Linealización de función de 2 variables: ejemplo Matlab (Symbolic Toolbox)

Presentaciones en vídeo:

http://personales.upv.es/asala/YT/V/linmiso1.html

http://personales.upv.es/asala/YT/V/linmiso2.html

*Este código ejecutó sin errores en Matlab **R2022b** (Linux)

Objetivo: ilustrar el concepto de linealización, plano tangente, aproximación, en una función $y = f(x_1, x_2)$.

Tabla de Contenidos

Definición de función y punto de operación

```
syms x_1  x_2  y     real
```

Definamos $y = f(x_1, x_2)$ como:

```
f(x_1,x_2)= 0.5*x_1^2 + x_2^2 + cos(x_1+exp(-x_2))
```

```
f(x_1, x_2) =
```

$$\cos\left(x_1 + \mathrm{e}^{-x_2}\right) + \frac{{x_1}^2}{2} + {x_2}^2$$

Elegiremos punto de tangencia (punto de operación) en el espacio de entrada (x_1, x_2):

```
x_pf=[0.85; 0.5]; %valor arbitrario, al menos en este contexto (en física/control es pt
```

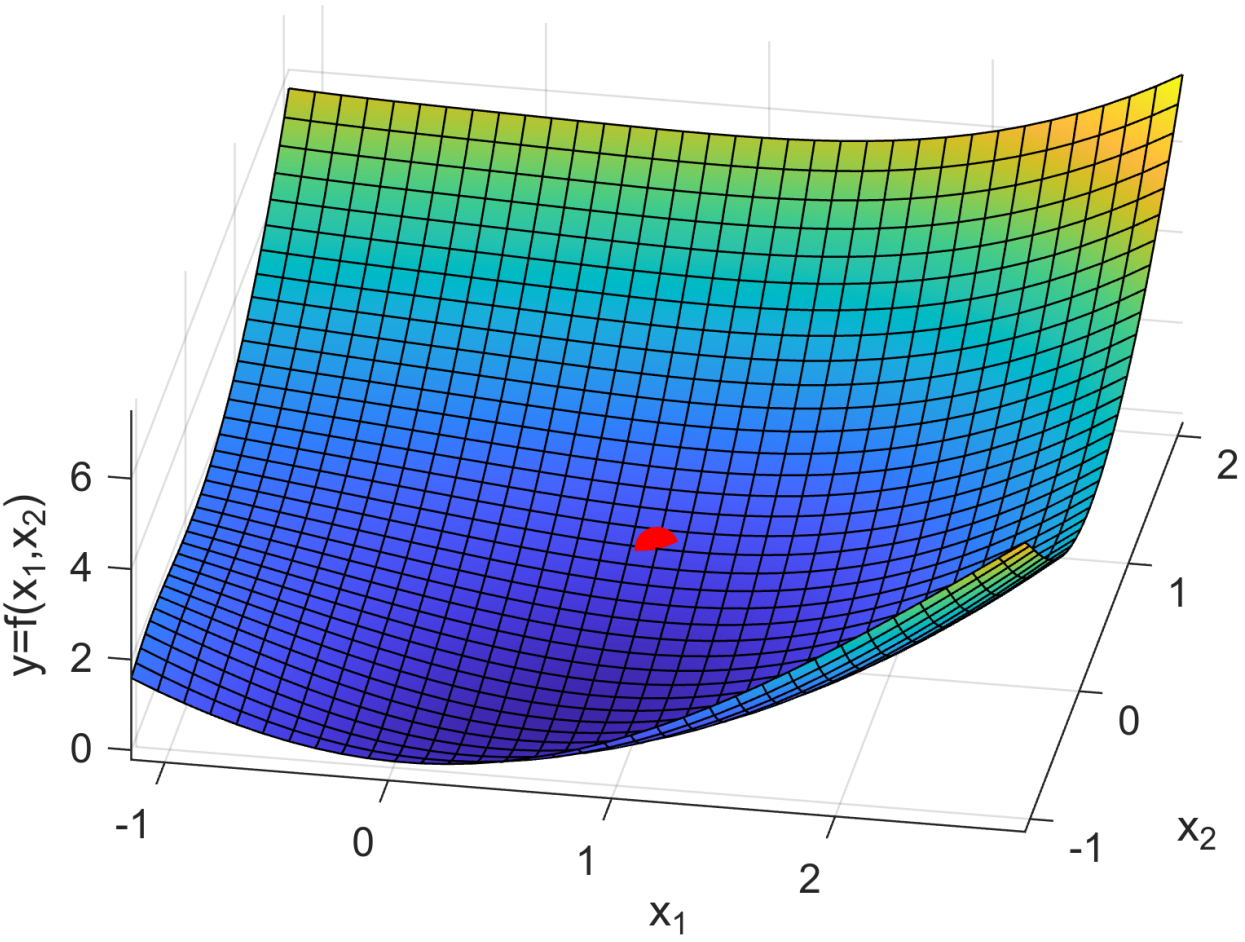
El valor de la función en ese punto es, por definición el "punto de operación de y". Su valor es:

```
y_pf=f( x_pf(1), x_pf(2) ); %simbólico
y_pf_num=eval(y_pf) %numérico
```

```
y_pf_num = 0.7253
```

Vamos a ver una gráfica (3D) de la función y el punto de operación:

```
rng1=[-2 2]+x_pf(1); %rango x1
rng2=[-1.6 1.6]+x_pf(2); %rango x2
fsurf(f,[rng1 rng2]) %superficie
hold on
plot3(x_pf(1),x_pf(2),y_pf_num,'or',LineWidth=5,MarkerSize=6) %Pto. func.
hold off
view([10 50]), xlabel("x_1"), ylabel("x_2"), zlabel("y=f(x_1,x_2)")
```



Modelo linealizado

A partir del desarrollo de Taylor alrededor de x_{pf}, definiendo la notación incremental $\Delta x = x - x_{pf}$:

$$\underbrace{f(x_{pf}) + \underbrace{\sum_{i=1}^{2} \frac{\partial f}{\partial x_i}_{|x_{pf}} \cdot \Delta x_i}_{\text{modelo linealizado incr.}}}_{\text{modelo linealizado abs. (afín)}} + \frac{1}{2} \cdot \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{\partial^2 f}{\partial x_i \partial x_j}_{|x_{pf}} \cdot \Delta x_i \Delta x_j + \frac{1}{3!} \cdot \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2} \frac{\partial^3 f}{\partial x_i \partial x_j \partial x_k}_{|x_{pf}} \cdot \Delta x_i \Delta x_j \Delta x_k$$

Hasta los términos de grado 1 y grado 2 en Δx, como son muy usados, suele expresarse con unas definiciones matriciales:

$$f(x) = \underbrace{f(x_{pf}) + \underbrace{\frac{df}{dx}_{|x_{pf}} \cdot \Delta x}_{\text{modelo linealizado incr.}}}_{\text{modelo linealizado abs. (afín)}} + \Delta x^T \cdot \frac{1}{2}\frac{d^2 f}{dx^2}_{|x_{pf}} \cdot \Delta x + \ldots$$

siendo $\Delta x := x - x_{pf}$ (vector 2×1), y donde, abusando la notacion, se han definido:

$J(x) := \frac{df}{dx} := \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2}\right]$ es la matriz **JACOBIANA** (1×2) , o "***gradiente***", y

$H(x) := \frac{d^2 f}{dx^2} := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$ es la matriz **HESSIANA** (2×2).

Escogiendo hasta primeras derivadas obtendremos el modelo linealizado:

```
jacob_f=jacobian(f) %matriz de primeras derivadas parciales
```

```
jacob_f(x_1, x_2) =
```

$$\left(x_1 - \sin\left(x_1 + e^{-x_2}\right) \quad 2\,x_2 + e^{-x_2}\sin\left(x_1 + e^{-x_2}\right)\right)$$

Por supuesto, coincide con las derivadas parciales primeras, organizadas en el sitio adecuado:

```
[diff(f,x_1) diff(f,x_2)] %jacobiano "a mano"
```

```
ans(x_1, x_2) =
```

$$\left(x_1 - \sin\left(x_1 + e^{-x_2}\right) \quad 2\,x_2 + e^{-x_2}\sin\left(x_1 + e^{-x_2}\right)\right)$$

Las derivadas parciales en el punto de funcionamiento es lo que hay que calcular al linealizar:

```
J=jacob_f(x_pf(1),x_pf(2)); %derivada en el punto de funcionamiento, pendiente de la re
jac_num=eval(J) %jacobiano numérico
```

```
jac_num = 1×2
   -0.1435    1.6026
```

El modelo incremental es $\Delta y = \frac{df}{dx}_{|x_{pf}} \cdot \Delta x$, y es con esta notación como suele ser utilizado en aplicaciones.

Obviamente, lo podemos escribir como la ecuación de un PLANO tangente:

```
x=[x_1; x_2]
```

```
x =
```

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

```
AproxLinV1=  ...
   (y-y_pf)  ==  J(1) * (x(1)-x_pf(1))  +  J(2) * (x(2)-x_pf(2))  ;
vpa(AproxLinV1,4) %veamos esto con números decimales (4 cifras significativas)
```

ans = $y - 0.7253 = 1.603\,x_2 - 0.1435\,x_1 - 0.6793$

Utilizando la matriz jacobiana, la expresión se parece más a la fórmula 1D:

```
AproxLin=   (y-y_pf) == J*(x-x_pf);
vpa(AproxLin,4) %veamos esto con números decimales (4 cifras significativas)
```

ans = $y - 0.7253 = 1.603\,x_2 - 0.1435\,x_1 - 0.6793$

Es una ecuación "implícita"... Escribiendo "explícitamente" y en función de x, esto es:

$y_{aprox} = f(x_{pf}) + \frac{df}{dx}\Delta x$ resulta:

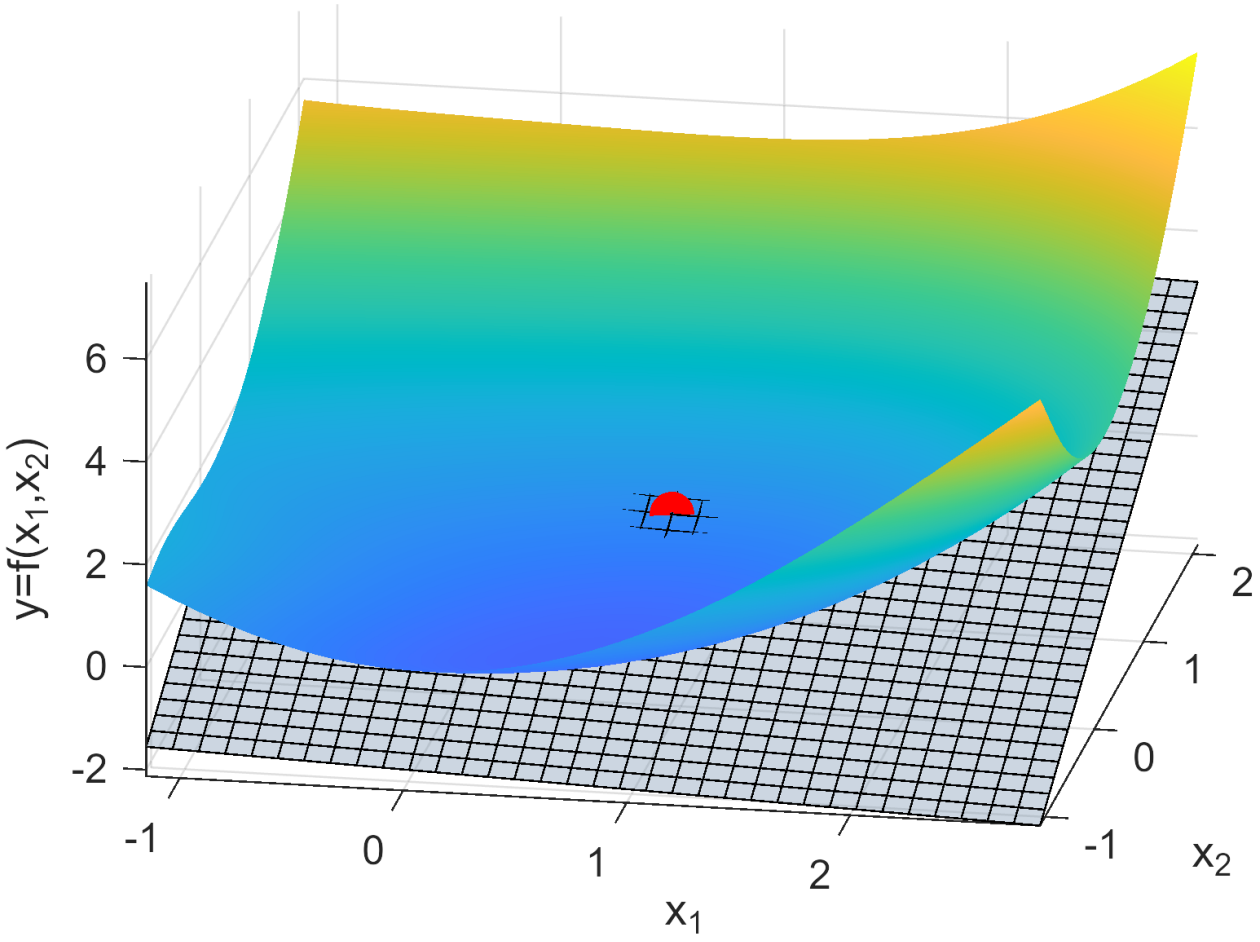
```
AproxLin2= solve(AproxLin,y); %despeja "y"
y_aprox_linealizada=vpa(AproxLin2,4)  %con números decimales
```

y_aprox_linealizada = $1.603\,x_2 - 0.1435\,x_1 + 0.04594$

*Las coordenadas x_1 y x_2 son ABSOLUTAS en esa expresión.

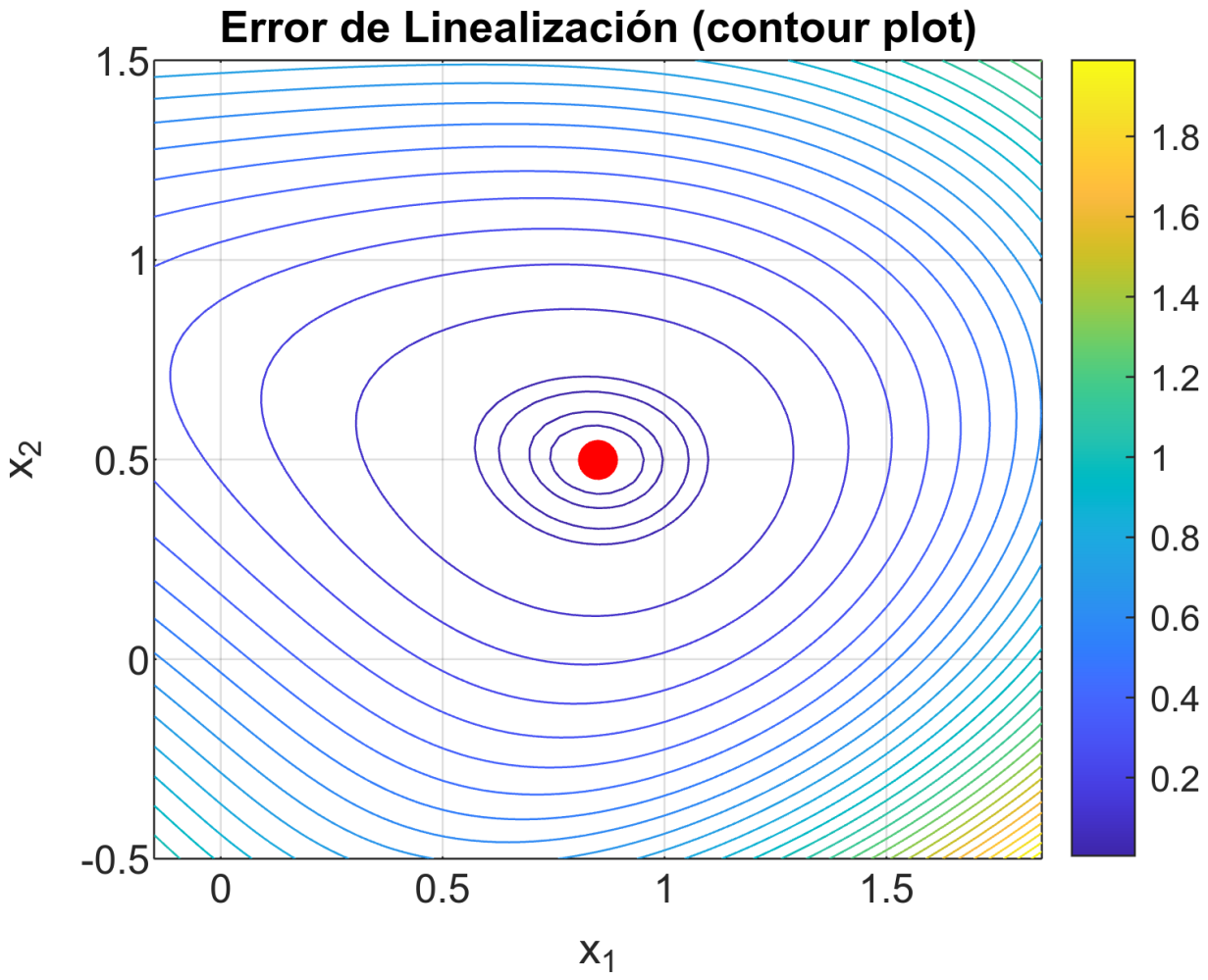
Representación/comparación gráfica con modelo original

```
rng1=[-2 2]+x_pf(1); %rango x1 en gráfica
rng2=[-1.6 1.6]+x_pf(2); %rango x2 en gráfica
fsurf(f,[rng1 rng2],LineStyle="none") %la superficie
hold on
fsurf(AproxLin2,[rng1 rng2],FaceAlpha=0.4,FaceColor=[.5 .6 .7]) %plano
plot3(x_pf(1),x_pf(2),y_pf_num,'or',LineWidth=5,MarkerSize=6) %bola marca pto. func.
hold off
view([10 30]), xlabel("x_1"), ylabel("x_2"), zlabel("y=f(x_1,x_2)")
```



¿Cuál es el error de linealización?

```
fc=fcontour((f-AproxLin2),[[-1 1]+x_pf(1) [-1 1]+x_pf(2)]);
fc.LevelList=[ 0.005 0.01 0.02 0.03:0.07:6];
grid on
colorbar, title("Error de Linealización (contour plot)")
hold on
plot(x_pf(1),x_pf(2),'or',LineWidth=5,MarkerSize=5)
hold off
xlabel("x_1"), ylabel("x_2")
```



Se parece a un paraboloide/hiperboloide cerca del punto de linealización (curvas de nivel elipses/hipérbolas según valores propios del hessiano).

En este caso, elipses porque:

```
H=hessian(f);
Hnum=eval(H(x_pf(1),x_pf(2)))
```

```
Hnum = 2×2
    0.8860    0.0692
    0.0692    1.3555
```

```
eig(Hnum)
```

```
ans = 2×1
    0.8760
    1.3655
```

Comprobación numérica de la linealización

Para aproximar $f(x_{pf} + \Delta x)$ calcularíamos la salida del modelo linealizado como:

```
incx=[-0.04; 0.03];
incyaprox=jac_num*incx
```

```
incyaprox = 0.0538
```

```
yaprox=y_pf_num+incyaprox %aproximación linealizada
```

```
yaprox = 0.7791
```

Comparemos con la función original:

```
newx=x_pf+incx
```

```
newx = 2×1
    0.8100
    0.5300
```

```
y_correcta=eval(f(newx(1),newx(2))) %valor numérico exacto
```

```
y_correcta = 0.7803
```

```
error_linealizar=y_correcta-yaprox %error de linealización
```

```
error_linealizar = 0.0012
```

```
incy_correcta=eval(y_correcta-y_pf)
```

```
incy_correcta = 0.0550
```