EBook Gratis

APRENDIZAJE scipy

Free unaffiliated eBook created from
Stack Overflow contributors.

#scipy

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con scipy

Observaciones

Acerca de Scipy

SciPy es una colección de algoritmos matemáticos y funciones de conveniencia construidas en la extensión Numpy de Python. Agrega un poder significativo a la sesión interactiva de Python al proporcionarle al usuario comandos y clases de alto nivel para manipular y visualizar datos. Con SciPy, una sesión interactiva de Python se convierte en un entorno de procesamiento de datos y prototipos de sistemas que compite con sistemas como MATLAB, IDL, Octave, R-Lab y SciLab.

El beneficio adicional de basar SciPy en Python es que también hace que un potente lenguaje de programación esté disponible para su uso en el desarrollo de programas sofisticados y aplicaciones especializadas. Las aplicaciones científicas que utilizan SciPy se benefician del desarrollo de módulos adicionales en numerosos nichos del panorama de software por parte de desarrolladores de todo el mundo. Todo desde programación paralela hasta subrutinas y clases de base de datos y web se ha puesto a disposición del programador de Python. Todo este poder está disponible además de las bibliotecas matemáticas en SciPy.

Versiones

Versión	Fecha de lanzamiento
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

Versión	Fecha de lanzamiento
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

Instalación o configuración

Scipy contiene partes escritas en C, C ++ y Fortran que deben compilarse antes de su uso. Por lo tanto, asegúrese de que estén instalados los compiladores necesarios y los encabezados de desarrollo de Python. Al haber compilado el código también significa que Scipy necesita pasos adicionales para importar desde las fuentes de desarrollo, que se explican a continuación.

Coloque una copia del repositorio principal de Scipy en Github en su propia cuenta, luego cree su repositorio local a través de:

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

Para crear la versión de desarrollo de Scipy y ejecutar pruebas, genere shells interactivos con las rutas de importación de Python configuradas correctamente, etc. Haz una de las siguientes:

```
$ 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
```

Esto construye primero a Scipy, por lo que puede llevar un tiempo la primera vez. Al especificar `-n` se ejecutarán las pruebas en la versión de Scipy (si existe) que se encuentra en el PYTHONPATH actual.

El uso de runtests.py es el enfoque recomendado para ejecutar pruebas. También hay una serie de alternativas a la misma, por ejemplo, la compilación in situ o la instalación en un entorno virtual. Algunas pruebas son muy lentas y deben habilitarse por separado.

Enlace a API

Ubuntu y Debian

Ejecutar comando

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

Las versiones en Ubuntu 12.10 o más nuevas y Debian 7.0 o más nuevas cumplen con la especificación actual de la pila SciPy. Los usuarios también pueden querer agregar el repositorio de NeuroDebian para paquetes adicionales de SciPy.

Convierte una matriz dispersa en una matriz densa usando 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]])
```

Versiones

El primer lanzamiento de SciPy, vsn 0.10, se lanzó el 14 de agosto de 2001. El lanzamiento actual de SciPy (correcto el 26 de julio de 2016) es v 0.17 (estable) con v .18 próximamente. Los detalles de los lanzamientos anteriores se enumeran aquí.

Manipulación de imágenes usando Scipy (tamaño de imagen básico)

SciPy proporciona funciones básicas de manipulación de imágenes. Estas incluyen funciones para leer imágenes del disco en matrices numpy, para escribir matrices numpy en el disco como imágenes y para redimensionar imágenes.

En el siguiente código, solo se usa una imagen. Es teñido, redimensionado y guardado. Las imágenes originales y resultantes se muestran a continuación:

```
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)
```

Referencia

Hola mundo básico

Cree un archivo (por ejemplo, hello_world.py) en un editor de texto o en un editor de python si tiene uno instalado (elija uno si no lo tiene : SublimeText, Eclipse, NetBeans, SciTe ... ¡hay muchos!)

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

Tenga en cuenta que las variables de Python no necesitan ser declaradas explícitamente; la declaración ocurre cuando asigna un valor con el signo igual (=) a una variable.

El resultado de las dos líneas de código anteriores es que se mostrará la cadena "Hello World".

Las funciones escritas en Python también se pueden usar en iPython.

En este caso, puede usar ejecutar el archivo guardado 'hello_world.py' en IPython así:

```
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
```

Si lo desea, puede usar dos variables, por ejemplo, una para hola y otra para world y concatenarlas usando el signo más (+):

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

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

Lea Empezando con scipy en línea: https://riptutorial.com/es/scipy/topic/2128/empezando-con-scipy

Capítulo 2: Ajustando funciones con scipy.optimize curve_fit

Introducción

El ajuste de una función que describe la ocurrencia esperada de puntos de datos a datos reales a menudo se requiere en aplicaciones científicas. Un posible optimizador para esta tarea es curve_fit de scipy.optimize. A continuación, se da un ejemplo de aplicación de curve_fit.

Examples

Ajustar una función a los datos de un histograma

Supongamos que hay un pico de datos distribuidos normalmente (gaussianos) (media: 3.0, desviación estándar: 0.3) en un fondo en descomposición exponencial. Esta distribución se puede ajustar con curve_fit en unos pocos pasos:

1.) Importar las bibliotecas requeridas.

2.) Defina la función de ajuste que se ajustará a los datos.

3.) Obtener datos del experimento o generar datos. En este ejemplo, se generan datos aleatorios para simular el fondo y la señal.

4.) Añadir la señal y el fondo.

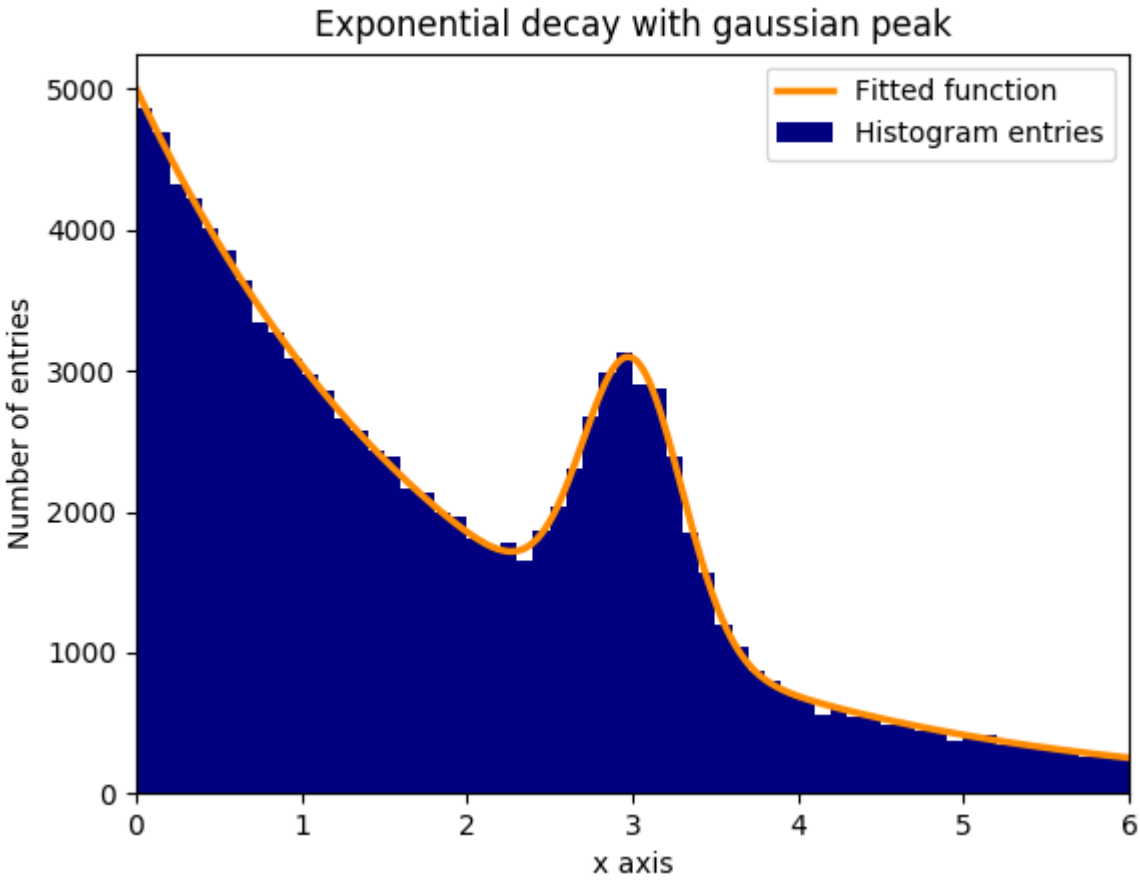
5.) Ajustar la función a los datos con curve_fit.

6.) (Opcionalmente) Graficar los resultados y los datos.

En este ejemplo, los valores de y observados son las alturas de los intervalos de histogramas, mientras que los valores de x observados son los centros de los `binscenters` histogramas (`binscenters`). Es necesario pasar el nombre de la función de ajuste, los valores x y los valores y a `curve_fit` . Además, con `p0` se puede proporcionar un argumento opcional que contiene estimaciones aproximadas para los parámetros de ajuste. `curve_fit` devuelve `popt` y `pcov` , donde `popt` contiene los resultados de ajuste para los parámetros, mientras que `pcov` es la matriz de covarianza, cuyos elementos diagonales representan la varianza de los parámetros ajustados.

```
# 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()
```



Lea Ajustando funciones con scipy.optimize curve_fit en línea:
https://riptutorial.com/es/scipy/topic/10133/ajustando-funciones-con-scipy-optimize-curve-fit

Capítulo 3: Cómo escribir una función jacobiana para optimizar.minimizar

Sintaxis

1. importar numpy como np
2. desde scipy.optimize import _minimize
3. desde scipy import especial
4. importar matplotlib.pyplot como plt

Observaciones

Tenga en cuenta el subrayado antes de 'minimizar' al importar desde scipy.optimize; '_minimize' Además, probé las funciones de este enlace antes de hacer esta sección y descubrí que tenía menos problemas / funcionó más rápido, si importaba 'especial' por separado. La función de Rosenbrock en la página vinculada era incorrecta: primero debe configurar la barra de colores; He publicado un código alternativo, pero creo que podría ser mejor.

Más ejemplos por venir.

Vea aquí para una explicación de Hessian Matrix

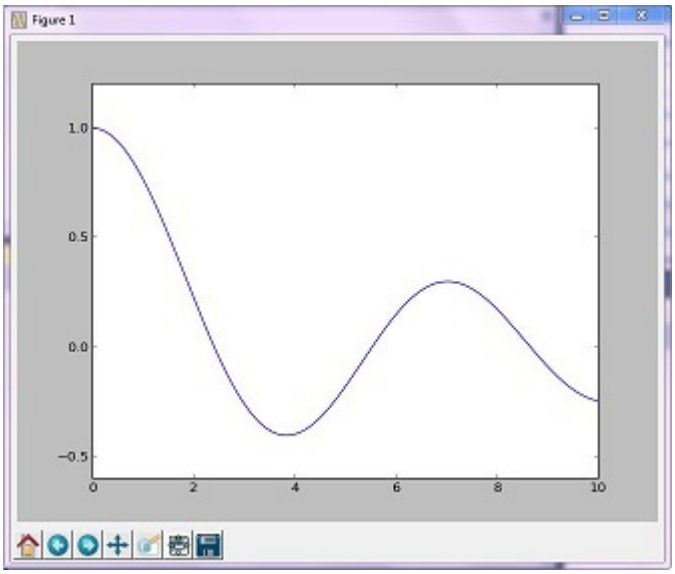
Examples

Ejemplo de optimización (dorado)

El método 'Golden' minimiza una función unimodal al reducir el rango en los valores extremos

```
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()
```

Imagen resultante

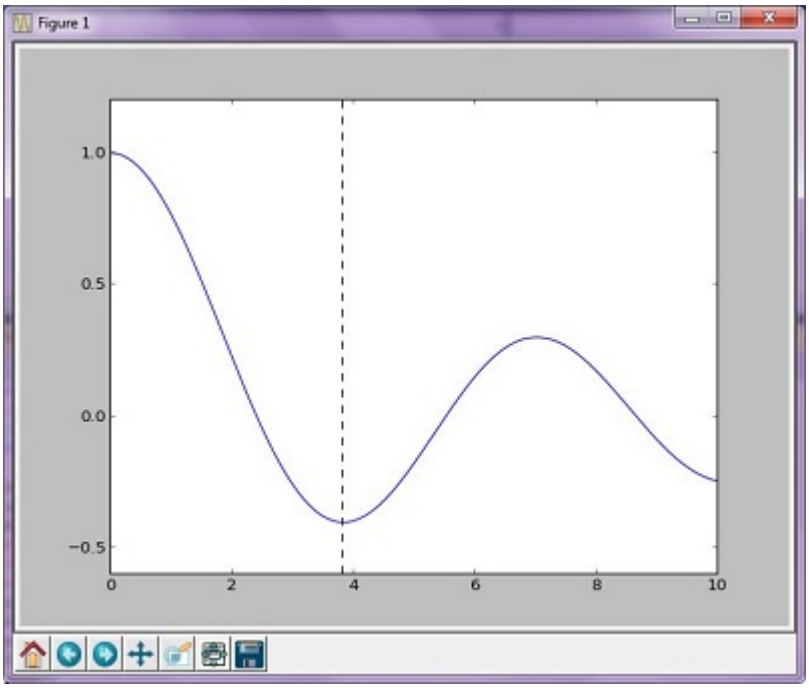


Ejemplo de optimización (Brent)

El método de Brent es una combinación de algoritmos más complejos de otros algoritmos de búsqueda de raíces; sin embargo, el gráfico resultante no es muy diferente del gráfico generado a partir del método de oro.

```
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))
```

Gráfica resultante



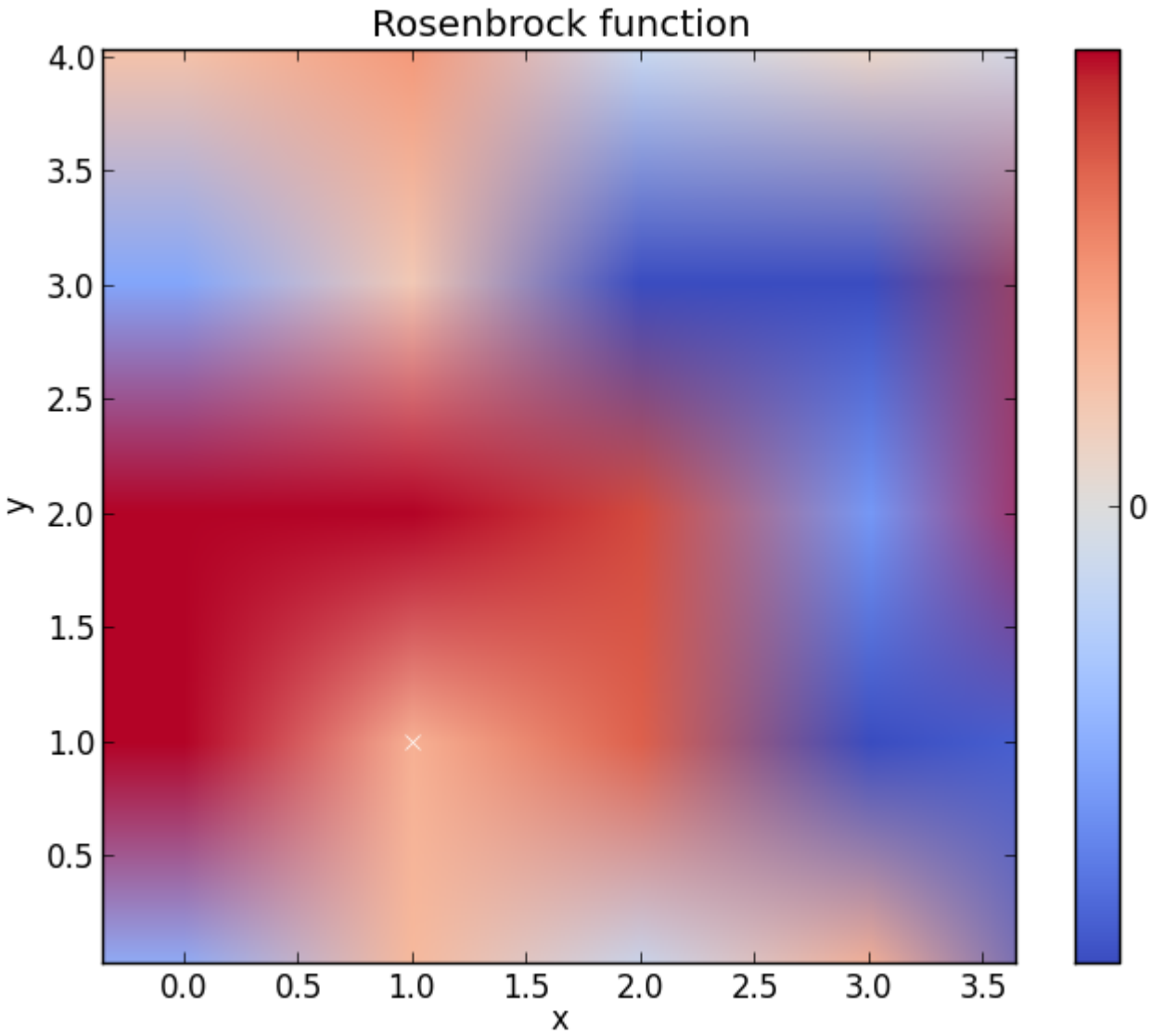
Salidas:

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

Función de Rosenbrock

Creo que este ejemplo podría ser mejor pero obtienes la esencia

```
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
```



Lea Cómo escribir una función jacobiana para optimizar.minimizar en línea: https://riptutorial.com/es/scipy/topic/4493/como-escribir-una-funcion-jacobiana-para-optimizar-minimizar

Capítulo 4: rv_continua para distribución con parámetros

Examples

Binomio negativo en reales positivos

```
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))
```

Es esencial definir _pdf o _cdf porque scipy infiere los parámetros de la otra función (que no define), y el orden de estos parámetros en cualquier función que realice, desde su definición. En este caso solo hay un parámetro de distribución, lambda. La variable que representa el valor de la variable aleatoria aparece primero en la definición de _pdf o _cdf.

Cuando defina solo una de estas funciones, scipy calculará la otra numéricamente. Para posible mayor eficiencia, defina ambos. De manera similar, defina _stats en términos de parámetros conocidos para la mejor eficiencia; De lo contrario scipy utiliza métodos numéricos.

Observe que la compatibilidad de la distribución se define cuando se crea **una** instancia de la clase (la variable **a** se establece en cero **yb** se establece en infinito de forma predeterminada), en lugar de cuando se subclasifica. Tenga en cuenta también que los parámetros de la distribución se establecen solo cuando se llama a una de las instancias de la clase, como en las últimas cinco líneas de código.

Lea rv_continua para distribución con parámetros en línea:
https://riptutorial.com/es/scipy/topic/6873/rv-continua-para-distribucion-con-parametros

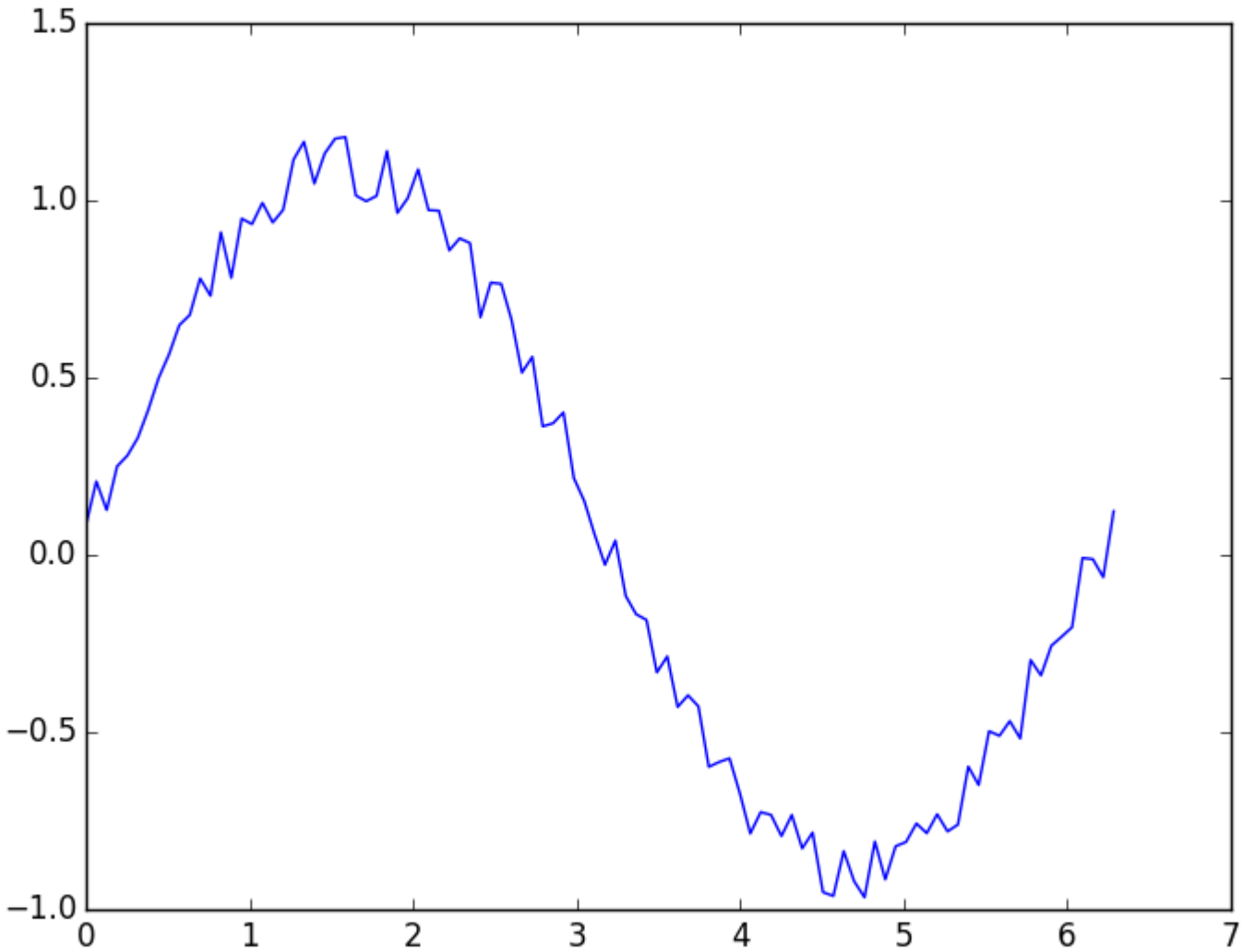
Capítulo 5: Suavizar una señal

Examples

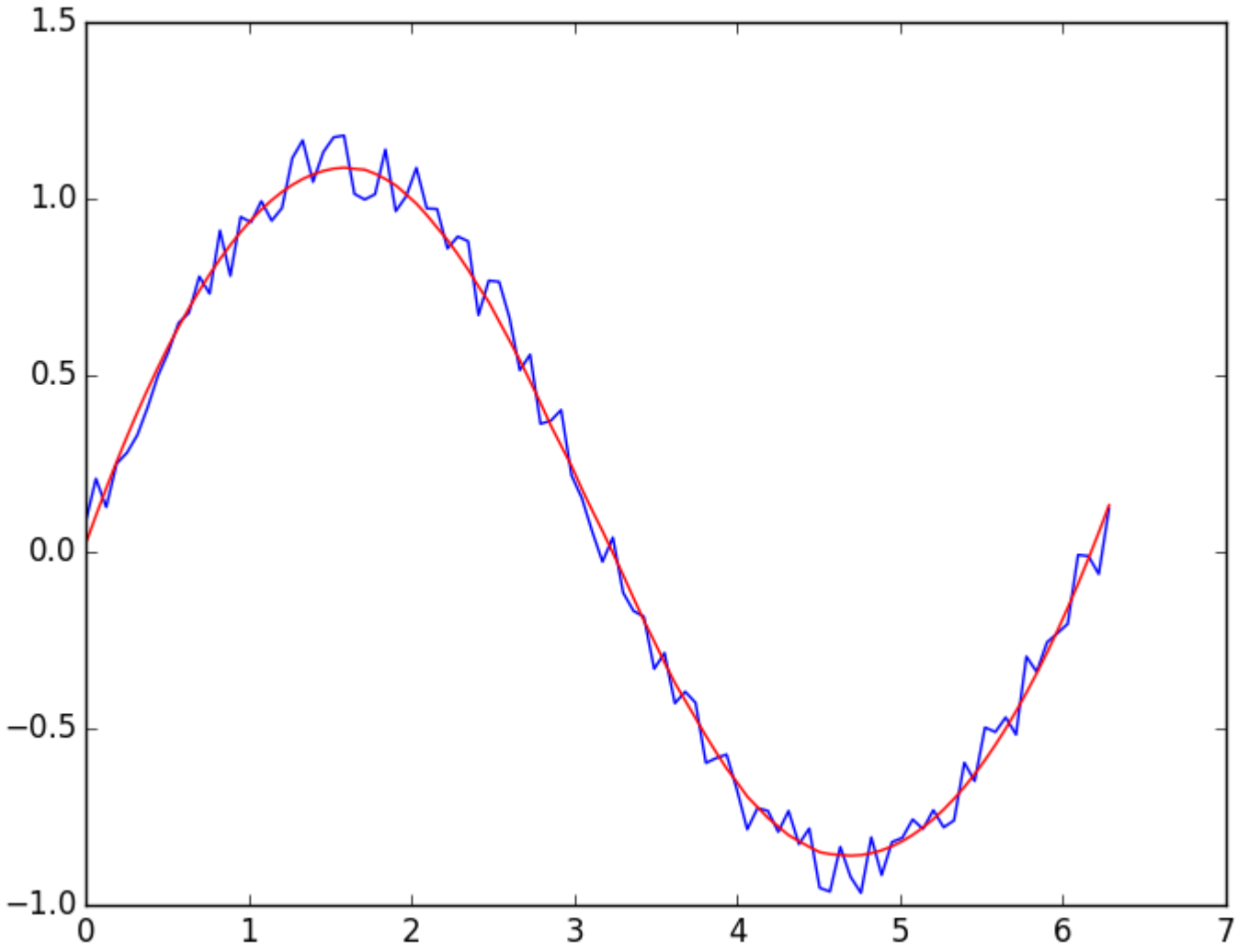
Usando un filtro Savitzky-Golay

Dada una señal ruidosa:

```
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()
```



uno puede suavizarlo usando un filtro Savitzky – Golay usando el método `scipy.signal.savgol_filter()` :




```
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()
```

Lea Suavizar una señal en línea: https://riptutorial.com/es/scipy/topic/4535/suavizar-una-senal

Creditos

S. No	Capítulos	Contributors
1	Empezando con scipy	4444, AIB, Community, edwinksl, Rachel Gallen
2	Ajustando funciones con scipy.optimize curve_fit	ml4294
3	Cómo escribir una función jacobiana para optimizar.minimizar	Rachel Gallen, Richard Fitzhugh
4	rv_continua para distribución con parámetros	Bill Bell
5	Suavizar una señal	Bill Bell, Franck Dernoncourt