Temario

- Graficación usando matplotlib.
- Método de cuadrados mínimos
- Determinación de raíces
- Ajuste de curvas
  - Método de bisección
  - Método de Newton

Miercoles 8 Noviembre. 2do Parcial.

Graficación en python

La librería de graficación se llama matplotlib. Dentro de esta tenemos dos opciones: pylab y pyplot.

- pylab. Interface de graficación inspirada en Matlab.
- pyplot. Librería de graficación basada en objetos.

Graficación en python

plot es para graficar curvas 2D.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 20, 1000)
y = np.sin(x)

plt.plot(x, y)
plt.savefig('fig.png')

plt.show() # Solo en la laptop no en el servidor
```

savefig para guardar la figura en un archivo (png,jpq,eps,etc.)

show para mostrar la figura. OJO NO usar en el servidor

Límites de los ejes

xlim(xmin,xmax), ylim(ymin,ymax)

En el caso anterior el plot acomoda los ejes a los máximos. Definamos nosotros los límites de los ejes. [xmin, xmax, ymin, ymax]

```
plt.plot(x, y)
plt.axis((5, 15,-1.2, 1.2))
```

Títulos de gráficos

xlabel('x'), ylabel('y'), title('Titulo')

```
plt.plot(x, y)

plt.xlabel('this is x!')
plt.ylabel('this is y!')
plt.title('My First Plot')
```

Entiende latex para escribir fórmulas:

```
y = np.sin(2 * np.pi * x)
plt.plot(x, y)
plt.title(r'$\sin(2 \pi x)$')
```

Tipos de curvas/líneas

```
plt.plot(x, y, '-r')
```

Colores disponibles:

'r' = red - rojo

'g' = green - verde

'b' = blue - azul

'c' = cyan - celeste

'm' = magenta - violeta

'y' = yellow - amarillo

'k' = black - negro

'w' = white - blanco

Las líneas pueden tener distintos estilos:

'-' = línea continua

'–' = línea a trazos

':' = línea punteada

'-.' = punteada a trazos

'.' = puntos

'o' = círculos

' ^ ' = triángulos

Múltiple curvas y leyendas

```
x = np.linspace(0, 20, 1000)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1, '-b', label='sine')
plt.plot(x, y2, '-r', label='cosine')
plt.legend(loc='upper right')
plt.ylim(-1.5, 2.0)
```

Hasta aca el uso de las librerias pyplot o pylab sería indistinto.

Múltiple plots

subplot(filas, columnas, nro de plot)

El nro empieza en 1, y sigue la numeración en orden de lectura (izq a der arriba a abajo).

Con pylab

```
pylab.subplot(2, 2, 1)
pylab.plot(x, np.sin(x))
pylab.subplot(2, 2, 2)
pylab.plot(x, np.cos(x))
pylab.subplot(2, 1, 2)
pylab.plot(x, x**2-x)
```

Con pyplot

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(9,3))
ax1 = fig.add_subplot(1,2,1)
ax1.plot(x, np.sin(x))
ax1 = fig.add_subplot(1,2,2)
ax1.plot(x, np.cos(x))
```

Guardar el gráfico en un archivo

Para guardar la figura en un archivo de imagen tenemos el comando:
savefig(fname, dpi=None, facecolor='w', edgecolor='w',
orientation='portrait', papertype=None, format=None):
Ejemplo:

```
savefig('fig05.png', dpi=80)
```

Formatos recomendados: png o eps (conservan la resolución de fuentes y líneas cuando se cambia el tamaño).

Esta es la opción recomendada en el servidor (en lugar del show).

Gráficos de densidades

Si lo que queremos graficar es una “imagen” o gráfico de densidad 2d se utiliza el comando:

```
imshow(data)
show()
```

Para cambiar el origin: `origin=”lower”`
Si se quiere el gráfico en blanco y negro en lugar del “heatmap” se utiliza:

```
imshow(data)
gray()
show()
```

Para cambiar la escala del gráfico:

```
imshow(data,extend=[0,10,0,5])
```

Si queremos cambiar el aspecto (la razón de distancia entre x e y)

`aspect=2`

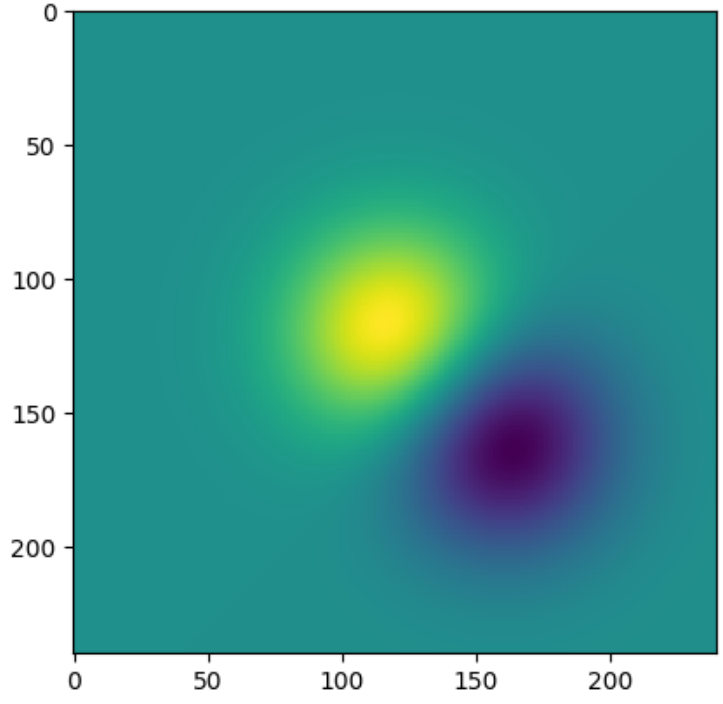
Ejemplo densidades

```
delta = 0.025
x = y = np.arange(-3.0, 3.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

fig, ax = plt.subplots()
im = ax.imshow(Z)
plt.show()

im = ax.imshow(Z,
       interpolation='bilinear',
       cmap=cm.RdYlGn,origin='lower',
       extent=[-3,3,-3,3],
       vmax=abs(Z).max(),
       vmin=-abs(Z).max())

plt.show()
```

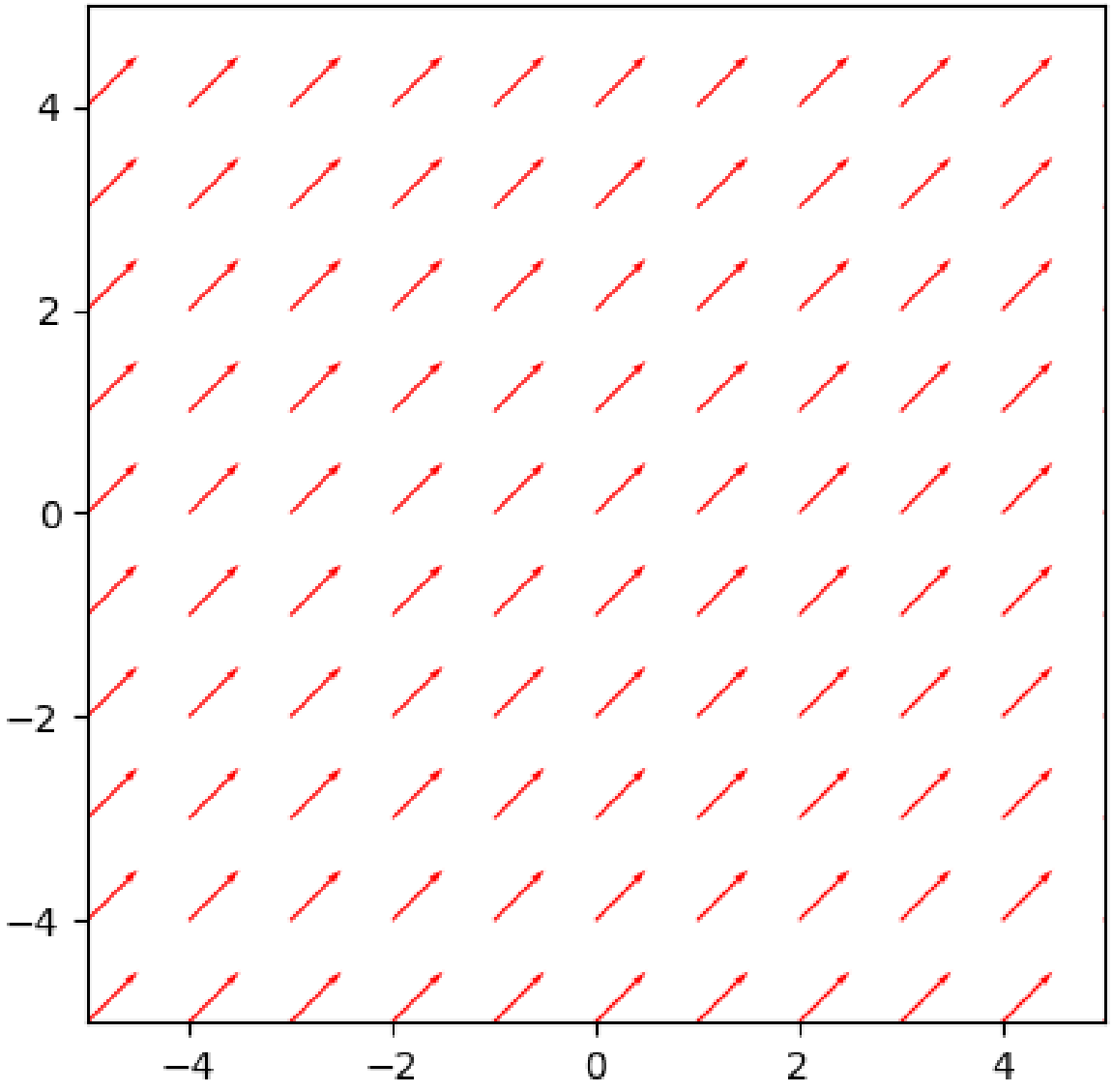


Campos de vectores

Función `quiver`. pyplot! Grafica vectores en un plano.

```
import matplotlib.pyplot as plt
import numpy as np
X, Y = np.meshgrid(np.arange(-10, 10, 1), np.arange(-10, 10, 1))
x_shape = X.shape
U=np.ones(x_shape)
V=np.ones(x_shape)
fig, ax = plt.subplots()
q = ax.quiver(X, Y, U, V, units='xy' ,scale=2, color='red')
ax.set_aspect('equal')
ax.set_xlim(-5,5)
ax.set_ylim(-5,5)
plt.show()
```

Campos de vectores



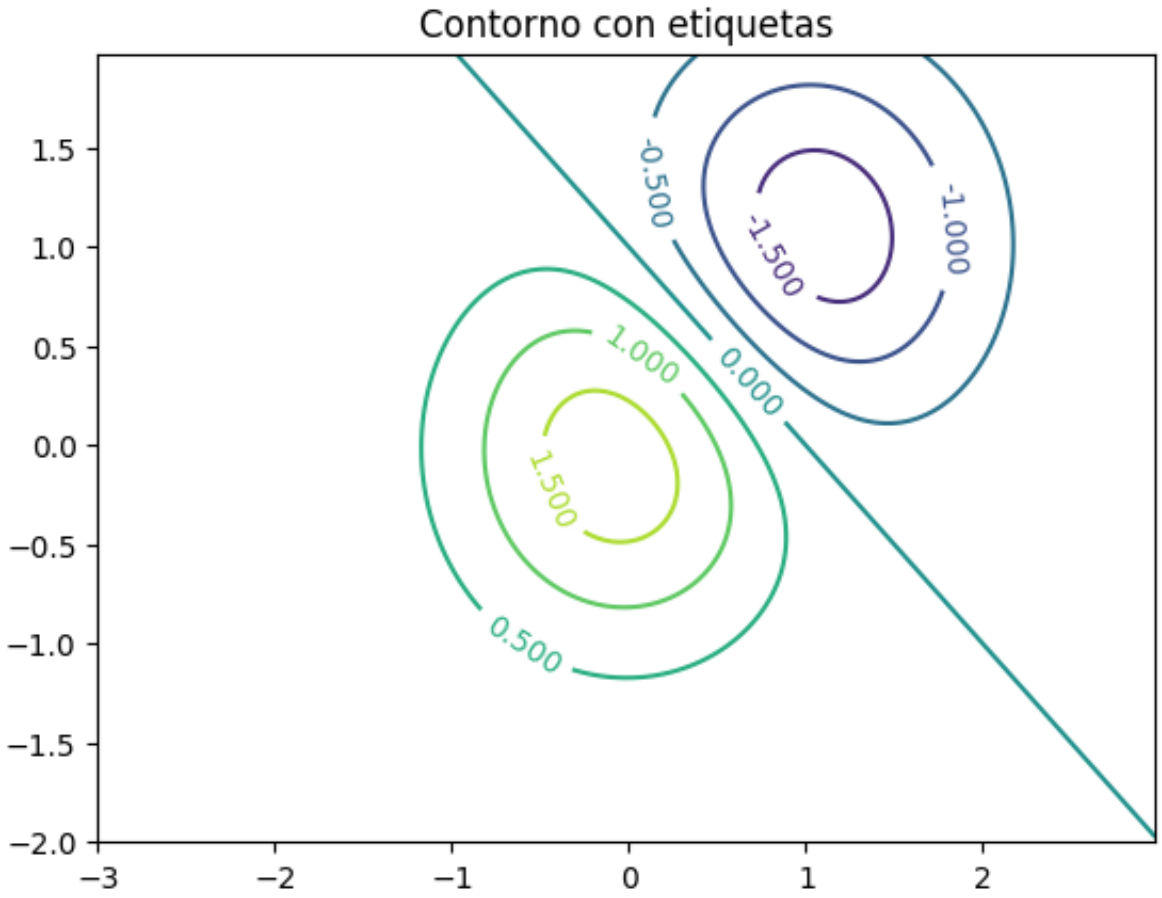
Contornos

Función `contour`. pyplot!

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
CS = ax.contour(X, Y, Z)
ax.clabel(CS, inline=True, fontsize=10)
ax.set_title('Contorno con etiquetas')
plt.savefig('contorno.png', bbox_inches='tight')
plt.show()
plt.close()
```

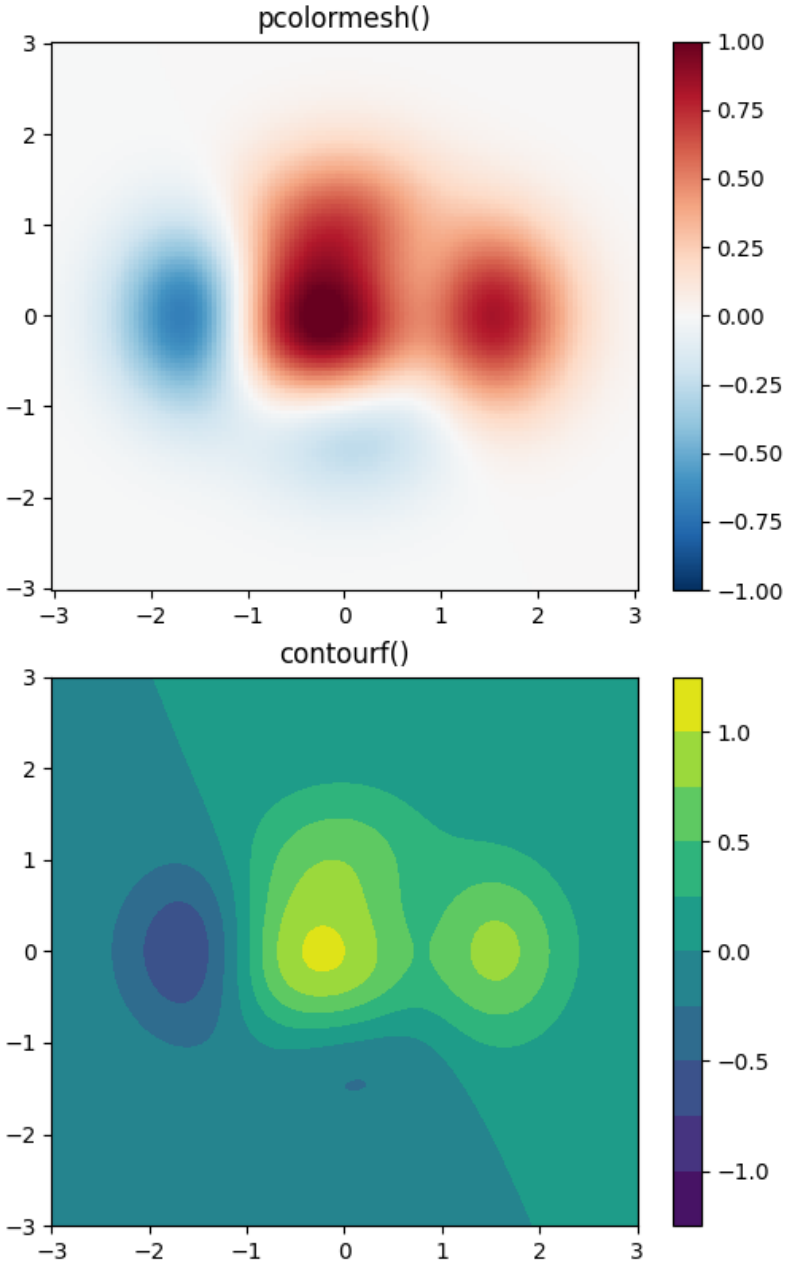
Contorno


Contorno con etiquetas
-0.500
-1.000
-1.500
0.000
1.000
1.500
0.500
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2

Contornos: barra de colores y pcolormesh

```
fig, axs = plt.subplots(1, 2,
    layout='constrained')
pc = axs[0].pcolormesh(X, Y, Z,
    vmin=-1, vmax=1, cmap='RdBu_r')
fig.colorbar(pc, ax=axs[0])
axs[0].set_title('pcolormesh()')
co = axs[1].contourf(X, Y, Z,
     levels=np.linspace(-1.25, 1.25
                        , 11))
fig.colorbar(co, ax=axs[1])
axs[1].set_title('contourf()')

fig.savefig('contorno2.png')
```



Mas alla!

Hay un montón de formas de graficar.

- Se pueden hacer animaciones. Ver `animation`
- Se pueden hacer gráficos interactivos.

  Widgets adaptativos.

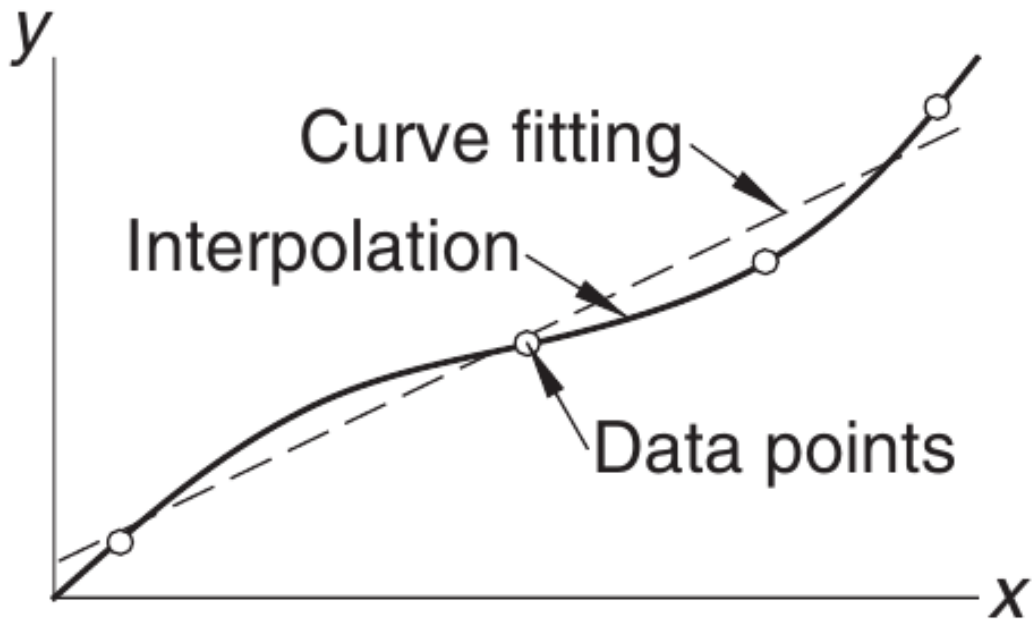
  Muy bueno para cambiar los parametros del gráfico y lo veo

```
from matplotlib.widgets import Slider, Button,
RadioButtons
```

`https://matplotlib.org/3.4.2/`

Aproximación de funciones

Dados un conjunto de n puntos x_i, y_i, cual es la función que mejor representa a estos puntos?



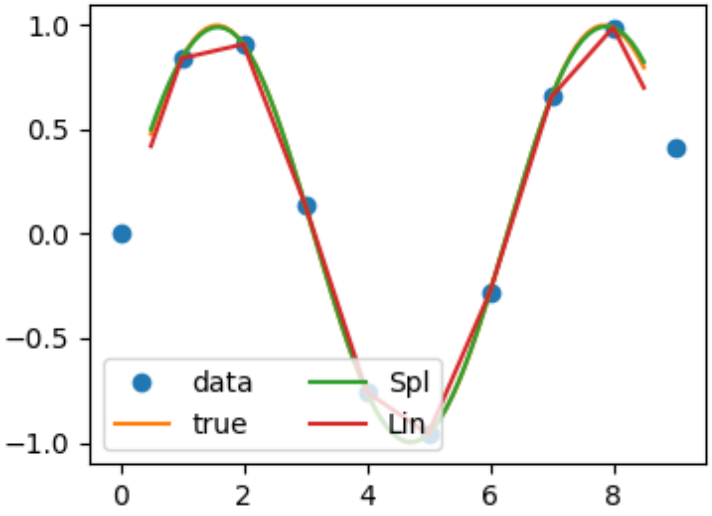
Lineal por pedazos

Usando el método de Lagrange:

$$P_1(x) = y_0 l_0(x) + y_1 l_1(x)$$

donde $l_0(x) = \frac{x-x_1}{x_0-x_1}$, $l_1(x) = \frac{x-x_0}{x_1-x_0}$.
Esto sería en forma recursiva para cada par de puntos.

Interpolación con splines cúbicos



Modelado de datos experimentales

Supongamos que tenemos una lista de datos experimentales: $(x_j, y_j)\ j = 1, \ldots, N$. N pares de datos.

Además conocemos una ley física

$$y = f(x, a_1, \cdots, a_M)$$

De esta queremos que la ley física ajuste a los datos y determinar los parámetros $a_1, \cdots, a_M$.

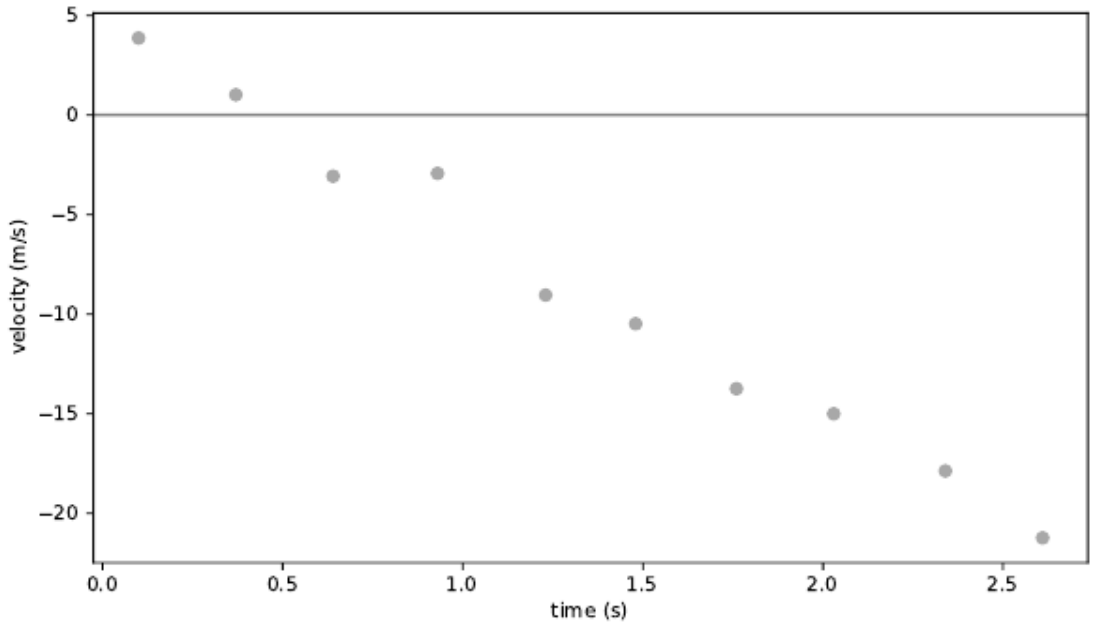
Ejemplo. Caida libre

Supongamos medimos la velocidad de un objeto que cae y la hemos tomado cada 0.25 segundos.

Si se desprecia la resistencia del aire, el objeto cae en caida libre con la aceleración de la gravedad:

$$v(t) = v_0 - gt$$

Entonces los parámetros a determinar son v_0, g.



Función de costo

Quiero encontrar la curva $f(x, a_1, \cdots, a_M)$ que mejor ajuste a todos los puntos que tengo $(x_j, y_j)\ j = 1, \ldots, N$.
Si hago la suma de las diferencias cuadráticas entre los puntos y la curva

$$J(a_1, \ldots, a_M) = \sum_i^N [y_i - f(x_i, a_1, \ldots, a_M)]^2$$

Esta J sería el error que tiene la curva en ajustar los puntos, se denomina función de costo.
El método de cuadrados mínimos o regresión consiste en determinar los parametros $a_1, \ldots, a_M$ que minimizan la función de costo $J(a_1, \ldots, a_M)$.
Sabemos que la función tendrá un mínimo cuando la derivada se anula:

$$\frac{\partial J}{\partial a_i} = 0 \quad \forall i$$

Regresión lineal

En el caso que la funcion de ajuste sea una recta $y = a_0 + a_1 x$:

$$J(a_0, a_1) = \sum_i^N (y_i - a_0 - a_1 x_i)^2$$

$$\frac{\partial J}{\partial a_0} = \sum -2(y_i - a_0 - a_1 x_i) = 2(Na_0 + a_1 \sum_i x_i - \sum_i y_i) = 0 \qquad (1)$$

$$\frac{\partial J}{\partial a_1} = \sum -2(y_i - a_0 - a_1 x_i)x_i = 2(a_0 \sum_i x_i + b \sum_i x_i^2 - \sum_i x_i y_i) = 0 \qquad (2)$$

Calculando las medias: $\bar{x} = \frac{1}{N}\sum_i x_i$, $\bar{y} = \frac{1}{N}\sum_i y_i$
Luego despejando determinamos los parámetros/coeficientes:

$$a_1 = \frac{\sum_i (x_i - \bar{x})y_i}{\sum_i (x_i - \bar{x})x_i}, \qquad a_0 = \bar{y} - a_1 \bar{x}$$

Regresión lineal con errores no-uniformes

Si cada medición tiene un error distinto, en la funcion de costo tenemos que considerar su peso. Las mediciones mas precisas tienen mas pesos que las mediciones con mas errores es decir:

$$J = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2$$

Las medias pesadas son:

$$\hat{x} = \frac{\sum_i x_i/\sigma_i^2}{\sum_i 1/\sigma_i^2}, \qquad \hat{y} = \frac{\sum_i y_i/\sigma_i^2}{\sum_i 1/\sigma_i^2},$$

En este caso las ecuaciones para los coeficientes son:

$$a_1 = \frac{\sum_i (x_i - \hat{x}) y_i/\sigma_i^2}{\sum_i (x_i - \hat{x}) x_i/\sigma_i^2}, \qquad a_0 = \hat{y} - a_1 \hat{x}$$

Ajuste de curvas no-lineales

En el caso general lo que tenemos es que

$$J = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2$$

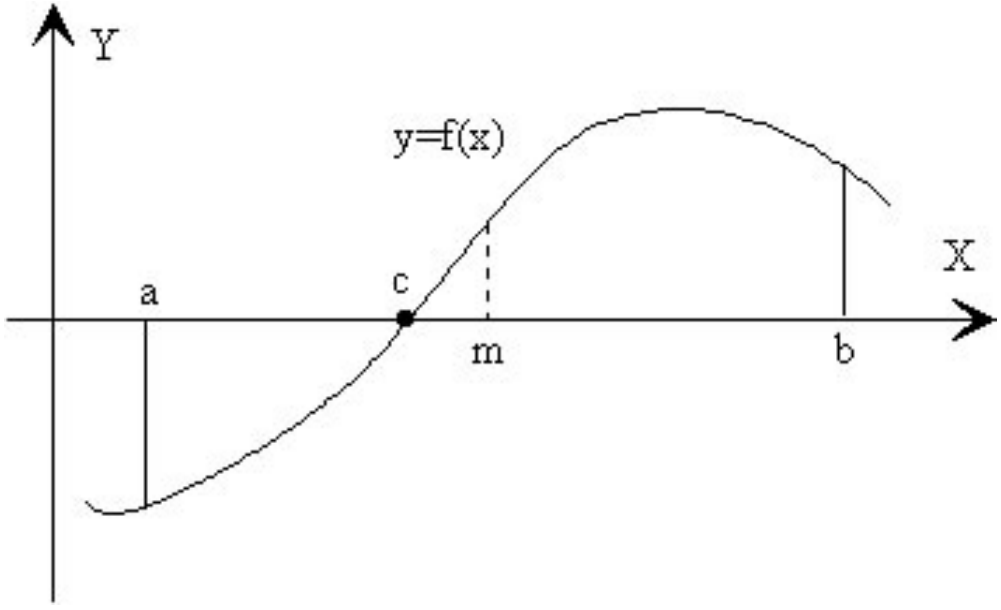
con $f(x)$ una función cualquiera que tiene un conjunto de parámetros a determinar:

$$f(x) = a\cos(bx) + c\log(dx)$$

Todo lo que pensamos es que $J(a, b, c, d)$ mas allá de como se produce.
Tenemos que encontrar (a, b, c, d) tal que se encuentre el mínimo de J. Esto es un optimizador o minimizador.
Si tenemos información de la derivada de la J (y por lo tanto necesitamos la derivada de f), entonces el método puede usar esa información para encontrar la raiz de ∇J.
Esta es la base de todo el aprendizaje automático para funciones f que son en general redes neuronales.

Raíces de ecuaciones

Dada una función f queremos encontrar un x_r tal que $f(x_r) = 0$.



Teorema de Bolzano: si tenemos una función continua $f(x)$ en el intervalo $[a, b]$, y el signo de la función en el extremo inferior a es distinto al signo de la función en el extremo superior b, $f(a)f(b) < 0$, entonces existe al menos un valor c dentro de dicho intervalo tal que $f(c) = 0$.

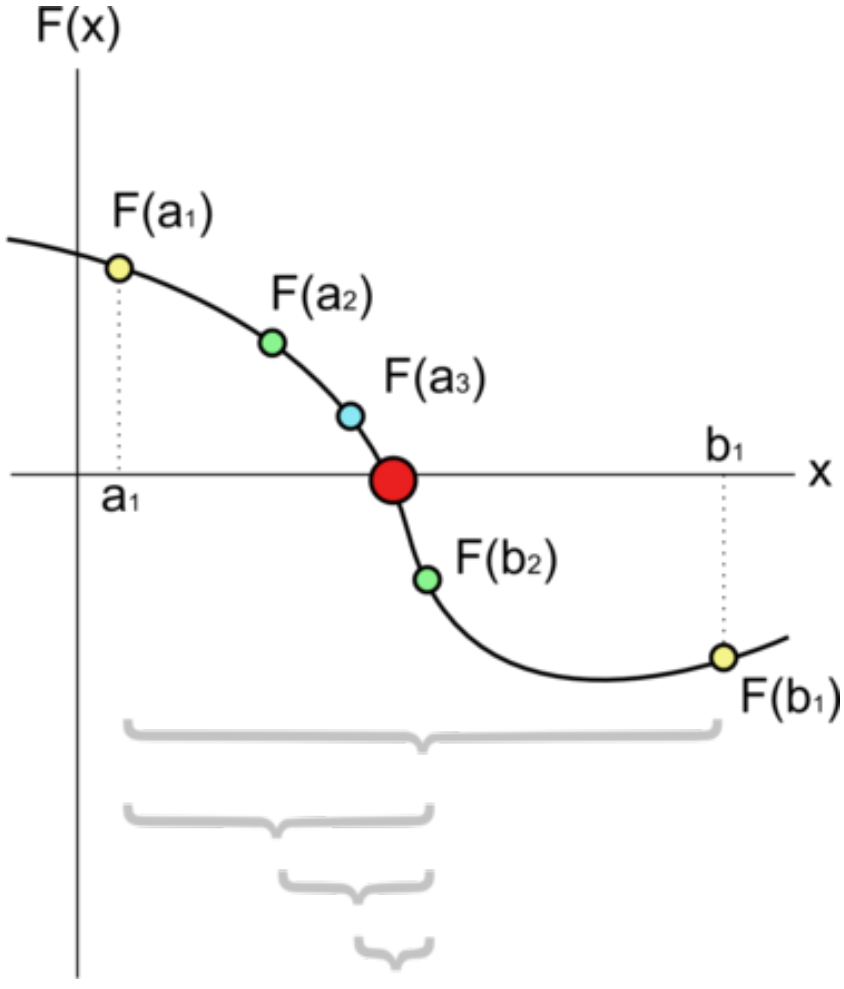
Método de bisección

Requisito: Conocemos un $f(a) < 0$ y $f(b) > 0$ (o viceversa), es decir $f(a)f(b) < 0$.

En que lugar del interior se puede encontrar la raiz?

Vamos a ver en la mitad del intervalo:
$x_m = [f(a) + f(b)]/2$
Si $f(x_m) > 0$ entonces la raiz se encuentra entre a y x_m.
Entonces elegimos como nuevo extremo superior $b = x_m$.

Ahora volvemos a repetir el procedimiento en el intervalo mas chiquito.



Método de bisección

```
      from math import ceil, log
def bisect(fn,a,b,epsilon=1.0e-5):
    fa = fn(a); fb = fn(b)
    # Chequeos inputs
    if fa == 0.0: return xa
    if fb == 0.0: return xb
    if fa*fb > 0.0: quit('No se cumple Bolzano')
    n = ceil(log(abs(b - a)/epsilon)/log(2.0)) # nro de intervalos
    for i in range(n):
        xm = 0.5*(xa + xb); fm = f(xm) # miro en el medio
        if fm == 0.0: return xm
        if fb*fm < 0.0:
            xa = xm; fa = fm # nuevo inferior
        else:
            xb =xm; fb = fm # nuevo superior
     return (xa + xb)/2.0
```

Alternativas similares a método de bisección

Python scipy.optimize.bisect

Positivo: Es un método seguro. Solo usa información de la función.

Negativo: Es muy lento. No utiliza la información del valor de la función.

Si $|f(a)| \gg |f(b)|$ esperamos que la raiz este mas cerca de $f(b)$.

Alternativas para mejorar la convergencia: Uso el método de la secante para determinar el x_m. Esto es una interpolación lineal. Próximo punto $y = 0$:

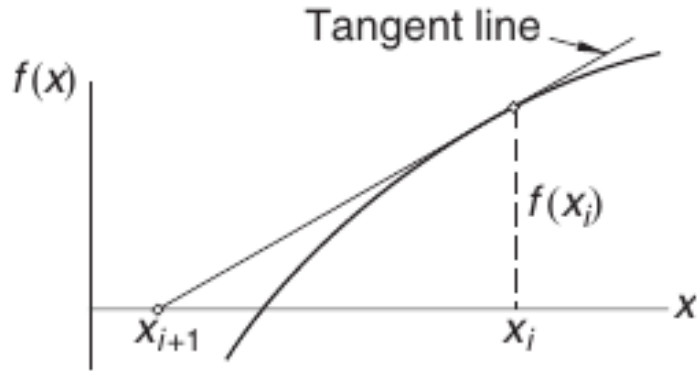
$$x_m = a - \frac{b-a}{f(b)-f(a)} f(a)$$

Método de Brent. Interporla un polinomio cuadrático entre los tres puntos, x_a, x_m y x_b y se fija cual es la raiz del polinomio cuadrático.

Método de Newton-Raphson

Si tenemos información de la derivada, sabemos cuanto esta cambiando la función en el punto.
Podemos poner como el próximo punto el lugar donde apuntaría la recta tangente al punto actual:

$$f(x_{i+1}) = 0 = f(x_i) + f'(x_i)(x_{i+1} - x_i)$$



Entonces el próximo punto esta en:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Esto es aproximado, mientras mas cerca esta la función a la tangente mas precisa será la estimación. Podemos seguir evaluando de esta manera.
Este algoritmo converge cuadraticamente.

Método de Newton-Raphson

1. Supongamos tenemos un valor inicial $x = x_0$
2. Loop for/while precisión requerida? $\epsilon < \epsilon_{target}$.
3. Evaluamos función y derivada de la función $f(x)$ y $f'(x)$.
4. xold=x
5. Determinamos próximo punto $x = xold - \frac{f(xold)}{f'(xold)}$
6. Precisión actual $\epsilon = \text{abs}(x - xold)$.

En python tenemos: scipy.optimize.newton(func, x0)