M. Essert, T. Žilić

MATLAB – **Mat**rični **lab**oratorij

Zagreb, 2004.

Iskrenom podupiratelju
računalne matematike
Prof. dr. sc. Ivanu Mahalec-u
Posvećujemo

Predgovor

Ovaj udžbenik za matematički jezik ***Matlab*** izrastao je iz skripti nastalih 1992. godine, za vrijeme srpske agresije na Hrvatsku, dok smo često sa studentima čekali prestanak zračne opasnosti, kako bismo mogli nastaviti s nastavom. Bila je to skripta skromnih mogućnosti, iako su studenti tvrdili kako im je bila razumljiva i vrlo korisna.

Još od mog povratka iz Heidelberga 1990. godine, gdje sam na Matematičkom institutu upoznao važnost računalne matematike, nastojao sam to iskustvo prenijeti i u svoju sredinu - Fakultet strojarstva i brodogradnje u Zagrebu. Trud je bio ogroman, a rezultati vrlo blijedi. Laboratoriju *Centra za računalnu matematiku* potporu je pružalo samo par pojedinaca. Među njima posebno mjesto pripada *prof.dr.sc. Ivanu Mahalec*-u, kojemu je ovaj udžbenik i posvećen.

Treći poticaj dogodio se služenjem civilnog roka, negdašnjeg studenta, a sada dragog kolege i asistenta, *Tihomira Žilića* na našoj Katedri. Zahvaljujući njegovom oduševljenju i velikom trudu za ovakav posao, načinjen je ovaj udžbenik.

Nastojali smo povezati korisničke i referentne priručnike kojih Matlab ima jako puno u jednu zaokruženu cjelinu, koja bi trebala dati ne samo osnove, nego i nadogradnju u radu s Matlab-om.

Nadamo se da će ovaj udžbenik biti koristan ne samo u upoznavanju Matlab-a nego i drugih, nekomercijalnih rješenja, kao što su Octave i Scilab, koji pokazuju vrijednost i važnost računalne matematike u inženjerskom i znanstvenom radu.

Naša želja je da taj rad donese i radost čitatelju.

Autori

Zagreb, studenog 2004.

UVOD 1

Matlab je matematički programski paket za znanstveni i inženjerski numerički račun, a izrastao je iz desetljećima usavršavanih fortranskih paketa LINPACK-a i EISPACK-a. Stoga nije ni čudo da se smatra standardom sveučilišnog matematičkog alata, iako se intenzivno koristi kako u industrijskom razvoju tako i praktičnom inženjerstvu. Prva verzija Matlab-a napisana je krajem 1970. godine na sveučilištima *University of New Mexico i Stanford University* s ciljem primjene u matričnoj teoriji, linearnoj algebri i numeričkoj analizi.

Budući da je zamišljen i razvijen (od 1986. u C-jeziku) kao interpreterski programski jezik visoke razine, koji se u početku temeljio na kompleksnoj matrici kao osnovnom tipu podatka, imenovan je kraticom od **Mat**rični **lab**oratorij. Mogućnost povezivanja s programima pisanim u C jeziku ili Fotranu, čini ga otvorenim za složene projekte, a gotova (funkcijska) rješenja za različita područja primjene kontinuirano mu proširuju domet. Po svojoj formi blizak je načinu na koji inače zapisujemo matematičke formule, pa jedan redak u Matlab-u može zamijeniti stotine redaka napisanih u nekom programskom jeziku opće namjene (Fortrana, C-a ili Jave).

Matlab je stoga jezik visoke učinkovitosti u tehničkom računanju. On objedinjuje računanje, vizualizaciju i programiranje u prozorskom okolišu vrlo ugodnom za korisnika, gdje su problemi i rješenja izraženi uobičajenim matematičkim zapisom. Tipična upotreba Matlaba uključuje:

- Matematiku i računanje
- Razvitak algoritama
- Modeliranje, simulaciju i izgradnju prototipova
- Analizu, obradu i vizualizaciju podataka
- Znanstvenu i inženjersku grafiku
- Razvitak gotovih rješenja (aplikacija) sa GUI (Graphical User Interface) alatima.

Danas svojstva Matlab-a daleko prelaze originalni "matrični laboratorij".Uz osnovni paket postoje i brojni programski alati (toolboxes) koji pokrivaju gotovo sva područja inženjerske djelatnosti: obradu signala, slike, 2D i 3D grafičko oblikovanje, automatsko upravljanje, identifikaciju sustava, statističke obrade, neuronske mreže, financijsku matematiku, simboličku matematiku i mnogo drugih. Paket SIMULINK je dodatak Matlab-u koji omogućuje simulaciju kontinuiranih i diskretnih sustava pomoću funkcijskih blok dijagrama i dijagrama stanja. Matlab je otvoren sustav u kojem korisnik može graditi svoje vlastite alate i biblioteke te modificirati postojeće, jer su dostupni u obliku izvornog koda.

Instalacija Matlab-a dostupna je za različite vrste strojeva, od osobnih računala do paralelnih strojeva, te za različite operacijske sustave (Windows, Unix/Linux, VMS).

Licencu Matlab-a drži tvrtka The MathWorks, Inc., 21. Eliot St., South Natick, MA 01760. Korisnici se putem elektronske pošte (e-mail) mogu uključiti u besplatnu korisničku knjižnicu na koju su povezani i najvažniji svjetski sveučilišni centri.
Razlog ovakve popularnosti leži u nekoliko činjenica:

- Matlab se odlikuje elegancijom, praktičnošću i preglednošću, pa se poput pseudokoda primjenjuje u mnogim knjigama kod opisivanja računskih postupaka.
- Matlab posjeduje veliku fleksibilnost: od običnog stolnog računala za računanje brojeva i matrica do sredstva za rješavanje zahtjevnijih zadataka.
- MathWorks nudi jako dobru 'On-line' potporu.
- Složeniji programi mogu biti postavljeni u kraćem vremenu, za razliku od vremena koje je potrebno kod "klasičnih" programskih jezika.
- Matlab se brzo uči: već nakon nekoliko dana mogu se rješavati zahtjevnije zadaće.
- Matlab posjeduje jaku grafičku potporu, koja se sa svojim jednostavnim funkcijama može brzo naučiti, međutim, uz značajno više uloženog truda – nudi nevjerojatne mogućnosti.

Sva ta svojstva čine Matlab omiljenim među matematičarima, kao i među praktičarima različitih pravaca (prirodne-, privredne-, inženjerske znanosti). Tome pridonosi Matlabova proširljivost kroz samoizgradnju ili dodavanje pripremljenih funkcija od matične tvrtke i/ili sveučilišnih centara.
Osim originalne, komercijalne verzije Matlaba, postoje i vrlo dobra, za sveučilišta besplatna rješenja sličnih paketa, među kojima se ističu Scilab i Octave.

PRVI KORACI 2

Matlab se može koristiti na više načina: kao "džepno računalo", kao programski jezik ili kao vrlo složeni matematički alat. Korisnik se za prvu primjenu može osposobiti u roku od pola sata, za drugu mu treba nekoliko sati (ako već ima iskustva u programiranju) ili dana (ako tog iskustva nema), dok se u trećoj mogu uložiti mjeseci i godine rada, jer mogućnosti rastu sa složenošću problema, a svaka riješena zadaća rađa novu. Sav se taj trud naplaćuje radošću. Matlab je, kako piše u njegovom priručniku, dovoljno snažan da promijeni tok Vašeg života.

U Windows okolišu Matlab se pokreće dvostrukim klikom na Matlab ikonu na radnoj površini (desktopu) ili izborom Matlab programa preko Start menu-a. Otvaranjem Matlab prozora, pojavljuje se radni prostor tzv. 'command window' u kojem Matlab ispisuje '»' kao svoj upit (engl. *prompt*) i očekuje korisnikovu naredbu. U ovakvom, interaktivnom načinu rada, korisnik upisuje naredbu koja se potom izvršava pritiskom na 'Enter' tipku. Imena tipki redovito ćemo u daljnjem tekstu pisati unutar jednostrukih navodnika.

Nakon što Matlab izvrši (izračuna) naredbu, ispisuje se ili iscrtava odgovor, te ponovno pojavljuje prompt za nastavak komunikacije s korisnikom. U slučaju da naredba završava znakom ';' sprječava se ispis rješenja na zaslonu. Naredbe koje su duže od širine zaslona nastavljaju se u novom redu utipkavanjem triju točkica '...' u prethodnom retku, a znak '%' služi kao komentar naredbe. Sve znakove desno od tog znaka Matlab neće interpretirati.

Tablica 2.1: *Znakovi na komandnoj liniji*

Znakovi	Opis znaka	Upotreba znakova
>>	Upit (engl. prompt)	>> x=1 <Enter>-tipka x = 1
,	Odvajanje naredbi	>> x=0.5, y=sin(x) x = 0.5000 y = 0.4794
;	Sprječavanje ispisa	>> x=1; y=6, z=3; s=7; y = 6
...	Nastavak naredbe u idućoj liniji	>> x=0.5, y=sin... (x) x = 0.5000 y = 0.479

%	Tekst koji slijedi iza znaka % je komentar	>> x=2 % x je varijabla x = 2

Matlab-ov prozor podijeljen je u više područja:

- **command window** – prozor koji se koristi za interaktivnu komunikaciju
- **command history** – linije unesene u glavnom (command window) području spremaju se nakon izvođenja u ovo područje za povijest naredbi iz kojeg se mogu bilo kad ponovno izvesti dvostrukim klikom.
- **launch pad** – omogućava brži pristup dostupnim alatima (*tools*), demo primjerima i dokumentaciji
- **current directory** – sve datoteke koje se pokreću, učitavaju ili spremaju, nalaze se u mapi (direktoriju ili folderu) izabranom u tom području
- **workspace** – radni prostor koji pokazuje skup trenutačno raspoloživih varijabli.

Budući da svaka nova inačica programa najviše zahvata načini baš na korisničkom sučelju, ovdje ćemo izostaviti detaljan opis ponuda (menu-a) i mogućnosti koje grafičko sučelje nudi, nego ćemo ih u obliku filmova spremiti na CD koji prati ovaj udžbenik. Promjena sučelja novih inačica mijenjat će samo odgovarajuće edukacijske filmove na CD-u.

Iako se do pomoći u radu s Matlabom može doći preko izborne ponude (Help) na vrhu Matlabovog prozora ili preko Matlabovog Helpdesk-a s CD-a koji sadrži svu Matlabovu dokumentaciju, pomoć se može dobiti i preko naredbe `help` koju utipkamo kao naredbu u naredbenoj liniji.

Matlab odgovara s popisom osnovnih područja (engl. *topics*), te popisom svih instaliranih dodatnih paketa (*toolboxes*). Ako uz naredbu `help` upišemo i neko ime iz navedenog popisa, Matlab će ispisati sve funkcije koje nam stoje na raspolaganju u tom području. Po istom pravilu će `help` s upisanim imenom funkcije dati informaciju o toj funkciji.

Tablica 2.2: *Korištenje* `help` *naredbe*

Naredba	Ispis rezultata naredbe
`>> help` %daje popis svih programskih %paketa	HELP topics: matlab\general - General purpose commands. matlab\ops - Operators and special characters. …..
`>> help ops` %daje popis svih operatora	Operators and special characters. Arithmetic operators. plus - Plus + uplus - Unary plus + …..

`>> help inv` %daje upute za funkciju inv %(inverzija kvadratne %matrice)	INV Matrix inverse. INV(X) is the inverse of the square matrix X. A warning message is printed if X is badly scaled or nearly singular.
`>> help ime_naredbe` %daje upute za navedenu %naredbu	`>> help abs` ABS Absolute value. ABS(X) is the absolute value of the elements of X. …..

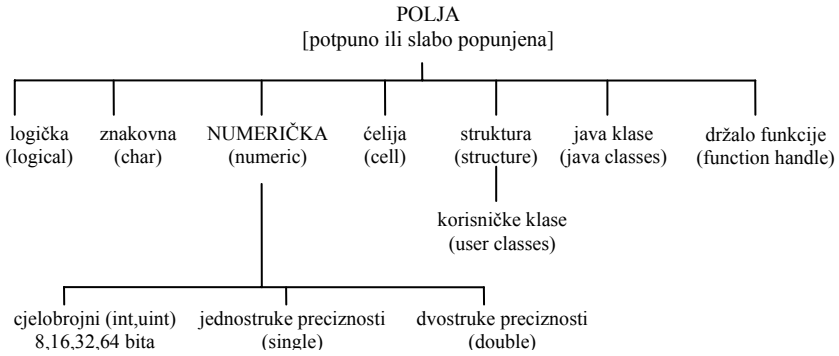
Vrlo korisna naredba za pomoć je također `lookfor` naredba, koja za zadanu ključnu riječ traži sve matlab naredbe koje u svom opisu (pomoći) imaju napisanu tu riječ. S obzirom na velik broj osnovnih i još veći dodanih naredbi, ovo pretraživanje može biti prilično dugotrajno.
Od pomoći stoje nam još na raspolaganju naredbe `info` za informacije o tvrtki MathWorks i njihovoj potpori, te naredba `demo` za upoznavanje različitih demonstracijskih programa iz različitih područja i paketa Matlaba.
Ako želite da čitav vaš razgovor s Matlab-om ostane spremljen za kasniju analizu (ili milu uspomenu), savjetujem Vam na početku razgovora uključiti dnevnik (engl. *diary*) s imenom datoteke po želji. Na primjer:
>> `diary` razg.txt
otvorit će zapisnik s imenom 'razg.txt' i u njega upisati sve (osim grafike) što se bude događalo (na zaslonu) između Vas i Matlab-a. Privremeni prekid upisa u otvorenu dnevničku datoteku postiže se s naredbom `diary`, ali bez imena datoteke. Na isti način nastavljate s upisom dnevnika u isti (već otvoreni) zapisnik.
Privremeno (ili trajno) zatvoreni zapisnik može se pročitati s naredbom `type`:
>> `type` razg.txt
ispisat će sve što je do tada u njega upisano, a to ovisi o Vašoj dotadašnjoj aktivnosti.

Tablica 2.3: *Korisne uvodne naredbe*

Naredba	
>> `demo`	%pokreće demonstracijske programe
>> `help`	%On-line dokumentacija
>> `info`	%informacija o Matlabu i tvrtki MathWorks
>> `lookfor`	%traži ključnu riječ u help informaciji
>> `type`	%ispisuje tekstualne datoteke
>> `ver`	%trenutačna inačica Matlaba i programskih paketa
>> `whatsnew`	%ispisuje README datoteku za Matlab i programske pakete
>> `which`	%pronalazi funkciju i ispisuje stazu do njene datoteke
>> `quit`	%završava rad Matlab programa

Ako smo zadovoljni već samim tim što smo u Matlab ušli, vidjeli njegov pozdrav, uočili inačicu programa i programskih paketa, pokrenuli demo programe ili tražili pomoć na različite načine, na koncu se možemo i oprostiti od njega s naredbom `quit` ili `exit`.

2.1. Vrste ili tipovi podataka



Slika 2.1: *Tipovi podataka u Matlab-u*

U Matlabu postoji 14 vrsta ili tipova podataka. Svaki od ovih tipova pojavljuje se u obliku nekog polja (engl. *array*). Najmanje polje je reda 1 x 1, a može narasti do veličine n-te dimenzije bilo koje vrste podatka. Jednodimenzionalne verzije ovih polja zovemo vektorima, a dvodimenzionalne verzije zovu se matrice. Polja mogu biti potpuna (engl. *full*) ili slabo popunjena (engl. *sparse*). U slabo popunjenim poljima stvarni podaci su vrlo rijetki s obzirom na ništične (nul) elemente, pa je spremanje takvih polja usavršeno u smislu zgusnutog zapisa kojim se štedi memorija računala. Svako polje, bez obzira koji se tip podatka u njemu nalazi ima standardni način indeksiranja (dohvaćanja elementa u polju) i automatskog proširivanja veličine. Dinamičko proširivanje polja, npr. upisom nekog elementa izvan granica zadanog polja, predstavlja važnu značajku Matlaba kao programskog jezika.

Polja se označuju alfanumeričkim imenima koja ne smiju počinjati znamenkom. Elementi se u poljima istovrsnih elemenata grupiraju unutar uglastih '[ ]' zagrada, a u poljima raznovrsnih elemenata unutar vitičastih '{ }' zagrada. Elementi se međusobno odjeljuju zarezom ili prazninom, a za prijelaz iz jednog retka matrice u drugi koristi se znak točka-zarez ';'. Dohvaćanje nekog elementa unutar polja postiže se ispisom njegovog imena i njegovog indeksa dopisanog imenu unutar običnih '( )' zagrada. Isti simboli (obične zagrade) koriste se i u grupiranju funkcijskih argumenata kod definicije i poziva funkcije, a također imaju standardno značenje kod grupiranja aritmetičkih izraza u eksplicitnom definiranju prioriteta matematičkih operacija.

Tablica 2.4: *Zagrade*

Zagrade	Opis korištenja zagrada	Upotreba zagrada
[ ]	Grupiranje vektorskih (matričnih) elemenata	>> x=[1 2 4] x = 1 2 4
{ }	Ćelije = polja različitih tipova podataka s klasičnim indeksiranjem	>> x={2, 'Miro'} x = [2] 'Miro'
( )	Indeksiranje vektora (matrica), grupiranje izraza, grupiranje funkcijskih argumenata	>> x=[1 2 4]; >>x(3) %treći element ans = 4

Primjer 2.1: *Osnovni primjeri vektora, matrice i funkcije*

Ispis vektora s tri elementa u retku, kvadratnu matricu od četiri elementa i funkciju kosinus sa argumentom 0.5.

```
>> x1=[1 2 3]   % vektor
>> x2=[1 2; 3 4]  % kvadratna matrica

>> y= cos(0.5)   %kosinus funkcija
```

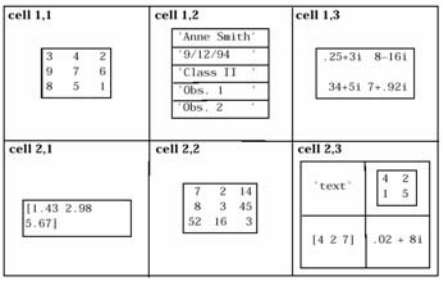
```
x1 =    1 2 3
x2 =    1 2
        3 4
y =     0.8776
```

Numerički tipovi podataka predstavljaju cijele brojeve i realne brojeve jednostruke (single) i dvostruke (double) preciznosti. Cijeli brojevi mogu zauzimati 8, 16, 32 i 64 bita u memoriji, s prvim bitom za predznak (int) ili bez njega (uint). Cijeli brojevi bez predznaka imaju dvostruko veći doseg pozitivnih brojeva od broja iste veličine s predznakom, ali nemaju negativne vrijednosti. Tako int8 obuhvaća cijele brojeve od -255 do +255, a uint8 od 0 do 511. Cijeli brojevi i realni brojevi jednostruke preciznosti zauzimaju manje memorijskog mjesta od brojeva dvostruke preciznosti, ali se kod računanja pretvaraju u brojeve dvostruke preciznosti, jer se sve matematičke operacije izvode s dvostrukom preciznosti. Kompleksni brojevi sastoje se od realnog i imaginarnog dijela označenog s imaginarnom konstantom 'i' ili 'j'. Realni i imaginarni dio mogu biti cjelobrojnog ili realnog tipa.

Logički (logical) tip podatka predstavljen je s dvije veličine: 1 i 0, gdje 1 odgovara TRUE (istina) vrijednosti, a 0 odgovara FALSE (laž). Numeričke vrijednosti različite od 0 odgovaraju vrijednosti TRUE, a samo one jednake ništici odgovaraju vrijednosti FALSE.

Znakovni (char) tip podataka sadrži Unicode (16 bitni) zapis. Tim zapisom mogu se osim uobičajenih alfanumeričkih znakova prikazati i slova drugih abeceda (grčke, ćirilične, katakana i sl.). Niz znakova zove se string, a predstavljan je kao vektor – jednoredčana ili jednostupčana matrica. Stringovi se mogu povezivati u matrice, samo ako su jednake duljine. Za polja nejednake duljine stringova potrebno je koristiti ćeliju (cell).

Polje ćelija omogućuje pohranu nejednakih vrsta podataka. Mogu se pohraniti vektori različitih tipova i/ili veličina unutar ćelije (engl. *cell*). Na primjer, možete pohraniti vektor 1 x 50 char, 7 x 13 double i 1 x 1 unit32 u jednu ćeliju. Pristup podacima u ćeliji jednako je indeksiranju bilo kojeg vektora ili matrice.



Slika 2.2: *Polje ćelija omogućuje pohranu nejednakih vrsta podataka*

Strukturalni tip (engl. *structure*) jednak je polju ćelija jer služi također za spremanje podataka različitih tipova. U ovom slučaju podaci se spremaju u imenovanim područjima, pa se dohvat podataka provodi ne po cjelobrojnom indeksu, nego po tom imenu područja. Prvo se navede ime strukture, pa točka '.', a potom ime područja. Kao podskup strukture na ovaj se način mogu definirati i Matlab klase.

Matlab omogućuje vezu s Java programskim jezikom, što znači da možemo koristiti Java klase i objekte, te koristiti metode tih objekata. *Java klasa* je Matlabov tip podatka, što znači da se mogu koristiti bilo koje već načinjene Java klase ili nove možemo načiniti pod Matlabom.
Svi relevantni podaci o funkciji (ime funkcije, argumenti, tipovi argumenata, informacija o ulaz/izlazu funkcije) spremaju se u posebnom tipu podatka koji se zove *držalo funkcije* (engl. *function handle*). Tipična primjena držala funkcije je njegova upotreba kao funkcijskog argumenta neke druge funkcije. To znači da Matlab funkcija može imati drugu funkciju kao svoj argument.

Tablica 2.5: *Opis tipova podataka*

Vrste podataka	Primjer	Opis
Realni broj jednostruke preciznosti (single)	>> 3*10^38	Numeričko polje jednostruke preciznosti. Jednostruka preciznost zahtijeva manje prostora za pohranu podataka nego dvostruka preciznost. Ovaj tip podataka služi samo za spremanje i ne koristi se u matematičkim operacijama.

Realni broj dvostruke preciznosti (double)	>> 3*10^300 >> 5+6i	Numeričko polje dvostruke preciznosti. Ovo je najčešći tip Matlab varijabli.
Raspršena, slabo popunjena matrica (sparse)	>> `speye`(5)	Raspršena (kvadratna) matrica dvostruke preciznosti. Raspršena matrica pohranjuje samo elemente različite od nule u daleko manjem memorijskom prostoru od onog koji je nužan za ekvivalentnu potpunu matricu. Raspršene matrice pozivaju posebne metode kojima se obrađuju takvi 'raspršeni' ne-nul elementi. Na primjer, matrice 10000 x 10000 elemenata sa samo par tisuća ne-nul elemenata.
Cijeli brojevi: int8, uint8 int16, uint16 int32, uint32	>> `uint8`(`magic`(3))	Polja cijelih brojeva s predznakom ili bez njega i dužinom od 8,16 ili 32 bita. Odgovarajući cjelobrojni raspon je 2^8, 2^16, 2^32. Služe za optimizirano spremanje cjelobrojnih podatka.
Znak (char)	>> 'Hello'	Polje znakova ili string (gdje je svaki znak 16 bita dug).
Ćelija (cell)	>> {17 'hello' `eye`(2)}	Elementi polja ćelije sadrže druge tipove podataka i/ili nove ćelije.
Struktura (structure)	>> a.day = 12; >> a.color = 'crvena'; >> a.mat = `magic`(3);	Strukture se sastoje od imenovanih polja ili područja. Polja sadrže bilo koje tipove podataka ili funkcije, pa odgovaraju klasama objektnih jezika.
Korisnička klasa (user class)	>> `inline`('`sin`(x)')	Matlab klasa definirana od strane korisnika napravljena je koristeći Matlab funkcije.
Java klasa (java class)	>> java.awt.frame	Java klasa koristi klase već definirane u Java API ili drugom izvoru. Također je moguće stvarati i vlastite klase u Java jeziku i koristiti ih u Matlabu.
Držalo funkcije (function handle)	>> @humps	Držalo funkcije može se proslijediti u listu argumenata neke funkcije i potom izvesti koristeći `feval()` funkciju.

2.2. Konstante i varijable

Konstanta (ili literal) je brojni iznos koji se ne mijenja, a odgovara nekom numeričkom tipu podatka. Tako razlikujemo ove tipove konstanti:
- cjelobrojne: 3, -700, 541312, ...
- realne: 4.6, 2.34e-4, -0.00123, ...
- kompleksne: 2 + 3i, 4.5 - 0.5j, 7.3e4 + 3.12j, ...

gdje su 'i' i 'j' imaginarne jedinice ($\sqrt{-1}$), točka '.' predstavlja decimalnu točku realnog broja, a 'e' označuje bazu 10 nakon koje slijedi eksponent. Tako 2.34e-4 znači 2.34 ·10-4 što je jednako 0,000234.

Varijabla ili promjenjivica je skladište u kojem se podatak tijekom rada može mijenjati. Varijabla ima ime sastavljeno od bilo koje kombinacije alfanumeričkih znakova u kojoj prvi znak ne smije biti znamenka, niti interpunkcijski znak, a sadržaj varijable ovisi o pridružbi koja je načinjena. Pridružba se postiže upotrebom operatora pridružbe '='. Tako će x=7 pridružiti cjelobrojnu konstantu 7 varijabli s imenom *x*, dok će y=2.58e-3 pridružiti realni broj 0,00258 varijabli *y*. Varijabli se može pridružiti bilo koji tip podatka, općenito polje bilo kojeg tipa. Više varijabli povezanih operatorima čine izraz. Operatori mogu biti uobičajeni algebarski operatori (za množenje, dijeljenje, potenciranje i sl.), ali i relacijski, logički, na razini bita, operatori skupova i dr.
U interaktivnom radu Matlab izvedenu pridružbu odmah ispisuje na zaslonu, osim ako ispis nije spriječen upotrebom već spomenutog znaka ';' na koncu naredbe. Matlab razlikuje velika i mala slova, pa varijabla nazvana *mojava*r nije ista što i *MojaVar*.
Ispis sadržaja varijable u bilo kojem trenutku postiže se upisom njenog imena zaključenog tipkom 'Enter'. Ako se konstanta ili neki izraz ne pridruži varijabli, onda Matlab automatski pridružuje izlaz varijabli pod imenom *ans* (od engl. *answer* – odgovor).
Ako drugačije nije navedeno Matlab će realne numeričke vrijednosti ispisati zaokružene na 4 decimalna mjesta. Ako želimo povećati ispis (na 14) decimalnih mjesta, onda koristimo naredbu `format long` ili `format e long`. Moguće je koristiti i druge formate ispisa. Format ostaje aktivan (nepromijenjen) sve do nove, drugačije `format` naredbe.

Tablica 2.6: *Naredbe za formatiranje ispisa*

Naredba	Opis naredbe	Primjer
>> `format short`	Zapis s 4 decimalna mjesta	0.2000
>> `format long`	Zapis s 14 decimalna mjesta	0.20000000000000
>> `format short e`	Eksponencijalni zapis s 4 decimalna mjesta	2.0000e-001
>> `format long e`	Eksponencijalni zapis s 14 decimalna mjesta	2.00000000000000e-001
>> `format hex`	Hexadecimalni zapis	3fc999999999999a

>> `format bank`	Zapis primjenjiv kod novčanih transakcija (dvije decimale)	0.20
>> `format rat`	Aproksimacija omjerom malih cijelih brojeva	1/5
>> `format +`	Ispisuje + odnosno minus za brojeve	0.2 -----> + -0.2 ------> - 0 ------> nema ispisa
>> `format compact`	Izbacivanje praznih linija iz ispisa rezultata	
>> `format loose`	Dodavanje praznih linija u ispis rezultata	
>> `format`	Vraćanje na standardan ispis rezultata	

Primjer 2.2: *Osnovni primjeri varijabli*

Pregled korištenja raznovrsnih imena varijabli (slovo uvijek treba biti na početku imena varijable)

```
>> a1_1=1   % skalar
>> A1_1=[1 2]  % vektor
>> Mirko_je_dobio_ocjenu=4
>> 3A=7  % nedozvoljeno 3 (broj na početku)
>> @a1_1=1 % nedozvoljeno @
>> Mirko-je_dobio_ocjenu=4 % nedozvoljeno -
```

```
a1_1 =       1
A1_1 =       1 2
Mirko_je_dobio_ocjenu =   4
Error (greška)
Error (greška)
Error (greška)
```

Osim *ans* varijable, Matlab poznaje još neke varijable kojima je pridružio konstantne vrijednosti.

Tablica 2.7: *Ime varijabli i konstanti*

Ime varijable, konstante	Opis
`ans`	Automatsko spremište rezultata koji nije pridružen varijabli
`eps`	Relativna preciznost decimalne točke. To je tolerancija koju Matlab koristi u svojim proračunima (najmanji iznos između dva realna broja). Na primjer, 2.2204e-016
`realmax`	Najveći realni broj kojeg vaše računalo može prikazati, npr. 1.797693134862316e+308.
`bitmax`	Najveći cijeli broj kojeg vaše računalo može prikazati.
`realmin`	Najmanji realni broj kojeg vaše računalo može prikazati, npr. 2.225073858507202e-308..

`pi`	3.1415926535897...
`i , j`	Imaginarna jedinica kompleksnog broja
`inf`	Beskonačnost. Na primjer, izraz n/0 gdje je n bilo koji realan broj osim nule, daje rezultat inf.
`NaN`	Nije broj (engl. ***n**ot **a** **n**umber),* nema numeričku vrijednost. Izraz kao 0/0 i inf/inf rezultiraju u NaN. Ako je n kompleksan broj, onda n/0 također vraća NaN.
`computer`	Vrsta računala
`flops`	Broj numeričkih operacija (s pomičnom točkom, engl. *floating point*) – inačica Matlaba 6 i sljedeće ne koriste više tu funkciju.
`version`	Matlab inačica

Brisanje varijable iz radnog prostora Matlaba postiže se naredbom `clear`. Naredba `clear` bez argumenata briše sve varijable iz radnog prostora. Za brisanje samo određenih varijabli potrebno je iza naredbe navesti popis varijabli koje se žele obrisati, a varijable koje se nalaze na raspolaganju možemo vidjeti upišemo li naredbu `who`. Brisanjem varijabli oslobađa se zauzeti radni prostor (memorija računala).

Spremanje varijabli i njihovih sadržaja na disk izvodi se naredbom `save`. Naredba `save` bez daljnjih argumenata sprema sve varijable iz radnog prostora Matlaba u datoteku 'moje.mat' koja se smješta u trenutačni radni folder, direktorij. U slučaju navođenja imena (i staze) datoteke za spremanje iza naredbe `save`, varijable se spremaju u navedenu datoteku. Ako se ne navede proširenje (ekstenzija) imena datoteke, onda datoteka dobiva proširenje '.mat'. Ako se iza naredbe `save` napišu imena varijabli, onda se na disk spremaju samo te varijable. Zapis na disku je u binarnom obliku, osim ako se na koncu naredbe ne napiše argument -ascii, kad se spremanje izvodi u tekstualnom obliku.
Na adekvatan način se spremljene datoteke varijabli učitavaju u Matlabov radni prostor s pomoću naredbe `load`. U slučaju da se hoće pročitati spremljena tekstualna datoteka potrebno je koristiti naredbu `type`.

Primjer 2.3: *Primjeri rada s varijablama*

Pregled naredbi za rad s varijablama (spremanje, brisanje, učitavanje).
Tekstualna datoteka (.txt) sa brojčanim podacima koji upisanim tako da tvore pravilno stupce i retke mogu se učitati u Matlab matricu. Npr. učitajmo podatke koji su u tekstualnoj datoteci (proba.txt) složeni tako da tvore dva stupca podataka koji su razdvojeni jednim razmakom (prazninom). Datoteku spremimo u radni direktorij Matlaba, tako da ne moramo upisivati putanju do nje. Onda u Matlab promptu (znači, iza >>) upišemo: `load` *proba.txt -ascii. Matlab je tada spremio sve podatke iz proba.txt u Matlab varijablu (matricu) pod istim nazivom kao i što je naziv tekstualne datoteke, a to je proba.*

```
>> a=1; b=2, c='Ante'; % ispis a i c zaustavljen s ;
>> who  % postojeće varijable u memoriji
>> clear a c  % briše varijable a i c iz memorije
>> who  % provjera varijabli u memoriji
>> a=1; c='Ante'  % varijable a i c u memoriju
>> who  % provjera varijabli u memoriji
>> save moje.mat  %sprema sve varijable a,b,c
>> save moje a c %sprema a,c u moje.mat
>> save tekst.txt c b –ascii  % c,b u tekst.txt
>> save C:\Matlab6p5\miro.txt b –ascii %sprema b
>> load moje % učitava sve varijable u memoriju
>> load moje a b % učitava varijable a,b
>> load miro.txt -ascii  %učitava b u varijablu miro
>> type tekst.txt % string Ante je u tekstualnoj
%datoteci spremljen pod odgovarajućim ascii
%brojevima (A-65, n-110, t-116, e-101)
```

```
b=   2
Your variables are:  a b c
--------------
Your variables are:  b
c= Ante
Your variables are:  a b c
--------------
--------------
--------------
--------------
--------------
--------------
--------------
--------------
65  110  116  101
2
--------------
```

2.3. Operatori i izrazi

Unarni operator (-) i binarne aritmetičke operatore (+, -, *, /,) upoznajemo već u prvim susretima s matematikom. Njihovom svezom s konstantama (literalima) i varijablama nastaju izrazi. Već smo promotrili operator pridružbe '=' koji omogućuje da rezultat izraza s desne strane pridružimo varijabli s lijeve. Matlab vodi računa o prioritetu operacija na koje korisnik može utjecati grupiranjem (pod)izraza unutar običnih '( )' zagrada.

MATLAB-ovi operatori dijele se u tri vrste:

• *Aritmetički operatori* koji izvode numeričke proračune, na primjer zbrajanje dva broja ili potenciranje nekog elementa polja na zadanu veličinu.
• *Relacijski operatori* koji uspoređuju operande kvantitativno, koristeći odnose kao što su "manji od" i "ne jednak kao" i sl.
• *Logički operatori* koji koriste logičke operacije I, ILI i NE (AND, OR i NOT), a mogu se primijeniti i na razini bita cjelobrojnih varijabli.

Svakom operatoru pripada odgovarajuća Matlab funkcija koja ima svoje ime i jedan ili više argumenata 'arg1', 'arg2' itd. Opći oblik poziva funkcije je:

`ime_funkcije` (arg1, arg2, ...)

S obzirom da je matrica temeljna Matlab struktura, treba uočiti da operator množenja '*' djeluje nad matricama na način množenja matrica (i-ti redak množi se s j-tim stupcem i

zbroj sprema na 'i,j'-to mjesto matrice rezultata), kod čega matrične dimenzije lijeve i desne matrice moraju međusobno odgovarati. Tako ćemo množenjem (m x k) matrice s (k x n) matricom dobiti (m x n) matricu. Ukoliko je kod množenja jedan od operanada skalar, tad ne postoje ograničenja na dimenzije matrice, a kao rezultat se dobije matrica čiji je svaki element pomnožen navedenim skalarom.

Treba se prisjetiti i lijevog, odnosno desnog dijeljenja s matricom, za što postoje i odgovarajući operatori, odnosno funkcije. Dijeljenje matrica odgovara množenju inverznom matricom (odnosno pseudo inverznom matricom kod nekvadratnih matrica). Desno dijeljenje matrica:

x=b/A daje rješenje jednadžbe x*A=b (x=b*A^{-1} ili x=b*`inv(A)`),

dok lijevo dijeljenje

x=A\b daje rješenje jednadžbe A*x=b (x=A^{-1}*b ili x=`inv(A)`*b).

Ako pak želimo množenje ili dijeljenje provesti na svakom odgovarajućem elementu u matrici (po mjestu gdje se pojedinačni element nalazi u polju), onda ispred operatora treba staviti točku '.', čime se dobiva tzv. Hadamarov operator.

Potenciranje matrice moguće je izvesti samo kod kvadratnih matrica, dok potenciranje među elementima matrica postavlja potrebu da matrice budu istih dimenzija, ali ne moraju nužno biti kvadratne.

2.3.1. Aritmetički operatori

Tablica 2.8: *Aritmetički operatori*

Naredba *ili* operator	Opis naredbe	Opći primjer
`plus()` *ili* +	binarni plus (zbrajanje)	`plus`(A,B) *ili* A+B
`minus()` *ili* -	binarni minus (oduzimanje)	`minus`(A,B) *ili* A-B
`uplus()` *ili* +	unarni plus (predznak)	`uplus`(A) *ili* +A
`uminus()` *ili* -	unarni minus (predznak)	`uminus`(A) *ili* -A
`mtimes()` *ili* *	matrično množenje	`mtimes`(A,B) *ili* A*B
`times()` *ili* .*	množenje elemenata u polju	`times`(A,B) *ili* A.*B
`mldivide()` *ili* \	matr. dijeljenje s lijeva	`mldivide`(A,B) *ili* A\B
`ldivide()` *ili* .\	lijevo dijeljenje u polju	`ldivide`(A,B) *ili* A.\B
`mrdivide()` *ili* /	matr. dijeljenje s desna	`mrdivide`(A,B) *ili* A/B
`rdivide()` *ili* ./	desno dijeljenje u polju	`rdivide`(A,B) *ili* A./B
`mpower()` *ili* ^	matrično potenciranje	`mpower`(A,k) *ili* A^k
`power()` *ili* .^	potenciranje nad poljem	`power`(A,k) *ili* A.^k
`ctranspose()` *ili* '	kompleksno transponiranje	`ctranspose`(A) *ili* A'
`transpose()` *ili* .'	transponiranje	`transpose`(A) *ili* A.'

gdje su:

A, matrica sa elementima a(i,j)

B, matrica sa elementima b(i,j)
C, matrica sa elementima c(i,j)
k, skalar

C=A.*B znači	c(i,j)=a(i,j)b(i,j)
C=A./B znači	c(i,j)=a(i,j)/b(i,j)
C=A.\B znači	c(i,j)=b(i,j)/a(i,j)
C=A.^B znači	c(i,j)=a(i,j)^{b(i,j)}
C=A.^k znači	c(i,j)=a(i,j)^k , k je skalar.
C=k.^A znači	c(i,j)=k^{a(i,j)}

Tablica 2.9: *Primjeri upotrebe aritmetičkih operatora*

Operacije nad matricama		**Operacije nad elementima matrice**	
A	1 2 3	B	4 5 6
A'	1 2 3	B'	4 5 6
A + B	5 7 9	A - B	-3 -3 -3
A + 2	3 4 5	A - 2	-1 0 1
A * B	Greška, jer ne odgovaraju dimenzije matrice	A .* B	4 10 18
A' * B	32	A' .* B	Greška
A * B'	4 5 6 8 10 12 12 15 18	A .* B'	Greška
A * 2	2 4 6	A .* 2	2 4 6
A \ B	16/7	A .\ B	4 5/2 2
2 \ A	1/2 1 3/2	2 ./ A	2 1 2/3
A / B	0 0 1/6 0 0 1/3 0 0 1/2	A ./ B	1/4 2/5 1/2

A / 2	1/2 1 3/2	A ./ 2	1/2 1 3/2
A ^ B	Greška	A .^ B	1 32 729
A ^ 2	Greška	A .^ 2	1 4 9
2 ^ A	Greška	2 .^ A	2 4 8
(A + i *B)'	1 - 4i 2 - 5i 3 – 6i		
(A + i *B).'	1 + 4i 2 + 5i 3 + 6i		

2.3.2. Relacijski ili odnosni operatori

Relacijski operatori su operatori odnosa među dvjema varijablama, a kao rezultat daju logičku varijablu. Logička varijabla je cjelobrojna varijabla koja ima dvije moguće vrijednosti i to: 0 (ništicu, nulu), ako operator nije zadovoljen i 1 (jedinicu) ako je uvjet zadovoljen. Ovakav način definiranja logičke varijable omogućuje da se rezultat relacijskih operatora direktno koristi kako u logičkim tako i u aritmetičkim izrazima. Relacijski operatori mogu se primjenjivati među matricama (vektorima) i među skalarima, a isto tako i između matrice (vektora) i skalara.
Kod primjene relacijskih operatora između dviju matrica, matrice moraju biti istih dimenzija, a operator djeluje među odgovarajućim elementima matrice. Rezultat je matrica istih dimenzija kao matrice između kojih je operator primijenjen. Operator između matrice i skalara daje kao rezultat matricu dimenzija jednakih ulaznoj matrici, a dobije se primjenom operatora između skalara i svakog elementa matrice. Matlab podržava sljedeće operatore:

Tablica 2.10: *Relacijski operatori*

Funkcija *ili* operator	**Značenje**	**Korištenje operatora**	**Primjeri korištenja operatora za: a=[1 2 3]; b=[4 2 6];**
`eq()` *ili* ==	jednako	`eq`(a,b) *ili* a == b	ans = 0 1 0
`ne()` *ili* ~=	nejednako	`ne`(a,b) *ili* a ~= b	ans = 1 0 1
`lt()` *ili* <	manje nego	`lt`(a,b) *ili* a < b	ans = 1 0 1
`gt()` *ili* >	veće nego	`gt`(a,b) *ili* a > b	ans = 0 0 0
`le()` *ili* <=	manje ili jednako	`le`(a,b) *ili* a <= b	ans = 1 1 1
`ge()` *ili* >=	veće ili jednako	`ge`(a,b) *ili* a >= b	ans = 0 1 0

Relacijski operatori rade s poljima za koje bilo koja dimenzija ima veličinu nula, sve dok su oba polja iste veličine ili je jedan od njih skalar (vektor reda 1 x 1). Međutim izrazi kao A == [ ] vraćaju pogrešku ako A nije 0 x 0 ili 1 x 1. Da bismo testirali je li polje A prazno polje, koristi se funkcija `isempty`(A).

2.3.3. Logički operatori

Logički operatori primjenjuju se između logičkih varijabli i kao rezultat daju logičku varijablu. Logički se operatori mogu primijeniti i između realnih varijabli, no tad se realne varijable interno pretvaraju u logičke prije samih operacija i to na način da realna '0' prelazi u logičku '0' (false), a svi ostali realni brojevi prelaze u logičku jedinicu '1'. Ukoliko su varijable među kojima se provode logički operatori matrice ili kombinacije matrica i skalara, vrijede ista pravila kao i kod relacijskih operatora. To znači da matrice moraju biti istih dimenzija, a rezultantna matrica je rezultat operacija među elementima matrica. U slučaju da je jedan operand skalar, a drugi nije, Matlab ispituje skalar prema svakom elementu drugog operanda. Mjesto gdje je zadana relacija istinita dobiva vrijednost 1, a gdje nije dobiva vrijednost 0.

Tablica 2.11: *Logički operatori*

Funkcija *ili* operator	**Značenje**	**Korištenje operatora**	**Primjeri korištenja operatora za: a=[0, 0, pi, 0]; b=[0, -3.1, 0, 1];**
`and()` *ili* &	logički I	`and`(a,b) *ili* a & b	ans = 0 0 0 0
`or()` *ili* |	logički ILI	`or`(a,b) *ili* a | b	ans = 0 1 1 1
`not()` *ili* ~	logički NE	`not`(a) *ili* ~a	ans = 1 1 0 1
`xor()`	logički ekskluzivni ILI	`xor`(a,b)	ans = 0 1 1 1

Neki izraz koji koristi operator AND (&) je istinit ako su oba operanda logički istinita. U numeričkoj terminologiji izraz je istinit ako su oba operanda različita od nule.
Neki izraz koji koristi operator OR (|) je istinit ako je barem jedan operand logički istinit. OR izraz je neistinit samo ako su oba operanda neistinita. U numeričkoj terminologiji izraz je neistinit samo ako su oba operanda jednaka nuli.

Neki izraz koji koristi NOT operator (~), negira operanda. Ovaj daje neistinit (false) rezultat ako je operand istinit, odnosno istinit (true) ako je operand neistinit. U numeričkoj terminologiji svaki operand različit od nule postaje nula, dok svaki nula operand postaje jednak jedan.

Neki izraz koji koristi XOR operator je istinit ako su njegovi operandi logički različiti, a neistinit, ako su operandi logički isti. U numeričkoj terminologiji izraz je neistinit samo

ako je jedan operand jednak nuli, a drugi jednak jedinici, a istinit, ako su oba jednaka nuli ili oba jednaka jedinici.

Logičke funkcije AND i OR česte su u programiranju kad povezuju i ispituju uvjetne izraze. U tom slučaju koriste se dvostruki simboli spomenutih operatora AND (&&) i OR (||) :

Primjer 2.4: *Upotreba short-circuit AND (&&) operatora*

Upotreba logičkog AND (&&) operatora na uspoređivanje izraza. Zahtijeva se od oba operanda da budu istinita (true) da bi cjelokupni izraz bio istinit (vrijedi i za & i za &&). Razliku između AND (&&) i AND (&) leži u tome da se uspoređivanje dvaju izraza pomoću && ponaša tako da se provjeri prvi izraz i u slučaju da je neistinit (false ili 0) drugi se izraz neće ni provjeravati nego će cijeli izraz biti neistinit. Koristi se za izbjegavanje ispisa grešaka kod izraza koji bi mogli u nekim uvjetima javljati određenu pogrešku

```
>> b=1; a=20;  %spriječen ispis
>> x = (b ~= 0) && (a/b > 18.5) % oba istinita
>> z = (b ~= 0) & (a/b > 18.5) % oba istinita
>> b=1; a=5;
>> x = (b ~= 0) && (a/b > 18.5) % drugi neistinit
>> z = (b ~= 0) & (a/b > 18.5) % drugi neistinit
>> b=0; a=5;
>> x = (b ~= 0) && (a/b > 18.5) % prvi neistinit,
%drugi dijeljenje s 0
>> z = (b ~= 0) & (a/b > 18.5)
```

```
x=  1
z=  1

x=  0
z=  0

x=  0

Warning: Divide by zero.
```

Primjer 2.5: *Upotreba short-circuit OR (||) operatora*

Upotreba logičkog OR (||) operatora na uspoređivanje izraza. Zahtijeva se da oba operanda ili samo jedan budu istinita (true) da bi cjelokupni izraz bio istinit (vrijedi i za & i za &&). Razliku između OR (||) i OR (|) leži u tome da se uspoređivanje dvaju izraza pomoću || ponaša tako da se provjeri prvi izraz i u slučaju da je istinit (true ili 1) drugi se izraz neće ni provjeravati nego će cijeli izraz biti istinit.

```
>> b=1; a=20;  %spriječen ispis
>> x = (b ~= 0) || (a/b > 18.5) % oba istinita
>> z = (b ~= 0) | (a/b > 18.5) % oba istinita
>> b=1; a=5;
>> x = (b ~= 0) || (a/b > 18.5) % drugi neistinit
>> z = (b ~= 0) | (a/b > 18.5) % drugi neistinit
>> b=0; a=5;
>> x = (a ~= 0) || (a/b > 18.5) % prvi istinit, drugi
%dijeljenje s 0
>> z = (a ~= 0) | (a/b > 18.5)
>> b=0; a=5;
>> x = (b ~= 0) || (a/b > 18.5) % prvi neistinit, drugi
%dijeljenje s 0
>> z = (b ~= 0) | (a/b > 18.5)
```

```
x=  1
z=  1

x=  1
z=  1

x=  1

Warning: Divide by zero.

Warning: Divide by zero.
--------------
Warning: Divide by zero.
```

Postoji čitav niz različitih logičkih funkcija, kao što su `all()`, `any()`, `isnan()`, `isinf()` i sl.

Tablica 2.12: *Neke dodatne logičke funkcije*

Funkcija	Opis	Primjer
`all()`	Vraća 1 ako su svi elementi u vektoru istiniti tj. različiti od nule. U slučaju matrice, stupci predstavljaju vektore.	>> A = [0 1 2;3 5 0] >> `all` (A) ans = 0 1 0
`any()`	Vraća 1 ako je bilo koji element vektora istinit. U slučaju matrice, stupci predstavljaju vektore.	>> v1 = [5 0 8]; >> `any` (v1) ans = 1

2.3.4. Operatori na razini bita

Logički operatori mogu se primijeniti i na razini bita i to tako da se provede zadana logička operacija nad svakim odgovarajućim bitom zadanih cjelobrojnih varijabli ili literala.

Tablica 2.13: *Operatori na razini bita*

Funkcija	Značenje	Korištenje funkcije	Primjeri korištenja funkcije za: **>> a=28 % (11100)** **>> b=21 % (10101)**
`bitand()`	I (AND) na razini bita	>> `bitand`(a,b)	ans = 20 % (10100)

`bitor()`	ILI (OR) na razini bita	>> `bitor`(a,b)	ans = 29 % (11101)
`bitcmp()`	komplement bitova	>> `bitcmp`(a,n) % n je broj bitova	ans = 3 % (00011)
`bitxor()`	EX-ILI (XOR) na razini bita	>> `bitxor`(a,b)	ans = 9 % (01001)
`bitset()`	Postavi bit na mjestu v na vrijednost 1	>> `bitset`(b,v) %neka je v=4	ans = 29 % (11101)
`bitget()`	Provjeri vrijednost bita na mjestu v	>> `bitget`(a,v) %neka je v=2:4	ans = 0 1 1 % (11100)
`bitshift()`	Pomak (shift) bitova za +k ili –k vrijednost	>>`bitshift`(a,k) %neka je k= 2 %neka je k= -2	ans = 112 % (1110000) ans = 7 % (111)

2.3.5. Operatori skupova

Skupovi se mogu promatrati kao elementi vektora ili matrice, nad kojima se onda mogu obavljati pripadajuće skupovne operacije. Operatori skupova predstavljeni su funkcijama.

Tablica 2.14: *Operatori skupova*

Funkcija	Značenje	Korištenje funkcije
`union()`	Unija skupova	`union`(a,b)
`unique()`	Jedinstvenost (unique) skupa	`unique`(a)
`intersect()`	Presjek (intersection) skupova	`intersect`(a,b)
`setdiff()`	Razlika (difference) skupova	`setdiff`(a,b)
`setxor()`	Ex-Ili (exclusive-or) skupova	`setxor`(a,b)
`ismember()`	Istina (True) ako je član (member) skupa	`ismember`(a,b)

Primjer 2.6: *Upotreba operatora skupova*

Pregled korištenja operatora skupova na dva različita vektora

```
>> a = [-1 0 2 4 6 2 1 4]; b = [-1 0 1 3];
>> c1 = union(a,b) % rang vrijednosti od a i b
>> c2 = unique(a) % jednostruke vrijednosti elemenata
>> c3 = intersect(a,b) % zajednički elementi
>> c4 = setdiff(a,b)%elementi u a koji nisu u b
>> c5 = setxor(a,b)%elementi koji nisu zajednički
>> c6 = ismember(a,b)% ako element iz a postoji u b
% daje vrijednost 1 inače 0
```

```
c1= -1 0 1 2 3 4 6
c2= -1 0 1 2 4 6
c3= -1 0 1
c4= 2 4 6
c5= 2 3 4 6
c6= 1 1 0 0 0 0 1 0
```

U operacijama sa skupovima i logičkim operatorima česta je upotreba funkcije `find()`. Ona vraća indekse nekog vektora u kojima je zadani uvjet zadovoljen. Tako na primjer, I = `find(A>100)`, vraća vektor indeksa u kojima vektor A ima vrijednost veću od 100.

2.3.6. Posebni operatori u matricama (vektorima)

Neke operatore u tvorbi vektora i matrice već smo upoznali. No, upisivanje elemenata matrice jednog za drugim nije lak posao, pogotovo ako je matrica većih dimenzija. Za neke primjene, kad je međusobni razmak elemenata poznat ili se od postojećih vektora (matrica) želi napraviti novi, postoje operatori i niz funkcija koje nam u tomu mogu puno pomoći.

Tablica 2.15: *Posebni operatori u matricama [vektorima]*

Funkcija ili operator	Značenje	Korištenje operatora
`horzcat()`	[a b], [a, b] nadoveza elemenata u retku	`horzcat`(a,b,...)
`vertcat()`	[a; b] stupčana nadoveza elemenata	`vertcat`(a,b,...)
`colon()` *ili* :	: raspon elemenata vektora	`colon`(a,k,b) *ili* a:k:b
`transpose()` *ili* .'	.' za transpoziciju vektora (matrice)	`transpose`(a) *ili* a.'
`ctranspose()` *ili* '	' konjugirano kompleksna transpozicija	`ctranspose`(a) *ili* a'

Na primjer, da bismo načinili stupčasti vektor (matrica sa jednim stupcem) morali bismo ubacivati poznati točka-zarez simbol između svaka dva elementa vektora ili 'Enter' tipkom pisati svaki element u novom redu. Drugi način je da napišemo jednoredčani vektor i onda ga uz pomoć operatora transpozicije (') prebacimo u stupčasti.

Za vektore u kojima je razlika između susjednih elemenata poznata, koristi se operator dvotočka ':' za njegovu tvorbu. Opći oblik je

{početna vr.} : {korak} : {konačna vr.}

A ako je korak=1 može se pisati samo

{početna vr.} : {konačna vr.}

Korak može biti i negativan, ali onda je početna vrijednost veća od konačne. No, ako je {početna vr.} manja od {konačna vr.} onda će generirani vektor biti prazan ([ ]).

Drugi način automatske tvorbe vektora je preko funkcija `linspace()` i `logspace()`. U oba slučaja prva dva argumenta su početak i dočetak vektora (u drugom kao potencije broja 10), a treći je broj elemenata između njih. Razlika je što su za prvu funkciju elementi linearno, a za drugu funkciju logaritamski međusobno udaljeni.

Primjer 2.7: *Upotreba operatora skupova*

Pregled posebnih operatora u matricama [vektorima] kao i tvorba vektora i matrica

```
>> a = [-1 0 2]; b = [0 1 3];
>> c1=horzcat(a,b)  %spaja vektore a i b horizontalno
>> c2=vertcat(a,b)  % spaja vektore a i b vertikalno

>> a1=-5; k=2; b1=4;
>> c3 = colon(a1,k,b1)  % isto kao i c3=a1:k:b1
>> z=a+i*b   % vektor s imaginarnim elementima
>> c4 = transpose(z)  % isto kao i c4=z.'


>> c5 = ctranspose(z)  % isto kao i c5=z'

>> a=3; b=7; n=4;
>> c6 = linspace(a,b,n)  % n je broj točaka od a do b
>> a=1; b=2; n=3;
>> c7 = logspace(a,b,n)  % n točaka od 10^a do 10^b
```

```
c1= -1 0 2 0 1 3
c2= -1 0 2
       0 1 3

c3= -5 -3 -1 1 3
z=-1 i 2+3i
c4= -1
        i
       2+3i
c5= -1
       -i
       2-3i
c6= 3 4.33 5.67 7

c7=10 31.62 100
```

Postoje slučajevi u kojima nam treba veličina polja ili duljina vektora. Te vrijednosti možemo dobiti na jednostavan način:

Tablica 2.16: *Funkcije za određivanje veličine (ili duljine) matrice i vektora*

Funkcija	Značenje	Korištenje funkcije	Primjeri korištenja funkcije >> a=[1 2;3 5;6 1] >> b=[1 2 3 4]
`size()`	nađi veličinu polja	`size`(a)	ans = 3 2
`length()`	nađi duljinu vektora	`length`(b)	ans = 4

Sve funkcije u Matlabu kao što je npr. `sqrt()` su "dovoljno pametne" da djeluju na varijable, skalare, vektore ili matrice. U računarstvu to se zove polimorfizam. Pojam koji označava primjenu matematičkih operatora nad različitim tipovima podataka zove se *overloading*.

Primjer 2.8: *Jednostavan primjer iz termodinamike*

Razmotrimo pojednostavljeni izraz iz termodinamike: $z = 1 + b/v + c/v^2$, gdje je v specifični volumen, b i c su konstante. Koristeći proizvoljne vrijednosti možemo izračunati faktor kompresije z za vrijednosti v u rasponu od 30 do 100.

```
>> b=12, c=1.2;
>> v=30:10:100  %vektor
%v = [30 40 … 100]
>> v2=v.^2  % kvadrat
%svakog elementa vektora v
>> z = 1 + b./v + c./v2
```

```
b= 12
v= 30 40 50 60 70 80 90 100
--------------
v= 900 1600 2500 3600 4900 6400 8100 10000
--------------
z= 1.40  1.30  1.24  1.20  1.17  1.15  1.13  1.12
```

Uočite kako smo dodali točke u zadnja dva izraza. Također uočite kako smo obavili posao bez da smo se koristili petljama. Ako želimo nacrtati *z* u ovisnosti o *v* koristili bi manji inkrementalni korak u generiranju vektora **v** te bi dodali točku-zarez kako bi izbjegli dugačak ispis tako tvorenog vektora.

U određenim slučajevima potrebno je spojiti (ili *konektirati*) male matrice i od njih načiniti velike. Na primjer, želimo načiniti matricu koja se sastoji od vektora **v1** u jednom stupcu i vektora **v2** u drugom. To postižemo ovako:

```
>> A = [v1' v2']
```

Ovdje smo prvo transponirali redove u stupce, a zatim ih spojili u matricu **A**. Ako naknadno želimo dodati treći stupac **v3** onda to možemo načiniti ovako:

```
>> format short e     % koristiti kratki eksponencijalni format
>> A = [A v3']
```

Dodali smo izraz za formatiranje kako bi koristili eksponencijalni format, jer na taj način možemo lakše pročitati brojeve. MATLAB ima loše rješenje prikaza brojeva kada su njihovi redovi veličina jako različiti.

Napravimo sada drugačiju "tablicu" sa sva tri vektora u redovima:

```
>> B = [v1 ; v2 ; v3]
```

```
>> format short       % vratimo prijašnji format
```

Točka-zarez ima ulogu odvajanja redaka. Dakako, matrica **B** je transponirana matrica **A** pa smo do istog rješenja mogli doći i na taj način.
Na isti način u matricu možemo osim vektora dodati i pojedinačne elemente.

Primjer 2.9: *Spajanje matrica i vektora*

Spajanje manjih vektora i matrica u veće matrice

```
>> v1=[8 9], v2= [4 1]; % dva vektora

>> A=[v1',v2']  %spajanje dva vektora u matricu

>> v3=[7 4.1]
>> A1= [A v3']  %dodavanje vektora matrici

>> B= [v1;v2;v3]   %spajanje tri vektora u matricu
```

```
v1= 8 9
v2= 4 1
A= 8 4
     9 1
v3= 7 4.1
A1= 8 4 7
      9 1 4.1
B= 8 9
     4 1
     7 4.1
```

Sad kada znamo spajati matrice i vektore želimo saznati kako izvući vektore i matrice iz matrica? To se postiže indeksiranjem elementa, vektora ili podmatrice:

```
v1(3)          % vrijednost trećeg elementa vektora v1
A(3,2)         % element u 3. retku i 2. stupcu matrice A
```

Interesantno je kada treba izvući redak ili stupac ili cijelu podmatricu. Da bi to postigli poslužit ćemo se operatorom dvotočka. Na primjer:

```
v1(2:4)        % izvlači od 2. do 4. elementa iz vektora v1
A(3:4,1:2)     % izvlači 2x2 podmatrica u presjeku 3, i 4. retka i 1 i 2. stupca
A(:,2)         % izvlači drugi stupac
A(2,:)         % izvlači drugi redak
```

Kada koristimo dvotočku bez brojeva na nekom indeksu, to onda znači od početka do kraja (za sve stupce ili za sve retke). Brisanje stupaca ili redaka iz matrice, vrši se tako da željene vektore napunimo nulvektorom.

```
B(:,[2])=[]        % izbriši drugi stupac
B(:,[1 3])=[]      % izbriši stupac 1 i 3
```

Primjer 2.10: *Izvlačenje matrica i vektora iz matrica*

Izvlačenje pojedinačnih elemenata, redaka ili stupaca ili cijelih podmatrica iz matrica

```
>> m=[1 2 3; 4 5 6; 7 8 9] % matrica 3x3

>> vr=m(2,:)   %izvlačenje drugog retka
>> vs=m(:,3)   %izvlačenje trećeg stupca

>> em=m(3,2)  %element iz trećeg retka i drugog stupca
>> evr=vr(1)  %prvi element iz vektora vr
>> evs=vs(2:3)  %od drugog do trećeg elementa iz vs
>> pm=m(1:2,2:3)  % podmatrica od prvog do drugog
%retka i od drugog do trećeg stupca
>> m(2:3,1)=[5;1]  %pridružba novih vrijednosti
%elementima u prvom stupcu na mjestu 2 i 3

>> m(:,[1 3])=[]  %brisanje prvog i trećeg stupca
```

```
m= 1 2 3
     4 5 6
     7 8 9
vr= 4 5 6
vs= 3
     6
     9
em= 8
evr= 4
evs= 6 9
pm= 2 3
      5 6
m= 1 2 3
     5 5 6
     1 8 9
m= 2
     5
     8
```

2.3.7. Prioritet operatora

Svaki izraz može biti bila kakva kombinacija aritmetičkih, relacijskih i logičkih operacija nad literalima, varijablama ili podizrazima. Mnogi takvi izrazi ne bi se mogli riješiti bez jasno definiranog prioriteta operacija.
Prioritet operatora definira poredak kojim Matlab izvršava izraz. Postoji lista od 9 razina prioriteta operatora. Unutar svake razine operatori imaju jednak prioritet i rješavaju se s lijeva na desno. Pravila prioriteta za Matlab operatore prikazana su na ovom popisu, a poredani su od najvišeg stupnja prioriteta prema najnižem:

Operacije unutar izraza izvršavaju se počevši od najvišeg prioriteta prema najnižem, a izrazi istog prioriteta izvršavaju se od lijeva prema desno.

Tablica 2.17: *Prioritet operatora*

Prioritet	Operatori
1	Zagrade ( )
2	Transponiranje (.'), potenciranje (.^), kompleksno konjugirano transpozicija ('), potenciranje matrice (^)
3	Unarni plus (+),unarni minus (-), logička negacija (~)
4	Množenje (.*), desno dijeljenje (./), lijevo dijeljenje (.\), desno dijeljenje matrice (/), lijevo dijeljenje matrice (\)
5	Zbrajanje (+), oduzimanje (-)
6	Operator koraka u tvorbi vektora (:)

7	Manje od (<), manje od ili jednako kao (< =), veće od (>), veće od ili jednako kao (> =), jednako kao (= =), različito od (~ =)
8	Logičko AND (&)
9	Logičko OR (|)

Primjer 2.11: *Prioritet operatora*

Pregled redoslijeda izvršavanja operacija za funkciju:
f = 11+a(x+y)^3*2*

```
>> x=2; y=5; a=3;
>> rez1 =x+y          %zagrade
>> rez2 = rez1^3      %potenciranje
>> rez3 = a*rez2      %množenje
>> rez4 = rez3*2      %množenje
>> rez5 = 11+rez4     %zbrajanje
```

```
rez1= 7
rez2= 343
rez3= 1029
rez4= 2058
rez5= 2069
```

2.4. Funkcije

Funkcije uz operatore i tipove podataka čine temelj Matlaba, a njihovo mnoštvo razvrstano po različitim toolbox-ovima čini Matlab tako snažnim matematičkim i tehničkim alatom. Funkcije se prema svom podrijetlu mogu svrstati u 3 kategorije:

- nutarnje ili interne funkcije,
- funkcije u toolbox-ima,
- funkcije definirane od korisnika.

Porijeklo funkcije može se odrediti naredbom `which` iza koje se upiše ime funkcije. Ako je funkcija nutarnja, matlab to i napiše, dok za funkcije iz toolbox-ova i definirane od korisnika ispiše mjesto na disku gdje je smještena. Funkcije koje nisu nutarnje mogu se iščitati (možemo saznati njihov programski kôd) s pomoću naredbe `type`. Za upotrebu pak funkcija njihovo podrijetlo nije bitno, jer se sve pozivaju na isti način:

`ime_funkcije`(arg1,arg2,...argn).

Osim imena funkcije potrebno je znati broj i tip argumenata koje funkcija očekuje. Ista funkcija može biti napisana tako da može prihvaćati različit broj argumenata. Svaka funkcija vraća jednu ili više vrijednosti, pa je opći oblik

[iz1, iz2, ...] = `ime_funkcije` (ul1, ul2, ...)

gdje su:

ul1, ul2, ... - ulazni argumenti u funkciju
iz1, iz2, ... - rješenja funkcije

Treba se prisjetiti da je najčešći tip podataka u Matlabu matrica, pa rješenja funkcije mogu biti izlazne varijable, vektori, matrice, ćelije, strukture i sl.
U Matlabu je omogućena rekurzija, što znači da funkcija može pozivati samu sebe (uz kontrolu završetka).
Većina Matlab funkcija napisana je i spremljena u m-datotekama koje su dostupne kako za analizu (algoritma i programa), tako i za (eventualnu) promjenu (ili nadgradnju). Svakom je korisniku pružena mogućnost da nove funkcije načini sam.
Broj i vrsta raspoloživih funkcija u Matlabu ovisi o broju instaliranih toolbox-ova. No, već i standardni toolbox-ovi sadrže velik broj funkcija među kojima možemo izdvojiti samo neke: Elementarne matematičke funkcije, Funkcije za obradu vektora i matrica, Funkcije za obradu stringova i funkcije za rad s polinomima.

Primjer 2.12: *Poziv funkcije*

*Napisat ćemo funkciju u matlabovom editoru i spremiti ju pod naziv test.m, te ju pozvati sa ulaznim parametrima (a i b) da dobijemo potrebna rješenja kao njene izlaze(zbroj, umnožak). Ime spremljene .m datoteke mora se poklapati sa imenom funkcije (**test**.m i **test**(a,b))*

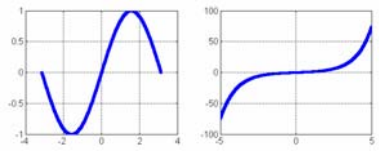
```
% sljedeća 3 retka su sadržaj datoteke test.m
function [zbroj, umnozak]=test(a,b)
zbroj= a+b;
umnozak= a*b;
% poziv funkcije test iz command windowa
>> [izlaz1, izlaz2]=test(7,3) % izlaz1 i izlaz2 su proizvoljna
%imena u koja se spremaju izlazne varijable zbroj i umnožak

% sljedeća 3 retka su sadržaj datoteke test.m
function  test(a,b) % nema izlaznih varijabli
zbroj= a+b %uklonjena je ; pa će se kod poziva test() ispisati
umnozak= a*b;
% poziv funkcije test iz command windowa
>> test(7,3)
```

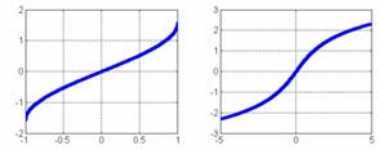
```
izlaz1= 10
izlaz2= 21

zbroj= 10
```

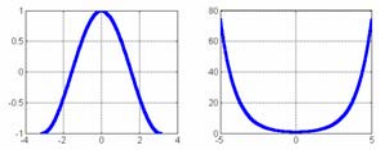
Neke elementarne matematičke funkcije:

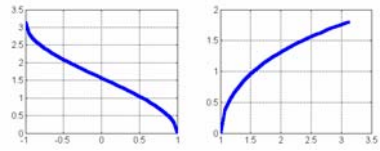
Tablica 2.18: *Neke trigonometrijske elementarne matematičke funkcije*

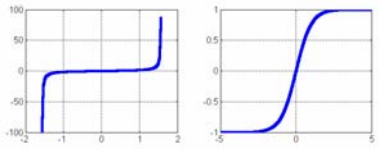
Funkcija i naziv	Primjer	Grafički prikaz funkcije

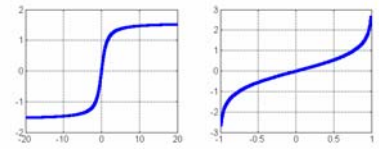
`sin()`, `sinh()`
 	
sinus, hiperbolični sinus
	x = -pi:0.01:pi;
	`plot`(x,`sin`(x))
	x = -5:0.01:5;
	`plot`(x,`sinh`(x))
		

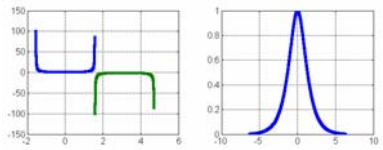
`asin()`, `asinh()`
 	
inverzni sinus, inverzni hiperbolični sinus
	x = -1:0.01:1;
	`plot`(x,`asin`(x))
	x = -5:0.01:5;
	`plot`(x,`asinh`(x))
		

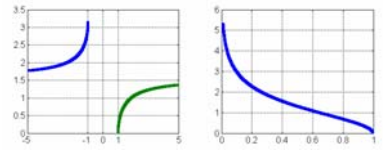
`cos()`, `cosh()`
 	
kosinus, hiperbolični kosinus
	x = -pi:0.01:pi;
	`plot`(x,`cos`(x))
	x = -5:0.01:5;
	`plot`(x,`cosh`(x))
		

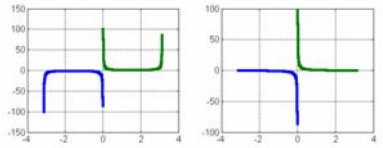
`acos()`, `acosh()`
 	
inverzni kosinus, inverzni hiperb. kosinus
	x = -1:0.05:1;
	`plot`(x,`acos`(x))
	x = 1:pi/40:pi;
	`plot`(x,`acosh`(x))
		

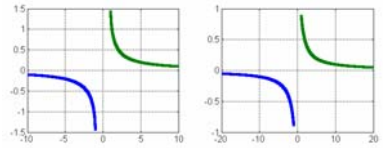
`tan()`, `tanh()` tangens, hiperbolični tangens	x = (-pi/2)+0.01… :0.01:(pi/2)-0.01; `plot`(x,`tan`(x)) x = -5:0.01:5; `plot`(x,`tanh`(x))	

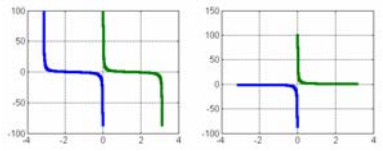
`atan()`, `atanh()`
 	
inverzni tangens, inverzni hiperb. tangens
	x = -20:0.01:20;
	`plot`(x,`atan`(x))
	x = -0.99:0.01:0.99;
	`plot`(x,`atanh`(x))
		

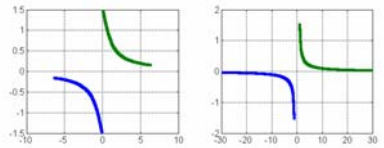
`sec()`, `sech()` sekans, hiperbolični sekans	x1 = -pi/2+0.01… :0.01:pi/2-0.01; x2 = pi/2+0.01… :0.01:(3*pi/2)-0.01; `plot`(x1,`sec`(x1)… ,x2,`sec`(x2)) x = -2*pi:0.01:2*pi; `plot`(x,`sech`(x))	

`asec()`, `asech()`
 	
inverzni sekans, inverzni hiperbolični sekans
	x1 = -5:0.01:-1;
	x2 = 1:0.01:5;
	`plot`(x1,`asec`(x1)…
	,x2,`asec`(x2))
	x = 0.01:0.001:1;
	`plot`(x,`asech`(x))
		

`csc()`, `csch()`
 	
sekans, hiperbolični sekans
	x1 = -pi+0.01:0.01:-0.01;
	x2 = 0.01:0.01:pi-0.01;
	`plot`(x1,`csc`(x1)…
	,x2,`csc`(x2))
	x1 = -pi+0.01:0.01:-0.01;
	x2 = 0.01:0.01:pi-0.01;
	`plot`(x1,`csch`(x1)…
	,x2,`csch`(x2))
		

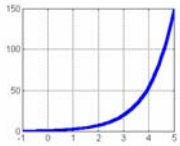
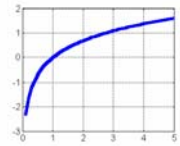
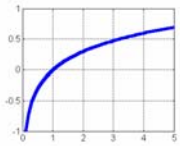
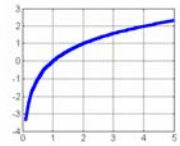
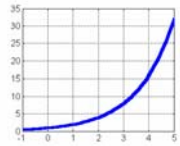
`acsc()`, `acsch()`
 	
inverzni kosekans, inverzni hiperb. kosekans
	x1 = -10:0.01:-1.01;
	x2 = 1.01:0.01:10;
	`plot`(x1,`acsc`(x1)…
	,x2,`acsc`(x2))
	x1 = -20:0.01:-1;
	x2 = 1:0.01:20;
	`plot`(x1,`acsch`(x1)…
	,x2,`acsch`(x2))
		

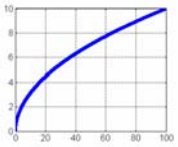
`cot()`, `coth()`
 	
kotangens, hiperbolični kotangens
	x1 = -pi+0.01:0.01:-0.01;
	x2 = 0.01:0.01:pi-0.01;
	`plot`(x1,`cot`(x1)…
	,x2,`cot`(x2))
	x1 = -pi+0.01:0.01:-0.01;
	x2 = 0.01:0.01:pi-0.01;
	`plot`(x1,`coth`(x1)…
	,x2,`coth`(x2))
		

`acot()`, `acoth()`
 	
inverzni kotangens, inverzni hiperb. kotangens
	x1 = -2*pi:pi/30:-0.1;
	x2 = 0.1:pi/30:2*pi;
	`plot`(x1,`acot`(x1)…
	,x2,`acot`(x2))
	x1 = -30:0.1:-1.1;
	x2 = 1.1:0.1:30;
	`plot`(x1,`acoth`(x1)…
	,x2,`acoth`(x2))
		

`atan2()` inverzni tangens u četvrtom kvadrantu	z= 4 + 3i; y= `atan2`… (`imag`(z),`real`(z)) % ili y1= atan(3/4)	y= 0.6435 y1= 0.6435

Tablica 2.19: *Neke eksponencijalne elementarne matematičke funkcije*

Funkcija i naziv	Primjer	Grafički prikaz funkcije
`exp()` eksponencijalna	x=-1:0.1:5; `plot`(x,`exp`(x))	
`log()` prirodni logaritam	x=0.1:0.1:5; `plot`(x,`log`(x)	
`log10()` obični (baza 10) logaritam	x=0.1:0.1:5; `plot`(x,`log10`(x))	
`log2()` logaritam po bazi 2	x=0.1:0.1:5; `plot`(x,`log2`(x)	
`pow2()` potencija po bazi 2	x=-1:0.1:5; `plot`(x,`pow2`(x))	

`realsqrt()` kvadratni korijen broja >= 0	A=[1 2 3; 4 5 6; 9 9 9];	A= 1.000 1.414 1.732 2.000 2.236 2.449 3.000 3.000 3.000
`sqrt()` kvadratni korijen	x=0:0.1:100; `plot`(x,`sqrt`(x))	

Tablica 2.20: *Neke kompleksne elementarne matematičke funkcije*

Funkcija i naziv	Primjer	Ispis primjera
`abs()` apsolutna vrijednost	y1= `abs`(-5) y2= `abs`(3+4i) % sqrt(real(X).^2 + imag(X).^2)	y1 = 5 y2 = 5
`angle()` fazni kut	Z = [1-i 2+i; 3-i 4+i]; A=`angle`(Z) %angle(Z)=atan2(imag(Z),real(Z))	A= -0.7854 0.4636 -0.3218 0.2450

`complex()`
 	
komplesni broj iz realnog i imaginarnog dijela
	a = [1;2;3;4];
	b = [2;2;7;7];
	c = `complex`(a,b)
		c =
		1.0000 + 2.0000i
		2.0000 + 2.0000i
		3.0000 + 7.0000i
		4.0000 + 7.0000i

`conj()` konjugirani kompleksni broj	Z = [1-i 2+i; 3-i 4+i]; C= `conj`(Z)	C= 1+i 2-i 3+i 4-i
`imag()` imaginarni dio kompleksnog broja	Z = [1-i 2+i; 3-i 4+i]; I= `imag`(Z)	I= -1 1 -1 1
`real()` realni dio kompleksnog broja	Z = [1-i 2+i; 3-i 4+i]; R= `real`(Z)	R= 1 2 3 4
`isreal()` true (istina) za realna polja	F1= 74; y1= `isreal`(F1) F2= 2+3i; y2= `isreal`(F2)	y1=1 y2=0

Tablica 2.21: *Neke elementarne matematičke funkcije za zaokruživanje i ostatak*

Funkcija i naziv	Primjer	Ispis primjera
`fix()` zaokruživanje prema nuli	A = [-1.9, -0.2, 2.4+3.6i]; y= `fix`(A)	y =-1.0 0 2.0 + 3.0i
`floor()` zaokruživanje prema minus beskonačnom	A = [-1.9, -0.2, 2.4+3.6i]; y= `floor`(A)	y = -2.0 -1.0 2.0+ 3.0i
`ceil()` zaokruživanje prema plus beskonačnom	A = [-1.9, -0.2, 2.4+3.6i]; y= `ceil`(A)	y = -1.0 0 3.0+ 4.0i
`round()` zaokruživanje na najbliži cijeli broj	A = [-1.9, -0.2, 2.4+3.6i]; y= `round`(A)	y = -2 0 2.0+ 4.0i
`rem()` ostatak nakon dijeljenja	a=13; % broj b=5; % djelitelj y= `rem`(a,b) y1=`rem`(-a,b)	y = 3 % 13/5=2 i ostatak 3 y1= -3 % -13/5=-2 i ostatak -3
`mod()` modulo funkcija (kao rezultat ostatka dijeljenja	a=13; % broj b=5; % djelitelj y= `mod`(a,b) y1=`mod`(-a,b)	y = 3 % 13/5=2 i ostatak 3 y1= 2 % -13/5=-2 i ostatak -3, %pa se ostatak -3 zbroji s %djeliteljem 5 i dobije se 2
`sign()` funkcija predznaka	X=[1 -12 -1 6]; y=`sign`(X)	y= 1 -1 -1 1

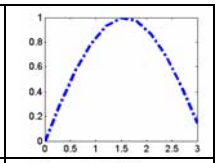
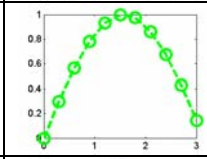
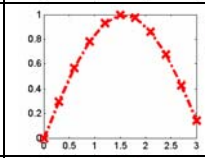
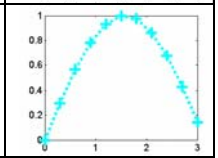
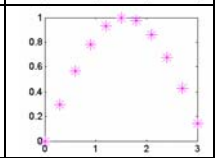
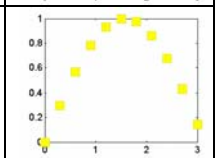
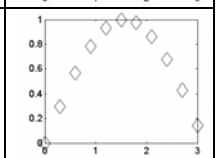
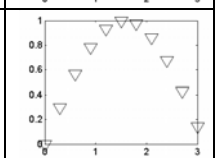
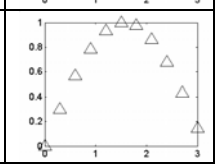
Osnovna grafička naredba je plot() funkcija čija je sintaksa:

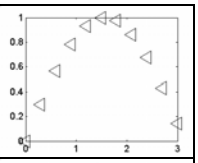
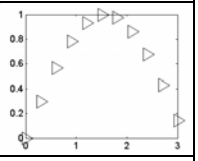
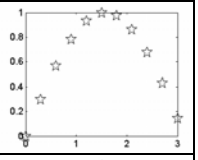
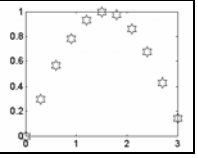
`plot`(X,Y) crta vektor Y prema vektoru X.

Različiti tipovi linija, simbola i boja mogu se postići s naredbom `plot`(X,Y,S) gdje je S niz znakova (string) dobiven iz bilo kojeg od tri (ili svih) stupaca ispod:

Tablica 2.22: *Osnovna grafička naredba* `plot()` *i njene karakteristike*

Boja linije	Stil linije	Oznake (markeri) na liniji	Primjer: >> x=0:0.3:pi; >> y=sin(x);	Grafički prikaz

b plava (blue)	**-** čvrsto (solid)	**.** točka (point)	`plot`(x,y,'b-.')	
g zelena (green)	**--** crtkano (dashed)	**o** kružić (circle)	`plot`(x,y,'g--o')	
r crvena (red)	-. crta točka (dash-dot)	**x** x-znak (cross)	`plot`(x,y,'r-.x')	
c cyan (cyan)	**:** točkasto (dotted)	+ plus (plus sign)	`plot`(x,y,'c:+')	
m magenta (magenta)		*** zvijezda (asterisk)	`plot`(x,y,'m*')	
y žuta (yellow)		**s** kvadratić (square)	`plot`(x,y,'ys')	
k crna (black)		**d** romb (diamond)	`plot`(x,y,'kd')	
		v trokut dolje (triangle down)	`plot`(x,y,'kv')	
		^ trokut gore (triangle up)	`plot`(x,y,'k^')	

		< trokut lijevo (triangle left)	`plot`(x,y,'k<')	
		> trokut desno (triangle right)	`plot`(x,y,'k>')	
		p pentagram (pentagram)	`plot`(x,y,'kp')	
		h heksagram (hexagram)	`plot`(x,y,'kh')	

Na primjer, `plot`(X,Y,'c+:') crta cyan točkastu liniju s + znakom na svakoj točki; `plot`(X,Y,'bd') crta rombove na svakoj točki podatka, ali ih ne povezuje linijom.
`plot`(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) kombinira grafove definirane trojcima (X,Y,S), gdje su X i Y vektori ili matrice, a S su stringovi.

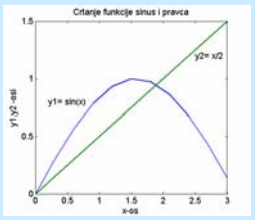
Na primjer, `plot`(X,Y,'y-',X,Y,'go') crta podatke dvaput, s punom žutom linijom interpolirajući zelenim kružićima točke podataka.

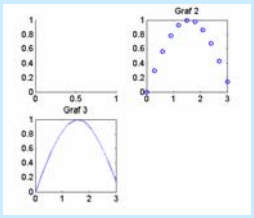
Uz `plot()` obično se koriste i funkcije:
`title`('*tekst*') dodaje *tekst* na vrh grafa.
`xlabel`('*tekst*'), `ylabel`('*tekst*') dodaje *tekst* uz X-os ili Y-os.
`text`(X,Y,'*string*') dodaje *string* u navodnicima na (X,Y) poziciju.
`subplot`(n,m,k) crta k-ti podgraf u (n x m) matrici grafova

Primjer 2.13: *Označavanje grafa i crtanje podgrafova na jednoj slici (figure)*

Pregled načina označavanja grafova, crtanje višestrukih podgrafova (subplots)

```
>> x=0:0.3:pi;
>> y1=sin(x); y2=x/2;
>> plot(x,y1,x,y2)  %prva slika
>> title('Crtanje funkcije sinus i pravca')
>> xlabel('x-os'), ylabel('y1,y2 -osi')
>> text(0.2,0.8,'y1= sin(x)')
>> text(2.5,1.2,'y2= x/2')
>> subplot(2,2,2)   %druga slika
>> plot(x,y,'o')
>> title('Graf 2')
>> subplot(2,2,3)   %druga slika
>> plot(x,y,':')
>> title('Graf 3')
>> subplot(2,2,1)   %druga slika
```





M-DATOTEKE: SKRIPTE I FUNKCIJE 3

Sve naredbe koje smo dosad interaktivno unosili u Matlab-ovu naredbenu liniju mogu se izvesti i tako da se spreme u datoteku (zapisnik) s pomoću Matlabovog editora (= programa za upis i promjenu teksta) ili nekog drugog ASCII tekst editora, a potom se datoteka pozove iz Matlab-a. Koliko god puta to načinili, slijed naredbi iz te datoteke izvest će se na isti način. Takvu datoteku nazivamo *skripta* (engl. *script-files*). Skripta nema konstrukcijsku mogućnost prenošenja ulaznih podataka u naredbe koje se u skripti nalaze, nego radi s postojećim varijablama radnog prostora, uz mogućnost stvaranja novih varijabli.

Druga mogućnost je stvaranje korisničkih funkcija, koje se po pozivu ne razlikuju od nutarnjih (Matlabovih) funkcija, a po ustroju odgovaraju funkcijama od kojih su napravljeni svi toolbox-i. Takve funkcije imaju konstrukcijsku mogućnost prenošenja ulaznih podataka u naredbe koje te podatke obrađuju, te na koncu vraćaju rezultat.
Budući da Matlab automatski dodaje '.m' za proširenje imena ovakvih datoteka (skripta i funkcija), ove se datoteke još nazivaju *m-datoteke* (engl. *m-files*).

3.1. M-skripta

Kod poziva skript-datoteke, u naredbenoj se liniji napiše samo njeno ime (bez '.m' proširenja!), nakon čega se njezine spremljene naredbe izvršavaju jedna za drugom. Na zaslonu se pojavljuju samo rješenja koje naredbe proizvode, a ne i same naredbe. Da bismo istodobno pratili i naredbe i njihovo izvršenje potrebno je uključiti ispis s naredbom `echo on`. Poništenje se postiže isključenjem iste naredbe - `echo off`. Bilo koji tekst koji je komentar u skripti ne ispisuje se na zaslonu. Popis m-datoteka u trenutačnom imeniku (folderu, direktoriju) dobiva se s pomoću naredbe `what`.

Primjer 3.1: *M- skripta*

Imamo situaciju u kojoj često moramo grafički prikazati ovisnost funkcije x o y za različite skupove mjerenih podataka, pa stoga želimo automatizirati taj posao grafičkog prikazivanja uz pomoć M-datoteke.

```
% ___ M-file skripta: plotxy.m ____
% Skripta za grafički prikaz ovisnosti x o y  uz dodavanje oznaka na
%grafu (x,y-osi i naslov).
```

```
plot(x,y)
grid
xlabel('Vrijeme [min]')
ylabel('Odgovor')
title('PID Simulacija')
% Kraj plotxy.m -  end izraz nije potreban.
```

Spremimo to u datoteku pod imenom '*plotxy.m*'. Sve što je iza "%" znaka smatra se komentarom koji se može, a ne mora staviti. Kao što znamo, on služi samo korisniku za opis što i kako program radi. Nakon što smo u naredbenom prozoru definirali nove ili promijenili stare vrijednosti za x i y (bilo da smo ih utipkali kao vektore ili učitali mjernim uređajima), sve što je potrebno učiniti je upisati `plotxy` i MATLAB će obaviti svoj dio. Važno je napomenuti da ovakva M-datoteka nema mehanizama za učitavanje ili eksplicitno unošenje podataka, u ovom slučaju vektora x ili y vrijednosti. To će se moći s pomoću m-funkcija. Sve naredbe iz skriptne m-datoteke izvršavaju se u naredbenom prozoru i mogu raditi sa svim do tada poznatim varijablama. Isto tako, sve varijable koje stvori skripta, ostaju u radnom prostoru i mogu se vidjeti u 'workspace' prozoru ili pozivom `who` ili `whos` naredbi.
Kad radimo s m-datotekom njen *put* (*staza*, engl. *path*) mora biti definiran u Matlab-ovom popisu putova po kojem pretražuje. Matlab po definiciji najprije traži u folderu iz kojega je on sam startan i u trenutačnom folderu koji je odabran u 'current directory' prozoru. Dobra praksa je držati sve što napravimo na jednom mjestu, u jednom folderu, te se pozicionirati u taj folder preko 'current directory' prozora. Druga je mogućnost naredbom `path` mijenjati put (stazu) ili je ugraditi u 'startup.m' datoteku koju Matlab izvršava na početku rada. U njoj su navedene i staze svih toolbox-a koji su instalirani zajedno s Matlabom.
Sadržaj bilo koje m-datoteke može se ispisati s pomoću `type` naredbe. Dakako, ime datoteke kod ispisa mora sadržavati i '.m' nastavak.

S obzirom da skripta nema mogućnost prenošenja ulaznih podataka, osim onih koje se već nalaze u varijablama radnog prostora, kod skripta je česta upotreba `input()` funkcije koja se stavlja obično među prve naredbe i preko nje učitavaju željeni podaci.
Ova naredba ima opći oblik:

vr = `input`('poruka')

gdje se ispisuje 'poruka' na zaslon i očekuje unos podataka. Poruka obično govori o kojoj vrsti ulaznih podataka je riječ. Ako se očekuje unos varijable, onda se ona upisuje kao literal očekivanog tipa, a ako se očekuje unos vektora, onda korisnik mora upisati elemente vektora unutar uglastih zagrada, kao što je to uobičajeno u radu s vektorima (ili matricama).

Primjer 3.2: *Naredba* `input` *u M- skripti*

```
Ovo je m-datoteka nazvana 'test.m' koja koristi trapezno pravilo za računanje
```

integrala. Egzaktna vrijednost integrala je 1/3, a korak 'h' djeluje na točnost. Provjerite algoritam pozivom skripte s različitim vrijednostima 'h' varijable

```
h=input('korak integracije h = ');
x=(0:h:1);
lx=length(x);
y=x.^2;
integral=(h/2)*(y(1)+2*sum(y(2:(lx-1)))+y(lx) )
```

Ako želimo u skriptu interaktivno unijeti string varijablu, onda u `input` naredbi trebamo dodati 's' argument:

```
st = input('poruka','s')
```

Često se u skript datotekama koristi i naredba `pause` koja zaustavlja izvođenje programa na određeno vrijeme ili do pritiska bilo koje tipke.

Primjer 3.3: *Naredba* `pause` *u M- skripti*

Skriptna m-datoteka za proizvodnju 'cvjetnih' grafova

```
theta = -pi : 0.01: pi ;                % kut 'theta'
rho (1,:) = 2*sin(5*theta).^2;          % kut 'rho'
polar(theta,rho(1,:))                   % graf u polarnim koordinatama
pause (2);                               % pauza od 2 sekunde
rho (2,:) = cos(10*theta).^3;
polar(theta,rho(2,:))
pause(3);                                % pauza od 3 sekunde
rho (3,:) = sin(theta).^2;
polar(theta,rho(3,:))
pause;                                   % pauza do pritiska tipke
rho (4,:) = 5*cos(3.5*theta).^3;
polar(theta,rho(4,:))
pause(2);
clf;                                     % brisanje grafike
close all;                               % zatvaranje grafičkog prozora
```

3.2. M-funkcija

M- funkcija je m-datoteka koja dopušta ulazne argumente i vraća izlazne argumente. Ona radi s varijablama u svom vlastitom radnom prostoru i sve njene varijable (osim

globalnih) se brišu čim funkcija završi izvođenje. Globalne varijable su takve varijable koje dopuštaju čitanje sadržaja i mijenjanje iz različitih funkcija i skripta. Ispred njihovog imena definiranog unutar m-funkcije nalazi se ključna riječ '*global*'.

Funkcije se zapisuju kao ASCII datoteke na disk, a njihovo ime obavezno mora imati ekstenziju m. Ime funkcije treba odgovarati imenu datoteke bez ekstenzije, u koju se funkcija sprema. Da bi se izbjegla kolizija s postojećim funkcijama Matlaba, korisničkim je funkcijama potrebno davati različita imena od imena internih funkcija i funkcija u toolboxovima. Također, kako je već spomenuto, da bi se funkcija mogla izvoditi, mora biti smještena u trenutačni direktorij ili u direktorij koji je u putu pretraživanja Matlaba.

M-funkcija ima sljedeći oblik:

```
function izl_var = ime_funkcije(ul_var_1, ul_var_2, ... ul_var_n)
% linije komentara koji se ispisuje s help naredbom
funkcijski kod – izrazi, kontrolne strukture i pozivi drugih funkcija
izl_var = izraz konačnog rješenja;
```

Zaglavlje funkcije počinje propisanom riječju '**function**' iza koje slijedi ime izlazne varijable ili niza varijabli koje funkcija vraća kao rezultat. Ako funkcija vraća više varijabli, tada se umjesto imena jedne izlazne varijable postavlja definicija izlaznog vektora. Na primjer, vektor od tri izlazne varijable imao bi oblik, od kojih svaka, dakako, može sadržavati polje bilo kojeg tipa podatka:

```
[izl_1, izl_2, izl_3].
```

Iza imena izlazne varijable ili vektora varijabli slijedi znak jednakosti i ime funkcije, a zatim u okruglim zagradama slijedi popis ulaznih varijabli koje će služiti kao argumenti funkcije kod njenog poziva. Ulazne varijable nazivaju se također formalni argumenti i parametri, s kojima se radi u tijelu funkcije (u programskom kodu koji slijedi). Svaki formalni argument bit će zamijenjen stvarnim argumentom kod poziva funkcije.

U novom, drugom po redu i idućim redovima slijedi komentar s opisom funkcije. Komentar počinje znakom %. Ovaj komentar, sve do prve naredbe, predviđen je kao pomoć (*help*) za tu funkciju. Iza njega slijedi niz naredbi kojima se na temelju ulaznih varijabli određuje izlazna. Svakoj izlaznoj varijabli mora se u funkciji pridružiti izračunata vrijednost, da bi se nakon poziva funkcije moglo doći do rezultata .
M-funkcija se poziva interaktivno, iz naredbenog prozora Matlaba ili iz druge funkcije tako da se napiše njeno ime, a u zagradi se navedu potrebne vrijednosti ulaznih argumenata.
Jedna takva jednostavna funkcija s imenom `pov_trokuta()` s tri ulazna i jednim izlaznim argumentom prikazana je u sljedećem primjeru:

Primjer 3.4: *M- funkcija*

Programski kod (napisan u nekom tekstualnom editoru i spremljen pod .m ekstenziju) za funkciju koja će izračunavati površinu trokuta

```
function [A]=pov_trokuta(a,b,c)
%Izracunava povrsinu trokuta sa stranicama duzine a, b, c
% Ulazi:  a, b,c – duljine stranica
% Izlaz:   A – povrsina trokuta
% Poziv:  Pov = pov_trokuta (x,y,z)
% Napisao M.M., FSB Zagreb, 05.03.2003.
s = (a+b+c)/2;
A=sqrt(s*(s-a)*(s-b)*(s-c));
%------------ konac -------------------------
```

Ako smo osim površine A željeli dobiti i sadržaj varijable s, to smo mogli postići jednostavnim proširenjem izlaznih argumenata funkcije, promjenom početne linije u

function [A,s]=pov_trokuta(a,b,c).

Da bi dobili pomoć o nekoj funkciji (u ovom slučaju korisničkoj funkciji `pov_trokuta`) koristimo naredbu `help` , a kao ispis pomoći dobit ćemo točno ono što smo uz pomoć komentara od druge linije do prve naredbe napisali. Takvim jednostavnim načinom riješeno je temeljno dokumentiranje Matlab funkcija.

Primjer 3.5: *Poziv pomoći (help) za korisničku funkciju* `pov_trokuta`

Dokumentiranje funkcije `pov_trokuta`

```
>> help pov_trokuta
```

```
Izračunava površinu trokuta sa stranicama dužine a,b,c
Ulazi:  a, b,c – duljine stranica
Izlaz:   A – površina trokuta
Poziv:  Pov = pov_trokuta(x,y,z)
Napisao M.M., FSB Zagreb, 05.03.2003.
```

Primijetite nadalje da funkcija mora biti spremljena na disk pod imenom pov_trokuta.m, dok poziv funkcije može izgledati na primjer ovako:

A1= `pov_trokuta`(2,5,6) ili A1= `pov_trokuta`(14,11.2,17.8).

U jednom slučaju Matlab će dati jedno (4.6837), a u drugom slučaju drugo rješenje (78.3916), po dobro poznatoj Heronovoj formuli. Međutim, korisno je uočiti kako napisani algoritam nije dovoljno zaštićen: loši ulazni podaci dat će nepredviđene rezultate,
npr. A1= `pov_trokuta`(2,5,1) daje 0 + 4.8990i što je istina s obzirom na kvadratni korijen iz negativnog broja, ali ne i s obzirom na površinu trokuta koji se uz ovakve

podatke ne može konstruirati. Dakle, algoritam bi trebao voditi računa o točnosti ulaznih podataka.
Idući primjer average.m, za traženje srednje vrijednosti ulaznog vektora, vodi računa o ulaznim podacima:

Primjer 3.6: *Korisnička funkcija za traženje srednje vrijednosti ulaznog vektora*

Traženje srednje vrijednosti za podatke ulaznog vektora, pri čemu se vodi računa o točnosti tih podataka. Ime datoteke je average.m, što je u biti i ime funkcije.

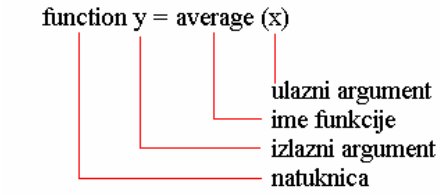
```
function y = average (x)
%  AVERAGE srednja vrijednost vektorskih elemenata.
%  AVERAGE (X), gdje je je X vektor
%  Ne-vektorski ulaz rezultira pogreskom
[m, n] = size (x);
if  (~((m = = 1) | (n = = 1)) | (m = = 1 & n = = 1))
  error ( 'Pogreska: ulaz mora biti vektor!')
end
% Izlaz y = zbroj_elemenata/duljina_ vektora
y = sum (x)/length (x);
```

Prvo se uzimaju dimenzije (size) ulaznog podatka, a potom se provjerava uvjet (*ako*, engl. *if*) - radi li se o redčanom ili stupčanom vektoru. Ako uvjet nije ispunjen poziva se funkcija `error()` kojom se ispisuje pogreška, kako bi korisnik znao što treba unijeti. Poziv ove funkcije može biti ovakav:

```
>> z = 1 : 99 ;      % prvo definiramo vektor
>> average(z)       % pa pozovemo funkciju s vektorom kao argumentom.
```

Funkcija ispisuje rezultat:

```
ans =
        50
```



Slika 3.1: *Opis načina rada s funkcijom*

Kao što **Pogreška! Izvor reference nije pronađen.** pokazuje, ulazni argument x u pozivu funkcije `average`., zamijenjen je stvarnim argumentom z, a izlazni argument y pridružen je nutarnjoj varijabli ans. Ime funkcije (`average`) mora odgovarati (biti jednako!) imenu datoteke na disku (average.m) u kojoj je funkcija spremljena, a natuknica **function** govori u definiciji m-datoteke da se radi o funkcijskoj konstrukciji, a ne o m-skripti.

Unutar funkcija moguće je definirati i druge podfunkcije koje međusobno mogu pozivati jedna drugu. Svaka podfunkcija vidljiva je i dostupna svim podfunkcijama definiranima u istoj m-datoteci. M-datoteka zove se po prvoj funkciji koja je u njoj upisana i ona je vidljiva drugim funkcijama unutar i izvan m-datoteke, dok ostale podfunkcije nisu vidljive izvan datoteke u kojoj su definirane.

Primjer 3.7: *Poziv podfunkcije iz funkcije*

Funkcija `stat()` poziva funkciju `avg()` koja je definirana u njoj kao podfunkcija.

```
function [mean,stdev] = stat(x)
%STAT - osnovna statistika.
n = length(x);
mean = avg(x,n);
stdev = sqrt(sum((x-avg(x,n)).^2)/n);
%---------------------------------
function mean = avg(x,n)
%MEAN subfunction
mean = sum(x)/n;
```

Primijetite kako je obje funkcije imaju varijablu s istim imenom (mean), ali te varijable ne dijele isti memorijski prostor. Isti prostor dijelile bi u slučaju kad bismo ih definirali kao globalne (s atributom global).
Međusobni pozivi funkcija mogu biti i rekurzivni, na primjer. Prva funkcija poziva drugu, a druga prvu. Dakako, moguća je i autorekurzija, kad funkcija poziva samu sebe, kao što je to slučaj u klasičnoj funkciji za izračunavanje faktorijela. Po definiciji znamo:

$$fact(0)=1 \quad i \quad fact(n) = n * fact(n-1) \text{ za } n>2$$

što se lako pretvara u matlabov program, funkciju `fact`() koja se sprema u datoteku fact.m:

Primjer 3.8: *Korisnička funkcija `fact` (autorekurzivna) za izračun faktorijela*

Funkcija `fact` je autorekurzivna korisnička funkcija koja poziva samu sebe

```
function y = fact(n)
% FACT - autorekurzivna korisnicka funkcija
% fact(n) = n * fact(n-1), fact(0) = 1;
if (n == 0)
  y = 1;
else
  y = n * fact(n-1);
end
```

Pozivom funkcije sa stvarnim argumentom jednakim 5, dobit ćemo:

```
>> fact(5)
ans =
            120
```

Doista, 5! = 120.

Primjer 3.9: *Korisnička funkcija `nrfact` (nerekurzivna) za izračun faktorijela*

Funkcija nr`fact` je nerekurzivna korisnička funkcija i koristi se programskom petljom za izračun faktorijela

```
function y = nrfact(n)
% NRFACT - nerekurzivni algoritam za faktorijele
y = 1;
for i=2:n
  y = y * i;
end
```

Očevidno, treba nam nešto programiranja. Tu već susrećemo prvu programsku petlju (`for`) koja 'vrti' naredbu y=y*i točno n-1 puta. Hoće li nerekurzivni i rekurzivni algoritam dati isti rezultat? Dakako!

```
>> nrfact(5)
ans =
            120
```

Na koncu spomenimo i dvije vrlo važne varijable pridružene funkcijama. Jedna govori o broju ulaznih argumenata i zove se `nargin` (**n**umber of **arg**uments, **in**put), a druga - `nargout` (**n**umber of **arg**uments, **out**put) sadrži broj izlaznih argumenata funkcije. To omogućuje programsku kontrolu svakog pojedinačnog argumenta unutar tijela funkcije.

Primjer 3.10: *Kontrola broja ulaznih i izlaznih argumenata (nargin i nargout)*

Pregled funkcije koja ima četiri mogućih izlaznih argumenata i jedan ulazni

```
function [x_sort, x_mean, x_med, x_std]=marks(x)
% MARKS  statistička analiza vektora
x_sort=sort(x);
if nargout>1, x_mean=mean(x); end
if nargout>2, x_med=median(x); end
if nargout>3, x_std=std(x); end
```

Razmotrimo ponašanje funkcije `marks()` (Primjer 3.10) s obzirom na različit broj izlaznih argumenata (Primjer 3.11):

Primjer 3.11: *Ponašanje funkcije s varijabilnim brojem izlaznih argumenata*

Pozivanje funkcije `marks()` (Primjer 3.10) u svrhu izračunavanja zahtijevanih izlaznih argumenata na temelju vektora ulaznog argumenta. Ako u pozivu funkcije zahtijevamo npr. tri izlazna argumenta onda je `nargout`=3, bez obzira što u marks.m (tj. kodu funkcije `marks()` postoje moguća 4 izlazna argumenta).

```
>> marks([12 3 4 5 -4 5])
>> [a,b]=marks([12 3 4 5 -4 5])

>> [a,b,c]=marks([12 3 4 5 -4 5])


>> [a,b,c,d]=marks([12 3 4 5 -4 5])
```

```
ans =   -4    3    4    5    5   12
a =     -4    3    4    5    5   12
b =    4.1667
a =     -4    3    4    5    5   12
b =    4.1667
c =    4.5000
a =     -4    3    4    5    5   12
b =    4.1667
c =    4.5000
d =    5.1153
```

Postoje i funkcije s varijabilnim brojem argumenata, koje će biti opisane nakon što steknemo više znanja o programiranju.
Funkcije sa argumentima koji su znakovne (string) varijable, mogu se pisati bez zagrada. Pogledajmo funkciju `comfun`() koja ima tri ulazna argumenta koje želimo samo prikazati.

Primjer 3.12: *Funkcije s znakovnim (string) varijablama*

Prikaz funkcije `comfun()` s tri ulazna string argumenta, te njezino pozivanje iz

command windowa

```
function comfun(x,y,z)
%COMFUN prikaz funkcije s tri string argumenta
disp(x), disp(y), disp(z)

%poziv funkcije comfun iz command windowa
%Sa zagradama i bez njih rezultat je isti:
>> comfun('ab','ac','ad')
>> comfun ab cd eg
```

```
ab
ac
ad
ab
ac
ad
```

Slične funkcije već smo upoznali, kod `diary()` ili `disp()`:

```
disp('hello'), disp hello
diary ('moja.txt'), diary moja.txt
```

No, što bi se dogodilo kad bismo pozvali neku funkciju koja ne očekuje znakovni argument bez zagrada, na primjer:

```
>> sqrt 2
ans =
            7.0711
```

Naputak: Kolika je ASCII vrijednost brojke 2? Provjerite s `abs('2')` i potom izvedite zaključak.

Nakon poziva funkcije Matlab prvo traži funkciju u stazama definiranima u varijabli `path` koja se formira tijekom instalacije programa. U toj varijabli nalazi se i put do trenutačnog foldera u kojem korisnik radi. Ako postoje m-datoteke s istim imenom, ali u različitim stazama, onda će biti prvo izvedene one koje se ranije nalaze u nizu unutar varijable `path`.
Ukoliko se želi dodati novi folder (direktorij) u put pretraživanja, moguće je to načiniti na ovaj način:

```
>> p=path;
>> path(p,'novi_put');
```

Prva naredba postojeći put pretraživanja pridružuje varijabli p. Druga naredba definira novi put pretraživanja tako da starom putu iz varijable p doda novi put pretraživanja definiran nizom 'novi_put'. Definirani put pretraživanja ostaje aktivan do izlaska iz matlaba. Ukoliko se taj put želi trajno pohraniti u računalo, potrebno je navedene naredbe dodati u startup.m datoteku u folderu gdje je instaliran Matlab.

Primjer 3.13: *Rad sa funkcijama i skriptama, te globalnim varijablama*

Povezivanje rada sa skriptama i funkcijama. Napisat ćemo programski kod za tri skripte (skripta1.m i skripta2.m i poziv.m) i dvije funkcije (funkcija1.m i funkcija2.m). Tih peti datoteka spremit ćemo u jedan zajednički direktorij, tj. datoteku koja će se nalaziti na C:\ukupno. Nakon toga ćemo postaviti putanju u Matlabu prema tom mjestu i postavit ga kao radni direktorij tako da se poziv funkcija i skripti izvodi samo upisivanjem njihovih imena. Pozivom skripte pod nazivom `skripta1` *pokreće se niz naredbi koje će napraviti novi direktorij na C: i koje će upisati putanju u matlab te ga postaviti kao radni direktorij. Skripta pod nazivom* `poziv` *će pozvati funkcije* `funkcija1` *i* `funkcija2` *i u interakciji s korisnikom dati neke rezultate te pokazati rad s globalnim varijablama. Skripta pod nazivom* `skripta2` *ima funkciju da obriše stvoreni direktorij C:\ukupno i sav njen sadržaj.*

%Programski kod za skripta1.m

```
echo on %ispisuje naredbe koje izvrsava
mkdir C: ukupno; %stvaranje direktorija na C:\ukupno
path(path,'C:\ukupno');  %postavljanje putanje u matlab
cd C:\ukupno; %postavlja putanju za radni direktorij u matlabu
echo off
disp ('Ako zelis vidjeti postojece putanje u Matlabu upisi naredbu pathtool')
```

%Programski kod za poziv.m

```
A=input('Upisi vektor s tri broja, npr. [4,1,5] : ')
echo on
[br1,br2,br3]=funkcija1(A(1),A(2),A(3)) %poziv funkcija1
B=input('Upisi broj, npr. 4 : ')
[z1,z2]=funkcija2(B)  %poziv funkcije funkcija2
echo off
disp('Ako zelis obrisati sve kako je bilo prije pokretanja ovog primjera
pokreni kod iz skripta2.m')
```

%Programski kod za funkcija1.m

```
function [iz1,iz2,iz3]=funkcija1(ul1,ul2,ul3)
global recenica broj  %dvije globalne varijable
recenica='Postoje dvije globalne varijable';
broj=ul1;
iz1=ul1+ul2+ul3;
iz2=ul1*ul2*ul3;
iz3=-ul1-ul2-ul3;
disp(recenica)
```

%Programski kod za funkcija2.m

```
function [izlaz1,izlaz2]=funkcija2(ulaz)
global recenica broj
```

```
izlaz1=ulaz*2+broj;
izlaz2=ulaz*6+broj;
disp(recenica);

%Programski kod za skripta2.m
echo on
cd ..  %iz C:\ukupno ide u C: tako da 'ukupno' nije vise radni direktorij i da
%se može obrisati
rmdir ('C:\ukupno','s'); %brise direktorij i sav njegov sadrzaj
rmpath C:/ukupno  %uklanja putanju iz Matlaba
```

Primjer 3.14: *Tijek poziva funkcija i skripti iz prethodnog primjera (*Primjer 3.13*)*

Primjer 3.13 sadrži samo programski kod koji se trenutno može samo proučavati. Svrha ovog primjera je da taj kod pretvorimo u m-datoteke i nekim ih redom pozovemo.

1. *Programski kod za skriptu skripta1.m kopirate u matlabov editor i spremite je pod imenom* `skripta1` *sa ekstenzijom .m (znači skripta1.m). Matlab će tu datoteku spremiti u radni direktorij (npr. C:\Matlab6p5\work). Sada pokrenite tu skriptu upisivanjem u command window matlaba samo njeno ime (*`skripta1`*) i sve naredbe u njoj će se izvršiti. Ta skripta će umjesto vas napraviti novi direktorij na C particiji pod nazivom "ukupno" (C:\ukupno) u koji ćete spremiti sve m-datoteke ovog primjera. Sada kada imate i taj novi direktorij "ukupno" odmah prekopirajte skripta.m u njega. Ta skripta je ujedno promijenila trenutačni radni direktorij Matlaba u C:\ukupno, te dodala tu putanju među ostale matlabove putanje.*
2. *Sada spremite i ostale skripte (poziv.m i skripta2.m) i funkcije (funkcija1.m i funkcija2.m) kao m-datoteke u C:\ukupno tako da se sada u tom direktoriju nalazi pet m-datoteka. Pozovite skriptu poziv.m u command windowu upisivanjem samo imena (*`poziv`*). Slijedi interakcija korisnika i programskog koda, te zaključke o radu skripti, funkcija i glodalnih varijabli donesite sami.*

```
>>skripta1
```

```
echo on %ispisuje naredbe koje izvrsava
mkdir C: ukupno; %stvaranje direktorija na C:\ukupno
path(path,'C:\ukupno');  %postavljanje putanje u matlab
cd C:\ukupno; %postavlja putanju za radni direktorij u matlabu
echo off
Vidjeti postojece putanje u Matlabu upisi naredbu pathtool
```

```
>> poziv
>> [4,1,5]
```

```
Upisi vektor s tri broja, npr. [4,1,5] :
A =    4    1    5
[br1,br2,br3]=funkcija1(A(1),A(2),A(3)) %poziv funkcija1
Postoje dvije globalne varijable
```

```
                                br1 =    10
                                br2 =    20
                                br3 =   -10
                                B=input('Upisi broj, npr. 4 : ')
                                Upisi broj, npr. 4 :
>> 4                            B =     4
                                [z1,z2]=funkcija2(B)  %poziv funkcije funkcija2
                                Postoje dvije globalne varijable
                                z1 =    12
                                z2 =    28
                                echo off
                                Ako zelis obrisati sve kako je bilo prije pokretanja ovog
                                primjer pokreni kod iz skripta2.m
                                cd ..  %iz C:\ukupno ide u C: tako da 'ukupno' nije vise radni
>>skripta2                      %direktorij i da se može obrisati
                                rmdir ('C:\ukupno','s'); %brise direktorij i sav njegov sadrzaj
                                rmpath C:/ukupno  %uklanja putanju iz Matlaba
```

UPRAVLJANJE TIJEKOM PROGRAMA 4

Primjena naredbi na strukturu podataka zove se program, a slijed naredbi algoritam. MATLAB je potpun programski jezik u kojem je, kako smo vidjeli, moguće pisati vlastite programske odsječke. Pojedine naredbe moguće je izvršiti uvjetno ili ponoviti više puta. Pomoću ugrađenih funkcija i programskih paketa moguće je graditi nove programe. Osnovna struktura podataka u Matlab-u je matrica čiji elementi mogu biti kompleksni, realni ili cjelobrojni. Vektor se u Matlab-u predočuje kao matrica od jednog retka s n-stupaca (ili jednog stupca s m-redaka), a skalar kao matrica od jednog retka i jednog stupca. Sve što je definirano za matrice odgovarajuće će se prenijeti i na njezine podskupove. Osim matrice Matlab poznaje i druge tipove podataka (klase i objekte, strukture, ćelije i dr.). Da bi se postupak obradbe podataka mogao uvjetno mijenjati ili iterativno ponavljati mora postojati programska kontrola. Programska kontrola je nuždan uvjet svakog programskog jezika. Matlab nam omogućuje napisati funkciju točno po formuli koju razumijemo i znamo. Time njegova pedagoška strana postaje neusporedivo snažnija. Čak se i funkcije koje je proizvođač načinio, a nalaze se u m-zapisnicima, dadu čitati, proučavati i poboljšavati. Tako Matlab postaje izvrstan alat kako da korisnik nauči programirati. Visoka matematička razina s koje polazi poticajno će djelovati na svakog početnika. Prijelaz s njega na niži programski jezik predstavljat će samo nadogradnju programskog iskustva i poticaj za nova, brža i elegantnija rješenja.

4.1. Uvjetne naredbe

4.1.1. IF-naredba:

Uvjetna naredba `if` (hrvatski *ako*) dolazi u više oblika.
Najjednostavniji oblik je:

```
if (uvjet ili relacija)
    naredba(e);
end
```

dok je nešto složeniji:

```
if ( uvjet ili relacija)
    naredba(-e)
elseif (uvjet ili relacija)
```

```
    naredba(-e)
else
    naredba(-e)
end
```

Uvjet je definiran preko relacija (izraza s relacijskim operatorima), pa ukoliko je istinit (tj. nije jednak nuli), programska kontrola prenosi se na naredbu(e) koja(e) iza uvjeta slijedi(e). `If` naredba završava ključnom riječi `end` iza koje se program nastavlja, ako uvjet nije zadovoljen ili, ako je uvjet bio zadovoljen, nakon što su naredbe `if` bloka obavljene.

Primjer 4.1: *If- petlja*

Pregled rada if – petlje

```
>> x = 2
>> if (x<5)
      x=3;
   end
>> x
```

```
x=     2


x=     3
```

Budući da je uvjet bio ispunjen (2<5 je istinito) ostvarila se naredba x=3, što je posljednja naredba potvrdila ispisom te varijable.

Primijetite da Matlab u ovakvom (interaktivnom) radu ne ispisuje svoj znak '>>' sve dok se ne završi `if` naredba riječju `end`. Ovaj i svi slični primjeri u ovom poglavlju imaju samo pedagoški smisao za upoznavanje programskog upravljanja. Njezina stvarna vrijednost i potreba dolazi do izražaja tek u tvorbi programa, skripta i funkcija u m-datotekama.
Proširenje jednostavne `if` naredbe postiže se s pomoću riječi `else` (hrvatski *inače*) iza koje slijede naredbe koje se obavljaju ako uvjet `if` naredbe nije zadovoljen. Ako je potrebno `inače` dio još proširiti novim `ako`, onda se koristimo spregnutim izrazom `elseif` (hrvatski *inače ako*). Tako se uvjetna kontrola produbljuje do bilo koje dubine.

Primjer 4.2: *If- elseif-else petlja*

Pregled rada if–else-elseif-end petlje

```
>> x=2, y=8

>> if (x >= y)
      x = -x;
   elseif (x > y/x) | (y <= x^3)
      y = y/x; x = y-1;
```

```
x=     2
y=     8
```

```
   else
     y=0; x=5;
   end
>> x
>> y
```

```
x=      3
y=      4
```

Dobro je opet napomenuti da provjera uvjeta ne mora biti samo s obzirom na cjelobrojne vrijednosti ili cjelobrojne varijable. Moguć je, na primjer, čak i uvjet: `if` a ~= b, gdje su a i b matrice. Ovaj uvjet je ispunjen ako matrice nisu iste. To isto vrijedi i za vektore.

4.1.2. SWITCH-naredba:

Struktura `switch` naredbe izabire vrijednost između više ponuđenih izraza.

```
switch {izraz}
        case {slučaj_1},
        naredba, ..., naredba
        case {slučaj2, slučaj3, ...}
        naredba, ..., naredba
                ...
        otherwise,
        naredba, ..., naredba

        end
```

Izraz može biti skalar ili string.

Primjer 4.3: *Switch – case – otherwise petlja sa izrazom kao string*

Pregled rada switch – case - otherwise - end petlje

```
>> x=2.7, units='m',

>>switch units
   case {'feet','ft'}
        y=x*2.54*12;
   case {'inch','in'}
        y=x*2.54;
   case {'metar','m'}
        y=100*x;
   case {'centimetar','cm'}
        y=x;
   case {'millimetar','mm'}
```

```
x=      2.7
units=      m
```

```
        y=x/10;
   otherwise
   disp(['Nepoznata mjerna jedinica:  ' units]);
        y=nan;
   end
>> y
```

```
y=      270
```

Primjer 4.4: *Switch – case – otherwise petlja sa izrazom kao skalar*

Pregled rada switch – case – otherwise - end petlje

```
>> var=3
>> switch var
    case 1
        disp('1');
    case {2,3,4}
        disp('2 ili 3 ili 4');
    case 5
        disp('5');
    otherwise
        disp('izaberi broj od 1÷5');
    end
```

```
var=     3

2 ili 3 ili 4
```

4.2. Petlje

4.2.1. FOR – petlja:

Sintaksa je:

```
for {varijabla} = {vektor vrijednosti brojila}
    naredba(-e)
end
```

Ako je broj iteracija poznat, preporuča se koristiti naredbu `for` (hrvatski *za*) petlju. `For` petlja mijenja vrijednost svoje kontrolne varijable od početnog do konačnog iznosa s odabranim korakom.
U daljnjim primjerima o petljama primijetit ćete da se varijabla iza naredbe petlje (npr. `for`) ponekad označava kao "ii" ili "jj", a ne kao uobičajeno "i" ili "j". To je zato što slova (i,j) Matlabu predstavljaju imaginarne jedinice. Naime, može se koristiti i "i" i "j" i

"ii" i "jj" i bilo koje drugo slovo ili riječ i to će funkcionirati, ali bi kod kompliciranijeg programiranja moglo Matlabu otežati normalan rad.

Primjer 4.5: *For – petlja*

Pregled rada for - end petlje

```
>> for x = 1:3
       x
   end
```

```
x=     1
x=     2
x=     3
```

Vrijednost kontrolne varijable x mijenjala se od 1 do uključno 3, s korakom 1, implicitno sadržanim u sintaksi. Promjena koraka posve je jednostavna:

Primjer 4.6: *For – petlja s promjenom koraka*

Pregled rada for - end petlje s negativnim korakom

```
>> x = 0
>> for k = 5:-0.5:4   %negativan korak (-0.5)
         x = x + k
   end
```

```
x=     0
x =    5
x =    9.5000
x =    13.5000
```

Petlja se može definirati i unutar neke druge petlje. Dubina "zapetljavanja" nije ograničena.

Za konac priče o `for` petljama još samo jedan primjer koji pokazuje kako je iterativna varijabla `for` petlje ustvari vektor (ili matrica):

Primjer 4.7: *Iterativna varijabla for- petlje*

Pregled iterativne varijable for-petlje kao vektora, tj.matrice

```
>> x = [ 3.4 5 -5 6*i ]
>> for j=x
       j
   end
```

```
x=     3.4000   5.0000   -5.0000   0+6.000i
j =    3.4000
j =    5
j =    -5
j =    0 + 6.0000i
```

4.2.2. WHILE – petlja:

Odgovara na pitanje "sve dok je istina – čini":

```
while {uvjet}
   naredbe
end
```

Sve dok je uvjet ispunjen (istinit) u ovoj će se petlji obavljati naredbe redom kako su napisane. Pretpostavlja se da naredbe koje slijede mijenjaju uvjet koji je postavljen. Inače, petlja postaje beskonačna (uz istinit uvjet na početku). Ako uvjet na početku nije zadovoljen naredbe unutar petlje se neće obaviti nijedanput.

Primjer 4.8: *While – petlja*

Pregled rada while - end petlje

```
>> x = 5
>> while x < 5000
        x
        x = x^2;
   end
```

```
x=     5

x=     5
x=     25
x=     625
```

`While` petlja može se prekinuti i prije nego uvjet postane lažan (=0), a to se postiže povezivanjem već poznate `if` naredbe i `break` naredbe (hrvatski *prekid*) za prekid petlje. `Break` prenosi programsku kontrolu na naredbe koje slijede iza `end` naredbe petlje.
Slijedeći primjer samo je izmijenjena verzija prethodnog u kojem je uključena i beskonačna petlja:

Primjer 4.9: *While – petlja kao beskonačna petlja*

Pregled rada while - end petlje kao beskonačne petlje i uvjetovano iskakanje iz nje

```
>> x = 5
>> while 1 > 0
        x
        x = x^2;
   if x > 5000
   break;
   end
   end
```

```
x=     5

x=     5
x=     25
x=     625
```

Uvjet (1>0) je uvijek istinit, pa je petlja beskonačna. Budući da se istinitost uvjeta provjerava (C-programeri to dobro znaju) samo s obzirom na različitost ili jednakost s nulom, isti efekt se postiže ako napišemo samo `while` 1 ... `end` (jer je 1 različito od 0).

4.3. Funkcije, funkcije, funkcije ...

Ovdje su tablično prikazane naredbe i funkcije koje najčešće koristimo u stvaranju novih (korisničkih) funkcija. O svakoj od njih Matlab daje detaljniji opis, najčešće i s primjerom.

4.3.1. Tijek programskog upravljanja

Tablica 4.1: *Opis naredbi tijeka programskog upravljanja*

Naredba	Opis naredbe
`if`	Uvjetno izvršavanje naredbi (ako).
`else`	Uvjetno izvršavanje naredbi (inače).
`elseif`	Uvjetno izvršavanje naredbi (inače, ako).
`end`	Završava doseg FOR, WHILE, SWITCH, TRY i IF naredbi.
`for`	Ponavlja naredbe zadani broj puta
`while`	Ponavlja naredbe neodređeni broj puta.
`break`	Prekida izvođenje WHILE ili FOR petlje.
`continue`	Prosljeđuje upravljanje na iduću iteraciju za FOR ili WHILE petlju.
`switch`	Izabire na temelju izraza između nekoliko slučaja.
`case`	Slučaj za SWITCH naredbu.
`otherwise`	Pretpostavljeni slučaj SWITCH naredbe.
`try`	Početak TRY skupa naredbi.
`catch`	Početak CATCH skupa naredbi.
`return`	Povratak iz pozvane funkcije.
`error`	Ispisuje poruku o pogreški i prekida izvođenje funkcije
`rethrow`	Ponavlja `error` naredbu i prekida izvođenje funkcije

4.3.2. Izračunavanje i izvršavanje

Tablica 4.2: *Opis naredbi za izračunavanje i izvršavanje*

Naredba	Opis naredbe
`eval`	Izvršava string kao da se radi o izrazu ili naredbi.
`evalc`	Isto kao `eval`, osim što se izlaz umjesto na zaslon šalje u polje znakova.

`feval`	Izračunava funkciju određenu držalom funkcije ili funkcijskim imenom
`run`	Izvršava skripte. Omogućuje definiranje staze gdje se skripta nalazi.
`evalin`	Izračunava izraz koji je spremljen u radnom prostoru (workspace)
`builtin`	Izvršava ugrađene (built-in) funkcije koje se pozivaju iz prekrivajuće (overloaded) metode.
`assignin`	Pridružuje vrijednost varijabli u radnom prostoru.

4.3.3. Skripta, funkcije i varijable

Tablica 4.3: *Opis naredbi za skripte, funkcije i varijable*

Naredba	Opis naredbe
`script`	O skriptama i M-datotekama
`function`	Dodaje novu (korisničku) funkciju.
`global`	Definira globalnu varijablu
`persistent`	Definira trajnu varijablu - konstantnu.
`mfilename`	Ime m-datoteke koja se trenutačno izvodi
`lists`	Popis (lista, listina) s članovima odijeljenim zarezom
`exist`	Provjerava jesu li varijable ili funkcije definirane
`isglobal`	Vraća istinu za globalne varijable
`mlock`	Zaštićuje m-datoteku od brisanja
`munlock`	Omogućuje brisanje m-datoteke
`mislocked`	Vraća istinu ako se m-datoteka ne može brisati
`precedence`	Operator prioriteta u Matlabu
`isvarname`	Provjerava valjanost imena varijable
`iskeyword`	Provjerava je li ulaz ključna riječ
`namelengthmax`	Najveća duljina imena

4.3.4. Rad s argumentima

Tablica 4.4: *Opis naredbi za rad s argumentima*

Naredba	Opis naredbe
`global`	Definira globalne varijable
`return`	Povratak iz pozvane funkcije
`nargin`	Broj ulaznih argumenata funkcije
`nargout`	Broj izlaznih argumenata funkcije
`varargin`	Lista promjenljive duljine ulaznih argumenata
`varargout`	Lista promjenljive duljine izlaznih argumenata
`nargchk`	Provjerava broj ulaznih argumenata.

`nargoutchk`	Provjerava broj izlaznih argumenata.
`inputname`	Ulazno ime argumenta, pozvano unutar funkcije

4.3.5. Message display (ispis poruke na ekranu)

Tablica 4.5: *Opis naredbi za ispis poruke na ekranu*

Naredba	Opis naredbe
`error`	Ispis pogreške poruke ili prekidne funkcije
`warning`	Prikaz poruke upozorenja
`lasterr`	Posljednja poruka o pogreški
`lasterror`	Posljednja poruka o pogreški i pridružena informacija
`lastwarn`	Posljednja poruka upozorenja
`disp`	Ispis polja (obično stringa)
`display`	Prekrivena funkcija za prikaz polja
`fprintf`	Ispis formatirane poruke
`sprintf`	Ispis formatirane poruke u string

4.3.6. Interaktivni ulaz

Tablica 4.6: *Opis naredbi za interaktivni ulaz*

Naredba	Opis naredbe
`input`	Ispisuje poruku i čeka korisnički unos
`keyboard`	Poziva tipkovnicu iz m-datoteke
`pause`	Čekanje na korisnikov odziv (pauza)
`uimenu`	Stvara padajuće izborne ponude
`uicontrol`	Stvara korisnički međusklop upravljanja

4.4. Primjeri, primjeri, primjeri...

Svaki od primjera koji slijede pokazuju jednu ili više naredbi ili funkcija koje su prethodno navedene, uz detaljnije objašnjenje.

Primjer 4.10: *Try – catch - end petlja*

Try - catch - end petlja se koristi za izvršavanje neke naredbe pod uvjetom da se vodi briga oko moguće pogreške u izvršavanju te naredbe. Nakon naredbe `try`

postavljaju se izrazi i naredbe koje će se izvršavati i u slučaju da se dogodi bilo kakva greška, programski kod skače na naredbu `catch` *i izvršava njene naredbe. Na ovaj način se mogu hvatati (engl. catch) pogreške koje su nastale tijekom izvršavanja petlje i tada se može njima rukovati na potreban način. U ovom primjeru su na početku postavljene kvadratna matrica i vektor redak koje se ne mogu matrično pomnožiti, pa će u* `try` *petlji odmah nastati greška i izvršavanje koda će se prebaciti na* `catch` *. Proizvedena greška će se spremiti u varijablu greska i uz pomoć naredbe (*`strfind`*) za traženje stringa unutar stringa će se ispisati razlog nastale pogreške.*

```
>> A=[1 2;3 4], B=[1 2]


>> try
     X = A*B
catch
     greska = lasterr

if(strfind(greska, 'Inner matrix dimensions'))
  disp('** Pogresne dimenzije za mnozenje matrica')

end
end

>>A=[1 2;3 4]; B=[1 2]'
>> try
     X = A * B
catch
     greska = lasterr
if(strfind(greska, 'Inner matrix dimensions'))
  disp('** Pogresne dimenzije za mnozenje matrica')
end
end
```

```
A=  1   2
    3   4
B=  1   2

greska =
Error using ==> *
Inner matrix
dimensions must agree.

** Pogresne dimenzije
za mnozenje matrica




B= 1
   2

X=  5
   11
```

Primjer 4.11: *Ugniježđene try – catch - end petlje*

Ugniježđene try - catch - end petlje se mogu koristiti za slučaj ponovnog pokušaja rješavanja nekog izraza u slučaju da se dogodi greška kod izračunavanja izraza iza prve `try` *naredbe. Ako se greška pojavi odmah se izvršavanje prebacuje sa naredbe* `try` *na* `catch`*.. Tamo ponovo postoji petlja try - catch - end koja ponovo pokušava riješiti početni problem množenja matrica tako što će u ovom primjeru vektor redak zamijeniti sa vektorom stupca (transponiranje) što će rezultirati uspjehom množenja matrica. Ako bi npr. vektor B imao tri elementa ne bi ga bilo*

moguće pomnožiti sa matricom A i rezultat ovog programskog koda bi bio: "Neuspjela operacija množenja".

```
>> A=[1 2;3 4]; B=[1 2];
>> try
      X= A*B
catch
  try
    if (strfind(lasterr,'Inner matrix dimensions'))
        B=B';
        X= A*B
    end
  catch
     disp('Neuspjela operacija mnozenja')
  end
end
```

```
X= 5
     11
```

Primjer 4.12: *For - petlje*

For - petlja se izvršava tako što brojilo unutar petlje poprima sve vrijednosti stupaca unutar zadane matrice. Ukoliko se radi o redčanom vektoru brojilo poprima sve vrijednosti tog retka. Obično se koristi operator koji omogućuje jednostavno postavljanje početne i konačne vrijednosti, te koraka. Ugniježđene FOR petlje se u mnogo slučaja koriste za rad s elementima matrice (i-redaka, j-stupaca)

```
>> X=[1 2;3 4]

>> for ii = X     % za svaki stupac matrice X
      disp(ii);   % izvrši tijelo petlje
   end


>> for ii = [1 2 5 7]   % za svaki element vektora
        disp(ii);         % izvrši tijelo petlje
    end


>> a = zeros(1,5) % inicijaliziraj vektor a
>> for i = 1:2:5   % za svaki i od 1 do 5 s korakom 2
  a(i) = i*i -3;  % na i-to mjesto vektora a ubaci vrijednost i*i-3
    end           %  kraj for petlje
>> a
```

```
X=    1   2
      3   4

1
3
2
4

1
2
5
7
a= 0 0 0 0 0



a= -2 0 6 0 22
```

```
>> for i=1:3
      for j=1:3
          a(i,j)=1/(i+j)
      end
   end
%usporedi sa:
>> aa=hilb (3)   % elementi matrice s vrijednoscu  1/(i+j-1)
```

```
a =
0.50 0.33 0.25
0.33 0.25 0.20
0.25 0.20 0.16
aa =
1.00 0.50 0.33
0.50 0.33 0.25
0.33 0.25 0.20
```

Primjer 4.13: *While - petlje*

WHILE petlja se izvršava dok je ispunjen logički uvjet petlje. Naredba `break` *koristi se samo za bezuvjetni izlazak iz FOR ili WHILE petlje. Ako se vektor nađe u petlji koja počinje sa brojem većim od jedan (npr.* `for` *ii=3) Matlab će mu automatski dodati prva dva elementa sa vrijednostima jednakim nuli.*

```
>> a=[-2 5 6 5 22]
>> i = 5                  % inicijaliziraj varijablu i
>> while (i > 0)          % dok je i veći od 0 ponavljaj
      if (a(i) ~= 5)      % ako a(i) nije jednako 5
        a(i) = a(i) - 3; % tada a(i) smanji za tri
      else                % inače
        a(i) = 127;       % u a(i) spremi 127
      end                 % kraj if naredbe
   i = i - 1;             % smanji i za 1
   end                    % kraj while petlje
>> a

>> ii = 7                % inicijalizacija ii
>> while (ii >= 0)
    if (ii <= 4)     % ako je i manji ili jednak od 4
       break         % bezuvjetno prekini petlju
    end              % kraj if naredbe
       ii=ii-1       % u svakoj iteraciji ispisi ii
   end               % kraj while naredbe

>> j=3
>> while j <=8
       k(j)=cos(j*pi);
       j=j+1;
   end
```

```
a=  -2 5 6 5 22
i=  5

a =  -5  127  3  127  19

ii=  7

ii =  6
ii =  5
ii =  4

j=  3
```

```
>> k
    % usporedi sa:
>> kk=cos((3:8)*pi)
```

```
k = 0  0  -1  1  -1  1  -1  1

kk =  -1  1  -1  1  -1  1
```

Primjer 4.14: *Argumenti funkcije*

Funkcija koja provjerava broj ulaznih argumenata, te ih zbraja.

```
function c = zbroji(a, b)
      % ZBROJI - Zbrajanje dva broja
      % Provjera ulaznih argumenata.
  if nargin ~= 2
    error('Zbrajamo dva broja.');
  end
      % Kod programa.
    c = a + b;
%--- kraj funkcije. Spremiti kao zbroji.m datoteku

>> zbroji(5)    % poziv funkcije zbroji()

>> zbroji(5,4)      % poziv funkcije zbroji()
```

```
??? Error using ==> zbroji
Zbrajamo dva broja.
ans =  9
```

Primjer 4.15: *Ispis u string*

Formatirani ispis dobiva se upotrebom `sprintf()` *funkcije. Npr.:* `sprintf`*(%5.3f/n,X) će formatirati ispis vektora X tako da će prikazati decimalni broj od 5 znamenaka od kojih su 3 znamenke iza decimalne točka, dok ostale znamenke otpadaju na decimalnu točku i znamenka ispred decimalne točke. Oznaka "/n" (new line ili novi redak) postavlja ispis formatiranog vektora X tako da svaki njegov element bude ispisan u novom retku.*

```
>> disp(' n      sum(1:n)      n*(n+1)/2 ')
>> disp('-----------------------------------')
>>for n=1:5
disp(sprintf(' %3.0f   %5.0f   %5.0f',...
n,sum(1:n), n*(n+1)/2));
    end

>> x=(1:4)*2*pi

>> sprintf(' %5.3f',x)
```

```
  n      sum(1:n)      n*(n+1)/2
-----------------------------------
  1        1             1
  2        3             3
  3        6             6
  4        10            10
  5        15            15
x =
6.2832 12.5664 18.8496 25.1327
ans =
6.283  12.566  18.850  25.133
```

```
>> x=(1:4)*2*pi;
>> sprintf(' %5.3f\n',x)
```

```
ans =
    6.283
   12.566
   18.850
   25.133
```

Primjer 4.16: *Najmanji cijeli broj*

U prvom dijelu primjera potrebno je naći najmanji pozitivni cijeli broj q za koji vrijedi $2^{-q}=0$. U drugom dijelu primjera potrebno je naći najmanji cijeli broj p tako da je $1+2^{-p}=1$ u aritmetici s pomičnim zarezom

```
>> x=1, q=0

>> while x>0
       x=x/2; q=q+1;
   end
>> q
>> x

>> x=1; p=0; y=1; z=x+y
>> while x~=z
       y=y/2; p=p+1; z=x+y;
   end
>> p
>> 2*y  %najmanji realni broj u Matlabu
>> eps   %najmanji realni broj u Matlabu
```

```
x=  1
q=  0



q=  1075
x=  0

z=  2


p=  53
ans =
   2.220446049250313e-016
ans =
   2.220446049250313e-016
```

Primjer 4.17: *Switch naredba*

Naredbom `switch` izabire se vrijednost između više ponuđenih

```
>> method = 'bilinear';
>> switch method
   case {'linear','bilinear'}
      disp('Method is linear')
   case 'cubic'
      disp('Method is cubic')
   case 'nearest'
      disp('Method is nearest')
   otherwise
```

```
Method is linear
```

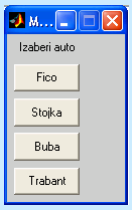
```
    disp('Unknown method.')
end
```

Primjer 4.18: *Unos varijabli*

Funkcijom `input()` *ostvaruje se komunikacija s korisnikom, te unos varijabli u program. Naredbom* `menu` *ostvaruje se grafička komunikacija s korisnikom, te pritiskom miša na jednu od ponuđenih vrijednosti (dugme) Matlab sprema u varijablu redni broj odabrane vrijednosti.*

```
>> n=input('Unesi cijeli broj n = '); %upisi broj npr.8
>> v=(1:2:n).^2

>> k=menu('Izaberi auto','Fico',
'Stojka','Buba','Trabant')  % Izaberi npr. Buba

>> switch k
   case 1
     disp('Odlican odabir, komfor cak za petoricu!')
   case 2
     disp('Da, da, superbrza masina!')
   case 3
     disp('Siguran si, mozes piti!')
   case 4
    disp('Hm, cudna limarija!')
   end
```

```
Unesi cijeli broj n =  8
v =    1    9   25   49
```



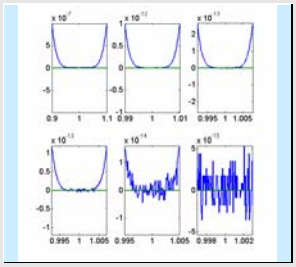

```
k=  3


Siguran si, mozes piti!
```

Primjer 4.19: *Zumiranje slike (figure)*

Nacrtajte (x-1)^6 neposredno uz točku x=1 sa slijedno povećanom skalom. Izračunavanje (x-1)^6 preko x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x +1 dovodi do ozbiljnih promišljanja.

```
>> close all  %briše sve aktivne slike (figures)
>> k=0;
>> for delta = [.1 .01 .008 .007 .005 .003 ]
        x = linspace(1-delta,1+delta,100)';
        y = x.^6 - 6*x.^5 + 15*x.^4 - 20*x.^3 ...
       + 15*x.^2 - 6*x + ones(100,1);
        k=k+1;
        subplot(2,3,k)
        plot(x,y,x,zeros(1,100))
```

```
axis([1-delta 1+delta -max(abs(y)) max(abs(y))])
  end
```



Primjer 4.20: *Nul-točke funkcije*

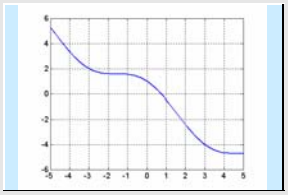
Funkcija: cos (x)-x , 0 < x < π/2

Program cbisekt.m , služi za traženje nultočaka funkcije u traženom intervalu.
Pojašnjenje programa: Prvi `if` *– blok predstavlja mjeru opreza – podjela intervala nema smisla kada funkcija ima iste predznake na oba kraja intervala. While – naredba radi podjelu tako dugo dok nije postignuta željena granica tolerancije. Slijedeća if – naredba provjerava da li je dobiveni c već nultočka, ako jeste, tada će a = c; b = c; biti podijeljeno sa* `while` *i zatim treba prestati. Inače (* `else` *) provjerava, na kojem dijelu intervala funkcija mijenja predznak, sa kojim će se dalje raditi. Kratko objašnjenje poslije* `end` *– naredbe kao i odgovarajući poredak poboljšavaju čitljivost.*
Uputa: Izvedite c = cbisekt(0, pi/2, tol) sa različitim vrijednostima tolerancije.

```
function c = cbisekt (a, b, tol)
% funkcija x0 = cbisekt (a, b, tol)
% izracunava nultocke z funkcije cos(x) - x
% u intervalu a < x < b sa
% pribliznom tolerancijom
if sign (cos(a) - a ) == sign (cos(b) - b )
       error ( 'Nema nultocaka!' );
end      % if sign
while b -a > tol
     c = (b + a)/2;
  if cos (c) - c == 0
    a = c; b = c;
  else
     if sign (cos(a) - a ) ~= sign (cos(c) - c )
          b = c;
      else
          a = c;
      end  % if sign
  end     % if fn
end    % while b - a
```

```
ans =    0.7391
ans=     0.7324
```

```
%-------------------------------------
>> cbisekt(-5,5,0.0001)  %poziv funkcije cbisect()
>> cbisekt(-5,5,0.01)
>> fplot('cos(x)-x ',[-5 5])  %crtanje funkcije
```



Primjer 4.21: *Podjela intervala i traženje nul-točaka za različite funkcije*

Funkcija `cbisekt()` *[Primjer 4.20] obrađuje samo funkciju cos(x) – x. Izvedite program (bisekt.m) za podjelu intervala, koji obrađuje različite funkcije.*
Novost je ovdje pozivanje funkcije

`feval` *(fn, a)*

koja funkciju sa imenom fn izračunava na mjestu a . Upravo time će se postići željena fleksibilnost kod obrade odgovarajućih funkcija – ugrađenih (kao sin, cos, exp, itd.) i samoizgrađenih.

```
function c = bisekt ( fn, a, b, tol )
% funkcija x0 = bisekt ( fn, a, b, tol )
% izracunava nultocke z funkcije, cija imena
% sadrzavaju niz fn u intervalu a < x < b sa
% približnom tolerancijom
if sign(feval(fn, a)) == sign(feval(fn, b))
      error ( 'Nema nultocaka!' );
end      % if sign
while b – a > tol
        c = (b + a)/2;
  if feval ( fn, c ) == 0
        a = c; b = c;
   else
     if sign(feval(fn, a)) ~= sign(feval(fn, c))
        b = c;
     else
        a = c;
     end  % if sign
   end   % if fn
end       % while b – a
%-------------------------------------
>>N1= bisekt('sin',1,5,0.1)
>>N2= bisekt('log',0.5,5,0.1)
>>N3= bisekt('tanh',-0.5,5,0.1)
>>N4= bisekt('tanh',0.5,5,0.1)
```

```
N1=   3.1875
N2=   0.9922
N3=  -0.0703
N4=
Error using ==> bisekt
Nema nultocaka!
```

Primjer 4.22: *Program za određivanje nul-točaka za proizvoljno odabrane samoizgrađene funkcije*

Određivanje nul-točaka funkcije `cos(x)–x` *korištenjem funkcije* `bisekt()` *[Primjer 4.21]. Da bi se ova jednakost mogla izvršiti kao* `cbisekt()` *[Primjer 4.20] , potrebno je još dodati funkciju cosxx.m.*

```
function y=cosxx ( x )
% function y=cosxx ( x ) izračunava cos( x ) – x
y=cos ( x ) – x;

>> c=bisekt('cosxx',0,pi/2,1e-10 );
>> c = cbisekt( 0 , pi/2, 1e-10 );
```

```
c=  0.7391
c=  0.7391
```

GRAFIKA 5

Matlab crta grafiku u posebnom prozoru, različitom od prozora u kojem se unose naredbe u interaktivnom radu. Matlab takav grafički prozor zove *slika* (eng. *figure*). Grafičke funkcije automatski otvaraju novi prozor sa slikom, ako takav prozor ne postoji, inače koriste postojeći. Moguće je naredbom `figure` otvarati nove prozore sa slikom (grafičkim izlazom). Trenutačni prozor (engl. *current figure*) odgovara zadnje otvorenom grafičkom prozoru, pa se sve sljedeće grafičke operacije izvršavaju u njemu. Matlabova grafika je složeno i široko područje koje sadrži nekoliko velikih cjelina:

- **Osnovno crtanje** – obuhvaća prikaz vektorskih i matričnih podataka u 2-D prikazbi.
- **Posebni (specijalni) grafovi** - bave se prikazom podataka u obliku stupaca (bar graphs), histograma i konturnih crteža.
- **3-D vizualizacija** – pokazuje kako se izborom svjetla i kuta gledanja postižu najsloženiji grafički efekti na crtežima stvorenim s pomoću različitih 3-D grafičkih funkcija.
- **Tvorba GUI-a (grafičkog korisničkog sučelja, engl. *Graphical User Interface*)** – pokazuje kako načiniti grafičko sučelje za primijenjene programe pisane u MATLAB-u.
- **Animacije** – bave se stvaranjem pokretne, animirane grafike.

Tu su svakako i posebna, pomoćna područja (zastupljena raznovrsnim Matlab funkcijama) za:

- **Formatiranje grafova** – koja nam pomažu u formatiranju grafova koji najbolje opisuju zadane podatke.
- **Ispis i prijenos grafova** – koja se bave ispisom na izlaznim napravama kao što su različite vrste štampača ili prijenosom grafova u različitim formatima pogodim za druge primjene.

U ovom poglavlju razmatrat ćemo samo osnove prvih triju cjelina, a ostatak će se promotriti u posebnom poglavlju temeljne, *handle* grafike.

Već dobro nam poznata `plot` funkcija može nas uvesti u složenost Matlab grafike. Kao i ostale grafičke funkcije, ona prihvaća ulazne argumente u obliku vektora ili matrica i automatski ih umjerava kako bi se podaci prilagodili osima. Ona će, zavisno o ulaznim argumentima, poprimati različite oblike i različito interpretirati podatke.
Na primjer, ako je **y** vektor, onda će `plot(`**y**`)` nacrtati linearni graf od y elemenata, s obzirom na njihove indekse, redoslijedu u vektoru **y**. Ako su zadana dva vektora kao argumenti, onda će `plot(`**x**`,`**y**`)` nacrtati graf od **y** vektora s obzirom na vektor **x**.

Kada se pozove funkcija `plot` sa samo jednim matričnim argumentom, npr.

`plot(`**Y**`)`

onda Matlab crta jednu liniju za svaki stupac matrice. X-os je označena s indeksima vektora retka, *1:m*, gdje je *m* broj redaka u **Y**.

Općenito, ako se `plot` koristi sa dva argumenta i ako bilo **X** bilo **Y** imaju više od jednog retka ili stupca, onda vrijedi:

- Ako je **Y** matrica, a **x** vektor, onda `plot(`**x**,**Y**`)` slijedno crta retke ili stupce od **Y** u odnosu na vektor **x**, koristeći različite boje ili tipove linija za svaku liniju.
- Ako je **X** matrica a **y** vektor, onda funkcija `plot(`**y**,**X**`)` svaki redak ili stupac od **X** u odnosu na vektor **y**.
- Ako su **X** i **Y** matrice iste veličine, onda funkcija `plot(`**X**,**Y**`)` crta stupce matrice **X** u odnosu na stupce matrice **Y**.

Kada su argumenti funkcije plot kompleksni (tj., imaginarni dio nije jednak nuli), MATLAB ignorira imaginarni dio, osim u slučaju kad je funkciji `plot` zadan samo jedan kompleksni argument. U tom posebnom slučaju crta se realni dio u odnosu na imaginarni, pa je naredba

`plot(`**Z**`)` ekvivalentna s `plot(real(`**Z**`),imag(`**Z**`))`

gdje je *Z* kompleksan vektor ili matrica.

Ako se želi nacrtati više kompleksnih matrica, onda tu nema nikakve prečice – realni i imaginarni dijelovi moraju se eksplicitno navesti.

Višestruki grafovi dobiju se pozivanjem `plot` funkcije s više x-y parova ulaznih argumenata. Matlab automatski izabire unaprijed boju iz unaprijed definiranog popisa, kako bi se nacrtane linije bolje razlikovale.
Vrste linija još posebnije se upravljaju specificiranjem različitih svojstava u više različitih kategorija:

- Širina linije – definira širinu linije u broju točaka kojim se linija crta.
- Rubna boja markera – definira boju ili rub ispunjenih markera (kružića, kvadratića, romba, pentagrama, heksagrama i četiriju vrsta trokutića).
- Boja markera – definira boju kojom je marker ispunjen.
- Veličina markera – definira veličinu markera izraženu u točkama.

Na taj način funkcija plot prihvaća string argumente ('vrstalinije_marker_boja' unutar jednostrukih navodnika) koji određuju vrste linija (npr. crtkane, točkaste i sl.), markerskih oznaka (npr. x, *, o, i sl.) i boja (c, m, y, k, r, g, b, w) za svaku liniju koja se crta. U općem obliku funkcija plot može se pozvati kao:

`plot(`**x**,**y**,'vrstalinije_marker_boja'`)`

Unutar string argumenta značajke ili svojstva linije mogu se navesti u bilo kojem redoslijedu, npr. prvo vrsta, pa oblik markera, pa boja ili drugačije. Pravo razumijevanje svojstava grafičkih objekata postiže se tek nakon svladavanja temeljne, handle, grafike.

5.1. 2D grafovi

Postoji mnogo grafičkih funkcija koje se nadovezuju ili povezuju sa `plot` funkcijom (npr. `axis()`, `grid()`, `legend()`, `line()`, `LineSpec()`, `loglog()`, `plotyy()`, `semilogx()`, `semilogy()`, `subplot()`, `xlabel()`, `xlim()`, `ylabel()`, `ylim()`, `zlabel()`, `zlim()`, `stem()`). Na primjer funkcija `plotyy()` omogućuje stvaranje crteža s dva skupa podataka, tako da se za jedan skup koristi lijeva, a za drugi desna strana Y-osi. Ili funkcija `plotyy()` može se iskoristit za crtanje linearne i logaritamske skale za usporedbu dva skupa podataka, koji imaju različita područja funkcijskih vrijednosti.
Kod crtanja grafova, Matlab automatski odabire granice ili međe koordinatnih osi, kao i crtice na osima ovisne o nacrtanim podacima. Međutim, moguće je naredbama odrediti drugačije granice ili mjesta crtica (podioka, engl. *tick marks*).

Postoji više različitih funkcija, od kojih su neke već korištene, koje služe za označavanje i dodatno objašnjenje nacrtanog grafa:

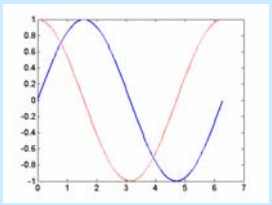
- Funkcija `title()` ispisuje naslov poviše grafa.
- Legenda (nacrtana s pomoću `legend()` funkcije) dodatno označuje svaku različitu liniju u grafu. U legendi je nacrtana i opisana ta linija s kojom je predstavljen zasebni skup podataka, u istoj boji i stilu koji se pojavljuju u grafu.
- Mogu se također dodati x-, y-, i z-oznake koordinatnih osi koristeći `xlabel()`, `ylabel()` i `zlabel()` naredbe.
- Ako je uključen editorski način crtanja, moguće je interaktivno dodavati strelice i linije bilo gdje na grafu u prozoru slike. Slično se s pomoću naredbe `gtext()` može upisati željeni tekst na mjestu koje se odredi klikom na lijevu tipku miša.
- U neinteraktivnom načinu moguće je preko naredbe `text()` i posebne simbole (npr. grčka slova) upisivati u graf s pomoću TeX znakova.
- Moguće je s pomoću `XTick` i `YTick` naredbi specificirati (kao vektor rastućih vrijednosti) mjesta podioka na x i y koordinatnim osima.
- `grid on`/`off` naredba upravlja iscrtavanjem mreže u dvodimenzionalnim i trodimenzionalnim grafovima.

Primjer 5.1: *Crtanje dvodimenzionalnih funkcija*

plot()

plot - crta 2D funkcije u kartezijskim koordinatama

```
>> x=0:pi/100:2*pi;
>> y1=sin(x);
>> y2=cos(x);

%crtanje plave pune linije i crvene tockaste
>> plot(x,y1,'b-',x,y2,'r:');
```



Matlab može crtati grafove u polarnim koordinatama koristeći funkciju:

`polar`(**th**,**r**)

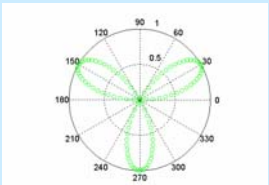
gdje je **th** vektor kutova (u radijanima), a **r** je vektor pripadajućih vrijednosti polumjera. Crtanje u 3D-sfernim koordinatama postiže se prvo pretvorbom sfernih (th,fi,r) u kartezijske (x,y,z) koordinate koristeći funkciju `sph2cart`, a potom prikazom koristeći `plot3` naredbu.

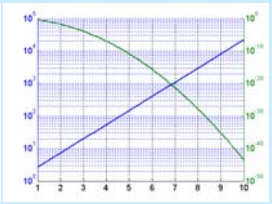
Primjer 5.2: *Crtanje dvodimenzionalnih funkcija u kartezijskim, polarnim i logaritamskim koordinatama.*

polar(), plotyy(), loglog(), semilogy()

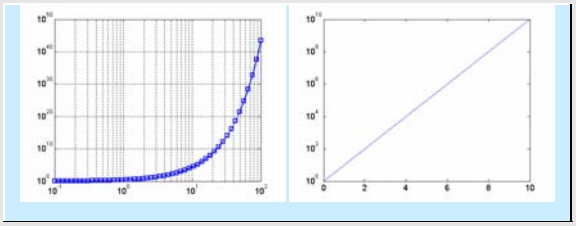
`polar()` - crta 2D funkcije u polarnim koordinatama
`plotyy()` - crta 2D funkcije na grafu sa y-osi na obje strane
`loglog` ()- crta 2D funkcije u logaritamskim koordinatama
`semilogy()` - crta 2D funkcije u logaritamskim koordinatama na y-osi i kartezijskim na x-osi (`semilogx()` - obrnuto od `semilogy()`).

```
%slika na mjestu (1,1)
>> theta=0:pi/100:2*pi;
>> rho=sin(theta*3);
>> polar(theta,rho,'go');
%slika na mjestu (1,2)
>> x=1:0.1:10;
>> y1=exp(x);
>> y2=exp(-x.^2);
>> plotyy(x,y1,
x,y2,'semilogy');
>> grid on
%slika na mjestu (2,1)
>> x = logspace(-1,2);
```





```
>> loglog(x,exp(x),'-s')
>> grid on
%slika na mjestu (2,2)
>> x = 0:.1:10;
>> semilogy(x,10.^x);
```



Kod stvaranja grafa, Matlab automatski određuje veličinu osi s obzirom na raspon podataka koji se crtaju i raspoloživog prostora za sliku. Odnos širine i dužine (engl. *aspect ratio*) generiranog grafa ostaje sačuvan i onda kad korisnik razvlači ili suzuje prozor sa slikom.
Naredbom `axis` moguće je na grafu mijenjati maksimalne i minimalne vrijednosti na pojedinačnim osima i na taj način povećavati ili smanjivati (engl. *zoom*) grafa na slici:

```
axis([xmin xmax ymin ymax zmin zmax])
```

Naredba `axis()` ima više argumenata, od kojih u **Pogreška! Izvor reference nije pronađen.** izdvajamo najčešće korištene:

Tablica 5.1: *Opis funkcija za upravljanje koordinatnim osima*

Funkcija	Opis funkcije
`axis auto`	Vraća upravljanje osima u automatski način rada.
`axis square`	Čini svaku os da bude iste dužine.
`axis equal`	Mijenja umjeravanje grafa tako da su podioke na svakoj osi jednake duljine To čini da npr. površina kugle izgleda doista kao kugla, a ne kao elipsoid.
`axis vis3d`	Zamrzava odnos stranica (aspect ratio) u grafu kako bi se omogućila rotacija u 3-D grafici.
`axis image`	Čini odnos stranica u grafu istim kao što je prikazana slika (image).
`axis normal`	Vraća trenutačnu veličinu osi na puni oblik i uklanja sva ograničenja na umjeravanje.
`semilogx()`	Graf s logaritamskom skalom za x-os, a linearnom za y-os.
`semilogy()`	Graf s logaritamskom skalom za y-os, a linearnom za x-os.
`loglog()`	Graf s logaritamskim skalama za obje osi.

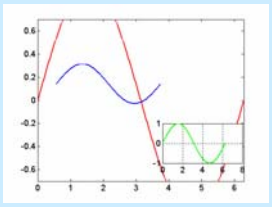
Primjer 5.3: *Kontrla osi (eng. axes) i označavanje grafova*

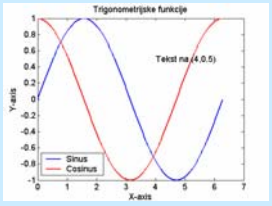
axis(), axes(), legend(), xlabel(), ylabel(),title(), text(), subplot()

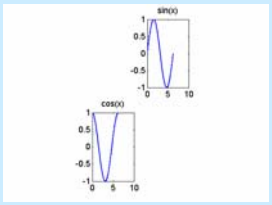
`axis()` - *postavlja osi i kontrolira njen rang vrijednosti*

`axes()` -postavlja osi oko grafičkog objekta
`legend()` -legenda koja služi za prepoznavanje pojedinih funkcija na grafu
`xlabel()` -ispisuje tekst uz os x
`ylabel()` -ispisuje tekst uz os y
`title()` -postavlja naslov na vrh grafa
`text()` -upisuje tekst na zadanom mjestu na grafu
`subplot()` -prikazuje više grafova na jednoj slici (figure)

```
% prva slika
>> x=0:pi/100:2*pi; y=sin(x);
>> plot(x,y,'r');
>> axis([0 2*pi -0.7 0.7])
>> axes('Position',[0.6 0.2 0.3 0.2])
>> plot(x,y,'g'), box on, grid on
>> axes('Position',[0.2 0.5 0.5 0.2])
>> plot(x,y,'b')
>> axis off

% druga slika
>> x=0:pi/100:2*pi;
>> y1=sin(x); y2=cos(x);
>> plot(x,y1,'b',x,y2,'r');
>> legend('Sinus','Cosinus',0);
>> xlabel('X-axis'); ylabel('Y-axis');
>> title('Trigonometrijske funkcije');
>> text(4, 0.5,'Tekst na (4,0.5)');

% treća slika
>> x=0:pi/100:2*pi; y=sin(x);
>> subplot(2,4,3), plot(x,y), title('sin(x)')
>> subplot(2,4,6), z=cos(x);
>> plot(x,z), title('cos(x)')
```







5.2. Specijalizirani 2-D grafovi

MATLAB podržava raznolike tipove grafova kojima se djelotvorno prikazuje informacija spremljena u numeričkim podacima. Tip odabranog grafa ovisit će o prirodi podataka. Na primjer, statističke podatke neke raspodjele prikazat ćemo uz pomoć `bar()`, `pie()` ili `hist()` naredbi, a topološke podatke neke planine naredbom `contour()`.
Stupčasti (bar) ili površinski (area) grafovi prikazuju vektor ili matricu podataka. Stupčasti grafovi su pogodni za prikaz diskretnih podataka, a površinski za kontinuirane

podatke. Oni prikazuju stupce od (m x n) matrice kao m skupina od n vertikalnih ili horizontalnih (u tom slučaju naredba ima 'h' u imenu) 2D ili 3D stupaca, gdje je svaki element (stupac) u matrici prikazan stupčastim prikazom. Visina stupca grafa (ili dužina histograma u polarnim koordinatama – funkcija `rose`) odgovara broju vrijednosti koje padaju u prikazani opseg.
Funkcija `bar3` crta svaki element kao posebni 3-D blok, s elementima svakog stupca raspodijeljenima duž y-osi. Budući da postoji situacija u kojoj veći stupci mogu zakloniti manje, problem se može riješiti korištenjem svojstva 'grouped'.

Tablica 5.2: *Opis funkcija specijaliziranih dvodimenzionalnih grafova*

Funkcija	Opis funkcije
`bar()`	Stupčasti (bar) graf.
`barh()`	Horizontal stupčasti graf.
`bar3()`	3-D bar graf.
`bar3h()`	Horizontalni 3-D stupčasti graf.
`fill()`	Ispunjeni 2-D poligon.
`errorbar()`	Crtež error stupaca.
`hist()`	Histogram.
`pareto()`	Pareto crtež.
`plotmatrix()`	Raspršena (scatter) matrica.
`ribbon()`	Crta 2-D linije kao vrpce u 3-D prikazu.
`stem()`	Discretni niz ili "stem" crtež.
`stairs()`	Stepeničasti crtež.

Histogram broji elemente unutar nekog opsega i prikazuje svaki opseg kao pravokutni stupac. Funkcija `hist()` pokazuje raspodjelu elemenata u jednom vektoru (ili svakom stupcu matrice) kao histogram s jednako udaljenim snopovima (bins, do 10 binova za svaki vektor) između minimalne i maksimalne vrijednosti vektora ili matričnog stupca.

Funkcija `stem()` crta diskretni graf kao linije (stems) koje su zaključene markerom za svaku vrijednost podatka. Function `stem3()` prikazuje 3-D stem crtež proširujući ga iz xy-ravnine. Funkcija s`tairs()` crta podatke kao vodeće rubove (grane) nad konstantnim intervalima.

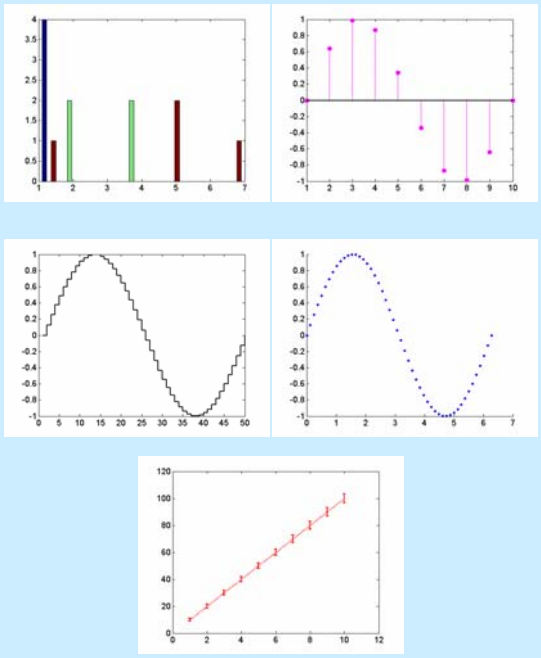
Primjer 5.4: *Specijalizirani grafovi*

hist(), stem(), stairs(), scatter(), errorbar()

`hist()` - *crta histogram. Zadanu matricu podataka crta na taj način da svaki podatak iz istog stupca matrice prikazuje u istoj boji i to tako da na x-os grafa postavlja brojčanu vrijednost podatka, a na y-os postavlja broj jedinki istih brojčanih vrijednosti u pojedinom stupcu matrice.*

`stem ()`- crta podatke kao točke koje su spojene linijom s horizontalnom nul-linijom
`stairs()` -crta stepeničastu funkciju od zadanih točaka
`scatter()` - crta točku na mjestu zadanom s x,y koordinatom
`errorbar()` - crta odstupanje od neke vrijednosti

```
%slika na mjestu (1,1)
y=[1 2 7;1 2 5;1 4 5;1 4 1];
hist(y);
%slika na mjestu (1,2)
y = linspace(0,2*pi,10);
z=sin(y);
stem(z,'fill','m:')
%slika na mjestu (2,1)
y = linspace(0,2*pi,50);
z=sin(y);
stairs(z,'k')
%slika na mjestu (2,2)
y = linspace(0,2*pi,50);
z=sin(y);
d=5;  %promjer kruzica
scatter(y,z,d,'filled')
% zadnja slika
>> x = 1:10;
>> y=10:10:100;
>> e=sqrt(x);
>> errorbar(x,y,e,'r:')
```



Pogled (engl. *view*) je određena orijentacija koju odabiremo za prikazivanje grafa ili grafičke scene. Motrenje se odnosi na proces prikaza grafičke scene iz različitih smjerova, uvećavanje ili smanjivanje prikazanog objekta (engl. *zoom in*, *zoom out*), promjena perspektive i odnosa duljina osi, prelijetanje (s pomoću kamere) i slično.
Matlab motrenje sastoji se od dva temeljna područja:

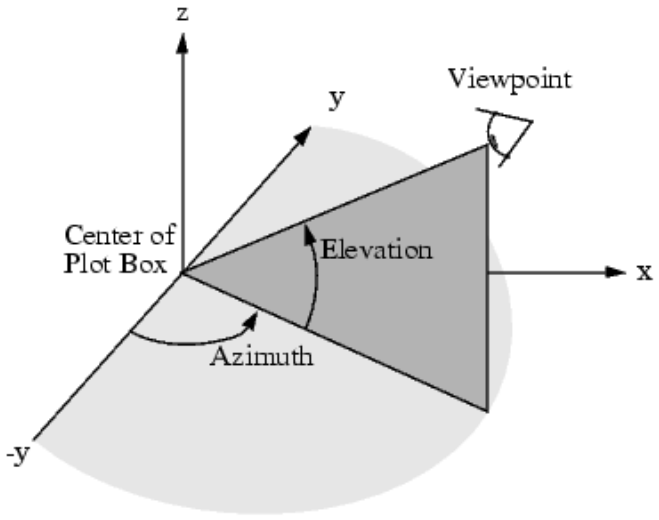
- Pozicionirane točke gledišta (engl. *viewpoint*) za orijentaciju scene.
- Postavljanje odnosa duljine osi (engl. *aspect ratio*) i relativno umjeravanje osi da bi se kontrolirao oblik objekta koji se prikazuje.

Matlab omogućuje upravljanje orijentacije grafike koja se prikazuje u zadanim osima. Može se specificirati točka gledišta, objekt koji se promatra, orijentacija i proširenje pogleda u prozoru slike.

Upravljanje točkom gledišta postiže se s jednom od naredbi `view()`, `viewmtx()`, `rotate3d()`.

Tablica 5.3: *Opis funkcija za upravljanje točkom gledišta*

Funkcija	Opis funkcije
`view()`	3-D zadavanje točke gledišta.
`viewmtx()`	Transformacijska matrica pogleda.
`rotate3d()`	Interaktivna rotacija pogleda u 3-D crtežu.



Slika 5.1: *Određivanje točke gledišta preko azimuta i elevacije*

Funkcija `view()` određuje točku gledišta definiranjem azimuta i elevacije s obzirom na ishodište koordinatnog sustava. Azimut je polarni kut u x-y ravnini, s pozitivnim kutom koji ukazuje na pomak točke gledišta suprotno smjeru kazaljke na satu. Elevacija je kut iznad (pozitivan kut) ili ispod (negativan kut) x-y ravnine. Dijagram ispod pokazuje koordinatni sustav s ugrađenim kutovima za definiranje točke gledišta. Strelice označuju pozitivan smjer.

Matlab automatski određuju točku gledišta s obzirom na 2-D ili 3-D crtež.

Za 2-D crtež, pretpostavljena vrijednost za azimut je 0°, a za elevaciju je 90°.

Za 3-D crtež, pretpostavljena vrijednost za azimut je 37.5°, a za elevaciju je 30°.

Funkcija

```
>> view([180 0])
```

postavlja točku gledišta tako da se gleda iz negativnog y-smjera uz z=0 elevaciju. Moguće je pomaknuti točku gledišta ispod točke ishodišta koristeći negativnu elevaciju, na primjer pogled:

```
>> view([-37.5 -30]).
```

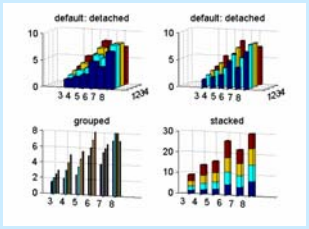
Zadavanje točke gledišta s pomoću azimuta i elevacije je koncepcijski jednostavno, ali ima ograničenja. Ne može se naime zadati stvarna točka gledišta, nego samo smjer, a z-os uvijek gleda prema gore. Ono ne dozvoljava rotacije ili translacije, kao ni povećavanje ili smanjivanje scene. Takve mogućnosti uvedene su preko naredbi koje rade s kamerom.

Primjer 5.5: *Trodimenzionalni stupčasti grafikon*

bar3()

`bar3()` - *crta 3D stupčasti grafikon, gdje je svaka vrijednost zadane matrice jedan stupac.* `bar3h()` *je ista funkcija kao* `bar3()` *samo što stupce crta u horizontalnoj ravnini.*

```
>> X=3:8;
>> Y=[1.5   2    2.5   3
      2     3    4     5
      2.5   3.5  4.5   5.5
      5     6    7     8
      4     5.5  6     6.5
      7     8    8     7];
>> subplot(2,2,1); bar3(X,Y);
>> view([-70 8]); title('default: detached')
>> subplot(2,2,2); bar3(X,Y,0.4);
>> view([-70 8]); title('default: detached');
>> subplot(2,2,3); bar3(X,Y,0.4,'grouped');
>> view([-70 8]); title('grouped')
>> subplot(2,2,4); bar3(X,Y,0.4,'stacked');
>> view([-70 8]); title('stacked');
```



Grafovi površine najčešće se dobivaju funkcijama `area()`, `pie()` i `pie3()`.

Tablica 5.4: *Opis funkcija za prikazivanje grafova površina*

Funkcija	Opis funkcije
`area()`	Crtež s ispunjenom površinom.
`pie()`	Crtež u obliku torte, pite (engl. *pie*).
`pie3()`	3-D crtež u obliku torte.

Funkcija `area()` crta vrijednosti svakog stupca matrice kao posebne krivulje i bojom ispunja površinu između krivulje i x-osi. Visina površine u grafu je zbroj elemenata u svakom retku. Svaka iduća krivulja koristi prethodnu krivulju kao svoju bazu.

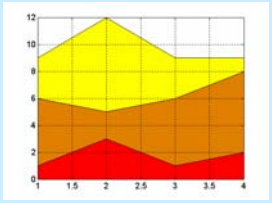
Pie crtež prikazuje (suprotno smjeru kazaljke na satu) postotak koji svaki element ili matrica doprinosi u zbroju svih elemenata. Kad je zbroj manji od 1, funkcije `pie()` i `pie3()` ne normaliziraju elemente i crtaju djelomičnu pitu.
Moguće je izdvojiti krišku iz pite, koristeći `explode()` ulazni argument. Ovaj argument je vektor od ništičnih ili neništičnih vrijednosti. Neništične vrijednosti crtaju izdvojenu krišku iz crteža pite.

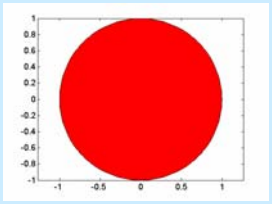
Primjer 5.6: *Bojom ispunjeni grafovi*

area(), fill(), pie(), pie3()

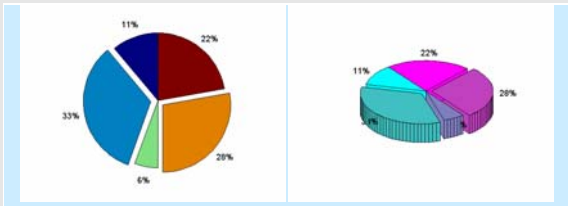
`area()` - *crta 2D područja na grafu jedan iznad drugoga u svrhu prikazivanja relativnog doprinosa pojedinog elementa retka u ukupnoj visini krivulje.*
`fill()` - *ispunjava bojom 2D poligone*
`pie()` - *crta grafikon u obliku pite (eng. pie) čije komade možemo isjeći i odvojiti. Brojčane vrijednosti iz vektora retka koje želimo prikazati funkcija* `pie` *zbraja u jedan broj koji tada predstavlja cijelu pitu (100%) i svaki element prikazuje kao njen udio (komad pite) u toj cjelini od 100%.*
`pie3()` - *3D prikaz funkcije* `pie`

```
%slika na mjestu (1,1)
>> Y = [  1  5  3;
          3  2  7;
          1  5  3;
          2  6  1  ];
>> area(Y)
>> grid on
>> colormap autumn
>> set(gca,'Layer','top')
%slika na mjestu (2,1)
>> x=0:pi/100:2*pi;
>> y=sin(x);
>> z=cos(x);
>> fill(y,z,'r')
>> axis equal
%slike u trecem retku
>> x = [1 3 0.5 2.5 2];
>> izvucen = [0 1 0 1 0];
>> pie(x,izvucen)
```





```
>> colormap jet
>> figure
>> pie3(x,izvucen)
>> colormap cool
```



Posebne funkcije za brzo i jednostavno crtanje grafova su `ezplot()` i `fplot()`:

Tablica 5.5: *Opis funkcija za brzo i jednostavno crtanje grafova*

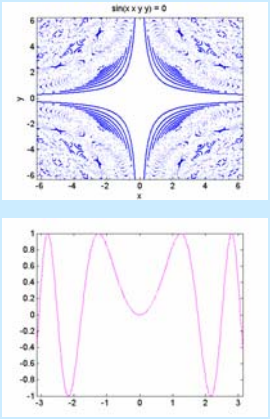
Funkcija	Opis funkcije
`ezplot()`	Jednostavno crtanje funkcijskih grafova.
`fplot()`	Crtanje funkcija

Primjer 5.7: *Jednostavno crtanje funkcija*

ezplot(), fplot()

`ezplot()` *- jednostavno crtanje različitih funkcija sa jednom ili dvije varijable. Crta implicitno zadane funkcije f(x,y)=0 npr.* $x^2-y^5=0$ *ili* $\sin(x^2 \cdot y^2)=0$. *Pritom automatski postavlja naziv funkcije na graf, te daje nazive osima x,y.*
`fplot()` *-crta funkcije unutar zadanih granica bez automatskog ispisivanja naziva osi te naslova na grafu. Može prikazati funkcije u ovisnosti o jednoj varijabli.*

```
% prva slika, f(x,y)=sin(x^2*y^2)
>> ezplot('sin(x*x*y*y)')


% druga slika, f(x)=sin(x^2)
>> fplot('sin(x*x)', [-pi pi],'m:')
```



5.3. 3D grafovi

Za prikaz funkcija dviju varijabli ili vizualizacije velikih matrica koristimo temeljne 3-D grafičke funkcije iz razreda `mesh()` i `surf()` funkcija. Korisno je prvo upoznati način na koji se podaci spremaju i obrađuju. Neka je na primjer, zadana funkcija

$$z = f(x, y) = x \cdot e^{\left(-x^2 - y^2\right)}$$

nad površinom razapetom vektorima **x**=[-1:0.5:1] i **y**=[-1:0.5:1]. Matrica **xx** može se dobiti kao:

```
>> xx=[ 1 1 1 1 1]' * [-1 -0.5 0 0.5 1]
xx =
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
```

ili xx=`ones(length`(x),1)*x. Na sličan način za **yy** vrijedi:

```
>>  yy=y' * ones(1,length(y))
yy =
  -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
  -0.5000   -0.5000   -0.5000   -0.5000   -0.5000
         0         0         0         0         0
    0.5000    0.5000    0.5000    0.5000    0.5000
    1.0000    1.0000    1.0000    1.0000    1.0000
```

To je identično djelovanju funkcije `meshgrid()`, kojom se površina pripravlja za 3-D prikaz.

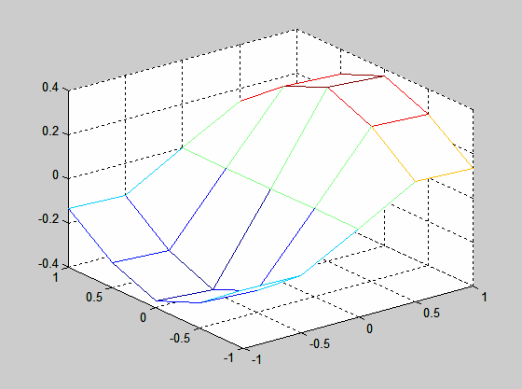
```
>> [xx,yy]=meshgrid(x,y)
xx =
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
  -1.0000   -0.5000         0    0.5000    1.0000
yy =
  -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
  -0.5000   -0.5000   -0.5000   -0.5000   -0.5000
         0         0         0         0         0
    0.5000    0.5000    0.5000    0.5000    0.5000
    1.0000    1.0000    1.0000    1.0000    1.0000
```

Nakon toga se izračunava **z**, točku po točku, prema zadanoj formuli:

```
>> z=xx .* exp(-xx .^ 2- yy.^2)
z =
  -0.1353   -0.1433         0    0.1433    0.1353
  -0.2865   -0.3033         0    0.3033    0.2865
  -0.3679   -0.3894         0    0.3894    0.3679
  -0.2865   -0.3033         0    0.3033    0.2865
  -0.1353   -0.1433         0    0.1433    0.1353
```

Funkcija `mesh()` nacrtat će izračunatu površinu:

```
>> mesh(x,y,z)
```



Kako je vidljivo na slici, površina nije glatka. Povećanje glatkoće površine dobilo bi se povećanjem broja točaka vektora x i y (tj. smanjenjem inkrementa, u ovom primjeru 0.5 i trebalo smanjiti na primjer na 0.01). Upotrebom funkcije `surf`(x,y,z) nacrtala bi se obojena površina u kojoj je boja proporcionalni visini (ili dubini) površine. Hladne boje odgovaraju nižim, a tople višim vrijednostima. O mapi korištenih boja ovisit će o njihov početak i kraj.

S funkcijom `pcolor()` dobila bi se gruba obojena površina slična šahovskoj ploči, a finije konture s pomoću `contour()` funkcija. Tako ulazimo u svijet raznovrsnih 3-D, 2-D i 2 ½ -D funkcija od kojih su još 3-D elementarni `plot3()` i `fill3()`:

Tablica 5.6: *Elementarne trodimenzionalne funkcije*

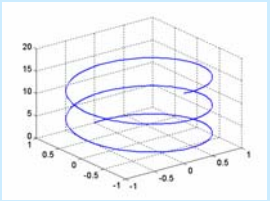
Funkcija	Opis funkcije
`plot3()`	Crta linije i točke u 3-D prostoru
`fill3()`	3-D površina prekrivena poligonima

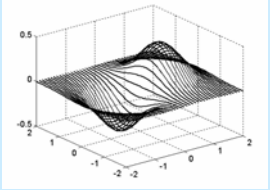
Primjer 5.8: *Trodimenzionalni grafički linijski prikaz.*

plot3()

`plot3()` -crta jednu ili više linija u trodimenzionalnom prostoru spajajući točke zadane koordinatama x,y,z

```
% prva slika
>> t = 0:pi/50:5*pi;       %vektor
>>plot3(sin(t),cos(t),t);
>> grid on


% druga slika
>> [X,Y] = meshgrid([-2:.1:2]);
>> Z = X.*exp(-X.^2-Y.^2);       %matrica
>> plot3(X,Y,Z,'-k')
```





Opisana `mesh()` funkcija ima nekoliko zanimljivih i korisnih podfunkcija koje je vrijedno proučiti:

Primjer 5.9: *Trodimenzionalni grafički prikaz mreže matematičke funkcije koja ovisi o dvije varijable.*

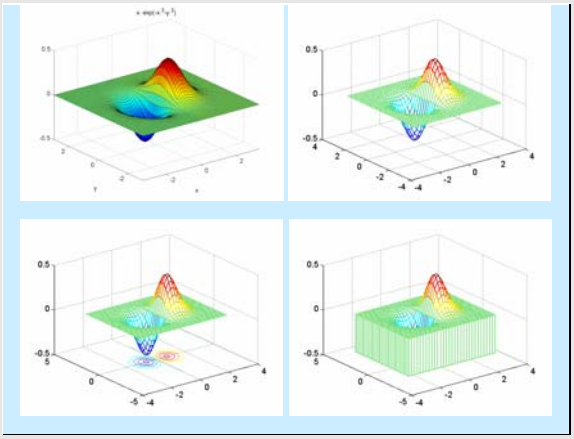
ezsurf(), meshgrid(), mesh(), meshc(), meshz()

`ezsurf()` - prepoznaje nepoznanice u jednadžbi (z=f(y,x)), te grafički prikazuje njihovu zavisnost u 3D prostoru. Automatski postavlja koordinate i ime funkcije.
`meshgrid()` - konstruira mrežu točaka u prostoru
`mesh()` - Trodimenzionalni prikaz mreže funkcije
`meshc()` - 3D prikaz mreže funkcije zajedno sa projekcijom njene konture na x-y ravninu
`meshz()` - 3D prikaz mreže funkcije zajedno s zavjesom po njenim rubovima

Prikaz funkcije $z = f(x,y) = x \cdot e^{\left(-x^2-y^2\right)}$, *tj. njene površine u 3D prostoru*

```
% slika na mjestu (1,1)
>> ezsurf('x.*exp(-x.^2-
y.^2)')
```

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
% slika na mjestu (1,2)
>> mesh(X,Y,Z);
% slika na mjestu (2,1)
>> meshc(X,Y,Z);
% slika na mjestu (2,2)
>> meshz(X,Y,Z);
```



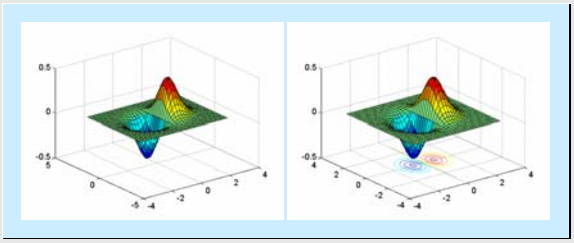
Primjer 5.10: *Trodimenzionalni grafički prikaz površine matematičke funkcije koja ovisi o dvije varijable.*

surf(), surfc()

`meshgrid()` - *konstruira mrežu točaka u prostoru*
`surf()` - *Trodimenzionalni prikaz površine (eng. surface) funkcije*
`surfc()` - *3D prikaz površine funkcije zajedno sa projekcijom njene konture na x-y ravninu*

Prikaz funkcije $z = f(x, y) = x \cdot e^{\left(-x^2 - y^2\right)}$, *tj. njene površine u 3D prostoru*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> surf(X,Y,Z);
>> surfc(X,Y,Z);
```

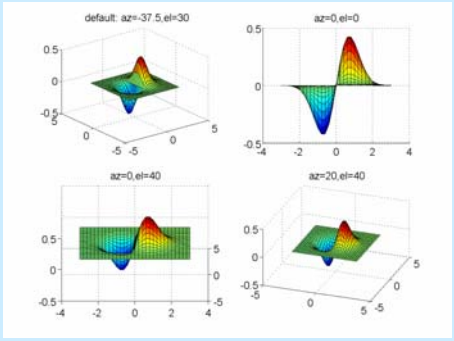


Primjer 5.11: *Crtanje i kontrola višestrukih osi na jednoj slici (figure), te namještanje kuta gledanja trodimenzionalnih grafova.*

subplot(), view()

`subplot()` - *crta mnogostruke osi (axes) na jednoj slici (figure). Osi su na slici raspoređeni kao matrica (retci i stupci).*
`view()` - *prilagođava kut gledanja na 3D grafove. Mogu se koristiti dva kuta kojim se namještava pogled. To su azimuth (horizontalna rotacija, tj. rotacija oko z-osi u smjeru kazaljke na satu za njene pozitivne vrijednosti) i elevation (vertikalna rotacija pri kojoj pozitivna vrijednost označava pogled iznad objekta i obratno).*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> subplot(2,2,1);surf(X,Y,Z);
>> title('default: az=-37.5,el=30')
>> subplot(2,2,2);surf(X,Y,Z);
>> title('az=0,el=0')
>> az=0;el=0;
>> view([az,el]);
>> subplot(2,2,3);surf(X,Y,Z);
>> title('az=0,el=40')
>> az=0;el=40;
>> view([az,el]);
>> subplot(2,2,4);surf(X,Y,Z);
>> title('az=20,el=40')
>> az=20;el=40;
>> view([az,el]);
```



5.4. Konturni i 2 ½-D grafovi

Među raznovrsnim 2- i 2 ½- D grafovima ističu se: `contour()`, `contourf()`, `contour3()`, `clabel()`, `pcolor()`, `quiver()` i `voronoi()`.

Tablica 5.7: *Raznovrsni 2- i 2 ½- D grafovi*

Funkcija	Opis funkcije
`contour()`	Konturni crtež.
`contourf()`	Ispunjeni konturni crtež.
`contour3()`	3-D konturni crtež.
`clabel()`	Konturni crtež s visinskim oznakama.
`pcolor()`	Crtež sa pseudobojom (šahovska ploča).
`quiver()`	Quiver crtež.

`voronoi()`	Voronoi dijagram.

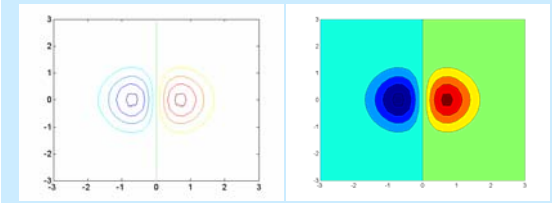
Primjer 5.12: *Dvodimenzionalni konturni grafički prikaz trodimenzionalne matematičke funkcije.*

contour(),contourf()

`meshgrid()` - *konstruira mrežu točaka u prostoru*
`contour()` - *2D prikaz konture (eng. contour) trodimenzionalne funkcije*
`contourf()` - *2D obojani(eng. filled) konturni prikaz trodimenzionalne funkcije*

Prikaz funkcije $z = f(x, y) = x \cdot e^{\left(-x^2 - y^2\right)}$, *tj. njene konture u 2D prostoru*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> contour(X,Y,Z);
>> contourf(X,Y,Z);
```



Matlabove funkcije: `compass()`, `feather()` i `quiver()` prikazuju podatke koji se sastoje od usmjerenih vektora i vektora brzine. One su posebno korisne za crtanje elemenata sa strelicama iz kompleksnog područja.

- `compass()` crta strelice koje izlaze iz ishodišta polarnog crteža.
- `feather()` prikazuje vektore koji izlaze iz jednako udaljenih točaka uzduž horizontalne osi..
- `quiver()` i `quiver3()` crtaju strelice koje izlaze iz različitih (x,y), odnosno (x,y,z) lokacija.

Ako je ulazni argument matrica kompleksnih vrijednosti Z, onda funkcija `feather` interpretira realne dijelove od Z kao x komponente vektora, a imaginarne dijelove kao y komponente vektora.

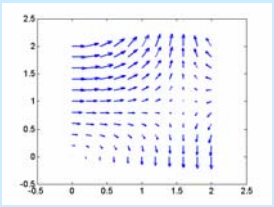
Obje funkcije, `compass()` i `feather()` crtaju vektore koristeći iznos (modul) kompleksnog broja kao dužinu vektora. Međutim, `quiver()` funkcija umjerava sve vektore tako da se ne proširuju izvan ishodišta najbližeg susjednog vektora. Quiver crtež je koristan kad se prikazuje s još nekim usporednim crtežom.

Primjer 5.13: *Prikaz vektorskih veličina pomoću strelica*

quiver()

Prikladan crtež koji pokazuje strelice s veličinom i smjerom koje odgovaraju vrijednostima nekog vektora, npr. brzine vjetra ili tijeka tekućine.

```
>> [x,y] = meshgrid ( 0:0.2:2, 0:0.2:2 );

%nagib funkcije u smjeru osi x
>> u = cos(x) .* sin(y);

%nagib funkcije u smjeru osi y
>> v = sin(x) .* y - log(x+1);
>> quiver ( x, y, u, v )
```



Funkcije kontura stvaraju, prikazuju i označuju izolinije određene s jednom ili s više matrica. `contour()` i `contour3()` prikazuju 2- , odnosno 3-D konture. One zahtijevaju jedan ulazni argument – matricu koja se interpretira kao visine s obzirom na ravninu.

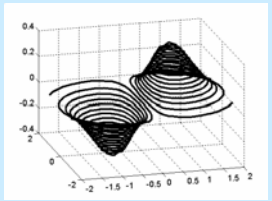
Primjer 5.14: *Trodimenzionalni grafički linijski prikaz konture, te diskretni prikaz vrijednosti točaka funkcije.*

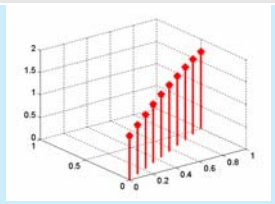
contour3(), stem3()

`contour3()` - *3D konturni prikaz površine funkcije* $z = f(x, y) = x \cdot e^{\left(-x^2 - y^2\right)}$

`stem3()` - *diskretni prikaz trodimenzionalnih vrijednosti koje izlaze iz x,y ravnine i produžuju linijski do same vrijednosti na z-osi prikazane oznakom (kružić, zvjezdica, itd.).*

```
% prva slika
>> [X,Y] = meshgrid([-2:.25:2]);
>> Z = X.*exp(-X.^2-Y.^2);
>> contour3(X,Y,Z,30,'-k')
>> view(-15,25)

% druga slika
>> X = linspace(0,1,10);
>> Y = X./2;
>> Z = sin(X) + cos(Y);
>> stem3(X,Y,Z,'*-r');
```

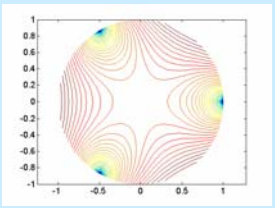




Primjer 5.15: *Prebacivanje kontura iz polarnih u kartezijske koordinate*

Moguće je podatke kontura definirati u polarnom koordinatom sustavu, postaviti mrežu u polarnim koordinatama i zatim ih konvertirati u kartezijske.
Neka je zadaća stvoriti i prikazati konture funkcije $f = (Z^3-1)^{1/3}$.

```
>> [th,r] = meshgrid((0:5:360)*pi/180,0:.05:1);
>> [X,Y] = pol2cart(th,r);

%Stvaranje kompleksne matrice na nutarnjoj
%jediničnoj kružnici.
>> Z=X+i*Y;   %X, Y, i Z su točke unutar kružnice.

%Stvaranje i prikazivanje površine zadane
%funkcijom f.
>> f = (Z.^3-1).^(1/3);
>> contour(X,Y,abs(f),50)
>> axis equal
```

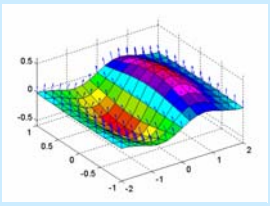


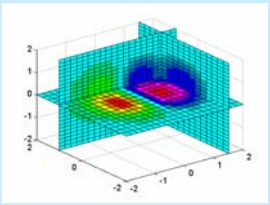
Primjer 5.16: *Trodimenzionalni grafički prikaz funkcije u ovisnosti o tri prostorne koordinate, te grafički prikaz polja pomoću vektora (strelica).*

quiver3(), surfnorm(), slice()

`quiver3()` - *grafički prikaz funkc.* $z = f(x,y) = x \cdot e^{\left(-x^2-y^2\right)}$ *pomoću strelica*
`surfnorm()`- *izračunava i grafički prikazuje normale površine u obliku linija*
`slice3()` - *prikaz funkcije* $v = f(x,y,z) = x \cdot e^{\left(-x^2-y^2-z^2\right)}$ *u 3D prostoru*

```
% prva slika
>> [X,Y] = meshgrid(-2:0.25:2,-1:0.2:1);
>> Z = X.* exp(-X.^2 - Y.^2);
>> [U,V,W] = surfnorm(X,Y,Z); %računa normale
>> quiver3(X,Y,Z,U,V,W,0.5);  %crta strelice
>> hold on %zadrzava nacrtane strelice (normale)
>> surf(X,Y,Z);  %crta povrsinu uz strelice
>> colormap hsv
>> view(-35,45)
>> axis ([-2 2 -1 1 -.6 .6])
>> hold off

% druga slika
>> [x,y,z] = meshgrid(-2:.2:2,-2:.25:2,-2:.16:2);
>> v = x.*exp(-x.^2-y.^2-z.^2);
>> slice(x,y,z,v,1.5,1,0)
```





5.5. Upravljanje bojama

Postoje dva načina kako se zadaju boje u Matlabu:

- RGB trojac – je troelementni redčani vektor čiji elementi određuju intenzitet crvene, zelene i plave komponente boje. Intenziteti moraju biti u opsegu [0 1].
- Kratkim ili dugim imenom - Matlab riječi (stringovi) koje definiraju jednu od osam predodređenih boja

Tablica 5.8: *Označavanje boja*

RGB vrijednost	Kratko ime	Dugo ime
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[0 1 1]	c	cyan
[1 0 0]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue
[1 1 1]	w	white
[0 0 0]	k	black

Osam unaprijed zadanih boja i bilo koja boja koja se zadaje RGB vrijednostima zovu se čvrste (fiksne) boje. S njima se mogu crtati osi, linije i oznake.

Druga kategorija boja poznata je kao mapa boja (engl. *colormap*). Mapa boja je jednostavna trostupčana matrica čija duljina je jednaka broju boja koje ona definira. Svaki redak matrice određuje pojedinačnu boju specificiranjem triju vrijednosti u opsegu od 0 do 1. Ove vrijednosti, kako znamo, definiraju RGB komponente (tj. intenzitet crvene, zelene i plave video komponente)

Tablica 5.9: *Mape boja*

Mapa boja	Opis mape boja
hsv	Hue-saturation-value mapa. Njezine boje počinju s crvenom, pa preko žute, zelene, cyan, plave i magente opet se vraćaju na crvenu
hot	Crna-crveno-žuto-bijela mapa boja
gray	Linearna mapa sivih boja
bone	Mapa sivih boja s prozirno plavim obrisima (poput kostiju)
copper	Linearna bakrenasta mapa boja
pink	Pastelne sjene ljubičaste mape boja
white	Mapa bijele boje
flag	Izmjenljiva crvena, bijela, plava i crna mapa boja
lines	Mapa zadana s ColorOrder svojstvom osi i sivim sjenčanjem.
colorcube	Pojačana mapa boja kocke
vga	Mapa sa 16 boja
jet	Varijanta od HSV. Prolaz od plave do crvene i prolazi bojama cyan, žuta i narančasta.
prism	Mapa boje prizme. Ponavljanje 6 boja: crvena, narančasta, žuta, zelena, plava i ljubičasta.
cool	Sjenčanje s cyan i magenta mapom boja (hladne boje)
autumn	Sjenčanje (vrlo postupno) sa crvenom i žutom mapom boja (boje jeseni)
spring	Sjenčanje s magenta i žutom mapom boja (boje proljeća)
winter	Sjenčanje s plavom i zelenom mapom boja (boje zime)
summer	Sjenčanje s zelenom i žutom mapom boja (boje ljeta)

Samo površine, mnogokuti i slike mogu koristiti mape boja. Matlab boja ove objekte na temelju poretka boja u mapi. Svaki Matlab grafički prozor ima pridruženu mapu boja. Funkcija `colormap()` bez argumenata vraća trenutačno postavljenu mapu boja radnog grafičkog prozora. Dok

```
colormap(map)
```

postavlja mapu boja zadanu matricom **map**.

Mape boja mogu se stvarati nove ili koristiti postojeće. Svaka mapa ima dodatni parametar koji određuje broj redaka rezultirajuće mape. Tako će na primjer,

```
>> hot(m)
```

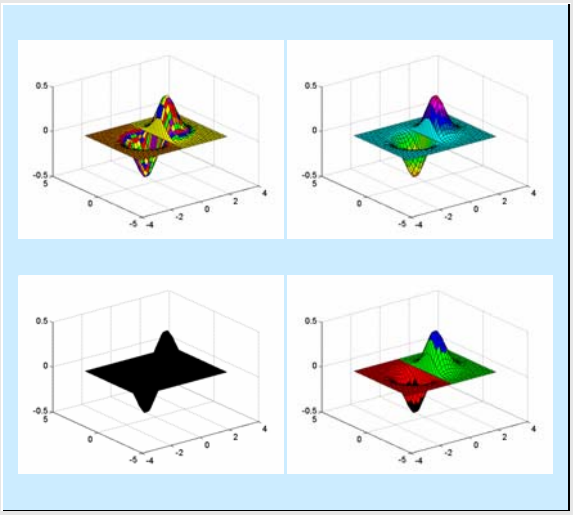
stvoriti jednu (m x 3) matricu čiji se redovi mijenjaju od crne, preko sjena crvene, narančaste i žute do bijele. Ako se ne definira duljina mape boja, onda Matlab stvara mapu boja isti duljine kao trenutačna mapa. Pretpostavljena mapa je `jet`(64).

Primjer 5.17: *Mapa boja (colormap)*

colormap()

`colormap()` - *sadrži matricu sa tri stupca (RGB, tj .red, green, blue) i proizvoljan broj redaka koji označavaju nijanse, tj. kombinacije tih triju boja. Ti brojevi se kreću od 0÷1 i npr. redak s komponentama RGB boja koji se sastoji od [0 0 0] označava crnu boju, [1 0 0] crvenu, [0 1 0] zelenu, a [0 0 1] plavu boju.*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> surf(X,Y,Z);
% slika na mjestu (1,1)
>> colormap (prism);
% slika na mjestu (1,2)
>> colormap (hsv);
% slika na mjestu (2,1)
>> boja1=[0 0 0];
>> colormap (boja1);
% slika na mjestu (2,2)
>> boja2=[0 0 0;1 0 0;
          0 1 0;0 0 1];
>> colormap (boja2);
```



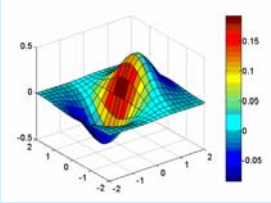
Funkcija `colorbar` prikazuje trenutačnu mapu boja bilo vertikalno ili horizontalno, u istom grafičkom prozoru zajedno s grafom.

Primjer 5.18: *Grafički prikaz mape boja na grafu*

colorbar()

Na grafu zadane površine će nacrtati vertikalnu vrpcu s bojama i numeričkim oznakama. To nam omogućuje lako određivanje iznosa z-koordinate površine u grafu, dovoljno je uočiti boju i očitati numeričku vrijednost sa vrpce boja.

```
>> [x,y] = meshgrid([-2:.2:2]);
>> Z = x.*exp(-x.^2-y.^2);
>> surf(x,y,Z,gradient(Z))
>> colorbar
```



U ovoj kategoriji mogu se navesti još neke zanimljive grafičke naredbe, koje se brinu za poboljšanje crteža: `caxis()`, `shading()` i `hidden`.

Tablica 5.10: *Grafičke funkcije za poboljšanje crteža*

Funkcija	Opis funkcije
`caxis()`	Umjeravanje osi preko pseudoboje.
`shading()`	Sjenčanje s bojama.
`hidden`	Prikaz ili skrivanje sakrivenih linija.

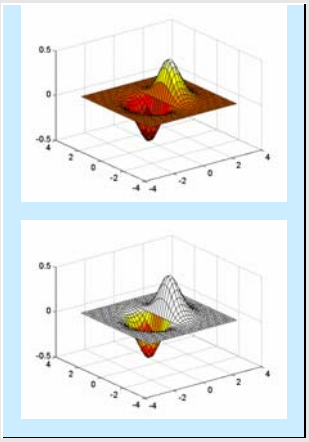
Primjer 5.19: *Kontrola prikaza boja na površinama, plohama i slikama*

caxis()

`caxis()` - *kontrolira nivo vrijednosti boja u mapi boja. U ovom primjeru postoji matrica vrijednosti (npr.* $z = f(x,y) = x \cdot e^{\left(-x^2-y^2\right)}$*, gdje "z" predstavlja točke površine funkcije u 3D prostoru) koja ima vrijednosti u rangu između neke minimalne i maksimalne vrijednosti. Kada bi pritom pridijelili tim vrijednostima jednu od mapa boja, onda bi se ta mapa boja prilagodila vrijednostima matrice "z", te bi za najmanju vrijednost matrice "z" pridijelila najmanju vrijednost iz mape boja, a za najveću vrijednost matrice "z" pridijelila najveću vrijednost iz mape boja. Vrijednosti između bile bi interpolirane. Time bi se dobila kontrola boja nad proizvoljnom matricom "z" (ta matrica može biti npr. matrica učitane slike u Matlab). U slučaju da želimo mapu boja pridijeliti nekom drugom rangu u matrici "z" koji se nalazi između minimuma i maksimuma naredba* `caxis()` *će to napraviti tako da će vrijednostima koje nisu uključene u taj rang pridijeliti granične boje iz boje mape.*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> surf(X,Y,Z);
```

```
>> colormap hot
>> minimum=min(min(Z));  %minimum = -0.4218
>> maximum=max(max(Z));  %maximum = 0.4218

% prva slika
>> caxis([minimum maximum])

% druga slika
>> caxis([minimum 0])
```

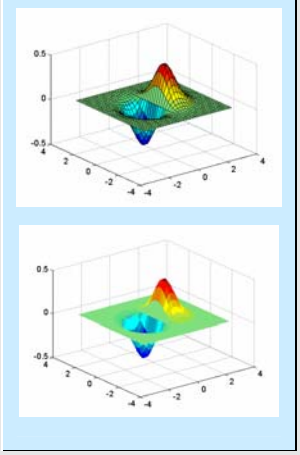


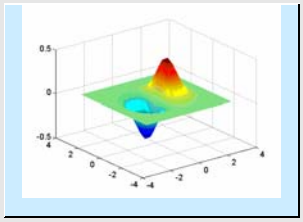
Primjer 5.20: *Sjenčanje trodimenzionalne površine*

shading - (hrv. sjenčanje)

`shading faceted()` - *sjenčanje 3D površine tako da svaki segment mreže ima konstantnu nijansu boje, dok se mreža ističe posebnom bojom (standardna je crna)*
`shading flat()` - *sjenčanje 3D površine tako da svaki segment mreže ima konstantnu nijansu boje, dok je segment mreže u istoj boji kao i segment površine.*
`shading interp()` - *sjenčanje 3D površine tako da se različito obojani segmenti površine interpoliraju u nijanse, pa je površina prikazana pomoću nijansi.*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> surf(X,Y,Z);
>> colormap jet

% prva slika
>> shading faceted

% druga slika
>> shading flat

% treca slika
>> shading interp
```



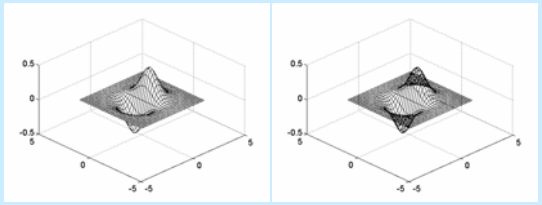


Primjer 5.21: *Prikaz skrivenih linija mreže na prostornim površinama*

hidden

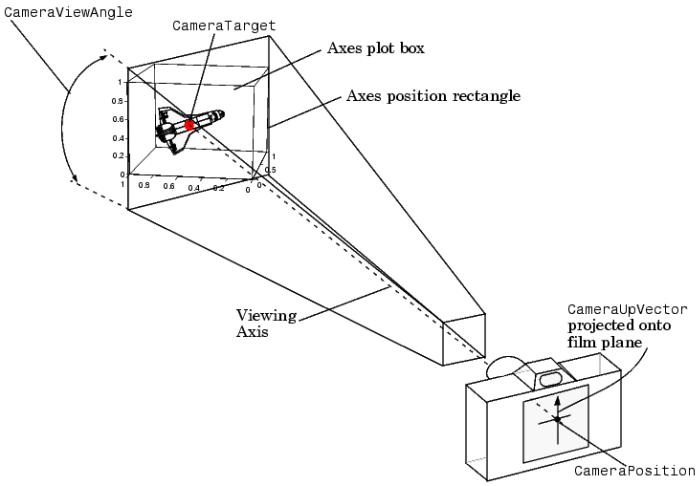
`hidden` *on - ne prikazuje skrivene linije mreže*
`hidden` *off - prikazuje skrivene linije mreže*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> mesh(X,Y,Z);
% prva slika
>> hidden on
% druga slika
>> hidden off
```



5.6. Kamera

MATLABova grafika s pomoću kamere daje veće upravljanje pogledom nego podešavanje preko azimuta i elevacije. Kad se gleda neki grafički objekt prikazan u osima, onda se vidi grafička scena iz neke točke u prostoru koja ima stanovitu orijentaciju s obzirom na scenu. Matlab ima ugrađenu funkcionalnost koja je analogna onoj u radu s kamerom, kako bi se što bolje upravljalo pogledom na scenu. Kamera ima mogućnost zumiranja, pomicanja i okretanja. Slika ispod pokazuje sve važne značajke koje ima kamera s obzirom na osi u kojima promatra neku grafičku scenu.



Slika 5.2: *Upravljanje pogledom pomoću kamere*

Naredba `campos()` daje ili postavlja poziciju kamere:

```
>> campos([camera_position])
```

camera_position je tročlani vektor koji sadrži x-, y- i z-koordinate željene lokacije u koju želimo smjestiti kameru.
Postoji niz funkcija za rad s kamerom, od kojih će neke biti detaljnije pokazane u poglavlju o temeljnoj, handle grafici.

Tablica 5.11: *Funkcije za rad s kamerom*

Funkcija	Opis funkcije
`camdolly()`	Pomiče poziciju kamere i objekt gledanja.
`camlookat()`	Gleda zadani objekt.
`camorbit()`	Kruži kamerom oko promatranog cilja.
`campan()`	Rotira ciljni objekt oko pozicije kamere.
`camproj()`	Dobiva ili postavlja tip projekcije (ortografska ili perspektiva).
`campos()`	Dobiva ili postavlja položaj kamere (x,y,z koordinate).
`camroll()`	Rotira kameru oko osi promatranja.
`camtarget()`	Dobiva ili postavlja lokaciju ciljnog objekta.
`camup()`	Dobiva ili postavlja vrijednost Up vektora kamere (okomiti v.).
`camva()`	Dobiva ili postavlja vrijednost kuta kamere.
`camzoom()`	Zumira scenu kamerom (povećanje ili smanjenje scene).

5.7. Osvjetljenje

Osvjetljenje je tehnika dodavanja realnog pogleda u grafičku scenu. To se postiže simulacijom osvijetljenih i tamnih površina koje se pojavljuju pod objektima osvijetljenih prirodnim svjetlom (npr. usmjerenog svjetla koje dolazi od sunca). Simulacija se može podešavati stvaranjem posebnog izvora svjetla, izborom materijala površine, kutom upadnih i odbijajućih zraka izvora svjetla, načinima osvjetljavanja i sl.

Među temeljnim funkcijama za osvjetljenje ističu se: `surfl()`, `lighting()`, `material()`, `specular()`, `diffuse()`, `brighten()`

Tablica 5.12: *Funkcije za osvjetljenje*

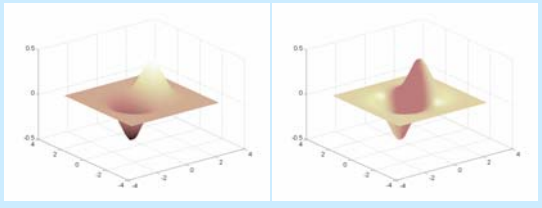
Funkcija	Opis funkcije
`surfl()`	3-D osjenčana površina s osvjetljenjem
`lighting()`	Načini osvjetljavanja
`material()`	Vrste svjetlosno odbijajućih materijala
`specular()`	Odbijanje svjetla
`diffuse()`	Difuzno odbijanje svjetla
`brighten()`	Posvjetljenje ili zatamnjenje mape boja

Primjer 5.22: *Osvjetljavanje površina objekata*

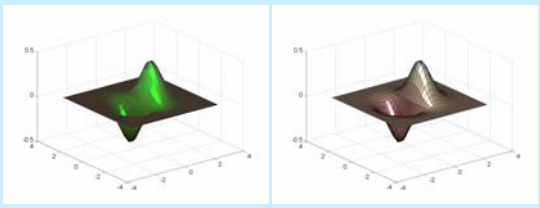
surfl(), light(), camlight()

`surfl()` - *crta 3D površinu s osvjetljenjem*
`light()` - *postavlja izvor svjetlosti koji obasjava 3D površinu. Mogu se odrediti mjesto izvora svjetla (kao vektor koordinata x,y,z-osi), boja osvjetljenja i još nekoliko drugih svojstava.*
`camlight()` - *postavlja izvor svjetlosti u kamera koordinatama. To znači da se izvor svjetlosti postavlja u odnosu na kameru kroz koju se promatra objekt površine. Zamislimo kameru kroz koju gledamo objekt, te imamo reflektor koji postavljamo da bismo ga osvijetlili. Tako reflektor može biti usmjeren kamo i kamera (headlight), lijevo od nje (left) ili desno od nje (right), a može se odrediti položaj svjetla zadavanjem azimuth-a i elevation-a([az,el]).*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
%slika na mjestu (1,1)
>> surf(X,Y,Z);
>> colormap pink
```



```
>> shading interp
%slika na mjestu (1,2)
>> surfl(X,Y,Z);
>> shading interp
%slika na mjestu (2,1)
>> surf(X,Y,Z);
>> shading interp
>> light('Position',[-5 -5
0],'Color','g');
%slika na mjestu (2,2)
>> surf(X,Y,Z);
>> shading interp
>> camlight('headlight')
```

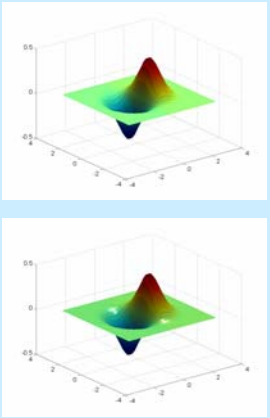


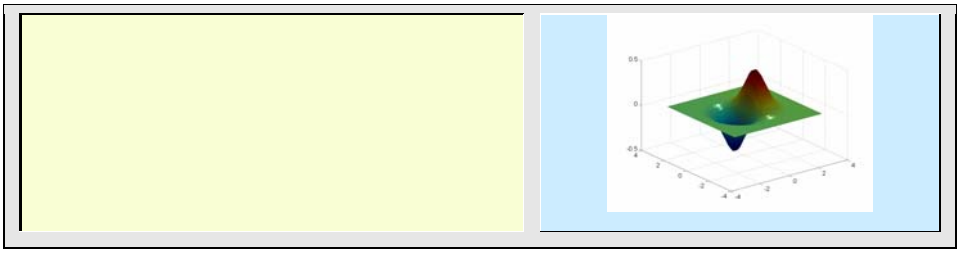
Primjer 5.23: *Refleksija površina objekata*

material

`material()` - postavlja refleksiju površine objekta tako, da izgleda kao da je obrađena s nekom određenom preciznošću i napravljena od nekog određenog materijala (npr. sjajan, bez sjaja, metalan itd.).

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> surf(X,Y,Z);
>> shading interp
>> light

% prva slika
>> material dull      % bez sjaja

% druga slika
>> material shiny     % sjajan

% treća slika
>> material metal     % kao metal
```



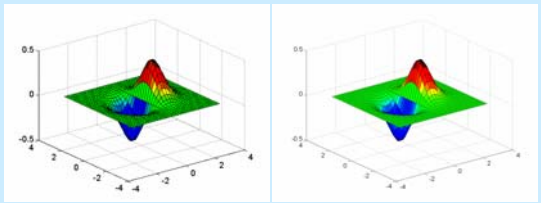


Primjer 5.24: *Posvjetljenje i potamnjenje mape boja*

brighten()

`brighten()` - *koristi se za posvjetljenje ili potamnjenje mape boja. Kada je argument ove funkcije negativan (između -1 i 0) onda se radi o potamnjenju, a kada je između 0 i 1 o posvjetljenju.*

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> surf(X,Y,Z);
% prva slika
>> colormap jet
>> brighten(-0.8)
% druga slika
>> colormap jet
>> brighten(0.8)
```



5.8. Modeliranje površina

MATLAB posjeduje funkcije za generiranje posebnih površina, među kojima se ističu: `cylinder()`, `sphere()` i `patch()`

Tablica 5.13: *Funkcije za generiranje posebnih površina*

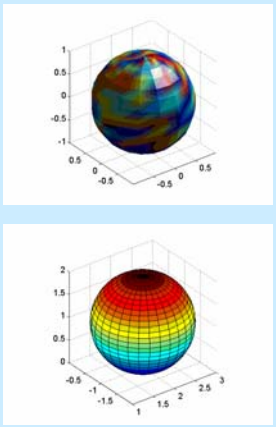
Funkcija	Opis funkcije
`cylinder()`	Generira stožac.
`sphere()`	Generira kuglu.
`patch()`	Stvara mnogokut (patch).

Centar sfere možemo translatirati vrlo lako. U slijedećem primjeru, nacrtati ćemo graf jedinične sfere sa centrom u (2,-1,1)

Primjer 5.25: *Generiranje kugle*

Naredba `sphere`*(n) generira točke jedinične kugle s centrom u ishodištu koristeći* $(n+1)^2$ *točaka. Ako se funkcija pozove bez argumenata, onda Matlab pretpostavlja n=20. Točke se s pomoću funkcije* `surf()` *mogu nacrtati, a funkcijom* `shading` *i prikladnim svjetlom postići trodimenzionalni efekt.*
U ovom slučaju korišteno je tzv. Gouraud-ovo sjenčanje u kojem se boja svakog segmenta linearno mijenja, a na rubovima dobiva interpolirane vrijednosti.
Centar sfere možemo translatirati vrlo lako. Na drugoj slici, nacrtati ćemo graf jedinične sfere sa centrom u (2,-1,1).

```
%prva slika
>> [X,Y,Z]=sphere(15);
>> C=rand(16,16);
>> surf(X,Y,Z,C)
>> axis equal
>> shading interp
>> light

%druga slika
>> [x,y,z]=sphere(30);
>> surf(x+2,y-1,z+1)
>> axis equal
```



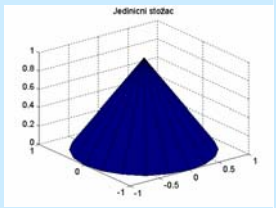
Funkcija `sphere()` zajedno sa funkcijama `surf()` ili `mesh()` može biti iskorištena za crtanje grafa kugle ili elipsoida.

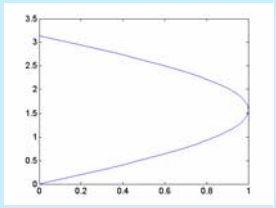
Funkcija `cylinder()` koristi se za crtanje površine stošca. Moguća su dva ulazna parametra. U naredbi `cylinder`(r,n) argument **r** predstavlja vektor koji definira polumjer stošca duž z-osi, a **n** specificira broj točaka korištenih za definiranje obodnice stošca. pretpostavljene vrijednosti ovih parametara su **r = [1 1]** i **n=20**. Nacrtani stožac ima jediničnu visinu.

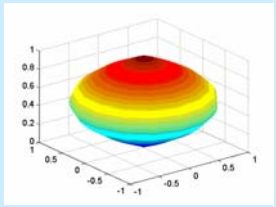
Primjer 5.26: *Generiranje stošca*

Na prvoj slici prikazan je stožac s osnovnim polumjerom jednakim jedan i jediničnom visinom.Na drugoj i trećoj slici prikazan je graf površine nastao rotacijom krivulje r(t)=`sin`*(t), t>0 oko y osi. Graf generirane krivulje i površine rotacijskog tijela dobije se s par linija programskog koda:*

```
% stozac s polumjerom 1 do polumjera 0 po z-osi
>> cylinder([1 0])
>> title('Jedinicni stožac')

%druga slika
>> t=0:pi/100:pi;
>> r=sin(t);
>> plot(r,t)

%treca slika
>> cylinder(r,15)
>> shading interp
```







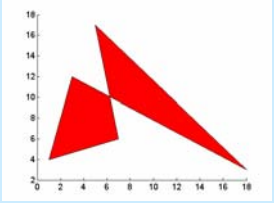
Jedan vrlo jednostavni poligon (patch) može se dobiti upotrebom opće sintakse:

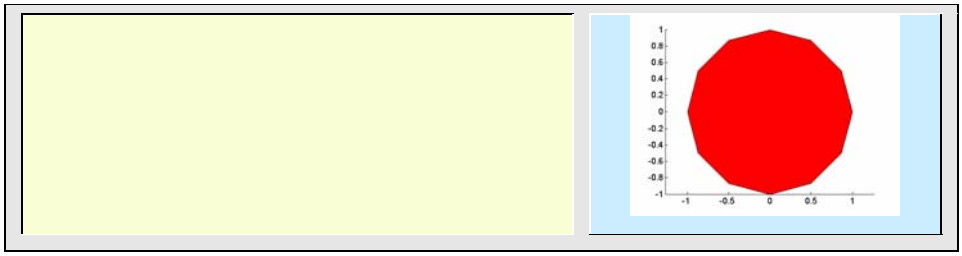
```
patch(x-coordinates,y-coordinates,[z-coordinates],colordata)
```

Primjer 5.27: *Generiranje poligona*

Poligoni se mogu generirati određivanjem točaka tj. koordinata putem zadanih vektora ili zadavanjem točaka funkcijama

```
%dva vektora
>> patch([7,1,3,18,5],[6,4,12,3,17],'r')

%ili generiranjem
>> t = 0:pi/6:2*pi;
>> patch(sin(t),cos(t),'r')
>> axis equal
```





5.9. Rad sa slikama

Profesionalna obrada slika u Matlabu uključuje instalaciju *Image processing toolbox*-a, dok osnovni rad traži poznavanje samo nekoliko naredbi za čitanje slika, poznavanje formata slike, osnovne transformacije i spremanja.

Tablica 5.14: *Osnovne funkcije za obradu slike*

Funkcija	Opis funkcije
`image()`	Prikazuje sliku
`imagesc()`	Umjerava podatke i prikazuje sliku
`contrast()`	Povećava kontrast slike umjeravanjem sive mape boja
`imread()`	Čitanje slike iz datoteke
`imwrite()`	Spremanje slike u grafičku datoteku
`imfinfo()`	Informacija o grafičkoj datoteci

Matlab podupire različite formate slika, kao što su:

BMP (Microsoft Windows Bitmap)
HDF (Hierarchical Data Format)
JPEG (Joint Photographic Experts Group)
PCX (Paintbrush)
PNG (Portable Network Graphics)
TIFF (Tagged Image File Format)
GIF (Graphics Interchange Format)

Matlab podržava četiri tipova slika:

- binarni (eng. *binary*)
- indeksirani (eng. *indexed*)
- tonalitetna (eng. *intensity*)
- RGB (Crvena-Zelena-Plava eng. *Red-Green-Blue*)

Tipovi slike predstavljaju tipove odnosa između vrijednosti elemenata matrice slike i boje piksela (eng. *pixel*). Piksel (picture element) predstavlja jednu točku slike te kao primjer jedna slika dimenzija 600x480 piksela će biti pohranjena u matricu sa 600 redaka i 480 stupaca. Da bi se prikazala boja jednog piksela na ekranu potrebno je imati određenu memoriju (npr. 1 bit, 8 bita, 24 bita ili neku drugu vrijednost bitova) u koju će se pohraniti vrijednost te boje.
Ako se za jedan piksel odredi 1 bit onda se mogu prikazati samo vrijednosti 0 i 1 što predstavlja dvije boje. Ako se odredi 8 bita onda se mogu prikazati 256 boja, dok se za 16 bita 65536 boja, a za 24 bita 16.777.216 (ili 2^{24}) boja. Kod 24 bitne boje za 1 piksel se odredi 3 puta po 8 bitova (po 256 kombinacija) i na taj način se prikazuje RGB slika. Svaka od te tri boje dobije po 8 bitova (256 nijansi te boje) i u kombinaciji se dobije 256*256*256=16.777.216 nijansi boja za 1 piksel.
Budući da Matlab za skoro sve svoje radnje koristi tip podatka *double* (64-bitni tip sa pomičnim zarezom) onda slike sa npr. 1000x1000 piksela tvore matricu od 1 miliona elemenata (8 bita=1 bajt, =>64 bita=8 bajtova => 1000.000 elemenata = 8000.000 bajtova = 8MB). Spremanje slika s takvim memorijskim zauzećem zauzima puno memorije pa se vrlo često koristi pretvorba npr. *double* - *uint8* ili *uint16*. Time se smanjuje memorija za pohranjivanje slike 8 ili 4 puta.

Tablica 5.15: *Tipovi slika*

Tip slike	Opis tipa slike
binarni	Binarni tip boja (1 bit) sadrži dvije vrijednosti 0 i 1, što najčešće označava crno i bijelo.
indeksirani	Indeksirana slika ima vrijednosti piksela koji predstavljaju pokazivače na RGB mapu boja. Indeksirana slika je matrica pokazivača na mapu boja, dok je mapa boja matrica sa 3 stupca (prvi stupac su nijanse crvene boje, drugi zelene, a treći plave boje). To znači da ako neki element (piksel) indeksirane matrice ima vrijednost npr.4, onda će se na ekranu prikazati boja koja je predstavljena trećim retkom iz mape boja (3 elementa se nalaze u 3.retku).
tonalitetni	Slika stvorena pomoću tonalitetne matrice je dobivena tako da vrijednosti svakog njenog elementa (piksela) prikazuju vrijednosti boje piksela. Budući da ne postoji mapa boja koja je direktno povezana s vrijednostima matrice onda se tim vrijednostima pridružuje automatski trenutnu sistemsku mapu boja. Najčešće se koristi siva mapa boja za prikazivanje ovog tipa slike.
RGB	Slika kojoj je svaki piksel prikazan pomoću 3 elementa (R,G,B). Matrica RGB slike je trodimenzionalna matrica dimenzije m-n-3 (zamislimo 3 matrice jedna iz druge pri čemu prva predstavlja crvenu boju, druga zelenu, a treća plavu boju, te je boja jednog piksela sastavljena od elemenata na istim mjestima tih triju matrica).

U slučaju da se nad učitanom slikom primjenjuju neke operacije (npr. filtriranje) onda će se vrlo često trebati konvertirati iz jednog u neki drugi tip slike. Npr. ako je učitana slika indeksiranog tipa (tip i ostale informacije provjeriti naredbon `imfinfo`) i hoćemo primijeniti neki filter na nju, onda će filter djelovati na matricu pokazivača koji pokazuju na matricu mape boja, pa se dobiti drugačija slika od željene. Stoga je potrebno prvo pretvoriti indeksiranu sliku u RGB sliku (naredba `ind2rgb`) i tek onda primijeniti filter.
Budući da Matlab vrlo često učitava sliku kao matricu *uint8* tipa podatka (kod indeksirane matrice mape boje uvijek će učitavati elemente kao *double* tip podatka) onda tu sliku možemo prikazati nekom od naredbi za njeno prikazivanje (npr. `imshow()`), ali ako želimo raditi neke matematičke operacije nad tim elementima npr. zbrajanje to neće biti moguće i Matlab će javit grešku (??? Error using ==> + Function '+' is not defined for values of class 'uint8'.), jer Matlab radi samo s *double* tipom podatka i zato će biti potrebno matricu slika pretvoriti u *double* naredbom `im2double`.
U slučaju da se indeksirana slika u boji učita u Matlab (naravno naredbom `imread`) sa svojom matricom pokazivača i matricom mape boja i želi se pretvoriti (naredbom `ind2gray`) u tonalitetni tip slike onda će se dogoditi to da će slika u boji postati slika u sivim nijansama. Automatski se prvo RGB mapa boja indeksirane slike prebaci u NTSC koordinate (tj. u definiranje boja pomoću osvjetljenja- eng. *luminance*, nijansi boja- eng. *hue* i zasićenja- eng. *saturation*, pri čemu osvjetljenje predstavlja informacije o sivim nijansama, a ostala dva faktora predstavljaju informacije o bojama). Tada se u novoj mapi boja u NTSC koordinatama postave vektori nijansi boja i zasićenja na nulu, a ostanu jedino vrijednosti osvjetljenja tj. sivih nijansi boja. Nakon toga se matrica pokazivača indeksirane matrice zamijeni odgovarajućim vrijednostima iz nove (sive) mape boja i tada je stvorena nova matrica slike tonalitetnog tipa.

5.9.1. Formati prikazivanja slika

Postoje dva formata slika: rasterski i vektorski.

Rasterski format prikazivanja (tzv. bitmape) je vrlo raširen u internetskoj tehnologiji. Temelji se na pikselnom prikazivanju slike. Slika se promatra kao skup točaka (piksela). Neki od formata su BMP, GIF, JPEG, TIFF, PNG itd.
Vektorski format prikazivanja slike prikazuje sliku pomoću matematički definiranih oblika (krugom, pravcem, trokutom, kvadratom…). Format je neovisan o razlučivosti slike, kvaliteta slike je visoka, ali i memorijsko zauzeće je veliko. Budući da je crtanje oblika koje prati matematička podloga moguća s programima za crtanje tako i slike koje se spremaju u vektorskom formatu dobivaju se iz tih aplikacija. Npr. CorelDraw(.cdr), Hewlett-Packard Graphic Language (.hgl), Windows Metafile (.wmf), Adobe Illustrator (.ai), Enhanced Metafiles(.emf) - Matlabov vektorski format za Microsoft aplikacije, itd. Vektorski formati omogućuju puno preciznije rukovanje s elementima slike budući da su zasnovani na matematičkim izrazima, za razliku od rasterskih slika.

Tipovi komprimiranja podataka (slika): komprimiranje bez gubitaka (eng. *lossless compression*) i sa gubitkom (eng. *lossy compression*)

Komprimiranje bez gubitaka se odnosi na takvo komprimiranje podataka pri kojem se ne gubi niti jedan bit iz informacijskog sadržaja što pritom omogućava dekomprimiranje sadržaja u potpunosti. Za komprimiranje slika koristi se tzv. Run Length Encoding (RLE). RLE kompresijski algoritam komprimira podatke na taj način da traži u nizu podataka one podatke koji su isti i koji se ponavljaju jedan za drugim, te ih zamjenjuje sa dva podatka, brojem ponavljanja tog podatka i samog podatka (npr. AAAAACDD će komprimirati u 5A1C2D). Dobro komprimiranje podataka će biti u slučaju mnogo istovrsnih podataka dok u slučaju mnogo raznovrsnih podataka nakon kompresije veličina datoteke može čak i porasti (npr. A -> 1A, vidimo da od jednog slova nakon komprimiranja nastanu dva!).
Zbog svojih svojstva RLE komprimiranje slike je jako korisno ako se želi komprimirati sliku bez gubitaka informacijskog sadržaja i u slučaju da se na slici nalazi veća područja s istobojnim pikselima. Taj tip komprimiranja koristi algoritam takav da dijelove slike koji sadrže određen broj piksela u istoj boji sažme u jedan piksel te boje. Pritom će algoritam uvijek moći dekomprimirati sliku u izvornu bez gubitaka.
Postoji još jedna metoda koja se koristi pri komprimiranju slika, a to je LZW komprimiranje (ime dobiveno po autorima Lempel-Ziv-Welch). Metoda se temelji na indeksiranom zapisu tipa slike, i to na taj način da se pronađu na slici sve postojeće boje i pohrane u mapu boja, a pritom se u matrici indeksa pohrane za svaki piksel reference na boju iz mape boja. Kao i RLE kompresija ova metoda je pogodna za jednostavne slike s većim područjima obojenim istom bojom. U slučaju da je originalna slika 24-bitna onda se iz mogućih 16.7 miliona nijansi boja ovom kompresijom svodi na mapu od 256 boje (8-bita) čime se trostruko smanjuje veličina slike. U slučaju da u 24-bitnom zapisu slike nema sveukupno različitih boja više od 256 onda LZW komprimiranje je bez gubitka. Ako slika ima mnogo nijansi i ukupan broj boja je veći od 256 boja onda sažimanjem tih boja na 256 će se neke nijanse izgubiti i onda je to komprimiranje uz gubitak. LZW se najviše koristi kod GIF formata slika gdje su velike površine obojane istom bojom i ima ih 256 ili manje.

Komprimiranje s gubitkom se odnosi na takvo komprimiranje pri kojem se neki podaci na neki način zanemaruju ali sama jezgra informacije ostaje sačuvana. Nakon dekomprimiranja nisu sačuvani svi podaci kao na originalu ali su izgubljene informacije zanemarujuće za glavni dio informacije koji se prenosi. Tzv. *lossy compression* je vrlo bitna i jako korištena metoda komprimiranja osobito ma WEB-u je uz veću kompresiju veličina datoteke ili slike se smanjuje. Prednost ove metode je i u tome što se može izabrati stupanj kompresije, a komprimiranje se izvodi na način da se na slici pronađu pikseli sličnih nijansi boje i zamijene se jednom bojom. Najpoznatiji format je JPEG.

Neki tipovi rasterskih formata slika:

BMP – **B**asic **M**ultilingual **P**lane, poznatija pod nazivom Windows BitMap. BMP je standardni Microsoft Windows rasterski format slike. Svaki piksel prikazivanja slike može biti 1,4,8 ili 24-bita, pa se tada svaki piksel može prikazati maksimalno u nešto više od 16 miliona boja (kod 24-bitnih slika). BMP format slike nije komprimiran, pa time nije ni pogodan za prikazivanje na WEB-u jer takva slika zauzima dosta memorije.

BitMap slika se može i komprimirati te se za to koristi komprimiranje bez gubitaka, tzv. run-length encoding (RLE). RLE komprimiranje BMP formata podržava slike s 4 (16 boja) i 8-bita (256 boja) po pikselu, dok je ekstenzija komprimirane slike ili bmp ili rle. Najčešće se koristi u Windows aplikacijama, te npr. kao wallpaper (slika na Desktopu).

GIF – **G**raphics **I**nterchange **F**ormat, 8-bitni format (256 boje) koji se koristi u internet tehnologiji. Prvi GIF format na WEB-u je bio GIF87a što je predstavljalo godinu 1987 i verziju a. Godine 1989 je predstavljen novi GIF format nazvan GIF89a. Noviji format ima svojstva ispreplitanja (interlacing), prozirnost (transparentnost) i animaciju.
Ispreplitanje se odnosi na prividnom bržem učitavanju slike na WEB-u, upravo zbog sporosti učitavanja nekih većih slika. Kada se slika prikazuje bez ispreplitanja onda se pojavljuju piksel po piksel u redu i tako od vrha stranice prema dnu. Zato se mora čekati da se slika u potpunosti učita tj. prikaže da bi se vidjelo što je na njoj. Kod ispreplitanja se slika učitava kroz više prolazaka. U prvom prolasku se prikazuje samo oko 12.5% slike (svaki osmi red). Zatim se slika prikazuje još tri puta, što znači da se u drugom prolasku učita još 12.5% (prikazan je pritom svaki četvrti red), u trećem sljedećih 25% (popunjen svaki drugi red) i tek u četvrtom prolasku se učita ostalih 50% slike. Prednost ovog učitavanja je da korisnik već kod 30-50% učitane slike (slika još uvijek izgleda zamućeno) može steći uvid u cijelu sliku. Ovo svojstvo podržavaju i stariji i noviji format.
Prozirnost nudi mogućnost prikazivanja slike na nekoj pozadini koja će se vidjeti kroz sliku. Oni pikseli koji su proglašeni prozirnima se jednostavno ne prikazuju na ekranu pa na tim mjestima ostaje boja podloge. Prozirnost podržava samo noviji format.
Animacija nudi mogućnost da nekoliko GIF sličica, promatranih u nizu čine sliku dinamičnom. Animaciju podržava samo GIF89a.
GIF slike su LZW komprimirane slike. GIF slike se koriste za prikazivanje logo slika, ilustracija, ikona, gumba itd.

JPEG – **J**oint **P**hotographic **E**xpert **G**roup, 24-bitni format (oko 16.7 miliona nijansi boja po pikselu), te je najkorišteniji format prikazivanja slika na WEB-u. JPEG slike su slike komprimirane s gubitkom, ali ti gubici su gotov neprimjetni ljudskom oku. Format ne podržava ispreplitanje, prozirnost i animaciju te nije dobar za slike s većim područjima obojenih istom bojom kao ni za one slike koje nemaju puno nijansi i prijelaza boja. Ako se hoće prikazati fotografija (što je učestala pojava na WEB-u) sa velikim spektrom boja i nijansi onda JPEG itekako dolazi u obzir. Algoritam kompresije koristi metodu grupiranja sličnih nijansi boja i zamjenjuje ih jedinstvenom vrijednošću boje koja se dobiva diskretnom kosinusovom transformacijom i drugim algoritmima koji određuju važnost određene informacije koja se prikazuje. Ako se JPEG slika uređuje (eng. *edit*) dolazi do gubitaka, te čak i rotacija nosi gubitke. Stoga JPEG sliku nikad ne treba uređivati niti višestruko komprimirati i dekomprimirati jer nakon toga slika neće više dobro izgledati. Velika prednost ovog formata je da se može izabrati stupanj kompresije, te da i pri većim kompresijama slika dobro izgleda, a veličina joj se izuzetno smanjila.
Progresivni JPEG prikaz slike je baziran na ispreplitanju, što je slično GIF ispreplitanju ali se razlikuju po tome da je JPEG ispreplitanje dvodimenzionalno.

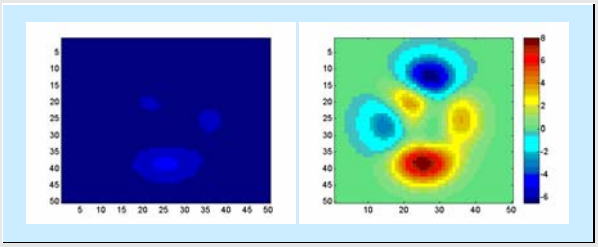
TIFF – **T**agged **I**mage **F**ile **F**ormat, maksimalno 24-bitni format. Ovaj format je izvrstan za čuvanje i spremanje slika na računalu te se koristi u prijenosu slika između aplikacija i kompjuterskih platformi (Windows, Unix, Macintosh …). TIFF format podržava proizvoljnu dubinu piksela (do max. 24-bita) za prikazivanje boja i podržava velik broj metoda komprimiranja (RLE, LZW, JPEG, ZIP itd.). Kao nekomprimiran TIFF format sadrži visoku kvalitetu slike, ali zato i slika zauzima puno memorije. To ograničava da se taj format prikazuje na WEB-u. Za prikaz slike na WEB-u potrebno je sliku iz TIFF formata komprimirati u JPEG ili GIF format. TIFF se može proizvesti iz gotovo svakog skenera, te ispis iz ovog formata nudi kvalitetu pa se on također koristi programima za stolno izdavaštvo.

Funkcija `image()` prikazuje matricu kao sliku. Vrijednosti iz matrice koristi kao indekse koji pokazuju na vrijednost boje iz mape boja. Ako je `image()` zadana s jednim argumentom, matricom, onda se svaki element matrice prikazuje kao linearni pravokutnik (patch). Za više argumenata ista funkcija ima drugačije prezentacije. Moguće je i graf funkcije predstaviti kao sliku. To se postiže funkcijom `imagesc()`, koja rang vrijednosti matrice shvaća kao mapu boja te ih tako i prikazuje.

Primjer 5.28: *Prikazivanje matrice kao slike*

Funkcijama `image()` i `imagesc()` možemo prikazati matricu kao sliku. Argument funkcije `image()` je matrica indeksa čiji elementi pokazuju na neku boju iz mape boja, dok `imagesc()` prikazuje svaki element iz matrice kao vrijednosti mape boja.

```
>> Z=peaks(50);
>> image(Z);
>> figure(2)
>> imagesc(Z);
>> colorbar;
```



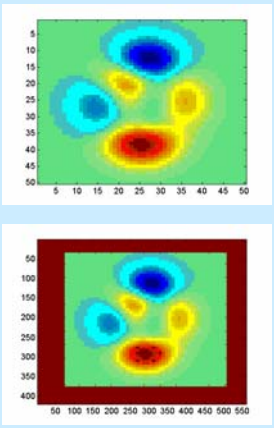
Matrica se može spremiti u vanjsku datoteku s pomoću funkcije `imwrite()`. Dakako, moguć je i obrnut postupak, čitanje matrice slike iz vanjske datoteke, koristeći funkciju `imread()`.

Primjer 5.29: *Spremanje i učitavanje slike*

Spremanje slike trenutačnog grafičkog prozora (gcf) u datoteku 'slika.tif'. Pritom će se koristiti indeksirani zapis (slika i mapa boja). Nakon toga ćemo učitati tu

sliku u Matlab i prikazati je pomoću funkcije `image()`.

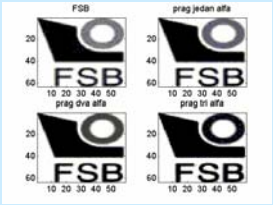
```
>> Z=peaks(50);
>> imagesc(Z);

>> [X,map]=capture(gcf);
>> imwrite(X,map,'slika.tif')

>> W=imread('slika.tif');
>> figure(2)
>> image(W)
```



Primjer 5.30: *Obrada slike mijenjajući okidnu razinu*

Bez ulaženja u detalje, pogledajmo jednu obradu slike, u kojoj se programski mijenja okidna razina (engl. treshold).

```
>> clear;
% Razina praga - parametar alfa:
>> alfa=0.1;% manji od 1/3

>> [x]=imread('fsb.jpg');
>> ix=rgb2gray(x);
>> ix=double(ix);
>> I_max=max(max(ix));
>> I_min=min(min(ix));
>> level1=alfa*(I_max-I_min)+I_min;
>> level2=2*level1;
>> level3=3*level1;
>> thix1=max(ix,level1.*ones(size(ix)));
>> thix2=max(ix,level2.*ones(size(ix)));
>> thix3=max(ix,level3.*ones(size(ix)));
>> figure(1);colormap(gray);
>> subplot(2,2,1);imagesc(ix)
>> title('FSB');
>> subplot(2,2,2);imagesc(thix1)
>> title('prag jedan alfa');
```



```
>> subplot(2,2,3);imagesc(thix2)
>> title('prag dva alfa');
>> subplot(2,2,4);imagesc(thix3)
>> title('prag tri alfa');
```

Na sličan način nad slikom se mogu provoditi različiti algoritmi obrade.

MATRICE, VEKTORI I POLINOMI 6

Premda pod pojmom matrica najčešće podrazumijevamo dvodimenzionalne matrice, Matlab podržava i višedimenzionalne matrice s kojima najčešće radi preko ćelija. U dvodimenzionalnoj matrici razlikujemo stupce i retke, kao jednodimenzionalne veličine koje zovemo vektorima. Svaki polinom prikazuje se kao vektor čiji elementi su koeficijenti.

6.1 Matrice

Matlab ima tri velike skupine funkcija koje se bave samo matricama. Te skupine su:

- **matlab\elmat** - Elementarne matrice i obrada matrica
- **matlab\matfun** - Matrične funkcije iz numeričke linearne algebre
- **matlab\sparfun** - Slabo popunjene (slabo popunjene) matrice

U ovom potpoglavlju dotaknut ćemo se samo najvažnijih funkcija, a posebna pozornost posvetit će se slabo popunjenim i višedimenzionalnim matricama.

6.1.1 Elementarne matrične funkcije

S pomoću `help` naredbe može se dobiti pomoć o svakoj od ove tri skupine, na primjer:

```
>> help matlab\elmat
```

navest će sve elementarne matrične funkcije:

Tablica 6.1: *Elementarne matrice*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`zeros()`	Polje ništica (nula)	`>> zeros(2)`	ans = 0 0 0 0
`ones()`	Polje jedinica	`>> ones(2)`	ans = 1 1 1 1

`eye()`	Matrica identiteta	>> `eye`(2)	ans = 1 0 0 1
`rand()`	Uniformno raspodijeljeni slučajni brojevi	>> `rand`(2,3)	ans = 0.4565 0.8214 0.6154 0.0185 0.4447 0.7919
`randn()`	Slučajni brojevi raspodijeljeni po normalnoj (Gaussovoj) razdiobi	>> `randn`(2,3)	ans = -0.4326 0.1253 -1.1465 -1.6656 0.2877 1.1909
`repmat()`	Kopira matrice. Koristi se i za ispunjenje matrice konst. vrijednostima	>> `repmat`(7,3)	ans = 7 7 7 7 7 7 7 7 7

Tablica 6.2: *Osnovne informacije o poljima*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`size()`	Veličina polja	>> A=[1 2;3 4;5 6]; >> `size`(A)	---------- ans = 3 2
`length()`	Dužina vektora	>> A=[1 1 5 5 8]; >> `length`(A)	---------- ans = 5
`ndims()`	Broj dimenzija	>> A=[1 1;2 2;3 3]; %2D >> A(:,:,2)=[7 7;8 8;9 9]; %3D >> `ndims`(A)	---------- ---------- ans = 3
`numel()`	Broj elemenata	>> A=[1 1;2 2;3 3]; %2D >> A(:,:,2)=[7 7;8 8;9 9]; %3D >> `numel`(A)	---------- ---------- ans = 12
`isempty()`	Istina, ako je polje prazno	>> A=[1 2 3]; B=[]; >> `isempty`(A) >>`isempty`(B)	---------- ans = 0 ans = 1
`isequal()`	Istina, ako su polja numerički jednaka	>> A=[1 2]; B=[1 2]; C=[1 3]; >> `isequal`(A,B,C) >>`isequal`(A,B)	---------- ans = 0 ans = 1

Tablica 6.3: *Matrična obrada*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`cat()`	Povezivanje polja	>> A=[1 2]; B=[1 3]; >> C1= `cat`(1,A,B) %1D >> C2= `cat`(2,A,B) %2D >> C3= `cat`(3,A,B) %3D	C1 = 1 2 1 3 C2 = 1 2 1 3 C3(:,:,1) = 1 2 C3(:,:,2) = 1 3

`reshape()`	Promjena veličine	>> A=[1 1 5 5 4 4] % naredba reshape popunjeva %R1 po stupcima uzimajuci %elemente iz A po redcima >> R1=`reshape`(A,2,3) >> R2=`reshape`(A,3,2)	A = 1 1 5 5 4 4 R1 = 1 5 4 1 5 4 R2 = 1 5 1 4 5 4
`diag()`	Dijagonalna matrica i dijagonalizacija matrice	% matrica 3x3 >> A=[5 7 9;4 5 2;1 2 5] % glavna dijagonala matrice A >> D1=`diag`(A) % prva dijagonala ispod glavne >> D2=`diag`(A,-1) % prva dijagonala iznad glavne >> D3=`diag`(A,1)	A = 5 7 9 4 5 2 1 3 5 D1 = 5 5 5 D2 = 4 3 D3 = 7 2
`end()`	Posljednji indeks polja	>> A=[5 7 9;4 5 2;1 14 17] >> B=A(end,2:end)	A = 5 7 9 4 5 2 1 14 17 B = 14 17
`squeeze()`	Vektore iz više dimenzija spaja u matricu s manje dimenzija	>> A(:,:,1)=[1 2 3]; %2D >> A(:,:,2)=[5 6 7]; %3D >> S = `squeeze`(A) %2D	S = 1 5 2 6 3 7

Tablica 6.4: *Specijalizirane matrice*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`gallery()`	Higham-ove test matrice. Sadrži više od 50 test matrica korisnih za testiranje algoritama	>> v = [1 2 3] % circul -> test matrica >> C = `gallery`('circul',v)	v = 1 2 3 C = 1 2 3 3 1 2 2 3 1
`hadamard()`	Hadamard-ova matrica (H).	>> H = `hadamard`(2) >> A = H'*H	H = 1 1 1 -1 A = 2 0 0 2
`hankel()`	Hankel-ova matrica. Nule ispod antidijagonale	>> A = [1 2 3 4 5] >> H = `hankel`(A)	H = 1 2 3 4 2 3 4 0 3 4 0 0 4 0 0 0

`hilb()`	Hilbert-ova matrica. Njeni elementi su H(i,j)=1/(i+j-1), i-redak, j-stupac	>> H = `hilb`(4)	H = 1 .50 .33 .25 .5 .33 .25 .20 .3 .25 .20 .16 .2 .20 .16 .14
`magic()`	Magični kvadrat. Suma elemenata bilo kojeg retka ili stupca je ista.	>> M = `magic`(3)	M = 8 1 6 3 5 7 4 9 2
`pascal()`	Pascal matrica	>> P=`pascal`(4)	P = 1 1 1 1 1 2 3 4 1 3 6 10 1 4 10 20
`rosser()`	Klasični test problem za simetrične svojstvene vrijednosti.	% test matrica (8x8) koja se %koristi za algoritme koji %se bave svojstvenim %vrijednostima >> R=`rosser`	% 8 x 8 matrica
`toeplitz()`	Toeplitz-ova matrica. Elementi su isti na pozitivnoj dijagonali	>> C = [1 2 3 4]; >> R = [1 7 8 9]; >> T = `toeplitz`(C,R)	T = 1 7 8 9 2 1 7 8 3 2 1 7 4 3 2 1
`wilkinson()`	Wilkinsonova test matrica za svojstvene vrijednosti	>> W = `wilkinson`(3)	ans = 1 1 0 1 0 1 0 1 1

6.1.2 Matrična analiza

Linearna algebra podržana je raznolikim matričnim funkcijama o kojima preko pomoći (help) čitamo ovo:

>> `help` matlab\matfun

Tablica 6.5: *Matrična analiza*

Funkcija	Opis funkcije	Primjer	Rezultat primjera

`norm()`	Matrična ili vektorska norma	>> A=[1 2] %vektorska 2-norma, %isto što i sqrt(1^2+2^2) >> N = `norm`(A)	A= 1 2 N= 2.2361
`normest()`	Procjena matrične 2-norme. Inače se koristi kod velikih matrica.	>> A=[1 2] %vektorska 2-norma >> N = `normest`(A)	A= 1 2 N= 2.2361
`rank()`	Rang matrice. Broj linearno nezavisnih stupaca ili redaka matrice	>> A=[4 4 4;0 8 9;0 8 9] >> R=`rank`(A)	A = 4 4 4 0 8 9 0 8 9 R = 2
`det()`	Determinanta	>> A=[1 2 3;4 5 6;7 8 9] >> D = `det`(A)	A = 1 2 3 4 5 6 7 8 9 D = 0
`trace()`	Trag matrice. Zbroj dijagonalnih elemenata matrice	>> A=[1 2 3;4 5 6;7 8 9]; >> T=`trace`(A)	A = 1 2 3 4 5 6 7 8 9 T = 15
`null()`	Daje ortonormirana bazu Z za nul- prostor matrice A. A*Z=nul matrica Z'*Z=I, I-jedinična matrica	>> A=[1 1 1;1 2 3] >> Z=`null`(A) >> N=A*Z %nul-prostor	A = 1 1 1 1 2 3 Z = 0.4082 -0.8165 0.4082 N = 0 0
`orth()`	Ortonormirana baza za rang matrice A. Svojstvo ortonorm.. matr. je da umnožak nje i njene transponirane daje jediničnu matricu B'*B=`eye`(`rank`(A))	>> A=[1 5 5;0 8 9] %ortonorm. matrica B >> B=`orth`(A) %umnozak -> jedin. mat. >> B'*B %jedinicna matr. ranga A >> `eye`(`rank`(A))	A = 1 5 5 0 8 9 B = -0.508 -0.861 -0.861 0.508 ans = 1 0 0 1 ans = 1 0 0 1

`rref()`	Reducirana redčana forma (row echelon form). Primjena Gauss-Jordanove eliminacije s parcijalnim pivotiranjem	>> A = `magic`(4) >> R = `rref`(A)	A = 16 2 3 13 5 11 10 8 9 7 6 12 4 14 15 1 R = 1 0 0 1 0 1 0 3 0 0 1 -3 0 0 0 0
`subspace()`	Kut između dva podprostora. Matrice koje čine ravninu ne moraju biti istih dimenzija da bi se odredio kut između njih.	>> H=`hadamard`(4) %hadamard-ova matrica %ima ortogonalne stupce >> A=H(:,1)' >> B=H(:,2:4)' %kut između dva %okomita podprostora A %i B je 90° >> fi=`subspace`(A',B')	H = 1 1 1 1 1 -1 1 -1 1 1 -1 -1 1 -1 -1 1 A = 1 1 1 1 B = 1 -1 1 -1 1 1 -1 -1 1 -1 -1 1 fi = 1.5708

Tablica 6.6: *Svojstvene i singularne vrijednosti*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`eig()`	Svojstvene vrijednosti i svojstveni vektori. Netrivijalna rješenja -sv.vr.: det(A-L*I)=0. Sustav: A*X=L*X , tj. (A-L*I)*X=0	>> A=[4 2;2 4]; %X - svojstveni vektori %L - svojstvene vrijednosti >> [X L]=`eig`(A)	X = -0.7071 0.7071 0.7071 0.7071 L = 2 0 0 6
`svd()`	Dekompozicija singularnih vrijednosti (kvadratni korijeni matrice A*A') matrice A. Ako u matrici S postoji jedan redak s elementima 0, onda je A singularna matr.	>> A=[4 2;2 4]; %U-unitarna($U^*=U^{-1}$,gdje je % U^* kompleksno %transponirana matrica, a %U^{-1} inverzna matrica. %S-matrica singularnih vr. %V-unitarna matrica >> [U,S,V]=`svd`(A)	U =-0.707 -0.707 -0.707 0.707 S = 6 0 0 2 V = -0.707 -0.707 -0.707 0.707

`poly()`	Karakteristični polinom matrice A. Korijeni tog polinoma su svojstvene vr. matrice A.	>> A=[4 2;2 4] %B-koeficijenti karakterist. %polinoma matrice A >> B=`poly`(A) %C-korijeni kar.polinoma >> C=`roots`(B)	A = 4 2 2 4 B = 1 -8 12 C = 6 2
`qz()`	QZ faktorizacija za generalizirane svojstvene vrijednosti. Netrivijalna rješenja -sv.vr.:det(A-L*B)=0. Sustav: A*X=L*B*X , tj. (A-L*B)*X=0	>> A=[4 2;2 4] >> B=[2 1;4 1] % AA=Q*A*Z % BB=Q*B*Z >> [AA,BB,Q,Z]=`qz`(A,B)	AA = -2.68 3.57 0 4.47 BB = 0.89 4.02 0 2.23 Q = -0.00 -1.00 -1.00 0.00 Z = -0.44 -0.89 0.89 -0.44

Tablica 6.7: *Matrične funkcije*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`expm()`	Matrična eksponencijala	>> A=[1 2;3 4]; %matrična eksponenc. >> B=`expm`(A) %obična eksponenc. >> C=`exp`(A) %dijagonalna matr. D >> D=[1 0;0 2]; >> E=`expm`(D)	B = 51.9690 74.7366 112.1048 164.0738 C = 2.7183 7.3891 20.0855 54.5982 E = 2.7183 0 0 7.3891
`logm()`	Matrični logaritam. A=`logm`(`expm`(A))	>> A=[1 2;3 4]; >> B=`expm`(A); >> C=`logm`(B)	C = 1 2 3 4
`sqrtm`	Matrični kvadratni korijen. A=C=X*X	>> A=[1 2;3 4]; >> X=`sqrtm`(A) >> C=X*X	X = 0.55+0.46i 0.80-0.21i 1.21-0.31i 1.76+0.14i C = 1 2 3 4
`funm()`	Izračunavanje generalizirane matrične funkcije. Za proizvoljnu funkciju računa matričnu verziju.	>> A=[1 2;3 4]; %sinusna funkcija >> S=`funm`(A,@`sin`) %exponencijalna funkc >> E=`funm`(A,@`exp`) >> B=`expm`(A)	S = -0.4656 -0.1484 -0.2226 -0.6882 E = 51.9690 74.7366 112.1048 164.0738 B = 51.9690 74.7366 112.1048 164.0738

Napomena: Ako imamo funkciju e^x, pri čemu je x-skalar, onda tu funkciju možemo prikazati kao red (koristeći Taylor-ov razvoj u red) koji će za beskonačan broj elemenata težiti upravo vrijednosti te funkcije: $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} = \sum_{n=0}^{\infty} \frac{x^n}{n!}$. U slučaju da se umjesto skalara x nalazi kvadratna matrica A onda bi se na isti način moglo napisati : $e^A = I + \frac{A}{1!} + \frac{A^2}{2!} + \frac{A^3}{3!} + ... + \frac{A^n}{n!} = \sum_{n=0}^{\infty} \frac{A^n}{n!}$. Postoji razlika između obične eksponencijale `exp`(**A**) i matrične eksponencijale `expm`(**A**) u tome što se obična eksponencijala primjenjuje na svaki element matrice **A**, a matrična na matricu **A** u cjelini. Za dijagonalnu matricu **A**, obična i matrična eksponencijala su jednake (npr. $A = \begin{pmatrix} a11 & 0 \\ 0 & a22 \end{pmatrix}$, $exp(A) = \begin{pmatrix} e^{a11} & 0 \\ 0 & e^{a22} \end{pmatrix} = expm(A) = \begin{pmatrix} e^{a11} & 0 \\ 0 & e^{a22} \end{pmatrix}$).

Tablica 6.8: *Faktorizacijske pomoćne funkcije*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`qrdelete()`	Vraća QR (Q1,R1) faktorizaciju matrice A1, gdje je A1 ustvari matrica A sa uklonjenim *j*-im redcima ili stupcima	>> A=[8 1 6;3 5 7;4 9 2]; >> [Q,R] = `qr`(A); >> j = 2; >> [Q1,R1] = `qrdelete`(Q,R,j,'row') %usporedba A i A1 >> A1 = Q1*R1	Q1 = 0.8944 0.4472 0.4472 -0.8944 R1 = 8.94 4.91 6.26 0 -7.60 0.89 A1 = 8 1 6 4 9 2
`qrinsert()`	Vraća QR (Q1,R1) faktorizaciju matrice A1, gdje je A1 ustvari matrica A sa umetnutim vektorom x ispred *j*-og retka (ili stupca)	>> A = [1 3;4 2]; >> [Q,R] = `qr`(A); >> j = 2; >> x = 1:2; >> [Q1,R1] = `qrinsert`(Q,R,j,x,'row') %usporedba A i A1 >> A1 = Q1*R1	Q1 = 0.23 0.82 -0.51 0.23 0.46 0.85 0.94 -0.32 -0.08 R1 = 4.24 3.06 0 2.75 0 0 A1 =1 3 1 2 4 2

`balance()`	Poboljšanje točnosti svojstvenih vrijednosti kod loše uvjetovanih matrica (matrica A) dijagonalnim skaliranjem (matrica T) tako što će se tražiti takva matrica B koja će imati puno bolju uvjetovanost nego A, a iste svojstvene vrijednosti. B=T\A*T. Ako je A simetrična onda je A=B, a T je jedinična matrica (nema skaliranja).	%lose uvjetovana matrica >> A=[1 1000;0.01 1] %E-svojstveni vektori %L-svojstvene vrijednosti >> [E,L]=`eig`(A) %T-matrica za skaliranje %B-matrica A s %poboljsanom %uvjetovanoscu >> [T,B] = `balance`(A) %E2-svojstveni vektori %L2-svojst. vrijednosti >> [E2,L2] = `eig`(B) %losa uvjetovanost jer %CA>>1 >> CA = `cond`(A) %dobra uvjetovanost jer %CA je oko 1 >> CB = `cond`(B)	A = 1 1000 0.01 1 E = 1.0000 -1.0000 0.0032 0.0032 L = 4.1623 0 0 -2.1623 T = 256 0 0 1 B = 1 3.9063 2.56 1 E2 = 0.7772 -0.7772 0.6292 0.6292 L2 = 4.1623 0 0 -2.1623 CA = 1.1e+005 CB = 2.19
`planerot()`	Givenove rotacije ravnine, pri čemu se od dva elementa vektora stupca želi dobiti vektor istih dimenzija samo s drugim elementom jednakim nuli.	>> x = [3 4]; %G-ortogonalna %(det(G)=1) matr rotacije >> [G,y] =`planerot`(x') %y=G*x' , y(2)=0 >> y1 = G*x'	x = 3 4 G = 0.6 0.8 -0.8 0.6 y = 5 0 y1 = 5 0
`cholupdate`	Rang 1 poboljšanje (smanjuje broj operacija za 1 red veličina)u Cholesky faktorizaciji. Matrica mora biti ispunjena (a ne sparse matrica) , te pozitivno definitna. Isto što i Cholesky faktorizacija na matrici A+x*x', gdje je x vektor.	%pozitivno definitna mat. >> A = `pascal`(3) >> x = [0 0 1]'; >> R = `chol`(A) >> R1 = `cholupdate`(R,x) %isto sto i : %R1 = chol(A + x*x');	A = 1 1 1 1 2 3 1 3 6 R = 1 1 1 0 1 2 0 0 1 R1 = 1 1 1 0 1 2 0 0 1.414

`qrupdate()`	Rang 1 poboljšanje (smanjuje broj operacija za 1 red veličina) u QR faktorizaciji. Isto što i QR fakt. na matrici A+u*v', gdje su u i v vektori odgovarajuće veličine.	>> A = `pascal`(3); >> u = [-1 0 0]'; >> v = `ones`(3,1); >> [QT,RT] = `qr`(A + u*v') % isto sto i: [Q1,R1] = qrupdate(Q,R,u,v)	QT =0 0 -1 -0.7 -0.7 0 -0.7 0.7 0 RT = -1.4 -3.5 -6.3 0 0.7 2.1 0 0 0

Napomena : pozitivno definitna matrica **A** čija su svojstva: da su joj svi elementi na dijagonali pozitivni, da joj sve podmatrice koje se prostiru od lijevog gornjeg ugla pa do desnog donjeg ugla imaju determinante veće od nule, te da vrijedi svojstvo **x'*A*x**>0 za realne matrice **A**, dok se za imaginarne matrice **A**, umjesto transponiranja matrice **x**, umeće u isti izraz $\mathbf{x}^*$ (kompleksno konjugirana i transponirana matrica **x**).

Osim nabrojenih funkcija u istoj skupini postoje i funkcije iz područja linearnih jednadžbi, iterativnih metoda, permutacijskih algoritama i operacija na grafovima (stablima).

6.1.3 Determinanta

U nekim primjenama linearne algebre potrebno je izračunavanje determinanti. Matlab ima ugrađenu funkciju `det()` koja izračunava determinante.

Neka je

```
>> A = magic(3);
```

Determinanta od **A** jednaka je:

```
>> det(A)
   ans =
        -360
```

Primjer 6.1: *Računanje determinante preko kofaktorskog razvoja*

*Jedna od klasičnih metoda računanja determinanti je ona preko kofaktorskog razvoja i u ovom primjeru će biti prikazana korisnička funkcija za potreban izračun. Funkcija ckj = cofact(A, k, j) računa kofaktor 'ckj' od matrice **B** koja je ustvari matrica **A** samo umanjena za k-ti redak i j-ti stupac. Funkcija d = mydet(A) koristi metodu kofaktorskog razvoja za računanje determinante.*

```
function ckj = cofact(A,k,j)
[m,n] = size(A); %dimenzija matrice A (mxn)
if m ~= n    %ako nije kvadratna
```

```
A =
0.4416  0.1760  0.5136
0.4831  0.0020  0.2132
```

```
  error('Matrica mora biti kvadratna')
end
%B -> brisanje k-tog retka i j-tog stupca matrice A
B = A([1:k-1,k+1:n],[1:j-1,j+1:n]);
ckj = (-1)^(k+j)*det(B);  %kofaktor na mjestu (k,j)

function d = mydet(A)
% Determinanta d od matrice A. Funkcija cofact()
%mora biti u MATLAB-ovom putu traženja
[m,n] = size(A);
if m ~= n
  error('Matrica mora biti kvadratna')
end
a = A(1,:)  %prvi redak matrice A
c = [];   %definiranje c kao matricne forme
for j=1:n     %petlja od prvog do zadnjeg stupca
 c1j = cofact(A,1,j);  %prvi vektor redak kofaktora
 c = [c;c1j]  %vektor stupac kofaktora
end
d = a*c;   %determinanta matrice A

%Testiranje funkcija
>> A=rand(3)
>> de=det(A)
>> De=mydet(A)
```

```
0.6081  0.7902  0.1034

de =  0.1351

a =
0.4416  0.1760  0.5136

c = -0.1683
     0.0797
     0.3805

De =  0.1351
```

Primijetite da funkcija `mydet()` koristi kofaktorski razvoj uz redak **1** od matrice **A**. Metoda kofaktora ima visoku računalnu kompleksnost, pa se ne preporučuje za velike matrice. Ovdje se prikazuje isključivo zbog pedagoških razloga.

6.1.4 Adjungirana matrica

Adjungirana matrica `adj`(**A**) matrice **A** je također zanimljiva u linearnoj algebri. Adjungirana matrica i inverzna matrica zadovoljavaju jednadžbu:
$\mathbf{A}^{-1}$=`adj`(**A**)/`det`(**A**)
Zbog visoke računalne kompleksnosti ova formula se ne preporučuje za računanje inverzne matrice.

Primjer 6.2: *Računanje adjungirane matrice i inverza matrice*

Adjungirana matrica i inverzna matrica zadovoljavaju jednadžbu:
$\mathbf{A}^{-1}$*=adj(**A**)/det(**A**). Korisnička funkcija adj() izračunava adjungiranu matricu od*

*matrice **A**, a pritom se koristi korisničkom funkcijom cofact() definirane u Primjeru 5 kojom se izračunavaju kofaktori matrice **A**.*

```
function B = adj(A)
% Adjungirana matrica B od kvadratne matrice A.
[m,n] = size(A);
if m ~= n
  error('Matrica mora biti kvadratna')
end
B = [];
for k = 1:n
 for j=1:n
%B ->vektor svih kofaktora matrice A
   B = [B;cofact(A,k,j)]
 end
end
%vektor B ->kvadratnu matricu B
B = reshape(B,n,n)

%Testiranje funkcije
>> A=[1 2 3;4 5 6; 7 7 8]
>> Ad=adj(A)  %adjungirana matrica A
>> Ainv=adj(A)/det(A)  %inverzna matrica A
>> A=inv(Ainv)  %matrica A
```

```
A =  1    2    3
     4    5    6
     7    7    8
B = -2
     10
    -7
     5
    -13
     7
    -3
     6
    -3
B = -2    5   -3
     10  -13   6
    -7    7   -3
Ad = -2    5   -3
      10  -13   6
     -7    7   -3
Ainv =
0.6667  -1.6667
1.0000
-3.3333  4.3333  -2.000
2.3333  -2.3333
1.0000
A = 1    2    3
    4    5    6
    7    7    8
```

6.2 Slabo popunjene (sparse) matrice

Slabo popunjene (sparse) matrice su takve matrice koje sadrže uglavnom nule, tj. obično samo 5% ili manje ne-ništićnih elemenata. Za matrice reda (1000 x 1000) cjelobrojnih ne-ništićnih elemenata (svaki po 8 okteta (64 bitni broj-*double* tipa je 8 bajta) zauzeća memorije) po glavnoj i dvjema usporednim dijagonalama trebalo bi 8 MB memorije za potpuno spremanje. Međutim, za spremanje sparse matrice bit će potrebno samo oko 40KB, dakle 200 puta manje prostora. Kod tog će i operacije obradbe također biti brže.

Matlab ima velik broj funkcija za obradbu sparse matrica:

```
>> help matlab\sparfun
```

Tablica 6.9: *Elementarne slabo popunjene matrice*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`speye()`	Slabo popunjena jedinična matrica. Zauzima oko 16kB memorijskog prostora.	>> S=`speye`(3,3)	S = (1,1) 1 (2,2) 1 (3,3) 1
`sprand()`	Slabo popunjena jednoliko distribuirana slučajna matrica. Ima istu strukturu kao matrica **S**, samo s uniformnom razdiobom slučajno odabranih vrijednosti.	>> S=[0 0 0;0 5 6; 7 0 8]; >> B=`sprand`(S)	B = (3,1) 0.6038 (2,2) 0.2722 (2,3) 0.1988 (3,3) 0.0153
`sprandn()`	Slabo popunjene normalno distribuirana slučajna matrica. Ima istu strukturu kao matrica **S**, samo s normalnom razdiobom slučajno odabranih vrijednosti .	>> S=[0 0 0;0 5 6; 7 0 8]; >> B=`sprandn`(S)	B = (3,1) 0.3481 (2,2) -0.4937 (2,3) 0.4728 (3,3) 1.9141
`sprandsym()`	Slabo popunjena simetrična slučajna matrica. Njen donji trokut i dijagonala imaju istu strukturu kao matrica **S**, s normalnom razdiobom slučajno odabranih vrijednosti .	>> S=[0 0 0;0 5 6; 7 0 8]; >> B=`sprandsym`(S)	B = (3,1) 0.3711 (2,2) 0.4515 (1,3) 0.3711 (3,3) -0.3896

`spdiags()`	Stvara matricu **B** sa dijagonalnim pojasom ne-ništićnih elemenata, stvorene preko dijagonala matrice **S** Glavna dijagonala je ujedno i nulta. Ne-ništične dijagonale iz **S** postaju stupci matrice **B**, dok vektor **d** sadrži redoslijed ne-ništičnih dijagonala iz **S**.	>> S=[0 0 0;0 5 6; 7 0 8]; >> [B,d]=`spdiags`(S)	S = 0 0 0 0 5 6 7 0 8 B = 7 0 0 0 5 0 0 8 6 d =-2 0 1

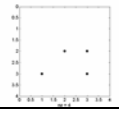
Tablica 6.10: *Pretvorba punih u slabo popunjene matrice*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`sparse()`	Tvorba slabo popunjene matrice. Popunjena matrica **A** se u Matlab-u prevodi u oblik manjeg memorijskog zauzeća (*sparse* matrica **S**)	>> A=[0 0 0;0 5 6; 7 0 8]; >> S=`sparse`(A)	S = (3,1) 7 (2,2) 5 (2,3) 6 (3,3) 8
`full()`	Pretvorba slabo popunjene matrice **S** u puni oblik **F**.	>> A=[0 0 0;0 5 6; 7 0 8]; >> S=`sparse`(A); >> F=`full`(S) %F=A	F = 0 0 0 0 5 6 7 0 8
`find()`	Traženje indeksa ne-ništićnih elemenata.	>> A=[0 0 0;0 5 6; 7 0 8]; >> [i,j] = `find`(S)	i = 3 2 2 3 j = 1 2 3 3

`spconvert()`	Uvlačenje iz slabo popunjenih matrica vanjskog formata.	>> C=`load`('spar.txt') %spar.txt je tekstualna %datoteka koja sadrži 3 %stupca po 4 elemenata >> S=`spconvert`(C)	C = 1 1 0 1 2 3 2 1 7 2 2 0 S = (2,1) 7 (1,2) 3

Tablica 6.11: *Rad sa slabo popunjenim matricama*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`nnz()`	Broj ne-ništićnih elemenata slabo popunjene matrice.	>> A=[0 0 0;0 5 6; 7 0 8]; >> Bn=nnz(A)	Bn = 4
`nonzeros()`	Ne-ništićni matrični elementi. Elementi se izdvajaju redom po stupcima.	>> A=[0 0 0;0 5 6; 7 0 8]; >> S=`nonzeros`(A)	S = 7 5 6 8
`nzmax()`	Broj mjesta prostora rezerviranog za ne-ništične matrične elemente.	>> A=[0 0 0;0 5 6; 7 0 8] >> M=`nzmax`(A) >> S=`sparse`(A) >> N=`nzmax`(S)	A =0 0 0 0 5 6 7 0 8 M = 9 S =(3,1) 7 (2,2) 5 (2,3) 6 (3,3) 8 N =4
`spones()`	Zamjenjuje ne-ništične elemente slabo popunjene matrice s jedinicama.	>> A=[0 0 0;0 5 6; 7 0 8]; >> S=`spones`(A)	S =(3,1) 1 (2,2) 1 (2,3) 1 (3,3) 1
`spalloc()`	Zauzima prostor za slabo popunjene matrica. Time se izbjegava neprekidno dodjeljivanje memorijskog prostora kada se u matici pojavi ne-ništićni element.	>> m=3,n=5,nzmax=15; >> S=`spalloc`(m, n,nzmax) >> SS=`full`(S)	S = All zero sparse: 3-by-5 SS = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

`issparse()`	True (istina) za slabo popunjene matrice.	>> A=[0 0 0;0 5 6; 7 0 8] >> `issparse`(A) >> S=`sparse`(A) >> `issparse`(S)	A = 0 0 0 0 5 6 7 0 8 ans = 0 S =(3,1) 7 (2,2) 5 (2,3) 6 (3,3) 8 ans = 1
`spfun()`	Primjenjuje funkciju na ne-ništične matrične elemente.	>> A=[0 0 0;0 5 6; 7 0 8]; >> S=`sparse`(A) >> F=`spfun`(@`sqrt`,A)	S =(3,1) 7 (2,2) 5 (2,3) 6 (3,3) 8 F = (3,1) 2.6458 (2,2) 2.2361 (2,3) 2.4495 (3,3) 2.8284
`spy()`	Vizualizira uzorak raspršenja slabo popunjenih matrica	>> S=[0 0 0;0 5 6; 7 0 8]; >> `spy`(S)	

Primjer 6.3: *Načini tvorbe sparse matrice*

Svaka matrica može se pretvoriti u sparse matricu, jer je takvim načinom pohrane podataka ušteđeno na memorijskom prostoru.

```
%Unesimo slabo popunjenu matricu na standardan
%način:
A=[ -2 1 0 0 0
     1 -2 1 0 0
     0 1 -2 1 0
     0 0 1 -2 1
     0 0 0 1 -2 ];

%a zatim je pretvorimo u sparse matricu preko
%sparse() funkcije:
>> S=sparse(A)

%sparse matricu mozemo dobiti i na nacin da se
%definiraju mjesta na kojima ce biti ne-nisticni
%element
>> i=[ 1 2  1 2 3   2 3 4  3 4 5  4 5];  %i-ti redak
```

```
S = (1,1)    -2
    (2,1)     1
    (1,2)     1
    (2,2)    -2
    (3,2)     1
    (2,3)     1
    (3,3)    -2
    (4,3)     1
    (3,4)     1
    (4,4)    -2
    (5,4)     1
    (4,5)     1
    (5,5)    -2
a = (1,1)    -2
    (2,1)     1
    (1,2)     1
```

```
>> j=[ 1 1  2 2 2   3 3 3  4 4 4  5 5];   %j-ti stupa
>> s=[-2 1  1 -2 1  1 -2 1  1 -2 1  1 -2]; %vrijednost
>> a=sparse(i,j,s,5,5)

%pretvorba iz sparse matrice u popunjenu matricu, pri
%cemu je matrica B jednaka matrici A
>> B=full(a)

%broj ne-nisticnih elemenata matrice B
>> NN=nnz(B)
```

```
    (2,2)      -2
    (3,2)       1
    (2,3)       1
    (3,3)      -2
    (4,3)       1
    (3,4)       1
    (4,4)      -2
    (5,4)       1
    (4,5)       1
    (5,5)      -2
B = -2   1   0   0   0
     1  -2   1   0   0
     0   1  -2   1   0
     0   0   1  -2   1
     0   0   0   1  -2
NN =  13
```

Primjer 6.4: *Tvorba sparse matrice*

*Načiniti slabo popunjenu matricu **S** dimenzija 7 x 5 koja ima 3 ne-ništična elementa $S_{1,2}$=20, $S_{3,3}$=41 i $S_{7,5}$=72.*

```
>> i = [1,3,7]; j = [2,3,5];
>> v = [20 41 72];
>> S = sparse(i,j,v)

%Punu matricu iz slabo popunjene (sparse tipa)
%dobivamo s pomoću funkcije full():
>> T = full(S)
```

```
S = (1,2)      20
    (3,3)      41
    (7,5)      72
T =
     0   20    0    0    0
     0    0    0    0    0
     0    0   41    0    0
     0    0    0    0    0
     0    0    0    0    0
     0    0    0    0    0
     0    0    0    0   72
```

Primjer 6.5: *Napisati Matlab kôd za stvaranje tridiagonalne matrice za bilo koju vrijednost n-a*

*Definiraju se tri stupčasta vektora, jedan za svaku 'dijagonalu', nakon čega se povezuju uporabom spdiags() funkcije. Vektori su označeni sa **l**, **d** i **u**. Oni moraju biti iste dužine, a u **l** vektoru koriste se samo prvih (n-1) članova, dok se u vektoru u koristi zadnjih (n-1) članova.*
Spdiags() postavlja ove vektore u dijagonale označene -1,0,1, gdje je 0 glavna dijagonala, a negativna je ona ispod nje.

```
>> n=3;
>> l=-(2:n+1)'; d=(1:n)'; u=((n+1):-1:2)';
>> A= [l d u]

%ako je stupac matrice A dulji od dijagonale koju
%zamjenjuje u B onda funkcija spdiags() uzima
%elemente za gornju dijagonalu matrice B iz donjeg
%dijela stupca (npr. stupca l), a za elemente za donju
%dijagonalu uzima iz gornjeg dijela stupca (npr.
%stupca u)
>> B=spdiags(A,-1:1,n,n)
>> full(B)


%Jos jedan primjer tvorbe trodijagonalne matrice
>> n=5; e=ones(n,1);
>> S=spdiags([-e 4*e -e],[-1 0 1],n,n);
>> C=full(S)
```

```
A = -2   1   6
    -3   2   5
    -4   3   4
    -5   4   3
    -6   5   2
B =  (1,1)      1
     (2,1)     -2
     (1,2)      5
     (2,2)      2
     (3,2)     -3
     (2,3)      4
     (3,3)      3
     (4,3)     -4
     (3,4)      3
     (4,4)      4
     (5,4)     -5
     (4,5)      2
     (5,5)      5
ans = 1   5   0   0   0
     -2   2   4   0   0
      0  -3   3   3   0
      0   0  -4   4   2
      0   0   0  -5   5
C =   4  -1   0   0   0
     -1   4  -1   0   0
      0  -1   4  -1   0
      0   0  -1   4  -1
      0   0   0  -1   4
```

6.3 Višedimenzionalne matrice preko polja ćelija

Polje ćelija je MATLABova matrica čiji elementi su ćelije, spremnici koji mogu sadržavati druga MATLAB polja (matrice ili vektore različitih tipova podataka). Na primjer, jedna ćelija može sadržavati uobičajenu matricu, druga može sadržavati tekstualni string (niz znakova), treća vektor kompleksnih vrijednosti i sl.

6.3.1 Stvaranje polja ćelija

Polje ćelija može se stvoriti dodavanjem podatka individualnim ćelijama, ćeliju po ćeliju. MATLAB automatski proširuje polje kako se pojedina ćelija dodaje. Postoje dva načina kako dodati podatke ćelijama:

a) Indeksiranjem ćelija

Kod tog načina, treba staviti indekse ćelija u zagradu koristeći standardno obilježavanje polja. Sadržaj ćelije stavlja se s desne strane pridružbe u vitičastu zagradu,"{ }".Na primjer, stvaranje nekog 2 x 2 polja ćelije A može izgledati ovako:

```
>> A(1,1) = {[1 4 3;0 5 8;7 2 9]};
>> A(1,2) = {'Anne Smith'};
>> A(2,1) = {3+7i};
>> A(2,2) = {-pi : pi / 10 : pi};
```

b) Indeksiranjem sadržaja

Kod ovog načina, treba indeks ćelije staviti u vitičastu zagradu koristeći standardno obilježavanje polja. Pridruženi sadržaj ćelije pridružuje se s desne strane jednakosti na standardan način.

```
>> A{1,1} = [1 4 3;0 5 8;7 2 9]
>> A{1,2} = 'Anne Smith';
>> A{2,1} = 3+7i;
>> A{2,2} = -pi : pi / 10 : pi;
```

Ova dva oblika mogu se naizmjence koristiti. Kod prikaza ćelija, MATLAB prikazuje redak ćelije u zgusnutom obliku:

```
>> A
A =
            [3x3 double]      'Anne Smith'
      [3.0000+7.0000i]     [1x21 double]
```

Za potpuni sadržaj ćelije koristi se funkciju `celldisp()`. Za grafički pak prikaz strukture ćelije koristi se funkcija `cellplot()` . Ako korisnik doda neki podatak polju ćelija na mjesto koja premašuje dimenzije dotičnog polja MATLAB će automatski proširiti polje uključivanjem novih indeksa. Ćelije koje pritom nemaju sadržaja bit će ispunjene praznim matricama. Na primjer, ako prethodno (2 x 2) polje ćelija pretvorimo u (3 x 3) polje:

```
>> A (3,3) = {5};
```

onda će novo polje ćelija izgledati ovako:

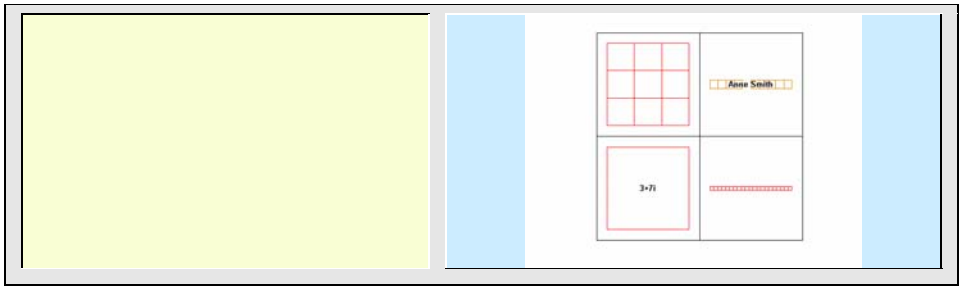
cell 1,1 1 4 3 0 5 8 7 2 9	cell 1,2 'Anne Smith'	cell 1,3 [ ]
cell 2,1 3+ 7i	cell 2,2 [-3.14...3.14]	cell 2,3 [ ]
cell 3,1 [ ]	cell 3,2 [ ]	cell 3,3 5

Primjer 6.6: *Prikaz sadržaja i strukture ćelije*

Prikaz potpunog sadržaja ćelije funkcijom celldisp(), te grafički prikaz strukture ćelije funkcijom cellplot().

```
>> A(1,1) = {[1 4 3;0 5 8;7 2 9]};
>> A(1,2) = {'Anne Smith'};
>> A(2,1) = {3+7i};
>> A(2,2) = {-pi : pi / 10 : pi};

>> A

>> celldisp(A)

>> cellplot(A)
```

```
A = [3x3 double]            'Anne Smith'
      [3.0000+ 7.0000i]    [1x21 double]

A{1,1} = 1     4     3
             0     5     8
             7     2     9
A{2,1} = 3.0000 + 7.0000i
A{1,2} = Anne Smith
A{2,2} =
Columns 1 through 7
-3.1416   -2.8274   -2.5133   -2.1991
-1.8850   -1.5708   -1.2566
Columns 8 through 14
-0.9425   -0.6283   -0.3142    0    0.3142
0.6283     0.9425
Columns 15 through 21
1.2566    1.5708    1.8850    2.1991  2.5133
2.8274    3.1416
```



6.3.2 Sintaksa polja ćelija

Vitičaste zagrade "{ }" su konstrukcije polja ćelija jednako kao što su uglaste zagrade konstrukcije numeričkog polja. Vitičaste zagrade ponašaju se slično kao i uglaste zagrade s tom razlikom da možete skladištiti vitičastu zagradu da bi odredili skladištenje ćelija. Vitičaste zagrade koriste zareze ili prazan prostor da bi odvojili stupce i točku-zarez da bi odvojili retke unutar polja ćelija. Na primjer:

```
>> C = {[1 2],[3 4];[5 6],[7 8]};
```

rezultira kao

cell 1,1 [1 2]	cell 1,2 [3 4]
cell 2,1 [5 6]	cell 2,2 [7 8]

Funkcija `cell()` dozvoljava nam da prethodno odredimo prazne redove ćelija određene veličine. Na primjer, donji izraz stvara prazno polje ćelija dimenzija 2 x 3 (dva retka i tri stupca).

```
>> B = cell(2,3);
```

Popunjavanje ćelije B na nekom mjestu ostvaruje se na već opisani način, npr.

```
>> B (1,3) = {1:3};
```

6.3.3 Pristupanje ćeliji koristeći indeksiranje sadržaja

Željenim podacima u ćeliji pristupa se indeksiranjem sadržaja na desnoj strani izraza. Najčešće se s lijeve strane odredi varijabla u koju se taj sadržaj želi spremiti. Indeks ćelije pritom se s desne strane izraza stavlja u vitičaste zagrade.
Na primjer, neka je polje ćelija N dimenzija 2 x 2 zadano sa:

```
>> N{1,1} = [1 2;4 5];
>> N{1,2} = 'Neko ime';
>> N{2,1} = 2-4i;
>> N{2,2} = 7;
```

Sadržaj ćelije N{1,2} možemo spremiti u neku varijablu 'c':

```
>> c = N{1,2}
c =
Neko ime
```

Želi li se dobiti neki podskup sadržaja ćelije onda se dodaje izraz za indeksiranje polja. Na primjer, da bi dobili element (2,2) matrice spremljene u prvoj ćeliji koristi se:

```
>> d = N{1,1} (2,2)
d =
     5
```

6.3.4 Pristupanje podskupu ćelija koristeći indeksiranje ćelija

Moguće je stvoriti i polje ćelija izvlačenjem ćelija iz nekog drugog pola ćelija. Situacija je analogna stvaranju standardnih podmatrica, samo su u ovom slučaju ćelije sadržaji elemenata matrice. Na primjer matrica **A**:

cell 1,1 3	cell 1,2 5	cell 1,3 9
cell 2,1 5	cell 2,2 6	cell 2,3 0
cell 3,1 4	cell 3,2 7	cell 3,3 2

cell 1,1 6	cell 1,2 0

B = A(2:3,2:3)		
	cell 2,1 7	cell 2,2 2

6.3.5 Brisanje ćelija

Brisati se mogu samo cijele dimenzije ćelija unutar nekog polja. Kao i kod brisanja standardnih polja i ovdje se indeksu ćelije dodaje prazna matrica. Primijetite da se u izrazu ne pojavljuju vitičaste zagrade, nego je indeksiranje klasično.

```
>> A(indeks_ćelije) = [ ]
```

6.3.6 Preoblikovanje polja ćelije

Poput standardnih polja i polje ćelija se može preoblikovati koristeći funkciju `reshape()`. Broj ćelija mora ostati jednak nakon preoblikovanja, što znači da se `reshape()` ne može koristiti da bi se dodale ili uklonile ćelije.

Primjer 6.7: *Preoblikovanje polja ćelije*

Funkcijom cell() stvorit ćemo prazno polje ćelija, dok ćemo njegovo preoblikovanje učiniti funkcijom reshape().

```
>> A = cell(3,4)
>> SA = size(A)
>> B = reshape(A,6,2)
>> SB = size(B)ž

%popunjavanje polja ćelija u A
>> A{3,1}='Jozo'
```

```
A =  [ ]   [ ]   [ ]   [ ]
     [ ]   [ ]   [ ]   [ ]
     [ ]   [ ]   [ ]   [ ]
SA =  3    4
B =  [ ]   [ ]
     [ ]   [ ]
     [ ]   [ ]
     [ ]   [ ]
     [ ]   [ ]
     [ ]   [ ]
SB =  6    2
A =  [ ]   [ ]   [ ]   [ ]
     [ ]   [ ]   [ ]   [ ]
   'Jozo'  [ ]   [ ]   [ ]
```

6.3.7 Primjena funkcija i operatora na poljima ćelija

Ćelije koriste analogna pravila nabrajanja i djelovanja operatora i funkcija kao standardna polja. Tako će izraz T{1:5} odgovarati listi vektora odvojenih zarezom u prvih pet ćelija iz T. Isto tako se indeksiranjem sadržaja ćelije možemo na nju primijeniti funkcije i operatore.

Primjer 6.8: *Primjena funkcija i operatora na poljima ćelija*

Rad na poljima ćelija je se izvodi na analogan način kao i kod rada sa poljima matrica.

```
>> A{1,1} = [1 2 ; 3 4];
>> A{1,2} = randn (3,3);
>> A{1,3} = 1:5;
>> B = sum (A{1,1})
>> for i = 1: length(A)
        M{i} = sum(A{1,i});
   end
>> M
>> celldisp(M)
>> M{1:2}
```

```
B =   4     6

M = [1x2 double]    [1x3 double]    [15]

M{1} =   4     6
M{2} =  -0.6336    1.0017    1.4202
M{3} =   15

ans =   4     6
ans =  -0.6336    1.0017    1.4202
```

6.3.8 Gniježđenje polja ćelija

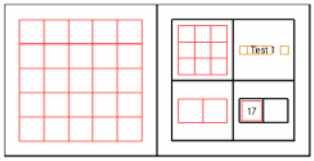
Polja ćelija korisna su za organiziranje podataka koji se sastoje od različitih veličina ili vrsta podataka, kad se želi pristupati mnogostrukim poljima ili podskupovima podataka. Ćelija može sadržavati također polje ćelija ili čak polje polja ćelija. Pritom se za gniježđenje ćelija mogu koristiti vitičaste zagrade, funkcija `cell()` ili direktni izrazi za stvaranje ugniježđenih ćelija.

Primjer 6.9: *Gniježđenje polja ćelija*

Primijetite da je desna strana zadatka stavljena u dva skupa vitičastih zagrada. Prvi skup predstavlja ćeliju (1,2) reda ćelije A. Drugi skup sprema ("pakira") 2 x 2 polje ćelije unutar vanjske ćelije.

```
%brisanje varijable A iz radne memorije
>> clear A

>> A(1,1) = {magic(5)};
>> A(1,2) = {{[5 2 8;7 3 0;6 7 3] 'Test
```

```
1';[2- 4i 5+7i] {17 [ ]}};

%graficki prikaz ugnijezdenih celija
>> cellplot(A)

>> A
```

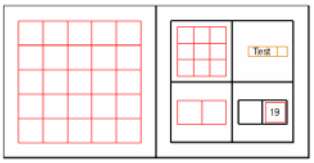



```
A =   [5x5 double]    {2x2 cell}
```

Primjer 6.10: *Način gniježđenja polja ćelija*

Da bi ugnijezdili polja ćelija pomoću funkcije cell() treba izlaz iz cell() funkcije dodati već postojećoj ćeliji. Vodite računa o upotrebi vitičastih zagrada sve do zadnje razine ugniježđenog indeksa.

```
% Stvorite prazno polje velicine 1 x 2
>> A = cell(1,2);
% Stvorite 2 x 2 polje celije unutar A(1,2)
>> A(1,2) = {cell(2,2)};

%Popunite A ukljucujući ugnijezdeno
%polje
>> A(1,1) = {magic(5)};
>> A{1,2}(1,1) = {[5 2 8;7 3 0;6 7 3]};
>> A{1,2}(1,2) = {' Test'};
>> A{1,2}(2,1) = {[2-4i 5+7i]};
>> A{1,2}(2,2) = {cell(1,2)};
>> A{1,2}{2,2}(2) = {19};

%graficki prikaz ugnijezdenih celija
>> cellplot(A)
>> A
```




```
A =   [5x5 double]    {2x2 cell}
```

6.3.9 Pretvorba između polja ćelija i numeričkih polja

Primjer 6.11: *Pretvorba između formata ćelija i numeričkih formata*

Pretvorba između formata ćelija i numeričkih formata ostvaruje se najčešće `for` *petljom.*

```
%prvo stvorimo polje F
>> F{1,1} = [1 2 ; 3 4];
>> F{1,2} = [-1 0 ; 0 1];
>> F{2,1} = [7 8 ; 4 1];
>> F{2,2} = [4i  3+2i ; 1-8i  5];

%zatim koristimo for petlje za kopiranje
%sadrzaj F u numericki red NUM.
for k = 1:4
  for i = 1:2
    for j = 1:2
       NUM(i,j,k) = F{k}(i,j);
    end
  end
end
>> NUM

%slicno mozemo koristiti for petlju kako bi
%dodali svaku vrijednost numerickog polja
%svakoj celiji u polju celija.
>> G = cell(1,16);
>> for m = 1:16
     G{m} = NUM(m);
   end
>> G
```

```
NUM(:,:,1) = 1     2
             3     4
NUM(:,:,2) =  7     8
              4     1
NUM(:,:,3) = -1     0
              0     1
NUM(:,:,4) =
0 + 4.0000i     3.0000 + 2.0000i
1.0000 - 8.0000i      5.0000

G =
  Columns 1 through 11
[1]    [3]    [2]    [4]    [7]    [4]
[8]    [1]    [-1]    [0]    [0]
  Columns 12 through 16
[1]    [0+ 4.0000i]    [1.0000-
8.0000i]    [3.0000+ 2.0000i]
[5]
```

6.4 Linearni sustav

Linearni sustav: **A*X=b**, gdje je **A** matrica (mxn), **X** vektor rješenja (mx1), a **b** vektor desne strane (mx1).

6.4.1 Rješavanje sustava linearnih jednadžbi

U tablici 12. prikazane su i s kratkim primjerima opisane glavne funkcije koje se koriste u rješavanju sustava linearnih jednadžbi. Nakon toga, rješavanje je pokazano s različitim metodama i pod različitim uvjetima.

Tablica 6.12: *Linearne jednadžbe*

Funkcija	Opis funkcije	Primjer	Rezultat primjera
`\ i /`	Rješenje sustava linearnih jednadžbi A*X=B , pozovi također "help slash".	%sustav linearnih jednadžbi % x+2y=1, 5x+3y=2 >> A=[1 2;5 3] >> B=[1;2] >> X=A\B % X=inv(A)*B	A = 1 2 5 3 B = 1 2 X = 0.1429 0.4286
`inv()`	Inverzna matrica A (nxn) sustav A*X=B, X=A\B ili X=inv(A)*B	>> A1=[1 2;5 3] %dijagonalna matrica A2 >> A2=[2 0;0 3] >> I1=inv(A1) % inverz za A2 matricu %znači recipročne %vrijednosti >> I2=inv(A2)	A1 = 1 2 5 3 A2 = 2 0 0 3 I1 = -0.42 0.28 0.71 -0.14 I2 = 0.50 0 0 0.33
`pinv()`	Pseudoinverzna matrica A^+ (mxn) sustava A*X=B, $X=A^+\backslash B$, $A^+=(A'*A)^{-1}*A'$.	>> A=[1 2 3;4 5 6] >> b=ones(2,1) %pinv(A) daje ono rjesenje %X koje ima minimalnu %normu,min(norm(A*x -b)) >> X=pinv(A)*b	A = 1 2 3 4 5 6 b = 1 1 X = -0.5000 -0.0000 0.5000
`rcond()`	Procjena recipročne uvjetovanosti matrice. LAPACK - reciprocal condition estimator.	>> A=[1 2;5 3] >> I=inv(A) %reciprocna uvjetovanost %matrice A, ako je R blizu %1 onda je dobra, ako je %blizu 0 onda je loša >> R=rcond(A) %1-norma	A= 1 2 5 3 I = -0.4286 0.2857 0.7143 -0.1429 R = 0.1458
`cond()`	Kondicijski broj s obzirom na inverziju.	>> A=[1 2;5 3] >> I=inv(A) % uvjetovanost matrice A, %ako je R blizu 1 onda je %dobra, ako jeR >>1 onda %je loša >> R=cond(A,1) %1norma	A= 1 2 5 3 I = -0.4286 0.2857 0.7143 -0.1429 R = 6.8574

`chol()`	Cholesky factorizacija. Matrica mora biti pozitivno definitna i simetrična.	>> X = `pascal`(n) %R - gornje trokutasta %matrica >> R = `chol`(X) >> X = R'*R	X = 1 1 1 1 2 3 1 3 6 R = 1 1 1 0 1 2 0 0 1 X = 1 1 1 1 2 3 1 3 6
`lu()`	LU factorizacija. L-donje trokutasta (Lower) matrica, U-gornje (Upper) tr. mat.	>> A=[1 2 3;4 5 6;7 8 9]; %P-permutacijska matrica %koja biljezi zamjene %redaka u A %P*A=L*U >> [L,U,P]=`lu`(A)	L =1 0 0 .14 1 0 .57 .5 1 U = 7 8 9 0 .85 1.71 0 0 0 P = 0 0 1 1 0 0 0 1 0
`qr()`	Ortogonalna-trokutna dekompozicija.	>> A=[1 2 3;4 5 6;7 8 9]; %A=Q*R %R-gornjetrokutasta %matrica dimenzije A %Q-unitarna matrica >> [Q,R]=`qr`(A)	Q = -0.12 0.90 0.40 -0.49 0.30 -0.81 -0.86 -0.30 0.40 R = -8.12 -9.60 -11.1 0 0.90 1.80 0 0 0
`lsqnonneq`	Linearna optimizacija najmanjih kvadrata s nenegativnim ograničenjima. Naredba minimizira izraz `norm`(C*X-d) u X>=0	>> C = [0.0372 0.2869 0.6861 0.7071 0.6233 0.6245 0.6344 0.6170]; d = [0.8587 0.1781 0.0747 0.8405]; >> X = `lsqnonneg`(C,d) >> U = C\d	X = 0 0.6929 U = -2.5627 3.1108
`lscov()`	Najmanji kvadrati s poznatom kovarijancom.	%A*X = b+e, gdje e ima %kovarijancu V >> X = `lscov`(A,b,V)	

Matlab ima nekoliko alata za traženje rješenja sustava linearnih jednadžbi. Neka je **A** (m x n) matrica i neka je **b** m-dimenzionalni (stupčasti) vektor. Riješiti linerani sustav **Ax = b** može se upotrebom '*backslash' operatora* ' \ ', što se još zove dijeljenje s lijeva. Za

potpuno razumijevanje idućih primjera, potrebno je osnovno znanje iz područja linearne algebre. Razmotrimo različite slučajeve međusobnog odnosa stupaca i redaka matrice.

Primjer 6.12: *Slučaj m = n*

*Rješavanje linearnog sustava jednadžbi pri čemu je **A** kvadratna matrica (m=n)*

```
%U ovom slučaju dobiva se egzaktno rješenje sustava (na
%razini pogreške zaokruživanja). Neka je matrica A:

>> A = [1 2 3;4 5 6;7 8 10]

%Vektor b:
>> b = ones(3,1)

%Onda je rješenje
>> x = A\b

%Da bi se provjerila ispravnost izračunatog rješenja može
%se izračunati residuani vektor r
>> r = b - A*x

%Dobivena rješenja residuala 'r' teorijski bi sva trebala biti
%jednaka nuli. No, vidi se da postoje pogreške
%zaokruživanja kod takvog  izračunavanja.
```

```
A = 1 2 3
    4 5 6
    7 8 10
b=  1
    1
    1
x = -1.0000
     1.0000
     0.0000
r =
  1.0e-015 *
  0
  0.2220
 -0.6661
```

Primjer 6.13: *Slučaj m > n*

*Rješavanje linearnog sustava jednadžbi pri čemu je **A** nekvadratna matrica (m>n)*

```
%Ako je m > n, onda je sustav Ax = b u većini slučajeva
%nekonzistentan. Rješenje sustava Ax = b, dobivenog uz
%pomoć spomenutog operatora \  je rješenje najmanjih
%kvadrata.

%Neka je
>> A = [2 -1; 1 10; 1 2];
>> b = ones(3,1)

>> x = A\b

%Residualni vektor r izračunatog rješenja je ovog puta:
>> r = b - A*x
```

```
A =  2    -1
     1     10
     1     2
b=   1
     1
     1
x =   0.5849
      0.0491
r =  -0.1208
     -0.0755
      0.3170
ans =
  1.0e-014 *
  -0.0999
```

```
%Teorijski residualni vektor r je ortogonalan na vektorski
%prostor stupaca u matrici A, što se može pokazati sa:
>> r'*A
```

```
0.4885
```

Primjer 6.14: *Slučaj m < n*

*Rješavanje linearnog sustava jednadžbi pri čemu je **A** nekvadratna matrica (m<n)*

```
%Ako broj nepoznanica prelazi broj jednadžbi, za linearni
%sustav kažemo da je neodređen. U tom slučaju MATLAB
%računa pojedinačno rješenje za koje je sustav
%konzistentan. Na primjer:

>> A = [1 2 3; 4 5 6]
>> b = ones(2,1)

%onda je rjesenje:
>> x = A\b

%Opće rješenje danog sustava postiže se linearnom
%kombinacijom x sa stupcima nul prostora od  A. Potonje
%se računa upotrebom MATLAB funkcije null()
>> z = null(A)

%Pretpostavimo da želimo naći rješenje za linearnu
%kombinaciju x i z, s koeficijentima 1 i –1. upotrebom
%funkcije  lincomb dobiva se:
>> w = lincomb({1,-1},{x,z})

%Rezidual  r računa se na uobičajen način:
>> r = b - A*w

%Funkcija lincomb (lincomb.m) je korisnička funkcija koja
izgleda ovako:

function M = lincomb(v,A)
% Linearna kombinacija M od nekoliko matrica iste veličine.
% Koeficijenti v = {v1,v2,…,vm} od linearne kombinacije i
% matrice A = {A1,A2,...,Am} moraju se unijeti kao ćelije.
m = length(v);
[k, l] = size(A{1});
M = zeros(k, l);
for i = 1:m
  M = M + v{i}*A{i};
```

```
A = 1    2    3
    4    5    6

b =  1
     1

x = -0.5000
         0
    0.5000

z =   0.4082
     -0.8165
      0.4082

w = -0.9082
     0.8165
     0.0918

r =
  1.0e-015 *
   0.4441
   0.8882
```

```
end
```

Nadalje, ugrađena funkcija `rref()` nudi korisniku mogućnost rješavanja više problema linearne algebre. S njom je također moguće riješiti sustav linearnih jednadžbi i također rang matrice.

Funkcija `rref()` uzima matricu te vraća reduciranu redčanu formu (*reduced row echelon form*) svojih argumenata. Sintaksa `rref()` funkcije je

B = `rref`(**A**) ili [**B**, **pivot**] = `rref`(**A**)

Drugi izlazni parametar **pivot** čuva indekse pivot stupaca.

Primjer 6.15: *Reducirana redčana forma argumenata matrice*

Reducirana redčana forma argumenata matrice koristeći Gauss-Jordanove eliminacije s parcijalnim pivotiranjem. Rješenje ***x*** *za linearni sustav* ***Ax = b*** *postiže se u dva koraka. Prvo se (augmented) matrica sustava transformira u reducirani redčani oblik, a poslije se izvlači zadnji stupac matrice kao rješenje sustava:*

```
>> A = magic(3)
>> b = ones(3,1)

%prosirena (augmented) matrica -> [A b]
>> [x, pivot] = rref([A b])
>> x = x(:,4)

%Rezidual izračunatog rješenja je:
>> r=b - A*x

%Informacija spremljena u izlazni parametar 'pivot'
%može se iskoristiti za izračunavanje ranga matrice A
>> R=length(pivot)
```

```
 A = 8     1     6
     3     5     7
     4     9     2
b = 1
    1
    1
x =
1     0     0     0.0667
0     1     0     0.0667
0     0     1     0.0667
pivot = 1     2     3
x = 0.0667
    0.0667
    0.0667
r =  0
     0
     0
R = 3
```

Napomena: Dva vektora su ortogonalna, ako je kut između njih pravi, tj. ako je skalarni produkt tih vektora jednak nuli. Dva vektora su ortonormirana ako su ujedno ortogonalni i normirani na dužinu 1. Jedinični vektori kartezijskog 3D prostora su ortonormirani

vektori (**X**=[1 0 0], **Y**=[0 1 0], **Z**=[0 0 1], što daje ortonormiranu matricu **A**=[**X**;**Y**;**Z**] iz koje se odmah može vidjeti da su to tri linearno nezavisna vektora u trodimenzionalnom Euklidskom prostoru i zato je rang te matrice jednak 3). Ako komponente vektora kvadriramo i zbrojimo onda dobijemo duljinu vektora. Ortogonalana matrica ima ortonormirane stupce. U ortogonalnoj matrici duljine svakog vektora moraju biti jedinične, a to provjeravamo kada svaku komponentu vektora (npr. redak matrice) pomnožimo samu sa sobom (kvadriramo komponentu) i zbrojimo te umnoške. Taj postupak se može jednostavno napraviti tako da se ortogonalna matrica pomnoži s svojom transponiranom što mora rezultirati jediničnom matricom. Ortogonalna matrica je ona čija je transponirana matrica jednaka inverznoj, a umnožak matrice s njenim inverzom daje jediničnu matricu.

Napomena: Podsjetnik o povezanosti između determinante i inverza matrice: Želimo naći neku kvadratnu matricu **B** kojom bismo s desna matrično pomnožili kvadratnu matricu **A**, a da pritom dobijemo dijagonalnu matricu s elementima koji imaju vrijednost kao determinanta matrice **A**. To bi bilo ovako **A*B**=`det`(**A**)***I**, gdje je **I** jedinična matrica istih dimenzija kao **A** i **B**. Iz sljedećeg izraza se može zaključiti da inverz matrice postoji uz uvjet da je determinanta različita od nule. Pomnožimo sada matrično s lijeva gornju jednadžbu (**A**$^{-1}$***A*B**=**A**$^{-1}$*`det`(**A**)***I**), a onda ju podijelimo sa determinantom (skalar) i dobijemo $\frac{\mathbf{A}^{-1}*\mathbf{A}*\mathbf{B}}{\det(\mathbf{A})} = \mathbf{A}^{-1}*\mathbf{I}$. Uz svojstva invertibilnih matrica vrijedi **A**$^{-1}$***A=A*A**$^{-1}$=**I**, te da je umnožak matrice s jediničnom opet ta ista matrica onda dobijemo izraz za inverznu matricu: $\mathbf{A}^{-1} = \frac{\mathbf{B}}{\det(\mathbf{A})}$. Matrica **B** se naziva *adjungirana* matrica i može da dobiti direktno iz matrice **A** . Ako iz **A** uklonimo i-ti redak i j-stupac, determinanta od preostalih elemenata naziva se *minor* m(i,j). Matrica **M**(i,j) sadrži kao elemente minore m(i,j). Znači da ako iz **A** uklonimo 3. redak (i=3) i 2. stupac (j=2) onda se u **M**(3,2) umeće element koji je u stvari determinanta preostalih elemenata iz **A**. Nakon što smo dobili matricu **M**(i,j), moramo je pomnožiti s $(-1)^{i+j}$ što je u stvari promjena predznaka svakog drugog elementa u **M**. Takvu promijenjenu matricu **M** nazivamo matricom *kofaktora* **C**(i,j). Transponiranjem **C**(i,j) dobili smo adjungiranu matricu **B** (**B**=**C**').

6.4.2 Homogen linearni sustav $\mathbf{b} = \mathbf{0}$

Homogen linearni sustav **A*X=0** ima samo trivijalno rješenje (**X=0**) kada je matrica **A** regularna (kvadratna, ima inverz **A**$^{-1}$, tj. `det`(**A**)~=0). Budući da je determinanta matrice sustava jednaka nuli to znači da je rang kvadratne matrice **A** (nxn) potpun (`rank`(**A**)=n) što isto znači da ima n nezavisnih nepoznanica koje tada razapinju n-dimenzionalan prostor. Iz toga slijedi da linearni sustav ima jedinstveno rješenje i to samo za n-dimenzionalan vektor rješenja kojem su svih n komponenti jednake nuli (**X=0**).

Ako linearni sustav od n jednadžbi ima dvije jednake jednadžbe (jedna jednadžba je samo pomnožena s nekim brojem i predstavlja drugu jednadžbu, onda te dvije jednadžbe i dalje predstavljaju isti pravac u n-dimenzionalnom prostoru) onda taj sustav nema puni rang (`rank`(**A**)<n) i onda je matrica sustava **A** neregularna, tj. singularna i `det`(**A**)=0. U tom slučaju će osim trivijalnog postojati i beskonačno mnogo netrivijalnih rješenja. Grafički zamislimo linearni sustav čije su jednadžbe pravci u n-dimenzionalnom prostoru. Budući da u tim jednadžbama nema slobodnog člana koji bi mogao predstavljati desnu stranu sustava onda ti pravci (jednadžbe) prolaze kroz središte koordinatnog sustava. Točka u kojoj se svi pravci sijeku predstavlja grafičko rješenje homogenog linearnog sustava i to je ishodište koordinatnog sustava (trivijalno rješenje).

6.4.3 Nehomogen linearni sustav $\mathbf{b} \neq \mathbf{0}$

1. Matrica **A** (nxn) je kvadratna, regularna (nesingularna), znači da ima inverz (mora vrijediti `det`(**A**)$\neq$**0**), a pritom je i rang matrice **A** potpun (`rank`(**A**)=n). Inverznu matricu mogu imati samo kvadratne matrice. Rješenje sustava: $\mathbf{X}=\mathbf{A}^{-1}*\mathbf{b}$, ili **X**=`inv`(**A**), ili **X**=**A****b**. Matlab najbrže izvodi matričnu operaciju sa "backslash-om". Kod ovako definiranog linearnog sustava postoji jedinstveno rješenje.

Primjer 6.16: *Nehomogen linearni sustav s jedinstvenim rješenjem*

Dva pravca koja se sijeku u jednoj točki
2x-3y=-2
4x+y=24

```
>> A=[2 -3;4 1]
>> b=[-2;24]

%X -> rjesenje linearnog sustava
>> X=A\b
%D -> determinanta matrice A
>> D=det(A)
%R ->reducirana forma (Gauss-Jordan)
>> R=rref(A)
%Aug -> prosirena (augmented) matrica
>> Aug=[A,b]
>> RA=rref(Aug)
>> Rg=rank(A)  %rang matrice A
```

```
A = 2    -3
    4     1
b = -2
    24
X = 5
    4
D = 14
R = 1     0
    0     1
Aug = 2    -3    -2
      4     1    24
RA = 1     0     5
     0     1     4
Rg =  2
```

2. Matrica **A** (nxn) je kvadratna, neregularna (singularna), znači da ima nema inverz (mora vrijediti `det`(**A**)=0), a pritom je rang matrice **A** nepotpun (`rank`(**A**)<n). Rješenje sustava: $\mathbf{X}=\mathbf{A}^{-1}*\mathbf{b}$, ili **X**=`inv`(**A**), ili **X**=**A****b** ne može se dobiti ovim načinima budući da

je **A** singularna. Matlab najbrže izvodi matričnu operaciju sa "backslash-om". Kod ovako definiranog linearnog sustava postoji beskonačan broj rješenja budući da se ta dva pravca sijeku u beskonačno točaka. To se može provjeriti korištenjem Gauss-Jordanovim eliminacijama nad proširenom (augmented, **Aug**=[**A**,**b**]) matricom, **RA**=`rref`(**Aug**).
a). Ako su svi elementi zadnjeg retka matrice **RA** jednaki nuli, znači da je matrica **A** singularna i onda sustav ima beskonačan broj rješenja. Provjera: `rank`(**A**)=`rank`(**Aug**)<n.
b). Ako su elementi zadnjeg retka matrice **RA** jednaki nuli osim zadnjeg elementa koji je različit od nule, to znači da je matrica **A** singularna i rješenje linearnog sustava ne postoji. Provjera: `rank`(**A**)<`rank`(**Aug**).
c). Jedinstveno rješenje postoji kada je u zadnjem retku **RA** predzadnji element jednak jedinici, a zadnji različit od nule. Provjera: `rank`(**A**)=`rank`(**Aug**)=n.
Matlab u slučaju a) i b) ne prikazuje ni jedno od rješenja nego javlja upozorenje da je matrica **A** singularna.

Da bi se našlo i prikazalo makar jedno rješenje Matlab se koristi metodom najmanjih kvadrata u funkciji `pinv`(**A**), pseudo-inverzu matrice **A**. Pseudo-inverz se koristi za dobivanje rješenja kod linearnih sustava koji imaju singularnu matricu **A** i onih linearnih sustava koji imaju pravokutnu matricu **A** (mxn, m~=n).
Dobivanje rješenja korištenjem pseudo-inverzne matrice dat će određene iznose rješenja i u slučaju kad linearni sustav nema rješenja, pa to može zavarati misleći da smo dobili rješenje linearnog sustava. Zato je prije traženja rješenja potrebno provjeriti pomoću matrice **RA** da li rješenje uopće postoji.
Inverzna matrica postoji samo kod kvadratnih matrica i rješenje linearnog sustava dobije se iz izraza $\mathbf{X}=\mathbf{A}^{-1}*\mathbf{b}$. U slučaju da je matrica **A** nekvadratna (mxn, m~=n), onda se rješenje sustava ne može dobiti na isti način kao i za kvadratne matrice. Zato će se posegnuti za određenom transformacijom kojom će se nekvadratna matrica **A** transformirati u kvadratnu matricu. Kada transponiranom matricom **A'** (nxm) pomnožimo matricu **A** (mxn) dobijemo kvadratnu matricu tipa (nxn). Primijenimo to na linearni sustav **A*X=b** tako što ćemo s **A'** pomnožiti s lijeva sustav i dobit ćemo **A'*A*X=A'*b** , te sređivanjem izraza dobijemo rješenje linearnog sustava $\mathbf{X}=(\mathbf{A'*A})^{-1}*\mathbf{A'*b}$. Sada inverzni dio tog izraza $(\mathbf{A'*A})^{-1}$ može postojati budući da umnožak **A'*A** daje kvadratnu matricu. Sada se može reći da je rješenje linearnog sustava $\mathbf{X}=\mathbf{A}^{+}*\mathbf{b}$, pri čemu je $\mathbf{A}^{+}=(\mathbf{A'*A})^{-1}*\mathbf{A'}$ tj. **Ap** = `inv`(**A'*A**)***A'*b** i naziva se pseudo-inverzna matrica (Matlabova naredba za $\mathbf{A}^{+}$ je `pinv`(**A**)). Tako se može riješiti linearni sustav s pravokutnom matricom **A**. Na ovaj način mogu se riješiti i svi ostali tipovi linearnog sustava (npr. za kvadratnu matricu **A**), samo se ne preporučuje korištenje funkcije `pinv`() tamo gdje se može linearni sustav riješiti pomoću funkcije `inv`(), "backslasha" itd., jer `pinv`() koristi puno više koraka za dobivanje rješenja nego `inv`(), "backslash" i ostali matrični operatori.
Budući da umnožak dviju pravokutnih matrica **A'*A** daje kvadratnu matricu, to ne znači da i ta kvadratna matrica ne može biti singularna (determinanta jednaka nuli). Ako se to desi onda se ne može na način **X**=`inv`(**A'*A**)***A'*b** dobiti rješenje linearnog sustava, jer se ne može izračunati `inv`(**A'*A**). Ipak, u slučaju da je ta kvadratna matrica **A'*A** singularna na način da postoji beskonačno rješenja, onda će izraz **A=**`pinv`(**A**)***b** dati

jedno od rješenja i to ono koje se dobije metodom najmanjih kvadrata (minimizacijom norme, `norm`(**A*x-b**)).
Razlika između $\mathbf{A}^{+}$=`inv`(**A'*A**)***A'** i funkcije `pinv`(**A**) je u tome što se funkcija `pinv`() ne koristi u računanju matricu **A** nego njene matrice koje nastaju dekompozicijom singularnih vrijednosti (**A** = **U*S*V'**, tj. funkcija u Matlabu je `svd`(**A**)).

Primjer 6.17: *Nehomogen linearni sustav s beskonačnim brojem rješenja*

Dva pravca koja leže jedan na drugom
9x+y=36
3x+(1/3)y=12

```
>> A=[9 1;3 1/3]
>> b=[36;12]

%X -> rjesenje linearnog sustava
>> X=A\b
%D -> determinanta matrice A
>> D=det(A)
%R ->reducirana forma (Gauss-Jordan)
>> R=rref(A)
%Aug -> prosirena (augmented) matrica
>> Aug=[A,b]
>> RA=rref(Aug)
>> Rg=rank(A)  %rang matrice A
>> X=pinv(A)*b
>> b=A*X
```

```
A = 9.0000    1.0000
    3.0000    0.3333
b = 36
    12
Warning: Matrix is singular to
working precision.
(Type "warning off
MATLAB:singularMatrix" to
suppress this warning.)
X = Inf
    Inf
D = 0
R =  1.0000    0.1111
     0         0
Aug = 9.000    1.000   36.000
      3.000    0.333   12.000
RA = 1.0000    0.1111    4.0000
     0         0         0
Rg =  1
X = 3.9512
    0.4390
b = 36.0000
    12.0000
```

Primjer 6.18: *Nehomogen linearni sustav bez rješenja*

Dva pravca koja leže paralelno jedan naspram drugog
7x+2y=16
-21x-6y=24

```
>> A=[7 2;-21 -6]
>> b=[16;24]
```

```
A = 7     2
   -21   -6
```

```
%X -> rjesenje linearnog sustava
>> X=A\b

%D -> determinanta matrice A
>> D=det(A)

%R ->reducirana forma (Gauss-Jordan)
>> R=rref(A)

%Aug -> prosirena (augmented) matrica
>> Aug=[A,b]
>> RA=rref(Aug)
>> Rg=rank(A)  %rang matrice A

%pseudo-inverz daje rjesenja kod sustava
%koji nema rjesenja!
>> X=pinv(A)*b
%provjera vektora desne strane lin. sustava
>> b=A*X
```

```
b = 16
      24
Warning: Matrix is singular to
working precision.
(Type "warning off
MATLAB:singularMatrix" to
suppress this warning.)
X = Inf
      Inf
D = 0
R = 1.0000    0.2857
      0            0
Aug = 7       2    16
       -21    -6    24
RA = 1.0000    0.2857        0
       0            0          1.0000
Rg = 1
X = -0.7396
      -0.2113
b = -5.6000
      16.8000
```

Primjer 6.19: *Rješavanje linearnog sustava s ne-kvadratnom matricom*

*Da bi se riješio linearni sustav **A***__X__=__b__ s ne-kvadratnom matricom **A**(mxn), koristi se pseudo - inverzna matrica, __X__=`pinv`(**A**)*__b__ ili operator matričnog dijeljenja s lijeva, __X__=**A**__b__. Budući da **A** nije kvadratna onda ima različit broj redaka (m) i stupaca (n). Broj stupaca ujedno predstavlja i broj nepoznanica linearnog sustava, a broj redaka predstavlja broj jednadžbi istog sustava. Ako je n>m onda je broj nepoznanica veći od broja jednadžbi pa sustav ima proizvoljan broj rješenja. Ako je m>n onda jednadžbi ima više nego nepoznanica, što ipak ne znači da postoji jedinstveno rješenje sustava. Da bi jedinstveno rješenje postojalo potrebno je toliko nezavisnih jednadžbi koliko ima nepoznanica (`rank`(**A**)=n). Za m>n jednadžbi ima više od nepoznanica te linearni sustav može imati sve kombinacije rješenja, jedinstveno rješenje, bezbroj rješenja ili da ono ne postoji. Slijedeći primjeri će pokazati takve različite mogućnosti.*

```
%tri pravca se sijeku u tri razlicite tocke (ne
%postoji rjesenje linearnog sustava)
>> A=[1 -1;2 1;1 3];
>> b=[0;-1;4];
>> Aug=[A,b]
>> R1=rref(Aug)
```

```
Aug =  1    -1     0
       2     1    -1
       1     3     4
R1 =   1     0     0
       0     1     0
       0     0     1
```

```
>> X1=A\b  %daje netocno rjesenje
>> X11=pinv(A)*b  %daje netocno rjesenje

%dva pravca leze jedan na drugom i sijeku se
%s trecim, pri cemu postoji tocka sjecista svih
%triju pravaca (jedinstveno rjesenje)
>> A=[1 -1;2 1;4 2];
>> b=[0;-1;-2];
>> Aug=[A,b]
>> R2= rref(Aug)
>> X2=A\b
>> X22=pinv(A)*b

%sva tri pravca leze jedan na drugom i sijeku
%se u beskonacnom broju tocaka
>> A=[6 3;2 1;4 2];
>> b=[-3;-1;-2];
>> Aug=[A,b]
>> R3=rref(Aug)
>> X3=A\b %daje samo jedno od rjesenja
>> X33=pinv(A)*b %daje jedno od rjesenja


%tri razlicita pravca prolaze kroz istu tocku
%(jedinstveno rjesenje)
>> A=[1 1;1 0.5; 1 -0.5];
>> b=[1;1;1];
>> Aug=[A,b]
>> R4=rref(Aug)
>> X4=A\b
>> X44=pinv(A)*b
```

```
X1 =   -0.4400
        1.1600
X11 = -0.4400
        1.1600
Aug =  1    -1     0
       2     1    -1
       4     2    -2
R2 =  1.0000        0    -0.3333
      0          1.0000   -0.3333
      0               0        0
X2 =   -0.3333
       -0.3333
X22 = -0.3333
       -0.3333
Aug =  6     3    -3
       2     1    -1
       4     2    -2
R3 =  1.0000    0.5000   -0.5000
      0           0           0
      0           0           0
Warning: Rank deficient,
rank = 1  tol =   4.9849e-015.
X3 = -0.5000
      0
X33 = -0.4000
       -0.2000
Aug = 1.0000    1.0000    1.0000
      1.0000    0.5000    1.0000
      1.0000   -0.5000    1.0000
R4 =   1     0     1
       0     1     0
       0     0     0
X4 = 1
     0
X44 = 1.0000
       -0.0000
```

6.5 Polinomi

Matlab predočuje polinome preko vektora i to tako da za polinom N-tog reda koristi N+1 članova dug vektor, čiji su elementi koeficijenti polinoma, poredani u opadajućem nizu polinomnih eksponenata. Tako će se polinom

$$x^4 - 12\,x^3 + 25x + 116$$

u Matlab-u predočiti s:

```
>> p = [ 1 -12  0  25  116]
```

Primijetite da svaki član polinoma mora biti uključen, bez obzira koliki je njegov koeficijent: ako je jednak nuli, onda se na njegovom mjestu u vektoru koeficijenata stavlja nula. U protivnom, načinit će se pogreška, jer će se smanjiti red polinoma.

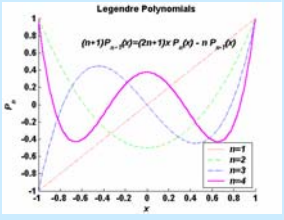
Polinom je moguće zadati i preko rekurzivnih formula. Takvi su, na primjer, Legeandre-ovi polinomi zadani s rekurzivnom jednadžbom:

$$(n+1)P_{(n+1)}(x)=(2n+1)xP_n(x) - nP_{n-1}(x),\ \ P_0=0;$$

Primjer 6.20: *Legeandreovi polinomi*

Program rješava prva četiri Legeandreova polinoma.
Napomena: ***legend****-a se može pozicionirati uz pomoć zadnjeg argumenta:*
-1: nadesno od crteža, 0: automatski (na 'najboljoj' poziciji), 1: gore desno (pretpostavljena vrijednost), 2: gore lijevo; 3: dolje lijevo, 4: dolje desno

```
>> x=-1:.01:1;
>> p1=x;
>> p2=(3/2)*x.^2 - 1/2;
>> p3=(5/2)*x.^3-(3/2)*x;
>> p4=(35/8)*x.^4-(15/4)*x.^2+3/8;
>> plot(x,p1,'r:',x,p2,'g--',x,p3,'b-.',x,p4,'m-')
>> box off

>> legend('\itn=1','n=2','n=3','n=4',4)
>> xlabel('x','Fontsize',12,'FontAngle','italic')
>> ylabel('P_n','Fontsize',12,'FontAngle','italic')

>> title('Legendre Polynomials','FontSize',14)
>> text(-.6,.7,'(n+1)P_{n+1}(x)=(2n+1)x P_n(x) -
n P_{n-1}(x)','FontSize',12, 'FontAngle', 'Italic')
```



Tablica 6.13: *Osnovne funkcije za rad s polinomima*

Funkcija	Opis funkcije	Primjer	Rezultat primjera

`roots()`	Nalazi korijene polinoma.	%polinom A 2.stupnja >> A=[1 -4 3] >> R = `roots`(A)	A = 1 -4 3 R = 3 1
`poly()`	Za zadane korijene polinoma nalazi polinom.	>> A=[1 -4 3]; >> R = `roots`(A) >> P=`poly`(R)	R = 3 1 P = 1 -4 3
`polyval()`	Izračunava vrijednost polinoma u zadanoj točki	>> A=[1 -4 3]; >> T=polyval(P,1) >> T=polyval(P,2)	T = 0 T = -1
`polyvalm()`	Izračunava polinom za zadanu matricu kao argument.	>> X=[0 1;2 3] >> T=`polyvalm`(P,X)	X = 0 1 2 3 T = 5 -1 -2 2
`residue()`	Razvoj polinoma u parcijalne razlomke (residui).	>> A=[1 -4 3];%nazivnik >> B=[1 -4]; %brojnik %elementi iz vektora **R** su %brojnici parcij. razlom. >> R=`residue`(B,A)	R = -0.5000 1.5000
`polyfit()`	Nadomještanje (fit) podataka polinomom.	>> X=[1 2 3]; %tocke x-osi >> Y=[1 2 4]; %tocke y-osi %aproksimacija točaka iz %X,Y polinomom P 1.reda >> P=`polyfit`(X,Y,1)	P = 1.5 -0.6667
`polyder()`	Diferencijacija polinoma.	>> A=[1 -4 3] >> D=`polyder`(A)	A = 1 -4 3 D = 2 -4
`polyint()`	Analitička integracija polinoma.	>> A=[1 -4 3] >> I=`polyint`(A)	A = 1 -4 3 I = 0.33 -2 3 0
`conv()`	Množenje polinoma.	>> A=[1 -4 3]; >> B=[1 -4]; >> C=`conv`(A,B)	C = 1 -8 19 -12
`deconv()`	Dijeljenje polinoma.	>> A=[1 -4 3]; >> B=[1 -4]; >> [Q,R]=`deconv`(A,B) >> K=`conv`(B,Q)+R	Q = 1 0 R = 0 0 3 K = 1 -4 3

Primjer 6.21: *Nul-točke polinoma*

Traže se nul-točke (korijeni) polinoma $P=x^4-12x^3+25x+116$. Korijeni polinoma mogu biti realni i imaginarni. Primijetite da funkcija roots() vraća stupčasti vektor. Inverzna funkcija od roots() je poly(), pa će ona dati početne koeficijente polinoma.

```
%Za polinom zadan vektorom
>> p = [1 -12 0 25 116];
```

```
R =  11.7473
      2.7028
```

```
%funkcija roots() naći će korijene polinoma
>> R=roots(p)
>> pp = poly(R)
```

```
    -1.2251 + 1.4672i
    -1.2251 - 1.4672i
pp = 1  -12   0   25  116
```

Dvije posebno značajne funkcije za rad s polinomima su `polyval()` i `polyfit()`. `polyval()` funkcija vraća vrijednost polinoma p(x) u zadanoj vrijednosti x. Sintaksa je:

`polyval`(**kpol**, **toc**)

gdje je **kpol** vektor koeficijenata polinoma, a **toc** je vektor točaka u kojima tražimo vrijednost polinoma. Tako će za

```
>> p = [1 -12 0 25 116]
>> polyval(p,1)
ans =
      130
```

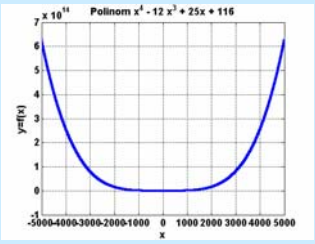
što je vrijednost polinoma u točki x=1, tj. $P(x) = x^4 - 12\ x^3 + 25x + 116 = 1^4 - 12*1^3 + 25*1 + 116 = 130$.

Na isti način funkcija `polyval()` izračunat će p(x) za bilo koju točku ili niz točaka.

Primjer 6.22: *Računanje vrijednosti polinoma u zadanim točkama*

Funkcijom polyval() možemo izračunati vrijednosti polinoma u željenim točkama.

```
>> p = [1 -12 0 25 116];
>> x=-5000:5000;
>> fx=polyval(p,x);

%crtanje grafa
>> plot(x,fx)
>> title('Polinom x^4 - 12 x^3 + 25x + 116')
>> xlabel('x'),ylabel('y=f(x)'),grid
```



Jasno je da se `polyval()` funkcija mogla zamijeniti jednostavnim programskim odsječkom. Za polinom n-tog reda:

$$y=p(x)= a_1x^n + a_2x^{n-1} + ... + a_nx^1 + a_{n+1}$$

uz poznate koeficijente $[a_1\ a_2\ ...\ a_n\ a_{n+1}]$ za neku vrijednost x može se p(x) izračunati direktno po gornjoj formuli sa sljedećim programom:

```
n = length(a) - 1;
nx = length(x);
y = zeros(1,nx );
for  k = 1 : n + 1
   y = y + a(k)* x.^(n - k + 1)
end
```

Inverzan problem je, ako imamo zadani niz točaka i njihovih vrijednosti, te želimo saznati kojem polinomu odgovaraju, onda se moramo poslužiti interpolacijskim formulama kao što su Lagrangeove, Newton-ove i sl. Matlab posjeduje funkcija `polyfit` koja koeficijente polinoma aproksimira metodom najmanjih kvadrata za niz podataka zadanih s parovima točaka (x,y).
Drugim riječima, neka je zadano m-točaka: $\{X_i, Y_i\}_{i=1:m}$. Treba naći polinom p stupnja n tako da je $p(x_i)$ približno jednako y_i za sve $i=1:m$. Funkcija `polyfit()` za zadani red polinoma daje koeficijente polinoma p tako da je zbroj kvadrata razlike odstupanja $(p(x_i)-y_i)^2$ minimalan, za $i=1:m$.

Sintaksa funkcije je:

```
polyfit(x,y,n)
```

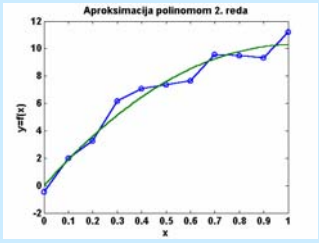
gdje je **x** vektor x-kordinata, **y** vektor vrijednosti funkcije f(**x**), a n je red polinoma po kojem aproksimiramo podatke. Funkcija vraća niz koeficijenata polinoma zadanog reda.

Primjer 6.23: *Aproksimiranje podataka polinomom*

Skup podataka može se aproksimirati polinomom n-tog reda i za to se koristi funkcija polyfit(). Podaci su definirani koordinatama x,y.

```
>> x = [0:0.1:1];
>> y = [-0.447  1.978  3.28  6.16  7.08  7.34
7.66  9.56  9.48  9.30  11.2];
>> p=polyfit(x,y,2) %polinom 2. reda
%aproksimacija podataka
>> xi=linspace(0,1,100);
>> z=polyval(p,xi);
>> plot(x,y,'-o',xi,z,'-');
>> xlabel('x'); ylabel('y=f(x)');
>> title('Aproksimacija polinomom 2. reda');
```

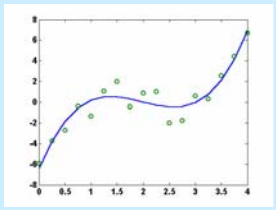
```
p =  -9.810   20.129   -0.031
```

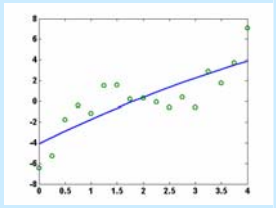


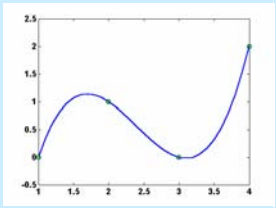
Primjer 6.24: *Interaktivno određivanje reda polinoma*

Ovaj primjer pokazuje kako se nekom polinomu kojemu su (x,y) podacima dodani slučajni podaci (šum), interaktivno određuje red polinoma i promatra promjena.

```
>> p=[1 -6 11 -6];
>> x=0:.25:4;
>> y=polyval(p,x)+randn(1,length(x));
>> while 1
        n=input('Red polinoma n=' );
        if n<=0, break, end
        c=polyfit(x,y,n);
        fit=polyval(c,x);
        plot(x,fit,x,y,'o'), pause
end
%za promjenu reda polinoma pritisni Enter
%za izlaz iz beskonacne while petlje Ctrl+C
%prva slika rezultata je za n=3, druga za n=2


%Da raspršenja podataka nema, aproksimacija bi
%bila potpuno točna (treca slika).
>> xd = 1 : 4;  yd = [0 1 0 2];
>> x = 1 : 0.1 : 4;
>> y = polyval(polyfit(xd,yd,length(xd)-
1),x);
>> plot(x,y,xd,yd,'o')
```



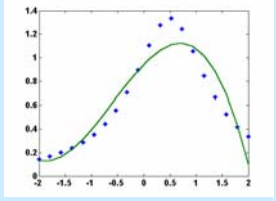


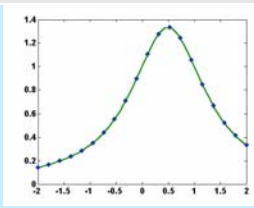


Primjer 6.25: *Aproksimiranje funkcije polinomom*

Traži se polinom trećeg reda koji u 20 točaka u intervalu [-2,2] po metodi najmanjih kvadrata aproksimira funkciju $f(x)=1/(x+(1-x)^2)$. Egzaktna interpolacija zadanih podataka može se dobiti preko funkcije spline().

```
>> x=linspace(-2,2,20);
>> y=1./(x+(1-x).^2);
>> p=polyfit(x,y,3);
>> plot(x,y,'*',x,polyval(p,x),'-'); %prva slika

%egzaktna interpolacija
>> x=linspace(-2,2,20);
>> y=1./(x+(1-x).^2);
>> xx=linspace(-2,2,60);
>> yy=spline(x,y,xx);
>> plot(x,y,'*',xx,yy,'-'); %druga slika
```





Primjer 6.26: *Analitička integracija i derivacija polinoma*

Za analitičku integraciju i derivaciju polinoma dolaze dvije funkcije u paru: polyint() – integrira, a polyder() – derivira polinom.
Neka je zadan polinom $p = x^3 + 7x - 2$. Pozivom polyint(p,5) s drugim argumentom (5) koji odgovara konstanti integracije što odgovara polinomu $p=1/4\ x^4 + 7/2\ x^2 - 2\ x + 5$. Pozivom polyder() dobit ćemo koeficijente polinoma $3x^2+7$.
Lako je uočiti poznatu činjenicu da integracijom red polinoma raste, a derivacijom pada za 1.

```
>> p = [ 1    0    7   -2 ]
>> integral = polyint(p,5)
>> deriv = polyder(p)
```

```
p =     1    0    7   -2
integral = 0.2500    0    3.5000   -2.0000   5.0000
deriv =     3    0    7
```

Primjer 6.27: *Množenje i dijeljenje polinoma*

Funkcije za množenje (conv()) i dijeljene (deconv()) polinoma.
Neka su zadani polinomi $a = x^3 + 2x^2 + 3x + 4$ i $b= 4x^2 + 9x + 16$
Njihov umnožak je $c = 4\ x^5 + 17\ x^4 + 46\ x^3 + 75\ x^2 + 84x + 64$, a kvocijent $q = 0.2500\ x - 0.0625$.

```
>> a = [1 2 3 4];
>> b = [4 9 16];
>> c = conv(a,b)
>> q=deconv(a,b)
```

```
c =     4    17    46    75    84    64
q =    0.2500   -0.0625
```

HANDLE GRAFIKA, GUI, ANIMACIJA 7

Handle grafika je grafika najniže razine s kojom se mogu postići najsloženiji grafički efekti. Ime je dobila po držalu (engl. *handle*) grafike, što je kompjutorski gledano obično kazalo (engl. *pointer*) na grafički objekt. Grafički objekti su osnovni elementi za crtanje, zovu se još i primitivama, koje Matlab koristi za prikaz podataka i stvaranje korisničkog grafičkog sučelja (graphical user interfaces - GUI). Ti objekti su:

- **Root**
- **Figure**
- **Axes**, **Uicontrol**, **Uimenu**, **Uicontextmenu**
- **Image**, **Light**, **Line**, **Patch**, **Rectangle**, **Surface**, **Text**

U ovom poglavlju bit će opisani navedeni objekti tako da će se prvo proučiti Root, Figure i Axes objekti, a zatim njihovi sljedbenici: Image, Light, Line, Patch, Rectangle, Surface i Text. Potom će se u posebnom potpoglavlju opisati korisničko grafičko sučelje GUI, s objektima: Uicontrol, Uimenu, Uicontextmenu. Na koncu, u posebnom potpoglavlju, obradit će se animacija.

7.1 Handle grafika

Kao što se iz redaka nabrojenih objekata može zaključiti, radi se o hijerarhijskoj strukturi objekata, gdje je viša razina nadređena nižoj. Svakom stvorenom objektu pridružuje se spomenuto držalo, jedinstveni broj po kojem se objekti razlikuju. Upotrebom držala mogu se mijenjati značajke ili svojstva (engl. *properties*) postojećih grafičkih objekata. Handle grafika omogućuje i animaciju, jer se na brz i lak način, mijenja svojstvo vidljivosti ili nevidljivosti nekog grafičkog objekta.

7.1.1 Dohvaćanje i postavljanje objektnih svojstava

Objektna svojstva uključuju opću informaciju (oko 15 svojstava), kao što je tip objekta (iz gornje navedene strukture), zatim prethodnike i sljedbenike (engl. *parent and children*), je li objekt vidljiv ili nije, kao i informaciju svojstvenu samo tom objektu. Korisnik može tražiti (funkcija `get()`) trenutačnu vrijednost bilo kojeg svojstva željenog objekta ili postaviti (funkcija `set()`) vrijednosti svih svojstava koja se mogu postaviti (neka svojstva se mogu samo čitati, dohvatiti). Svaki tip grafičkog objekta ima odgovarajuću naredbu, funkciju, za njegovo stvaranje: na primjer, `figure()` će stvoriti

objekt slike, `text()` funkcija stvorit će text objekt i sl. Dosad upoznate grafičke funkcije, npr. `plot()`, stvorit će istodobno više objekata, npr. prozor, sliku, osi, liniju itd. Naredbom `findobj` dobiju se handle brojevi, držala, pojedinih objekata. Tako će:

```
>> plot(1:10,'o-')
>> h=findobj
h =
     0
     1.0000
     100.0011
     3.0012
```

što znači da je `plot()` funkcija stvorila četiri objekta i pridijelila im četiri jedinstvena broja. Želimo li saznati o kojim se objektima radi, koristit ćemo funkciju `get()`.

```
>> get(h,'type')
ans =
     'root'
     'figure'
     'axes'
     'line'
```

Dakle, radi se o prozoru (root), slici (figure), osima (axes) i liniji (line) – sve je to stvoreno izvršenjem naredbe `plot()`. Root odgovara zaslonu računala, uvijek ima identifikacijski broj 0. Ostali objekti dobivaju brojeve ovisno u dotadašnjem radu (engl. *session*) u Matlabu, handle brojevi, naime, generiraju se uvijek novi, pa se ne može unaprijed znati (osim za root) koji objekt će dobiti koji broj.

Osnovna sintaksa za dohvaćanje specifičnog svojstva nekog objekta je:

```
vracena_vrijednost = get(handle,'Ime_svojstva');
```

a za postavljanja vrijednosti je:

```
set(handle,'Ime_svojstva','Nova_vrijednost_svojstva')
```

Ako se izostavi 'Nova_vrijednost_svojstva', onda `set` naredba samo vraća vrijednosti za zadano ime svojstva. Na primjer:

```
>> set(h(4))
     Color
     EraseMode: [ {normal} | background | xor | none ]
     LineStyle: [ {-} | -- | : | -. | none ]
     LineWidth
```

```
        Marker: [ + | o | * | . | x | square | diamond | v | ^ | > | < | pentagram |
    hexagram | {none} ]
        MarkerSize
    ….
        UserData
        Visible: [ {on} | off ]
```

Ispisat će sva svojstva za četvrti handle prošlog primjera (Line). Zbog velikog broja svojstava, navedena su samo početna i krajnja. Kao što vidimo neka svojstva imaju i po nekoliko vrijednosti na raspolaganju. Uobičajena vrijednost (defaultna) je stavljena u vitičaste zagrade, dok su vrijednosti svojstava razdvojene oznakom za logički operator | (*ili*, eng. *or*). Ako se žele vidjeti tj. ispisati sve vrijednosti nekog svojstva, onda se navodi samo ime tog svojstva, uz pridruženo držalo.

```
>> set(h(4),'Marker')
        [ + | o | * | . | x | square | diamond | v | ^ | > | < | pentagram | hexagram |
{none} ]
```

Ako se želi promijeniti neka vrijednost, onda se uz ime svojstva upisuje i nova vrijednost, kako je to općom sintaksom pokazano. Na primjer:

```
>> set(h(4),'Marker','s','MarkerSize',16)
```

promijenit će izgled markera u nacrtanoj liniji u kvadratić (*square* - dovoljno je navesti samo početne, jednoznačne znakove imena : 's') i njegovu veličinu (u 16 točaka, pixela).

Sve funkcije za stvaranje objekata imaju sličan format:

```
handle = function('Ime_svojstva', 'Vrijednost_svojstva',...)
```

Moguće je u istom pozivu funkcije zadati vrijednosti za svako svojstvo, jednostavno navodeći parove *Ime_svojstva / Vrijednost_svojstva*. Funkcija vraća držalo (handle) objekta koji stvara, a s kojim kasnije možemo mijenjati bilo koje njegovo svojstvo.

Važan pojam u handle grafici je značenje trenutačnog objekta. Trenutačni (engl. *current*) objekt je zadnje načinjeni grafički objekt ili onaj na kojeg je načinjen zadnji klik mišem. Razlikuju se barem tri trenutačna objekta: otvoreni prozor na zaslonu (root), slika u okviru, te osi u slici. Unutar osi obično postoji još neki grafički objekt, kao što je linija, površina, patch ili sl. Pripadna držala spremaju se u listu, a mogu se dohvatiti s punom naredbom ili kraticom:

```
>> get(0,'CurrentFigure');     %ili
>> gcf;  % za prozor sa slikom (get current figure)

>> get(gcf,'CurrentAxes');   % ili
```

```
>> gca;  % za osi u prozoru (get current axes)

>> get(gcf,'CurrentObject');  %ili
>> gco;  %za objekt u osima (get current object)
```

Tako se naredbom:

```
>> set(gca,'XScale')
      [ {linear} | log ]
```

Za trenutačne osi vidi se da svojstvo 'Xscale' (vrsta X osi) ima dvije mogućnosti (može biti linearna ili logaritamska), dok će naredba:

```
>> set(gca,'XScale','log')
```

postaviti X-os u logaritamski prikaz, bez obzira u kakvom je bila do tada.

7.1.2 Grafički objekti i (neka) njihova svojstva

U ovom odlomku istaknut će se samo osnovna svojstva objekata i to ona koja se najčešće koriste.

7.1.2.1 Korijen (engl. *root*)

Na vrhu hijerarhijske ljestvice nalazi se jedan root (korijen) objekt koji se automatski stvara kad god se pokrene Matlab. On odgovara Matlabovom osnovnom prozoru, radnom okolišu. Njegovo držalo, identifikacijski broj je, kako znamo, uvijek jednak 0, i root nema prethodnika (roditelja), za razliku od ostalih objekata. Root objekt ne može se programski stvoriti, niti izbrisati. Sva njegova svojstva moguće je dohvatiti preko `get(0)` naredbe, a neku pojedinačnu s već opisanom sintaksom. Na primjer,

```
>> v = get (0, 'ScreenSize')
```

vratit će u vektoru v dimenzije zaslona, točnije vektor s komponentama [*left, bottom, width, height*]. Ako je u našem slučaju to [1, 1, 1024, 768] znači da radimo na zaslonu rezolucije 1024 x 728 točaka, a elementi *left* i *bottom* su uvijek jednaki 1 (ako su jedinice u točkama, pixelima). Moguće je jedinice postaviti u npr. centimetrima, pa će i vektor **v** biti drugačiji.

```
>> set(0,'Units','centimeters')
```

7.1.2.2 Slika (eng. *figure*)

Objekt slike (Figure) je svaki pojedinačni grafički prozor koji se otvara pod korijenom Matlabovog osnovnog okoliša. Ako postoji više takvih prozora, onda je jednom od njih

(zadnje stvorenom ili otvorenom) pridruženo svojstvo trenutačne slike ('CurrentFigure') čije se držalo lako dobije preko:

```
>> h=get(0,'CurrentFigure')   % ili jednostavnije:
>> h=gcf
```

S naredbom `get(gcf)` dobio bi se popis mnogobrojna svojstva tog objekta. Upotrebom *Position* svojstva na primjer, trenutačna slika željene veličine može se postaviti na zaslonu Matlabovog prozora na kojoj god želimo poziciji. Donji lijevi kut prozora slike određen je *left* i *bottom* parametrima, a veličina prozora s parametrima *width* i *height*:

```
>> set (gcf, 'Position', [left, bottom, width, height])
```

Sve funkcije koje crtaju grafiku, npr. `plot()` ili `surf()`, automatski stvaraju sliku ako ona ne postoji, ili ucrtavaju druge grafičke objekte u trenutačnu.

7.1.2.3 Osi (eng. *axes*)

Objekt *Osi* definira prostor unutar prozora slike i orijentaciju svojih sljedbenika (children) unutar tog područja. Osi su sljedbenici objekta slike, a prethodnici (roditelji) važnih objekata kao što su *slika, svjetlo, linija, mnogokut, pravokutnik, površina* i *tekst*. Ako unutar slike ima više osi, onda je uvijek jednima od njih pridruženo svojstvo trenutačnih osi (CurrentAxes), što se lako provjeri naredbom:

```
>> h=get(gcf, 'CurrentAxes')   % ili
>> h=gca
```

Promjenom trenutačne slike, mijenjaju se i trenutačne osi. Naime, općenito vrijedi da operacije na jednom objektu automatski se prenose i na sve njegove nasljednike.

MATLAB automatski određuje granice osi (XLim, YLim, Zlim), raspored markica (podioka) na osima (XTick, YTick, ZTick) i njihove oznake (XTickLabel, YTickLabel, ZTickLabel) kad god se stvara neki graf. Međutim, moguće je također, postavljanjem vrijednosti pojedinih svojstava, naknadno mijenjati bilo koju od tih veličina.

7.1.2.4 Linije (eng. *lines*)

Linijski objekti su osnovne grafičke primitive koje se koriste u stvaranju većine 2-D i 3-D crteža. Grafičke funkcije više razine (npr. `plot()`, `plot3()`, `logolog()` i sl.) stvaraju linijske objekte, brišući osi ili stvarajući novu sliku. Nasuprot tomu, temeljna funkcija `line()` jednostavno stvara određeni grafički objekt i postavlja ga unutar prethodnog objekta, tj. trenutačnih osi. Koordinatni sustav i pozicija osi definirana je u prethodnom objektu, pa ona usmjeruje i pozicionira linijski objekt.
Na primjer, ako se pozove `line()` funkcija s argumentima:

```
>> line('XData',x,'YData',y,'ZData',z,'Color','r')
```

Matlab će nacrtati crvenu liniju u trenutačnim osima koristeći zadane podatke (vektore *x*, *y* i *z*). Ako ne postoje osi ili prozor sa slikom, onda će Matlab to stvoriti automatski. Dobro je primijetiti da `line()` funkcija prihvaća parove ulaznih argumenata (PropretyName/PropertyValue) što se sintaksi više razine može pojednostavniti na:

```
>> line(x,y,z,'Color','r')
```

Ako se `line()` funkcija poziva po drugi put, onda Matlab crta drugu liniju u trenutačnim osima bez brisanja prve linije. Ovakvo ponašanje različito je od funkcija više razine, kakva je npr. funkcija plot koja briše grafičke objekte i poništava skoro sva svojstva osi.
Funkcije niske razine prvenstveno su zamišljene za primjenu unutar m-datoteka kad se želi stvoriti slika ovisna o dotadašnjim izračunima i interakciji s korisnikom. Česta naredba u takvim programima je `newplot`. Tako će

```
>> h = newplot
```

prirediti tj. postaviti stanovita svojstva objekta slike i njezinih osi, za iduće grafičke naredbe, te vratiti (h) držalo trenutačnih osi.
Grafički objekt briše se funkcijom `delete()` koristeći držalo objekta kao argument. Na primjer:

```
>> delete(gca)
```

izbrisat će trenutačne osi i sve što se ispod njih nalazi. U slučaju da se želi brisati samo neki podobjekt ili svojstvo objekta, onda se s naredbom `findobj` nalazi držalo objekta koje sa svojstvom koje se briše poziva u naredbi za brisanje. Na primjer:

```
>> line_handle = findobj('LineStyle',':');
```

naći će točkastu liniju koja se može izbrisati s pomoću `delete` funkcije:

```
>> delete(line_handle)
```

Obje naredbe moguće je povezati u jednu:

```
>> delete(findobj('LineStyle',':'))
```

7.1.2.5 Tekst (eng. *text*)

Već smo upoznali funkcije više razine za stvaranje tekst objekata. Bile su to: `title()`, `xlabel()`, `ylabel()`, `zlabel()`, `gtext()`. S funkcijama niže razine moguće je nad tekst objektom ostvarivati finije operacije. Tekst objekti su znakovni nizovi (eng. *string*). Naredbom:

```
text(x,y,'string')
```

postavlja se niz znakova unutar jednostrukih navodnika na mjesto određeno (x,y) pozicijom. U slučaju dodatne koordinate z, ova funkcija dodaje string u 3-D koordinatama. Postoji i ekvivalenta sintaksa na nižoj razini.
Promjenom svojstava *Extent*, *VerticalAlignment*, i *HorizontalAlignment* upravlja se postavljanje znakovnog niza s obzirom na poziciju dobivenu x,y(,z) koordinatama ili s pomoću vektora u *Position* svojstvu. Orijentacija tekstovnog objekta postavlja se podešavanjem *Rotation* svojstva.

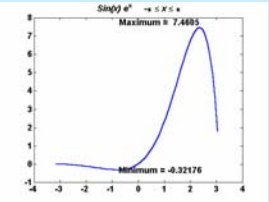
Za oblik slova (engl. *font*) koriste se sljedeća svojstva tekstovnog objekta: *FontName* (vrsta slova u kompjutorskom sustavu, npr. Helvetica, Arial, Courier,...), *FontSize* (definira veličinu u jediniciama FontUnits), *FontUnits* (jedinice za font: točke, normalizairano, inch, centimetri, pixeli), *FontWeight* (težina fonta: light, normal, demi, bold) i *FontAngle* (kut fonta: normal, italic, oblique). Tekst objekt podržava podskup TeX znakova, pa je moguće dobiti specijalne znakove kao što su grčka slova i matematički simboli.

Primjer 7.1: *Tekstualne oznake na grafu*

Zadaća je tekstovno označiti minimum i maksimum funkcije, zajedno s iznosima. M-funkcija MinMax() ima dva ulazna argumenta, prvi je držalo grafičke funkcije koja se crta, a drugi naslov koji se ispisuje nad grafom.

```
function MinMax (h,naslov)
x = get(h,'XData');    % dohvatiti podatke
y = get(h,'YData');
% naći indeks minimuma i maksimuma
imin = find(min(y) == y)
imax = find(max(y) == y)
% povezati tekst s min i max numeričkim podacima
% num2str() pretvara numerički podatak u string
text(x(imin),y(imin),...
['Minimum = ',num2str(y(imin))],...
'VerticalAlignment','middle',...
'HorizontalAlignment','left',  'FontSize',14)
```

```
imin =    25
imax =    56
```




```
ans =   -0.7416
ans =    2.3584
```

```
text(x(imax),y(imax),...
['Maximum = ',num2str(y(imax))],...
'VerticalAlignment','bottom',...
'HorizontalAlignment','right', 'FontSize',14)
title(naslov)

%poziv funkcije MinMax() iz komandne linije
>> X =-pi:0.1:pi;
>> handle=plot(X, exp(X).*sin(X))
>> MinMax(handle,'{\it Sin(x) e}^x     -\pi \leq
{\itx} \leq \pi')   %string argument napisan TeX-om
%provjera max i min na x-osi iz dobivenih rjesenja
>> X(25), X(56)
```

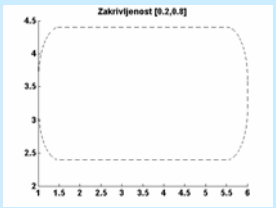
7.1.2.6 Pravokutnik (eng. *rectangle*)

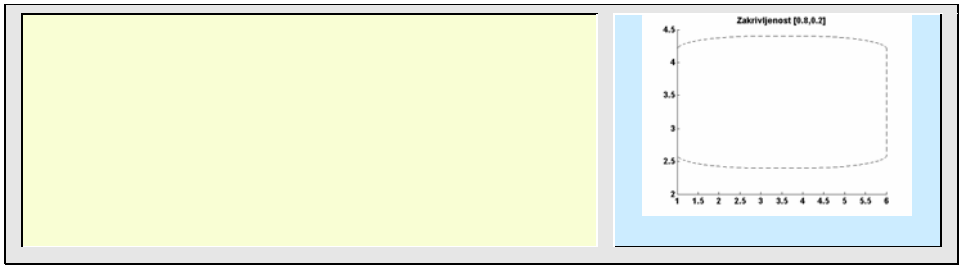
Pravokutni (Rectangle) objekt je 2-D ispunjena površina koja ima oblik koji se može mijenjati od pravokutnika do elipse, ovisno o *Curvature* svojstvu [x,y]. *Horizontalna zakrivljenost (curvature x)* je odnos širine pravokutnika koja se zakrivljuje uzduž gornjih i donjih rubova. V*erticalna zakrivljenost (curvature y*) je odnos visine pravokutnika i zakrivljenosti uzduž lijevih i desnih rubova. Vrijednosti x i y mogu se mijenjati od 0 (bez zakrivljenosti) do 1 (najveća zakrivljenost). Vrijednost od [0,0] stvara pravokutnik s stranicama kvadrata. Vrijednost od [1,1] generira elipsu. Pravokutni objekt je koristan za stvaranje crteža s dijagramima tijeka programa.

Primjer 7.2: *Zakrivljenost pravokutnika*

Treba nacrtati pravokutni objekt na poziciji (1,2) koordinate, dimenzija 5 x 2 jedinice, te sa [0.2,0.8] i [0.8,0.2] zakrivljenosti.

```
%prva slika
>> rectangle('Position',[1,2.4,5,2], 'Curvature',
[0.2, 0.8], 'LineWidth',2,'LineStyle','--' )
>> title('Zakrivljenost [0.2,0.8]')

%druga slika
>> figure(2)
>> rectangle('Position',[1,2.4,5,2], 'Curvature',
[0.8, 0.2], 'LineWidth',2,'LineStyle','--' )
>> title('Zakrivljenost [0.8,0.2]')
```





7.1.2.7 Mnogokuti (eng. *patch*)

Patch objekt je jedan ili više poligona, mnogokuta, definiranih koordinatama svojih čvorova. Poligoni mogu biti povezani ili odvojeni. Svaki patch objekt može sadržavati više lica, od kojih svako može biti nezavisno obojeno s čvrstim ili interpolacijskim bojama. Uz bojanje moguće je i osvjetljavanje mnogokuta. Mnogokuti su korisni za modeliranje objekata realnog svijeta i za crtanje 2-D i 3-D poligona bilo kojeg oblika. Među osnovnim svojstvima razlikujemo:

Tablica 7.1: *Osnovna svojstva mnogokuta*

Svojstvo	Namjena
CData	Određuje jednu, po licu ili po kutu, boju u sprezi sa x, y i z podacima.
CDataMapping	Određuje da li je podatak boje skaliran ili korišten direktno kao što je zapisano u mapi boja oblika.
FaceVertexCData	Određuje jednu, po licu ili po kutu, boju u sprezi sa podacima lica i kutova
EdgeColor	Određuje da li su rubovi nevidljivi, u jednoj boji, površinske boje određene bojama kuta, ili da li su interpolirane boje određene bojama kuta.
FaceColor	Određuje da li su lica nevidljiva, u jednoj boji, površinske boje određene bojama kuta, ili da li su interpolirane boje određene bojama kuta.
MarkerEdgeColor	Određuje boju markera ili boju ruba za ispunjene markere
MarkerFaceColor	Određuje boju ispunjenja za markere koji su zatvorenog oblika

Da bi se stvorio poligon moraju se odrediti koordinate kutova i podatak boje prema:

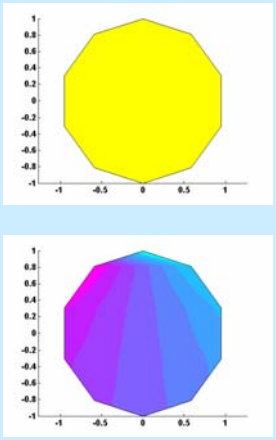
```
patch(x-coordinates,y-coordinates,[z-coordinates],colordata)
```

Primjer 7.3: *Crtanje poligona*

Prva slika prikazuju 10-erostrani poligon sa žutim licem koje zatvara crni rub.

Naredbom `axis equal` *se dobiva pravilni (proporcionalni) poligon. Prvi i zadnji kutovi ne moraju se poklapati, ali se to događa automatski. Ustvari, općenito je bolje definirati svaki kut samo jednom osobito ako se koristimo interpoliranim bojanjem lica. Moguće je, naime, upravljati mnogim aspektima patch bojanja. Primjerice, može se umjesto određivanja jedne boje osigurati raspon numeričkih vrijednosti koje prikazuju boju svakog kuta u boji koja je u mapi boja oblika, što se vidi na drugoj slici.*

```
%prva slika
>> t = 0:pi/5:2*pi ;
>> patch(sin(t),cos(t),'y')
>> axis equal

%druga slika
% mice suvisnu definiciju kuta
>> a = t(1:length(t)-1) ;
>> patch(sin(a),cos(a),1:length(a),
'FaceColor','interp')
>> colormap cool
>> axis equal
```



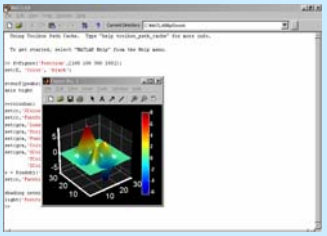
7.1.2.8 Površina (eng. *surface*)

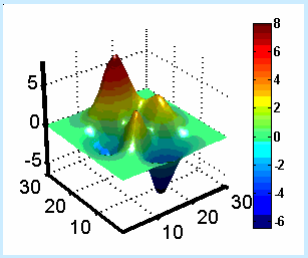
Surface objekti su 3-D reprezentacije matričnih podataka dobivenih crtanjem vrijednosti svakog matričnog elementa kao visine nad x-y ravninom. Crteži površina sastoje se od četverokuta čiji vrhovi su određeni matričnim podacima. Matlab crta površine s čvrstim ili interpolacijskim bojama ili samo s mrežom linija koja povezuje točke.
Funkcije `pcolor()`, `surf()` i skupina `mesh()` funkcija crta površinske objekte. U odnosu na mnogokute, površinski objekti su pravokutne mreže četverokuta i prikladniji su za prikaz ravninske topologije kao što su matematičke funkcije, konture podataka u ravnini ili parametrizirane površine kao što su kugle.

Primjer 7.4: *Crtanje površinskih objekata*

U ovom primjeru je prikazano crtanje površinskih objekata. Da bi se vidjelo gdje se slika nalazi na ekranu i kojih je dimenzija te kako izgleda prije pohranjivanja prikazano je na prvoj slici. Druge slika se odnosi na onu vidljivu na prvoj samo nakon pohranjivanja (save). Vidimo razliku u pozadinskoj boji.

```
>> f=figure('Position',[100 100 300 250]);
>> set(f, 'Color', 'black')

>> s=surf(peaks(30));
>> axis tight

>> c=colorbar;
>> set(c,'XColor','white','YColor','white');
>> set(c,'FontSize',10,'FontWeight','bold');
>> set(gca,'LineWidth',3);
>> set(gca,'Projection','perspective')
>> set(gca,'FontSize',14);
>> set(gca,'Color',[0.1 0.1 0.1]);
>> set(gca,'XColor','white',...
      'YColor','white',...
      'ZColor','white');
>> o = findobj('Type','surface');
>> set(o,'FaceLighting','phong')

>> shading interp
>> light('Position',[0 -1 2])
```

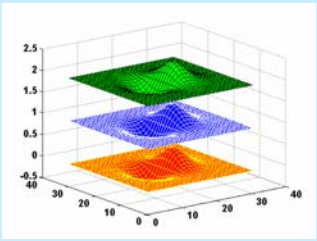




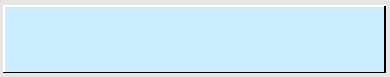
Primjer 7.5: *Crtanje površinskih objekata u odvojenim ravninama*

Da bismo 3D funkcije nacrtali na istom grafu samo u različitim ravninama potrebno je svakom površinom postupati kao zasebnim objektom i pojedinačno ih crtati funkcijom surface().

```
>> x=[-3:0.2:3];
>> y=[-3:0.2:3];
>> [X,Y]=meshgrid(x,y);
>> z1=X.*exp(-X.^2-Y.^2);
>> z2=X.*exp(-X.^2-Y.^2)+1;
>> s1=surface(z1,'FaceColor',[1 0 0]);
>> view(-35,45)
>> hold on
>> s2=surface(z2,'FaceColor',[0 1 0]);
>> s2=surface(z2,'FaceColor',[0 0 1],
'EdgeColor',[1 1 1]);
>> s1=surface(z1,'FaceColor',[1 0 0],
'EdgeColor','y');
>> z3=X.*exp(-X.^2-Y.^2)+2;
```



```
>> s3=surface(z3,'FaceColor','g');
>> view([-37,20]), grid on
```



7.1.2.9 Osvjetljenje (eng. *light*)

Objekt osvjetljenja definira izvor svjetla koje utječe na sve mnogokute (patches) i objekte površina (surfaces) unutar koordinatnih osi. Tri važna svojstva osvjetljenja su:

- *Color* – boja svjetla objekta koje osvjetljava moguće obojeni mnogokut ili površinu.
- *Style* – način osvjetljenja i to iz beskonačne udaljenosti (to je ujedno i pretpostavljena vrijednost) što znači da su zrake osvjetljavanja paralelne iz smjera zadane pozicije ili je lokalno, što znači da zrake izviru iz zadane pozicije u svim smjerovima
- *Position* – smjer (za beskonačno udaljene izvore svjetla) ili pozicija, mjesto (za lokalne izvore).

Tablica 7.2: *Svojstva za osvjetljenje objekta*

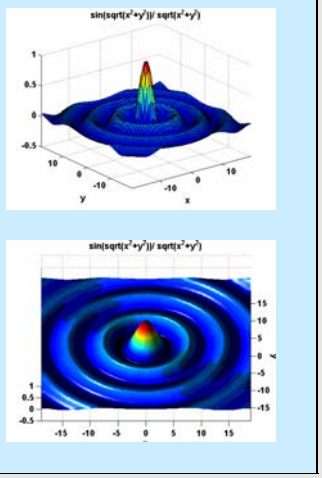
Svojstvo	Namjena
AmbientLightColor	Ovo je svojstvo osi koje zadaje boju pozadinskog svjetla u sceni, koje nema smjer i na sve objekte djeluje uniformno. Ambijentno svjetlo pojavljuje se samo ako postoje vidljivi objekti unutar osi.
AmbientStrength	Određuje intenzitet ambijentne komponente svjetla koji se odbija od objekta.
DiffuseStrength	Određuje intenzitet difuzne komponente svjetla koji se odbija od objekta.
SpecularStrength	Određuje intenzitet zrcalne (specular) komponente svjetla koji se odbija od objekta.
SpecularExponent	Određuje veličinu vidljivog zrcaljenja (specular highlight).
SpecularColorReflectance	Određuje stupanj koliko je zrcalno (specularly) reflektirano svjetlo obojeno s bojom objekta ili bojom svjetlosnog izvora.
FaceLighting	Određuje metodu koja se koristi za izračunavanje efekta svjetla na licu objekta. Mogući izbor je: bez svjetla, površinsko (flat), Gouraud, ili Phong svjetlosni algoritmi.

EdgeLighting	Određuje metodu koja se koristi za izračunavanje efekta svjetla na rubovima objekta. Mogući izbor je: bez svjetla, površinsko (flat), Gouraud, ili Phong svjetlosni algoritmi.
BackFaceLighting	Određuje kako lica objekata svijetle kada njihove normale površine izlaze iz kamere. Ovo svojstvo je korisno za razlikovanje nutarnjih i vanjskih površina nekog objekta.
FaceColor	Određuje boju lica nekog objekta.
EdgeColor	Određuje boju rubova nekog objekta.
VertexNormals	Svojstvo koje sadrži vektore normala za svako čvorište (vertex) objekta. MATLAB koristi te vektore da bi izveo proračune osvjetljenja. Dok Matlab automatski generira te podatke, korisnik također može zadati vlastite normale čvorišta.
NormalMode	Određuje da li preračunava normale čvorišta ako se promijene podaci objekta (auto) ili koristi trenutačnu vrijednost od VertexNormals svojstva (manual). Ako se zadaju vrijednosti VertexNormals, onda Matlab postavlja ovo svojstvo na manual.

Primjer 7.6: *Primjena svojstava osvjetljenja na objekt površine*

U ovom primjeru ćemo vidjeti neka od svojstava osvjetljenja objekta koja pokazuju da slika može prikazati gotovo onako kako zamislimo.

```
>> ezsurf('sin(sqrt(x^2+y^2))/
sqrt(x^2+y^2)', [-6*pi,6*pi])

%dodavanjem svjetla i promjenom kuta gledišta
>> view(0,75)
>> shading interp
>> lightangle(-45,30)
>> set(findobj(gca,'type','surface'),...
'FaceLighting','phong',...
'AmbientStrength',.3,'DiffuseStrength',.8,...
'SpecularStrength',.9,'SpecularExponent',25,...
'BackFaceLighting','unlit')
```



7.1.2.10 Slike (eng. *image*)

Matlab pruža naredbe za čitanje, spremanje i prikazivanje slika različitih grafičkih formata. Slika se prikazuje u pretpostavljenom okviru sa zadanim osima. Pritom se slika širi ili sažima kako bi stala u zadani okvir. Kao i svi grafički objekti i slika ima nekoliko svojstava čijim mijenjanjem postiže se fino ugađanje njenog pojavljivanja na zaslonu. Najvažnija svojstva slikovnog objekta s obzirom na pojavljivanje su:

- *Cdata* – koje sadrži polje s podacima
- *CdataMapping* – koji služe za prikaz, umjeravanje i preslikavanje
- *Xdata* i *Ydata* – koji upravljaju koordinatnim sustavom slike
- *EraseMode* – koji služi za osvježavanje slike na zaslonu, ako se promijenilo *Cdata* svojstvo.

Ponekad je potrebno prikazati sliku tako da svakom elementu u matrici podataka odgovara jedan pixel (točkica) na zaslonu. Pritom je nužno promijeniti objekt okvira i osi (figure i axes objekt). Na primjer:

Primjer 7.7: *Prikazivanje slike*

Svojstvo slike Position je vektor s četiri elementa koji određuju mjesto slike na zaslonu, kao i njenu veličinu.
Donji lijevi kut slike postavljen je na poziciju (100,100) na zaslonu tako da njezina visina i širina odgovaraju veličini slike. Postavljanje osi u poziciju [0 0 1 1] stvaraju se osi u normaliziranim jedinicama koje popunjavaju okvir što rezultira donjom slikom.

```
>> clear  %brisanje trenutnih varijabli iz memorije
%slika zemlje (earth.mat) ugradjena u Matlab
>> load earth
>> who %pregled učitanih varijabli
>> [m,n] = size(X);
>> figure('Units','pixels','Position',[100 100 n m])
>> image(X); colormap(map)
>> set(gca,'Position',[0 0 1 1])

%dodatak: radius planete Zemlje
>> radijus=almanac('earth','radius','kilometers')
```

```
Your variables are:
                X   map
```

```
radijus =       6371
```

7.1.3 Pretpostavljene (eng. *default*) vrijednosti

Svi objekti imaju pretpostavljene (default) vrijednosti za svoja svojstva. To su ugrađene (factory-defined) vrijednosti koje Matlab donosi s objektima. Ideja je da ako se neko objektno svojstvo ne mijenja eksplicitno, onda Matlab uzima njegovu pretpostavljenu vrijednost. Dobra zamisao je da i korisnik može promijeniti pretpostavljene vrijednosti. Ako se to načini na višoj objektnoj razini, onda će to imati većeg odjeka, budući da objekti niže razine zadržavaju svojstva objekata višeg ranga. To se postiže dodavanjem riječi *Default* ispred imena nekog svojstva kojem se želi promijeniti pretpostavljena vrijednost. Na primjer, ako se želi promijeniti pretpostavljena debljina linija na 1.5 točku i to na razini slike, figure, onda se ispred imena *LineWidth* napiše riječ *Default*, i pokrene postavljanje sa:

```
>> set(gcf,'DefaultLineLineWidth',1.5)
```

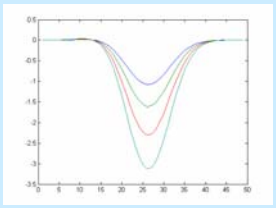
Ako se žele saznati pretpostavljene vrijednosti nekog objekta, na primjer trenutačne slike (`gcf`), onda se to dobiva već poznatom `get()` funkcijom:

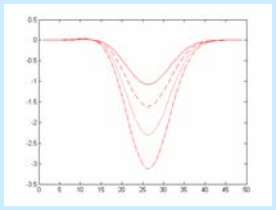
```
>> get(gcf,'default')
```

Primjer 7.8: *Pretpostavljene (defaultne) vrijednosti svojstava objekta*

Tvornički pretpostavljene vrijednosti za crtanje krivulja na grafu su takve da se crtanje više krivulja na jednom grafu funkcijom plot() ostvaruje različitim bojama. Ako bismo htjeli da svaka krivulja na grafu bude u istoj boji ali prikazana različitim stilovima onda moramo promijeniti defaultne vrijednosti kako je prikazano u ovom primjeru.

```
%prva slika
>> Z = peaks; plot(1:49,Z(4:7,:))

%[ 1 0 0] je crvena boja
>> set(0,'DefaultAxesColorOrder',[1 0 0],...
'DefaultAxesLineStyleOrder','-|--|:|-.')

%druga slika
>> Z = peaks; plot(1:49,Z(4:7,:))
```





Moguće je dakako i brisanje pretpostavljenih vrijednosti sa 'remove' argumentom, ali i vraćanje na tvorničke vrijednosti sa 'factory' argumentom, na primjer:

```
>> set(0,'DefaultSurfaceEdgeColor','remove')
```

odnosno:

```
>> set(0,'DefaultSurfaceEdgeColor','factory')
```

7.2 GUI – Graphical User Interface: korisničko grafičko sučelje

Korisničko grafičko sučelje omogućuje izradu takvih programa za koje korisnik neće morati učiti programiranje, niti složene postupke njihove upotrebe: dovoljno je tipkom miša kliknuti na gumb, upisati podatke u okvir, izabrati neku ponudu i sl. GUI predstavlja samo grafički okoliš ugodan čovjeku za komunikaciju sa strojem, a ne i program koji rješava neki problem. Postoji veza GUI s programom za koji je okoliš izgrađen. To se postiže definiranjem pozivnih rutina (eng. *callback routines*) kao vrijednosti svojstava posebnih UI objekata. Već su u trećoj razini handle grafike nabrojeni su ti GUI objekti. To su: *Uicontrol*, *Uimenu*, *Uicontextmenu*.

Uicontrol su objekti za kontrolu kao što su gumbi, liste, tekst okviri, mjerila i sl. Svaki objekt očekuje neku interakciju s korisnikom, podatak, izbor ili klik mišem.

Uimenu objekti su padajuće izborne ponude koje izvršavaju pozivne rutine na temelju korisnikovog izbora. *Uimenu* i *Uicontrol* su u hijerarhijskoj ljestvici sljedbenici slika (figure) i prema tomu su neovisni o osima.

Uicontextmenu sa sintaksom

```
Uicontextmenu('svojstvo_1', vrijednost_1, 'svojstvo_2', vrijednost_2, 'svojstvo_3', vrijednost_3,...)
```

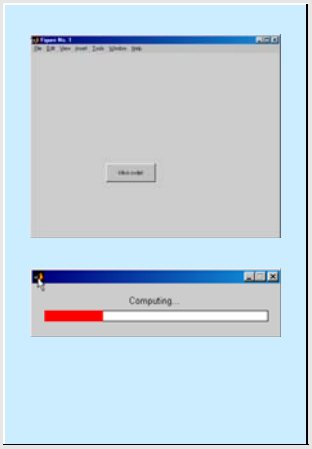
stvara držalo koje se koristi u funkciji `uimenu()`, kako bi se izgradio sustav izbornih ponuda sa zadanim svojstvima.

Osim nabrojenih objekata, postoji velik broj Matlab funkcija koje su s njima povezane, od različitih dijaloških okvira, radnih rastućih vrpci, okvira za rad s datotekama i sl. Najbolje je sve funkcije i objekte upoznati na jednostavnim primjerima.

Primjer 7.9: *Prikaz gumba (button), te grafički prikaz tijeka procesa*

Pomoću GUI-a stvorit će se gumb u sredini prozora sa slikom i vratiti držalo na novi objekt. Treba primijetiti i razumjeti svaki od Svojstvo/Vrijednost parova.
Za grafički prikaz trajanja izvršavanja programskog koda može se koristiti funkcija waitbar()

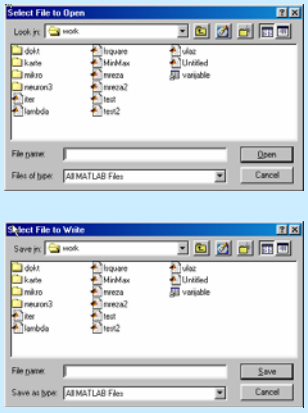
```
%prva slika
>> b = uicontrol('Style','pushbutton',...
'Units','normalized', 'Position',...
[.3 .3 .2 .1], 'String','klikni ovdje!');

%druga slika
>> h=waitbar(0,'Computing...');
% najčešće se koristi u programima:
>> n=20000;
>> for j=1:n
    %neki proračun ...
   waitbar(j/n)
  end
>> close(h)
```



Primjer 7.10: *Otvaranje dijaloških okvira*

Funkcijom uigetfile() se otvara dijaloški okvir za otvaranje datoteka, dok se funkcijom uiputfile() otvara dijaloški okvir za spremanje datoteka

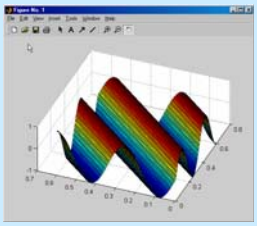
```
%prva slika
>> uigetfile

%druga slika
>> uiputfile
```

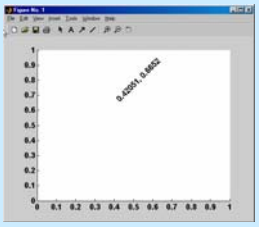


Primjer 7.11: *Rotacija objekata na grafu*

U slučaju da hoćemo postaviti objekt na grafu tako da ga možemo mišem 3D rotirati (prva slika) koristimo naredbu `rotate3d`.
Izbor koordinate (klikom miša na graf) na kojoj se sprema tekst (x,y) koordinate, rotiran za 45° prikazan je na drugoj slici.

```
%prva slika
>> x = 0:.01:.6;
>> [X, Y] = meshgrid(x,x);
>> Z = sin(X^2 - Y^2);
>> surf(X, Y, Z);
>> rotate3d on;

%druga slika
>> [x y] = ginput(1);
>> txthandle = text(x, y, [num2str(x) ', '
num2str(y)]);
>> set(txthandle, 'rotation', 45);
```

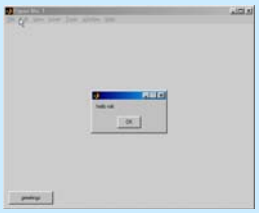




Primjer 7.12: *Interakcija među grafičkim prozorima*

U ovom primjeru stvara se gumb (pushbutton) s natpisom 'greetings'. Klikom na njega poziva se (callback) metoda 'msgbox("hello mik")' koja u pop-up prozoru ispisuje poruku koja se zatvara klikom na OK.

```
>> uicontrol('style', 'pushbutton', ...
      'parent', figure, ...
      'string', 'greetings', ...
      'callback','msgbox("hello mik")', ...
      'position', [10 10 100 30]);
```



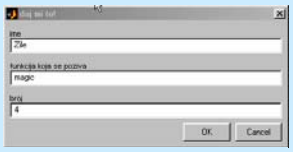
Primjer 7.13: *Interakcija grafičkog sučelja s korisnikom*

U ovom primjeru poziva se ulazni dijaloški okvir s tri polja u kojima su pretpostavljene vrijednosti za ime, funkciju i ulazni argument. Korisnik može

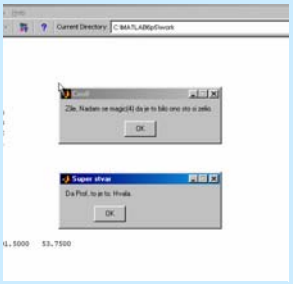
mijenjati bilo koji od podataka. Klikom na OK, izvodi se funkcija (eval(mycall)) sa ulaznim argumentom (broj) i na koncu ispisuje pop-up okvir s prikladnom porukom. Zadana akcija ispisuje izračunate rezultate u Matlab okolišu, a na temelju ulaznih podataka.

```
>> prompt={'ime', 'funkcija koja se poziva', 'broj'}
>> defans={'Zile', 'magic', '4'}
>> fields = {'name','func', 'num'}
>> info = inputdlg(prompt, 'daj mi to!', 1,
defans)

>> if ~isempty(info)    %pritisnut cancel?

   info = cell2struct(info,fields)
   myname = info.name
   myfunc = info.func
%pretvorba stringa u broj
   mynum = str2num(info.num)
   mycall = [myfunc '(' num2str(mynum) ')']
   eval(mycall)              %racunanje izraza
   h1=msgbox([myname ', Nadam se ' mycall ' da
je to bilo ono sto si zelio.'], 'Cool!')

>> end

>> msg1=get(h1,'Position') %polozaj i dimenzije
>> h2=msgbox('Da Prof, to je to. Hvala. ','Super
stvar')
>> set(h2,'Position',[msg1(1),msg1(2)-2*msg1(4),
msg1(3),msg1(4)]) %polozaj malo ispod msg1
```

```
prompt =
'ime'    [1x23 char]    'broj'
defans =
'Zile'    'magic'    '4'
fields =
'name'    'func'    'num'
info =  'Zile'
        'magic'
        '4'
info =  name: 'Zile'
        func: 'magic'
        num: '4'
myname = Zile
myfunc = magic
mynum = 4
mycall = magic(4)
ans =  16     2     3    13
        5    11    10     8
        9     7     6    12
        4    14    15     1
h1 =    3.0082
```



```
msg1 =
199.2  264.1  201.5  53.7
h2 =    7.0039
```



Primjer 7.14: *Izrada GUI predloška*

U ovom primjeru postupno se pokazuje izrada predloška (engl. layout), GUI okoliša. Postupak je da se problem prvo zamisli na papiru, a onda korak po korak izvodi na zaslonu. Od većih cjelina prema manjim detaljima. Na koncu izgrađuju se Matlab funkcije koje se povezuju (callback) sa željenim kontrolama, gumbima, kazalima i sl.

```
%prva slika
>> set(gcf,'DefaultUIcontrolUnits','Normalized')
>> frame1_=uicontrol(gcf,'Style','Frame',
'Position',[0.1 0.1 0.8 0.3]);
>> frame2_=uicontrol(gcf,'Style','Frame',
'Position',[0.1 0.6 0.8 0.3]);
>> set(frame1_,'BackgroundColor',[0.5 0.5 0.5]);
>> set(frame2_,'BackgroundColor',[0.5 0.5 0.5]);
%druga slika
>> text_f_=uicontrol(gcf,'Style','Text',...
 'String', 'Farenheit:','Position',[0.3 0.7 0.2 0.05],...
'HorizontalAlignment','Left');
>> edit_f_=uicontrol(gcf,'Style','Edit','String',...
 '68.0','Position',[0.6 0.7 0.1 0.05],...
'HorizontalAlignment','right','Callback','fc_calc');
>> text_c1_=uicontrol(gcf,'Style','Text',...
'String', 'Celsius:','Position',[0.3 0.3 0.2 0.05],...
'HorizontalAlignment','Left');
>> text_c2_=uicontrol(gcf,'Style','Text',...
'String', '20.0','Position',[0.6 0.3 0.1 0.05],...
'HorizontalAlignment','Right');
%treca slika, prikazan je samo dio gornjeg dijela
>> slider_f=uicontrol(gcf,'Style','Slider',...
'Min', 32.0,'Max',212.0,'Value',68.0,'Position',...
[0.6 0.8 0.2 0.05], 'Callback','fc_slider; fc_calc');
```

Iako je moguće razviti GUI primjene koristeći samo funkcije niske razine Matlab pruža mogućnost grafičkog sučelja za stvaranje GUI programa. To je GUIDE program (u komandnu liniju se upiše `guide`), koji se poziva iz Matlabovog naredbenog sučelja. GUIDE uključuje izborne ponude za generiranje zaslona, aranžiranje objekata u prozoru, podešavanje udaljenosti i pozicija među objektima, padajuće izborne ponude i sl. Posebno mjesto u GUIDE programu zauzima editor svojstava (Property Inspector) gdje se grafičkim putem mogu promatrati, zadavati i mijenjati vrijednosti svojstava bilo kojeg objekta.

7.3 Animacija

Postoje dva načina na koji Matlab može prikazati animaciju:

- Spremiti određen broj različitih sličica i prikazivati ih u nizu kao film
- Kontinuirano brisati i ponovo crtati objekte na ekranu, te pri svakom ponovnom crtanju praviti određene pomake ili promjene onog što se prikazuje

Animacija u Matlabu temelji se na više različitih funkcija koje je omogućuju.

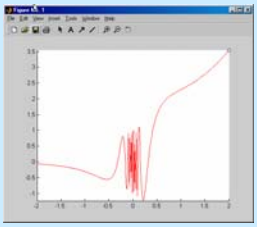
Tablica 7.3: *Funkcije za animaciju*

Funkcija	Namjena
`moviein()`	Inicijalizira memoriju za okvire ili sličice animacije
`getframe()`	Uzima filmsku sličicu i sprema je u memorijsko polje
`movie()`	Igra zapisane filmske sličice jednu za drugom
`rotate()`	Rotira objekt oko zadanog ishodišta i smjera
`frame2im()`	Pretvara film u indeksirane slike
`im2frame()`	Pretvara indeksirane slike u filmski format

Primjer 7.15: *Animacija preko ugrađene funkcije - comet()*

Funkcija comet() pokazuje tijek izvođenja neke putanje, gdje se točka (plavi kružić) giba po formulom zadanoj putanji određenom brzinom.

```
>> x=linspace(-2,2,500);
>> y=exp(x) .* sin(1./x);
>> comet(x,y)
```



Animacija u Matlabu preko funkcije `movie()` temelji se na principu rada filma. U filmu se određenom brzinom pojavljuje sličica iza sličice, pa ako je frekvencija podešena ljudskom oku (npr. 25 sličica u minuti) onda nemamo osjećaj treperenja, nego kontinuiranog igranja filma. Film se u Matlabu sprema kao niz sličica, okvira (eng. *frames*). Sličica je graf ili slika koja se sprema kao matrica ili polje. Svaka filmska sličica zauzima jedan stupac matrice. Sintaksa funkcije je:

`movie(`**m**, n, fps)

gdje je:

- **m** – matrica koja sprema sve sličice, okvire
- n – koliko broj puta se film ponavlja
- fps – broj prikazanih sličica po sekundi

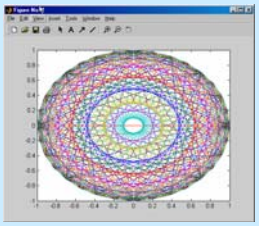
Naredba `moviein`(n) stvara matricu koja će spremiti n sličica filma, dok će m(:,j)=`getframe` uzimati iz koordinatnih osi sliku objekta i spremati je kao pojedinačnu sličicu u j-ti stupac matrice.

Na taj način, moguće je sa 16 sličica pokazati jednostavan film stvaranja različitih grafova nastalih FFT transformacijom (fast fourier transformation) nad jediničnom matricom:

Primjer 7.16: *Prikaz animiranog filma, slika na sliku*

Nakon spremanja sličica pomoću `getframe` *u matricu* **m**, *izvrtit će se jedan put svih 16 sličica s prikazom od 2 sličice u sekundi pomoću funkcije movie(). Na slici prikazan je samo konac filma.*

```
>> m=moviein(16); %stvaranje matrice
>> for j=1:16
      plot(fft(eye(j+16))); %slicice
      m(:,j)=getframe; %punjenje matrice
>> end

%nakon sto su slicice spremljene pokrenu se u film
>> movie(m,1,2)
```



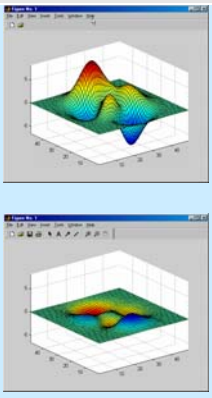
Primjer 7.17: *Slika na sliku, drugi primjer*

Ovim načinom u polje F sprema se funkcijom getframe() određeni broj slika (u ovom slučaju 11). Nakon što su slike spremljene, pozivaju se s funkcijom movie(), koja ih prikazuje poput filma s pretpostavljenim (default) vrijednostima. Na sličan način generira se i izvodi i sljedeći film. Prikazana su samo dva uhvaćena trenutka (prva i druga slika).

```
>> clear  % Brisanje postojecih varijabli
>> Z=peaks; surf(Z)
>> axis tight
>> set(gca,'nextplot','replacechildren')
>> disp('Creating the movie...')
>> for j=1:11
```

```
Creating the movie...
Playing the movie...
```

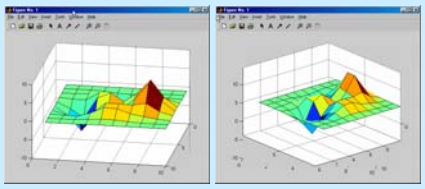
```
      surf(cos(2*pi*(j-1)/10) .* Z,Z)
      F(j)=getframe;
>> end
>> disp('Playing the movie...')
>> movie(F)
```



Primjer 7.18: *Slika na sliku, treći primjer*

U ovom primjeru treba primijetiti upotrebu promjene točke gledišta na isti objekt (funkcija view()). Prikazana su samo dva uhvaćena trenutka (prva i druga slika).

```
>> M=moviein(10);
>> set(gca, 'NextPlot',
'replacechildren')
>> for j=1:10
      surf(peaks(10))
      view([j*10 30])
      M(:,j) = getframe;
>> end
>> movie(M,4);
```

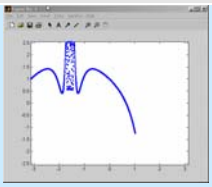


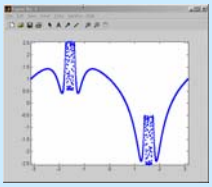
Primjer 7.19: *Crtanje točke po točku*

Ovim načinom crta se točka po točka u okviru, a okvir se pritom drži 'zaleđen' (hold on). Točka se crta promjenom vrijednosti svojstava (x,y) parova - set(p,'Xdata',x(i),'Ydata',y(i)). Prikazana su samo dva uhvaćena trenutka (prva i druga slika).

```
>> x=linspace(-pi,pi,2000);
>> y=cos(tan(x)) - tan(sin(x));
>> p=plot(x(1),y(1),'.','EraseMode','none',...
'MarkerSize',5);
>> axis([min(x) max(x)  min(y)  max(y)])
>> hold on
```

```
>> for i=2:length(x)
       set(p,'Xdata',x(i),'Ydata',y(i))
       drawnow
>> end
>> hold off
```



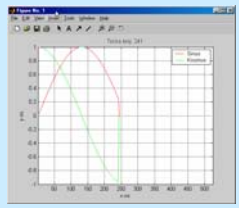


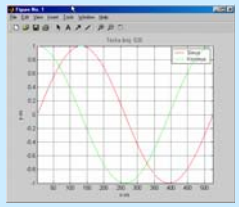
Postoji još načina na koji se može ostvariti animacija. Najrobusnija je metoda da se svaki okvir sprema na disk, koristeći `imwrite()` naredbu, a onda se upotrijebi vanjski program (npr. dmconvert ili sl.) da se od slika načini MPEG ili neka druga animacija. Od najfinijih je pak ona najniže razine.

Primjer 7.20: *Animacija na niskoj razini programiranja*

U ovom primjeru se želi pokazati kako je programiranje animacije na niskoj razini vrlo efikasno u smislu kontrole objekata, te brzine izvođenja animacije. Prikazana su samo dva uhvaćena trenutka (prva i druga slika).

```
%prikaz sinusne i kosinusne animacije
>> elementi=525; %broj elemenata polja
%inicijalizacija pocetnog stanja sinusa
>> Y10=zeros(elementi,1); Y1=Y10;
%inicijalizacija pocetnog stanja kosinusa
>> Y20=zeros(elementi,1);
>> Y20(1)= 1; %stanje prve tj. pocetne tocke
>> Y2=Y20;

>> clf  %brisanje postojece slike
>> handleSin=plot(Y10,'r-');
%postavlja drzalo za sliku
>> hold on, grid on
>> axis tight  %osi u rangu vrijednosti podataka
>> handleCos=plot(Y20,'g-');
%postavlja drzalo za naslov
>> handleNaslov=title('Tocka broj 0');
>> legend('Sinus','Kosinus');
```





```
>> xlabel('x-os'); ylabel('y-os');

%postavlja svojstvo brisanja na xor
>> set(handleSin,'EraseMode','xor');
>> set(handleCos,'EraseMode','xor');
>> set(handleNaslov,'EraseMode','xor');

>> for ii=2:elementi
     Y1(ii)=sin((ii-1)*2*pi/(elementi-1));
     Y2(ii)=cos((ii-1)*2*pi/(elementi-1));
%promjena samo Y vrijednosti tocaka
     set(handleSin,'YData',Y1);
     set(handleCos, 'YData',Y2);
     set(handleNaslov,'String',['Tocka broj: '
num2str(ii)]); %promjena naslova
     drawnow
>> end
```

Naredba

```
>> handleSin=plot(Y10,'r-');
```

ne samo da crta crtež, nego i sprema držalo grafičkog objekta u varijabli *handleSin*. Kasnije je u skript datoteci moguće koristiti držalo kako bi se mijenjale ostala svojstva grafa. Da bi se osvježavali novim vrijednostima samo Y podaci u grafu, potrebno je izvesti naredbu:

```
>> set (handleSin,'EraseMode','xor')
```

koja navodi Matlab kako će izbrisati liniju, tj. navodi Matlab da pri promjeni koordinata određene prikazane točke ne briše sve elemente te sliku (figure) nego da obriše samo taj objekt (tj. točku) . Stvarni trik u animaciji dolazi s naredbom:

```
>> set(handleSin,'YData',Y1)
```

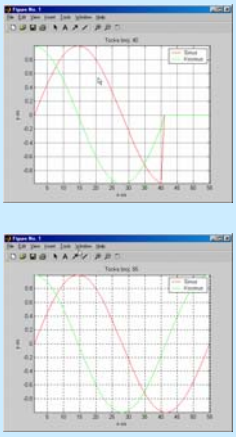
To mijenja samo Y vrijednosti u grafu s držalom *handleSin*. Koristeći ovu metodu program jednostavno precrtava liniju bez ponovnog crtanja koordinatnih osi, naslova, legende i sl. To je i razlog da se takva animacija događa puno brže, nego kao što bi bila ova u nastavku:

Primjer 7.21: *Animacija na višoj razini programiranja*

U ovom primjeru se želi pokazati kako je programiranje animacije na višoj razini

efikasno u smislu pojednostavljenja programskog koda, ali je brzina izvođenja animacije znatno manja nego za programiranje istog na nižoj razini programiranja. Prikazana su samo dva uhvaćena trenutka (prva i druga slika).

```
%prikaz sinusne i kosinusne animacije
>> elementi=55; %broj elemenata polja
%inicijalizacija pocetnog stanja sinusa
>> Y10=zeros(elementi,1); Y1=Y10;
%inicijalizacija pocetnog stanja kosinusa
>> Y20=zeros(elementi,1);
>> Y20(1)= 1; %stanje prve tj. pocetne tocke
>> Y2=Y20;

>> for ii=2:elementi
     Y1(ii)=sin((ii-1)*2*pi/(elementi-1));
     Y2(ii)=cos((ii-1)*2*pi/(elementi-1));
     clf  %brisanje postojece slike
     plot(Y1,'r-');
%postavlja drzalo za sliku
     hold on, grid on
     axis tight  %osi u rangu vrijednosti podataka
     plot(Y2,'g-');
%postavlja drzalo za naslov
     title(['Tocka broj: ' num2str(ii)]);
     legend('Sinus','Kosinus');
     xlabel('x-os'); ylabel('y-os');
     drawnow
>> end
```



U svim animacijama pojavljuje se naredba `drawnow` koja ima jedno vrlo korisno svojstvo, a to je da svakim prolaskom kroz petlju, tj. svakim crtanjem slike, slika se prikaže i zadrži na ekranu dok ju ne prekrije nova slika. Da te naredbe nema onda bi za vrijeme učitavanja i prikazivanja pojedinačnih slika, prozor (figure) u kojem se prikazuje slika bio bi prazan, te bi se tek na kraju prikazala zadnja slika.

NUMERIČKA I SIMBOLIČKA MATEMATIKA

8

Matlab je izvrstan alat za rješavanje numeričkih problema. Tek dodatkom simboličkog toolbox-a iz Mapple programa počelo je njegovo približavanje simboličkoj matematici. Istaknuta područja numeričke matematike svakako su traženja nultočki funkcija (korjenovanje), interpolacija, diferencijalni i integralni račun, te numerička rješavanja običnih diferencijalnih jednadžbi. Dakako, to nisu jedina područja numeričke matematike i Matlab-ovih funkcija koje ih podupiru. Ovdje su obrađena samo najznačajnija. Mnoga područja pokrivena su posebnim toolbox-ima, npr. optimizacija, spline, parcijalne diferencijalne jednadžbe, obrada signala, obrada slike i sl.

8.1 Traženje nultočki funkcija

Problem je sljedeći: uzimajući u obzir funkciju realne vrijednosti $\mathbf{f}: \Re^{\mathbf{n}} \to \Re^{\mathbf{n}}, \mathbf{n} \geq 1$, treba naći vektor **r** takav da je f(**r**) = **0**. Vektor **r** zove se korijen ili nula od **f**. Proširenje problema traženja nultočki vodi na vektorske funkcije. Pretpostavlja se da su parcijalne derivacije prvog reda vektorskih funkcija **f** kontinuirane na otvorenom intervalu domene funkcije i sadrže sve nultočke od **f** .

8.1.1 Korjenovanje

Već je u poglavlju o polinomima pokazano kako se s pomoću funkcije `root()` traži korijen polinoma. Međutim, točnost izračunatog korijena predstavlja velik problem.

Primjer 8.1: *Dobivanje nultočki funkcije traženjem korijena polinoma*

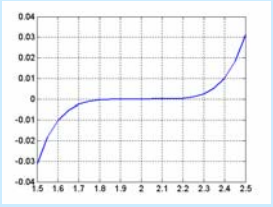
Neka se na primjer promotri polinom 5 stupnja:
$p(x) = x^5 - 10x^4 + 40x^3 - 80x^2 + 80x - 32$ koji je ustvari $p(x) = (x - 2)^5$.
*Koristeći funkciju roots() dobiju se rezultati koji nisu zadovoljavajući. Točno rješenje je jedan korijen petog reda u x=2. Da bi se to izračunalo potrebno je koristiti funkciju fzero() koja će naći višestruki korijen od p(**x**). Potrebno je definirati funkciju s polinomom, npr. peti r.m*

```
>> p=[1 -10 40 -80 80 -32]
```

```
p =  1   -10    40   -80    80   -32
```

```
>> korijeni = roots(p)

%M-datoteka, peti_r(x).m
function y = peti_r(x)
% Funkcija s polinomom petog reda
y = (x - 2)^5;


>> korijeni = fzero('peti_r', 1.5)
%crtanje polinoma
>> x=1.5:0.05:2.5;
>> vr=polyval(p,x);
>> plot(x,vr)
>> grid
```

```
korijeni =
    2.0020 + 0.0014i
    2.0020 - 0.0014i
    1.9992 + 0.0023i
    1.9992 - 0.0023i
    1.9975
korijeni =     2.0000
```



Upotrebom funkcije `fzero()` u većini slučajeva dobiju se točniji rezultati. Na primjer, razvijeni oblik istog polinoma ne bi dao potpuno točan rezultat. Treba također primijetiti da funkcija zahtjeva od korisnika pretpostavljanje mjesta korijena, početnu točku oko koje se očekuje rješenje (u ovom slučaju pretpostavljena je vrijednost 1.5). Postoji mogućnost zadavanja više različitih argumenata `fzero()` funkcije.

Primjer 8.2: *Traženje nultočki funkcije*

U ovom primjeru neka je f(x) = cos(x) – x. Prvo se definira funkcija y = fcos(x). Pripravom formata i pozivom te funkcije dobije se nultočka u okolini apscise 0.5 Rezultat se može se provjeriti koristeći funkciju feval().
Moguće je također pretpostavljenu vrijednost nultočke zadati u obliku dvoelementog vektora u kojem prvi element predstavlja početnu, a drugi konačnu vrijednost pretpostavljenog raspona mogućih vrijednosti. U slučaju loše izabranog intervala prikazuje se poruka o pogrešci (vrijednosti funkcije na krajevima intervala moraju se razlikovati u predznaku, inače ne može postojati nultočka). Dodavanjem trećeg ulaznog parametra 'optimset' može se izračunati nultočka funkcije s različitim mogućnostima, na primjer ispisom iteracijskog postupka:

```
function y = fcos(x)
%Funkcija s jednostrukom
nultockom
y = cos(x) - x;

%poziv funkcije fcos()
>> format long
>> format compact
```

```
k =   0.73908513321516
pogreska =     0
k_raspon =     0.73908513321516

??? Error using ==> fzero
The function values at the interval
endpoints must differ in sign.
Func-count     x          f(x)         Procedure
```

```
>> k = fzero('fcos', 0.5)

%provjera rezultata
>> pogreska = feval('fcos', k)

%nultocka unutar raspona
>> k_raspon = fzero('fcos', [0 1])
%U slučaju loše izabranog intervala:
>> k_raspon = fzero('fcos', [1 2])

%dodavanje optimset -a
>> rt = fzero('fcos', .5,
optimset('disp','iter'))

%Moguće je također postaviti mjeru
%tolerancije, kod koje se iteracijski
%postupak završava, na primjer:

>> rt1 = fzero('fcos', .5,
optimset('disp','iter','TolX',0.05))
```

```
  1        0.5      0.37758     initial
  2        0.48     0.39841     search
  3        0.51     0.35657     search
  4        0.48     0.40699     search
  5        0.52     0.34781     search
  6        0.47     0.41907     search
  7        0.52     0.33538     search
  8        0.46     0.43605     search
  9        0.54     0.31770     search
 10        0.44     0.45985     search
 11        0.55     0.29250     search
 12        0.42     0.49308     search
 13        0.58     0.25646     search
 14        0.38     0.53923     search
 15        0.61     0.20471     search
 16        0.34     0.60275     search
 17        0.66     0.12999     search
 18        0.27     0.68904     search
 19        0.72     0.02137     search
 20        0.18     0.80384     search
 21        0.82    -0.13777     search
Looking for a zero in the interval [0.18,
0.82]
 22        0.72     0.0212455   interp
 23        0.73     0.0003671   interp
 24        0.73    -6.042e-8    interp
 25        0.73     2.927e-12   interp
 26        0.       0           interp
rt =   0.73908513321516

Func-count     x       f(x)        Procedure
  1            0.5     0.37758     initial
  2            0.48    0.39841     search
…
 22            0.72    0.03180     interp
Zero found in the interval: [0.18, 0.82].
rt1 =   0.72000000000000
```

```
>> help optimset
```

dobije se popis svih mogućnosti trećeg argumenta.

8.1.2 Newton – Raphson-ova metoda za rješavanje sustava nelinearnih jednadžbi.

Problem traženja nultočki vektorskih funkcija odgovara rješavanju sustava nelinearnih jednadžbi. Bit će opisana metoda poznata pod imenom *Newton-Raphson-ova metoda.*
Neka stupčasti vektor **f** sadrži vektorske funkcije $\mathbf{f = [f_1; \ldots ; f_n]}$ gdje je svaka f_k funkcija od $\Re^{\mathbf{n}}$ na $\Re$ Postavljajući početnu aproksimaciju $\mathbf{x}^{(0)} \in \Re^{\mathbf{n}}$ od vektora rješenja **r,** ova metoda stvara niz vektora $\{\mathbf{x}^{(k)}\}$ koristeći iteraciju

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J_f}\,(\mathbf{x}^{(k)})^{-1}\, f(\mathbf{x}^{(k)}), \quad k = 0, 1, \ldots$$

gdje $\mathbf{J_f}$ predstavlja Jakobijevu matricu od **f** za koju vrijedi:

$$\mathbf{J_f}\,(\mathbf{x}) = \left\lfloor \partial \mathbf{f_i}\,(\mathbf{x}) / \partial \mathbf{x_j} \right\rfloor,\ 1 \le \mathbf{i}, \mathbf{j} \le \mathbf{n}$$

Primjer 8.3: *Rješavanje sustava nelinearnih jednadžbi*

Funkcija NR() izračunava nultočke sustava nelinearnih jednadžbi.
Nultočka r nelinearnog sistema jednadžbi f(x) = 0. J je Jacobijeva matrica od f a x0 je početna pretpostavka nultočke r. Izračunavanje se prekida ako je norma od f kod trenutačne aproksimacije manja (po iznosu) nego broj tol, ili ako je relativna pogreška dvije uzastopne aproksimacije manja nego je pogreškom točno propisano, ili ako je načinjen broj zadanih iteracija. Izlazni parametar 'niter' predstavlja broj obavljenih iteracija.
U ovom primjeru je zadan nelinearni sustav:

$$f_1(x) = x_1 + 2x_2 - 2,\ f_2(x) = x_1^2 + 4x_2^2 - 4$$

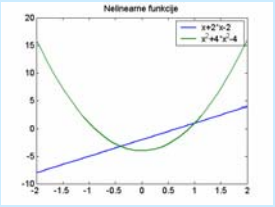
za koji znamo da ima nultočke u $r = [0\ 1]^T$ *i* $r = [2\ 0]^T$. *Funkcije fun1() i J1() definiraju sustav jednadžbi i njihove Jacobijane. Za loše odabranu procjenu početne nultočke Jacobijeva matrica je singularna (u kôdu programa to se ispituje pozivom funkcije rcond()). Promjenom početne procjene dobije se točan rezultat i broj potrebnih iteracija da se to ostvari.*

```
function [r, niter] = NR(f, J, x0, tol, maxiter)
Jc = rcond(feval(J,x0));
if Jc < 1e-10
  error('Pokušajte zadati novu početnu
aproksimaciju x0')
end
  xold = x0(:);
  xnew = xold - feval(J,xold)\feval(f,xold);
for k=1:maxiter
  xold = xnew;
  niter = k;
```

```
x0 =     0     0
??? Error using ==> nr
Pokušajte zadati novu
pocetnu aproksimaciju
x0

x0 =     1     0
r1 =    2.0000
         0.0000
niter1 =     6

x0 =    -2     4
```

```
  xnew = xold - feval(J,xold)\feval(f,xold);
if (norm(feval(f,xnew)) < tol) |...
  norm(xold-xnew,'inf')/norm(xnew,'inf') < tol|...
  (niter == maxiter)
break,end,end
r = xnew;

function z = fun1(x)
% sustav jednadzbi
z = zeros(2, 1);
z(1) = x(1) + 2*x(2) - 2;
z(2) = x(1)^2 + 4*x(2)^2 - 4;

function s = J1(x)
% parcijalne derivacije prikazane matricno
s = [1 2;2*x(1) 8*x(2)];

%Pozivom funkcije dobije se upozorenje za losu
%procjenu pocetne nultocke:
>> x0 = [0 0]
>> [r, niter] = NR('fun1', 'J1', x0, eps, 10)
%Promjena pocetnih uvjeta
>> x0 = [1 0]
>> [r1, niter1] = NR('fun1', 'J1', x0, eps, 10)
%Za drugačije početne uvjetne pretpostavke moguće
%je dobiti i drugo rješenje
>> x0=[-2 4]
>> [r2, niter2] = NR('fun1', 'J1', x0, eps, 10)
%uvrštenjem u početnu jedn. potvrđuje ispravnost
>> F=feval('fun1', r2)
%crtanje grafa
>> fplot('[x+2*x-2,x^2+4*x^2-4]',[-2 2])
>> title('Nelinearne funkcije' )
>> legend('x+2*x-2','x^2+4*x^2-4')
```

```
r2 =    0.0000
        1.0000
niter2 =     6

F =     0
        0
```



8.2 Interpolacija

Interpolacija funkcije je jedan od klasičnih problema numeričke analize. Problem jednodimenzionalne interpolacije formuliran je ovako: za zadanih n+1 točaka $\{x_k, y_k\}$, $0 \le k \le n$, sa $x_0 < x_1 < \ldots < x_n$, potrebno je naći funkciju f(x) čiji graf interpolira zadane točke, tj. $f(x_k) = y_k$, za k = 0, 1, …, n.

8.2.1 Matlab-ova funkcija interp1()

Općeniti oblik funkcije `interpl()` je

yi = `interp1` (**x**,**y**,**xi**, metoda)

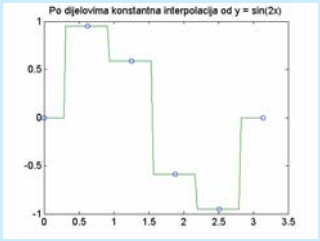
gdje su vektori **x** i **y** vektori spremišta koordinata točaka koje se interpoliraju, **xi** je vektor točaka koje se računaju, tj. **yi** = f(**xi**), a *metoda* je string koji određuje interpolacijsku metodu. Postoji više metoda koje nudi `interpl` funkcija:

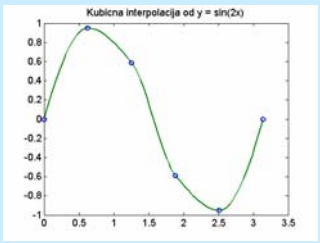
- Interpolacijska metoda najbližeg susjeda - metoda = 'nearest' .
- Metoda linearne interpolacije - metoda = 'linear'.
- Kubna spline interpolacija - metoda = 'spline' .
- Kubna interpolacija - metoda = 'cubic'.

Primjer 8.4: *Interpolacija podataka*

Za točke (x_k, y_k) = ($k\pi/5$, $\sin(2x_k)$), k = 0, 1, ..., 5, koriste se dvije metode interpolacije 'nearest' i 'cubic'. Interpolacija je izračunata za niz točaka xi.

```
>> x = 0:pi/5:pi;
>> y = sin(2.*x);

%racunanje medjutocaka
>> xi = 0:pi/100:pi;
>> yi = interp1(x, y, xi, 'nearest');

%crtanje tocaka kod 'nearest' interpolacije
>> plot(x, y, 'o', xi, yi)
>> title('Po dijelovima konstantna
interpolacija od y = sin(2x)')

>> yi = interp1(x, y, xi, 'cubic');

%crtanje tocaka kod 'cubic' interpolacije
>> plot(x, y, 'o', xi, yi), title('Kubicna
interpolacija od y = sin(2x)')
```





8.2.2 Interpolacija s pomoću algebarskih polinoma

Pretpostavimo da je interpolacijska funkcija algebarski polinom $p_n(x)$ n-tog stupnja, gdje je *n* broj točaka interpolacije umanjen za 1. Dobro je znati da interpolacijski polinom p_n uvijek postoji i jedinstven je. Da bi se odredio interpolacijski polinom mogu se koristiti Vandermond-ova, Lagrange-ova, Newton-ova ili Aitken-ova metoda. Promotrit ćemo samo Newton-ovu, čiji interpolacijski polinom je reda k≤n, a prolazi kroz n+1 točku s koordinatama $(x_k,y_k) = (x_k,f(x_k))$, za k=0,1…,n

Prvo će se $p_n(x)$ napisati kao

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)(x - x_1) \ldots (x - x_{n-1})$$

Koeficijenti $a_0, a_1, \ldots, a_n$ zovu se podijeljene razlike (engl. *divided differences*) i mogu se izračunati rekurzivno. Prikaz $p_n(x)$ naziva se Newton-ov oblik interpolacijskog polinoma. Podijeljene razlike k-tog reda temeljene su na točkama $x_0, \ldots x_k$, označuju se sa $[a_0, \ldots, a_k]$, i definirane su rekurzivno kao

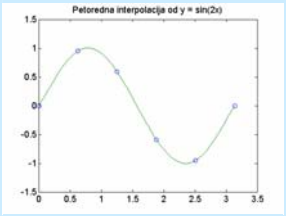
$$p_n(x_0)=a_0=y_0, \text{ tj.}[a_k] = y_k \text{ , ako je } k = 0$$

$$[a_0, \ldots, a_k] = ([a_1, \ldots, a_k] - [a_0, \ldots, a_{k-1}])/(x_k - x_0) \text{ , ako je } k > 0$$

Primjer 8.5: *Izračunavanje interpolacijskog polinoma*

Funkcija Newtonpol() izračunava interpolacijski polinom kod točaka koje korisnik unese. Vrijednost interpolacijskog polinoma yi u točkama xi. Koordinate točaka interpolacije sadržane su u vektorima x i y. Horner-ova metoda se koristi za izračun polinoma. Drugi izlazni parametar 'a' vraća koeficijente interpolacijskog polinoma u Newton-ovoj formi.

```
function [yi, a] = Newtonpol(x, y, xi)
a = divdiff(x, y);
n = length(a);
val = a(n);
for m = n-1:-1:1
  val = (xi - x(m)).*val + a(m);
end
yi = val(:);

function a = divdiff(x, y)
% Podijeljene razlike bazirane na točkama
%spremljene su pod x i y.
n = length(x);
for k=1:n-1
```



```
   y(k+1:n) = (y(k+1:n) - y(k))./(x(k+1:n) - x(k))
end
a = y(:);

%poziv funkcija
>> x = 0:pi/5:pi;
>> y = sin(2.*x);
>> xi = 0:pi/100:pi;
>> [yi, a] = Newtonpol(x, y, xi);
>> plot(x, y, 'o', xi, yi);
>> title('Petoredna interpolacija od y = sin(2x)')
```

Proces interpolacije ne daju uvijek dijelove polinoma koji jednoliko konvergiraju interpolacijskoj funkciji kad stupanj interpolacijskog polinoma teži u beskonačnost. Često se događa da se za točke koje su dovoljno blizu krajnjim točkama intervala interpolacije pojavljuje divergencija.

8.2.3 Interpolacija spline-om

Posebno značajne aproksimacije postižu se polinomnim spline funkcijama. Dan je interval [a, b]. Podjela **Δ** intervala [a, b] s točkama prekida $\{x_l\}_1^m$ je definirana kao $\Delta = a = x_1 < x_2 < \ldots < x_m = b$, gdje je $m > 1$. Neka su još k i n, gdje je $k < n$, nenegativni cijeli brojevi. Funkcija s(x) je spline funkcija stupnja n s glatkoćom k, ako su zadovoljeni sljedeći uvjeti:

(i) Na svakom podintervalu $[x_l, x_{l+1}]$, spline funkcija s(x) podudara se s algebarskim polinomom do najviše stupnja **n**
(ii) s(x) i njene derivacije do najviše reda k su kontinuirane na intervalu [a, b]

Matlab-ova funkcija `spline()` napravljena je za računanje s pomoću kubičnih spline-ova (n = 3) koji su dvostruko kontinuirano derivabilni (k = 2) na intervalu $[x_1, x_m]$.
Naredba

yi=`spline`(**x**, **y**, **xi**)

izračunava kubični spline s(x) u točkama koje se spremaju u vektor **xi**. Vektori **x** i **y** sadrže koordinate točaka koje treba interpolirati. Djelomično, tj. po odsječcima, polinomski prikaz spline interpolacije, može se dobiti s pomoću naredbe

pp=`spline`(**x**, **y**)

Naredbom **zi**=`ppval`(**pp**, **xi**) potom se izračunavaju odsječci polinoma spline interpolacije. Točke proračuna sadržane su u vektoru **xi**. Ako je potrebno proračunati

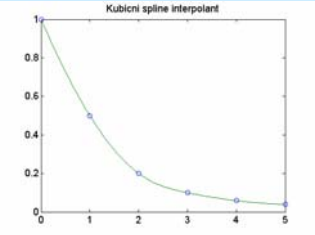
spline interpolant od nekoliko vektora **xi**, onda se svakako preporučuje korištenje `ppval()` funkcije.

Primjer 8.6: *Interpolacija podatka spline-om*

U ovom primjeru interpolirati će se Runge-ova funkcija g(x) =1/(1 + x^2) na intervalu [0, 5] koristeći šest jednakih mjesta točki prekida.
*Detaljne informacije o prikazu polinoma po odsječcima spline interpolanta mogu se dobiti pokretanjem funkcije spline() s dva ulazna parametra **x** i **y** i funkcijom za izvršavanje unmkpp().*

pp = spline(x, y); [brpts, coeffs, npol, ncoeff] = unmkpp(pp);

```
>> x = 0:5;
>> y = 1./(1 + x.^2);

>> xi = linspace(0, 5);
>> yi = spline(x, y, xi);
>> plot(x, y, 'o', xi, yi),
>> title('Kubicni spline interpolant')

%Maksimalna pogreška na odjeljku xi
>> err = norm(abs(yi-1./(1+xi.^2)),'inf')

%prikazu polinoma po odsječcima spline-a
>> pp = spline(x, y);
>> [brpts, coeffs, npol, ncoeff]=unmkpp(pp)
```




```
err =    0.0859
brpts =  0     1     2     3     4     5
coeffs =
0.0074   0.0777   -0.5852   1.00
0.0074   0.1000   -0.4074   0.50
-0.0371  0.1223   -0.1852   0.20
-0.0002  0.0110   -0.0519   0.10
-0.0002  0.0104   -0.0306   0.06
npol =     5
ncoeff =     4
```

Izlazni parametri funkcije `unmkpp()` za prikaz polinoma po odsječcima spline-a su:

- **brpts -** točke prekida spline interpolanta,
- **coeffs** - koeficijenti od s(x) na uzastopnom odsječku, podintervalu,
- npol - broj dijelova polinoma koji tvore spline funkciju
- ncoeff - broj koeficijenata koji predstavljaju svaki dio polinoma zasebno.

Na subintervalu $[x_l,x_{l+1}]$ spline interpolant prikazan je kao

$$s(x) = c_{l1}(x - x_l)^3 + c_{l2}(x - x_l)^2 + c_{l3}(x - x_l) + c_{l4}$$

gdje su $[c_{l1}, c_{l2}, c_{l3}, c_{l4}]$ l-ti red matrice **coeffs**. Ovaj oblik naziva se *djelomični polinomski oblik* spline funkcije.

8.2.4 Dvodimenzionalna interpolacija

Zadana je pravokutna mreža $\{x_k,y_j\}$ i njima pridruženi skup brojeva z_{kj}, $1 \le k \le m$, $1 \le j \le n$. Potrebno je naći bivarijantnu funkcija $z = f(x,y)$ koja interpolira podacima, tj. $f(x_k, y_j) = z_{kj}$ za sve vrijednosti *k* i *j*. Točke mreže moraju biti raspoređene monotono, tj. $x_1 < x_2 < \ldots < x_m$ sa sličnim rasporedom po y ordinati.

Matlab-ova funkcija

zi = `interp2`(**x**, **y**, **z**, **xi**, **yi**, 'method')

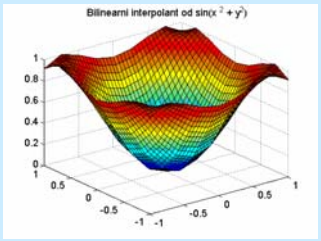
generira bivarijantni interpolant na pravokutnoj mreži i proračunava ih u točkama zadanih vektorima **xi** i **yi**. Šesti ulazni parametar 'method' je mogućnost kojom se zadaje metoda interpolacije. Dostupne metode su, kao i u slučaju funkcije `interp1()`:

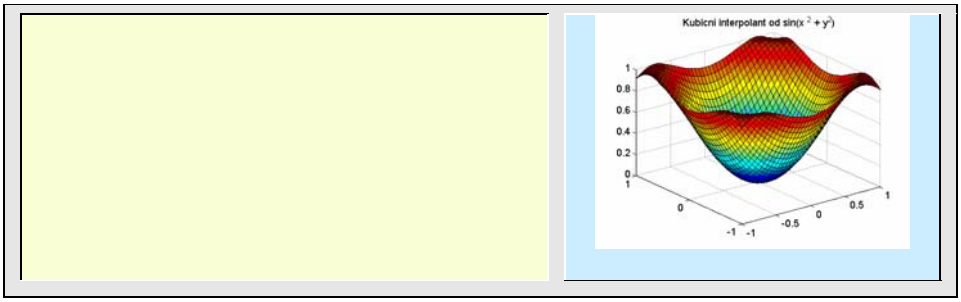
- 'nearest' – najbliža susjedna interpolacija
- 'linear' – bilinearna interpolacija
- 'cubic' – bikubična interpolacija
- 'spline' – spline interpolacija

Primjer 8.7: *Dvodimenzionalna interpolacija*

U ovom primjeru bivarijantna funkcija $z = sin(x^2 + y^2)$ interpolira se nad kvadratom $-1 \le x \le 1$, $-1 \le y \le 1$ s pomoću 'linear' i 'cubic' metode.

```
>> [x, y] = meshgrid(-1:.25:1);
>> z = sin(x.^2 + y.^2);

>> [xi, yi] = meshgrid(-1:.05:1);
>> zi = interp2(x, y, z, xi, yi, 'linear');

>> surf(xi, yi, zi), title('Bilinearni
interpolant od sin(x^2 + y^2)')

%dok je bikubični interpolant
>> zi = interp2(x, y, z, xi, yi, 'cubic');
>> surf(xi, yi, zi), title('Kubicni
interpolant od sin(x^2 + y^2)')
```





8.3 Numerički integralni račun

Klasičan problem numeričkog integralnog računa (kvadrature) je sljedeći: neka je zadana kontinuirana funkcija f(x), na intervalu $a \le x \le b$. Zadaća je naći koeficijente $\{w_k\}$ i čvorišta $\{x_k\}$, $1 \le k \le n$ tako da kvadraturna formula

$$\int_a^b f(x)dx \approx \sum_{k=1}^{n} w_k f(x_k)$$

vrijedi za polinome najvišeg mogućeg stupnja.

Za jednako raspoređena čvorišta $\{x_k\}$, skup kvadraturnih formula zove se *Newton-Cote*-ove formule. Ako se pretpostavi da su svi koeficijenti $\{w_k\}$ jednaki tada se kvadraturne formule zovu *Chebyshev*-ljeve kvadraturne formule. Ako su pak i koeficijenti $\{w_k\}$ i čvorišta $\{x_k\}$ određeni isključivo s pomoću opće (gornje) formule, onda se formule zovu *Gauss*-ove kvadraturne formule.

8.3.1 Funkcije quad() i quad8()

Dvije Matlabove funkcije načinjene za numeričko integriranje su:

```
quad('f', a, b, tol, trace, p1, p2, ...)   %sa Simpsonovom i
quadl('f', a, b, trace, p1, p2, ...)       %s rekurzivnim kvadraturama višeg reda.
```

Ulazni parametar *f* je niz koji sadrži ime funkcije ili držalo funkcije, koja se integrira od točke *a* do točke *b*. Četvrti ulazni parametar, *tol* može se zadati, ali ne mora, a označuje relativnu pogrešku prilikom integriranja. Parametar *tol* može sadržavati dvodimenzionalni vektor tol=[rel_tol, abs_tol] s apsolutnom i relativnom pogreškom tolerancije, koju zadaje korisnik. Parametar *trace* je također izboran, te ako je zadan, onda funkcija ispisuje korak po korak izvođenje funkcije integriranja. Ako se žele uzeti pretpostavljene vrijednosti za *tol* i *trace,* onda se na mjestu njihovih parametara stavlja

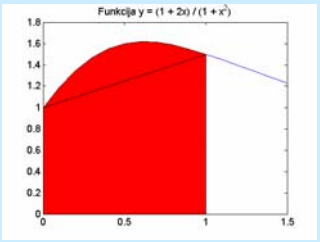
prazna matrica ([ ]). Parametri p1, p2 itd. su također izborni (ne nužni) parametri, a koriste se za zadavanje parametara integranda koji o njima ovisi.

Primjer 8.8: *Numeričko integriranje*

U ovom primjeru zadana je jednostavna racionalna funkcija: $\mathbf{f(x)} = \dfrac{\mathbf{a} + \mathbf{b} \cdot \mathbf{x}}{1 + \mathbf{c} \cdot \mathbf{x}^2}$.

Integracija od 0 do 1 za parametra a=1, b=2 i c=1, može se nacrtati , kao i numerički izračunati što je i prikazano ispod. Pretpostavljene relativne i apsolutne pogreške pohranjene su u vektoru tol, dok su parametri a, b i c, zadani preko dodatnih parametara p1, p2 i p3. Drugi izlazni parametar 'nfev' daje informaciju o broju funkcijskih izračuna (koraka) potrebnih za računanje integrala.

```
function y = rfun(x, a, b, c)
% Jednostavna racionalna funkcija koja ovisi o
%tri parametra a, b and c.
y = (a + b.*x)./(1 + c.*x.^2);
y = y';


%crtanje integracije pozivanjem rfun.m
>> x=0:0.1:1.5;
>> plot(x,rfun(x,1,2,1))
>> patch(0:0.1:1,rfun(0:0.1:1,1,2,1),'r')
>> patch([0 0 1 1],[0 rfun(0,1,2,1)
rfun(1,1,2,1) 0],'r')
>> title('Funkcija y = (1 + 2x) / (1 + x^2)')

>> format long
%dok je numerički integral izračunat sa:
>> tol = [1e-5 1e-3];
>> [q, nfev] = quad('rfun', 0, 1, tol,[ ], 1, 2, 1)

%koristeći funkciju quadl() dobiva se:
>> [ql, nfev]=quadl('rfun', 0, 1, tol,[ ],1,2,1)

%točna vrijednost integrala je:
>> stvarno=log(2)+pi/4

%pa se može izračunati relativna pogreška u
%izračunatim približnim vrijednostima q i ql
>> rel_pogreska = [abs(q - stvarno)/stvarno;
abs(ql - stvarno)/stvarno]
```




```
q =   1.47854599762368
nfev =    13


ql =   1.47854546052199
nfev =    18

stvarno = 1.47854534395739

rel_pogreska =
  1.0e-006 *
      0.44210093672034
      0.07883735021191
```

8.3.2 Newton-Cote-ove kvadraturne formule

Jednu od najstarijih metoda za numeričko računanje približne vrijednosti integrala na intervalu [a,b] otkrili su Newton i Cote. Čvorišta Newton-Cote-ovih formula odabrana su tako da budu jednako raspoređena u intervalu integrala. Postoji dva tipa Newton-Cote-ovih formula: zatvorene i otvorene. U prvom slučaju, krajnje točke intervala integrala uključene su u skup čvorišta, dok u otvorenim formulama nisu uključene. Težina $\{w_k\}$ određena je zahtjevom da je kvadraturna formula točna za polinome najvišeg mogućeg stupnja.

Za Newton-Cote-ove formule zatvorenog tipa vrijedi da se čvorišta n-točaka računaju prema: $x_k = a + (k-1)h$, za $k = 1,2,...n$; gdje je $h = (b-a)/(n-1)$, $n>1$. Težine kvadraturne formule određuju se iz uvjeta da su sljedeće jednadžbe zadovoljene za funkcije $f(x) = 1, x, ... x^{n-1}$.

$$\int_a^b f(x)dx = \sum_{k=1}^{n} w_k f(x_k)$$

Primjer 8.9: *Newton - Cote-ove kvadraturne formule*

U ovom primjeru s funkcijom pogreške $\mathbf{Erf(x)} = \frac{2}{\sqrt{\pi}} \cdot \int_0^x \mathbf{e^{-t^2}}\,\mathbf{dt}$ *načinit će se aproksimacija na x = 1 (tj. integrirat će se na intervalu [0 1]) koristeći zatvorene Newton-Cote-ove formule s parametrima n = 2 (trapezoidno pravilo), n = 3 (Simpson-ovo pravilo), n = 4 (Boole-ovo pravilo). Integrand će se procjenjivat koristeći funkciju exp2().*

```
function [s, w, x] = cNCqf(fun, a, b, n, varargin)
% Numer. aproksimacija konacnog integrala f(x).
%'fun' je string koji sadrzi ime integranda f(x).
% Integracija se provodi nad intervalom [a, b].
% Metoda koristena:  n-tocaka Newton-Cotes
%zatvorene kvadraturne formule
% Tezine i cvorista kvadraturne formule
% spremaju se u vektore w i x.

if n < 2
  error(' Broj cvorista mora biti veci od 1')
end
x = (0:n-1)/(n-1);
f = 1./(1:n);
V = Vander(x);
```

```
V = rot90(V);
w = V\f';
w = (b-a)*w;
x = a + (b-a)*x;
x = x';
s = feval(fun,x,varargin{:});
s = w'*s;

function w = exp2(x)
% Tezinska funkcija w Gauss-Hermitske
%kvadraturne formule
w = exp(-x.^2);

%mijenjajuci n od n=2 do n=4 i mnozeci s
faktorom 2/sqrt(pi) dobiva se:
>> approx_v = [];
>> for n =2:4
        approx_v = [approx_v; (2/sqrt(pi))*
cNCqf('exp2', 0, 1, n)]
>> end

%Za usporedbu, koristeci Matlabovu ugradjenu
%funkciju erf() dobivamo priblizno vrijednost
%funkcije pogreske za x = 1:
>> tocna_v = erf(1)
%sto pokazuje da je najbolje rjesenje s pomocu
%cNCqf za parametar n=4.
```

```
approx_v =
   0.77174333225805
   0.84310283004298
   0.84289057143172

tocna_v =
   0.84270079294971
```

8.3.3 Gauss-ove kvadraturne formule

Gauss-ove formule su tipa

$$\int_a^b p(x)f(x)dx \approx \sum_{k=1}^{n} w_k f(x_k)$$

gdje p(x) označava *težinsku funkciju*. Tipičan izbor težinskih funkcija s pripadajućim intervalima integracije je :

Tablica 8.1: *Težinske funkcije s pripadajućim intervalima integracije*

Težinska funkcija p(x)	Interval [a, b]	Ime kvadrature
1	[-1, 1]	Gauss-Legendre
$1/\sqrt{1-x^2}$	[-1, 1]	Gauss-Chebyshev
e^{-x}	[0, ∞)	Gauss-Laguerre
e^{-x^2}	(-∞, ∞)	Gauss-Hermite

Dobro je znano da su težine Gauss-ovih formula sve pozitivne, a čvorišta i korijeni polinoma su ortogonalni. Funkcije `Gquad` načinjena je za izračun konačnih integrala koristeći Gauss-ove kvadraturne formule.

Primjer 8.10: *Gauss-ove kvadraturne formule*

Numerička integracija koja koristi ili Gauss-Legendre (tip = 'L') ili Gauss-Chebyshev (tip = 'C') kvadratura sa n (n > 0) čvorišta. 'fun' je string koji predstavlja ime funkcije koja se integrira od a do b. Za Gauss - Chebyshev kvadraturu pretpostavlja se da je a = -1 i b = 1. Izlazni parametri s, w, i x pohranjuju izračunatu aproksimaciju integrala, popis težina i popis (listu) čvorišta.

I u ovom primjeru izračunati ćemo funkciju pogreške $\mathbf{Erf(x)} = \frac{2}{\sqrt{\pi}} \cdot \int_0^x \mathbf{e^{-t^2}\,dt}$

koristeći Gauss-Legendre-ove formule sa n = 2, 3, ... 8.

```
function [s, w, x] = Gquad(fun, a,b,n, type,
varargin)
d = zeros(1,n-1);
if type == 'L'
  k = 1:n-1;
  d = k./(2*k - 1).*sqrt((2*k - 1)./(2*k + 1));
  fc = 2;
  J = diag(d,-1) + diag(d,1);
  [u,v] = eig(J);
  [x,j] = sort(diag(v));
  w = (fc*u(1,:).^2)';
  w = w(j)';
  w = 0.5*(b - a)*w;
  x = 0.5*((b - a)*x + a + b);
else
  x = cos((2*(1:n) - (2*n + 1))*pi/(2*n))';
  w(1:n) = pi/n;
end
```

```
f = feval(fun,x,varargin{:});
s = w*f(:);
w = w';

function w = exp2(x)
% Tezinska funkcija w Gauss-Hermitske
%kvadraturne formule
w = exp(-x.^2);

>> approx_v = [ ];
>> for n =2:8
        approx_v = [approx_v; (2/sqrt(pi))*
Gquad('exp2', 0, 1, n,'L')]
   end
>> tocna_v=erf(1) %tocna vrijednost
```

```
approx_v =
   0.84244189252255
   0.84269001848451
   0.84270117131620
   0.84270078612733
   0.84270079303742
   0.84270079294882
   0.84270079294972

tocna_v =
   0.84270079294971
```

8.3.4 Romberg-ova metoda

Romberg predlaže metodu koja ne pripada razredu metoda koje su dosad opisane. Ta metoda sastoji se iz dvije faze. Prva faza stvara niz približnih vrijednosti koristeći pravilo kompozitnog trapeza (engl. *composite trapezoidal rule*). Druga faza popravlja rezultate dobivene u prvoj fazi koristeći Richardson-ovu ekstrapolaciju. Ovaj proces je rekurzivan i broj ponovljenih operacija ovisi o vrijednosti parametra integrala *n*. U većini slučaja nije nužan veliki *n*.
Korisnička funkcija

```
Romberg(fun, a, b, n, varargin)
```

primjenjuje Rombergov algoritam.

Primjer 8.11: *Rombergova metoda*

Numerička aproksimacija 'rn' konacnog integrala od a do b koja se dobiva s pomoću Romberg-ove metode sa n redaka i n stupaca.' fun' je string koji označuje integrand. Ako integrand ovisi o parametrima, npr. p1, p2, ... , onda se oni trebaju navesti odmah nakon parametra n. Drugi izlazni parametar r1 čuva aproksimirajuće vrijednosti izračunatog integrala dobivene uz pomoć kompozitnog trapezoidnog pravila koristeći 1, 2, ... ,n podintervale.
Funkcija Romberg() ispitat će se integrirajući racionalnu funkciju rfun() obrađivanu već sa quad() i quadl() funkcijama. Interval integracije je [a,b] = [0,1], n = 10, a vrijednost parametara a, b i c su 1, 2, i 1.

```
function [rn, r1] = Romberg(fun, a, b, n, varargin)
h = b - a;
d = 1;
r = zeros(n,1);
r(1) = .5*h*sum(feval(fun,[a b],varargin{:}));
for i=2:n
  h = .5*h;
  d = 2*d;
  t = a + h*(1:2:d);
  s = feval(fun, t, varargin{:});
  r(i) = .5*r(i-1) + h*sum(s);
end
r1 = r;
d = 4;
for j=2:n
  s = zeros(n-j+1,1);
  s = r(j:n) + diff(r(j-1:n))/(d - 1);
  r(j:n) = s;
  d = 4*d;
end
rn = r(n);

function y = rfun(x, a, b, c)
% Jednostavna racionalna funkcija koja ovisi o tri
%parametra a, b and c.
y = (a + b.*x)./(1 + c.*x.^2);
y = y';

%Ovog puta, apsolutne i relativne pogreske u 'rn' su:
>> [rn, r1] = Romberg('rfun', 0 , 1, 10, 1, 2, 1)
>> stvarno=log(2)+pi/4
>> [abs(stvarno - rn); abs(rn - stvarno)/stvarno]
```

```
rn =
   1.47854534395739

r1 =
   1.25000000000000
   1.42500000000000
   1.46544117647059
   1.47528502049722
   1.47773122353730
   1.47834187356141
   1.47849448008531
   1.47853262822223
   1.47854216503816
   1.47854454922849

stvarno =
   1.47854534395739

ans =
     0
     0
```

8.3.5 Numeričko integriranje dvostrukih i trostrukih integrala

Funkcija `dblquad()` računa približnu vrijednost dvostrukog integrala

$$\iint_D f(x, y)dxdy$$

gdje je:

$D=\{(x,y): a \le x \le b,\ c \le y \le d\}$

domena integrala. Sintaksa funkcije `dblquad()` je:

```
dblquad(fun, a, b, c, d, tol)
```

gdje parametar *tol* ima isto značenje kao i u funkciji `quad()`.

Primjer 8.12: *Integriranje dvostrukih i trostrukih integrala*

Neka je $f(x,y) = e^{-xy} \sin(xy)$, $-1 \le x \le 1$, $0 \le y \le 1$. Funkciju f zapisuje se u m-datoteci ili se koristi inline() funkcija. Integrirajući funkciju f s pomoću naredbe dblquad() nad zadanim intervalom dobiva se rezultat.
Funkcija f(x,y) može se zadati na još dva poznata načina:

- *zadavanjem držala funkcije*
- *inline() sintaksom*

*Na sličan način računaju se i trostruki određeni integrali, npr. integrirat će funkciju f(x,y,z)= y*sin(x)+z*cos(x) nad intervalom: 0 <= x <= pi, 0 <= y <= 1, -1 <= z <= 1.*

```
function z = esin(x,y);
z = exp(-x*y).*sin(x*y);

>> result = dblquad('esin', -1, 1, 0, 1)
>> rez_drzalo = dblquad(@esin, -1, 1, 0, 1)
>> rez_inline = dblquad(inline('exp(-
x*y).*sin(x*y)'), -1, 1, 0, 1)

%trostruki integral
>> Q = triplequad(inline('y*sin(x)+
z*cos(x)'), 0, pi, 0, 1, -1, 1)
```

```
result =
  -0.22176852942704
rez_drzalo =
  -0.22176852942704
rez_inline =
  -0.22176852942704

Q =
   1.99999999436264
```

8.4 Numerički diferencijalni račun

Numerički diferencijalni račun jednako je važan kao i integralni. Na tom polju u Matlabu su načinjeni veliki koraci ne samo u općim, nego i parcijalnim diferencijalnim jednadžbama. Postoji niz "solvera" koji rješavaju mnoge diferencijalne zadaće.

8.4.1 Funkcija diff()

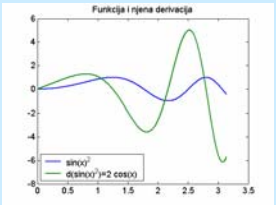
Funkcija `diff()` u Matlabu nalazi razliku susjednih vrijednosti vektora, pa će `diff`(X) za vektor X izračunati [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)] što se može iskoristiti u računanju aproksimacije derivacije funkcija, po formuli:

$$f'(x) = (f(x+h)-f(x)) /h$$

Primjer 8.13: *Aproksimacija derivacije funkcije*

Derivaciju funkcije se može numerički aproksimirati funkcijom diff()

```
>> h = .001; x = 0:h:pi;
>> der=diff(sin(x.^2))/h;
>> plot(x,sin(x.^2),x,[0 der])
>> title('Funkcija i njena derivacija')
>> legend('sin(x)^2','d(sin(x)^2)=2 cos(x)',3)
```



Treba primijetiti da funkcija `diff()` daje jednu vrijednost manje od ukupnog broja elemenata vektora, pa je treba dodati na početku ili kraju vektora razlika.

Problem diferencijalnog računa je dakle za zadanu funkciju f(x), naći približnu vrijednost $f'(x)$. Točniji algoritam koji računa niz približnih vrijednosti za derivaciju, koristeći aproksimaciju konačnih razlika, bio bi:

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$$

gdje je h početni korak. Prva faza ove metode računa niz aproksimacija $f'(x)$ koristeći nekoliko vrijednosti za h. Kada se sljedeća aproksimacija riješi, vrijednost h se prepolovljuje. U ovoj drugoj fazi koristi se Richardson-ovu ekstrapolacija. Funkcija `numder()` koristi ovu metodu.

Primjer 8.14: *Aproksimacija konačnim razlikama*

U ovom primjeru traži se numerička aproksimacija derivacije prvog reda funkcije $f(x) = e^{-x2}$. *Ona se računa koristeći funkciju numder() i uspoređuje s točnim (eksplicitno izračunatim) vrijednostima* $f'(x)$ *za x = 0.1, 0.2, ...1.0. Vrijednosti parametara h i n su 0,01 i 10.*
Rezultat kôda kojim se pozivaju funkcije u ovom primjeru biti će prikazan ispod

primjera zbog neprimjerene veličine.

```
function der = numder(fun, x, h, n, varargin)
% Aproksimacija 'der' prvog reda derivacije, u tocki x, od funkcije
%zadane stringom 'fun'. Parametri h i n su inicijalne vrijednosti za korak
%i broj iteracija koje zadaje korisnik. Iteracije se izvode Richardson-
%ovom ekstrapolacijom. Za funkcije koje ovise o parametrima, njihovi
%parametri  zadaju se iza parametra n.
d = [ ];
for i=1:n
  s = (feval(fun,x+h,varargin{:})-feval(fun,x-h,varargin{:}))/
(2*h);
  d = [d;s];
  h = .5*h;
end
l = 4;
for j=2:n
  s = zeros(n-j+1,1);
  s = d(j:n) + diff(d(j-1:n))/(l - 1);
  d(j:n) = s;
  l = 4*l;
end
der = d(n);

function testnder(h, n)
% Test file za funkciju  numder(). Inicijalni korak je h i broj iteracija je
n. Testira se f(x) = exp(-x^2).
format long
disp(' x        numder              tocno')
disp(sprintf('_____________________________________________\n'))
for x=.1:.1:1
  s1 = numder('exp2', x, h, n);
  s2 = derexp2(x);
  disp(sprintf('%1.4f    %1.14f     %1.14f',x,s1,s2))
end

%gdje je za usporedbu s tocnim vrijednostima:
function y = derexp2(x)
% Derivacija prvog reda od f(x) = exp(-x^2).
y = -2*x.*exp(-x.^2);

%rezultati dobiveni pozivom sljedece funkcije
>> testnder(0.01, 10)
```

Rezultat poziva funkcije `testnder`(0.01, 10):

x	numder	tocno
0.1000	-0.19800996675001	-0.19800996674983
0.2000	-0.38431577566308	-0.38431577566093
0.3000	-0.54835871116311	-0.54835871116274
0.4000	-0.68171503117432	-0.68171503117297
0.5000	-0.77880078306967	-0.77880078307140
0.6000	-0.83721159128023	-0.83721159128524
0.7000	-0.85767695185697	-0.85767695185818
0.8000	-0.84366787846708	-0.84366787846888
0.9000	-0.80074451919839	-0.80074451920129
1.0000	-0.73575888234189	-0.73575888234288

8.4.2 Eulerova aproksimacija

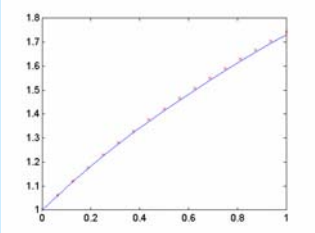
Temelj metode je u rekurzivnom izračunavanju jednadžbe diferencija:

y(i)=y(i-1)+h*f(i-1)

Primjer 8.15: *Eulerova aproksimacija*

*Na jednostavnom primjeru diferencijalne jednadžbe y' = dy/dx = 1/y može se pokazati rješavanje Eulerovom metodom aproksimacije. Potrebno je zadati početnu vrijednost y(0) koja će se u programu označiti varijablom 'starty'. Po varijabli x načinit će se mreža od 0 do 1 s korakom h. Točno rješenje za pretpostavljeni primjer je sqrt(2*x+1).*

```
>> h = 1/16;
>> starty = 1;
>> x = [0:h:1];

>> y = 0*x;
>> y(1) = starty;

>> for i=2:max(size(y)),
       y(i) = y(i-1) + h/y(i-1);
>> end

>> tocno = sqrt(2*x+1);
>> plot(x,y,'rx',x,tocno)
```

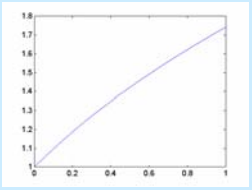


Ovu najjednostavniju metodu rješavanja diferencijalne jednadžbe Eulerovom metodom aproksimacije moguće je ugraditi u funkciju koja će, zahvaljujući parametrima, imati općenitije značenje.

Primjer 8.16: *Funkcija za Eulerovu aproksimaciju*

U ovom primjeru dif. jednadžba y'=1/y će biti zapisana u vanjsku datoteku, a početni uvjeti će biti zadani kao parametri kod poziva funkcije eulerApprox()

```
function [x,y] = eulerApprox(startx,h,endx,starty,func)
% Ova funkcija nalazi aproksimaciju za dif. jedn:
% y' = func(x,y)
% y(startx) = starty
% Kod poziva potrebno je navesti
% startx  : pocetna vrijednost za x
% h       : velicina koraka
% endx    : konacna vrijednost za x
% starty  : inicijalna vrijednost za y, y(0)
% func    : ime datoteke koja sadrzi dif. jednd.
%         Primjer: [x,y] = eulerApprox(0,1/16,1,1,'f');
%Vraca aproksimaciju dif. jedn. gdje je x od 0 do 1 u
%koracima po 1/16. Pocetna vrijednost je 1, a desna
%strana jedn. racuna se po funkciji spremljenoj u f.m
%datoteci.
%Funkcija vraca dva vektora. Prvi vektor je x koji
%sadrzi mrezu od x0=0 i ima korak h. Drugi vektor je
%aproksimacija rjesenja te dif. jedn.

x = [startx:h:endx];
y = 0*x; y(1) = starty;
for i=2:max(size(y)),
    y(i) = y(i-1) + h*feval(func,x(i-1),y(i-1));
end

function [f] = f(x,y)
% zadavanje desne strane diferencijalne jednadzbe
f = 1/y;

%poziv funkcije eulerApprox()
>> [x,y] = eulerApprox(0,1/16,1,1,'f');
>> plot(x,y)
```



Eulerova metoda može imati više varijanti, npr. može se načiniti pomak s obzirom na točke pa se radi sa $y(xk+1)$ da se izbjegnu negativni indeksi. Jedan takav program koji od

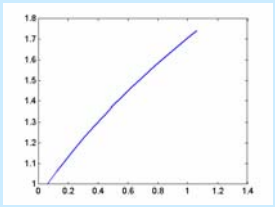
korisnika interaktivno traži unos početnih postavki, a dif. jednadžbu ima spremljenu u f.m datoteci:

$$y(xk+1) = y(xk) + hf(xk; y(xk)).$$

Primjer 8.17: *Eulerova aproksimacija, varijanta 2*

U interakciji s korisnikom programski kôd će zahtijevati unos parametara, te će u ovom primjeru parametri biti: a=0, b=1, N=16, ya=1, što će rezultirati pripadajućom slikom.

```
function [f] = f(x,y)
% zadavanje desne strane diferencijalne jednadzbe
f = 1/y;

>> clear
% ovaj program poziva jednadzbu iz datoteke f.m
>> a=input('Lijeva pocetna tocka intervala a = ');
>> b=input('Desna konacna tocka intervala b = ');
>> N=input('Broj podintervala, N = ');
>> ya=input('Pocetna vrijednost u x=a, ya= ');
>> h=(b-a)/N;
>> x=a+h*(1:(N+1));
>> lx=length(x);
>> y(1)=ya;
>> for j=1:N
       y(j+1)=y(j)+h*f(x(j),y(j));
>> end
>> plot(x,y)
```



Međutim, u Matlabu nije potrebno graditi vlastite programa za numeričko rješavanje diferencijalnog računa, jer Matlab ima već izgrađene brze i djelotvorne funkcije.

8.4.3 Obične diferencijalne jednadžbe

Mnogi fizikalni problemi u znanosti i inženjerstvu zahtijevaju poznavanje funkcije y = y(t) koja zadovoljava prvi red diferencijalne jednadžbe y' = f(t,y) s početnim uvjetima $y(a) = y_0$; gdje su a i y realni brojevi, a f je dvovarijabilna funkcija koja zadovoljava određene uvjete neprekinutosti i glatkoće.
Općenitiji problem formulira se na sljedeći način. Za zadanu funkciju *f* s *n* varijabli, treba naći funkciju y = y(t) koja zadovoljava n-ti red obične diferencijalne jednadžbe

$$y^{(n)} = f(t,y,y',...y^{(n-1)})$$

s početnim uvjetima $y(a) = y_0$, $y'(a) = y'_0$, $y^{(n-1)}(a) = y_0^{(n-1)}$. Taj se problem često transformira u problem rješavanja sustava diferencijalnih jednadžbi prvog reda. Naziv *obične diferencijalne jednadžbe* (engl. *ordinary differential equations*) označava se obično kraticom **ODE**.

Matlab ima nekoliko funkcija za numeričko računanje ODE problema:

Tablica 8.2: *ODE funkcije*

Funkcija	Opis	Korištena metoda
`ode23()`	Ne-krute (nonstiff) ODE	Eksplicitna Runge-Kutta 2 i 3 reda
`ode45()`	Ne-krute (nonstiff)ODE	Eksplicitna Runge-Kutta 4 i 5 reda
`ode113()`	Ne-krute (nonstiff) ODE	Adams-Bashforth-Moulton solver
`ode15s()`	Krute (stiff) ODE	Solver temeljen na jednadžbama numeričke diferencijacije
`ode23s()`	Krute (stiff) ODE	Solver temeljen na modificiranoj Rosenbrock-ovoj jednadžbi drugog reda

Najjednostavniji oblik sintakse Matlab ODE je:

[t,y] = *solver* (fun,tspan,y0)

gdje je *fun* string koji sadrži ime ODE-ove m-datoteke koja opisuje diferencijalnu jednadžbu, **tspan** je interval integracije, a **y0** je vektor koji sadrži početnu(e) vrijednost(i). Izlazni parametri t i y su vektori koji sadrže točke procijenjenih i izračunatih vrijednosti za y na tim točkama.

Primjer 8.18: *ODE*

U ovom primjeru tražit će se numeričko rješenje za y ; t = 0, .25, .5, .75, 1. ; za zadanu dif. jednadžbu $y' = -2ty^2$ s početnim uvjetom y(0) = 1.Koriste se ode23() i ode45() funkcije ("solvers") kako bi se načinila usporedba. Točno rješenje ovog problema je $y(t) = 1/(1+t^2)$. Potrebna ODE m-datoteka je eq1.m.

```
function dy = eq1(t,y)
% m-file za ODE y' = -2ty^2.
dy = -2*t.*y(1).^2;

%poziv funkcije eq1()
>> tspan = [0 .25 .5 .75 1];
>> y0 = 1;
>> [t1 y1] = ode23('eq1',
tspan, y0);
>> [t2 y2] = ode45('eq1',
```

```
ans =

0      1.00000000000000  1.00000000000000
0.25   0.94118221525751  0.94117646765650
0.50   0.80002280597122  0.79999999678380
0.75   0.64001788410487  0.63999998775736
1.00   0.49999658522366  0.50000000471194
```

```
tspan, y0);

%dobiveni rezultati
>> [t1 y1 y2]
```

Primjer 8.19: *Obične dif. jed. prvog reda*

Umjesto ODE m-datoteke za ovaj sustav može se koristiti naredba inline(). Sustav::

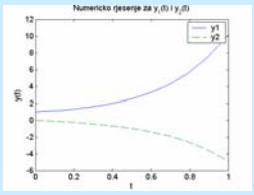
$$y1'(t) = y1(t) - 4y2(t),\ y2'(t) = -y1(t) + y2(t),$$
$$y1(0) = 1;\ y2(0) = 0.$$

Inline() naredbe pišu se u naredbenom, interaktivnom Matlabovom prozoru. Interval za koji se traži numeričko rješenje i početne vrijednosti za ODE sustav pohranjeni su u vektorima ***tspan*** *i* ***y0****.*

Numeričko rješenje sustava dobiva se pozivom ode23() naredbe. Dobiveni su grafovi of y1(t) (puna linija) i y2(t) (crtkana linija). Točna rješenja za ovaj sustav dobivena su uz pomoć funkcije dsolve() iz simboličkog toolbox-a:

```
>> dy = inline('[1 -4; -1 1]*y', 't', 'y')
>> tspan = [0 1]; y0 = [1 0];
>> [t,y] = ode23(dy, tspan, y0);
>> plot(t,y(:,1),t,y(:,2),'--')
>> legend('y1','y2'),
>> xlabel('t'), ylabel('y(t)'),
>> title('Numericko rjesenje za y_1(t) i y_2(t)')

>> S = dsolve('Df = f - 4*g','Dg = -f + g','f(0) =
1','g(0) = 0')
>> S.f
>> S.g
```

```
dy =
    Inline function:
    dy(t,y) = [1 -4; -1 1]*y
```




```
S =   f: [1x1 sym]
      g: [1x1 sym]
ans =
1/2*exp(-t)+1/2*exp(3*t)
  ans =
-1/4*exp(3*t)+1/4*exp(-t)
```

8.4.4 Nelinearne obične diferencijalne jednadžbe

Dakle, za numeričko integriranje diferencijalnih jednadžbi, najčešće se koriste dvije funkcije: `ode23()`, i `ode45()`. Funkcija `ode23()` koristi par formula drugog i trećeg stupnja, dok `ode45()` koristi Runge-Kutta-Fehlberg formule četvrtog i petog stupnja za automatsko koračno integriranje. Matlab u drugim funkcijama koristi i druge oblike algoritama za rješavanje diferencijalnih jednadžbi.

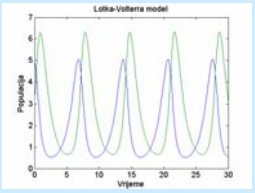
Primjer 8.20: *Integriranje nelinearnih običnih dif. jednadžbi*

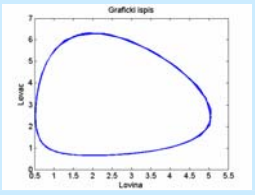
U ovom primjeru uzet ćemo dvije diferencijalne jednadžbe koje opisuju dinamiku hvatanja u modelu grabežljivac (ili lovac)-lovina:

$$dy_1/dt = (1 - \alpha y_2)\, y_1 \qquad i \qquad dy_2/dt = (-1 + \alpha y_1)\, y_2$$

Zbog jednostavnosti neka parametri za određivanje granica tolerancije budu onakvima kakvi jesu. Obje funkcije (ode23() i ode45()) koriste iste parametre. Ako želimo koristiti ode45() funkciju, sve što moramo učiniti je zamijeniti ode23() sa ode45() u pozivu. Neka se program spremi u skriptnu m-datoteku pod imenom "odesolver.m". Diferencijalne jednadžbe spremaju su u drugu m-datoteke pod imenom "lotka.m". Program se pokreće pozivom "odesolver".

```
function dydt = lotka(t,y, flag,alpha,beta)
% lotka.m je datoteka koja vraća derivacije dydt[]
%(dy/dt) funkcijama ode23() or ode45(). Varijabla
%'flag' je prazna varijabla, a lotka() uzima samo
%parametre 'alpha' and 'beta'.
yp1 = (1 - alpha*y(2))*y(1);
yp2 = (-1 + beta*y(1))*y(2);
dydt = [yp1; yp2]; % dydt je vektor stupca

% pocetak programskog kôda datoteke odesolver.m
alpha=0.4;
beta=0.5;
% integriraj od t=0 do t=30 [vrem. jedinice]
tspan = [0 30];
% pocetni uvjeti y0 u trenutku t=0 za obje jednadžbe
y0 = [5 3];
[t,y] = ode23('lotka',tspan,y0, [],alpha,beta);

% Prikaz grafickih rezultata
plot(t,y)
title('Lotka-Volterra model')
xlabel('Vrijeme'),ylabel('Populacija')

figure(2)
plot(y(:,1),y(:,2))
title('Graficki ispis')
xlabel('Lovina'),ylabel('Lovac')
figure(1) %povratak na fig 1
% kraj programskog kôda datoteke odesolver.m

%Rješenja ovog sustava dobit će se pozivom skripte:
```





```
>> odesolver
```

Nelinearne obične diferencijalne jednadžbe opisane su sljedećim graničnim problemom s dvije točke (engl. two-point boundary value problem):

y y'' + 2 y''' = 0
x = 0 , y=0, y'=0
x = inf, y' = 1

Primjer 8.21: *Obične dif. jednadžbe u prostoru stanja*

Neka se pretpostavi da je inicijalna vrijednost f''(0) = 0.33206. Radi se o rješavanju laminarnog toka na ravnoj ploči (tzv. Blaziusov problem).
Prvo se treba pretvoriti jednadžba trećeg reda u 3 jednadžbe prvog, to je definicija u prostoru stanja. Uobičajena je praksa da se koriste slobodne (engl. dummy) varijable y1, y2, y3:

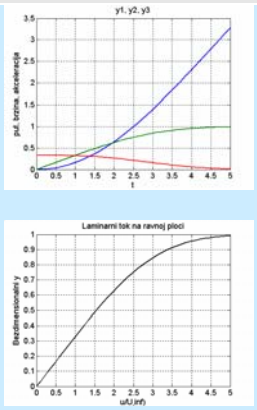
$$y1 = y$$
$$y1' = y2$$
$$y2' = y3, \text{ tj. } y'' = y3$$
$$y3' = -y1*y3/2, \text{ tj. } y''' = -0.5\, y\, y''$$
$$y1(0) = 0, \quad y2(0) = 0, \quad y3(0) = 0.33206$$

Zatim se dobiveno stanje upiše u m-datoteku stanje.m koja će se pozvati s ODE funkcijom:

```
function st = stanje(t,y)
% Postavlja jednadzbe stanja kao stupcasti
%vektor.
% Prethodno stanje spremljeno je u y
% Derivacije prvog reda vraćaju se u 'st'
st = [ y(2) , y(3) , -0.5*y(1)*y(3)]';


% Pocetak kôda skripte runge.m
%Skripta za probleme pocetnih uvjeta  koja
%koristi Matlab ODE45
ti = 0.0; % pocetak integracije
tf = 5.0; % konacna vrijednosti integracije
tint = [ti tf];  % vektor pocetne i krajnje tocke
pocvr = [0.0 0.0 0.33206];  % pocetne vrijed.
[t y ] = ode45('stanje',tint,pocvr);

% u y su spremljeni stupci za y1, y2, y3
[t y]   %vektori koji ce se ispisati pri pozivu
plot(t, y), grid, title('y1, y2, y3')
```

```
ans =
0         0         0         0.3321
0.0002    0.0000    0.0001    0.3321
0.0003    0.0000    0.0001    0.3321
0.0005    0.0000    0.0002    0.3321
0.0006    0.0000    0.0002    0.3321
0.0014    0.0000    0.0005    0.3321
0.0021    0.0000    0.0007    0.3321
0.0029    0.0000    0.0010    0.3321
0.0036    0.0000    0.0012    0.3321
0.0074    0.0000    0.0025    0.3321
0.0112    0.0000    0.0037    0.3321
...
4.4726    2.7634    0.9786    0.0353
4.6045    2.8927    0.9828    0.0293
4.7363    3.0225    0.9863    0.0241
4.8682    3.1527    0.9892    0.0197
5.0000    3.2833    0.9915    0.0159
```

```
xlabel('t'),
ylabel('put, brzina, akceleracija')
figure(2);
plot(t,y(:,2),'k-') % profil brzine
grid
title('Laminarni tok na ravnoj ploci')
xlabel('u/U_(inf)')
ylabel('Bezdimensionalni y'), figure(2);
%Kraj kôda skripte runge.m

%Poziv skripte runge.m
>> runge
```



Problem graničnih vrijednosti dviju točaka općeniti je problem numeričkih metoda. Za ODE drugog reda vrijedi:

$$y''(t) = f(t, y, y')$$
$$y(a) = ya,\ y(b) = yb.$$

Metoda koja se koristi zove se metoda konačnih razlika (engl. *finite difference method)*. Neka se pretpostavi da je funkcija *f* u obliku f(t, y, y') = g0(t) + g1(t)y + g2(t)y'. Funkcija *f* je linearna i po *y* i po *y′*. Koristeći standardnu aproksimaciju drugog reda za *y′* i *y″* može se relativno jednostavno konstruirati linearni sustav jednadžbi za izračunavanje aproksimativnih vrijednosti *y* na skupu jednoliko raspoređenih točaka. Funkcija `bvp2ode()` koristi ovu metodu:

Primjer 8.22: *Problem graničnih vrijednosti*

Numeričko rješenje y problema graničnih vrijednosti

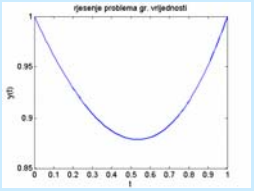
$$y'' = g0(t) + g1(t)*y + g2(t)*y'$$
$$y(a) = ya,\ y(b) = yb,$$

na n+2 jednoliko raspoređenih točaka t na intervalu tspan = [a b]. g0, g1 i g2 su stringovi koji predstavljaju funkcije g0(t), g1(t) i g2(t). Granične vrijednosti ya i yb spremaju se u vektor bc = [ya yb]. Kao primjer, može se uzeti jednadžba drugog stupnja: y″(t) = 1 +sin(t)y + cos(t)y′, s početnim uvjetima: y(0) = y(1) = 1.

```
function [t, y] = bvp2ode(g0, g1, g2, tspan, bc, n)
a = tspan(1); b = tspan(2);
t = linspace(a,b,n+2); t1 = t(2:n+1);
u = feval(g0, t1);v = feval(g1, t1);
w = feval(g2, t1);
h = (b-a)/(n+1);  %broj medjuintervala
```

```
d1 = 1+.5*h*w(1:n-1); d2 = -(2+v(1:n)*h^2);
d3 = 1-.5*h*w(2:n);
A = diag(d1,-1) + diag(d2) + diag(d3,1);
f = zeros(n,1);
f(1) = h^2*u(1) - (1+.5*h*w(1))*bc(1);
f(n) = h^2*u(n) - (1-.5*h*w(n))*bc(2);
f(2:n-1) = h^2*u(2:n-1)';
s = A\f; y = [bc(1);s;bc(2)]; t = t';

%Umjesto u datoteci, funkcije se mogu definirati i
%inline().
>> g0 = inline('ones(1, length(t))', 't'), g1 =
inline('sin(t)', 't'), g2= inline('cos(t)', 't')

%Poziva funkcija bvp2ode()
>> [t, y] = bvp2ode(g0, g1, g2, [0 1],[1 1],100);
>> plot(t, y), axis([0 1 0.85 1])
>> title('rjesenje problema gr. vrijednosti')
>> xlabel('t'), ylabel('y(t)')
```

```
g0 =
    Inline function:
    g0(t) = ones(1, length(t))

g1 =
    Inline function:
    g1(t) = sin(t)

g2 =
    Inline function:
    g2(t) = cos(t)
```



Dakako, moguće su kombinacije rješavanja diferencijalnih jednadžbi s drugim numeričkim metodama. Na primjer, trapezno pravilo s korekcijskim članom često se koristi kod numeričkih integracija funkcija koje su derivabilne na intervalima integracije, pa vrijedi:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(a)+f(b)\right]-\frac{h^2}{2}\left[f'(a)-f'(b)\right]$$

gdje h = b – a. Ova formula se lako implementira u MATLAB.

Primjer 8.23: *Korigirano trapezno pravilo*

*U ovom primjeru integrirat će se funkciju sinus preko intervala čije su završne točke spremljene u vektorima **a** i **b**.*
Pošto integralna funkcija i njezina prva derivacija pripadaju Matlab ugrađenim funkcijama nije bilo potrebe koristiti dsolve() funkciju i rješenje spremati u vanjsku m-datoteku.

```
function y = corrtrap(fname, fpname, a, b)
% Korigirano trapezno pravilo y.
% fname - ime m-datoteke u kojem je spremljen integrand
% fpname - ime m-datoteke u kojoj je spremljena prva
```

```
%derivacija integranda
% a,b - krajnje točke intervala integracije.
h = b - a;
y = (h/2).*(feval(fname,a) + feval(fname,b))+...
(h.^2)/12.*(feval(fpname,a) - feval(fpname,b));

>> a = [0 0.1]
>> b = [pi/2 pi/2 + 0.1]
>> y = corrtrap('sin', 'cos', a, b)
```

```
a =
    0         0.1000
b =
    1.5708    1.6708
y =
    0.9910    1.0850
```

8.5 Simbolička matematika

Matlab svoju simboličku matematiku temelji na Mapple proizvodu koji je ugradio kao jedan od toolbox-a pod imenom 'symbolic'. Da bi se bilo koji, pa prema tomu i simbolički toolbox, mogao izvoditi, treba biti naveden u Matlab-ovoj stazi, 'matlabpath' putu. U tom slučaju se funkcije toolbox-a pozivaju na isti način kao i osnovne Matlab funkcije.
Temeljna stvar simboličkog pristupa je simbolički objekt, koji se stvara upotrebom `sym()` funkcije. Tako će na primjer

```
>> sqrt(3)
ans =
   1.7321
```

dati numerički rezultat drugog korijena iz 3, dok će

```
>> simb = sqrt(sym(3))
simb =
   3^(1/2)
```

varijabli *simb* pridružiti simbolički objekt drugog korijena iz 3.
Pretvorba u numerički oblik sa zadanim brojem znamenki ostvaruje se preko `vpa()` funkcije:

```
>> vpa(simb,8)
ans =
   1.7320508
>> vpa(simb,18)
ans =
   1.73205080756887729
```

Na sličan način, bez numeričkog izračunavanja, moguće je raditi s razlomcima:

```
>> sym(2)/sym(5) + sym(1)/sym(3)
ans =
   11/15
```

Da bi varijabla postala simbolička potrebno je pozvati funkciju `sym()` s imenom željene simboličke varijable. Treba, međutim, uočiti da naredba

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

pridjeljuje simbolički atribut samo varijabli f, ali ne i ostalima. Ostale varijable postaju simboličke samo ako se napiše

```
>> a = sym('a')
>> b = sym('b')
>> c = sym('c')
>> x = sym('x')
```

ili, što je puno jednostavnije, samo sa:

```
>> syms a b c x
```

Pojedinim simboličkim varijablama moguće je dodavati i druge atribute, pa će na primjer:

```
>> x = sym('x','real'); y = sym('y','real');
```

učiniti da x i y varijable budu simboličke, ali s pretpostavljenim realnim sadržajem. Tako ćemo kompleksnu simboličku varijablu generirati s naredbom:

```
>> syms x y real  % ovo je sažetiji zapis istih atributa
>> z = x + i*y
```

Na sličan način moguće je raditi sa matricama čiji su elementi simboličke veličine. Na primjer:

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

će vratiti

```
A =
   [ a, b, c ]
   [ b, c, a ]
   [ c, a, b ]
```

Ako zadamo naredbu, na primjer:

```
>> sum(A(1,:))
```

dobit ćemo, dakako, rezultat u simboličkom obliku.

```
ans =
    a+b+c
```

Slično će:

```
>> syms a b c d
>> Asym=[a, b ;c, d], Asymsq = sym(Asym)*sym(Asym)
```

dati:

```
Asym =
   [ a, b]
   [ c, d]
Asymsq =
   [ a^2+b*c, a*b+b*d]
   [ c*a+d*c, b*c+d^2]
```

Simbolički toolbox raspolaže mnoštvom različitih funkcija koje se mogu primijeniti na simbolički zadane varijable. Tako će:

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

funkcija `diff()` derivirati funkciju:

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

Ako se ne želi derivirati po varijabli *x*, nego po *a*, onda treba pisati:

```
>> diff(f,a)
```

što vraća:

```
ans =
   cos(a*x)*x
```

Derivacija se može zadati i preko limesa, pa će

```
>> syms h n x
>> limit( (cos(x+h) - cos(x))/h,h,0 )
```

vratiti derivaciju cos(x) funkcije po definiciji:

```
ans =
  -sin(x)
```

dok će:

```
>> limit( (1 + x/n)^n,n,inf )
```

vratiti:

```
ans =
  exp(x)
```

Mogućnosti simboličkog toolbox-a mogu se sažeti u tablici:

Tablica 8.3: *Mogućnosti simboličkog toolbox-a*

Mogućnost	koja pokriva
Integralni i diferencijalni račun	Diferenciranje, integriranje, limese, sumacije, redove, Taylorove nizove
Linearna algebra	Determinante, svojstvene vrijednosti, dekompozicija, inverzi, kanonske forme simboličkih matrica
Pojednostavljenja	Metode pojednostavljivanja algebarskih izraza
Rješenja jednadžbi	Simboličko i numeričko rješavanje algebarskih i diferencijalnih jednadžbi
Posebne matematičke funkcije	Specijalne funkcije klasične matematičke primjene
Aritmetička točnost promjenljive preciznosti	Numeričko izračunavanje matematičkih izraza na bilo koju zadanu točnost
Transformacija	Laplace-ove, Fourier-ove, z-transformacije i pripadajućih inverznih transformacija

U primjeru s Tayler-ovim redom može se koristiti `pretty()` funkcija koja ljepše ispisuje 8 članova razvijenog reda:

```
>> syms x
>> f = 1/(5+4*cos(x))
>> T = taylor(f,8)
f =
```

```
   1/(5+4*cos(x))
T =
   1/9+2/81*x^2+5/1458*x^4+49/131220*x^6
>> pretty(T)

                    2             4       49        6
  1/9 + 2/81 x  + 5/1458 x  +  ---------  x
                                131220
```

U ovom toolboxu mogu se koristiti i naredbe koje Matlab nema, ali Mapple ima (stoga je nužno poznavati Mapple program!). Takva je na primjer `discrim()` funkcija za traženje diskriminante:

```
>> syms a b c x
>> maple('discrim', a*x^2+b*x+c, x)
ans =
   -4*a*c+b^2
```

Funkcija `dsolve()` izračunava simboličko rješenje običnih diferencijalnih jednadžbi. Jednadžbe se zadaju simboličkim izrazima koje sadrže slovo D za označavanje derivacije. Tako će simboli D2, D3, ... DN, odgovarati, drugoj, trećoj, ... N-toj derivaciji. Zavisne varijable imaju oznaku D ispred sebe (dakle, one same se ne smiju zvati 'd' ili 'D'), a pretpostavljeno ime nezavisne varijable je t. Ovo ime se može promijeniti navodeći neko drugo ime, kao posljednji ulazni argument.

Primjer 8.24: *Simboličko rješavanje dif. jednadžbi*

U ovom primjeru riješit ćemo običnu dif. jednadžbu sa i bez početnog uvjeta koristeći funkciju dsolve().

```
% Rijesiti Dy=1/(t-1) za razlicite pocetne uvjete
>> y=dsolve('Dy=1/(t-1)')
   % pocetni uvjet t=2
>> y1=dsolve('Dy=1/(t-1)','y(2)=-10')
>> y2=dsolve('Dy=1/(t-1)','y(2)=0')
   % pocetni uvjet t=4
>> y3=dsolve('Dy=1/(t-1)','y(4)=8')
```

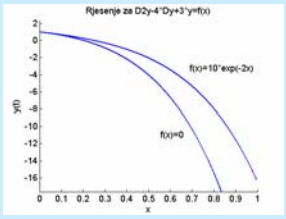
```
y = log(t-1)+C1
 y1 = log(t-1)-10
 y2 = log(t-1)
 y3 = log(t-1)-log(3)+8
```

Primjer 8.25: *Rješavanje dif. jed. drugog reda*

Funkcijom dsolve() mogu se simbolički riješiti i diferencijalne jednadžbe višeg reda.

```
% Riješiti jednadžbu y''-4*y'+3*y = f(x)
%         za f(x)=0 i  f(x)=10*exp(-2x)
% Koristiti gtext naredbu da se oznace osi
%
>> y1=dsolve('D2y-
4*Dy+3*y=0','y(0)=1','Dy(0)=-3','x')
>> tf=1.0
>> clf, hold on
>> ezplot(y1,[0,tf]);
>> gtext('f(x)=0'), pause
>> y2=dsolve('D2y-4*Dy+3*y=10*exp(-
2*x)','y(0)=1','Dy(0)=-3','x')
>> ezplot(y2,[0,tf])
>> gtext('f(x)=10*exp(-2x)')
>> title('Rjesenje za D2y-4*Dy+3*y=f(x)')
>> ylabel('y(t)')
>> zoom, hold off
```

```
y1 = 3*exp(x)-2*exp(3*x)

tf =  1

y2 =
2/3*exp(-2*x)+ 4/3*exp(x)-
exp(3*x)
```



Usporedbom analitičkog rješenja dobivenog s `dsolve()` i numeričkog dobivenog s `ode23()` Matlab funkcijom može se vidjeti da pogreška u numeričkom rezultatu naglo raste za x > 8.

Na isti način rješava se i sustav diferencijalnih jednadžbi:

Primjer 8.26: *Rješavanje sustava dif. jednadžbi*

Simboličko rješavanje sustava diferencijalnih jednadžbi također se rješava funkcijom dsolve().

```
>> fprintf('Sustav dif.jednadzbi \n')

>> [x,y]=dsolve('Dx=x+exp(t)','Dy=-x+3*
y+1' 'x(0)=2,y(0)=1')
```

```
Sustav dif.jednadzbi
x =
exp(t)*(2+t)

y =
1/12*exp(3*t)+5/4*
exp(t)+1/2*exp(t)*t-1/3
```

Po istom uzorku, rješavaju se i ostali matematički problemi u simboličkom obliku. Na koncu spomenimo samo integriranje.

Primjer 8.27: *Simboličko integriranje*

*Treba riješiti dvostruku integraciju funkcije: x*sin(y)+y*cos(x) nad konačnim intervalom pi < y < 2pi, 0 < x < pi. Za integraciju se koristi funkcija int().*

```
% nutarnji integral
>> inty=int('x*sin(y)+y*cos(x)','y',pi,2*pi)
>> intx=int(inty,'x',0,pi)  % vanjski integral
% pretvorba u numericki rezultat
>> numericki=numeric(intx)
```

```
inty =  -2*x+3/2*pi^2*cos(x)

intx = -pi^2

numericki =   -9.8696
```

Extended Symbolic Math Toolbox je dodatak simboličkom toolbox-u koji dopušta pristup Maple paketima, programiranju i svim Maple procedurama osim grafičkih. Tako Matlab uz snažnu numeričku ima i simboličku potporu.

PODACI I NJIHOVA OBRADBA 9

Iako se pod pojmom podataka podrazumijevaju bilo kakvi digitalni podaci u računalu, npr. realne varijable, vektori i sl. ovdje će biti više riječi o podacima koji će najčešće biti mnogobrojni, te pritom umjetno načinjeni, mjereni ili dohvaćeni iz datoteke. Matlab u svojim toolbox-ima ima rutine za prihvaćanje mjernih podataka iz različitih realnih uređaja (voltmetara, signal generatora, mosnih spojeva, univerzalnih mjernih instrumenata i sl.) kao i čitanje podataka iz različitih tipova datoteka. Često se za potrebe načinjenih algoritama podaci prvo simuliraju, umjetno stvaraju na nizu poznatih uzoraka, upisanih ili generiranih, a potom na realnim uzorcima. Još češće se, zahvaljujući snažnom toolbox-u s nazivom Simulink i njegovim mnogobrojnim dodacima (block setovima), simulacije obavljaju u virtualnom okruženju, prije nego se provedu u realnom s pomoću rtw-a (real time workshop-a), cilju svih nastojanja - obradbe podataka u realnom vremenu.

9.1 Čitanje i spremanje podataka u datoteke

Razlikujemo podatke koje se tiču varijabli u programu, od podataka koji se čitaju ili spremaju u datoteku na disku. Za rad s čuvanjem i dohvaćanjem matlab radnog okoliša koriste se `save` i `load()` naredbe:

SAVE - sprema varijable iz radnog prostora (workspace) na disk.

```
save IME %sprema sve varijable u binarnu tzv. MAT-datoteku IME.mat.
save  %stvara datoteku 'matlab.mat'.
save IME X   %sprema samo varijablu X.
save IME X Y Z  % sprema X, Y, i Z.
save IME.txt X Y Z -ascii  % sprema X, Y, i Z u IME.txt
```

Oznaka za sve '*' može se koristiti za sva imena koja zadovoljavaju uvjet.

LOAD – puni (eng. *load*) varijable radnog prostora s podacima spremljenim naredbom `save`.

```
load IME   %vraća sve varijable spremljene u datoteci IME.mat.
```

U slučaju da IME nije specificirano otvara se pretpostavljena datoteka s imenom 'matlab.mat'.

```
load IME.txt -ascii   %vraća sve varijable iz datoteke IME.txt u varijablu IME
A=load ('IME.txt');   %vraća sve varijable iz datoteke IME.txt u varijablu A
```

Za čitanje i spremanje formatiranih podataka na disk koriste se funkcije `fscanf()`, `fprintf()` i `textread()`, a za formatirano čitanje iz stringa funkcija `strread()`. Osim njih postoji još nekoliko manje važnih funkcija za rad s datotekama.

FSCANF – čita formatirane podatke iz datoteke.

```
[A,COUNT] = fscanf(FID,FORMAT,SIZE)
```

čita podatke iz datoteke određene FID identifikatorom datoteke, pretvara te podatke s obzirom na zadani string FORMAT, i vraća tako obrađene podatke u matricu A. COUNT je dozvoljeni (ali ne nužni) argument koji vraća broj elemenata koji su uspješno pročitani.
FID je cijeli broj dobiven iz naredbe FOPEN koja otvara datoteku.
SIZE je također dozvoljen, ali ne nužan argument, koji stavlja ograničenje na broj elemenata koji se čitaju iz datoteke. Ako nije specificiran onda se čita cijela datoteka, a ako se specificira, onda je moguće:

N čita najviše N elemenata u stupčasti vektor
inf čita najviše do konca datoteke
[M,N] čita najviše M * N elemenata koji popunjavaju MxN matricu u stupčastom redoslijedu (stupac po stupac). *N* može biti *inf*, ali ne može M.

Primjer 9.1: *Čitanje formatiranih podataka iz datoteke*

Prikazan je postupak od spremanja varijabli u datoteku do njenog otvaranja i čitanja formatiranih podataka iz nje pa do zatvaranja datoteke.

```
>> clear  %brisanje postojecih varijabli
>> x=2; y=33; s=1:4;   %varijable
>> save brojevi.txt -ascii  %spremanje varijabli x,y,s
>> A=fopen('brojevi.txt','r')  %otvaranje datoteke za citanje
>> B=fscanf(A,'%g',[1 inf]) %cita podatke iz datoteke
>> fclose(A)  %zatvaranje datoteke
```

```
A =     5
B =
1  2  3  4  2  33
ans =     0
```

FPRINTF - upisuje formatirane podatke u datoteku.

```
COUNT = fprintf (FID,FORMAT,A,...)
```

formatira podatke u realnom dijelu matrice A (i bilo kojeg dodatnog matričnog argumenta), pod kontrolom zadanog FORMAT stringa, i upisuje podatke u datoteku specificiranu FID identifikatorom. COUNT je broj okteta (byte-ova) koji su uspješno upisani u datoteku.
FID je cijeli broj dobiven kao identifikator u `fopen()` naredbi. Može također biti 1 za standardni izlaz (zaslon) ili 2 za standardnu pogrešku. Ako je FID izostavljen, onda izlaz ide na zaslon.
FORMAT je niz znakova (string) koji sadrži konverzijske specifikacije iz C programskog jezika. One uključuju znak %, dozvoljene zastavice (eng. *flags*), bilo kakvu širinu (eng. *width*) i polja preciznosti, proizvoljni podtip i konverzijske znakove d, i, o, u, x, X, f, e, E, g, G, c, kao i znak s.
Specijalni formati \n,\r,\t,\b,\f mogu se iskoristiti za tvorbu posebnih linefeed, carriage return, tab, backspace, i formfeed znakova.
Koriste se dvostruke lijeve kose crte ('\\') za tvorbu znaka '\' i '%%' za tvorbu znaka postotka '%'.

Primjer 9.2: *Upisivanje formatiranih podataka u datoteku*

Ovaj primjer načinit će tekst datoteku exp.txt koja će sadržavati skraćenu tablicu eksponencijalne funkcije:

```
>> x = 0:.1:1; y = [x; exp(x)];
>> fid = fopen('exp.txt','w');
>> fprintf(fid,'%6.2f  %12.8f\n',y);
>> fclose(fid);
>> A=load('exp.txt')
```

```
A =
         0    1.0000
    0.1000    1.1052
…
    0.9000    2.4596
    1.0000    2.7183
```

TEXTREAD – čita formatirane podatke iz tekst datoteke

A = `tekstread`('IME')
A = `tekstread` ('IME','',N)
A = `tekstread` ('IME','',param,value, ...)
A = `tekstread` ('IME','',N,param,value, ...)

čitaju numeričke podatke iz datoteke IME u jednu varijablu.

[A,B,C, ...] = `tekstread` ('IME','FORMAT')
[A,B,C, ...] = `tekstread` ('IME','FORMAT',N)
[A,B,C, ...] = `tekstread` ('IME','FORMAT',param,value, ...)
[A,B,C, ...] = `tekstread` ('IME','FORMAT',N,param,value, ...)

čita podatke iz datoteke IME u varijable A,B,C, itd. Tip svakog argumenta je specificiran FORMAT stringom. Broj ulaznih argumenata mora odgovarati broju pretvorbenih članova iz FORMAT stringa.

Ako je N zadan, onda se format string ponavlja N puta. Ako je N -1 ili nije zadan, onda `tekstread()` čita cijelu datoteku.
Ako su specificrani (param,value) parovi, onda korisnik posebno ugađa ponašanje funkcije `tekstread()`.

`tekstread()` traži podudarnost u skupinama i čini zadane pretvorbe. Postoje znakovi koji definiraju ograde ili međe (razdjelnike, engl. *delimiters*) među znakovima.

Podržane su sljedeće specifikacije:

%n - čita broj - float ili integer (vraća double array)
%5n čita do 5 znamenki ili do idućeg razdjelnika
%d - čita integer vrijednost s predznakom (vraća double array)
%5d čita do 5 znamenki ili do idućeg razdjelnika
%u - čita cjelobrojnu vrijednost (vraća double array)
%5u čita do 5 znamenki ili do idućeg razdjelnika
%f - čita realni broj (vraća double array)
%5f čita do 5 znamenki ili do idućeg razdjelnika
%s - čita string omeđen prazninom (vraća cellstr)
%5s čita do 5 znakova ili do iduće praznine
%q - čita (moguće s dvostrukim navodnicima omeđeni) string (i vraća cellstr)
%5q čita do 5 non-quote znakova ili do iduće praznine
%c - čita znak ili prazninu (vraća char array)
%5c čita do 5 znakova uključujući praznine
%[...] - čita znakove koji se podudaraju sa znakovima unutar zagrada sve do prvog znaka koji se ne podudara ili praznine (vraća cellstr)
%5[...] čita do 5 znakova
%[^...] - čita znakove koji se ne podudaraju sa znakovima između zagrada sve do prvog znaka koji se podudara (vraća cellstr)
%5[^...] čita do 5 znakova

Primjedba: Formatirani stringovi interpretiraju se kao sa naredbom `sprinf()` prije parsiranja (prepoznavanja i pretvorbe).
Na primjer,

```
textread('mydata.dat','%s\t')
```

tražit će znak tab ('\t'),a ne znak '\' iza kojeg slijedi znak 't'. Upotreba %* umjesto % u pretvorbi uzrokuje `tekstread()` da preskoči podudarne znakove na ulazu (i pritom ne nastaje izlaz za tu konverziju).

Primjeri:

Pretpostavimo da datoteka 'mydata.dat' sadrži podatke u sljedećem obliku:

```
Sally   Type1 12.34 45 Yes
Joe     Type2 23.54 60 No
```

Bill Type1 34.90 12 No

Čita svaki stupac u pojedinu varijablu:
```
[names,types,x,y,answer] = textread('mydata.dat','%s%s%f%d%s');
```

Čita prvi stupac u polje ćelija (preskačući ostatak linije)
```
[names]=textread('mydata.dat','%s%*[^\n]')
```

Čita prvi znak u polje znakova (preskačući ostatak linije)
```
[initials]=textread('mydata.dat','%c%*[^\n]')
```

Čita datoteku kao datoteku čvrstog formata preskačući pritom *double* podatke
```
[names,types,y,answer] = textread('mydata.dat','%9c%5s%*f%2d%3s');
```

Čita datoteku i traži literal 'Type'
```
[names,typenum,x,y,answer]=textread('mydata.dat','%sType%d%f%d%s');
```

Čita m-datoteku u ćeliju polja stringova
```
file = textread('fft.m','%s','delimiter','\n','whitespace','');
```

Ako se želi pročitati sve podatke iz tekst datoteke s razdjelnicima, koristi se jedan izlazni argument, prazni format string i prikladni razdjelnik (u našem slučaju razdjelnik (delimiter) je zarez).

Na primjer, pretpostavimo da 'data.csv' sadrži:
```
1,2,3,4
5,6,7,8
9,10,11,12
```

Pročitati čitavu matricu u jednoj varijabli:
```
[data] = textread('data.csv','','delimiter',',');
```

Pročitati prva dva stupca u dvije varijable:
```
[col1, col2] = textread('data.csv','%n%n%*[^\n]','delimiter',',');
```

Za datoteke s praznim mjestima koristi se parametar 'emptyvalue'
Pretpostavimo da datoteka 'data.csv' sadrži:
```
1,2,3,4,,6
7,8,9,,11,12
```

Pročitati ovu datoteku koristeći NaN na praznim mjestima:
```
[data] = textread('data.csv','','delimiter',',','emptyvalue',NaN);
```

Za datoteke s razdjelnicima, postoje posebne funkcije `dmlread`() i `dlmwrite`()

DLMREAD čita ASCII datoteku s razdjelnicima.

RESULT = `dlmread`(IME,DELIMITER)

čita numeričke podatke iz ASCII datoteke IME koristeći razdjelnik DELIMITER. Rezultat se sprema u RESULT.

DLMWRITE piše ASCII datoteku s razdjelnicima između podataka.

`dlmwrite`(IME,M,DLM)

piše matricu M u datoteku IME koristeći znak DLM kao razdjelnik.

Na sličan način `wk1write()` i `wk1read()` spremaju i čitaju podatke u tabličnom obliku u Lotus (ili Excell) formatu.

Posebna funkcija za rad s formatiranim podacima iz stringa je:

STRREAD čita formatirane podatke iz stringa, na sličan način kao i s naredbom `textread()`.

Primjer 9.3: *Čitanje formatiranih podataka iz stringa*

Prvo će se formatirani podaci upisati u string 's' pomoću naredbe sprintf(), a onda će se funkcijom strread() rastaviti string 's' na dijelove odvojene zarezima i te dijelove pridružiti varijablama a,b,c

```
%sprintf() upisuje formatirane podatke u string 's'
>> s = sprintf('a,1,2\nb,3,4\n')

>> [a,b,c] = strread(s,'%s%d%d','delimiter',',')
```

```
s =   a,1,2
      b,3,4
a =   'a'
      'b'
b =    1
       3
c =    2
       4
```

9.2 Formati Matlab datoteka

U tablici su prikazani formati datoteka s naredbama za čitanje i tipovima podataka koje vraćaju.

Tablica 9.1: *Formati podataka*

Format	Naredba	Vraća
MAT, MATLAB workspace	`load()`	Varijable u datoteci
CSV, Comma separated numbers	`csvread()`	Double array
DAT, Formatted text	`importdata()`	Double array
DLM, Delimited text	`dlmread()`	Double array
TAB, Tab separated text	`dlmread()`	Double array

Tablica 9.2: *Spreadsheet (tablični kalkulator) formati*

Format	Naredba	Vraća
XSL, Excell worksheet	`xslread()`	Double array i cell array
WK1, Lotus 123 worksheet	`wk1read()`	Double array i cell array

Tablica 9.3: *Znanstveni (Scientific data) podatačni formati*

Format	Naredba	Vraća
CDF, Common Data Format	`cdfread()`	Cell array od CDF recorda
FITS, Flexible Image Transport System	`fitsread()`	Double array
HDF, Hierarchical Data	`hdfread()`	Data set

Tablica 9.4: *Filmski (movie) formati*

Format	Naredba	Vraća
AVI, Movie	`aviread()`	MATLAB movie

Tablica 9.5: *Slikovni (image) formati*

Format	Naredba	Vraća
TIFF, TIFF slika	`imread()`	Truecolor, grayscale, indeksirana
PNG, PNG slika	`imread()`	Truecolor, grayscale, indeksirana
BMP, BMP slika	`imread()`	Truecolor, grayscale, indeksirana
HDF, HDF slika	`imread()`	Truecolor, grayscale, indeksirana
JPEG, JPEG slika	`imread()`	Truecolor, grayscale
GIF, GIF slika	`imread()`	Indeksirana slika
PCX, PCX slika	`imread()`	Indeksirana slika
XWD, XWD slika	`imread()`	Indeksirana slika
CUR, Cursor slika	`imread()`	Indeksirana slika
ICO, Icon slika	`imread()`	Indeksirana slika
RAS, Sun raster slika	`imread()`	Truecolor, indeksirana
PBM, PBM slika	`imread()`	Grayscale slika
PGM, PGM slika	`imread()`	Grayscale slika
PPM, PPM slika	`imread()`	Truecolor

Tablica 9.6: *Zvučni (audio) formati*

Format	Naredba	Vraća
AU, NeXT/Sun sound	`auread()`	zvuk i brzina uzrokovanja
SND, NeXT/Sun sound	`auread()`	zvuk i brzina uzrokovanja
WAV, Microsoft Wave sound	`wavread()`	zvuk i brzina uzrokovanja

9.3 Obradba

Neka nam podaci za obradbu bude neki zvučni zapis. Matlab ima funkciju `sound()` koja izvodi (na kompjutorske zvučnike) zapisani zvuk. Tako će:

```
>> load handel
>> sound(y,Fs)
```

odsvirati korsku izvedbu jednog odlomka Handel-ovg "Aleluja". Zvučni zapis spremljen je u 'handel.mat' datoteci i ima *y* podatke amplitude zvuka i varijablu *Fs* koja govori o frekvenciji uzorkovanja podataka (u ovom slučaju 8192), kako bi se zvuk mogao vjerno reproducirati.

Ako imamo spremljenu npr. wav datoteku, onda se ona može pročitati s funkcijom `wavread()`:

```
>> [x,fs,bits]=wavread('cow.wav');
```

gdje prve dvije izlazne varijable odgovaraju već opisanima, a varijabla 'bits' nakon poziva sadrži broj bitova po uzorku, tj. finoću digitalne pretvorbe analognog signala (više bitova točniji zapis). Učitani zvuk može se reproducirati na opisani način:

```
>> sound(x,fs)
```

pa će u slučaju 'caw.wav' biti reproducirano mukanje krave.

Najčešća je statistička obrada, koja uključuje srednju vrijednost, varijancu (standardnu devijaciju), srednju vrijednost amplitude, srednju snagu signala i broj prolazaka kroz nulu. Ako je niz podataka
$x(n)$, $n = 1, \ldots, N$, onda su formule za njegovu obradu sljedeće:

$$\text{mean} = \mu = \frac{1}{N}\sum_{n=1}^{N} x(n)$$

$$\text{s tan dard deviation} = \sigma = \left[\frac{1}{N}\sum_{n=1}^{N}(x(n)-\mu)^2\right]^{1/2}$$

$$\text{average magnitude} = \frac{1}{N}\sum_{n=1}^{N}|x(n)|$$

$$\text{average power} = \frac{1}{N}\sum_{n=1}^{N}(x(n))^2$$

pa će program za obradbu zvučnog zapisa izgledati ovako:

Primjer 9.4: *Obrada zvučnog zapisa*

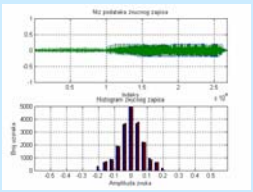
Skripta 'zvuk.m' koja čita zvučni zapis sa zadanim imenom (u ovom slučaju cow.wav) i računa različite statistike zvuka.

```
%pocetak pisanja skripte 'zvuk.m'
file = input('Utipkajte ime datoteke sa zvukom: ','s');
[x,fs,bits] = wavread(file);
n = length(x);
% Obradba
fprintf('\n')
fprintf('Digitalna statistika \n\n')
fprintf('uzorci (samples): %.0f \n',n)
fprintf('Frekvencija uzorkovanja (sampling >>
frequency): %.1f \n',fs)
fprintf('Broj bitova po uzorku: %.0f \n',bits)
fprintf('Srednja vrijednost: %.4f \n',mean(x))
fprintf('Standardna devijacija: %.4f \n', std(x))
fprintf('Prosjecni iznos amplitude: %.4f \n',
mean(abs(x)))
fprintf('Srednja snaga: %.4f \n', mean(x.^2))
prod = x(1:n-1).*x(2:n);
crossings = length(find(prod<0));
fprintf('Presjecanje nul osi: %.0f \n', crossings)
subplot(2,1,1),plot(x),...
axis([1 n -1.0 1.0]),...
title('Niz podataka zvucnog zapisa'),...
xlabel('Indeks'), grid,...
```

```
Utipkajte ime datoteke
sa zvukom: cow

Digitalna statistika

uzorci (samples):
26848
Frekvencija
uzorkovanja (sampling
>> frequency): 22050.0
Broj bitova po uzorku:
8
Srednja vrijednost:
0.0007
Srednja vrijednost:
0.0006
Standardna devijacija:
0.0648
Standardna devijacija:
0.0592
Prosjecni iznos
amplitude: 0.0439
Prosjecni iznos
amplitude: 0.0403
```

```
subplot(2,1,2),hist(x,linspace(-1,1,51)),...
axis([-0.6,0.6,0,5000]),...
title('Histogram zvucnog zapisa'),...
xlabel('Amplituda zvuka'),...
ylabel('Broj uzoraka'),grid
%kraj pisanja skripte 'zvuk.m'

%poziv skripte 'zvuk.m'
>> zvuk
%Utipkajte ime datoteke sa zvukom: cow
```

```
Srednja snaga: 0.0042
Srednja snaga: 0.0035
Presjecanje nul osi:
2122
```



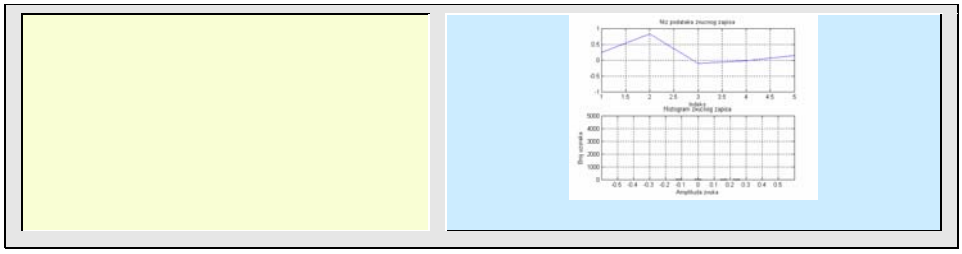
Obično se provjera programa obavlja na podacima koje samo generiramo tako da obradbu možemo ručno provjeriti.

Primjer 9.5: *Testiranje programa zvučnog zapisa*

Može se zvučni signal od samo 5 uzoraka (x = [0.25 0.82 -0.11 -0.02 0.15]) i spremiti u datoteku 'test.wav'. Ručno ćemo izračunati potrebne vrijednosti.
Srednja vrijednost = μ = 1/5 (0.25 + 0.82 − 0.11 − 0.02 + 0.15) = 0.218
Standardna devijacija = σ = 1/5 [(0.25 − μ)^2 + (0.82 − μ)^2 + (−0.11 − μ)^2 + (−0.02 − μ)^2 + (0.15 − μ)^2]$^{1/2}$ = 0.365
Srednja amplituda = 1/5 (|0.25| + |0.82| + | − 0.11| + | − 0.02| + |0.15|) = 0.270
Srednja snaga =1/5 (0.25^2 + 0.82^2 + (−0.11)^2 + (−0.02)^2 + 0.15^2) = 0.154
Broj presjecišta apscise = 2

```
>> x = [0.25 0.82 -0.11 -0.02 0.15]
>> wavwrite(x,'test.wav')

%Pozivom skripta dobiju se
%rezultati koje je moguće lako
%provjeriti:
>> zvuk
```

```
x =
0.2500   0.8200   -0.1100   -0.0200   0.1500

Utipkajte ime datoteke sa zvukom: test

Digitalna statistika

uzorci (samples): 5
Frekvencija uzorkovanja (sampling >>
frequency): 8000.0
Broj bitova po uzorku: 16
Srednja vrijednost: 0.2180
Standardna devijacija: 0.3648
Prosjecni iznos amplitude: 0.2700
Srednja snaga: 0.1540
Presjecanje nul osi: 2
```



U mjernim podacima (dobivenih iz mjernih instrumenata) redovito je prisutna i pogreška, superponirani, neželjeni signal kojeg zovemo šum (engl. *noise*). Stvarno zapisani realni signal je, dakle:

$$x = s + n$$

gdje je *s* signal, a *n* – šum.

U nedostatku mjernih uređaja i za potrebe testiranja algoritama, signal se najčešće generira harmonijskim funkcijama sinusa i kosinusa, a šum s pomoću generatora slučajnih brojeva. Slučajni brojevi u Matlabu se mogu generirati kao:

`rand()` - uniformni (jednolika raspodjela)
`randn()` - normalni (Gauss-ova raspodjela)

Ako se želi generirati slučajni uniformni niz od n podataka u granicama [xmin, xmax], onda je to lako načiniti s pomoću formule:

```
x = (xmax - xmin)*rand(1,n) + xmin
```

U slučaju normalne raspodjele podataka, funkcija `rand()` se zamjenjuje sa `randn()`.
Odnos signala i šuma redovito se izražava u decibelima (dB) kao logaritamski odnos varijanci signala i šuma:

$$SNR = 10 \log_{10} \sigma^2_s/\sigma^2_n = 20 \log_{10} \sigma_s/\sigma_n$$

S obzirom da varijanca sinusnog oblika jednaka $\sigma^2_s = A/2$ lako je izračunati da je za amplitudu A=1 odnos signala i šuma SNR=17 dB.
Česta su i djelovanja različitih filtera na signal zagađen šumom. Jedan od najjednostavnijih je tročlani filtar s pomakom sredine (engl. *3-point moving average filter*) koji radi po formuli:

$$y(k) = 1/3\ [x(k) + x(k-1) + x(k-2)]$$

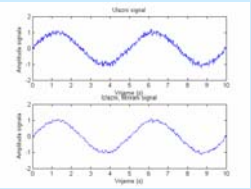
Dakle, signal se usrednjava tri po tri točke s pomakom od početka do konca. Sljedećim program može se uočiti djelovanje filtra na oblik signala i SNR:

Primjer 9.6: *Djelovanje filtera na oblik signala*

Pomoću filtera djelovat će se na signal zagađen šumom u svrhu smanjivanja šuma. Vidi se da se filtracijom odnos poboljšava (u korist signala). Treba primijetiti da ćemo pozivom skripte filtar.m svaki put dobivati drugačije (makar slične) rezultate, zbog slučajnog karaktera koji je u program ugrađen (šum generiran slučajnim brojem).

```
% FILTAR.m – program za jednostavnu obradbu
%signala
% Generiranje signala sa sumom i filtriranje
t = linspace(0,10,512); % vremenska baza
s = sin(2*pi/5*t); % signal
n = 0.1*randn(size(t)); % sum, std dev 0.1
x = s + n; % signal + sum
disp('Odnos ulaznog signala i suma (SNR),dB')
% ulaz SNR, dB
SNR_ulaz = 20*log10(std(s)/std(n))
% initializacija izlaznog signala
y = zeros(size(t));
% filtriranje
y(1) = x(1); y(2) = (x(2)+x(1))/2;
for k = 3:length(t);
   y(k) = (x(k)+x(k-1)+x(k-2))/3;
end
% analiza filtriranog signala
disp('Odnos izlaznog signal i suma (SNR), dB')
% izlaz SNR, dB
SNR_izlaz = 20*log10(std(s)/std(y-s))
% crtez signala
subplot(2,1,1),plot(t,x), xlabel('Vrijeme (s)')
ylabel('Amplituda signala '), title('Ulazni signal')
subplot(2,1,2),plot(t,y), xlabel('Vrijeme (s)')
ylabel(' Amplituda signala')
title('Izlazni, filtrirani signal')

%Pozivom:
>> filtar
```

```
Odnos ulaznog signala
i suma (SNR),dB

SNR_ulaz =
   17.4577

Odnos izlaznog signal
i suma (SNR), dB

SNR_izlaz =
   21.7848
```



Prijelaz na složeniju obradu signala vodi nas na Fourier-ovu transformaciju, s vremenske prelazi se u frekvencijsku domenu. Ne ulazeći u detalje, ovdje će biti prikazan samo jedan primjer, temeljen na podacima iz Matlab-ovog demo programa o sunčevim pjegama (`fftdemo`). MATLAB koristi oznake izvedene od matrične teorije, pa s obzirom da se tamo ne koristi indeks 0, nego je početni indeks 1, i ovdje će se stoga

umjesto uobičajene matematičke oznake y_k koristiti y_{k+1}. Konačna ili diskretna Fourier-ova transformacija kompleksnog vektora **y** sa n elemenata y_{j+1}; $j = 0; \ldots n-1$ je drugi kompleksni vektor **Y** sa *n* elemenata Y_{k+1} za koji vrijedi:

$$Y_{k+1} = \sum_{j=0}^{n-1} y_{j+1} \cdot e^{-2ijk\pi/n}, \; k = 0,\ldots,n-1$$

Eksponencijale su svi kompleksni n-tog reda korijeni jedinične kružnice, tj. sve su potencije od

$$\omega = e^{-2\pi i/n} = \cos\delta - i\sin\delta$$

gdje je: $\delta = 2\pi/n$.

Transformacija se također može izraziti s matrično-vektorskom predodžbom:

$$\mathbf{Y} = \mathbf{F}\mathbf{y}$$

Gdje je **F** matrica konačne Foureir-ove transformacije i ima elemente:

$$f_{k+1,j+1} = \omega_{jk}$$

Kompleksno konjugirana matrica **F** zadovoljava
$\mathbf{F}^H\mathbf{F} = n\mathbf{I}$, pa je $\mathbf{F}^{-1} = 1/n\,\mathbf{F}^H$ što omogućuju invertiranje Fourirerove transformacije:

$$\mathbf{y} = 1/n\,\mathbf{F}^H\mathbf{Y}$$

$$y_{j+1} = \frac{1}{n}\sum_{k=0}^{n-1} Y_{k+1} e^{2ijk\pi/n}, \; j = 0,\ldots,n-1$$

U Matlabu se po gornjim formulama za zadani *n* može izračunati Fourierova transformacija **F** kao:

```
>> omega = exp(-2*pi*i/n);
>> j = 0:n-1;
>> k = j'
>> F = omega.^(k*j)
```

Umnožak *k* i *j* predstavlja vanjski umnožak vektora koji daju *(n x n)* matricu. Međutim, ugrađena `fft()` funkcija načini konačnu FT na svakom stupcu matrice u argumentu, tako da je jednostavnija i brža. S obzirom na brzi algoritam još se zove FFT (engl. *Fast Fourier Transformation*). Na primjer:

```
>> F = fft(eye(n))
```

naći će brzu F. transformaciju na kvadratnoj (nxn) matrici identiteta I, što će dati identičan rezultat kao i par redaka gornjeg programa.

Stoljećima su ljudi primjećivali da izgled sunca nije konstantan ili uniforman, već da se ciklički na njemu pojavljuju tamna područja, koja su nazvana sunčevim mrljama. Njihova aktivnost povezana je s vremenom i nekim Zemljinim fenomenima. 1848 godine Rudolf Wolfer definirao je pravilo za bilježenje ovih mrlja, njihove veličine i broja u jedinstvenom indeksu. Tako se zahvaljujući višegodišnjem bilježenju poznata je sunčeva aktivnost unatrag do 1700 godine. Danas se indeks nadopunjava mjerenjima mnogih astronoma koji se koordinira preko ustanove "The Sunspot Index Data Center" na Royal Observatory u Belgiji. (http://www.astro.oma.be/SIDC/index.html). Tekst datoteka 'sunspot.dat' u Matlab-ovom folderu demo programa ima dva stupca brojeva. Prvi stupac su godine od 1700. do 1987., a drugi stupac su srednje vrijednosti Wolfer-ovih sunčanih mrlja za svaku godinu.

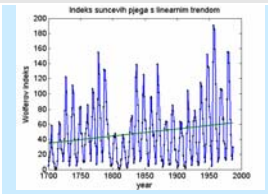
Primjer 9.7: *Prikaz grafa sunčevih pjega*

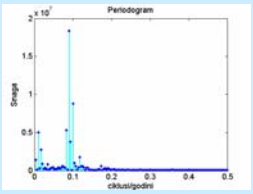
Matlabova datoteka 'sunspot.dat' sadrži srednje vrijednosti Wolfer-ovih sunčanih mrlja za svaku godinu od 1700. do 1987. Podaci su dakle zapisani za 288 godina. Postoji slabi rast trenda podataka, što se vidi ako se načini 'fitting' po metodi najmanjih kvadrata, prikazano na prvoj slici. Već se na njoj mogu uočiti ciklička priroda pojave, gdje se svakih otprilike 10 godina pojavljuju maksimumi. Točna analiza provest će se pretvorbom pojave u frekvencijsku domenu, upotrebom FFT. Prvo treba oduzeti linearni trend, a onda na podacima načiniti FFT. Kvadrirani kompleksni iznos od Y zove se snaga, a graf snage u odnosu na frekvenciju zove se periodogram. Frekvencija je indeks polja umjeren (engl. scaled) sa n, brojem točaka podataka. Pošto je vremenski korak jednak jednoj godini, jedinica za frekvenciju su ciklusi po godini (druga slika). Najveća snaga pojavljuje se blizu frekvencije jednake 0.09 ciklusa/godini. Ako se želi saznati odgovarajući period u jedinicama godine/ciklus, onda treba povećati (engl. zoom) dio osi i koristiti recipročnu frekvenciju za oznake na x-osi (treća slika). Kao što se očekivalo, postoji jako izraženi ciklus dužine oko 11 godina. To pokazuje da se u posljednjih 300 godina, period ciklusa sunčevih mrlja je nešto malo preko 11 godina.

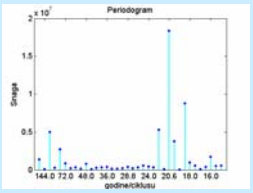
```
>> load sunspot.dat
>> t = sunspot(:,1)';
>> wolfer = sunspot(:,2)';
>> n = length(wolfer)
%prva slika
>> c = polyfit(t,wolfer,1);
>> trend = polyval(c,t);
>> plot(t,[wolfer; trend],'-',t,wolfer,'k.')
>> xlabel('year')
```

```
n =   288
```

```
>> ylabel('Wolferov indeks')
>> title('Indeks suncevih pjega s linearnim
trendom')

>> y = wolfer - trend;
>> Y = fft(y);
%Prvi Fourier-ov koeficijent, Y(1), treba izbrisati, jer
%se oduzimanjem trenda dogodilo da je Y(1) =
%sum(y)=0;
>> Y(1) = [];
%druga slika
>> pow = abs(Y(1:n/2)).^2;
>> pmax = 20e6; f = (1:n/2)/n;
>> plot([f; f],[0*pow; pow],'c-', f,pow,'b.', ...
'linewidth',2,'markersize',16)
>> axis([0 .5 0 pmax]), xlabel('ciklusi/godini')
>> ylabel('Snaga'), title('Periodogram')
%treca slika
>> k = 1:36; pow = pow(k);
>> ypk = n./k(2:2:end); % godine/ciklusu
>> plot([k; k],[0*pow; pow],'c-',k,pow,'b.', ...
'linewidth',2,'markersize',16)
>> axis([0 max(k)+1 0 pmax])
>> set(gca,'xtick',k(2:4:end))
>> xticklabels = sprintf('%4.1f|',ypk);
>> set(gca,'xticklabel',xticklabels)
>> xlabel('godine/ciklusu'), ylabel('Snaga')
>> title('Periodogram')
```







9.4 Matlab-ovi programski paketi – toolboxes

Izgrađen na dobroj osnovi matričnog laboratorija, Matlab najveću popularnost zahvaljuje brojnošću i stalnom izgradnjom svojih toolbox-a, zaokruženih programskih cjelina nekog područja iz znanosti ili tehnike. Grupirani u cjeline koje se međusobno isprepliću, možemo nabrojiti neke od njih:

Tablica 9.7: *Toolbox općeg značenja*

Toolbox	Opis
toolbox\symbolic	Simbolička matematika
toolbox\pde	Parcijalne diferencijalne jednadžbe

toolbox\splines	Spline aproksimacije

Tablica 9.8: *Toolbox prikupljanja podataka, mjerenja i obradbe*

Toolbox	Opis
daq\daq	Prikupljanje podataka (Dana Acuisition)
instrument\instrument	Upravljanje instrumentima
signal\signal	Obrada signala
filterdesign\filterdesign	Projektiranje filtera
images\images	Obrada slike
wavelet\wavelet	Valovi
map\map	Kartografija

Tablica 9.9: *Toolbox upravljanja i regulacije*

Toolbox	Opis
daq\daq	Prikupljanje podataka (Dana Acuisition)
instrument\instrument	Upravljanje instrumentima
signal\signal	Obrada signala
filterdesign\filterdesign	Projektiranje filtera
images\images	Obrada slike
wavelet\wavelet	Valovi
map\map	Kartografija

Tablica 9.10: *Toolbox upravljanja i regulacije*

Toolbox	Opis
control\control	Upravljanje sustava
fuzzy\fuzzy	Fuzzy logika
toolbox\ncd	Nelinearno upravljanje
nnet\nnet	Neuronske mreže
toolbox\optim	Optimizacija
toolbox\robust	Robusno upravljanje
lmi\lmilab	LMI upravljanje
ident\ident	Identifikacija sustava
mpc\mpccmds	Upravljanje predviđanjem modela
mutools\commands	Mu-Analiza i sinteza

Tablica 9.11: *Toolbox financijske obradbe*

Toolbox	Opis
finance\finance	Financijski alati
finderiv\finderiv	Financijske derivacije

ftseries\ftseries	Financijski vremenski nizovi
garch\garch	GARCH
toolbox\stats	Statistika
datafeed\datafeed	Prikupljanje podataka
curvefit\curvefit	Aproksimacija krivulja (fitting)
toolbox\exlink	Veza Matlab-Excel
matlabxl\matlabxl	MATLAB Excel graditelj (Builder)
database\database	Baza podatka

Tablica 9.12: *Toolbox simulacijskog alata (Simulink) s dodacima*

Toolbox	Opis
simulink\simulink	Simulink
aeroblks\aeroblks	Zrakoplovni dodatak (Aerospace Blockset)
simdemos\automotive	Automobilski modeli
simulink\blocks	Simulink biblioteka blokova
commblks\commblks	Kommunikacijski dodatak
simulink\dee	Editor diferencijslnih jednadžbi
mech\mech	Mehaničke simulacije
powersys\powersys	Elektrotehničke simulacije
stateflow\stateflow	Tijek i stanja algoritama
database\vqb	Visualne query builder funkcije
dspblks\dspblks	DSP dodatak
cdma\cdma	CDMA referentni dodatak
toolbox\fixpoint	Dodatak mat. čvrste točke
ccslink\ccslink	MATLAB Link za Code Composer Studio(tm)
mbc\mbc	Kalibracija modela
mpc555dk\mpc555dk	Mikrokontroleri za Motorola MPC555
toolbox\reqmgt	Zahtjevani Management Interface
toolbox\rptgenext	Simulink Report Generator
toolbox\runtime	MATLAB Runtime Server razvojni Kit
tic6000\tic6000	Mikrokontroleri za Texas Instruments
xpc\xpc	xPC temeljni konstruktor objekata
rtw\rtw	Real-Time Workshop
rtw\accel	Simulink ubrzavač
vr\vr	Virtualna stvarnost (Virtual Reality)

Tablica 9.13: *Toolbox programskih alata*

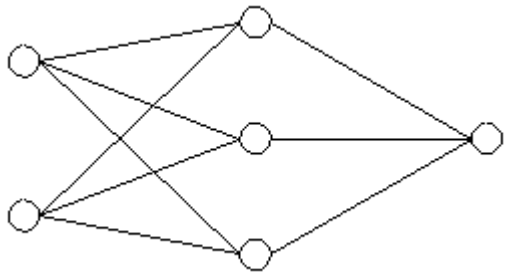
Toolbox	Opis
toolbox\compiler	MATLAB compiler
combuilder\combuilder	MATLAB COM Builder.
webserver\webserver	MATLAB Web Server.

Svakako treba još jednom navesti mnoštvo primjera spremljenih u datoteci matlab\demos koji se preporučuju za pokretanje kod početnog upoznavanja s nekim toolbox-om. Međutim, opće je mišljenje da osim fascinirajućeg efekta ti primjeri nemaju veću korisnost za početnika. Čak i priručnici (obično u .pdf) formatu koji dolaze uz toolbox više su referentni, nego edukacijski materijal. Štoviše, toolbox-i su zamišljeni da budu pomoć profesionalnim korisnicima tog područja, pa je potrebno prvo naučiti teoriju, a tek ona preći na programe. Preporuka je mnogih autora da se znanje toolboxa stekne malim programima, a tek postupno pređe na složenije, one iz priručnika ili demo programa. Ovdje će, zbog prevelikog broja toolboxa, biti pokazana samo tri primjera s načelnim savjetima što i kako raditi.

Primjer korištenja toolbox-a neuronskih mreža.

Ovaj primjer koristi objekt definiran u neuronske mrežama (toolbox nnet). O njemu u priručniku, a pogotovo ne demo programima nema ni spomena, a na njemu se temelje svi programi. Riječ je o *network* objektu. Za njegovo stvaranje koriste se `newp()` i `newff()` funkcije na visokoj razini, koja je za početnika preteška. Stoga je jednostavnije poći od najniže razine, samog objekta i njegovih svojstava, te načiniti slabu, ali jednostavnu neuronsku mrežu koju poznajemo "iznutra". Evo koraka za takvu izgradnju:

1. Mrežni objekt neka se zove *net*:
>> net = `network`;
2. Neka se izgradi samo jedan ulazni sloj:
>> net.`numInputs` = 1;
3. Neka se definira ulazni sloj (broj njegovih ulaza) :
>> net.`inputs`{1}.`size` = 2; % broj{1} govori da se radi o prvom (i jedinom) ulaznom sloju
4. Neka se definiraju ostali slojevi:
>> net.`numLayers` = 2;
>> net.`layers`{1}.`size` = 3;
>> net.`layers`{2}.`size` = 1; % broj 3, odnosno 1, govore o broju neurona doticnog sloja (sakriveni sloj ima 3, a vanjski 1 neuron)
To bi bez definiranih međusobnih spojeva neurona dalo neuronsku mrežu otprilike ovakvu:



Slika 9.1: *Shema neuronske mreže*

5. Neka se slojevi povežu (1 – povezuju se, 0 – ne povezuju):
```
>> net.inputConnect(1) = 1;  % 1. sloj je spojen s ulaznim
>> net.layerConnect(2, 1) = 1;
>> net.outputConnect(2) = 1;
>> net.targetConnect(2) = 1; % izlazni sloj spjen je s drugim
```
6. Neka se postave težinske funkcije slojeva:
```
>> net.layers{1}.transferFcn = 'logsig';
>> net.layers{2}.transferFcn = 'purelin';
```
7. Neka se postave pomaci (engl. *bias*):
```
>> net.biasConnect = [ 1 ; 1];
```
8. Neka se mreža inicijalizira:
```
>> net = init(net);
>> net.initFcn = 'initlay';
```

I mreža je spremna za učenje. Sad bi trebalo krenuti na funkcije `train()` ili `adapt()` i pokazati, kako mreža radi. No, to nije bila svrha ovog primjera. U njemu se željelo pokazati kako pristupati toolbox funkcijama – graditi ih od temelja. Slično, kao korisniku nakon naučene handle grafike odjednom sve složene 2-D i 3-D grafičke funkcije postanu bliske.

Primjer 9.8: *Korištenje toolbox-a za optimiziranje*

U ovom primjeru koristeći funkcije iz optimizirajućeg toolbox-a, pokazati kako se s jednostavnim primjerom mogu naučiti važne stvari.
Optimizacija bez ograničenja. Treba naći skup vrijednosti [x1, x2] koje minimiziraju funkciju $f(x_1,x_2)$*:* $F(x_1,x_2)=e^{x1}(4\,x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)$.
Proširenje primjera nastaje ako se uzme ista funkcija, ali ovog puta uz ograničenja: $1.5 + x_1x_2 - x_1 - x_2 <= 0$, $-x_1x_2 - 10 <= 0$
U m-datoteku (fun2.m) sad treba spremiti osim funkcije još i ograničenja:

```
%Prvo se spremi funkcija u m-datoteku:
function f=fun(x)
f=exp(x(1))*(4*x(1).^2+2*x(2)*x(2)+4*
x(1)*x(2)+2*x(2) + 1);

%Zatim se napisu pretpostavljena rjesenja:
X0=[-1,1];
%I pozove funkcija fminunc():
>> X=fminunc('fun',X0)
a=inline('exp(x1)*(4*x1^2+2*x2^2+4*
x1*x2+ 2*x^2+1)')
%Rjesenje se uvrsti u pocetnu jednadzbu
>> fun(X)  %što daje minimalni iznos
%funkcije za dobivena rjesenja.

function [f,g]=fun2(x)
f=exp(x(1))*(4*x(1).^2 + 2*x(2)*x(2) +
4*x(1)*x(2) + 2*x(2) + 1);
g(1)=1.5 + x(1)*x(2)- x(1)- x(2);
g(2)=-x(1)*x(2)-10;

%Za ista pretpostavljena rjesenja poziva se
>> X0=[-1,1];
>> X=constr('fun2',X0)
%ili s novom funkcijom fmincon()
%bogatijih mogućnosti.

%Ponovnim uvrstenjem rjesenja u jedn.
>> fun2(X)
%Vidi se da je rjesenje uz ogranicenja
%drugacije od onog s ogranicenjima.
```

```
Warning: Gradient must be
provided for trust-region method;
   using line-search method
instead.
Optimization terminated
successfully:
Current search direction is a
descent direction, and magnitude
of
directional derivative in search
direction less than
2*options.TolFun

X =    0.5000   -1.0000

ans =
    1.302793198827844e-010




X =
-9.547405025   1.047405025


ans =
   0.02355037962418
```

Primjer 9.9: *Složena rješenja pomoću funkcija niže razine*

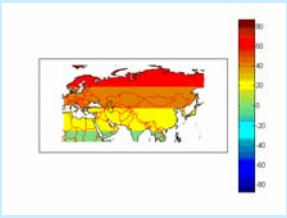
U ovom primjeru koristeći funkcije iz mapping toolbox-a, treba pokazati kako se složena kartografska rješenja mogu naći jednostavnim pozivima funkcija niže razine. Proučavanje GUI rješenja u toolbox demonstraciji jednostavno nije moguće.

```
% postaviti početne veličine (matricu mreže
%(latitudes)).
>> xgrid = [ 2.5: 5: 360 ]' ;
>> ygrid = [ 87.5: -5: -87.5 ]' ;
>> nx = length(xgrid)
```

```
nx =
    72
```

```
>> ny = length(ygrid)
>> xval = ones(ny,1)*(xgrid');
>> yval = ygrid*ones(1,nx);

% Zadati projekciju mapa, u ovom slučaju
%cilindričnu. Zadati domenu.
>> axesm('eqdcylin','MapLatLimit',[ 10 80 ],
'MapLonLimit', [0 150])
% Osjenčati područja od interesa.
>> contourfm( yval, xval, yval ); %ravninsko
%sjencanje
% primjena oceanske maske.
>> h3 = displaym(worldlo('oceanmask'));
% za neke worldlo atribute:
>> set(h3, 'FaceColor', [ 1 1 1 ] );
% set(h3, 'EdgeColor', 'none' );
>> caxis( [ -87.5 87.5 ] ), colorbar
```

```
ny =
    36
```



Kako vidite, zahvaljujući Matlab-u svijet Vam je na dlanu. Iskoristite to!