# Linearization of a function of 2 variables: Matlab example (Symbolic Toolbox)

Presentations in video:

http://personales.upv.es/asala/YT/V/linmiso1EN.html

http://personales.upv.es/asala/YT/V/linmiso2EN.html



*The code below ran without errors in Matlab **R2022b** (Linux)

**Objective:** illustrate the concept of linearization, tangent plane, approximation, in a function $y = f(x_1, x_2)$.

**Table of Contents**



## Function and operating point definitions

```
syms x_1  x_2  y     real
```

Let us define $y = f(x_1, x_2)$ as:

```
f(x_1,x_2)= 0.5*x_1^2 + x_2^2 + cos(1.1*x_1+exp(-x_2))+0.28
```

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

$$\cos\left(\frac{11\,x_1}{10} + \mathrm{e}^{-x_2}\right) + \frac{{x_1}^2}{2} + {x_2}^2 + \frac{7}{25}$$

We will choose point of tangency (operation point) in input space $(x_1, x_2)$:

```
x_op=[0.85; 0.5]; %arbitrary value here (in control/physics, say, this would be an equi
```

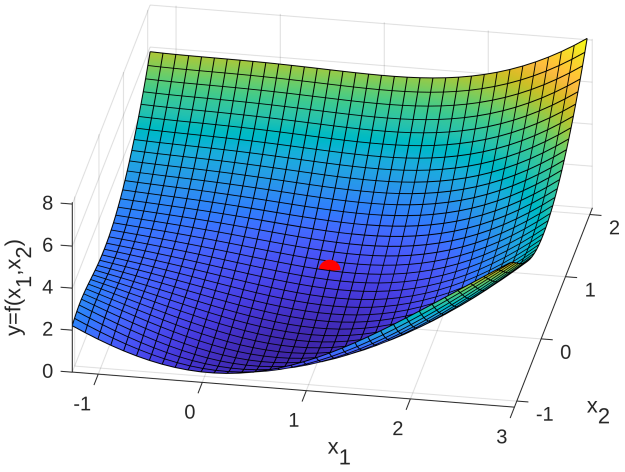
The value of the function at that point is by definition the "operating point of $y$". Its value is

```
y_op=f( x_op(1), x_op(2) ); %symbolic expression
y_op_num=eval(y_op) %numeric
```

```
y_op_num = 0.9205
```

Let's see a 3D plot of the function and the operating point:

```
rng1=[-2.1 2.1]+x_op(1); %range x1
rng2=[-1.6 1.6]+x_op(2); %range x2
fsurf(f,[rng1 rng2]) %surface plot
hold on
plot3(x_op(1),x_op(2),y_op_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)")
```




## Linearised model

From Taylor series around $x_{op}$, using notation $\Delta x := x - x_{op}$, we get:

$$= \underbrace{f(x_{op}) + \underbrace{\sum_{i=1}^{2} \frac{\partial f}{\partial x_i}_{|x_{op}} \Delta x_i}_{\text{linearised model (incr.)}}}_{\text{linearised model (abs.; affine)}} + \frac{1}{2} \cdot \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{\partial^2 f}{\partial x_i \partial x_j}_{|x_{op}} \Delta x_i \Delta x_j + \frac{1}{3!} \cdot \sum_{i=1}^{2} \sum_{j=1}^{2} \sum_{k=1}^{2} \frac{\partial^3 f}{\partial x_i \partial x_j \partial x_k}_{|x_{op}} \Delta x_i \Delta x_j \Delta x_k \quad +$$

The grade 1 and grade 2 terms in $\Delta x$, widely used, are usually expressed with matrix definitions:

$$f(x) = \underbrace{f(x_{op}) + \underbrace{\frac{df}{dx}_{|x_{op}} \cdot \Delta x}_{\text{linearised model incr.}}}_{\text{linearised model abs. (affine)}} + \Delta x^T \cdot \frac{1}{2}\frac{d^2 f}{dx^2}_{|x_{op}} \cdot \Delta x + \ldots$$

being $\Delta x := x - x_{op}$ ( $2 \times 1$ vector), where, abusing the notation, we used:

$J(x) := \frac{df}{dx} := \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix}$ is the **JACOBIAN matrix** ( $1 \times 2$ ) , or "***gradient***", and

$$H(x) = \frac{d^2 f}{dx^2} := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$ is the **HESSIAN** matrix ( $2 \times 2$ ).

Keeping up to first derivatives we will obtain the linearized model.

In our particular example function:

```
jacob_f=jacobian(f) %first partial derivatives arranged in matrix form
```

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

$$\left( x_1 - \frac{11 \sin\left(\frac{11 x_1}{10} + e^{-x_2}\right)}{10} \quad 2 x_2 + e^{-x_2} \sin\left(\frac{11 x_1}{10} + e^{-x_2}\right) \right)$$

Of course, it matches the first partial derivatives, suitably arranged:

```
[diff(f,x_1) diff(f,x_2)]
```

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

$$\left( x_1 - \frac{11 \sin\left(\frac{11 x_1}{10} + e^{-x_2}\right)}{10} \quad 2 x_2 + e^{-x_2} \sin\left(\frac{11 x_1}{10} + e^{-x_2}\right) \right)$$

The numerical value of partial derivatives at the operating point are what you have to calculate when linearizing:

```
J=jacob_f(x_op(1),x_op(2)); %derivatives at operating point (symbolic expression)
```

```
jac_num=eval(J) %numerical evaluation
```

```
jac_num = 1×2
   -0.2495    1.6063
```

The incremental model is $\Delta y = \frac{df}{dx}_{|x_{op}} \cdot \Delta x$, and it is with this notation that it is usually used in applications, specially control ones.

Obviously, we can write it as the equation of a tangent PLANE:

```
x=[x_1; x_2] %symbolic vector arranging the variables in column form
```

```
x =
```

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

```
ApproxLinV1=  ...
   (y-y_op)  ==  J(1) * (x(1)-x_op(1))  +  J(2) * (x(2)-x_op(2))  ;
vpa(ApproxLinV1,4) %let's see it in "pretty" numerical format
```

```
ans =
```

$$y - 0.9205 = 1.606\,x_2 - 0.2495\,x_1 - 0.591$$

By using the Jacobian matrix, the expression is more like the 1D formula:

```
ApproxLin=   (y-y_op) == J*(x-x_op);
vpa(ApproxLin,4) %let's see it in "pretty" numerical format
```

```
ans =
```

$$y - 0.9205 = 1.606\,x_2 - 0.2495\,x_1 - 0.591$$

This is an "implicit" expression relating $(y, x_1, x_2)$... Solving for $y$ "explicitly" as a function of $x$, i.e.: $y_{aprox} = f(x_{op}) + \frac{df}{dx}_{|x_{op}} \Delta x$ results in:

```
ApproxLin2= solve(ApproxLin,y); %despeja "y"
y_approx_linear=vpa(ApproxLin2,4)  %con números decimales
```

```
y_approx_linear =
```

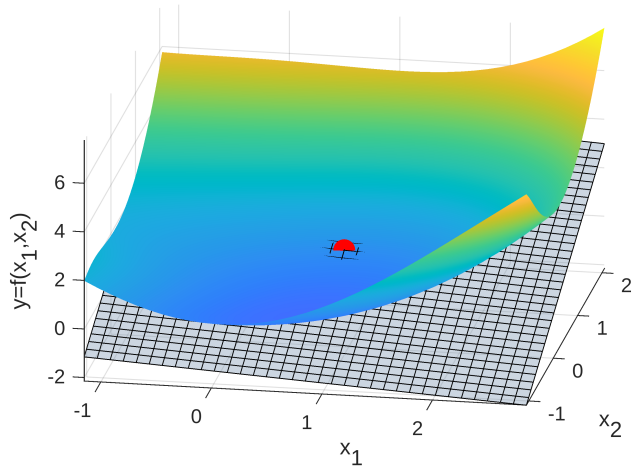
$$1.606\,x_2 - 0.2495\,x_1 + 0.3295$$

*The $x_1$ y $x_2$ variables are in ABSOLUTE coordinates above.

### Graphical representation and comparison with original model

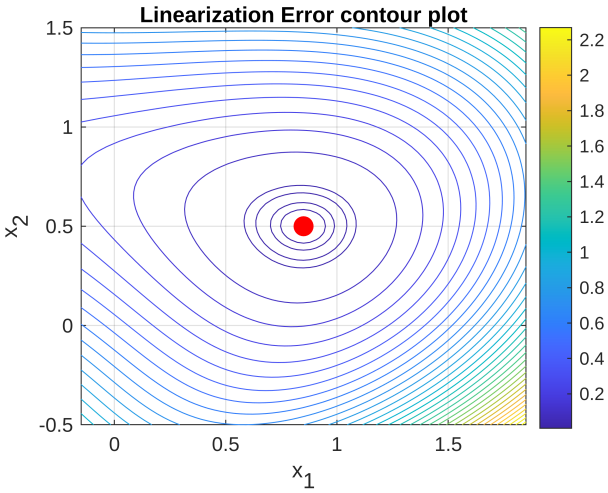
```
rng1=[-2 2]+x_op(1); %range x1 for plots
rng2=[-1.6 1.6]+x_op(2); %range x2 for plots
fsurf(f,[rng1 rng2],LineStyle="none") %surface plot original f
```

```
hold on
fsurf(ApproxLin2,[rng1 rng2],FaceAlpha=0.4,FaceColor=[.5 .6 .7]) %plane
plot3(x_op(1),x_op(2),y_op_num,'or',LineWidth=5,MarkerSize=6) %operating point, ball
hold off
view([10 30]), xlabel("x_1"), ylabel("x_2"), zlabel("y=f(x_1,x_2)")
```




Which is the linearization error?

```
fc=fcontour((f-ApproxLin2),[[-1 1]+x_op(1) [-1 1]+x_op(2)]);
fc.LevelList=[ 0.005 0.01 0.02 0.03:0.07:6];
grid on
colorbar, title("Linearization Error contour plot")
hold on
plot(x_op(1),x_op(2),'or',LineWidth=5,MarkerSize=5)
hold off
xlabel("x_1"), ylabel("x_2")
```




It looks like a paraboloid/hyperboloid near the linearization point (contour lines ellipses/ hyperbolas according to Hessian eigenvalues).

In this case ellipses, because:

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

```
Hnum = 2×2
    0.9646    0.0195
    0.0195    1.3830
```

```
eig(Hnum)
```

```
ans = 2×1
    0.9637
    1.3839
```

## Numerical check of linearization accuracy

To approximate $f(x_{op} + \Delta x)$ we would compute the linearised model output as:

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

```
incy_approx = 0.0582
```

```
yapprox=y_op_num+incy_approx %linearised approximation value
```

```
yapprox = 0.9787
```

Let's compare with the original function:

```
newx=x_op+incx
```

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

```
y_correct=eval(f(newx(1),newx(2))) %exact numerical value
```

```
y_correct = 0.9800
```

```
error_linearize=y_correct-yapprox %linearization error
```

```
error_linearize = 0.0013
```

```
incy_correct=eval(y_correct-y_op)
```

```
incy_correct = 0.0595
```