

# Métodos de optimización

## BIG DATA CON PYTHON



**INTRODUCTION TO  
PYTHON FOR BIG DATA**



# Métodos de optimización

## BIG DATA CON PYTHON

### 1. Introducción a la optimización

- Tipos de problemas
- Big Data

### 2. Resolver problemas de optimización

- Solución analítica, grid, algoritmos
- Optimizar con Python

### 3. Optimización lineal

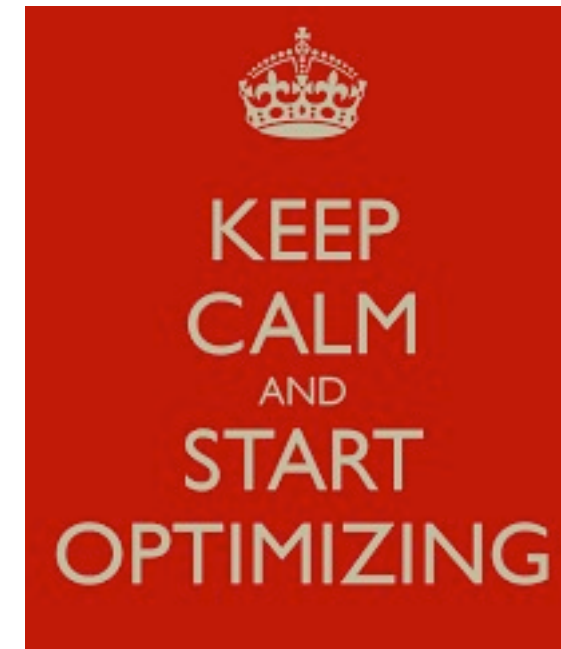
- Solución geométrica
- Métodos numéricos
- Ejemplos con python (`scipy.optimize`)

### 4. Optimización no lineal

- Programación cuadrática

### 5. Modelado de datos

- mínimos cuadrados
- maximum likelihood
- Ejemplos con python (`numpy.polyfit` and `scipy.curve_fit`)



# OPTIMIZACIÓN NO-LINEAL

(o Programación No-  
Lineal)



# Optimización No Lineal

Se presentan problemas parecidos en programación lineal, pero en este caso la función objetivo o las ligaduras no son lineales con respecto a los parámetros que buscamos. Ej.

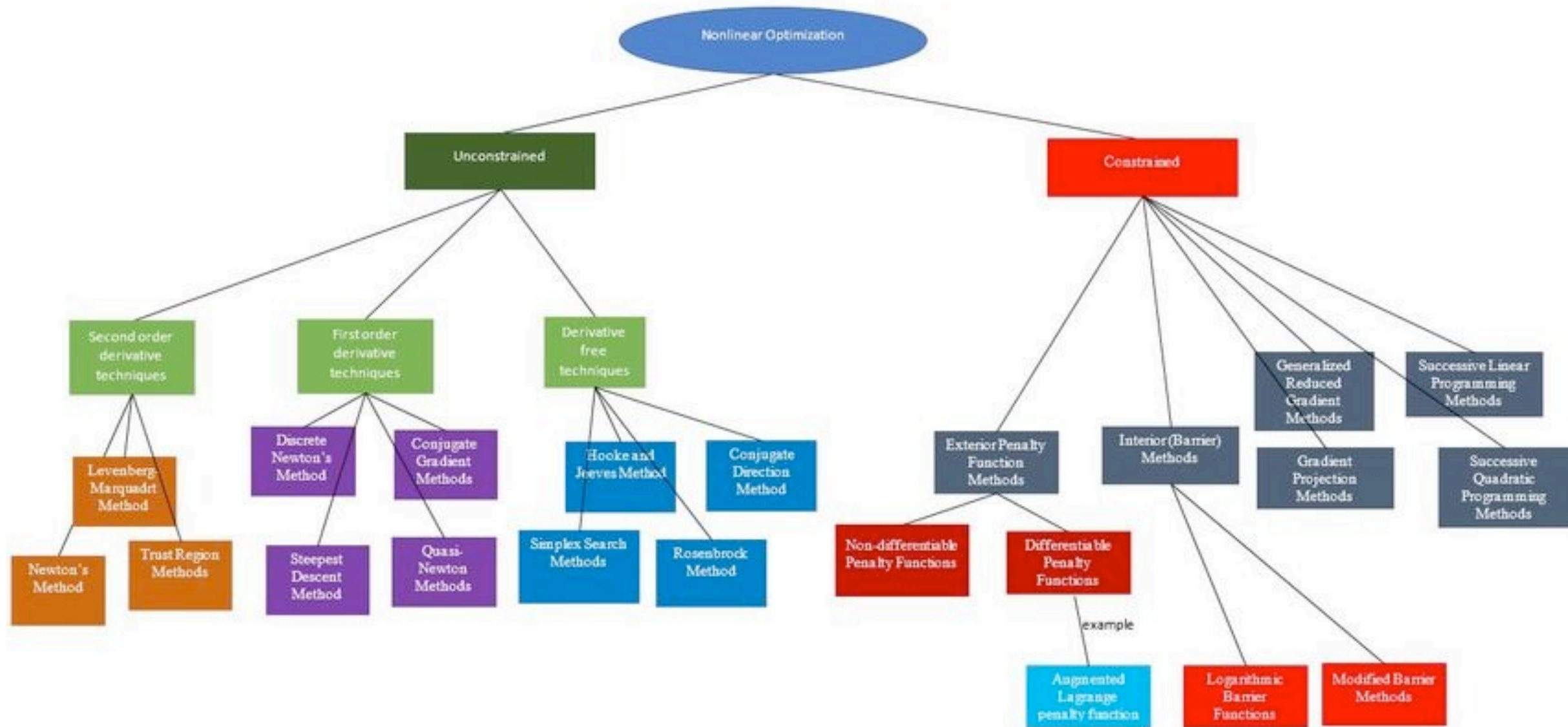
$$f(x) = Ax^\beta + Cx^2$$

En realidad, optimización no-lineal es el caso general, siendo la optimización lineal un caso específico de no-lineal.

Los algoritmos que son útiles para no-lineal pueden resolver un problema lineal, mientras que al revés no siempre es posible.

Los problemas típicos en Física, Ingeniería, ...

# Optimización No Lineal



# Optimización No Lineal

## Optimización cuadrática:

Un caso específico de optimización no lineal es lo que se llama **programación cuadrática** que en este caso la función a minimizar es cuadrática y las desigualdades son lineales. Se pueden usar los mismos algoritmos o variantes del mismo que en el caso de programación lineal.

$$\underset{x_1, x_2}{Min} \quad \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2$$

*subject to:*

$$x_1 + x_2 \leq 2$$

$$-x_1 + 2x_2 \leq 2$$

$$2x_1 + x_2 \leq 3$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

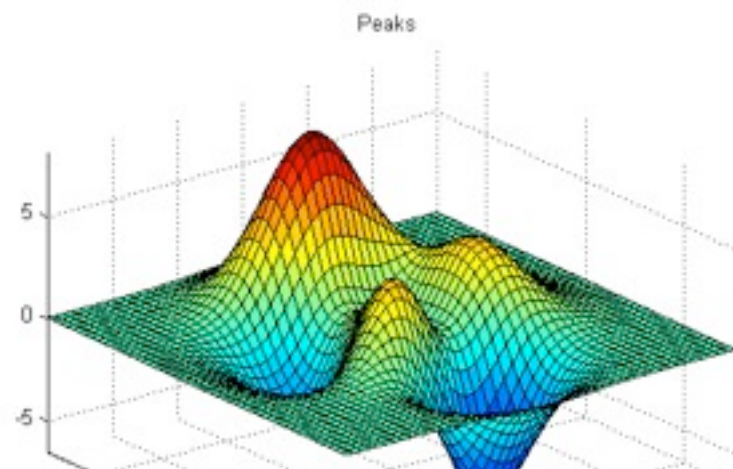
En el caso de problemas sin ligaduras o ligaduras lineales, se intenta conseguir una función objetivo cuadrática, mediante **expansión de Taylor a segundo orden**.

Cuando esto no es una buena aproximación, los métodos clásicos pueden fallar. Se puede usar un **sampleador** para explorar el espacio de parámetros.

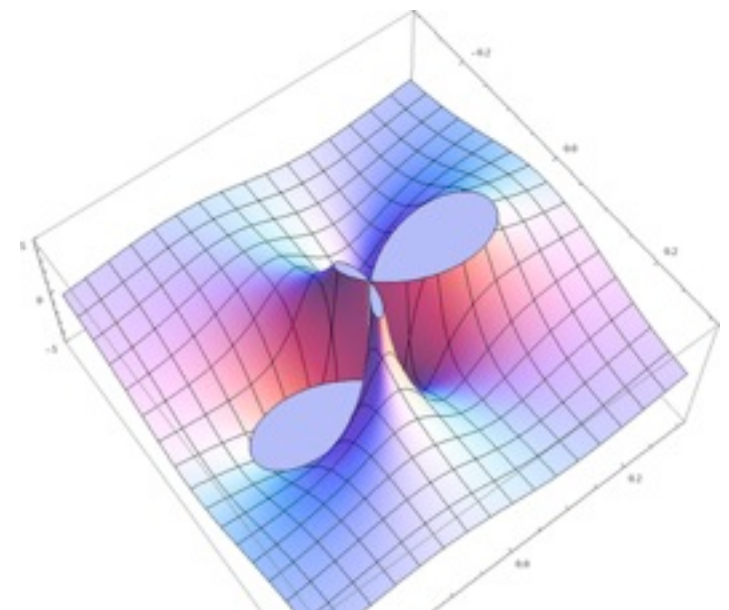
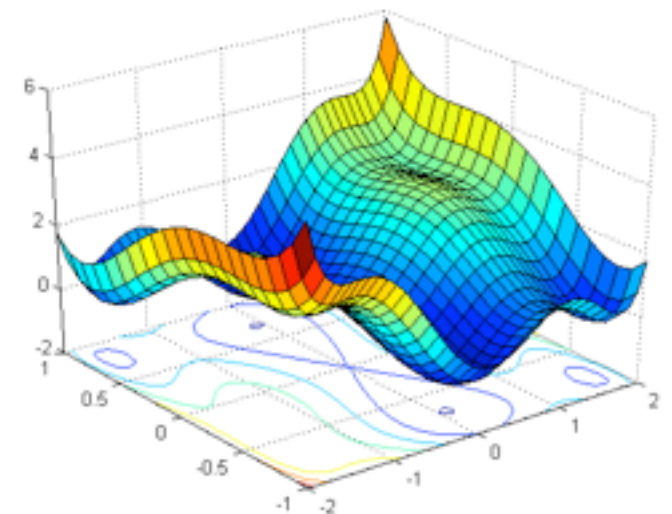
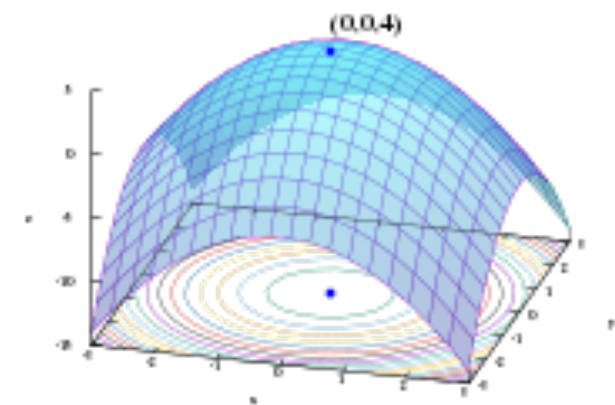
# FUNCIONES OBJETIVO TIPO

- Diferentes tipos de problemas necesitarán diferentes soluciones:
  - **convex problems** (mínimo local = mínimo global)
  - Más complicados, puedes caer en un mínimo local
  - Discontinuas, (gradient free methods, sampling methods)
  - Problemas con ligaduras pueden dar funciones discontinuas

**Problemas No-Lineales con  
derivada definidas -->  
gradiente based algoritms  
(BFGS, conjugate gradient,...)**



**Problemas No-Lineales sin  
derivada definidas -->  
Búsqueda directa**



# TIPOS de ALGORITMOS de OPTIMIZACIÓN

- **BÚSQUEDA DIRECTA:** Grid, Monte Carlo sampling, simplex method, genetic algorithms,...
  - normalmente más lentos
  - pueden lidiar con funciones discontinuas o no derivables
- **BASADOS EN EL GRADIENTE:** Hace falta que la función sea suave y continua: Gradiente conjugado, gradient descent, newton methods, BFGS, Levenberg-Marquardt,...
- **INTELIGENCIA ARTIFICIAL:** Cuando no conocemos el modelo o los otros algoritmos son muy lentos. Pero necesitamos un set de entrenamiento (mañana!)
- **+ CREATIVOS:** Cuando no hay modelo ni set de entrenamiento. (Ant colony,...)



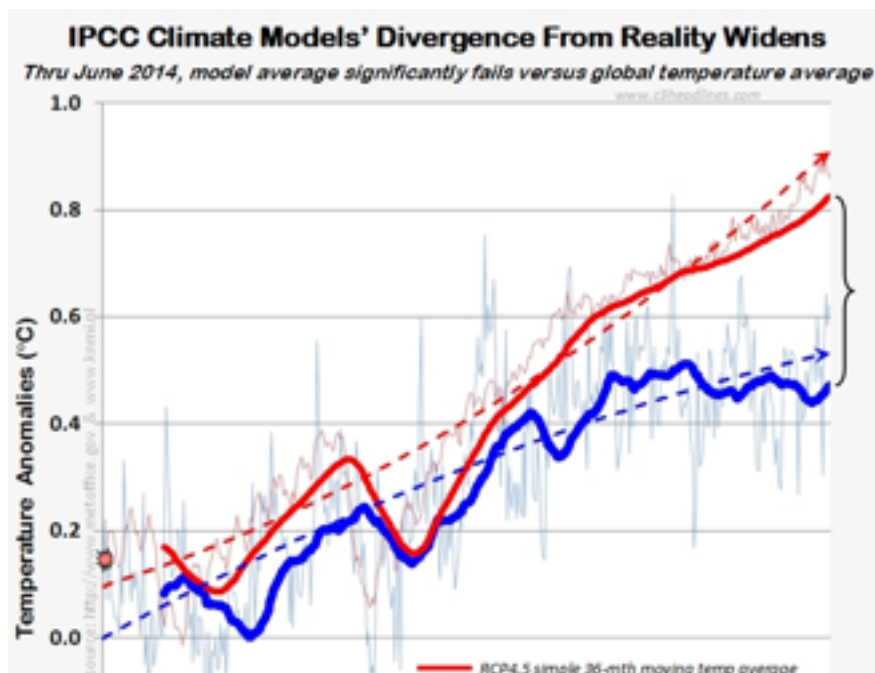
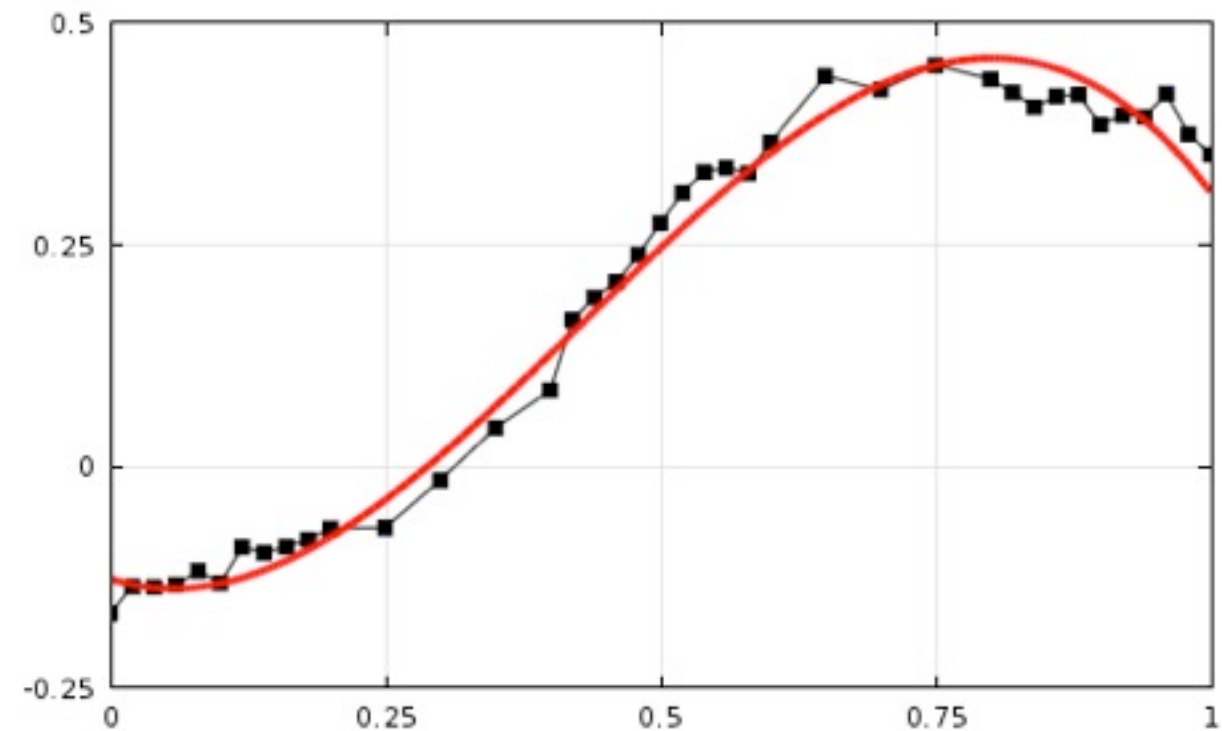
# MODELIZAR

(o encontrar parámetros de un modelo teórico)

# Modelizar

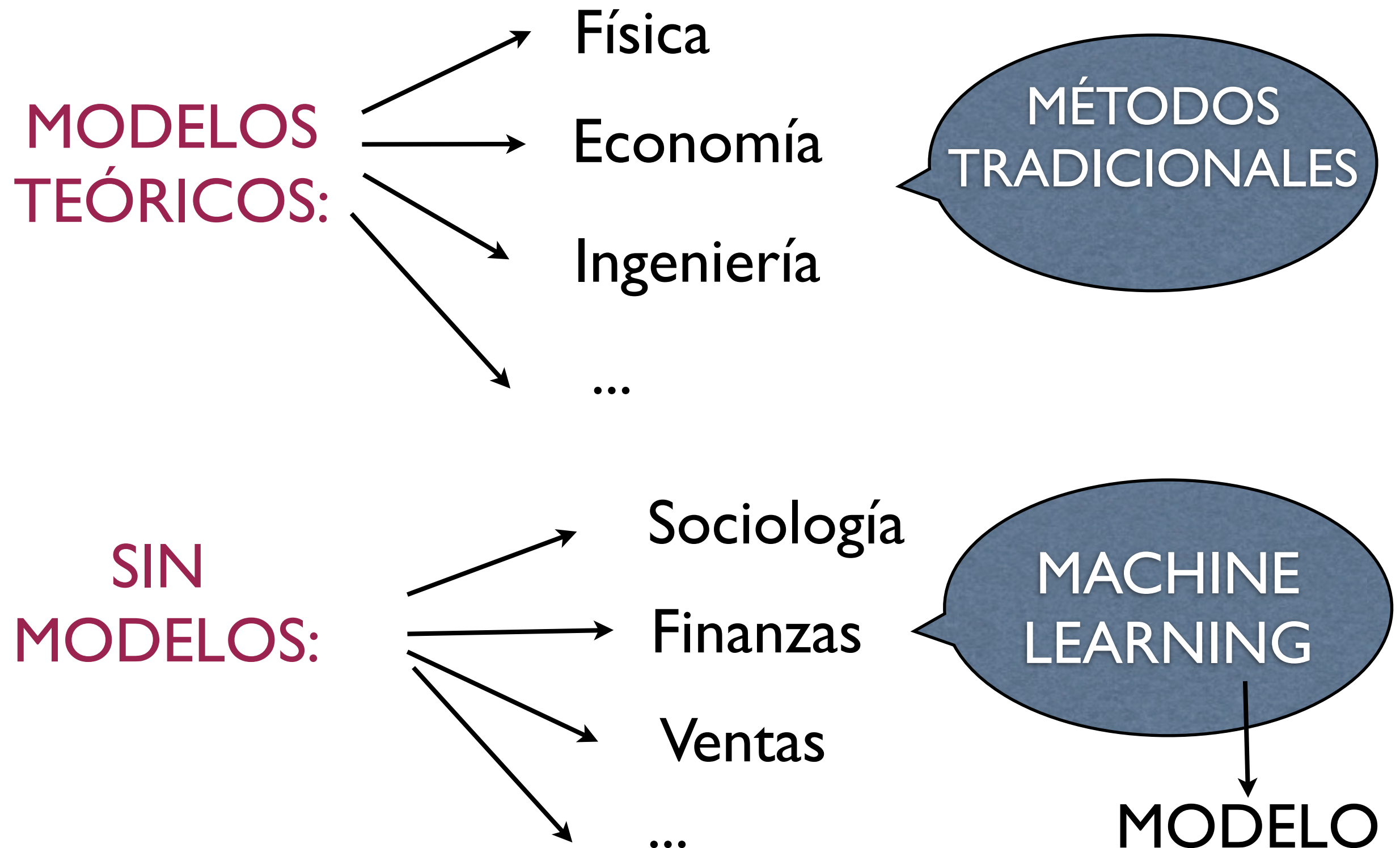
Aparte de los problemas de optimización que hemos visto hasta ahora, muchas veces tenemos una serie de observaciones y queremos explicarlas.

**Modelo teórico:** Puede ser que tengamos algún motivo para esperar que se comporten de cierta manera. Es decir, tenemos un **modelo**, que puede estar dado por leyes físicas, o por hipótesis sacadas de otras observaciones, en general tiene una teoría sólida detrás.



**Modelo predictivo:** O puede ser que queramos sacar un **modelo** nosotros. Es decir, que seamos capaces de predecir el resultado que nos darán unos nuevos datos. Esto se hace normalmente con un set de entrenamiento, Big Data permite tener grandes sets de entrenamiento. Se suelen usar técnicas de *machine learning* (**MAÑANA**)

# TIPOS de PROBLEMAS de OPTIMIZACIÓN



# Modelizar

- Al final es un problema específico de **optimización** donde la función objetivo compara los datos con unas predicciones.

Ej. mse

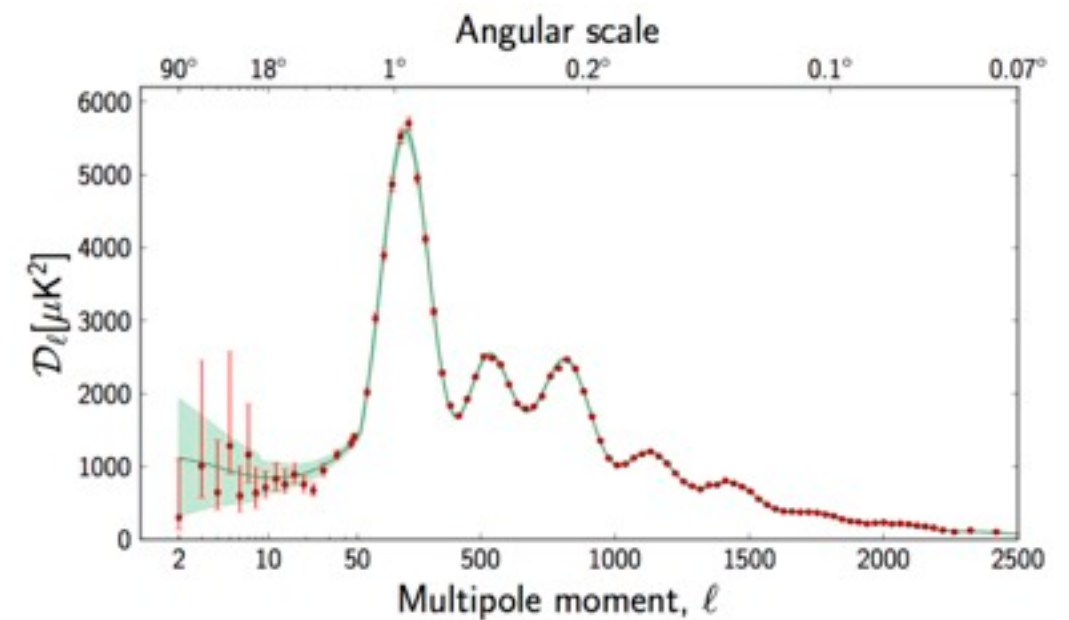
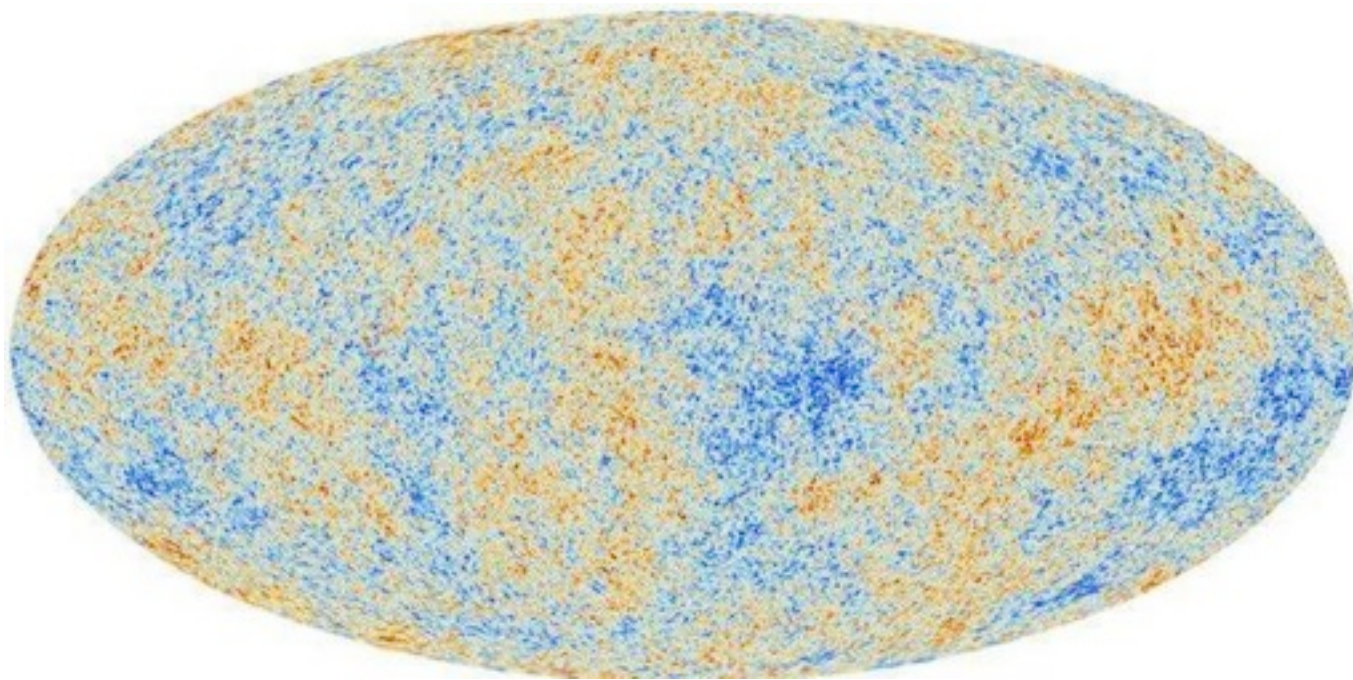
$$E = \sum_i \frac{(d - y)^2}{N}$$

- Y el modelo es la ligadura. Ej.  $y = ax + b$



# Modelizar

En física buscamos los parámetros de modelos constantemente. Ej. **Fondo cósmico de microondas**




Los puntos rojos se sacan de los datos (mapa izquierda), y la línea verde es el modelo teórico con los parámetros que mejor se ajustan a las observaciones. (Edad Universo, energía oscura, materia oscura, velocidad de expansión,...)

# Modelizar

- En general, necesitaremos como mínimo tantas observaciones como parámetros.
- Una función objetivo en el caso de modelización nos tiene que dar cuenta del parecido entre el modelo y los datos

Típicas:

Mínimos cuadrados   $(\text{datos} - \text{modelo})^2$

Maximum likelihood   $\text{Pr}(\text{datos} | \text{modelo})$

Maximum posterior   $\text{Pr}(\text{modelo} | \text{datos})$

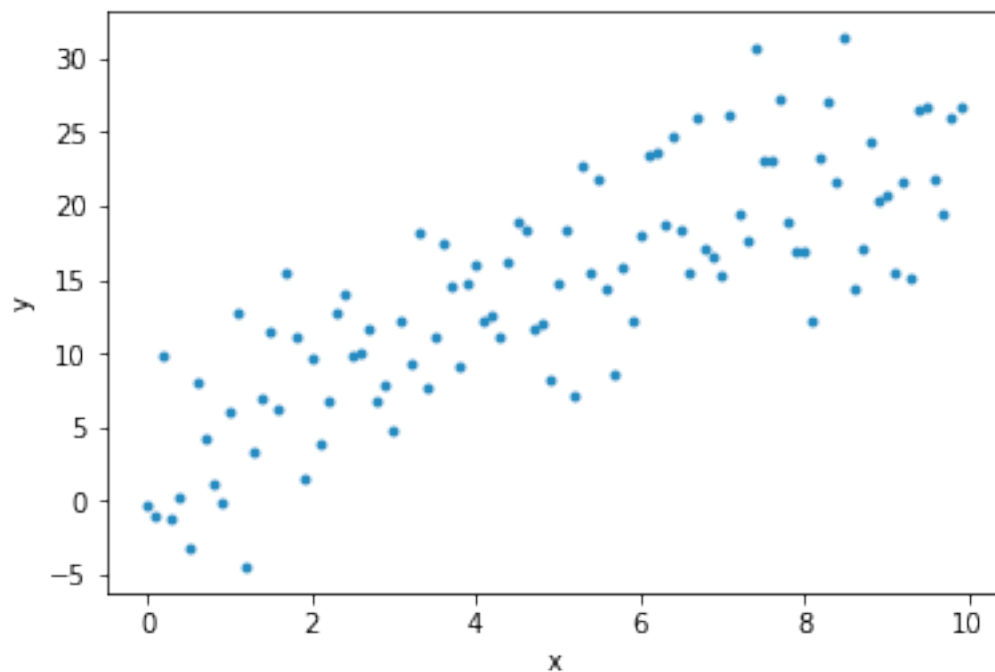
(acercamiento Bayesiano)

- Y escoger el algoritmo que nos permita encontrar la solución (valor de los parámetros óptimos) de la manera más eficiente.
- Una de las ventajas de las técnicas de machine learning es que no es necesario tener un modelo para los datos.

# MÍNIMOS CUADRADOS

Es una de las funciones objetivo más usadas, y la que está implementada en varias funciones de python.

Se busca el mejor ajuste entre unos datos y un modelo, mediante la minimización del error cuadrático (o residuos).



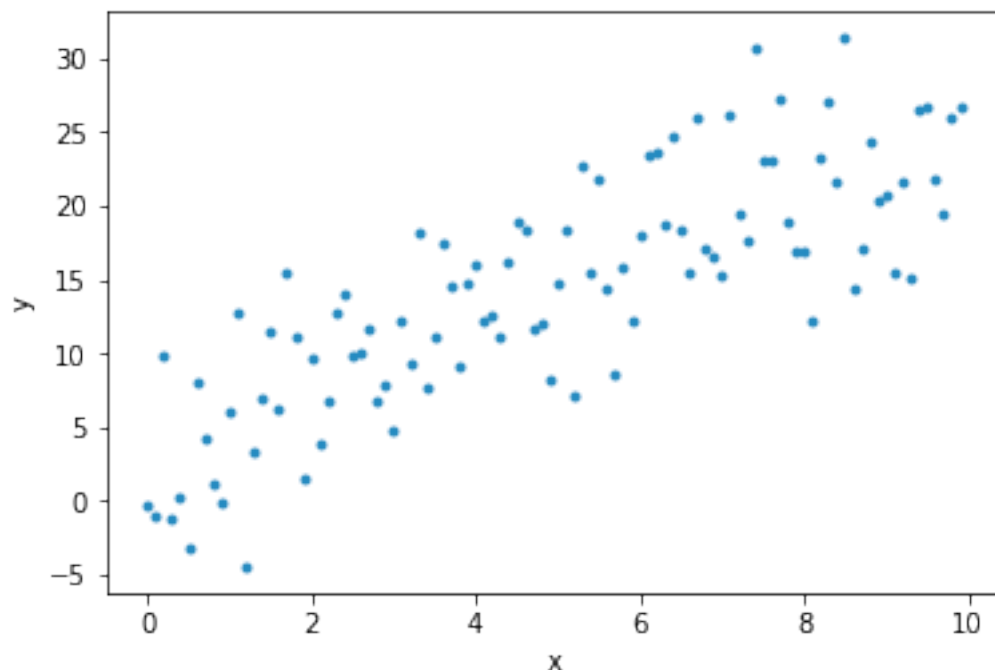
$$E = \sum_i^N \frac{(y_i - f(x_i))^2}{N}$$

En este caso,  $f(x) = ax + b$

# MÍNIMOS CUADRADOS

Es una de las funciones objetivo más usadas, y la que está implementada en varias funciones de python.

Se busca el mejor ajuste entre unos datos y un modelo, mediante la minimización del error cuadrático (o residuos).



$$E = \sum_i^N \frac{(y_i - f(x_i))^2}{N}$$

En este caso,  $f(x) = ax + b$

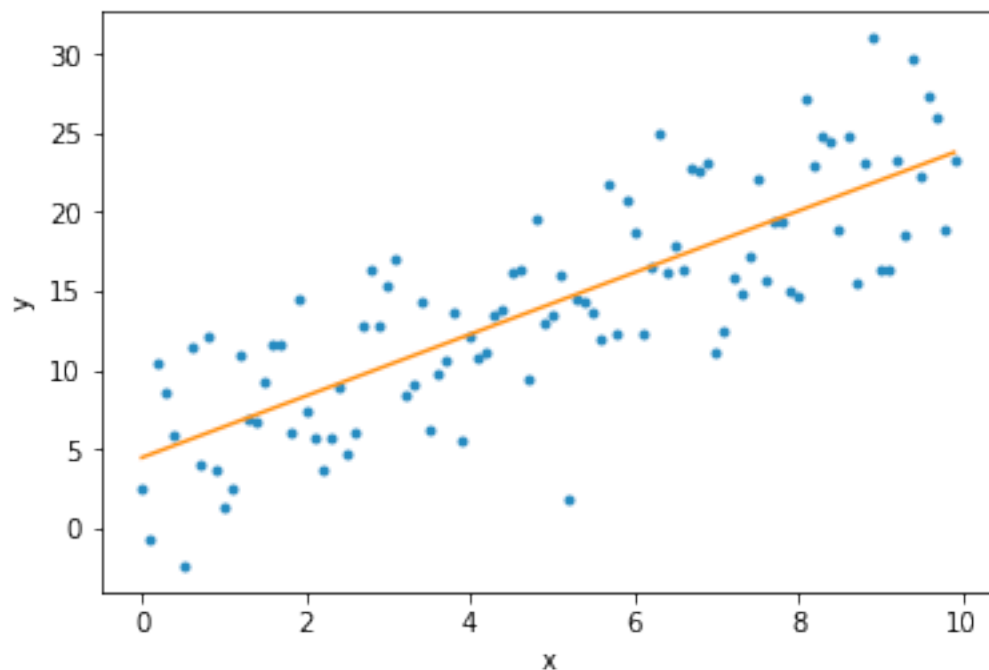
Fijaros que x e y son los datos,  
son valores conocidos. E al final  
es una función  $E(a,b)$



# MÍNIMOS CUADRADOS

Es una de las funciones objetivo más usadas, y la que está implementada en varias funciones de python.

Minimizamos la función  $E$  con respecto de  $a$  y  $b$ , con cualquier método



$$E = \sum_i^N \frac{(y_i - f(x_i))^2}{N}$$

En este caso ajustamos a una recta

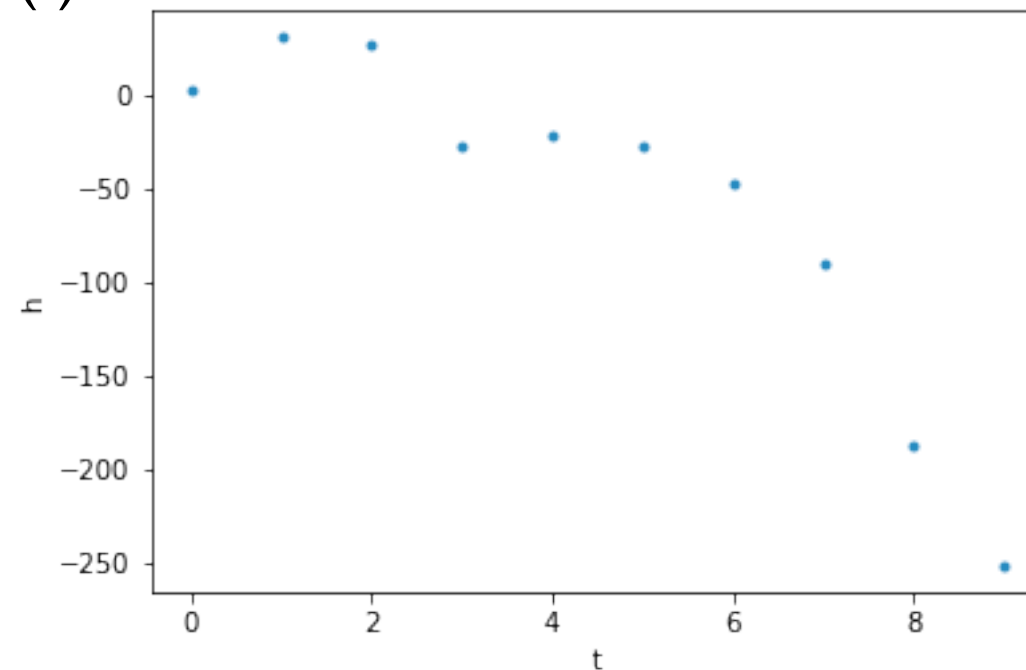
$$f(x) = ax + b$$

Y encontramos  $a$  y  $b$  que mejor ajustan  $f(x)$  a los datos.

# MÍNIMOS CUADRADOS

**Modelo teórico.** Tratamos de ajustar un modelo teórico (lineal o no-lineal) a los datos que tenemos.

**Ejemplo.** Tomamos una serie de datos: altura (h) que coge un objeto lanzado hacia arriba a cada tiempo (t).

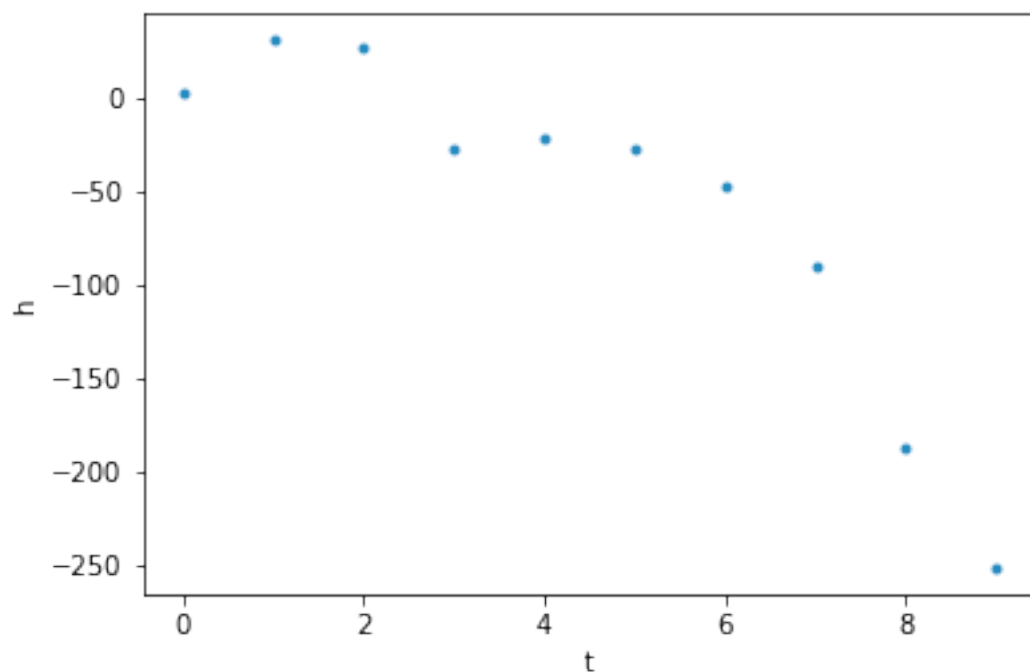


Y sabemos por las ecuaciones de movimiento que la trayectoria que seguirá será:

$$x(t) = v_0 t + \frac{1}{2} a t^2$$

# MÍNIMOS CUADRADOS

Debido a errores en la medida, no tenemos la ley exacta, pero podemos hacer un ajuste para obtener los parámetros

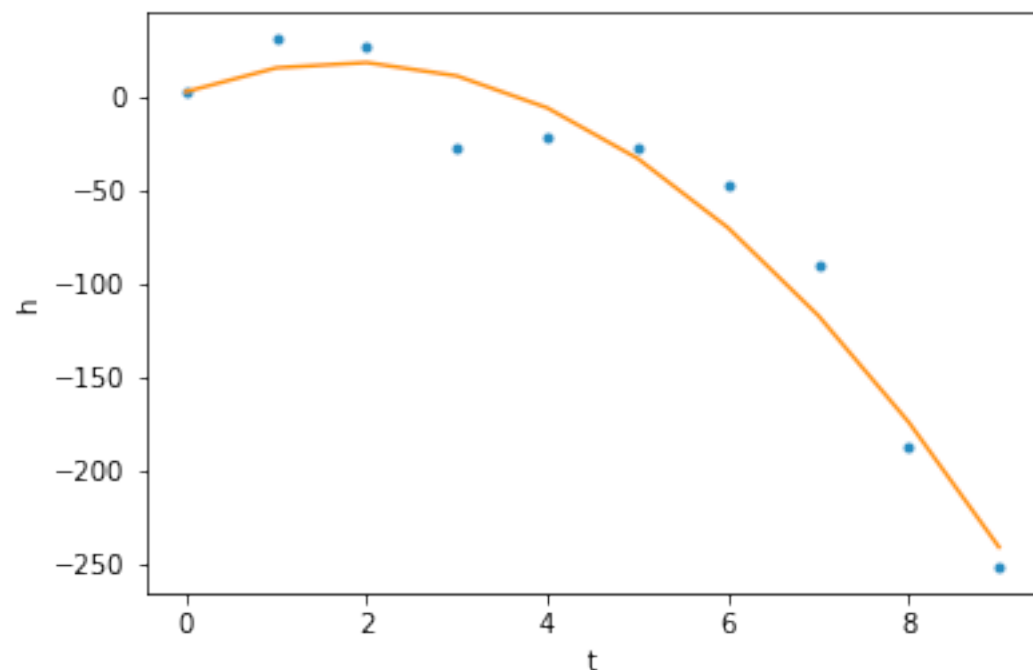


El error cuadrático será menor cuando la velocidad inicial y aceleración sean las que hacen que el modelo  $x(t)$  se ajuste más a los datos.

$$Err = \sum_i^N \frac{(h(t_i) - x(t_i))^2}{N}$$

# MÍNIMOS CUADRADOS

Minimizamos Err y encontramos la velocidad inicial y la aceleración



Los parámetros del modelo que mejor se ajustan a los datos son:

$$v_0 = 18m/s, a = -10m/s^2$$

Para generar los puntos utilicé:

$$v_0 = 20, a = -9.8$$



# Modelizar

Como sabemos si un ajuste está bien hecho?

**Goodness of fit.** Hay varios tests que nos pueden decir cuan bueno es un ajuste. Dependiendo del método o estadístico que estemos usando:

R-squared

Minimos cuadrados (sigue distribución  $\sigma\chi^2$ )  
Mínimos cuadrados pesados ( $\chi^2$ )

## R-squared

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2, \quad R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

R-squared puede ir entre 0 y 1 . (Los errores en la estimación deberían ser menores a la varianza de la muestra)

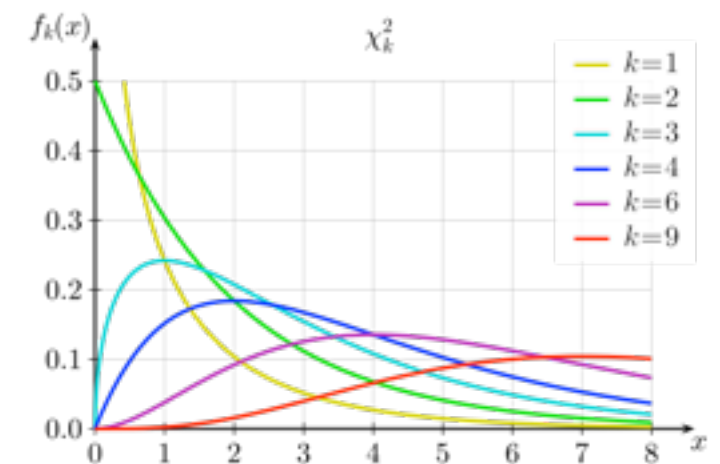
$$\chi^2$$

- **chi cuadrado.** Es un estadístico que compara unos datos con su valor esperado dividido por la varianza de estos datos

$$\chi^2 = \sum_i \frac{(d_i - m_i(\theta))^2}{\sigma_i^2}$$

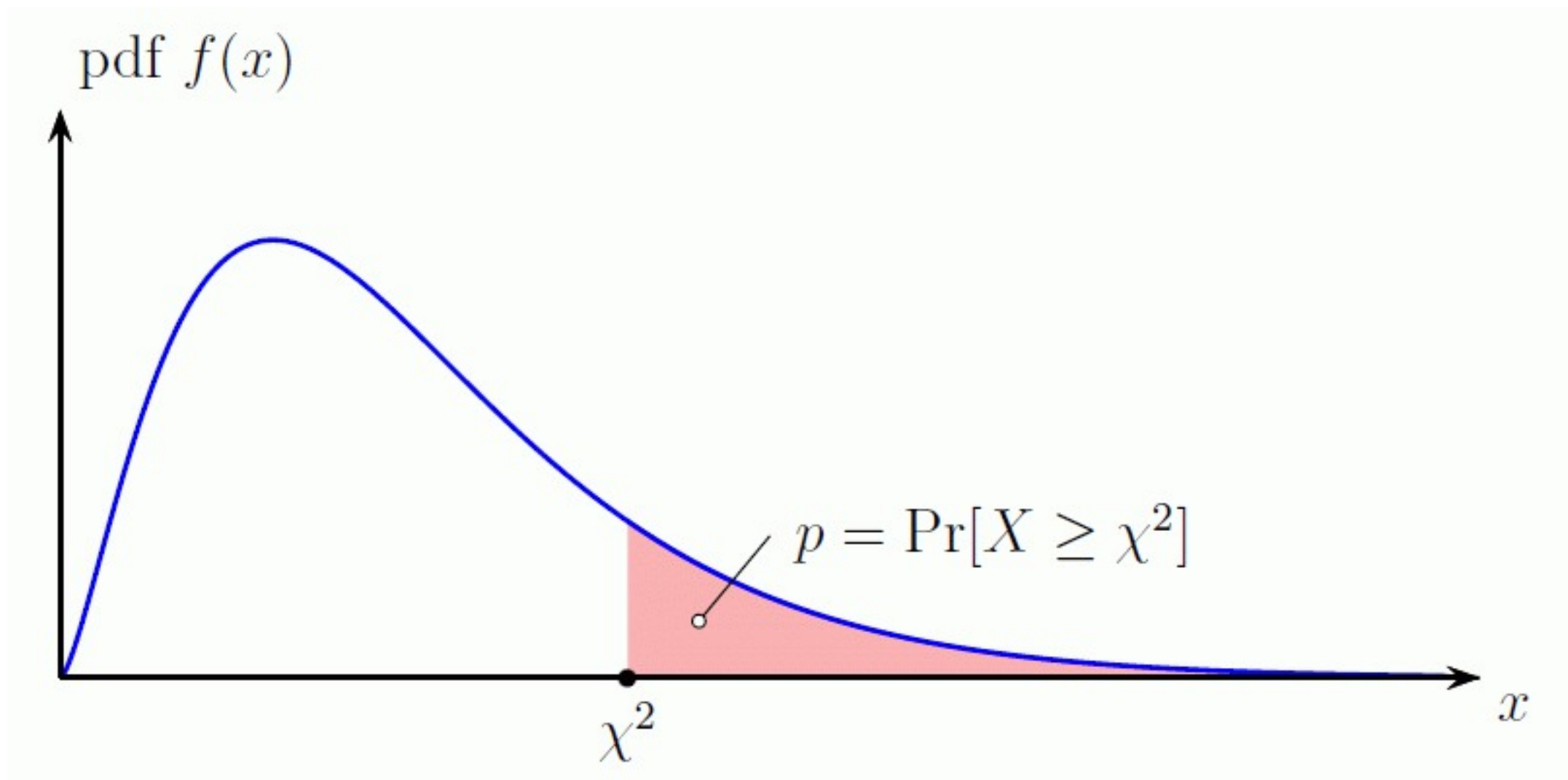
- **Grados de libertad.** Número de variables menos el número de parámetros. N-p
- **Distribución.** Dado unos grados de libertad, la distribución del  $\chi^2$  es conocida y esto nos permite estimar si nuestro fit
- **mínimos cuadrados.** En el caso donde al caso la varianza es igual en todas las variables, el chi2 es igual a los residuos cuadráticos
- **Correlaciones.** Si las variables están correlacionadas, la forma general es:

$$\chi^2 = x C^{-1} x^T$$



$$\chi^2$$

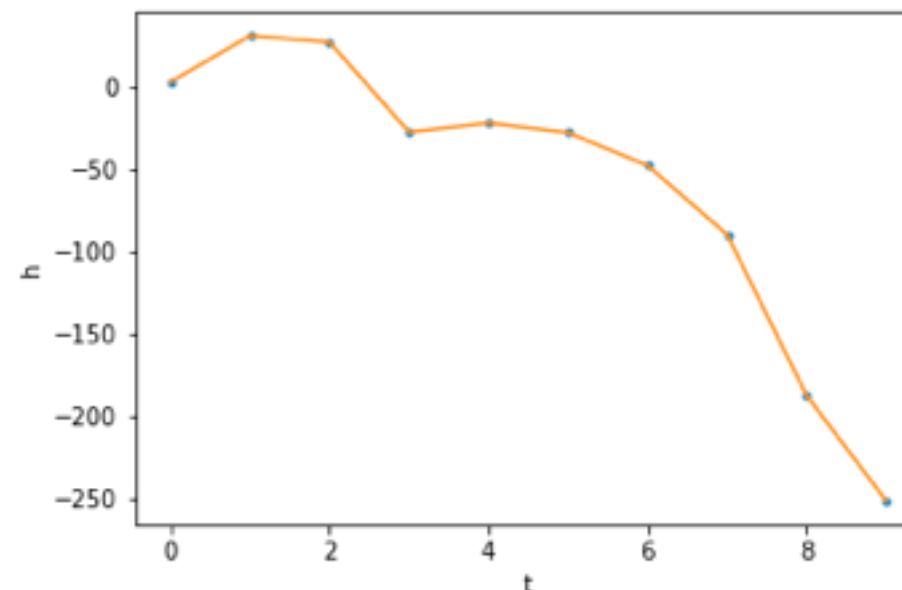
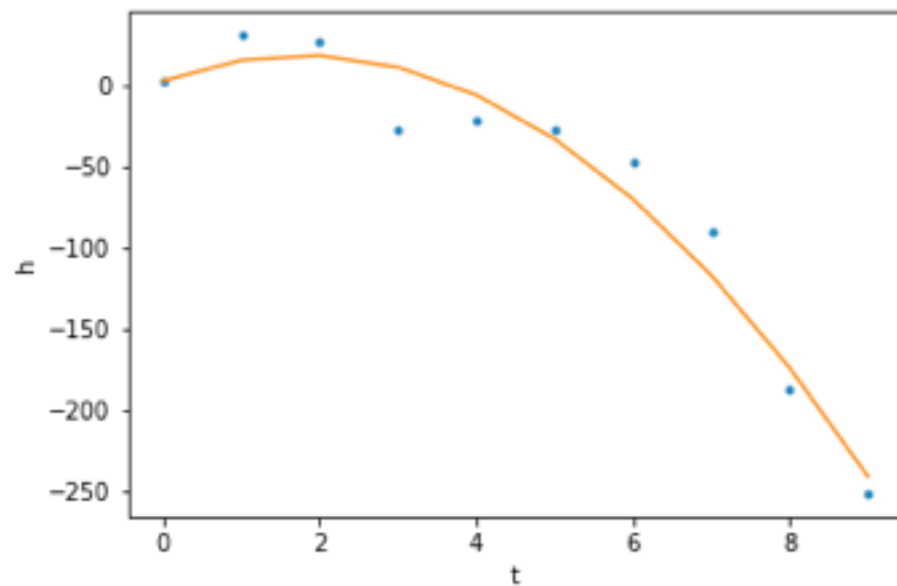
Comparamos el valor mínimo de la función (mínimos cuadrado o chi cuadrado) que nos da nuestro ajuste, con su distribución teórica, si está muy a la cola no nos creemos el ajuste, si está dentro de la distribución nos creemos el ajuste.



# Modelizar

**PERO.** Que un ajuste esté bien hecho no implica que el modelo sea el correcto. El modelo tiene que estar bien motivado!!

OVERFITTING!!



Si el modelo no es del todo conocido es necesario tener muchos datos para hacer el ajuste con una muestra y luego probarlo en la otra. Si tenemos muy buen fit en una y en la otra falla es que hemos sobreajustado (más en *machine learning*)

El ajuste de la derecha nos dará mejor R-squared y mejor chi-squared



# Modelizar

**Maximum likelihood.** (o máxima verosimilitud)

La **likelihood** se entiende como la probabilidad de los datos suponiendo un modelo. Es decir, nos cuantifica cuan creíbles son unos datos, asumiendo que el modelo que asumimos es el correcto.

Si nuestra muestra es lo suficientemente grande (tenemos una cantidad grande de datos) los parámetros que maximizan esa probabilidad son:

- **Inesgados** (el valor que encontramos es el valor esperado.)
- **Óptimos** (son los de menor error que podemos encontrar)

Si los errores en los datos son Gaussianos, la verosimilitud viene dada por:

$$P(d|\theta) \propto e^{-x C^{-1} x^T}$$
$$x = d - m(\theta)$$

$C$  matriz de covarianza

Maximizar la verosimilitud sería equivalente a minimizar

$$\chi^2 = x C^{-1} x^T$$

# Modelizar

**Maximum likelihood.** (o máxima verosimilitud)

Si no hay correlaciones la matriz de covarianza es diagonal con valores  $\sigma_i^2$

$$\chi^2 = \sum_i \frac{(d_i - m_i(\theta))^2}{\sigma_i^2}$$

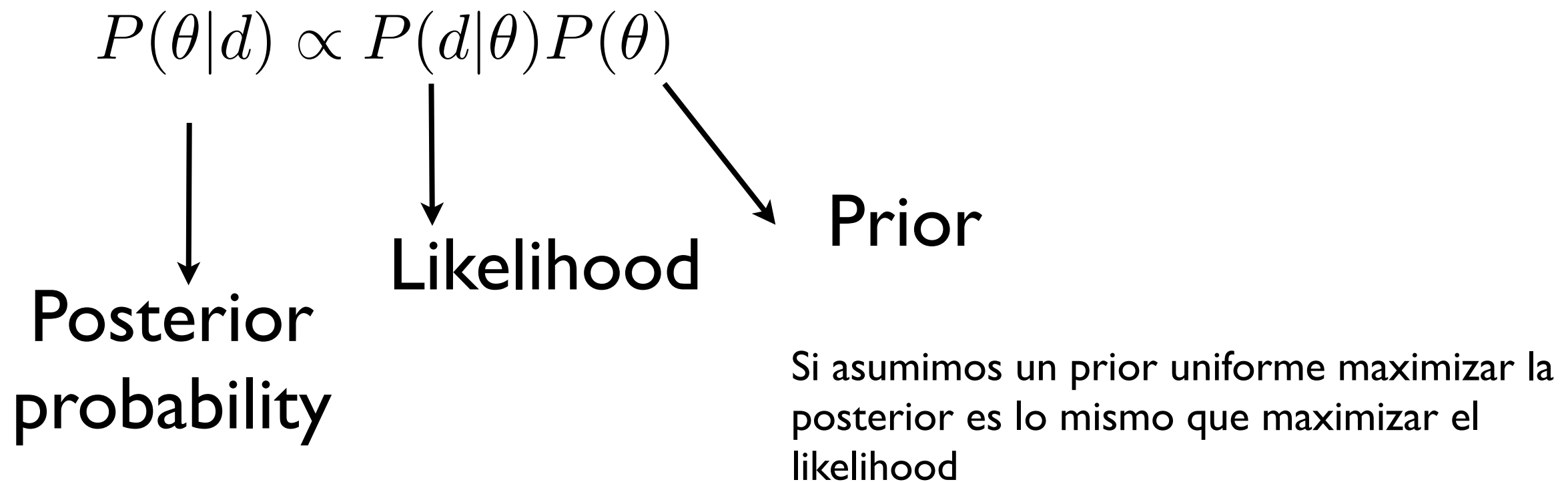
y se parece mucho al problema de mínimos cuadrados (exactamente igual si la varianza de las variables es la misma para todas)

Tanto si hay correlaciones como no se resuelve minimizando la función  $\chi^2$

# Modelizar

Hay varios tests para saber si el ajuste está bien hecho, o para saber si los datos son verosímiles, pero ninguno de ellos nos dirá si el modelo escogido es el bueno.

En **teorema de Bayes** nos permite obtener la probabilidad de un modelo dadas unas observaciones. Normalmente se hace un sampleado para poder obtener la distribución de probabilidad.



# Ajustes con Python

- Las funciones para ajustar de python usan como función objetivo el error mínimo cuadrado. (Least Squares method o mínimos cuadrados)  $E = \sum_i^N \frac{(y_i - f(x_i))^2}{N}$ 
  - si es un polinomio: `numpy.polyfit(x,y, grado)` --> resultado : las constantes del polinomio
  - curva general: `optimize.curve_fit(fun,x,y)` fun=función que queremos minimizar, --> encuentra los parámetros libres de fun
- Usando cualquier otra función objetivo:
  - `optimize.minimize(Objefunc,x0)`

Vemos ejemplo en `leastSquares.ipynb`