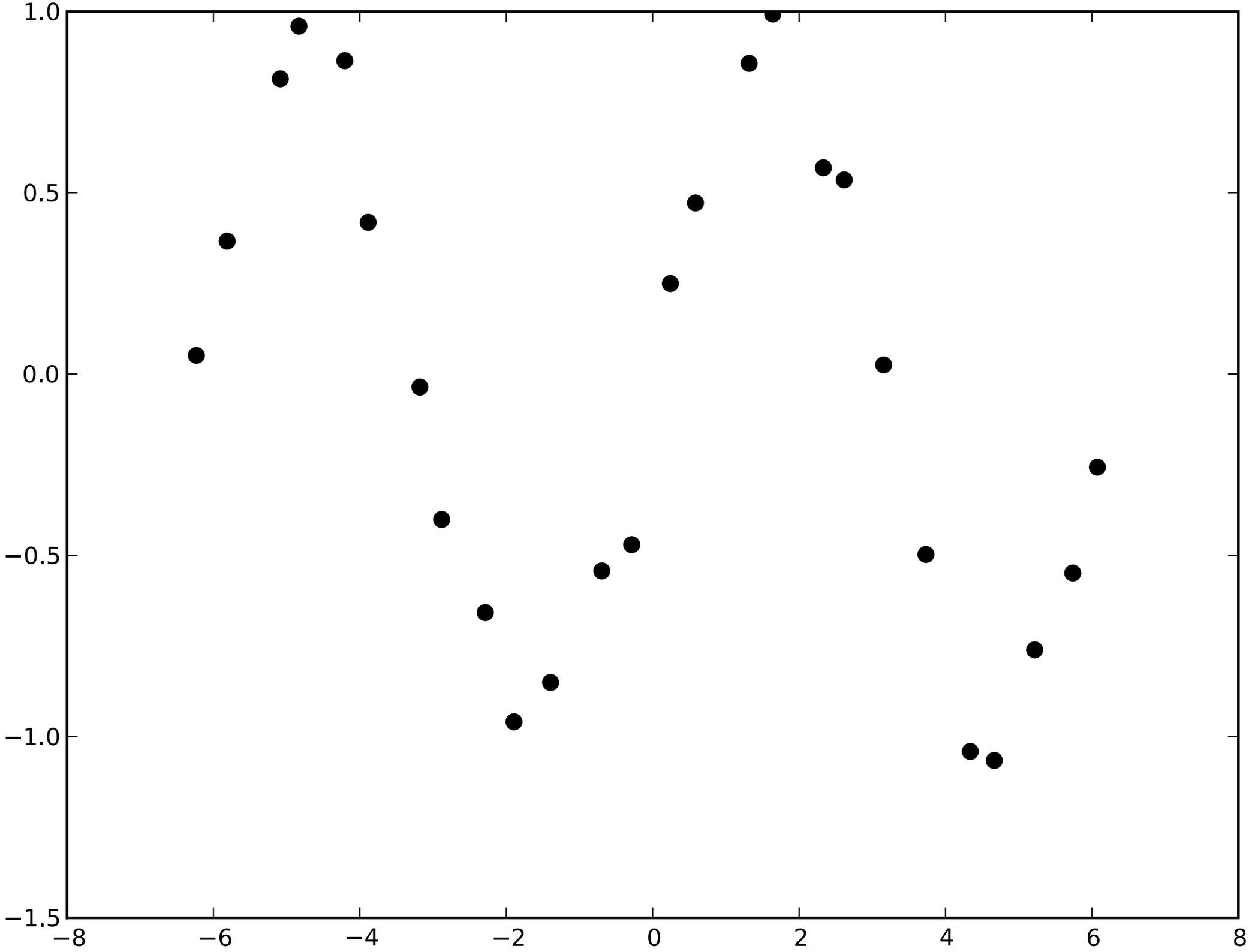
SciPy

- Extends NumPy with common scientific computing tools
  - optimization, linear algebra, integration, interpolation, FFT, signal and image processing, ODE solvers
- Heavy lifting done by C/Fortran code

Generate Signal & Plot:

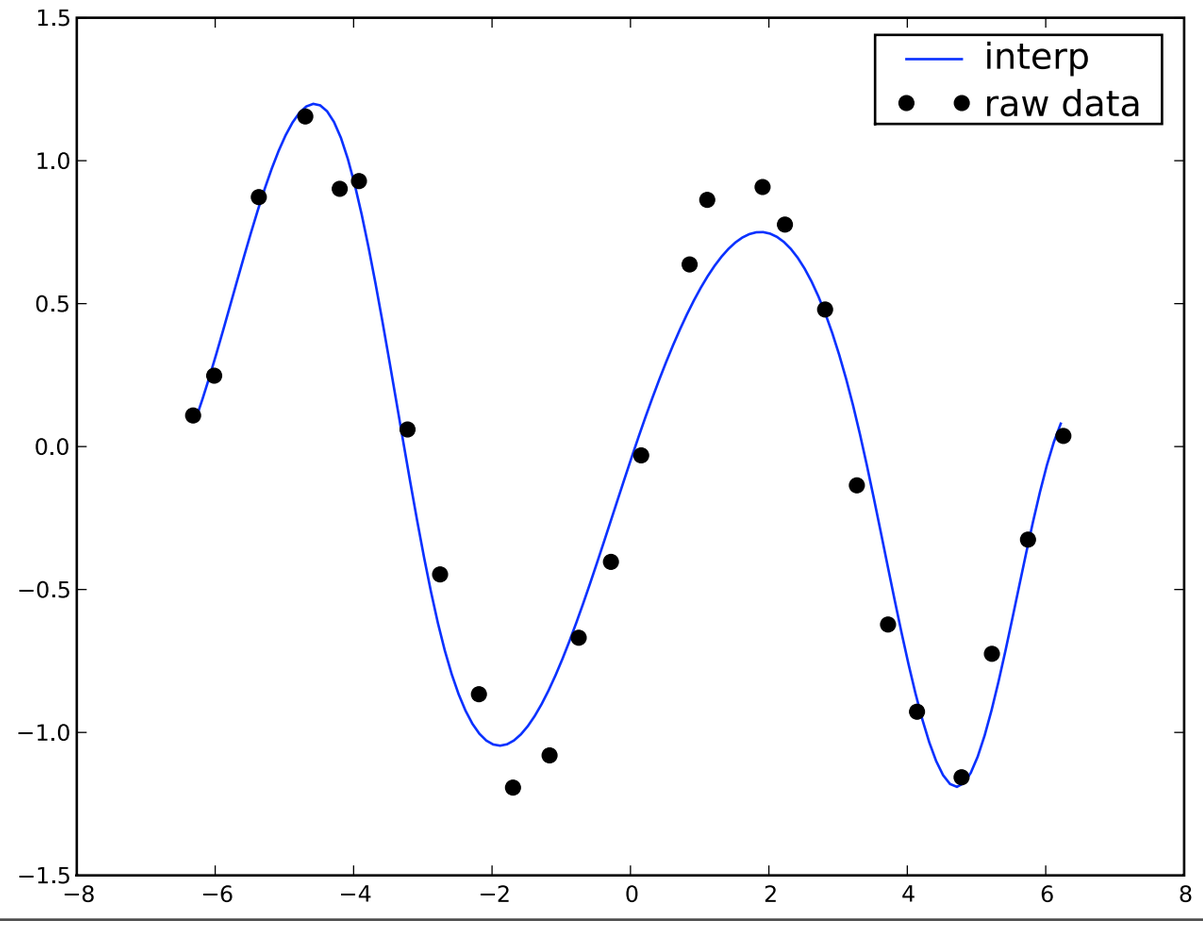
```
import numpy as np
from scipy import optimize, interpolate
import matplotlib.pyplot as plt

# Create range -2pi,2pi with step=0.5,
x = np.arange(-np.pi*2,np.pi*2,0.5)
# add noise
x += np.random.standard_normal(np.shape(x))*0.1
# make sinusoid
y = np.sin(x)
y += np.random.standard_normal(np.shape(x))*0.1

# plot x and y, with black (k) circles (o)
plt.plot(x, y, 'ko')
plt.show()
```

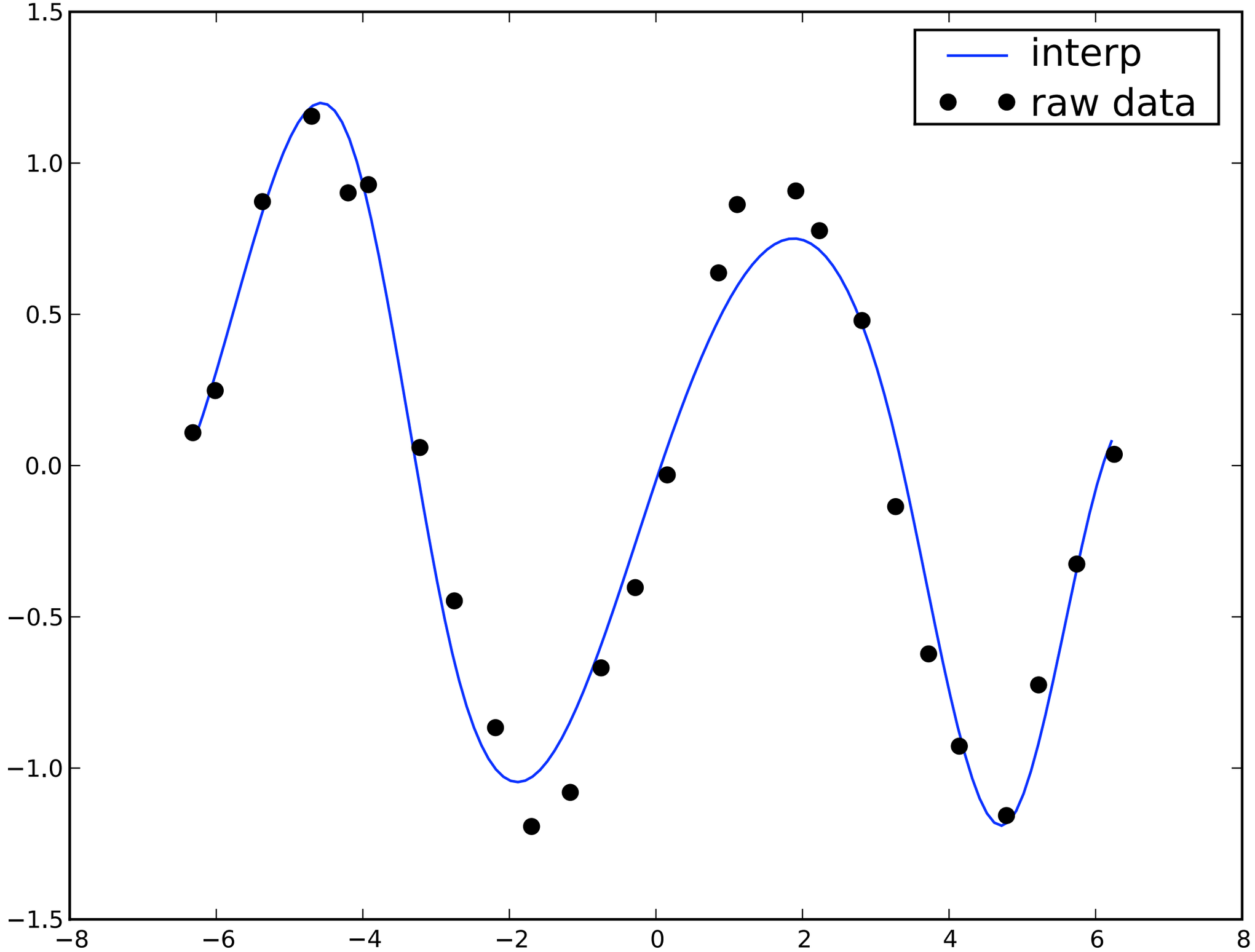



1.0
0.5
0.0
−0.5
−1.0
−1.5
−8
−6
−4
−2
0
2
4
6
8

Interpolating Signal:

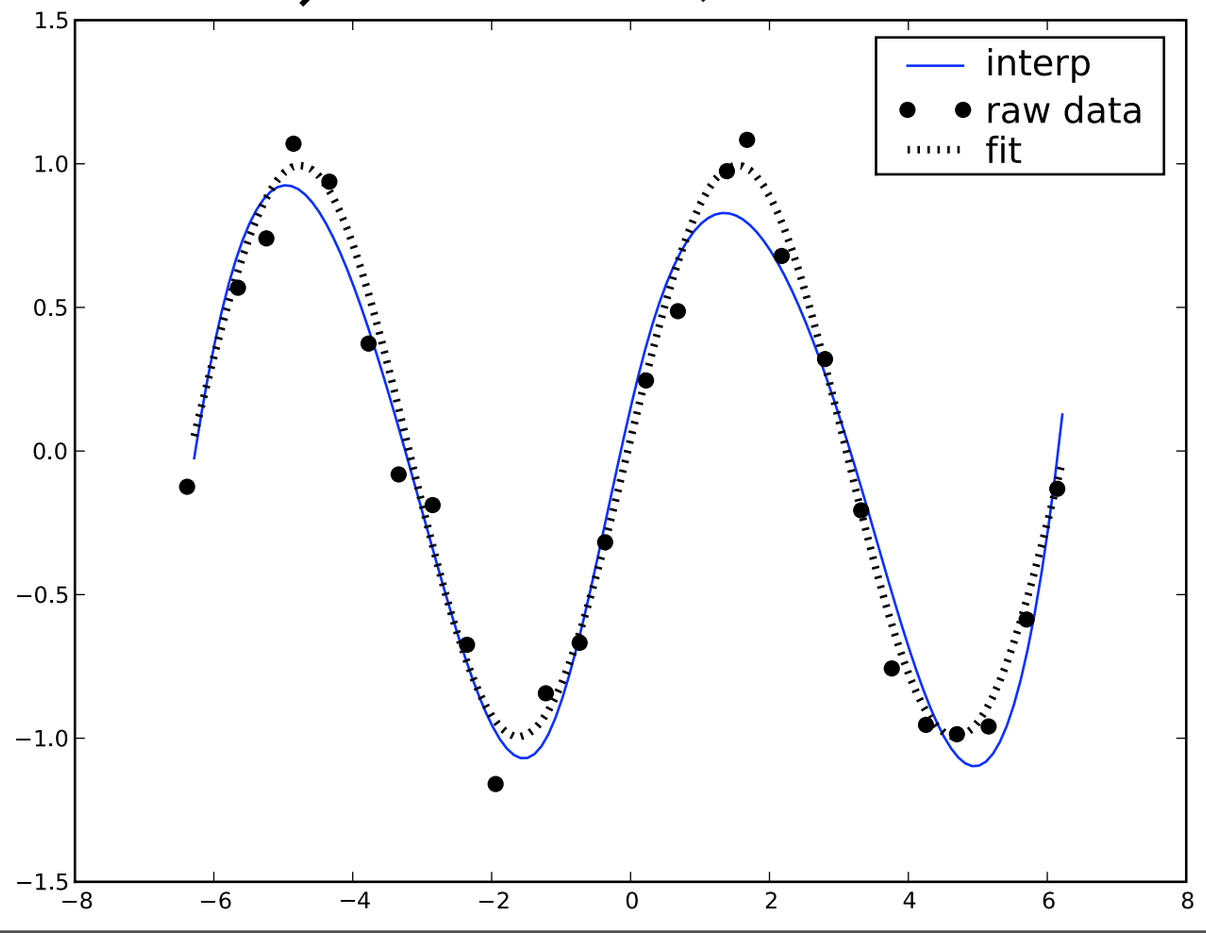
```
# Model signal with spline interp (smoothing set to 0.5)
interp_function = interpolate.UnivariateSpline(x,y,s=0.5)

# New x range for data -2pi,2pi with step size of 0.1
newx = np.arange(-np.pi*2, np.pi*2, 0.1)
# get interpolated version
newy = interp_function(newx)

# New Figure
plt.figure()
# plot blue (b) line (-) of interpolated signal
plt.plot(newx, newy,'b-',label='interp')
# plot original raw data
plt.plot(x, y, 'ko',label='raw data')
# add a legend (uses labels)
plt.legend()
plt.show()
```

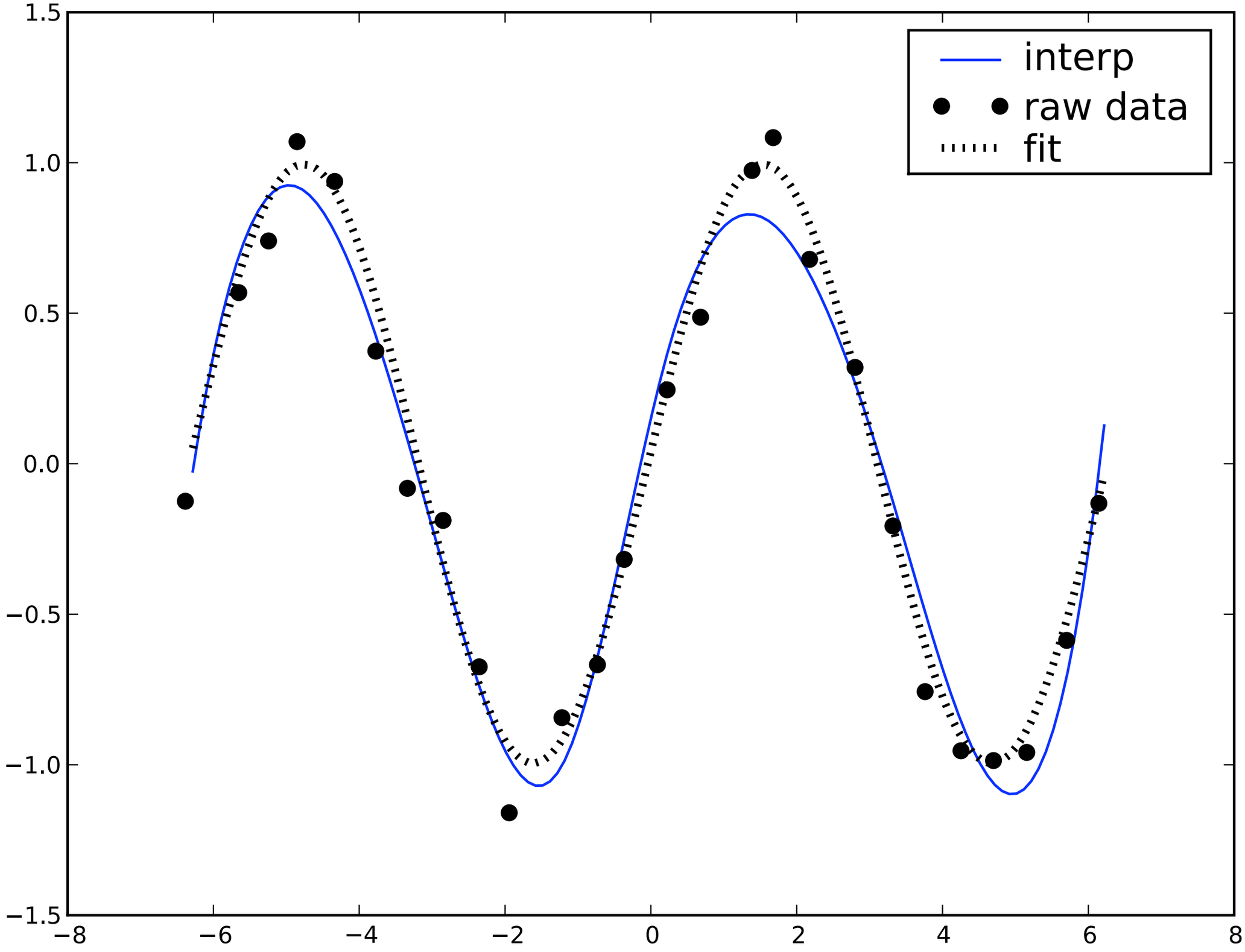



interp
raw data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−8
−6
−4
−2
0
2
4
6
8

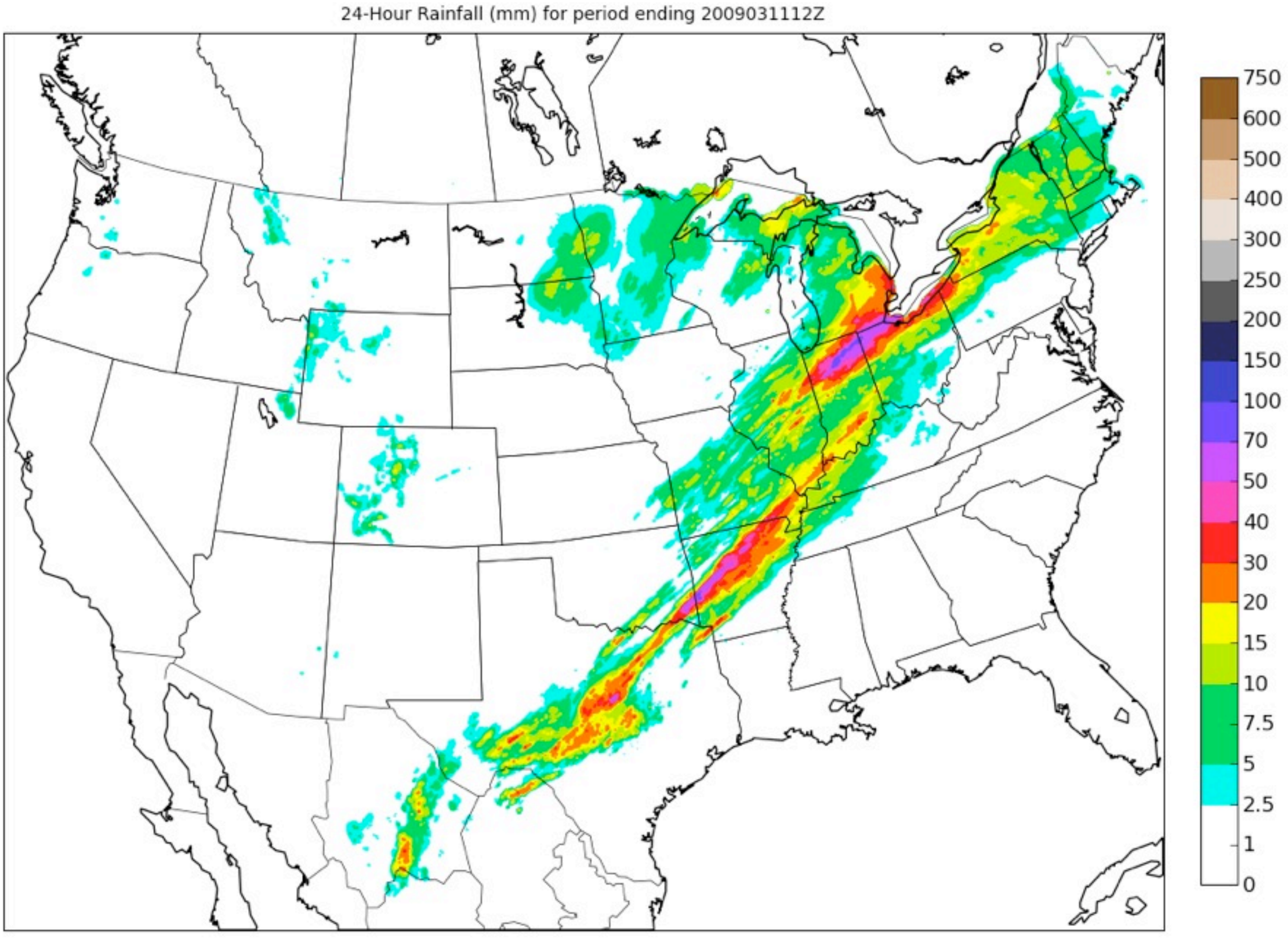
Fitting Signal:

```
# Fit sin function to the data
fitfunc = lambda p, x: p[0]*np.sin(2*np.pi/p[1]*x+p[2])
# err = difference raw data - fit, goal is to minimize this
errfunc = lambda p, x, y: fitfunc(p, x) - y

p0 = [1., 5., 0.] # Initial guess for the parameters
p1 = optimize.leastsq(errfunc, p0[:], args=(x, y))

plt.figure()
plt.plot(newx, newy,'b-',label='interp')
plt.plot(x, y, 'ko',label='raw data')
# plot new fit function data
plt.plot(newx, fitfunc(p1, newx),'k:',label='fit',linewidth=3)
plt.legend()
plt.show()
```




interp
raw data
fit
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−8
−6
−4
−2
0
2
4
6
8

Visualize Public Data Products: Rainfall Mar 11



Saturday, April 11, 2009

Visualize Public Data Products: Rainfall Mar 12