FREE eBook

# LEARNING scipy

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#scipy

# Table of Contents

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: scipy

It is an unofficial and free scipy ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official scipy.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapter 1: Getting started with scipy

## Remarks

About Scipy

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 and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.

The additional benefit of basing SciPy on Python is that this also makes a powerful programming language available for use in developing sophisticated programs and specialized applications. Scientific applications using SciPy benefit from the development of additional modules in numerous niches of the software landscape by developers across the world. Everything from parallel programming to web and data-base subroutines and classes have been made available to the Python programmer. All of this power is available in addition to the mathematical libraries in SciPy.

## Versions

| Version | Release Date |
|---|---|
| 0.19.0 | 2017-03-09 |
| 0.18.0 | 2016-07-25 |
| 0.17.0 | 2016-01-22 |
| 0.16.1 | 2015-10-24 |
| 0.16.0 | 2015-07-23 |
| 0.16b2 | 2015-05-24 |
| 0.16b1 | 2015-05-12 |
| 0.15.1 | 2015-01-18 |
| 0.15.0 | 2015-01-11 |
| 0.14.1 | 2014-12-30 |
| 0.14.1rc1 | 2014-12-14 |
| 0.14.0 | 2014-05-03 |

| Version | Release Date |
| --- | --- |
| 0.14.0rc2 | 2014-04-23 |
| 0.14.0rc1 | 2014-04-02 |
| 0.14.0b1 | 2014-03-15 |
| 0.13.3 | 2014-02-04 |
| 0.13.2 | 2013-12-07 |
| 0.13.1 | 2013-11-16 |
| 0.13.0 | 2013-10-19 |
| 0.13.0rc1 | 2013-10-10 |
| 0.12.1 | 2013-10-08 |
| 0.12.0 | 2013-04-06 |
| 0.12.0rc1 | 2013-03-29 |
| 0.12.0b1 | 2013-02-16 |
| 0.11.0 | 2012-09-24 |
| 0.11.0rc2 | 2012-08-12 |
| 0.11.0rc1 | 2012-07-17 |
| 0.11.0b1 | 2012-06-12 |
| 0.10.1 | 2012-02-26 |
| 0.10.1rc2 | 2012-02-19 |
| 0.10.1rc1 | 2012-02-10 |
| 0.10.0 | 2011-11-13 |
| 0.10.0rc1 | 2011-11-03 |
| 0.10.0b2 | 2011-09-16 |
| 0.10.0b1 | 2011-09-11 |
| 0.9.0 | 2011-02-27 |

# Examples

## Installation or Setup

Scipy contains parts written in C, C++, and Fortran that need to be compiled before use. Therefore make sure the necessary compilers and Python development headers are installed. Having compiled code also means that Scipy needs additional steps to import from development sources, which are explained below.

Fork a copy of the main Scipy repository in Github onto your own account, then create your local repository via:

```
$ git clone git@github.com:YOURUSERNAME/scipy.git scipy
$ cd scipy
$ git remote add upstream git://github.com/scipy/scipy.git
```

To build the development version of Scipy and run tests, spawn interactive shells with the Python import paths properly set up, and so on. Do one of the following:

```
$ python runtests.py -v
$ python runtests.py -v -s optimize
$ python runtests.py -v -t scipy/special/tests/test_basic.py:test_xlogy
$ python runtests.py --ipython
$ python runtests.py --python somescript.py
$ python runtests.py --bench
```

This builds Scipy first, so it may take a while the first time. Specifying `-n` will run the tests against the version of Scipy (if any) found on the current PYTHONPATH.

Using runtests.py is the recommended approach to running tests. There are also a number of alternatives to it, for example in-place build or installing to a virtual environment. Some tests are very slow and need to be separately enabled.

Link to API

**Ubuntu & Debian**

Run command

```
sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook
python-pandas python-sympy python-nose
```

The versions in Ubuntu 12.10 or newer and Debian 7.0 or newer meet the current SciPy stack specification. Users might also want to add the NeuroDebian repository for extra SciPy packages.

## Convert a sparse matrix to a dense matrix using SciPy

```
from scipy.sparse import csr_matrix
A = csr_matrix([[1,0,2],[0,3,0]])
```

```
>>>A
<2x3 sparse matrix of type '<type 'numpy.int64'>'
   with 3 stored elements in Compressed Sparse Row format>
>>> A.todense()
  matrix([[1, 0, 2],
          [0, 3, 0]])
>>> A.toarray()
     array([[1, 0, 2],
            [0, 3, 0]])
```

## Versions

The first release of SciPy, vsn 0.10, was released on August 14th 2001. The current release of SciPy (correct at 26th July 2016) is v 0.17 (stable) with v .18 forthcoming soon. Details of former releases are listed here

## Image Manipulation using Scipy (Basic Image resize)

SciPy provides basic image manipulation functions. These include functions to read images from disk into numpy arrays, to write numpy arrays to disk as images, and to resize images.

In the following code, only one image is used. It is tinted, resized, and saved. Both original and resulting images are shown below:

```
import numpy as np  //scipy is numpy-dependent

from scipy.misc import imread, imsave, imresize   //image resizing functions

# Read an JPEG image into a numpy array
img = imread('assets/cat.jpg')
print img.dtype, img.shape  # Prints "uint8 (400, 248, 3)"

# We can tint the image by scaling each of the color channels
# by a different scalar constant. The image has shape (400, 248, 3);
# we multiply it by the array [1, 0.95, 0.9] of shape (3,);
# numpy broadcasting means that this leaves the red channel unchanged,
# and multiplies the green and blue channels by 0.95 and 0.9
# respectively.
img_tinted = img * [1, 0.95, 0.9]

# Resize the tinted image to be 300 by 300 pixels.
img_tinted = imresize(img_tinted, (300, 300))

# Write the tinted image back to disk
imsave('assets/cat_tinted.jpg', img_tinted)
```


Reference

## Basic Hello World

Create a file (e.g. hello_world.py) in a text editor or a python editor if you have one installed (pick one if you don't - SublimeText, Eclipse, NetBeans, SciTe... there's many!)

```
hwld = 'Hello world'
print(hwld)
```

Note that python variables do not need to be explicitly declared; the declaration happens when you assign a value with the equal (=) sign to a variable.

The output of the above two lines of code is that the string "Hello World" will be displayed.

Functions written in Python can be used in iPython also.

In this instance, you can use run your saved file 'hello_world.py' in IPython like so:

```
In [1]: %run hello_world.py
#run file to get output below
Hello world
In [2]: wld
#show what value of wld var is
Out[2]: 'Hello world'
In [3]: %whowld
#display info on variable wld (name/type/value)

Variable     Type     Data/Info
-----------------------------
```

```
wld         str     Hello world
```

If you wish you can use two variables, e.g one for hello and one for world and concatenate them using the plus (+) sign:

```
h = 'Hello '
w = "world!'
print(h+w)

#this will also output Hello World, only this time with an exclamation mark..
```

Read Getting started with scipy online: https://riptutorial.com/scipy/topic/2128/getting-started-with-scipy

# Chapter 2: Fitting functions with scipy.optimize curve_fit

## Introduction

Fitting a function which describes the expected occurence of data points to real data is often required in scientific applications. A possible optimizer for this task is curve_fit from scipy.optimize. In the following, an example of application of curve_fit is given.

## Examples

### Fitting a function to data from a histogram

Suppose there is a peak of normally (gaussian) distributed data (mean: 3.0, standard deviation: 0.3) in an exponentially decaying background. This distribution can be fitted with curve_fit within a few steps:

1.) Import the required libraries.

2.) Define the fit function that is to be fitted to the data.

3.) Obtain data from experiment or generate data. In this example, random data is generated in order to simulate the background and the signal.

4.) Add the signal and the background.

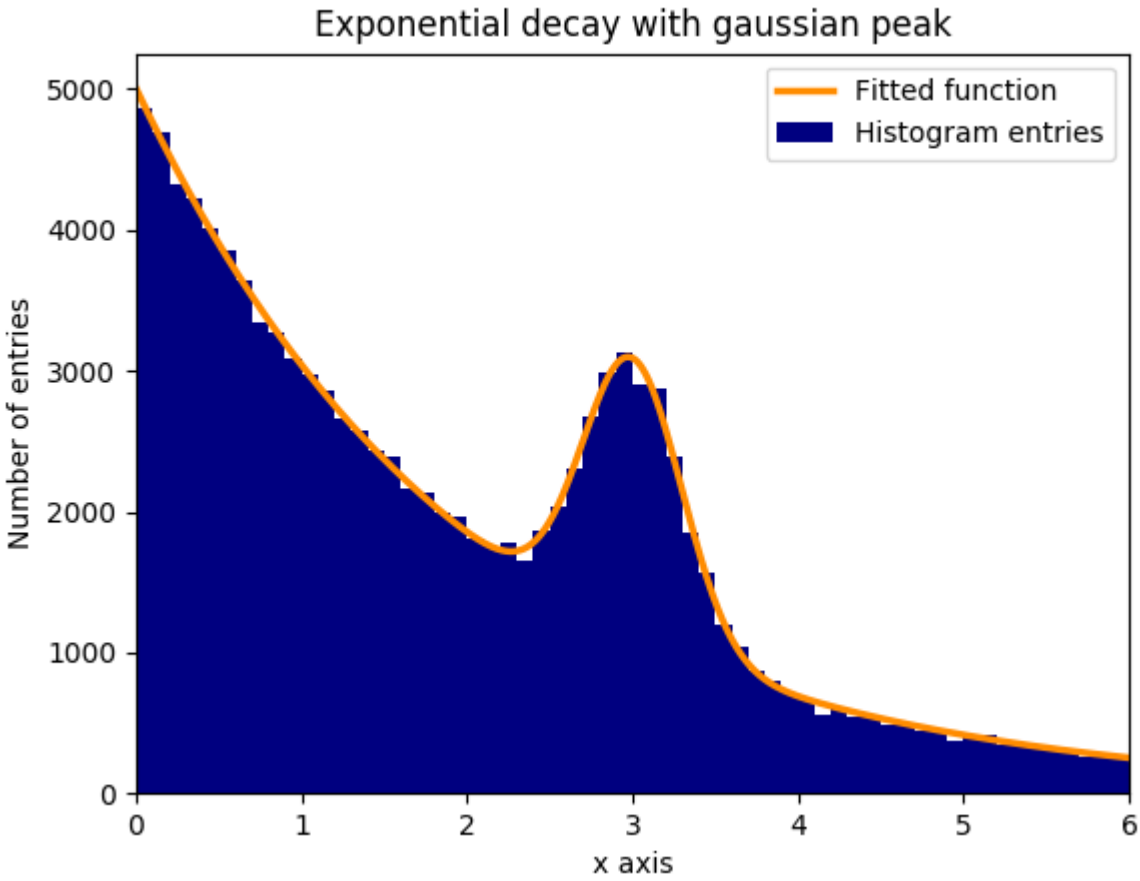
5.) Fit the function to the data with curve_fit.

6.) (Optionally) Plot the results and the data.

In this example, the observed y values are the heights of the histogram bins, while the observed x values are the centers of the histogram bins (`binscenters`). It is necessary to pass the name of the fit function, the x values and the y values to `curve_fit`. Furthermore, an optional argument containing rough estimates for the fit parameters can be given with `p0`. `curve_fit` returns `popt` and `pcov`, where `popt` contains the fit results for the parameters, while `pcov` is the covariance matrix, the diagonal elements of which represent the variance of the fitted parameters.

```
# 1.) Necessary imports.
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# 2.) Define fit function.
def fit_function(x, A, beta, B, mu, sigma):
    return (A * np.exp(-x/beta) + B * np.exp(-1.0 * (x - mu)**2 / (2 * sigma**2)))

# 3.) Generate exponential and gaussian data and histograms.
```

```
data = np.random.exponential(scale=2.0, size=100000)
data2 = np.random.normal(loc=3.0, scale=0.3, size=15000)
bins = np.linspace(0, 6, 61)
data_entries_1, bins_1 = np.histogram(data, bins=bins)
data_entries_2, bins_2 = np.histogram(data2, bins=bins)

# 4.) Add histograms of exponential and gaussian data.
data_entries = data_entries_1 + data_entries_2
binscenters = np.array([0.5 * (bins[i] + bins[i+1]) for i in range(len(bins)-1)])

# 5.) Fit the function to the histogram data.
popt, pcov = curve_fit(fit_function, xdata=binscenters, ydata=data_entries, p0=[20000, 2.0,
2000, 3.0, 0.3])
print(popt)

# 6.)
# Generate enough x values to make the curves look smooth.
xspace = np.linspace(0, 6, 100000)

# Plot the histogram and the fitted function.
plt.bar(binscenters, data_entries, width=bins[1] - bins[0], color='navy', label=r'Histogram
entries')
plt.plot(xspace, fit_function(xspace, *popt), color='darkorange', linewidth=2.5,
label=r'Fitted function')

# Make the plot nicer.
plt.xlim(0,6)
plt.xlabel(r'x axis')
plt.ylabel(r'Number of entries')
plt.title(r'Exponential decay with gaussian peak')
plt.legend(loc='best')
plt.show()
plt.clf()
```




Read Fitting functions with scipy.optimize curve_fit online:
https://riptutorial.com/scipy/topic/10133/fitting-functions-with-scipy-optimize-curve-fit

# Chapter 3: How to write a Jacobian function for optimize.minimize

## Syntax

1. import numpy as np
2. from scipy.optimize import _minimize
3. from scipy import special
4. import matplotlib.pyplot as plt

## Remarks

Note the underscore before 'minimize' when importing from scipy.optimize; '_minimize' Also, i tested the functions from this link before doing this section, and found I had less trouble/it worked faster, if I imported 'special' separately. The Rosenbrock function on the linked page was incorrect - you have to configure the colorbar first; I've posted alternate code but think it could be better.

Further examples to come.

See here for an explanation of Hessian Matrix

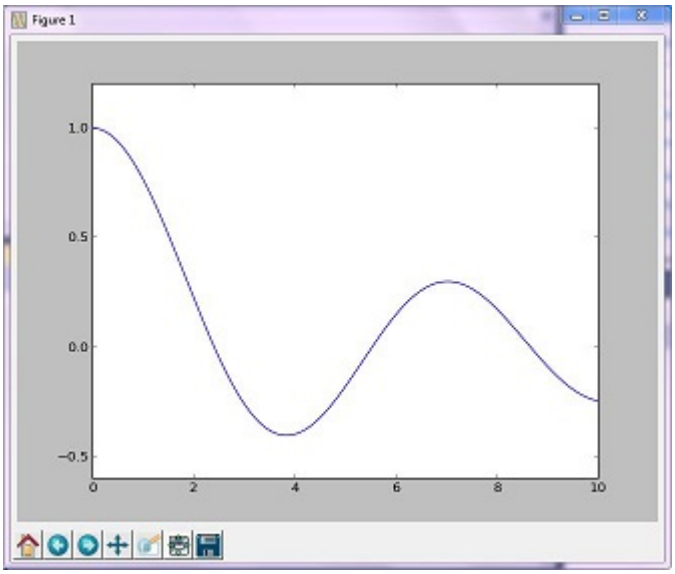
## Examples

### Optimization Example (golden)

The 'Golden' method minimizes a unimodal function by narrowing the range in the extreme values

```
import numpy as np
from scipy.optimize import _minimize
from scipy import special
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 500)
y = special.j0(x)
optimize.minimize_scalar(special.j0, method='golden')
plt.plot(x, y)
plt.show()
```

Resulting image

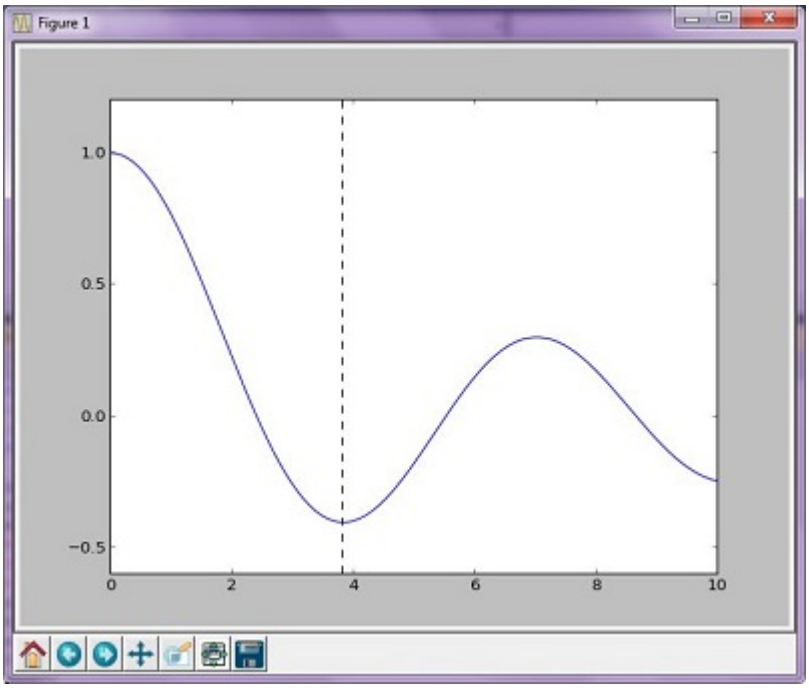



## Optimization Example (Brent)

Brent's method is a more complex algorithm combination of other root-finding algorithms; however, the resulting graph isn't much different from the graph generated from the golden method.

```
import numpy as np
import scipy.optimize as opt
from scipy import special
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 500)
y = special.j0(x)
# j0 is the Bessel function of 1st kind, 0th order
minimize_result = opt.minimize_scalar(special.j0, method='brent')
the_answer = minimize_result['x']
minimized_value = minimize_result['fun']
# Note: minimize_result is a dictionary with several fields describing the optimizer,
# whether it was successful, etc. The value of x that gives us our minimum is accessed
# with the key 'x'. The value of j0 at that x value is accessed with the key 'fun'.
plt.plot(x, y)
plt.axvline(the_answer, linestyle='--', color='k')
plt.show()
print("The function's minimum occurs at x = {0} and y = {1}".format(the_answer,
minimized_value))
```

Resulting graph



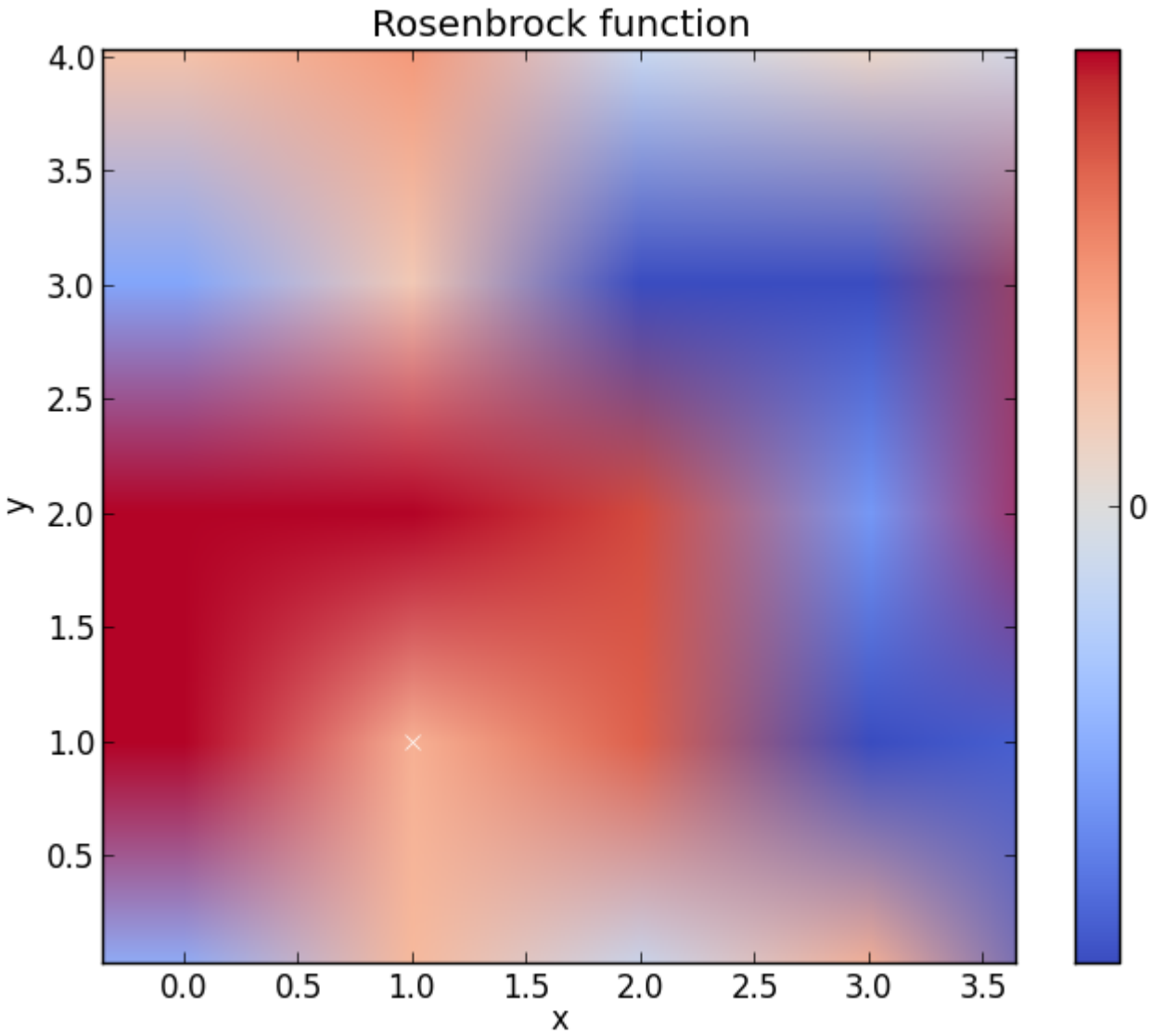

Outputs:

```
The function's minimum occurs at x = 3.8317059554863437 and y = -0.4027593957025531
```

## Rosenbrock function

Think this could example could be better but you get the gist

```
import numpy as np
from scipy.optimize import _minimize
from scipy import special
import matplotlib.pyplot as plt
from matplotlib import cm
from numpy.random import randn

x, y = np.mgrid[-2:2:100j, -2:2:100j]
plt.pcolor(x, y, optimize.rosen([x, y]))
plt.plot(1, 1, 'xw')

# Make plot with vertical (default) colorbar
data = np.clip(randn(100, 100), -1, 1)
cax = plt.imshow(data, cmap=cm.coolwarm)

# Add colorbar, make sure to specify tick locations to match desired  ticklabels
cbar = plt.colorbar(cax, ticks=[-2, 0, 2])  # vertically oriented colorbar
plt.axis([-2, 2, -2, 2])
plt.title('Rosenbrock function') #add title if desired
plt.xlabel('x')
plt.ylabel('y')

plt.show()  #generate
```




Read How to write a Jacobian function for optimize.minimize online:
https://riptutorial.com/scipy/topic/4493/how-to-write-a-jacobian-function-for-optimize-minimize

# Chapter 4: rv_continuous for Distribution with Parameters

## Examples

### Negative binomial on positive reals

```
from scipy.stats import rv_continuous
import numpy

class Neg_exp(rv_continuous):
    def _cdf(self, x, lamda):
        return 1-numpy.exp(-lamda*x)

neg_exp = Neg_exp(name="Negative exponential", a=0)

print (neg_exp.pdf(0,.5))
print (neg_exp.pdf(5,.5))
print (neg_exp.cdf(5,.5))
print (neg_exp.stats(0.5))
print (neg_exp.rvs(0.5))
```

It's essential to define either _pdf or _cdf because scipy infers the parameters of the other function (that you do not define), and the order of these parameters in any functions calls that you make, from the your definition. In this case there is only one distribution parameter, lambda. The variable representing the random variable value appears first in the definition of _pdf or _cdf.

When you define just one of these functions scipy will calculate the other numerically. For possible greater efficiency, define both. Similarly, define _stats in terms of known parameters for best efficiency; otherwise scipy uses numerical methods.

Notice that the distribution's support is defined when the class is instantiated (variable **a** is set to zero and **b** is set to infinity by default), rather then when it is subclassed. Notice too that the distribution's parameters are set only when one of the class instances are called, as in the final five lines of code.

Read rv_continuous for Distribution with Parameters online:
https://riptutorial.com/scipy/topic/6873/rv-continuous-for-distribution-with-parameters

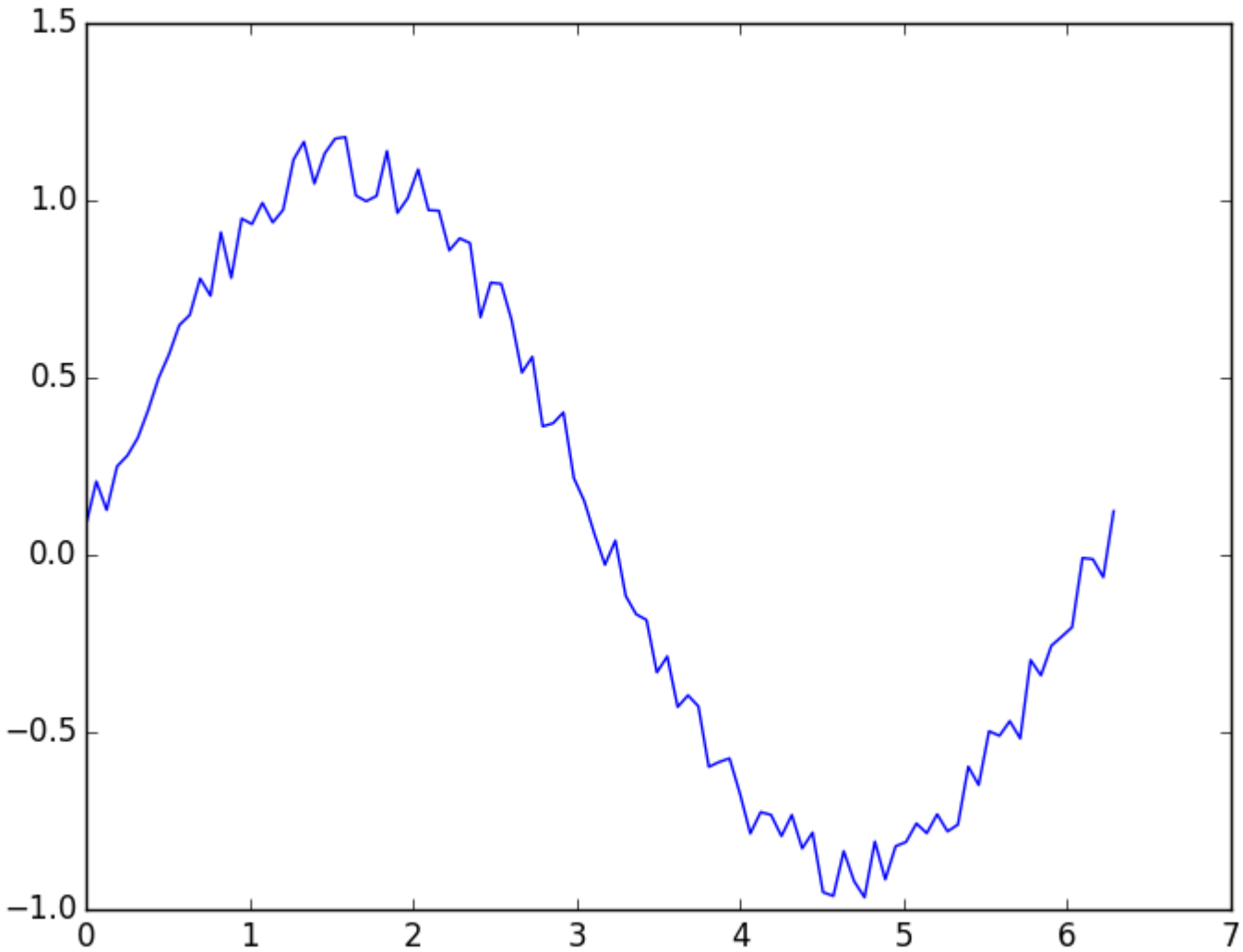
# Chapter 5: Smoothing a signal

## Examples

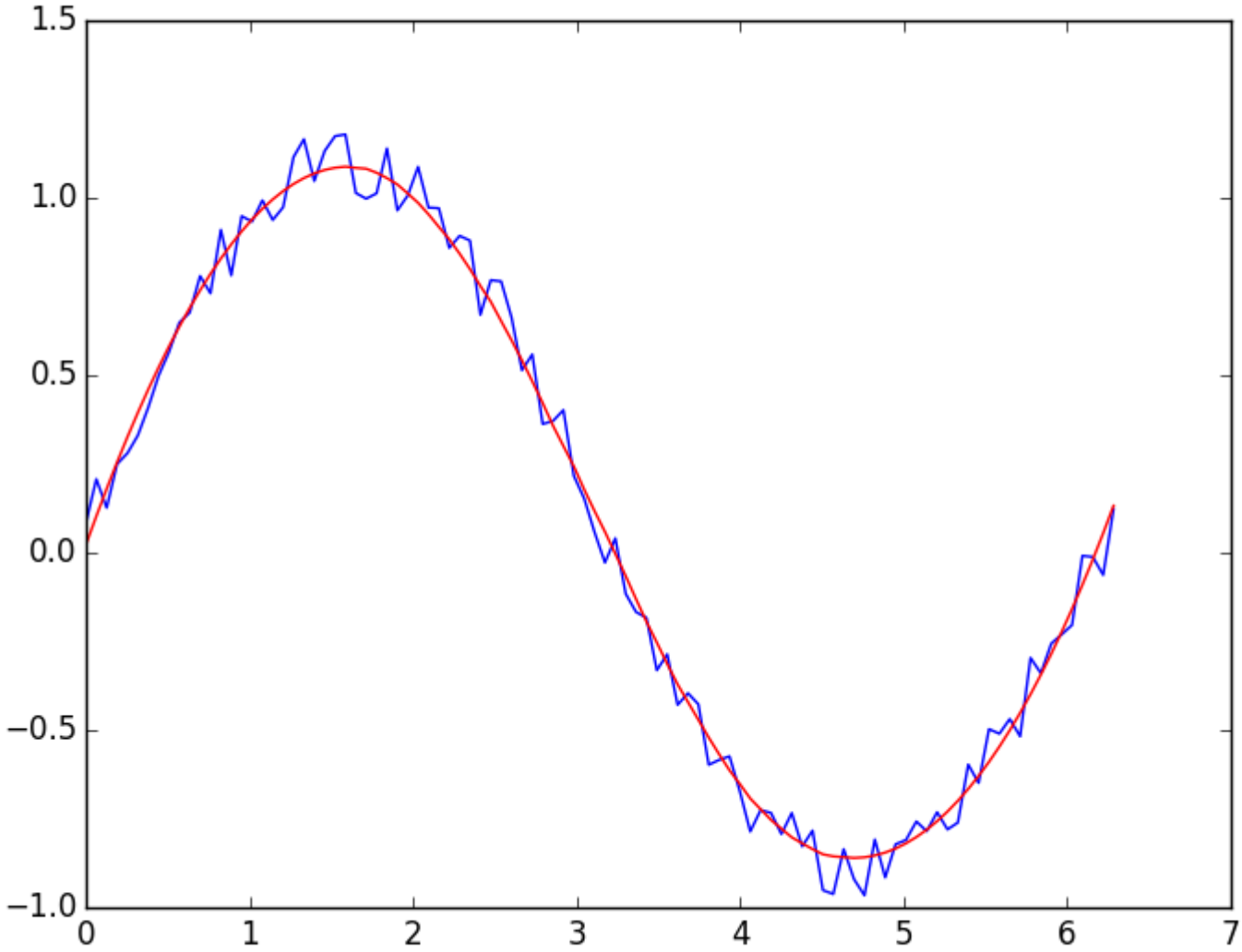
### Using a Savitzky–Golay filter

Given a noisy signal:

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1)

x = np.linspace(0,2*np.pi,100)
y = np.sin(x) + np.random.random(100) * 0.2

plt.plot(x,y)
plt.show()
```



one can smooth it using a Savitzky–Golay filter using the `scipy.signal.savgol_filter()` method:




```
import scipy.signal
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1)

x = np.linspace(0,2*np.pi,100)
y = np.sin(x) + np.random.random(100) * 0.2
yhat = scipy.signal.savgol_filter(y, 51, 3) # window size 51, polynomial order 3

plt.plot(x,y)
plt.plot(x,yhat, color='red')
plt.show()
```

Read Smoothing a signal online: https://riptutorial.com/scipy/topic/4535/smoothing-a-signal

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with scipy | 4444, AIB, Community, edwinksl, Rachel Gallen |
| 2 | Fitting functions with scipy.optimize curve_fit | ml4294 |
| 3 | How to write a Jacobian function for optimize.minimize | Rachel Gallen, Richard Fitzhugh |
| 4 | rv_continuous for Distribution with Parameters | Bill Bell |
| 5 | Smoothing a signal | Bill Bell, Franck Dernoncourt |