CHAPTER 2: REGRESSION. LINEAR MODELS (SYSTEMS)

Grado en Ingeniería Informática

Curso 2014 / 15

Topics

1. Introduction
2. Linear model
3. Reducing to a linear model
4. Numerical optimization

INTRODUCTION

Regression

- It assign the input pattern the corresponding output to a continuous variable

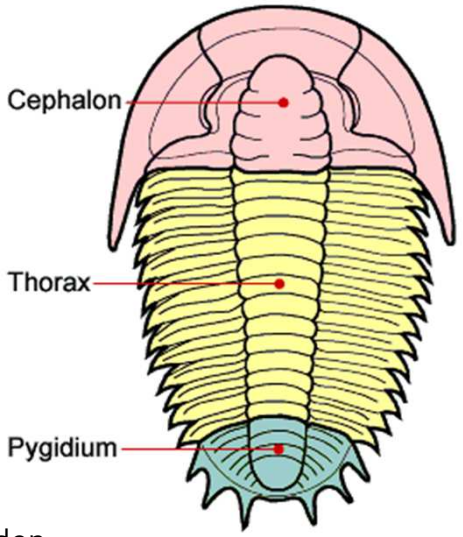
$$y = f : R^n \Rightarrow R$$

- General case:

$$y = f(x) \qquad x \in R^n, y \in R^m$$

Regression

- Example: Size estimation of Trilobites
  - ✓ Under most preservation conditions, it is difficult to find complete copies of Trilobites
  - ✓ The head (cephalon) is more common
  - ✓ Therefore, it is useful for being able to estimate the body from measure of the head, to establish whom prove the better determination of total size.



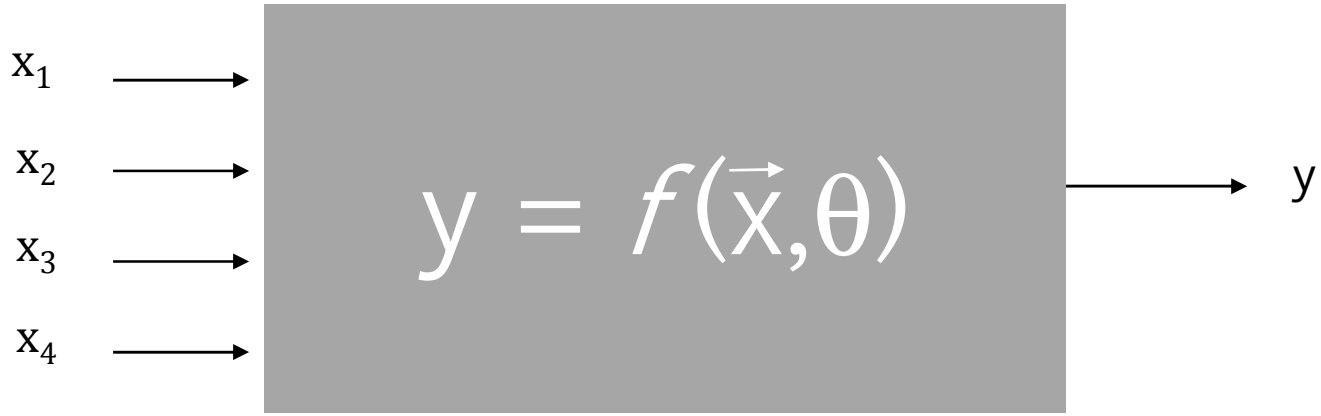
Norman MacLeod. Keeper of Palaeontology, The Natural History Museum, London

Table 1. Trilobite Data[1]

Genus	Body Length (mm)	Glabellar Length (mm)	Glabellar Width (mm)
Acaste	23.14	3.50	3.77
Balizoma	14.32	3.97	4.08
Calymene	51.69	10.91	10.72
Ceraurus	21.15	4.90	4.69
Cheirurus	31.74	9.33	12.11
Cybantyx	36.81	11.35	10.10
Cybeloides	25.13	6.39	6.81
Dalmanites	32.93	8.46	6.08
Delphion	21.81	6.92	9.01
Ormathops	13.88	5.03	4.34
Phacopdina	21.43	7.03	6.79
Phacops	27.23	5.30	8.19
Placopoaria	38.15	9.40	8.71
Pricyclopyge	40.11	14.98	12.98
Ptychoparia	62.17	12.25	8.71
Rhenops	55.94	19.00	13.10
Sphaerexochus	23.31	3.84	4.60
Toxochasmops	46.12	8.15	11.42
Trimerus	89.43	23.18	21.52
Zacanthoides	47.89	13.56	11.78
Mean	36.22	9.37	8.98
Std. Deviation	18.63	5.23	4.27

Generic system (model)

System = Model

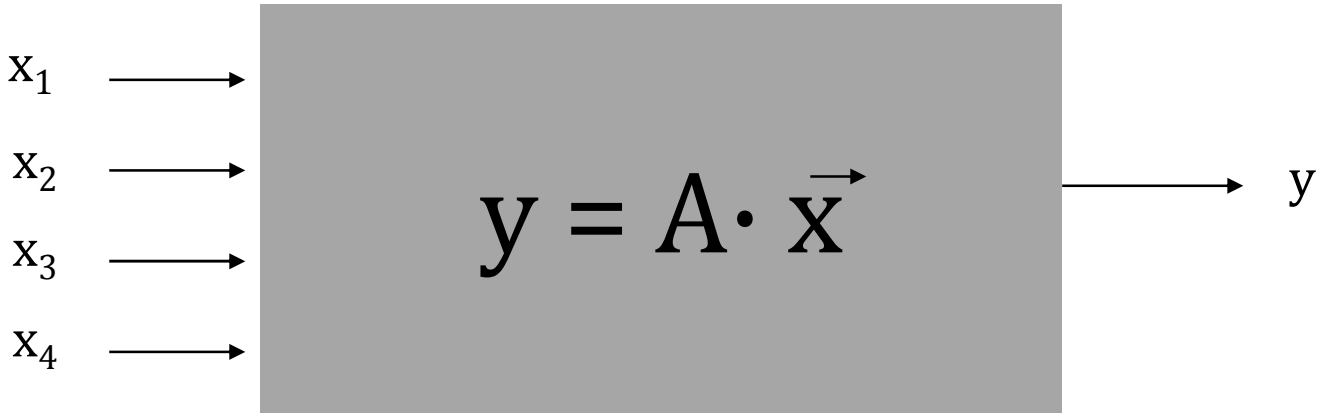


x : system logons

θ : system internal parameters

y : system output

Linear model



$$y = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + a_0$$

LINEAR MODEL

Fitting a line (N = 1)

- Initially, we consider only one *x*. Thus, from this data set

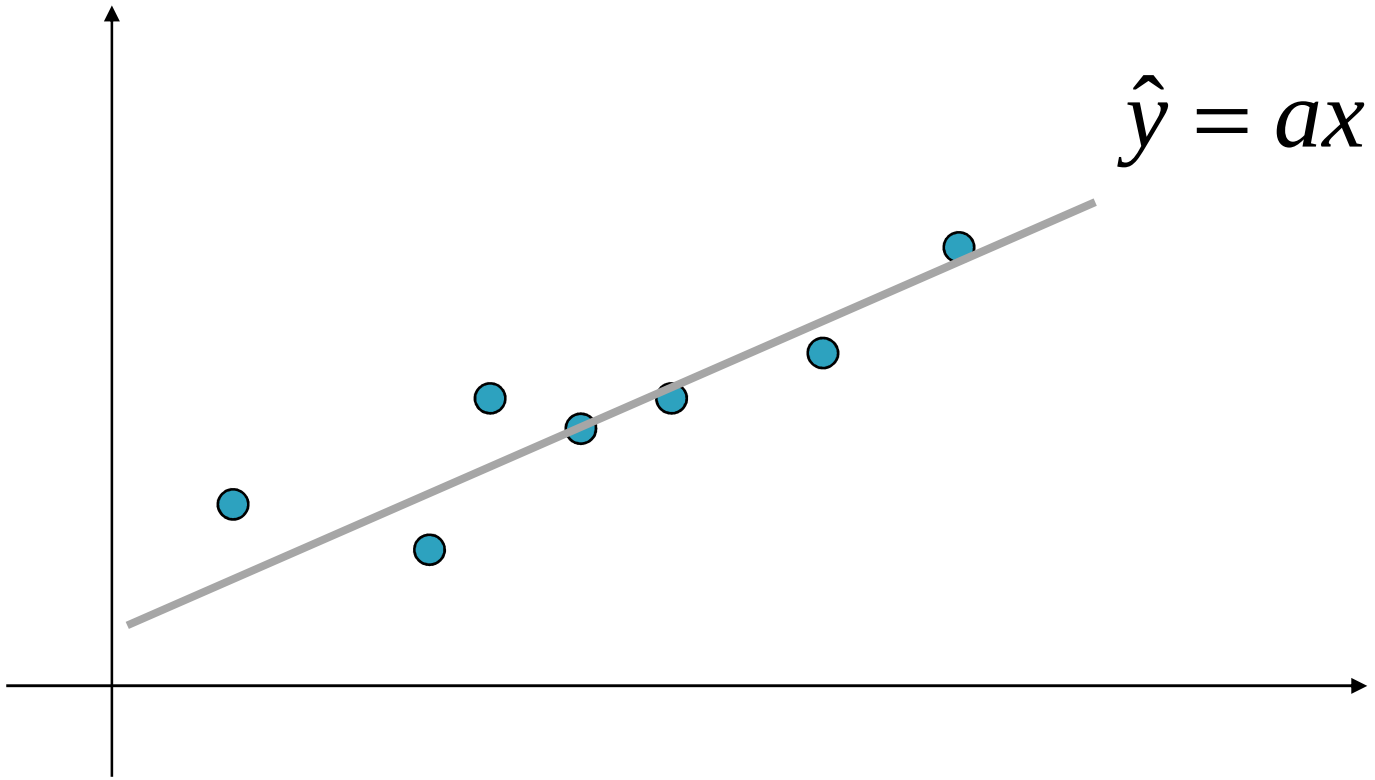
$$\{(x_i, y_i) \quad i = 1, \ldots, n\}$$

- we look for the function

$$\hat{y} = ax + b$$

- that most closely approximates this data
- In other words, what values do a and b take?

Fitting a line (N = 1)

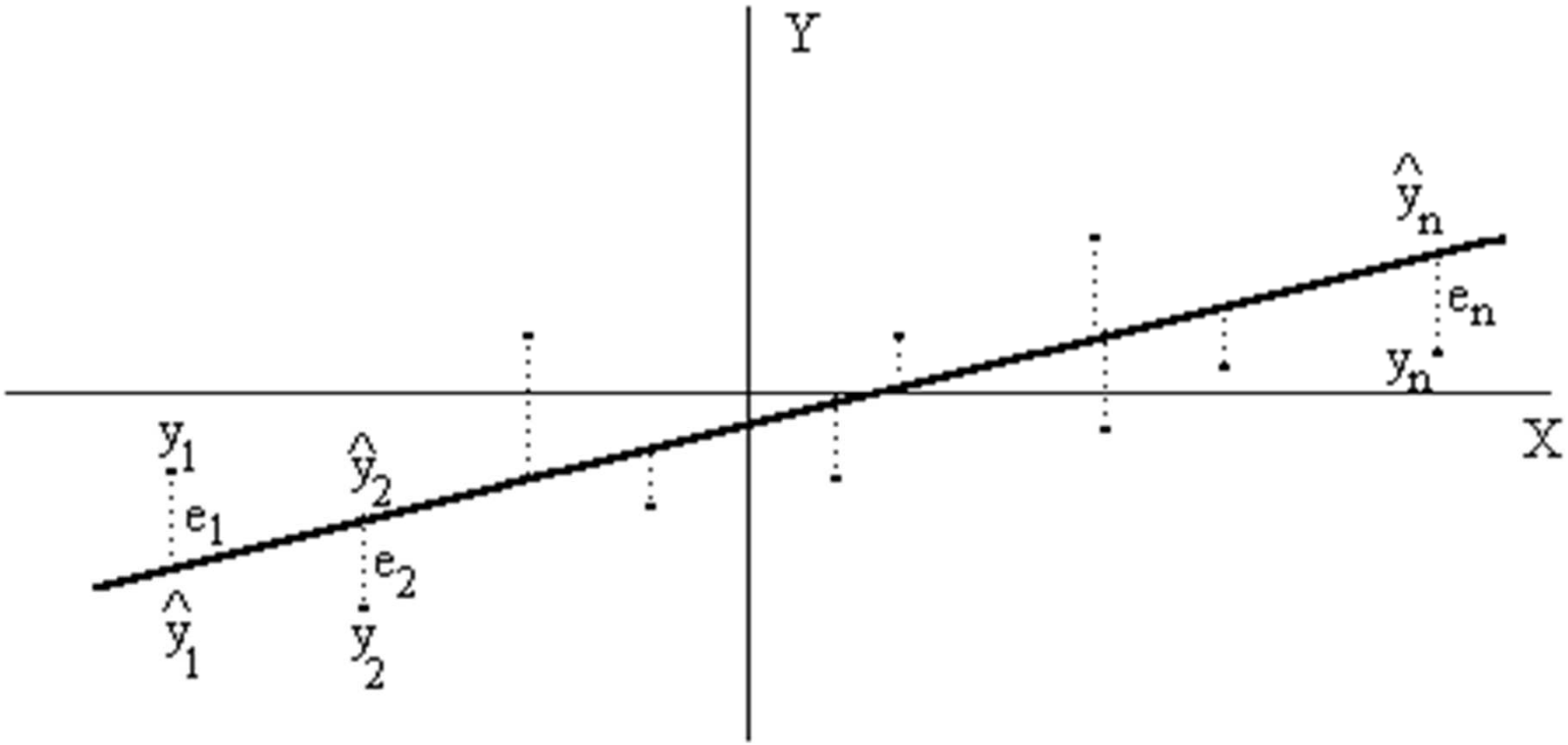


$$\hat{y} = ax + b$$

Fitting a line (N = 1)

1. To define the error in each point

$$e_i = y_i - \hat{y}(x_i) = y_i - (a_i x + b)$$



Fitting a line (N = 1)

2. To define a criterion of the error

$$E(a,b) = \sum_{i=i}^{n} e_i^{\,2} = \sum_{i=i}^{n} (y_i - (ax_i + b))^2$$

- Other criterion could be the absolute values sum of errors, for example
- The use of square errors sum is the possibility to calculate easily the error derivative

Fitting a line (N = 1)

3. To take derivative with respect to each parameter and equating to zero

$$\left.\begin{aligned}\frac{\partial E(a,b)}{\partial a}=0\\ \frac{\partial E(a,b)}{\partial b}=0\end{aligned}\right\}$$

$$\frac{\partial E}{\partial a}=2\sum_{i=1}^{n}\left(y_i-(ax_i+b)\right)(-x_i)$$

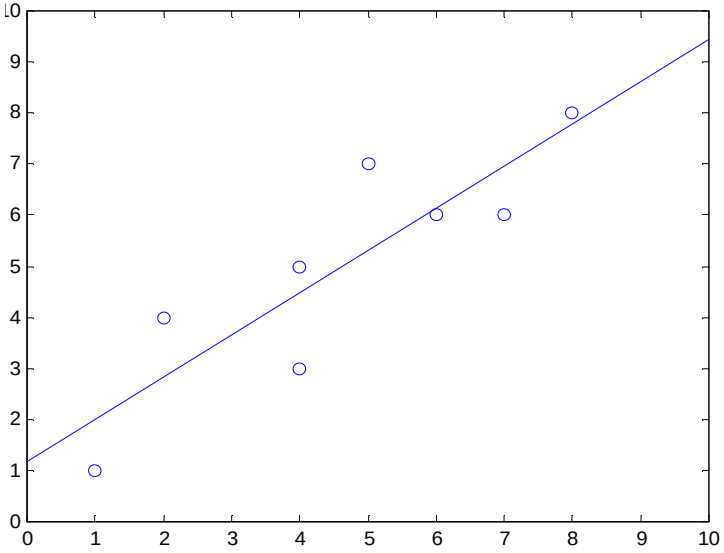
$$\frac{\partial E}{\partial b}=2\sum_{i=1}^{n}\left(y_i-(ax_i+b)\right)(-1)$$

Equating to zero

$$\begin{pmatrix}\sum_{i=1}^{n}x_i^{\,2} & \sum_{i=1}^{n}x_i\\ \sum_{i=1}^{n}x_i & n\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix}=\begin{pmatrix}\sum_{i=1}^{n}y_i x_i\\ \sum_{i=1}^{n}y_i\end{pmatrix}$$

Fitting a line (N = 1)

```
x=[1 2 4 4 5 6 7 8];
y=[1 4 3 5 7 6 6 8];
Sxx=sum(x.*x); Sx=sum(x);
Sxy=sum(x.*y); Sy=sum(y);
n=length(x);
A=[Sxx Sx; Sx n]; b=[Sxy Sy]';
sol=A\b;
plot (x,y,'o'); hold on;
axis([0 10 0 10]);
xr=[0 10];
yr=[sol(2) sol(1)*10+sol(2)];
plot (xr,yr); hold off;
E=sum((y-(sol(1)*x+sol(2))).^2);
» Sol =
    0.8276
    1.1724
» E = 8.6897
```



Linear model (M > 1)

- Given

$$\{(\vec{x}_i, y_i) \quad i = 1,\ldots,n\}$$

were $$\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$$

- In this case:

$$\hat{y} = a_1 x_1 + a_2 x_2 + \ldots + a_m x_m + a_0$$

Linear model (N > 1)

1. To define the error in each point:

$$e_i = y_i - \hat{y}(\vec{x}_i) = y_i - \left(a_1 x_{i1} + a_2 x_{i2} + \ldots + a_m x_{im} + a_0\right)$$

$$\underbrace{\begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}}_{r} = \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{b} - \underbrace{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} & 1 \\ x_{21} & x_{22} & \cdots & \vdots & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ x_{n1} & x_{n2} & \cdots & x_{nm} & 1 \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_0 \end{pmatrix}}_{x}$$

Linear model (N > 1)

2. The error criterion will be:

$$E = \sum_{i=i}^{m} e_i^{\,2} = \begin{pmatrix} e_1 & e_2 & \cdots & e_m \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{pmatrix} = r^t \cdot r$$

$$E = r^t \cdot r = (b - Ax)^t (b - Ax) = b^t b - b^t (Ax) - (Ax)^t b + (Ax)^t (Ax) = b^t b - b^t Ax - x^t A^t b + x^t A^t Ax$$

Since $b^t Ax$ has dimensions 1 x 1, it is satisfied that

$$b^t Ax = \left(b^t Ax\right)^t = x^t A^t b$$

Therefore $$E = b^t b - 2x^t A^t b + x^t A^t Ax$$

Linear model (N > 1)

- Given a column vector z and a matrix M, we have to remember that:

$$\frac{\partial z^t M}{\partial z} = M \qquad \frac{\partial Mz}{\partial z} = M^t$$

$$\frac{\partial z^t Mz}{\partial z} = \left(M + M^t\right) \cdot z$$

Linear model (N > 1)

3. To take derivative of error with respect to parameters vector *x:*

$$\frac{\partial E}{\partial x} = \frac{\partial\left[b^t b - 2b^t Ax + x^t A^t Ax\right]}{\partial x} = \frac{\partial\left[b^t b\right]}{\partial x} - \frac{\partial\left[2b^t Ax\right]}{\partial x} + \frac{\partial\left[x^t A^t Ax\right]}{\partial x}$$

$$\left.\begin{aligned} &\frac{\partial\left[b^t b\right]}{\partial x} = 0 \\ &\frac{\partial\left[2x^t A^t b\right]}{\partial x} = 2A^t b \\ &\frac{\partial\left[x^t A^t Ax\right]}{\partial x} = \left(A^t A + \left(A^t A\right)^t\right)x = 2A^t Ax \end{aligned}\right\} \Rightarrow \quad \frac{\partial E}{\partial x} = 0 - 2A^t b + 2A^t Ax$$

Linear model (N > 1)

3. … and aquating to zero:

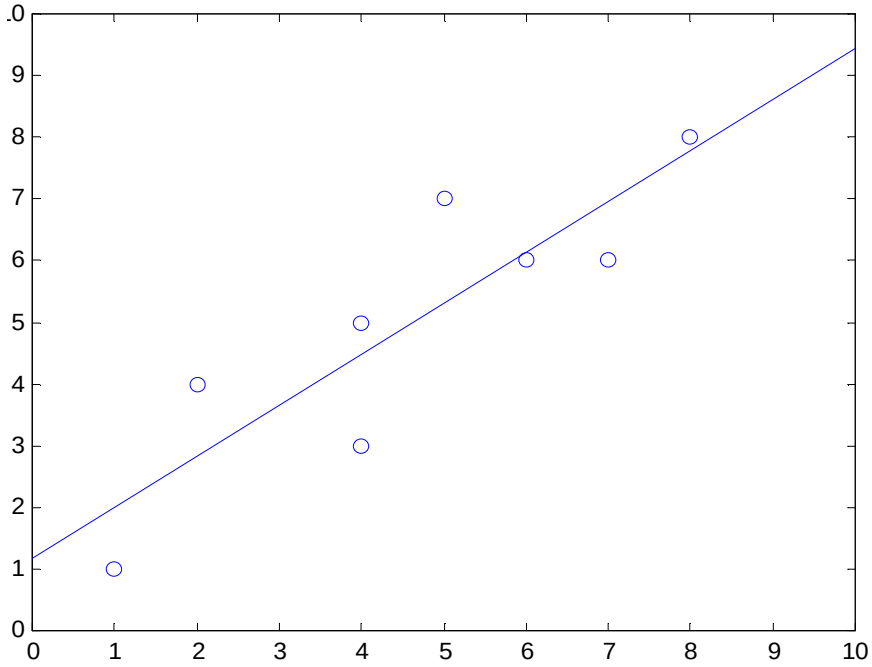
$$-2A^t b + 2A^t Ax = 0$$

$$A^t b = A^t Ax$$

$$x = \underbrace{\left(A^t \cdot A\right)^{-1} \cdot A^t}_{\text{Pseudoinversa de A}} \cdot b$$

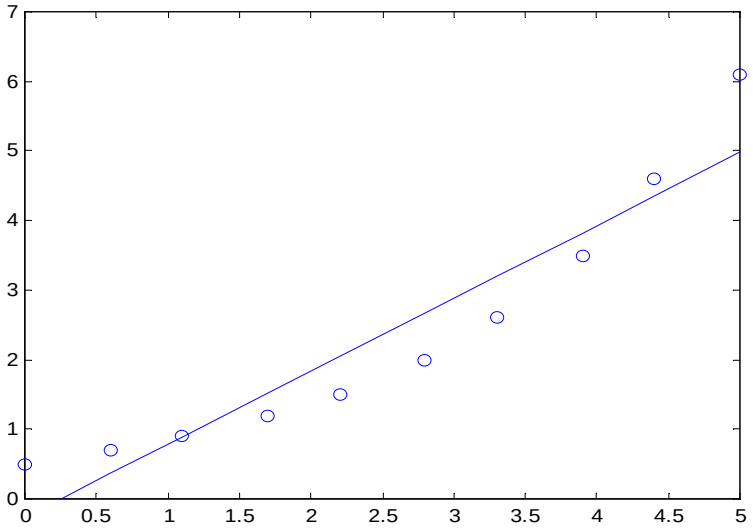
Fitting a line (N = 1)

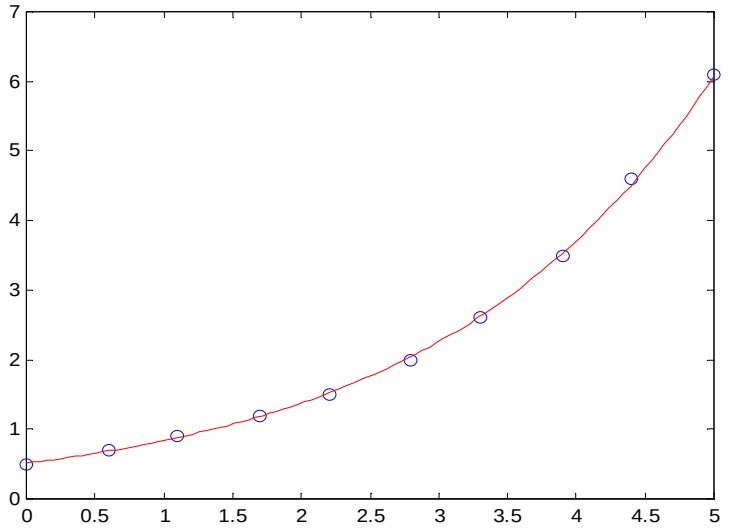
```
x=[1 2 4 4 5 6 7 8]';
y=[1 4 3 5 7 6 6 8]';
A = [x ones(size(x))];
sol = inv(A'*A)*(A'*y);
r = y-A*sol;
E = r'*r;
» Sol =
    0.8276
    1.1724
» E = 8.6897
```



REDUCING TO A LINEAR MODEL

Exponential model





- When the linear model is insufficient, it is possible to reduce some models to linear model using a variables change

Exponential model

- Given a curve

$$y = Ae^{Bx}$$

- Taking the logarithms

$$\ln(y) = Bx + \ln(A)$$

- And doing the variables change

$$\hat{y} = \ln(y) \qquad C = \ln(A)$$

$$\hat{y} = Bx + C$$

Exponential model

```
x=[0 0.6 1.1 1.7 2.2 2.8 3.3 3.9 4.4 5]';
y=[0.5 0.7 0.9 1.2 1.5 2 2.6 3.5 4.6 6.1]';
% cambio de variable en los datos
yp=log(y);
% solucion del problema lineal
A = [x ones(size(x))];
sol = inv(A'*A)*(A'*yp);
B=sol(1); C=sol(2);
% calculamos los parámetros de la exponencial
A=exp(C);
plot (x,y,'o');
axis([0 5 0 7]); hold on;
plot (x,A*exp(B*x),'r'); hold off;
```

Other models

- Similarly, it is possible to use the below models:

Function	Model	Vars. change
$y = C \cdot x^{A}$	$\ln(y) = A \cdot \ln(x) + \ln(C)$	$\hat{y} = \ln(y)$ $\hat{x} = \ln(x)$ $C = e^{B}$
$y = C \cdot x \cdot e^{Ax}$	$\ln\left(\frac{y}{x}\right) = A \cdot x + \ln(C)$	$\hat{y} = \ln\left(\frac{y}{x}\right)$ $C = e^{B}$

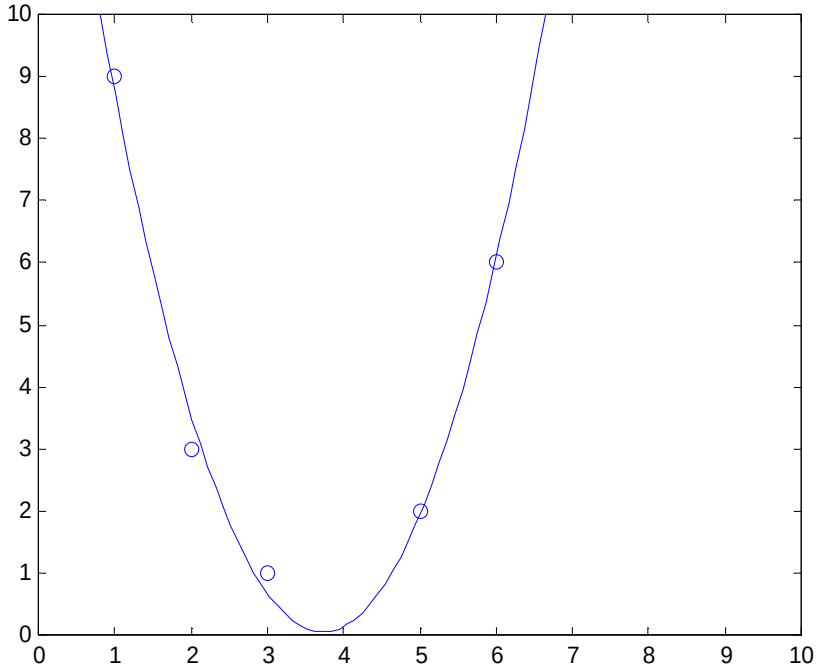
Polynomial model (N = 1)

- We have to transform our data set in the following way:

$$\bar{x}_i = \left(x_i, x_i^2, x_i^3, \ldots, x_i^m \right)$$

Polynomial model (N = 1)

```
x=[1 2 3 5 6]';
y=[9 3 1 2 6]';
x2=x.*x;
A = [x2 x ones(size(x))];
polisol = inv(A'*A)*(A'*y);
plot (x,y,'o');
axis([0 10 0 10]); hold on;
xp=linspace(0,10);
plot (xp,polyval(polisol,xp));
hold off;
```



NUMERICAL OPTIMIZATION

Introduction

- Suppose the error surface has a known expression

$$E = x^4 - 3 \cdot x^2 + x + y^2 + x \cdot y$$

- The analytical solution for possible minimum are determined by:

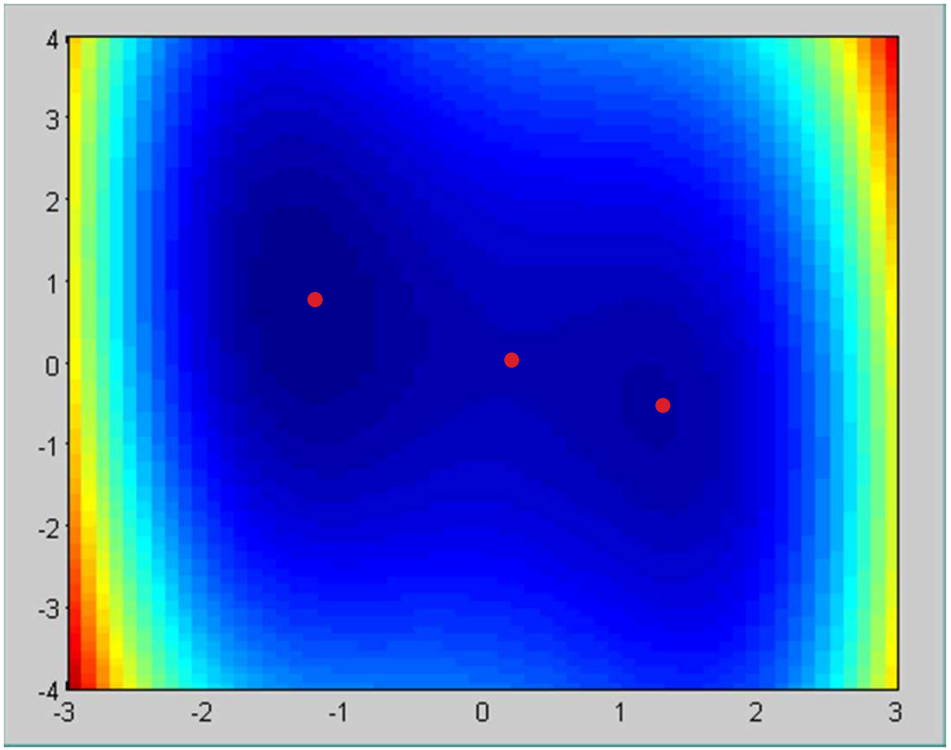
$$\frac{\partial E}{\partial x} = 4 \cdot x^3 - 6 \cdot x + 1 + y = 0 \qquad \frac{\partial E}{\partial y} = 2 \cdot y + x = 0$$

- Example:

```
fzero(inline('4*x.^3 - 6*x + 1 - x/2'),-1) = -1.3457
fzero(inline('4*x.^3 - 6*x + 1 - x/2'),0)  =  0.1562
fzero(inline('4*x.^3 - 6*x + 1 - x/2'),2)  =  1.1895
```

Iterative solution

```
[x,y]=meshgrid(-3:0.1:3,-4:0.1:4);
z = (x.^4 - 3*x.^2 + x + y.*y + x.*y);
pcolor(x,y,z),shading flat, colorbar
```



Possible minimum:

[-1.3457 , 0.6728]

[0.1562 , -0.0781]

[1.1895 , -0.5947]

The most obvious algorithm: Gradient descent

- It is a optimization algorithm of first order
- It is used to find the local maximum or minimum in a function
- Gradient descent is based on the observation that if the multivariable function $F(x)$ is defined and differentiable in a neighborhood of **a** point , then $F(x)$ decreases fastest if one goes from **a** in the direction of the negative gradient of F at **a**

The most obvious algorithm: Gradient descent

- Let $-\nabla F(a)$ be the negative gradient of function $F(x)$ at **a** point. It follows that, if:

$$b = a - \gamma \nabla F(a)$$

- For γ small enough, then $F(a) \geq F(b)$
- With this observation in mind, one starts with a guess x_0 for a local minimum of F, and considers the sequence $x_o, x_1, x_2, \ldots$ such that

$$x_{n+1} = x_n - \gamma \nabla F(n), n \geq 0$$

The most obvious algorithm: Gradient descent

- The negative gradient is always in downward direction
- The algorithm:
  1. Choose a initial point, x_0
  2. The direction is equal to negative gradient
  3. Step length:

$$p_k = -\nabla f(x_k)$$

  4. Modify x: $x_{k+1} = x_k + \gamma_k p_k$
  5. Return to step 2

Gradient descent

```
[x,y]=meshgrid(-3:0.1:3,-4:0.1:4);
z = (x.^4 - 3*x.^2 + x + y.*y + x.*y);
while(1),
    close all
    pcolor(x,y,z),shading flat;hold on;
    sol = ginput(1)';
    for i=1:100,
       x=sol(1,i);
       y=sol(2,i);
       dx = 4 * x.^3 - 6*x + 1 + y;
       dy = 2 * y + x;
       sol(:,i+1) = sol(:,i) - 0.1 * [dx;dy];
       plot([sol(1,i) sol(1,i+1)],[sol(2,i) sol(2,i+1)],'r',LineWidth',3)
       pause
    end
end
```

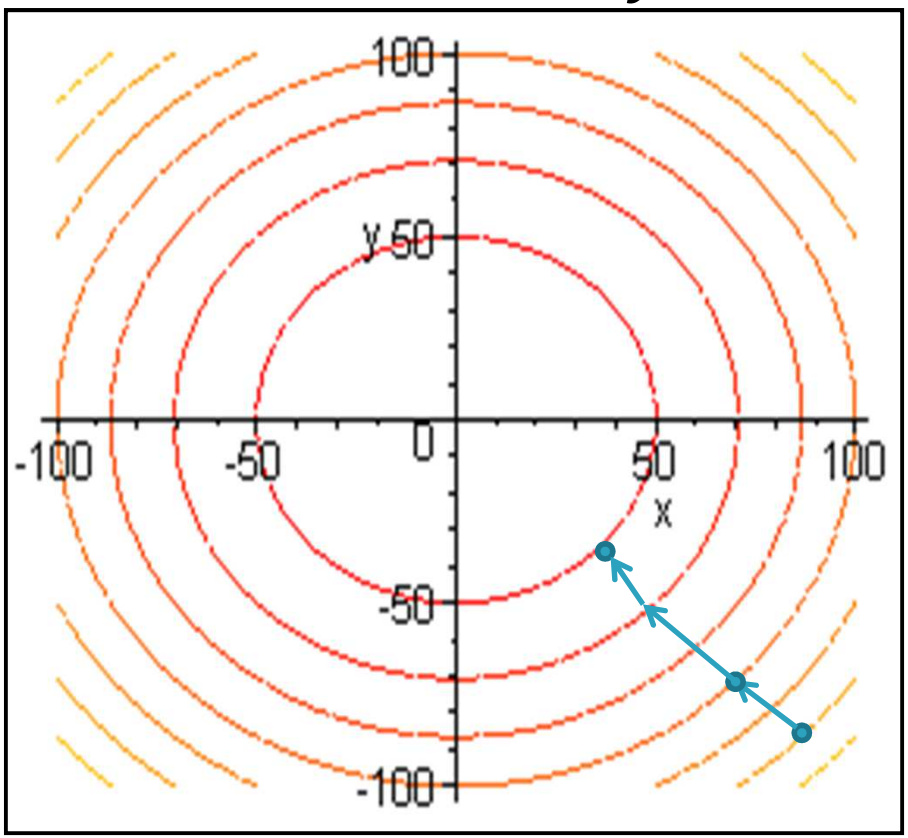
Gradient descent. Linear model

```
clear all
x = [1 2 4 4 5 6 7 8]';
y = [1 4 3 5 7 6 6 8]';
A = [x ones(size(x))];

weights = [0.8 1]';
P = weights;

m = -1:0.05:2;
b = -1:0.05:3;
[bgrid,mgrid] = meshgrid(b,m);
for i=1:length(m),
   for j=1:length(b),
      weights = [m(i) b(j)]';
      estim = A * weights;
      error(j,i) = log(sumsqr(y - estim));
   end
end
```

```
while(1),
    close all
    pcolor(m,b,error),shading interp,colorbar,hold on
    xlabel('b','FontSize',12)
    ylabel('m','FontSize',12)
    weights=ginput(1)';
    P=weights;
    for i=1:1000,
        estim = A * weights;
        aux = repmat(estim-y,1,size(A,2));
        gradiente = sum(aux.* A);
        weights = weights - 0.001 * gradiente';
        P = [P weights];
    end;
    comet(P(1,:),P(2,:));
    pause
end;
```

Well-conditioned problem

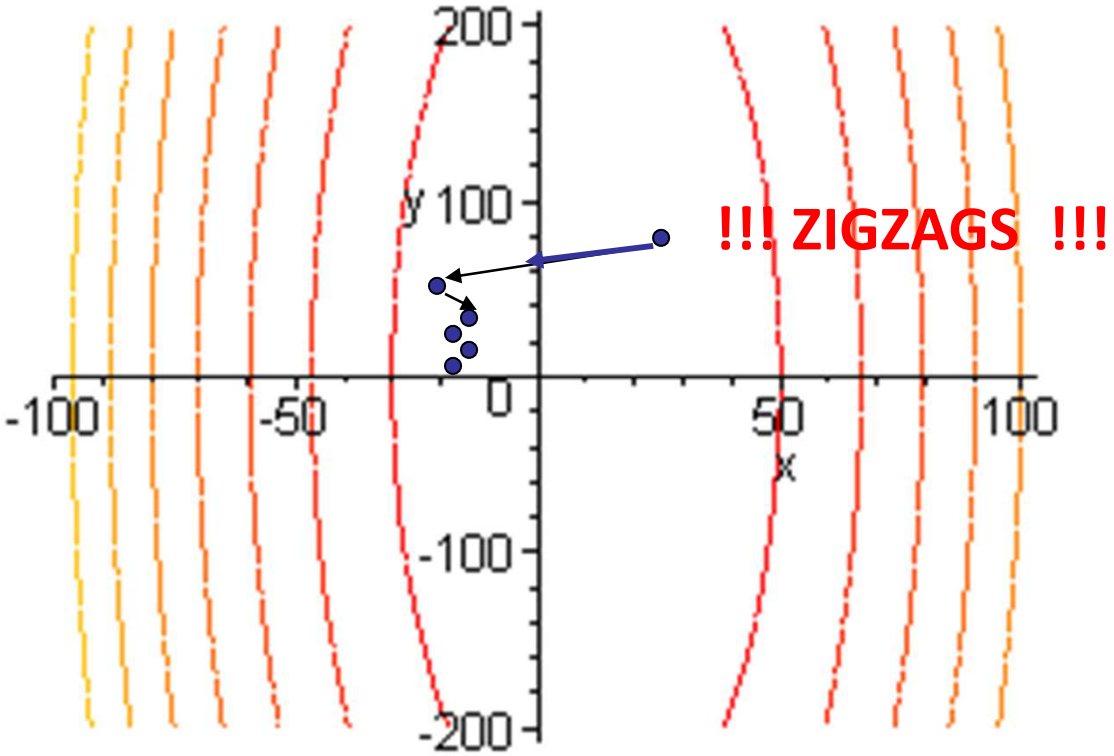
$$E = x^2 + y^2$$



- E is defined in the plane and its graph has a bowl shape
- Red curves are contour lines, that is, the regions in which the value of E is constant
- The blue arrow that originates at a point shows the direction of negative gradient at that point.
- The (negative) gradient at a point is orthogonal to the contour line passing through that point.
- The gradient descent takes us to the point where the value of the function E is minimal.

Ill-conditioned problem

$$50(x-10)^2+y^2$$



Limitations

- For some examples, gradient descent is relatively slow close to the minimum
- Its asymptotic rate of convergence is inferior to many other methods
- For poorly conditioned convex problems, gradient descent increasingly 'zigzags' as the gradients point nearly orthogonally to the shortest direction to a minimum point

Newton method

- It is a method for finding successively better approximations to the roots (or zeroes) of a real-valued function

- Newton method can be used to find a minimum or maximum of a function

- Newton method is an extremely powerful technique—in general the convergence is quadratic: as the method converges on the root, the difference between the root and the approximation is squared at each step

Newton method

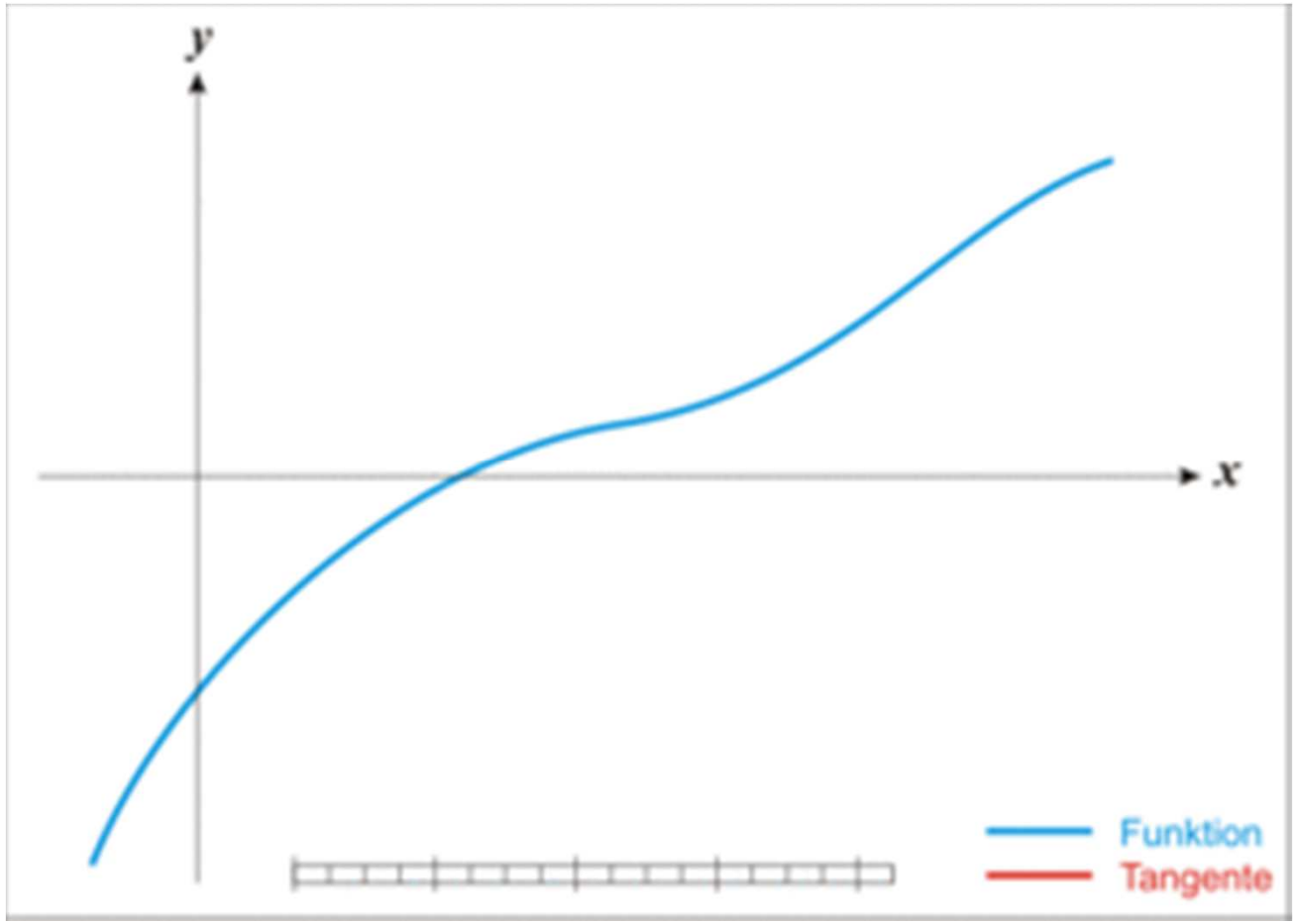
1. A large error in the initial estimate can contribute to non-convergence of the algorithm. So, we have to select a x_0 value close to zero

2. The tangent line at x_0 point is calculated

3. Computes the x-intercept of this tangent line. This x-intercept will typically be a better approximation to the function root than the original guess

4. Return to step 2

Newton method



Newton method

```
[x,y]=meshgrid(-3:0.1:3,-4:0.1:4);
z = (x.^4 - 3*x.^2 + x + y.*y + x.*y);

while(1),
    close all
    pcolor(x,y,z),shading flat;hold on;
    sol = ginput(1)';
    for i=1:100,
       x=sol(1,i);
       y=sol(2,i);
       dx = 4 * x.^3 - 6*x + 1 + y;
       dy = 2 * y + x;
       dxx = 12 * x.^2 - 6;          dxy = 1;
       dyx = 1;                      dyy = 2;
       H = [dxx dxy;dyx dyy];
       sol(:,i+1) = sol(:,i) - 0.1 * inv(H)  * ([dx;dy]);
       plot([sol(1,i) sol(1,i+1)],[sol(2,i) sol(2,i+1)],'r')
       pause
    end
end
```