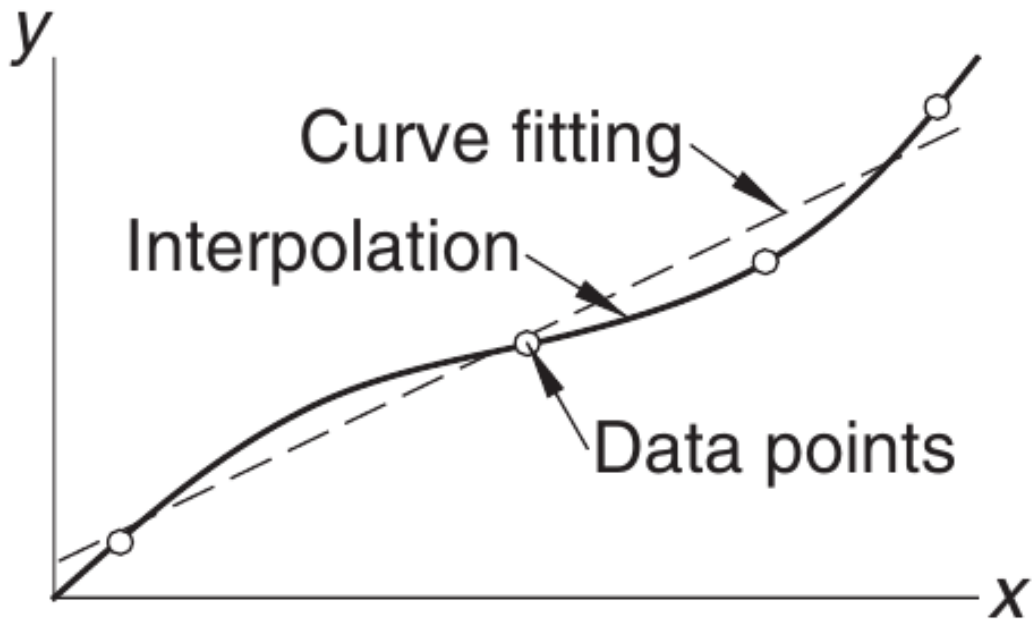
Métodos numéricos útiles para la experimentación

Temario

- Método de cuadrados mínimos
- Ajuste de curvas
- Determinación de raíces
    - Método de bisección
    - Método de Newton

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

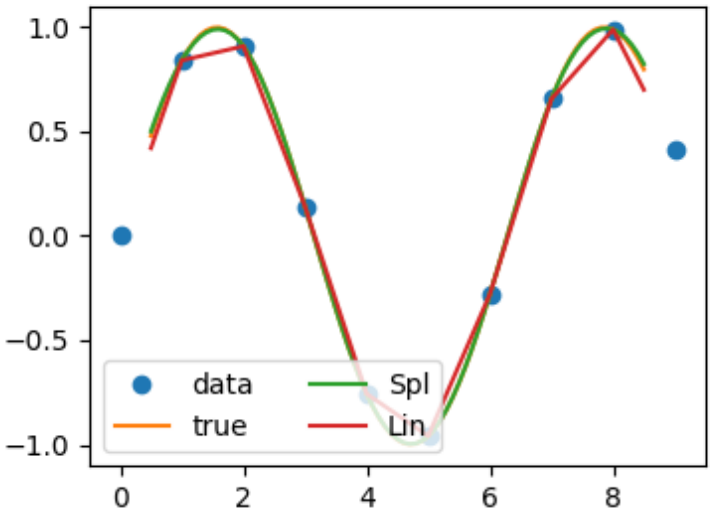
Dado los puntos los representamos por rectas entre cada par de puntos.
Usando el método de Lagrange:

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

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, x_i, x_{i+1}.

Interpolación con splines cúbicos

- Los polinomios cúbicos deben pasar por los puntos (x_i, y_i).
- Las derivadas primeras de dos polinomios debe ser la misma en los puntos. Continuidad de la funcion y de la derivada primera.




```
import numpy as np
from scipy import interpolate as interp
x =np.arange(10)
y =np.sin(x)
cs =interp.CubicSpline(x, y)
y_cs=cs(xs)
```

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 $a_1, \cdots, a_M$.

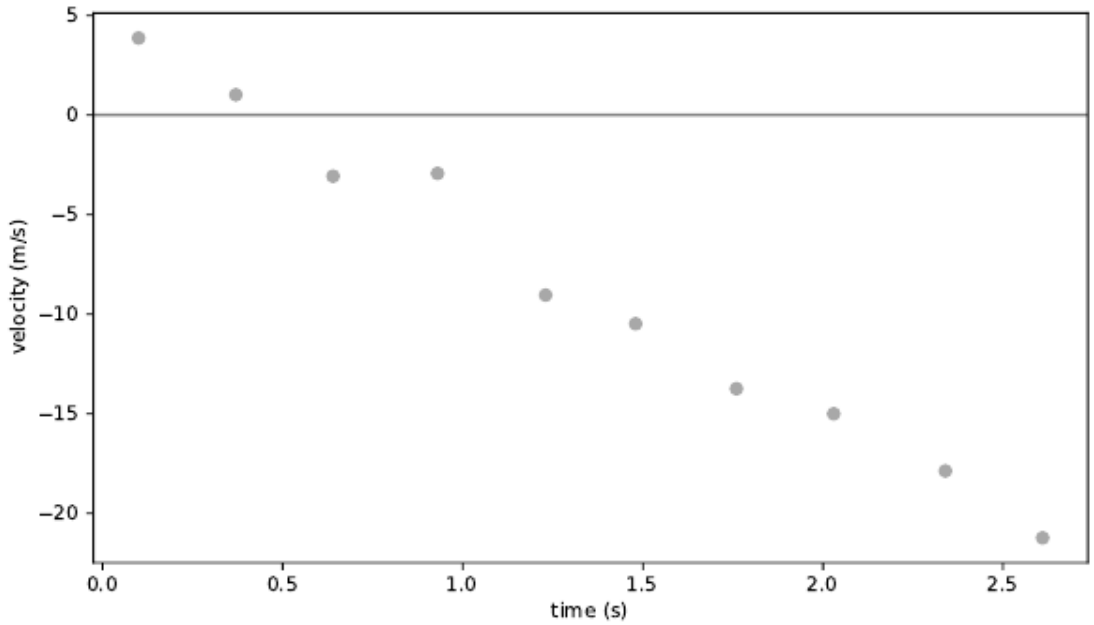
Ejemplo. Caida libre

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

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

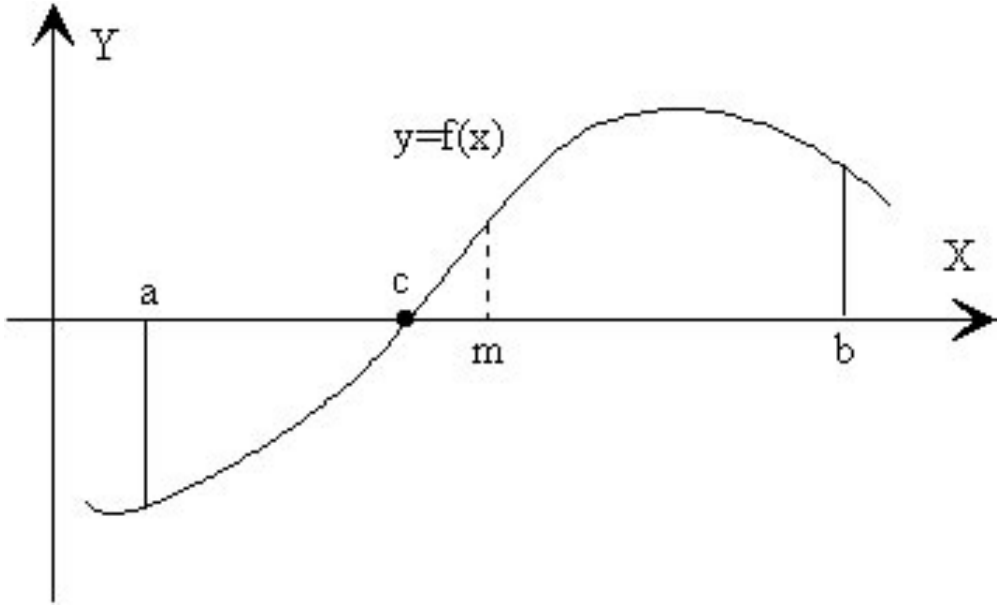
con $f(x)$ una función cualquiera que tiene un conjunto de parámetros a determinar:

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

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 $f(c) = 0$.

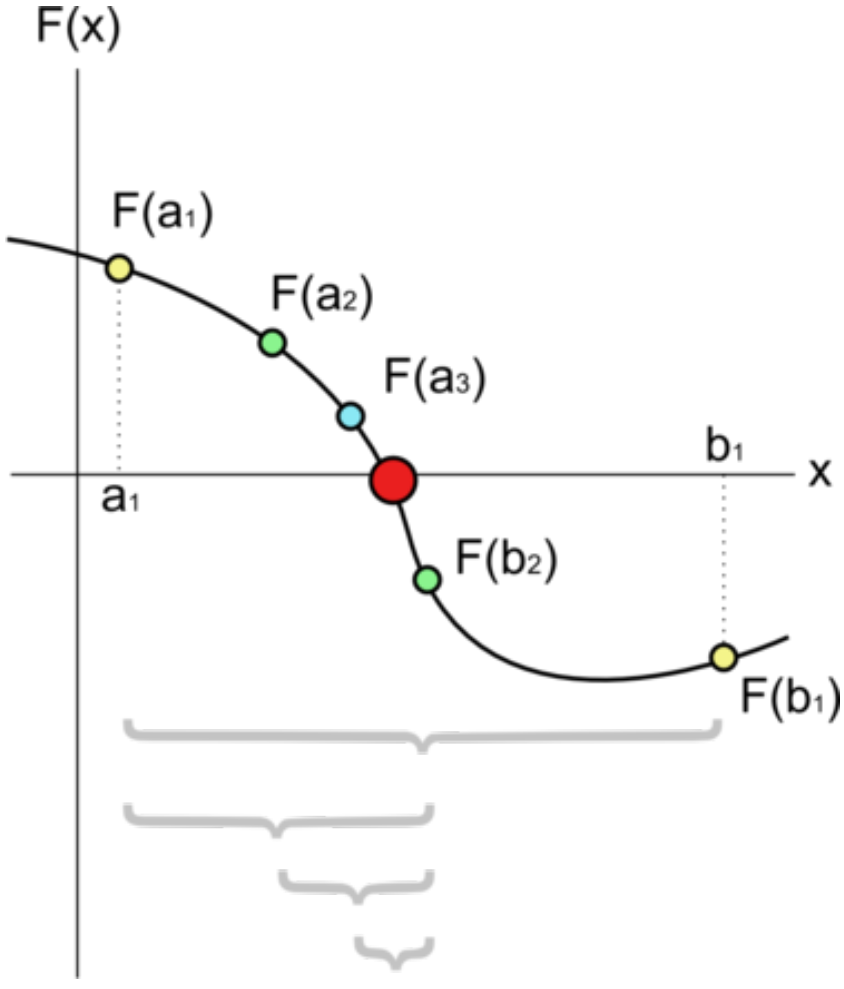
Método de bisección

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

```
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 $y = 0$:

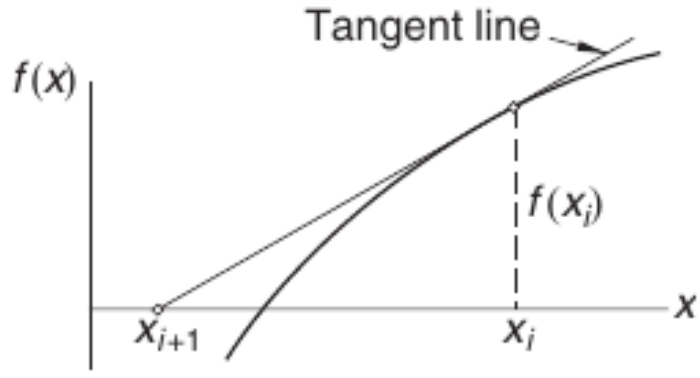
$$x_m = a - \frac{b-a}{f(b)-f(a)} f(a)$$

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)