Curso Python 3 Científico: **SCIPY**

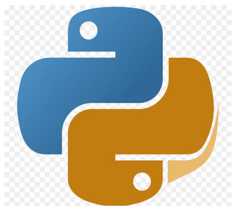
Módulo 3

César Husillos Rodríguez

IAA-CSIC

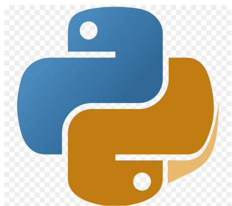
30 sep - 4 oct 2019

Índice



1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. Constantes
10. Agrupamiento

Índice



1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. Constantes
10. Agrupamiento

¿Qué es SCIPY?

- **Paquete** que extiende la funcionalidad de NUMPY.
- SCIPY implementa módulos que realizan tareas de **optimización**, **integración** , **agrupamientos**, **álgebra lineal** y que dan acceso a **funciones especiales**, **distribuciones de probabilidad** y **constantes científicas**, entre otros.

La última versión estable es la `v1.3.1` *(08/08/2019)*.

scipy.__version__

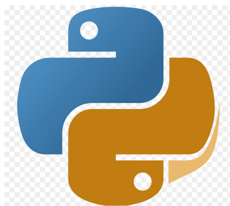
Subpaquetes SCIPY

Subpaquete	Descripción
cluster	Algoritmos de clustering
constants	Constantes físicas y matemáticas
fftpack	Rutinas de Transformadas Rápidas de Fourier
integrate	Para integrar y resolver ecuaciones diferenciales ordinarias
interpolate	Rutinas de interpolación.
io	Entrada y salida (Input/Output)
linalg	Álgebra Lineal
maxentropy	Métodos de Máxima Entropía
ndimage	Procesado de imágenes n-dimensionales
odr	Ajustes a distancias ortogonales (Orthogonal Distance Regression)
optimize	Procedimientos de optimización y búsqueda de ceros en funciones
signal	Procesado de señal
sparse	Matrices dispersas y rutinas asociadas
spatial	Datos con estructura espacial y algorimos
special	Funciones especiales
stats	Distribuciones estadísticas y funciones
weave	Integración con C/C++

¿Qué vamos a aprender?

1. Determinar los valores mínimos de funciones.
2. Localizar las raíces de una función.
3. Interpolación n-dimensional.
4. Ajuste de curvas.
5. Integración de funciones multivariables.
6. Estadística descriptiva.
7. Manejar distribuciones estadísticas. Verificar qué parámetros de una distribucción se ajustan a una muestra experimental.
8. Realizar operaciones sobre matrices.
9. Resolver sistemas de ecuaciones lineales.
10. Clasificación de datos.
11. Acceso a constantes físicas.

Índice



1. Introducción
2. **Búsqueda de mínimos**
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. Constantes
10. Agrupamiento

Búsqueda de mínimos

scipy.optimize

Paquete **scipy.optimize**

```
minimize(fun, x0, args=(), method='BFGS',
jac=None, hess=None, hessp=None, bounds=None,
constraints=(), tol=None, callback=None,
options=None)
```

Proporciona una **interfaz común** para todos los **algoritmos de minimización** que se apliquen sobre **funciones escalares multivariables**.

Parámetros

minimize

Parámetro	Tipo	Descripción
fun	Función	El nombre de una función.
x0	ndarray	Punto inicial al partir del que se busca el mínimo.
args	tupla	Parámetros extra si los necesitan la fun y sus derivadas.
method	cad. caracteres	Método de búsqueda de mínimo. Posibles: 'Nelder-Mead', 'Powell', 'CG', 'BFGS', 'Newton-CG', 'Anneal', 'L-BFGS-B', 'TNC','COBYLA' y 'SLSQP'.
jac	Booleano o función	Si es True se supone que fun devuelve el valor del Jacobiano junto al de la función. Si es False, jac se estima numéricamente. Puede ser una función, en cuyo caso recibe los mismos parámetros que fun.
hess	Función	Hessiano de fun. (Sólo para Newton-CG.)
hessp	Función	Hessiano de fun por un vector arbitrario p. (Newton-CG.)
bounds	secuencia	(Sólo para L-BFGS-B, TNC, COBYLA y SLSQP) Límites para las variables. Por cada variable de fun, un par (min|"min"|None, max|"max"|None)
constraints	dict o seq. of dict	(Consultar documentación).
tol	float	Tolerancia por debajo de la cual se detienen las iteraciones.
options	diccionario	Las opciones aceptadas por todos los métodos son: maxiter (int); disp (bool) Si True, info de convergencia.

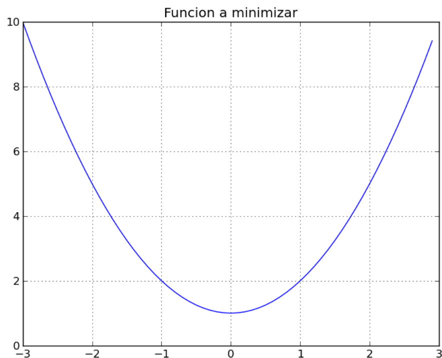
Ejemplo 1

Minimización

Ejemplo

```
# Funcion a minimizar
import math
def parabola(x):
    return math.pow(x, 2) +

# jacobiano de la funcion
def parabola_derivada(x):
    return 2 * x
```



Método SIMPLEX (Nelder-Mead)

Ejemplo 1

Es la forma más simple de determinar el mínimo de una función. Sólo necesita de una función a evaluar. Es lento.

```
>>> res = minimize(parabola, 202.34, method='nelder-mead', \
    options={'xtol': 1e-8, 'disp': True})
Optimization terminated successfully.
         Current function value: 1.000000
         Iterations: 38
         Function evaluations: 76
>>> res
  status: 0
    nfev: 76
 success: True
     fun: 1.0
       x: array([ -9.09494702e-13])
 message: 'Optimization␣terminated␣successfully.'
     nit: 38
```

Método Broyden-Fletcher-Goldfarb-Shanno

Ejemplo 1

Converge más rápidamente porque usa el gradiente de la función. Si no es dado, se estima.

Algoritmo de BFGS

```
>>> res = minimize(parabola, 202.34, method='BFGS', \
    options={'disp': True})
Optimization terminated successfully.
         Current function value: 1.000000
         Iterations: 4
         Function evaluations: 15
         Gradient evaluations: 5
```

Método Newton del gradiente conjugado

Ejemplo 1

Algoritmo de Newton-CG

```
>>> res = minimize(parabola, 202.34,method='Newton-CG', \
    jac=parabola_derivada, options={'xtol': 1e-8, 'disp': True})
Optimization terminated successfully.
         Current function value: 1.000000
         Iterations: 3
         Function evaluations: 4
         Gradient evaluations: 7
         Hessian evaluations: 0
```

Más rápido aún. Posibilidad de insertar el Hessiano de la función.

Minimización

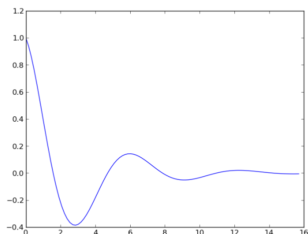
scipy.optimize

Pregunta...

¿Qué sucede si hay mínimos locales?

Ejemplo

```
def coseno_amortiguado(x):
    return math.cos(x) * \
    math.exp(- x / math.pi)
```



Minimización

scipy.optimize

Ejemplo

```
def cos_amort_jacobiano(x):
    return -1 * (math.sin(x) +
math.cos(x) / math.pi) * math.exp(- x / math.pi)

def cos_amort_hessiana(x):
    return math.exp(-x/math.pi)*((1/math.power(math.pi,2)-1) \
    * math.cos(x)+2*math.sin(x)/math.pi)
```

Minimización

Mínimos locales

```
>>> res = scipy.optimize.minimize(coseno_amortiguado, \
    scipy.pi/2,method='Newton-CG', \
    jac=cos_amort_jacobiano, hess=cos_amort_hessiano, \
    options={'xtol': 1e-8, 'disp': True})
>>> res
Optimization terminated successfully.
         Current function value: -0.386678
         Iterations: 6
         Function evaluations: 76
         Gradient evaluations: 6
         Hessian evaluations: 6
```

Minimización

Mínimos locales

```
>>> res1 =scipy.optimize.minimize(coseno_amortiguado, \
    3.22 * scipy.pi, method='Newton-CG',
    jac=cos_amort_jacobiano, hess=cos_amort_hessiano, \
    options={'xtol': 1e-8, 'disp': True})
>>> res1
Optimization terminated successfully.
         Current function value: -0.052331
         Iterations: 2
         Function evaluations: 45
         Gradient evaluations: 2
         Hessian evaluations: 2
```

Minimización

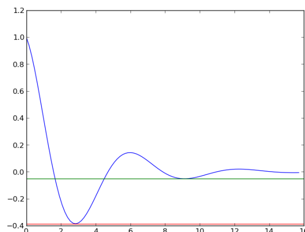
Mínimos locales

Pregunta...

¿Qué sucede si hay mínimos locales?

Respuesta

Depende del punto inicial para la evaluación de la función.

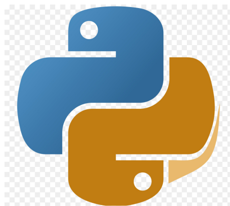


Ejercicios

Ejercicios 1 y 2.

http://python.iaa.es/content/ejercicios-curso-científico

Índice



1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. Constantes
10. Agrupamiento

Búsqueda de raíces

(`scipy.optimize`)

1. Para una función escalar.
2. Para un conjunto de ecuaciones.

Búsqueda de raíces
(scipy.optimize)

```
scipy.optimize.root(fun, x0, args=())
```

Permite resolver ecuaciones escalares y sistemas de ecuaciones (lineales o no).

Parámetro	Tipo	Descripción
fun	Función	El nombre de una función.
x0	ndarray	Punto inicial al partir del se busca la raíz.
args	tupla	Parámetros extra si los necesitan la fun y sus derivadas.

Búsqueda de raíces

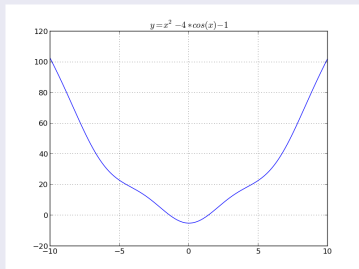
Ecuación escalar

Dada la función

$$f(x) = x^2 - 4 * cos(x) - 1$$

se trata de resolver la ecuación

$$f(x) = x^2 - 4 * cos(x) - 1 = 0$$



Búsqueda de raíces

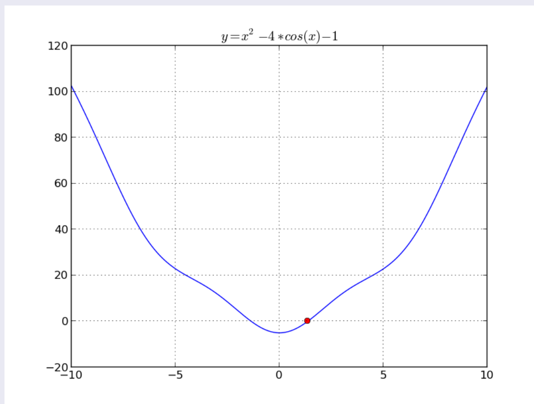
Ecuación escalar

```
>>> import scipy
>>> import scipy.optimize
>>> def nonlineq(x):
...     return x ** 2 - 4 * math.cos(x) - 1
...
>>> res = scipy.optimize.root(nonlineq, 2)
>>> res
  status: 1
 success: True
     qtf: array([ -6.64964233e-08])
    nfev: 7
       r: array([-6.62606427])
     fun: array([  2.50466314e-13])
       x: array([ 1.35809235])
 message: 'The␣solution␣converged.'
    fjac: array([[-1.]])
>>> print res['x']
[ 1.35809235]
```

Búsqueda de raíces

Ecuación escalar

Pinto la solución...

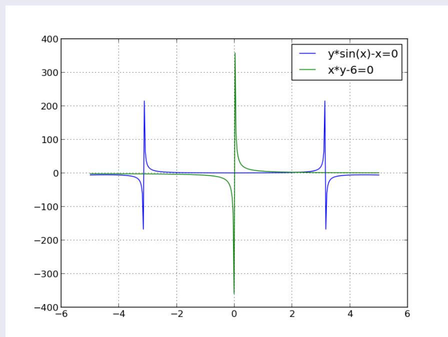


Búsqueda de raíces

Sistema de ecuaciones

Determinar los valores de x e y que cumplen

$$y * sin(x) - x = 0$$
$$x * y - 6 = 0$$



Búsqueda de raíces

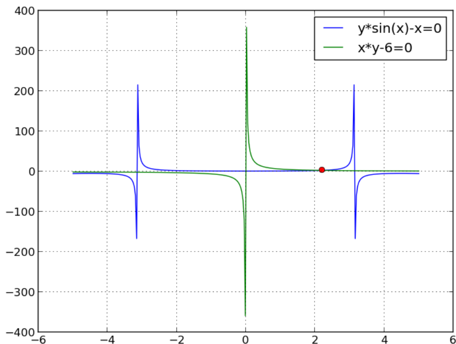
Sistema de ecuaciones

```
>>> import scipy
>>> import scipy.optimize
>>> def func_2var(x):
...     return [x[1] * math.sin(x[0]) - x[0], x[0] * x[1] - 6]
...
>>> sol = scipy.optimize.root(func_2var, [2, 3])
>>> print sol
  status: 1
 success: True
     qtf: array([ -2.16014189e-10,  -4.83007432e-11])
    nfev: 9
       r: array([ 3.76235438,  1.04271611, -2.1105447 ])
     fun: array([ -1.33848488e-12,  -5.95079541e-13])
       x: array([ 2.20138341,  2.72555883])
 message: 'The␣solution␣converged.'
    fjac: array([[-0.68619494,  0.7274177 ],
       [-0.7274177 , -0.68619494]])
>>> >>> sol['x']
array([ 2.20138341,  2.72555883])
```

Búsqueda de raíces

Ecuación escalar

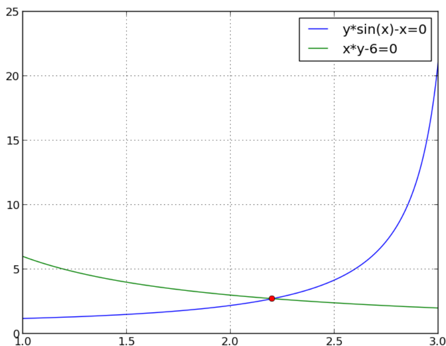
Pinto la solución...



Búsqueda de raíces

Ecuación escalar

Hago zoom en la zona de la solución

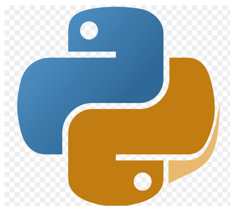


Ejercicios

Ejercicio 3.

http://python.iaa.es/content/ejercicios-curso-científico

Índice



1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. Constantes
10. Agrupamiento

Integración

scipy.integrate

El subpaquete

`scipy.integrate`

proporciona rutinas para integrar funciones de una o varias variables. También para integrar ecuaciones diferenciales ordinarias.

Integración en una variable

`scipy.integrate.quad`

La función

```
quad(func, a, b, args=())
```

sirve para integrar la función `func` de una variable entre dos puntos (`a` y `b`). `args` es una tupla con parámetros adicionales para `func`.
Los puntos pueden ser `scipy.inf` para indicar los límites infinitos.
Retorna una tupla:

```
(valor de la integral, estimación del error
          absoluto en el resultado)
```

Integración de una función de una variable

scipy.integrate.quad

```
>>> def const(x):
...     return 1
...
>>> import scipy.integrate
>>> res = scipy.integrate.quad(const, 0, 1)
>>> print(res)
(1.0, 1.1102230246251565e-14)
```

Integración de una muestra x-y

scipy.integrate.simps

La función

```
simps(y, x=None, dx=1, even='avg')
```

Integra la muestra `y(x)` mediante la regla de Simpson.
Si `x=None`, tiene en cuenta el intervalo `dx`.
Posibles valores de `even`: `avg`, `first` y `last`.
Retorna un `float` con el valor de la integral.

Integración de una muestra x-y

scipy.integrate.simps

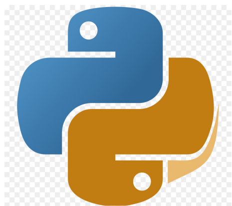
```
>>> from scipy.integrate import simps
>>> res = simps(scipy.ones(5), scipy.arange(5))
>>> print(res)
4.0
```

Ejercicios

Ejercicios 4 y 5.

`http://python.iaa.es/content/ejercicios-curso-científico`

Índice



1 Introducción
2 Búsqueda de mínimos
3 Búsqueda de raíces
4 Integración
5 Interpolación
6 Estadística y Distribuciones
7 Ajuste de curvas
8 Álgebra lineal
9 Constantes
10 Agrupamiento

Interpolación

scipy.interpolate

Definiciones

- “En el subcampo matemático del **análisis numérico**, se denomina interpolación a la obtención de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos.”
- “En **ingeniería** y **algunas ciencias** es frecuente disponer de un cierto número de puntos obtenidos por muestreo o a partir de un experimento y pretender construir una función que los ajuste.”

(Obtenidas de la Wikipedia (textithttp://es.wikipedia.org/wiki/Interpolación))

Interpolación

`scipy.interpolate`

Veremos ejemplos de

1. Interpolación-1D.
2. Interpolación de datos multivariables (2D).
3. Interpolación con *splines*:
   - 1D.
   - 2D.

Interpolación-1D

scipy.interpolate.interp1d

```
scipy.interpolate.interp1d(x, y, kind='linear')
```

Parámetro	Tipo	Descripción
`x`	array	Array de números ordenados de forma monótona creciente.
`y`	array	Array o matríz de valores reales. El tamaño de `y` a lo largo del eje de interpolación debe ser el mismo que el de `x`.
`kind`	string o entero	Si es `string` puede tomar valores entre: 'linear','nearest', 'zero', 'slinear', 'quadratic' o 'cubic'. Si es `entero` especifica el orden del *spline* usado para interpolar.

.

Retorna una **función**. Cuando se la llama, realiza la interpolación.

Ejemplo 1

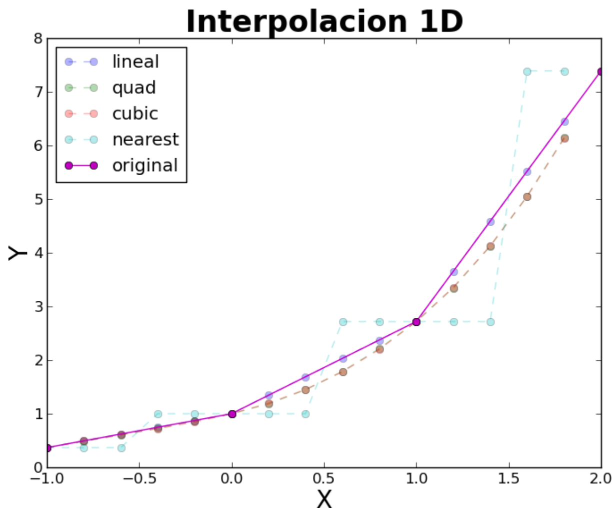
Interpolación-1D

Código del ejemplo

```
>>> from scipy.interpolate import interp1d
>>> x = scipy.array([-1, 0, 1, 2])
>>> y = scipy.exp(x)
>>> flin = interp1d(x, y)
>>> fqua = interp1d(x, y,kind='quadratic')
>>> fcub = interp1d(x, y,kind='cubic')
>>> fnea = interp1d(x, y,kind='nearest')
>>> newx = scipy.arange(-1, 2, 0.2)
>>> newylin = flin(newx)
>>> newyqua = fqua(newx)
>>> newycub = fcub(newx)
>>> newynea = fnea(newx)
```

Ejemplo 1

Interpolación-1D



Interpolación-2D

scipy.interpolate.griddata

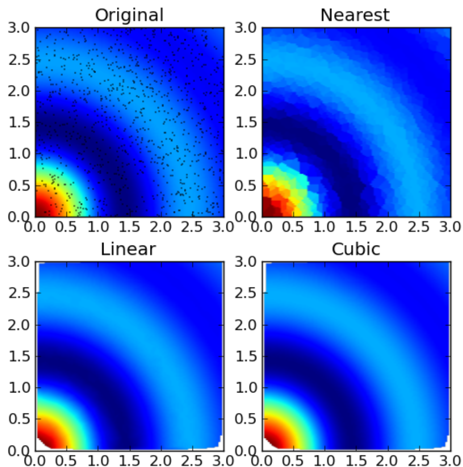
```
scipy.interpolate.griddata(points, values, xi,
        method='linear', fill_value=nan)
```

Parámetro	Tipo	Descripción
points	array	Coordenadas de los puntos. Puede ser un array de tamaño (n, D), o una tupla de ndim arrays.
values	array	Array de float o complex y dimensión (n,).
xi	array	Puntos a los cuales interpolar los datos. Shape (M,D).
method	string	A elegir entre 'linear', 'nearest' o 'cubic'. (Ver documentación).
fill_value	float	Valor usado para asignar los puntos fuera del dominio convexo dado por points. No afecta al método "nearest".

Devuelve un array con los valores interpolados.

Ejemplo1

Interpolación-2D



Ejemplo1

Interpolación-2D

```
>>> import scipy
>>> from scipy.interpolate import griddata
>>> def func(x, y):
...     return scipy.sinc(scipy.sqrt(scipy.power(x, 2) + \
...     scipy.power(y, 2)))
>>> intvx = scipy.linspace(0, 3, 100)
>>> intvy = scipy.linspace(0, 3, 200)
>>> grid_x, grid_y = scipy.meshgrid(intvx, intvy)
>>> points = scipy.random.rand(1000, 2) * 3
>>> values = func(points[:,0], points[:,1])
>>> grid_z0 = griddata(points, values, (grid_x, grid_y), \
method='nearest')
>>> grid_z1 = griddata(points, values, (grid_x, grid_y), \
method='linear')
>>> grid_z2 = griddata(points, values, (grid_x, grid_y), \
method='cubic')
```

Ejemplo1 (continuación)

Interpolación-2D

```
>>> # plotting routines
>>> pyplot.subplot(221)
>>> pyplot.imshow(func(grid_x, grid_y).T, extent=(0,3,0,3), \
origin='lower')
>>> pyplot.plot(points[:,0], points[:,1], 'k.', ms=1)
>>> pyplot.title('Original')
>>> pyplot.subplot(222)
>>> pyplot.imshow(grid_z0.T, extent=(0,3,0,3), origin='lower')
>>> pyplot.title('Nearest')
>>> pyplot.subplot(223)
>>> pyplot.imshow(grid_z1.T, extent=(0,3,0,3), origin='lower')
>>> pyplot.title('Linear')
>>> pyplot.subplot(224)
>>> pyplot.imshow(grid_z2.T, extent=(0,3,0,3), origin='lower')
>>> pyplot.title('Cubic')
>>> pyplot.gcf().set_size_inches(6, 6)
>>> pyplot.show()
```

Splines

Una función `spline` está formada por varios polinomios, cada uno definido sobre un subintervalo, que se unen entre sí obedeciendo a ciertas condiciones de continuidad.

Supongamos que disponemos de $n+1$ puntos, a los que denominaremos **nudos**, tales que $t_0 < t_1 < ... < t_n$. Supongamos además que se ha fijado un entero $k > 0$. Decimos entonces que una función spline de **grado k** con nudos en $t_0, t_1, ..., t_n$ es una función S que satisface las condiciones:

1. en cada intervalo (t_{i-1}, t_i), S es un polinomio de grado menor o igual a k.
2. S tiene una derivada de orden $(k-1)$ continua en $[t_0, t_n]$.

Splines-1D

scipy.interpolate.splrep

Generación del *spline* a partir de una muestra de puntos.

```
scipy.interpolate.splrep(x, y, xb=None, xe=None, k=3)
```

En el que, dado un conjunto de puntos $(x[i], y[i])$ se aproxima a un spline de grado k en en intervalo $xb <= x <= xe$.

Devuelve los coeficientes, c, y los nudos, t.

Splines-1D

scipy.interpolate.splev

Interpolación a cualquier punto dentro del rango definido por el spline.

```
scipy.interpolate.splev(x, tck, der=0, ext=0)
```

Evalúa el spline (der=0) o sus derivadas (*der* > 0), según el spline "tck" en los puntos dados por el array "x".

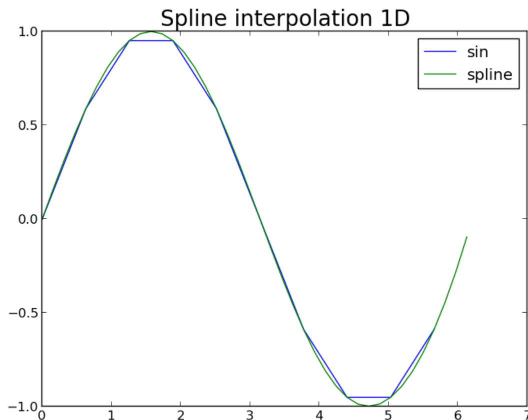
Splines-1D

Ejemplo 1

```
>>> import scipy
>>> from scipy import interpolate
>>> x = scipy.arange(0, 2 * scipy.pi, scipy.pi / 5)
>>> y = scipy.sin(x)
>>> tck = interpolate.splrep(x, y, s=0)
>>> newx = scipy.arange(0, 2 * scipy.pi, scipy.pi / 20)
>>> newy = interpolate.splev(newx, tck, der=0)
```

Splines-1D

Ejemplo 1



Ejercicios

Ejercicios 6 y 7.

`http://python.iaa.es/content/ejercicios-curso-científico`

Índice

1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. Constantes
10. Agrupamiento

Estadística descriptiva

Valores centrales y medidas de dispersión

SCIPY usa a NUMPY adoptando las funciones de las que éste dispone para caracterizar medidas de valores centrales y dispersión de un array cualquiera.

Función	**Descripción**
`scipy.average(a, axis=None, weights=None)`	Devuelve un escalar o array con la media "pesada" del array *a* por los valores "weights" en el eje "axis" seleccionado.
`scipy.mean(a, axis=None)`	Devuelve un escalar o array con la media aritmética del array sobre el "`axis`" dado.
`scipy.median(a, axis=None)`	Devuelve un escalar o array con la mediana del array para el eje seleccionado.
`scipy.std(a, axis=None)`	Devuelve un escalar o array con la desviación estándar en el "`axis`".
`scipy.var(a, axis=None)`	Devuelve un escalar o array con la varianza de los elementos en "`axis`" eje seleccionado.

Distribuciones de probabilidad

scipy.stats

SCIPY dispone de un número enorme de distribuciones de probabilidad, tanto discretas (12 distribuciones) como contínuas (unas 84).
(Consultar documentación)

Entre las posibilidades más atractivas están

1. Capacidad de generar muestras aleatorias de cualquiera de esas casi 100 disponibles.
2. Dada una muestra aleatoria obtenida experimentalmente, determinar los parámetros de una determinada distribución que mejor ajustan.

Distribuciones de probabilidad

scipy.stats

De los métodos disponibles de las distribuciones, los que más nos interesan son:

Método	Descripción
`rvs(loc=0, scale=1, size=1)`	Genera una muestra aleatoria de esta distribución. El tamaño viene dado `size`.
`pdf(x, loc=0, scale=1)`	Función Densidad de Probabilidad.
`cdf(x, loc=0, scale=1)`	Función de distribución acumulativa.
`ppf(q, loc=0, scale=1)`	Percentil.
`stats(loc=0, scale=1, moments='mv')`	Devuelve la media, varianza, sesgo (de Fisher) o kurtosis (de Fisher), según el valor de `moments`: media('m'), varianza('v'), sesgo('s'), y kurtosis('k')
`fit(data, loc=0, scale=1)`	Estima parámetros de la distribución a partir de `data`.

Distribuciones de probabilidad

scipy.stats

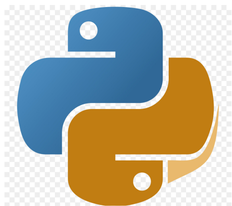
```
>>> import scipy.stats
>>> print scipy.stats.norm.cdf(scipy.array([-2, 0, 2]))
[ 0.02275013  0.5         0.97724987]
>>> print scipy.stats.norm.stats(moments="mvsk")
(array(0.0), array(1.0), array(0.0), array(0.0))
>>> print scipy.stats.norm.ppf(0.5)  # mediana = percentil 50
0.0
>>> print scipy.stats.norm.rvs(size=3)  # muestra de 3 elementos
[ 1.37629104  0.24356188 -1.7217918 ]
>>> sample = scipy.stats.norm.rvs(loc=3, scale=2, size=3)
>>> print sample
[ 4.79340559  3.46683487  4.74428621]
>>> print sample.mean(), sample.std(), sample.var()
4.33484222524 0.614101380386 0.377120505392
>>> a = scipy.stats.norm.rvs(loc=4, scale=0.5, size=1000)
>>> norm = scipy.stats.norm
>>> print norm.fit(a)
(4.014113735638515, 0.4946152010101 4156)
```

Ejercicios

Ejercicios 8 y 9.

`http://python.iaa.es/content/ejercicios-curso-científico`

Índice



1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. **Ajuste de curvas**
8. Álgebra lineal
9. Constantes
10. Agrupamiento

Ajuste de curvas

scipy.optimize.curve_fit

La función

```
scipy.optimize.curve_fit(func, xdata, ydata, p0=None)
```

emplea un ajuste por mínimos cuadrados no lineales para establecer los parámetros de la función `func` que mejor se ajustan a los datos `(xdata, ydata)`.
El parámetro `p0` es una tupla que representa los valores iniciales para los parámetros de `func`.
Devuelve una tupla de 2 elementos:

- el primer elemento es un array con los parámetros de la función que minimizan la suma cuadrática de los errores.
- la matriz de covarianza de la solución. La diagonal corresponde a las varianzas de cada uno de los parámetros.

Ajuste de curvas

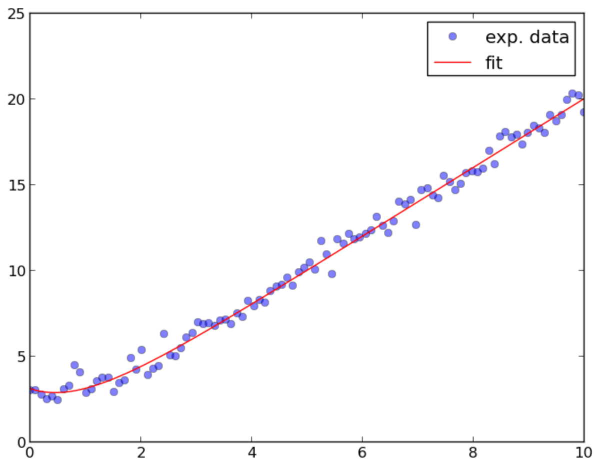
scipy.optimize.curve_fit

Ajuste la función con ruido normal gausiano escalado por 0.5 a la función $y = 3 \cdot e^{-x} + 2 \cdot x$.

```
>>> def func_fit(x, a, b):
...     return a * scipy.exp(-x) + b * x
...
>>> x = scipy.linspace(0, 10, 100)
>>> y = 3 * scipy.exp(-x) + 2 * x
>>> noise = scipy.randn(100) * 0.5
>>> y += noise
>>> res = scipy.optimize.curve_fit(func_fit, x, y)
>>> print res
(array([ 2.97253275,  1.99997092]), array([[  3.61334133e-02,
-1.06635306e-04],
       [ -1.06635306e-04,   5.89627071e-05]]))
```

Ajuste de curvas

scipy.optimize.curve_fit

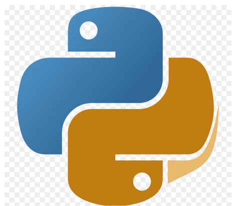


Ejercicios

Ejercicio 10.

http://python.iaa.es/content/ejercicios-curso-científico

Índice



1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. **Álgebra lineal**
9. Constantes
10. Agrupamiento

Determinante de una matriz

scipy.linalg.det

```
scipy.linalg.det(ndarray)
```

```
>>> from scipy import linalg
>>> A = scipy.arange(1, 5).reshape((2,2))
>>> A
array([[1, 2],
       [3, 4]])
>>> linalg.det(A)
-2.0
```

Resolución de sistemas lineales

scipy.linalg.solve

scipy.linalg.solve(ndarray)

Dado el sistema lineal

$x + y + 2z = 3$

$2x - y + 3z = 0$

$x - 6y - z = 6$

Se toma por matriz A los coeficientes de x, y y z, y por vector b el de los términos independientes.

```
>>> from scipy import linalg
>>> import scipy
>>> A = scipy.array([[1, 1, 2], [2, -1, 3], [1, -6, -1]])
>>> b = scipy.array([[3, 0, 6]])
>>> solucion = linalg.solve(A, scipy.transpose(b))
>>> solucion
array([[ 43.5],
       [ 10.5],
       [-25.5]])
```

Autovalores y autovectores

scipy.linalg.

La función

```
scipy.linalg.eig(A)
```

realiza la descomposición del array (N x N) dimensional A.
Devuelve una tupla con dos elementos:

```
(valores_propios, vectores_propios)
```

Autovalores y autovectores

scipy.linalg.

```
>>> import scipy.linalg
>>> A = scipy.array([[2, 4], [1, 3]])
>>> eigval, eigvect = scipy.linalg.eig(A)
>>> eigval[0], eigval[1]
((0.43844718719116971+0j), (4.5615528128088307+0j))
>>> eigvect[:,0]  # eigenvectores en columnas
array([-0.93153209,  0.36365914])
>>> eigvect[:,1]
array([-0.84212294, -0.5392856 ])
>>> mod_eigvect1 = scipy.dot(eigvect[:,0],eigvect[:,0].T)
>>> mod_eigvect1  # modulo del vector propio 1
1.0
>>> mod_eigvect2 = scipy.dot(eigvect[:,1],eigvect[:,1].T)
>>> mod_eigvect2  # modulo del vector propio 2
0.99999999999999978
```

Álgebra lineal

scipy.linalg

- `scipy.linalg` contiene y amplía las funcionalidades de `numpy.linalg`.
- las rutinas están construidas sobre las librerías optimizadas `ATLAS`, `LAPACK` y `BLAS`, lo cual garantiza el resultado y un rendimiento en operaciones espectacular.

Producto escalar de matrices

scipy.dot

```
scipy.dot(array1, array2)
```

Las dimensiones han de ser compatibles.

```
>>> from scipy import linalg
>>> A = scipy.arange(1, 7).reshape((2,3))
>>> A.shape
(2, 3)
>>> b = scipy.ones(3)
>>> b.shape
(3,)
>>> b = scipy.array([b])
>>> b.shape
(1, 3)
```

Producto escalar de matrices

scipy.dot

```
>>> prod = scipy.dot(A, b.transpose())
>>> print prod
[[  6.]
 [ 15.]]
>>> print prod.shape
(2, 1)
```

Inversa de una matriz

scipy.linalg.inv

```
scipy.linalg.inv(ndarray)
```

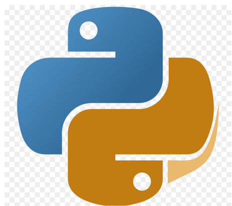
```
>>> from scipy import linalg
>>> A = scipy.arange(1, 5).reshape((2,2))
>>> A
array([[1, 2],
       [3, 4]])
>>> B = linalg.inv(A)
>>> B
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>> A.dot(B)  # verificacion
array([[ 1.,  0.],
       [ 0.,  1.]])
```

Ejercicios

Ejercicio 11.

`http://python.iaa.es/content/ejercicios-curso-científico`

Índice



1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. **Constantes**
10. Agrupamiento

Constantes *SCIPY*

`scipy.constants`

Este módulo contiene sobre constantes matemáticas y físicas.
Incluye un diccionario "`scipy.constants.physical_constants`" en el que

- cada clave es el nombre de una constante física, y
- el valor es una tupla que incluye (valor de la constante, unidades, precisión).

Dispone de 399 constantes físicas.

Constantes *SCIPY*

scipy.constants

El siguiente fragmento de código muestra diez de ellas:

```
>>> import scipy.constants
>>> keys = scipy.constants.physical_constants.keys()
>>> print len(keys)
399
>>> for j in range(10):
...     print keys[j]
...
joule-electron volt relationship
conductance quantum
standard acceleration of gravity
electron volt-hertz relationship
shielded helion to proton mag. mom. ratio
lattice parameter of silicon
neutron mag. mom. to nuclear magneton ratio
deuteron mass in u
kelvin-joule relationship
shielded helion magn. moment
```

Constantes *SCIPY*

scipy.constants

Cada constante lleva asociados tres valores, almacenados en forma de tupla en el diccionario scipy.constants.physical_constants

```
>>> from scipy.constants import physical_constants as pc
>>> pc['joule-electron volt relationship']
(6.24150965e+18, 'eV', 160000000000.0)
# (VALOR, UNIDADES, PRECISION)
```

Constantes *SCIPY*

scipy.constants

Existen 3 funciones que permiten obtener esos valores por separado para una constante dada:

- `scipy.constants.value(nombre_constante)`,
- `scipy.constants.unit(nombre_constante)`,
- `scipy.constants.precision(nombre_constante)`.

```
>>> scipy.constants.value('electron mass')
9.10938291e-31
>>> scipy.constants.unit('electron mass')
'kg'
>>> scipy.constants.precision('electron mass')
4.3910768045648e-08
```

Constantes *SCIPY*

scipy.constants

¿Tengo que leer todas las claves para saber si una constante existe y cuál es su nombre?

No es necesario. Podemos hacer una búsqueda aproximada mediante el uso de la función

```
scipy.constants.find(regexp)
```

```
>>> scipy.constants.find('light')
['speed of light in vacuum']
```

Prefijos de unidades

scipy.constants

Disponemos de órdenes para prefijos de unidades físicas.

```
>>> scipy.constants.yotta
1e+24
>>> scipy.constants.exa
1e+18
>>> scipy.constants.mega
1000000.0
>>> scipy.constants.kilo
1000.0
>>> scipy.constants.hecto
100.0
>>> scipy.constants.micro
1e-06
>>> scipy.constants.femto
1e-15
```

Unidades de información binaria

scipy.constants

También para unidades de información en sistema binario.

```
>>> scipy.constants.kibi
1024
>>> scipy.constants.gibi
1073741824
>>> scipy.constants.tebi
1099511627776
```

Funciones de conversión

scipy.constants

Existen funciones de conversión específicas para determinadas magnitudes.

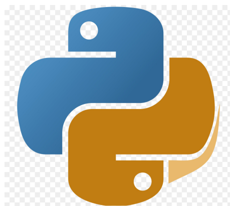
```
>>> scipy.constants.lambda2nu(3000)
99930.8193333333
>>> # Equivalente a ...
>>> scipy.constants.find('light')
['speed␣of␣light␣in␣vacuum']
>>> scipy.constants.value('speed␣of␣light␣in␣vacuum')/3000
99930.81933333333
```

Ejercicios

Ejercicio 12.

http://python.iaa.es/content/ejercicios-curso-científico

Índice

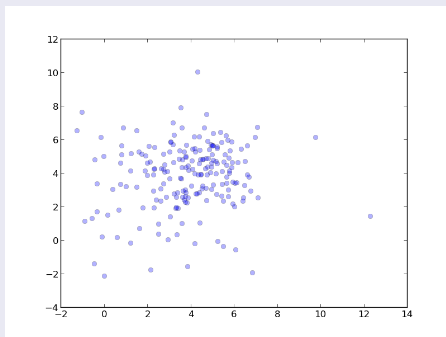


1. Introducción
2. Búsqueda de mínimos
3. Búsqueda de raíces
4. Integración
5. Interpolación
6. Estadística y Distribuciones
7. Ajuste de curvas
8. Álgebra lineal
9. Constantes
10. **Agrupamiento**

Agrupamiento

Caso práctico

Tengo el siguiente conjunto conjunto de datos



¿Cómo agrupo las observaciones en 2 o más grupos?

Agrupamiento

scipy.clustering

Dos paquetes realizan este tipo de tareas:

1. scipy.clustering.vq
   Soporta técnicas de *vector quantization* y *algoritmo k-means*.
2. scipy.clustering.hierarchy
   Agrupamiento por aglomeración y jerárquico.

Agrupamiento `k-means`

`scipy.clustering.vq`

En el algoritmo *k-means* se necesitan dos parámetros de entrada:

1. el conjunto de **datos** a agrupar.
2. el **número de grupos** a generar.

Los datos son normalmente arrays-2D de dimensión `M x N`, donde `M` es el número de objetos y `N` es el número de datos del que se dispone para cada objetos

Dado el número de grupos, el algoritmo trabaja asignando unas coordenadas al centro de cada uno de los grupos y asignando el identificador a cada objeto en función del grupo cuyo centro esté más próximo. Esto es lo que se denomina *cuantificar vectores* (*vector quantization*)

k-means

scipy.clustering.vq.kmeans

```
scipy.cluster.vq.kmeans(obs, k_or_guess, iter=20,
thresh=1e-05)
```

Parámetro	Tipo	Descripción
obs	array	de dimensión M x N (M observaciones con N datos cada una).
k_or_guess	entero o array	Si es entero, es el número de grupos. Los centros se inicializan aleatoriamente entre las observaciones. Si es un array, se inicializan los centros a esas k posiciones (k x N).
iter	entero	Número de iteracciones
thresh	entero	Termina las iteraciones si la distorsión (suma de las distancias al cuadrado entre cada observación y su *centroide dominante*) cambia menos que este valor.

Retorna

Parámetro	Tipo	Descripción
codebook	array	Array k x N, con las coordenadas de los centroides que minimizan la distorsión
distorsion	float	Suma de las distancias al cuadrado entre cada observación y su *centroide dominante*

Vector quantization

scipy.clustering.vq.vq

```
scipy.cluster.vq.vq(obs, code_book))
```

Parámetro	Tipo	Descripción
obs	array	de dimensión M x N (M observaciones con N datos cada una).
code_book	array	Array k x N, con las coordenadas de los centroides que minimizan la distorsión

Retorna

Parámetro	Tipo	Descripción
code	array	Array de longitud N, con el código del centroide dominante para cada observación
dist	array	Array de longitud N, con la distanca de cada observación al centroide dominante.

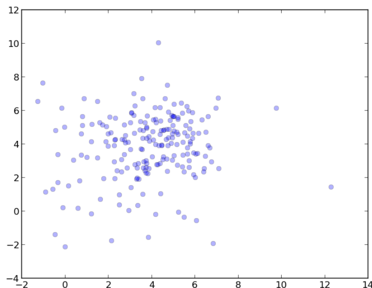
Ejemplo 1

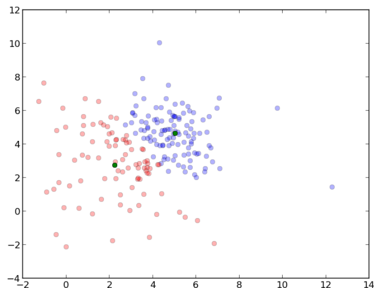
Agrupamiento por k-means

```
>>> # genero la muestra
>>> x = scipy.concatenate((random.normal(2, 3, 100), \
random.normal(5, 1, 100)))
>>> y = scipy.concatenate((random.normal(4, 1, 100), \
random.normal(3, 2, 100)))
>>> data = scipy.concatenate((x, y))
>>> data = data.reshape((200,2))
>>> Agrupo
>>> codebook, distorsion = cluster.vq.kmeans(data, 2)
>>> code, dist = cluster.vq.vq(data, codebook)
>>> # puntos de un grupo
>>> pyplot.plot(data[:,0].compress(code==0), \
data[:,1].compress(code==0), 'bo', alpha=0.3)
>>> # puntos del otro grupo
>>> pyplot.plot(data[:,0].compress(code==1), \
data[:,1].compress(code==1), 'ro', alpha=0.3)
>>> # centroides
>>> pyplot.plot(codebook[:,0], codebook[:,1], 'go')
```

Ejemplo 1

Agrupamiento por k-means





Ejercicios

Ejercicio 13.

http://python.iaa.es/content/ejercicios-curso-científico

FIN