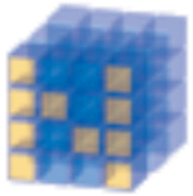
NumPy + SciPy

A Python based Ecosystem of Open-source Software for Math, Science, and Engineering

http://www.numpy.org/

https://www.scipy.org/

Core Packages



NumPy

Base N-dimensional array package

SciPy library

Fundamental library for scientific computing

Matplotlib

Comprehensive 2D Plotting

IP[y]:
IPython

IPython

Enhanced Interactive Console

Sympy

Symbolic mathematics

pandas

Data structures & analysis

Numpy

- Offers Matlab-ish capabilities within Python
- A powerful N-dimensional array object.
- Useful linear algebra, Fourier transform, and random number capabilities.
- Can also be used as an efficient multi-dimensional container of generic data.

Numpy Core component

The ndarray Array object

- An *n*-dimensional array of homogeneous data types,
- Operations are performed in compiled code for better performance
- NumPy arrays have a fixed size. Modifying the size means creating a new array.
- NumPy arrays must be of the same data type, but this can include Python objects.
- More efficient mathematical operations than built-in python sequence types.

Variables in a Numpy Array are stored with a dtype equivalent to python basic types

Basic Type	Available NumPy types	Comments
Boolean	bool	Elements are 1 byte in size
Integer	int8, int16, int32, int64, int128, int	int defaults to the size of int in C for the platform
Unsigned Integer	uint8, uint16, uint32, uint64, uint128, uint	uint defaults to the size of unsigned int in C for the platform
Float	float32, float64, float, longfloat,	Float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform dependent.
Complex	complex64, complex128, complex	The real and complex elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	str, unicode	Unicode is always UTF32 (UCS4)
Object	object	Represent items in array as Python objects.

Numpy Datatype examples

```
>>> import numpy as np
>>> x = np.float32(1.0)
>>> x
1.0
>>> y = np.int_([1,2,4])
>>> y
array([1, 2, 4])
>>> z = np.arange(3, dtype=np.uint8)
>>> z
array([0, 1, 2], dtype=uint8)
>>> z.dtype
dtype('uint8')
```

Numpy Arrays

There are different ways for creating Numpy arrays, for example

- Conversion from other Python structures (e.g., lists, tuples).
- Built-in NumPy array creation (e.g., arange, ones, zeros, etc.).
- Reading arrays from disk, (e.g. reading in from a CSV file).

In general, any python container, eg: list, or tuple can be converted to a numpy array

```
>>> x = np.array([2,3,1,0])
>>> print x
[2 3 1 0]
>>> x = np.array([[1,2,0], [0,0], (1+1j, 3.0)])
>>> print x
[[1, 2, 0] [0, 0] ((1+1j), 3.0)]
>>> x = np.array([[1+0j, 2+0j], [1+1j, 3+0j]])
>>> x
array([[ 1.+0.j,  2.+0.j],
       [ 1.+1.j,  3.+0.j]])
>>> print x
[[ 1.+0.j  2.+0.j]
 [ 1.+1.j  3.+0.j]]
```

Creating Numpy Arrays using functions

- There are a couple of built-in NumPy functions which will create arrays from scratch.
- zeros(shape) -- creates an array filled with 0 values with the specified shape. The default dtype is float64.

```
>>> np.zeros((2, 3))
array([[ 0., 0., 0.], [ 0., 0., 0.]])
```

- ones(shape) -- creates an array filled with 1 values.
- arange() -- creates arrays with regularly incrementing values.

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2, 10, dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8., 9.])
>>> np.arange(2, 3, 0.1)
array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
```

Numpy Array Operations

```
>>> a = np.arange(5)
>>> b = np.arange(5)
>>> a+b
array([0, 2, 4, 6, 8])
>>> a-b
array([0, 0, 0, 0, 0])
>>> a**2
array([ 0,  1,  4,  9, 16])
>>> a>3
array([False, False, False, False,  True], dtype=bool)
>>> 10*np.sin(a)
array([ 0., 8.41470985, 9.09297427, 1.41120008, -7.56802495])
>>> a*b
array([ 0,  1,  4,  9, 16])
```

Basic operations apply element-wise. The result is a new array with the resultant elements.

Operations like *= and += will modify the existing array.

Other Numpy Array operations

Useful built in methods

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.68166391, 0.98943098, 0.69361582],
       [ 0.78888081, 0.62197125, 0.40517936]])
>>> a.sum()
4.1807421388722164
>>> a.min()
0.4051793610379143
>>> a.max(axis=0)
array([ 0.78888081, 0.98943098, 0.69361582])
>>> a.min(axis=1)
array([ 0.68166391, 0.40517936])
```

Shape manipulation

```
>>> a = np.floor(10*np.random.random((3,4)))
>>> print a
[[ 9. 8. 7. 9.]
 [ 7. 5. 9. 7.]
 [ 8. 2. 7. 5.]]
>>> a.shape
(3, 4)
>>> a.ravel()
array([ 9., 8., 7., 9., 7., 5., 9., 7., 8., 2., 7., 5.])
>>> a.shape = (6,2)
>>> print a
[[ 9. 8.]
 [ 7. 9.]
 [ 7. 5.]
 [ 9. 7.]
 [ 8. 2.]
 [ 7. 5.]]
>>> a.transpose()
array([[ 9., 7., 7., 9., 8., 7.],
       [ 8., 9., 5., 7., 2., 5.]])
```

The Numpy Linear Algebra Module
Notice the similarity to Matlab

```
>>> from numpy import *
>>> from numpy.linalg import *
>>> a = array([[1.0, 2.0], [3.0, 4.0]])
>>> print a
[[ 1. 2.]
 [ 3. 4.]]
>>> a.transpose()
array([[ 1., 3.],
       [ 2., 4.]])
>>> inv(a) # inverse
array([[-2. , 1. ],
       [ 1.5, -0.5]])
```

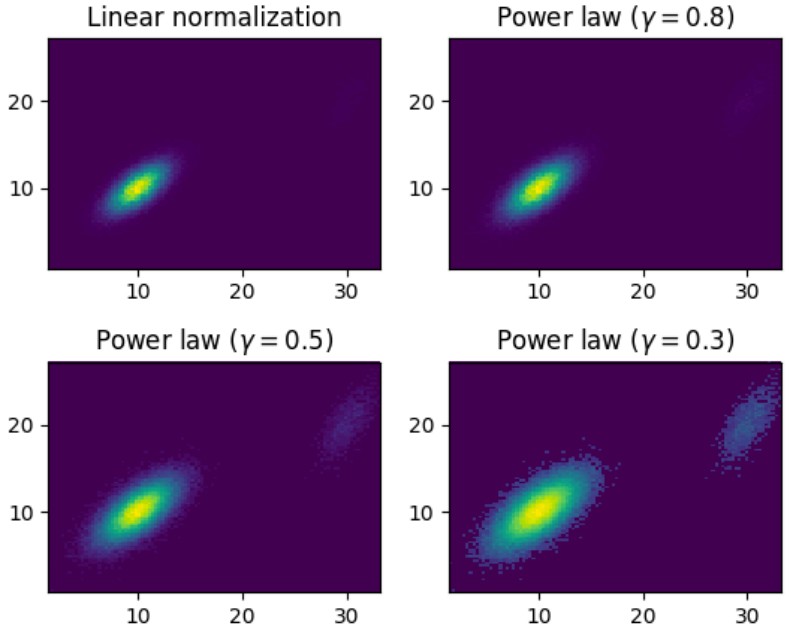
```
>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1., 0.],
       [ 0., 1.]])
>>> j = array([[0.0, -1.0], [1.0, 0.0]])
>>> dot(j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
>>> trace(u) # trace
2.0
>>> y = array([[5.], [7.]])
>>> solve(a, y) # solve linear matrix equation
array([[-3.],
       [ 4.]])
>>> eig(j) # get eigenvalues/eigenvectors of matrix
(array([ 0.+1.j, 0.-1.j]),
 array([[ 0.70710678+0.j, 0.70710678+0.j],
        [ 0.00000000-0.70710678j, 0.00000000+0.70710678j]]))
```

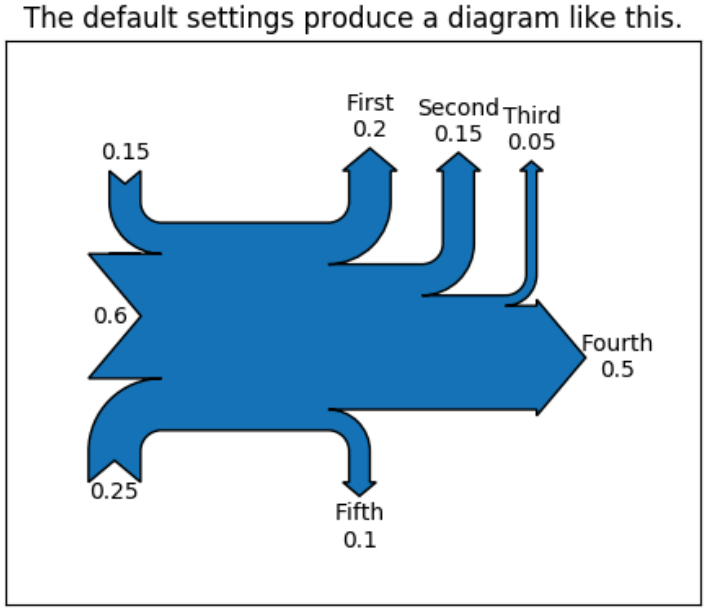
Creating matrices using the Numpy Array Linear Algebra module

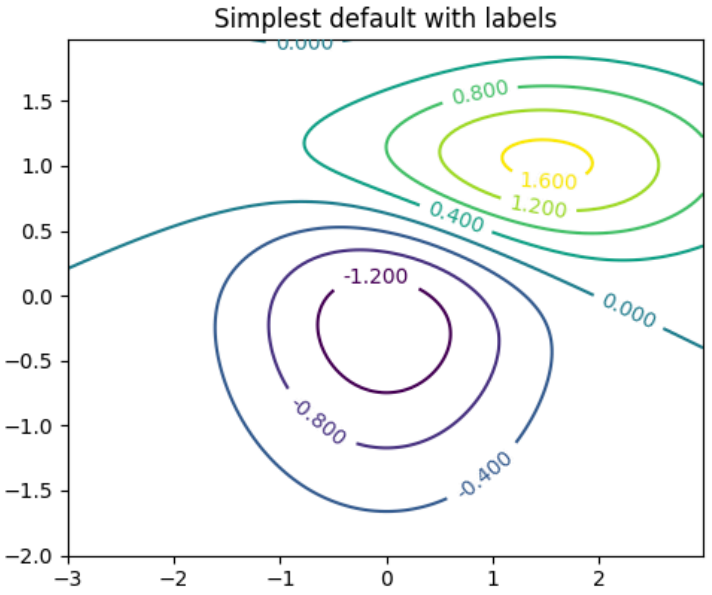
```
>>> A = matrix('1.0 2.0; 3.0 4.0')
>>> A
[[ 1. 2.]
 [ 3. 4.]]
>>> type(A)
<class 'numpy.matrixlib.defmatrix.matrix'>
>>> A.T # transpose
[[ 1. 3.]
 [ 2. 4.]]
>>> X = matrix('5.0 7.0')
>>> Y = X.T
>>> print A*Y # matrix multiplication
[[19.]
 [43.]]
>>> print A.I # inverse
[[-2. 1. ]
 [ 1.5 -0.5]]
>>> solve(A, Y) # solving linear equation
matrix([[-3.], [ 4.]])
```

Math Plot Library (`matplotlib`)

- Powerful package for creating 2D and 3D math visualizations
- The pyplot module allows for MATLAB like plotting
- Can create basically any type of plot or graph you can think of







SciPy Packages build on Numpy

- SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python.
- It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.
- With SciPy, an interactive Python session becomes a data-processing environment similar to systems like MATLAB

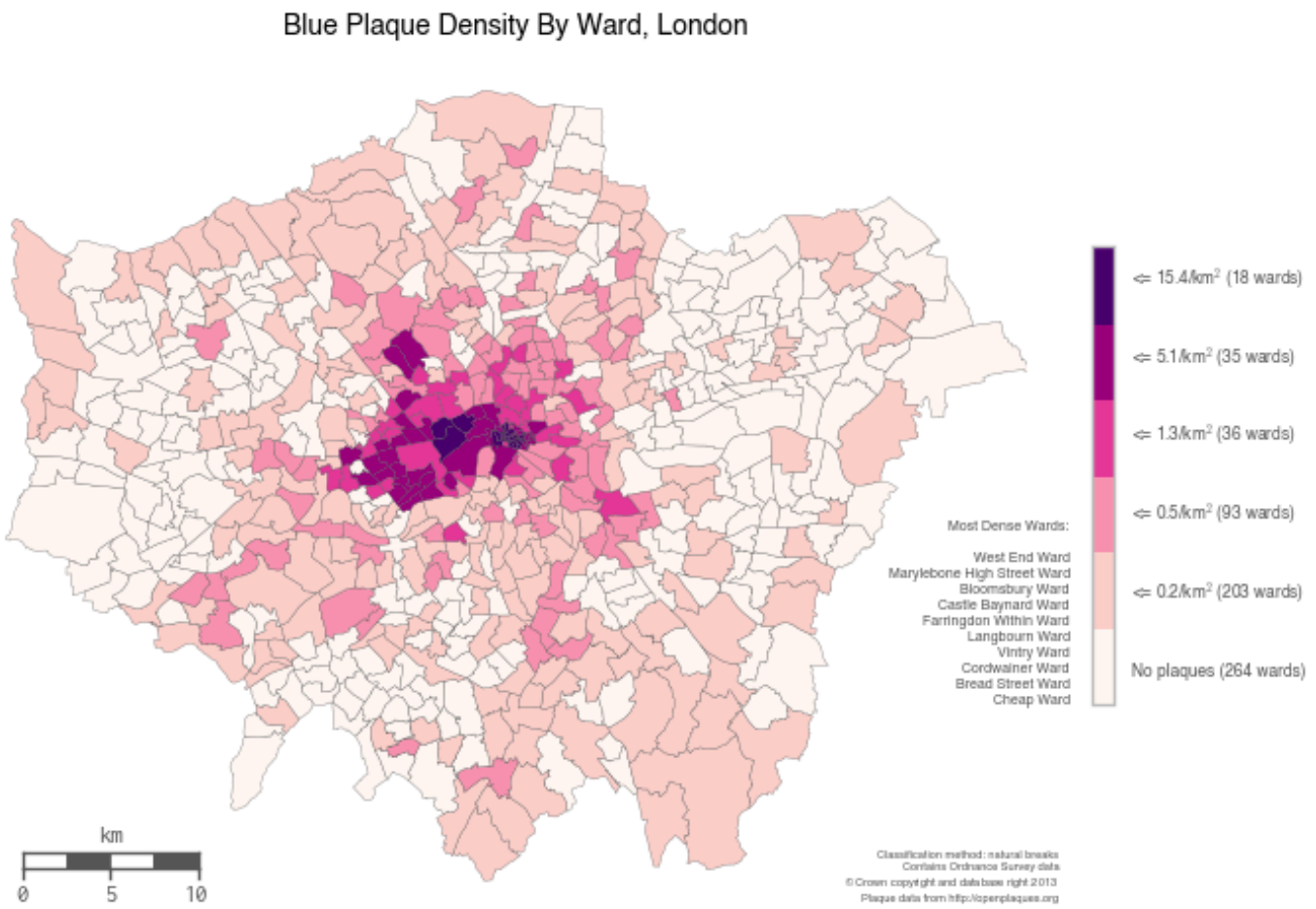
Examples of SciPy modules, this is only the tip of the iceberg

- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Spatial data structures and algorithms (scipy.spatial)
- Statistics (scipy.stats)
- Multidimensional image processing (scipy.ndimage)
- Data IO (scipy.io)

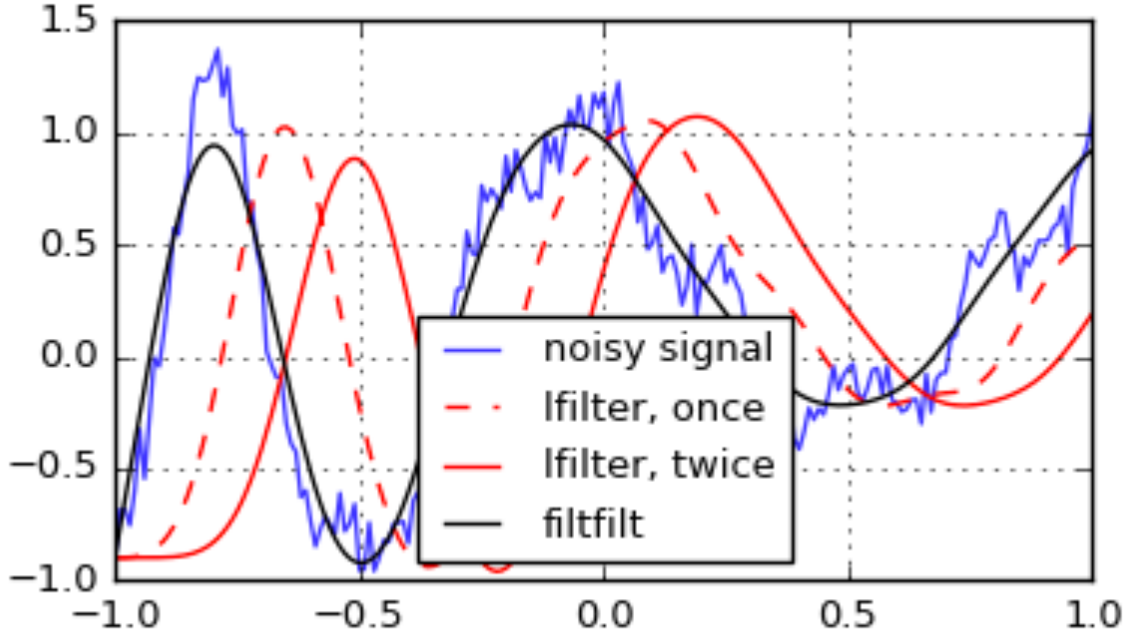
Examples of much more specialized and powerful modules

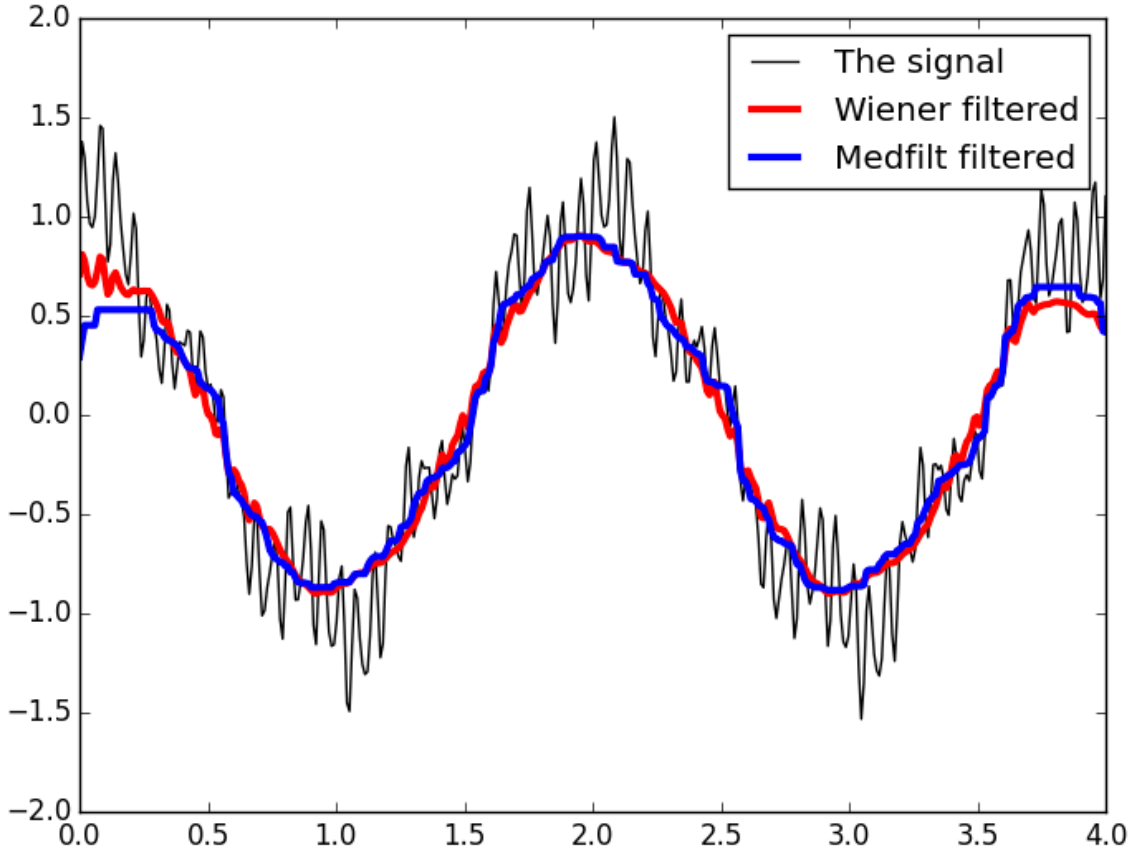
- Astronomy
- Artificial Intelligence and Machine Learning
- Biology (including neuroscience)
- Dynamical Systems
- Economics
- Geo Sciences
- Molecular modelling
- Quantum Mechanics

Blue Plaque Densities in London using Geopython/Geopandas

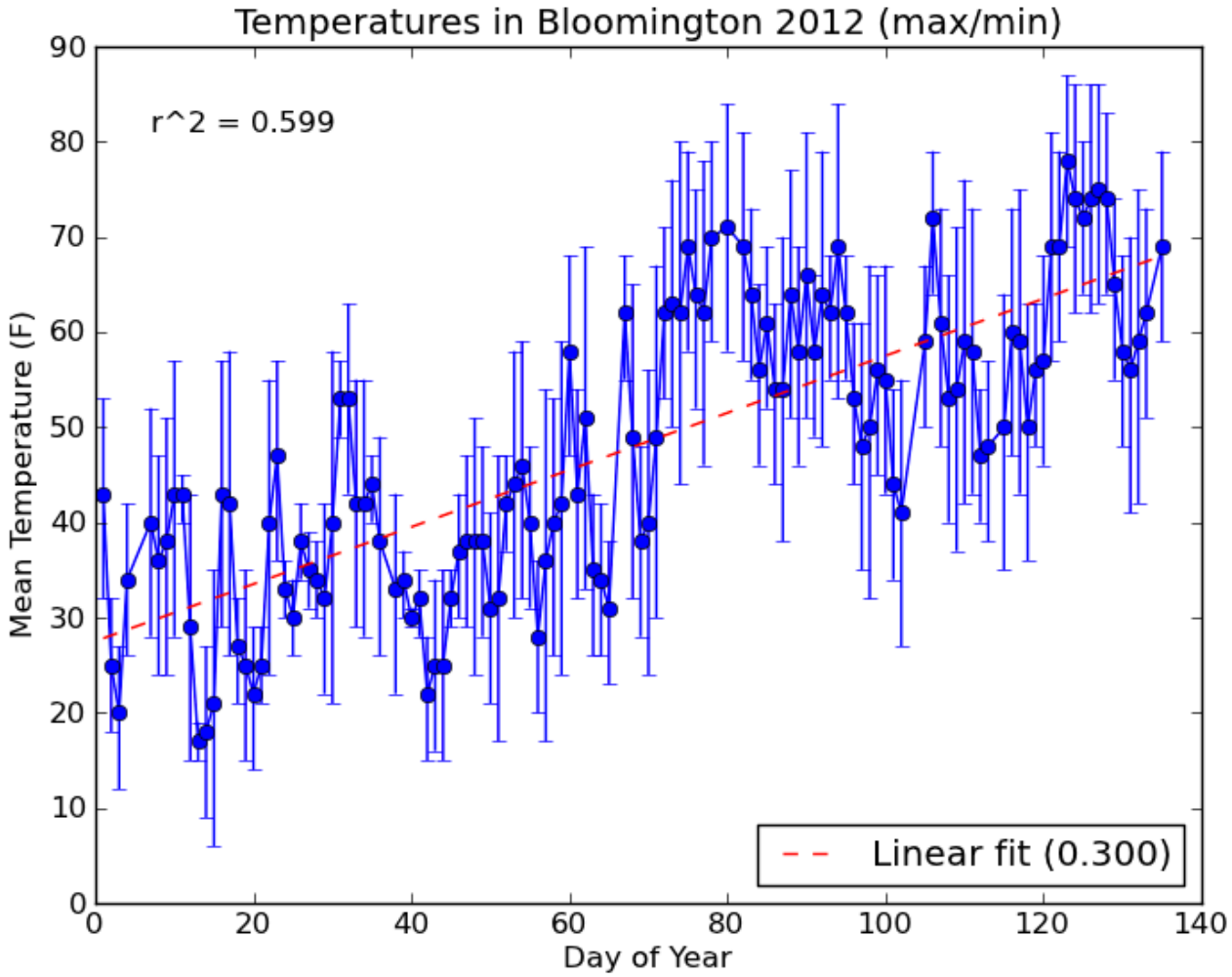


A Signal with Filters Applied using SciPy's Signal Module



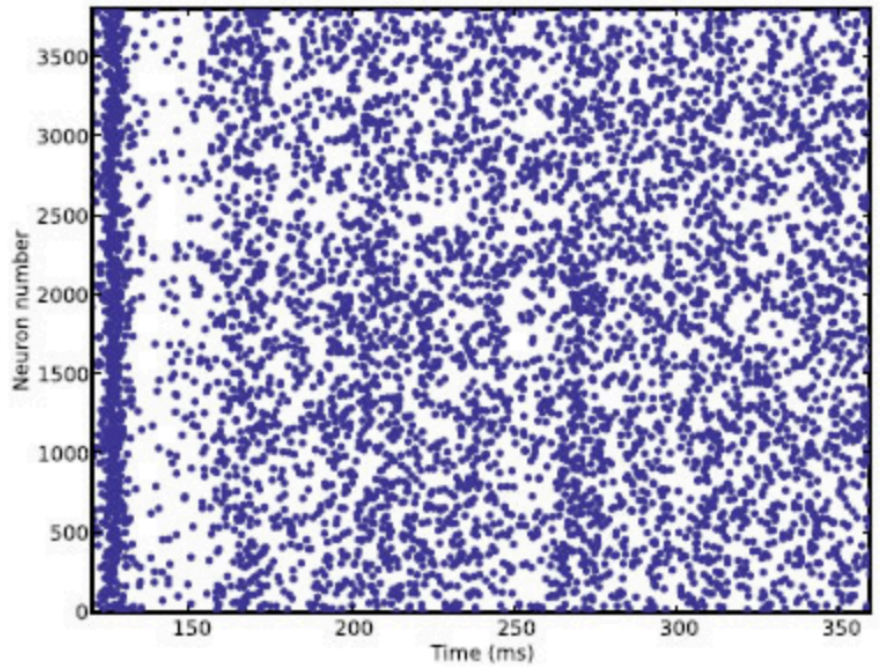


Line Fitting using NumPy's Polyfit Function



Neuroscience simulation using the brian module for spiking neural networks in the brain

```
from brian2 import *
eqs = '''
dv/dt  = (ge+gi-(v+49*mV))/(20*ms) : volt
dge/dt = -ge/(5*ms)                : volt
dgi/dt = -gi/(10*ms)               : volt
'''
P = NeuronGroup(4000, eqs, threshold='v>-50*mV', reset='v=-60*mV')
P.v = -60*mV
Pe = P[:3200]
Pi = P[3200:]
Ce = Synapses(Pe, P, on_pre='ge+=1.62*mV')
Ce.connect(p=0.02)
Ci = Synapses(Pi, P, on_pre='gi-=9*mV')
Ci.connect(p=0.02)
M = SpikeMonitor(P)
run(1*second)
plot(M.t/ms, M.i, '.')
show()
```



Real Time Visualization of Molecular Graphics, Images, and Animations using the PyMol