An Introduction to Matlab

Quantitative methods for Economics and Business

University of Ferrara

Academic year 2017-2018

An Introduction to Matlab

An Introduction to Matlab

An Introduction to Matlab

An Introduction to Matlab

An Introduction to Matlab

An Introduction to Matlab

An Introduction to Matlab

Part I

An Introduction to Matlab

What is MATLAB?

- Matlab is an interactive system for doing numerical computations.
- A numerical analyst called Cleve Moler wrote the first version of Matlab in the 1970s. Then it has evolved into a successful commercial software package.
- Powerful operations can be performed using just one or two commands.
- Matlab makes use of highly respected algorithms; hence you may be confident about your results.
- Excellent graphics facilities are available and the pictures can be inserted into LATEX and Word documents.

Starting up

Matlab can be used in a number of different ways or modes: for instance

- as an advanced calculator in the calculator mode,
- in a high level programming language mode,
- as a subroutine called from a C-program.

When used in calculator mode, all Matlab commands are entered to the command line from the keyboard at the "command line prompt" indicated with

'>>'

Type

```
>> quit
```

at any time to exit from Matlab.

Help

Extensive documentation is available, either via the command line by using the "help topic" command (as we are going to see) or via the internet.

We recommend starting with the command

```
>> demo
```

This brings up a separate window which gives access to a short video entitled "Getting Started" that describes the purpose of the various panes in the main Matlab window.

Help is available from the command line prompt. Type

`help`

for getting "help" in a list of topics. The first few lines of this read

```
>> help
HELP topics:

matlab\datafun     - Data analysis and Fourier transforms.
matlab\datatypes     - Data types and structures.
matlab\elfun     - Elementary math functions.
matlab\elmat     - Elementary matrices and matrix
manipulation.
matlab\funfun     - Function functions and ODE solvers.
matlab\general     - General purpose commands.
matlab\iofun     - File input and output.
matlab\lang     - Programming language constructs.
matlab\matfun     - Matrix functions - numerical linear
algebra.
```

Then to obtain help on "Elementary math functions", for instance, type

```
>> help elfun
```

Clicking on a key word, for example

```
sin
```

will provide further information together with a link to

```
Reference page for sin
```

which provides the most extensive documentation on the key word itself along with examples of its use.

Matlab as a calculator

The basic arithmetic operators are

$$+ \quad - \quad * \quad / \quad \hat{}$$

and these are used in conjunction with brackets:

$$(\qquad)$$

The symbol ^ is used to get exponents (powers).
You should type the operators in the commands shown at the prompt: >>.

Type

```
>> 2 + 3/4*5
ans =
5.7500
>>
```

Is this calculation $2 + 3/(4 * 5)$ or $2 + (3/4) * 5$?

Matlab works according to the priorities:

1. quantities in brackets,
2. powers $(2 + 3\hat{}2 = 2 + 9 = 11)$,
3. $*$ $/$, working left to right $(4 * 2/5 = 8/5, \quad 4/2 * 5 = 2 * 5 = 10)$,
4. $+$ $-$, working left to right $(3 + 4 - 5 = 7 - 5)$.

Thus, the earlier calculation was for $2 + (3/4) * 5$ by priority 3.

Numbers and formats

Matlab recognizes different kinds of numbers:

- Integer
  Examples: 1375, -21975
- Real
  Examples: 1.234, -10.76
- Complex
  Examples: $3.21 - 4.3i$ $(i = \sqrt{-1})$
- Inf
  Infinity (result of dividing by 0)
- NaN
  Not a Number, $0/0$

The "e" notation is used for very large or very small numbers:

$$-1.3412e+03 = -1.3412 \cdot 10^{3} = -1341.2$$

$$1.3412e-01 = 1.3412 \cdot 10^{-1} = 0.13412$$

All computations in MATLAB are done in double precision, which means about 15 significant figures.

How Matlab prints numbers is controlled by the "format" command. For full list type

```
>> help format
```

As an example:

```
>> format
```

with no inputs sets the output format to the default appropriate for the class of the variable.

```
>> format short
```

Scaled fixed point format with 5 digits.

```
>> format long
```

Scaled fixed point format with 15 digits.

Variables

```
>> 3-2^4
ans =
-13
>> ans*5
ans =
-65
```

The result of the first calculation is labelled "ans" by Matlab and is used in the second calculation, where its value is changed.

We can use our own names to store numbers:

```
>> x = 3-2^4
x =
-13
>> y = x*5
y =
-65
```

In order to represent variables, it is convenient to use names that respect them.
Legal names for variables consist of any combination of letters and digits, starting with a letter.

These are allowable:

`NetCost, Left2Pay, x3, X3, z25c5`

They are **not** allowable:

`Net-Cost, 2Pay, %x, @sign`

Special names: you should avoid using

`eps`
(which has the value $2.2204e-16 = 2^{-52}$. The smallest number greater than 1 which can be represented in Matlab is 1+eps)

`pi` $= 3.14159\cdots = \pi$

Suppressing output

One often does not want to see the result of intermediate calculations.

In that case, one terminates the assignment statement or expression with semi-colon.

```
>> x=-13; y = 5*x, z = x ^ 2+y
y =
-65
z =
104
>>
```

In the previous example, the value of `x` is hidden.
Note that we can place several statements on one line, separated by commas or semicolons.

Built-in Functions

Matlab provides many scalar functions; they are divided into Trigonometric, Exponential, Complex, Rounding and remainder.

```
>> x = 9;
>> sqrt(x), exp(x), log(sqrt(x)), log10(x ^ 2+6)
ans =
3
ans =
8.1031e+03
ans =
1.0986
ans =
1.9395
```

`exp(x)` denotes the exponential function $exp(x) = e^x$ and the inverse function is `log`.

Try to type:

```
>> format long, exp(log(9)), log(exp(9))
ans = 9.000000000000002e+00
ans = 9
>> format short
```

In the previous example, we see a tiny rounding error in the first calculation.

log10 gives logs to the base 10.

A complete list of elementary built-in functions is provided when typing

```
>> help elfun
```

Vectors

At first, we describe **row vectors**: they are lists of numbers separated by either commas or spaces.

The number of entries is known as the "length" of the vector and the entries are often referred to as "elements" or "components" of the vector.

The entries must be enclosed in square brackets.

For instance:

```
>> v = [3 1,sqrt(2)]
v =
   3.0000    1.0000    1.4142
>> length(v)
ans =
   3
```

Spaces can be vitally important:

```
>> v2 = [4+ 5 6]
v2 =
   9    6
>> v3 = [4 +5 6]
v3 =
   4    5    6
>>
```

A vector may be multiplied by a scalar (i.e. a number), or added/subtracted to another vector of the same length. The operations are carried out elementwise.

In this respect, we perform some arithmetic operations with vectors of the same length, such as `v` and `v3` in the previous example.

```
>> v + v3
ans =
   7.0000    6.0000    7.4142
>> v4 = 3* v
v4 =
   9.0000    3.0000    4.2426
>> v5 = 2*v -3*v3
v5 =
  -6.0000   -13.0000   -15.1716
>> v + v2
Error using +
Matrix dimensions must agree.
```

The error is due to `v` and `v2` having different lengths.

We can build row vectors from existing ones:

```
>> w = [1 3 2], z = [9 7], cd = [2*z,-w], sort(cd)
w =
   1   3   2
z =
   9   7
cd =
   18   14   -1   -3   -2
ans =
   -3   -2   -1   14   18
>>
```

Notice the last command `sort`'ed the elements of `cd` into ascending order.

We can also change or look at the value of particular entries:

```
>> w, w(2) = -5, w(3)
w =
   1    3    2
w =
   1   -5    2
ans =
   2
>>
```

The colon notation

This is a shortcut for producing row vectors:

```
>> 0:2
ans =
  0   1   2
>> 3:7
ans =
  3   4   5   6   7
>> 2:-1
ans =
  Empty matrix:  1-by-0
```

In general, `a:b:c` produces a vector of entries starting from the value `a`, incrementing by the value `b` until it gets `c`. (it will not produce a value beyond `c`).

This is why `2:-1` has given the empty vector.

For instance:

```
>> 0.31:0.1:0.7
ans =
  0.3100   0.4100   0.5100   0.6100
>> -1.4:-0.3:-2
ans =
  -1.4000   -1.7000   -2.0000
>>
```

Extracting parts of vectors

Define:

```
>> r = [2:3:9, -1:-2:-7]
r =
  2   5   8   -1   -3   -5   -7
```

To get the 3rd to 6th entries:

```
>> r(3:6)
ans =
  8   -1   -3   -5
```

To get alternate entries:

```
>> r(1:2:7)
ans =
  2   8   -3   -7
```

```
>> r
r =
   2    5    8   -1   -3   -5   -7
```

What does `r(6:-2:1)` give?

```
>> r(6:-2:1)
ans =
   -5   -1    5
```

For a fuller description, see

```
>> help colon
```

Column vectors

Column vectors have similar constructs to row vectors except that entries are separated by ";" or "newlines".

```
>> c = [3; 1; sqrt(2)]
c =
   3.0000
   1.0000
   1.4142
>> c2 = [3
4
5]
c2 =
   3
   4
   5
```

Column vectors may be added or subtracted, **provided that they have the same length**.

```
>> c3 = 3*c2-2*c
c3 =
   3.0000
  10.0000
  12.1716
```

The length of a vector (number of elements) can be determined by

```
>> length(c)
ans =
   3
>> length(r)
ans =
   7
```

`length` command does not distinguish between row and column vectors.

The size might be needed to determine the last element in a vector but this can be found by using the word `end`:

```
>> c(end), c(end-1:end)
ans =
  1.4142
ans =
  1.0000
  1.4142
```

Transposing

A row vector can be converted into a column vector (and viceversa) by a process which is called "transposing" and denoted by '

```
>> w, c'
w =
   1    3    2
ans =
   3.0000    1.0000    1.4142
```

and

```
>> t = 2*w-c'
t =
   -1.0000    5.0000    2.5858
>> T = 5*w'+2*c
T =
   11.0000
   17.0000
   12.8284
```

How to keep a record

The command

```
>> diary mysession
```

causes that all subsequent text that appears on the screen is saved into the file `mysession` located in the directory in which Matlab is invoked.

Instead of `mysession`, you may use any legal filename except the names `on` and `off`.
The record may be terminated by

```
>> diary off
```

The file `mysession` may be edited with your favourite editor (the Matlab editor, emacs, or even Word).

A list of variables used in the current session may be seen by the command:

```
>> whos
```

Keyboard accelerators

It is possible to recall previous Matlab commands in the Command Window by using the $\uparrow$ and $\downarrow$ cursor keys.
Repeatedly pressing $\uparrow$ will review the previous commands and, if you wish to re-execute the command, simply press the return key.

Once a command has been recalled, it may be edited (changed). You can use $\leftarrow$ and $\rightarrow$ to move backwards and forwards through the line, characters may be inserted by typing at the current cursor position or deleted using the `Del` key.

This procedure is most commonly used when long command lines have been mistyped or when you want to re-execute a command that is very similar to one previously used.

Inner product ($*$)

Let $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ be two vectors.
We recall that the **scalar** or **inner product** of $\mathbf{x}$ and $\mathbf{y}$ is given by

$$\langle \mathbf{x}, \mathbf{y} \rangle = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

It is a scalar (i.e. a number)!!!

In Matlab environment, define

```
>> x = [10 -11 12], y = [2 1 3]
x =
   10   -11    12
y =
   2    1    3
```

In order to evaluate $\langle \mathtt{x}, \mathtt{y} \rangle$, we consider the transpose vector of `y`. In this way, we have a row vector `x` and a column vector `y'`; thus, we may perform the matrix product $\mathtt{x} * \mathtt{y}'$.

```
>> x*y'
ans =
  45
```

Notice that:

```
>> x*y
Error using   *
Inner matrix dimensions must agree.
```

An error results because `y` is not a column vector.

Elementwise product (.∗)

Let $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ be two vectors.
The **Hadamard product** of **x** and **y** is given by

$$\mathbf{x} .* \mathbf{y} = (x_1 y_1, x_2 y_2, \ldots, x_n y_n)$$

It is the vector which is obtained by simply multiplying elementwise the corresponding entries of **x** and **y**.

In Matlab, the product is computed by means of the operator `.*`:

```
>> x.*y
ans =
  20   -11   36
```

Summing the entries in the resulting vector would give the inner product $\langle \mathrm{x}, \mathrm{y} \rangle$:

```
>> sum(ans)
ans =
  45
```

Perhaps the most common use of the Hadamard product is in the evaluation of mathematical expressions so that they may be plotted.

Example: *Tabulate the function* $y = x \cdot e^{\pi x}$ *for* $x = 0, 0.25, 0.5, 0.75, 1$.

At first we define a column vector of x-values:

```
>> x = (0:0.25:1)'
x =
   0
   0.2500
   0.5000
   0.7500
   1.0000
```

Then, to evaluate y we have to multiply each element of the vector x by the corresponding element of the vector $e^{\pi x}$:

x	$\times$	$e^{\pi x}$	$=$	$xe^{\pi x}$
0	$\times$	1	$=$	0
0.25	$\times$	2.1933	$=$	0.5483
0.5	$\times$	4.8105	$=$	2.4052
0.75	$\times$	10.5507	$=$	7.9130
1	$\times$	23.1407	$=$	23.1407

To carry this out in Matlab:

```
>> y = x.*exp(pi*x)
y =
   0
   0.5483
   2.4052
   7.9130
   23.1407
```

Note:

1. the use of `pi`
2. `x` and `exp(pi*x)` are both column vectors (the `exp` function is applied to each element of the vector).
   Thus, the Hadamard product of these is also a column vector.

Elementwise division (./)

The operator ./ is defined to give element by element division of one vector by another one.
Therefore, this kind of calculation is only defined for vectors of the same size and type.

```
>> a = 1:5, b = 6:10
a =
   1   2   3   4   5
b =
   6   7   8   9   10
>> a./b
ans =
   0.1667   0.2857   0.3750   0.4444   0.5000
```

If we change to `format rat` (short for rational), we get:

```
>> format rat
>> a./b
ans =
  1/6   2/7   3/8   4/9   1/2
```

Moreover:

```
>> format
>> c = -2:2, a
c =
  -2   -1   0   1   2
a =
  1   2   3   4   5
>> a./c
ans =
  -0.5000   -2.0000   Inf   4.0000   2.5000
```

The previous calculation required division by 0; notice the `Inf`, denoting infinity, in the answer.

As another example:

```
>> a.*b -24
ans =
  -18   -10    0    12    26
>> ans./c
ans =
  9    10    NaN    12    13
```

Here we are warned about 0/0 giving a `NaN` (Not a Number).

Elementwise power (.^)

The operator .^ performs the power elmentwise.
For instance:

```
>> u = [10, 11, 12]
u =
   10   11   12
>> u.^2
ans =
   100   121   144
>> u.^4
ans =
   10000   14641   20736
>> u.^(1/2)
ans =
   3.1623   3.3166   3.4641
```

Plotting Functions

In order to plot the graph of a function $y = f(x)$, it is sampled at a sufficiently large number of points and the points (x, y) are joined by straight lines.

For instance, we would like to plot the values of $y = \sin(3\pi x)$ for $0 \leq x \leq 1$.
With this aim, suppose we take $N + 1$ sampling points equally spaced a distance h:

$$x = 0, h, 2h, \ldots, 1 - h, 1$$

with $h = 1/N$.

```
>> N = 10; h = 1/N; x = 0:h:1
```

Then corresponding y values are computed by

```
>> y = sin(3*pi*x);
```

and finally, we can plot the points with

```
>> plot(x,y)
```

In the previous example, the value of N is too small. Thus we change the value of N to 100:

```
>> N = 100; h = 1/N; x = 0:h:1;
>> y = sin(3*pi*x);
>> plot(x,y)
```

We get a smooth graph.

To put a title and label the axes, we use:

```
>> title('Graph of y = sin(3 pi x)')
>> xlabel('x-axis')
>> ylabel('y-axis')
```

The strings enclosed in single quotes, can be anything of our choosing.

The default is to plot solid lines.
A dashed red line is produced by

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

A solid green line, with the simbol 'x' drawn at each data point, is produced by

```
>> plot(x,y,'g-x')
```

The third argument is a string where the characters specify the colour (green), the line style (solid) and the symbol (x) to be drawn at each data point. The order in which they appear is unimportant and any, or all, may be omitted.

The options for colours, styles and symbols are in a full list that is shown by means of command

```
>> help plot
```

Exercises:

Draw graphs of the functions

$$f(x) = 1 - x, \qquad 0 \leq x \leq 2$$

$$f(x) = \frac{x}{x^2 + 1}, \qquad -2 \leq x \leq 2$$

$$f(x) = \frac{x^2 + 1}{x^2 - 4}, \qquad -1 \leq x \leq 1$$

Two-dimensional arrays

In a mathematical setting it is usual to enclose the entries of any matrix in either round or square brackets: Matlab insists on square ones!
For instance, consider

$$A = \begin{pmatrix} 5 & 7 & 9 \\ 1 & -3 & -7 \end{pmatrix}$$

To enter such a matrix into Matlab we type it in row by row using the same syntax as for vectors:

```
>> A = [5 7 9
1 -3 -7]
```

Rows may be separated by semi-colons rather than a new line:

```
>> B = [-1 2 5; 9 0 5]
B =
  -1   2   5
   9   0   5
>> C = [0, 1; 3, -2; 4, 2]
C =
   0   1
   3  -2
   4   2
>> D = [1:5; 6:10; 11:2:20]
D =
   1   2   3   4   5
   6   7   8   9  10
  11  13  15  17  19
```

Size of a matrix

We can get the size (dimensions) of a matrix with the command `size`

```
>> size(A)
ans =
  2   3
>> size(D)
ans =
  3   5
```

Transpose of a matrix

Transposing a vector changes it from a row to a column vector and vice versa. The extension of this idea to matrices is that transposing interchanges rows with the corresponding columns: the 1st row becomes the 1st column, and so on.

```
>> D, D'
D =
   1    2    3    4    5
   6    7    8    9    10
   11    13    15    17    19
ans =
   1    6    11
   2    7    13
   3    8    15
   4    9    17
   5    10    19
```

In order to get the sizes:

```
>> size(D)
ans =
   3    5
>> size(D')
ans =
   5    3
```

Special matrices

Matlab provides a number of useful built-in matrices of any desired size. For instance:

`ones(m,n)` gives an $m \times n$ matrix of 1's

```
>> P = ones(2,3)
P =
   1   1   1
   1   1   1
```

`zeros(m,n)` gives an $m \times n$ matrix of 0's

```
>> Z = zeros(3,2)
Z =
   0   0
   0   0
   0   0
```

Moreover:

`eye(m)` gives the $m \times m$ identity matrix

```
>> I=eye(3)
I =
   1   0   0
   0   1   0
   0   0   1
```

`rand(n)` gives an $n \times n$ matrix with entries randomly distributed in $[0, 1]$

```
>> R = rand(2)
R =
   0.8147    0.1270
   0.9058    0.9134
```

Matrix product

Consider two matrices A and B.
Assume that A is $m \times n$ and B is $n \times p$.
In Matlab environment, product $A \cdot B$ ($m \times p$) is easily performed.

```
>> A = [5 7 9; 1 -3 -7], B = [0, 1; 3, -2; 4, 2]
A =
   5    7    9
   1   -3   -7
B =
   0    1
   3   -2
   4    2
>> C=A*B
C =
   57    9
  -37   -7
```

Elementwise operations (.* ./ .^)

Elementwise operations work as for vectors: corresponding elements are multiplied-divided together (so the matrices involved must have the same size). Furthermore, each element is powered as for vectors.

```
>> A = [1 2 3; 4 5 6], B = [0 -1 2; 1 -1 1]
A =
   1   2   3
   4   5   6
B =
   0   -1   2
   1   -1   1
>> A.*B
ans =
   0   -2   6
   4   -5   6
```

Then

```
>> C = eye(2)
C =
   1   0
   0   1
>> A.*C
Error using .*
Matrix dimensions must agree.
>> A./B
ans =
   Inf   -2.0000   1.5000
   4.0000   -5.0000   6.0000
>> A.^3
ans =
   1   8   27
   64   125   216
```

Determinant of a matrix

Determinant of any square matrix is produced by built-in function `det`.

```
>> A = 100*rand(3)
A =
   17.1187   27.6923   82.3458
   70.6046    4.6171   69.4829
    3.1833    9.7132   31.7099
>> det(A)
ans =
   -9.6593e+03
```

Eigenvalues of a matrix

Eigenvalues of any square matrix are produced by built-in function `eig`.

```
>> A = [1 3; 3 1]
A =
   1   3
   3   1
>> eig(A)
ans =
  -2
   4
```

Size of A may be increased, without any effort:

```
>> A = rand(4), B = A+A'
A =
   0.5830   0.2653   0.3439   0.8797
   0.2518   0.8244   0.5841   0.8178
   0.2904   0.9827   0.1078   0.2607
   0.6171   0.7302   0.9063   0.5944
B =
   1.1660   0.5171   0.6343   1.4967
   0.5171   1.6488   1.5667   1.5480
   0.6343   1.5667   0.2155   1.1670
   1.4967   1.5480   1.1670   1.1887
>> eig(B)
ans =
  -0.8263
  -0.6219
   1.0093
   4.6578
```

An application

Consider function

$$f(x, y, z) = x^3 - y^3 + xy + z^2.$$

Notice that the partial derivatives of f are

$$f_x(x, y, z) = 3x^2 + y, \qquad f_y(x, y, z) = x - 3y^2, \qquad f_z(x, y, z) = 2z.$$

Thus, by solving the following system

$$\begin{cases} 3x^2 + y = 0 \\ x - 3y^2 = 0 \\ 2z = 0 \end{cases}$$

we get the stationary points

$$P_1 = (0, 0, 0) \qquad P_2 = \left(\frac{1}{3}, -\frac{1}{3}, 0\right)$$

In order to investigate the nature of P_1 and P_2, we notice that Hessian matrix is defined at any point (x, y, z) and is

$$H(x, y, z) = \begin{pmatrix} 6x & 1 & 0 \\ 1 & -6y & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

In Matlab environment, we evaluate it at P_1:

```
>> x = 0; y = 0; z = 0;
>> H = [6*x 1 0; 1 -6*y 0; 0 0 2]
H =
   0   1   0
   1   0   0
   0   0   2
>> eig(H)
ans =
  -1
   1
   2
```

The first eigenvalue is negative, the other ones are positive, then $H(P_1)$ is not definite and P_1 represents a saddle point.

In Matlab environment, we evaluate it at P_2:

```
>> x = 1/3; y = -1/3; z = 0;
>> H = [6*x 1 0; 1 -6*y 0; 0 0 2]
H =
   2   1   0
   1   2   0
   0   0   2
>> eig(H)
ans =
   1
   2
   3
```

Eigenvalues are positive, then $H(P_2)$ is positive definite and P_2 represents a minimum point.

Plotting surfaces

A surface is defined by a function $f(x, y)$. For each value of (x, y) we compute the height of the function by

$$z = f(x, y)$$

In order to plot it we have to decide on the ranges of x and y: suppose $2 \leq x \leq 4$ and $1 \leq y \leq 3$. This gives us a rectangle in the (x, y)-plane.

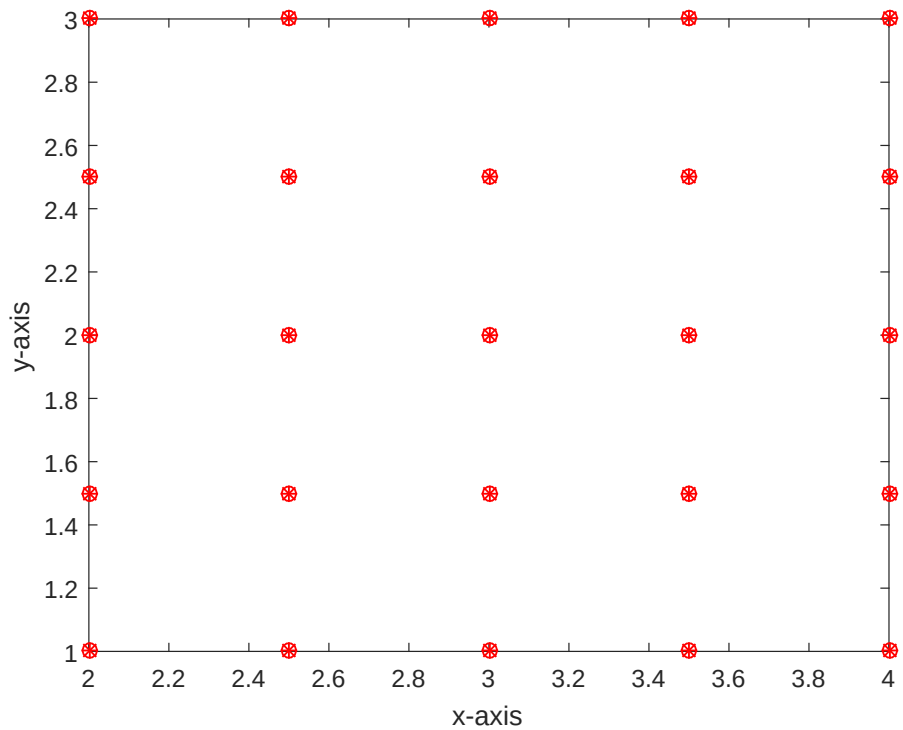
Next, we need to choose a grid on this domain.

Finally, we have to evaluate the function at each point of the grid and "plot" it.

As an example, we choose a grid with intervals 0.5 in each direction (i.e both in x and in y direction).
The x and y coordinates of the grid lines are

```
x = 2:0.5:4,    y = 1:0.5:3
```

in Matlab notation.



We construct the grid by:

```
>> x = 2:0.5:4; y = 1:0.5:3;
>> [X,Y] = meshgrid(x,y);
X =
   2.0000   2.5000   3.0000   3.5000   4.0000
   2.0000   2.5000   3.0000   3.5000   4.0000
   2.0000   2.5000   3.0000   3.5000   4.0000
   2.0000   2.5000   3.0000   3.5000   4.0000
   2.0000   2.5000   3.0000   3.5000   4.0000
Y =
   1.0000   1.0000   1.0000   1.0000   1.0000
   1.5000   1.5000   1.5000   1.5000   1.5000
   2.0000   2.0000   2.0000   2.0000   2.0000
   2.5000   2.5000   2.5000   2.5000   2.5000
   3.0000   3.0000   3.0000   3.0000   3.0000
```

For any i and j, pair $(X(i,j), Y(i,j))$ represents the coordinates of a node in the grid.

We need to evaluate function f using X and Y in place of x and y, respectively.

Exercise: Plot the surface defined by the function

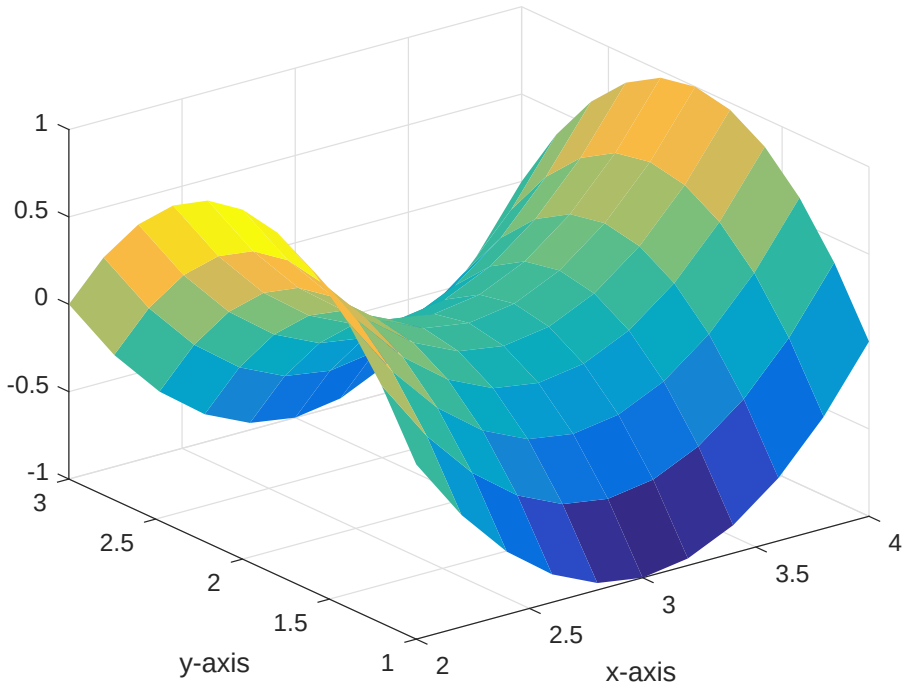
$$f(x,y) = (x-3)^2 - (y-2)^2$$

for $2 \leq x \leq 4$ and $1 \leq y \leq 3$.

Type:

```
>> x = 2:0.2:4; y = 1:0.2:3;
>> [X,Y] = meshgrid(x,y);
>> f = (X-3).^2-(Y-2).^2;
>> surf(X,Y,f)
>> shading flat
>> xlabel('x-axis')
>> ylabel('y-axis')
```

We obtain



Example

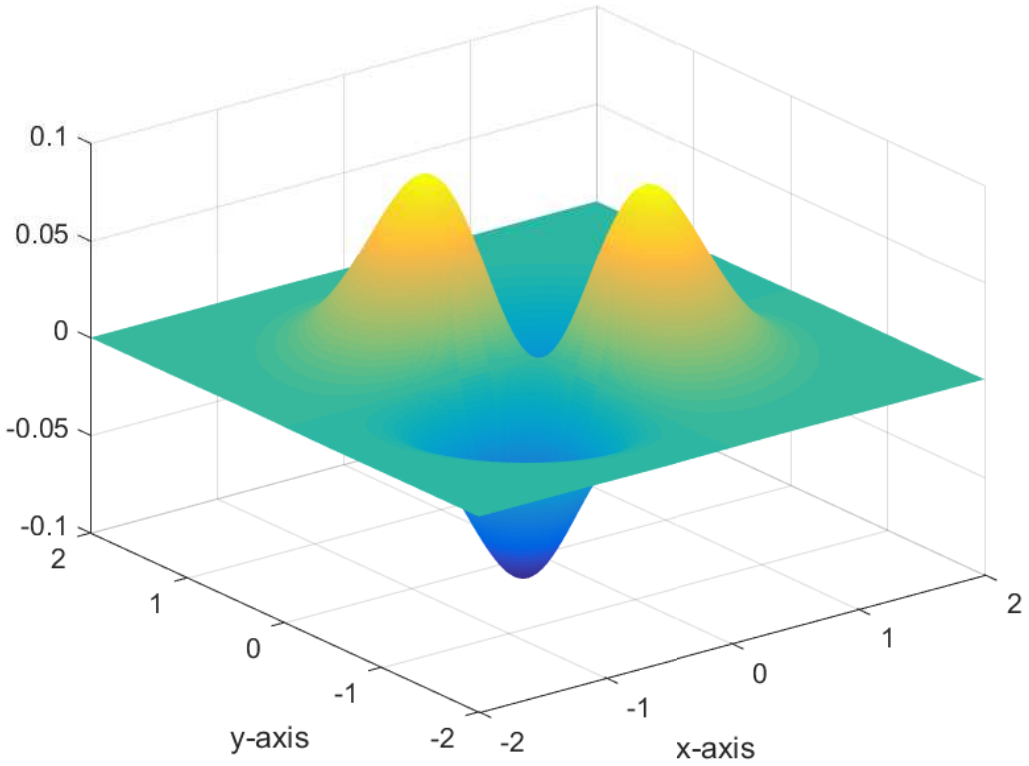
Plot the surface defined by the function

$$f(x, y) = -xye^{-2(x^2+y^2)}$$

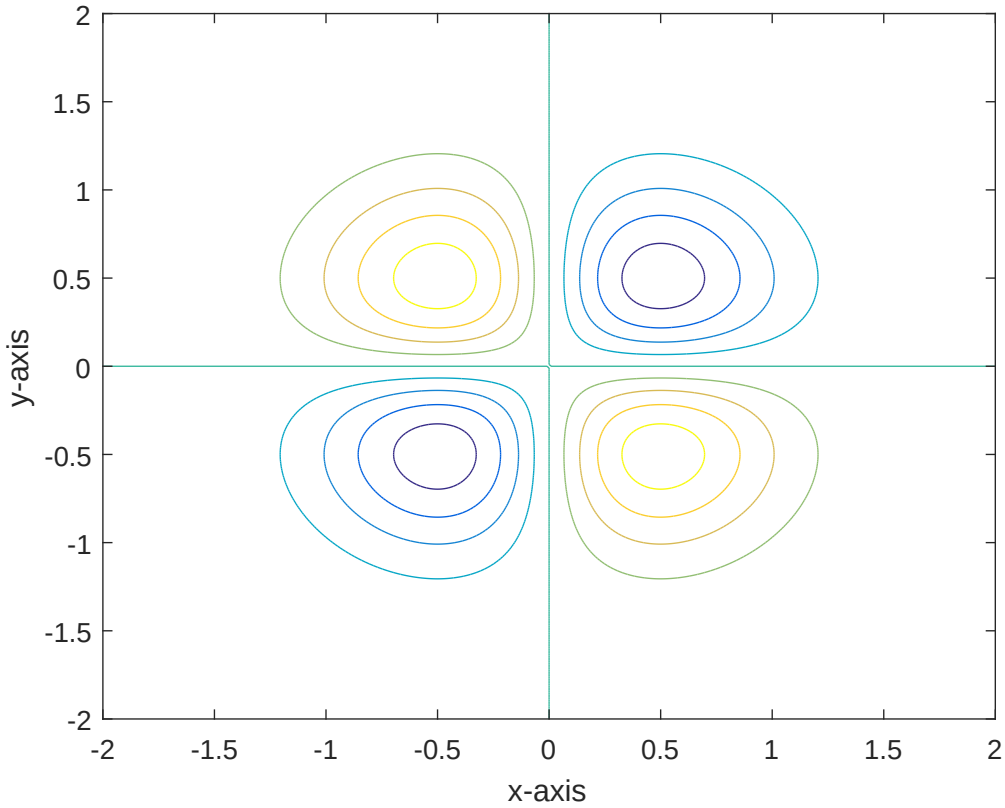
on the domain $-2 \leq x \leq 2$, $-2 \leq y \leq 2$.

```
>> x = -2:.01:2; y = -2:.01:2;
>> [X,Y] = meshgrid(x,y);
>> f = -X.*Y.*exp(-2*(X.^2+Y.^2));
>> surf(X,Y,f)
>> shading flat
>> xlabel('x-axis'), ylabel('y-axis')
>> contour(x,y,f)
>> xlabel('x-axis'), ylabel('y-axis')
```

We obtain



and



An application: Example 1

Consider function

$$f(x, y) = 3x^4 + 3x^2y - y^3.$$

Notice that the partial derivatives of f are

$$f_x(x, y) = 12x^3 + 6xy, \qquad f_y(x, y) = 3x^2 - 3y^2.$$

Thus, by solving the following system

$$\begin{cases} 2x^3 + xy = 0 \\ x^2 - y^2 = 0 \end{cases}$$

we get the stationary points

$$P_1 = (0, 0) \qquad P_2 = \left(-\frac{1}{2}, -\frac{1}{2}\right) \qquad P_3 = \left(+\frac{1}{2}, -\frac{1}{2}\right)$$

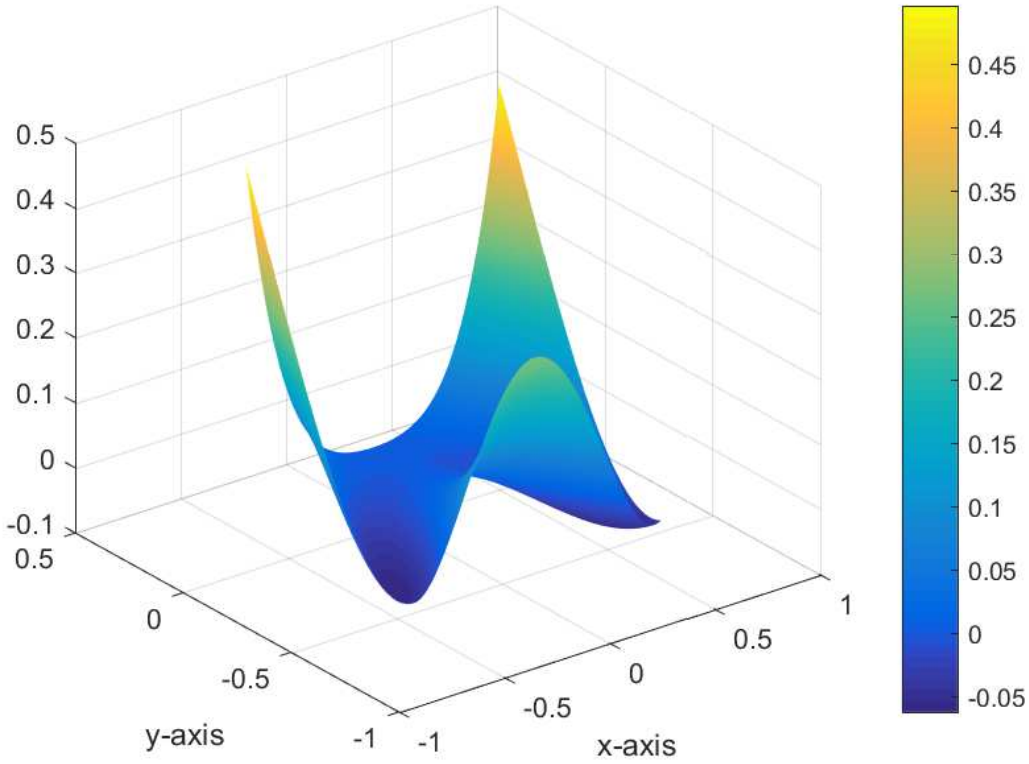
P_1, P_2, P_3 are candidate points to be extrema.

Notice that they lie in the rectangle $[-0.6, 0.6] \times [-0.65, 0.1]$. Then we plot surface and contour lines of f in the domain

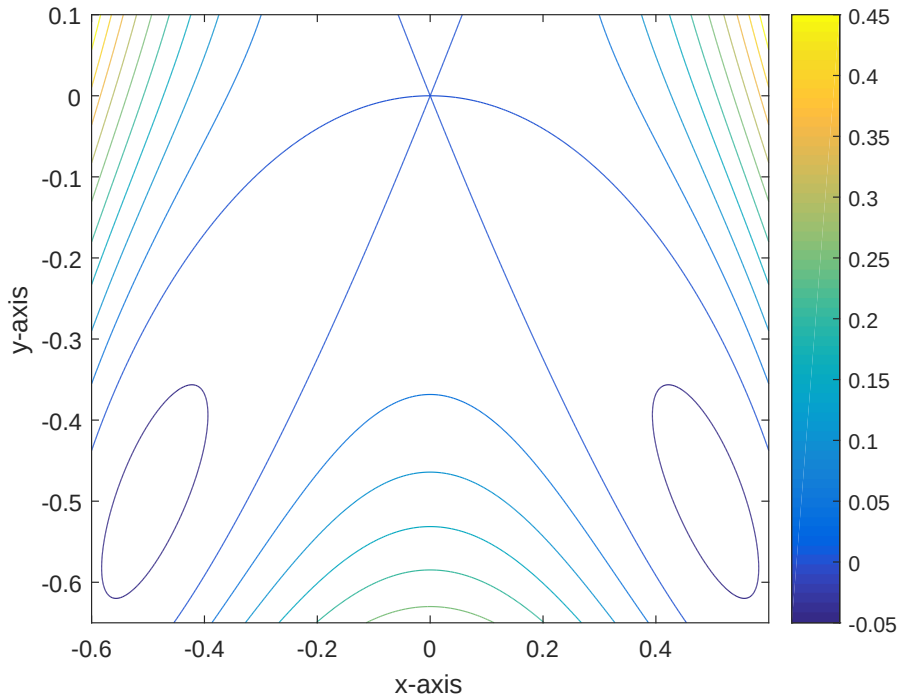
$$-0.6 \leq x \leq 0.6, \qquad -0.65 \leq y \leq 0.1$$

```
>> x = -0.6:0.001:0.6; y = -0.65:0.001:0.1;
>> [X,Y] = meshgrid(x,y);
>> f = 3*X.^4+3*X.^2.*Y-Y.^3;
>> surf(X,Y,f)
>> shading flat
>> xlabel('x-axis'), ylabel('y-axis'), colorbar
>> figure(2)
>> contour(x,y,f)
>> xlabel('x-axis'), ylabel('y-axis'), colorbar
```

We obtain


0.5
0.4
0.3
0.2
0.1
0
-0.1
0.5
0
-0.5
-1
y-axis
-1
-0.5
0
0.5
1
x-axis
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05

and



P_1 represents a saddle point. Indeed, it is evident that contour lines cross at P_1.
Contour lines are closed around P_2 and P_3 and $f(x, y)$ gets low values around them (which correspond to blue colour in colour bar): P_2 and P_3 are minimum points.

On the other hand, in analytical way, Hessian matrix is defined at any point (x, y) and is

$$H(x,y) = \begin{pmatrix} 36x^2 + 6y & 6x \\ 6x & -6y \end{pmatrix}$$

In Matlab environment, we evaluate it at P_2:

```
>> x = -1/2; y = -1/2;
>> H = [36*x^2+6*y 6*x; 6*x -6*y]
H =
   6   -3
  -3    3
>> eig(H)
ans =
   1.1459
   7.8541
```

Both eigenvalues are positive, then $H(P_2)$ is positive definite and P_2 represents a minimum point. That is expected!

In Matlab environment, we evaluate Hessian matrix at P_3:

```
>> x = 1/2; y = -1/2;
>> H = [36*x^2+6*y 6*x; 6*x -6*y]
H =
   6   3
   3   3
>> eig(H)
ans =
   1.1459
   7.8541
```

Both eigenvalues are positive, then $H(P_3)$ is positive definite and P_3 represents a minimum point. That is expected!

Hessian matrix at P_1 is

$$H(P_1) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Both eigenvalues $\lambda_1 = \lambda_2 = 0$ nullifies, hence $H(P_1)$ is semidefinite and we cannot draw any conclusion about its nature.

In analytical way: we notice that $f(P_1) = 0$ and

$$\Delta f_{P_1}(x, y) := f(x, y) - f(0, 0) = f(x, y).$$

We consider the case when $x = 0$ and we get $\Delta f_{P_1}(0, y) = -y^3$.

- $\Delta f_{P_1}(0, y) > 0$ if $y < 0$
- $\Delta f_{P_1}(0, y) < 0$ if $y > 0$

Thus, $\Delta f_{P_1}(x, y)$ attains both positive and negative values in any neighbourhood of P_1.
It follows that P_1 is a saddle point. That is expected!

An application: Example 2

Consider function

$$f(x, y) = \log(1 + x^2 + y^2).$$

It is defined at any point (x, y) in $\mathbb{R}^2$. Notice that the partial derivatives of f are

$$f_x(x, y) = \frac{2x}{1 + x^2 + y^2}, \qquad f_y(x, y) = \frac{2y}{1 + x^2 + y^2}.$$

We get stationary point

$$P = (0, 0)$$

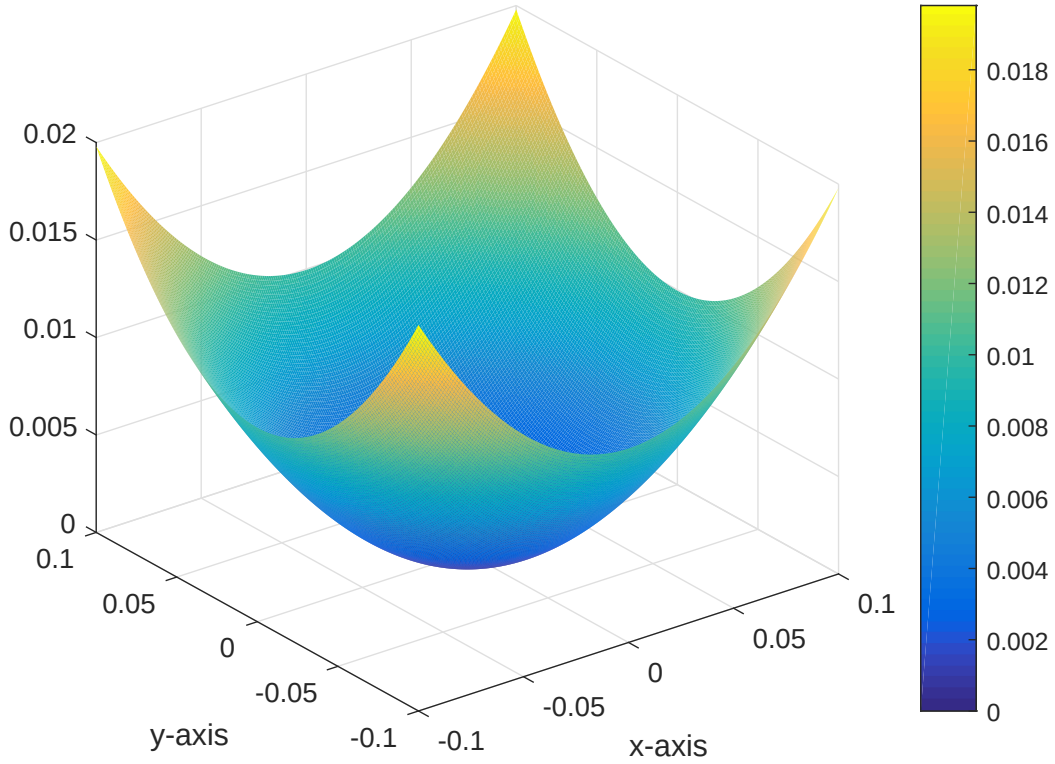
It is candidate to be an extremum.

We plot surface and contour lines of f in the rectangle

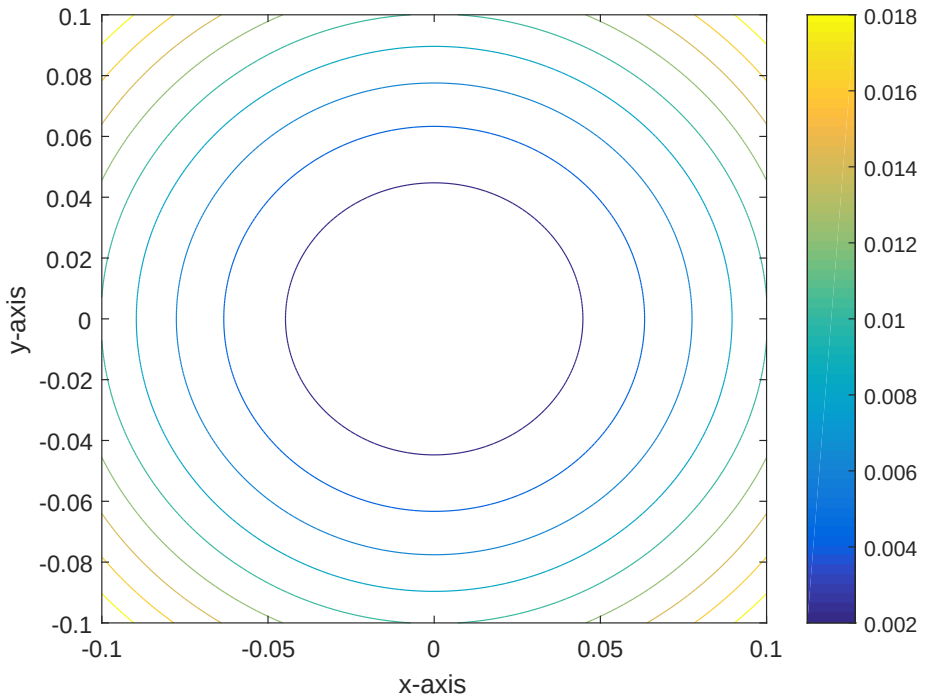
$$-0.1 \leq x \leq 0.1, \qquad -0.1 \leq y \leq 0.1$$

```
>> x = -0.1:0.001:0.1; y = -0.1:0.001:0.1;
>> [X,Y] = meshgrid(x,y);
>> f = log(1+X.^2+Y.^2);
>> surf(X,Y,f)
>> shading flat
>> xlabel('x-axis'), ylabel('y-axis'), colorbar
>> figure(2)
>> contour(x,y,f)
>> xlabel('x-axis'), ylabel('y-axis'), colorbar
```

We obtain


0.02
0.015
0.01
0.005
0
0.1
0.05
0
-0.05
-0.1
y-axis
-0.1
-0.05
0
0.05
0.1
x-axis
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0

and



$P = (0, 0)$ represents a minimum point. It is evident that contour lines are closed around P and $f(x, y)$ gets low values around it (which correspond to blue colour in colour bar).

On the other hand, in analytical way, Hessian matrix is defined at any point (x, y) and is

$$H(x,y) = \begin{pmatrix} \dfrac{2-2x^2+2y^2}{(1+x^2+y^2)^2} & \dfrac{-4xy}{(1+x^2+y^2)^2} \\ \dfrac{-4xy}{(1+x^2+y^2)^2} & \dfrac{2+2x^2-2y^2}{(1+x^2+y^2)^2} \end{pmatrix}$$

At stationary point P, we get the diagonal matrix

$$H(P) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

Both eigenvalues $\lambda_1 = \lambda_2 = 2$ are positive, then $H(P)$ is positive definite and P represents a minimum point. That is expected!