Matlab-ის ელემენტები

ასისტენტ პროფესორი ნანა ოდიშელიძე

თბილისის სახელმწიფო უნივერსიტეტი, ზუსტ და საბუნემისმეტყველო მეცნიერებათა ფაკულტეტი, ინტერდისციპლინური (მათემატიკა, კომპიუტერული მეცნიერებები) ქვემიმართულება: მათემატიკური ლოგიკა და დისკრეტული სტრუქტურები

ბაკალავრიატი
თბილისი 2012

შესავალი.

პროგრამა “Matlab” –ი წარმოადგენს მაღალდონიან ტექნიკურ გამომთვლელ ენას და ინტერაქტიულ გარემოს ალგორითმების დასამუშავებლად. ის გამოიყენება მონაცემების ანალიზისთვის და მისი ვიზუალიზაციისთვის, რიცხვითი გათვლებისათვის. ტექნიკური გამომთვლელი ამოცანების უფრო სწრაფად ამოხსნა შესაძლებელია პროგრამა “Matlab”–ის გამოყენებით ვიდრე, ტრადიციული პროგრამირების ენების საშუალებით, ისეთი როგორიცაა C, C++ , Fortran.

პროგრამა “Matlab”–ის საშუალებით შესაძლებელია ფუნქციის გრაფიკების აგება, განტოლებების ამოხსნა, სტატისტიკური ტესტების შესრულება და ბევრი სხვა. შესაძლებელია გააერთიანოთ მათემატიკური გამოთვლები ტექსტთან და გრაფიკასთან იმისათვის, რომ მიიღოთ სრულყოფილი ინტეგრირებადი ინტერაქტიული დოკუმენტები.

შესალებელია სიმულაციისა და მოდელირების წარმოქმნა (თუ გაქვთ წვდომა დამატებით პროგრამასთან Simulink-თან). შესაძლებელია ხმის და ანიმაციური გრაფიკის შექმნა. ასევე შესაძლებელია მასალების შექმნა ინტერნეტში ექსპორტირებისათვის.

MATLAB (Matrix LABarotatory- მატრიცული ლაბორატორია) დამუშავებულია ფირმით The Math Works Inc. (აშშ, ქ. ნეიტიკი, შტ. მასაჩუსეტი).

სამუშაოს დაწყება.

პროგრამა “Matlab”-ის გაშვება

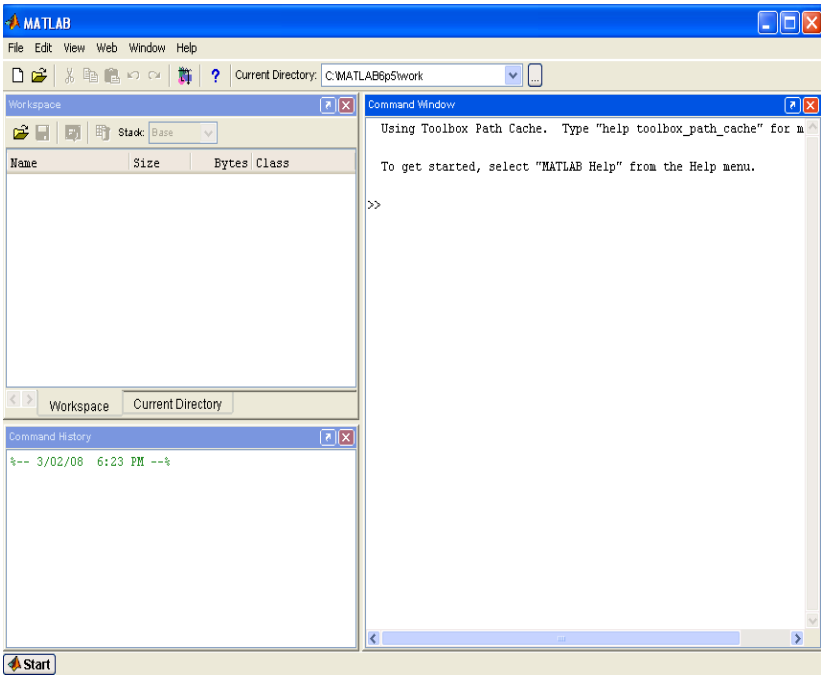
პროგრამა “Matlab”-ის გაშვება ხდება ისევე როგორც ნებისმიერი პროგრამის გაშვება: Windows -ის ოპერაციულ სისტემაში მთავარი მენიუდან (Start), სადაც პროგრამა აღნიშნულია სახელით “MATLAB 7” ან “ Student MATLAB”, Linux ან Unix ოპერაციულ სისტემებში საზოგადოდ საკმარისია ტერმინალის ფანჯარაში შევიტანოთ სიტყვა matlab-ი. შესაძლებელია აგრეთვე ისარგებლოთ პროგრამის იარლიყით სამუშაო მაგიდაზე, რომლის საშუალებითაც შესრულდება იგივე ამოცანა.

პროგრამა “Matlab” ის ფანჯრები

პროგრამის გაშვების შედეგად გაიშვება პროგრამა “Matlab” –ის ძირითადი ფანჯარა, რომელსაც სამუშაო მაგიდას ვუწოდებთ. ეს ფანჯარა შეიცავს სათაურის სტრიქონს (პროგრამის სახელწოდებას- **MATLAB**), მენიუს სტრიქონს, ინსტრუმენტთა პანელს და ოთხ შიდა ფანჯარას რომელთაგან ერთი დაფარულია. ყველაზე მნიშვნელოვანი ფანჯარაა **Command Window** (ბრძანების ფანჯარა) მარჯვენა მხარეს. სამი დანარჩენი ფანჯრებია: **Command History** (ბრძანებათა ისტორია), **Current Directory** (მიმდინარე საქაღალდე), **Workspace** (სამუშაო არე).

ეს ფანჯრები და სხვა ზოგიერთი ფანჯარაც შესაძლებელია გაშვებული იყოს ცალკეც და ჩამაგრებულიც სამუშაო მაგიდის ფანჯარაში. მათი ჩამაგრება

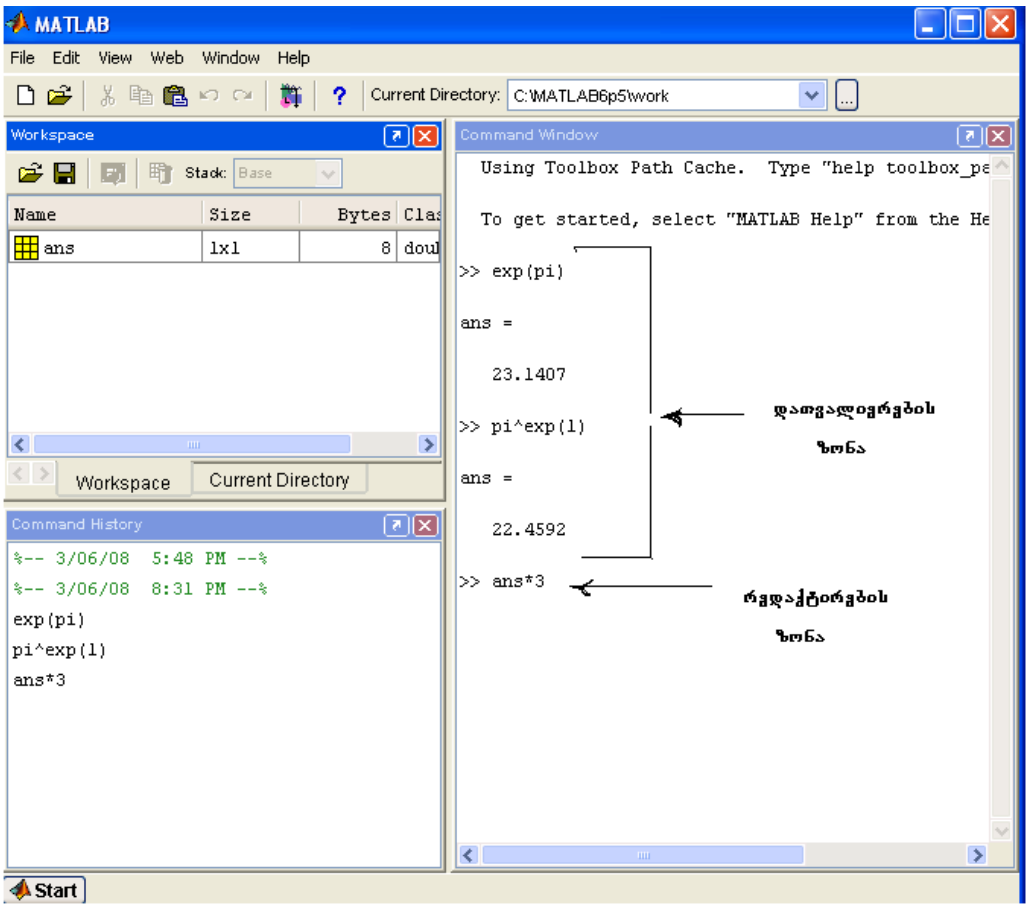
ხდება Desktop (სამუშაო მაგიდის(View-Desktop Layout---)) - ის საკუთარი მენიუდან. შეგიძლიათ გააქროთ ან ცალკე გამოიტანოთ თუ მისი ფანჯრის მარჯვენა კუთხის გაღუნულ ისარზე დააწკაპუნებთ.



ფანჯარა Command Window (ბრძანების ფანჯარა)

Command Window ფანჯრის ზედა ნაწილში შესაძლოა დავინახოთ ზოგადი ინფორმაცია **MATLAB** – თან დაკავშირებით, გარკვეული ინსტრუქცია სამუშაოს დაწყებისას, მაგრამ რაც მთავარია თქვენ დაინახავთ ბრძანების სტრიქონის მიწვევას (>> ან EDU >>) ამ არეში შეგვაქვს ბრძანებები. ფანჯარა რომ გავააქტიუროთ, მასზე დავაწკაპუნოთ ნებისმიერ ადგილას.

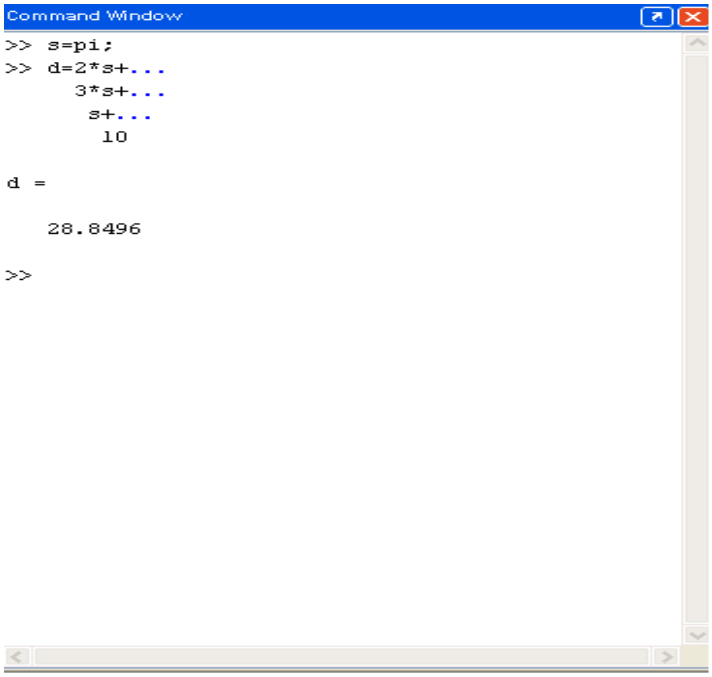
ფანჯარა **Command Window** გაყოფილია ორ ზონად: დათვალიერებისა და რედაქტირების ზონებად.



დათვალიერების ზონაში ჩასწორება შეუძლებელია, მისი ნებისმიერი ფრაგმენტი შეიძლება გამოვყოთ თაგვით, შემდგომ დავაკოპიროთ ბუფერში და მერე ჩავსვათ ბრძანების სტრიქონში **MATLAB** ან სხვა დოკუმენტში.

ფანჯარა **Command Window** -ს სუფთავდება ბრძანებით - **clc.**
ლოგიკური სტრიქონის სიგრძე არ აღემატება 256 სიმბოლოს.

საბრძანებო სტრიქონის გაგრძელება რამდენიმე ფიზიკურ სტრიქონად ხდება სამი წერტილის დაბოლოებით ყოველი ბრძანებისა და ENTER –ით.

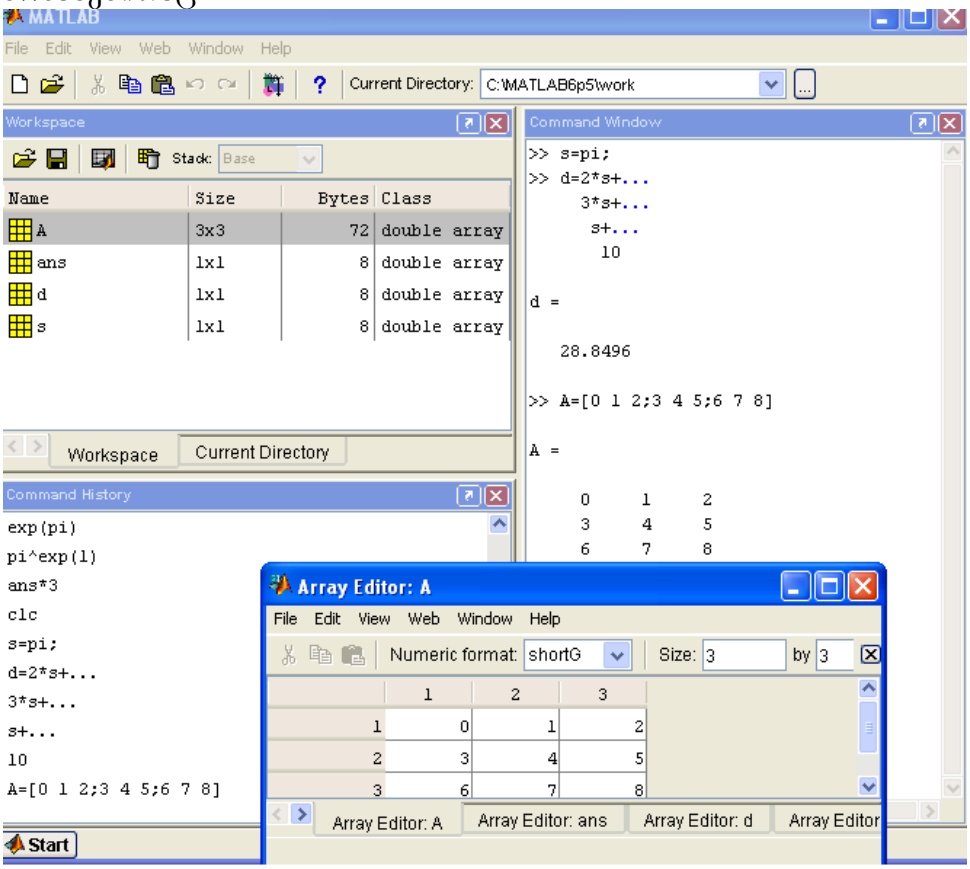


კლავიშების ↑↓ მოქმედებით ხდება წინა ან მომდევნო ბრძანებების შესრულება.

ფანჯარა Workspace (სამუშაო არე).

ცვლადების მნიშვნელობები, რომლებიც გამოითვლება სამუშაო სეანსის დროს ინახება სპეციალურად გამოყოფილ კომპიუტერის ოპერატიული მეხსიერების არეში, ეგრეთ წოდებულ **MATLAB** -ის **სამუშაო არეში –Workspace.**

იმისათვის რომ დავადგინოთ ნებისმიერი ცვლადის მნიშვნელობა საკმარისია ბრძანების სტრიქონში ავკრიფოთ ცვლადის სახელი და დავაჭიროთ **ENTER**, ანდა ფანჯარა **Workspace**-ში , რომელშიც ასახულია ყველა ცვლადები, რომლებიც გამოყენებულ იქნენ მუშაობისას, ორჯერ დავაწკაპუნოთ მათზე თაგვის მარცხენა ღილაკით. ამის შედეგად ჩნდება ფანჯარა **Array Editor** (მასივის რედაქტორი), რომელშიც შეიძლება დავათვალიეროთ ცვლადები და შევცვალოთ მათი მნიშვნელობები. ეს ფანჯარა იხსნება აგრეთვე **openvar** ბრძანებითაც.



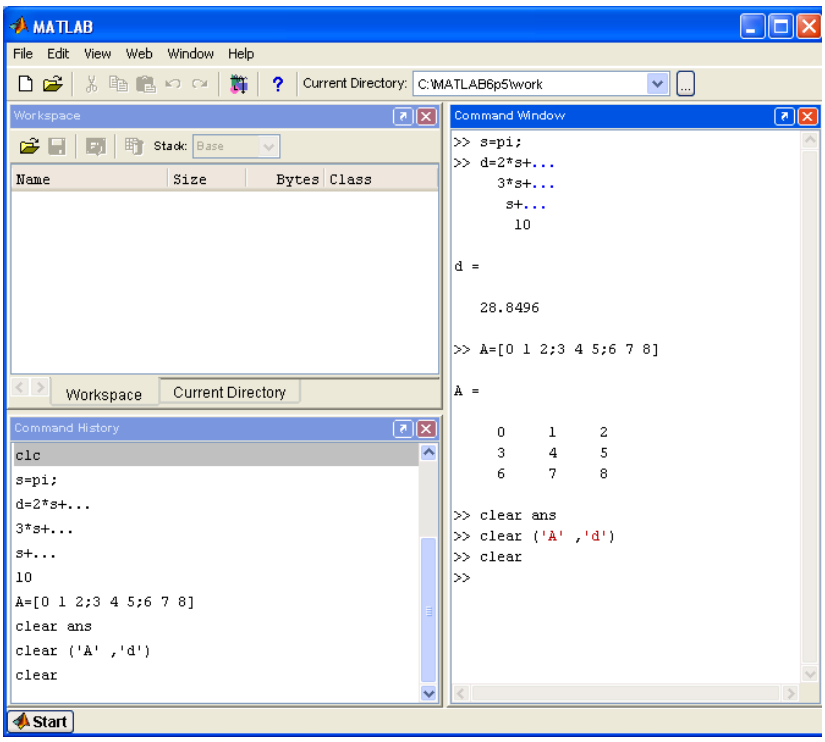
უნდა აღინიშნოს, რომ მუშაობის ეფექტურობა მცირდება იმის და მიხედვით თუ რამდენად იზრდება სამუშაო სივრცე. ამიტომ, თუ საჭიროება მათ შენახვას აღარ მოითხოვს, საჭიროა მათი წაშლა მეხსიერებიდან.

clear name1 name2…

name1, name2- წასაშლელი ცვლადების სახელები. ან როგორც

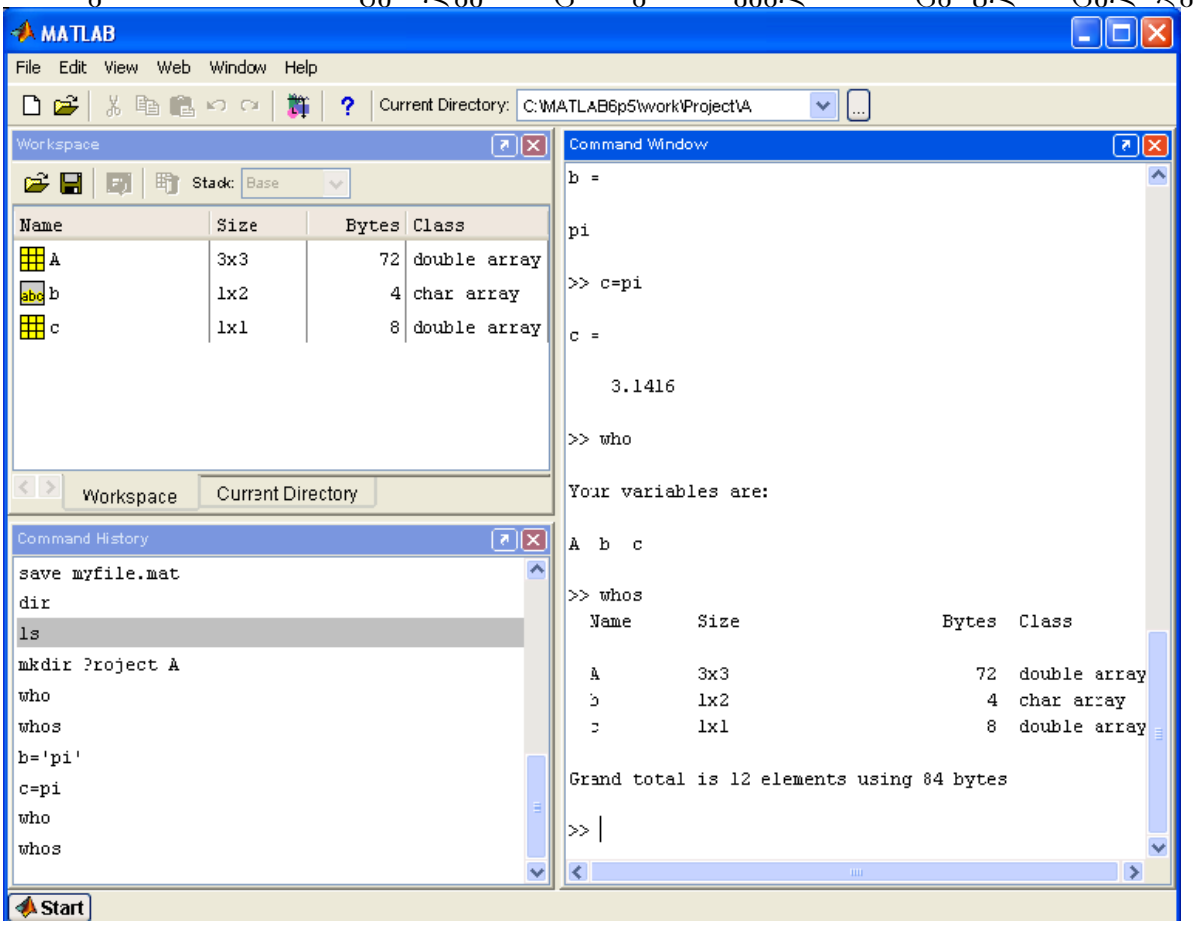
clear (‘name1’ ,‘name2’)

ყველა ცვლადის ერთბაშად წაშლა: **clear**



ყველა ცვლადთა სია **Workspace** ფანჯარაში აისახება სამუშაო ფანჯარაში ბრძანებით – **who.**

ბრძანება – **whos** გვაძლევს ცნობებს ყველა მოცემულ ცვლადებზე



ცვლადებს **A** და **c** მიენიჭათ რიცხვითი მნიშვნელობები და აღნიშნულნი არიან როგორც **‘double array’** (‘ორმაგი მასივი’), მოცემულ შემთხვევაში მასივს **c** აქვს

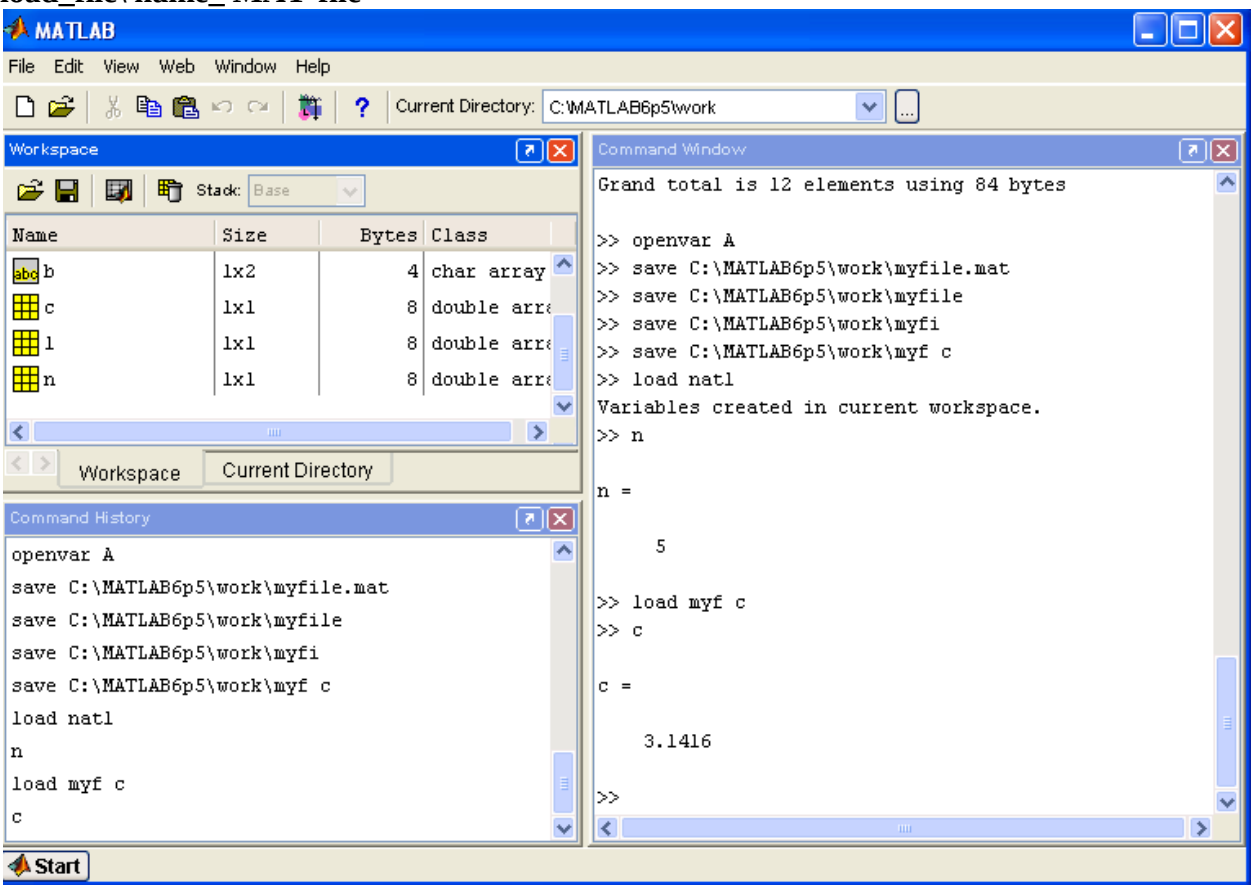
განზომილება **1x1**, ე. ი. არის სკალარი. ცვლადი **b** არის სიმბოლური **'char array'** ('სიმბოლური მასივი').

ბრძანება – **whos** არ გვაძლევს მოცემული ცვლადების მნიშვნელობებს. სამუშაოს დამთავრების შემდეგ ყველა გამოთვლილი ცვლადები რომ შევინახოთ ფაილში და შემდგომ გამოვიყენოთ უნდა შევასრულოთ:

File-Save Workspace as…, ან **save path_to_file\ name_ MAT-file**

ასეთ ფაილებს აქვთ გაფართოება mat, მათ უწოდებენ **'MAT- ფაილებს'.**

თავიდან რომ აღვადგინოთ ყველა ცვლადის მნიშვნელობა ვიყენებთ ბრძანებას: **load_file\ name_ MAT-file**



save C:\MATLAB6p5\work\myf c - აქ ფაილში სახელწოდებით **myf.mat** შენახულია მხოლოდ **c** ცვლადის მნიშვნელობა და აღდგება მხოლოდ **c** -ს მნიშვნელობა: **load myf c.**

load natl - აქ აღდგება ყველა ცვლადის მნიშვნელობა **natl.mat** ფაილში.

მიმდინარე კატალოგის სახელის დასადგენად შეასრულეთ ბრძანება: **cd**

მიმდინარე კატალოგის შეცვლა ხდება ბრძანებით: cd_ path_to_new catalog

მიმდინარე კატალოგში **(Current Directory)** შეგიძლიათ შექმნათ ახალი კატალოგი, ვთქვათ **ProjectA:**

mkdir ProjectA.

ფანჯარა Command History (ბრძანებათა ისტორია)

ფანჯარაში Command History არის ყველა ის ინფორმაცია რაც შეგიტანეთ **Command Window** ფანჯარაში, რაც გვაძლევს საშუალებას დავათვალიეროთ ყველა ის ბრძანება რაც შეგიტანეთ ადრე. თუ მათზე დავაწკაპუნებთ, ეს

ბრძანება დაუყონებლივ შესრულდება **Command Window** ფანჯარაში. ასევე, შეგვიძლია მოვნიშნოთ ბრძანებები **Command History** ფანჯარაში, მერე ავირჩიოთ ბრძანება **Copy** და ბრძანებით **Paste** ჩავწერთ **Command Window** ფანჯარაში და მზად იქნება რედაქტირებისათვის.

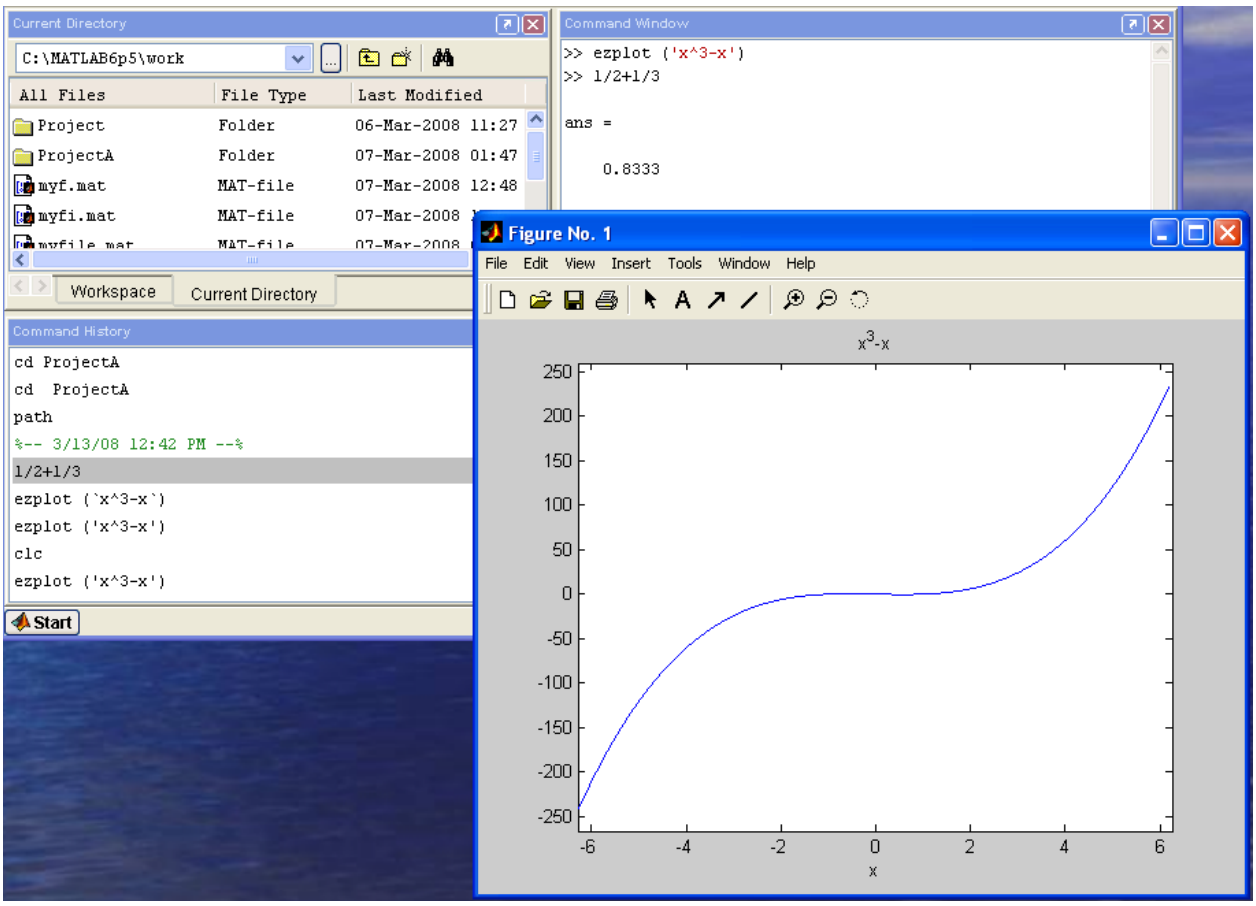
სამუშაოს დასრულება

სამუშაოს დასრულება ხდება
FILE-EXIT MATLAB
HELP.
HELP NAVIGATOR: CONTENTS, INDEX, SEARCH, DEMOS.
doc sin, help factor, help general
lookfor factor

პროგრამა "Matlab"ის საფუძვლები.

შეტანა გამოტანა.

ბრძანებები პროგრამა "Matlab"-ში შეგვყავს **Command Window** ფანჯარაში**.** პროგრამა აბრუნებს მნიშვნელობებს ორი გზით: ტექსტი და რიცხვითი შედეგები ჩნდება იმავე ფანჯარაში **Command Window,** მხოლოდ გრაფიკული შედეგი აღიბეჭდება სხვა ფანჯარაში



ასეთი ეკრანის მისაღებად აკრიფეთ: 1/2+1/3 და ezplot ('x^3-x').

არითმეტიკა.

პროგრამა “Matlab” -ის გამოყენება შეიძლება როგორც კალკულატორი.

ძირითადი ოპერატორები:

მინიჭების ოპერატორი =

მიმატება +

გამოკლება –

გამრავლება *

გაყოფა /

ახარისხება ^.

მაგალითი

9-(5+4)/2+6*3

ans =

22.5000

პროგრამა “Matlab” -ს გამოაქვს პასუხი და აღნიშნავს მნიშვნელობას ცვლადით ans. თუ მოგინდებათ ჩაატაროთ შემდგომი გამოთვლები მიღებულ პასუხთან შეგიძლიათ გამოიყენოთ ცვლადი **ans**, იმის მაგივრად რომ ხელახლა შემოიყვანოთ რიცხვითი მნიშვნელობა. მაგალითად, შეგიძლიათ გამოთვალოთ მიღებული პასუხის კვადრატის და მისი ფესვის ჯამი:

ans^2+sqrt(ans)

ans =

510.9934

იმისათვის, რომ ჩავატაროთ რთული გამოთვლები თქვენ შეგიძლიათ ცვლადებს მიანიჭოთ გამოთვლილი მნიშვნელობები, მაგალითად:

```
>> u=cos(10)
u =
  -0.8391
>> v=sin(10)
v =
  -0.5440
>> u^2+v^2
ans =
       1
```

აღსანიშნავია, რომ პროგრამა "Matlab" -ში ტრიგონომეტრიულ ფუნქციებში გამოიყენება რადიანები და არა გრადუსები.

ნამდვილ რიცხვთა წარმოდგენის ფორმატი

პროგრამა "Matlab" -ში რიცხვების წარმოსადგენად გამოიყენება ფორმატი მცოცავი წერტილით, რომელშიც ნებისმიერი რიცხვი მოიცემა მანტისისა და ხარისხის მაჩვენებლის საშუალებით და ჩაიწერება შემდეგი სახით, მაგალითად:
8.134838545e+10; 581e-1; 5.4; -312
სადაც **e** არის 10 -ის ხარისხის ფუძე.

მონაცემთა ასეთ ტიპს ეწოდება **Double.** მანტისისათვის და ხარისხის მაჩვენებლისათვის (მანქანურ დონეზე გამოიყენება ჩანაწერის ორობითი სისტემა) გამოიყოფა მეხსიერების 8 ბაიტი.

პროგრამა "Matlab" -ის მოდულით უდიდესი **realmax** და უმცირესი **realmin** ნამდვილი რიცხვებია:

```
>> realmax
ans =
   1.797693134862316e+308
>> realmin
ans =
   2.225073858507201e-308
```

რიცხვების წარმოსადგენად **Command Window** ფანჯარაში გამოიყენება ფორმატები: **short, long, rat, hex**.

გაჩუმების პრინციპის თანახმად პროგრამა "Matlab" -ი მონაცემს წარმოაჩენს 5 ნიშნის სიზუსტით. რომ წარმოჩინდეს მეტი ნიშნით, შეიტანეთ ბრძანება **format long**. მაშინ ყველა რიცხვით მნიშვნელობებს წერტილის შემდეგ ექნება 15 ნიშანი. რომ დავუბრუნდეთ ისევ 5 ნიშნა ჩვენებას, შეიყვანეთ ბრძანება **format short.** მაგალითი:

```
>> format long
>> v
v =
  -0.54402111088937
>> format short
>> v
v =
  -0.5440
```

ნამდვილი რიცხვის წარმოსაჩენად წილადის სახით გამოიყენება ფორმატი **rat**

```
>> r =1.3
```

```
>> format rat
>> r
r =
   13/10
```

ქვემოთ წარმოდგელია რიცხვების სხვადასხვა ფორმატები, რომლებიც გამოყენებულია x ვექტორის წარმოსადგენად:

```
>> x=[4/3 1.2345e-6];

>> format short
   1.3333    0.0000
>> format short e
  1.3333e+000  1.2345e-006
>> format short g
      1.3333  1.2345e-006
>> format long
  1.33333333333333   0.00000123450000
>> format long e
   1.333333333333333e+000    1.234500000000000e-006
>> format long g
        1.33333333333333             1.2345e-006
>> format bank
        1.33         0.00
>> format rat
    4/3         1/810045
>> format hex
  3ff5555555555555   3eb4b6231abfd271
```

კომლექსური რიცხვები

პროგრამა “Matlab” -ში გაჩუმების პრინციპის თანახმად ცვლადებს **i, j** ენიჭებათ მნიშვნელობა $\sqrt{-1}$. ხშირად ისინი გამოიყენებიან ინდექსებად და ენიჭებათ სხვა მნიშვნელობები. ამის თავიდან ასაცილებლად სამუშაოს დაწყების წინ სასურველია გამოვიყენოთ ბრძანება: **clear i, j;**

```
 >> clear i, j;
>> a=1+i
>> format short
>> a
a =
  1.0000 + 1.0000i.

>> abs(a)
ans =
   1.4142
>> real(a)
ans =
    1
>> imag(a)
ans =
    1
>> angle(a)
```

```
ans =
   0.7854
>> conj(a)
ans =
  1.0      - 1.0000i
```

შექმნილი პრობლემების გადაჭრა: წარმოქმნილი შეცდომები შეტანის დროს.

შეცდომის დაშვების შემთხვევაში პროგრამა “Matlab” -ს შეტანის სტრიქონში გამოაქვს ეკრანზე შენიშვნა მაგალითად:

```
>> 3u^2
??? 3u^2
   |
Error: Missing operator, comma, or semicolon.
```

შეცდომა მდგომარეობს გამრავლების ოპერატორის * არ არსებობაში. გამოსახულების სწორად ჩაწერის ფორმაა: **3*u^2.** მარკერი | განათავსდება იმ ადგილას სადაც უნდა იყოს შეცდომა. მაგრამ რეალური შეცდომა შეიძლება იყოს გამოსახულებაში ამ ნიშნის შემდგომ ან მანამდე.

დამწყები მომხმარებლის ხშირი შეცდომებია ფრჩხილებისა და გამრავლების ოპერატორის გამოტოვება.

გამოთვლების შეწყვეტა

თუ პროგრამა “Matlab” -ი ვეღარ ფუნქციონირებს ისე კარგად, ანუ ოპერაციის შესრულებას ანდომებს დიდ დროს, მაშინ შეგიძლიათ შეწყვიტოთ ოპერაცია კლავიშით: **Ctrl +C** .

ალგებრული და სიმბოლური გამოთვლები

თუ გამოიყენებთ ინსტრუმენტს **Simbolic Math Toolbox (**სიმბოლური მათემატიკის ინსტრუმენტი**)** თქვენ შეგიძლიათ განახორციელოთ ალგებრული და სიმბოლური გამოთვლები. იმისათვის, რომ დარწმუნდეთ რომ დაყენებული გაქვთ კომპიუტერში **Simbolic Math Toolbox ,** ბრძანების სტრიქონში შეიყვანეთ **help symbolic.**

სიმბოლური გამოთვლების ჩასატარებლად საჭიროა გამოვიყენოთ ბრძანება **syms.** იმისათვის, რომ ვაწარმოოთ ალგებრული და სიმბოლური გამოთვლები, ისეთი როგორიცაა მამრავლებად დაშლა, ან ალგებრული განტოლებების ამოხსნა. განვიხილოთ ბრძანებების ნუსხა:

```
>> syms x y
>> (x-y)*(x-y)^2
 ans =
 (x-y)^3

 >> expand(ans)
 ans =
```

x^3-3*x^2*y+3*x*y^2-y^3

>> factor(ans)
ans =
(x-y)^3

ბრძანება **expand** -ით ხდება გამოსახულების მამრავლებად დაშლა. ბრძანება **factor** -ით კი ხდება აღდგენა საწყის შეკუმშულ ფორმაში. ბრძანება **simplify** გამოიყენება ფორმულის გასამარტივებლად.

>> simplify((x^3-y^3)/(x-y))
ans =
x^2+x*y+y^2

პროგრამა "Matlab" -ს გააჩნია აგრეთვე ბრძანება **simple** რომელიც ხანდახან უფრო უკეთ მოქმედებს ბრძანება **simplify** -ზე. იხილეთ მაგალითი:

sin(x)*cos(y)+sin(y)*cos(x)
syms x y
>> simple(sin(x)*cos(y)+sin(y)*cos(x))
.
.
ans =
sin(x+y)

და

simplify(sin(x)*cos(y)+sin(y)*cos(x))
ans =
sin(x)*cos(y)+sin(y)*cos(x)

ერთი და იგივე ხარისხის მქონე ცვლადებთან მდგომი კოეფიციენტების დაჯგუფება ხდება ბრძანებით **collect.** მაგალითები:

გავამარტივეთ გამოსახულება $a\log x - x\log x - x$

>> syms a x
>> f=a*log(x)-log(x)*x-x
f =
a*log(x)-log(x)*x-x
>> collect(f,log(x))
ans =
(a-x)*log(x)-x

გავამარტივეთ გამოსახულება x*y+a*x*y+y*x^2-a*y*x^2+x+a*x

>> syms x y a
>> p=x*y+a*x*y+y*x^2-a*y*x^2+x+a*x
p =
x*y+a*x*y+y*x^2-a*y*x^2+x+a*x
>> collect(p,x)
ans =
(y-a*y)*x^2+(y+a*y+1+a)*x
>> collect(p,y)
ans =
(x^2-a*x^2+x+a*x)*y+x+a*x

შეტანა სიმბოლურ გამოსახულებებში.

როდესაც მოცემული გვაქვს სიმბოლური გამოსახულება რომელიც შეიცავს სიმბოლურ ცვლადს და გვინდა ის შევცვალოთ სხვა რაიმე ცვლადით, რიცხვით ან გამოსახულებით ვიყენებთ ბრძანებას **subs.** მაგალითად, მოცემული გვაქვს სიმბოლური გამოსახულება **w=u^2-v^2,** რომელიც შეიცავს სიმბოლურ ცვლადს **u.** მაშინ ბრძანებით **subs(w,u,2)** ცვლადი **u** შეიცვლება **2-**ით.

აი რამდენიმე მაგალითი:

```
d=1, syms u v
d =
    1
>> w=u^2-v^2
w =
 u^2-v^2
 >> subs(w,u,2)
 ans =
 4-v^2
 >> subs(w,v,d)
 ans =
 u^2-1
 >> subs(w,v,u+v)
 ans =
 u^2-(u+v)^2
 >> simplify(ans)
 ans =
 -2*u*v-v^2
```

სიმბოლური გამოსახულებები, ცვლადის სიზუსტე და ზუსტი არითმეტიკა.

როგორც აღვნიშნეთ, პროგრამა “Matlab” -ი იყენებს თავის გამოთვლებში არითმეტიკას მცოცავი წერტილით. თუ გამოიყენებთ **Simbolic Math Toolbox,** შეგიძლიათ ჩაატაროთ ზუსტი არითმეტიკული გამოთვლები სიმბოლური გამოსახულების საშუალებით. განიხილეთ მაგალითი:

```
>> cos(pi/2)
ans =
  6.1232e-017
```

პასუხი მიღებულია მცოცავი წერტილის ფორმატში და აღნიშნავს 6.1232e-017. მაგრამ ვიცით, რომ **cos(pi/2)** უდრის ნოლს. ეს ხდება იმის გამო, რომ კონსტანტა **pi** მოიცემა პროგრამა “Matlab” -ში მიახლოებით π -თან სიზუსტით 15 ნიშნამდე. იმისათვის რომ მივიღოთ ზუსტი მნიშვნელობა, უნდა შევქმნად ზუსტი სიმბოლური წარმოდგენა $\left(\frac{\pi}{2}\right)$. ამისთვის შემოვიტანოთ ბრძანება **sym('pi/2').** ეხლა განვიხილოთ კოსინუსი $\left(\frac{\pi}{2}\right)$ სიმბოლური წარმოდგენისა:

```
>> cos(sym('pi/2'))
ans =
0
```

ეს კი არის მოსალოდნელი პასუხი.

sym('pi/2') ბრძანებაში, ბრჭყალები - რომელშიც არის მოთავსებული **pi/2,** ქმნიან სიმბოლურ სტრიქონს, რომელიც შეიცავს **pi/2** სიმბოლოებს და პროგრამა "Matlab" –ში ვეღარ გამოითვლება როგორც რიცხვი მცოცავი წერტილით.

ბრძანება **sym** -ს სტრიქონი გადაყავს სიმბოლურ გამოსახულებაში. ბრძანება **syms x** ეკვივალენტურია ბრძანებისა **x=sym('x').** მაგალითი: შევკრიბოდ სიმბოლურ ფორმაში **1/2+1/3**

```
>> sym('1/2')+sym('1/3')
ans =
5/6
```

შეგიძლიათ აგრეთვე მოახდინოთ გამოთვლები მოცემული სიზუსტით **vpa** ბრძანების საშუალებით.

```
>> vpa('sqrt(2)',50)
ans =
1.4142135623730950488016887242096980785696718753769
```

გამოვთვალეთ $\sqrt{2}$ სიზუსტით 50 ნიშნამდე მძიმის შემდეგ.

მაგალითები:

```
>> 3^45
ans =
  2.9543e+021
>> vpa(3^45)
 ans =
29543127065508336107 52.
 >> vpa('3^45')
 ans =
 29543127065508336986 43.
```

პირველი გამოსახულება იძლევა მიახლოებით შედეგს მცოცავი წერტილით, რადგან პროგრამა "Matlab"-ი აწარმოებს გამოთვლებს ხარისხიანი გამოსახულებისა 16 ნიშნამდე, ამიტომ მეორე პასუხი იქნება ზუსტი პირველი 16 ნიშნის ფარგლებში მძიმის შემდეგ, მესამე გამოსახულება იძლევა ზუსტ პასუხს.

სიმბოლური ცვლადები, გამოსახულებები და მატრიცები.

როგორც უკვე ავღნიშნეთ, სიმბოლური გამოსახულება იქმნება ბრძანებით **sym(´სიმბოლური გამოსახულება´).**

სიმბოლური გამოსახულების შესაქმნელად, რომელიც შეიცავს გამოსახულებას ax^2+bx+c, საჭიროა შესრულდეს შემდეგი ბრძანება:

f=sym('a*x^2+b*x+c')

უნდა აღინიშნოს, რომ აქ შემოტანილი გამოსახულება განხილული იქნება როგორც ერთი ცვლადი. იმისათვის, რომ გვქონდეს საშუალება, რომ ვცვალოთ ax^2+bx+c გამოსახულებაში შემავალი კოეფიციენტების და ცვლადების მნიშვნელობები, საჭიროა შესრულდეს შემდეგი ბრძანებები:

```
>> syms a b c x
>> f=sym('a*x^2+b*x+c')
 f =
 a*x^2+b*x+c
```

აბსტრაქტული ფუნქციის **f(x)** შესაქმნელად საჭიროა შესრულდეს შემდეგი ბრძანება:

```
>> clear
>> f=sym('f(x)')
 f =
 f(x)
```

შექმნილი **f(x)** აბსტრაქტული ფუნქციის გამოყენებით შეიძლება შევქმნათ ახალი ფუნქცია **df**, ამისათვის საჭიროა შესრულდეს შემდეგი ბრძანება:

```
 >> df=(subs(f,'x','x+h')-f)/'h'
 df =
 (f(x+h)-f(x))/h
```

ზემოთ შექმნილი ფუნქცია $df(x)$ განიხილება როგორც ერთეული ცვლადი. იმისათვის რომ გვქონდეს საშუალება ვცვალოთ **x,h** მნიშვნელობები, საჭიროა აღიწეროს ცვლადები როგორც სიმბოლური

```
>> syms x h
df=(subs(f,x,x+h)-f)/h
 df =
 (f(x+h)-f(x))/h
```

აქ $df(x)$ ფუნქციის გარეგანი სახე რჩება შეუცვლელი. ქვევით გამოთვლილია $df(x)$ როცა $h=1$

```
>> subs(df,h,1)
 ans =
 f(x+1)-f(x)
```

როცა $x=1$

```
>> subs(df,x,1)
 ans =
 (f(1+h)-f(1))/h
```

ეხლა შევქმნად სიმბოლური მატრიცა

```
 >> syms a b c
>> A=[a b c; b c a; c a b]
 A =
 [ a, b, c]
[ b, c, a]
[ c, a, b]
```

სიმბოლური დიფერენცირებადობა.

f(x) ფუნქციის გაწარმოება რომ მოვახდინოთ, საჭიროა 1) წარმოვადგინოთ გამოსახულება, რომელიც აღწერს ფუნქციას, 2) გამოვიყენოთ **diff** ფუნქცია**.** მაგალითი: $f=x^3$

```
>> syms x
>> f=x^3
 f =
 x^3
 >> diff(f)
 ans =
 3*x^2
```

შეიძლება ასეც ჩაიწეროს

```
>> diff(f,x)
 ans =
 3*x^2
```

f(x) ფუნქციის მეორე რიგის წარმოებულია

```
>> diff(f,2)
 ans =
 6*x
```

მაგალითი: გამოთვალეთ $f = \sin(ax)$ წარმოებული a პარამეტრით.

```
>> syms a x
>> f=sin(a*x)
 f =

sin(a*x)
 >> diff(f,a)
 ans =
 cos(a*x)*x
```

მაგალითი: გამოთვალეთ წარმოებული $\begin{pmatrix} \cos(ax) & \sin(ax) \\ -\sin(ax) & \cos(ax) \end{pmatrix}$ მატრიცის

```
>> syms a x
>> A=[cos(a*x),sin(a*x);-sin(a*x),cos(a*x)]
 A =
 [  cos(a*x),  sin(a*x)]
[ -sin(a*x),  cos(a*x)]
 >> diff(A)
 ans =
 [ -sin(a*x)*a,  cos(a*x)*a]
[ -cos(a*x)*a, -sin(a*x)*a]
```

სიმბოლური ინტეგრირება.

პროგრამა **MATLAB**-მა შეუძლია აწარმოოს განსაზრვრული და განუსაზღვრელი ინტეგრალების გამოთვლები. სიმბოლური ინტეგრირებისათვის გამოიყენება ფუნქია **int,** რომელსაც გააჩნია შემდეგი სინტაქსი

$\text{int}(f)$

$\text{int}(f,[u]),$

$\text{int}(f,[u,a,b]),$

სადაც **f** –სიმბოლური ინტეგრალქვეშა ფუნქციაა, **u**-ინტეგრირების ცვლადი, ***a***-ინტეგრირების ქვედა ზღვარი, ***b***- ინტეგრირების ზედა ზღვარი.

განვიხილოთ განუსაზღვრელი ინტეგრალი

```
>> int('x^2','x')
 ans =
 1/3*x^3
```

შეიძლება ასეც ჩაიწეროს

```
>> syms x,
>> int(x^2,x)
 ans =
 1/3*x^3
```
შეიძლება ასეც ჩაიწეროს
```
>> int(x^2)
ans =
 1/3*x^3
```
განვიხილოთ განსაზღვრული ინტეგრალი
```
>> syms x,int(asin(x),0,1)
 ans =
 1/2*pi-1
```
გამოთვალეთ $\int \frac{dx}{a^2+(bx)^2}$
```
>> syms a b x
>> int(1/(a^2+(b*x)^2))
 ans =
 1/a/b*atan(b*x/a)
```
გამოთვალეთ $\int_0^{a/b} \frac{dx}{a^2+(bx)^2}$
```
>> syms a b x
>> int(1/(a^2+(b*x)^2),0,a/b)
 ans =
 1/4*pi/a/b
```

ორმაგი ინტეგრალის დათვლა.

ვთქვათ გამოსათვლელია ინტეგრალი
$\int_0^{\pi}\int_0^{\sin x}(x^2+y^2)dydx$
```
>> syms x y,int(int(x^2+y^2,y,0,sin(x)),0,pi)
 ans =
 pi^2-32/9
```

ზღვრების გამოთვლა

მარჯვენა და მარცხენა ზღვრების დასათვლელად შეიძლება გამოიყენოთ ბრძანება $\lim it$. მაგალითად $\lim_{x\to 0}\sin(x)/x$
```
>> syms x, limit(sin(x)/x,x,0)
 ans =
 1
```
ცალმხრივი ზღვრების დასათვლელად გამოიყენეთ პარამეტრები **'right'** და **'left'**

```
>> syms x, limit(sin(x)/x,x,0,'left')
 ans =
 -1

>> syms x, limit(sin(x)/x,x,0,'right')
 ans =
 1
```

უსასრულო ზღვრის დათვლა შეიძლება **Inf** სიმბოლოს გამოყენებით.

გამოთვალეთ $\lim_{x\to\infty}\frac{x^4+x^2-3}{3x^4-\log x}$:

```
>> limit((x^4+x^2-3)/(3*x^4-log(x)),x,inf)
 ans =
 1/3
```

გამოთვალეთ $\lim_{n\to\infty}(1+\frac{x}{n})^n$:

```
>> syms x n
>> limit((1+x/n)^n,n,inf)
 ans =
 exp(x)
```

გამოთვალეთ $\lim_{x\to 1}\frac{1}{1-x}$:

```
>> syms x
>> limit(1/(1-x),x,1)
 ans =
 NaN
```

აქ ცვლადი **NaN** აღნიშნავს რომ ზღვარი ფუნქციისა $\frac{1}{(1-x)}$ წერტილში **x=1** არ არსებობს..

ჯამი და ნამრავლი.

სასრული რიცხვითი ჯამები და ნამრავლები შეიძლება ადვილად გამოითვალოს თუ გამოიყენებთ პროგრამა **MATLAB**-ის ვექტორულ შესაძლებლობას და **sum** და **prod** ბრძანებებს. მაგალითად

```
>> x=1:7
x =
    1    2    3    4    5    6    7
>> sum(x)
ans =
   28
>> prod(x)
ans =
       5040
```

symsum ბრძანებით შეგიძლიათ განსაზღვროთ სასრული და უსასრულო ჯამები სიმბოლურ ფორმაში. მაგალითად $\sum_{k=1}^{n}\left(\frac{1}{k}-\frac{1}{k+1}\right)$

```
>> syms k n;symsum(1/k-(1/(k+1)), 1, n)
 ans =
 -1/(n+1)+1
```

გამოთვალეთ $\sum_{k=1}^{\infty}\frac{1}{k^4}$:

```
>> syms x k
>> s1=symsum(1/k^4,1,inf)

s1 =
 1/90*pi^4
```

გამოთვალეთ $\sum_{k=1}^{10}\frac{1}{k^4}$:

```
>> syms x k
>> s1=symsum(1/k^4,1,10)
 s1 =
43635917056897/40327580160000
```

ტეილორის მწკრივი

შესაძლებელია გამოიყენოთ ბრძანება **taylor** იმისათვის რომ შექმნათ ტეილორის მწკრივის გამოსახულება მოცემული ხარისხით მოცემულ წერტილში. მაგალითად, იმისათვის, რომ $\sin x$ ფუნქციისათვის შევქმნად ტეილორის მრავალწევრი $x=0$ წერტილში ხარისხად 9, გამოიყენეთ ბრძანებები:

```
>> syms x; taylor(sin(x),x,10)
 ans =
x-1/6*x^3+1/120*x^5-1/5040*x^7+1/362880*x^9
```

- **ვექტორები და მატრიცები**.
- **ფუნქციები.** ჩაშენებული ფუნქციები.

ვექტორები და მატრიცები

ვექტორი

პროგრამა “Matlab” –ში თქვენ შეგიძლიათ შემოიტანოთ ნებისმიერი სიგრძის **ვექტორი,** სადაც რომლი კომპონენტები გამოიყოფებიან მძიმით ან პრობელით და კვადრატულ ფრჩხილებში განთავსდებიან. მაგალითად

```
>> Z=[2,4,6,8]
Z =
   2    4    6    8
```

```
>> Y=[4 -3 5 -2 1]
Y =
   4   -3    5   -2    1
```
თუ გინდათ შექმნათ ვექტორი კომპონენტებით 1 დან 9 მნიშვნელობით, ისე რომ ყოველი რიცხვი არ შევიტანოდ შესაძლებელია ასეთ ნაირად
```
>> X=1:9
X =
     1     2     3     4     5     6     7     8     9
```
აქ ბიჯია 1 ტოლი. თუ გვინდა რომ ბიჯი სხვა რაიმე რიცხვი იყოს მაგალითად 2 ტოლი , მაშინ გვექნება
```
>> X=1:2:9
X =
     1     3     5     7     9.
```
ბიჯი შესაძლებელია იყოს წილადი და უარყოფითიც, მაგალითად -0.2
```
>> X=1:-0.2:0.1
X =
    1.0000    0.8000    0.6000    0.4000    0.2000
```

X ვექტორის ელემენტები შეიძლება გამოვყოთ ასეც **X(3), X(5)**

```
>> X(3)
ans =
   0.6000
>> X(5)
ans =
   0.2000
```
იმისათვის რომ ვექტორი სტრიქონის ფორმიდან გადავიყვანოთ ვექტორულ ფორმაში გამოიყვანება სიმბოლოს (') გამოყენებით ვექტორის შემდეგ
```
>> X'
ans =
   1.0000
   0.8000
   0.6000
   0.4000
   0.2000
```
შეგიძლიათ ვექტორებზე შეასრულოთ მატემატიკური ოპერაციები. მაგალითად X ვექტორის ელემენტების კვადრატში აყვანა ხდება ბრძანებით .^
```
>> X.^2
ans =
   1.0000    0.6400    0.3600    0.1600    0.0400
```
ანალოგიურად თუ გვინდა ვექტორების ელემენტი ელემენტზე გავამრავლოთ ან გავყოთ შემოგვაქვს ბრძანებები .*, ./ (აქ ვექტორები ან სვეტად ან სტრიქონად არის წარმოდგენილი)
```
>> y=[4 -3 5 -2 1]
y =
   4   -3    5   -2    1
>> X.*y
ans =
   4.0000   -2.4000    3.0000   -0.8000    0.2000
>> X./y
ans =
```

0.2500 -0.2667 0.1200 -0.2000 0.2000.

თუ გვინდა ვექტორების ელემენტი ელემენტზე გავამრავლოთ ისე რომ ერთი სტრიქონის და მეორე სვეტის სახით არის წარმოდგენილი მაშინ წერტილის გარეშე ვწერთ გამრავლებას, მხოლოდ*.

```
>> y=[4 -3 5 -2 1]
y =
    4   -3    5   -2    1
>> x=[4; -3; 5 ;-2 ;1]
x =
    4
   -3
    5
   -2
    1
>> x*y
ans =
   16  -12   20   -8    4
  -12    9  -15    6   -3
   20  -15   25  -10    5
   -8    6  -10    4   -2
    4   -3    5   -2    1
>> y*x
ans =
   55
```

ვექტორის სიგრძის დადგენა ხდება ბრძანებით **length.** მაგალითად

```
>> length(y)
ans =
    5
>> whos y
  Name      Size                Bytes  Class
  y         1x5                    40  double array
Grand total is 5 elements using 40 bytes.
```

მატრიცა

მატრიცა იწერება კვადრატული ფრჩხილების საშუალებით სადაც სტრიქონის თვითეული ელემენტი მძიმით ან ჰარით გამოიყოფა, ხოლო ყოველი ახალი სტრიქონები გამოიყოფა წერტილ მძიმით.

პროგრამა **MATLAB**-ი უპირატესობას ანიჭებს მატრიცის სვეტებთან მუშაობას. მაგალითად

```
>> A=[1 2 3 4;5 6 7 8;9,10,11,12]
A =
    1    2    3    4
    5    6    7    8
    9   10   11   12
```

A მატრიცის ჯამის, მინიმუმის და მაქსიმუმის პოვნა ხთება სვეტების მიმართ

```
>> sum(A)
ans =
```

```
   15    18    21    24
>> max(A)
ans =
    9    10    11    12
>> min(A)
ans =
    1     2     3     4
```
შეგვიძლია გამოვყოთ სტრიქონი ან სვეტი მატრიცაში სიმბოლოს **:** მეშვეობით. მაგალითად
```
>> A(:,1)
ans =
    1
    5
    9
>> A(2,:)
ans =
    5     6     7     8
```
ინდექსი

i სტრიქონში და **j** სვეტში მდგომი **ელემენტი აღინიშნება A(i,j).** მაგალითად
```
>> A=[1 2 3; 4 5 6; 8 9 10]
A =
    1     2     3
    4     5     6
    8     9    10
>> A(1,3)
ans =
    3
```
ორი წერტილიანი ოპერატორის გამოყენებით შეიძლება გამოყოთ ელემენტები. მაგალითად
```
>> A(1:2,3)
ans =
    3
    6
```
>> მატრიცის დიაგონალის ელემენტები
```
>> diag(A)
ans =
    1
    5
   10
>> sum(diag(A))
ans =
   16
```
სტრიქონების და სვეტების ამოჭრა
```
>> X=A
X =
    1     2     3
    4     5     6
    8     9    10

>> X(:,2)=[]
X =
```

```
   1    3
   4    6
   8   10
>>
X =
   1    3
   4    6
   8   10

>> X(1,:)=[]

X =

   4    6
   8   10
```

გამოსახულებაში იღებთ შეცდომას

```
>> X(1,1)=[]
??? Indexed empty matrix assignment is not allowed.
```

მაგრამ თუ გამოიყენებთ ერთ ინდექს, მაშინ შეცდომას აღარ მიიღებთ მაგალითად

```
>> A
A =
   1    2    3
   4    5    6
   8    9   10
>> A(2:2:9)=[]
A =
   1    8    5    3   10.
>> A+1
ans =

   2    3    4
   5    6    7
   9   10   11
>> A*3
ans =
   3    6    9
  12   15   18
  24   27   30
```

მატრიცების გენერირება

eye - ერთეულოვანი მატრიცაა.

rand -შემთხვევითი ელემენტების თანაბარი განაწილება

randn-შემთხვევითი ელემენტების ნორმალური განაწილება

```
>> eye(3)
ans =
   1    0    0
   0    1    0
   0    0    1
>> Z=zeros(2,4)
```

```
Z =
    0     0     0     0
    0     0     0     0
>> F=5*ones(3,3)
F =
    5     5     5
    5     5     5
    5     5     5

>> N=fix(10*rand(1,10))

N =

    9     2     6     4     8     7     4     0     8     4
>> R=randn(4,4)
R =
   -0.4326   -1.1465    0.3273   -0.5883
   -1.6656    1.1909    0.1746    2.1832
    0.1253    1.1892   -0.1867   -0.1364
    0.2877   -0.0376    0.7258    0.1139

>> R=randn(4,4)
R =
    1.0668    0.2944   -0.6918   -1.4410
    0.0593   -1.3362    0.8580    0.5711
   -0.0956    0.7143    1.2540   -0.3999
   -0.8323    1.6236   -1.5937    0.6900

>> R=randn(4,4)
R =

    0.8156    1.1908   -1.6041   -0.8051
    0.7119   -1.2025    0.2573    0.5287
    1.2902   -0.0198   -1.0565    0.2193
    0.6686   -0.1567    1.4151   -0.9219
>>მაგიური მატრიცა
> B=magic(4)
B =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> sum(B)
ans =
    34    34    34    34
>> sum(diag(B))
ans =
    34
```

მატრიცის გაერთიანება

```
>> B=[A+1 A+3; A+45 A*6]
B =
```

```
   2    3    4    4    5    6
   5    6    7    7    8    9
   9   10   11   11   12   13
  46   47   48    6   12   18
  49   50   51   24   30   36
  53   54   55   48   54   60
```

თუ გვინდა **გავამრავლოთ მატრიცა B ვექტორზე Z,** ამისათვის უნდა მოვახთინოდ ვექტორის ტრანპოზიცია **Z'** და მასზე გამრავლება :

```
>> Z=[1 3 5]
Z =
   1    3    5
B =
   1    2    3
   5    6    7
   9   10   12
>> B*Z'
ans =
   22
   58
   99
```

B მატრიცის **B':**

```
>> B'
ans =
   1    5    9
   2    6   10
   3    7   12
```

B მატრიცის **დეტერმინანტის** გამოსათვლელად გამოიყენება ბრძანება **det**

```
>> det(B)
ans =
   -4
```

B მატრიცის **შებრუნება** ხდება ბრძანებით **inv(B):**

```
>> inv(B)
ans =

  -0.5000   -1.5000    1.0000
  -0.7500    3.7500   -2.0000
   1.0000   -2.0000    1.0000
```

A-მატრიცის რანგის გამოსათვლელად ვიყენებთ rank(A). მაგალითად

A=[1 3 5; 1 7 9; 4 -3 9]

```
 >> rank(A)
ans =
   3
```

თუ დიდი განზომილების ვექტორები ან მატრიცები გვაქვს, ანდა რეზულტატი არ გვინდა გამოვიდეს ეკრანზე, მაშინ ბრძანების შეტანის ბოლოს წერტილ მძიმეს დაუსვავთ ; მაგალითად

X=-1:0.1:2;

კონკატენაცია ვერტიკალური იწერება1, ხოლო ვერტიკალური 2

```
x=[[1;3;5],[2;4;6]]
x =
    1    2
    3    4
    5    6
>> x=cat(2,[1;3;5],[2;4;6])
x =
    1    2
    3    4
    5    6
>> x=[[1 2];[3 4];[5 6]]
x =
    1    2
    3    4
    5    6
>> x=cat(1, [1 2];[3 4];[5 6])
x =
    1    2
    3    4
    5    6
```

თუ საჭიროა ამოვხსნათ ალგებრულ განტოლებათათა სისტემა **Ay=b,** სადაც **A** – მოცემული კვადრატული მატრიცაა, **b**- ვექტორ -სვეტია , ამისათვის საკმარისია გამოვითვალოთ გამოსახულება **A\b.**

მაგალითად

```
>> A=[1 3 5; 1 7 9; 4 -3 9]
A =
    1    3    5
    1    7    9
    4   -3    9
>> b=[1 3  9]'
b =
    1
    3
    9
>> A\b
ans =
   -6.7500
   -2.6250
    3.1250
```

იმისათვის რომ სიმბოლური ფორმით განვსაზღვროთ ამოხსნა, თუ **A** და **b** – სიმბოლურები არიან, უნდა განვსაზღვროთ შემდეგნაირად **x=sym(A)\b.**

მაგალითად

```
>> syms a1 a2 a3 a4,   A=[a1 a2; a3 a4]
 A =
 [ a1, a2]
[ a3, a4]
 >> syms b1 b2 , b=[b1; b2]
```

```
 b =
 [ b1]
[ b2]
 >> x=sym(A)\b
 x =
 [ -(a4*b1-b2*a2)/(a3*a2-a1*a4)]
[  (a3*b1-a1*b2)/(a3*a2-a1*a4)]
```

მომხმარებლის მიერ მოცემული ფუნქციები

განვიხილოთ ორი ხერხი საკუთარი ფუნქციის მოცემისა პროგრამა ‘ Matlab’-ში. პირველი ხერხი იყენებს ბრძანება inline, ხოლო მეორე გამოიყენებს ოპერატორს @ მისათვის რომ შექმნას ეგრეთწოდებული ’ანონიმური ფუნქცია’. ფუნქციის წარმოდგენა აგრეთვე მოხდეს ეგრეთ წოდებული M- ფაილების სახით. მაგალითში ნაჩვენებია $\mathbf{f(x)=x^2}$ მოცემა ამ ბრძანებების გამოყენებით :

```
>> f1=inline('x^2','x')
f1 =
     Inline function:
     f1(x) = x^2
>>  f1(4)
ans =
    16
>> f=@(x)x^2
>>  f(4)
ans =
    16
```

რადგან უმეტეს ფუნქციებს შეუძლიათ პროგრამა ‘ Matlab’-ში მოქმედებდეს როგორც ვექტორებზე ასევე სკალარებზეც, ამიტომ წერტილი ჩასვით ოპერატორების ***, /, ^** წინ. იმისათვის რომ მივიღოთ ვექტორული ვერსია $\mathbf{f(x)=x^2}$ შემოიყვანეთ სტრიქონი

```
>> f=@(x)x.^2
      |

>> f1=inline('x.^2','x')
f1 =
     Inline function:
     f1(x) = x.^2
>> f1(1:5)
ans =
     1     4     9    16    25
```

ფუნქციის წარმოდგენა შეიძლება ორი ან მეტი ცვლადებით. მაგალითად

```
>> g=@(x,y)x^2+y^2;g(1,2);
>> g1=inline('x^2+y^2','x','y');g1(1,2)
ans =
     5
```

აგრეთვე შეგიძლიათ გამოთვალოთ ვექტორები თუ გამოვიყენებთ წერტილს ოპერატორების ***, /, ^** წინ.

```
>> g=@(x,y)x.^2+y.^2; g([1 2],[3 4])
>> g1=inline('x.^2+y.^2','x','y');g1([1 2],[3 4])
```

```
ans =
   10    20
```

ელემენტარული ფუნქციები.

ზოგიერთი ძირითადი ელემენტარული მათემატიკური ფუნქციები ჩაშენებული **MATLAB** ში.

sin, asin ,cos, acos,tan, atan, exp,

log ითვლის მასივის ელემენტების ნატურალურ ლოგარითმს , **log10** ითვლის მასივის ელემენტების ათობით ლოგარითმს,

pow2: Y= pow2(X) არის მაჩვენებლიანი ფუნქცია 2 ის ფუძით.

sqrt: y= sqrt(X) - ითვლის მასივის ელემენტების კვადრატულ ფესვს.

fix: fix(A) ამრგვალებს **A** მასივის ნამდვილ ელემენტებს 0-კენ

```
>> A=[11 0.007; 19978 -0.9989]
A =
  1.0e+004 *
    0.0011    0.0000
    1.9978   -0.0001
>> fix(A)s =
        11         0
     19978         0
```

floor: floor (A) ამრგვალებს **A** მასივის ნამდვილ ელემენტებს **--inf** კენ

```
>> floor(A)
ans =
        11         0
     19978        -1
```

ceil: ceil (A) ამრგვალებს **A** მასივის ნამდვილ ელემენტებს **inf** კენ

```
>> ceil(A)
ans =
        11         1
     19978         0
```

round: round(A) ამრგვალებს **A** მასივის ნამდვილ ელემენტებს უახლოეს მთელამდე.

```
>> round(A)
ans =
        11         0
     19978        -1
```

mod: M=mod(X,Y) აბრუნებს ნაშთს მიღებულს გაყოფისას **X :Y**

```
>> X=7, Y=3
X =
     7
Y =
     3
>> M=mod(X,Y)
M =
     1
```

rem: M= rem (X,Y) აბრუნებს მთელ ნაწილს მიღებულს გაყოფისას **Y: X**

```
>> Y=55,X=7, rem(X,Y)
Y =
   55
```

X =

7

ans =

7

sign: sign(x) აბრუნებს -1, თუ x<0, 0 თუ x=0 და 1 თუ x>0.

მიმართების ოპერაციები და ლოგიკური ოპერაციები რიცხვებზე

მიმართების ოპერატორები

<	ნაკლები
<=	ნაკლები ან ტოლი
>	მეტი
>=	მეტი ან ტოლი
==	უდრის
~ =	არ უდრის

მიმართების ოპერატორების გამოყენებისას დარდება ორი მასივი ერთი და იგივე განზომილების ან დარდება მასივი სკალართან. ამ მეორე შემთხვევაში სკალარი დარდება მასივის ყველა ელემენტს და შედეგად ვიღებთ იმავე ზომის მასივს.

>> A=1:9,B=8-A

A =

1 2 3 4 5 6 7 8 9

B =

7 6 5 4 3 2 1 0 -1

>> tf1=A<=4

tf1 =

1 1 1 1 0 0 0 0 0

>> tf2=A>B

tf2 =

0 0 0 0 1 1 1 1 1

>> tf3=(A==B)

tf3 =

0 0 0 1 0 0 0 0 0

>> tf4=B-(A>2)

tf4 =

7 6 4 3 2 1 0 -1 -2

>>

შევნიშნოთ რომ = და== სხვადასხვა მნიშვნელობა აქვს. == ადარებს ორ ცვლადს და აბრუნებს მნიშვნელობას 1 სადაც ისინი ტოლებია და ნულს სადაც ისინი ტოლები არ არიან, ხოლო = მიანიჭებს გამოსატან ოპერატორს მნიშვნელობას.

ნულზე გაყოფის აცილება

განვიხილოთ შემდეგი:

>> x=(-3:3)/3

x =

Columns 1 through 5

-1.0000 -0.6667 -0.3333 0 0.3333

Columns 6 through 7

0.6667 1.0000

>> sin(x)./x

ans =

0.8415 0.9276 0.9816 NaN 0.9816 0.9276 0.8415

აქ მეოთხე ელემენტი არის ***NaN*** ***(***ნიშნავს ***Not a Number)*** , რადგან მეოთხე ელემენტი ***x*** ში უდრის ***0*** და ***sin(0)./0*** არ არის განსაზღვრული. ამის ასაცილებლად ***0-*** ლი ***xX-*** ში შევცვალოთ სპეციალური რიცხვით ***eps,*** რომელიც მიახლოებით ***2.2X10*** $^{-16}$

>> x=(-3:3)/3

x =

-1.0000 -0.6667 -0.3333 0 0.3333 0.6667 1.0000

>> x=x+(x==0)*eps

x =

-1.0000 -0.6667 -0.3333 0.0000 0.3333 0.6667 1.0000

```
>> sin(x)./x

ans =
  0.8415   0.9276   0.9816   1.0000   0.9816   0.9276   0.8415
>>
```

ლოგიკური ოპერატორები (იხ. **help relop**)

&	და
|	ან
~	არა

მეოთხე ლოგიკური ოპერატორი წარმოდგენილია შემდეგი ფუნქციით: განვიხილოთ ლოგიკური ფუნქცია **xor.** ეს ფუნქცია ორ არგუმენტიანია და განიხილავს მათ მიმართ ოპერაციას 'ან –ის გამორიცხვა', ღებულობს ერთის ტოლ მნიშვნელობას (´ჭეშმარიტი´) მხოლოდ მაშინ, როდესაც ერთერთი რიცხვითი არგუმენტი ჭეშმარიტია (´არ ურდრის ნოლს´), ხოლო მეორე მცდარი (´ურდრის ნოლს´).

A	B	~A	A|B	A&B	xor(A,B)
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0

პრიორიტეტი მიმართებების ოპერატორების იწყება ~, &, |. ფრჩხილები ცვლიან პრიორიტეტს.
მაგალითები:

```
>> A=1:9

A =
   1   2   3   4   5   6   7   8   9

>> tf1=A>4

tf1 =
   0   0   0   0   1   1   1   1   1

>> tf2=~(A>4)
tf2 =

   1   1   1   1   0   0   0   0   0

>> tf3=(A>2)&(A<6)
tf3 =
   0   0   1   1   1   0   0   0   0
```

```
>> tf4=xor((A>2),(A<6))

tf4 =
   1    1    0    0    0    1    1    1    1

>>
>> a=1; b=0;
>> xor(a,b)
ans =
   1
```
იმ შემთხვევაში როდესაც ორივე არგუმენტი `ჭეშმარიტია´ ან ორივე `მცდარი´, მოცემული ფუნქცია **xor** ურდრის ნოლს
```
>> a=1; b=1;
>> xor(a,b)
ans =
   0
```
მაგალითები. განვიხილოთ:
```
>> A=[1 1 1;2 2 2;3 3 3]
A =
   1    1    1
   2    2    2
   3    3    3
>> B=[0 0 0;7 7 7 ;1 2 3]
B =
   0    0    0
   7    7    7
   1    2    3
>> A<=B
ans =
   0    0    0
   1    1    1
   0    0    1
>> xor(A,B)
ans =
   1    1    1
   0    0    0
   0    0    0
>> v=[1 2 3 4 0]
v =
   1    2    3    4    0
>> ~v
ans =
   0    0    0    0    1
```
ოპერატორი ´ ლოგიკური ან ` იღებს ნოლს თუ ორივე ოპერანდი **0** – ია, და იღებს ერთს თუ ეს ასე არ არის.
```
>> 2|3
ans =
   1
```
ლოგიკური გამოსახულება გამოთვლისას ღებულობს მნიშვნელობას 0 (თუ მცდარია) და 1 (თუ ჭეშმარიტია)

```
>> 2>3
ans =
    0
```
ლოგიკური ოპერატორი მოქმედებს თვითეულ ელემენტზე
```
>> [2 3]<[3 2]

ans =
    1     0
>> x=-2:2; x>=0
ans =
    0     0     1     1     1
```
ვთქვათ
```
>> a=3;b=5;c=7;
>> (a<b)+(c~=b)+(b==a)
ans =
    2
```

ლოგიკური ფუნქციები

ლოგიკური ფუნქციები მოქმედებენ სკალარზე, ვექტორებზე, და მატრიცებზე. განვიხილოთ ზოგიერთი მათგანი:

any(x) აბრუნებს სკალარს მნიშვნელობით 1 (**true**) თუ ნებისმიერი (რომელიმე, ერთი მაინც) ელემენტი **x** ვექტორისარისა არის ნული, ხოლო თუ ყველა ელემენტი **x** ვექტორის არისა არის ნული აბრუნებს სკალარს მნიშვნელობით 0(**false**).

```
>> x=0:3:9

x =

    0    3    6    9

>> any(x)

ans =

    1

>> y=[0 0 0 0]

y =

    0    0    0    0

>> any(y)

ans =

    0
```

მატრიცის შემთხვევაში, აბრუნებს ვექტორს (row) 1- ბითა(true) და 0-ებით (false), იმის და მიხედვით თუ მისს სვეტში ყველა ელემენტი ნულია (მაშინ ვექტორში ჩაიწერება 0) თუ ერთი მაინც განსხვავდება ნულისაგან (მაშინ ვექტორში ჩაიწერება 1)

```
>> A=[1,0,3;1,0,0;4,0,0]

A =

   1   0   3
   1   0   0
   4   0   0

>> any(A)

ans =

   1   0   1
```

all(x) აბრუნებს სკალარს მნიშვნელობით 1 (true) თუ ყველა ელემენტი x ვექტორისა არისა არის არანულოვანი, ხოლო თუ რომელიმე ელემენტი x ვექტორის არის ნული აბრუნებს სკალარს მნიშვნელობით 0(false).

მაგალითები:

```
>> y=[5 3 4 7]
y =
   5   3   4   7

>> all(y)
ans =
   1
x=0:3:9
x =
   0   3   6   9

>> all(x)
ans =
   0
```

მატრიცის შემთხვევაში, აბრუნებს ვექტორს (row) 1- ბითა(true) და 0- ებით (false), იმის და მიხედვით თუ მისს სვეტში თუ ერთი ელემენტი მაინც ნულია (მაშინ ვექტორში ჩაიწერება 0) ყველა ელემენტი განსხვავდება ნულისაგან (მაშინ ვექტორში ჩაიწერება 1)

```
A =

   1   0   3
   1   0   0
   4   0   0

>> all(A)
```

```
ans =
   1   0   0
```
მეორე მაგალითი
```
>> A=[1,0,3;1,0,5;4,0,0]

A =
   1   0   3
   1   0   5
   4   0   0

>> all(A)
ans =
   1   0   0
```
find(x) აბრუნებს ვექტორს, რომლის ელემენტები წარმოადგენენ **x** ვექტორის არანულოვანი ელემენტების ინდექსებს.

```
y=[0 3 0 7]
y =
   0   3   0   7
>> find(y)
ans =
   2   4

y=[10 3 4 7]
y =
  10   3   4   7
>>  find(y)
ans =
   1   2   3   4
y =
   0   0   0   0
>> find(y)
ans =
  Empty matrix: 1-by-0
```

მატრიცის შემთხვევაში, აბრუნებს ვექტორს (column) რომლის ელემენტები წარმოადგენენ A მატრიცის სვეტების არანულოვანი ელემენტების ინდექსებს.

```
>> A=[1,0,3;1,0,5;4,0,0]
A =
   1   0   3
   1   0   5
   4   0   0

>> find(A)
ans =

   1
   2
   3
```

```
   7
   8
>> A=[1,0,3;1,0,5;4,1,1]
A =

   1   0   3
   1   0   5
   4   1   1
>> find(A)
ans =
   1
   2
   3
   6
   7
   8
   9
```

>>მიღებული ვექტორების სიგრძე დამოკიდებულია A მატრიცაში არანულოვანი ელემენტების რაოდენობაზე.
კიდევ ერთი მაგალითი:

```
A =

   1   0   3
   1   0   5
   4   1   1
>> find(A>3)
ans =
   3
   8

y=[5 3 4 7]
y =
   5   3   4   7
>> find(y>3)
ans =
   1   3   4

y=[5 3 4 7]
y =
   5   3   4   7
>> find(y<4)
ans =
   2

y=[5 3 4 7]
y =
   5   3   4   7
>> find(y<3)
ans =
  Empty matrix: 1-by-0
```

isnan(x) აბრუნებს მასივს 0 და1 -ებით, იმის და მიხედვით თუ რომელი **x** ელემენტები **იქნება NaN.,** მასივში **- ჩაიწერება 1-ები,** და **ჩაიწერება** 0-ბი, თუ **x** ვექტორის ელემენტები არ არიან **NaN.**

```
>> y=[5 3 4 NaN]
y =
   5   3   4  NaN
>> isnan(y)
ans =
   0   0   0   1

>> y=[5 3 4 7]
y =
   5   3   4   7
>> isnan(y)
ans =
   0   0   0   0
```

isfinite(x) აბრუნებს მასივს 0 და1 -ებით, იმის და მიხედვით თუ სად არის **x** ვექტორის ელემენტები სასრულები, ჩაიწერება **1-ები,** და **იქნება** 0-ბი, თუ **x** ვექტორის ელემენტები არ არიან სასრულები.

```
>> y=[pi NaN Inf -Inf]
y =
  3.1416    NaN    Inf    -Inf
>> isfinite(y)
ans =
   1   0   0   0
```

isinf(x) აბრუნებს მასივს 0 და1 -ებით, იმის და მიხედვით თუ სად არის **x** ვექტორის ელემენტები **Inf ან -Inf**, ჩაიწერება **1-ები,** და ჩაიწერება 0-ბი, თუ **x** ვექტორის ელემენტები არ არიან **Inf ან -Inf.**

```
>> y=[pi NaN Inf -Inf]
y =
  3.1416    NaN    Inf    -Inf
>> isinf(y)
ans =
   0   0   1   1

>> y=[3 7 Inf 9 -Inf]
y =
   3   7  Inf   9  -Inf
>> isinf(y)
ans =
   0   0   1   0   1
```

isempty(x) აბრუნებს 1 თუ **x** არის ცარიელი მასივი და 0 თუ **x** არ არის ცარიელი მასივი

```
>> y=[3 7 7 9]
y =
   3   7   7   9
```

```
>> isempty(y)
ans =
   0

>> y=[]
y =
   []
>> isempty(y)
ans =
   1
```

განტოლებების ამოხსნა

ერთი ცვლადის მქონე განტოლებების ამოხსნა შესაძლებელია ბრძანებებით **solve** და **fzero.**

```
>> solve('x^2-2*x-4=0')
 ans =
 [ 5^(1/2)+1]
[ 1-5^(1/2)]
```

აქ განტოლებას რომელსაც ვხსნით მოცემულია ბრჭყალებში, ამიტომ ამოხსნა მოიცემა ზუსტად (სიმბოლური) $1+\sqrt{5}$. იმისათვის რომ მიიღოთ რიცხვითი ამოხსნები შეიყვანეთ **double (ans)** ან **vpa (ans)** რომ უფრო მეტ ნიშნით წარმოადგინოთ . გამოსახულება ბრძანება solve ში შეიძლება იყოს სიმბოლურადაც წარმოდგენილი. მაგალითად, ამოვხსნათ განტოლება $x^2-3x=-7$, სინტაქსისი ამოხსნისა იქნება შემდეგი

```
>> syms x
>> solve(x^2-3*x+7)
 ans =
 [ 3/2+1/2*i*19^(1/2)]
[ 3/2-1/2*i*19^(1/2)]
```

აქ ამოხსნა არის ზუსტი (სიმბოლური) $(3+(\sqrt{19}i))/2$.

ბრძანება **solve** შეიძლება ამოვხსათ პოლინომიალური და სხვა ტიპის განტოლები. თუ განტოლებების რაოდენობა ნაკლებია უცნობების რაოდენობაზე, მაშინ საჭიროა განსაზღვროთ (როგორც სტრიქონები) რომელი ცვლადია (ცვლადები) გამოსათვლელი. მაგალითად ამოვხსანად $2x-\log y=1$ ჯერ **y** ცვლადის მიმართ **x** ის პირობით და პირიქით

```
 >> solve('2*x-log(y)=1','y')
 ans =
 exp(2*x-1)

 >> solve('2*x-log(y)=1','x')
 ans =
 1/2*log(y)+1/2
```

ანალოგიურად შეიძლება განისაზღვროს ერთზე მეტი განტოლება. მაგალითად:

```
 >> [x,y]=solve('x^2-y=2','y-2*x=5')
 x =
 [ 1+2*2^(1/2)]
[ 1-2*2^(1/2)]
  y =
```

[7+4*2^(1/2)]
[7-4*2^(1/2)]
ამ სისტემას აქვს ორი ამონახსნი.პირველი ამონახსნია x და y პირველი მნისვნელობები, ანუ **x(1), y(1)**

```
>> x(1),y(1)
ans =
1+2*2^(1/2)
ans =
7+4*2^(1/2)
>> x(2),y(2)
ans =
1-2*2^(1/2)
 ans =
7-4*2^(1/2)
```

მეორე ამოხსნა შეიძლება მივიღოთ თუ შევიყვანოთ **x(2), y(2).**
ყურადღება მიაქციეთ იმას რომ ბრძანება **solve** ამოხსნის გამოტანა მოვითხოვეთ ვექტორულ ფორმაში **[x,y].** თუ გამოიყენებთ ბრძანება **solve -** ს განტოლებათა სისტემაში და არ მიუთითებთ გამოტანას ვექტორული ფორმით, ამ შემთხვევაში პროგრამა **MATLAB** -ი ავტომატურად არ წარმოსახავს ამოხსნის მნიშვნელობებს:

```
>> sol=solve('x^2-y=2','y-2*x=5')
sol =
   x: [2x1 sym]
   y: [2x1 sym]
```

იმისათვის რომ დავინახოთ x და y მნიშვნელობების ვექტორები , შეიყვანეთ sol.x და sol.y. იმისათვის რომ დავინახოთ ცალკეული მნიშვნელობები, შეიყვანეთ **sol.x(1)** და **sol.y(1),** და ასე შემდეგ.
ზოგიერთი განტოლების ამოხსნა სიმბოლურად შეუძლებელია და ასეთ შემთხვევაში ბრძანება **solve** ცდილობს განსაზღვროს რიცხვითი პასუხი.

```
>> solve('sin(x)=2-x')
ans =
1.1060601577062719106167372970301

>> solve('exp(-x)=sin(x)')
ans =
-2.012775662931511163336070699O971+2.703074511590962213931614804426 5*i
```

იგი წარმოადგენს კომპლექსურ რიცხვს. მაგრამ არსებობენ აგრეთვე ნამდვილი რიცხვები.

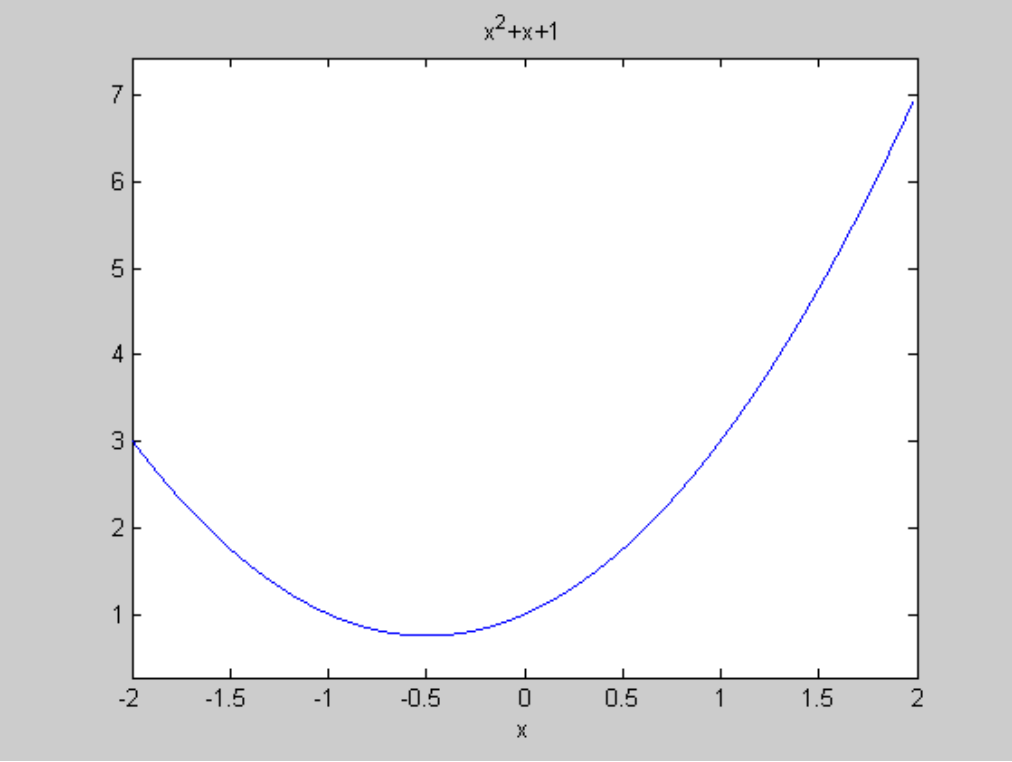
```
>> h=@(x)exp(-x)-sin(x)
    h=inline (´exp(-x)-sin(x)´, ´x´)
.>> fzero(h,0.5)
>> h=inline ('exp(-x)-sin(x)','x')
h =
    Inline function:
    h(x) = exp(-x)-sin(x)
>> fzero(h,0.5)
ans =
   0.5885
```

გრაფიკა

გრაფიკების აგება ezplot ბრძანებით

უმარტივესი ხერხი იმისათვის რომ ავაგოთ ერთი ცვლადის ფუნქციის გრაფიკი გამოიყენეთ ბრძანება **ezplot**, რომელსაც შეუძლია იმუშაოს სტრიქონთან, სიმბოლურ გამოსახულებასთან ან ანონიმურ ფუნქციასთან, რომელიც წარმოადგენს ფუნქციას გრაფიკული სახით წარმოსადგენად. მაგალითად იმისათვის რომ ავაგოთ გრაფიკი ფუნქციისა x^2+x+1 ინტერვალში -2 დან 2-მდე (გამოიყენეთ რა სტრიქონული ფორმა **ezplot** ბრძანებისა), შემოიყვანეთ შემდეგი

>> ezplot ('x^2+x+1',[-2,2])



გამოიყენეთ რა სიმბოლური გამოსახულება , შეგიძლიათ წარმოსახოთ ნახაზი, თუ შეიყვანთ ბრძანების სტრიქონში შემდეგს,

>> syms x

>> ezplot ('x^2+x+1',[-2,2])

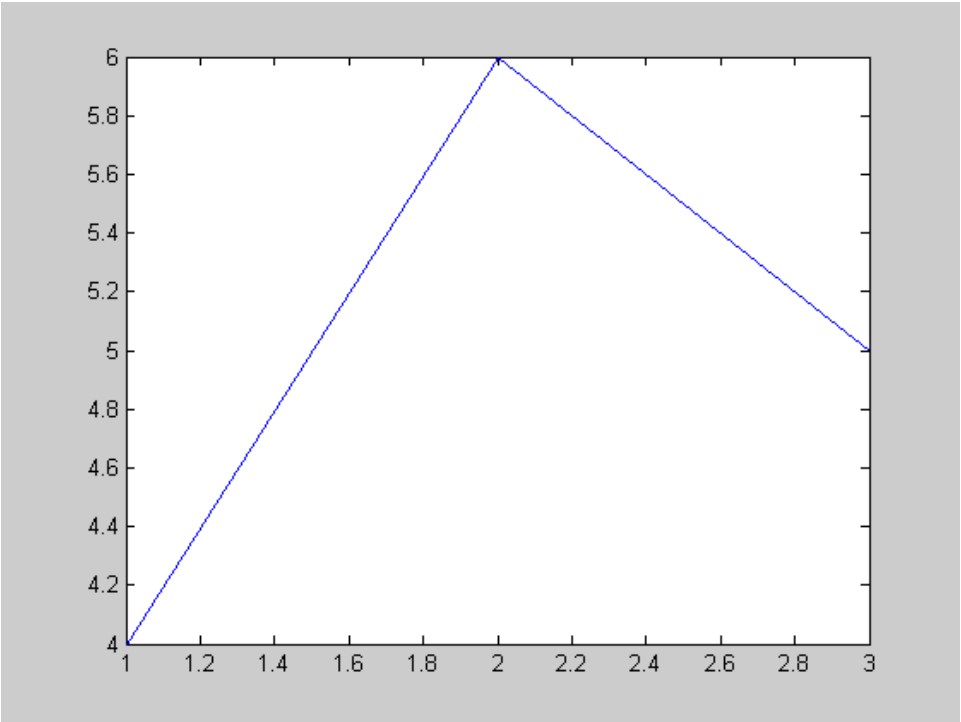
შეგიძლიათ გამოიყენოთ ანონიმური ფუნქცია როგორც ბრძანება ezplot -ის არგუმენტი, მაგალითად:

>> ezplot(@(x)x.^2+x+1,[-2,2])

გრაფიკების აგება ბრძანებით plot

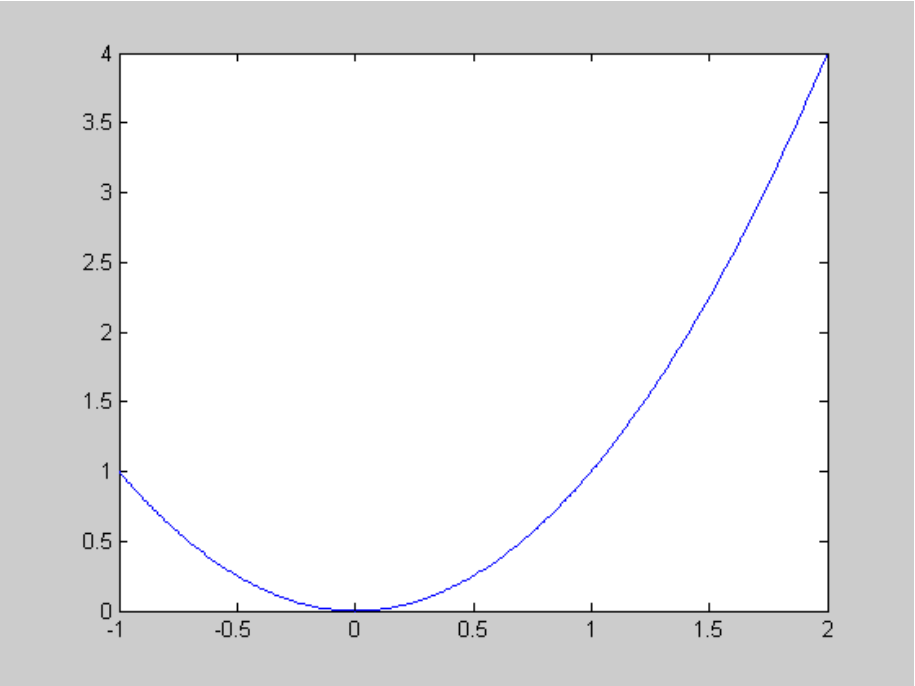
ბრძანება **plot** მუშაობს რიცხვითი მონაცემთა ვექტორებთან. ბრძანების სინტაქსია: **plot (X,Y),** სადაც **X** და **Y** არიან ვექტორები ერთნაირი სიგრძე აქვს. მაგალითად

>> X=[1 2 3]; Y=[4 6 5]; plot(X,Y)



იმისათვის რომ ავაგოთ გრაფიკი x^2 ფუნქციისა ინტერვალში -1 დან 2, საჭიროა შეიქმნას X ჩამონათვალი x მნიშვნელობებიდან, და მერე შეიყვანეთ **plot (X,X.^2).** აქ აუცილებელია წერტილით X .^ რადგან თითოეული ელემენტი ახარისხთება კვადრატში და აგრეთვე საჭიროა საკმარისი წერტილების რაოდენობა რადგან გრაფიკს ჩონდეს მისაღები სახე (გლუვი წირი და არა ტეხილი). ჩვენ ვიღებთ ბიჯს 0.01. შეიყვანეთ

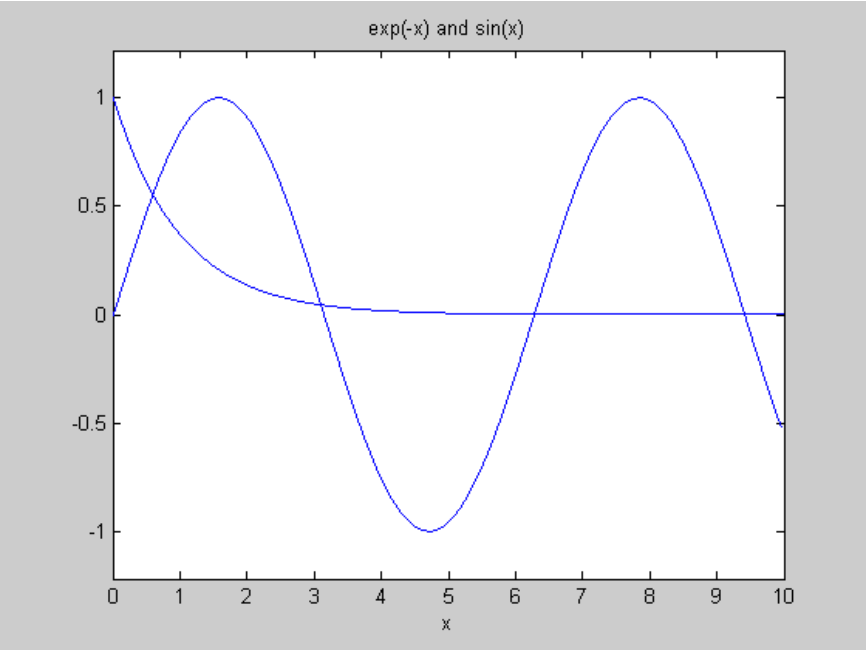
>> X=-1:0.01:2; plot(X,X.^2)



რამდენიმე მრუდის აგება

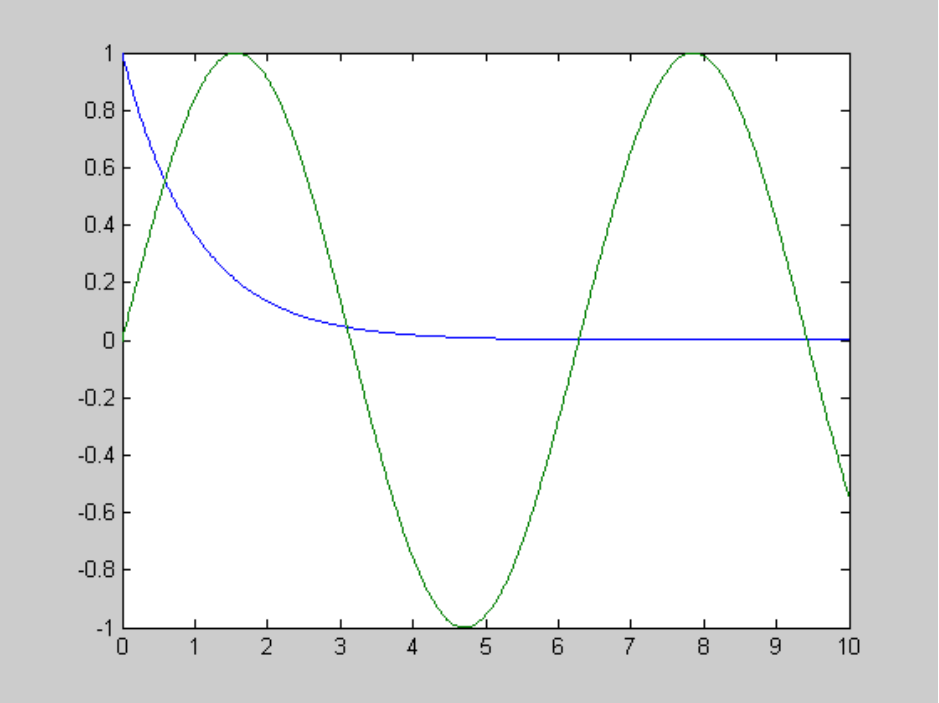
ყოველთვის, როდესაც სრულდება ნახაზი, ძველი ისლე ა და მის ნაცვლად იხატება ახალი. თუ გსურთ მოახთინოდ ერთ ნახაზზე მეორე ნახაზის ზედ დადება, გამოყენეთ ბრძანება **hold on.** ეს ბრძანებით შეინახება ძველი ნახაზი და მასზე ნებისმიერი ახალი ნახაზები. ეს შესრულდება მანამ სანამ არ შეიყვანთ ბრძანებას **hold off.** ქვემოთ მოყვანილია მაგალითი **ezplot** ბრძანების გამოყენებით

```
>> ezplot('exp(-x)', [0 10])
>> hold on
>> ezplot('sin(x)', [0 10])
>> hold off
>> title 'exp(-x) and sin(x)'
```

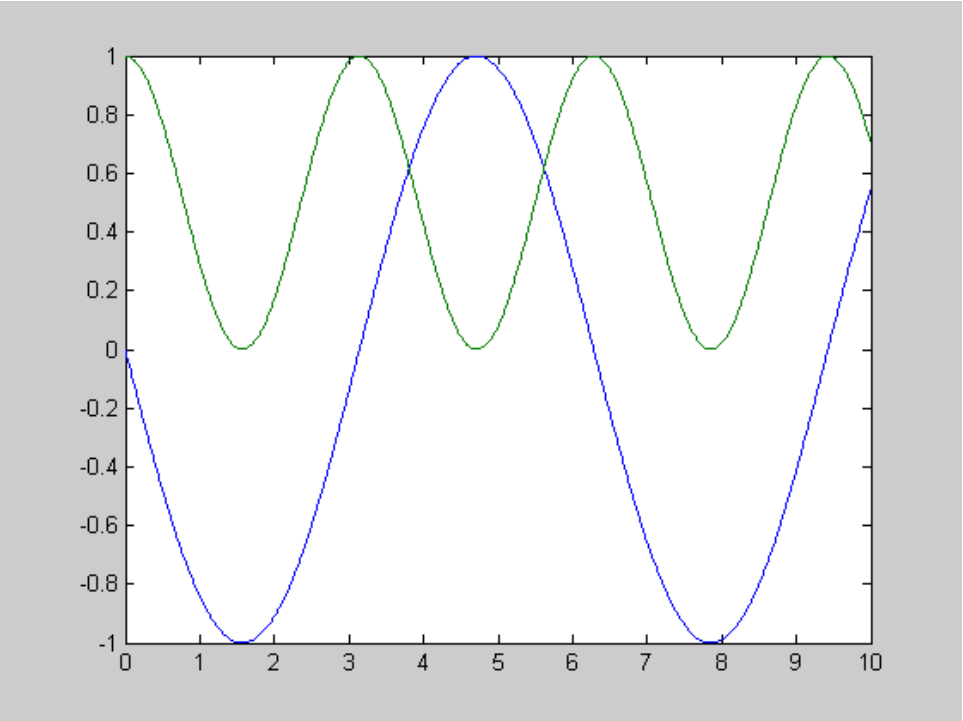


ბრძანება **plot** ით შეიძლება დახაზოთ ერთდროულად რამოდენიმე წირის გრაფიკი

X=0:0.01:10; plot(X,exp(-X),X,sin(X))



>> X=0:0.01:10; plot(X,sin(-X),X,cos(X).^2)



ყურადღება მიაქციეთ რომ x კოორდინატთა ღერძის ვექტორი მოიცემა თითოჯერ ყოველი ფუნქციისათვის , რომელიც იქნება გამოტანილი გრაფიკის სახით.

M-ფაილები

მ-ფაილები გვაძლევენ საშუალებას შევინახოთ ერთ ფაილში პროგრამა **MATLAB** ის ბევრი ბრძანებები. არსებობს ორი სხვა დასხვა ტიპი მ-ფაილებისა: **მ-ფაილი-სცენარები** და **მ-ფაილები-ფუნქციები.**

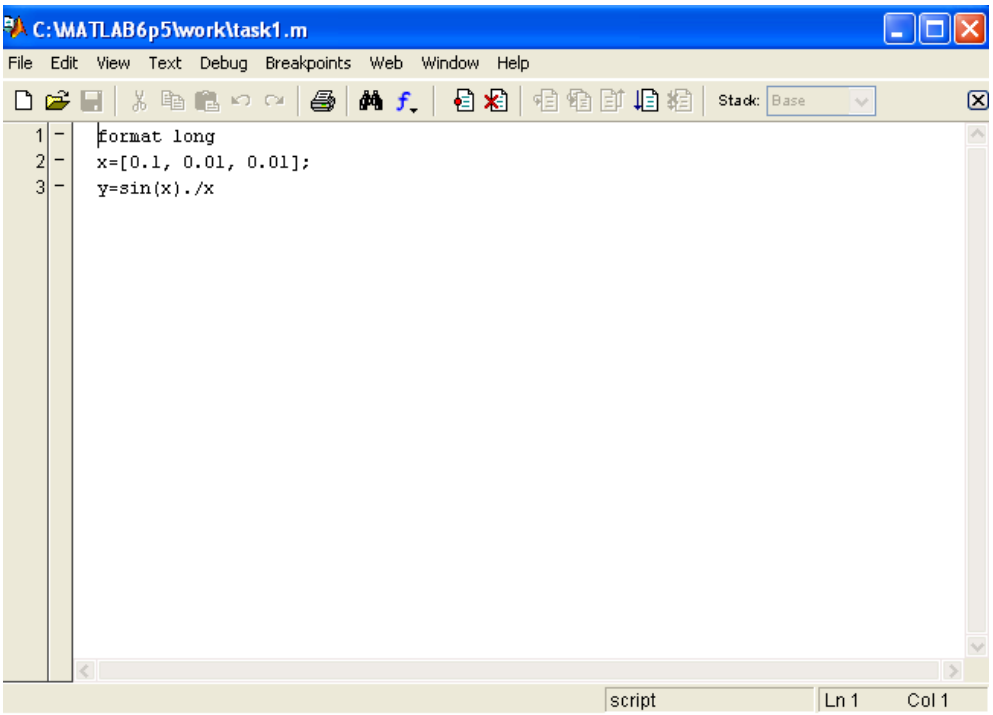
მ-ფაილები წარმოადგენენ თავის მხრივ ჩვეულებრივ ტექსტურ ფაილებს, რომლებიც მოიცავენ პროგრამა **MATLAB** ის ბრძანებებს.შეგიძლიათ შექმნათ და მათი მოდიფიცირება მოახთინოდ ნებისმიერი ტექსტური რედაქტორის მეშვეობით ან ტექსტური პროცესორი, რომელსაც შეუძლია შეინახოს ფაილები როგორც ჩვეულებრივი ტექსტი **SCII** ფორმატში (ეს ისეთი რედაქტორებია როგორც ან **Wordpad** სისტემა **Word** ში და emacs და **vi** სისტემა **UNIX** ში).

M-ფაილი-სცენარები

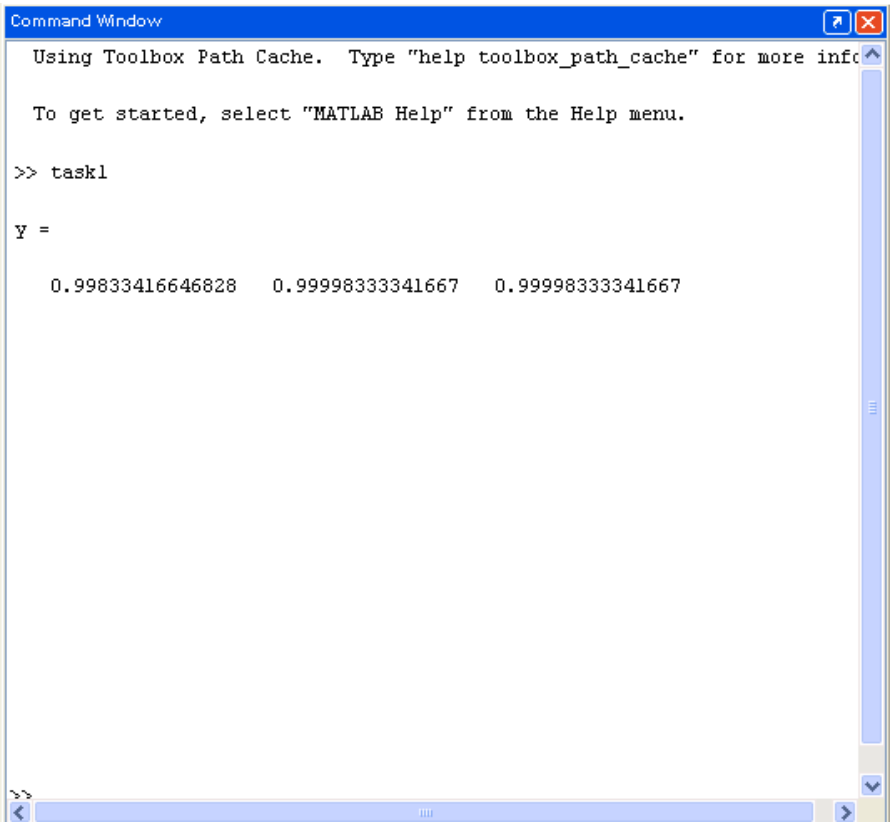
მ-ფაილი-სცენარები შეიცავენ პროგრამა **MATLAB** ის ბრძანებათა მიმდევრობას, რომლებიც გაიშვება გარკვეული მიმდევრობით. შექმენით ფაილი, რომელიც შეიცავს შემდეგ სტრიქონებს:

```
>> format long
>> x=[0.1,0.01,0.001];
>> y=sin(x)./x
```

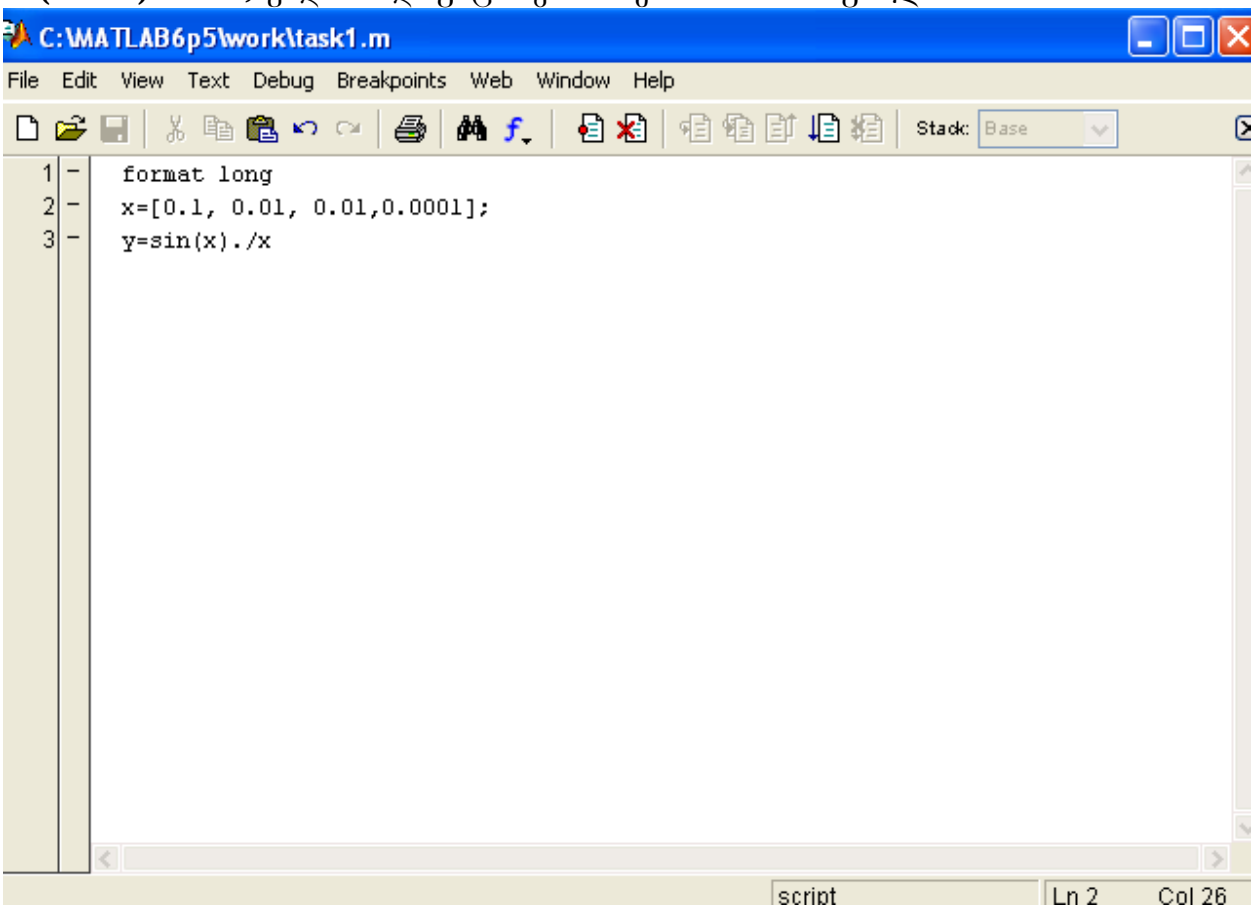
დაუშვათ რომ ეს ფაილი შევინახეთ სახელით **task1.m** თქვენს მიმდინარე კატალოგში ან რომელიმე კატალოგში. შეგიძლია მიანიჭოთ ნებისმიერი სახელი რაც გნებავთ, მხოლოდ გაფართოვება აუცილებლად უნდა იყოს **.m** .



შეიძლება პროგრამა **MATLAB-** ით ამ სცენარის გაშვება თუ ბრძანებათა ფანჯარაში **Command Window** -ში შეიყვანთ ბრძანებას **task1** გაფართოვების გარეშე. რეზულტატი (და არა ბრძანებები, რომლის მეშვეობითაც იქნა ეს შედეგი მიღებული) იქნება გამოსახული ფანჯარაში **Command Window.**



ბრძანებების მიმდევრობა ადვილად შეიძლება შეიცვალოს მ–ფაილის **task1.m** მოდიფიკაციის საშუალებით. მაგალითად, თუ გინდათ გამოთვალოთ **sin(0.0001)/0.0001,** უნდა მოდიფიცირება მოვახთინოთ მ–ფაილის **task1.m:**

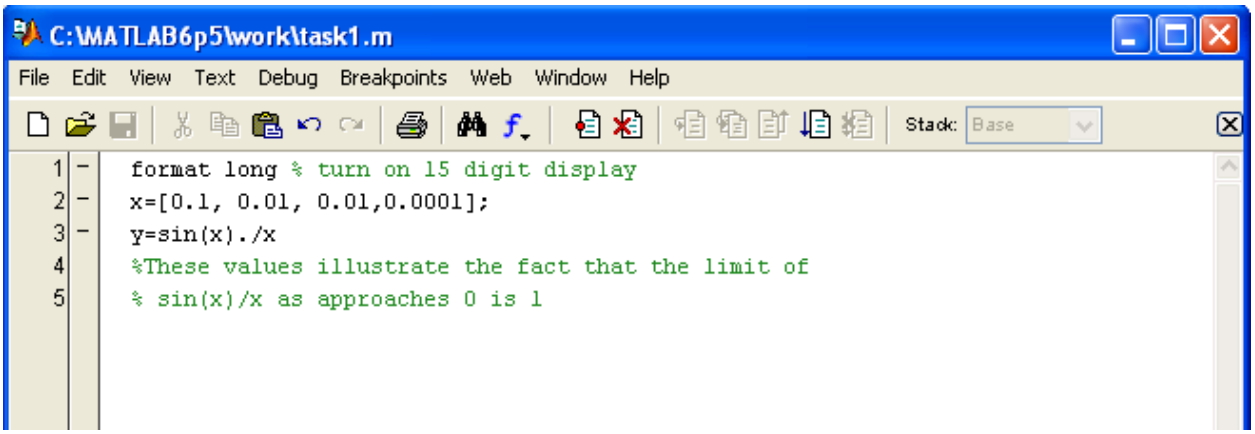


და მერე თავიდან გაუშვათ მოდიფიცირებული სცენარი ფანჯარაში **Command Window:**

```
>> task1
y =
  0.99833416646828   0.99998333341667   0.99998333341667   0.99999999833333
 >> sin(0.0001)/0.0001
ans =
  0.99999999833333
```

კომენტარების დამატება

რეკომენდირებულია რომ მ-ფაილებს დაურთოდ კომენტარები. კომენტარები იწყება პროცენტის ნიშნით %; კომენტარი არის მწვანე ფერის, რომ გამოირჩეს ბრძანებისგან, რაც შავი ფერისაა. ქვემოთ მოყვანილია მაგალითი **task1.m** ფაილისა კომენტარების დამატებით.



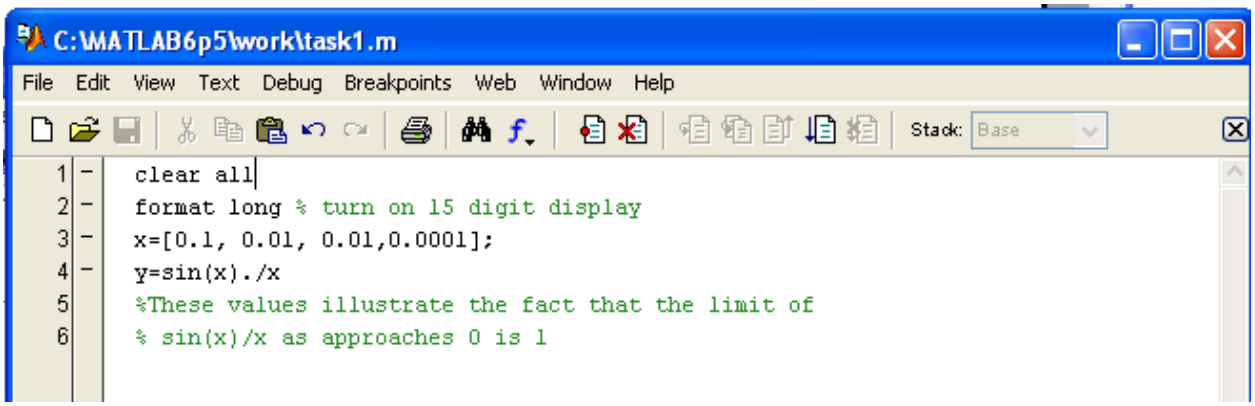
სცენარების ინიციალიზაცია

იმისათვის რომ რეზულტატი მ-ფაილები-სცენარებისა იყოს აღსადგენი სცენარები უნდა იყვნენ დამოუკიდებელი.განვიხილოთ მაგალითი, თუ შემოვიყვანთ ფანჯარაში **Command Window** ცვლადს სახელით **sin,** და მერე გაუშვებთ სცენარს **task1.m** მივიღებთ შეცდომას რადგან მიმდინარე მომენტში **sin** იქნება ცვლადი და არა ჩაშენებული ფუნქცია

```
>> sin=5
sin =
    5
>> task1
??? Subscript indices must either be real positive integers or logicals.
Error in ==> C:\MATLAB6p5\work\task1.m
On line 3  ==> y=sin(x)./x
```

იმისათვის რომ მ-ფაილები-სცენარები იყოს ავტონომიური, დამოუკიდებელი სხვა გამოყენებული ცვლადებისაგან თქვენ შეგიძლიათ გამოიყენოთ მ-ფაილები-

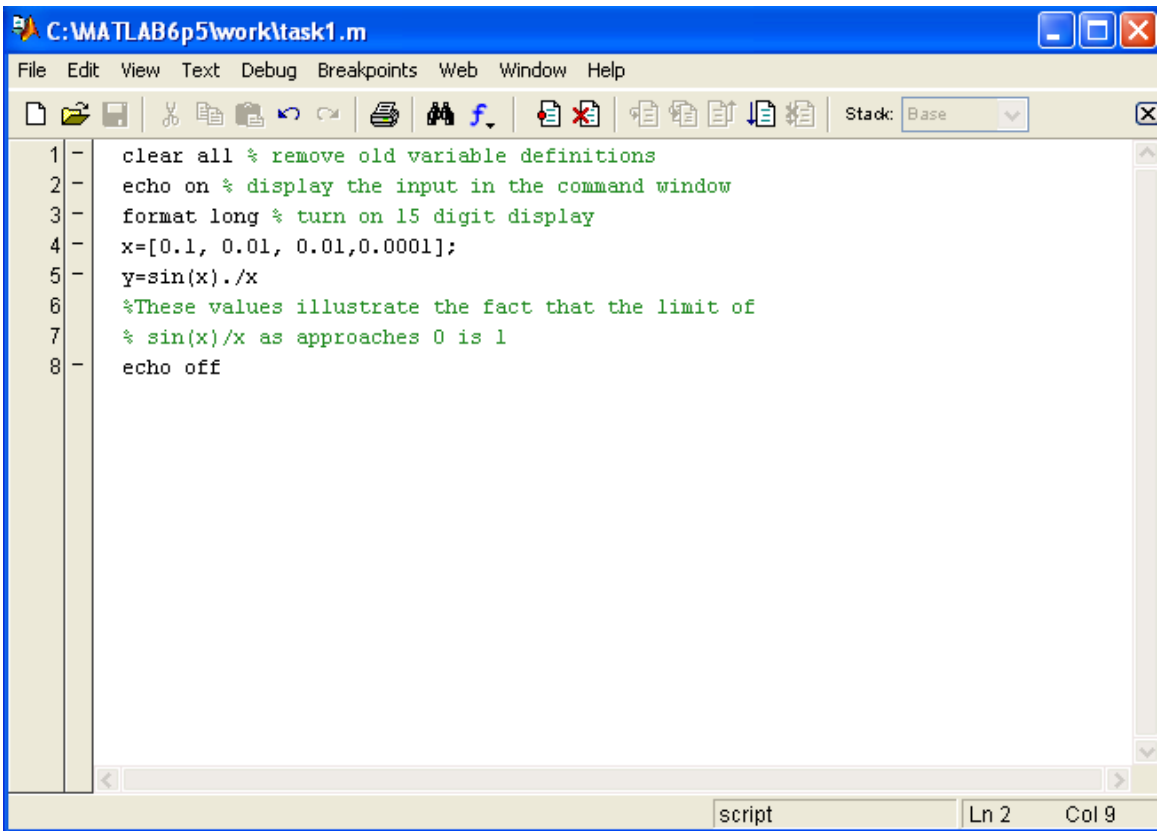
სცენარის დასაწყისში **clear all** სტრიქონი, რომ დარწმუნებული ვიყოთ რომ წინანდელი გამოყენებული მოქმედებები გავლენას არ მოახდენენ შედეგზე. აგრეთვე შეგიძლიათ გამოიყენოთ მ-ფაილები-სცენარის დასაწყისში **close all** სტრიქონი.



ეხლა მივიღებთ შედეგს, რაც არ არის დამოკიდებული ადრე გამოყენებულ მონაცემებზე

>> sin=5;
>> task1
y =
0.99833416646828 0.99998333341667 0.99998333341667 0.99999999833333

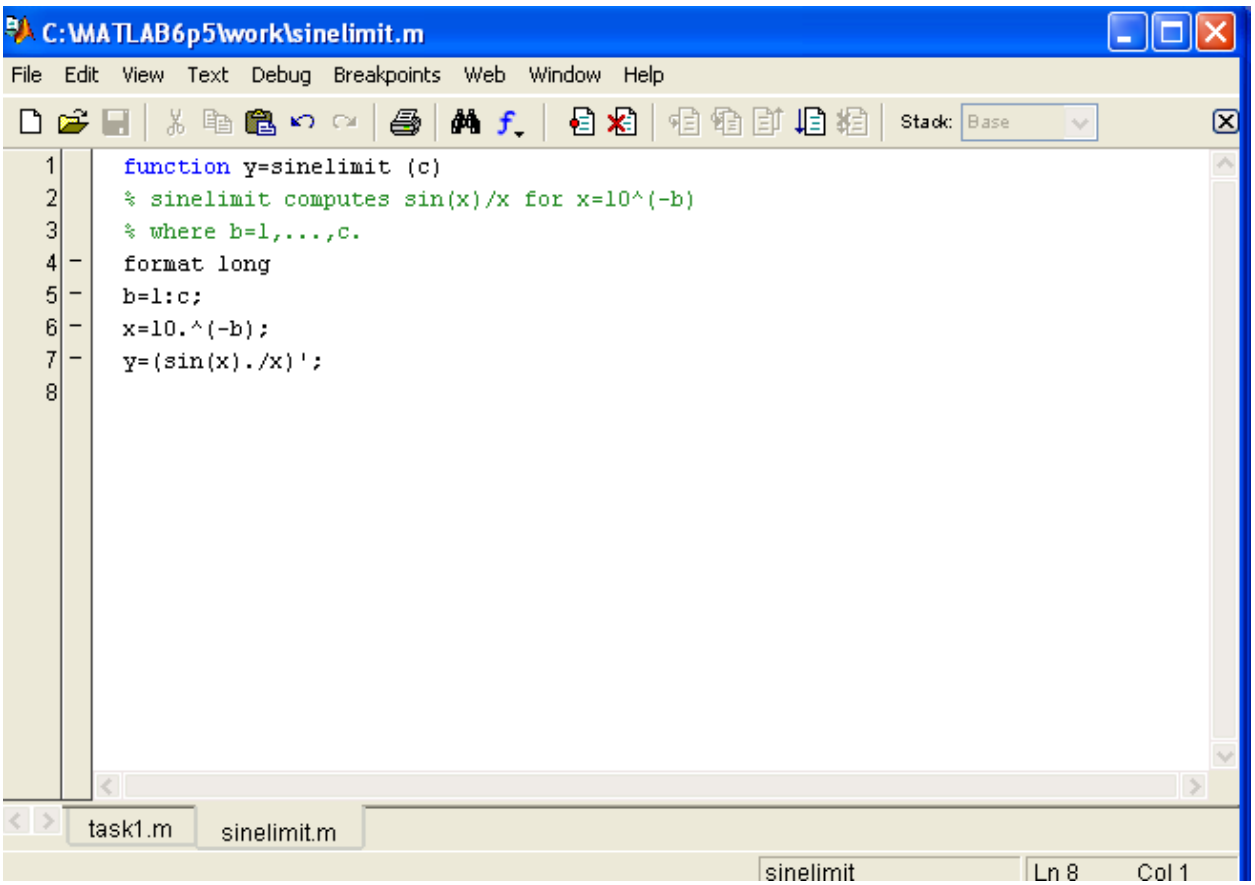
ზემოთ როგორც იყო აღნიშნული ფანჯარაში **Command Window** ში წარმოჩინდება მხოლოდ მ-ფაილები-სცენარების შედეგები, ბრძანებების გარეშე. თუ გინდათ რომ **Command Window** ფანჯარაში ერთად წარმოჩინდეს შედეგები და ბრძანებები, დაუმატეთ სცენარის დასაწყისში და ბოლოში ბრძანებები **echo on** და **echo off**



ეხლა გაუშვათ **Command Window** ფანჯარაში ბრძანება **task1,** შედეგები და ბრძანებები ერთად წარმოჩინდებიან:

M-ფაილ-ფუნქციები

m-ფაილი-ფუნქციები m-ფაილი-სცენარებიდან განსხვავებით გაშვებისას საშუალება გვაქვს მივცეთ შემავალი მნიშვნელობები. ქვემოთ მოცემულია მაგალითი m-ფაილი-ფუნქციისა სახელით **sinelimit.m:**



ფაილის პირველი სტრიქონი იწყება სიტყვით **function** (ლურჯი ფერისაა), რომელიც ფაილის იდენტიფიცირებას ახდენს მ-ფაილ-ფუნქციასთან. მ-ფაილ-ფუნქციის სახელი უნდა ემთხვეოდეს ფუნქციის სახელს. ამ მაგალითში ფუნქციის სახელიც **sinelimit** –ია. მას გააჩნია ერთი შემავალი ელემენტი **c** . რეზულტატიც ერთი ელემანტია **y.** ცვლადები **x,y, b** რომლებიც გამოვიყენეთ მ-ფაილი-ფუნქციაში **sinelimit.m** არიან ლოკალური ცვლადები.პროგრამა **MATLAB**-ი არ იმახსოვრებს ამ ცვლადების მნიშვნელობებს, მას შემდეგ რაც შესრულდება მ-ფაილ-ფუნქცია. ყურადღება მივაქციოთ იმას რომ სტრიქონები რომელშიც მოცემულია **b, x** და **y** ბოლოვდებიან ;. ეს ნიშნავს რომ ამ სტრიქონის შესრულების შედეგები არ წარმოჩნდება **Command Window** ფანჯარაში.

მაგალითი:

```
>> sinelimit (5)
ans =
   0.99833416646828
   0.99998333341667
   0.99999983333334
   0.99999999833333
   0.99999999998333
```

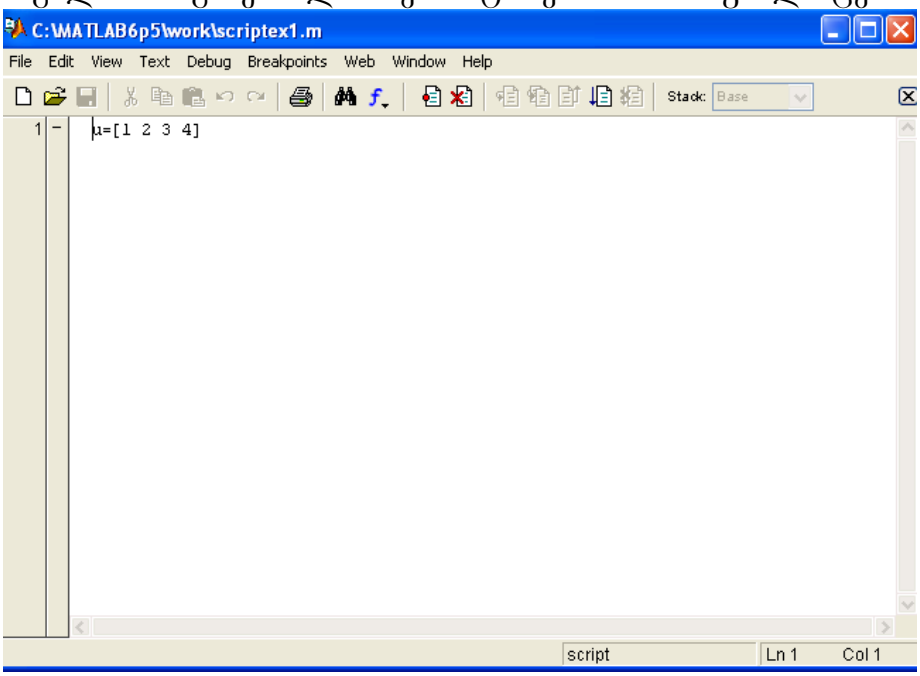
ბრძანება **help**-ით შეგვიძლია გამოვიტანოთ კომენტარები

```
>> help sinelimit
  sinelimit computes sin(x)/x for x=10^(-b)
  where b=1,...,c.
```

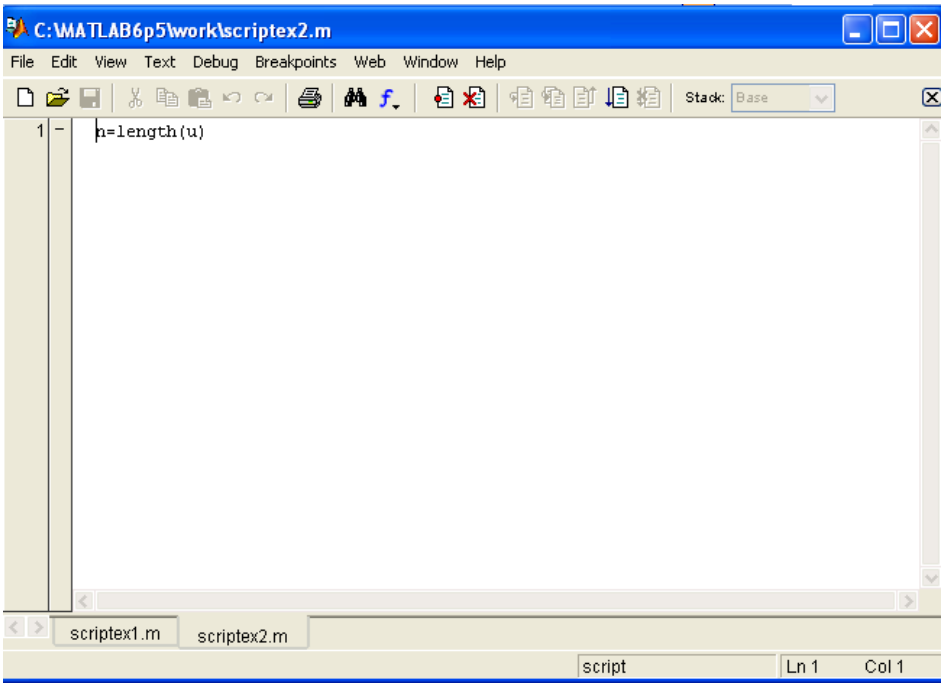
ცვლადები M-ფაილ-სცენარებში

M-ფაილ-სცენარების შესრულების დროს გამოიყენება ცვლადები , რომლებიც ეკუთვნიან სამუშაო ფანჯარას (გამოისახებიან ფანჯარა Workspace-ში) და ინახებიან სცენარის შესრულების შემდგომაც.

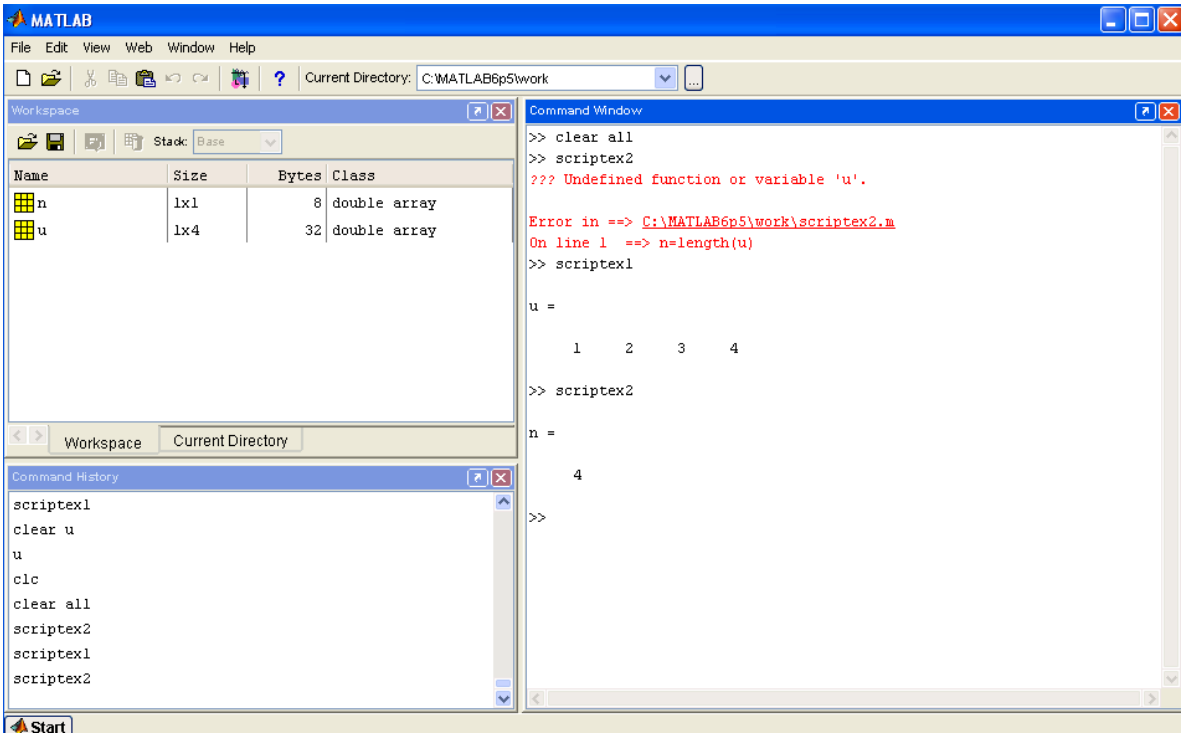
მაგალითი : განვიხილოთ ერთსტრიქონიანი M-ფაილ-სცენარი სახელით **scriptex1.m:**



ამ ბრძანებას თუ შეიყვანთ **Command Window** ფანჯარაში მოცემული ვექტორი მიენიჭება **u** ცვლადს**.** ეხლა განვიხილოთ მეორე M-ფაილ-სცენარი სახელით **scriptex2.m:**



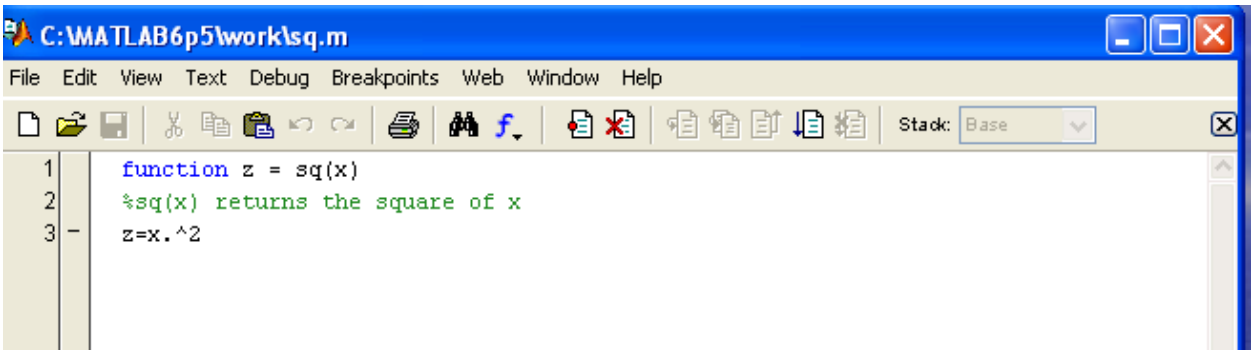
თუ თქვენ წინასწარ არ მიეცით **u,** მაშინ ბრძანება **scriptex2** შეყვანის შემდეგ მოგცემთ შეცდომას. მაგრამ შეიყვანთ მას **scriptex1** შეყვანის შემდეგ მაშინ მიიღებთ შედეგს ეკრანზე



როგორც ზემოთ იყო აღნიშნული, თუ არ გინდათ რომ ადრე გამოყენებულ ცვლადებს არ ჰქონეთ გავლენა შედეგზე გამოიყენეთ ბრძანება **clear all** M–ფაილის დასაწყისში

ცვლადები M-ფაილ ფუნქციებში

ცვლადები, რომლებიც გამოიყენება M-ფაილ ფუნქციებში არიან ლოკალურები, ე.ი არც თვითონ ახდენენ გავლენას ცვლადებზე რომლებიც არიან სამუშაო არეში **(Work space)** და არც მათზე არ ხდება იმ ცვლადების მხრიდან გავლენა. მაგალითი: განვიხილოთ ერთ სტრიქონიანი M-ფაილ ფუნქცია სახელით **sq.m:**



თუ შევიყვანთ ფანჯარა **Command Window** –ში ბრძანებას **sq(3)**, მივიღებთ:

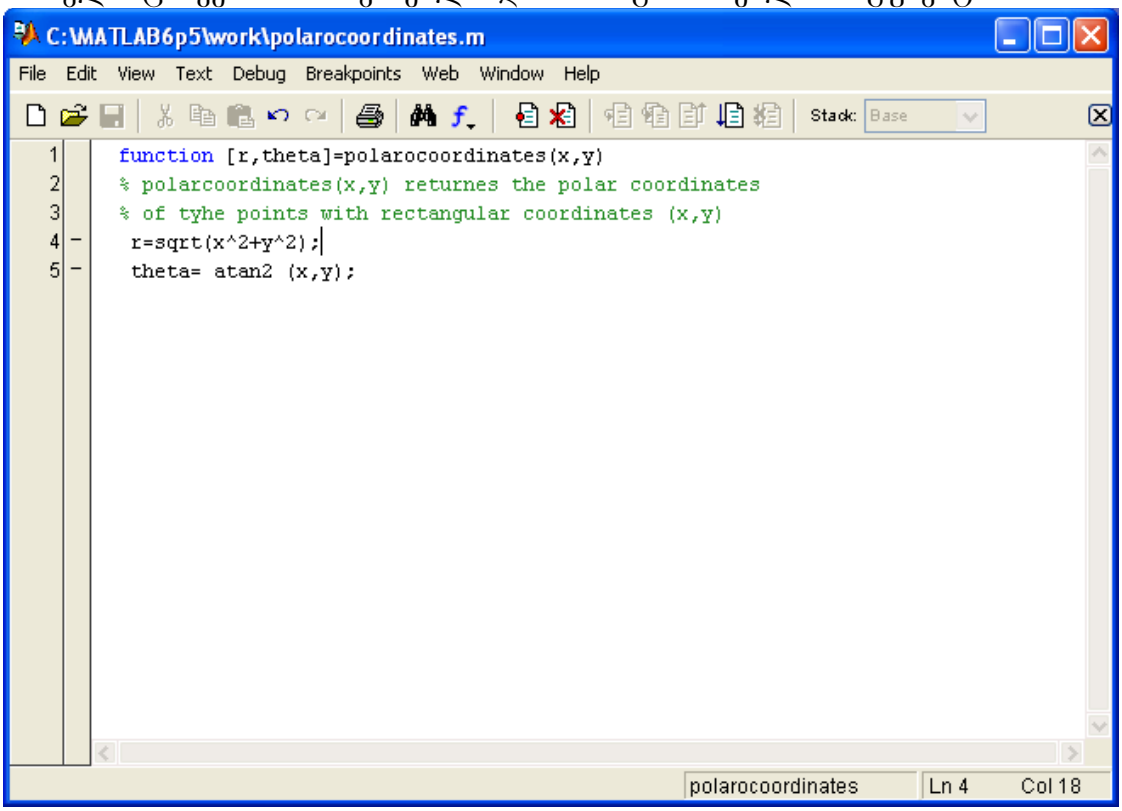
```
>> sq(3)
z =
    9
ans =
    9
```

იმის და მიუხედავათ მოცემულია თუ არა **x** და **y** ცვლადები სამუშაო არეში **(Work space).** მ-ფაილ ფუნქციის გაშვება არც გვაძლევს ცვლადებს და არც ცვლის მის პარამეტრებს **Work space** –ში, თუ ისინი ადრე იყვნენ მოცემულნი.

M-ფაილ ფუნქციების სტრუქტურა

მ-ფაილ ფუნქციის პირველი სტრიქონს უწოდებენ ფუნქციის განსაზღვრის სტრიქონს; ეს სტრიქონი იძლევა ფუნქციის სახელს , აგრეთვე შემავალი და გამოვალი არგუმენტების რაოდენობას და მიმდევრობას. ფუნქციის განსაზღვრის სტრიქონის შემდეგ შეიძლება მოდიოდეს რამოდენიმე სტრიქონი კომენტარებისა რომლებიც იწყებიან % ნიშნით. ამ სტრიქონებს უწოდებენ ცნობარის ტექსტს და გამოისახება ბრძანება **help** შეყვანის შემდგომ. დანარცენი სტრიქონები წარმოადგენენ ფუნქციის ტანს. საზოგადოდ ყველა ბრძანება რომელსაც გამოჰყავს შედეგი ; ბოლოვდება.

მ-ფაილ ფუნქციებს შეიძლება ჰქონდეს ბევრი შემავალი და გამომავალი არგუმენტი. მოყვანილია მაგალითი მ-ფაილ ფუნქციისა **polarcoordinates.m** სახელით, რომელსაც აქვს ორი შემავალი და ორი გამომავალი არგუმენტი:



თუ შეიყვანთ **polarcoordinates (3,4)** მხოლოდ პირველი არგუმენტი იქნება გამოტანილი და შენახული სახელით **ans-** ით:

>> polarcoordinates (3,4)

ans =

5

იმისათვის რომ რომ ორივე შედეგი დავინახოთ საჭიროა კვადრატულ ფრჩხილებში მოთავსებულ ცვლადებს უნდა მივანიჭოთ ეს მნიშვნელობები:

>> [r,theta]=polarcoordinates (3,4)

r =

5

```
theta =
  0.64350110879328
```
თუ
```
>> r=polarcoordinates (3,4)
r =
    5
>> theta=polarcoordinates (3,4)
theta =
    5
>>
```

ციკლები

ყველაზე მარტივი ხერხი ციკლების შექმნისა არის **for** გამოსახულების გამოყენება. მოცემულია მაგალითი 10!=10*9*8 *2*1 გამოთვლისა:

```
f=1;
for n=2:10
   f=f*n;
end
f
```

გრაფიკების შენახვა და ამობეჭდვა

შეგიძლიათ გრაფიკის შენახვა თუ აირჩევთ ბრძანებას მენიუდან **File-Save As….** გაჩუმების პრინციპით ინახება **(.fig)** ფორმატით, მაგრამ შესანახად გამოიყენება სხვა გრაფიკული ფორმატებიც. დასაბეჭდად ვირჩევთ ბრძანებას მენიუდან **File-Print…**

მაგალითი:

```
 Close all%remove old figure windows
% graph sin(x) from 0 to 2*pi:
x=2*pi*(0:0.01:1);
plot(x,sin(x))
title ('Figura A: Sine Curve') %title the figure
print -deps figureA %store the graph in figureA.eps
```

აქ მიმდინარე გამოსახულება იწერება ფორმატში **EPS-Encapsulated PostScript** ფაილში სახელით **figureA.eps.** თუ გსურთ **web-** გვერდზე განათავსოთ გრაფიკი, უმჯობესია შეინახოთ გამოსახულება ფაილში ფორმატით **.png.** გამოვიყენებთ რა ბრძანება **print -dpng.**

ნაკადის მართვა

მეტი ინფორმაციისთვის იხ.: help lang.

მარტივი IF დებულება

ზოგადი ფორმა მარტივი **if** დებულებისა არის:

```
if ლოგიკური გამოსახულება
ბრძანებები
end
```

თუ ლოგიკური გამოსახულება ჭეშმარიტია, მაშინ სრულდება ბრძანებები **if** და **end** ს შორის. თუ გამოსახულება არის მცდარი, მაშინ გამოიტოვება ბრძანებები **if** და **end** ს შორის და პირდაპირ **end** -ზე გადადის მოქმედება.
მაგალითი:

```
>> d=8
d =
   8
>> count=0
count =
   0
>> if d<50
count=count+1;
disp(d);
end
   8
>> count
count =
   1
```

აქ d -სკალარია და d <50 და შესრულდება count -ის 1 -ანი ნაზრდით გაზრდა და d-ს დისპლეიზე გამოტანა. თუ (d <50) ეს პირობა დაირღვეოდა, მაშინ ეს ორი ბრძანება არ შესრულდებოდა. თუ d - არ არის სკალარი, მაშინ if -ს ლოგიკური დებულება ჭეშმარიტი იქნება, თუ მისი ყოველი ელემენტი ნაკლები იქნება 50 -ზე.

```
>> d=56
d =
  56
>> if d<50
count=count+1;
disp(d);
end
```

აქ პირობა d<50 მცდარია და ეს ორი ბრძანება

```
count=count+1;
disp(d);
```

არ სრულდება. ვექტორის შემთხვევაში:

```
>> d=[12 34 45]
d =
   12   34   45
>> if d<50
count=count+1;
disp(d);
end
   12   34   45
```

სხვა მაგალითი.

```
>> d=[12 34 55]
d =
   12   34   55
>> if d<50
count=count+1;
disp(d);
end
```

აქ პირობა d<50 მცდარია, 55 მეტია 50 -ზე.

ჩალაგებული IF დებულებები

დებულებები შეიძლება იყოს ჩალაგებული, როგორც ნაჩვენებია მაგალითში:

```
>> d=3
d =
   3
>> b=5
b =
   5
>> count
count =
   5

>> if d<50
count=count+1;
disp(d);
if b>d;
b=0;
end
end
   3

>> b
b =
   0
```

აქ b და d სკალარებია და პირობა d<50 შესრულებულია. თუ ეს პირობა სრულდება მაშინ შესრულდება count ის ზრდა 1 -ნი ნაზრდით და d -ს დისპლეიზე გამოტანა. ამის გარდა თუ პირობა b>d სრულდება მაშინ შესრულდება b=0.

თუ ეს პირობა d<50 არ სრულდება მაშინ გამოვტოვებთ ყველა ბრძანებას მეორე end დებულებამდე.

მაგალითი: ვთქვათ

```
>> d=56
d =
  56

>> b=57
b =
  57

>> count
count =
   5


>>  if d<50
count=count+1;
```

```
disp(d);
if b>d;
b=0;
end
end
>> b
b =
  57
```

არცერთი ბრძანება არ სრულდება, რადგან პირობა d<50 მცდარია და გადადის მეორე end-ზე, მიუხედავათ იმისა რომ მეორე პირობა b>d ჭესმარიტია (57>56 , b ისევ 57 რჩება და არ ხდება 0-ის ტოლი).

else and elseif Clauses (პირობებით-ოპერატორებით)

else clause (else პირობით): თუ **if -**ის ლოგიკური გამოსახულება ჭეშმარიტია, მაშინ შესრულდება ბრძანებები **else** -მდე და მერე გადახტება **end** -ზე, და თუ კი **if -**ის ლოგიკური გამოსახულება მცდარია, მაშინ შესრულდება **else** -შემდეგ ბრძანებები. მაგალითი:

```
function xinc=nnn(interval)
%UNTITLED4 Example of else Clauses
 if interval<1
xinc=interval/10;
else
xinc=0.1;
end
end
```

მაშინ თუ შევიყვანთ `interval` -ის მნიშვნელობებს, მივიღებთ:

```
>> interval
interval =
     3

>> xinc=nnn(interval)
xinc =
    0.1000
```

აქ `if` -ს შემდეგ ჩაწერილი ლოგიკური გამოსულება მცდარია (3 მეტია 1 -ზე), ამიტომ ბრძანება `xinc=interval/10` გამოიტოვება და შესრულდება ბრძანება `xinc=0.1`, რომელიც `else` -ის შემდეგ არის ჩაწერილი.

თუ

```
>> interval=0.7
interval =
    0.7000

>> xinc=nnn(interval)
xinc =
    0.0700
```

აქ `if` -ს შემდეგ ჩაწერილი ლოგიკური გამოსულება ჭეშმარიტია (`0.7` ნაკლებია 1 -ზე), ამიტომ სრულდება ბრძანება `xinc=interval/10` და გადავა `end` -ზე გამოიტოვება `else` -ის შემდეგ ჩაწერილი ბრძანება `xinc=0.1`.

ლოგიკურ გამოსახულებაში აგრეთვე დაშვებულია მასივების გამოყენება. იმ შემთხვევაში, როდესაც მნიშვნელობები ასეთი გამოსახულებისა იქნება მასივი, მაშინ ჭეშმარიტება დგება, როდესაც ჭეშმარიტია (არა ნულოვანი) ყველა ელემენტი მასივისა. თუ ერთი ელემენტი მაინც ასეთი მასივისა იქნება ნულის ტოლი, მაშინ პირობა მცდარია. გარდა ამისა, პირობა ითვლება მცდარი თუ გამოიყენება ცარიელი სიმრავლე.

მაგალითი 2.

```
  function b = AAA( A )
%Example2 of Else if Clauses
 if A
b=1;
else
b=2;
end
```

თუ

```
>> A=[1 2;4 0]
A =
   1   2
   4   0

>> b = AAA( A )
b =
   2
```

აქ მატრიცა **A** შეიცავს ერთ ნულოვან ელემენტს, ამიტომ პირობა **A** `if A` -ში მცდარია და სრულდება პირობა `else`-ს შემდეგ, `b=2`. ქვევით მაგალითში კი პირიქით

```
>> A=[1 2;4 5]
A =
   1   2
   4   5

>> b = AAA( A )
b =
   1
```

გამოსახულება **elseif** -ის გამოყენება ხელსაყრელია,თუ ორზე მეტი ალტერნატივაა. გამოვიყენოთ გამოსახულება **elseif** -ი **sign** ფუნქციის წარმოსადგენად

$$sgn = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

განვიხილოთ

```
function y = signum( x )
%Presentation of sign function
 if x>0
     y=1;
```

```
    elseif x==0
        y=0;
    else
        y=-1;
    end

end
```

თუ x=5
y = signum(5)
y =

1

თუ x=0
>> y = signum(0)
y =
0

თუ x=-7
>> y = signum(-7)
y =
-1

ე.ი თუ **x** დადებითია, მაშინ **y** ღებულობს მნიშვნელობას 1, და ყველა ბრძანება `elseif` გამოსახულებიდან `end` -მდე გამოიტოვება. თუ **x** არა დადებითია, მაშინ პროგრამა **MATLAB** გადადის `elseif` გამოსახულებასთან და შემოწმდება **x** უდრის თუ არა ნულს. თუ ეს ასეა მაშინ **y** მიიღებს მნიშვნელობას 0, თუ არა და მაშინ **y** მიიღებს მნიშვნელობას -1.
როდესაც რამდენიმე დონე `if else` ჩალაგებულია, ხელსაყრელია `elseif` გამოყენება გასამარტივებისათვის.
მაგალითი. გამოვიყენოთ ბრძანება `if else` იმისათვის რომ დავწეროთ **M-file** ფუნქცია ერთი შემავალი არგუმენტით და ტექსტის სახით აღწერეთ,არგუმენტი არის თუ არა სკალარი, ვექტორი ან მატრიცა. მნიშვნელობა არ ბრუნდება.

```
function testvar(x)
%Display text indicating whether x is a
%   scalar, vector, or matrix
[m,n]=size(x);
if m==n&m==1
    disp('Argument is a scalar')
elseif m==1| n==1
    disp ('Argument is a vector')
else
    disp('Argument is a matrix')
end
end
```

>> x=9
x =
9
>> testvar(x)
Argument is a scalar

```
>> x=[2 3]
x =
   2   3

>> testvar(x)
Argument is a vector

>> x=[4 5;6 7]
x =
   4   5
   6   7

>> testvar(x)
Argument is a matrix

   >>
```

Swich Selection Structure (გადამრთველი ოპერატორი)

მორიგი განშტოების ოპერატორი არის გადამრთველი ოპერატორი. ის გამოიყენებს შემდეგ გასაღების სიტყვებს **Swich** (გადამრთველი), **case** (პირობა), **otherwise** (წინააღმდეგ შემთხვევაში) და აქვს შემდეგი კონსტრუქცია:

swich (გამოსახულება)
    case მნიშვნელობა 1
...
    case {მნიშვნელობა 2, მნიშვნელობა 3}
....
    otherwise
...
    end

აქ ჯერ გამოითვლება გამოსახულება, რომელიც ღებულობს სკალარულ რიცხვით მნიშვნელობას, შემდეგ მიღებული შედეგი დარდება მნიშვნელობა 1, მნიშვნელობა2, მნიშვნელობა 3 და ა.შ. თუ რომელიმეს დაემთხვა მასინ სრულდება მომდევნო ქვეშ მდგომი განშტოება და თუ არცერთ მნიშვნელობას არ დაემთხვა, მაშინ გადადიხართ განშტოებაზე რომელიც არის **otherwise** -ისშემდეგ. სტრიქონების რაოდენობა საგასაღებო სიტყვით **case** შეზღუდული არ არის, ხოლსო სტრიქონი საგასაღებო სიტყვით **otherwise** უნდა მხოლოდ ერთი იყოს.

მაგალითი

```
function y = count( x)
%Switch
%   Detailed
switch x
    case 1
        y='one';
    case 2
        y='two';
    otherwise
        y='many';
    end

 >> x=1
x =
```

```
     1

>> y = count( x)
y =
one

>> x=3
x =
     3

>> y = count(x)
y =
many

>> x=2
x =
     2

>> y = count(x)
y =
two
```

მაგალითი 2.

```
function xinc = SWITCHEXAMPLE( interval )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
switch interval<1
    case 1
        xinc=interval/10;
    case 0
        xinc=0.1;

end
```

```
>> interval=0.5
interval =
   0.5000

>> xinc = SWITCHEXAMPLE( interval )
xinc =
   0.0500

>> interval=3
interval =
    3

>> xinc = SWITCHEXAMPLE( interval )
xinc =
   0.1000
```

Switch syntax is

```
switch expression
     case test expression1
          commands
     Case {test expression 2, test expression 3}
          Commands
```

```
        .
        .
        .
Otherwise
     Commands
end
```

For

ციკლი **for** იმეორებს ოპერატორთა ჯგუფის შესრულებას წინასწარ განსაზღვრულ ფიქსირებულ რიცხვის ჩათვლით. საგასაღებო სიტყვა **end** აბოლოვებს ციკლის ტანს.

```
>> for n=3:32
r(n)=rank(magic(n));
end
>> r

r =
 Columns 1 through 24
   0   0   3   3   5   5   7   3   9   7  11   3  13   9  15   3  17  11  19   3  21  13
23   3
 Columns 25 through 32
  25  15  27   3  29  17  31   3
```

ციკლის ტანში **;** - გამოყენება გამოსახულების შემდეგ, უზრუნველყოფს იმას რომ, ეკრანზე არ გამოვიდეს შუალედური გათვლები, ციკლის შემდეგ **r** კი გამოაქვს შედეგი.

ჩალაგებული ციკლები:

```
for i=1:3
for j=1:2
H(i,j)=1/(i+j);
end
end
```

While

ციკლი **While** იმეორებს ოპერატორთა ჯგუფის შესრულებას იმდენჯერ სანამ სრულდება ლოგიკური პირობა (მანამ იგი ჭეშმარიტია). საგასაღებო სიტყვა **end** აბოლოვებს გამოყენებულ ოპერატორებს.

ქვემოთ მოცემულია პროგრამა, სადაც ილუსტრირებულია ოპერატორების **while, if, else** და **end** ის მუშაობა, რომლებიც გამოყენებულია მონაკვეთის შუაზე გაყოფის მეთოდში $x^3 - 2x - 5$ პოლინომის ფესვების მოსაძებნად

```
>> eps
ans =

 2.2204e-016
```

```
>> a=0; fa=-inf;
b=3;fb=inf;
while b-a>eps*b
x=(a+b)/2;
fx=x^3-2*x-5;
if sign(fx)==sign(fa)
a=x;fa=fx;
else
b=x;fb=fx;
end
end
x

x =
   2.0946
```

break

ოპერატორი **break** გვაძლევს საშუალებას ნაადრევად გამოვიდეთ ციკლებიდან. ჩალაგებულ ციკლებში აწარმოებს გამოსვლას ყველაზე შიდა ციკლიდან. ქვემოთ მოცემულია წინა მაგალითის გაუმჯობესებული ვარიანტი. რატომ არის უფრო ხელსაყრელი ოპერატორი **break** –ის გამოყენება?

```
>>  a=0; fa=-inf;
b=3;fb=inf;
while b-a>eps*b
x=(a+b)/2;
fx=x^3-2*x-5;
if fx==0
break
elseif sign(fx)==sign(fa)
a=x;fa=fx;
else
b=x; fb=fx;
end
end
>> x

x =
   2.0946
```

სიმბოლოები და ტექსტი

შემოვიტანოთ ტექსტი ერთჯერადი ფრჭყალებით, მაგალითად
s='Hello'
შედეგად მივიღებთ სიმბოლურ მასივს 1X5.
სიმბოლოები ინახება როგორც რიცხვები, მაგრამ არა მცოცავ მძიმიან ფორმატს.
ოპერატორი

>> a=double(s)
გარდაქმნის სიმბოლოთა მასივს რიცხვით მასივში, რომელსაც გააჩნია ASCII კოდი მცოცავ მძიმიან წარმოდგენას ყოველი სიმბოლოსთვის. შედეგად იქნება
a =

72 101 108 108 111
ქვემოთ გამასახულებას კი პირიქით, უკუ გარდაქმნის:
>> s=char(a)

s =
Hello
კვადრატული ფრჩხილები აერთიანებს ტექსტურ ცვლადებს ერთად ერთდიდ სტრიქონში

>> h=[s,'world']

h =
Helloworld

>> h=[s,' world']

h =
Hello world
ოპერატორი

>> v=[s;'world']
აერთიანებს სტრიქონებს ვერტიკალში, რასაც მივყავართ
v =

Hello
world

შევნიშნოთ, რომ h ცვლადში w სიმბოლოს წინ აუცილებლად უნდა იყოს პრობელი, ხოლო v ცვლადში ორივე სიტყვები უნდა იყვნენ თანაბარი სიგრძის. შედეგად გვაქვს მასივი სიმბოლოების: h ცვლადი – 1X11, ხოლო v ცვლადი -2X5.

აქ სიტყვები არ არიან თანაბარი სიგრძის
>> v=[s;' world']
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
არსებობს ორი ხერხი, იმისათვის რომ ვმართოდ ტექსტის ჯგუფი, რომელთაც სხვადასხვა სიგრძის სტრიქონებს შეიცავენ: შევქმნათ სიმბოლოთა მასივი, ან

სტრიქონთა უჯრედიანი მასივები. ფუნქცია char – ი იღებს ნებისმიერ რაოდენობა სტრიქონებს, უმატებს პრობელებს ყოველ სტრიქონში, იმისათვის რომ ყველა მათგანი ტოლი სიგრძისა იყვნენ და ქმნიან სტრიქონთა მასივს სიმბოლური სტრიქონით ყოველ სტრიქონში. მაგალითად

```
>> S=char ('A' , 'rolling', 'stone', 'gathers', 'momentum.')

S =

A
rolling
stone
gathers
momentum.
```

არის საკმარისი რაოდენობა პრობელებისა პირველ ოთხ სტრიქონში, იმისათვის რომ ყველა სტრიქონები ტოლი სიგრძისა იყვნენ. მეორე ხერხი –არის ტექსტის შენახვა უჯრედთა მასივში

```
>> C= {'A' ; 'rolling'; 'stone'; 'gathers'; 'momentum.'}
```

იქნება უჯრედთა მასივი 1X5

```
C =

   'A'
   'rolling'
   'stone'
   'gathers'
   'momentum.'
```

შეგვიძლია გარვდაქმნათ შევსებული სიმბოლური მასივი უჯრედთა მასივში სტრიქონებით შემდეგ ნაირად

```
>> C=cellstr(S)

C =

   'A'
   'rolling'
   'stone'
   'gathers'
   'momentum.'
```

უკუგარდაქმნა

```
>> S=char(C)

S =

A
rolling
stone
```

gathers
momentum.

ფუნქციები, რომლებიც ასრულებენ ბიტურ ოპერაციებს

ნებისმიერი მთელი რიცხვის გარდაქმნისათვის ოროობით წარმოდგენაში გამოიყენება ფუნქცია dec2bin:

```
>> dec2bin(12)

ans =
1100
```

უკუ გარდაქმნისათვის, ანუ რიცხვის ოროობითი წარმოდგენიდან ათობით წარმოდგენაში გამოიყენება ფუნქცია bin2dec:

```
>> bin2dec('1100')

ans =
   12

>> bin2dec('1000')

ans =
    8
```

ათობითი რიცხვის თექვსმეტობითში წარმოსადგენათ გამოიყენება ფუნქცია dec2hex

```
>> dec2hex(193)

ans =
C1
```

პირიქით, თექვსმეტობითიდან ათობითში წარმოსადგენათ გამოიყენება ფუნქცია hex2dec

```
>> hex2dec('C1')

ans =
  193
```

ზემოთ ჩამოთვლილი ფუნქციები შეიძლება გამოვიყენოთ მასივებისთვისაც

```
>> A=[10,20]

A =
   10   20

>> dec2bin(A)

ans =

01010
10100
```

გარდაქმნა ხდება ელემენტებისა სვეტების და მიხედვით

```
>> B=[1 2 3;4 5 6]

B =

   1   2   3
   4   5   6

>> dec2bin(B)

ans =

001
100
010
101
011
110

>> dec2bin([12,56])

ans =
001100
111000
```

სამ ძირითად ბიტურ ოპერაციებს „ ბიტური და”, „ ბიტური ან”, „ ბიტური ან –ის უარყოფა” ასრულებენ შესაბამისად შემდეგი ფუნქციები: bitand, bitor, bitaxor

```
>> bitand(12,56)

ans =
   8

>> bitor(12,56)
ans =

  60
```

ლიტერატურა

1. Brian R. Hunt/ Ronald L. Lipsman/ Jonathan M.Rosenberg,Matlab: Официальный учебный курс Кембриджского университета,Москва, Издательство ТРИУМФ, 2008
2. С.В. Поршнев, Учебник 'Matlab7' основы работы и программирования, Москва, Издательство БИНОМ,2006
3. D.M.Etter.Engineering Problem Solving with Matlab. Prentice Hall, Englewood Clifs,New Jersey,1993

სარჩევი

შესავალი. 2
სამუშაოს დაწყება. 2
პროგრამა “Matlab”-ის გაშვება 2
ფანჯარა Command Window (ბრძანების ფანჯარა) 3
ფანჯარა Workspace (სამუშაო არე). 5
ფანჯარა Command History (ბრძანებათა ისტორია) 7
სამუშაოს დასრულება 8
პროგრამა “Matlab”ის საფუძვლები. 8
შეტანა გამოტანა. 8
არითმეტიკა 9
კომლექსური რიცხვები 11
გამოთვლების შეწყვეტა 12
ალგებრული და სიმბოლური გამოთვლები 12
შეტანა სიმბოლურ გამოსახულებებში. 13
სიმბოლური გამოსახულებები, ცვლადის სიზუსტე და ზუსტი არითმეტიკა. 14
სიმბოლური ცვლადები, გამოსახულებები და მატრიცები. 15
სიმბოლური დიფერენცირებადობა. 16
სიმბოლური ინტეგრირება. 17
ორმაგი ინტეგრალის დათვლა. 18
ზღვრების გამოთვლა 18
ჯამი და ნამრავლი. 19
ტეილორის მწკრივი 20
ვექტორები და მატრიცები 20
ვექტორი 20
მატრიცა 22
მომხმარებლის მიერ მოცემული ფუნქციები 28
ელემენტარული ფუნქციები. 29
მიმართების ოპერაციები და ლოგიკური ოპერაციები რიცხვებზე 30
ნულზე გაყოფის აცილება 31
ლოგიკური ოპერატორები 32
ლოგიკური ფუნქციები 34

განტოლებების ამოხსნა 39
გრაფიკა 41
გრაფიკების აგება ezplot ბრძანებით 41
გრაფიკების აგება ბრძანებით plot 42
რამდენიმე მრუდის აგება 43
M-ფაილები 45
M-ფაილი-სცენარები 45
კომენტარების დამატება 48
სცენარების ინიციალიზაცია 48
M-ფაილ-ფუნქციები 50
ცვლადები M-ფაილ-სცენარებში 52
ცვლადები M-ფაილ ფუნქციებში 53
M-ფაილ ფუნქციების სტრუქტურა 54
ციკლები 55
გრაფიკების შენახვა და ამობეჭდვა 55
მარტივი IF დებულება 55
ჩალაგებული IF დებულებები 57
else and elseif Clauses (პირობებით-ოპერატორებით) 58
>> 61
Swich Selection Structure (გადამრთველი ოპერატორი) 61
For 63
While 63
break 64
სიმბოლოები და ტექსტი 65
ფუნქციები, რომლებიც ასრულებენ ბიტურ ოპერაციებს 67
ლიტერატურა 69