

---

# Mathematical Methods and Modeling Laboratory class

---

## Graphs and Optimization in MATLAB

---

## Topics

---

1. Graph of a scalar function
2. Graph of a function of 2 variables
3. Level Curves and Gradient Field
4. Functions and handles
5. Unconstrained Optimization
6. Constrained Optimization

---

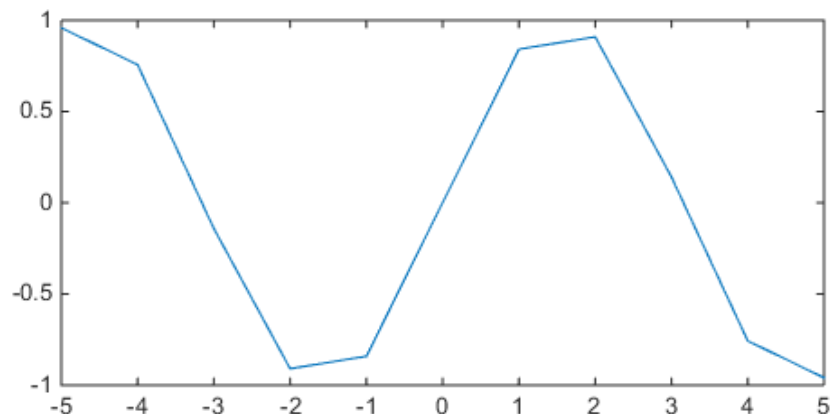
## Graph of a scalar function

---

- The task of drawing the graph of a function in MATLAB is not as simple as inserting its expression
- First, we have to decide in what range we draw the function, and the discretized points to draw.
- For those points we have to construct the **abscissa vector** and its corresponding **ordinate vector**
- Then using the **plot** function MATLAB draws the piece-wise line interpolating the discretized points
- Eg.: *draw the sine function in  $[-5,5]$  with 11 points:*

```
x = [-5:1:5];  
y = sin(x);  
plot(x,y);
```

The graph is very rough!  
→ try using more points



---

## Graph of a scalar function

---

- An additional string argument of `plot` function can be used to specify the style of the curve to be drawn:

```
plot(x,y,'-- * r')
```

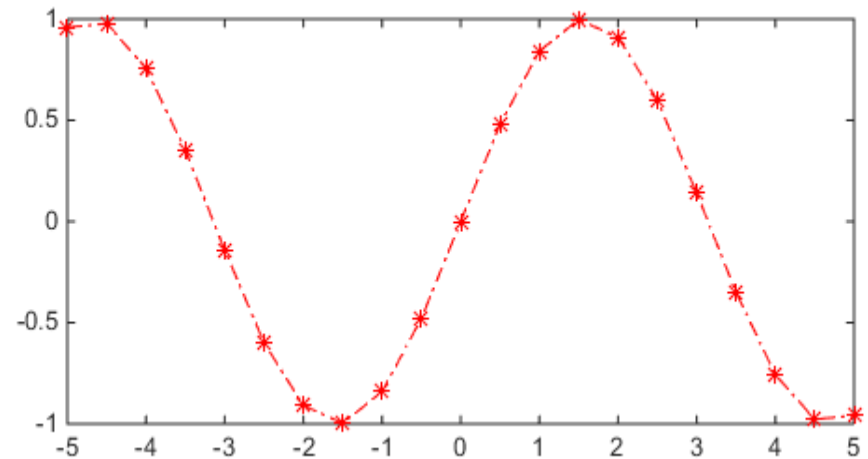
draws a red dashed-line curve with asterisks in place of points.

- Optional arguments can be supplied to `plot` function, by specifying their name-value pair. The syntax is:

```
plot(x,y,'name1', val1, 'name2', val2, ...)
```

E.g.: `plot(x,y,'LineWidth',3,'MarkerSize',5)` draws a 3 points width line, with points marker of size 5.

- See [doc LineSpec](#) for more information.



---

## Graph of a function of 2 variables

---

- In order to draw a function of 2 variables we have to decide the rectangular region  $[a,b] \times [c,d]$  in which we want the graph (surface).  $[a,b] \times [c,d]$  has to be discretized regularly into points  $(x,y)$ , thus producing a so-called grid (or lattice).
- Similarly to the previous 1-dimensional case we have to define 2 vectors containing the possible (discretized) values for the abscissa and ordinate, respectively.
- Then using `meshgrid` function, MATLAB produces 2 matrices  $X, Y$  whose elements are in one-to-one correspondence to the points of the grid:  $X(i,j)$  is the abscissa value for the grid point  $(i,j)$ , and analogously for  $Y$ .
- In terms of MATLAB instructions, for example:

```
x = [-pi:0.1:pi];           %a=-π, b=π
y = [-2:0.1:2];           %c=-2, d=2
[X,Y] = meshgrid(x,y);    %notice X,Y have the same size
```

---

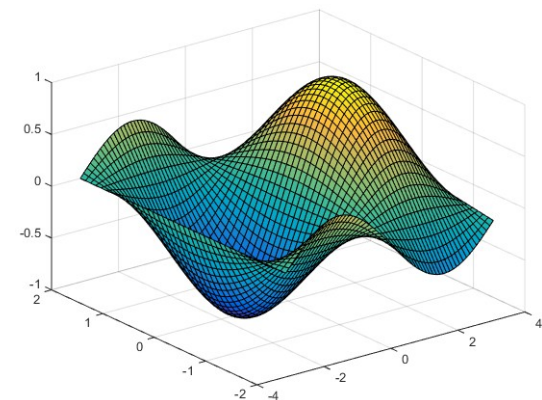
## Graph of a function of 2 variables

---

- At this step it is necessary to construct the matrix **Z** containing the z-values for the function evaluated at the discretized points (x,y), using the **element-wise** operations on X, Y to form the desired expression of the function
- Finally, draw the graph using MATLAB's **surf** function  
E.g.: *draw the function  $f(x,y) = \sin(x) \cos(y)$  after constructing the previous grid.*

```
Z = sin(X) .* cos(Y);  
surf(X,Y,Z);
```

- Optional arguments can be given to **surf** function in order to adjust some graphical details; see: **doc surf**
- In the figure window that appears, try using the “Rotate 3D” button in the toolbar to move the viewpoint of the surface



---

## Level Curves and Gradient Field

---

- Level sets of a function of two variables are also called level curves or contour lines
- Basically, MATLAB's `contour` function draws some level curves for a function  $z = f(x,y)$ . The base syntax is:  
`contour(X,Y,Z)`  
with X, Y, Z built through previous discretization method.
- To draw a specific collection of level curves, just give as additional argument an **increasing vector** `[z1, z2, ..., zm]`, that specifies we want level curves at z-values  $z_1, z_2, \dots, z_m$
- Eg. (continued): draw the level curves at z-values 0.7, -0.8  
`close all` %closes any open figure window  
`contour(X,Y,Z, [-0.8, 0.7], 'LineWidth', 3)`  
`xlabel('x axis')` %add the name next to the x-axis  
`ylabel('y axis')` %add the name next to the y-axis  
Note: the `LineWidth` option set to 3pt draws a thicker curve

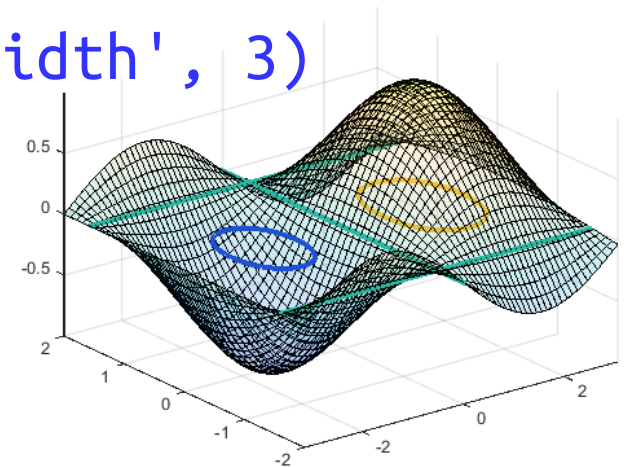
---

## Level Curves

---

- We are interested in drawing the surface and the level curves in the same graph → use `hold on`, that tells MATLAB to keep previous drawings in the same figure
- Unluckily, in a MATLAB figure one graphical object (eg. surface) can visually hide another object (eg. contour lines) → set the transparency through `FaceAlpha` option
- Eg.: draw  $f(x,y)=\sin(x)\cos(y)$  and its contour lines at z-values -0.8, 0 and 0.7.

```
surf(X,Y,Z, 'FaceAlpha', 0.2) %0=transparent,1=fill  
hold on  
contour(X,Y,Z, [-0.8,0,0.7], 'LineWidth', 3)
```





---

## Level Curves

---

- Note: the level curves are drawn in the  $z=0$  plane of the 3D space  $\mathbf{R}^3$
- Clearly level curves of other functions can be drawn in the same figure of  $f(x,y)$ . E.g. those of some possible constraint functions

---

## Gradient Field

---

- When the gradient  $\nabla f$  exists at a point  $(x,y)$ , it corresponds to the **direction of maximal increment** from  $(x,y)$ ; when it is  $\nabla f(x,y)=(0,0)$  we know that  $(x,y)$  is a stationary point.
- Therefore a figure in the  $xy$ -plane containing arrows proportional to  $\nabla f(x,y)$  can be helpful in visually localizing the stationary points, and hence candidate minimum/maximum points → such a graph is called **Gradient Field**
- In MATLAB a gradient field can be drawn by the **quiver** function, after constructing the **discrete gradient** (i.e. finite differences) of  $f$ :
  - 1) **gradient** function takes  $Z$  matrix and **the 2 discretization steps** (0.1 and 0.1 in our example)
  - 2) **quiver** takes  $X$ ,  $Y$  and  $DX$ ,  $DY$  matrices given by **gradient**

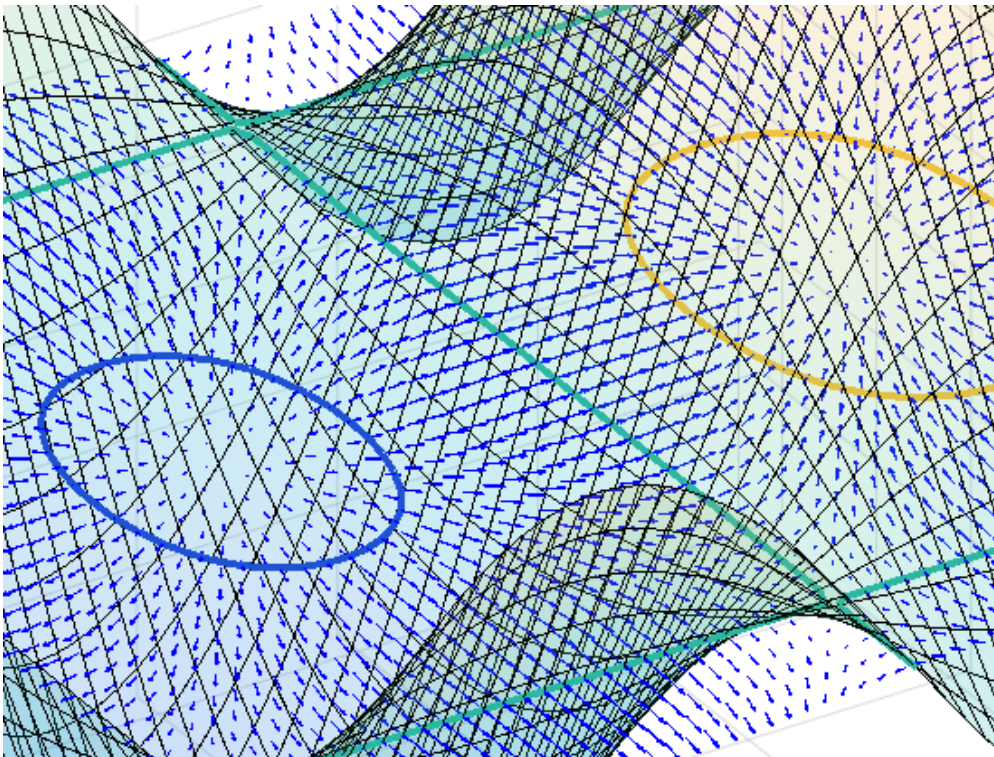
---

## Gradient Field

---

```
[DX, DY] = gradient(Z, 0.1, 0.1)
quiver(X, Y, DX, DY, 'LineWidth', 1)
```

After little rotation and zoom (using the buttons on the toolbar) we can see a figure like:



Note: gradient arrows (in blue) are **orthogonal** to the level curves

See [doc gradient](#) and [doc quiver](#) for more information

---

## Functions and handles

---

- MATLAB's optimizer needs both objective function and, possibly, constraint functions to be defined suitably
- There are mainly two ways to define a function in MATLAB:
  - 1) **anonymous** functions defined *inline*, i.e. as instruction in the code flow (actually these functions can have a name)
  - 2) common functions defined through **M-files (\*.m)**
- Syntax for an anonymous function is:
$$\text{@(x) expr(x)}$$
  - $x$  is the argument; it can be scalar, vector or matrix
  - $\text{expr}(x)$  is any admissible expression in MATLAB syntax, possibly accessing  $x$ 's elements
  - return value is the evaluation of the expression on  $x$
- Eg.: define an anonymous function for the magnitude (length) of a vector, and name it `mag`
$$\text{mag} = \text{@(x) sqrt(sum( x .^ 2))};$$
- Assigned variable (`mag`) is a function **handle** (reference). It can be used like normal functions, eg.: `mag([-2 4 3 0])`

---

## Functions and handles

---

- A common function is defined by means of a M-file with extension **.m**
- The syntax of such a file is:

```
function [out1, out2, ...] = fname(arg1, arg2, ...)
    ...
    % COMPUTATIONS POSSIBLY WITH arg1, arg2, ...
    ...
    out1 = ...
    out2 = ...
end
```
- (optional) input and output arguments **arg1, arg2, ..., out1, out2, ...** can be scalars, vectors or matrices
- an output argument has to be assigned before end
- **fname** can be a customary name for the function; it is recommended not to redefine a MATLAB built-in function
- Before using, save it to a file with **same name: fname.m**

---

## Functions and handles

---

- The usage is the same as with built-in functions:  
`[eval1, eval2, ...] = fname(val1, val2, ...)`
- Eg.: define the standard 2-dimensional gaussian function  

```
function value = gauss(x)
    num = exp(-(x(1)^2+x(2)^2) /2);
    den = 2*pi;
    value = num/den;
end
```
- After saving it to a file named gauss.m (**in the working directory**), we can calculate the 2-dimensional gaussian function evaluated at  $x=0.7$ ,  $y=0.2$  through the instruction:  
`gauss([0.7 0.2]) %returns 0.1221`
- A **handle** (reference) to a function can be obtained by `@`, and can be used like the former function. Eg.:  

```
normal = @gauss;
normal([0.7, 0.2]) %returns 0.1221
```

---

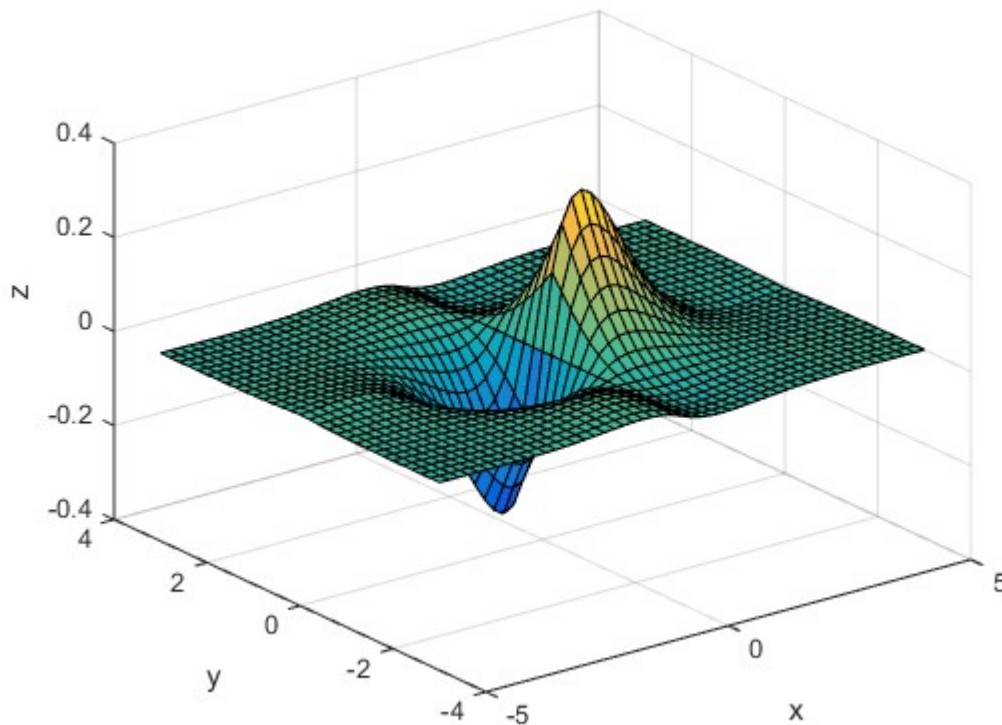
## Unconstrained Optimization

---

- Consider this unconstrained optimization problem in the form:

$$\min f(x, y) := \sin(x) \cos(y) e^{-|x|-|y|}$$

- First, it is useful to draw the qualitative graph of the function within a suitable region: just use the meshgrid discretization as done before



By rotating the surface through the Rotate 3D button it is easy to see that the minimum point is located in the region  $[-2, 0] \times [-3, 3]$  for instance

---

## Unconstrained Optimization

---

- The MATLAB function for doing unconstrained optimization is `fminunc`, which implements various numerical optimization algorithms that can be tuned with options
- 1. set the algorithm for `fminunc` with the `optimoptions` function:

```
opts = optimoptions(@fminunc, 'Algorithm', 'quasi-newton')
```

2. define the **objective function** through an M file

```
function out = f1(x)
    a = x(1);
    b = x(2);
    out = sin(a)*cos(b)*exp(-abs(a)-abs(b));
end
```

and save it with the same name of the function: `f1.m`

3. `fminunc` needs a point `x0` as **initial guess**: a trivial method is to look at the graph and choose a point sufficiently close to the minimum. E.g.: `x0 = [-0.8, 0.2];`



---

## Unconstrained Optimization

---

4. Call the function `fminunc` specifying also the handle to the objective function:

```
[xopt,fopt]=fminunc(@f1,x0,opts);
```

The calculated values are:

```
xopt =  
-0.7952  0.0000  
fopt =  
-0.3224
```

One can also check that `f1(xopt)` yields -0.3224 as expected

---

## Constrained Optimization

---

- The MATLAB function used for constrained optimization problems is `fmincon`.
- It implements (among others) the SQP (sequential quadratic programming) algorithm. We have to set it through the usual `optimoptions` function:

```
opts = optimoptions(@fmincon,'Algorithm','sqp')
```

- MATLAB assumes the following form for a constrained problem:

$$\min_x f(x) \text{ s.t. } \begin{cases} c(x) \leq 0 & \leftarrow \text{non-linear inequality constraints} \\ ceq(x) = 0 & \leftarrow \text{non-linear equality constraints} \\ A \cdot x \leq b & \leftarrow \text{linear ineq. constraints: matrix A,} \\ & \text{vector b (x is vector variable)} \\ Aeq \cdot x = beq & \leftarrow \text{linear eq. constraints: matrix Aeq,} \\ & \text{vector beq} \\ lb \leq x \leq ub & \leftarrow \text{vectors lb, ub defining bounds} \end{cases}$$

---

## Constrained Optimization

---

- To specify the desired constraints just provide them to `fmincon`. The other information can be left **empty**, namely the matrix `[]`.
- Matrices `A`, `Aeq` and vectors `b`, `beq`, `lb`, `ub` can be defined suitably (according to the problem) and given as argument to `fmincon` **directly**
- **Non-linear constraints** can be specified by an ad-hoc function in an M file, whose definition must comply to this scheme: take a (possibly vector) argument `x`, and return `c` and `ceq`. Eg.:

```
function [c,ceq] = nlcon(x)
...
c = ... %expression possibly depending on x
ceq = ... %expression possibly depending on x
end
```

Remember to name the file `nlcon.m`

---

## Constrained Optimization

---

- Syntax of `fmincon` is

`xopt = fmincon(@f,x0,A,b,Aeq,beq,lb,ub,@nlcon,opts)`

- `@f`: handle to objective function `f` defined in a M-file
- `x0`: initial guess point
- `A,b,Aeq,beq,lb,ub`: matrices/vectors as in the model
- `@nlcon`: handle to the function returning `[c,ceq]`
- `opts`: optional parameters for the algorithm
- 
- Example: add the constraint  $x+y+2=0$  to previous example

$$\begin{aligned} \min f(x, y) &:= \sin(x) \cos(y) e^{-|x|-|y|} \\ \text{subject to } g(x) &:= x + y + 2 = 0 \end{aligned}$$

- First, define matrix `Aeq` and vector `b`:

```
Aeq = [1 1];  
beq = -2;
```

---

## Constrained Optimization

---

- Then assign an initial point guess and call `fmincon` (provided options are set):

```
opts = optimoptions(@fmincon,'Algorithm','sqp')
x0 = [-1, -1];
[xo,fo]=fmincon(@f1,x0,[],[],Aeq,beq,[],[],[],opts);
```

The outputs are:

```
x0 =
    -1.7854    -0.2146
fo =
    -0.1292
```

- REMARK: SQP is effective on convex problems and also works well on other moderately non-linear problems
- REMARK: we only used linear (eq.) constraints, hence no handle to M function defining non-linear constraints was provided

---

## Constrained Optimization

---

- Now, we study the convex problem:

$$\min f(x, y) := 2x^2 + 4y^4 + x + 3$$

$$\text{subject to } g(x) := x^2 + \frac{y^2}{2} - 2 \leq 0$$

- This time we have to specify a non-linear inequality constraint through the file **g2.m**:

```
function [c,ceq] = g2(x)
    c = x(1)^2+x(2)^2/2-2;
    ceq = []; %empty matr. for no eq. constraint
end
```

- It can be seen that exact solution is  $x^* = (-1/4, 0)$ . For this reason, or also by plotting the graph, we can choose for instance the sufficiently close initial point  $x_0 = [-1, 1]$
- Then it is an easy task to define suitably the M-file **f2.m** for the function  $f(x,y)$

---

## Constrained Optimization

---

- Finally we can use the instruction:

```
[xo,fo] = fmincon(@f2,x0,[],[],[],[],[],[],@g2,opts)
```

- The results are:

xo =

-0.2500 -0.0015

fo =

2.8750

- REMARK: non-linear (equality and inequality) constraints have to be specified through an M-file function, whose **handle** is given into `fmincon` as argument
- For further documentation about the two main MATLAB functions for unconstrained and constrained optimization just look at [doc fminunc](#) and [doc fmincon](#)
- What if the domain of `f` is non rectangular?
- REMARK: drawing graph is an “independent” task