ს ა რ ჩ ე ვ ი

შესავალი

MATLAB–ი – რიცხვითი გამოთვლების, მონაცემთა ანალიზისა და მათი გრაფიკული ინტერპრეტაციის ერთ–ერთი წამყვანი პროგრამული პაკეტია. მძლავრი მათემატიკური აპარატისა და ვიზუალიზაციის ფართო შესაძლებლობების კომბინაცია განაპირობებს ამ პროგრამული პაკეტის ფართო გამოყენებას საინჟინრო ამოცანების გადაწყვეტაში.

MATLAB–ი შეიქმნა 1984 წელს. MathWorks კომპანიამ მუდმივად ავითარებს MATLAB–ის პროგრამულ პაკეტს, გამოდის მისი ახალი ვერსიები. მოცემულ სახელმძღვანელოში განხილულია 7.5.0 ვერსია.

აბრევიატურა MATLAB (MATrix LABoratory) ნიშნავს „მატრიცულ ლაბორატორიას“. გამოთვლები სისტემაში სრულდება ვექტორებზე ან ვექტორთა ერთობლიობაზე, რაც განასხვავებს MATLAB–ს სხვა პროგრამებისაგან, როგორებიცაა Mathcad, Maple, Mathematica. მონაცემთა ვექტორული დამუშავება განაპირობებს გამოთვლების სისწრაფეს, ციკლების ოპერატორების გამოყენების გარეშეც ციკლური გამოთვლების შესრულების საშუალებასა და ძალიან მაღალი სიზუსტის მიღებას. MATLAB–ის სხვა მნიშვნელოვანი თვისებაა მოდულური აგების პრინციპი, რაც იძლევა გამოყენებითი პროგრამების პაკეტების შექმნის შესაძლებლობებს. ამ პროგრამების შექმნაში მონაწილეობას იღებენ როგორც MATLAB სისტემის დამმუშავებლები, ისე მრავალრიცხოვანი პარტნიორული ორგანიზაციები.

დღეისათვის MATLAB–ი წარმოადგენს ინტერაქტიულ სისტემურ და პროგრამულ ენას სამეცნიერო და ტექნიკური გამოთვლებისათვის.

MATLAB–ის გამოყენების ტიპური სახეებია:

- მათემატიკური გამოთვლები;
- ალგორითმების შექმნა;
- მოდელირება;
- მონაცემთა ანალიზი, გამოკვლევა და ვიზუალიზაცია;
- მეცნიერული და საინჟინრო ამოცანების გადაწყვეტა;
- სხვადასხვა სფეროში გამოყენებითი პროგრამების დამუშავება გრაფიკული ინტერფეისის შექმნით.

MATLAB–ის ენა და მასში არსებული დაპროგრამების საშუალებანი ახლოსაა თანამედროვე უნივერსალური ალგორითმული ენების (Basic, C++, Java, Object Pascal) ბაზაზე დაფუძნებულ დაპროგრამების სისტემებთან. რიგ ასპექტში MATLAB–ი ჩამორჩება აღნიშნულ ენებს (ინტერპრეტაციის რეჟიმი, ვიზუალური კომპონენტების მცირე რაოდენობა), სამაგიეროდ მის რიცხვითი მეთოდების ბიბლიოთეკას – როგორც რაოდენობის, ისე ხარისხის მხრივ – ვერც ერთი დაპროგრამების სისტემა ვერ შეედრება. გარდა ამისა, MATLAB პაკეტში შედის რიცხვითი დამუშავების შედეგების ვიზუალიზაციისა და სხვადასხვა გრაფიკული ობიექტების

ასახვის მრავალფეროვანი შესაძლებლობები. MATLAB–ის ბაზაზე შექმნილია მრავალრიცხოვანი გაფართოებები, რომლებიც განაპირობებენ მოდელირებას და სისტემათა ანალიზს ადამიანის მოღვაწეობის სხვადასხვა სფეროში და ფაქტობრივად სისტემის შემადგენელ ნაწილს წარმოადგენენ.

თანამედროვე მეცნიერებასა და ტექნიკაში MATLAB–ი წარმოადგენს მოდელირებისა და სხვადასხვა გამოყენებითი სისტემების ანალიზის შეუდარებელ ინსტრუმენტს. ეს ეხება როგორც სისტემის მზა პროდუქტების გამოყენებას, ისე MATLAB სისტემაში არსებულ პროგრამულ და ალგორითმულ საშუალებათა ბაზაზე ახალი პროდუქტების შექმნას. ძალზე მნიშვნელოვანია აგრეთვე სხვადასხვა დაპროგრამების სისტემაში შემუშავებული მოდულების გაერთიანების შესაძლებლობების ათვისება.

MATLAB–ის უნიკალურობა განისაზღვრება მისი შემდეგი თვისებებით:

- სისტემის ორიენტირებით მატრიცულ ოპერაციებზე;
- ბიბლიოთეკური ფუნქციების ძალზე დიდი რაოდენობით, რის გამო MATLAB–ი შეიძლება განვიხილოთ როგორც სპეციალიზირებული მათემატიკური სისტემა სამეცნიერო და საინჟინრო ამოცანების ამოსახსნელად (მართვის სისტემების ანალიზი და სინთეზი, ექსპერიმენტის დაგეგმვა და უამრავი სხვა);
- სხვა მათემატიკურ სისტემებთან (Maple, Mathcad, MS Excel) დიალოგის გამართვის უნარით, რაც აფართოვებს MATLAB–ის შესაძლებლობებს და ანულირებს მის ერთ–ერთ ნაკლს – სხვა სისტემებთან შედარებით უფრო სუსტ სიმბოლურ მათემატიკას.

წინამდებარე სახელმძღვანელო განკუთვნილია როგორც სტუდენტებისათვის, ასევე მომხმარებელთა ფართო წრისათვის, ვინც დაინტერესებულია საინჟინრო და სამეცნიერო ამოცანების გადაწყვეტაში თანამედროვე პროგრამული პაკეტების გამოყენებით, რომელთა შორის MATLAB–ს უდავოდ ერთ–ერთი წამყვანი პოზიცია უკავია.

თავი I საწყისი ცნებები

1.1 MATLAB–ის შემადგენელი ნაწილები

MATLAB–ი შედგება ხუთი ძირითადი ნაწილისაგან:

MATLAB–ის ენა – ეს არის ობიექტზე ორიენტირებული, მატრიცებისა და მასივების დამუშავების მაღალი დონის ენა, რომლის საშუალებით შეიძლება განვახორციელოთ როგორც „მცირემასშტაბიანი“ დაპროგრამება „შავად ნაწერი“ პროგრამების დასაწერად, ასევე „დიდმასშტაბიანი“ – დიდი და რთული პროგრამების შესაქმნელად.

MATLAB–ის გარემო – ეს არის ინსტრუმენტების ერთობლიობა, რომელსაც მომხმარებელი ან პროგრამისტი იყენებს MATLAB–ში. აქ შედის სამუშაო არეში ცვლადების მართვის, მონაცემთა შეყვანა–გამოყვანის, M–ფაილებისა და გამოყენებითი პროგრამების შექმნის, კონტროლის და გაშვების საშუალებები და სხვ.

მმართველი გრაფიკა (GUI) – ეს არის MATLAB–ის გრაფიკული სისტემა, რომელიც შეიცავს ორ და სამგანზომილებიან მონაცემთა ვიზუალიზაციის საშუალებებს, გამოსახულების დამუშავების, გრაფიკის ანიმაციისა და ილუსტრირების შესაძლებლობებს.

მათემატიკური ფუნქციების ბიბლიოთეკა – გამოთვლითი ალგორითმების ფართო კოლექციაა, დაწყებული ელემენტარული ფუნქციებიდან (ჯამი, სინუსი, კოსინუსი, კომპლექსური რიცხვების არითმეტიკა), რთული ფუნქციებით (ბესელის ფუნქცია, ფურიეს და ლაპლასის გარდაქმნის, მატრიცებთან მუშაობასთან დაკავშირებული ფუნქციები და სხვ.) დამთავრებული.

დაპროგრამების ინტერფეისი – ბიბლიოთეკა, რომელიც საშუალებას გვაძლევს დავწეროთ პროგრამები C–სა და FORTRAN დაპროგრამების ენებზე, რომლებიც ურთიერთქმედითია MATLAB–თან. ის შეიცავს აგრეთვე MATLAB–ში პროგრამის გამოძახებისა და MAT–ფაილების წაკითხვა–ჩაწერის საშუალებებს.

Simulink – სიმულინკის პაკეტი MATLAB–ის გაფართოების ერთ–ერთი მნიშვნელოვანი შემადგენელი ნაწილია (მისი გაშვების ღილაკი გამოტანილია სისტემის ინსტრუმენტთა პანელზე). სიმულინკი გვაძლევს ანალოგური გამომთვლელი მანქანების იდეის თანამედროვე დონეზე განხორციელების საშუალებას.

სიმულინკი წარმოადგენს ინტერაქტიულ სისტემას დინამიური პროცესების მოდელირებისათვის. მაუსის მეშვეობით შესაძლებელია დიაგრამების ბლოკების გადაადგილება ეკრანზე და მანიპულირება. სიმულინკი მუშაობს წრფივ, არაწრფივ, უწყვეტ, დისკრეტულ, მრავალგანზომილებიან სისტემებთან. მომხმარებლისათვის ეს არის კონსტრუქტორი, რომლის მეშვეობითაც შესაძლებელია ბლოკების (რომლებიც სისტემის ცალკეულ ელემენტებს წარმოადგენენ) გაერთიანება ერთ მთლიანობაში და მიღებული სისტემის მოქმედების შესწავლა დინამიკაში.

სიმულინკის ბაზაზე აგებულია MATLAB–ის ბევრი გაფართოება. **Blocksets –** ეს არის სიმულინკის დამატება, რომელიც შეიძლება განვიხილოთ როგორც დასამოდელირებელი სისტემების კონსტრუირებაში მონაწილე ბლოკების (მოდელების) გაერთიანება. Blocksets შეიცავს მოდელების (ფაილები გაფართოებით mdl) ბიბლიოთეკას სპეციალური მონაცემებისათვის, როგორიცაა სიგნალების დამუშავება, კავშირი, ენერგეტიკული სისტემები.

MATLAB–ის ოჯახში შემავალი პროდუქტების შეთავაზებები მოდელირებაში არ შემოიფარგლება მხოლოდ სისტემის ანალიზით დინამიკაში, აგრეთვე შესაძლებელია შესაძლო შემთხვევების მოდელირება (**Stateflow** პაკეტი), რაც ტექნიკური სისტემების ექსპლუატაციისას სხვადასხვა სიტუაციათა იმიტაციას გვაძლევს.

MATLAB–ში მნიშვნელოვან როლს ასრულებს სპეციალური პროგრამების ჯგუფი **– Toolboxes** (სიტყვასიტყვით ინსტრუმენტთა ყუთი). Toolboxes – ეს MATLAB–ის ფუნქციათა ერთობლიობაა (M–ფაილების), რომლებიც გვაძლევს კერძო სახის ამოცანების ამოხსნის საშუალებას. Toolboxes გამოიყენება სიგნალების დამუშავებისათვის, მოდელირებისათვის და სხვ.

აღვნიშნოთ აგრეთვე MATLAB–ის ისეთი კომპონენტები, როგორიცაა **Code Generation Tools –** კოდების გენერირების ინსტრუმენტალური საშუალებანი, რომელთა მეშვეობით შესაძლებელია C ან Ada ენებზე დამოუკიდებლად შესასრულებადი კოდების შექმნა და აგრეთვე მიკროპროცესორებზე მათი აპარატურული რეალიზაცია; **Data Access Tools –** მონაცემთა ბაზებთან მუშაობის და რეალური დროის რეჟიმში ინფორმაციის შეგროვების პროგრამული უზრუნველყოფა; **Real-Time Workshop**, რომელიც საშუალებას გვაძლევს მოვახდინოთ პროგრამების C–ფორმატში გენერირება დიაგრამების ბლოკების საფუძველზე; **Real-Time Workshop Embedded Coder** – Real-Time Workshop პაკეტის პროგრამების ოპტიმიზაცია მუშაობის სისწრაფის, მეხსიერების გამოყენების, ინტერფეისის სიმარტივის, კოდების ადვილი წაკითხვის თვალსაზრისით; **Signal Processing Blockset –** ციფრული და ანალოგური სიგნალების დამუშავება; **Image Processing Toolbox –** გამოსახულებების დამუშავება; **Robust Control Toolbox –** შემთხვევითი ზემოქმედებების მიმართ მდგრადი მართვის სისტემების ანალიზი და სინთეზი; **Communications Toolbox –** საკომუნიკაციო სისტემების დამუშავება და მოდელირება დროის რეალურ მასშტაბში; **Communications Blockset –** ფუნქციების ბიბლიოთეკა Communications Toolbox–სათვის (მოდულაცია, კოდირება, დეკოდირება); **Control System Toolbox –** უკუკავშირის მქონე ავტომატური რეგულირების სისტემების მოდელირება, ანალიზი და პროექტირება; **RFBlockset** და **RF Toolbox –** უსადენო კავშირგაბმულობის სისტემების მოდელირება და გამოკვლევა; **Simulink Control Design –** მოდელების აგების პროცესის მართვა სიმულინკში; **Simulink Parameter Estimation –** მოდელების პარამეტრების შერჩევა სიმულინკში.

1.2 MATLAB–ის დოკუმენტის ფანჯრის სტრუქტურა

როგორც ყველა პროგრამას, რომელიც მუშაობს *Windows* ოპერაციული სისტემის გარემოში, MATLAB–საც გააჩნია მომხმარებლისთვის მოხერხებული მრავალფანჯრიანი ინტერფეისი. MATLAB–ის გაშვებისას წინასწარი შეთანხმებით მონიტორზე გამოჩნდება კომბინირებული ფანჯარა, რომელიც შეიცავს მენიუს, იარაღების პანელს და ოთხ ჩანართს: **Command Window** (ბრძანებების ფანჯარა), **Command History** (ბრძანებების ისტორია – ადრე შესრულებული ბრძანებების ჩამონათვალი), **Workspace** (სამუშაო არე) და **Current Directory** (მიმდინარე კატალოგი). აღნიშნული პანელების გააქტიურება/გაპასიურება შესაძლებელია ფანჯრის სათაურის სტრიქონზე ან მის სამუშაო სივრცეში დაწკაპუნებით. ფანჯრების ჩვენება/დამალვა შესაძლებელია მთავარი მენიუს **Desktop** პუნქტიდან. ფანჯრების სათაურების სტრიქონში მარჯვნივ 4 ღილაკი მდებარეობს: დაკეცვის, მთელ ეკრანზე გაშლის, ჩაკეტვის მოხსნისა (ამ შემთხვევაში შესაბამისი ფანჯრის ადგილმდებარეობა არ არის მკაცრად დაფიქსირებული და შესაძლებელია მისი გადაადგილება სასურველ ადგილას) და დახურვის. ადვილად შესაძლებელია აგრეთვე ფანჯრების ზომებით მანიპულირება. პროგრამაში მუშაობისას ვააქტიურებთ ზემოთ ჩამონათვალთაგან მხოლოდ იმ ფანჯარას, რომელიც მოცემულ მომენტშია საჭირო, რათა არ გადაიტვირთოს ეკრანი, რაც მუშაობაში ხელისშემშლელია. ეკრანის მარცხენა კუთხეში მდებარეობს MATLAB–ის მთავარი მენიუს **Start** ღილაკი, რომლის მეშვეობით ხდება პროგრამული პაკეტის ნებისმიერ ქვედანაყოფში გადასვლა.

ზემოაღნიშნული ჩანართებიდან ყველაზე მნიშვნელოვანია **Command Window**. აქ ხდება მომხმარებლის მიერ ბრძანებების ჩაწერა და შესრულებაზე გაშვება კლავიატურიდან Enter ღილაკის გამოყენებით. ამავე ფანჯარაში შეგვიძლია საჭირო დახმარების მიღება **doc**, **help** ან **lookfor** ბრძანებების გამოყენებით.

Workspace ფანჯარაში ასახულია მიმდინარე ცვლადები, რომლებიც მომხმარებლის მიერ შექმნილია ბრძანებების ფანჯარაში. სამუშაო არეში წინასწარი შეთანხმებით გამოტანილია მათი სახელები (**Name** ველში), მნიშვნელობები (**Value**–ში), მინიმუმი (**Min**) და მაქსიმუმი (**Max**). თუ გვსურს ცვლადების შესახებ დამატებითი ინფორმაციის მიღება (მონაცემთა ტიპი (**Class**), ზომა (**Size**) და სხვ.), მენიუს **View** პუნქტიდან დავაყენოთ შესაბამისი ველების ჩვენების რეჟიმი. ზოგადად ცვლადების შესახებ ინფორმაციის მიღება (თუმცა არა ისეთი დეტალურის) შესაძლებელია ბრძანებათა ფანჯარაშიც **whos** ბრძანების გამოყენებით. ამის გამო ზედმეტია, სამუშაო არის ფანჯარა მუდმივად გამოტანილი იყოს ეკრანზე. ამ ფანჯარაში ჩახედვა მიზანშეწონილია, თუ გვესაჭიროება რომელიმე მასივის ელემენტების რედაქტირება **Array Editor**–ის (მასივის რედაქტორის) გამოყენებით. ეს ახალი ინსტრუმენტი MATLAB 7–ის ვერსიაშია დამატებული. მისი გამოძახება შესაძლებელია სამუშაო არეში ცვლადის სახელზე ორმაგი დაწკაპუნებით. შესაძლებელია აგრეთვე ცვლადების მნიშვნელობების კორექტირება უშუალოდ სამუშაო არის ფანჯრიდან შესაბამისი ცვლადის კონტექსტური მენიუდან ან მისი ინსტრუმენტთა პანელიდან.

Command History ფანჯარა ინახავს მომხმარებლის მიერ აკრეფილ ყველა ბრძანებას, მაგრამ **Command Window** ფანჯრისგან განსხვავებით აქ ვერ დავინახავთ სისტემურ შეტყობინებებს და გამოთვლების შედეგებს.

Current Directory (მიმდინარე კატალოგი) ფანჯარა გვიჩვენებს მიმდინარე კატალოგს, საიდანაც შესაძლებელია სასურველი ფაილის გამოძახება.

ბრძანებების ფანჯრის მენიუს პუნქტებში თავმოყრილია როგორც ყველა ის სტანდარტული ბრძანება, რომელიც გვხვდება *Windows* ოპერაციული სისტემის გარემოში მომუშავე პროგრამებში, ასევე MATLAB–ისათვის დამახასიათებელი სპეციფიკური ბრძანებები. ასეთი ბრძანებები შემდგომში იქნება განხილული საჭიროებისამებრ.

მთავარი მენიუს ქვემოთ მდებარეობს ინსტრუმენტთა პანელი, რომლის ღილაკების დანიშნულება არ საჭიროებს დამატებით ახსნა–განმარტებას.

MATLAB–ის ბრძანებების ფანჯარაში შესვლისთანავე დავინახავთ >> ნიშანს, რაც იმის მანიშნებელია, რომ სისტემა მზად არის მომხმარებელთან დიალოგისათვის და ელოდება მისგან ბრძანებას. დიალოგური რეჟიმი არის MATLAB–ის მუშაობის ძირითადი რეჟიმი. დიალოგი ხორციელდება ოპერატორების მეშვეობით, რომელთა შესრულება სისტემის მიერ ხდება ინტერპრეტაციის რეჟიმში, ე.ი. ბრძანებათა მიმდევრობის ყოველი სტრიქონი გარდაიქმნება კოდში და მაშინვე სრულდება. აღვნიშნოთ, რომ MATLAB–ს აქვს კომპილატორი, რომელიც ასრულებს დაპროგრამების ენის საშუალებით შექმნილ პროგრამებს.

ოპერატორი წარმოადგენს ბრძანებას ან ფუნქციას ცხადი ან არაცხადი მინიჭებით.

ბრძანება ასრულებს მითითებულ მოქმედებას; როგორც წესი, ბრძანების შესრულებისას არ ხდება გამოსასვლელი ცვლადების ფორმირება. ბრძანების ფორმატი არ შეიცავს ფრჩხილებს. მას შემდეგი სახე აქვს:

ბრძანების_სახელი ოფცია1 ოფცია2 ...

ფუნქცია შეიცავს შესასვლელ და გამოსასვლელ ცვლადებს, რომლებიც ფუნქციის ჩაწერისას ფრჩხილებშია ჩასმული. ფუნქციის ფორმატს შემდეგი სახე აქვს:

[out1, out2, ...] = ფუნქციის_სახელი (arg1, arg2, ...).

აქ out1, out2, ... – გამოსასვლელი ცვლადებია, ხოლო arg1, arg2, ... – შესასვლელი ანუ ფუნქციის არგუმენტები.

გამოსასვლელი ცვლადების არარსებობის შემთხვევაში ფუნქციის ფორმატს ასეთი სახე აქვს: ფუნქციის_სახელი (arg1, arg2, ...). ამ დროს ფუნქცია შეიძლება ჩავწეროთ ბრძანების ფორმატში: ფუნქციის_სახელი arg1 arg2 .... ბრძანებაც ხშირ შემთხვევაში შეიძლება ჩაიწეროს ფუნქციის ფორმატში: ბრძანების_სახელი (ოფცია1, ოფცია2, ...).

აღვნიშნოთ, რომ ეს პრინციპი ყოველთვის არ მოქმედებს, ამიტომაც რეკომენდირებულია ბრძანებებისათვის ვისარგებლოთ ბრძანების ფორმატით, ხოლო ფუნქციებისათვის – ფუნქციის ფორმატით.

ბრძანებების ფანჯრის გარდა, სადაც, როგორც უკვე აღვნიშნეთ, ხდება ბრძანებების და ფუნქციების შეყვანა და აგრეთვე შედეგების მიღება, MATLAB–ს აქვს გრაფიკების ფანჯარაც.

home ბრძანებით ხდება კურსორის გადატანა ბრძანებათა ფანჯრის ზედა მარცხენა კუთხეში.

clear აუქმებს სამუშაო სივრცის ყველა ცვლადს, ხოლო **clear ცვლადების_სახელების_სია** – მითითებულ ცვლადებს.

clf ასუფთავებს გრაფიკების ფანჯარას, ხოლო **clc** ბრძანებით ხდება ბრძანებების ფანჯრის გასუფთავება და კურსორის გადატანა ბრძანებათა ფანჯრის ზედა მარცხენა კუთხეში.

shg ეკრანზე გამოიყვანს გრაფიკულ ფანჯარას.

იმ შემთხვევაში, როდესაც ბრძანებათა ფანჯარა არ ჩანს ეკრანზე (მაგ., მთელი ეკრანი დაკავებულია გრაფიკული ფანჯრით), ნებისმიერ კლავიშზე დაწკაპუნება დააბრუნებს ამ ფანჯარას ეკრანზე.

მომხმარებელს, რომელიც პირველ ნაბიჯებს აკეთებს MATLAB–ში, შეუძლია ისარგებლოს **demo** ბრძანებით იმისათვის, რომ გაეცნოს სადემონსტრაციო პროგრამების სიას, რომლებიც აჩვენებენ MATLAB–ის შესაძლებლობებს.

computer ბრძანება გვაძლევს ინფორმაციას კომპიუტერის ტიპის შესახებ.

who ბრძანებით მიიღება მომხმარებლის მიერ შექმნილი ცვლადების სია, ხოლო **whos** – დამატებითი ინფორმაცია მათი ზომის, მანქანური მოცულობის და კლასის შესახებ.

უკვე მოცემული ბრძანების შესაწყვეტად (მაგ., როცა რომელიმე ოპერატორის გამო კომპიუტერი გაუთავებლად იმეორებს რაიმე ციკლს) ხმარობენ **ctrl+c** კლავიშების კომბინაციას.

help ბრძანებით ხდება იმ საკითხების სიის გამოძახება, რომელთა შესახებ შეგვიძლია მივმართოთ MATLAB–ს დახმარების თხოვნით, ხოლო თუ **help**–ის შემდეგ აკრეფილი იქნება რომელიმე ბრძანების ან ფუნქციის სახელი, მაგ., **help clear**, სისტემა გამოიტანს ინფორმაციას ამ ბრძანების (ფუნქციის) შესახებ.

MATLAB–ი იხსომებს მუშაობის სეანსის დროს გამოთვლილი ყველა ცვლადის მნიშვნელობას. **save** ბრძანებით მომხმარებლის მიერ შექმნილი ცვლადები შეიძლება შენახული იქნას მუშაობის დამთავრებამდე. ცვლადები შეინახება ფაილში, რომლის სახელია matlab.mat. MATLAB–ის ხელახალი გახსნისას **load** ბრძანებით შესაძლებელია შენახული ცვლადების აღდგენა. **save ფაილის_სახელი** ბრძანებით ყველა ცვლადის მნიშვნელობა იქნება შენახული ფაილის_სახელი.mat ფაილში MATLAB–ის ძირითადი კატალოგის work ქვეკატალოგში. **File → Save Workspace As...** გამოიტანს დიალოგურ ფანჯარას, სადაც მისათითებელია ფაილის სახელი და შესანახი კატალოგის სახელი. **save ფაილის_სახელი ცვლადების_სახელების_სია** ბრძანებით შენახული იქნება მითითებული ცვლადები.

MATLAB–ში მუშაობის დასასრულებლად სარგებლობენ ბრძანებებით **quit** ან **exit**.

გარდა იმისა, რომ ოპერატორები შეიძლება მივცეთ ბრძანებების ფანჯრიდან, შეგვიძლია წინასწარ შევქმნათ ოპერატორთა მწკრივი ტექსტური ფაილის სახით, რომლის სახელს უნდა ჰქონდეს გაფართოება .m; ასეთ ფაილებს ეწოდება M–ფაილები. ფაილის სახით შენახული ოპერატორთა ასეთი მწკრივი შეიძლება შემდგომშიც გამოვიყენოთ. M–ფაილი წარმოადგენს

ტექსტურ ASCII ფაილს, რომელიც შეიცავს MATLAB–ის კოდებს. M–ფაილი შეიძლება შევქმნათ ტექსტურ რედაქტორში, რომლის გამოძახება ხდება მენიუს სტრიქონიდან **File → New → M-file** ბრძანებით. სამუშაო სივრცეში M–ფაილის გამოძახება ხდება მისი სახელის აკრეფით ბრძანებათა სტრიქონში, რის შემდეგ უნდა დავაჭიროთ Enter კლავიშს. აღსანიშნავია, რომ ამ დროს M–ფაილი უნდა იმყოფებოდეს ხელმისაწვდომ დირექტორიაში. თუ გვინდა გამოვიყვანოთ M–ფაილის ტექსტი ბრძანებების ფანჯარაში, უნდა ვისარგებლოთ **echo** ბრძანებით.

M–ფაილები ფართოდ გამოიყენება ახალი ფუნქციების შესაქმნელად.

what ბრძანებით მიიღება იმ ფაილების სია, რომლებიც განთავსებულია მიმდინარე დირექტორიაში, ხოლო **type** ბრძანება გვაძლევს მითითებული ფაილის შინაარსს.

1.3 ცვლადები, კონსტანტები, საკვანძო სიტყვები. ელემენტარული მათემატიკური ფუნქციები

MATLAB–ში მონაცემთა ძირითადი ტიპები არის მუდმივები (კონსტანტები) და ცვლადები.

ცვლადი არის სახელის მქონე ობიექტი, რომელსაც გააჩნია მნიშვნელობა და ეს მნიშვნელობა შეიძლება შეიცვალოს პროგრამის შესრულების მსვლელობისას.

ცვლადის ტიპი განისაზღვრება მისი მნიშვნელობით. იგი შეიძლება იყოს: **single** – ერთმაგი სიზუსტის რიცხვითი მასივი, **double** – ორმაგი სიზუსტის რიცხვითი მასივი, **char** – სტრიქონული, **logic** – ლოგიკური, **sparse** – ორმაგი სიზუსტის გამეჩხრებული მატრიცა, **cell** – უჯრედების მასივი, **structure** – ველებიანი სტრუქტურების მასივი, **function handle** – ფუნქციათა დესკრიპტორები, გამოიყენება ფუნქციების არაპირდაპირი გამოძახებისას, **int8**, **int16**, **int32** – 8-, 16 და 32 თანრიგიანი მთელი რიცხვები.

იდენტიფიკატორი (ცვლადის ან სხვა ობიექტის სახელი) წარმოადგენს ასოთი დაწყებულ ასოების, ციფრების და ქვეშხაზგასმის სიმბოლოების მიმდევრობას. მაგ., a, x1, alfa, Y_coordinate. იდენტიფიკატორის ჩანაწერში დაუშვებელია ჰარის გამოყენება. სისტემა განასხვავებს ნუსხურ და მთავრულ ასოებს. აღვნიშნოთ აგრეთვე, რომ სასურველია, იდენტიფიკატორი იყოს უნიკალური ანუ არ არის მიზანშეწონილი, თუმცა დასაშვებია, იდენტიფიკატორად ვიხმაროთ, მაგალითად, sin, რადგან ის ჩაშენებული ფუნქციის სახელია. იდენტიფიკატორში სიმბოლოთა რაოდენობა შეზღუდულია და მისი სიგრძე დამოკიდებულია MATLAB–ის ვერსიაზე. **namelenghthmax** ფუნქციის მეშვეობით შესაძლებელია ცვლადის სახელის მაქსიმალური დასაშვები სიგრძის გაგება. MATLAB 7.5.0 ვერსიისათვის

```
>> N=namelengthmax
N=
    63
```

MATLAB–ის ერთ–ერთი ყველაზე გავრცელებული ოპერატორია **მინიჭების ოპერატორი**. გამოთვლების შედეგი მიენიჭება ცვლადს, რომელიც მინიჭების ოპერატორში ჩაწერილია ტოლობის ნიშნის მარცხნივ. ასეთ მინიჭებას **ცხადი მინიჭება** ჰქვია. თუ ოპერატორს არ გააჩნია მარცხენა ნაწილში ცვლადი, MATLAB–ი ავტომატურად ქმნის სპეციალურ **ans (answer)** ცვლადს და ანიჭებს მას შედეგის მნიშვნელობას. ასეთ მინიჭებას **არაცხადი მინიჭება** ჰქვია. ans ცვლადის მნიშვნელობა შენახული იქნება მომდევნო არაცხად მინიჭებამდე. შესაძლებელია მისი როგორც ცვლადის გამოყენება.

შემდეგი მაგალითიდან ჩანს, რომ ცვლადის მნიშვნელობის გარეშე არითმეტიკული გამოსახულების შეყვანისას MATLAB–ის სისტემამ ჩაწერა შედეგი ans–ში.

```
>> sin(pi/6)
ans =
   0.5000
```

გამოსათვლელ ფორმულებში გამოიყენება ნიშნები: + (მიმატება); – (გამოკლება); * (გამრავლება); / (გაყოფა); ^ (ახარისხება). მათი შესრულების თანამიმდევრობა სტანდარტულია. თანამიმდევრობის შეცვლა შესაძლებელია ფრჩხილების დახმარებით.

კონსტანტა არის სახელის მქონე ობიექტი, რომლის მნიშვნელობა არ იცვლება. კონსტანტები შეიძლება სხვადასხვა ტიპის იყოს.

რიცხვითი კონსტანტების მაგალითებია: –682, 86.4, 5.5*10^2. MATLAB–ს შეუძლია მუშაობა როგორც ნამდვილ, ისე კომპლექსურ რიცხვებთან. კომპლექსური რიცხვითი კონსტანტების მაგალითებია: 2+3i, 3–10j, სადაც i და j სიმბოლოებით აღნიშნულია წარმოსახვითი ერთიანი ანუ $\sqrt{-1}$. კომპლექსური რიცხვების ჩაწერა შესაძლებელია სხვა ფორმითაც: 3+2i და complex(3,2) ერთ კომპლექსურ მნიშვნელობას ასახავს. კომპლექსური რიცხვების გამოყენება შეიძლება ელემენტარული ფუნქციების არგუმენტად. კომპლექსურ–შეუღლებული გამოსახულების გამოთვლა ხდება აპოსტროფით:

```
>> 5-3i'
ans =
   5.0000 + 3.0000i
```

სტრიქონული ტიპის მონაცემები წარმოადგენენ სიმბოლოების მიმდევრობას აპოსტროფებში, მაგ., 'Insert Matrix', 'The rezult is valid'. სტრიქონული ტიპის კონსტანტა შეიძლება შედგებოდეს ერთი ან რამდენიმე სიმბოლოსაგან ან სიტყვისაგან.

MATLAB–ში არსებობს სპეციალური კონსტანტებიც.

pi სპეციალურ კონსტანტას MATLAB–ი აღიქვამს როგორც ჩვეულებრივ ® რიცხვს. IEEE სტანდარტით pi=3.1416 short ფორმატით წარმოდგენისას; pi=3.141592653589793 long ფორმატით წარმოდგენისას და ა.შ.

i,j კონსტანტები გამოიყენება კომპლექსური რიცხვების ჩასაწერად. უნდა აღვნიშნოთ, რომ i და j სიმბოლოები შეიძლება წარმოადგენდნენ ციკლის პარამეტრს, მასივის ელემენტების ინდექსს ან სულადაც ჩვეულებრივ ცვლადსაც.

Inf სპეციალური კონსტანტაა, რომელიც IEEE სტანდარტით წარმოადგენს უსასრულობას. მაგ.,

```
>> a=1; a/0
ans =
  Inf
```

NaN – სპეციალური კონსტანტაა, რომლის მეშვეობით ფიქსირდება შედეგი, რომელიც არ წარმოადგენს რიცხვს. NaN კონსტანტა მიიღება, მაგ., 0*Inf გამრავლებით, Inf/Inf ან 0/0 გაყოფით, ნებისმიერი არითმეტიკული მოქმედების შესრულებისას NaN კონსტანტასთან ერთად და სხვ.

eps – რიცხვის მოცემის სიზუსტეა მცოცავი მძიმის რეჟიმში. ეს არის სხვაობა 1.0–სა და მის მომდევნო უახლოეს ათწილადს შორის, რომლის წარმოება შეუძლია MATLAB–ს.

საკვანძო სიტყვები

MATLAB–ის ენაში წარმოდგენილია 19 საკვანძო სიტყვა, რომლებიც გამოიყენება ოპერატორების ფორმირებისას, მაგ, continue, if, global და სხვ. ეს სიტყვები შესაძლებელია გამოვიყენოთ მხოლოდ პირდაპირი მნიშვნელობით. მათი გამოყენება სხვა მიზნებისათვის დაუშვებელია. მაგ., ცვლადის მნიშვნელობად global საკვანძო სიტყვის გამოყენებით გამოდის შეტყობინება შეცდომის შესახებ:

```
>> global=5
??? global=5
Error: The expression to the left of the equals sign is not a valid target for an assignment.
```

iskeyword ბრძანების მეშვეობით შესაძლებელია შემოწმდეს, წარმოადგენს თუ არა სტრიქონი საკვანძო სიტყვას. ამასთან, iskeyword–ით მიიღება MATLAB–ის ყველა საკვანძო სიტყვების ჩამონათვალი, ხოლო iskeyword('S') ფუნქციით მივიღებთ 1–ს, თუ S წარმოადგენს საკვანძო სიტყვას, 0–ს – წინააღმდეგ შემთხვევაში:

```
>> P=iskeyword('otherwise')
P =
    1
>> F=iskeyword('PAL')
F =
    0
```

MATLAB–ს აქვს მრავალი **ელემენტარული ფუნქცია**. **help elfun** ბრძანებით ეკრანზე გამოვა ყველა ელემენტარული ფუნქციის ჩამონათვალი მოკლე აღწერით, ხოლო **help specfun** ბრძანებით – სპეციალური ფუნქციების სია. MATLAB–ის ძირითადი ელემენტარული ფუნქციებია (ფუნქციების არგუმენტები შეიძლება იყოს როგორც სკალარული სიდიდეები, ისე მასივებიც):

abs(x) – x ცვლადის აბსოლუტური სიდიდე;

sqrt(x) – კვადრატული ფესვი x სიდიდიდან;

round(x) – x სიდიდის უახლოეს მთელამდე დამრგვალება;

fix(x) – x სიდიდის უახლოეს მთელამდე დამრგვალება 0 მიმართულებით;

floor(x) – x სიდიდის უახლოეს მთელამდე დამრგვალება $-\infty$ მიმართულებით;

ceil(x) – x სიდიდის უახლოეს მთელამდე დამრგვალება $+\infty$ მიმართულებით;

sign(x) – x–ის ნიშანი: sign(x) გვაძლევს –1, თუ x<0, 0 – თუ x=0, 1 – თუ x>0;

rem(x,y) – x სიდიდის y–ზე გაყოფის შედეგად მიღებული ნაშთი. rem(x,y)= x–n*y, სადაც n = fix(x/y), $y \neq 0$. rem(x,0) გვაძლევს NaN–ს. თუ $x \neq y$ და $y \neq 0$, rem(x,y) ფუნქციის ნიშანი ემთხვევა x–ის ნიშანს;

mod(x,y) = x–n*y, სადაც n = floor(x/y), $y \neq 0$. თუ $x \neq y$ და $y \neq 0$, mod(x,y) ფუნქციის ნიშანი ემთხვევა y–ის ნიშანს. mod(x,y) და rem(x,y) ფუნქციები გვაძლევს ერთსა და იმავე შედეგს, თუ x და y–ის ნიშნები ერთნაირია;

exp(x) – ექსპონენციალური ფუნქცია e^x, სადაც e ნატურალური ლოგარითმის ფუძეა (ნეპერის რიცხვი), $e \approx 2.718282$;

log(x), **log10(x)**, **log2(x)** – x ცვლადის ლოგარითმი ნატურალური, ათობითი და ორობითი ფუძით;

pow2(x) – მაჩვენებლიანი ფუნქცია 2^x;

sin(x), **cos(x)**, **tan(x)**, **cot(x)**, **sec(x)**, **csc(x)** – x ცვლადის სინუსი, კოსინუსი, ტანგენსი, კოტანგენსი, სეკანსი და კოსეკანსი; ამ ფუნქციების არგუმენტს MATLAB–ი აღიქვამს რადიანებში. იმისათვის, რომ რადიანი გადავიყვანოთ გრადუსებში ან პირიქით, გამოვიყენოთ შემდეგი ტოლობები:

კუთხე_გრადუსებში = კუთხე_რადიანებში*(180/pi);

კუთხე_რადიანებში = კუთხე_გრადუსებში *(pi/180);

sind(x), **cosd(x)**, **tand(x)**, **cotd(x)**, **secd(x)**, **cscd(x)** – x ცვლადის სინუსი, კოსინუსი, ტანგენსი, კოტანგენსი, სეკანსი და კოსეკანსი, როდესაც არგუმენტი მოცემულია გრადუსებში;

sinh(x), **cosh(x)**, **tanh(x)**, **coth(x)**, **sech(x)**, **csch(x)** – x ცვლადის ჰიპერბოლური სინუსი, კოსინუსი, ტანგენსი, კოტანგენსი, სეკანსი და კოსეკანსი;

asin(x) – x ცვლადის არკსინუსი, სადაც $-1 \le x \le 1$. ფუნქცია გვაძლევს კუთხეს რადიანებში [–π/2 ; π/2] ინტერვალში;

acos(x) – x ცვლადის არკკოსინუსი, სადაც $-1 \le x \le 1$. ფუნქცია გვაძლევს კუთხეს რადიანებში [0 ; π] ინტერვალში;

atan(x) – x ცვლადის არკტანგენსი. ფუნქცია გვაძლევს კუთხეს რადიანებში (–π/2 ; π/2) საზღვრებში;

acot(x), **asec(x)**, **acsc –** არკკოტანგენსი, არკსეკანსი და არკკოსეკანსი რადიანებში;

atan2(y,x) y/x სიდიდის არკტანგენსი, ამასთან, როდესაც x=0, მიღებული პასუხი იქნება π/2 (დადებითი y–ის შემთხვევაში) ან –π/2 (უარყოფითი y–ის შემთხვევაში). ფუნქცია იძლევა კუთხეს რადიანებში, რომელიც მდებარეობს [–π; + π] საზღვრებში, x და y ნიშნების მიხედვით;

asind(x), **acosd(x)**, **atand(x)**, **acotd(x)**, **asecd(x)**, **acscd(x)** – x ცვლადის არკსინუსი, არკკოსინუსი, არკტანგენსი, არკკოტანგენსი, არკსეკანსი და არკკოსეკანსი გრადუსებში;

asinh(x), **acosh(x)**, **atanh(x)**, **acoth(x)**, **asech(x)**, **acsch(x)** – x ცვლადის ჰიპერბოლური არკსინუსი, არკკოსინუსი, არკტანგენსი, არკკოტანგენსი, არკსეკანსი და არკკოსეკანსი.

1.4 ოპერატორების შეტანა და რედაქტირება

ბრძანებების ფანჯრის სტრიქონს ***ბრძანებათა სტრიქონი*** ჰქვია. MATLAB–ის ოპერატორები ჩვეულებრივ იკრიფება ახალ სტრიქონში, მაგრამ შესაძლებელია ერთ სტრიქონში რამდენიმე ოპერატორის ერთად ჩაწერა. ამ შემთხვევაში მათ მძიმით ან წერტილ–მძიმით გამოყოფენ. ამ უკანასკნელის გამოყენებისას მონაცემები პროგრამის მიერ აღქმული იქნება, მაგრამ მათ ვერ ვნახავთ ეკრანზე. ეს განსხვავება ნათლად ჩანს შემდეგი მაგალითიდან:

```
>> a=3, b=2,
a =
    3
b =
    2
>> a=3, b=2;
a =
    3
```

თუ ოპერატორის ჩაწერა ვერ ხერხდება ერთ სტრიქონზე მისი სიგრძის გამო, შესაძლებელია სტრიქონის ბოლოს სამი წერტილის (...) გამოყენება, რაც იმის მანიშნებელია, რომ მეორე სტრიქონზე გრძელდება წინა სტრიქონის ოპერატორის ჩაწერა. თუ სამი წერტილის მარჯვნივ იგივე სტრიქონში რაიმეს ავკრეფთ, შეტანილი ინფორმაცია MATLAB–ის მიერ იქნება აღქმული როგორც კომენტარი.

თუ გვსურს რამდენიმე სტრიქონის ერთი სტრიქონის სახით წარმოდგენა, ყოველი სტრიქონი დავამთავროთ Shift + Enter კლავიშების კომბინაციით.

ტექსტი, რომელიც მოყვება % ნიშანს, MATLAB–ის მიერ იგნორირდება და გამოიყენება კომენტარისათვის.

სიმბოლოების სტრიქონის, კომენტარების და სხვა ელემენტების ჩაწერისას MATLAB–ი იყენებს სხვადასხვა ფერს მათი შეყვანის სისწორის კონტროლისათვის:

- სიმბოლოების სტრიქონი, რომელიც იწერება აპოსტროფებში, გამოიტანება მოყავისფრო ფერში, ხოლო დასახური აპოსტროფის ნიშნის აკრეფა ცვლის სიმბოლოებს ფერს იისფრად;
- საკვანძო სიტყვები და ასევე სტრიქონის გაგრძელების სიმბოლო (...) გამოიტანება ლურჯად;
- კომენტარი მწვანე ფერად არის წარმოდგენილი;
- ოპერაციული სისტემის ბრძანებები, რომლებიც ძახილის ნიშნით იწყება, გამოიტანება ოქროსფრად;
- ჩანაწერში დაშვებული სინტაქსური შეცდომის შესახებ ინფორმაცია გამოიტანება წითლად.

ფერების შეცვლა შესაძლებელია **Preferences** დიალოგურ ფანჯარაში, რომელშიც შევდივართ მთავარი მენიუს **File** ì **Preferences...** ბრძანების მეშვეობით.

MATLAB–ში გამოიყენება რედაქტირებისთვის განკუთვნილი ყველა ის მოქმედებანი, რომლებიც Windows ოპერაციული სისტემის გარემოში მომუშავე პროგრამებშია მიღებული:

კლავიში	ბრძანება	დანიშნულება
↑	Ctrl+P	წინა ბრძანებათა სტრიქონის გამოძახება
↓	Ctrl+N	მომდევნო ბრძანებათა სტრიქონის გამოძახება რამდენიმეჯერ გამოყენებული ↑ კლავიშის ან Ctrl+P ბრძანების შემდეგ
→	Ctrl+F	კურსორის გადაადგილება ერთი სიმბოლოთი მარჯვნივ
←	Ctrl+B	კურსორის გადაადგილება ერთი სიმბოლოთი მარცხნივ
	Ctrl+→	კურსორის გადაადგილება ერთი სიტყვით მარჯვნივ
	Ctrl+←	კურსორის გადაადგილება ერთი სიტყვით მარცხნივ
Home	Ctrl+A	კურსორის გადაადგილება ბრძანებათა სტრიქონის დასაწყისში
End	Ctrl+E	კურსორის გადადგილება ბრძანებათა სტრიქონის ბოლოში
	Ctrl+ Home	კურსორის გადაადგილება ბრძანებათა ფანჯრის პირველ სტრიქონზე
	Ctrl+ End	კურსორის გადაადგილება ბრძანებათა ფანჯრის ბოლო სტრიქონზე
Esc	Ctrl+U	ბრძანებათა სტრიქონის წაშლა

1.5 მონაცემთა ფორმატირება

რიცხვითი მონაცემები MATLAB–ში წარმოდგენილია ნამდვილი ან კომპლექსური რიცხვებით. ნამდვილი რიცხვი იკავებს ოპერატიულ მეხსიერებაში 8 ბაიტს და იღებს მნიშვნელობებს $[10^{-308};10^{+308}]$ დიაპაზონიდან. ამ დროს ციფრების რაოდენობა ათობითი წერტილის შემდეგ 16 – 17–ია. კომპლექსური რიცხვი ასეთი წარმოდგენისას შესაბამისად 16 ბაიტს იკავებს. სწორედ ამ სიზუსტით ასრულებს MATLAB–ი გამოთვლებს.

ფორმატირება არ ახდენს გავლენას გამოთვლების მსვლელობაზე, მასზე დამოკიდებულია მხოლოდ და მხოლოდ გამოსატანი მონაცემების წარმოდგენის ფორმა. რიცხვების გამოტანისას ხდება ამ რიცხვების დამრგვალება და ასახვა დისპლეიზე მითითებული ფორმატის თანახმად.

მონაცემები, რომლებიც ეკუთვნის მთელ რიცხვთა კლასს (integer), ბუნებრივია, გამოდის როგორც მთელი, ხოლო ათწილადები წინასწარი შეთანმებით გამოდის ე.წ. მოკლე ფორმატით – ათობითი წერტილის შემდეგ 4 ნიშნით, თუ ჩვენ თვითონ არ მივუთითებთ განსხვავებულ ფორმატს. თუ გვსურს გამოვიყვანოთ რიცხვითი გამოსახულება მძიმის შემდეგ

15 ნიშნით, ვსარგებლობთ ბრძანებით **format long**. თუ კვლავ მოკლე ფორმატს გვინდა დავუბრუნდეთ – **format short**:

```
>>  format short; b=-5.63858
b =
   -5.6386
>> format long
 >> b
b =
  -5.638580000000000
```

format long და format short არ ახდენენ გავლენას მთელი რიცხვების გამოსახვაზე.

როცა რიცხვითი გამოსახულება ძალიან დიდია ან ძალზე მცირე, ჩაწერის ათწილადური ფორმა მოუხერხებელია. მაგალითად, ქიმიაში ხშირად გამოიყენება ავოგადროს რიცხვი, რომლის მნიშვნელობა უდრის 602300000000000000000000–ს. ასეთი რიცხვის ჩაწერისას მიმართავენ ექსპონენციალურ (მაჩვენებლიან) ფორმას: $6.023 \cdot 10^{23}$. MATLAB–ში ასეთი ფორმით რიცხვის ჩაწერისას მანტისა და ექსპონენტა გაყოფილია e ნიშნით. თუ გვსურს რიცხვის გამოტანა ექსპონენციალური ფორმატით, ვსარგებლობთ ბრძანებით: **format short e** ან **format long e**. თუ მიმდინარე ფორმატით რიცხვის ჩაწერა ვერ ხერხდება, MATLAB–ი თვითონ გამოიყენებს ექსპონენციალურ ფორმატს. ავოგადროს რიცხვის შემთხვევაში ყოველგვარი დამატებითი მითითების გარეშე ის გამოტანილი იქნება როგორც 6.0230e+023, გამოსახულება -0.000003*2 – როგორც -6.0000e-006.

format short g და **format long g** ბრძანებების შემთხვევაში **ა**ირჩევა საუკეთესო ვარიანტი რიცხვის ფიქსირებული ან ექსპონენციალური ფორმით წარმოდგენებს შორის.

format + ბრძანების შემთხვევაში დადებითი და უარყოფითი რიცხვების მაგიერ დაიბეჭდება + ან – ნიშანი, ხოლო თუ რიცვის მნიშვნელობა 0–ის ტოლია, მის მაგიერ ცარიელი ადგილი დარჩება. კომპლექსური რიცხვის წარმოსახვითი ნაწილი ამ ფორმატში იგნორირდება:

```
>> format +;  x=[-25 3 -6 -2 0 -4];  x
x =
-+-- -
```

format bank არის საბანკო ფორმატი ორი ციფრით რიცხვის ათობოთ ნაწილში:

```
>> format bank;  z=1200; z
z =
      1200.00
```

format rat გამოიყვანს რიცხვს რაციონალური წილადის სახით, ამასთან, თუ წილადი შესაკვეცია, მოახდენს შეკვეცას:

```
>> format rat;  y=1.55
y =
    31/20
```

format hex განკუთვნილია რიცხვების გამოსატანად ათვლის თექვსმეტობით სისტემაში:

```
>> format hex;  a=1
```

a =

3ff0000000000000

format ბრძანება ოფციების გარეშე გააუქმებს ფორმატირების წინა ბრძანებებს და დააყენებს წინასწარი შეთანხმების ფორმატებს.

format compact ბრძანება დააყენებს კომპაქტურად, ზედმეტი ცარიელი სტრიქონების გარეშე, ინფორმაციის გამოტანის რეჟიმს, ხოლო **format loose** დააბრუნებს არამკვრივ გამოტანას.

ფორმატების დაყენება შესაძლებელია აგრეთვე **Preferences** დიალოგურ ფანჯარაში.

ტექსტის ეკრანზე გამოსაყვანად MATLAB–ს აქვს ბრძანება **disp**, რომლის შემდეგ ფრჩხილებში იწერება ბრჭყალებში მოთავსებული ტექსტი. ეს ბრძანება ასევე გამოიყენება მატრიცის, ვექტორის ან სკალარის ეკრანზე გამოსაყვანად. მაგ., თუ სკალარი სახელით temp შეიცავს ტემპერატურის რიცხვით მნიშვნელობას 25 (ცელსიუსის გარდუსებში), ბრძანება disp(temp); disp('degrees C') გვაძლევს:

25

degrees C

ქვემოთ მოყვანილ მაგალითში ჯერ ხდება სათაურის სტრიქონის ფორმირება, ხოლო შემდეგ ტრიგონომეტრიული ფუნქციების მნიშვნელობათა ცხრილის გამოტანა (აქ blanks(n) – ცარიელი n პოზიციის გამოყვანაა, ხოლო x ვექტორის ტრანსპონირება საჭიროა მასივების ელემენტების სვეტებად წარმოდგენისათვის):

```
>> headers=[blanks(10)  'angle'  blanks(10)  'sin'  blanks(10)  'cos'  blanks(10)  'tg'];
>> format short g; x=(0:pi/6:2*pi)'; t=[x/pi*180  sin(x)  cos(x)  tan(x)];disp(headers),disp(t)
```

angle	sin	cos	tg
0	0	1	0
30	0.5	0.86603	0.57735
60	0.86603	0.5	1.7321
90	1	6.1232e-017	1.6331e+016
120	0.86603	-0.5	-1.7321
150	0.5	-0.86603	-0.57735
180	1.2246e-016	-1	-1.2246e-016
210	-0.5	-0.86603	0.57735
240	-0.86603	-0.5	1.7321
270	-1	-1.837e-016	5.4437e+015
300	-0.86603	0.5	-1.7321
330	-0.5	0.86603	-0.57735
360	-2.44936e-016	1	-2.4493e-016

ფორმატირების უფრო ფართო საშუალებებს იძლევა ოპერატორი **fprintf**. იგი ერთდროულად ტექსტის და მატრიცის დაბეჭდვისა და რიცხვითი გამოსახულების ფორმატირების საშუალებას იძლევა. მისი ზოგადი ფორმა ასეთია: **fprintf (იდენტიფიკატორი, 'ფორმატები', ცვლადები)**, სადაც ფორმატები შეიცავს ბრჭყალებში ჩასმულ ტექსტს და

ფორმატის სპეციფიკაციებს. თუ ფაილის იდენტიფიკატორი არ არის მითითებული, ცვლადების მნიშვნელობების გამოტანა ხდება ბრძანებათა ფანჯარაში.

ტექსტის შიგნით %e, %f და %g გამოიყენება იმისათვის, რომ მივუთითოთ, როგორი სახით უნდა დაიბეჭდოს მატრიცის რიცხვითი მნიშვნელობები: %e შემთხვევაში რიცხვითი გამოსახულება მოიცემა ექსპონენციალური ფორმით, %f შემთხვევაში – ათწილადური ფორმით, %g შემთხვევაში კი აირჩევა უმოკლესი ფორმა ექსპონენციალურსა და ათწილადურ ფორმებს შორის. თუ ტექსტში არის ნიშანი \n, ეს ნიშნავს, რომ მის მარჯვნივ მდგომი ინფორმაცია გადავა შემდეგ ხაზზე. მაგ., ბრძანებებით:

```
>> temp=-5;
>> fprintf ( 'The temperature is %f degrees C \n ', temp)
```

მივიღებთ:

```
The temperature is -5.000000 degrees C
```

თუ ოპერატორს ასე შევცვლით:

```
>> fprintf ( 'The temperature is  \n %f degrees C \n', temp)
```

მივიღებთ:

```
The temperature is
 -5.000000 degrees C
```

ფორმატის განმსაზღვრელი ნიშნები %f, %g, %e შეიძლება შეიცავდეს ინფორმაციას თუ რამდენი ნიშანი უნდა ჰქონდეს რიცხვს და რამდენი წილადური ციფრი დაიბეჭდოს ათობითი წერტილის შემდეგ. მაგ., ოპერატორი

```
>> fprintf ( 'The temperature is  %3.1f degrees C \n ', temp)
```

იძლევა:

```
The temperature is  -5.0 degrees C
```

ფორმატებში პროცენტის ნიშნის შემდეგ დამატებით შეიძლება განლაგდეს შემდეგი სიმბოლოები: + (დადებითი რიცხვის ნიშნის გამოსატანად); – (რიცხვის ჩასაწერად გამოყოფილი ველის მარცხენა კიდეზე); 0 (ველში დარჩენილი მარცხენა პოზიციების ნულებით შესავსებად). მაგ., ოპერატორებით

```
>> x=6; y=2.5; z=3.1;
fprintf('x=%+6.1f\n y=%-6.1f\n z=%06.1f\n',x,y,z)
```

მივიღებთ:

```
 x=  +6.0
 y=2.5
 z=0003.1
```

თუ გამოსატანი ცვლადი მატრიცაა, ფორმატების მითითება ხდება სვეტების მიხედვით, მაგ.,

```
>> A=[2.5  7.5  12]; fprintf('%6.2f  %6.2f  %6.2f\n',A)
```

მოგვცემს

```
 2.50    7.50   12.00
```

თავი II მატრიცებთან მუშაობა

2.1 მატრიცების შეტანის ხერხები

მასივი არის სახელის მქონე ერთგვაროვან მონაცემთა დალაგებული ერთობლიობა. მასივი შეიძლება იყოს სხვადასხვა განზომილების (ერთგანზომილებიანი, ორგანზომილებიანი და ა.შ) და ზომის (ყოველი განზომილების მიმართ ელემენტების რაოდენობა). მასივის ელემენტების ნუმერაცია იწყება 1–დან.

მატრიცა არის სტრიქონების და სვეტების სახით დალაგებული მონაცემთა მწკრივი, რომლისთვისაც შემოღებულია მოქმედებათა წარმოების სპეციალური წესები; ორგანზომილებიანი მასივებისათვის იგივე მოქმედებები სხვანაირადაა განსაზღვრული (იხ.2.2). მატრიცა, რომელიც შედგება ერთი სტრიქონისა და ერთი სვეტისაგან, წარმოადგენს სკალარულ სიდიდეს, ერთი სტრიქონისა და რამდენიმე სვეტისაგან შემდგარი მატრიცა – ვექტორ–სტრიქონს, ხოლო ერთი სვეტისა და რამდენიმე სტრიქონისაგან – ვექტორ–სვეტს.

ndims(A) ჩაშენებული ფუნქცია გამოიყენება A მასივის განზომილების, **size(A)** – ზომის, **length(A)** – ვექტორის სიგრძის დასადგენად (აღვნიშნოთ, რომ მატრიცის შემთხვევაში length(A) ფუნქცია მოგვცემს სტრიქონის სიგრძეს ანუ სტრიქონში ელემენტების რაოდენობას), ხოლო **numel(A)** ფუნქციით მივიღებთ A მასივში ელემენტების რაოდენობის მნიშვნელობას.

არსებობს მატრიცების (მასივების) შეტანის რამდენიმე ხერხი:

- მატრიცის ყველა ელემენტის თანმიმდევრული შეტანა მინიჭების ოპერატორის გამოყენებით;
- მატრიცის წაკითხვა ფაილიდან;
- კოლონოპერატორის – ორწერტილოვანი (:) ოპერატორის გამოყენება;
- მატრიცის შეტანა კლავიატურიდან **input** ოპერატორის საშუალებით.

მინიჭების ოპერატორის გამოყენებით მატრიცის შექმნის დროს ჯერ იწერება მატრიცის სახელი, შემდეგ ტოლობის ნიშანი, რომლის მარჯვნივ კვადრატულ ფრჩხილებში ჩაისმება მატრიცის ელემენტების მნიშვნელობები. ამასთან, სტრიქონის ელემენტები ერთმანეთისაგან გამოიყოფა ჰარით ან მძიმით, ხოლო სტრიქონები – წერტილ–მძიმით. თუ ოპერატორის ბოლოში არ იქნება დასმული წერტილ–მძიმე, Enter კლავიშზე დაწკაპუნების შემდეგ მოხდება მატრიცის ავტომატური გამოტანა ეკრანზე, წინააღმდეგ შემთხვევაში მატრიცა იქნება დამახსოვრებული, მაგრამ არ იქნება გამოტანილი ეკრანზე.

მაგ.,

```
>> Z=[1 2.5 6;  4 -5 6.1]
Z =
   1.0000    2.5000    6.0000
   4.0000   -5.0000    6.1000
```

>> P=[1 -23 4 0];

მატრიცის შეტანისას შეიძლება სტრიქონები წერტილ–მძიმით კი არ გამოვყოთ ერთმანეთისაგან, არამედ გადავიტანოთ შემდეგ ხაზზე Enter კლავიშის მეშვეობით:

>> Q=[2 3 -3
1 4 5
-3 2 5];

თუ მატრიცის სტრიქონში ბევრი ელემენტია, მათი ჩაწერა შეიძლება გაგრძელდეს მომდევნო ხაზზე. ამისათვის წინა ხაზზე გამყოფის (ჰარის, მძიმის ან წერტილ–მძიმის) შემდეგ ავკრიფოთ სამი წერტილი. მაგ.,

>> W=[1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0 10 0];

იგივეა, რაც

>> W=[1 0 2 0 3 0 ...
4 0 5 0 6 0 ...
7 0 8 0 9 0 10 0];

A მატრიცის i–ური სტრიქონისა და j–ური სვეტის გადაკვეთაზე მდგომი ელემენტი შემდეგნაირად აღინიშნება: A(i,j). მატრიცის ელემენტთან მიმართვა შესაძლებელია ერთი ინდექსის გამოყენებითაც. ასეთი მიმართვისას მატრიცა განიხილება როგორც საწყისი მატრიცის სვეტებისაგან ფორმირებული გრძელი ვექტორი. ასე რომ, მაგ., ზემოთ მოყვანილი Z მატრიცისთვის Z(2,1)–ის მნიშვნელობა იგივეა, რაც Z(2)–ის და უდრის 4.0000–ს.

ინდექსიანი ცვლადების მეშვეობით შესაძლებელია მატრიცის ელემენტების მნიშვნელობების შეცვლა. მაგ., ოპერატორი

>> Z(2,3)=2

შეცვლის შესაბამისი ელემენტის მნიშვნელობას და Z მატრიცა გახდება

Z=
1.0000 2.5000 6.0000
4.0000 -5.0000 2.0000

ასევე შესაძლებელია მატრიცას დავუმატოთ ახალი ელემენტები. ამ დროს მატრიცის ზომა იზრდება. მაგ., ოპერატორი Z(3,2)=7 მოგვცემს შემდეგ მატრიცას:

Z =
1.0000 2.5000 6.0000
4.0000 -5.0000 2.0000
0 7.0000 0

როგორც ვხედავთ, მატრიცის ზომა გაიზარდა, ხოლო ის ახალი ელემენტები, რომელთა მნიშვნელობა არ იყო ცხადად განსაზღვრული, გახდა 0–ის ტოლი.

ახალი მატრიცის განსაზღვრა შესაძლებელია უკვე შექმნილი მატრიცის ბაზაზე. მაგ., ოპერატორები

>> Y=[0 1 1]; Z=[Y;Z]

შემდეგნაირად გარდაქმნის Z მატრიცას:

Z =

```
     0       1.0000       1.0000
1.0000       2.5000       6.0000
4.0000      -5.0000       2.0000
     0       7.0000            0
```

ოპერატორები

```
>> K=[2.4  8.25];  M=[4.5 3.0  K];
```

ეს იგივეა, რაც

```
>>  M=[4.5 3.0  2.4  8.25];
```

აქვე აღვნიშნოთ, რომ არარსებული ინდექსების მქონე ელემენტთან მიმართვა დაუშვებელია. ამ შემთხვევაში MATLAB–ი მოგვცემს შეტყობინებას შეცდომის შესახებ:

```
>>  s =Z(5,1)
???  Index exceeds matrix dimensions.
```

სიმბოლური მატრიცის ფორმირებისას მისი ელემენტების სიგრძე ტოლი უნდა იყოს. თუ ელემენტების სიგრძეები განსხვავებულია, საჭიროა სტრიქონებს დავუმატოთ ჰარები:

```
>>  z=['abcdef'; 'ABCDEF'; '12345 ']
z =
abcdef
ABCDEF
12345
```

თუ სტრიქონულ მნიშვნელობებს გავყოფთ ერთიმეორისაგან ჰარებით, მოხდება მათი გაერთიანება:

```
>>  z=['abcdef '  'ABCDEF '  '123456']
z =
abcdefABCDEF123456
```

მატრიცა შეიძლება აგრეთვე წავიკითხოთ ფაილიდან. ამისათვის, რა თქმა უნდა, საჭიროა, რომ მატრიცა არსებობდეს ფაილში. როგორც უკვე იყო ნათქვამი, მონაცემთა შესანახად გამოიყენება **save** ბრძანება. მაგ., ოპერატორების მიმდევრობა

```
>>  x=10;  y=11;
>>  v=x+y;
>>   z=[10 2 3];
>>  save datavar1 x z; save datavar2; save
```

შეინახავს datavar1.mat ფაილში x, z ცვლადებს, datavar2.mat ფაილში x, y, z და v ცვლადებს, ხოლო matlab.mat ფაილში – სამუშაო არის ყველა ცვლადს.

ფაილიდან მონაცემთა წასაკითხად ვსარგებლობთ **load** ბრძანებით.

load ბრძანებით ხდება სამუშაო არეში ცვლადების ჩატვირთვა დისკიდან. ამ დროს სისტემა ეძებს matlab.mat ფაილს; **load ფაილის_სახელი** ბრძანებით ჩაიტვირთება ყველა ცვლადი მითითებული ფაილიდან, ამასთან, თუ ფაილის სახელში არ არის მითითებული გაფართოება, სისტემა ეძებს ფაილს .mat გაფართოებით და თავისთავად განიხილავს მას როგორც ორობითს, ხოლო თუ მითითებულ ფაილს აქვს სხვა გაფართოება, სისტემა

განიხილავს მას როგორც ASCII ტიპის ფაილს; **load ფაილის_სახელი ცვლადების_სია** ბრძანებით ხდება ამ ცვლადების ჩატვირთვა მითითებული ფაილიდან.

თუ მონაცემთა წასაკითხად გვსურს გამოვიყენოთ ASCII ფაილი, ის უნდა შეიცავდეს მხოლოდ რიცხვით მნიშვნელობებს, ამასთან, ტექსტური ფაილი უნდა წარმოადგენდეს რიცხვების მართკუთხა ცხრილს ანუ ყოველი სტრიქონი ელემენტთა ერთნაირ რაოდენობას უნდა შეიცავდეს. ეს ფაილი შეიძლება შეიქმნას როგორც ნებისმიერი ტექსტური რედაქტორის საშუალებით, ასევე MATLAB–ის მეშვეობით. მაგ., შევქმნათ Notepad–ში ტექსტური ფაილი:

```
16.0    3.0    2.0    13.0
 5.0   10.0   11.0    -3.2
```

და შევინახოთ ეს ფაილი prov.dat სახელით. MATLAB–ის ბრძანება load prov.dat წაიკითხავს ამ ფაილს და შექმნის prov ცვლადს, სადაც იქნება ჩაწერილი მოყვანილი მატრიცა.

იგივეს მივაღწევთ Notepad–ში შექმნილი ფაილის შენახვით MATLAB–ის ძირითადი კატალოგის work ქვეკატალოგში .txt გაფართოებით და პროგრამაში შემდგომი ჩატვირთვით A=load('ფაილის_სახელი'.txt) ბრძანებით. მატრიცა იქნება ჩაწერილი A ცვლადში.

თუ ტექსტურ ფაილს ვქმნით MATLAB–ში, ის უნდა შევინახოთ **save** ბრძანებით და მივუთითოთ შესაბამისი გაფართოება. მაგ., ოპერატორების შემდეგი მიმდევრობა

```
>> R=[2 1 3 4; 3 5 6 7]; save  pr.dat  R  -ASCII
```

შეინახავს R მატრიცას pr.dat ტექსტურ ფაილში. ბრძანება load pr.dat წაიკითხავს pr.dat ფაილს და შექმნის pr ცვლადს, სადაც იქნება ჩაწერილი R მატრიცა.

სამუშაო არეში არსებული R მასივის ტექსტურ ფაილში გადასატანად შეიძლება გამოვიყენოთ აგრეთვე save 'ფაილის_სახელი.txt' R -ASCII ბრძანება ან, ვთქვათ, save 'ფაილის_სახელი.doc' R -ASCII (მაგ., save 'liza.txt' R -ASCII ან save 'liza.doc' R -ASCII). ბრძანება A=load('liza.txt') (ან შესაბამისად A=load('liza.doc')) ჩატვირთავს მითითებულ ფაილს და შექმნის A ცვლადს, სადაც იქნება ჩაწერილი R მატრიცა.

მატრიცის შექმნა შესაძლებელია აგრეთვე M–ფაილში. მაგ., გამოვიძახოთ M–ფაილის ფანჯარა და ავკრიფოთ

```
F=[...
2  45  -4
1  2  35];
```

შევინახოთ შეტანილი ოპერატორი სახელით LP.m. მაშინ ბრძანება LP წაიკითხავს ამ ფაილს და შექმნის სამუშაო სივრცეში F ცვლადს, რაც ილუსტრირებულია შემდეგ მაგალითში:

```
>> LP
>> X1=F
X1 =
   2   45   -4
   1    2   35
```

ორწერტილოვანი ოპერატორი – MATLAB–ის ერთ–ერთი მნიშვნელოვანი ოპერატორია. მას აქვს სხვადასხვა ფორმა. გამოსახულება

```
>> 1 : 10
```

შექმნის ვექტორ–სტრიქონს, რომელშიც ჩაწერილია მთელი რიცხვები 1–დან 10–მდე:

```
    1    2    3    4    5    6    7    8    9   10
```

თუ ორწერტილოვანი ოპერატორი შეიცავს 3 რიცხვს, შეიქმნება ვექტორი, რომლის პირველი რიცხვი გვაძლევს ვექტორის პირველ ელემენტს, მეორე რიცხვი აჩვენებს ბიჯს, ხოლო მესამე – ვექტორის ბოლო ელემენტის ზღვრულ მნიშვნელობას. მაგ.,

```
>> kl=150 : -11 : 10
```

შექმნის შემდეგ kl ვექტორს:

```
kl =
   150   139   128   117   106    95    84    73    62    51    40    29    18
```

ორწერტილოვანი ოპერატორის მეშვეობით შეიძლება არსებული მატრიცის საფუძველზე შევქმნათ ახალი მატრიცა, რომელიც საწყისი მატრიცის ნაწილს წარმოადგენს. მაგ., შემდეგი ოპერატორების მიმდევრობა

```
>> T=[1  2  6  5 ; 3  1  1  0 ; 2  3  5  6 ; 4  4  0  0];
>> T_1=T(2:3,1:2);
>> T_2=T(3:4,:);
>> T_3=T(:, 1:2:3);
>> T_4=T(3,1:3);
```

ჯერ შექმნის T მატრიცას, შემდეგ T_1 მატრიცას, რომელიც T მატრიცის მე–2 და მე–3 სტრიქონების პირველ ორ სვეტს წარმოადგენს, T_2 მატრიცას, რომელიც T მატრიცის მე–3 და მე–4 სტრიქონების ყველა სვეტს შეიცავს, T_3 მატრიცას, რომელიც შედგება საწყისი მატრიცის ყველა სტრიქონის 1–ლი და მე–3 სვეტისაგან, და T_4 მატრიცას, რომელიც შექმნილია მე–3 სტრიქონის პირველი სამი ელემენტისაგან:

T	T_1	T_3	T_4
1 2 6 5	3 1	1 6	2 3 5
3 1 1 0	2 3	3 1	
2 3 5 6		2 5	
4 4 0 0		4 0	

T(:) გამოსახულება შექმნის ვექტორ–სვეტს, რომლის დასაწყისში განლაგებულია T მატრიცის პირველი სვეტის ელემენტები, შემდეგ – მეორის და ა.შ.:

შესაძლებელია მატრიცის მნიშვნელობები შევიტანოთ *უშუალოდ კლავიატურიდან* **input** *ოპერატორის საშუალებით*. **input** ოპერატორი გამოიტანს ეკრანზე ტექსტს და დაელოდება მონაცემებს, რომლის საფუძველზე შეიქმნება მატრიცა. მაგ.,

```
>> D=input('Enter D')
Enter D [1 22 -3]
D =
   1   22   -3
```

თუ მონაცემებს არ შევიყვანთ და დავაჭერთ Enter კლავიშს, შეიქმნება ცარიელი მატრიცა [ ].

2.2 ოპერაციები მასივებზე და მატრიცებზე

ოპერაციები მასივებზე ზოგადად განსხვავდება მატრიცული ოპერაციებისაგან. მასივებზე ოპერაციების განმსაზღვრელი სიმბოლოა წერტილი მოქმედების ნიშნის წინ.

მატრიცების შეკრება და გამოკლება შემდეგი გამოსახულების თანახმად სრულდება და არ განსხვავდება მასივების შეკრება–გამოკლებისაგან:

$$A \pm B = [a(i,j) \pm b(i,j)] \quad .$$

აქ A და B – საწყისი მატრიცებია, a(i,j) და b(i,j) – მატრიცების ელემენტები, შეკრება (გამოკლება) წევრ–წევრად სრულდება. საწყისი მატრიცების ზომა, ცხადია, ერთნაირი უნდა იყოს (იგივეა ერთგანზომილებიანი მატრიცების – ვექტორების შემთხვევაში).

თუ გვსურს ორი ერთნაირი ზომის მატრიცა (ან ვექტორი) ერთმანეთზე წევრ–წევრად გადავამრავლოთ, ვასრულებთ *მასივების გამრავლებას*:

$$A. * B = [a(i,j) * b(i,j)] \quad .$$

წერტილის გამოტოვება სერიოზული შეცდომაა, რადგან წერტილის არდაწერისას მოხდება *მატრიცების გადამრავლება*, რომელიც სრულდება შემდეგი გამოსახულების თანახმად:

$$A \cdot B = [\sum_{k=1}^{p} a(i,k) * b(k,j)] \quad .$$

ანუ m x p ზომის A მატრიცის p x n ზომის B მატრიცაზე ნამრავლი წარმოადგენს m x n ზომის მატრიცას, რომლის i–ური სტრიქონის და j–ური სვეტის გადაკვეთაზე მდგომი ელემენტი A მატრიცის i–ური სტრიქონისა და B მატრიცის j–ური სვეტის რიგრიგობით აღებული ელემენტების ნამრავლის ჯამის ტოლია.

მასივების გაყოფა წევრ–წევრად სრულდება. საწყისი მასივების (ვექტორების) ზომა აქაც ერთნაირი უნდა იყოს. არსებობს *მარჯვენა გაყოფა* და *მარცხენა გაყოფა*:

$$A. / B = [a(i,j) / b(i,j)] \quad ;$$

$$A. \backslash B = [b(i,j) / a(i,j)] \quad .$$

მატრიცებში მარჯვენა და მარცხენა გაყოფა შემდეგნაირადაა განსაზღვრული:

$$B / A = B \cdot A^{-1} \quad ; \quad A \backslash B = A^{-1} \cdot B \,,$$

სადაც A^{-1} – A–ს შებრუნებული მატრიცაა. A კვადრატული მატრიცის შებრუნებული არის ისეთი A^{-1} მატრიცა, რომლისთვისაც სრულდება პირობა $A \cdot A^{-1} = A^{-1} \cdot A$ და ეს ნამრავლი უდრის ერთეულოვან მატრიცას ანუ მატრიცას, რომლის მთავარი დიაგონალის ელემენტები ერთის ტოლია, ყველა სხვა ელემენტი კი ნულის; A კვადრატული და იგივე ზომის I ერთეულოვანი მატრიცებისათვის სრულდება შემდეგი ტოლობა: $A \cdot I = I \cdot A = A$.

მასივური ახარისხება წევრ–წევრად ხდება:

$$A. \wedge B = [a(i,j) \wedge b(i,j)] \quad .$$

მაგ.,

>> A=[2 5 6]; B=[1 3 2];		
>> C=A.^2 C = 4 25 36	>> D=A.^B D = 2 125 36	>> W=3.^A W = 9 243 729

დასაშვებია, რომ მასივების ელემენტები კომპლექსური რიცხვებიც იყოს.

მატრიცული ახარისხება. როდესაც ხარისხის მაჩვენებელი *k* მთელი დადებითი რიცხვია, მატრიცული ახარისხება შეესაბამება მატრიცის *k*–ჯერ თავის თავზე გამრავლებას, თუ *k* მთელი უარყოფითი რიცხვია, შებრუნებული მატრიცის *k*–ჯერ თავის თავზე გამრავლებას, ხოლო თუ ხარისხის მაჩვენებელი p ნამდვილი ან კომპლექსური რიცხვია, ვსარგებლობთ ფორმულით [3]:

$$A^{\wedge} p = R \cdot \Delta^{p} \cdot R^{-1}.$$

იმისათვის რომ მატრიცა ავახარისხოთ, ის აუცილებლად კვადრატული უნდა იყოს.

მასივის ელემენტების ტრანსპონირება მდგომარეობს მისი სტრიქონების და სვეტების ადგილების ურთიერთშეცვლაში. ტრანსპონირებული მასივია ეს არის ახალი მასივი, რომლის სვეტები საწყისი მასივის სტრიქონებია და პირიქით. A მასივის ტრანსპონირებისათვის ხმარობენ აპოსტროფის ნიშანს, რომლის მომდევნო სიმბოლოა წერტილი. შემდეგ მაგალითში F საწყისი მასივია, R – ტრანსპონირებული.

```
>> F=[1 2 3 5; 0 1 3 2];
 >> R=F.'
R =
   1   0
   2   1
   3   3
   5   2
```

მატრიცის ტრანსპონირებისას ხდება სტრიქონებისა და სვეტების ადგილების ურთიერთგაცვლა და კომპლექსური ელემენტების გარდაქმნა: ტრანსპონირებული მატრიცის კომპლექსური ელემენტების შესაბამისი ელემენტები გახდება საწყისი კომპლექსური რიცხვების შეუღლებული. მატრიცის ტრანსპონირება აღინიშნება აპოსტროფის ნიშნით. თუ გვსურს ტრანსპონირებული მატრიცის მიღება მისი ელემენტების მნიშვნელობების შეცვლის გარეშე, უნდა ვისარგებლოთ A.' ბრძანებით ან **conj**(A') ფუნქციით.

მაგ.,

>> P=[1+3i, -2+3i, 4; -2i, 3-5i, 6];	
>> X=P' X = 1.0000 - 3.0000i 0 + 2.0000i -2.0000 - 3.0000i 3.0000 + 5.0000i 4.0000 6.0000	>> X=P.' X = 1.0000 + 3.0000i 0 - 2.0000i -2.0000 + 3.0000i 3.0000 - 5.0000i 4.0000 6.0000

MATLAB–ში *შესაძლებელია მასივის ყველა ელემენტისათვის გამოვთვალოთ ნებისმიერი ერთცვლადიანი მათემატიკური ფუნქცია*: f(A) = [f(a(i,j))]. შედეგად მიიღება საწყისი მასივის ზომის მქონე მასივი, რომლის ყველა ელემენტი წარმოადგენს საწყისი მასივის შესაბამისი ელემენტის f ფუნქციას. ამ თვისებას – ფუნქციის ერთდროული გამოთვლის შესაძლებლობას მასივის ყველა ელემენტისათვის – *გამოთვლების ვექტორიზაცია* ჰქვია. ეს არის MATLAB სისტემის უნიკალური თვისება, რომელიც უზრუნველყოფს მუშაობის დიდ სისწრაფეს ინტერპრეტაციის რეჟიმში. მაგ.:

```
>> A=[0 pi/6 pi/4 pi/3 pi/2 2*pi/3 5*pi/4 5*pi/6 pi];
>> B=sin(A)
B =
      0   0.5000   0.7071   0.8660   1.0000   0.8660  -0.7071   0.5000   0.0000
>> C=sin(A)+cos(B)
C =
   1.0000   1.3776   1.4674   1.5139   1.5403   1.5139   0.0531   1.3776   1.0000
```

2.3 მონაცემთა დამუშავების ძირითადი ფუნქციები

help elfun ბრძანებით ეკრანზე აისახება მონაცემთა დამუშავებასთან დაკავშირებული ყველა ფუნქციის ჩამონათვალი მოკლე აღწერით. განვიხილოთ ძირითადი ამ ფუნქციებისაგან.

sum(X) ფუნქცია ვექტორებისათვის გამოთვლის ვექტორის ელემენტების ჯამს. მატრიცებისათვის ეს ფუნქცია გამოთვლის ელემენტების ჯამს თითოეულ სვეტში და მოათავსებს შედეგს ვექტორ–სტრიქონში.

sum(X,k) გამოთვლის ჯამს k–ური განზომილების მიხედვით. ასე რომ, ორგანზომილებიანი მასივისათვის sum(X,1) გამოთვლის თითოეული სვეტის ელემენტების ჯამს ანუ იგივე შედეგს მოგვცემს, რაც sum(X), ხოლო sum(X,2) გამოთვლის ელემენტების ჯამს თითოეულ სტრიქონში და მოათავსებს შედეგს ვექტორ–სვეტში. თითოეული სტრიქონის ელემენტების ჯამის ვექტორ–სტრიქონში მოსათავსებლად ვსარგებლოთ sum(X') ფუნქციით:

```
>> g=[1 -22; 4 -2];
```

>> sum(g)	>> sum(g,1)	>> sum(g,2)	>> sum(g')	>> sum(g(:,end))
ans =	ans =	ans =	ans =	ans =
5 -24	5 -24	-21	-21 2	-24
		2		

ბოლო ოპერატორით სისტემა მატრიცის ბოლო სვეტის ელემენტის ჯამს გამოთვლის.

იგივე შეიძლება ვთქვათ **prod** ფუნქციაზე, რომლის მეშვეობით გამოითვლება ნამრავლი:

>> prod(g,1)	>> prod(g,2)
ans =	ans =
4 44	-22
	-8

cumsum(X) ფუნქცია ვექტორებისათვის წარმოადგენს ვექტორს, რომლის თითოეული ელემენტი უდრის მის წინ მდგომი ელემენტების ჯამს. მატრიცებისათვის ეს ფუნქცია გვაძლევს საწყისი მატრიცის ზომის მქონე მატრიცას, რომლის ელემენტები აკუმულირებენ ჯამს სვეტების მიხედვით. იგივე შეიძლება გაკეთდეს **cumsum(X,1)** ფუნქციის მეშვეობით, ხოლო **cumsum(X,2)** ფუნქციით ჯამის აკუმულირება ხდება სტრიქონების მიხედვით.

>> h=[1 2 3 5 10];	>> g=[1 -22; 4 -2];	
>> cumsum(h)	>> cumsum(g,1)	>> cumsum(g,2)
ans =	ans =	ans =
1 3 6 11 21	1 -22	1 -21
	5 -24	4 2

cumprod(X) ფუნქციის მეშვეობით ხდება ნამრავლის აკუმულირება:

>> cumprod(h)	>> cumprod(g,1)	>> cumprod(g,2)
ans =	ans =	ans =
1 2 6 30 300	1 -22	1 -22
	4 44	4 -8

diff(X) ფუნქცია ვექტორისათვის წარმოადგენს [X(2)–X(1) X(3)–X(2) ... X(n)–X(n–1)] ვექტორს, ხოლო მატრიცისათვის გვაძლევს [X(2:n,:)–X(1:n-1,:)] სტრიქონების სხვაობის მატრიცას:

>> h=[1 2 3 5 10]; U=[1 0 2; 55 4 7; 8 9 10];						
>> U	>> diff(U,1)	>> diff(U,2)	>> diff(U,3)	>> diff(U,4)	>> diff(U,5)	>> diff(U,6)
U =	ans =	ans =	ans =	ans =	ans =	ans =
1 0 2	54 4 5	-101 1 -2	102 -3	-105	[]	[]
55 4 7	-47 5 3					
8 9 10						

dot(A,B) ფუნქცია გამოიყენება A და B ერთნაირი ზომის ვექტორების *სკალარული ნამრავლის* გამოსათვლელად. სკალარული ნამრავლი (სხვანაირად მას *შიდა ნამრავლს* უწოდებენ) განისაზღვრება ფორმულით:5

$$A \cdot B = \sum_{i=1}^{n} a_i \cdot b_i \ ,$$

სადაც n – ვექტორების ზომაა.

dot(A,B) ფუნქცია გამოითვლის სკალარულ ნამრავლს იმისდა მიუხედავად, თუ როგორი სახით არიან წარმოდგენილი A და B ვექტორები: ორივე ვექტორ–სვეტია, ორივე ვექტორ–სტრიქონი თუ ერთ–ერთი ვექტორ–სვეტია, მეორე კი ვექტორ–სტრიქონი. თუ A ვექტორ–სტრიქონია, ხოლო B – ვექტორ–სვეტი, სკალარული ნამრავლი შეიძლება სხვა

მეთოდითაც გამოვთვალოთ – როგორც sum(A'.* B) ან sum(A.* B'). ორი ვექტორ–სტრიქონის სკალარული ნამრავლის გამოსათვლელად ვსარგებლობთ A*B' , ხოლო ორი ვექტორ–სვეტის შემთხვევაში A'*B გამოსახულებით ან sum(A.* B) ფუნქციით.

A და B ვექტორ–სტრიქონების A'*B ნამრავლით მიიღება nxn ზომის მატრიცა, რომელსაც ორი ვექტორის *გარე ნამრავლი* ჰქვია. ანალოგიურად A და B ვექტორ–სვეტების A*B' ნამრავლით მიიღება nxn ზომის მატრიცა.

min(X) და **max(X)** ფუნქციებით ხდება მინიმალური და მაქსიმალური ელემენტის მოძებნა ვექტორში. როდესაც X მატრიცაა, მოიძებნება თითოეული სვეტის მინიმალური (მაქსიმალური) ელემენტი და შედეგი მოთავსდება ვექტორ–სტრიქონში. NaN კონსტანტები იგნორირდება მინიმუმის და მაქსიმუმის გამოთვლისას. თუ არგუმენტის ყველა ელემენტია NaN, მინიმალური (მაქსიმალური) ელემენტიც NaN–ს უდრის.

[m,i]=min(X) და **[m,i]=max(X)** ოპერაციების შესრულების შედეგად m ცვლადში მოთავსდება X ვექტორის მინიმალური (მაქსიმალური) ელემენტი, ხოლო i–ში – მისი ინდექსი. თუ ვექტორი შეიცავს რამდენიმე მინიმალური (მაქსიმალური) მნიშვნელობის მქონე ელემენტს, i–ში მოთავსდება პირველი ასეთი ელემენტის ინდექსი. ოპერაცია განსაზღვრულია მატრიცებისთვისაც.

min(X,Y) და **max(X,Y)** ფუნქციით მიიღება X და Y მასივების სიგრძის მქონე მასივი, რომლის თითოეული ელემენტი წარმოადგენს უმცირესს (უდიდესს) X და Y–ის შესაბამის ელემენტთა შორის.

```
>> X=[1 4 -1 2 -1 5 6]; Y=[3 2 3;3 -1 5]; D=[NaN NaN]; A=[4 1 3;6 -1 5]; F1[2 4 1/0]; F2=[2 4 0/0];
```

```
>> X
X =
   1   4  -1   2  -1   5   6
>> min(X)
ans =
  -1
>> min(X)+max(X)*2
ans =
  11
>> D
D =
  NaN  NaN
>> min(D)
ans =
  NaN
```

```
>> F1
F1 =
   2   4 Inf
>> max(F1)
ans =
  Inf
F2 =
   2   4 NaN
>> max(F2)
ans =
   4
```

```
>> [m,i]=min(X)
m =
  -1
i =
   3
>> Y
Y =
   3   2   3
   3  -1   5
>> A
A =
   4   1   3
   6  -1   5
```

```
>> min(Y)
ans =
   3  -1   3
>> [z,j]=max(Y)
z =
   3   2   5
j =
   1   1   2
>> [w,p]=min(Y)
w =
   3  -1   3
p =
   1   2   1
```

```
>> min(Y,A)
ans =
   3   1   3
   3  -1   5
>> max(Y,A)
ans =
   4   2   3
   6  -1   5
```

sort(X) ფუნქციით ხდება ვექტორის ელემენტების დალაგება ზრდადობით. როდესაც X მატრიცაა, ფუნქცია დაალაგებს ზრდადობით მატრიცის თითოეული სვეტის ელემენტებს.

sort(X,dim,mode) ფუნქციას აქვს ორი ოფცია: dim და mode. როდესაც dim=1, ხდება მატრიცის თითოეული სვეტის ელემენტების დალაგება ზრდადობით, ხოლო როდესაც dim=2 –

თითოეული სტრიქონის. mode ოფციით აირჩევა დალაგების მიმართულება: 'ascend' – ზრდადობით, 'descend' –კლებადობით. კლებადობით ელემენტების დალაგება შესაძლებელია აგრეთვე **–sort(-X)** ოპერატორით.

>> Y=[3 2 3;3 -1 5];					
>> Y Y = 3 2 3 3 -1 5	>> H=sort(Y) H = 3 -1 3 3 2 5	>> K=sort(Y,1) K = 3 -1 3 3 2 5	>> F=sort(Y,2) F = 2 3 3 -1 3 5	>> -sort(-Y) ans = 3 2 5 3 -1 3	>> sort(Y,2,'descend') ans = 3 3 2 5 3 -1

mean(X) ფუნქცია ვექტორებისათვის გამოთვლის ვექტორის ელემენტების საშუალო არითმეტიკულს. მატრიცებისათვის ეს ფუნქცია გამოთვლის თითოეული სვეტის ელემენტების საშუალო არითმეტიკულს და მოათავსებს შედეგს ვექტორ–სტრიქონში.

median(X) ფუნქცია ვექტორებისათვის გამოთვლის ვექტორის ელემენტების მედიანას, ხოლო მატრიცებისათვის – თითოეული სვეტის ელემენტების მედიანას შედეგის ვექტორ–სტრიქონში მოთავსებით.

diag(X) ფუნქციით მიიღება ვექტორ–სვეტად ჩაწერილი X მატრიცის მთავარი დიაგონალი. მთავარი დიაგონალი იწყება ზედა მარცხენა კუთხიდან, მისი ელემენტების სვეტის და სტრიქონის მიმთითებელი ინდექსები ერთნაირია: x_{11}, x_{22}, და ა.შ. X მატრიცის მთავარი დიაგონალის ელემენტების ჯამი გამოითვლება sum(diag(x)) ფუნქციით. diag(X) ფუნქციით შეიძლება აგრეთვე მთავარ დიაგონალზე მოვათავსოთ ნებისმიერი რიცხვები, ხოლო დანარჩენი ელემენტები გავუტოლოთ ნულს. ამ შემთხვევაში diag ფუნქციის არგუმენტი ვექტორია და არა მატრიცა. მაგ.,

>> X=[1 3 5 4; -1 -2 -5 -10; 1 2 22 0]; diag(X)	>> z=[1; 3; 5; -1]; A=diag(z)
ans = 1 -2 22	A = 1 0 0 0 0 3 0 0 0 0 5 0 0 0 0 -1

diag(X,k) ფუნქციით იმ შემთხვევაში, როდესაც k>0, ხდება მთავარი დიაგონალის ზემოთ k–ური დიაგონალის შერჩევა; როდესაც k<0, ამოღებული იქნება მთავარი დიაგონალის ქვემოთ k–ური დიაგონალი. ცხადია, diag(X,0) diag(X) –ის ტოლფასია.

>> X = 1 3 5 4 -1 -2 -5 -10 1 2 22 0	>> diag(X,1) ans = 3 -5 0	>> diag(X,2) ans = 5 -10	>> diag(X,-1) ans = -1 2

triu(X) ფუნქციით მიიღება საწყისი მატრიცის ზომის მატრიცა, რომელიც შეიცავს X მატრიცის მთავარი დიაგონალის და მის ზემოთ განლაგებულ ელემენტებს, დანარჩენი ელემენტი კი 0–ის ტოლია.

triu(X,k) ფუნქცია შექმნის საწყისი მატრიცის ზომის მატრიცას, რომელიც შეიცავს k–ური დიაგონალის და მის ზემოთ მდებარე X მატრიცის ელემენტებს, სხვა ელემენტი 0–ის ტოლია.

tril(X) და **tril(X,k)** ფუნქციებიი მსგავსია **triu(X)** და **triu(X,k)** ფუნქციებისა, მხოლოდ ამოღებული იქნება საწყისი მატრიცის ქვედა ნაწილი.

```
>>  F=[1:10:40; 1 2  3 4; 3:2:9];
```

```
>> F
F =
   1   11   21   31
   1    2    3    4
   3    5    7    9
```

```
>> triu(F)
ans =
   1   11   21   31
   0    2    3    4
   0    0    7    9
```

```
>> triu(F,1)
ans =
   0   11   21   31
   0    0    3    4
   0    0    0    9
```

```
>> tril(F)
ans =
   1    0    0    0
   1    2    0    0
   3    5    7    0
```

```
>> tril(F,-1)
ans =
   0    0    0    0
   1    0    0    0
   3    5    0    0
```

det(X) ფუნქციით გამოითვლება X კვადრატული მატრიცის დეტერმინანტი.

inv(X) ფუნქციით მიიღება კვადრატული X მატრიცის შებრუნებული მატრიცა ანუ X^{-1}.

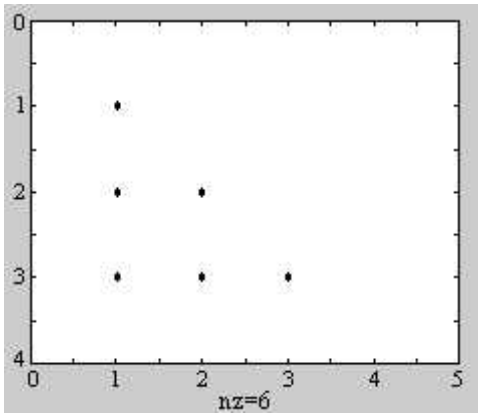
ახლა განვიხილოთ ფუნქციები, რომელთა საშუალებით ხდება მატრიცის ნაწილის ამოღება ახალი მატრიცის სახით.

spy(X) ფუნქციის საშუალებით გრაფიკულ ეკრანზე გამოჩნდება არანულოვანი ელემენტების ადგილმდებარეობა.

მაგ.,

```
>> spy(tril(F))
```

მოგვცემს ნახ.2.1–ზე ნაჩვენებ სქემას. საკოორდინატო ღერძებზე მონიშნულია სტრიქონებისა და გვერდების ნომრები, ხოლო ქვემოთ მითითებულია არანულოვანი ელემენტების რაოდენობა.



ნახ.2.1

2.4 მატრიცების და ვექტორების გაერთიანება და გარდაქმნა

მატრიცების გაერთიანება წარმოადგენს პატარა მატრიცებისაგან დიდი მატრიცის შექმნის პროცესს. ამისათვის ვსარგებლობთ გაერთიანების ოპერატორით – კვადრატული ფრჩხილებით. იგივე ითქმის ვექტორებზეც. საილუსტრაციოდ ქვემოთ მოყვანილია D მატრიცისაგან ახალი G მატრიცისა და S ვექტორისაგან S2 მატრიცის შექმნის მაგალითი:

```
>> D=[ 1 5 25 6; -1 -3 0 0; 4 5 -6 -8]
D =
   1   5  25   6
  -1  -3   0   0
   4   5  -6  -8
```

```
>>G=[D  D+3; D-2  D*2]
G =
    1    5   25    6    4    8   28    9
   -1   -3    0    0    2    0    3    3
    4    5   -6   -8    7    8   -3   -5
   -1    3   23    4    2   10   50   12
   -3   -5   -2   -2   -2   -6    0    0
    2    3   -8  -10    8   10  -12  -16
>>S=[1  2  3]; S2=[S   2*S   4-S; S+1   S   S]
S2 =
    1    2    3    2    4    6    3    2    1
    2    3    4    1    2    3    1    2    3
```

კვადრატული ფრჩხილებით ვსარგებლობთ აგრეთვე *ვექტორის ელემენტებისა* და *მატრიცის სტრიქონებისა და სვეტების მოსაშორებლად*:

```
>> S2(:,3)=[]; S2
>> S2
S2 =
    1    2    2    4    6    3    2    1
    2    3    1    2    3    1    2    3
>> S2(:,1:2:7)=[]; S2
S2 =
    2    4    3    1
    3    2    1    3
>>S(1:2:3)=[]; S
S =
    2
```

დაუშვებელია მატრიციდან ერთი ელემენტის ამოღება. S2(1,2)=[] ოპერატორის შედეგად მივიღებთ შეტყობინებას შეცდომის შესახებ. მაგრამ თუ ანალოგიურ ოპერატორში მივუთითებთ ერთ ინდექსს, შესაბამისი ელემენტი იქნება ამოღებული და დანარჩენი ელემენტების მიმდევრობა ვექტორ–სტრიქონად გარდაიქმნება:

```
>> S2(2)=[]; S2
S2 =
    2    4    2    3    1    1    3
```

ვექტორების ელემენტების და *მატრიცების სვეტების (სტრიქონების) ადგილების ურთიერთშეცვლა* ხდება კვადრატულ ფრჩხილებში მათი ახალი ადგილის მითითებით:

```
>> K=[-1 -3 -32 -4; 0 5 0 7]; P=K( : , [1 3 2 4]);
```

```
>> P
P =
   -1   -32    -3    -4
    0     0     5     7
```

```
>> S2([1 4 3 7 2 6 5])
ans =
     2     3     2     3     4     1     1
```

ბოლო ოპერატორში მივუთითეთ S2 ვექტორის *ელემენტების ახალი თანამიმდევრობა.* ანალოგიური ბრძანება შეიძლება მივცეთ მატრიცისათვისაც:

```
>> P=[-1 -3 -32 -4; 0 5 0 -7]; P (: , [3 4 2 2])
ans =
   -32    -4    -3    -3
     0    -7     5     5
```

2.5 სპეციალური მატრიცების გენერირება

MATLAB–ს აქვს ფუნქციები, რომლის საშუალებით სპეციალური ტიპის მატრიცები იქმნება.

magic(n) ფუნქციით შეიქმნება ე.წ. მაგიური კვადრატი. მაგიური კვადრატი არის nxn ზომის მატრიცა, რომლის ელემენტები მთელი რიცხვებია 1–დან n^2–მდე. ეს რიცხვები ისეა განლაგებული, რომ ყოველ სტრიქონში, სვეტში და ორივე დიაგონალში ელემენტების ჯამი ერთი და იგივე რიცხვია.

```
>> magic(3)
ans =
   8    1    6
   3    5    7
   4    9    2
```

magic ფუნქცია გვაძლევს სწორ შედეგს, როდესაც n>2.

zeros(n) ფუნქციით მიიღება კვადრატული nxn ზომის მატრიცა, რომლის ელემენტები 0–ის ტოლია. **zeros(m,n)**–ით (ამ ფუნქციის ჩაწერის სხვა ფორმებია **zeros([m n])**, **zeros([m,n])**) შეიქმნება mxn ზომის მატრიცა, რომლის ყველა ელემენტი ნულია. იგივეს გააკეთებს **zeros(m,n,p,..)** (**zeros([m n p ...])**, **zeros([m,n,p...])**)) ფუნქცია მრავალგანზომილებიანი მასივისთვის. **zeros(size(X))** ფუნქცია შექმნის ისეთივე ზომის მატრიცას, როგორიც X–ია და ამ მატრიცის ელემენტებიც ნულებია. მაგ.,

```
>>  M=zeros(2,3); F=zeros(size(M));
```

ოპერატორებით შეიქმნება 2x3 ზომის M და F მატრიცები, რომლებშიც ყველა ელემენტი უდრის ნულს.

ones ფუნქციით მიიღება კვადრატული nxn ზომის მატრიცა, რომლის ელემენტები 1–ის ტოლია. ამ ფუნქციის არგუმენტები **zeros** ფუნქციის არგუმენტების ანალოგიურია. მაგ.,

```
>>  K=5*ones(size(M))
K =
   5    5    5
   5    5    5
```

eye ფუნქციის საშუალებით შეიძლება შევქმნათ ერთეულოვანი მატრიცა ანუ მატრიცა, რომლის მთავარი დიაგონალის ელემენტები 1–ის ტოლია, ყველა დანარჩენი კი ნულებია. ამ

ფუნქციის არგუმენტები **ones** და **zeros** ფუნქციების არგუმენტების ანალოგიურია. მაგ., eye(3) და eye(3,2) ფუნქციები შესაბამისად შემდეგ მასივებს შექმნიან:

1	0	0		1	0
0	1	0	და	0	1
0	0	1		0	0

rand(n) ფუნქციით მიიღება კვადრატული nxn ზომის მატრიცა, რომლის ელემენტები (0.0,1.0) ინტერვალში მდებარეობენ და ახასიათებენ შემთხვევითი რიცხვების თანაბარ განაწილებას. შემთხვევითი რიცხვების თანაბარ განაწილებას მივიღებთ აგრეთვე **rand(m,n)** (**rand([m,n])**, **rand([m n])**) და **rand(size(X))** ფუნქციებით, ოღონდ rand(m,n) გენერირებს mxn ზომის მატრიცას, rand(size(X)) კი – ისეთივე ზომის, როგორიცაა X მატრიცა. **rand(m,n,p,...)** (**rand([m,n,p,...])**, **rand([m n p...])**) გენერირებს მრავალგანზომილებიან მასივს.

randn(n) ფუნქციით მიიღება კვადრატული nxn ზომის მატრიცა, რომლის ელემენტები ახასიათებენ შემთხვევითი რიცხვების ნორმალურ განაწილებას. ამ ფუნქციის არგუმენტები rand ფუნქციის არგუმენტების ანალოგიურია.

მაგ.,

```
>> rand(1,10)
ans =
   0.6405   0.2091   0.3798   0.7833   0.6808   0.4611   0.5678   0.7942   0.0592   0.6029
>> T=randn(2,3)
T =
   0.1746   0.7258   2.1832
  -0.1867  -0.5883  -0.1364
```

pascal(n) ფუნქციით მიიღება კვადრატული მატრიცა, რომლის ელემენტები პასკალის სამკუთხედს შეესაბამება:

```
                1
            1       1
        1       2       1
    1       3       3       1
1       4       6       4       1
```

მაგ., pascal(4) ფუნქციით მივიღებთ შემდეგ მატრიცას:

```
1   1   1   1
1   2   3   4
1   3   6  10
1   4  10  20
```

2.6 მანიპულირება მატრიცებთან

rot90(X) ფუნქციით ხდება X მატრიცის 90 გრადუსით მობრუნება საათის ისრის საწინააღმდეგო მიმართულებით, ხოლო **rot90(X,k)** ფუნქციით – k·90 გრადუსით, სადაც k – მთელი რიცხვია.

მაგ.,

```
>> A=[1 2 3; 4 5 6; 7 8 9]; B=rot90(A); C=rot90(A,2); D=rot90(A,-1); W=rot90(A,0);
```

ოპერატორთა მიმდევრობით მივიღებთ შემდეგ მატრიცებს:

A			W			B			C			D		
1	2	3	1	2	3	3	6	9	9	8	7	7	4	1
4	5	6	4	5	6	2	5	8	6	5	4	8	5	2
7	8	9	7	8	9	1	4	7	3	2	1	9	6	3

აღვნიშნოთ, რომ არ არის აუცილებელი, საწყისი მატრიცა კვადრატული იყოს:

```
>> T=[3 6; 2 5; 1 4]; TR=rot90(T)
TR =
   6   5   4
   3   2   1
```

fliplr(X) ფუნქციით ხდება X მატრიცის (ვექტორის) სარკისებური ასახვა მარცხნიდან მარჯვნივ, ხოლო **flipud(X)** –ით – ზევიდან ქვევით. მაგ.,

```
>> X=[1 2 3; 4 5 6]; Y=fliplr(X);
>> Z=flipud(X);  a= sum(diag(Z));
```

ოპერატორებით მიღებული იქნება X, Y და Z მატრიცები, ხოლო ბოლო ოპერატორით Z მატრიცის დიაგონალის (ანუ საწყისი X მატრიცის ე.წ. ანტიდიაგონალის) ელემენტების ჯამი გამოითვება (a=6):

X			Y			Z		
1	2	3	3	2	1	4	5	6
4	5	6	6	5	4	1	2	3

შემდეგი ოპერატორებით სრულდება ვექტორის სარკისებური ასახვა. ცხადია, ვექტორის ასახვა ზემოდან ქვევით არ შეცვლის საწყის ვექტორს:

```
>> S=[1 2 3 4 5];  fliplr(S)
ans =
   5   4   3   2   1
>> flipud(S)
ans =
   1   2   3   4   5
```

flipdim(X,k) ფუნქცია ასრულებს მატრიცის ასახვას k–ური განზომილების მიმართ. ანუ flipdim(X,1) მოგვცემს იგივე შედეგს, რაც flipud(X), ხოლო flipdim(X,2) – რაც fliplr(X).

reshape(X,m,n) ფუნქცია ცვლის საწყისი X მატრიცის ფორმას ანუ თანაფარდობას სტრიქონებისა და სვეტების რაოდენობას შორის. m და n არგუმენტები განსაზღვრავს ახალი მისაღები მატრიცის სტრიქონებისა და სვეტების რაოდენობას, ამასთან, m·n ნამრავლი საწყისი მატრიცის სტრიქონებისა და სვეტების რაოდენობის ნამრავლის ტოლი უნდა იყოს. ანალოგიურად მოქმედებს **reshape(X,m,n,p...)** ფუნქცია მრავალგანზომილებიან მასივებში.

m·n·p... ნამრავლი prod(size(X)) ფუნქციის მნიშვნელობის ტოლი უნდა იყოს. **reshape(X,[m,n])** და **reshape(X,[m,n,p...])** ფუნქციები reshape ფუნქციების ჩაწერის სხვა ფორმას წარმოადგენს. ქვემოთ მოყვანილ მაგალითებში ილუსტრირებულია reshape ფუნქციის მოქმედება:

>> P=[1 3 5 28; -1 4 7 3];		
>> P P = 1 3 5 28 -1 4 7 3	>> reshape(P,[1,8]) ans = 1 -1 3 4 5 7 28 3 >> reshape(P,1,8) ans = 1 -1 3 4 5 7 28 3	>> reshape (P, 4, 2) ans = 1 5 -1 7 3 28 4 3

2.7 მოქმედებები პოლინომზე

ერთცვლადიანი პოლინომის (მრავალწევრის) ზოგადი ფორმულაა:

$$f(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n \quad .$$

პოლინომის ხარისხი მისი ცვლადის ხარისხის უმაღლესი მაჩვენებლის ტოლია.

MATLAB–ში გამოიყენება პოლინომის ჩაწერის ორი ხერხი:

- კოეფიციენტების ვექტორ–სტრიქონით;
- სიმბოლური გამოხატულებით.

პოლინომის განმსაზღვრელ კოეფიციენტების ვექტორ–სტრიქონში ელემენტთა რაოდენობა ერთით უნდა აღემატებოდეს მის ხარისხს. 0–ის ტოლი კოეფიციენტის იგნორირება დაუშვებელია. მაგ., $p = -2x^3 + x - 1$ მრავალწევრის განსაზღვრისათვის გამოიყენება ბრძანება

```
>> p=[-2 0 1 -1];
```

იგივე პოლინომი სიმბოლური სახით შემდეგნაირად ჩაიწერება:

```
>> syms x
>> -2*x^3+x-1
 ans =
 -2*x^3+x-1
```

აქ syms ოპერატორი გამოყენებულია სიმბოლური ობიექტის კონსტრუირებისათვის.

ვექტორ–სტრიქონის სახით ჩაწერილი პოლინომის წარმოდგენა სიმბოლური გამოხატულებით შესაძლებელია **poly2sym(X)** ფუნქციის მეშვეობით, სადაც X – პოლინომის კოეფიციენტების ვექტორ–სტრიქონია:

```
>> poly2sym(p)                ან        >> poly2sym([-2 0 1 -1])
ans =                                    ans =
-2*x^3+x-1                               -2*x^3+x-1
```

poly2sym(X,'v') და **poly2sym(X,sym('v'))** ფუნქციებით მიიღება პოლინომი, რომლის სიმბოლური ცვლადი მითითებულია მეორე არგუმენტში:

```
>> poly2sym([-2 0 1 -1],'t')
ans =
-2*t^3+t-1
```

პოლინომის სიმბოლური წარმოდგენიდან კოეფიციენტების ვექტორ–სტრიქონის სახით ჩანაწერზე გადასვლა ხდება **sym2poly(X)** ფუნქციით, სადაც X – პოლინომის სიმბოლური გამოხატულებაა:

```
>> syms z; sym2poly(3*z^4-2*z^3 - 10*z - 5)
ans =
    3   -2    0   -10   -5
```

ფუნქცია **polyval(X,Y)** გამოთვლის X პოლინომის მნიშვნელობას Y ცვლადისათვის, ამასთან, პოლინომი კოეფიციენტების ვექტორ–სტრიქონით უნდა იყოს წარმოდგენილი:

```
>> polyval([3   -2    0   -10   -5],1.2)
ans =
  -14.2352
```

თუ ფუნქციის მეორე არგუმენტი ვექტორი ან მატრიცაა, პოლინომის მნიშვნელობის გამოთვლა ხდება ყოველი ელემენტისათვის:

```
>> polyval([2 -3 5],[1 -1 ; 2 -5])
ans =
    4   10
    7   70
```

რა თქმა უნდა, პოლინომის მნიშვნელობის გამოთვლა შეიძლება განვახორციელოთ მინიჭების ოპერატორებითაც (შესაბამისად z=1.2; 3*z^4-2*z^3-10*z–5 და x=[1 -1; 2 5]; 2*x.^2-3*x+5 ბრძანებებით).

დავუშვათ, პოლინომები განსაზღვრულია თავისი კოეფიციენტებით. თუ პოლინომების ხარისხები ერთმანეთის ტოლია ანუ კოეფიციენტების ვექტორ–სტრიქონები ერთნაირი სიგრძისაა, მხოლოდ ამ დროს არის შესაძლებელი მათი შეკრება–გამოკლება. ისეთი პოლინომების შეკრება–გამოკლებისას, რომელთა ხარისხები არ ემთხვევა ერთიმეორეს, საჭიროა ჯერ მათი გათანაბრება და შემდეგ მოქმედების შესრულება.

მაგ., გამოვთვალოთ $f(x1)+f(x2)-f(x3)$, სადაც $f(x1)=x^4+3x^2-1$, $f(x2)=x^5+3x-10$ და $f(x3)=2x^2+x-3$:

```
>> x1=[0 1 0 3 0 -1]; x2=[1 0 0 0 3 -10]; x3=[0 0 0 2 1 -3];
>> x=x1+x2-x3
x =
    1    1    0    1    2   -8
>> poly2sym(x)
ans =
x^5+x^4+x^2+2*x-8
```

პოლინომების შეკრება–გამოკლება შესაძლებელია იმ შემთხვევაშიც, როდესაც ისინი სიმბოლურად არიან წარმოდგენილი:

```
>> syms s;  A = s^4 -3*s^3 -s +2;  B = 4*s^3 -2*s^2 +5*s -16;  C = A + B
C =
s^4+s^3+4*s-14-2*s^2
```

პოლინომის სკალარულ სიდიდეზე გასამრავლებად მისი კოეფიციენტების ვექტორი უნდა გავამრავლოთ სკალარზე. ვთქვათ, გვაქვს $f(x) = 3x^4 + 6x^2 - 10$ პოლინომი. იმისათვის, რომ მივიღოთ $g(x) = -3f(x)$ პოლინომის კოეფიციენტები, ვსარგებლობთ ცხრილის 1–ლი სვეტის ბრძანებებით. გამრავლება სკალარზე შესაძლებელია იმ შემთხვევაშიც, როდესაც პოლინომი სიმბოლურად არის წარმოდგენილი, რაც ცხრილის მე–2 სვეტში ასახულია:

>> f=[3 0 6 0 -10]; >> g = -3*f g = -9 0 -18 0 30	>> syms x F = 3*x^4 +6*x^2-10; G = -3*F G = -9*x^4-18*x^2+30

პოლინომების გასამრავლებად ვიყენებთ **conv(X,Y)** ფუნქციას, სადაც X და Y – პოლინომების კოეფიციენტების ვექტორ–სტრიქონებია. გამრავლებისას მიიღება ნამრავლის ვექტორი, რომლის სიგრძეა length(A)+length(B)-1. მაგ., ზემოთ მოყვანილი $f(x1)$ და $f(x2)$ პოლინომების ნამრავლის გამოსათვლელად ვსარგებლობთ ოპერატორებით:

```
>> x1=[1 0 3 0 -1]; x2=[1 0 0 0 3 -10]; conv(x1,x2)
ans =
  1   0   3   0   2  -10   9  -30  -3   10
```

პოლინომების გამრავლება შესაძლებელია მათი სიმბოლურად გამოხატვის შემთხვევაშიც.

პოლინომების გაყოფის ფუნქციაა **deconv(X,Y)**, რომელიც შედეგად გვაძლევს ორ ვექტორს: პირველი ვექტორი შეიცავს განაყოფი პოლინომის კოეფიციენტებს, ხოლო მეორე – ნაშთი პოლინომის კოეფიციენტებს. ბრძანება შემდეგნაირად ჩაიწერება: [q,r]=deconv(x1,x2), სადაც q განაყოფია, ხოლო r – ნაშთი. ზემოთ მოყვანილი $f(x2)$ და $f(x3)$ პოლინომების გაყოფით მივიღებთ:

```
>> x2=[1 0 0 0 3 -10]; x3=[2 1 -3];
>> [q,r]=deconv(x2,x3)
q =
  0.5000  -0.2500   0.8750  -0.8125
r =
       0       0       0       0  6.4375 -12.4375
```

polyder(X) ფუნქციით ხდება კოეფიციენტების ვექტორ–სტრიქონით წარმოდგენილი პოლინომის წარმოებულის გამოთვლა, ამასთან, შედეგიც იქნება მიღებული ვექტორ–სტრიქონის სახით. **polyder(X,Y)** ფუნქცია განკუთვნილია ორი პოლინომის ნამრავლის წარმოებულის გამოსათვლელად, ხოლო ორი პოლინომის შეფარდების წარმოებულის საპოვნელად გამოიძახება polyder ფუნქცია 2 გამომავალი ცვლადით: პირველი წარმოადგენს შედეგის მრიცხველის კოეფიციენტებს, ხოლო მეორე – მნიშვნელის.

diff(X) ახდენს X სიმბოლური გამოსახულების დიფერენცირებას მისი სიმბოლური ცვლადით, **diff(X,v)** – v ცვლადით. **diff(X,n)** ახდენს დიფერენცირებას n–ჯერ (n – მთელი დადებითი რიცხვია), **diff(X,v,n)** – n–ჯერ v ცვლადით. აღვნიშნოთ, რომ diff ფუნქციებით წარმოებულის მნიშვნელობა მიიღება არა მხოლოდ პოლინომებისათვის.

ქვემოთ მოყვანილი ცხრილის 1–ლ სვეტში დიფერენცირება განხორციელებულია polyder ფუნქციით, მე–2 სვეტში კი – diff ფუნქციით.

>> x2=[1 0 0 0 3 -10]; x3=[2 1 -3]; polyder(x2) ans = 5 0 0 0 3 >> polyder(x2,x3) ans = 14 6 -15 0 18 -34 -19 >> [w1,w2]=polyder(x2,x3) w1 = 6 4 -15 0 -6 40 1 w2 = 4 4 -11 -6 9	>> syms s >> p = s^3 + 4*s^2 -7*s -10; >> d = diff(p) d = 3*s^2+8*s-7 >> e = diff(p,2) e = 6*s+8

int(X) ფუნქციით გამოითვლება X სიმბოლური გამოსახულების განუსაზღვრელი ინტეგრალი მისი სიმბოლური ცვლადის მიმართ, **int(X,v)**–ით – v ცვლადის მიმართ.

int(X,a,b) ფუნქციით მიიღება X–ის განსაზღვრული ინტეგრალის მნიშვნელობა a–დან b–მდე მისი სიმბოლური ცვლადის მიმართ, ხოლო **int(X,v,a,b)**–ით **–** v ცვლადის მიმართ.

int ფუნქციებით ინტეგრალის მნიშვნელობა მიიღება არა მხოლოდ პოლინომებისათვის.

```
>> syms x  t; int(x^2+3*x-1)
ans =
1/3*x^3+3/2*x^2-x
>> int(4*x*t,x)
ans =
 2*x^2*t
 >> int(4*x*t,x,1,10)
 ans =
 198*t
```

roots(X) ფუნქცია განკუთვნილია კოეფიციენტების ვექტორ–სტრიქონით განსაზღვრული პოლინომის ყველა ფესვის მოსაძებნად, რომელთა რაოდენობა ემთხვევა პოლინომის ხარისხს. მაგ., roots([1 0 2 0 3 -5]) ოპერატორით მივიღებთ ფესვებს: -0.9432 + 1.1306i; -0.9432 - 1.1306i; 0.4822 + 1.5063i; 0.4822 - 1.5063i; 0.9220. პოლინომის ფესვების გამოთვლა შესაძლებელია აგრეთვე **solve('X')** ფუნქციით, სადაც X – პოლინომის სიმბოლური წარმოდგენაა:

```
>> solve('x^2-7*x+12')
ans =
 4
 3
```

poly(X) ფუნქციით მიიღება იმ პოლინომის კოეფიციენტები, რომლის ფესვები X ვექტორის ელემენტებია:

```
>> poly([3 4])
ans =
```

1 -7 12

თავი III გრაფიკების აგება და რედაქტირება

3.1 ორგანზომილებიანი X–Y გრაფიკის აგება

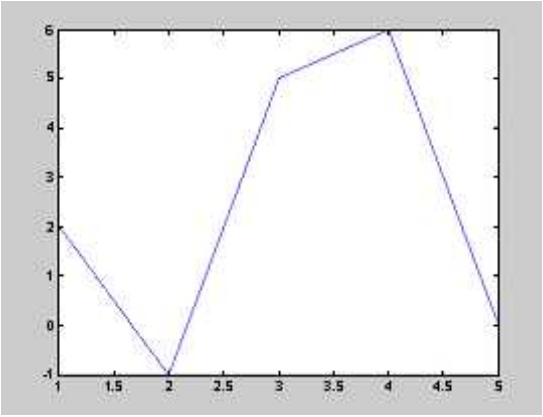
გრაფიკის ასაგებად ვსარგებლობთ **plot** ფუნქციით, რომელსაც რამდენიმე ნაირსახეობა გააჩნია.

plot(Y) მართკუთხა საკოორდინატო სისტემაში შექმნის გრაფიკს, რომელიც ასახავს Y ვექტორის ელემენტების დამოკიდებულებას მათ ინდექსებზე. ამასთან, მნიშვნელობა არა აქვს, ვექტორ–სტრიქონთან გვაქვს საქმე თუ ვექტორ–სვეტთან.

მაგ.,

```
>> Y=[2 -1 5 6 0];  plot(Y)
```

ოპერატორებით მიიღება შემდეგი გრაფიკი, რომლისთვისაც MATLAB–ი ცალკე Figure 1 ფანჯარას გახსნის:



ნახ.3.1

თუ ვექტორის ერთ–ერთი ელემენტი მაინც კომპლექსური რიცხვია, plot(Y) ფუნქცია ექვივალენტურია plot(real(Y),imag(Y))–ისა ანუ შეიქმნება გრაფიკი, რომლის აბსცისათა ღერძზე აღებული იქნება საწყისი ვექტორის ელემენტების ნამდვილი ნაწილი, ხოლო ორდინატთა ღერძზე – წარმოსახვითი (ნახ.3.2,ა):

```
>>  B=[2-2i; 3-i; 5; -1+2i; 3+4i; 5-i; 5];
>> plot(B)
```

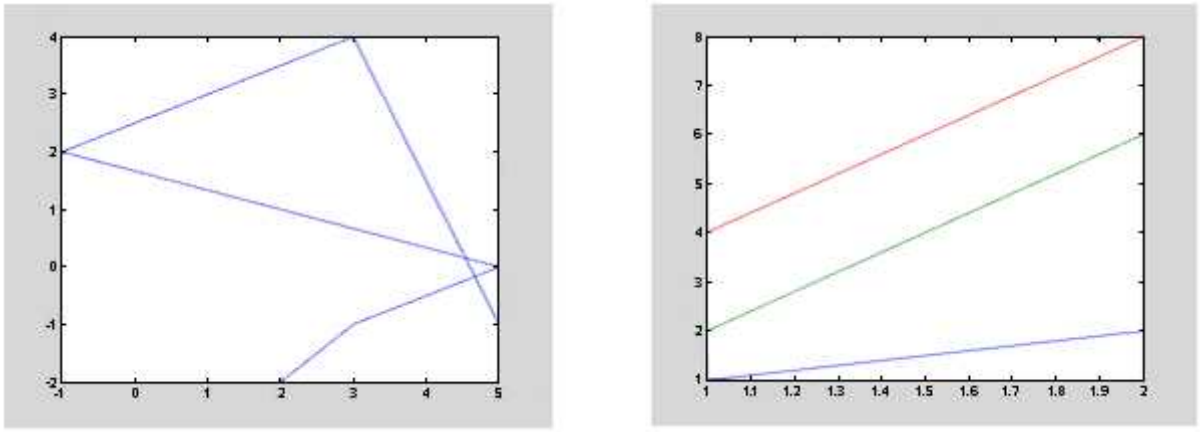
plot(Y)–ის არგუმენტი შეიძლება მატრიცაც იყოს. ამ შემთხვევაში ერთ გრაფიკულ არეში აიგება წირები მატრიცის თითოეული სვეტისათვის. მაგ.,

```
>>A=[1 2 4;2 6 8];  plot(A)
```

ოპერატორებით შეიქმნება ნახ.3.2,ბ–ზე ნაჩვენები გრაფიკი.

წინასწარი შეთანხმებით ახალი გრაფიკი შეიქმნება მიმდინარე გრაფიკულ ფანჯარაში და „წაშლის“ იქ წინა გრაფიკს. თუ გვსურს გრაფიკის აგება ახალ გრაფიკულ არეში, წინასწარ

მივცეთ **File → New → Figure** ბრძანება, რის შედეგად შეიქმნება Figure 2 გრაფიკული არე. ახალი გრაფიკული ფანჯარა შეიძლება აგრეთვე შევქმნათ **figure** ბრძანებით. ბოლო შექმნილი ფანჯარა ითვლება მიმდინარედ. თუ გვსურს გრაფიკის აგება არსებულ n–ურ გრაფიკულ ფანჯარაში, საკმარისია მასზე დავაწკაპუნოთ ან ვისარგებლოთ **figure(n)** ფუნქციით. შემდგომი plot ოპერატორი ააგებს გრაფიკს მითითებულ გრაფიკულ ფანჯარაში და „წაშლის“ იქ არსებულ გრაფიკს.



ა ბ

ნახ.3.2

hold on ბრძანებით შესაძლებელია ახალი გრაფიკების აგება არსებული გრაფიკების წაშლის გარეშე, ამასთან შენარჩუნებული იქნება ღერძების ყველა თვისება. ეს ბრძანება ინარჩუნებს მიმდინარე გრაფიკულ ფანჯარას აქტიურ მდგომარეობაში და ყოველი შემდგომი plot ოპერატორი ახალ წირს უმატებს მას. **hold** ბრძანებითაც მომდევნო ასაგები წირები მიმდინარე ფანჯარაში იქნება ასახული. hold ბრძანების ხელმეორე მიცემა **(**ან **hold off** ბრძანება) შეწყვეტს ამ პროცესს.

თუ პროგრამაში გათვალისწინებულია რამდენიმე ნახაზის შექმნა ერთ ფანჯარაში ან გრაფიკის აგების შემდეგ შემდგომი გამოთვლების გაგრძელება, MATLAB–ი გამოიყვანს ნახაზს და გააგრძელებს პროგრამაში მითითებულ ოპერატორებს, მაგრამ თუ გვინდა ნახაზის დათვალიერება პროგრამის მიერ მუშაობის გაგრძელებამდე, გრაფიკის აგების ოპერატორის შემდეგ უნდა მივცეთ **pause** ბრძანება. ასეთ შემთხვევაში MATLAB–ი დაელოდება მომხმარებლის დასტურს (Enter კლავიში) პროგრამის გასაგრძელებლად.

plot(X,Y) ფუნქციით მართკუთხა საკოორდინატო სისტემაში აიგება წირი, რომელიც აერთებს X და Y ვექტორებით მოცემულ წერტილებს, ამასთან, *x* და *y* ღერძები ტოლ ინტერვალებად იქნება დაყოფილი (ასეთ გრადაციას წრფივს უწოდებენ). plot ფუნქციის პირველი არგუმენტი დამოუკიდებელ ცვლადს შეესაბამება, ხოლო მეორე – დამოკიდებულ ცვლადს.

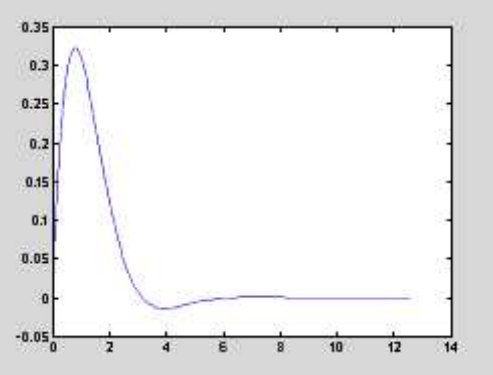
როდესაც ცვლადების მნიშვნელობები ძალიან ფართო დიაპაზონშია განსაზღვრული, იყენებენ ლოგარითმულ გრადაციას. **semilogx(X,Y)** ფუნქციით მართკუთხა საკოორდინატო სისტემაში აიგება გრაფიკი ლოგარითმული გრადაციით *x* ღერძისათვის და წრფივი *y*–სათვის,

semilogy(X,Y) ფუნქციით აიგება გრაფიკი წრფივი გრადაციით *x* ღერძისათვის და ლოგარითმული *y*–სათვის, ხოლო **loglog(X,Y)**–ით მიიღება გრაფიკი ლოგარითმული გრადაციით ორივე ღერძისათვის. ამ ოპერატორების გამოყენებისას საჭიროა გვახსოვდეს, რომ ნულზე ნაკლები ან ტოლი სიდიდის ლოგარითმი არ არსებობს. თუ შესაბამისი მონაცემები შეიცავს ასეთ მნიშვნელობებს, MATLAB–ი მიგვითითებს შეცდომაზე და გაგვაფრთხილებს, რომ გრაფიკის აგების დროს ეს მნიშვნელობები გამოტოვებული იქნება.

შემდეგ მაგალითში მოყვანილია ოპერატორები მილევადი სინუსოიდის გრაფიკის ასაგებად. ფუნქციის არგუმენტი იცვლება 0–დან 4® –მდე ბიჯით ® /50 (ცხადია, რომ რაც ნაკლებია ბიჯი, მით უფრო ზუსტად აისახება ასაგები ფუნქცია. დიდი ბიჯის შემთხვევაში შესამჩნევი ხდება უბან–უბან წრფივი აპროქსიმაცია):

```
>> x=0:pi/50:4*pi;  y=sin(x).*exp(-x); plot(x,y)
```

შედეგად მივიღებთ ნახ.3.3–ზე ნაჩვენებ გრაფიკს:

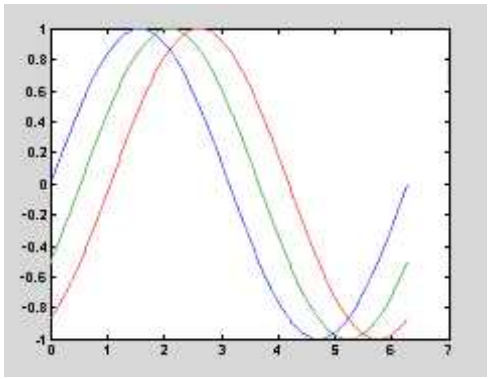


ნახ.3.3

plot(X1,Y1,X2,Y2,...) ფუნქციით, სადაც X1,Y1,X2,Y2,... ვექტორებია, ერთ გრაფიკულ არეში აიგება რამდენიმე წირი, ამასთან, საჭირო არაა წერტილების რაოდენობა ერთმანეთს ემთხვეოდეს. MATLAB–ი ავტომატურად მიუჩენს წირებს სხვადასხვა ფერს. მაგ.,

```
>> t=0:pi/100:2*pi; y1=sin(t); y2=sin(t-pi/6); y3=sin(t-pi/3);
>> plot(t,y1,t,y2,t,y3)
```

ოპერატორებით შეიქმნება სხვადასხვა ფერის 3 სინუსოიდა (ნახ.3.4,ა).

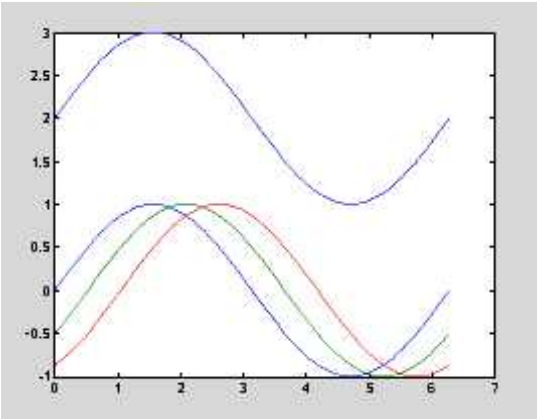


ნახ.3.4

თუ ამის შემდეგ დავწერთ

```
>> hold on
>> plot(t,y1+2)
```

ოპერატორებს, გრაფიკულ არეში შემდეგ სურათს დავინახავთ(ნახ.3.5):



ნახ.3.5

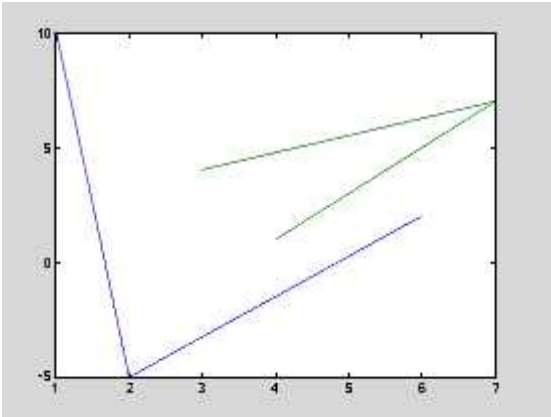
აქვე აღვნიშნოთ, რომ თუ გრაფიკის აგების ფუნქციაში არგუმენტების რაოდენობა ერთს აღემატება, კომპლექსური რიცხვების შემთხვევაში წარმოსახვითი ნაწილი იგნორირდება.

თუ გრაფიკის აგების ოპერატორებში ერთ–ერთი არგუმენტი სკალარია, ხოლო მეორე ვექტორი, აიგება ერთმანეთთან არადაკავშირებული წერტილებისაგან შემდგარი გრაფიკი, ამასთან წერტილების რაოდენობა, ცხადია, ვექტორის ელემენტთა რაოდენობის ტოლია.

თუ ეს არგუმენტები ერთნაირი ზომის მატრიცებია, აიგება Y მატრიცის ყოველი სვეტი X მატრიცის შესაბამისი სვეტის მიმართ. მაგ., შემდეგი ოპერატორებით

```
>> X=[1 3; 2 7; 6 4]; Y=[10 4; -5 7; 2 1];
>> plot(X,Y)
```

მიიღება ნახ.3.6–ზე ნაჩვენები გრაფიკი:



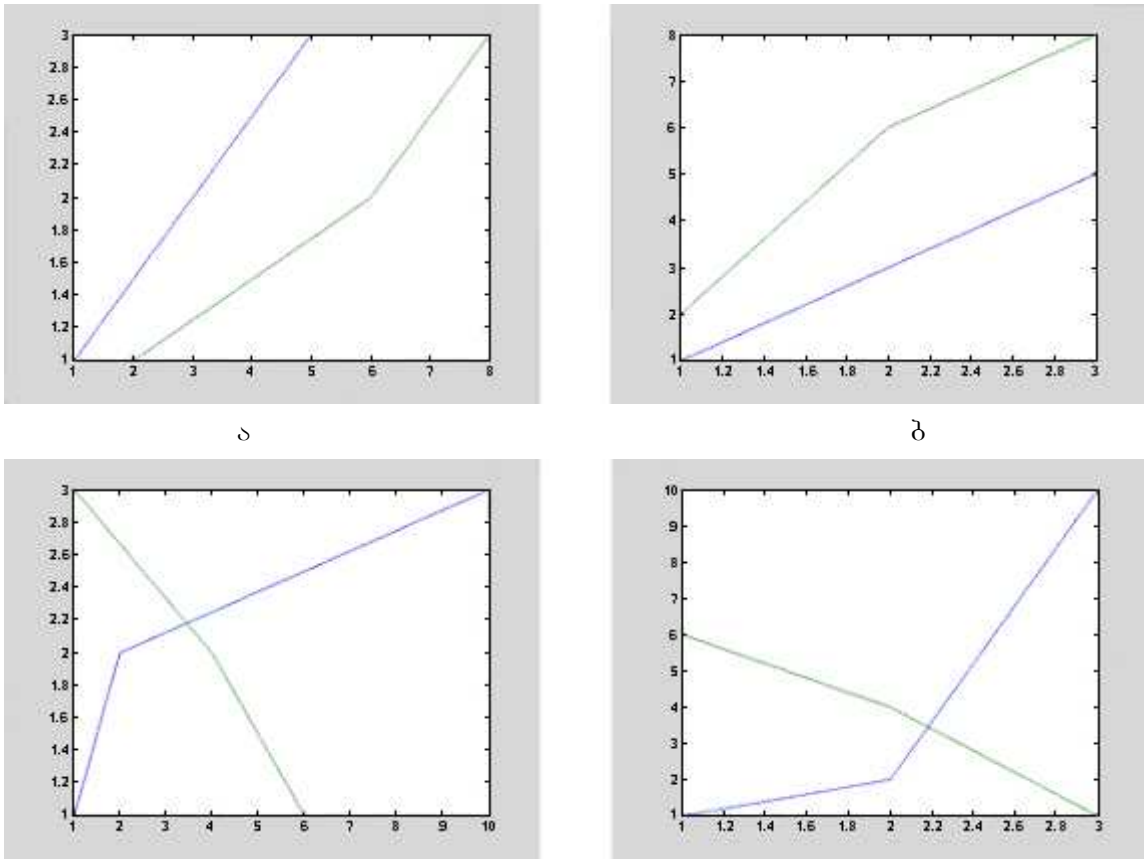
ნახ.3.6

თუ plot(X,Y) ფუნქციაში ერთ–ერთი არგუმენტი მატრიცაა, მაშინ წირები აიგება მატრიცის თითოეული სტრიქონის ან სვეტის მიმართ ერთი და იგივე გრაფიკულ არეში, ამასთან,

მატრიცაში სტრიქონების ან სვეტების რაოდენობა უნდა უდრიდეს ვექტორის ელემენტების რაოდენობას. წირების აგება სტრიქონების ან სვეტების მიმართ დამოკიდებულია იმაზე, თუ რომლის რაოდენობა ვექტორის სიგრძის ტოლია. ასე მაგ.,

```
>>  A=[1 3 5; 2 6 8]; B=[1 2 3]; plot(A,B)
>> figure;  plot(B,A)
>> figure; C=[1 6; 2 4; 10 1];  plot(C,B)
>> figure;  plot(B,C)
```

ოპერატორებით აიგება ნახ.3.7–ზე მოყვანილი წირები (ნახ.3.7,ა შეესაბამება plot(A,B)–ს, 3.7,ბ – plot(B,A))–ს, 3.7,გ – plot(C,B)–ს და 3.7,დ – plot(B,C)–ს).



ა ბ

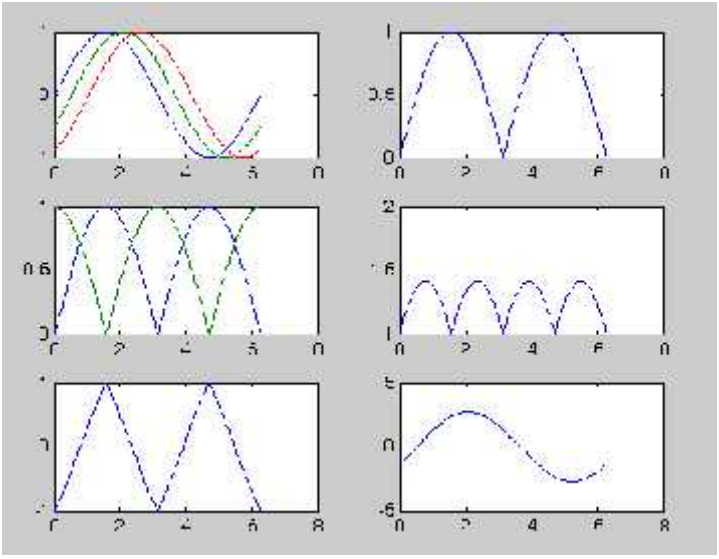
გ დ

ნახ.3.7

subplot(m,n,p) (ან **subplot(mnp)**) ფუნქცია საშუალებას გვაძლევს გრაფიკული ფანჯარა დავყოთ რამდენიმე ქვეფანჯარად. ქვეფანჯრების რაოდენობა m·n ნამრავლის ტოლია. m – ქვეგრაფიკების რაოდენობაა ვერტიკალის გასწვრივ, n – ჰორიზონტალის გასწვრივ, ხოლო p–თი განისაზღვრება თუ რომელ ქვეფანჯარაში აიგება შემდგომი plot ფუნქციით მითითებული გრაფიკი. ქვეფანჯრები დანომრილია მარცხნიდან მარჯვნივ და ზემოდან ქვემოთ. მაგ., შემდეგი ოპერატორებით ერთ გრაფიკულ არეში მიიღება ნახ.3.8–ზე ნაჩვენები გრაფიკები:

```
>> t=0:pi/100:2*pi; y1=sin(t); y2=sin(t-pi/6); y3=sin(t-pi/3);
>> subplot(3,2,1); plot(t,y1,t,y2,t,y3); subplot(3,2,2); z1=abs(sin(t)); plot(t,z1);
```

>> subplot(3,2,3); z2=abs(cos(t)); plot(t,z1,t,z2); subplot(3,2,4); plot(t,z1+z2);
>> subplot(3,2,5); plot(t,z1-z2); subplot(3,2,6); plot(t,y1+y2+y3)



ნახ.3.8

დაბოლოს, განვიხილოთ plot ფუნქციის მოდიფიკაცია – fplot ფუნქცია. მას ორი არგუმენტი გააჩნია – ასაგები ფუნქციის სახელი (Y) და საზღვრები: **fplot(@Y,[საზღვრები])** (ან **fplot('Y',[საზღვრები])**). ფუნქციის მეორე არგუმენტი წარმოადგენს ორ [xmin xmax] ან ოთხკომპონენტიან [xmin xmax ymin ymax] ვექტორს. პირველ შემთხვევაში მომხმარებელი აწვდის MATLAB–ს ინფორმაციას დამოუკიდებელი ცვლადის მინიმალურ და მაქსიმალურ მნიშვნელობაზე, მეორე შემთხვევაში კი ამ ინფორმაციას ემატება ფუნქციის მნიშვნელობების ქვედა და ზედა საზღვარი. მაგ., შემდეგი ოპერატორებით აიგება სინუსოიდის გრაფიკი, რომლის არგუმენტი იცვლება 0–დან 2® –მდე: fplot(@sin,[0 2*pi]) ან fplot('sin',[0 2*pi]).

plot–თან შედარებით, fplot ფუნქცია უფრო „ინტელექტუალურია": იქ, სადაც ასაგები ფუნქციის მნიშვნელობები მკვეთრად იცვლება, fplot–ი შეამცირებს დამოუკიდებელი ცვლადის ცვლილების ბიჯს. ამრიგად, fplot–ით რაიმე ფუნქციის გრაფიკის აგებისას მომხმარებელს გარანტია აქვს, რომ მიღებულ გრაფიკსა და ფუნქციის იდეალურ გრაფიკებს შორის განსხვავება 0,2%–ს არ აღემატება.

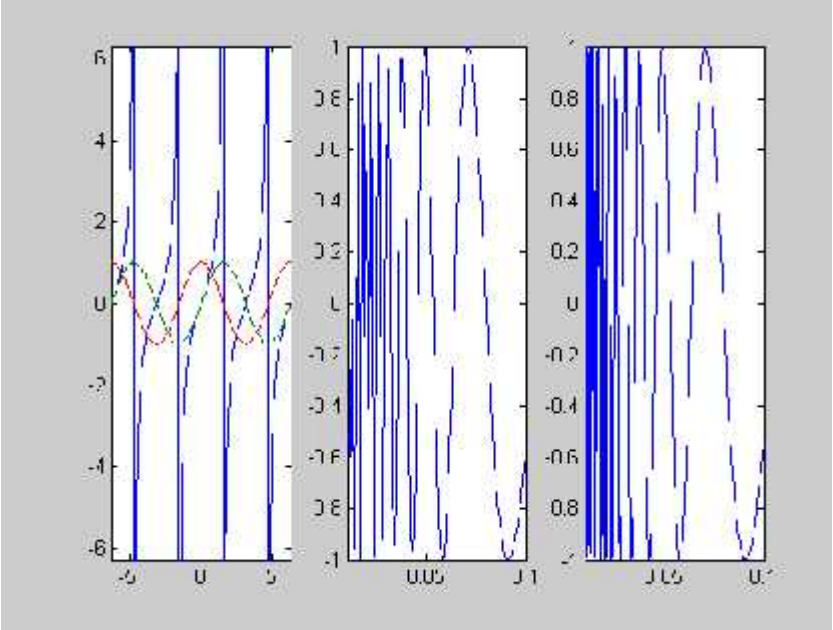
თუ გვსურს, შეგვიძლია მივუთითოთ სასურველი ფარდობითი ცდომილება – 1–ზე ნაკლები რიცხვი, მაგ., fplot(@sin, [0 2*pi], 0.05). ამ ოპერატორით აიგება სინუსოიდა, რომელიც განსხვავდება იდეალურისაგან არაუმეტეს 5%–ისა.

fplot–ს შეიძლება კიდევ ერთი პარამეტრი ჰქონდეს – მთელი ნატურალური რიცხვი n, რომლითაც მომხმარებელი მიუთითებს MATLAB–ს, რომ გრაფიკის ასაგებად საჭიროა მინიმუმ n+1 წერტილის გამოყენება. ამასთან, მნიშვნელობა არა აქვს, თუ რა თანმიმდევრობით არის მითითებული fplot–ში ფარდობითი ცდომილება და n რიცხვი: fplot(@sin, [0 2*pi], 0.05, 20) და fplot(@sin, [0 2*pi], 20, 0.05) ფუნქციები ტოლფასია.

ნახ.3.9–ზე მოყვანილია შემდეგი ოპერატორებით მიღებული გრაფიკები (აქ [-1 1 -1 1] საზღვრების წინ მოთავსებულია 2® საერთო კოეფიციენტი, რომელზედაც უნდა გამრავლდეს ყოველი საზღვარი):

>> subplot(1,3,1); fplot(‘[tan(x),sin(x),cos(x)]’, 2*pi*[-1 1 -1 1])

>> subplot(1,3,2); fplot(’sin(1./x)’, [0.01 0.1], 10e-3); subplot(1,3,3); fplot(’sin(1./x)’, [0.01 0.1], 1e-3)



ნახ.3.9

3.2 გრაფიკების რედაქტირება

წარწერები გრაფიკზე

ცხრილი 3.1

უნქცია	ფუნქციის აღწერა
title(’ტექსტი’)	გრაფიკს გაუკეთებს სათაურს
xlabel(’ტექსტი’)	*x* ღერძის ქვემოთ წააწერს ბრჭყალებში მოთავსებულ ტექსტს
ylabel(’ტექსტი’)	*y* ღერძის გასწვრივ წააწერს ბრჭყალებში მოთავსებულ ტექსტს
text(x,y, ’ტექსტი’)	ბრჭყალებში ჩაწერილ ტექსტს წააწერს გრაფიკს წერტილში, რომლის კოორდინატებია (x, y). თუ x, y ვექტორებია, წარწერას გააკეთებს ყოველ წერტილში
gtext(’ტექსტი’)	წააწერს ტექსტს გრაფიკზე იმ წერტილში, რომელსაც თაგვით ან კლავიატურის ისრებით მივუთითებთ

აქვე აღვნიშნოთ, რომ **grid** ან **grid on** ბრძანება დაიტანს გრაფიკზე საკოორდინატო ბადეს, ხოლო **grid off** – „წაშლის“ მას.

წირის ფერის შერჩევა

გრაფიკის აგების ოპერატორებში ძირითადი პარამეტრების შემდეგ შეიძლება მოვათავსოთ დამატებითი ოფციები მისი გაფორმების გასაუმჯობესებლად. ამისათვის ბრჭყალებში უნდა მივუთითოთ ფერის, სტილის და მარკერის განმსაზღვრელები. ამასთან, სიმბოლოების თანმიმდევრობას მნიშვნელობა არა აქვს. ასე რომ plot(x,y,'g:') და plot(x,y,':g') ფუნქციები ტოლფასია. 3.2 ცხრილში მოყვანილია წირის ფერის განმსაზღვრელები:

ცხრილი 3.2

წირის ფერი	განმსაზღვრელი	წირის ფერი	განმსაზღვრელი
წითელი (ინგლ. – red)	r	შავი (black)	k
მწვანე (green)	g	ყვითელი (yellow)	y
ლურჯი (blue)	b	ჟოლოსფერი (magenta)	m
თეთრი (white)	w	ცისფერი (cyan)	c

წირის სტილის და მარკერების ფორმის შერჩევა

წინასწარი შეთანხმებით მართკუთხა და პოლარულ კოორდინატთა სისტემაში აგებული გრაფიკები უწყვეტი სტილისაა. შესაძლებელია შევცვალოთ წირის სტილი და წერტილების აღმნიშვნელი მარკერების ფორმა. 3.3 ცხრილში მოყვანილია წირის სტილის განმსაზღვრელები:

ცხრილი 3.3

წირის სტილი	განმსაზღვრელი	წირის სტილი	განმსაზღვრელი
უწყვეტი	-	პუნქტირული	:
წყვეტილი	--	წყვეტილ–პუნქტირული	-.

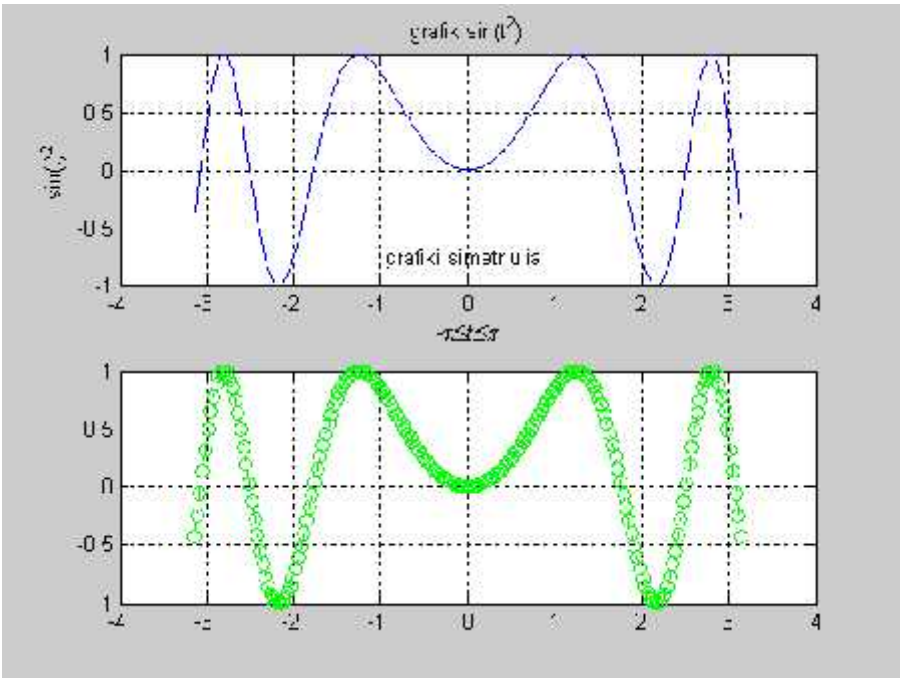
3.4 ცხრილში მითითებულია მარკერების ტიპის განმსაზვრელები, რომლებიც აგრეთვე უნდა ჩავსვათ ბრჭყალებში:

ცხრილი 3.4

მარკერის ფორმა	განმსაზღვრელი	მარკერის ფორმა	განმსაზღვრელი
წერტილი	.	v–აღნიშვნა	V
პლუსი	+	^–აღნიშვნა	^
ვარსკვლავი	*	<–აღნიშვნა	<
წრე	o	>–აღნიშვნა	>
X–აღნიშვნა	x	ხუთქიმიანი ვარსკვლავი	p
კვადრატი	s	ექვსქიმიანი ვარსკვლავი	h
რომბი	d		

ქვემოთ მოყვანილი ოპერატორებით აიგება ნახ.3.10–ზე ნაჩვენები გრაფიკები, სადაც ილუსტრირებულია წარწერების დატანა გრაფიკებზე და წირის გაფორმების შერჩევა:

```
>> t = -pi:pi/100:pi; y=sin(t.^2); subplot(2,1,1); plot(t,y)
>> title('grafik sin(t^2)');  xlabel('-\pi\leq\itt\leq\pi'); ylabel('sin(t)^2');
>> grid on;  text(-1, -0.75, ' grafiki simetriulia');
>> subplot(2,1,2); plot(t,y,'g-.o'); grid on
```



ნახ.3.10

ღერძების დაფორმატება

როგორც აღვნიშნეთ, მართკუთხა საკოორდინატო სისტემაში გრაფიკების აგებისას MATLAB–ი ავტომატურად ირჩევს ღერძების გრადაციას. **Axes** ფუნქციით შესაძლებელია შევცვალოთ გრადაციის პარამეტრები.

axis([xmin, xmax, ymin, ymax]) ფუნქციით შეიძლება შევცვალოთ *x* და *y* ღერძების მინიმალური და მაქსიმალური მნიშვნელობები.

axis square ბრძანებით დაყენდება ღერძების კვადრატული თანაფარდობა, ხოლო **axis normal** – ნორმალური. axis normal ბრძანება აუქმებს axis square და axis equal ბრძანებებით დაწესებულ შეზღუდვებს.

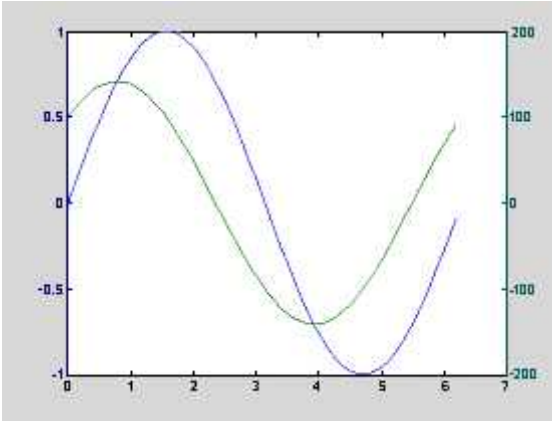
axis equal ბრძანებით ორივე ღერძზე შეიქმნემა ერთი და იგივე მასშტაბი. მაგ., თუ axis square ან axis equal ბრძანებების შემდეგ plot–ში მივუთითებთ ოვალის ამსახველ ფუნქციას, დავინახავთ არა ოვალს, არამედ წრეწირს.

axis on ბრძანება „გაყინავს“ ღერძების დაფორმატების მიმდინარე პარამეტრებს. ავტომატურ რეჟიმში დასაბრუნებლად ვიმეორებთ იგივე ბრძანებას. მიმდინარე დაფორმატების პარამეტრების გაუქმება შესაძლებელია აგრეთვე **axis off** ბრძანებით.

axis auto ბრძანებით MATLAB–ი ღერძების გრადაციის შერჩევას ავტომატურ რეჟიმში გააგრძელებს.

თუ ერთ ფანჯარაში ორი გრაფიკის აგებისას შეიქმნა გრადაციასთან დაკავშირებული პრობლემები (ეს შეიძლება მოხდეს მაშინ, როდესაც ან არგუმენტების, ან ფუნქციის მნიშვნელობები მკვეთრად განსხვავებულია), შესაძლოა ვისარგებლოთ **plotyy** ფუნქციით. ეს ფუნქცია საშუალებას გვაძლევს პირველი ფუნქციის არგუმენტის და თვით ფუნქციის გრადაცია მოვახდინოთ *x* ღერძის ქვემოთ და *y* ღერძის მარცხნივ, ხოლო მეორის – არგუმენტის გრადაცია ზემოთ, ხოლო ფუნქციის – მარჯვნივ (ნახ.3.11, რომელიც მიიღება ქვემოთ მოცემული ოპერატორებით):

```
>> x=0:0.1:6.28; yl=sin(x); y2=100*(sin(x)+cos(x));
>> plotyy(x,yl,x,y2)
```



ნახ.3.11

განმარტებების დატანა

თუ ერთ გრაფიკულ არეში აგებულია რამდენიმე წირი, სასურველია გრაფიკზე განმარტებების (წირების დახასიათების) დატანა.

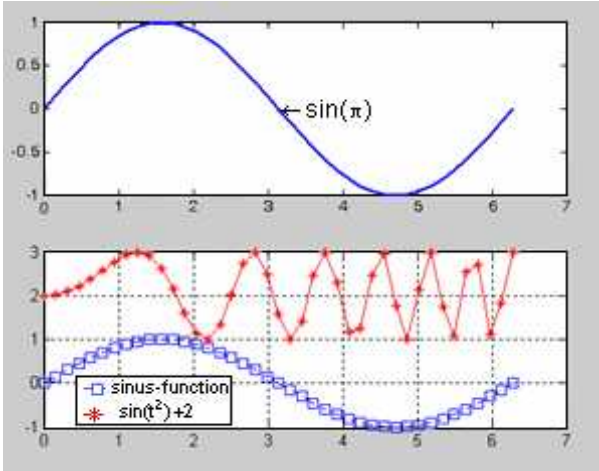
LEGEND(სტრიქონი1, სტრიქონი2, სტრიქონი3, ...) ფუნქციით განმარტება (legend) მოთავსებული იქნება მიმდინარე გრაფიკზე. განმარტების დატანა შესაძლებელია X–Y გრაფიკზე, bar გრაფიკზე, pie–ზე და ა.შ. ამასთან, legend–ის შრიფტი და მისი ზომა ემთხვევა ღერძების შრიფტსა და ზომას. ორმაგი დაწკაპუნება legend–ზე შეგვიყვანს მისი რედაქტირების რეჟიმში. შესაძლებელია განმარტების სხვა ადგილზე გადატანა, რისთვისაც ვსარგებლობთ *drag and drop* ტექნიკით. **legend(...,n)** ფუნქციის n პარამეტრით (რომელიც პრინციპში ფრჩხილების შიგნით ნებისმიერ ადგილას შეიძლება იყოს მოთავსებული) განისაზღვრება განმარტების ადგილმდებარეობა:

- 1 – განმარტება მოთავსებული იქნება ზედა მარჯვენა კუთხეში (თუ n პარამეტრს არ მივუთითებთ, წინასწარი შეთანხმებით (*by Default)*) legend–ის დატანა სწორედ აქ მოხდება);
- 2 – განმარტება მოთავსებული იქნება ზედა მარცხენა კუთხეში;
- 3 – განმარტება მოთავსებული იქნება ქვედა მარცხენა კუთხეში;

- 4 – განმარტება მოთავსებული იქნება ქვედა მარჯვენა კუთხეში;
- -1 – განმარტება მოთავსებული იქნება გრაფიკის არის გარეთ, მის მარჯვნივ და ზევით;
- 0 – სისტემა თვითონ შეირჩევს „საუკეთესო“ კუთხეს legend–სათვის.

ქვემოთ მოყვანილი ოპერატორებით აგებულია ნახ.3.12–ზე ნაჩვენები გრაფიკები. სისტემამ განმარტებებისათვის შეირჩია ქვედა მარცხენა კუთხე, მაგრამ რადგანაც მოხდა მონაცემების ნაწილობრივი გადაფარვა, *drag and drop* ტექნიკით განმარტებები ოდნავ მარჯვნივ გავწიეთ.

```
>> subplot(2,1,1); plot(0:pi/20:2*pi, sin(0:pi/20:2*pi), 'LineWidth',2)
>> text(pi,0,'\leftarrow sin(\pi)','fontsize',14)
>> subplot(2,1,2); t=0:pi/20:2*pi; plot(t,sin(t),'b-.s',t,sin(t.^2)+2,'r-*')
>> grid on; legend('sinus-function','sin(t)^2+2',0)
```



ნახ.3.12

3.3 გრაფიკული ფანჯრის მენიუს მოკლე მიმოხილვა

გრაფიკულ ფანჯარას აქვს თავისი მენიუ და ინსტრუმენტთა პანელი, რომელთა მეშვეობით შესაძლოა მივმართოთ გრაფიკული ობიექტის თვისებების რედაქტორს, შევინახოთ ის fig გაფართოებით, გადავიდეთ MATLAB–ის ბრძანებათა ფანჯარაში და იქ შევასრულოთ ნებისმიერი მოქმედება და ა.შ. განვიხილოთ ზოგიერთი ამ შესაძლებლობებიდან.

მენიუს პუნქტი File (ფაილი)

File პუნქტის პირველი ჯგუფის ბრძანებების მეშვეობით შეგვიძლია დავხუროთ გრაფიკული ფანჯარა (**Close**), გავხსნათ არსებული გრაფიკული ფაილი (**Open...**), შევქმნათ ახალი (**New**) M–ფაილი, გრაფიკული ფანჯარა და ა.შ. **New Figure** და **Open File** ღილაკები ინსტრუმენტთა პანელზედაც გვხვდება.

ბრძანებათა მეორე ჯგუფში შედის **Save** – გრაფიკული ფანჯრის შენახვა არსებული სახელით (ღილაკი **Save Figure** ინსტრუმენტთა პანელზედაც გვაქვს), **Save As...** – გრაფიკული ფანჯრის შენახვა ახალი სახელით და **Generate M-File...** – M–ფაილის გენერირება. ამ ბრძანებით MATLAB–ი ავტომატურად შექმნის პროგრამას გრაფიკულ ფანჯარაში არსებული წირების ასაგებად. როგორც წესი, შექმნილი პროგრამა საწყისთან შედარებით უფრო დიდია, რაც აიხსნება წირის ყველა ელემენტის დეტალური აღწერით. ასეთი პროცედურის ჩატარება გამართლებულია, როდესაც მომხმარებელს „ხელით“ აქვს შეცვლილი გრაფიკის ზოგიერთი პარამეტრი. მაგ., ნახ.3.12–ზე ნაჩვენები გრაფიკული არისათვის ამ ბრძანებით გენერირებული M–ფაილი ასე გამოიყურება:

```
function createfigure(X1, Y1, YMatrix1)
%CREATEFIGURE(X1,Y1,YMATRIX1)
%  X1:  vector of x data
%  Y1:  vector of y data
%  YMATRIX1:  matrix of y data
%  Auto-generated by MATLAB on 06-Mar-2011 22:41:20
% Create figure
figure1 = figure;
% Create subplot
subplot1 = subplot(2,1,1,'Parent',figure1);
box('on');
hold('all');
% Create plot
plot(X1,Y1,'Parent',subplot1);
% Create text
text('Parent',subplot1,'String','\leftarrow sin(\pi)',...
    'Position',[3.142 0 0],...
    'FontSize',14);
% Create subplot
subplot2 = subplot(2,1,2,'Parent',figure1);
box('on');
grid('on');
hold('all');
% Create multiple lines using matrix input to plot
plot1 = plot(X1,YMatrix1,'Parent',subplot2);
set(plot1(1),'Marker','square','LineStyle','-.','Color',[0 0 1],...
    'DisplayName','sinus-function');
set(plot1(2),'Marker','*','Color',[1 0 0],'DisplayName','sin(t)^2+2');
% Create legend
legend1 = legend(subplot2,'show');
set(legend1,'Location','Best');
```

Import Data... , **Save Workspace As…** და **Preferences** დუბლირებენ ბრძანებათა ფანჯრის მენიუს ანალოგიურ ბრძანებებს.

File პუნქტის ბოლო ჯგუფის ბრძანებებით ხდება გრაფიკების ბეჭდვისათვის მომზადება. **Export Setup...** ბრძანებით გამოიძახება დიალოგური ფანჯარა, სადაც შესაძლოა გრაფიკული ფანჯრის და წირების მიმდინარე პარამეტრების შეცვლა. **Print Preview...** ბრძანებით შევდივართ ბეჭდვისწინა დათვალიერების რეჟიმში, ხოლო **Print...** ბრძანება ემსახურება

გრაფიკების დაბეჭდვას. **Print Figure** ღილაკი ინსტრუმენტთა პანელზედაც გვხვდება. გრაფიკის დაბეჭდვა პრინტერზე შესაძლებელია აგრეთვე ბრძანებათა ფანჯრიდან **print** ბრძანებით.

მენიუს პუნქტი Edit (რედაქტირება)

Edit პუნქტის ბრძანებები ემსახურება წირების რედაქტირებას. აქ შედის როგორც სტანდარტული ბრძანებები, რომლებიც არ საჭიროებს დამატებით ახსნა–განმარტებას (**Undo** – გაუქმება, **Redo** – აღდგენა, **Cut** –ამოჭრა, **Copy** – კოპირება, **Paste** – ჩასმა, **Delete** – წაშლა, **Select All** – ყველაფრის გამოყოფა, **Clear Figure** – გრაფიკული ფანჯრის გასუფთავება, **Clear Workspace** – სამუშაო სივრცის გასუფთავება და სხვ.), ასევე MATLAB–ის გრაფიკული შესაძლებლობებისათვის დამახასიათებელი სპეციფიკური ბრძანებები.

Edit→Figure Properties... ბრძანებით გამოიძახება გრაფიკული ობიექტის თვისებების რედაქტორი (**Property Editor - Figure**), რომელიც მოთავსდება გრაფიკულ ფანჯარაში გრაფიკის ქვევით. ამ რედაქტორში შესაძლებელია გრაფიკული ობიექტის სახელის (**Figure Name**), მისი ფერის პალიტრის (**Colormap**) და გრაფიკული არის ფერის (**Figure Color**) შეცვლა.

ამა თუ იმ წირზე დაწკაპუნებით წირი გამოიყოფა, ერთდროულად შეიცვლება თვისებების რედაქტორის როგორც სათაური (**Property Editor - Lineseries**), ისე შინაარსიც. ამ რედაქტორში შესაძლებელია წირის გაფორმების ყველა მახასიათებლის კორექტირება და აგრეთვე საწყისი მონაცემების შეცვლა, რის შემდეგაც საჭიროა მოვახდინოთ მონაცემების განახლება **Refresh Data** ღილაკის დაჭერით, რათა ახალი მონაცემები აისახოს წირზე.

გრაფიკის რომელიმე ღერძის გამოყოფით თვისებების რედაქტორი გარდაიქმნება ღერძების პარამეტრების რედაქტორად. ამ რედაქტორის გამოძახება შესაძლოა აგრეთვე **Edit→Axes Prorerties...** ბრძანებით. **Property Editor - Axes** ჩანართებში შეიძლება მოვახდინოთ ღერძების სახელების, ცვლადების საზღვრებისა და მასშტაბის რედაქტირება, ღერძების მიმართულების საწინააღმდეგო მიმართულებით შეცვლა (**Reverse**) და ა.შ.

თვისებების ნებისმიერ რედაქტორს აქვს **More Properties...** ღილაკი. ამ ღილაკით გამოიძახება თვისებების ინსპექტორი (**Inspector**), რომელშიც ასახულია გრაფიკის ყველა თვისება. შეიძლება მოვახდინოთ ნებისმიერი მათგანის კორექტირება.

თვისებების რედაქტორის გამოძახებისას ინსტრუმენტთა პანელის მარჯვენა ნაწილში დავინახავთ **Tile**, **Left/Right Tile**, **Top/Bottom Tile**, **Float**, **Maximize** ღილაკებს, რომელთა მეშვეობით შესაძლებელია გრაფიკულ ფანჯარაში გრაფიკის ადგილმდებარეობის შეცვლა.

რედაქტირების რეჟიმში შესვლა შესაძლებელია აგრეთვე ინსტრუმენტთა პანელის **Edit Plot** ღილაკის დაჭერით, რის შემდეგ ობიექტზე დაწკაპუნება გამოყოფს მას, ხოლო ორმაგი წკაპუნით მოხდება შესაბამისი თვისებების რედაქტორის გამოძახება.

მენიუს პუნქტი View (ხედი)

View პუნქტის ქვეპუნქტებია: გრაფიკული ფანჯრის (**Figure Toolbar**), კამერის (**Camera Toolbar**) და გრაფიკის რედაქტირების (**Plot Editor Toolbar**) ინსტრუმენტთა პანელები, გრაფიკული ფანჯრის (**Figure Palette**) და გრაფიკული ობიექტის თვისებების (**Property Editor**) რედაქტორები, გრაფიკული ობიექტის დათვალიერება (**Plot Browser**). ამ ბრძანებების

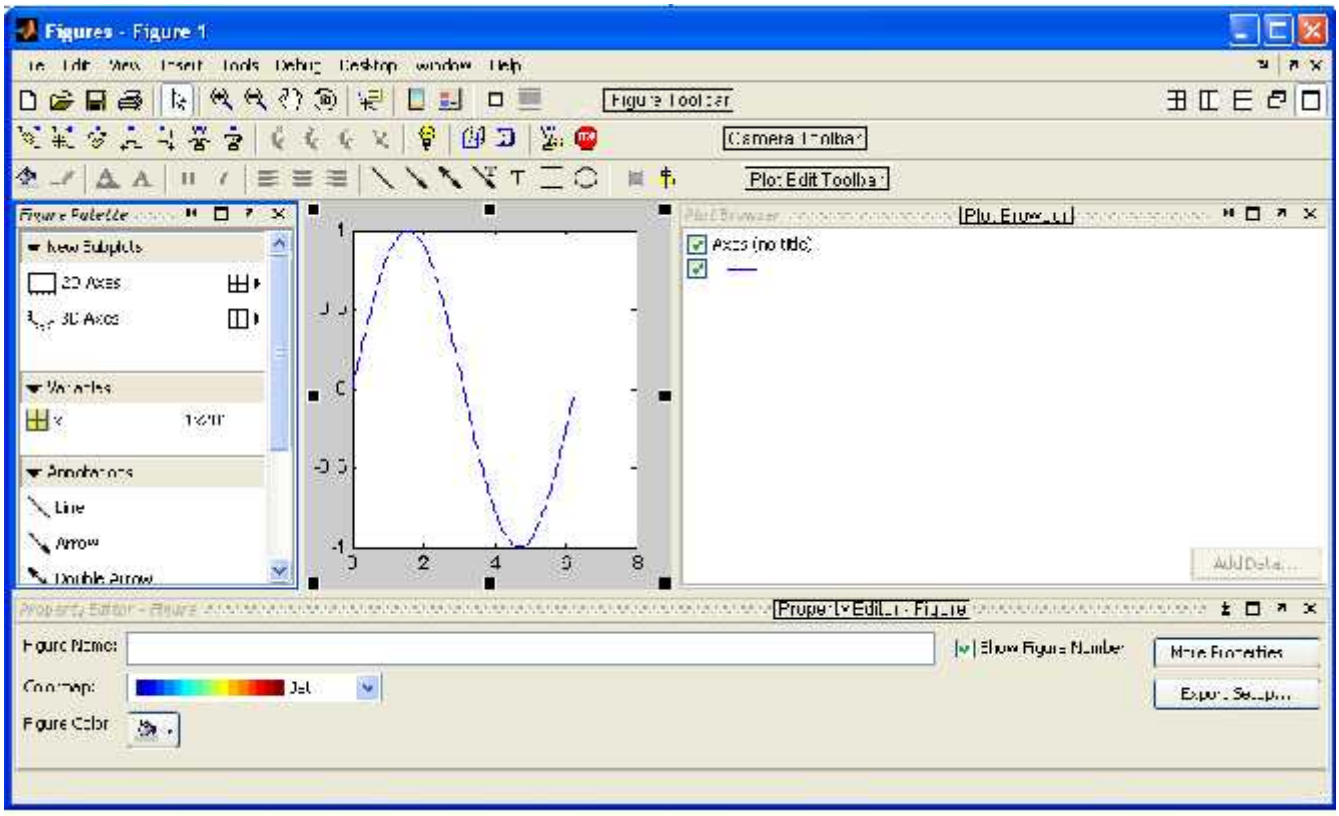
მონიშვნით ხდება შესაბამისი პანელების/რედაქტორების დაყენება, ხოლო ხელმეორე წკაპუნით – მოხსნა. ნახ.3.13–ზე ნაჩვენებია გრაფიკული ფანჯარა, სადაც დაყენებულია ყველა ინსტრუმენტთა პანელი და რედაქტორი.

Camera განსაზღვრავს დამზერის წერტილს – რაკურსს, რომლითაც ჩანს გრაფიკი, სინათლის სხივის წყაროს პარამეტრებს, სინათლის სხივების გავრცელებასა და არეკვლას.

Plot Browser ფანჯარაში ასახულია გრაფიკულ ფანჯარაში მოთავსებული ობიექტების სია და თვისებები.

გრაფიკული ფანჯრის რედაქტორში შესაძლებელია ახალი ქვეფანჯრების შექმნა (**New Subplots**), ისრების, ხაზების, ელიპსების ჩასმა და სხვ.

გრაფიკის რედაქტირების ინსტრუმენტთა პანელის ღილაკებითაც შესაძლებელია შევქმნათ ისრები, მართკუთხედები, ელიპსები და სხვ. აქვე გვაქვს საშუალება შევცვალოთ წირების ფერი, ჩასმული ტექსტის გაფორმება, ვამოვიძახოთ **Align Distribute Tool** პანელი და სხვ.



ნახ.3.13

მენიუს პუნქტი Insert (ჩასმა)

მენიუს ჩასმის პუნქტი შეიცავს ბრძანებებს, რომელთა მეშვეობით ხორციელდება გრაფიკულ არეში გაფორმების სხვადასხვა ელემენტების ჩასმა: ღერძებზე წარწერების – **Label**, სათაურის – **Title**, განმარტების – **Legend**, სწორი ხაზის – **Line**, სხვადასხვა სახის ისრების – **Arrow**, **Text Arrow**, **Double Arrow**, ჩარჩოს შიგ ტექსტით – **TextBox**, მართკუთხედის – **Rectangle**, ელიპსის – **Ellipse**, ფერების პანელის – **Colorbar** და სხვ. ფერების

პანელის და განმარტების ჩასმა შესაძლებელია აგრეთვე გრაფიკული ფანჯრის ინსტრუმენტთა პენელიდან, ხოლო ისრების, მართკუთხედების და სხვ. – გრაფიკის რედაქტირების პანელზე მდებარე ღილაკებით.

მენიუს პუნქტი Tools (ინსტრუმენტები, სერვისი)

სერვისის პუნქტი შეიცავს შემდეგ ღილაკებს: **Edit Plot** – გრაფიკის რედაქტირება; **Zoom In** და **Zoom Out** – მასშტაბის გადიდება და დაპატარავება**; Pan** – წირის „მიმაგრება“ თაგვის მაჩვენებელთან, რის შემდეგ შესაძლებელი ხდება მისი გადაადგილება (ამ ოპერაციის შესრულებისას თაგვის მაჩვენებელს ხელისგულის ფორმა აქვს); **Rotate 3D** – სივრცული მობრუნება (ამისთვის დავაჭიროთ თაგვის მარცხენა ღილაკს, არ გავუშვათ და ვამოძრაოთ); **Data Cursor** – წირის წერტილების კოორდინატების ამოკითხვა. ყველა ეს ბრძანება დუბლირებულია შესაბამისი ღილაკებით გრაფიკული ფანჯრის ინსტრუმენტთა პანელზე. **Reset View** ბრძანებით გადაადგილებული წირი საწყისს მდგომარეობაში ბრუნდება.

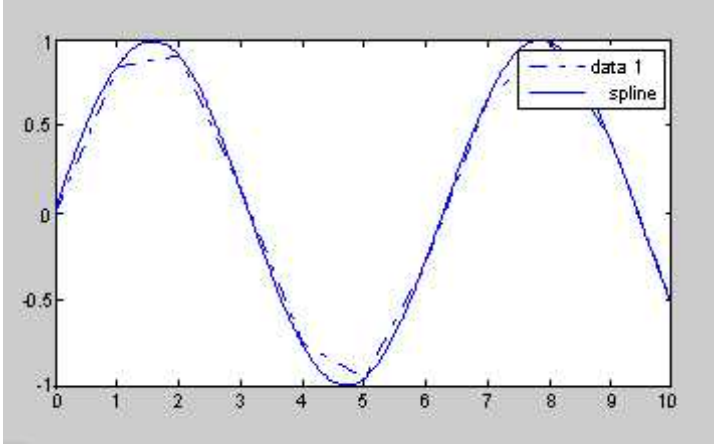
Options–ში შედის შემდეგი ოფციები: **Unconstrained Zoom**, **Unconstrained Pan** – ბუნებრივი მასშტაბირება (ერთდროულად ყველა ღერძისათვის) და წირის ბუნებრივი გადაადგილება (თაგვის მოძრაობის ტრაექტორიის შესაბამისად), **Horizontal Zoom**, **Horizontal Pan** – მასშტაბირება და გადაადგილება ჰორიზონტალის გასწვრივ, **Vertical Zoom**, **Vertical Pan** – მასშტაბირება და გადაადგილება ვერტიკალის გასწვრივ, **Display Cursor as Datatip** – წირის წერტილების კოორდინატების ჩვენება კურსორის გვერდით, **Display Cursor in Window** – წირის წერტილების კოორდინატების ჩვენება გრაფიკის ქვედა ნაწილში.

Pin to Axes (ღერძებთან „ჭიკარტით“ მიმაგრება) ბრძანებით ობიექტი – ეს შეიძლება იყოს ისარი, მართკუთხედი და სხვ. – შეიძლება წავიღოთ გრაფიკის ნებისმიერ ადგილას და იქ „მივამაგროთ“. თუ ამის შემდეგ, ვთქვათ, გადავაადგილებთ წირს ან შევასრულებთ სივრცულ მობრუნებას, მიმაგრებული ობიექტი გადაადგილდება წირთან ერთად.

სერვისის პუნქტი შეიცავს აგრეთვე მონიშნული გრაფიკული ობიექტების ბადის კვანძებთან „მიბმის“ ბრძანებებს, ურთიერთსწორებასთან (**Align**) და განლაგებასთან (**Distribute**) დაკავშირებულ შესაძლებლობებს. აქვე აღვნიშნოთ, რომ რამდენიმე გრაფიკული ობიექტის მონიშვნა მათზე დაწკაპუნებით და ერთდროულად Shift კლავიშის დაჭერით ხდება.

ძალზე საინტერესოა **Tools→Basic Fitting** (ბაზური დაგლუვება) ბრძანება. ვთქვათ, ფუნქციის აგებისას არგუმენტის ცვლილების ბიჯი ისეთია, რომ მიღებულია ფუნქციის არადამაკმაყოფილებელი აპროქსიმაცია. რა თქმა უნდა, შეიძლება შევცვალოთ ბიჯი. მაგრამ თუ ფუნქცია მიღებულია რთული გამოთვლების შედეგად, მომხმარებელს მაინცდამაინც არ სურს პროგრამის შეცვლა და ხელახალი გაშვება. ასეთ შემთხვევებში **Basic Fitting** ბრძანება ეკრანზე გამოიყვანს დიალოგურ ფანჯარას, ცხრილის სახით მოცემული ფუნქციისათვის ინტერპოლაციის მეთოდის შესარჩევად, რის შემდეგაც მოახდენს წირის დაგლუვებას.

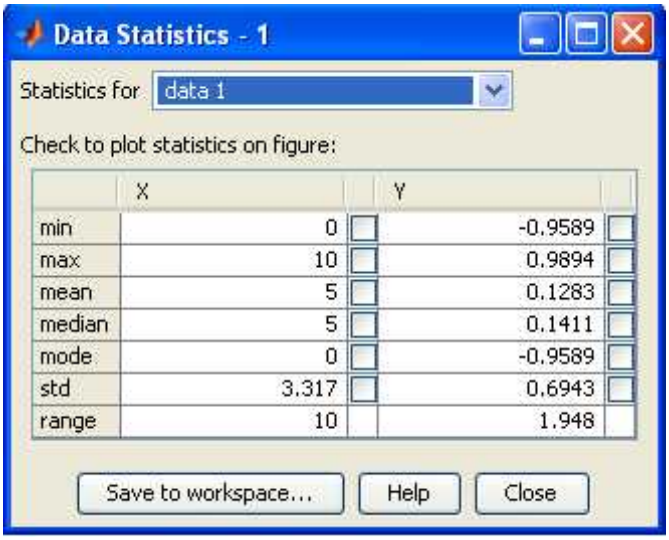
ნახ.3.14–ზე ნაჩვენებია x=0:10; plot(x,sin(x),'-.') ოპერატორებით აგებული სინუსოიდა (წყვეტილ–პუნქტირული წირი) და მისი სპლაინური ინტერპოლაციით მიღებული გაგლუვების შედეგი (უწყვეტი წირი).



ნახ.3.14

აქვე აღვნიშნოთ, რომ Command Window–ში მოცემული **[x,y]=ginput(n)** ოპერატორი საშუალებას გვაძლევს მაუსის მეშვეობით შევარჩიოთ გრაფიკულ ფანჯარაში n რაოდენობის ნებისმიერი წერტილი. მოხდება ამ წერტილების კოორდინატების დაფიქსირება – შეიქმნება x, y ვექტორები კოორდინატთა შესაბამისი მნიშვნელობებით. **[x,y]=ginput** საშუალებას გვაძლევს შევარჩიოთ წერტილთა შეუზღუდავი რაოდენობა. შეიქმნება x, y ვექტორები მითითებული წერტილების კოორდინატთა მნიშვნელობებით. ოპერატორის მოქმედებას წყვეტს Enter კლავიში. ამ ოპერატორით შეიძლება მივიღოთ, მაგ., გაგლუვებული წირის ან რთული გამოთვლების შედეგად მიღებული ფუნქციის ცხრილური სახე.

სერვის პუნქტის **Data Statistic** ბრძანება გამოიყვანს სტატისტიკურ მონაცემებს გრაფიკის შესახებ: მინიმალურ და მაქსიმალურ მნიშვნელობებს, მონაცემთა საშუალოს, მედიანას, მოდას, სტანდარტულ (საშუალო კვადრატულ) გადახრას და დიაპაზონს. ნახ.3.14–ზე მოყვანილი სინუსოიდისათვის (წყვეტილ–პუნქტირული წირი) სტატისტიკური მონაცემები შემდეგნაირად გამოიყურება (ნახ.3.15):



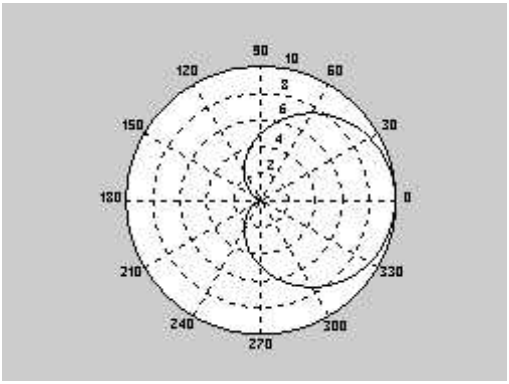
ნახ.3.15

3.4 პოლარული გრაფიკის აგება

M(r,θ) წერტილის ადგილმდებარეობა კოორდინატთა პოლარულ სისტემაში განისაზღვრება ორი სიდიდით: რადიუს–ვექტორის სიგრძით, რომელიც აერთებს კოორდინატთა სათავეს M წერტილთან, და რადიუს–ვექტორსა და *x* ღერძს შორის არსებული კუთხით. კუთხის მნიშვნელობათა დიაპაზონია 0–დან 2® რადიანამდე (0^0–დან 360^0–მდე). კუთხის ათვლა წარმოებს *x* ღერძის დადებითი მიმართულებიდან რადიუს–ვექტორამდე საათის ისრის საწინააღმდეგოდ.

MATLAB–ში პოლარული გრაფიკი აიგება **polar(theta,r)** ფუნქციით, რომლის პირველი არგუმენტია θ კუთხე, მეორე კი – r.

მაგალითისათვის ნახ.3.16–ზე მოყვანილია ე.წ. კარდიოიდის გრაფიკი, რომელიც alfa=0:0.1:2*pi; r=5*(1+cos(alfa)); polar(alfa,r,'k') ოპერატორებით აგებულია:

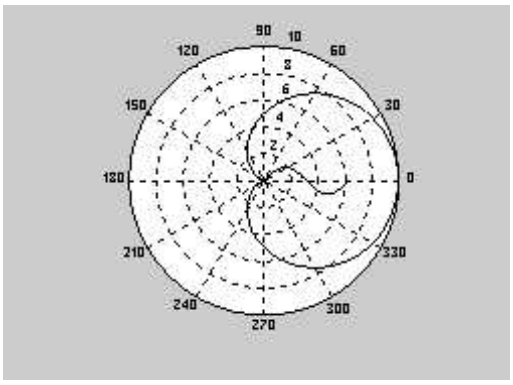


ნახ.3.16

როგორც ვიცით, **hold on** ბრძანებით მომდევნო გრაფიკები აიგება არსებულ გრაფიკულ არეში. საინტერესოა, რომ შესაძლებელია ერთ გრაფიკულ არეში ავაგოთ როგორც X–Y გრაფიკი, ისე პოლარულიც, რაც ილუსტრირებულია 3.17 ნახაზზე. გავაგრძელოთ წინა მაგალითის ოპერატორები:

>> hold on; plot(alfa,sin(alfa),'k')

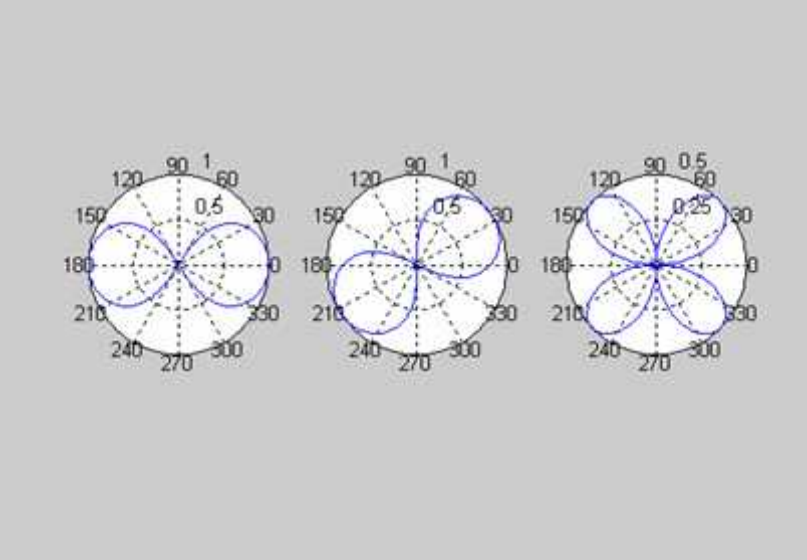
მივიღებთ



ნახ.3.17

ქვემოთ მოყვანილი ოპერატორებით ერთი გრაფიკული არის სამ ქვეარეში შექმნილია პოლარული გრაფიკები (ნახ.3.18):

```
>> gama=0:pi/100:2*pi; subplot(1,3,1); polar(sin(gama),cos(gama));
>> subplot(1,3,2); polar(sin(gama)+10,cos(gama)); subplot(1,3,3); polar(gama,sin(gama).*cos(gama))
```



ნახ.3.18

3.5 bar, stairs, stem, pie და hist გრაფიკები

სვეტოვანი დიაგრამის აგება

bar(Y) ააგებს bar ტიპის გრაფიკს – სვეტოვან დიაგრამას, რომელიც ასახავს Y ვექტორის ელემენტების მათ ინდექსებზე დამოკიდებულებას. იმ შემთხვევაში, როდესაც Y მატრიცაა, აიგება მატრიცის თითოეული სვეტის ელემენტების მათ პირველ ინდექსზე დამოკიდებულების ამსახველი სვეტოვანი დიაგრამა. **bar(X,Y)** ააგებს bar გრაფიკს, რომელიც ასახავს Y ვექტორის X ვექტორზე დამოკიდებულებას. იმ შემთხვევაში, როდესაც Y მატრიცაა, შეიქმნება გრაფიკი, რომელიც აჩვენებს მატრიცის სვეტების X ვექტორზე დამოკიდებულებას (ნახ.3.19,ა-1). **barh**–ით შეიქმნება ჰორიზონტალურად ორიენტირებული სვეტოვანი დიაგრამა (ნახ.3.19,ა-2).

bar ფუნქციაში შეიძლება მივუთითოთ სვეტების დაჯგუფების სასურველი ვარიანტი – grouped (**bar(...,'grouped') –** ნახ.3.19,ა-1) ან stacked (**bar(...,'stacked') –** ნახ.3.19,ა-3). ამასთან, მითითების არარსებობისას წინასწარი შეთანხმებით გამოყენებული იქნება grouped ვარიანტი. შეიძლება მივუთითოთ აგრეთვე სვეტების ფერი (ერთ–ერთი 'rgbymckw'–დან) და სიგანე. წინასწარი შეთანხმებით სვეტების სიგანე უდრის 0.8–ს.

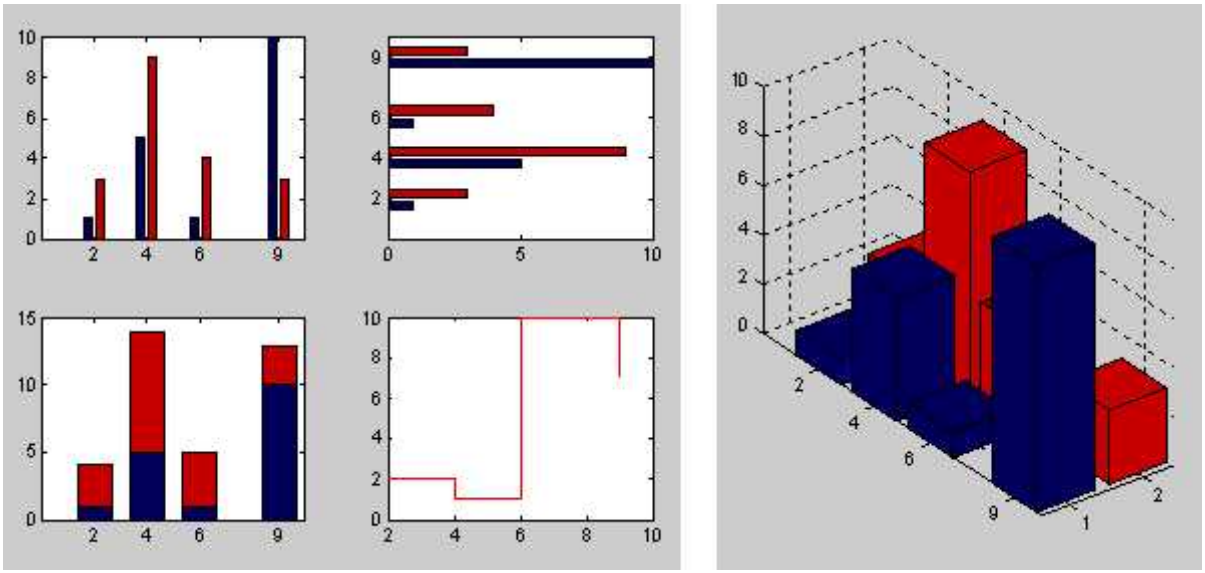
bar3 და **bar3h** ფუნქციებით სვეტოვანი დიაგრამა მიიღებს მოცულობით სახეს (ნახ.3.19,ბ), თუმცა შინაარსობრივად იგივე დამოკიდებულებას ასახავს.

საფეხურისებრი გრაფიკის აგება

stairs(Y) და **stairs(X,Y)**–ით მოხდება საფეხურისებრი გრაფიკების აგება. ამ ფუნქციების არგუმენტები bar–ის არგუმენტების ანალოგიურია. დასაშვებია, რომ X და Y არგუმენტები ერთნაირი ზომის მატრიცებიც იყოს. საფეხურისებრი გრაფიკების აგებისას შეიძლება მივუთითოთ ხაზის ფერი, სტილი და მარკერების ფორმა.

ქვემოთ მოყვანილი ოპერატორებით მიიღება ნახ.3.19 (ბოლო ორი ოპერატორით ახალ გრაფიკულ ფანჯარაში აგებულია მოცულობითი სახის სვეტოვანი დიაგრამა):

```
x=[2 4 6 9]; Z=[1 3; 5 9; 1 4; 10 3]; subplot(2,2,1); bar(x,Z, 'grouped')
>> subplot(2,2,2); barh(x,Z);  subplot(2,2,3); bar(x,Z,'stacked')
>> subplot(2,2,4);y=[2 1 10 7]; stairs(x,y,'r')
>> figure(2); bar3(x,Z)
```



ა ბ

ნახ.3.19

stem ტიპის გრაფიკის მიღება

stem(Y) და **stem(X,Y)**–ით აიგება გრაფიკი, სადაც ფუნქციის მნიშვნელობები წინასწარი შეთანხმებით წრიული ფორმის მარკერებით აღინიშნება. ფუნქციის არგუმენტები stairs ფუნქციის არგუმენტების ანალოგიურია. **stem(...,'filled')** ფუნქციით შეიქმნება stem გრაფიკი შევსებული მარკერებით. stem ფუნქციაში შეიძლება მივუთითოთ ორდინატების და მარკერების ფერი, ორდინატების სტილი და შევცვალოთ მარკერების ფორმა.

წრიული დიაგრამის აგება

pie(X) ფუნქციით შეიქმნება n სექტორისაგან შემდგარი წრიული დიაგრამა, სადაც n არის X ვექტორის ელემენტების რაოდენობა. წრიული დიაგრამა მთელის ნაწილებს შორის ფარდობით დამოკიდებულებას ასახავს. ვექტორის ელემენტების მნიშვნელობები 0–ზე მეტი და 1–ზე ნაკლები უნდა იყოს, ხოლო თუ X–ის ელემენტები 1–ზე მეტი რიცხვებია, MATLAB–ი თვითონ მიიყვანს ამ რიცხვებს $x_i/\mathrm{sum}(X)$ მნიშვნელობებამდე (ნახ.3.20,1).

თითოეული სექტორის ცენტრული კუთხე X და ფართობი ვექტორის შესაბამისი ელემენტის მნიშვნელობის პროპორციულია, ყველა ელემენტის ჯამი კი 360^0 –ს შეესაბამება. იმ შემთხვევაში, როდესაც $0 < x_i < 1$, აიგება ნაწილობრივი „წრე“ (ნახ.3.20,3). წინასწარი შეთანხმებით თითოეული სექტორის გვერდით აღინიშნება მისი პროცენტული „წვლილი“. სექტორები აიგება საათის ისრის საწინააღმდეგო მიმართულებით, დაწყებული „ჩრდილოეთიდან“.

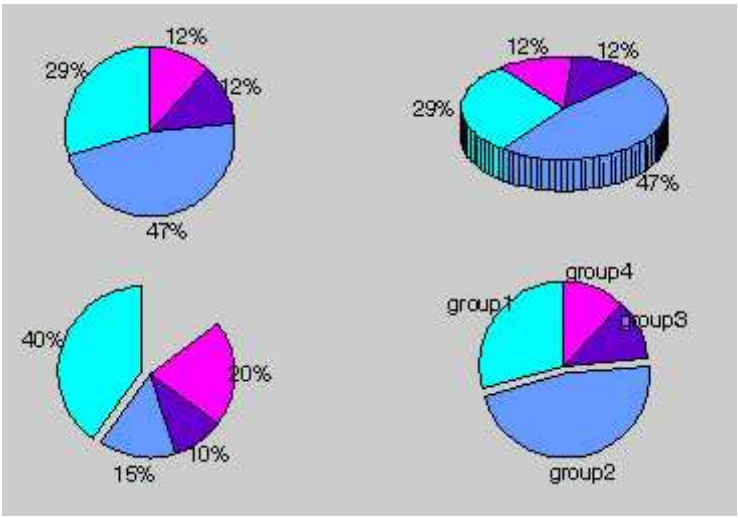
იმისათვის, რომ ყურადღება გავამახვილოთ ზოგიერთ სექტორზე ან სექტორთა ჯგუფზე, მიმართავენ მათი წრიდან ნაწილობრივ გამოწევას (ნახ.3.20,3 და 3.20,4). ამისათვის pie ფუნქციას ემატება კიდევ ერთი არგუმენტი – m ვექტორი, რომლის ელემენტების რაოდენობა X ვექტორის ელემენტების რაონების ტოლია. ამ ვექტორის 1–ის ტოლი მნიშვნელობები წრის გამოწეულ სექტორებს შეესაბამება.

იმისათვის, რომ პროცენტული „წვლილის“ მაგივრად დიგრამაზე აისახოს მომხმარებლის მიერ შექმნილი წარწერები (ნახ.3.20,4), pie ფუნქციას უნდა დაემატოს კიდევ ერთი არგუმენტი, სადაც მითითებული იქნება წარწერები. ცხადია, ეს უნდა იყოს სტრიქონული ტიპის მუდმივები.

pie3(X) ფუნქცია pie(X)–ის ანალოგიურია. წრიულ დიაგრამას ამ შემთხვევაში მოცულობითი სახე აქვს (ნახ.3.20,2). მოცულობითი წრიული დიაგრამის აგება „ჩრდილო–დასავლეთიდან“ იწყება.

ქვემოთ მოყვანილია ოპერატორები, რომელთა მეშვეობით მიღებულია ნახ.3.20. ბრძანება colormap cool ემსახურება ფერების „გაღიავებას“.

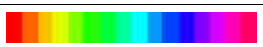
```
>> z=[25  40  10  10];  colormap cool;
>> subplot(2,2,1); pie(z); subplot(2,2,2); pie3(z);
>> subplot(2,2,3); m=[1 0 0 0];  pie([0.4  0.15  0.1  0.2], m);
>> s={'group1', 'group2', 'group3', 'group4'};  subplot(2,2,4); pie(z,[0 1 0 0],s)
```



ნახ.3.20

აქვე აღვნიშნოთ, რომ **colormap(პალიტრის სახელი)** ბრძანებით შეიძლება მივუთითოთ სასურველი პალიტრა. ცხრილში მოყვანილია ზოგიერთი პალიტრის დახასიათება.

ცხრილი 3.5

პალიტრის სახელი	პალიტრა	პალიტრის დახასიათება
autumn		შემოდგომის პალიტრა: წითლის, ნარინჯისფრისა და ყვითლის გრადაციები
bone		ნაცრისფრის პალიტრა, რომელშიც gray პალიტრასთან შედარებით ოდნავ მომატებულია ლურჯის კომპონენტი
colorcube		პალიტრა, რომელიც შეიცავს ყოველი ფერის მაქსიმალური რაოდენობის გრადაციას
cool		„ცივი“ ფერების პალიტრა: გრადაციები ცისფრიდან ჟოლოსფრამდე
copper		„სპილენძის“ პალიტრა: გრადაციები შავიდან სპილენძისფრამდე
gray		ფერები იცვლება მუქი ნაცრისფრიდან ღიამდე
hot		„ცხელი“ ფერების პალიტრა: შავის, წითლის, ნარინჯისფრის, ყვითლისა და თეთრის გრადაციები
hsv		შეიცავს ფერებს: წითელს, ნარინჯისფერს, ყვითელს, მწვანეს, ცისფერს, ლურჯს, ჟოლოსფერსა და ისევ წითელს
jet		წინასწარი შეთანხმების პალიტრა: გრადაციები ლურჯიდან წითელ ფერამდე
spring		გაზაფხულის პალიტრა: გრადაციები ჟოლოსფრიდან ყვითლამდე
summer		ზაფხულის პალიტრა: გრადაციები მწვანიდან ყვითლამდე
white		white პალიტრა შეიცავს მხოლოდ თეთრ ფერს
winter		ზამთრის პალიტრა: გრადაციები ლურჯიდან მწვანემდე

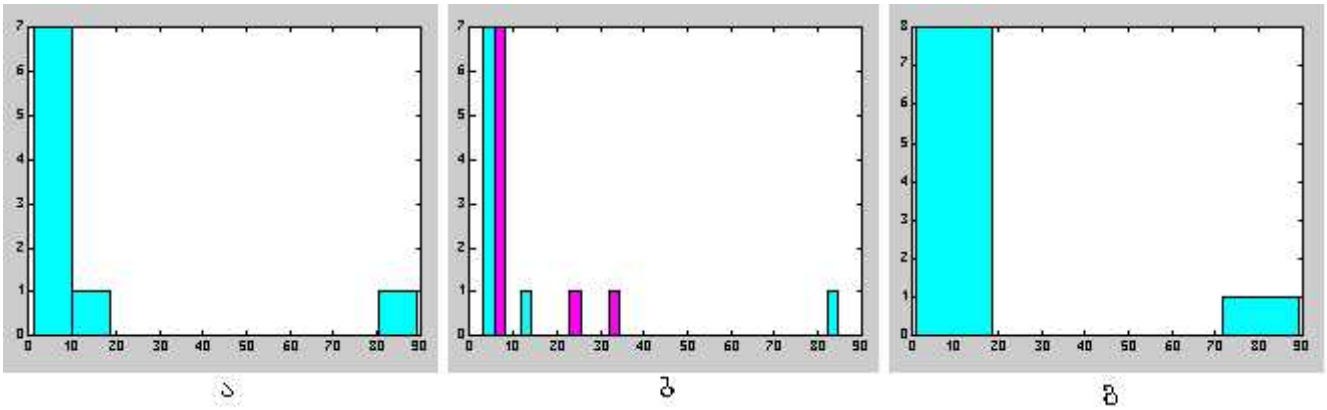
colormap('default') ბრძანება იწვევს წინასწარი შეთანხმებით გათვალისწინებული jet პალიტრის დაყენებას.

hist ფუნქცია ემსახურება ინტერვალებს შორის მონაცემთა განაწილებას. **n=hist(X)** ოპერატორის გამოყენებისას X ვექტორის ელემენტები 10 „თანაბარი მოცულობის კონტეინერში თავსდება“, ხოლო ელემენტების რაოდენობა „კონტეინერებში“ (მონაცემთა განაწილება) ჩაიწერება n ვექტორში. **hist(X)** ფუნქციით აიგება ე.წ. ჰისტოგრამა – გრაფიკი, რომელიც ასახავს 10 ინტერვალთა შორის მონაცემთა განაწილებას (ნახ.3.21,ა). თუ X მატრიცაა, მივიღებთ მონაცემთა განაწილების ვექტორს და ჰისტოგრამას ყოველი სვეტისათვის (ნახ.3.21,ბ). **n=hist(X,m)** და **hist(X,m)** ოპერატორები ახდენენ მონაცემთა განაწილებას და შესაბამისი გრაფიკის აგებას m „კონტეინერის“ შემთხვევაში (ნახ.3.24,გ). ცხადია, ეს ოპერატორები დამოუკიდებლად მუშაობენ ანუ ჰისტოგრამის ასაგებად განაწილების ვექტორის წინასწარი მიღება აუცილებელი არ არის.

ქვემოთ მოყვანილი ოპერატორებით მიღებულია განაწილების ვექტორები და ნახ.3.21.

```
>> colormap cool
>> X=[1  2  3  4  5  6  7  89  10]; z=hist(X)
```

```
z =
   7    1    0    0    0    0    0    0    0    1
>> hist(X)
>> A=[1  2  3  4  5  6  7  89  10;   2  5  4  22  8  35  2  5  7]';
>> m=hist(A); m=m'
m =
   7    1    0    0    0    0    0    0    0    1
   7    0    1    1    0    0    0    0    0    0
>> hist(A)
>> g=hist(X,5)
g =
   8    0    0    0    1
>> hist(X,5)
```



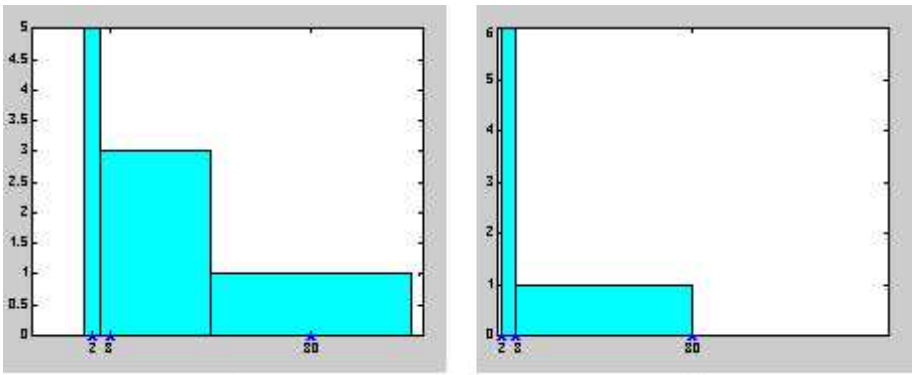
ნახ.3.21

n=hist(X,Y) და **hist(X,Y)** ოპერატორებით მიიღება X მონაცემთა განაწილება „კონტეინერებში“, რომელთა ცენტრები მოცემულია Y ვექტორში. პირველი „კონტეინერი“ შეიცავს მონაცემებს $-\infty$-დან პირველ მითითებულ ცენტრამდე, ბოლო „კონტეინერი“ – ბოლო მითითებული ცენტრიდან $+\infty$-მდე (ნახ.3.22,ა). თუ გვსურს უსასრულოების მაგივრად პირველი და ბოლო „კონტეინერებისათვის“ ვისარგებლოთ ნაპირა მნიშვნელობებით, მონაცემთა განაწილების მისაღებად უნდა გამოვიყენოთ **n=histc(X,Y)** ოპერატორი. ყოველ „კონტეინერში“ ელემენტების რაოდენობა მეტი ან ტოლია მარცხენა საზღვრის მნიშვნელობაზე და ნაკლებია მარჯვენასი, ხოლო ის მონაცემები, რომლებიც საზღვრებს გარეთაა, მხედველობაში არ მიიღება. ნაპირა მნიშვნელობების გათვალისწინებისას ჰისტოგრამის ასაგებად მივმართავთ **bar(Y,n, 'histc')** ფუნქციას (ნახ.3.22,ბ).

გავაგრძელოთ ოპერატორთა მიმდევრობა. შედეგად მივიღებთ განაწილების ვექტორებს და ნახ.3.22-ს.

```
>> z=[2  8  80];
>> v=hist(X,z)
>> v =
   5    3    1
>> hist(X,z)
>> w=histc(X,z)
```

```
w =
    6    1    0
>> bar(z,w,'histc')
```



ა ბ

ნახ.3.22

3.6 სამგამზომილებიანი ობიექტების ვიზუალიზაცია

სამგანზომილებიანი ობიექტის ვიზუალიზაცია წარმოადგენს XOY სიბრტყეზე Z მასივის ასახვას. თუ XOY სიბრტყის დაფარვის მართკუთხა ბადეს ორივე ღერძის გასწვრივ აქვს რეგულარული ბიჯი, ბადე შეიძლება განსაზღვრული იყოს ორი ვექტორით: n–ელემენტიანი x ვექტორით და m–ელემენტინი y ვექტორით. ბუნებრივია, Z მასივი ორგანზომილებიანია და მასში ელემენტების რაოდენობა mxn უდრის. ზოგადად ბიჯი შეიძლება იყოს არარეგულარულიც.

თუ x და y ვექტორების სახით არიან მოცემული, Z მასივის ელემენტების ფორმირების დაჩქარების მიზნით სასურველია ვექტორების საფუძველზე შევქმნათ მატრიცები. ამას ემსახურება **meshgrid** ფუნქცია.

[X,Y] = meshgrid(x,y) შეასრულებს x, y ვექტორების X, Y მატრიცებად ისეთ გარდაქმნას, რომლის შედეგად X მატრიცის ყველა სტრიქონი x ვექტორის ელემენტებს წარმოადგენს, ხოლო სტრიქონების რაოდენობა y ვექტორის ელემენტების რაოდენობის ტოლია, Y მატრიცაში კი ყველა სვეტი y ვექტორის ელემენტებს წარმოადგენს, ხოლო სვეტების რაოდენობა x ვექტორის ელემენტების რაოდენობის ტოლია.

მაგ.,

```
>> u=[-2,0,2]; v=[0,1,2,3]; [X,Y]=meshgrid(u,v)
X =
   -2    0    2
   -2    0    2
   -2    0    2
   -2    0    2
```

Y =

0	0	0
1	1	1
2	2	2
3	3	3

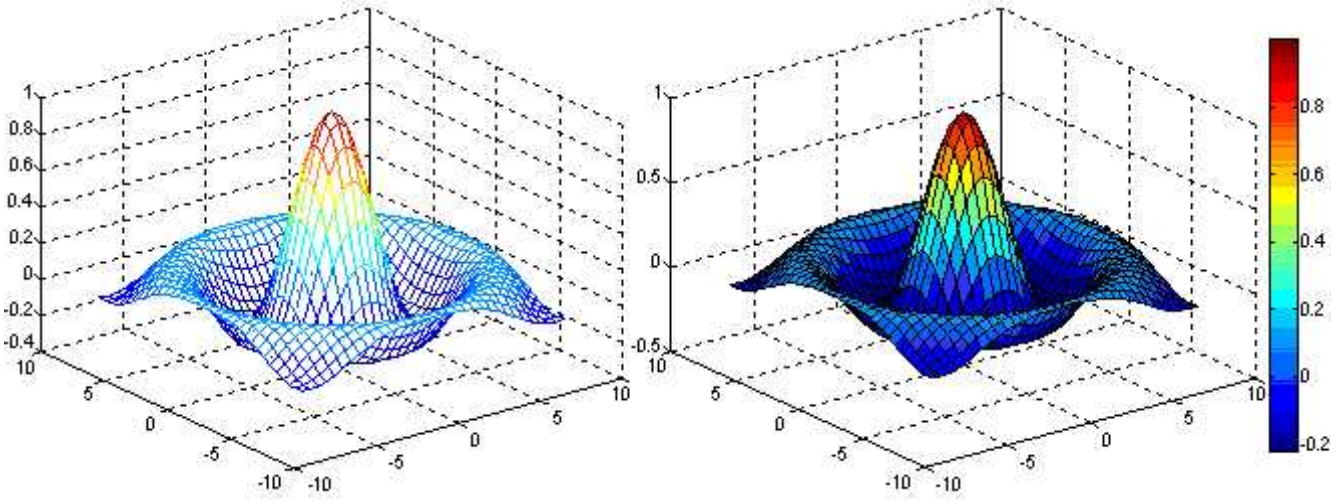
[X,Y] = meshgrid(x) შეასრულებს შემდეგ გარდაქნბას: [X,Y] = meshgrid(x,x).

აღვნიშნოთ, რომ x და y შეიძლება ელემენტების mxn რაოდენობის მქონე მატრიცების სახითაც იყოს წარმოდგენილი. ამ შემთხვევაში პირდაპირ გადავდივართ შემდეგ ეტაპზე – ბადის ხაზების გადაკვეთის წერტილებში Z მასივის ელემენტების მნიშვნელობების გამოთვლაზე.

X,Y,Z მასივების ფორმირების შემდეგ შეიძლება Z=f(X,Y) გრაფიკის შექმნა. **ზედაპირის გრაფიკის** ასაგებად ხმარობენ **mesh(X,Y,Z)** ან **surf(X,Y,Z)** ფუნქციას. ქვემოთ მოყვანილი ოპერატორებით mesh (ნახ.3.23,ა) და surf (ნახ.3.23,ბ) ფუნქციებით შექმნილია ზედაპირის გრაფიკები. ორივე გრაფიკი ერთ და იგივე დამოკიდებულებას გამოხატავს. განსხვავება ამ დამოკიდებულების ვიზუალიზაციაში ნათლად ჩანს მოყვანილი ნახატებიდან; mesh ფუნქცია აერთებს მეზობელ წერტილებს წრფის მონაკვეთებით, ამასთან, წინასწარი შეთანხმებით არ აჩენს ზედაპირის უკანა პლანზე მყოფ ხაზებს. surf ფუნქცია აფერადებს ზედაპირის თითოეულ ოთხკუთხა ელემენტს, რაც აუმჯობესებს მის აღქმას.

```
>> [X,Y]=meshgrid(-8:0.5:8); R=sqrt(X.^2+Y.^2)+eps;
>> Z=sin(R)./R; mesh(X,Y,Z)
>> figure(2)
>> surf(X,Y,Z)
```

ამ მაგალითში R – კოორდინატთა სათავიდან მანძილია, eps მუდმივის დამატება საჭიროა კოორდინატთა სათავეში განუსაზღვრელობის 0/0 ასარიდებლად.

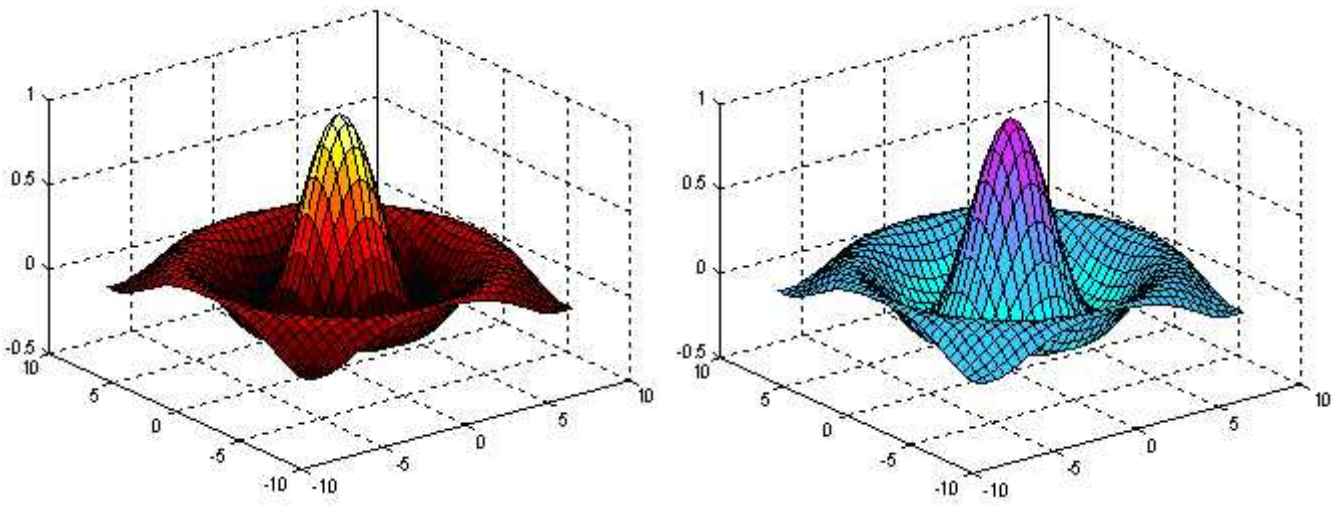


ა ბ

ნახ.3.23

როგორც ვხედავთ, ზედაპირის ელემენტები სხვადასხვა ფერისაა. წინასწარი შეთანხმებით ფერი დამოკიდებულია აპლიკატის მნიშვნელობაზე ანუ ზედაპირის სიმაღლის

პროპორციულია. ნახ.3.23,ბ–ზე გრაფიკული ფანჯრის **Insert→Colorbar** ბრძანებით ეკრანზე გამოტანილია პალიტრაც (იგივეს მივაღწევთ გრაფიკული ფანჯრის შესაბამისი ღილაკით ან პროგრამის **colorbar** ოპერატორით, რომელშიც შესაძლებელია აგრეთვე გრაფიკზე პალიტრის ადგილმდებარეობის მითითება). თავისთავად ცხადია, რომ ფერის ინდექსი, რომელიც ზედაპირის ელემენტებს მიენიჭება, შეესაბამება მიმდინარე პალიტრას. ასე, მაგ., იგივე Z ფუნქციის შესაბამისი ზედაპირი colormap hot პირობებში ნაჩვენებია ნახ.3.24,ა–ზე, ხოლო colormap cool პირობებში – ნახ.3.24,ბ–ზე.



ა ბ

ნახ.3.24

თუ მომხმარებელს არ აწყობს ფერის აპლიკატის მნიშვნელობაზე შერჩევის ხერხი, გრაფიკების აგების ფუნქციებს ამატებენ კიდევ ერთ პარამეტრს, რომელიც mxn მასივს წარმოადგენს, ზედაპირის გაფერადება ამ მასივის ელემენტების მნიშვნელობებით იქნება განპირობებული.

როგორც ვხედავთ, ეკრანზე ასახულია ზედაპირის მხოლოდ ხილული ნაწილი. **hidden off** ბრძანებით შესაძლებელია ფუნქციის გამოსახულებას დაემატოს უხილავი მხარეებიც, ხოლო **hidden on** ბრძანებით მოხდება საწყის ვარიანტზე დაბრუნება.

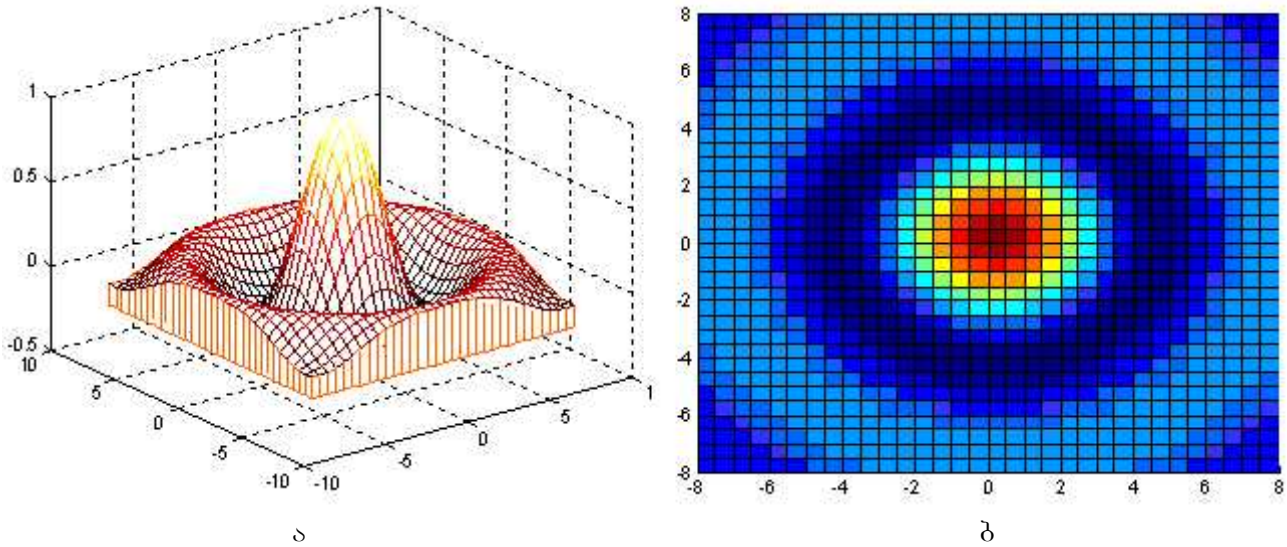
წინასწარი შეთანხმებით ყველა სახის ზედაპირის გრაფიკის აგებისას კოორდინატთა ღერძების დადებითი მიმართულებებია: *x* ღერძის – მარჯვნივ, *y* – მარცხნივ, *z* – ზემოთ, ხოლო დამკვირვებლის მდგომარეობა, რომელიც აზიმუტით და ამაღლების კუთხით განისაზღვრება, შემდეგნაირია: აზიმუტი = -37,5^0, ამაღლების კუთხე = 30^0 (აზიმუტი აითვლება *y* ღერძის უარყოფითი მიმართულებიდან, ამაღლების კუთხე კი – XOY სიბრტყიდან, ამასთან დადებითი მიმართულებებია შესაბამისად მარცხნიდან მარჯვნივ და ქვემოდან ზემოთ). ამ კუთხეების შეცვლით შესაძლებელია ზედაპირის დათვალიერება დამკვირვებლის სხვა მდგომარეობიდან. **[a,b]=view** ოპერატორით მივიღებთ აზიმუტის და ამაღლების კუთხის მიმდინარე მნიშვნელობებს, ხოლო **view(x,y)** ფუნქციით აზიმუტი გახდება x-ის ტოლი, ხოლო ამაღლების კუთხე – y-ის. გრაფიკის სივრცულ მობრუნებას და ამით ნახაზის ყოველმხრივ

დათვალიერებას მივაღწევთ გრაფიკული ფანჯრის **Tools→Rotate 3D** ბრძანებით ან ინსტრუმენტთა პანელის შესაბამისი ღილაკით.

surfl(X,Y,Z) ფუნქციით აიგება განათებული ზედაპირის გრაფიკი. ასეთი გრაფიკის აგებისას იგულისხმება, რომ სინათლის წყაროს აქვს 45^0–ზე მეტი აზიმუტი, ვიდრე დამკვირვებელს, და იგივე ამაღლების კუთხე.

meshz(X,Y,Z)–ით აიგება ზედაპირის გრაფიკი კვარცხლბეკზე (ნახ.3.25,ა – დაყენებულია colormap hot პალიტრა).

surface(X,Y,Z) აჩვენებს XOY სიბრტყის ნაწილს, რომლის ზედაპირის თითოეული ელემენტის ფერი Z მნიშვნელობაზე არის დამოკიდებული (ნახ.3.25,ბ, ფერები წინასწარი შეთანხმების პალიტრას შეესაბამება).



ა ბ

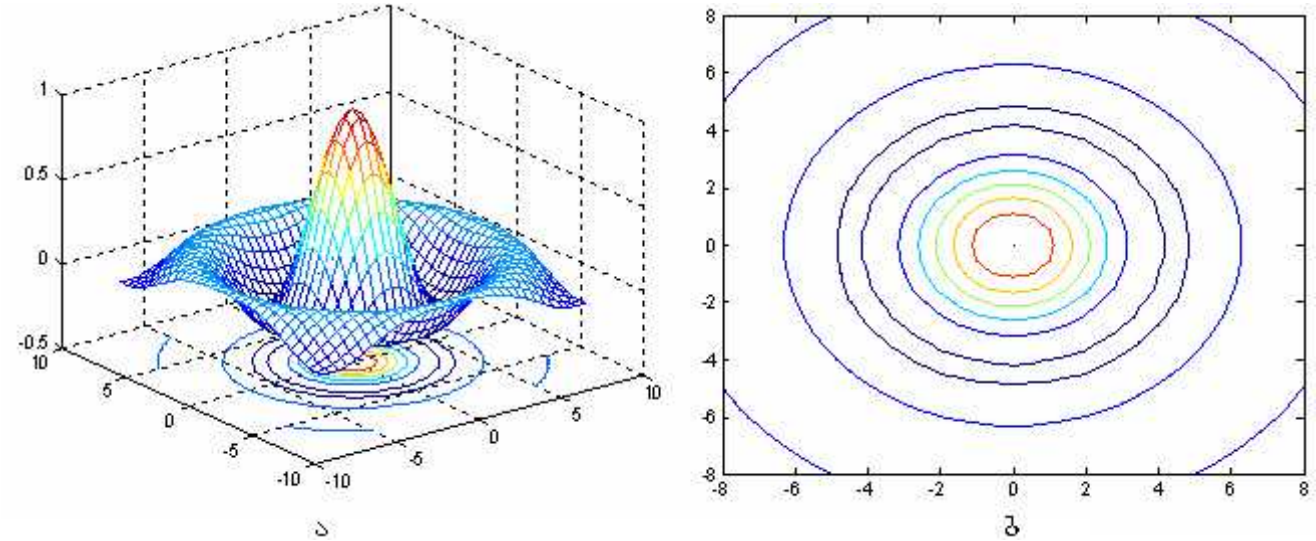
ნახ.3.25

meshc(X,Y,Z)–ით ერთ გრაფიკულ არეში აიგება ზედაპირის გრაფიკი და კონტურულიც (ზოგადად ერთ გრაფიკულ არეში რამდენიმე სამგანზომილებიანი გრაფიკის ასაგებად გამოიყენება hold on ბრძანება) – ნახ. 3.26,ა, ხოლო **contour(X,Y,Z)** –ით – **კონტურული** ანუ ერთნაირი დონეების წირების (იზოწირების) რუკა – ნახ.3.26,ბ.

იზოწირები წარმოადგენენ Z(X,Y) ზედაპირის გადაკვეთას Z=const სიბრტყეებთან. ამ წირეების პროექცია ერთ სიბრტყეზე გვაძლევს 2D გამოსახულებას. contour ფუნქციას გამოძახების რამდენიმე ვარიანტი აქვს: **contour(Z)**, **contour(Z,k)**, **contour(Z,v)**, **contour(X,Y,Z)**, **contour(X,Y,Z,k)**, **contour(X,Y,Z,v)** და სხვ.

გამოძახების ნებისმიერ ვარიანტში Z წარმოადგენს mxn ზომის ორგანზომილებიან აპლიკატების მასივს. X და Y შეიძლება იყოს შესაბამისად n და m ელემენტების მქონე ვექტორები ან იგივე ზომის მატრიცები, როგორც Z. X–ით და Y–ით განისაზღვრება ბადის კვანძების კოორდინატები, რომელთათვის მოცემულია აპლიკატების მნიშვნელობები. თუ ზედაპირის განსაზღვრის არე კვადრატულია და *x* და *y* ღერძების გასწვრივ ბიჯი ერთნაირია, X,Y არგუმენტების წყვილის მაგივრად შეიძლება მოცემული იყოს მხოლოდ ერთი მასივი. თუ

კოორდინატთა ბადე არ არის განსაზღვრული, ითვლება, რომ აბსცისის მნიშვნელობები იცვლება 1–დან n–მდე ბიჯით 1, ხოლო ორდინატის – 1–დან m–მდე ბიჯით 1.



ა ბ

ნახ.3.26

k სკალარით განისაზღვრება დონეების რაოდენობა ანუ z ღერძის გასწვრივ სიმაღლეების რაოდენობა, რომელთათვის აიგება იზოწირები. ამასთან, მიღებული იზოწირების რიცხვი შეიძლება k–ს ტოლი არ იყოს, რადგანაც შესაძლებელია შესაბამისი სიბრტყით ზედაპირის გადაკვეთით მიღებული იქნას რამდენიმე წირი (ნახ.3.27). თუ k–ს მნიშვნელობა არ არის მოცემული, MATLAB–ი ავტომატურად აირჩევს დონეების რაოდენობას [zmin,zmax] ინტერვალიდან. როგორც წესი, არჩეული მნიშვნელობა 20–ს არ აღემატება.

v ვექტორით განისაზღვრება დონეების მნიშვნელობები, რომელთათვის სასურველია იზოწირების აგება. თუ v ვექტორის რომელიმე კომპონენტი არ შედის [zmin,zmax] ინტერვალში, შესაბამისი იზოწირი არ იქნება აგებული მისი არარსებობის გამო.

გამოძახების ნებისმიერ ვარიანტში contour ფუნქციას შეუძლია 2 მასივის დაბრუნება: **[C h]=contour(...);**

აქ C მასივი წარმოადგენს მატრიცას, რომელშიც ჩაწერილია ინფორმაცია ყველა იზოწირზე: იზოწირის დონის მნიშვნელობა, მასში შემავალი წერტილების რაოდენობა თავისი კოორდინატებითურთ; h ვექტორი კი წარმოადგენს შექმნილ გრაფიკულ მასივებზე მიმთითებლების კრებულს. ეს ორი პარამეტრი საჭიროა გრაფიკზე იზოწირების დონეების მნიშვნელობების აღსანიშნავად. ამას ემსახურება **clabel(C,h)** ფუნქცია, რომლის საშუალებით ხდება კონტურულ გრაფიკზე დონეების მნიშვნელობების ავტომატური ჩაწერა.

გავაგრძელოთ ოპერატორთა მიმდევრობა (მე–3 ოპერატორი ემსახურება წირის სისქის განსაზღვრას):

```
>> figure
>> [C,h]=contour(X,Y,Z,5);
>> set(h,'LineWidth',3)
>> clabel(C,h)
```

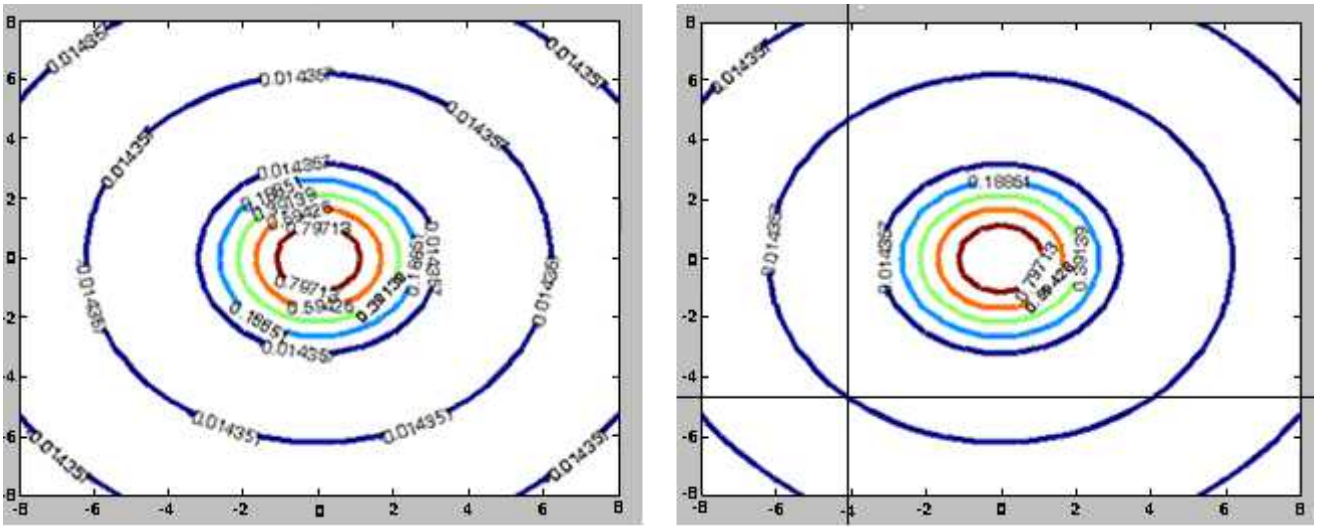
შედეგად მივიღებთ ნახ.3.27,ა.

როგორც ნახატიდან ჩანს, რიცხვების ავტომატური ჩაწერისას მიღებული გრაფიკი მაინცდამაინც ლამაზად არ გამოიყურება. განსაკუთრებით ეს ეხება არეებს იზოწირების დიდი კონცენტრაციით. შეიძლება დონეების მნიშვნელობების აღმნიშვნელი რიცხვები ჩავწეროთ არა ავტომატურად, არამედ „ხელით“. ამას ემსახურება **clabel(C,h,'manual')** ფუნქცია. ამ შემთხვევაში ეკრანზე დავინახავთ გრაფიკულ ფანჯარას კონტურული გრაფიკით, მაგრამ რიცხვების გარეშე. თაგვის მეშვეობით უნდა გადავაადგილოთ გადაჯვარედინებული წრფეების გადაკვეთის წერტილი იზოწირის ჩვენს მიერ შერჩეულ წერტილში და იქ დავაწკაპუნოთ. გამოჩნდება ციფრული წარწერა. უნდა აღვნიშნოთ, რომ „ხელით“ ჩატარებული ციფრების დატანის პროცედურაც ყოველთვის როდია გვაძლევს კარგ (ნახატის სილამაზის თვალსაზრისით) შედეგს.

გავაგრძელოთ ოპერატორების მწკრივი:

```
>> figure
>> [C,h]=contour(X,Y,Z,5);
>> set(h,'LineWidth',3)    '
>> clabel(C,h,'manual')
```

ნახ.3.27,ბ–ზე ნაჩვენებია რიცხვების დატანა „ხელით“ კონტურულ გრაფიკზე.



ა ბ

ნახ.3.27

ვექტორული ველის გრაფიკი მოკლე ისრების (ან კონუსების) სახით წარმოდგენილი ვექტორების ერთობლიობაა, რომელთა მიმართულება გვიჩვენებს სიჩქარის ვექტორის ზრდას, ხოლო სიგრძე სიჩქარის ვექტორის მოდულის მნიშვნელობის პროპორციულია. გრაფიკის ასეთი ფორმა ხშირად გამოიყენება ელექტრომაგნიტური, სითბური და გრავიტაციული ველების წარმოდგენისათვის. ვექტორული ველის გრაფიკის ასაგებად ვხმარობთ coneplot ფუნქციას, რომელსაც გამოძახების რამდენიმე ვარიანტი აქვს.

coneplot(X,Y,Z,U,V,W,Cx,Cy,Cz) ფუნქცია ასახავს U, V, W მასივებით მოცემულ ვექტორულ ველს კონუსების სახით. ვექტორული ველის პარამეტრებს MATLAB–ი გამოთვლის X, Y, Z მასივებით განსაზღვრულ კოორდინატებისათვის, ამიტომ ამ მასივების ზომა U, V, W

მასივების ზომას უნდა ემთხვეოდეს. Cx, Cy, Cz–ით განისაზღვრება წერტილები, რომლებშიც იქნება გამოსახული კონუსები, ამასთან, წინასწარი შეთანხმებით ამ წერტილებში კონუსების ასაგებად MATLAB–ი იყენებს წრფივ ინტერპოლაციას.

coneplot(U,V,W,Cx,Cy,Cz) ფუნქციაში გამოტოვებულია X,Y,Z არგუმენტები. ნაგულისხმევია, რომ [X,Y,Z]]=meshgrid(1:n,1:m,1:p), სადაც [m,n,p]=size(U).

coneplot(…, 'method') ფუნქციაში მისათითებელია სასურველი ინტერპოლაციის მეთოდი: 'linear', 'cubic' ან 'nearest'.

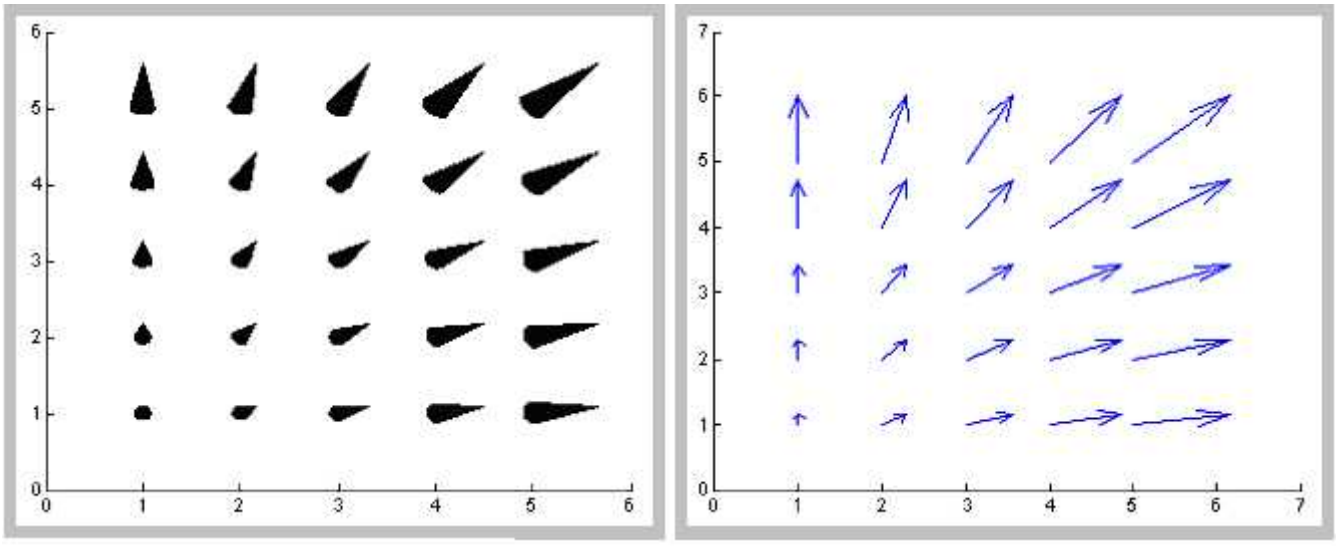
coneplot(X,Y,Z,U,V,W, 'nointerp')–ში არ ხდება კონუსების პოზიციების ინტერპოლაცია. კონუსების პოზიციები ამ შემთხვევაში განისაზღვრება X,Y,Z მასივებით, ხოლო მიმართულება – U, V, W–ით.

coneplot(…, 'color') ფუნქციით ხდება ვექტორული ველის გრაფიკის გაფერადება სამგანზომილებიანი color მასივის შესაბამისად, რომლის ზომა U–ს ზომას უნდა ემთხვეოდეს. გაფერადება შესაძლებელია მხოლოდ კონუსებით ველის ჩვენებისას და არა ისრებით.

coneplot(…,'quiver') ფუნქცია ააგებს ვექტორული ველის გრაფიკს, რომელშიც მიმართულება ისრებით ნაჩვენებია.

ქვემოთ მოყვანილი ბრძანებებით მიღებულია ნახ.3.28–ზე ნაჩვენები ვექტორული ველის გრაფიკები:

```
>> u=[0:2:10];v=[1 2 3 5 7 8];w=[4 5 8 2 6 12];[U,V,W]=meshgrid(u,v,w);
>> x=[0:5]; y=[0:5];z=[0:5:25];[Cx,Cy,Cz]=meshgrid(x,y,z);
>> coneplot(U,V,W,Cx,Cy,Cz);
>> figure
>> coneplot(U,V,W,Cx,Cy,Cz,'quiver');
```



ა ბ

ნახ.3.28

როგორც სამგანზომილებიანი გრაფიკების აგების ამ მოკლე მიმოხილვიდანაც ჩანს, MATLAB–ს გასაოცრად დიდი შესაძლებლობები აქვს ობიექტების ვიზუალიზაციის სფეროში.

3.7 ანიმაციური გრაფიკის აგება

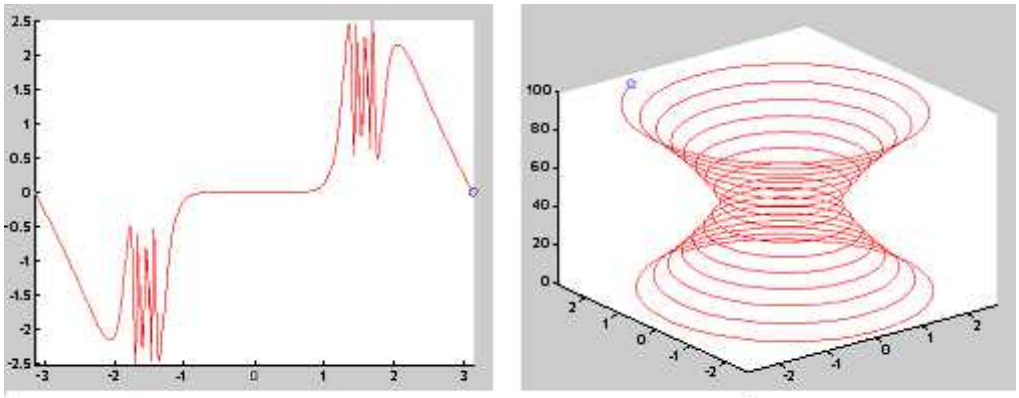
წერტილის მოძრაობის შესწავლისას ხშირად მნიშვნელოვანია არა მარტო მისი ტრაექტორიის შექმნა, არამედ აგების პროცესში ტრაექტორიის გასწვრივ წერტილის მოძრაობის დანახვა. ანიმაციური (დინამიური) გრაფიკების მისაღებად MATLAB–ს აქვს 2 ფუნქცია: comet და comet3. დინამიური გრაფიკების აგების პროცესში წერტილის აღმნიშვნელი რგოლი მოძრაობს და ტოვებს კვალს ხაზის სახით, რაც დამკვირვებელს მოძრავ კომეტას აგონებს.

comet(Y) მართკუთხა საკოორდინატო სისტემაში შექმნის დინამიურ გრაფიკს, რომელიც ასახავს Y ვექტორის ელემენტების დამოკიდებულებას მათ ინდექსებზე, ხოლო **comet(X,Y)** ფუნქციით აიგება ანიმაციური წირი, რომელიც აერთებს X და Y ვექტორებით მოცემულ წერტილებს ანუ ასახავს Y ვექტორის დამოკიდებულებას X–ზე.

ავაგოთ t = -pi:pi/200:pi; comet(t,tan(sin(t))-sin(tan(t))) ოპერატორებით განსაზღვრული გრაფიკი. მივიღებთ ნახ.3.29,ა–ს, ამასთან, გრაფიკის შექმნის პროცესს დავათვალიერებთ დინამიკაში. ამისათვის, რა თქმა უნდა, საჭიროა, რომ გრაფიკული ფანჯარა ეკრანზე სხვა ფანჯრების ზემოთ მდებარეობდეს. აღვნიშნოთ, რომ გრაფიკული ფანჯრის ზომის შეცვლისას ან მისი მინიმიზაციისა და მომდევნო აღდგენის შემდეგ მოძრაობის ტრაექტორია ქრება.

comet3(Z) ფუნქციით მიიღება Z ვექტორის ამსახველი ანიმაციური 3D გრაფიკი, ხოლო **comet3(X,Y,Z)**–ით – ანიმაციური 3D გრაფიკი, რომელშიც რგოლი მიმდევრობით გაივლის (X(i),Y(i),Z(i)) წერტილებს. ქვემოთ მოყვანილი ბრძანებებით მიღებულია მოძრავი წერტილის სამგანზომილებიანი ტრაექტორია (ნახ.3.29,ბ), რომელიც შეესაბამება შემდეგ პარამეტრულ ფუნქციას: $x = e^{|t-50|/50}\sin(t)$, $y = e^{|t-50|/50}\cos(t)$.

```
>> t=[0:0.05:100]; x=exp(abs(t-50)/50).*sin(t);
>>  y=exp(abs(t-50)/50).*cos(t); comet3(x,y,t)
```



ა ბ

ნახ.3.29

რგოლის („კომეტის“ თავის) მოძრაობის სიჩქარე დამოკიდებულია დროის ვექტორის ბიჯზე, ხოლო „კომეტის“ კუდის სიგრძე შეიძლება შევცვალოთ comet და comet3 ფუნქციებში დამატებითი p პარამეტრის ($0 \le p < 1$) შეტანით, რის შედეგადაც „კომეტის“ სიგრძე p·length(Y)–ის ტოლი გახდება. წინასწარი შეთანხმებით p=0,1.

თავი IV გამოთვლითი პროცესების დაპროგრამება

4.1 სკრიპტები და ფუნქციები

აქამდე ვიხილავდით ბრძანებათა ფანჯარაში მუშაობას დიალოგურ რეჟიმში, როდესაც მომხმარებელი კრეფს სასურველ ოპერატორებს და Enter ღილაკის დაჭერის შემდეგ სისტემა ასრულებს შესაბამის მოქმედებებს. აკრეფილი ბრძანებები ინახება Command History ფანჯარაში, საიდანაც საჭიროებისამებრ შესაძლებელია ადრე აკრეფილი ოპერატორების ბრძანებათა ფანჯარაში შეტანა მათი ხელმეორე შესრულებისათვის ან მსგავსი ბრძანებების ფორმირებისათვის. ასეთი მიდგომა კარგია, როცა გვჭირდება ნაბიჯ–ნაბიჯ ერთჯერადი გამოთვლების ჩატარება.

ბრძანებათა ფანჯარაში შესრულებული მოქმედებების შენახვა შესაძლებელია ტექსტურ ფაილში, რომლის შინაარსი MATLAB–ში გადატანის შემდეგ შესრულდება ავტომატურად, ოპერატორების ხელმეორე აკრეფისა და ყოველი ოპერატორის შემდეგ Enter კლავიშის დაჭერის გარეშე. ასეთი სახის პროგრამებს ჰქვია **სკრიპტები (script)**. სკრიპტების გარდა, არსებობს პროგრამების მეორე, უფრო მოხერხებული ნაირსახეობაც – **ფუნქციები (function)**. სკრიპტი ფუნქციასთან შედარებით უფრო მარტივია, რადგან მას არ გააჩნია შემავალი და გამომავალი პარამეტრები.

პროგრამა შეიძლება მომზადდეს ტექსტურ ფაილში და შენახული იქნას დისკზე m გაფართოებით. მაგრამ MATLAB–ს გააჩნია საკუთარი M–ფაილის რედაქტორი, რომელიც გამოიძახება **File → New → M-File** ბრძანებით. რედაქტორის ფანჯარაში აკრეფილი პროგრამის დასამახსოვრებლად ვსარგებლობთ რედაქტორის მენიუს **File → Save As...** ბრძანებით. აღსანიშნავია, რომ M–ფაილის შენახვისას ფუნქციის სახელი უნდა იქნას აღებული ფაილის სახელად.

M–ფუნქციის სტრუქტურა ხასიათდება შემდეგი კომპონენტებით:

1. ფუნქციის ჩაწერა იწყება **function** ოპერატორით, რომელსაც აქვს შემდეგი სინტაქსი:

function ცვლადის_სახელი = ფუნქციის_სახელი (შემავალი_პარამეტრების_სია)

თუ გამომავალი პარამეტრების რაოდენობა ერთზე მეტია, მათი ჩამონათვალი უნდა იყოს მოყვანილი კვადრატულ ფრჩხილებში:

function [გამომავალი_პარამეტრების_სია] = ფუნქციის_სახელი (შემავალი_პარამეტრების_სია)

function ოპერატორის შემავალი და გამომავალი პარამეტერები ფუნქციის ფორმალურ პარამეტრებს წარმოადგენს.

2. სასურველია, რომ function ოპერატორის წინ ან მის შემდეგ მოთავსებული იქნას კომენტარი, რომელიც განსაზღვრავს ფუნქციის დანიშნულებას და ინფორმაციას შემავალი და გამომავალი ცვლადების შესახებ. **help ფუნქციის_სახელი** ბრძანებით კომენტარის სტრიქონები

(პირველ ცარიელ სტრიქონამდე ან function ოპერატორამდე – თუ კომენტარი მოთავსებულია function ოპერატორის წინ, ხოლო თუ კომენტარი მოთავსებულია function ოპერატორის შემდეგ – პირველ ოპერატორამდე ფუნქციის ტანში) გამოდის ეკრანზე;

3. ფუნქციის ტანი – ეს არის MATLAB ენაზე დაწერილი პროგრამა, რომელიც ახორციელებს გამოთვლებს და ანიჭებს გამომავალ ცვლადებს მნიშვნელობებს.

M–ფაილში ჩაწერილი ბრძანებების შესრულებაზე გაშვება ხორციელდება:

- სკრიპტისათვის რედაქტორის ფანჯრიდან (კლავიატურიდან F5 ფუნქციონალური ღილაკის გამოყენებით ან მენიუდან **Debug** ì **Run** ბრძანებით) ან, თუ ფაილი დამახსოვრებულია, ბრძანებათა ფანჯრიდან ბრძანებათა სტრიქონში მისი სახელის აკრეფით და Enter კლავიშის დაჭერით;
- ფუნქციისათვის ბრძანებათა ფანჯრიდან ბრძანებათა სტრიქონში მისი სახელის აკრეფით და Enter კლავიშის დაჭერით, ამასთან, წინასწარ უნდა განსაზღვრული იყოს ფრჩხილებში მოთავსებული ფუნქციის ფაქტიური შემავალი პარამეტრები. ფაქტიური პარამეტრები აუცილებლად უნდა შეესაბამებოდეს ფუნქციის ფორმალურ პარამეტრებს როგორც რაოდენობის, ასევე ტიპისა და ჩამონათვალში მათი მდებარეობის მიხედვით. მიმართვისას ფაქტიური პარამეტრები ცვლის შესაბამის ფორმალურ პარამეტრებს. მათთვის განხორციელდება ფუნქციის ტანში აღწერილი ოპერატორები, რის შემდეგაც მოხდება პროგრამა–ფუნქციიდან გამოსვლა და პროგრამული მოდულის შესრულება გაგრძელდება მიმართვის შემდეგ მდგომი ოპერატორიდან.

M–ფაილში სტრიქონები ავტომატურად ინომრება და პროგრამის გაშვებისას შედეგი გამოდის ბრძანებათა ფანჯარაში. თუ სტრიქონში შეტანილი ბრძანებების ბოლოს დავსვამთ ; ნიშანს, პროგრამის შესრულებაზე გაშვებისას შუალედური შედეგები ბრძანებათა ფანჯარაში არ გამოჩნდება. ასეთი მიდგომით შესაძლებელია დაწერილი პროგრამის მხოლოდ საბოლოო შედეგების ჩვენება.

M–ფაილში სტრიქონების დანომვრა გვაძლევს დაშვებული სინტაქსური და სემანტიკური შეცდომების იდენტიფიცირების საშუალებას. ამასთან, ბრძანებების ბოლოს ; ნიშნის დასმა არ თრგუნავს შეცდომის შესახებ შეტყობინების გამოტანას. ბრძანებათა ფანჯარაში წითლად გამოტანილ შეცდომის შესახებ შეტყობინებაზე მაუსის დაწკაპუნებით გადავყავართ M–ფაილში შესაბამის სტრიქონზე.

სკრიპტების მთავარი თავისებურება მდგომარეობს იმაში, რომ ისინი მუშაობენ მხოლოდ სამუშაო არის (Workspace) ცვლადებთან. რამდენიმე სკრიპტი შეიძლება გავუშვათ შესრულებაზე ერთიმეორის მიყოლებით, ამასთან, მომდევნო სკრიპტი სარგებლობს წინა სკრიპტის მონაცემებით საერთო სამუშაო არის მეშვეობით ან ფაილებში შუალედური შედეგების დამახსოვრებით. მაგრამ უნდა აღინიშნოს, რომ ლოკალური ცვლადების არარსებობა სკრიპტებში აფერხებს ამისთანა რეჟიმში მუშაობას საერთო ცვლადების შეთანხმებული გამოყენების სიძნელეების გამო. ამ მხრივ გაცილებით უფრო მოხერხებულია ფუნქციები. ფუნქციებს ვაწვდით საწყის მონაცემებს შემავალი პარამეტრების სახით, ხოლო შედეგს ვიღებთ გამომავალ პარამეტრებში.

სრიპტებისაგან განსხვავებით, ფუნქციებს ახასიათებს ლოკალური ცვლადების აპარატი. გლობალური ცვლადების და შემავალი და გამომავალი პარამეტრების გარდა, ყველა ცვლადი, რომლებიც გვხვდება ფუნქციის ბრძანებებში, ითვლება ლოკალურად ანუ მათი მოქმედება ლიმიტირებულია კონკრეტული ფუნქციის ფარგლებით და არც ერთი სკრიპტი ან სხვა ფუნქცია მათ ვერ გამოიყენებს. ცხადია, ლოკალური ცვლადები Workspace–შიც არ არიან განსაზღვრული. საჭიროებისამებრ შესაძლებელია ზოგი ცვლადი გლობალურ ცვლადად (ოპერატორი **global**) გამოცხადდეს, თუმცა გლობალური ცვლადების გამოყენება ერთობ სარისკოა. მაგალითად, ვთქვათ, მომხმარებელმა გამოაცხადა რომელიმე ცვლადი გლობალურ ცვლადად. თუ სხვა M–ფაილში არსებობს ასეთივე სახელის მქონე გლობალური ცვლადი, მისი მნიშვნელობა ამით შეიცვლება. ამისთანა შეცდომის გამოვლინება ძალზე რთულია. სწორედ ამიტომ გლობალური ცვლადების გამოყენება დასაშვებია მხოლოდ განსაკუთრებულ შემთხვევებში. მათი გამოყენებისას საჭიროა დავრწმუნდეთ სახელის უნიკალურობაში.

კიდევ ერთი მნიშვნელოვანი განსხვავება მდგომარეობს იმაში, რომ M–სკრიპტის სახელი, M–ფუნქციის სახელისაგან განსხვავებით, არ შეიძლება გამოყენებული იქნას არითმეტიკული გამოსახულების ოპერანდად და ფუნქციების არგუმენტად.

M–ფაილი შეიძლება შეიცავდეს ერთზე მეტ ფუნქციას. მათგან პირველს ჰქვია ძირითადი. მხოლოდ ამ ფუნქციის გამოძახება ხდება გარედან. ყველა სხვა ფუნქციას ჰქვია ქვეფუნქცია – ეს არის შიდა ფუნქცია, რომლის მოქმედების არე ლიმიტირებულია მოცემული M–ფაილით ანუ ქვეფუნქციის გამოძახება შესაძლებელია მხოლოდ ძირითადი ფუნქციიდან და მოცემული M–ფაილის სხვა ქვეფუნქციებიდან.

M–ფაილის შესრულება ხდება ინტერპრეტაციის რეჟიმში, რაც საგრძნობლად ამცირებს ამოცანის ამოხსნის სისწრაფეს. სისწრაფის გადიდების მიზნით MATLAB–ი გარდაქმნის საწყის ფუნქციას ე.წ. ფსევდოკოდად, რათა ფუნქციის ხელმეორე გამოძახებისას სისტემას დასჭირდეს მინიმალური მუშაობის შესრულება შეცდომების გამოსავლენად. ფსევდოკოდი რჩება მეხსიერებაში მუშაობის სეანსის ბოლომდე ან ოპერატიული მეხსიერების გასუფთავებამდე, რომელიც ხორციელდება ბრძანებებით:

clear ფუნქციის_სახელი – კონკრეტული ფსევდოკოდის გაუქმება,

clear functions – ყველა ფუნქციის ფსევდოკოდის გაუქმება,

clear all – ყველა ფუნქციის და ცვლადის გაუქმება.

შესაძლებელია M–სკრიპტის ან M–ფუნქციის ფსევდოკოდის შენახვა მუშაობის მომდევნო სეანსებში გამოსაყენებლად. ამისთვის ვსარგებლობთ ბრძანებით **pcode ფუნქციის_სახელი**, რომლის შედეგია იმავე სახელის მქონე ორობითი ფაილის შექმნა p გაფართოებით. საწყისი ტექსტის p–ფაილით ჩანაცვლებით ხდება ფაილის თავდაპირველი ჩატვირთვისას პროგრამის სინტაქსური ანალიზის არიდება და აგრეთვე ალგორითმის საწყისი ტექსტის გასაიდუმლოება.

4.2 დაპროგრამება MATLAB–ში. ოპერატორთა სინტაქსი

MATLAB–ში გამოიყენება გამოთვლითი პროცესების დაპროგრამებასთან დაკავშირებული 9 ოპერატორი, რომლებშიც გამოყენებულია 14 საკვანძო სიტყვა (სიტყვა–გასაღები). რაც შეეხება მონაცემთა აღწერასა და შეტანას, ისინი საგრძნობლად გამარტივებულია დაპროგრამების ენებთან შედარებით. MATLAB პროგრამაში ცვლადის ტიპი არ ცხადდება. ის განისაზღვრება იმ გამოსახულებით, რომლის მნიშვნელობაც ენიჭება ცვლადს ანუ ახალი ცვლადი სისტემის მიერ აღიქმება როგორც განსაზღვრული კლასის მასივის სახელი (**single –** ერთმაგი სიზუსტის რიცხვითი მასივი, **double –** ორმაგი სიზუსტის რიცხვითი მასივი, **logic –** ლოგიკური მასივი, **char –** სიმბოლოებისგან შემდგარი სტრიქონული მასივი და ა.შ.), რომლის ზომები მისი შევსებისას განისაზღვრება.

4.2.1 მინიჭების ოპერატორი

მინიჭების ოპერატორს აქვს სახე:

ცვლადის_სახელი = გამოსახულება

მინიჭების ოპერატორი ითვლის გამოსახულების მნიშვნელობას და ანიჭებს შედეგს მითითებულ ცვლადს. მაგალითისათვის შევქმნათ პროგრამა–ფუნქცია, რომელიც წამებში მოცემულ დროის მონაკვეთს გადაიყვანს საათებში, წუთებში და წამებში:

```
function [saaTi,cuTi,cami]=dro(camebi)
%   camebSi mocemuli drois monakveTis
% saaTebSi, cuTebSi da camebSi gadayvana
saaTi=floor(camebi/3600);
a=camebi-saaTi*3600;
cuTi=floor(a/60);
cami=a-cuTi*60;
```

თუ, ვთქვათ, ბრძანებათა ფანჯრიდან შემდეგნაირად მივმართავთ ამ ფუნქციას:

```
>> [h m s]=dro(36650)
```

შედეგად მივიღებთ:

```
h =
  10
m =
  10
s =
  50
```

4.2.2 განშტოებადი ალგორითმების დაპროგრამება

პირობის ოპერატორს, რომელიც განშტოების ოპერათორთა ჯგუფს ეკუთვნის, შემდეგი სტრუქტურა აქვს:

```
if   პირობა_1
        ოპერატორი_1
[elseif პირობა_2
        ოპერატორი_2
elseif პირობა_3
        ოპერატორი_3
. . . . . .
else
        ოპერატორი_n]
end
```

აქ პირიბა_1, პირიბა_2,... – ლოგიკური გამოსახულებებია, ოპერატორი_1, ოპერატორი_2,... – ცალკეული ოპერატორებია ან ოპერატორთა ჯგუფებია. თუ პირობა_1 ჭეშმარიტია, სრულდება ოპერატორი_1, თუ მცდარია, მოწმდება პირობა_2, რომლის ჭეშმარიტების შემთხვევაში სრულდება ოპერატორი_2 და ა.შ. თუ ყველა აღნიშნული პირობა მცდარია, მაშინ სრულდება else–სა და end–ს შორის განლაგებული ოპერატორი_n. პირობითი ოპერატორი შეიძლება ნებისმიერი რაოდენობის elseif ქვეოპერატორს შეიცავდეს. დასაშვებია პირობითი ოპერატორის შემოკლებული სახეებიც:

```
if პირობა_1
    ოპერატორი_1
end
```

ან

```
if პირობა_1
    ოპერატორი_1
else
    ოპერატორი_2
end
```

პირველ შემთხვევაში თუ პირობა ჭეშმარიტია, სრულდება ოპერატორი_1, თუ მცდარია – end–ის შემდეგ მდგარი ოპერატორი.

მაგალითისათვის მოვიყვანოთ პროგრამა–ფუნქცია, რომელიც გამოთვლის ვექტორის ელემენტთა საშუალო მნიშვნელობას და მედიანას. მედიანის საპოვნელად გამოყენებულია if ოპერატორი.

```
function[avg,med]=statistics(A)
%   funqcia poulobs A veqtoris elementebis saSualo
% mniSvnelobas da medianas qvefunqciebis gamoyenebiT
n=length(A);
avg=saSualo(A,n);
med=mediana(A,n);
%
function s=saSualo(X,n)
% es qvefunqcia gamoTvlis saSualo mniSvnelobas
s=sum(X)/n;
%
```

```
function m=mediana(X,n)
%  es qvefunqcia gamoTvlis medianas
Y=sort(X);
if rem(n,2)==1
    m = Y((n+1)/2);
else
    m=(Y(n/2)+Y(n/2+1))/2;
end
```

განვიხილოთ პირობების (ლოგიკური გამოსახულებების) შედგენის წესები.

MATLAB–ს აქვს **შედარების ექვსი ოპერატორი**: < (ნაკლებია), <= (ნაკლებია ან ტოლი), > (მეტია), >= (მეტია ან ტოლი), == (ტოლია), ~= (არ უდრის).

მასივი და მასივური გამოსახულება შეიძლება შეგვხვდეს ამ ოპერატორთა ორივე მხარეს, მაგრამ შეიძლება შევადაროთ მხოლოდ ერთნაირი ზომის მასივები. შედეგად მიღებული იგივე ზომის მასივის შესაბამისი ელემენტი უდრის 1–ს (True – ჭეშმარიტება), თუ შერადების შედეგი ჭეშმარიტია, და 0–ს (False – მცდარობა), თუ იგი მცდარია. გამოსახულებას, რომელიც შეიცავს შედარების ოპერატორს, **ლოგიკური გამოსახულება** ეწოდება, რადგან შედეგად მიიღება მასივი, რომლის ელემენტების მნიშვნელობებია 1 და 0. მაგ., ლოგიკური გამოსახულება a<b, თუ a და b, ვთქვათ, ვექტორებია: a = [2 4 6] , b = [3 5 1], მოგვცემს შედეგად [1 1 0] ვექტორს, ხოლო a~=b – [1 1 1]. შედარების ოპერაციებში ერთ–ერთ არგუმენტად შესაძლებელია რიცხვის გამოყენება. ამ დროს ხდება მასივის ყოველი ელემენტის შედარება ამ რიცხვთან და შედეგად მიიღება იგივე ზომის მასივი.

ორი ლოგიკური გამოსახულება შეგვიძლია გავაერთიანოთ ლოგიკური ოპერატორებით, რომლებიც აღინიშნება სიმბოლოებით ან დარეზერვირებული სიტყვებით: **~, not** (ლოგიკური უარყოფა), **&**, **and** (ლოგიკური გამრავლება), |, **or** (ლოგიკური შეკრება), **xor** (გამომრიცხავი or). ოპერაციის სიმბოლო იწერება ოპერანდებს შორის, ხოლო დარეზერვებული სიტყვების გამოყენებისას ოპერანდები უნდა მოვათავსოთ ფრჩხილებში. ცხრილი შეიცავს A და B ლოგიკურ გამოსახულებებს შორის ლოგიკურ ოპერატორთა ყველა შესაძლო კომბინაციას:

A	B	~A	&	|	xor
F	F	T	F	F	F
F	T	T	F	T	T
T	F	F	F	T	T
T	T	F	T	T	F

MATLAB–ში შესაძლებელია ერთ გამოსახულებაში გამოვიყენოთ როგორც არითმეტიკული, ისე ლოგიკური ცვლადებიც. ოპერაციათა იერარქია მაღლიდან დაბლისაკენ ასეთია: 1) ლოგიკური უარყოფა; 2) ტრანსპონირება, ახარისხება, რიცხვის წინ + ან – ნიშნის გათვალისწინება; 3) გამრავლება და გაყოფა; 4) შეკრება და გამოკლება; 5) შედარების ოპერაციები; 6) and; 7) or. ერთი პუნქტის ოპერაციებს აქვს ერთნაირი პრიორიტეტი, ისინი თანმიმდევრობით სრულდება. ფრჩხილებით შესაძლებელია ოპერაციათა თანამიმდევრობის შეცვლა. მაგ., შემდეგ ლოგიკურ გამოსახულებაში: ~(b == c | b == 5.5) პირველ რიგში შესრულდება b == c და b == 5.5, შემდეგ | ოპერაცია და ბოლოს ლოგიკური უარყოფა.

დავუშვათ, b = 3, c = 5. შედეგად მივიღებთ ჭეშმარიტებას. დავუშვათ, ამ გამოსახულებაში ფრჩხილები არ გვაქვს: ~b == c | b == 5.5. ამ შემთხვევაში ჯერ შესრულდება ლოგიკური უარყოფა ~b (ამ დროს ანუ როდესაც b თვითონ რიცხვს წარმოადგენს, MATLAB–ი ყველა არანულოვან მნიშვნელობას შეაფასებს როგორც ჭეშმარიტს, ხოლო ნულოვანი მნიშვნელობა შეფასდება როგორც მცდარი), შემდეგ შესრულდება ~b ==c და b == 5.5 და ბოლოს ლოგიკური or. b და c მოცემულ მნიშვნელობათათვის ამ გამოსახულების მნიშვნელობა მცდარია. აღვნიშნოთ აგრეთვე, რომ პირველ რიგში სრულდება and, or და not ოპერაციები (რადგან ისინი MATLAB სისტემის ფუნქციებს წარმოადგენენ), თუ ისინი გამოყენებულია &, | და ~–ის ნაცვლად, ასე რომ and(A,B) + C და A&B + C ჩანაწერი არ არის ეკვივალენტური.

MATLAB–ს აქვს ლოგიკური ფუნქციები, რომლებიც ხშირად გამოიყენება if ოპერატორში. მოვიყვანოთ ზოგი მათგანი:

any(X) ფუნქცია X მატრიცის ყოველი სვეტისათვის გვაძლევს 1–ს, თუ ამ სვეტის რომელიმე ელემენტი მაინც არის არანულოვანი, სხვა შემთხვევაში გვაძლევს 0–ს;

all (X) X მატრიცის ყოველი სვეტისათვის გვაძლევს 1–ს, თუ ამ სვეტის ყველა ელემენტი არანულოვანია, სხვა შემთხვევაში გვაძლევს 0–ს;

find (X) გვაძლევს ვექტორს, რომელიც შეიცავს X ვექტორის არანულოვანი ელემენტების ინდექსებს. თუ X მატრიცაა, მაშინ ინდექსები შეირჩევა X(:) ვექტორიდან, რომელიც წარმოადგენს X მატრიცის ერთმანეთს მიყოლებული სვეტების ელემენტებისაგან შემდგარ ერთ გრძელ ვექტორს. მაგ., თუ A=[3 -4 0 0; 7 5 0 8], find(A) მოგვცემს შემდეგი ელემენტებისაგან შემდგარ ვექტორ–სვეტს: 1, 2, 3, 4, 8;

exist ('A') გვაძლევს 1–ს, თუ მოცემულ სამუშაო სივრცეში არსებობს A ცვლადი; 2–ს, თუ მოცემულ სამუშაო სივცეში არსებობს A ცვლადი ან A.m ფაილი; 0–ს, თუ ასეთი არც ცვლადი და არც .m ფაილი არ არსებობს;

isnan(X) გვაძლევს მატრიცას, რომლის ელემენტები 1–ის ტოლია, როცა X მატრიცის შესაბამისი ელემენტები წარმოადგენს განუსაზღვრელობას, სხვა შემთხვევაში ელემენტების მნიშვნელობები 0–ის ტოლია;

finite(X) გვაძლევს მატრიცას, რომლის ელემენტები უდრის 1–ს, როცა X მატრიცის შესაბამისი ელემენტები სასრული რიცხვებია, სხვა შემთხვევაში – 0–ს;

isempty (X) გვაძლევს 1–ს, თუ X არის ცარიელი მატრიცა, თუ არა, გვაძლევს 0–ს;

ischar (X) გვაძლევს 1–ს, თუ X სტრიქონული ტიპის მასივია, 0–ს – თუ ეს ასე არაა;

strcmp(y1,y2) ერთმანეთს ადარებს y1 და y2 სტრიქონს და გვაძლევს 1–ს, თუ ისინი იდენტურია, 0–ს – თუ განსხვავებულია.

დავუშვათ, A სამსტრიქონიანი და სამსვეტიანი მატრიცაა. განვიხილოთ შემდეგი ბრძანება:

```
if all (A) == 0
    disp ('A contains all zeros')
end
```

ტექსტი A contains all zeros დაიბეჭდება მხოლოდ იმ შემთხვევაში, როცა ყველა ელემენტის მნიშვნელობა არის 0–ის ტოლი.

განშტოების ოპერათორთა ჯგუფს ეკუთვნის აგრეთვე ***გადამრთველი ოპერატორი***, რომელსაც შემდეგი სტრუქტურა აქვს:

switch მმართველი გამოსახულება
case მნიშვნელობა_1
ოპერატორი_1
case მნიშვნელობა_2
ოპერატორი_2
. . . . . .
[othewise
ოპერატორი_n]
end

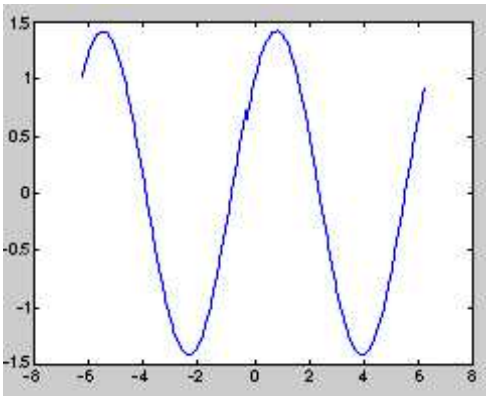
გადართვა ხდება მმართველი გამოსახულების მნიშვნელობის თანახმად. თუ ის ემთხვევა მნიშვნელობა_1 სიდიდეს, მაშინ სრულდება ჯგუფი ოპერატორი_1, თუ ის ემთხვევა მნიშვნელობა_2 სიდიდეს, მაშინ სრულდება ოპერატორი_2, ... თუ გამოსახულება არ ემთხვევა არც ერთ ჩამოთვლილ მნიშვნელობას, მაშინ სრულდება ოპერატორი, რომელიც განლაგებულია otherwise–სა და end–ს შორის. ერთ განშტოებაში შეიძლება ცვლადის რამდენიმე მნიშვნელობის ჩაწერა, მაგ.,

case(მნიშვნელობა_1, მნიშვნელობა_2, მნიშვნელობა_3, ...).

switch ოპერატორის ნაცვლად ყოველთვის შეიძლება გამოვიყენოთ if ოპერატორის ესა თუ ის ფორმა, მაგრამ, როცა ალგორითმი მრავალვარიანტულია და არსებობს შესაბამისობა რაღაცა ცვლადის დისკრეტულ მნიშვნელობებსა და შემდგომ მოქმედებებს შორის, switch ოპერატორის მეშვეობით შეიძლება შევადგინოთ უფრო თვალსაჩინო პროგრამული სტრუქტურები.

ქვემოთ, მარცხნივ, მოყვანილია პროგრამა, რომელიც ახორციელებს ამა თუ იმ გრაფიკის აგებას მმართველი ცვლადის მნიშვნელობის შესაბამისად, ხოლო მარჯვნივ – ბრძანებათა ფანჯრიდან x = -6.28:0.1:6.28; grafiki(x,3) ბრძანებებით მიღებული გრაფიკი.

```
function grafiki(x,k)
switch k
   case 1
       plot(x,sin(x));
   case 2
       plot(x,cos(x));
   case 3
       plot(x, sin(x)+cos(x));
end
```



ნახ.4.1

4.2.3 ციკლური ალგორითმების დაპროგრამება

მსგავსი განმეორებადი მოქმედებების ჩასატარებლად გამოიყენება ციკლის ოპერატორები **for** და **while**. for ციკლი არითმეტიკული პროგრესიის ტიპის ციკლია. მასში წინასწარ ცნობილია მოქმედებების გამეორების რიცხვი, ხოლო while ციკლში ცნობილია ციკლის გაგრძელების პირობა.

for ტიპის ციკლის ოპერატორს შემდეგი სტრუქტურა აქვს:

for ციკლის_პარამეტრი = e1:[e2:]e3
ოპერატორები
end

ციკლის პარამეტრი იცვლება e1–დან ბიჯით e2 საბოლოო e3 მნიშვნელობამდე (უფრო სწორედ, მანამ ციკლის პარამეტრის მნიშვნელობა არ გახდება e3–ზე მეტი, თუ e2 დადებითია, და e3–ზე ნაკლები, თუ e2 უარყოფითია). თუ ციკლის პარამეტრის ცვლილების ბიჯი 1–ის ტოლია, e2 შეიძლება საერთოდ არ მივუთითოთ.

შევადგინოთ ფაქტორიალის გამოთვლის პროგრამა–სკრიპტი და პროგრამა-ფუნქცია:

სკრიპტის შემთხვევა
(5!–ის გამოთვლა):

```
a=1;
for i=1:5
    a=a*i;
end
a
```

ფუნქციის შემთხვევა:

```
function y=factn(n)
k=1;
for i=1:n
    k=k*i;
end
y=k;
```

ცხადია, ნებისმიერი რიცხვის ფაქტორიალის გამოსათვლელად ფუნქციის შემთხვევაში საკმარისია ბრძანებათა ფანჯრის მიმდინარე სტრიქონში ავკრიფოთ ფუნქციის სახელი და ფრჩხილებში მივუთითოთ ეს რიცხვი, მაგ.,

```
>> k=factn(5)
```

პასუხად მივიღებთ

```
k=
   120
```

დასაშვებია, ციკლები ერთმანეთში იყოს ჩადგმული. ცხადია, ამ შემთხვევაში ციკლის პარამეტრად სხვადასხვა ცვლადი უნდა იყოს აღებული. მაგ., დავწეროთ პროგრამა, რომელიც ახორციელებს მასივის ელემენტების მოწესრიგებას მნიშვნელობათა კლებადობის მიხედვით:

```
% veqtoris elementebis mowesrigeba klebadobis mixedviT
function A=mowesrigeba(A)
n=length(A);
for i=1:n-1
    k=n+1-i;
    % veZebT minimalur elements k elementTa Soris da mis nomers
    minim=A(1); nom=1;
    for j=1:k
       if A(j)<minim
```

```
            minim=A(j);
            nom=j;
        end
    end
  %  minimaluri da k-uri elementebis adgilebis Secvla
  A(nom)=A(k); A(k)=minim;
end
```

პროგრამის მუშაობის სისწორის შესამოწმებლად ავკრიფოთ ბრძანებათა ფანჯარაში

```
>> z=[1 5 3 8 9 4 8 2 1 5];  z=mowesrigeba(z)
```

პასუხად მივიღებთ:

```
z =
    9     8     8     5     5     4     3     2     1     1
```

while ტიპის ციკლის (ციკლის პირობით) ოპერატორს შემდეგი სახე აქვს:

while პირობა
ოპერატორები
end

ციკლი while სრულდება მანამ, სანამ სრულდება პირობა. for ციკლისაგან განსხვავებით, მას არა აქვს ციკლის პარამეტრი. ამიტომ მის ტანში საჭიროა გათვალისწინებული იქნას პირობაში შესული რომელიმე ცვლადის ცვლილება, რათა პირობა ციკლში ყოველ შესვლისას განსხვავებული იყოს. მაგალითისათვის დავწეროთ პროგრამა–ფუნქცია, რომელიც მოცემული სიზუსტით (ანუ პროგრამა გააგრძელებს მუშაობას მანამ, სანამ მორიგი შესაკრები მოცემულ რიცხვზე მეტია) გამოთვლის შემდეგი მწკრივის მნიშვნელობას:

$$1+\frac{1}{1\times 2}+\frac{1}{2\times 3}+\frac{1}{3\times 4}+\ldots$$

```
function s_jami=sss(sizuste)
s_jami=1;
i=1;
while 1/(i*(i+1))>sizuste
    s_jami=s_jami+1/(i*(i+1));
    i=i+1;
end
```

ავკრიფოთ ბრძანებათა ფანჯარაში

```
>>  p=sss(0.0001);
```

მივიღებთ

```
p =
        1.9900
```

4.2.4 პროგრამების მუშაობის სისწორის კონტროლი

სხვა დაპროგრამების ენებისაგან განსხვავებით, ზოგი არაკორექტული მათემატიკური ოპერაცია არ იწვევს MATLAB პროგრამის მუშაობის დასრულებას (მაგ., ნულზე გაყოფის დროს მიიღება უსასრულობა **Inf**, ნულის ნულზე გაყოფისას **Nan**), თუმცა არსებობს შეცდომები, რომლებიც იწვევს მუშაობის დასრულებას. ასეთი შეცდომაა, მაგალითად, არარსებულ ფაილთან მიმართვა. მწყვეტავი სიტუაციის დამუშავება ხდება **try ... catch** ოპერატორით, რომელსაც შემდეგი სახე აქვს:

try

ოპერატორი_1

catch

ოპერატორი_2

end

ჩვეულებრივად სრულდება მხოლოდ try და catch საკვანძო სიტყვათა შორის განლაგებული ოპერატორები. შეცდომის შემთხვევაში შესრულდება ოპერატორები catch–სა და end–ს შორის, ხოლო შეცდომის შესახებ ინფორმაცია შენახული იქნება **lasterror**–ში. შემდეგ მაგალითში ფაილის არარსებობის შემთხვევაში გამოვა შესაბამისი შეტყობინება, რის შემდეგ პროგრამა გააგრძელებს მუშაობას.

```
try X=load('prov.dat');
    pie(X)
catch('faili prov.dat. ar arsebobs')
end
A= 1:10; B=21:30;
C=A.^2+B.^2
```

მმართველი კონსტრუქციების for, while, switch, try–catch შეწყვეტა ხდება **break** ოპერატორით.

მაგალითი: შევადგინოთ პროგრამა–სკრიპტი A ვექტორში პირველი 0–ის ტოლი ელემენტის ნომერის მოსაძებნად. თუ ასეთი ელემენტი A–ში არ არის, გამოვიტანოთ შესაბამისი შეტყობინება:

```
m=length(A);
nom=0;
for i=1:m
    if  A(i)==0
        nom=i
        break
    end
end
if  nom==0
    'A-Si ar aris 0-is toli elementebi'
end
```

4.2.5 function და return ოპერატორები

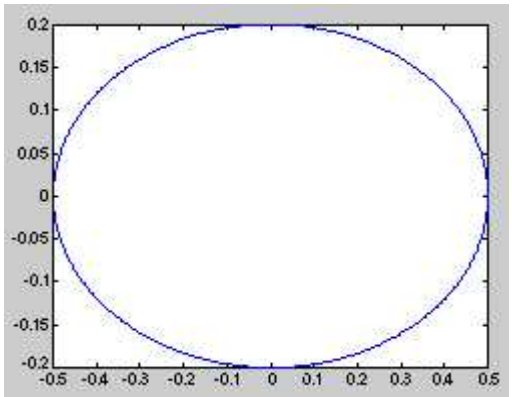
return ოპერატორის მეშვეობით შესაძლებელია პროგრამის მუშაობის შეწყვეტა. მაგალითისათვის მოვიყვანოთ შემდეგი პროგრამა–ფუნქცია, რომელიც ასრულებს ვექტორების გაყოფას წევრ–წევრად. პროგრამა გამოთვლის ორი ვექტორის სიგრძეს, ამასთან, თუ ვექტორების ზომა არ არის ერთნაირი, მოხდება პროგრამის მუშაობის შეწყვეტა შესაბამისი შეტყობინებით, ხოლო 0–ზე გაყოფისას მოხდება შეტყობინების გამოყვანა და ციკლის შეწყვეტა. პროგრამების შეწყვეტისას ცვლადებს მიენიჭება მიმდინარე ბრძანებით გამოთვლილი მნიშვნელობა.

```
function s=jjj(a,b)
n1=length(a); n2=length(b);
if n1~=n2
    'veqtorebis zoma ar aris erTnairi'
    return
end
for i=1:n1
    if b(i)==0
       '0-ze gayofaa'
       break
    else
        s(i)=a(i)/b(i);
    end
end
'programis muSaoba dasrulda'
```

დაბოლოს, დავუბრუნდეთ function ოპერატორს, რომლის ფორმები უკვე იყო მოცემული 4.1–ში. დასაშვებია function ოპერატორის კიდევ ორი კონსტრუქცია: **function ფუნქციის_სახელი** და **function ფუნქციის_სახელი (შემავალი_პარამეტრების_სია)**.

ქვემოთ, მარცხნივ, მოყვანილია პროგრამა–ფუნქცია გამომავალი პარამეტრების გარეშე. პროგრამა ახორციელებს პარამეტრული გრაფიკის აგებას ანუ გრაფიკის, რომლის აბსცისაც და ორდინატაც პარამეტრზეა (t –ზე) დამოკიდებული ($x(t) = k_1 \sin(t), y(t) = k_2 \cos(t)$). მარჯვნივ მოყვანილია ბრძანებათა ფანჯრიდან parametruli_grafiki(0.5,0.2) ბრძანებით მიღებული გრაფიკი.

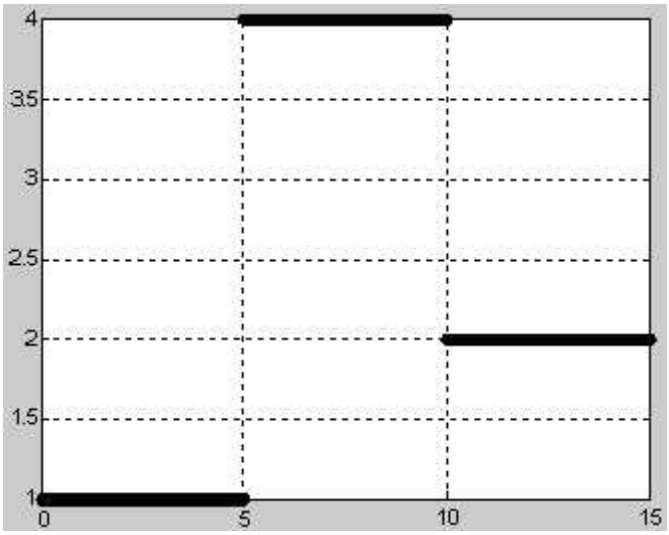
```
function parametruli_grafiki(k1,k2)
t=0:0.01:2*pi;
x=k1*sin(t);
y=k2*cos(t);
plot(x,y);
```

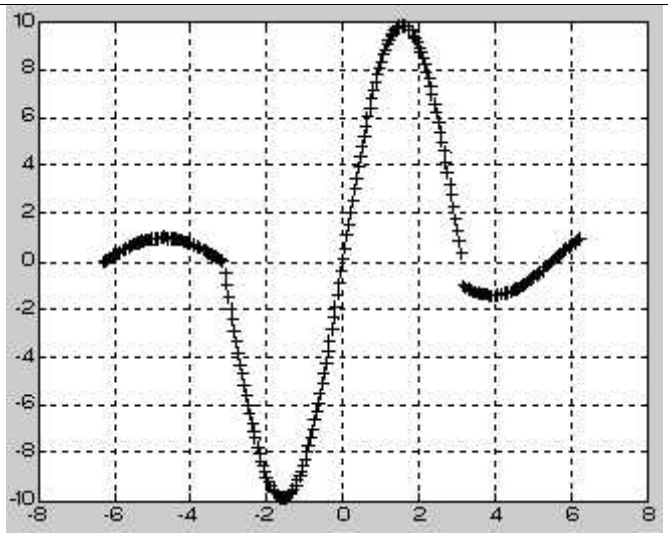


ნახ.4.2

მოვიყვანოთ პროგრამა–ფუნქციები შემავალი და გამომავალი პარამეტრების გარეშე. ორივე პროგრამა ახორციელებს უბან–უბან განსაზღვრული ფუნქციების გრაფიკების აგებას:

$$f_1(t)=\begin{cases}1, & 0\le t<5\\ 4, & 5\le t<10\\ 2, & 10\le t\le 15\end{cases} \qquad \text{და} \qquad f_2(x)=\begin{cases}|\sin(x)| & , \ -2\pi\le x\le -\pi\\ \pi^2\sin(x) & , \ -\pi<x\le\pi\\ \sin(x)+\cos(x), & \pi<x\le 2\pi\end{cases}$$

1	function uug_grafiki
	t=0:0.01:15;
	n=length(t);
	for i=1:n
	  if t(i)>=0&t(i)<5
	  y(i)=1;
	  elseif t(i)>=5&t(i)<10
	  y(i)=4;
	  else
	  y(i)=2;
	  end
	end
	plot(t,y,'ko'); grid on
	
	ბრძანებათა ფანჯარაში აკრეფილია
	>> uug_grafiki
		
		ნახ.4.3

2	function ububg_grafiki X1=-2*pi:0.05:-pi; Y1=abs(sin(X1)); X2=-pi+0.05:0.05:pi; Y2=pi^2*sin(X2); X3=pi+0.05:0.05:2*pi; Y3=sin(X3)+cos(X3); X=[X1 X2 X3]; Y=[Y1 Y2 Y3]; plot(X,Y,'k+'); grid on ბრძანებათა ფანჯარაში აკრეფილია >> ububg_grafiki	 ნახ.4.4

თავი V სიმბოლური ინფორმაციის დამუშავება

5.1 სიმბოლური ცვლადები და ფუნქციები

როგორც უკვე აღვნიშნეთ, მიუხედავად იმისა, რომ MATLAB სისტემა ორიენტირებულია რიცხვითი გამოთვლების საწარმოებლად, მისი საშუალებით შესაძლებელია სიმბოლური გამოთვლების ჩატარებაც. **help symbolic** ბრძანებით მიიღება დაწვრილებითი ინფორმაცია MATLAB–ის სიმბოლური ოპერაციებისა და ფუნქციების შესახებ, ხოლო ამ საკითხთან დაკავშირებული სადემონსტრაციო მასალის დასათვალიერებლად ვსარგებლობთ **symintro** ბრძანებით.

რიცხვითი ცვლადებისაგან განსხვავებით, სიმბოლური ინფორმაციის დამუშავებისას პირველ რიგში უნდა აღვწეროთ საჭირო მუდმივები და ცვლადები როგორც სიმბოლური ობიექტები (symbolic objects). როგორც უკვე იყო აღნიშნული 2.7 პარაგრაფში, ამას ემსახურება **syms** ოპერატორი. მაგ.,

```
>> syms x y
```

ბრძანებით შეიქმნება x და y სიმბოლური ცვლადები. სიმბოლური ცვლადისათვის მეხსიერებაში გამოიყოფა 126 ბაიტი. შედარებისათვის – რიცხვისათვის გამოიყოფა მხოლოდ 8.

სიმბოლური მუდმივების გამოსაცხადებლად გამოიყენება ფუნქცია **sym**. მისი არგუმენტია აპოსტროფებში ჩასმული სიმბოლური სტრიქონი, რომელიც შეიცავს სპეციალური ცვლადის სახელს, რიცხვით გამოსახულებას, ფუნქციის სახელს ან სიმბოლურ გამოსახულებას. შემდეგი ოპერატორებით:

```
>> pi = sym('pi'); delta = sym('1/10'); sqrt2 = sym('sqrt(2)');
```

შექმნილია 3 სიმბოლური მუდმივი. აღვნიშნოთ, რომ თუ pi სიმბოლურ მუდმივს ასე განვსაზღვრავთ, იგი მოცემულ სამუშაო სივრცეში შეცვლის pi–ს რიცხვით მნიშვნელობას.

თუ ზემოხსენებული ბრძანებების შემდეგ მივცემთ ბრძანებას **whos**, მივიღებთ:

```
Name      Size                Bytes      Class
delta     1x1                   132      sym object
pi        1x1                   128      sym object
sqrt2     1x1                   138      sym object
x         1x1                   126      sym object
y         1x1                   126      sym object
Grand total is 20 elements using 650 bytes
```

5.2 სიმბოლური გამოსახულება

სიმბოლური ცვლადი შეიძლება გამოვიყენოთ სიმბოლურ გამოსახულებაში როგორც ცვლადი ან ფუნქციის არგუმენტი. +, -, *, /, ^ ოპერაციებისა და MATLAB–ის ფუნქციების გამოყენება იგივეა, რაც რიცხვითი გამოთვლების წარმოებისას. მაგ.,

```
>> syms s t A
>> f = s^2 + 4*s + 5
f =
s^2+4*s+5
>> h = f*(s+2)
h =
(s^2+4*s+5)*(s+2)
>> y = A*exp(-s*t)
y =
A*exp(-s*t)
```

როგორც ამ მაგალითებიდან ჩანს, ახლადშექმნილი ცვლადები (f, h და v) არ საჭიროებენ გამოცხადებას. ეს სიმბოლური ცვლადებია და არა რიცხვითი გამოსახულებები.

სიმბოლურ გამოსახულებაში შეიძლება ერთდროულად გამოვიყენოთ ჩვეულებრივი (რიცხვითი) და სიმბოლური ცვლადები და მუდმივები. **findsym(S)** ფუნქცია S სიმბოლურ გამოსახულებაში ან მატრიცაში პოულობს სიმბოლურ ცვლადებს და გვაძლევს მათ, როგორც სიმბოლურ სტრიქონს, ანბანის მიხედვით დალაგებულს და მძიმეებით გამოყოფილს:

```
>> findsym(f)
ans =
s
>> findsym(y)
ans =
A, s, t
```

თუ S არ შეიცავს არც ერთ სიმბოლურ ცვლადს, მიიღება ცარიელი სტრიქონი.

სიმბოლური ცვლადები შეიძლება იყოს ვექტორის ან მატრიცის ელემენტები. ქვემოთ მოყვანილია სიმბოლური მატრიცის შექმნის მაგალითი:

```
>> n = 3;
>> syms x
>> B = x.^((0:n)'*(0:n))
B =
[1, 1, 1, 1]
[1, x, x^2, x^3]
[1, x^2, x^4, x^6]
[1, x^3, x^6, x^9]
```

sym ფუნქციით შესაძლებელია რიცხვითი ცვლადების მიღება სიმბოლური ფორმით. მაგ., t=8.2; w=sym(t) ოპერატორებით მივიღებთ w=41/5. რაციონალური წილადების გამოყენება სიმბოლურ გამოთვლებში ნიშნავს, რომ ყოველთვის მიღებული იქნება ზუსტი პასუხი, დამრგვალების ცდომილების გარეშე.

ქვემოთ მოყვანილია სიმბოლურ გამოსახულებებთან მომუშავე ძირითადი ფუნქციები (ზემოთ განხილული და 2.7 პარაგრაფში აღწერილი ფუნქციების გარდა):

ფუნქციის აღწერა	მაგალითი
char(x) არაუარყოფითი მთელი რიცხვებისაგან შემდგარ x მასივს, რომლის ელემენტები წარმოადგენს სიმბოლოთა კოდებს, გარდაქმნის სიმბოლოთა სტრიქონად	მაგალითი აჩვენებს თუ რომელი სიმბოლო შეესაბამება კოდს 32–დან 95–მდე: >> char(reshape(32:95, 32, 2)') ans = !"#$%&'()*+,-./0123456789:;<=>? @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
ischar(s) გვაძლევს 1-ს, თუ s სიმბოლოთა სტრიქონია, სხვა შემთხვევაში – 0-ს	>> x='MATLAB' >> ischar(x) ans = 1
isletter(s) გვაძლევს 1-ს s სტრიქონის ყოველი ელემენტისათვის, რომელიც ასოით გამოსახულებას წარმოადგენს, და სხვა შემთხვევაში 0-ს	>> R='group 845' >> isletter(R) ans = 1 1 1 1 1 0 0 0 0
blancs(n) გვაძლევს სტრიქონს, რომელიც შეიცავს n ცარიელ პოზიციას n სიმბოლოს ნაცვლად; გამოიყენება შედეგების გამოყვანის ფუნქციებში	>> disp(['xxx' blanks(20) 'yyy']) xxx yyy
deblank(s) შეკვეცს ცარიელ პოზიციებს სტრიქონის ბოლოში	>> A{1,1} = 'MATLAB '; A{1,2} = 'SIMULINK '; A = deblank(A) A = 'MATLAB' 'SIMULINK'
bin2dec(b) ახდენს b სტრიქონის სახით ჩაწერილი ორობითი რიცხვის ინტერპრეტირებას და გვაძლევს შესაბამის ათობით რიცხვს.	>> bin2dec('11011') ans = 27
dec2bin(d) გვაძლევს არაუარყოფითი 2^{52}–ზე ნაკლები d მთელი რიცხვის ორობით წარმოდგენას სტრიქონის სახით	>> dec2bin(23) ans = 10111
bitshift(x,k) მოგვცემს მთელ ათობით რიცხვს, რომელიც წარმოადგენს x რიცხვის ორობითი წარმოდგენის k ბიტით წანაცვლების შედეგს. თუ k>0, ეს ოპერაცია იგივეა, რაც გამრავლება 2^k–ზე, თუ k<0 – იგივეა, რაც გაყოფა $2^{\|k\|}$–ზე	ათობით რიცხვს 12 შეესაბამება ორობითი 1100. მისი წანაცვლებით მარცხნივ 2 პოზიციით მივიღებთ 110000 ანუ ათობით 48–ს. >> bitshift(12, 2) ans = 48
bitand(a,b) ბიტური and ოპერაცია	>> A=[1 0 0]; B=[1 1 0]; >> bitand(A,B) 1 0 0

bitor(a,b) ბიტური or ოპერაცია	>> A=[1 0 0]; B=[1 1 0]; bitor(A,B) ans = 1 1 0
bitcmp(a,n) a ათობითი მთელი რიცხვის n-ბიტიან ორობით წარმოდგენაში შეცვლის 0-ს 1-ით, ხოლო 1-ს 0-ით და მოგვცემს შესაბამის ათობით რიცხვს	>> A=5; bitcmp(A,3) ans = 2 >> c=8; bitcmp(c,4) ans = 7
pretty(S) ახდენს S სიმბოლური გამოსახულების გადაყვანას ზოგადად მიღებულ მათემატიკურ სახეზე. სამწუხაროდ, ფუნქციას არ გააჩნია სპეციალური მათემატიკური ნიშნები: ინტეგრალის, ჯამის, ნამრავლის და სხვ.	>> syms a b c >> t=(2-cos(b))^2*(sin(2*a))^(1/2)/abs(c)^3 t = (2-cos(b))^2*sin(2*a)^(1/2)/abs(c)^3 >> pretty(t) 2 1/2 (2 - cos(b)) sin(2 a) ---------- 3 | c |
[n,d]=numden(S) გვაძლევს რაციონალურ წილადად წარმოდგენილი S სიმბოლური მასივის ყოველი ელემენტის მრიცხველსა და მნიშვნელს მთელკოეფიციენტებიანი დაყვანილი პოლინომის სახით	[n,d] = numden(sym(4/5)) ოპერატორით მივიღებთ პასუხს: n = 4, d =5. ოპერატორებით syms x y; [n,d] = numden(x/y + y/x) მივიღებთ: n = x^2+y^2, d = y*x. ოპერატორებით syms s h; A = [s, 1/h]; [n,d] = numden(A) მივიღებთ: n = [s, 1], d = [1, h].
findstr(s1,s2) ეძებს ერთ სტრიქონს მეორის შიგნით. თუ ძებნა უშედეგოდ დამთავრდა, პასუხად მიიღება ცარიელი სიმრავლე.	>> s = 'Find the starting indices of the shorter string'; >> findstr(s, 'the') ans = 6 30 >> findstr('the', s) ans = 6 30 >> findstr('The', s) ans = []
eval(s) გაუშვებს s სტრიქონს როგორც MATLAB გამოსახულებას ან ბრძანებას. **num2str(n)** ფუნქციით n რიცხვიდან მიიღება სტრიქონი, ხოლო **str2num(s)**–ით s სტრიქონი, რომელიც უნდა იყოს რიცხვითი სიდიდის შესაბამისი ASCII კოდი, გარდაიქმნება შესაბამის რიცხვით მნიშვნელობად	>> for n = 4:6 eval(['M' num2str(n) ' = magic(n)']) end ამ ბრძანების შესრულების შედეგად მივიღებთ მაგიურ კვადრატებს სახელით M4, M5 და M6..

double(S) S სიმბოლურ გამოსახულებას გარდაქმნის double array ტიპის რიცხვით სიდიდედ	>> z=sym(sqrt(7)) z = sqrt(7) >> a=double(z) a = 2.6458 აქ a ცვლადი double ტიპისაა და იკავებს 8 ბაიტს, ხოლო სიმბოლური ცვლადი z იკავებს 138 ბაიტს, რაშიდაც ადვილად დავრწმუნდებით whos a z ბრძანების მეშვეობით
vpa(x,n) წარმოადგენს რიცხვს მითითებული n სიზუსტით (გამოყენებული ციფრების რაოდენობით), ამასთან x შეიძლება იყოს როგორც რიცხვითი გამოსახულება, ასევე სიმბოლური. ორივე შემთხვევაში პასუხი იქნება სიმბოლური ტიპისა. ზოგადად **digits** ფუნქციით შეიძლება გავიგოთ არითმეტიკული გამოთვლების მიმდინარე სიზუსტე (*by default* ის 32–ის ტოლია), ხოლო **digits(k)**–ით დავაყენოთ k სიზუსტე.	>> vpa(a,4) ans = 2.646 >> vpa(z,4) ans = 2.646 >> vpa(z,10) ans = 2.645751311
simplify(S) ფუნქცია ახორციელებს სიმბოლური გამოსახულების გამარტივებას. იგივეს აკეთებს **simple(S)** იმ განსხვავებით, რომ ეს უკანასკნელი გამარტივებისას სხვადასხვა ხერხებს იყენებს და ეკრანზე გამოჰყავს გამარტივების სხვადასხვა ვარიანტი	>> syms c a b >> simplify(exp(c*log(sqrt(a+b)))) ans = (a+b)^(1/2*c) >> S = [(a^2+5*a+6)/(a+2),sqrt(25)]; R = simplify(S) R = [a+3, 5]
expand(S)–ით ხდება გამოსახულების გაშლა. ეს ფუნქცია, შეიძლება ითქვას, გამარტივების საპირისპიროა. მისი მეშვეობით გამოსახულებაში შესული რთული ფუნქციები უფრო მარტივი ფუნქციებით გამოისახებიან და ხდება ალგებრული გამოსახულების წარმოდგენა გაშლილი ფორმით	>> syms x; expand((x-2)^2*(x-3)^3+6) ans = x^5-13*x^4+67*x^3-171*x^2+216*x-102 >> syms b; expand(3+2*exp(a+b)) ans = 3+2*exp(a)*exp(b) >> z=sym('sin(x+y)'); >> expand(z) ans = sin(x)*cos(y)+cos(x)*sin(y)
factor(S) ახდენს მათემატიკური გამოსახულების მამრავლებად დაშლას	>> syms x; z=sin(x)^4-1; >> factor(z) ans = (sin(x)-1)*(sin(x)+1)*(sin(x)^2+1) >> factor(204) ans = 2 2 3 17

collect(S) S გამოსახულებას მწკრივად გაშლის მსგავსი წევრების შეკრებით, ხოლო **collect(S,x)** გაშლას შეასრულებს x–ის მიმართ	>> syms x y; R1 = collect((exp(x)+x)*(x+2)) R1 = x^2+(exp(x)+2)*x+2*exp(x) >> R2 = collect((x+y)*(x^2+y^2+1), y) R2 = y^3+x*y^2+(x^2+1)*y+x*(x^2+1)
subs(S) S სიმბოლურ გამოსახულებაში შეცვლის ყველა ცვლადს მათი მნიშვნელობებით, რომლებიც აიღება სამუშაო არიდან. **subs(S,x)** შეცვლის S გამოსახულებაში სიმბოლურ ცვლადს x–ით, ხოლო **subs(S,y,x)** შეცვლის მითითებულ y ცვლადს x–ით, ამასთან, x და y შეიძლება სიის სახით იყოს მოცემული	>> w=8; b=7; >> subs(exp(-w*t)*b) ans = 7*exp(-8*t) >> syms x t >> z=x^3-7*x^2+1; >> subs(z,2.5) ans = -27.1250 >> subs(z,t) ans = t^3-7*t^2+1 >> z=x^3-7*x^2+1; subs(z,{3;11}) ans = -35 485 >> S=x^2-2*t^3+5; subs(S,{x,t},{[1,-1],[0,2]}) ans = 6 -10
finverse(f) გვაძლევს ერთცვლადიანი ფუნქციის შებრუნებულს, ხოლო **finverse(f,v)** – f ფუნქციის შებრუნებულს v ცვლადის მიმართ	>> finverse(1/tan(x)) ans = atan(1/x) >> syms v; finverse(exp(u-2*v),u) ans = 2*v+log(u)

სიმბოლური ფუნქციის გრაფიკის მიღება ხდება ezplot ფუნქციის საშუალებით.

ezplot(F), სადაც F – სიმბოლური გამოსახულებაა, აგებს F(x) გრაფიკს. წინასწარი შეთანხმებით $-2\pi < x < 2\pi$. ორარგუმენტიანი სიმბოლური ფუნქციის შემთხვევაში აიგება წირი F(x,y)=0, სადაც $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$.

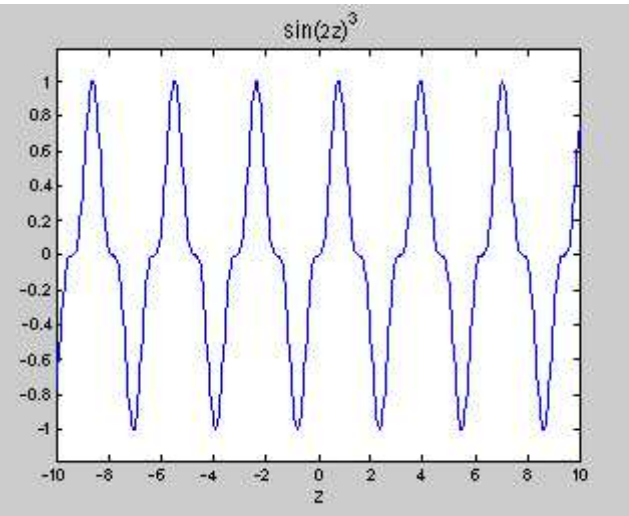
ezplot(F,[a,b]) გამოიყვანს F(x) გრაფიკს a<x<b ინტერვალში. ორარგუმენტიანი სიმბოლური ფუნქციის შემთხვევაში ezplot(F,[a,b]) აგებს F(x,y)=0 წირს, სადაც a<x<b და a<y<b, ხოლო **ezplot(F,[a,b,c,d])** აგებს იგივე წირს a<x<b და c<y<d ინტერვალებში.

ezplot(FX,FY), სადაც FX, FY – სიმბოლური გამოსახულებებია, გამოსახავს პარამეტრულად განსაზღვრულ წირს: FX(t) და FY(t), სადაც, წინასწარი შეთანხმების თანახმად, $0 < t < 2\pi$.

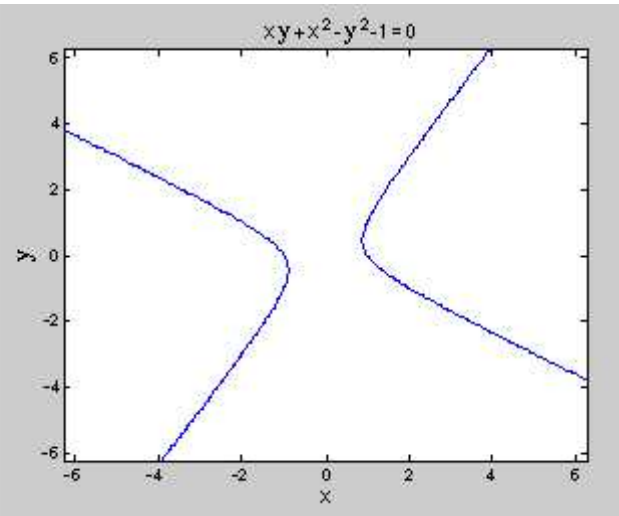
ezplot(FX,FY,[a,b]) აგებს პარამეტრულ წირს a<t<b ინტერვალში.

ნახ.5.1–ზე და 5.2–ზე მოყვანილია შემდეგი ოპერატორებით მიღებული წირები:

```
>> syms z; F=sin(2*z)^3; a=-10; b=10;
>> ezplot(F,[a,b])
>> figure
>> ezplot('x*y+x^2-y^2-1')
```



ნახ.5.1



ნახ.5.2

სიმბოლური ფუნქციების ვიზუალიზაციისათვის არსებობს სხვა ფუნქციებიც: **ezcontour**, **ezmesh**, **ezpolar**, **ezsurf** და ა.შ., რომლებიც შესაბამისი ჩვეულებრივი ფუნქციების ანალოგებს წარმოადგენენ.

5.3 MATLAB სისტემის სიმბოლური მათემატიკის საბაზო ოპერაციები – Symbolic Math Toolbox პაკეტი

Symbolic Math Toolbox პაკეტის საშუალებით შესაძლებელია ყველა ძირითადი მათემატიკური ამოცანის ამოხსნა. რა თქმა უნდა, გასათვალისწინებელია, რომ ზოგ შემთხვევაში ამოცანას შეიძლება არ ჰქონდეს ანალიზური ამოხსნა. აქვე აღვნიშნოთ, რომ ზოგიერთი ქვემოთ ახსნილი ფუნქციებიდან არ შედის MATLAB ფუნქციათა ბიბლიოთეკაში ან შედის სხვა მნიშვნელობით (მაგ, diff ფუნქცია), ამიტომ **help ფუნქციის_სახელი** მიმართვა უშედეგო აღმოჩნდება ან სხვა ინფორმაციას მოგვცემს. Symbolic Math Toolbox პაკეტი აკავშირებს MATLAB–ს Maple–ის ფუნქციების ბიბლიოთეკასთან. ამგვარ შემთხვევაში ინფორმაციას შესაბამისი ფუნქციის შესახებ მივიღებთ **mhelp ფუნქციის_სახელი** ბრძანებით.

ქვემოთ მოცემულია იმ ფუნქციების მოკლე მიმოხილვა, რომლებითაც ვსარგებლობთ ძირითადი მათემატიკური ამოცანების ამოსახსნელად; ამ ფუნქციების მინიმუმ ერთი პარამეტრია სიმბოლური გამოსახულება.

- განვიხილოთ *წრფივი ალგებრის* სიმბოლური ამოცანების ამოხსნა.

მატრიცული ოპერაციების განხორციელება შესაძლებელია სიმბოლური მატრიცებისთვისაც. სიმბოლურ მატრიცებთან და ვექტორებთან სამუშაოდ ვსარგებლობთ ფუნქციებით, რომლებსაც, როგორც წესი, იგივე შინაარსი აქვთ, რაც რიცხვითი არგუმენტის მქონე შესაბამის ფუნქციებს: **det(X)** ფუნქციას ეკრანზე გამოჰყავს სიმბოლური X მატრიცის დეტერმინანტი; **diag(X)** – მისი დიაგონალი (ამ ფუნქციით სარგებლობენ აგრეთვე დიაგონალური მატრიცის შესაქმნელად); **diag(X,k)** – k-ური დიაგონალი; **trace(X)** მოგვცემს დიაგონალური ელემენტების ჯამს; **sum(X)** – ელემენტების ჯამს თითოეულ სვეტში ან ვექტორისათვის – ვექტორის ელემენტების ჯამს; **sum(X,k)** – ჯამს k-ური განზომილების მიხედვით; **prod(X)** – ელემენტების ნამრავლს თითოეულ სვეტში ან ვექტორისათვის – მისი ელემენტების ნამრავლს; **prod(X,k)** – ნამრავლს k-ური განზომილების მიხედვით; **inv(X)** მოგვცემს კვადრატული მატრიცის შებრუნებულს.

X = sym('[a1 b1 c1;d1 e1 f1;g1 h1 k1]');	>> syms t1 t2 h1 h2; >> Z=[t1 t2; h1 h2];
>> det(X) ans = a1*e1*k1-a1*f1*h1-d1*b1*k1+d1*c1*h1+g1*b1*f1-g1*c1*e1 >> diag(X) ans = a1 e1 k1 >> trace(X) ans = a1+e1+k1 >> diag(X,1) ans = b1 f1 >> diag(diag(X)) ans = [a1, 0, 0] [0, e1, 0] [0, 0, k1]	>> sum(Z) ans = [t1+h1, t2+h2] >> prod(Z) ans = [t1*h1, t2*h2] >> sum(Z,2) ans = t1+t2 h1+h2 >> prod(Z,2) ans = t1*t2 h1*h2 >> inv(Z) ans = [-h2/(-t1*h2+t2*h1), t2/(-t1*h2+t2*h1)] [h1/(-t1*h2+t2*h1), -t1/(-t1*h2+t2*h1)]

- *ტეილორის მწკრივად ფუნქციის გაშლისათვის* ვსარგებლობთ **taylor(F)** ფუნქციით. წინასწარი შეთანხმებით მიღებული იქნება მე-5 ხარისხის პოლინომით F ფუნქციის აპროქსიმაცია. **taylor(F,n)**-ით მივიღებთ (n-1) ხარისხის აპროქსიმაციას, ხოლო **taylor(F,a)**-ით – აპროქსიმაციას a წერტილის მიდამოში. როდესაც F მრავალცვლადიანია, ამ ფუნქციებს შეიძლება დაემატოს y პარამეტრი, რომელიც აჩვენებს, თუ რომელი ცვლადის მიმართ უნდა მოხდეს გაშლა:

  >> syms z; taylor(exp(-z))

  ans =

```
1-z+1/2*z^2-1/6*z^3+1/24*z^4-1/120*z^5
>> taylor(exp(-z),7)
ans =
1-z+1/2*z^2-1/6*z^3+1/24*z^4-1/120*z^5+1/720*z^6
 >> taylor(log(z),6,1)
 ans =
 z-1-1/2*(z-1)^2+1/3*(z-1)^3-1/4*(z-1)^4+1/5*(z-1)^5
>> syms t; taylor(z^t,3,t)
ans =
1+log(z)*t+1/2*log(z)^2*t^2
```

- მწკრივის ჯამი გამოითვლება symsum ფუნქციით.
  symsum(S) ფუნქცია გამოთვლის მწკრივის ჯამს, როდესაც მისი სიმბოლური ცვლადი k იცვლება 0-დან (k-1)-მდე:

```
>> syms  n;  symsum(n^2)
ans =
1/3*n^3-1/2*n^2+1/6*n
```

  symsum(S,k) ფუნქცია გამოთვლის ჯამს, როდესაც ფუნქციაში მითითებული სიმბოლური ცვლადი k იცვლება 0-დან (k-1)-მდე.
  symsum(S,a,b) გამოთვლის მწკრივის ჯამს, როდესაც მისი სიმბოლური ცვლადი იცვლება a-დან b-მდე:

```
>> symsum(k^2,0,10)
ans =
385
```

  symsum(S,k,a,b) ფუნქციით გამოითვლება მწკრივის ჯამი, როდესაც ფუნქციაში მითითებული სიმბოლური ცვლადი k იცვლება a-დან b-მდე. მაგ., გამოვთვალოთ $\sum_{k=0}^{\infty}(-1)^k \frac{z^{2k}}{(2k)!}$ მწკრივის ჯამი:

```
>> syms z  k;   S=(-1)^k*z^(2*k)/sym('(2*k)!');
>> P=symsum(S,k,0,Inf)
P =
cos(z)
```

  ყველა ამ ფუნქციაში S – სიმბოლური გამოსახულებაა ან ინდექსზე დამოკიდებული ფუნქციის სახელი.

- *ზღვრის მოსაძებნად* გამოიყენება **limit(S,x,a)** ფუნქცია, რომელიც გამოთვლის სიმბოლური გამოსახულების ზღვარს, როდესაც $x \to a$; **limit(S)** გამოთვლის ზღვარს, როდესაც მისი ცვლადი $x \to 0$; **limit(S,x,a,'right')** და **limit(S,x,a,'left')** – შესაბამისად მარჯვენა და მარცხენა ზღვრებს. ასე მაგალითად, limit(sin(x)/x) ფუნქცია პასუხად მოგვცემს 1, limit((x-2)/(x^2-4),2) – 1/4, limit((1+2*x/t)^(3*t),t,inf) – exp(6*x), limit(1/x,x,0,'right') – inf, limit(1/x,x,0,'left') – -inf.

- *სიმბოლური გამოსახულების დიფერენცირება* ხდება **diff** ფუნქციით, რომლის სემანტიკა უკვე იყო აღწერილი 2.7–ში. diff('3^x+6') ფუნქციით მივიღებთ 3^x*log(3), diff('3^x+6',2)-ით – 3^x*log(3)^2. syms x y; diff('sin(x)^2+cos(y)',x)-ით მივიღებთ წარმოებულს x ცვლადით ანუ 2*sin(x)*cos(x), ხოლო diff('sin(x)^2+cos(y)',y)-ით – y ცვლადით ანუ -sin(y). diff('sin(x)^2+cos(y)',x,2) მოგვცემს მე-2 რიგის წარმოებულს x ცვლადით.

- *ინტეგრალის გამოსათვლელად* ვსარგებლობთ int ფუნქციით, რომლის სემანტიკა აგრეთვე უკვე აღვწერეთ 2.7–ში.

  int(sin(x)) მოძებნის sin(x))-ის *განუსაზღვრელ ინტეგრალს*, რომელიც უდრის -cos(x), int(sin(x)^2+cos(y)^3) და syms x; int(sin(x)^2+cos(y)^3,x) ოპერატორებით მივიღებთ ინტეგრალს x ცვლადის მიმართ: -1/2*sin(x)*cos(x)+1/2*x+cos(y)^3*x, ხოლო syms y; int(sin(x)^2+cos(y)^3,y)-ით – y ცვლადის მიმართ: sin(x)^2*y+1/3*cos(y)^2*sin(y)+2/3*sin(y). int(int(2*sin(x)^2)) ოპერატორი შედეგად მოგვცემს -1/2*sin(x)^2+1/2*x^2.

  რა თქმა უნდა, განუსაზღვრელი ინტეგრალის მოძებნა ყოველთვის როდია შესაძლებელი. ზოგიერთ შემთხვევაში კი პასუხი – პირველყოფილი ფუნქცია შეიცავს შეცდომის ფუნქციას. მაგ., syms x; int(exp(-x^2)) ოპერატორებით მივიღებთ:

  ans =

  1/2*pi^(1/2)*erf(x)

  აქ erf(x) შეცდომის ფუნქციაა: $erf(x) = \frac{2}{\sqrt{®}} \int_0^x e^{-t^2} dt$. შეცდომის ფუნქციებია აგრეთვე erfc, erfcx, erfinv, erfcinv. ასეთი სახის პასუხის მიღებისას ინტეგრალის გამოსახულების მისაღებად საჭიროა დამატებითი გარდაქმნების ჩატარება.

  int(sin(x)^2,-pi/2,pi/2) გამოითვლის sin(x)^2 ფუნქციის *განსაზღვრულ ინტეგრალს* -®/2-დან ®/2-მდე, პასუხად მივიღებთ 1/2*pi. int(sin(x)^3+2*cos(y)^3,y,-pi/2,pi/2) ოპერატორი მოგვცემს მითითებული ფუნქციის განსაზღვრულ ინტეგრალს -®/2-დან ®/2-მდე y სიმბოლური ცვლადის მიმართ: 8/3+sin(x)*pi-sin(x)*pi*cos(x)^2.

- *ალგებრული განტოლების ან განტოლებათა სისტემის ამოსახსნელად* განკუთვნილია solve ფუნქცია, ამასთან, მეოთხე რიგის ჩათვლით განტოლება ამოიხსნება ზუსტად, უფრო მაღალი რიგის შემთხვევაში – მიახლოებით.

  solve(E) ამოხსნის განტოლებას მისი ცვლადის მიმართ (თუ განტოლებაში შედის ერთზე მეტი ცვლადი, ის ამოხსნილი იქნება *by default* განსაზღვრული ცლადის მიმართ), ხოლო **solve(E,v)** – მითითებული v ცვლადის მიმართ. აქ E – სიმბოლური გამოსახულებაა (მაგ., x^2-2*x+1) ან სტრიქონი ('x^2-2*x+1'). თუ E არ შეიცავს ტოლობის ნიშანს, ამოხსნილი იქნება E=0 განტოლება:

  >> solve('a*x^4-2*x^3+a*x^2-2*x')

  ans =

```
  0
  i
 -i
 2/a
>> solve('5*x^2 + 10*x + 7=16')
ans =
 -1+1/5*70^(1/2)
 -1-1/5*70^(1/2)
>> w=solve('a*x^2 + b*x + c','b')
w =
-(a*x^2+c)/x
```

subs ოპერატორის საშუალებით შეგვიძლია ვიპოვოთ განტოლების ამონახსნი სიმბოლური ცვლადების სხვადასხვა მნიშვნელობათათვის:

```
>> subs(w,{a,b,c,x},{1,-2,1,5})
ans =
-26/5
```

განტოლებებს, რომლებიც შეიცავენ პერიოდულ ფუნქციებს, შესაძლოა ჰქონდეთ ფესვების განუსაზღვრელი რაოდენობა. ასეთ შემთხვევაში solve ფუნქცია შემოიფარგლება ფესვებით 0-ის მიდამოში:

```
>>  syms theta;  E = cos(2*theta)-sin(theta);  solve(E)
ans =
 -1/2*pi
  1/6*pi
  5/6*pi
```

განხილული solve ფუნქციის ანალოგიური შინაარსი აქვს **solve(E1,E2,...,En)** და **solve(E1,E2,...,En,v1,v2,...,vn**) ფუნქციებს, რომლებითაც ვსარგებლობთ განტოლებათა სისტემის ამოსახსნელად:

```
>> A = solve('a*u^2 + v^2', 'u - v = 1', 'a^2 - 5*a + 6')
A =
   a: [4x1 sym]
   u: [4x1 sym]
   v: [4x1 sym]
```

როგორც ვხედავთ, პასუხი მიღებულია სტრუქტურის სახით. ასეთი ტიპის პასუხი გამოდის, როდესაც სისტემის ამოხსნისას მითითებულია ერთი გამოსავალი პარამეტრი. A.a , A.u და A.v ოპერატორებით მივიღებთ შესაბამისად:

>> A.a	>> A.u	>> A.v
2	1/3+1/3*sqrt(-1)*2^(1/2)	-2/3+1/3*sqrt(-1)*2^(1/2)
2	1/3-1/3*sqrt(-1)*2^(1/2)	-2/3-1/3*sqrt(-1)*2^(1/2)
3	1/4+1/4*sqrt(-1)*3^(1/2)	-3/4+1/4*sqrt(-1)*3^(1/2)
3	1/4-1/4*sqrt(-1)*3^(1/2)	-3/4-1/4*sqrt(-1)*3^(1/2)

მოვიყვანოთ არაწრფივ განტოლებათა სისტემის ამოხსნის მაგალითი:

```
>> syms x y;  f1=sin(x)^2-cos(x)^2;  f2=x-y;
>> [p1,p2]=solve(f1,f2,x,y)
p1 =
  3/4*pi
  1/4*pi
p2 =
  3/4*pi
  1/4*pi
```

მოვიყვანოთ ტრანსცენდენტულ განტოლებათა სისტემის ამოხსნის მაგალითი:

```
>> syms x1 x2; f1=exp(x1)-x2;  f2=2*x^2+3*x-10-3*x2;
>> [z1 z2]=solve(f1,f2,x1,x2)
z1 =
   log(2/3*x^2+x-10/3)
z2 =
   2/3*x^2+x-10/3
```

დაბოლოს აღვნიშნოთ, რომ განტოლების ან განტოლებათა სისტემის ამოხსნისას იმ შემთხვევაში, როდესაც პასუხის მიღება სიმბოლური სახით შეუძლებელია, solve ფუნქცია გვაძლევს რიცხვით ამონახსნს.

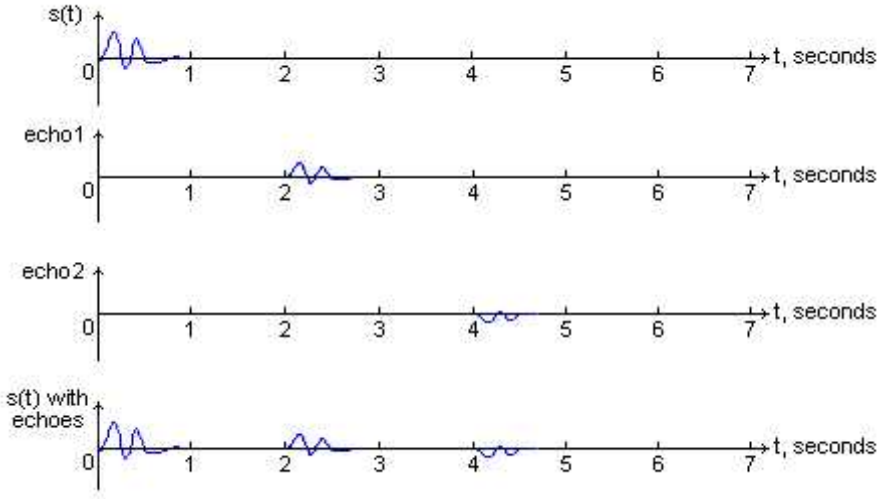
- *დიფერენციალური განტოლების ან განტოლებათა სისტემის ამოსახსნელად* განკუთვნილია dsolve ფუნქცია, რომელსაც აქვს შემდეგი სინტაქსი: **dsolve('E1,E2,...', 'P1,P2,...', 'v')** ან **dsolve('E1','E2',..., 'P1','P2',..., 'v')**. აქ E1, E2,... – განტოლებებია, P1, P2,... – სასაზღვრო პირობები (ცხადია, სასაზრვრო პირობები შედის dsolve ფუნქციაში მათი არსებობის შემთხვევაში), v – დამოუკიდებელი ცვლადი. თუ დამოუკიდებელი ცვლადი მითითებული არ არის, *by default* იგულისხმება t (დრო). D ასო აღნიშნავს წარმოებულს v ცვლადით. მაგ., Dy აღნიშნავს dy/dt, D2y – მეორე რიგის წარმოებულს და ა.შ. თუ სასაზღვრო პირობების რაოდენობა ნაკლებია დიფერენციალურ განტოლებათა რიცხვზე, მაშინ ამონახსნი შეიცავს C1, C2,... მუდმივებს. მოვიყვანოთ მაგალითები:

```
>> dsolve('Df = f + sin(t)')
ans =
-1/2*cos(t)-1/2*sin(t)+exp(t)*C1
>> dsolve('Dy = a*y', 'y(0) = b')
ans =
b*exp(a*t)
>> dsolve('(Dy)^2 + y^2 = 1','s')
ans =
        1
       -1
 sin(-s+C1)
-sin(-s+C1)
```

დანართი 1

სიგნალის გავრცელების პროცესში მიღებული ექოსიგნალები

ექოსიგნალი წარმოადგენს ძირითადი სიგნალის შესუსტებულ ანარეკლს, რომელიც მიმღებამდე აღწევს გარკვეული შეყოვნებით. მაგალითად, ნახ.1–ზე მოყვანილია საწყისი s(t) სიგნალი, echo1 სიგნალი, რომლის მნიშვნელობა შესუსტებულია 2–ჯერ (კოეფიციენტით 0.5) და 2 წამის დაგვიანებით დაერთვის საწყისს, echo2 სიგნალი -0.3 კოეფიციენტით და დაგვიანებული 5 წამით (ეს ე.წ. ჩახვეული (rolled) ექოა, რადგან ექოს მნიშვნელობები უარყოფითი სიდიდეებია) და ძირითადი სიგნალი თანდართული ორივე ექოთი.



ნახ.1 სიგნალი ექოებითურთ

დავუშვათ, სიგნალის მიღების პროცესი გრძელდება 10 წამი. პირველი წამის განმავლობაში შეგროვდა კოორდინატთა შემდეგი წყება:

დრო, წმ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
სიგნალის მნიშვნელობა	0.0	0.5	1.0	-0.5	0.75	0.0	-0.02	-0.10	0.0	0.0	0.0

სიგნალის შემდგომი მნიშვნელობები 0–ის ტოლია.

დავწეროთ პროგრამა–სკრიპტი, რომელიც შექმნის ძირითადი სიგნალისაგან და მის სამ ექოსიგნალისაგან შედგენილ სიგნალს: პირველი ექო – სიგნალი შესუსტებულია კოეფიციენტით 0.5 და დაგვიანებული 2 წამით, მეორე ექო – ჩახვეული ექოა 4 წამის დაგვიანებით და კოეფიციენტით -0.3, მესამე ექო დაგვიანებულია 7.5 წამით და შესუსტებული 10–ჯერ. ავაგოთ მიღებული სიგნალის გრაფიკი და შევინახოთ მიღებული სიგნალის მონაცემები MAT ფაილის სახით – echo.mat.

პროგრამის დაწერისას ექოს მნიშვნელობების გამოსათვლელად საჭიროა გავითვალისწინოთ შესუსტების კოეფიციენტი და დაგვიანების დრო. მაგ., სიგნალის პირველი 3 მონაცემისაგან მივიღებთ:

1–ლი ექო		მე–2 ექო		მე–3 ექო	
დრო	სიგნალი	დრო	სიგნალი	დრო	სიგნალი
2.0	0.5∗0.0 = 0.0	4.0	-0.3∗0.0 = 0.0	7.5	0.1∗0.0 = 0.0
2.1	0.5∗0.5 = 0.25	4.1	-0.3∗0.5 = -0.15	7.6	0.1∗0.5 = 0.05
2.1	0.5∗1.0 = 0.5	4.2	-0.3∗1.0 = -0.3	7.7	0.1∗1.0 = 0.1

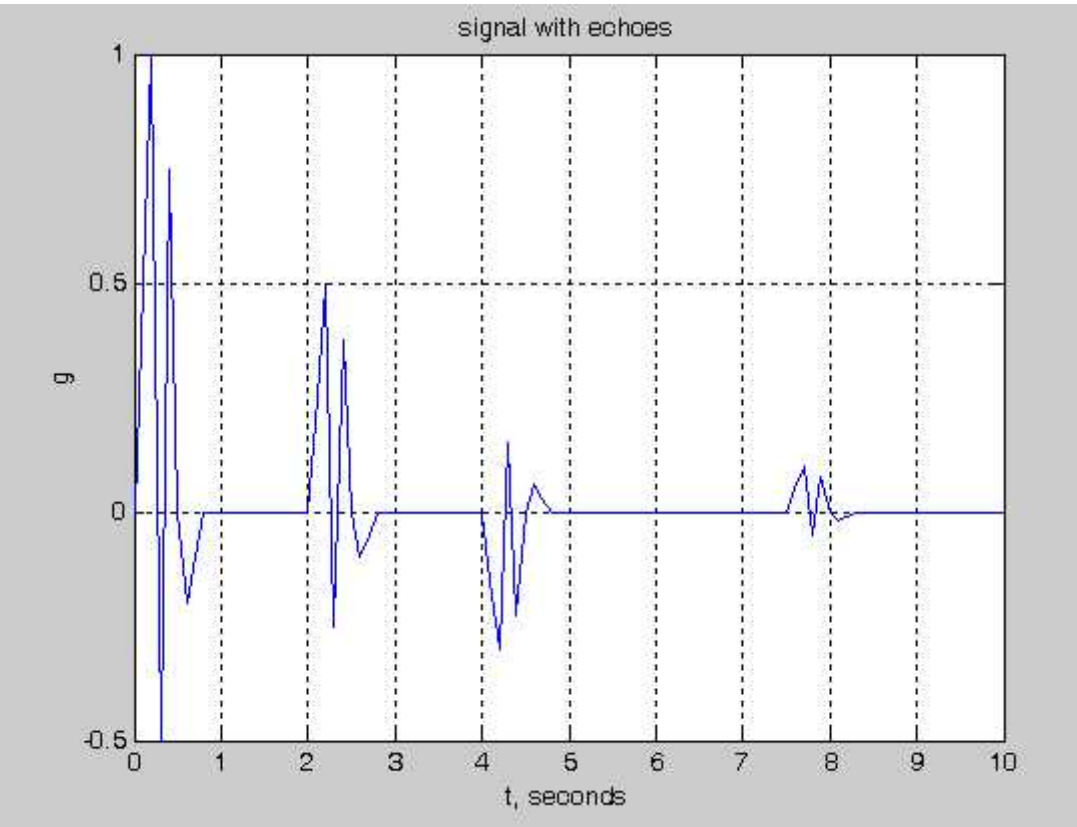
ანუ იმისათვის, რომ პირველი ექო იყოს 2 წამით დაგვიანებული, პროგრამაში უნდა გავითვალისწინოთ, რომ s(1) მნიშვნელობიდან მიიღება echo_1(21) მნიშვნელობა (რადგან ათვლის ინტერვალია 0.1), s(2)–დან – echo_1(22) და ა.შ. ზოგადად echo_1–ის მნიშვნელობების მისაღებად ძირითადი სიგნალის მნიშვნელობები უნდა გავამრავლოთ 0.5 კოეფიციენტზე და „გადავწიოთ“ ათვლის 20 წერტილით; echo_2–ის მნიშვნელობების მისაღებად ძირითადი სიგნალის მნიშვნელობები უნდა გავამრავლოთ -0.3–ზე და „გადავწიოთ“ ათვლის 40 წერტილით, ხოლო echo_3–ის – გავამრავლოთ 0.1–ზე და „გადავწიოთ“ 75 წერტილით.

M–ფაილი, რომელიც სკრიპტს წარმოადგენს, ასე გამოიყურება:

```
%          This program generates tree echoes from an  original signal
%          The sum of the original signal and the three  echoes
%          is stored in a data file and then plotted
%
t=0.0:0.1:10;
s=zeros(size(t));
s(1:8)=[0 0.5 1 -0.5 0.75 0 -0.2 -0.1];
%
%          Generation three echoes by scaling and  delaiing  original signal
%
echo_1=zeros(size(t));
echo_1(21:101)=0.5*s(1:81);
echo_2=zeros(size(t));
echo_2(41:101)=-0.3*s(1:61);
echo_3=zeros(size(t));
echo_3(76:101)=0.1*s(1:26);
%
%          Adding echoes to original signal
%          Saving new signal.
%
g=s+echo_1+echo_2+echo_3;
save echo3 t g;
%
%          Plotting new signal.
%
plot(t,g),...
    title('signal with echoes'),...
xlabel('t, seconds'),...
```

```
ylabel('g'),...
grid
```

მიღებული სიგნალი მოყვანილია ნახ.2–ზე.



ნახ.2 ძირითადი სიგნალი და სამი ექო

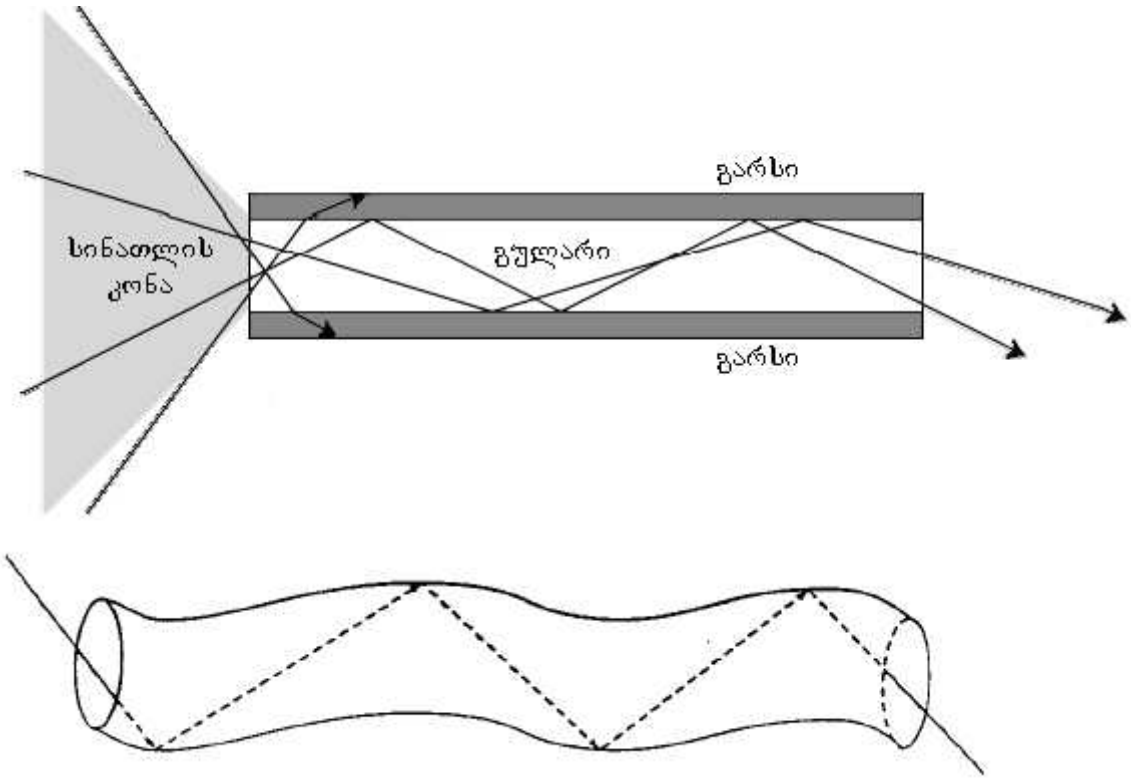
დანართი 2

ოპტიკურ ბოჭკოში სინათლის გავლის პირობების განსაზღვრა

ოპტიკური ბოჭკო გამჭვირვალე მინის ძაფია, რომელიც უფრო წვრილია, ვიდრე ადამიანის თმა, მაგრამ მას შეუძლია გაატაროს უფრო მეტი ინფორმაცია, ვიდრე რადიოტალღებმა სპინელძის სატელეფონო მავთულში. გარდა ამისა, ოპტიკურ–ბოჭკოვანი საკომუნიკაციო სიგნალი დაცულია ელექტრომაგნიტური ტალღებისაგან, რომელიც იწვევს ხმაურს და ამახინჯებს სიგნალს.

ოპტიკური ბოჭკო ამჟამად წარმოადგენს ინფორმაციის გადაცემის ყველაზე სრულყოფილ საშუალებას. კავშირგაბმულობის ოპტიკური სისტემები გამოირჩევა:

- ფართოზოლოვნებით (ერთი ბოჭკოს მეშვეობით შეიძლება ერთდროულად გადაიცეს 10 მილიონი სატელეფონო საუბარი და მილიონი ვოდეოსიგნალი);
- სიგნალის დაბალი მილევით;
- ბოჭკოს მასალის (კვარცის) სიიაფით;
- კომპაქტურობით, სიმსუბუქით და ექსპლუატაციის ხანგრძლივობით;
- დაბრკოლებამდგრადობით ელექტრომაგნიტური ველების მიმართ;
- მდგრადობით არასანქციონირებული შეღწევის მიმართ.



ნახ.1 ოპტიკური ბოჭკო

ოპტიკურ–ბოჭკოვანი კავშირის უარყოფით მხარეს წარმოადგენს ბოჭკოს შეპირაპირების უზრუნველმყოფელი ძვირი დანადგარები, თუმცა უპირატესობანი იმდენად მნიშვნელოვანია, რომ აღნიშნული ნაკლი შეიძლება უმნიშვნელოდ ჩაითვალოს.

ოპტიკური ბოჭკო შედგება განსხვავებული ოპტიკური თვისებების მქონე გარსისა და გულარისაგან.

თუ სინათლე ეცემა მინის ან პლასტიკის გრძელ ღეროს, იგი მრავალჯერ აირეკლება ღეროს კედლებიდან, ვიდრე მეორე ბოლოს მიაღწევს, როგორც ნახაზზეა ნაჩვენები.

ბოჭკოებში შიდა არეკვლის ფენომენი აიხსნება შნელის კანონით. ჩვეულებრივ, გულარის მასალა ოპტიკურად უფრო მკვრივია, ვიდრე გარსი. როცა სინათლე ერთი გარემოდან განსხვავებული სიმკვრივის მქონე მეორე გარემოში გადადის, იგი გარდატყდება გამყოფ ზედაპირზე. გარდატეხის კუთხე დამოკიდებულია გარემოს გარდატეხის კოეფიციენტზე და დაცემის კუთხეზე. როცა სინათლე უფრო მკვრივიდან ნაკლებ მკვრივ გარემოში გადადის, დაცემა რა ზედაპირს, მისი ნაწილი აირეკლება, ნაწილი კი გააღწევს მეორე გარემოში. სინათლის დაცემის კუთხეს, როცა იგი მთლიანად აირეკლება ზედაპირიდან, სრული არეკვლის კრიტიკული კუთხე ეწოდება. რადგანაც კრიტიკული კუთხე დამოკიდებულია ერთი გარემოს მეორის მიმართ გარდატეხის კოეფიციენტზე, შეგვიძლია გამოვითვალოთ ეს კუთხე და განვსაზღვროთ, მოცემული კუთხით დაცემული სინათლე გაივლის თუ არა ბოჭკოში.

დავუშვათ, n_2 გარსის გარდატეხის კოეფიციენტია, ხოლო n_1 – გულარის. თუ $n_2 > n_1$, ბოჭკო სინათლეს არ გაატარებს. კრიტიკული კუთხე განისაზღვრება ფორმულით:

$$\sin^{-}{}_{c} = \frac{n_2}{n_1}$$

მაგალითად, ჰაერის გარდატეხის კოეფიციენტია 1.0003, მინის – 1.5. ამ შემთხვევაში კრიტიკული კუთხე შეიძლება გამოვითვალოთ შემდეგნაირად:

$$^{-}{}_{c} = \arcsin\frac{n_2}{n_1} = \arcsin\frac{1.0003}{1.5} = \arcsin 0.66687 = 41.82^{0}$$

ასეთი გამტარი გაატარებს სინათლეს, რომელიც დაეცემა 41.82^{0}–ზე მეტი კუთხით.

დავწეროთ პროგრამა, რომელიც განსაზღვრავს, გაივლის თუ არა სინათლე ოპტიკურ ბოჭკოში, რომელიც გარემოცულია განსხვავებული სიმკვრივის მქონე მასალით.

დავუშვათ, გვაქვს ASCII მონაცემთა ფაილი სახელით indices.dat. ფაილის ყოველი სტრიქონი შეიცავს გულარისა და გარსის მასალის გარდატეხის კოეფიციენტების მნიშვნელობებს. პროგრამამ უნდა განსაზღვროს, გაატარებს თუ არა სინათლეს ასეთი მონაცემების მქონე ოპტიკური ბოჭკო და თუ გაატარებს, როგორია კრიტიკული კუთხის მნიშვნელობა.

ქვემოთ მოყვანილია პროგრამა–სკრიპტი:

```
%
% This program reads the indices of  refraction  for
% materials forming a light pipe.
% For each  pair of materials we determine if the pipe
```

```
%  will transmit light and at what
%  angles of  incidence in degrees.
%
factor = 180/pi;
load indices.dat
n1 = indices(:,1);
n2 = indices(:,2);
for k = 1: length(n1)
  if  n2(k) > n1(k)
    fprintf('light is not transmitted for \n')
    fprintf('rod index %g and medium index %g \n\n',n1(k), n2(k))
  else
    critical_angle = asin(n2(k)/n1(k))*factor;
    fprintf('light is not transmitted for \n')
    fprintf('rod index %g and medium index %g \n',n1(k), n2(k))
    fprintf('for angles less than %g degrees \n\n',critical_angle)
  end
end
```

შევინახოთ M–ფაილი opt_boWkos_krit_kuTxe სახელით.

შევქმნათ indices.dat ფაილი შემდეგი მონაცემებით:

```
1.31      1.473
1.5       1.0003
1.49      1.33
```

ბრძანებათა ფანჯარაში მივცეთ ბრძანება

```
>> opt_bowkos_krit_kuTxe
```

შედეგად მივიღებთ:

```
light is not transmitted for
rod index 1.31 and medium index 1.473

light is not transmitted for
rod index 1.5 and medium index 1.0003
for angles less than 41.8257 degrees

light is not transmitted for
rod index 1.49 and medium index 1.33
for angles less than 63.204 degrees
```

დანართი 3

გრაფის წვეროებს შორის უმოკლესი გზის განსაზღვრა

მონაცემების გადაცემისას მნიშვნელოვანია უმოკლესი გზის განსაზღვრა. ზოგადად უმოკლესი გზის პოვნის ამოცანა ძალზე აქტუალურია არა მარტო კავშირგაბმულობაში, არამედ მრავალ სხვა დარგშიც (მაგ., ტრანსპორტში).

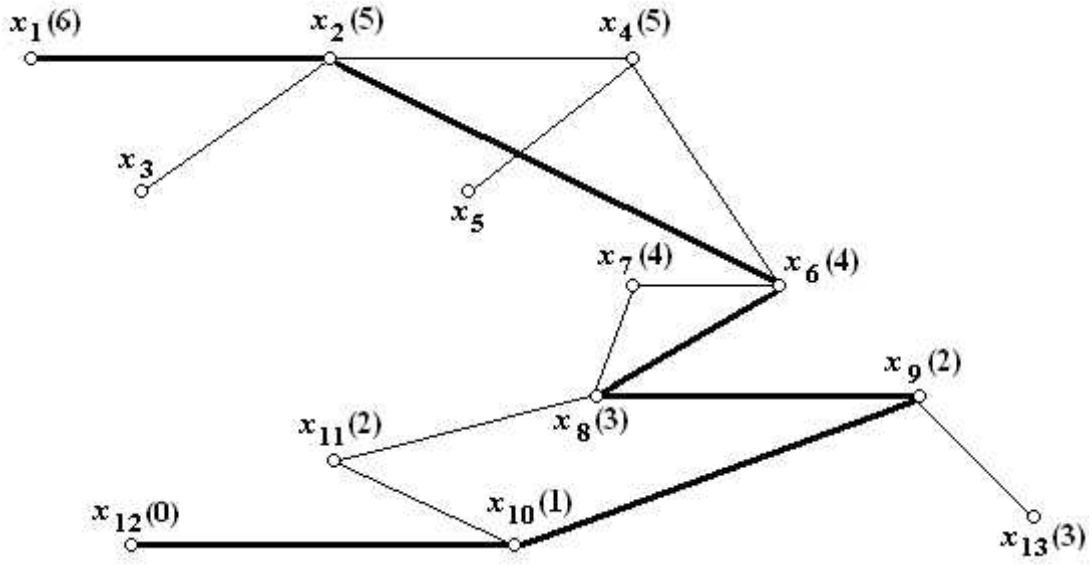
განვიხილოთ არაორიენტირებულ გრაფში უმოკლესი გზის პოვნის ალგორითმი.

ჩავთვალოთ, რომ გრაფის წიბოები ერთეულოვანი სიგრძისაა. ამ შემთხვევაში არაორიენტირებულ გრაფში (ორგრაფისათვის პროცესი ანალოგიურია) უმოკლესი გზის მოძებნის წესი შემდეგში მდგომარეობს: გრაფის თითოეულ წვეროს მიეწერება ინდექსი, რომელიც გამოსახავს მოცემული წვეროდან საბოლოო წვერომდე უმოკლესი გზის სიგრძეს. წვეროებისათვის ინდექსების მიწერა შემდეგი თანმიმდევრობით ხდება:

1. საბოლოო წვეროს (აღვნიშნოთ ის $X_{საბ}$–ით) მიეწერება ინდექსი 0, ხოლო ყველა წვეროს, რომელიც წიბოთი უშუალოდ უკავშირდება $X_{საბ}$–ს, – ინდექსი 1;
2. ყველა წვეროს, რომელსაც არ გააჩნია ინდექსი და წიბოთი უკავშირდება მიმდინარე ინდექსის მქონე წვეროს, მიეწერება ინდექსი, რომელიც 1–ით აღემატება მიმდინარეს. პროცესი გრძელდება მანამ, სანამ საწყის წვეროსაც ($X_{საწყ}$) არ მიეწერება ინდექსი. ეს უკანასკნელი კი უმოკლესი გზის სიგრძის ტოლი იქნება. თვით უმოკლესი გზა რომ განვსაზღვროთ, საჭიროა ვიმოძრაოთ საწყისი წვეროდან საბოლოო წვერომდე ინდექსების კლების მიმართულებით.

ზემოაღნიშნული პროცესის საილუსტრაციოდ ნახ.1–ზე ნაჩვენებია არაორიენტირებულ გრაფში უმოკლესი გზის მოძებნა $X_{საწყ}$ და $X_{საბ}$ წვეროებს შორის. ნახაზზე მოცემულ გრაფს შემდეგი შეერთებების მატრიცა შეესაბამება (შეერთებების მატრიცის ელემენტი უდრის 1–ს, როდესაც წვეროები დაკავშირებულია, წინააღმდეგ შემთხვევაში – 0–ს):

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
x_1	0	1	0	0	0	0	0	0	0	0	0	0	0
x_2	1	0	1	1	0	1	0	0	0	0	0	0	0
x_3	0	1	0	0	0	0	0	0	0	0	0	0	0
x_4	0	1	0	0	1	1	0	0	0	0	0	0	0
x_5	0	0	0	1	0	0	0	0	0	0	0	0	0
x_6	0	1	0	1	0	0	1	1	0	0	0	0	0
x_7	0	0	0	0	0	1	0	1	0	0	0	0	0
x_8	0	0	0	0	0	1	1	0	1	0	1	0	0
x_9	0	0	0	0	0	0	0	1	0	1	0	0	1
x_{10}	0	0	0	0	0	0	0	0	1	0	1	1	0
x_{11}	0	0	0	0	0	0	0	1	0	1	0	0	0
x_{12}	0	0	0	0	0	0	0	0	0	1	0	0	0
x_{13}	0	0	0	0	0	0	0	0	1	0	0	0	0



ნახ.1 x_1 და x_{12} წვეროებს შორის უმოკლესი გზის განსაზღვრა

ქვემოთ მოყვანილია აღწერილი მეთოდის საფუძველზე დაწერილი პროგრამა–ფუნქცია. პროგრამის საწყის მონაცემებად აღებულია გრაფის შეერთებების მატრიცა (sheert) და საწყისი და საბოლოო წვეროების ნომრები (xsawy, xsab).

```
function [sigrdze gza]=gzis_povna(sheert,xsawy,xsab)
%    araorientirebul grafshi wveroebs Soris umoklesi gzis da
% gzis sigrZis gansazRvra (grafis yvela wibo erTeulovani sigrZisaa)
% sheert - grafis sheertebebis matrica;
% xsawy - sawyisi wveros nomeri; xsab - saboloo wveros nomeri
%
n=length(sheert);
if xsawy==xsab
    sigrdze=0; gza=[xsawy];
    'sawyisi da saboloo wvero emTxveva erTmaneTs'
else
    [index sigrdze]=indeqsacia(sheert,xsawy,xsab,n);
    gza=[xsawy];
    for i=index(xsawy):-1:1
        for j=1:n
            if index(j)==i-1
                gza=[gza j];
                break
            end
        end
    end
end
%
function [index sigrdze]=indeqsacia(sheert,xsawy,xsab,n)
% wveroebis indeqsaciis programa
%
index=-1*ones(1,n);
index(xsab)=0;
```

```
ind=0;
k=xsab;
while k~=xsawy
    sia=[];
    for i=1:n
        if index(i)==ind
            sia=[sia i];
        end
    end
    m=length(sia);
    for i=1:m
        for j=1:n
            if sheert(sia(i),j)==1&index(j)==-1
                index(j)=ind+1;
                if j==xsawy
                     sigrdze=index(j);
                     k=xsawy;
                     return
                end
            end
        end
    end
    ind=ind+1;
end
```

ბრძანებათა ფანჯარაში შევიტანოთ საწყისი ინფორმაცია და მივმართოთ gzis_povna ფუნქციას:

```
>> sheert=[0 1 0 0 0 0 0 0 0 0 0 0 0; 1 0 1 1 0 1 0 0 0 0 0 0 0;
0 1 0 0 0 0 0 0 0 0 0 0 0; 0 1 0 0 1 1 0 0 0 0 0 0 0;
0 0 0 1 0 0 0 0 0 0 0 0 0; 0 1 0 1 0 0 1 1 0 0 0 0 0;
0 0 0 0 0 1 0 1 0 0 0 0 0; 0 0 0 0 0 1 1 0 1 0 1 0 0;
0 0 0 0 0 0 0 1 0 1 0 0 1; 0 0 0 0 0 0 0 0 1 0 1 1 0;
0 0 0 0 0 0 0 1 0 1 0 0 0; 0 0 0 0 0 0 0 0 0 1 0 0 0;
0 0 0 0 0 0 0 0 1 0 0 0 0];
>> [sigrdze gza]=gzis_povna(sheert,1,12)
```

მივიღებთ პასუხს:

```
sigrdze =
   6
gza =
   1    2    6    8    9   10   12
```

გრაფში უმოკლესი გზის განსაზღვრის ამოცანა რთულდება, თუ გრაფს აქვს ნებისმიერი სიგრძის წიბოები (რკალები). ამ ამოცანის ამოხსნისათვის არსებობს სხვადასხვა ალგორითმი. განვიხილოთ ერთ–ერთი მათგანი – ე.წ. დანციგის ალგორითმი:

1. მივაწეროთ საწყის წვეროს ნულის ტოლი ინდექსი: $^{a}(x_{საწყ}) = {}^{a}_{0} = 0$.

2. Q–თი აღვნიშნოთ იმ არაინდექსირებული წვეროების სიმრავლე, რომლებიც წარმოადგენენ ინდექსის მქონე წვეროების ასახვების გაერთიანებას (ანუ ამ წვეროებში

არსებობს ინდექსირებული წვეროებიდან გზა ერთი წიბოს (რკალის) გავლით). ყველა $x_i \in Q$ წვეროსათვის გამოვთვალოთ α_i მახასიათებლის მნიშვნელობა:

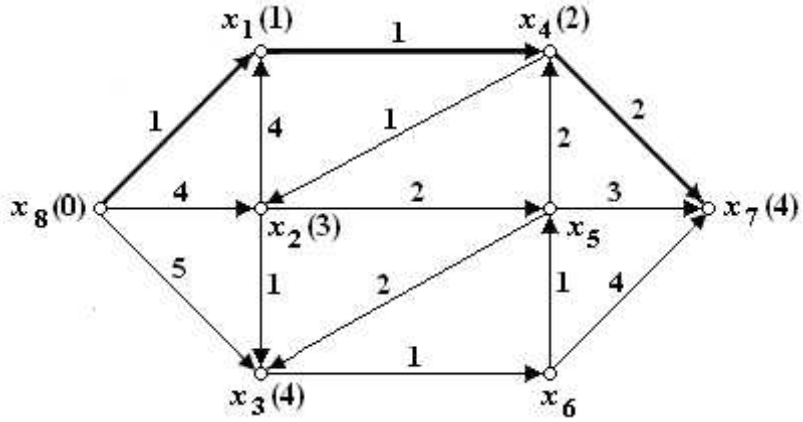
$$\mu_i = \min_j (\lambda(x_j) + d_{ji}),$$

სადაც მინიმუმს ვეძებთ ინდექსის მქონე ყველა x_j წვეროს გათვალისწინებით, რომლებიც x_i წვეროს უკუასახვის სიმრავლეს მიეკუთვნება. ის x_i წვერო, რომელსაც შეესაბამება მინიმალური α_i, აღვნიშნოთ ინდექსით α_i: $a(x_i) = \alpha_i$.

ეს პროცედურა მეორდება მანამ, სანამ ინდექსი არ მიეწერება საბოლოო წვეროს $x_{საბ}$. თუ რომელიმე ეტაპზე $Q = \varnothing$, ეს ნიშნავს, რომ $x_{საწყ}$ და $x_{საბ}$–ს შორის გზა არ არსებობს და შესაბამისი მანძილი უსასრულობას უდრის.

3. უმოკლესი გზის სიგრძე $a(x_{საბ})$ ინდექსის მნიშვნელობის ტოლია, ხოლო გზა შეიძლება განვსაზღვროთ, თუ $x_{საბ}$–დან $x_{საწყ}$–კენ გადავინაცვლებით ინდექსების კლების მიმართულებით შემდეგი წესის დაცვით: $a(x_j) - a(x_i) = d_{ij}$.

განვიხილოთ დანციგის ალგორითმის მუშაობა ნახ.2–ზე ნაჩვენები ორგრაფის მაგალითზე (არაორგრაფისათვის პროცესი ანალოგიურია). რკალების ზემოთ მითითებული რიცხვი მათ სიგრძეს აღნიშნავს. თუ რკალის სიგრძე მთელი რიცხვი არ არის, რა თქმა უნდა, ინდექსიც მთელი არ იქნება. ინდექსები მითითებულია ფრჩხილებში.



ნახ.2 დანციგის ალგორითმით უმოკლესი გზის განსაზღვრა x_8 და x_7 წვეროებს შორის

მოვძებნოთ გზა x_8 ($x_{საწყ}$) და x_7 ($x_{საბ}$) წვეროებს შორის.

<u>I ბიჯი</u>: $a(x_8) = 0$; $Q = \{x_1, x_2, x_3\}$.

გამოვთვალოთ α_i–ის მნიშვნელობები:

$\alpha_1 = a_0 + d_{81} = 0+1=1$; $\alpha_2 = a_0 + d_{82} = 0+4=4$; $\alpha_3 = a_0 + d_{83} = 0+5=5$.

« მახასიათებლის მინიმალური მნიშვნელობა x_1 წვეროს აქვს. ამ წვეროს min «$_i$–ის (1–ის) ტოლი ინდექსი მივაწეროთ.

II ბიჯი: $Q = \{x_2, x_3, x_4\}$.

«$_2$ = min($a_0 + d_{82}$)=0+4=4; «$_3$ = min($a_0 + d_{83}$)=0+5=5; «$_4$ = min($a(x_1) + d_{14}$)=1+1=2 .

x_4 წვეროს მივაწეროთ ინდექსი 2.

III ბიჯი: $Q = \{x_2, x_3, x_7\}$.

$$«_2 = \min\begin{Bmatrix} a_0 + d_{82} \\ a(x_4) + d_{42} \end{Bmatrix} = \min\begin{Bmatrix} 0+4 \\ 2+1 \end{Bmatrix} = 3;$$

«$_3$ = min($a_0 + d_{83}$)=0+5=5; «$_7$ = min($a(x_4) + d_{47}$)=2+2=4 .

x_2 წვეროს მივაწეროთ ინდექსი 3.

IV ბიჯი: $Q = \{x_3, x_5, x_7\}$.

$$«_3 = \min\begin{Bmatrix} a_0 + d_{83} \\ a(x_2) + d_{23} \end{Bmatrix} = \min\begin{Bmatrix} 0+5 \\ 3+1 \end{Bmatrix} = 4;$$

«$_5$ = min($a(x_2) + d_{25}$)=3+2=5; «$_7$ = min($a(x_4) + d_{47}$)=2+2=4 .

x_3 და x_7 წვეროებს მივაწეროთ ინდექსი 4.

რადგან საბოლოო x_7 წვეროს უკვე აქვს ინდექსი, პროცედურა დამთავრებულია. უმოკლესი გზის სიგრძე უდრის ოთხს, თვითონ გზა არის x_8 x_1 x_4 x_7.

ქვემოთ მოყვანილია აღწერილი მეთოდის საფუძველზე დაწერილი პროგრამა–ფუნქცია. პროგრამის საწყის მონაცემებად აღებულია გრაფის მანძილების მატრიცის (xdmatr) ნაირსახეობა (xdmatr(i, j) უდრის მანძილს i და j წვეროებს შორის, როდესაც არსებობს გზა i–ური წვეროდან j–ურ წვეროში ერთი რკალის გავლით, წინააღმდეგ შემთხვევაში – 0–ს) და აგრეთვე საწყისი და საბოლოო წვეროების ნომრები (xsawy, xsab).

```
function [sigrdze_danc gza_danc]=dancigis_algoriTmi(xdmatr,xsawy,xsab)
%   nebismieri sigrZis SeerTebebis mqone grafSi wveroebs Soris
% umoklesi gzis da gzis sigrZis gansazRvra dancigis algoriTmiT
% xdmatr - grafis mandzilebis matrica;
% xsawy - sawyisi wveros nomeri; xsab - saboloo wveros nomeri
% sigrdze_danc - gzis sigrZe; gza_danc - gza;
%
n=length(xdmatr);
if xsawy==xsab
    sigrdze_danc=0; gza_danc=[xsawy];
    'sawyisi da saboloo wvero emTxveva erTmaneTs'
else
    % siawver - indeqsis mqone wveroebis sia
    % ind - indeqsebis mniSvnelobebi
```

```
siawver=[xsawy];
ind=length(siawver); ind(xsawy)=0;
% ww cvladi gansaqRvravs, miRweulia Tu ara saboloo wvero:
%   rodesac ww=1, indeqsaciis procesi damTavrebulia
ww=0;
while ww==0
    %     indeqsis mqone wveroebTan SeerTebuli
    % araindeqsirebuli wveroebis Q veqtoris formireba
    Q=[];
    for j=1:length(siawver)
        vsp=find(xdmatr(siawver(j),:));
        for k=1:length(siawver)
            vr=vsp(find(vsp~=siawver(k)));
            vsp=vr;
        end
        for k=1:length(Q)
            vr=vsp(find(vsp~=Q(k)));
            vsp=vr;
        end
        Q=[Q vsp];
    end
    if length(Q)==0
        'miTiTebul wveroebs Soris gza ar arsebobs'
        sigrdze_danc=Inf;
        gza_danc=Inf;
        return
    else
        % Q-Si Semavali wveroebisaTvis miu maxasiaTeblis gansazRvra
        miu=[];
        for i1=1:length(Q)
            A=[];
            for j1=1:length(siawver)
                if xdmatr(siawver(j1),Q(i1))>0
                    A=[A ind(siawver(j1))+xdmatr(siawver(j1),Q(i1))];
                end
            end
            miu(i1)=min(A);
        end
        lambda=min(miu);
        for i2=1:length(Q)
            if miu(i2)==lambda
                siawver=[siawver Q(i2)];
                ind(Q(i2))=lambda;
                if Q(i2)==xsab
                    ww=1;
                end
            end
        end
    end
end
sigrdze_danc=ind(xsab);
% indeqsebis dalageba zrdadobiT da Sesabamisad
% siawver veqtoris mowesrigeba
```

```
    [ind siawver]=sort(ind);
    % xsawy da xsab Soris gzis gansazRvra
    gza_danc=[xsab];
    m=length(siawver);
    R=xsab;
    while R~=xsawy
        for k=m-1:-1:1
            if ind(m)-ind(k)>0&ind(m)-ind(k)==xdmatr(siawver(k),siawver(m))
                gza_danc=[gza_danc siawver(k)];
                break
            end
        end
        R=siawver(k);
        m=k;
    end
    m=length(gza_danc);
    for i=1:m/2
        c=gza_danc(i);
        gza_danc(i)=gza_danc(m+1-i);
        gza_danc(m+1-i)=c;
    end
end
```

ბრძანებათა ფანჯარაში შევიტანოთ ნახ.2–ის შესაბამისი საწყისი ინფორმაცია და მივმართოთ dancigis_algoriTmi ფუნქციას:

```
>> X=[0 0 0 1 0 0 0 0;
      4 0 1 0 2 0 0 0;
      0 0 0 0 0 1 0 0;
      0 1 0 0 0 0 2 0;
      0 0 2 2 0 0 3 0;
      0 0 0 0 1 0 4 0;
      0 0 0 0 0 0 0 0;
      1 4 5 0 0 0 0 0];
[sigrdze_danc  gza_danc]=dancigis_algoriTmi(X,8,7)
```

მივიღებთ პასუხს:

```
sigrdze_danc =
     4
gza_danc =
    8     1     4     7
```

მე–2 მაგალითი ასახავს სიტუაციას, როდესაც $x_{საწ}$ და $x_{საბ}$ წვეროებს შორის გზა არ არსებობს (მაგ., x_7 და x_8 წვეროებს შორის – იხ. ნახ.2). მაშინ მიღებული პასუხი იქნება:

```
ans =
miTiTebul wveroebs Soris gza ar arsebobs
sigrdze_danc =
   Inf
gza_danc =
     Inf
```

დანართი 4

კომპონირების ამოცანის ამოხსნა მაქსიმალური კონიუნქციის – მინიმალური დიზიუნქციის მეთოდით

კომპონირების ამოცანის (სქემის ნაწილებად გაყოფის) ამოხსნის ერთ–ერთ მიმდევრობით მეთოდს წარმოადგენს მაქსიმალური კონიუნქციის – მინიმალური დიზიუნქციის (მაქსიმალური შიგა კავშირების – მინიმალური გარე კავშირების) მეთოდი. ამ მეთოდით კომპონირებისას საწყისი სქემიდან ელემენტების (მოდულების) ჯგუფის გამოყოფა ხდება კონიუნქციის და დიზიუნქციის ოპერაციების განხორციელებით.

ვთქვათ, მოცემული სქემა წარმოდგენილია ელემენტური კომპლექსების მატრიცით – ეს არის $n \times m$ განზომილების მატრიცა (n – ელემენტების (მოდულების) რაოდენობაა, m – კავშირების (წრედების) რიცხვი), რომლის q_{ij} ელემენტი უდრის 1–ს, თუ j–ური წრედი შეერთებულია i–ურ ელემენტთან, წინააღმდეგ შემთხვევაში – 0–ს. სქემის გარე გამომყვანებს ერთ ან რამდენიმე ფიქტიურ მოდულს მიაკუთვნებენ. მოყვანილ პროგრამაში ფიქტიური მოდული ერთია.

მორიგი i–ური $(i = 1, 2, \ldots)$ კვანძის (ნაწილის) ფორმირება იწყება საბაზო z_i^* ელემენტის არჩევით გაუნაწილებელი ელემენტების Z_i სიმრავლიდან. პროცესის დასაწყისში ყველა ელემენტი ითვლება გაუნაწილებლად, ე.ი. გაუნაწილებელი ელემენტების სიმრავლე $Z_i = E$ (ფიქტიური მოდული $e_{n+1} \notin E$).

$x \in Z_i$ ელემენტებისათვის გამოითვლება ფუნქციონალი

$$L_1(x) = \left| x \cap Z_i^{'} \right|,$$

რომელიც განსაზღვრავს განსახილველი x ელემენტის კავშირების რაოდენობას დანარჩენ გაუნაწილებელ ელემენტებთან, ე.ი. ელემენტებთან $Z_i^{'} = Z_i \setminus x$ სიმრავლიდან.

საბაზო z_i^* ელემენტად ითვლება რიგით პირველი ელემენტი Z_i–დან, რომლისთვისაც L_1 ფუნქციონალი იღებს მაქსიმალურ მნიშვნელობას. z_i^* ელემენტი თავსდება i–ურ კვანძში, ხოლო დანარჩენი $Z_i \setminus z_i^*$ ელემენტები გვევლინება E_i სიმრავლეში შესაყვან კანდიდატებად ალგორითმის მომდევნო ბიჯებზე (E_i – i–ური ნაწილის ელემენტების სიმრავლეა).

კვანძის მომდევნო ფორმირება ხდება $L_2(x)$ და $L_3(x)$ ფუნქციონალების საშუალებით.

ვთქვათ, i–ურ კვანძში უკვე მოთავსებულია k ელემენტი: $E_i^{a} = \{z_i^*, e_j, e_k, \ldots\}$. $L_2(x)$ ფუნქციონალი მოიცემა მოცემული მომენტისათვის გაუნაწილებელი ელემენტების $x \in Z_i^{a}$

$(Z_i^{a} = Z_i \setminus E_i^{a})$ სიმრავლეზე და განსაზღვრავს გარე გამომყვანების რიცხვს $i-$ური კვანძისათვის, რომელიც წარმოადგენს $i-$ურ კვანძს $x \in Z_i^{a}$ ელემენტის დამატებით:

$$L_2(x) = \left| E_i^{\lambda}(x) \cap \left\{ \bigcup_{s=0}^{i-1} E_s \cup Z_i^{''} \right\} \right| ,$$

სადაც $Z_i^{''}$ არის წრედები, რომლებიც დაკავშირებულია Z_i^{a} სიმრავლის ელემენტებთან x ელემენტის გამორიცხვით, ხოლო $E_0 = \{e_{n+1}\}$.

$L_3(x)$ ფუნქციონალი მოიცემა გაუნაწილებელი ელემენტების Z_i^{a} სიმრავლეზე იმ ელემენტების გამორიცხვით, რომლებიც არ აკმაყოფილებენ პირობას $L_2(x) \le k$, სადაც k – კვანძის გამომყვანების, ანუ ნაწილის გარე კავშირების, მაქსიმალური დასაშვები რაოდენობაა. $L_3(x)$ ფუნქციონალი წარმოადგენს განსახილველი ელემენტის შიგა კავშირებს $E_i^{a}(x)$ სიმრავლეში შესულ ელემენტებთან:

$$L_3(x) = \left| x \cap E_i^{a} \right| .$$

კვანძში შევა ელემენტი, რომელსაც E_i^{a} სიმრავლესთან აქვს მაქსიმალური კონიუნქცია, ანუ რომელსაც შეესაბამება $L_3(x)-$ის მაქსიმალური მნიშვნელობა. თუ ასეთი ელემენტი რამდენიმეა, მათ შორის ამოირჩევა ის ელემენტი, რომლისთვისაც $L_2(x)$ ღებულობს მინიმალურ მნიშვნელობას. თუ მინიმალური დიზიუნქციის მქონე ელემენტიც რამდენიმეა, ამოირჩევა მინიმალური ინდექსის მქონე ელემენტი.

კვანძი დასრულებულად ითვლება, თუ კვანძში ელემენტების რაოდენობა მოცემული k რიცხვის ტოლია ანდა ნებისმიერი გაუნაწილებელი ელემენტის განაწილება კვანძში კვანძის გარე კავშირების რიცხვს ზრდის ისეთ მნიშვნელობამდე, რომელიც დასაშვებ v მნიშვნელობაზე მეტია.

განვიხილოთ ალგორითმის რეალიზაცია კონკრეტული მაგალითისათვის. ვთქვათ, მოცემულია შეერთებების სქემა შემდეგი Q მატრიცის სახით:

$$Q = \begin{vmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix}$$

საჭიროა მოხდეს სქემის კომპონირება იმ პირობით, რომ $k=3$ და $v=5$. ამოხსნა მოყვანილია ცხრილში.

i	Z_i	L_i	z_i^*	Z_i^1	L_2	L_3	E_i^2	Z_i^2	L_2	L_3	E_i^3
1	2	3	4	5	6	7	8	9	10	11	12
1	(e_1) e_2 e_3 e_4 e_5 e_6 e_7 e_8 e_9	3 3 3 3 3 2 2 3 2	e_1	(e_2) e_3 e_4 e_5 e_6 e_7 e_8 e_9	4 6 4 6 5 5 6 6	2 – 2 – 1 1 – –	e_1 e_2	e_3 (e_4) e_5 e_6 e_7 e_8 e_9	7 5 5 6 6 7 7	– 2 1 – – – –	e_1 e_2 e_4
2	(e_3) e_5 e_6 e_7 e_8 e_9	3 1 2 2 3 2	e_3	e_5 e_6 e_7 e_8 (e_9)	5 6 6 4 3	1 – – 2 2	e_3 e_9	e_5 e_6 e_7 (e_8)	5 6 6 3	1 – – 2	e_3 e_9 e_8
3	e_5 (e_6) e_7	0 2 2	e_6	e_5 (e_7)	6 4	– 2	e_6 e_7	e_5	7	–	e_6 e_7
4	e_5										e_5

ამგვარად, მივიღეთ შემდეგი დაყოფა:

$$E_1=\{e_1,e_2,e_4\}, E_2=\{e_3,e_8,e_9\}, E_3=\{e_6,e_7\}, E_4=\{e_5\}.$$

ქვემოთ მოყვანილია მაქსიმალური კონიუნქციის – მინიმალური დიზიუნქციის მეთოდის საფუძველზე დაწერილი პროგრამა–ფუნქცია:

```
function kv=kompon_mimd(Q, max_el_raod, max_gare_wib)
% komponirebis amocanis amoxsna maqsimaluri koniunqciis -
%        minimaluri diziunqciis meTodiT
% Q - elementuri kompleqsebis matrica; max_el_raod -
%    kvanZSi elementebis maqsimaluri raodenoba;
% max_gare_wib - gare kavSirebis maqsimaluri raodenba
```

```
%
[n m]=size(Q);
% Z0 - gaunawilebel elementTa sia
for i=1:n-1
      Z0(i)=i;
end
nn=length(Z0);
%  kv - yvela kvanZSi Sesuli elementebis sia
kv=[];
while nn>0
    for j=1:max_el_raod
        dam(j)=0;
    end
    kvanZi=[];
    L1= L1_gamoTvla(Q,Z0,nn);
    [maximal  nom]=max_povna(L1);
    kvanZi=[kvanZi  Z0(nom)];
    Z0(nom)=[];
    nn=nn-1;
    for el_raod=2: max_el_raod
        [L2 L3]=L2_da_L3_gamoTvla(Q,Z0,kvanZi,nn,max_gare_wib);
        fff=0;
        for t=1:length(L2)
           if L2(t)> max_gare_wib
               fff=fff+1;
           end
        end
        if fff==length(L2)
            break
        else
            nom_el=el_moZebna(L2,L3);
            kvanZi=[kvanZi  Z0(nom_el)];
            Z0(nom_el)=[];
            nn=nn-1;
        end
    end
    for j=1:length(kvanZi)
        dam(j)=kvanZi(j);
    end
    kv=[kv;dam];
end
'kvanZebSi elementebis ganawileba:'
%
function L1=L1_gamoTvla(Q,Z0,nn)
% L1-is gamoTvla: L1 - elementis kavSirebis raodenoba danarCen
% gaunawilebel elementebTan
% Z0 - gaunawilebeli elementebis sia
% nn - gaunaWilebeli elementebis raodenoba
%
```

```
[n m]=size(Q);
for t=1:nn
VSP(t,:)=Q(Z0(t),:);
% VSP – damxmare matrica, warmoadgens Q-s, ganawilebuli
%       elementebis striqonebis gamoricxviT
end
L1=zeros(1,nn);
for i=1:nn
    Y=VSP; VSP(i,:)=[];
    for j=1:m
        if Q(Z0(i),j)==1&any(VSP(:,j))==1
            L1(i)=L1(i)+1;
         end
    end
    VSP=Y;
end
%
function [maximal nom]=max_povna(A)
% veqtoris maximaluri elementis da misi nomeris povna
n=length(A);
maximal=A(1); nom=1;
for i=1:n
    if A(i)>maximal
        maximal=A(i); nom=i;
    end
end
%
function [L2 L3]=L2_da_L3_gamoTvla(Q,Z0,kvanZi,nn,max_gare_wib)
% L2-is gamoTvla: L2 - gare kavsirebis ricxvi;
% L3-is gamoTvla: L3 - Sida kavsirebis ricxvi; rodesac i-uri
% elementis L2(i)>max_gare_wib, CavTvaloT L3(i)=-1;
%
% Q - elementuri kompleqsebis matrica;
% Z0 - - gaunawilebeli elementebis sia;
% kvanZi - mimdinare kvanzSi Sesuli elementebis sia;
% nn - gaunawilebeli elementebis raodenoba;
% max_gare_wib - kvanZis gare kavsirebis maqsimaluri dasasvebi ricxvi,
[n m]=size(Q);
% VSP-s gamoTvla: VSP – damxmare matrica, warmoadgens Q-s, ganawilebuli
%  elementebis striqonebis gamoricxviT, masSi Sedis fiqtiuri modulic;
% damx-s gamoTvla: damx – damxmare matrica, warmoadgens Q-s striqonebs,
%   romlebic mimdinare kvanZSi ganawilebul elementebs Seesabameba;
% E - yvela elementis sia
for i=1:n
    E(i)=i;
end
for i=1:length(kvanZi)
    damx(i,:)=Q(kvanZi(i),:);
end
```

```
sia_VSP=simr_gamokleba(E,kvanZi);
for i=1:length(sia_VSP)
    VSP(i,:)=Q(sia_VSP(i),:);
end
L2=zeros(1,nn); L3=zeros(1,nn); Y1=VSP;
for i=1:nn
    Y=Q(Z0(i),:); Z=damx; damx=[damx;Y];
    for j=1:length(sia_VSP)
        if Z0(i)==sia_VSP(j)
            number=j;
            break
        end
    end
    VSP(number,:)=[];
    for j=1:m
        if any(damx(:,j))==1&any(VSP(:,j))==1
           L2(i)=L2(i)+1;
        end
    end
    if L2(i)>max_gare_wib
        L3(i)=-1;
    else
        L3(i)=0;
        for j=1:m
            if Y(j)==1&any(Z(:,j))==1
               L3(i)=L3(i)+1;
            end
        end
    end
    VSP=Y1;
    damx=Z;
end
%
function nom_el=el_moZebna(L2,L3)
% max L2-is da min L3-is mqone elementis serCeva
n=length(L2);
n1=[];
maxL3=max(L3);
for i=1:n
    if L3(i)==max(L3)
        n1=[n1 i];
    end
end
nom_el=n1(1);
if length (n1)>1
    minL2=L2(n1(1)); nom_el=n1(1);
    for j=2:length(n1)
        if L2(n1(j))<minL2
            nom_el=(n1(j));
```

```
        end
    end
end
%
function  s_gam=simr_gamokleba(A,B)
% A  simravlidan  B simravlis gamokleba
n1=length(A); n2=length(B); s_gam=[];
for  i=1:n1
    k=0;
    for j=1:n2
        if A(i)~=B(j)
            k=k+1;
        end
    end
    if k==n2
        s_gam=[s_gam  A(i)];
    end
end
```

ბრძანებათა ფანჯრიდან შევიტანოთ საწყისი ინფორმაცია (Q მატრიცა) და მივცეთ ბრძანება:

```
>> kv=kompon_mimd(Q, 3, 5)
```

მივიღებთ პასუხს:

```
ans =
kvanZebSi elementebis ganawileba:
kv =
     1     2     4
     3     9     8
     6     7     0
     5     0     0
```

`>> kv=kompon_mimd (Q, 5, 9)` ბრძანების შემთხვევაში მიღებული პასუხი იქნება:

```
ans =
kvanZebSi elementebis ganawileba:
kv =
     1     2     4     5     3
     6     7     8     9     0
```

აღვნიშნოთ, რომ `max_povna` ქვეფუნქციის დაწერა არ არის აუცილებელი, რადგან MATLAB–ს აქვს შესაბამისი ჩაშენებული ფუნქცია.

დანართი 5

საკომუტაციო ველზე ელემენტების განლაგების ამოცანის ამოხსნა ბმულობის მიხედვით – საწყისი განლაგების მიღება

განლაგების ამოცანა მდგომარეობს სიბრტყეზე ელემენტების ოპტიმალური მდგომარეობის შერჩევაში. საერთო ჯამში განლაგების ამოცანა შეიძლება ჩამოყალიბდეს შემდეგნაირად: სამონტაჟო სივრცეში მოცემულია არე, რომელიც იყოფა პოზიციების სიმრავლედ $P = \{p_1, p_2, ... p_q\}$, რომელთა რიცხვი განსალაგებელი ელემენტების რიცხვზე ნაკლები არ უნდა იყოს. ცხადია, რომ ყველა ელემენტი დაიკავებს მხოლოდ ერთ პოზიციას (დასამაგრებელ ადგილს), რომელთა ადგილმდებარეობა აღწერილია კოორდინატთა მატრიცის სახით. ელემენტების არსებული სიმრავლე $E = \{e_1, e_2, ... e_n\}$, რომელთა ერთმანეთთან დაკავშირება ასახულია მომიჯნავეობის R მატრიცით (მომიჯნავეობის მატრიცის r_{ij} ელემენტი i–ური და j–ური ელემენტებს შორის კავშირების რაოდენობის ტოლია), უნდა აისახოს P სიმრავლეში ისე, რომ უზრუნველყოფილი იქნას არჩეული მიზნობრივი ფუნქციის ექსტრემუმი. განლაგების ამოცანის ოპტიმალურობის მთავარი კრიტერიუმია შეერთებების სიგრძეების ჯამის მინიმიზაცია, რომელიც ითვალისწინებს ელემენტების განლაგებისა და მათი შეერთებების ტრასირებისადმი წაყენებულ მრავალრიცხოვან მოთხოვნას.

მანძილი დასამაგრებელ ადგილებს შორის გამოითვლება ევკლიდურ ან ორთოგონალურ მეტრიკაში:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

ან

$$d_{ij} = | x_i - x_j | + | y_i - y_j | \ .$$

საწყისი განლაგების მისაღებად გამოვიყენოთ ელემენტების ბმულობის მიხედვით ამოცანის ამოხსნის მეთოდი, რომელიც მიეკუთვნება მიმდევრობითი ალგორითმების ჯგუფს. საწყისი განლაგების მიღების პროცესი წარმოადგენს გადაწყვეტილებების მიღების ბიჯურ პროცესს, რომლის თითოეულ ბიჯზე ხდება ერთ–ერთი თავისუფალი ელემენტის არჩევა და მისი განლაგება თავისუფალ პოზიციაზე.

ვთქვათ, E_k არის იმ ელემენტების სიმრავლე, რომლებიც განლაგებულია k–ურ ბიჯამდე, ხოლო P_k – ამ ელემენტების პოზიციების სიმრავლე. $\overline{E_k}$ და $\overline{P_k}$ არის შესაბამისად თავისუფალი ელემენტებისა და თავისუფალი პოზიციების სიმრავლე.

k–ური ბიჯის დაწყებამდე შეიძლება ადგილი ჰქონდეს ორ სიტუაციას:

1. არ გვაქვს ადრე განლაგებული ან წინასწარ რომელიმე პოზიციაზე დამაგრებული ელემენტები;

2. გვაქვს წინასწარ რომელიმე პოზიციებზე დაფიქსირებული ელემენტები.

თუ საკომუტაციო ველზე არ არის არც ერთი ადრე განლაგებული ან წინასწარ დამაგრებული ელემენტი, გამოითვლება თითოეული ელემენტის შეერთებების ჯამური რიცხვი. საკომუტაციო ველის ცენტრალურ პოზიციაზე განლაგდება ელემენტი, რომლის შეერთებების ჯამი უდიდესია. ცენტრალურ პოზიციად ითვლება მინიმალური დაშორების მქონე პოზიცია, ე.ი. ისეთი, რომლისთვისაც $l_j = \sum_i d_{ij}$ მახასიათებლის მნიშვნელობა მინიმალურია. აქ d_{ij} მანძილია p_i და p_j პოზიციებს შორის. აღვნიშნოთ, რომ l_j შეიძლება აგრეთვე გამოვთვალოთ, როგორც $\max_i d_{ij}$.

განვიხილოთ მეორე სიტუაცია. ელემენტის არჩევის ნებისმიერი წესი ემყარება მისი კავშირების გამოთვლას უკვე განლაგებულ ელემენტებთან. ქვემოთ მოყვანილ პროგრამაში თითოეული თავისუფალი $e_j \in \overline{E_k}$ ელემენტისათვის გამოთვლილია c_j მახასიათებელი, რომელიც წარმოადგენს განსახილველი $e_j \in \overline{E_k}$ ელემენტის უკვე განაწილებულ $e_i \in E_k$ ელემენტებთან კავშირების მაქსიმალურ რაოდენობას:

$$c_j = \max_{e_i \in E_k} r_{ij} ,$$

სადაც r_{ij} – e_i და e_j ელემენტებს შორის კავშირების რაოდენობაა (c_j მახასიათებელი შეიძლება აგრეთვე გამოითვალოს როგორც თავისუფალი ელემენტის უკვე განაწილებულ ელემენტებთან ჯამური კავშირების რიცხვი). მორიგ ბიჯზე ხდება იმ ელემენტის არჩევა, რომლისთვისაც c_j მაქსიმალურია.

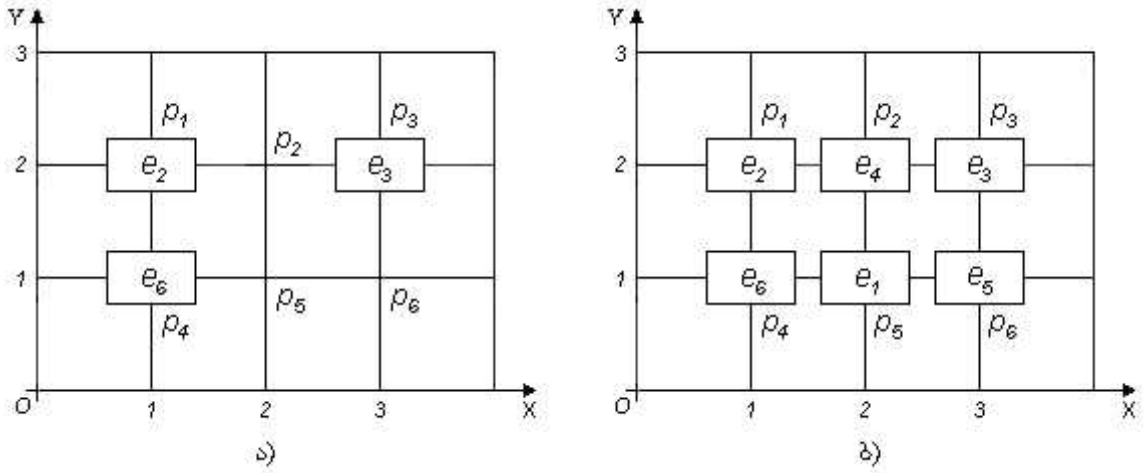
განლაგებისთვის ამორჩეული e_{i_0} ელემენტი უნდა მოთავსდეს ერთ–ერთ თავისუფალ პოზიციაზე $\overline{P_k}$ სიმრავლიდან. პოზიციის შერჩევა ხდება შეერთებების ჯამური სიგრძის გათვალისწინებით. კერძოდ, პოზიციის შესარჩევად ყველა $p_j \in \overline{P_k}$–თვის გამოითვლება ე.წ. ფსევდოსიგრძე:

$$F_j = \sum_{e_i \in E_k} r_{i_0 i} \cdot d_{p(e_{i_0})p(e_i)} ,$$

სადაც $p(e_{i_0}) = j$. ამ ფორმულით გამოთვლილი ფსევდოსიგრძე წარმოადგენს e_{i_0} ელემენტის იმ კავშირების ჯამურ სიგრძეს, რომლებიც e_{i_0}–ს აერთებენ საკომუტაციო ველზე უკვე მოთავსებულ ელემენტებთან. აირჩევა ის პოზიცია, რომლისთვისაც F_j მინიმალურია.

მაგალითი:

მოცემულია 6–ადგილიანი საკომუტაციო ველი (ნახ.1ა), რომელზეც უნდა მოვათავსოთ 6 ელემენტი. 3 მოდული (e_2, e_3, e_6) უკვე განლაგებულია შესაბამისად p_1, p_3 და p_4 პოზიციაზე. ეს ინფორმაცია მიეწოდება პროგრამას უკვე განლაგებული ელემენტების და დაკავებული პოზიციების ვექტორების მეშვეობით.



ნახ.1

ელემენტებს შორის კავშირები წარმოდგენილია მომიჯნავეობის მატრიცით, ხოლო საკომუტაციო ველი – დასამაგრებელი ადგილების კოორდინატების X და Y ვექტორებით (X = 1 2 3 1 2 3; Y = 2 2 2 1 1 1), რომელთა საფუძველზედაც შედგენილია ორთოგონალურ მეტრიკაში გამოთვლილი პოზიციებს შორის მანძილების მატრიცა:

$R =$

	e_1	e_2	e_3	e_4	e_5	e_6
e_1	0	1	2	1	0	0
e_2	1	0	1	3	0	0
e_3	2	1	0	1	1	1
e_4	1	3	1	0	1	1
e_5	0	0	1	1	0	0
e_6	0	0	1	1	0	0

$D =$

	p_1	p_2	p_3	p_4	p_5	p_6
p_1	0	1	2	1	2	3
p_2	1	0	1	2	1	2
p_3	2	1	0	3	2	1
p_4	1	2	3	0	1	2
p_5	2	1	2	1	0	1
p_6	3	2	1	2	1	0

ამოხსნა მოყვანილია ცხრილში, ხოლო მიღებული განლაგება – ნახ.1ბ–ზე.

k	$e_j \in \overline{E_k}$	c_j	e_{i_0}	$p_j \in \overline{P_k}$	F_j	არჩეული პოზიცია
1	e_1	2	e_4	p_2	6	p_2
	e_4	3		p_5	9	
	e_5	1		p_6	12	
2	e_1	2	e_1	p_5	7	p_5
		1			7	
3	e_5			p_6		

ქვემოთ მოყვანილია აღწერილი მეთოდის საფუძველზე დაწერილი პროგრამა–ფუნქცია:

```
function P_gan=ganlag_bmul(mom,coord,gan_el,dak_poz)
% sakomutacio velze elementebis ganlageba
%          bmulobis mixedviT
%
% mom - momijnaveobis matrica,
% coord - adgilebis koordinatebis matrica,
% gan_el - ukve ganlagebuli elementebis sia,
% dak_poz - dakavebuli poziciebis sia,
% P_gan - miRebuli ganlageba: 1-li striqoni Seesabameba
%       elementebs, me-2 - poziciebs
%
q=length(coord);
for i=1:q
    for j=1:q
        mandz_adg(i,j)=abs(coord(1,i)-coord(1,j))...
            +abs(coord(2,i)-coord(2,j));
    end
end
n=length(mom); k=length(gan_el);
if k==0
    for i=1:n
        b(i)=0;
        for j=1:n
            b(i)=b(i)+mom(i,j);
        end
    end
    nom=max_povna(b);
    gan_el=[gan_el  nom];
    for i=1:q
        dash(i)=0;
        for j=1:q
            dash(i)=dash(i)+mandz_adg(i,j);
        end
    end
    nomeri=min_povna(dash);
    dak_poz=[dak_poz  nomeri];
    k=k+1;
end
% E - yvela elementis, P - yvela poziciis siebi
for i=1:n
    E(i)=i;
end
for i=1:q
    P(i)=i;
end
Tav_el=simr_gamokleba(E,gan_el);
Tav_poz=simr_gamokleba(P,dak_poz);
mm=n-k;
nn=q-k;
```

```
while mm>=1
   c=[];
   for j=1:mm
       c(j)=-1;
       for i=1:k
           if mom(gan_el(i),Tav_el(j))>c(j)
               c(j)=mom(gan_el(i),Tav_el(j));
           end
       end
   end
   nom=max_povna(c);
   gan_el=[gan_el Tav_el(nom)];
   % fsevdosigrZeebis gamoTvla
   F=[];
   for j=1:nn
      F(j)=0;
          for i=1:k
          F(j)=F(j)+mom(Tav_el(nom),gan_el(i))*mandz_adg(Tav_poz(j),dak_poz(i));
      end
   end
   nomeri=min_povna(F);
   dak_poz=[dak_poz  Tav_poz(nomeri)];
   mm=mm-1; nn=nn-1; k=k+1;
   Tav_el=simr_gamokleba(Tav_el,Tav_el(nom));
   Tav_poz=simr_gamokleba(Tav_poz,Tav_poz(nomeri));
end
P_gan=[gan_el;dak_poz];
%
function nom=max_povna(A)
% veqtoris maximaluri elementis nomris povna
n=length(A);
maximal=A(1);
nom=1;
for i=1:n
    if A(i)>maximal
        maximal=A(i); nom=i;
    end
end
%
function nomeri=min_povna(A)
% veqtoris minimaluri dadebiTi elementis nomris povna
n=length(A);
minimal=A(1);
nomeri=1;
for i=1:n
    if A(i)>0&A(i)<minimal
        minimal=A(i); nomeri=i;
    end
end
```

```
%
function  s_gam=simr_gamokleba(A,B)
%  A  simravlidan  B simravlis gamokleba
n1=length(A); n2=length(B); s_gam=[];
for  i=1:n1
    k=0;
    for j=1:n2
        if A(i)~=B(j)
            k=k+1;
        end
    end
    if k==n2
        s_gam=[s_gam  A(i)];
    end
end
```

ბრძანებათა ფანჯარაში შევიტანოთ საწყისი ინფორმაცია და მივმართოთ ფუნქციას:

```
>> R=[0 1 2 1 0 0;
>> 1 0 1 3 0 0;
>> 2 1 0 1 1 1;
>> 1 3 1 0 1 1;
>> 0 0 1 1 0 0;
>> 0 0 1 1 0 0];
>> coord=[1 2 3 1 2 3;2 2 2 1 1 1];
>>  gan_el=[2 3 6];  dak_poz=[1 3  4];
>> P_gan=ganlag_bmul(R,coord,gan_el,dak_poz)
```

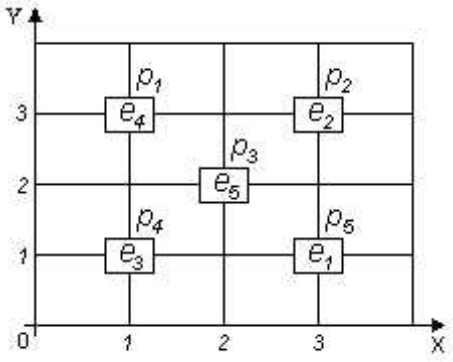
პასუხი წარმოადგენს მატრიცას, რომლის პირველ სტრიქონში ნაჩვენებია ელემენტების ნომრები, მეორეში – შესაბამისი პოზიციები:

```
    P_gan =
         2     3     6     4     1     5
         1     3     4     2     5     6
```

პასუხიდან ნათლად ჩანს მისი სრული იდენტურობა ზემოთ მოყვანილ ცხრილთან.

მეორე მაგალითი ასახავს სიტუაციას, როდესაც წინასწარ არ არის განლაგებული არც ერთი მოდული:

```
>> R=[0 1 2 0 0;1 0 0 1 1;2 0 0 0 1;0 1 0 0 2;0 1 1 2 0];
>> coord=[1 3 2 1 3;3 3 2 1 1];
>> El=[]; poz=[];
>> P_gan=ganlag_bmul(R,coord,El,poz)
```



ნახ.2

მიღებული პასუხია:

```
    P_gan =
  5     4     2     1     3
  3     1     2     5     4
```

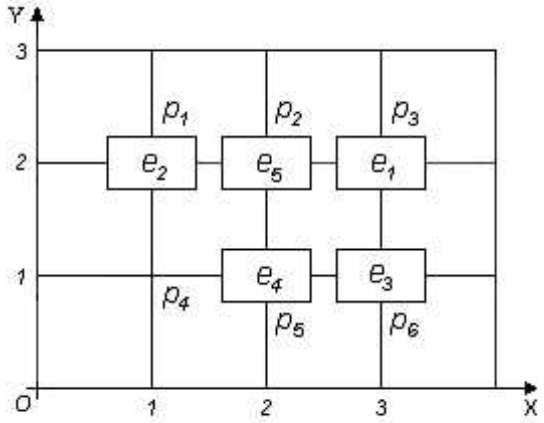
ნახ.2–ზე ნაჩვენებია საკომუტაციო ველი მასზე განლაგებული ელემენტებით, რომელთა ადგილმდებარეობა შეესაბამება მიღებულ პასუხს.

მესამე მაგალითშიც არც ერთი მოდული წინასწარ არ არის განლაგებული. ამის გარდა, აქ დასამაგრებელი პოზიციების რაოდენობა ელემენტების რაოდენობაზე მეტია. იმისათვის, რომ პასუხის გამოტანა მოხდეს ელემენტების ნომრების ზრდადობის შესაბამისად, ძირითადი პროგრამის ბოლო ოპერატორის ნაცვლად შეტანილია ოპერატორები:

```
% poziciebis sortireba elementebis nomrebis zrdadobis Sesabamisad
vvv=dak_poz;
for i=1:n
    for j=1:n
        if E(i)==gan_el(j)
            dak_poz(i)=vvv(j);
            break
        end
    end
end
P_gan=[E;dak_poz];
```

ბრძანებათა ფანჯარაში შევიტანოთ საწყისი ინფორმაცია და მივმართოთ ფუნქციას:

```
>> R=[0 1 1 0 2;
>> 1 0 0 0 3;
>> 1 0 0 1 0;
>> 0 0 1 0 1;
>> 2 3 0 1 0];
>> coord=[1 2 3 1 2 3; 2 2 2 1 1 1];
>> El=[]; poz=[];
>> P_gan=ganlag_bmul(R,coord,El,poz)
```



ნახ.3

მიღებული პასუხია (ნახ.3):

```
P_gan =
   1   2   3   4   5
   3   1   6   5   2
```

დანართი 6

საკომუტაციო ველზე ელემენტების განლაგების ამოცანის ამოხსნა წყვილ–წყვილი გადაადგილების მეთოდით

წყვილ–წყვილი გადაადგილების მეთოდი მიეკუთვნება იტერაციულ ალგორითმთა ჯგუფს. როგორც წესი, ამ მეთოდით სარგებლობენ ერთნაირგაბარიტებიანი მოდულების საკომუტაციო ველზე განლაგებისას.

დავუშვათ, რომ ინფორმაცია სქემის ელემენტებს შორის კავშირების შესახებ მოიცემა მომიჯნავეობის R მატრიცით, ხოლო საკომუტაციო ველის გეომეტრიული პარამეტრების შესახებ – დასამაგრებელი ადგილების კოორდინატებით, რომელთა საფუძველზედაც ხდება დასამაგრებელ ადგილებს შორის მანძილების მატრიცის შედგენა. მანძილი დასამაგრებელ ადგილებს შორის გამოითვლება ორთოგონალურ მეტრიკაში. ელემენტების საწყისი განლაგების შესახებ ინფორმაცია მოიცემა P ვექტორით, რომლის $p_i = j$ ელემენტი გვიჩვენებს, რომ j–ურ ადგილზე განლაგებულია e_i ელემენტი. ზოგჯერ საჭიროა განლაგების საწყის ვარიანტში ზოგიერთი ელემენტის დაფიქსირება. ამისათვის შემოღებულია ფიქსაციის F ვექტორი, რომლის ელემენტი $f_i = 1$, თუ e_i ვექტორი მითითებულ ადგილზეა დამაგრებული. თუ e_i არ არის დაფიქსირებული და შეუძლია მონაწილეობის მიღება გადაადგილებაში, მაშინ $f_i = 0$.

ოპტიმალურობის კრიტერიუმია შეერთებების სიგრძეების ჯამის მინიმიზაცია. შეერთებების ჯამური სიგრძე გამოითვლება ფორმულით:

$$L = \frac{1}{2}\sum_{e_i \in E}\sum_{e_j \in E} r_{ij} \cdot d_{p(e_i)p(e_j)},$$

სადაც $p(e_i)$, $p(e_j)$ – საკომუტაციო ველზე e_i და e_j ელემენტების პოზიციები, ხოლო $d_{p(e_i)p(e_j)}$ – მანძილი ამ ელემენტებს შორის.

აღვწეროთ წყვილ–წყვილი გადაადგილების მეთოდით განლაგების ალგორითმის სქემა:

1. საწყისი მონაცემებისათვის ორთოგონალურ მეტრიკაში მანძილების მატრიცის და L – შეერთებების ჯამური სიგრძის გამოთვლა;
2. $it \leftarrow 1$ (it ცვლადის ამ მნიშვნელობისათვის სრულდება მორიგი იტერაცია);
3. იტერაციის დასაწყისი. $it \leftarrow 0$ (ეს მინიჭება საჭიროა იმისათვის, რომ ოპტიმალური განლაგების მიღებისას არ შესრულდეს შემდეგი იტერაცია და პროცესი შეწყდეს);
4. განლაგების საწყის ვარიანტში i და j მოდულების წყვილის ($i = \overline{1, n-1}$; $j = \overline{i+1, n}$) ადგილების ურთიერთგაცვლა და გაცვლის შედეგად მიღებული შეერთებების ჯამური სიგრძის L_{ij}–ის გამოთვლა. (ცხადია, i და j მოდულები არ უნდა იყოს დაფიქსირებული).

5. ყველა შესაძლო გადაადგილების შედეგად e_q და e_w მოდულების წყვილის განსაზღვრა, რომელთა ადგილების ურთიერთგაცვლა იძლევა მიზნობრივი ფუნქციის მინიმუმს. e_q და e_w მოდულების გადაადგილება, ამასთან, $L \leftarrow L_{qw}$, $it \leftarrow 1$ (თუ მოდულების ასეთი წყვილი არ არსებობს, it რჩება 0–ის ტოლი);
6. მიმდინარე იტერაციის დასასრული: თუ $it = 1$, გადასვლა მე–3 ბიჯზე – შემდეგი იტერაციის შესრულებაზე; თუ $it = 0$ (იტერაციის შედეგია: არ არსებობს მოდულების წყვილი, რომელთა გადაადგილებით ხდება შეერთებების ჯამური სიგრძის მინიმიზაცია), ალგორითმი ამთავრებს მუშაობას.

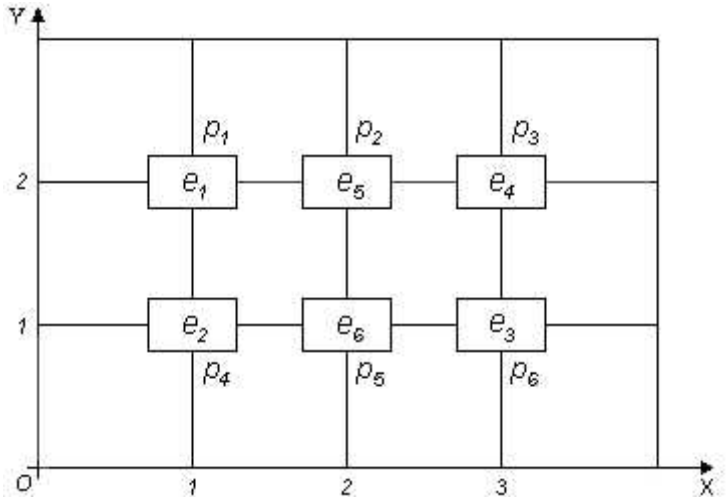
აღწერილ ალგორითმში დასამაგრებელი პოზიციების რაოდენობა ელემენტების რაოდენობის ტოლია.

მაგალითი:

ნახ.1–ზე მოყვანილია 6–პოზიციანი საკომუტაციო ველი მასზე განლაგებული ელემენტებით.

საწყისი ინფორმაციაა:

- მომიჯნავეობის R მატრიცა;
- საკომუტაციო ველის შესაბამისი კოორდინატთა *coord* მატრიცა: მატრიცის პირველი სტრიქონი შეესაბამება დასამაგრებელი ადგილების აბსცისებს, მეორე – ორდინატებს;
- ელემენტთა საწყისი განლაგების P ვექტორი;
- ელემენტთა ადგილმდებარეობის ფიქსაციის F ვექტორი.



$$R = \begin{array}{c|cccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ e_1 & 0 & 1 & 1 & 2 & 0 & 0 \\ e_2 & 1 & 0 & 0 & 4 & 0 & 1 \\ e_3 & 1 & 0 & 0 & 0 & 0 & 2 \\ e_4 & 2 & 4 & 0 & 0 & 1 & 1 \\ e_5 & 0 & 0 & 0 & 1 & 0 & 3 \\ e_6 & 0 & 1 & 2 & 1 & 3 & 0 \end{array};$$

$$coord = \begin{vmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 2 & 2 & 2 & 1 & 1 & 1 \end{vmatrix};$$

ნახ.1

P = | 1 4 6 3 2 5 | (1–ლი ელემენტი 1–ლ პოზიციაზე, მე–2 – მე–4–ზე და ა.შ);

F = | 1 0 0 1 0 1 | (დაფიქსირებულია 1–ლი, მე–4 და მე–6 ელემენტების პოზიციები).

ცხრილში მოყვანილია ამოცანის ამოხსნის მსვლელობა:

იტერაცი-ის ნომერი	P	L	შესაძლო გადაადგილებები	M	L_{ij}	$L-L_{ij}$	იტერაცი-ის პასუხი
1	1 4 6 3 2 5	29	$e_2\leftrightarrow e_3$ $e_2\leftrightarrow e_5$ $e_3\leftrightarrow e_5$	1 6 4 3 2 5 1 2 6 3 4 5 1 4 2 3 6 5	21 23 27	8 6 2	$e_2\leftrightarrow e_3$
2	1 6 4 3 2 5	21	$e_2\leftrightarrow e_3$ $e_2\leftrightarrow e_5$ $e_3\leftrightarrow e_5$	1 4 6 3 2 5 1 2 4 3 6 5 1 6 2 3 4 5	29 19 23	-8 2 -2	$e_2\leftrightarrow e_5$
3	1 2 4 3 6 5	19	$e_2\leftrightarrow e_3$ $e_2\leftrightarrow e_5$ $e_3\leftrightarrow e_5$	1 4 2 3 6 5 1 6 4 3 2 5 1 2 6 3 4 5	27 21 23	-8 -2 -4	პროცესი დამთავრე-ბულია

საბოლოო განლაგებაა $P = |\,1\,2\,4\,3\,6\,5\,|$, რომლის დროს $L = 19$.

ქვემოთ მოყვანილია აღწერილი ალგორითმის საფუძველზე დაწერილი პროგრამა–ფუნქცია:

```
function [Popt,kavsh_sigrZe]=ganlag_iter(mom,coord,Psawy,fixir)
% sakomutacio velze elementebis ganlageba
%   wyvil-wyvili gadaadgilebis meTodiT
%
% mom - momijnaveobis matrica,
% coord - adgilebis koordinatebis matrica,
% Psawy - sawyisi ganlagebis veqtori,
% fixir - fiqsaciis veqtori
%
n=length(coord);
for i=1:n
    for j=1:n
        mandz_adg(i,j)=abs(coord(1,i)-coord(1,j))+abs(coord(2,i)-coord(2,j));
    end
end
it=1;
P=Psawy;
kavsh_sigrZe=sig(mom,mandz_adg,Psawy,n);
while it==1
    it=0;
    for i=1:n-1
        if fixir(i)==0
            for j=i+1:n
                if fixir(j)==0
                    P1=P;
                    z=P1(i);
                    P1(i)=P1(j);
```

```
                    P1(j)=z;
                    kavsh_sigrZe_axali=sig(mom,mandz_adg,P1,n);
                    if kavsh_sigrZe_axali<kavsh_sigrZe
                       kavsh_sigrZe=kavsh_sigrZe_axali;
                       q=i;
                       w=j;
                       it=1;
                    end
                end
            end
        end
    end
    if it==1
        z=P(q);
        P(q)=P(w);
        P(w)=z;
    end
end
Popt=P;
%
function L=sig(A,D,P,n)
%       SeerTebaTa sigrZis gamoTvla
% A da D - momijnaveobis da manZilis matricebi,
% P - ganlagebis veqtori, n - elementTa raodenobaa
L=0;
for i=1:n-1
    for j=i+1:n
        L=L+A(i,j)*D(P(i),P(j));
   end
end
```

ბრძანებათა ფანჯარაში შევიტანოთ ინფორმაცია, რომელიც ნახ.1–ს შეესაბამება. მივიღებთ შედეგს:

```
Popt =
   1    2    4    3    6    5
kavsh_sigrZe =
       19
```

მე–2 მაგალითი:

ბრძანებათა ფანჯარაში შევიტანოთ შემდეგი საწყისი ინფორმაცია და მივმართოთ ganlag_iter ფუნქციას:

```
>> mom=[0 1 2 3 4 1 2 3 3 2 1 1; 1 0 2 3 3 1 2 1 3 1 0 1; 2 2 0 1 1 3 1 0 3 1 0 3; 3 3 1 0 1 1 2 2 1 1 2 1;...
4 3 1 1 0 1 1 1 2 2 1 2; 1 1 3 1 1 0 1 2 3 1 3 1; 2 2 1 2 1 1 0 4 2 0 0 1; 3 1 0 2 1 2 4 0 2 2 2 2;...
3 3 3 1 2 3 2 2 0 1 1 2; 2 1 1 1 2 1 0 2 1 0 1 1; 1 0 0 2 1 3 0 2 1 1 0 2; 1 1 3 1 2 1 1 2 2 1 2 0];
>> coord=[1 4 7 10 1 4 7 10 1 4 7 10; 5 5 5 5 3 3 3 3 1 1 1 1];
```

```
>> Psawy=[11 2 8 5 12 1 9 3 6 4 7 10];
>> fixir=[0 1 0 0 1 0 0 0 0 0 1 0];
>> [Popt,kavsh_sigrZe]=ganlag_iter(mom,coord,Psawy,fixir)
```

მივიღებთ პასუხს:

```
Popt =
   11    2    5    3   12    1    4    8    6    9    7   10
kavsh_sigrZe =
  592
```

თუ პროგრამაში გავითვალისწინებთ ყოველი იტერაციის შემდეგ შუალედური მონაცემების ეკრანზე გამოტანას, დავინახავთ, რომ ოპტიმალური განლაგების მისაღებად დასჭირდა 5 იტერაცია (ცხადია, იგივე საწყისი მონაცემებისათვის). ქვემოთ მოყვანილია ყოველი იტერაციის დასაწყისში არსებული განლაგების ვექტორი და კავშირების სიგრძე:

1) P = 11 2 8 5 12 1 9 3 6 4 7 10 ,
   kavsh_sigrZe = 649;
2) P = 11 2 5 8 12 1 9 3 6 4 7 10 ,
   kavsh_sigrZe = 616;
3) P = 11 2 5 8 12 1 4 3 6 9 7 10 ,
   kavsh_sigrZe = 603;
4) P = 11 2 5 3 12 1 4 8 6 9 7 10 ,
   kavsh_sigrZe = 598;
5) P = 11 2 1 3 12 5 4 8 6 9 7 10 ,
   kavsh_sigrZe = 592.

საბოლოო პასუხია: Popt = 11 2 1 3 12 5 4 8 6 9 7 10,
kavsh_sigrZe = 592.

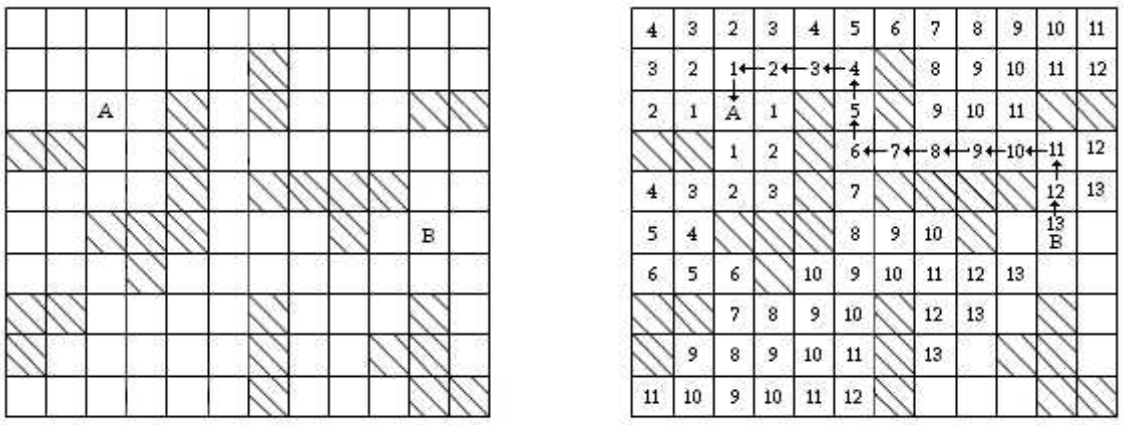
დანართი 7

გამტარების ტრასირება ლის ალგორითმით

ცალკეული გამტარების ტრასირების ყველაზე გავრცელებულ ალგორითმს წარმოადგენს ტალღური ალგორითმი (ლის ალგორითმი), რომელსაც მრავალრიცხოვანი მოდიფიკაცია გააჩნია. განვიხილოთ ლის ალგორითმის არსი.

საკომუტაციო ველი დავყოთ მართკუთხა ბადის უჯრედებად, როგორც ეს ნაჩვენებია ნახ.1,ა–ზე. აქ A და B შემაერთებელი კონტაქტებია, ხოლო დაშტრიხულია პლატის „აკრძალული“ უჯრედები, სადაც ტრასის გატარება არ შეიძლება.

პროცესი იწყება საწყის წერტილად ნებისმიერი ამ ორი კონტაქტის არჩევით. ავირჩიოთ, მაგ., A კონტაქტი. თითოეულ ცარიელ უჯრედში, რომელიც უშუალოდ ესაზღვრება (ჰორიზონტალურად ან ვერტიკალურად) A კონტაქტის უჯრედს, იწერება ციფრი 1. შემდეგ ყველა ცარიელ უჯრედში, რომელიც ესაზღვრება 1 ინდექსის მქონე უჯრედებს, იწერება ციფრი 2. იგივე პრინციპით იწერება ციფრები 3, 4 და ა. შ. უჯრედების დანომვრის (ინდექსაციის) პროცესი გრძელდება ან მიზნის – B უჯრედის მიღწევამდე, ან ყველა შესაძლო გზის ბლოკირების აღმოჩენამდე (*k*–ურ ბიჯზე არ აღმოჩნდება (*k*–1)-ნომრიანი უჯრედების მოსაზღვრე ცარიელი უჯრედები). ნახ.1,ბ–ზე ნაჩვენებია საკომუტაციო ველი მისი უჯრედების დანომვრის შემდეგ. B უჯრედის ინდექსი A–სა და B–ს შორის უმოკლესი გზის სიგრძის ტოლია.



ა) ბ)

ნახ.1

იმისათვის, რომ მოვძებნოთ უჯრედები, რომლებზედაც გაივლის ტრასა, B უჯრედი უნდა შევაერთოთ უჯრედთან, რომლის ნომერი B უჯრედის ინდექსზე 1–ით ნაკლებია, ანუ უჯრედთან ინდექსით 12, უჯრედი 12 – უჯრედ 11–თან და ა.შ., სანამ არ მივაღწევთ A უჯრედს. ნახ.1,ბ–ზე ისრებით ნაჩვენებია მიღებული ტრასა.

იმ შემთხვევაში, როდესაც *k* ინდექსის მქონე უჯრედს რამდენიმე (*k*–1)-ნომრიანი უჯრედი ესაზღვრება, საჭიროა მათ შორის ერთ–ერთის არჩევა. თეორიულად ალტერნატიული

უჯრედებიდან ნებისმიერის არჩევა შეიძლება. მაგრამ პრაქტიკაში, ტრასის მოხვევათა რიცხვის შემცირების მიზნით, რეკომენდირებულია წინა მიმართულების შენარჩუნება.

თუ წინა მიმართულების შეცვლა აუცილებელია ან, როგორც ეს არის ტრასის განსაზღვრის პროცესის დასაწყისში, წინა მიმართულება ჯერ არ არსებობს, გამოიყენება ე.წ. „კომპასის წესი“, რომლის მიხედვითაც მიმართულებებს ასეთი პრიორიტეტები გააჩნიათ: ჩრდილოეთი – აღმოსავლეთი – დასავლეთი – სამხრეთი. პრიორიტეტების სიაში მიმართულებების მიმდევრობა პირობითია. მთავარია, დავიცვათ ერთხელ არჩეული წესი, რათა მივიღოთ გამტარების „ბუდისებრი“ განლაგება, რომელიც ტრასირებისათვის ყველაზე ხელსაყრელ პირობებს ქმნის.

ქვემოთ მოყვანილია აღწერილი ალგორითმის საფუძველზე დაწერილი პროგრამა–ფუნქცია:

```
function  [sigrdze coord_tras]=tras_Li(plata,A,B)
% trasis koordinatebis gansazRvra
% plata – platis aRmweri matrica;
% A da B –trasis sawyisi da bolo wertilebi
%
[n m]=size(plata);
[index sigrdze zzz]=indeqsacia(plata,A,B,n,m);
if zzz==0
    'am monacemebisaTvis gza A-dan B-mde beWduri montaJiT ver Sesruldeba'
    return
end
coord=B; mimd=B;
% mimd – mimdinare gansahilveli ujredebis CamonaTvali
win_mim=0;
% win_mim – wina mimarTulebis cvladia: 1 – chrdiloeTi; 2 – aRmosavleTi;
%   3 – dasavleTi; 4 – samxreTi; 0 – wina mimarTulebis araryseboba
for i=sigrdze-1 :-1:1
    k=zeros(4,1);
%    k – veqtori, romelic aCvenebs  mimdinare etapze
%        trasis  SesaZlo  mimarTulebebs
    ujr=zeros(4,2);
% ujr – matrica, romelSic  Sedis ujredebi – trasis kandidatebi
% zeda wveros Semowbeba
    if  mimd(1)-1>=1&index(mimd(1)-1,mimd(2))==i
        ujr(1,:)=[mimd(1)-1  mimd(2)];
        k(1)=1;
    end
% marjvena wveros Semowbeba
    if mimd(2)+1<=m&index(mimd(1),mimd(2)+1)==i
        ujr(2,:)=[mimd(1)  mimd(2)+1];
        k(2)=1;
    end
% marcxena wveros Semowbeba
    if mimd(2)-1>=1&index(mimd(1),mimd(2)-1)==i
        ujr(3,:)=[mimd(1)  mimd(2)-1];
        k(3)=1;
    end
% qveda wveros Semowbeba
```

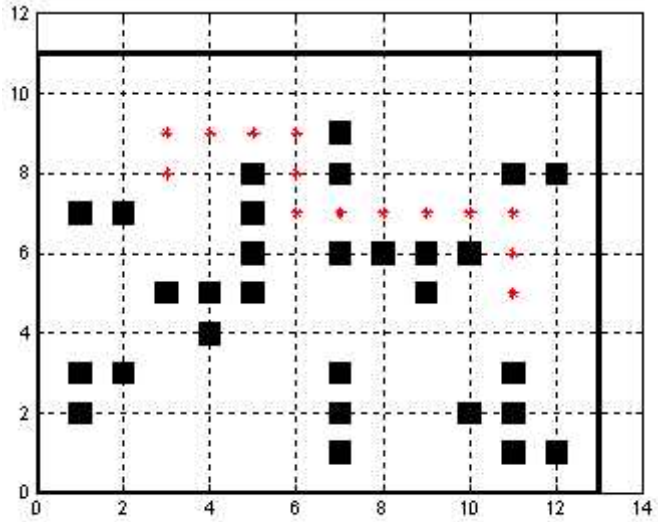
```
    if mimd(1)+1<=n&index(mimd(1)+1,mimd(2))==i
        ujr(4,:)=[mimd(1)+1  mimd(2)];
        k(4)=1;
    end
    zz=0;
% wina mimarTulebis wesis realizacia
% zz=1, Tu moxerxda wina mimarTulebis SenarCuneba,
%        winaaRmdeg SemTxvevashi zz=0
    if win_mim~=0
        zz=0;
        for  j=1:4
            if  win_mim==j&k(j)==1
                mimd=ujr(j,:); zz=1;
            end
        end
    end
   if win_mim==0|zz==0
% kompasis wesis realizacia
       for  j=1:4
           if  k(j)==1
               mimd=ujr(j,:); win_mim=j;
               zz=0;
               break
           end
       end
   end
coord=[coord;mimd];
end
coord=[coord;A];
coord=coord';
% grafikis ageba da trasis ujredebis adgilmdebareobis gadaangariSeba
%        dekartis sakoordinato sistemis Tanaxmad
X=[0 0 m+1 m+1 0];
Y=[0 n+1 n+1 0 0];
plot(X,Y,'k','LineWidth',3)
hold on
for i=1:n
    for j=1:m
        if plata(i,j)==-1
            plot([j],[n+1-i],'ks','LineWidth',7)
        end
    end
end
grid on
X=coord(1,:); Y=coord(2,:);
for i=1:sigrdze+1
    w=X(i);
    X(i)=Y(i);
    Y(i)=n+1-w;
end
plot(X,Y,'r*')
coord_tras=[X;Y];
%
```

```
function [index sigrdze zzz]=indeqsacia(plata,A,B,n,m)
% ujredebis indeqsacia
%
index=plata;
sigrdze=0;
sia=A;
% sia – mimdinare indeqsis (sigrZis) mqone ujredebis CamonaTvali
zzz=1;
mm=1;
index(A(1),A(2))=-1;
while zzz==1
% cikli grZeldeba saboloo B wertilis miRwevamde
% an yvela SesaZlo gzis blokirebis aRmochenamde;
    sigrdze=sigrdze+1;
    dam=[];
% dam – ujredebis CamonaTvali, romlebsac ciklshi miewereba indeqsi
    for k=1:mm
        % zeda wveros indeqsacia
        if sia(k,1)-1>=1&index(sia(k,1)-1,sia(k,2))==0
            index(sia(k,1)-1,sia(k,2))=sigrdze;
            ujr=[sia(k,1)-1,sia(k,2)];
            if ujr==B
                index(A(1),A(2))=0;
                return
            end
            dam=[dam; ujr];
        end
        % marjvena wveros indeqsacia
        if sia(k,2)+1<=m&index(sia(k,1),sia(k,2)+1)==0
            index(sia(k,1),sia(k,2)+1)=sigrdze;
            ujr=[sia(k,1),sia(k,2)+1];
            if ujr==B
                index(A(1),A(2))=0;
                return
            end
            dam=[dam;ujr];
        end
        % marcxena wveros indeqsacia
        if sia(k,2)-1>=1&index(sia(k,1),sia(k,2)-1)==0
            index(sia(k,1),sia(k,2)-1)=sigrdze;
            ujr=[sia(k,1),sia(k,2)-1];
            if ujr==B
                index(A(1),A(2))=0;
                return
            end
            dam=[dam; ujr];
        end
        % qveda wveros indeqsacia
        if sia(k,1)+1<=n&index(sia(k,1)+1,sia(k,2))==0
            index(sia(k,1)+1,sia(k,2))=sigrdze;
            ujr=[sia(k,1)+1,sia(k,2)];
            if ujr==B
                index(A(1),A(2))=0;
```

```
                return
            end
            dam=[dam;ujr];
        end
    end
    if size(dam)==0
        zzz=0;
    end
    sia=dam;
    [mm nn]=size(sia);
end
```

ბრძანებათა ფანჯარაში შევიტანოთ ინფორმაცია, რომელიც ნახ.1,ა–ს შეესაბამება. აღვნიშნოთ, რომ პლატის აღწერა გათვალისწინებულია მატრიცის სახით. ინფორმაცია ჩაიწერება თანმიმდევრობით (სტრიქონებად) ყველა ჰორიზონტალის გასწვრივ ზევიდან ქვევით: დაკავებულ უჯრედებს შეესაბამება -1, ხოლო თავისუფალს – 0. A უჯრედი იმყოფება მე–3 სტრიქონის და მე–3 სვეტის, ხოლო B – მე–6 სტრიქონის და მე–11 სვეტის გადაკვეთაზე.

```
>> P=[0 0 0 0 0 0 0 0 0 0 0 0 0;
 0 0 0 0 0 0 -1 0 0 0 0 0;
0 0 0 0-1 0 -1 0  0  0 -1 -1;
-1 -1 0 0 -1 0 0 0 0 0 0;
0 0 0 0 -1  0 -1 -1  -1 -1 0  0;
0 0 -1 -1 -1 0 0 0 -1 0 0 0;
0 0 0 -1 0 0 0 0 0 0 0;
-1 -1 0 0 0 0 -1 0 0 0 -1 0;
-1 0 0 0 0 0 -1 0 0 -1 -1 0;
 0 0 0 0 0 0 -1 0 0 0 -1 -1];
>> A=[3 3];  B=[6 11];
```



ნახ.2

მივმართოთ `tras_Li` ფუნქციას:

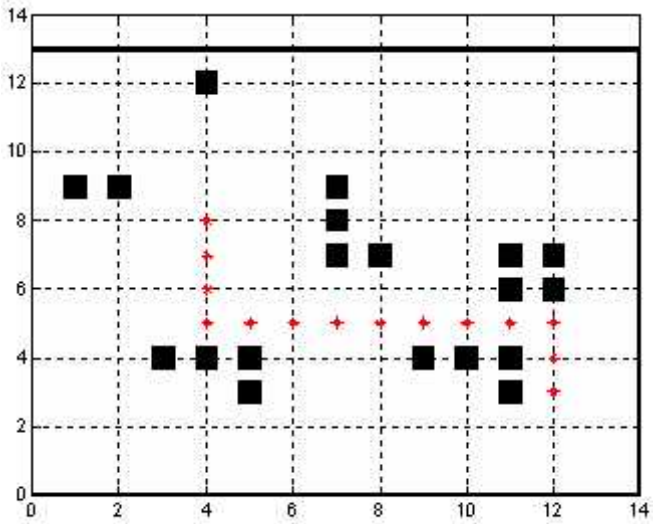
```
>> [sigrdze coord_tras]=tras_Li(P,A,B)
```

შედეგად მივიღებთ ტრასის სიგრძეს და თვით ტრასას. აღვნიშნოთ, რომ ტრასის უჯრედების ადგილმდებარეობა გადაანგარიშებულია დეკარტის საკოორდინატო სისტემის თანახმად. coord_tras მატრიცის პირველ სტრიქონში მოყვანილია ტრასის უჯრედების *x* კოორდინატები B–დან A–მდე, მეორეში – შესაბამისი *y* კოორდინატები. გათვალისწინებულია აგრეთვე პლატის ნახაზის აგება და მასზე ტრასის ჩვენება (ნახ.2).

```
sigrdze =
   13
coord_tras =
   11   11   11   10    9    8    7    6    6    6    5    4    3    3
    5    6    7    7    7    7    7    7    8    9    9    9    9    8
```

მე–2 მაგალითი (მიღებულია ნახ.3):

```
>> P=[0 0 0 -1 0 0 0 0 0 0 0 0 0;
 0 0 0 0 0 0 0 0  0 0 0 0 0;
 0 0 0 0 0 0 0 0 0 0 0 0 0;
 -1 -1 0 0 0 0 -1 0 0 0 0 0 0;
0 0 0 0 0 0 -1 0 0 0 0 0 0;
0 0 0 0 0 0 -1 -1 0 0 -1 -1 0;
0 0 0 0 0 0 0 0 0 0 -1 -1 0;
0 0 0 0 0 0 0 0 0 0  0 0 0;
0 0 -1 -1 -1 0 0 0 -1 -1 -1 0 0;
0 0 0 0 -1 0 0 0 0 0 -1 0 0 ;
0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0];
>> A=[5 4]; B=[10 12];
>> [sigrdze coord_tras]=tras_Li(P,A,B)
sigrdze =
   13
coord_tras =
   12   12   12   11   10    9    8    7    6    5    4    4    4    4
    3    4    5    5    5    5    5    5    5    5    5    6    7    8
```

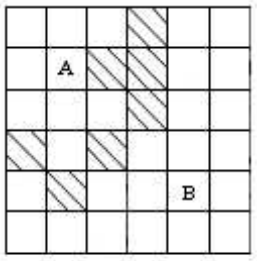


ნახ.3

მე–3 მაგალითი:

ამ მაგალითში ასახულია სიტუაცია, როდესაც A და B კონტაქტების შეერთება ბეჭდური მონტაჟით შეუძლებელია (ნახ.4).

```
>> S=[0  0  0  -1  0  0;
    0  0  -1  -1  0  0;
    0  0  0  -1  0  0;
   -1  0  -1  0  0  0;
    0  -1  0  0  0  0;
    0  0  0  0  0  0];
>> A=[2 2];
>> B=[5 5];
>> [L c]=tras_Li(S,A,B)
ans =
am monacemebisaTvis gza A-dan B-mde beWduri montaJiT ver Sesruldeba
```



ნახ.4

მოყვანილი პასუხის გარდა, გამოტანილი იქნება შეტყობინება შეცდომაზე, რადგან ამისთანა შემთხვევაში, როდესაც გზა არ არსებობს, ბუნებრივია, შეუძლებელია ტრასის სიგრძისა და მისი უჯრედების ადგილმდებარეობის განსაზღვრა.

ლიტერატურა

1. Дьяконов В.П. MATLAB 7. */R 2006/R 2007. Москва, ДМК ПРЕСС, 2008. 214 стр.
2. Кетков Ю., Кетков А., Шульц М. MATLAB 7 Программирование, численные методы. Санкт–Петербург, БХВ–Петербург, 2005. 737 стр.
3. Потемкин В.Г. Вычисления в среде MATLAB. Москва, Диалог–МИФИ, 2004. 720 стр.
4. ზ. ბაიაშვილი. MATLAB პროგრამული პაკეტის გამოყენების საფუძვლები. თბილისი, სტუ, 2010. 118 გვ.
5. ლ. ფერაძე, მ. ქურხული. რადიოელექტრონული და ელექტრონული გამომთვლელი აპარატურის საკონსტრუქტორო–ტექნოლოგიური დაპროექტების ავტომატიზაციის საფუძვლები. თბილისი, სტუ, 1977. 158 გვ.
6. Misza Kalechman. Practical MATLAB Basics for Engineers. London–NewYork, CRC Press, Taylor&Francis Group, 2009. 724 p.