Skripte za predmet
Programiranje 1 (smer M1, M2)
Departman za matematiku i informatiku
PMF Novi Sad, 31.12.2011.

Sadržaj

Unošenje naredbi preko komandne linije

Osnovni i najprostiji način rada u Matlab-u je preko komandne linije. Iako nam je krajnji cilj pisanje programa, najpre ćemo se upoznati sa nekim osnovnim naredbama, unoseći ih preko komandne linije. Dakle, nakon što unesemo naredbu u komandnoj liniji i pritisnemo taster Enter, naredba se izvršava, a u narednim redovima ispisuje se (eventualni) rezultat.

Na primer, ako unesemo

```
>> 1+4
```

u narednim redovima biće ispisano sledeće.

```
ans =
     5
```

Promenljive i dodela

Prvi novi koncept sa kojim se srećemo jesu promenljive. Promenljive koje ovde pominjemo nisu isto što i promenljive u matematici, štaviše – suštinski su drugačije, jer svaka promenljiva ima svoju brojnu vrednost. Kada otvorimo Matlab i počnemo sa radom, nemamo ni jednu promenljivu. Ako želimo da uvedemo novu promenljivu, potrebno je da joj damo ime i da joj po prvi put dodelimo vrednost. Na primer, ako želimo da uvedemo promenljivu pod imenom `x` i dodelimo joj vrednost 1, daćemo naredbu

```
>> x=1
```

a u narednim redovima biće ispisano sledeće.

```
x =
     1
```

Istovremeno, u delu prozora u kom su prikazane sve promenljive kojima je dodeljena vrednost (workspace) će se pojaviti promenljiva `x` sa svojom trenutnom vrednošću.

U opštem slučaju, za naredbu dodele vrednosti promenljivoj koristi se znak =, a sama naredba ima sledeću formu: pre znaka = nalazi se ime promenljive kojoj dodeljujemo vrednost, a posle znaka = nalazi se izraz koji ima brojnu vrednost. Ta brojna vrednost se dodeljuje promenljivoj. Istoj promenljivoj vrednost može biti dodeljena proizvoljan broj puta, i vrednost promenljive je uvek poslednja dodeljena vrednost.

Kada u komandnoj liniji unesemo konstantu ili izraz čija je vrednost konstanta, ta vrednost se smešta u specijalnu promenljivu `ans`.

Matlab razlikuje velika i mala slova. Kada imenujemo promenljive možemo koristiti i mala i velika slova, ali treba imati u vidu da se `x` i `X`, i `Pera`, `pera`, i `PERA`,

tretiraju kao različite promenljive. Takođe, neke reči su rezervisane, odnosno imaju specijalnu namenu, i njih nije moguće koristiti kao imena promenljivih.

Kada jednom dodelimo vrednost promenljivoj, njena vrednost se može koristiti navođenjem imena promenljive.

```
>> x=1
x =
     1
>> y=x+1
y =
     2
>> 1+2
ans =
     3
>> ans+ans
ans =
     6
```

U naredbi dodele, izraz koji definiše novu vrednost promenljive može da uključi i samu tu promenljivu (tačnije, njenu staru vrednost). U takvim slučajevima, najpre se izračunava vrednost izraza (koristeći staru vrednost promenljive), a zatim vrednost izraza postaje nova vrednost promenljive. To ćemo često koristiti za relativne promene vrednosti promenljive, bez obzira na njenu trenutnu vrednost. Na primer, ako želimo da uvećamo vrednost promenljive `x` za jedan, to radimo na sledeći način.

```
>> x=x+1
x =
     2
>> x=x+1
x =
     3
```

Ako želimo da promenljiva koja već ima vrednost izgubi tu svoju vrednost, koristimo naredbu `clear`.

```
>> clear A
```

Ako naredbu `clear` navedemo bez imena promenljive posle, biće uklonjene sve promenljive.

Naredbom `save` čuvamo u datoteci (čije ime navodimo) sve ili samo neke promenljive, zajedno sa njihovim trenutnim vrednostima. Kasnije vrednosti možemo učitati naredbom `load`. Ako umesto imena navedemo zvezdicu, čuvaju se sve promenljive koje u tom momentu imaju vrednost.

```
>> save ime_datoteke_2 x y

>> save ime_datoteke *

>> load ime_datoteke
```

Zadaci:

1. Dodeliti promenljivoj xyz vrednost 4, a promenljivoj abc vrednost 3. Zatim duplirati vrednost promenljive xyz, a nakon toga vrednost promenljive abc umanjiti za vrednost promenljive xyz.

Osnovne operacije i naredbe

Znak za operaciju množenja je *, i za razliku od nekih drugih programskih jezika ne možemo ga izostaviti (tj. ne možemo koristiti samo razmak između dva izraza koje želimo da pomnožimo, niti možemo „slepiti“ te izraze).

```
>> 1*2
ans =
     2
>> 1 2
??? 1 2
      |
Error: Missing MATLAB operator.
>> 2x
??? 2x
     |
Error: Missing MATLAB operator.
```

Kao što se iz prethodnog primera vidi, u slučaju sintaksne greške pri unosu naredbe dobijamo poruku o grešci u kojoj je najčešće ukratko naveden kratak opis greške – u prethodnom primeru, Matlab nam javlja da smo zaboravili da unesemo operator.

Za neke standardne konstante postoje naredbe koje vraćaju traženu vrednost. U slučaju naredbe `pi`, vraćena vrednost je približna vrednost konstante pi.

```
>> pi
ans =
    3.1416
```

Za većinu standardnih matematičkih operacija postoje naredbe. Dajemo primer izračunavanja kvadratnog korena, apsolutne vrednosti, logaritma sa prirodnom osnovom, logaritma sa osnovom dva, kosinusa, i ostatka pri celobrojnom deljenju. Argument na koji primenjujemo ove naredbe se navodi odmah nakon naredbe, u zagradama.

```
>> sqrt(5)
ans =
    2.2361
>> abs(-5)
ans =
     5
>> log(2)
ans =
    0.6931
>> log2(4)
ans =
     2
```

```
>> cos(pi)
ans =
    -1
>> mod(19,5)
ans =
     4
```

Elementarne operacije, sabiranje (koje smo već demonstrirali u prethodnim primerima), oduzimanje, deljenje, stepenovanje, obavljaju se na očekivan način, pomoću za to namenjenih znakova.

```
>> 5-4
ans =
    1
>> 2/7
ans =
    0.2857
>> 3^2
ans =
    9
```

Kao što se vidi iz prethodnog primera, sve numeričke vrednosti u Matlab-u se izračunavaju približno, na određen broj decimala. Samim tim, neke računske operacije neće biti izvršene tačno, već približno, i to treba imati u vidu. Način ispisa možemo promeniti naredbom `format`. U standardnom ispisu (`short`) broj je prikazan sa četiri decimale.

```
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
   3.14159265358979
>> format short e
>> pi
ans =
  3.1416e+000
>> pi^100
ans =
  5.1878e+049
>> format short
```

U radu sa komandnom linijom postoje načini da se ubrza i olakša rad. Pored standardnog korišćenja strelica u levo i u desno, tastera Backspace, Del, Home, End, možemo koristiti Esc za brisanje celog sadržaja linije. Strelicom na gore (a nakon toga i strelicom na dole) pozivamo prethodno otkucane naredbe. Ako otkucamo samo početak naredbe, onda se tasterom Tab dobija cela naredba (ako je jedinstveno određena), ili prolaskom kroz listu mogućih naredbi odabiramo onu koju želimo da koristimo.

Ako želimo da rezultat naredbe ne bude ispisan nakon što izvršimo naredbu, na kraju naredbe kucamo tačka zarez. Pored toga, možemo otkucati i više naredbi

odvojenih tačka zarezom, i izvršiti ih sve odjednom. Ovo ćemo pre svega koristiti kasnije, kada počnemo da pišemo programe.

```
>> a=2;
>> a=3; b=4; c=a+b
c =
     7
```

Za rad sa kompleksnim brojevima koristimo imaginarnu jedinicu koja se dobija naredbom `i`.

```
>> 1+2*i
ans =
   1.0000 + 2.0000i
>> 1/(2+3*i)
ans =
   0.1538 - 0.2308i
```

Za zaokruživanje brojeva postoji nekoliko naredbi, `round`, zaokruživanje na najbliži ceo broj, `fix`, odsecanje razlomljenog dela broja (dela iza decimalne tačke), `floor`, zaokruživanje na dole (najveći ceo broj koji nije veći od navedenog broja), `ceil`, zaokruživanje na gore (najmanji ceo broj koji nije manji od navedenog broja).

```
>> round(5.6)
ans =
     6
>> round(4.1)
ans =
     4
>> fix(-4.9)
ans =
    -4
>> floor(4.6)
ans =
     4
>> ceil(4.6)
ans =
     5
```

Opis naredbe, sa spiskom srodnih naredbi, dobija se naredbom `help`. U opisu je sama naredba ispisana velikim slovima, da bi se naglasilo njeno pojavljivanje u tekstu, ali se, naravno, prilikom korišćenja ona uvek piše malim slovima.

```
>> help ceil
 CEIL   Round towards plus infinity.
    CEIL(X) rounds the elements of X to the nearest
    Integers towards infinity.

    See also floor, round, fix.

    Overloaded functions or methods (ones with the same
    name in other directories)
       help sym/ceil.m
```

```
Reference page in Help browser
   doc ceil
```

Zadaci:

2. Izračunati u MATLAB-u:
   a) $\sin\frac{\pi}{2}+\cos\frac{\pi}{6}-tg\frac{\pi}{4}$
   b) $\log_2(\log_{10}10000)+\ln e^4$
   c) $5\pi+\sqrt{20}+3|-4|+\lfloor-2.2\rfloor+(2+6i)(2-6i)$

Matrice

U Matlab-u, čije ime je zapravo nastalo skraćivanjem imena Matrix Laboratory (laboratorija za matrice), mnogo pažnje je posvećeno radu sa matricama. Najprostije govoreći, matrica je tabela koja u svakoj svojoj ćeliji sadrži jedan broj. Svaka matrica ima određen broj vrsta (redova) i broj kolona.

Matrice se u Matlab-u unose između uglastih zagrada, vrstu po vrstu, vrste su odvojene tačka zarezom, a elementi u okviru vrste razmakom ili zarezom. Tako dve matrice, dimenzija 1x3 i 2x2, unosimo na sledeći način.

```
>> [1 2 3]
ans =
     1     2     3
>> [1 2;3 3]
ans =
     1     2
     3     3
```

Zapravo, Matlab sve brojne promenljive tretira kao matrice. U slučaju da promenljivoj dodelimo vrednost na način na koji smo to uradili u prvom primeru, `x=1`, ona će zapravo biti tretirana kao matrica sa jednom vrstom i jednom kolonom. Samim tim, sledeća naredba ima potpuno isti efekat.

```
>> x=[1]
x =
     1
```

Matrice koje imaju samo jednu vrstu zvaćemo vektor, a matrice koje imaju samo jednu kolonu zvaćemo vektor kolona. Matrice koje imaju dve dimenzije (i broj vrsta i broj kolona im je veći od jedan) ćemo najčešće označavati velikim slovima. Evo još nekoliko primera unošenja matrica.

```
>> v=[1 3 2 4]
v =
     1     3     2     4
>> A=[1 3 5 7;3 1 1 1]
A =
     1     3     5     7
     3     1     1     1
>> w=[1;2;3;4;5;6;7]
```

```
w =
     1
     2
     3
     4
     5
     6
     7
```

Apostrof se koristi za transponovanje matrice, tj. okretanje matrice preko glavne dijagonale. Glavna dijagonala je skup elemenata koji imaju isti broj vrste i broj kolone u kojima se nalaze.

```
>> [1 2 3;4 5 6]'
ans =
     1     4
     2     5
     3     6
>> v=[1 2 3]'
v =
     1
     2
     3
```

Dvotačka se koristi za dobijanje vektora koji sadrži aritmetički niz brojeva. Kada otkucamo `a:b`, i `a` je manje od `b`, dobijamo vektor koji sadrži niz brojeva `a`, `a+1`, `a+2`, …, `b`. Ako između dodamo još jedan parametar, `a:c:b`, dobijamo niz brojeva sa korakom `c`, dakle `a`, `a+c`, `a+2*c`,… , gde se niz prekida kada elementi niza postanu veći od `b`. Korak `c` može biti i negativan. Treba imati u vidu da vektor koji dobijamo može da bude i prazan, ako `a-b` i `c` imaju isti znak (oba su pozitivna ili oba negativna).

```
>> 2:5
ans =
     2     3     4     5
>> v=1:8
v =
     1     2     3     4     5     6     7     8
>> 0:10:45
ans =
     0    10    20    30    40
>> 10:1
ans =
   Empty matrix: 1-by-0
>> 1:-2:10
ans =
   Empty matrix: 1-by-0
```

Kada je matrica preširoka, pa zbog širine ne može da stane u prozor u kom se ispisuje, Matlab automatski prekida svaki red i to nam stavlja do znanja eksplicitno imenujući kolone u svakom od redova ispisa.

```
>> 1:17:200
ans =
```

```
  Columns 1 through 9
     1    18    35    52    69    86   103   120   137
  Columns 10 through 12
   154   171   188
>> 0:pi/4:2*pi
ans =
  Columns 1 through 5
         0    0.7854    1.5708    2.3562    3.1416
  Columns 6 through 9
    3.9270    4.7124    5.4978    6.2832
```

Zadaci:

3. Kreirati vektor koji sadrži parne cele brojeve iz intervala [22, 40].
4. Kreirati vektor čiji su elementi umnošci broja e, od e do 10e.
5. Kreirati vektor v koji sadrži brojeve od 9 do -4, koji se redom smanjuju za po 1. Transponovati taj vektor.
6. Kreirati matricu sa 2 vrste i 3 kolone koja sadrži proizvoljne elemente.
7. Kreirati matricu koja ima 3 vrste i 5 kolona takvih da se u prvoj vrsti nalaze neparni celi brojevi iz intervala [1, 9], u drugoj vrsti se nalaze vrednosti $\pi,2\pi,3\pi,4\pi,5\pi$, a u trećoj brojevi 6,5,4,3,2.

Osnovne matrične operacije i naredbe

Postoji više operatora koji se koriste za matrične operacije. Sabiranje i oduzimanje matrica se vrši slično kao i brojeva, operatorima + i –. Pri tome, matrice koje se sabiraju (ili oduzimaju) moraju biti istih dimenzija – mora im biti isti i broj vrsta i broj kolona. I rezultat je matrica istih dimenzija kao i matrice koje se sabiraju (ili oduzimaju), a svaki njen element je jednak zbiru elemenata na istoj toj poziciji u matricama koje se sabiraju – kažemo da se matrice sabiraju element-po-element.

```
>> A=[1 2 3;4 5 6]
A =
     1     2     3
     4     5     6
>> B=[1 1 2; 1 1 3]
B =
     1     1     2
     1     1     3
>> A+B
ans =
     2     3     5
     5     6     9
>> A-B
ans =
     0     1     1
     3     4     3
```

Množenje matrica (u matematičkom smislu) vrši se operatorom *. Kao što je poznato, za ovakvo množenje matrica broj kolona prve matrice mora biti jednak broju vrsta druge matrice. Ako pak želimo da množimo matrice element-po-element, na sličan način kao kod sabiranja, dodajemo tačku ispred operatora množenja. Tačka se

na isti način može dodati i pre bilo kog drugog aritmetičkog operatora, sa istim efektom. Naravno, u ovom slučaju matrice moraju biti istih dimenzija.

```
>> C=B'
C =
     1     1
     1     1
     2     3
>> A*C
ans =
     9    12
    21    27
>> A.*B
ans =
     1     2     6
     4     5    18
>> A./B
ans =
    1.0000    2.0000    1.5000
    4.0000    5.0000    2.0000
>> A.^B
ans =
     1     2     9
     4     5   216
```

Kao što smo već napomenuli, da bismo mogli da izvršimo navedene operacije neophodno je da dimenzije matrica budu kompatibilne. U suprotnom, dolazi do greške.

```
>> A
A =
     1     2     3
     4     5     6
>> B=A'
B =
     1     4
     2     5
     3     6
>> A+B
??? Error using ==> plus
Matrix dimensions must agree.
>> A*A
??? Error using ==> mtimes
Inner matrix dimensions must agree.
>> A.^B
??? Error using ==> power
Matrix dimensions must agree.
```

Za neke često korišćene matrice, postoje naredbe za generisanje tih matrica automatski. Za uneti vektor, naredbom `diag` dobijamo kvadratnu matricu čiji je broj vrsta i kolona jednak broju elemenata unetog vektora, i koja na glavnoj dijagonali ima elemente tog vektora, a van glavne dijagonale nule.

```
>> A=diag([1 3 2])
```

```
A =
     1     0     0
     0     3     0
     0     0     2
>> v=[1 2 3 4]
v =
     1     2     3     4
>> diag(v)
ans =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4
```

Naredba `ones` daje matricu koja sadrži sve jedinice. Ako joj prosledimo jedan parametar, dobijamo kvadratnu matricu tih dimenzija, a ako prosledimo dva, prvi parametar daje broj vrsta matrice, a drugi broj kolona. Naredba `zeros` funkcioniše analogno naredbi `ones`, s tim da daje nule umesto jedinica.

```
>> ones(6)
ans =
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
>> ones(3,5)
ans =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
>> zeros(4)
ans =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Naredba `eye` daje kvadratnu matricu sa jedinicama na glavnoj dijagonali, i nulama van glavne dijagonale. S obzirom da je matrica uvek kvadratna, naredbi treba proslediti samo jedan parametar koji određuje dimenziju matrice.

```
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Nije teško primetiti da smo umesto naredbe `eye` mogli koristiti i kombinaciju `diag` i `ones`.

```
>> diag(ones(1,4))
```

```
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Jedna od pogodnosti Matlab-a je to što većina operacija na brojevima može (u identičnoj ili sličnoj formi) da se primeni i na matrice. Ako navedemo matricu na onom mestu gde bi se inače očekivala jedna brojna vrednost, rezultat je matrica istih dimenzija kao navedena matrica, koja sadrži rezultate operacije primenjene na svaki od elemenata matrice.

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A+1
ans =
     2     3     4
     5     6     7
     8     9    10
>> log(A)
ans =
         0    0.6931    1.0986
    1.3863    1.6094    1.7918
    1.9459    2.0794    2.1972
>> sin([pi/2 pi 3*pi/2])
ans =
    1.0000    0.0000   -1.0000
>> A=ones(3)/2
A =
    0.5000    0.5000    0.5000
    0.5000    0.5000    0.5000
    0.5000    0.5000    0.5000
>> A.^2
ans =
     1     4     9
    16    25    36
    49    64    81
```

Dimenzije matrice dobijamo kao dvoelementni vektor naredbom `size`, gde prvi element predstavlja broj vrsta, a drugi broj kolona matrice. Ako među parametrima pored matrice navedemo i broj 1, ili 2, dobijamo samo broj vrsta, odnosno kolona, matrice. Ovu naredbu ćemo često koristiti kada budemo pisali programe, u situacijama u kojima budemo radili sa matricama za koje nam nisu poznate dimenzije.

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

```
>> size(A,1)
ans =
     2
>> size(A,2)
ans =
     3
>> ones(size(A))
ans =
     1     1     1
     1     1     1
>> A=A+ans
A =
     2     3     4
     3     4     5
>> size([1 2 3 4])
     1     4
```

Ako želimo da broj vrsta i broj kolona matrice `A` smestimo u promenljive `x` i `y`, to radimo na sledeći način.

```
>> [x y]=size(A)
x =
     2
y =
     3
```

Sada ćemo navesti par naredbi za izvršavanje nekih matematičkih funkcija na matricama. Rang matrice i determinanta (kvadratne) matrice dobijaju se naredbama `rank` i `det`.

```
>> A
A =
     1     1     1
     1     0    -1
     1     1     0
>> det(A)
ans =
     1
>> rank(A)
ans =
     3
```

Inverznu matricu dobijamo naredbom `inv`. Naravno, inverzna matrica postoji samo ako je matrica kvadratna i regularna (matrica sa svim jedinicama u sledećem primeru nije regularna).

```
>> inv(A)
ans =
   -0.5000   -0.5000    1.5000
         0    1.0000         0
    0.5000   -0.5000   -0.5000
>> inv(ones(3))
Warning: Matrix is singular to working precision.
ans =
```

```
   Inf   Inf   Inf
   Inf   Inf   Inf
   Inf   Inf   Inf
```

Naredbom `sum` sabiramo elemente vektora. U slučaju da je parametar naredbe sum matrica, sabiraju se elementi po kolonama, tj. nalazi se zbir elemenata u okviru svake kolone, a rezultat je vektor koji sadrži te zbirove. Ako želimo da uradimo slično za vrste, tj. da dobijemo vektor kolonu koji sadrži zbirove elemenata po vrstama, moramo dva puta koristiti operator za transponovanje matrice.

```
>> sum([1 2 3 4])
ans =
    10
>> A=[1 2 3;0 1 0; 1 1 1]
A =
     1     2     3
     0     1     0
     1     1     1
>> sum(A)
ans =
     2     4     4
>> sum(A')'
ans =
     6
     1
     3
```

Zadaci:

8. Kreirati kvadratnu matricu dimenzija 4x4 koja sadrži sve nule, sem na glavnoj dijagonali, gde su svi elementi 4.
9. Kreirati matricu dimenzija 4 x 4 koja sadrži sve 1 sem na glavnoj dijagonali, gde su svi elementi -1.
10. Kreirati 2 matrice: A – dimenzije 2x3 čiji su svi elementi 1, i B – dimenzije 2x3 čiji su elementi proizvoljni. Sabrati ove dve matrice, a zatim transponovati dobijen rezultat.
11. Kreirati matricu X dimenzije 3x2 i matricu Y dimenzije 2x3, obe sa proizvoljnim elementima, pa njihov matrični proizvod smestiti u matricu T. Zatim naći matricu istih dimenzija kao T koja na odgovarajućim mestima sadrži logaritam (sa osnovom 10) od elemenata u T. Kreirati matricu K koja ima isti broj vrsta kao matrica X, i isti broj kolona kao matrica Y i čiji su svi elementi 3. Na kraju kreirati matricu čiji su svi elementi 0, a dimenzije je kao matrica T.
12. Neka je A matrica dimenzija 5x5 čiji su svi elementi na glavnoj dijagonali 1 a van dijagonale su svi nula, B je matrica 5x5 čiji su svi elementi 6, i C matrica dimenzija 5x5 koja na glavnoj dijagonali ima vrednosti 5, 12, 17, 30, 60 dok svi ostali elementi imaju vrednost 0. Izračunati u MATLAB-u:
    a) R=A+B*C;
    b) Naći determinantu matrice R;
    c) Naći rang matrice R;
    d) Naći vektor čiji su elementi zbirovi elemenata kolona iz R;
    e) Naći zbir svih elemenata iz R.

Pristupanje delu matrice

Elementu matrice u određenoj vrsti i koloni pristupamo korišćenjem zagrada, navodeći redne brojeve (indekse) vrste i kolone u kojima se element nalazi. Elementu matrice moguće je i dodeliti vrednost, naredbom dodele, a vrednosti ostalih elemenata se pri tome ne menjaju. Elementima vektora pristupamo navodeći samo jedan parametar, redni broj (indeks) elementa u vektoru.

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A(3,2)
ans =
     8
>> A(3,2)=0
A =
     1     2     3
     4     5     6
     7     0     9
>> v=[2 3 4 5]
v =
     2     3     4     5
>> v(2)
ans =
     3
```

Da bismo pristupili većem broju elemenata iz iste matrice, umesto indeksa vrste/kolone navodimo vektor koji sadrži indekse vrsta/kolona kojima želimo da pristupimo. Vrednosti u okviru tog vektora mogu se i ponavljati.

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A([1 2],3)
ans =
     3
     6
>> A([1 3],[1 3])
ans =
     1     3
     7     9
>> A([1 1],[1 1 1])
ans =
     1     1     1
     1     1     1
>> A(1:2,1:3)
ans =
     1     2     3
     4     5     6
>> v([2 4])
```

```
ans =
     3     5
```

Ako umesto indeksa vrste (odnosno kolone) navedemo samo dvotačku, smatra se da želimo da pristupimo celim kolonama (odnosno vrstama). Jedna ili više vrsta (ili kolona) matrice se briše tako što tom bloku matrice dodelimo prazan vektor, `[]`.

```
>> A(2,:)
ans =
     4     5     6
>> A(:,2)
ans =
     2
     5
     8
>> A(2,:)=[]
A =
     1     2     3
     7     8     9
>> A(:,[1 3])=[]
A =
     2
     8
```

Sledećim naredbama zamenjujemo prvu i treću vrstu, odnosno prvu i treću kolonu, matrice `A`.

```
>> A=[1 2 3;4 5 6;7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A([1 3],:)=A([3 1],:)
A =
     7     8     9
     4     5     6
     1     2     3
>> A(:,[1 3])=A(:,[3 1])
A =
     9     8     7
     6     5     4
     3     2     1
```

Kada želimo da pristupimo elementu matrice ili delu matrice, možemo koristiti i naredbu `end`. Naredba `end` se navodi u okviru zagrada i vrednost koju vraća je ukupan broj vrsta, odnosno kolona matrice, zavisno od toga da li se pojavljuje pre ili posle zareza. Kako u sledećem primeru matrica `A` ima 3 vrste i 4 kolone, `end` pre zareza ima vrednost 3, a posle zareza 4. Istu naredbu možemo koristiti i u radu sa vektorima.

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

```
     9    10    11    12
>> A(end,2)
ans =
    10
>> A(1:end-1,2:end)
ans =
     2     3     4
     6     7     8
>> A(2,[1 end-1 end])
ans =
     5     7     8
>> v=[1 2 3 4]
v =
     1     2     3     4
>> v(end-1)
ans =
     3
```

Koristeći sličan pristup možemo izvršiti i neke komplikovanije modifikacije matrice. U sledeća dva primera pokazaćemo kako iz matrice izdvojiti samo parne kolone, i kako je okrenuti naopako u gore-dole smeru, tako da prva i poslednja vrsta zamene mesta, druga i pretposlednja, itd.

```
>> A(:,2:2:end)
ans =
     2     4
     6     8
    10    12
>> A(end:-1:1,:)
ans =
     9    10    11    12
     5     6     7     8
     1     2     3     4
```

Zadaci:

13. Iz matrice G, dimenzija 4x6, čiji su elementi proizvoljni brojevi izbaciti 1. i 5. kolonu.
14. Od date matrice A, dimenzije 4x5, kreirati novu koja sadrži elemente iz 2. i 4. vrste i neparnih kolona.
15. U matrici A, dimenzija 2x8 koja sadrži proizvoljne brojeve zameniti 3. i 6. kolonu.
16. Uneti proizvoljnu matricu X, a zatim kreirati matricu Y koja sadrži kolone matrice X u obrnutom redosledu – od poslednje do prve.

Blok matrice

Umesto elemenata matrice možemo unositi i matrice. Pri tome je neophodno voditi računa da broj vrsta matrica koje se stavljaju u isti red bude isti, kao i da broj kolona u svakom sledećem redu bude isti kao i u prvom redu. U suprotnom, doći će do greške.

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

```
A =
     1     2
     3     4
>> B=[A -A;-A A]
B =
     1     2    -1    -2
     3     4    -3    -4
    -1    -2     1     2
    -3    -4     3     4
>> C=[A [5;6] A; [9 8 7 6 5];A -A [3;3]]
C =
     1     2     5     1     2
     3     4     6     3     4
     9     8     7     6     5
     1     2    -1    -2     3
     3     4    -3    -4     3
>> D=[A [1 2 3]]
??? Error using ==> horzcat
All matrices on a row in the bracketed expression must
have the same number of rows.
```

Ovaj način definisanja matrice ćemo često koristiti za dodavanje elementa u vektor, sa leve ili desne strane, što ilustrujemo sledećim primerom.

```
>> v=[2 4 6]
v =
     2     4     6
>> v=[v 9]
v =
     2     4     6     9
>> v=[0 1 v]
v =
     0     1     2     4     6     9
```

Zadaci:

17. Kreirati matricu A sa 3 vrste i 2 kolone. Na kraj matrice A dodati novu kolonu koja sadrži samo dvojke. Zatim na kraj matrice A dodati novu vrstu sa proizvoljnim elementima.
18. Kreirati matricu S dimenzija 4x4 sa proizvoljnim brojevima. Kreirati blok matricu B=[S –S 04x4].
19. Kreirati proizvoljnu matricu X, a zatim ispod poslednje vrste matrice X dodati njenu prvu vrstu, i još jednu vrstu koja sadrži sve jedinice.

Slučajni brojevi

Naredbom `rand` dobijamo slučajno izabran broj između 0 i 1. Ako želimo da dobijemo matricu čiji su svi elementi slučajni brojevi između 0 i 1, koristimo dodatne parametre, slično kao kod naredbi `ones` i `zeros`.

```
>> rand
ans =
    0.9501
```

```
>> rand(3)
ans =
    0.9355    0.8936    0.8132
    0.9169    0.0579    0.0099
    0.4103    0.3529    0.1389
>> rand(2,3)
ans =
    0.2311    0.4860    0.7621
    0.6068    0.8913    0.4565
```

Naravno, ako želimo da dobijemo broj koji je slučajno izabran između neka druga dva broja, možemo koristiti odgovarajuće matematičke transformacije. Na primer, na sledeći način dobijamo slučajan broj između 1 i 11, i matricu slučajnih brojeva između 2 i 7.

```
>> 1+10*rand
ans =
    5.9872
>> 2+5*rand(2,3)
ans =
    6.7506    5.0342    6.4565
    3.1557    4.4299    5.8105
```

Ako kao želimo da dobijemo slučajan ceo broj iz nekog intervala, poslužićemo se nekom od naredbi za zaokruživanje. U sledećem primeru vidimo kako da dobijemo slučajan broj između 1 i 10, i slučajan paran broj između 1 i 100.

```
>> ceil(10*rand)
ans =
     6
>> 2*ceil(50*rand)
ans =
    76
```

Zadaci:

20. Kreirati matricu S, dimenzija 3x3, čiji su elementi slučajni brojevi između 0 i 3.
21. Kreirati matricu M, dimenzija 3x4 čiji su elementi slučajni brojevi iz intervala (3,6).
22. Kreirati matricu T dimenzija 3x5 koja sadrži slučajne cele brojeve u intervalu [2,11]. Obrisati iz T 2. i 5. kolonu, a zatim prikazati na ekranu elemente iz 1. i 3. vrste i 2. kolone.
23. Neka je A matrica dimenzija 4x4 čiji su elementi slučajni celi brojevi iz intervala [1,1000].

Skupovi

Skupovi se predstavljaju kao vektori koji u proizvoljnom redosledu sadrže elemente skupa. Ponavljanje elementa u okviru takvog vektora je moguće, i ignoriše se. Pomoću naredbi `union`, `intersect` i `setdiff` dobija se unija, presek i razlika skupova.

```
>> x=[1 2 3 5]
x =
     1     2     3     5
>> y=[1 4 8]
y =
     1     4     8
>> union(x,y)
ans =
     1     2     3     4     5     8
>> intersect(x,y)
ans =
     1
>> setdiff(x,y)
ans =
     2     3     5
>> union([1 1 2 1], [3 3 3 1])
ans =
     1     2     3
```

Zadaci:

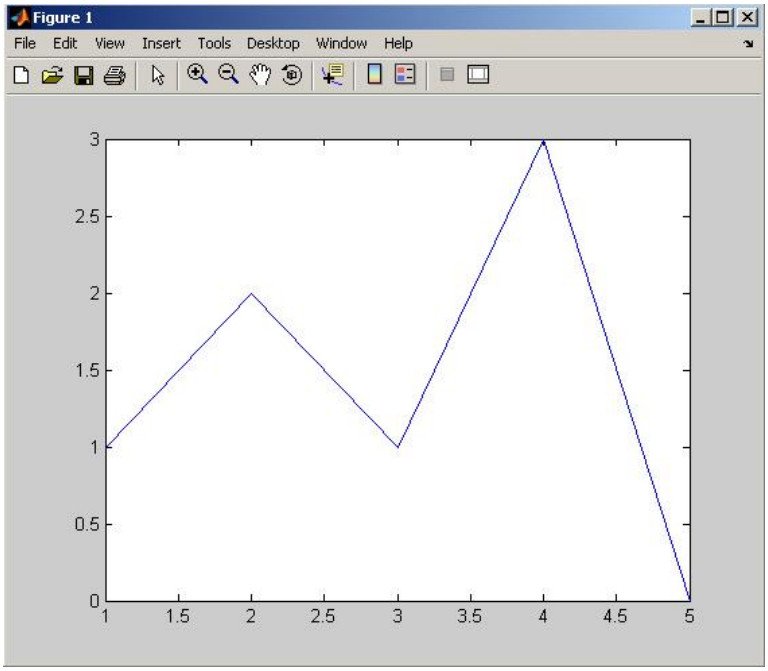
24. Kreirati skupove S1={1,6,3,4}, S2={2,3,5,8} i S3={2,5,7,9}. Naći $S1 \cup S2, S2 \cap S3, S1 \setminus S2, (S1 \cup S3) \setminus S2$ i simetričnu razliku skupova S2 i S3.

Grafički prikaz tačaka u ravni

Niz tačaka u ravni grafički se može predstaviti pomoću naredbe `plot`. Kao argumente dajemo dva vektora sa istim brojem elemenata koji sadrže prve, odnosno druge, koordinate tačaka koje želimo da predstavimo. Nakon izvršavanja naredbe

```
>> plot([1 2 3 4 5],[1 2 1 3 0])
```

grafički prikaz vidimo u posebnom prozoru, a tačke sa koordinatama (1,1), (2,2), (3,1), (4,3) i (5,0) su prikazane i povezane dužima.

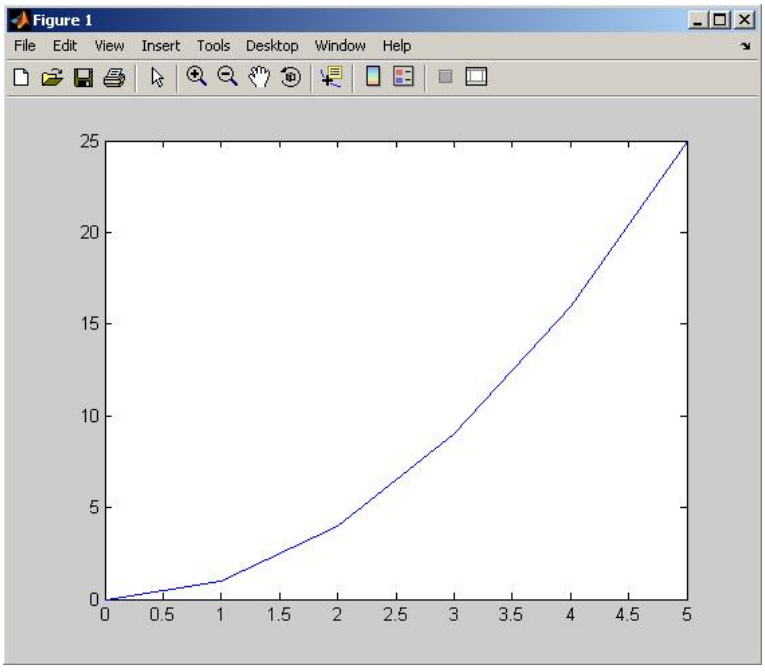


Naravno, u slučaju kada su *x*-koordinate tačaka na ravnomernom rastojanju, kao što je to slučaj u prethodnom primeru, prvi vektor koji prosleđujemo naredbi `plot` možemo kreirati i lakše:

```
>> plot(1:5,[1 2 1 3 0]).
```

U sledećem primeru prikazujemo tačke sa grafika kvadratne funkcije.

```
>> t=0:5
t =
     0     1     2     3     4     5
>> plot(t, t.^2)
```

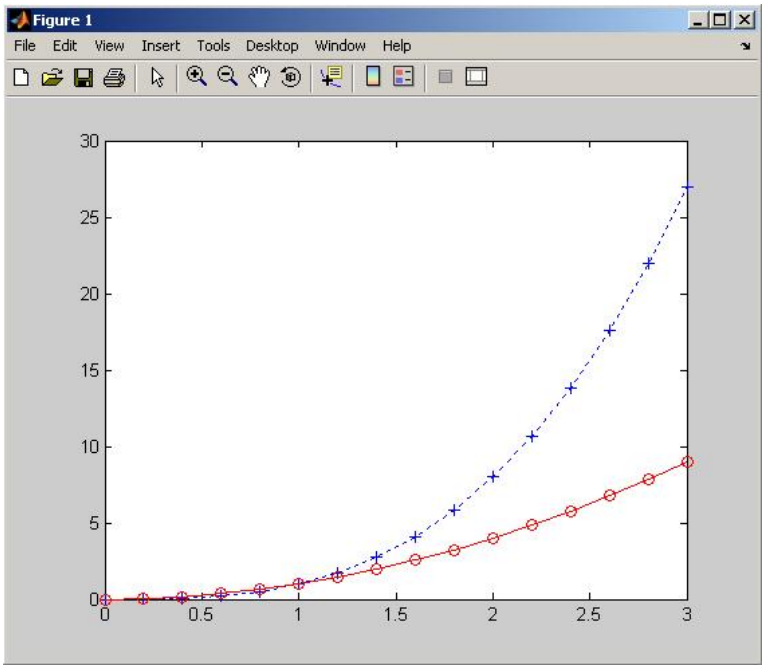


Ako među parametrima pored dva vektora dodamo i string kao treći parametar, menjamo način na koji se grafik iscrtava – pod jednostrukim navodnicima navodimo jedan ili više znakova za formatiranje (na primer, `+`, `o` ili `s` navodimo za naglašavanje tačaka na grafiku, `:` ili `-` za tip linije, `r`, `g`, `b`, `y` za boju grafika).

Ako želimo (ili ne želimo) da se svaki sledeći grafički prikaz pojavi preko prethodnih, kucamo `hold on` (ili `hold off`). Naredbe poput `xlabel` i `title` služe za dodavanje oznaka *x*-ose i naslova na grafik.

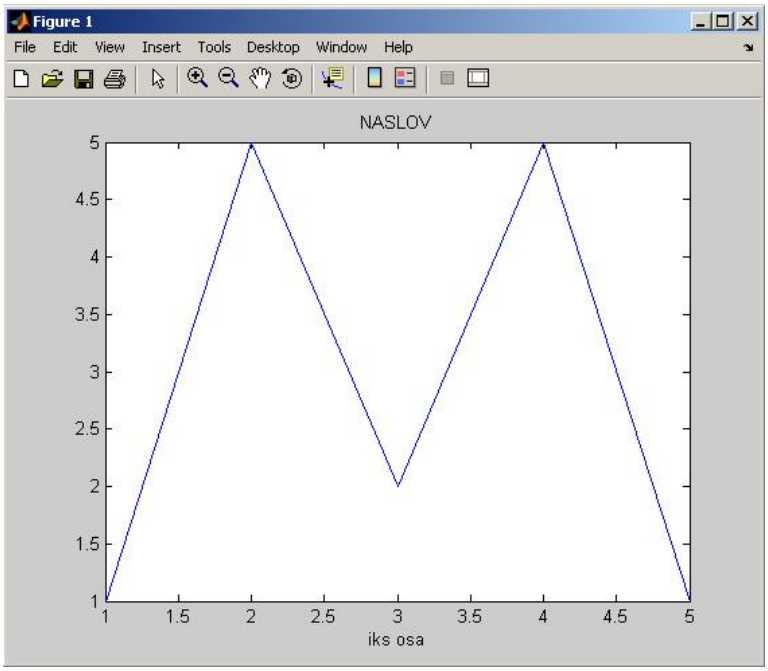
Sledi još par primera za korišćenje pomenutih naredbi. U prvom primeru istovremeno prikazujemo tačke sa grafika funkcija $f(x)=x^3$ i $g(x)=x^2$, formatiranih na različit način.

```
>> t=0:0.2:3;
>> plot(t, t.^3, '+:')
>> hold on
>> plot(t, t.^2, 'ro-')
```



U drugom primeru na grafički prikaz dodajemo i oznake. Vredi napomenuti da se grafički prikaz može menjati i ulepšavati i preko menija u prozoru u kom se nalazi.

```
>> hold off
>> plot(1:5,[1 5 3 5 1])
>> xlabel('iks osa')
>> title('NASLOV')
```



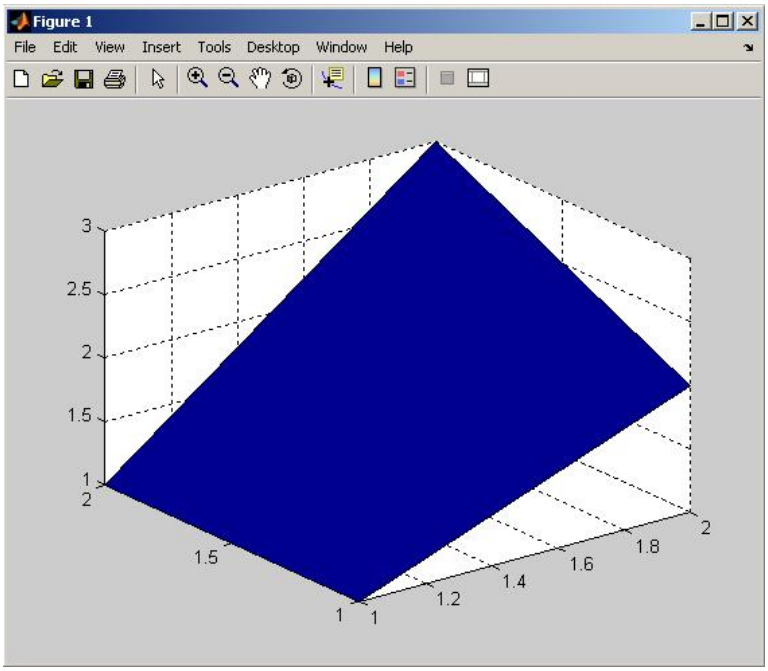
Zadaci:

25. Prikazati grafički skup tačaka u ravni {(1,1), (2,4), (3,2), (4,7), 5,6)}.
26. Prikazati grafik kvadratne funkcije x^2, za x odabrano između 0 i 1 sa korakom 0,1. Zatim prikazati isti grafik zelenom bojom.
27. Prikazati grafike funkcija t^2 i t^3 u istom prozoru tako da drugi grafik bude žute boje, iscrtan isprekidanom linijom, za t odabrano između 0 i 1 sa korakom 0,2.
28. Za x odabrano između 0 i 2 sa korakom 0,1, prikazati na istom crtežu grafike sledećih funkcija:
    a) $f(x)=x^2$, tako da je grafik zelene boje, sa zadatim tačkama označenim dijamantima i tačkastom linijom.
    b) $f(x)=x^4$, tako da je grafik crvene boje, sa zadatim tačkama označenim kružićima i isprekidanom linijom.

Grafički prikaz tačaka u prostoru

Za grafički prikaz skupa tačaka u tri dimenzije koristi se naredba `surf`. Slično kao i kod naredbe `plot`, naredbi `surf` prosleđujemo tri matrice istih dimenzija, i svaka od njih sadrži odgovarajuću koordinatu tačaka koje se prikazuju. U sledećem primeru, prikazujemo četiri tačke sa koordinatama (1,1,1), (2,1,2), (1,2,1) i (2,2,3).

```
>> surf([1 2;1 2],[1 1;2 2], [1 2; 1 3])
```



Ako je projekcija prikazanih tačaka na ravan prve dve koordinate kvadratna mreža, onda je umesto prve dve matrice dovoljno navesti vektor i vektor-kolonu, koji sadrže koordinate prve (vektor) i druge (vektor-kolona) koordinate tačaka. Tako smo prikaz iz prethodnog primera lakše mogli dobiti na sledeći način.

```
>> surf([1 2],[1;2], [1 2; 1 3])
```

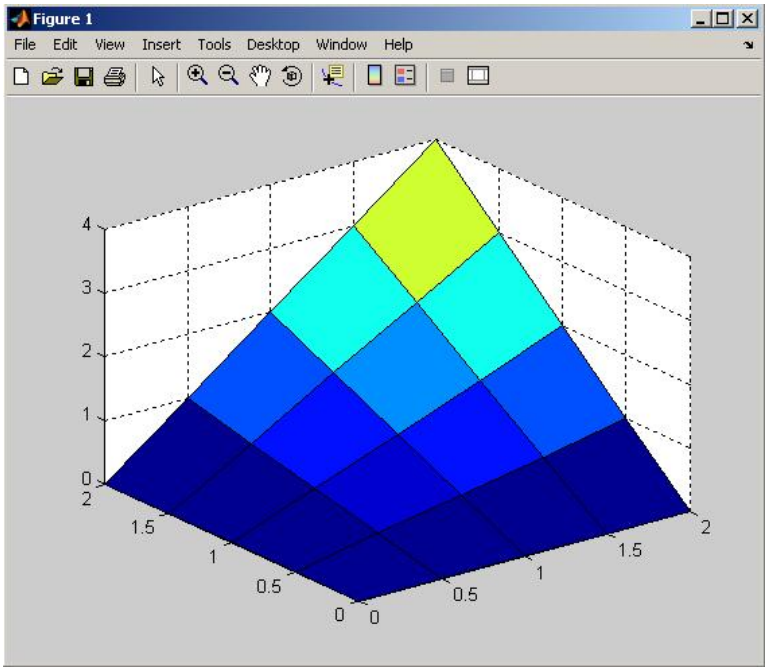
Ako za dobijanje treće matrice želimo da iskoristimo prve dve, možemo najpre iskoristiti naredbu `meshgrid` za generisanje matrica koje predstavljaju prve dve koordinate. Parametri koje prosleđujemo naredbi `meshgrid` su, slično prethodnom primeru, dva vektora koji sadrže prve, odnosno druge, koordinate kvadratne mreže. Naredba `meshgrid` vraća dve matrice, i ako želimo da ih smestimo u dve promenljive, `X` i `Y`, to radimo na sledeći način.

```
>> [X Y]=meshgrid(0:0.5:2,0:0.5:2)
X =
         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
         0    0.5000    1.0000    1.5000    2.0000
Y =
         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
    1.5000    1.5000    1.5000    1.5000    1.5000
    2.0000    2.0000    2.0000    2.0000    2.0000
```

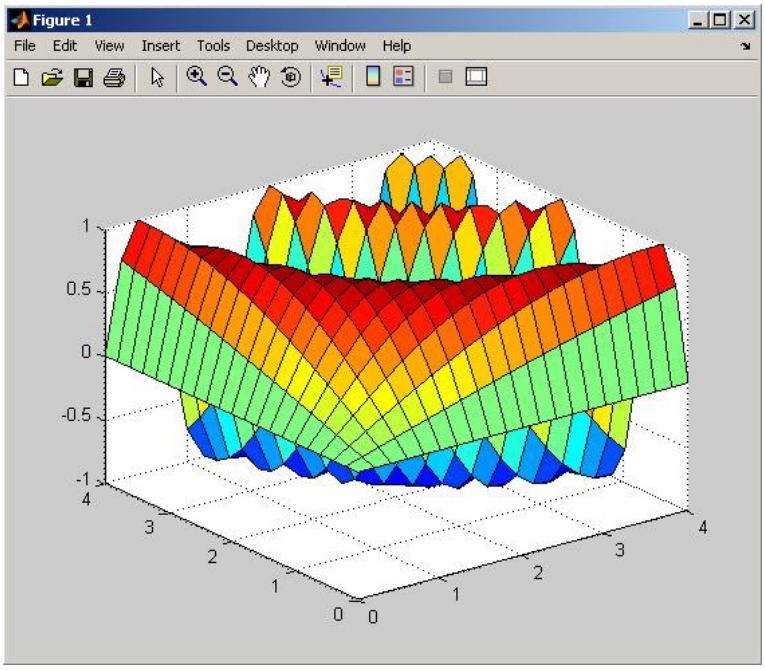
Nakon toga, matrice koje smo dobili na ovaj način možemo da iskoristimo kao prva dva parametra naredbe `surf`, dok treću matricu možemo da dobijemo pomoću te dve matrice. U sledećim primerima, dobijamo prikaz tačaka sa grafika funkcije

$f(x,y)=xy$, i funkcije $g(x,y)=\sin(xy)$. Naredbe u kojima smo koristili `meshgrid` smo završili tačka zarezom, da bismo sprečili glomazan ispis nakon izvršavanja naredbe.

```
>> [X Y]=meshgrid(0:0.5:2,0:0.5:2);
>> surf(X,Y, X.*Y)
```




```
>> [X,Y]=meshgrid(0:0.2:3,0:0.2:3);
>> surf(X,Y, sin(X.*Y))
```



Zadaci:

29. Prikazati grafik funkcije $f(x,y)=(xy)^3$, za x između -1 i 1 birano sa korakom 0,5, a y između -1 i 1 birano sa korakom 0,1.
30. Prikazati grafik funkcije:

a) *f(x,y)=sin(xy)* kao mrežu,
b) *g(x,y)=sin(x)cos(y)* kao površ,

ako su *x* i *y* za obe funkcije odabrani iz intervala [-1,1] sa korakom 0,2.

Ispis

Ispis u komandnom prozoru ćemo koristiti kada budemo pisali programe, pre svega za komunikaciju između programa i korisnika.

Naredba `disp` koristi se za ispis vrednosti izraza koji se navodi kao parametar naredbe. Nakon ispisa naredbom `disp`, eventualni naredni ispis nastavlja se u sledećem redu. Pored brojeva sa kojima smo do sada radili, izraz može biti i niz znakova koji nazivamo string. String u Matlab-u navodimo između jednostrukih navodnika i, neformalno govoreći, takav izraz se tretira kao konstanta tekstualnog tipa.

```
>> x=1
x =
     1
>> v=[1 2 3]
v =
     1     2     3
>> disp(x)
     1
>> disp(x); disp(x+1); disp(x+2);
     1
     2
     3
>> disp(v)
     1     2     3     5
>> disp(v(3))
     3
>> disp('Uuuujaaaa!')
Uuuujaaaa!
```

Naredbi `error` takođe prosleđujemo string, tj. niz znakova između jednostrukih navodnika, a kada se ona izvrši dobijamo poruku sličnu porukama prilikom sintaksnih grešaka, ali sa tekstom koji smo mi uneli.

```
>> error('Greska: Dati broj je negativan!')
??? Greska: Dati broj je negativan!
```

Naredba `fprintf` je naprednija verzija naredbe `disp`. Kao obavezan parametar ove naredbe, pod jednostrukim navodnicima navodimo niz znakova koji će biti ispisan. Pored običnih znakova možemo navoditi i specijalne oznake. Oznaka `\n` koristi se za prelaz u novi red. Za razliku od naredbe `disp`, eventualni naredni ispis nastavlja se u istom redu u kom je prethodni završen. Zato ćemo u situacijama u kojima ne želimo da više puta vršimo ispis u isti red (što je najčešće slučaj) koristiti oznaku `\n` na samom kraju niza karaktera.

```
>> fprintf('Tralala!\n')
```

```
Tralala!
>> fprintf('Tra\nla\nla!\n')
Tra
la
la!
>> fprintf('Ime: '); fprintf('Pera\n');
Ime: Pera
```

Postoji i nekoliko specijalnih oznaka za ubacivanje vrednosti u niz karaktera koji se ispisuje. Tako se oznaka `%d` koristi za celobrojni ispis broja, `%f` za decimalni ispis, a `%s` za ispis stringa. Same vrednosti koje ubacujemo se navode kao parametri naredbe `fprintf`, iza početnog niza znakova.

```
>> fprintf('Imam %d dinara.\n', 5)
Imam 5 dinara.
>> x=10.5
x =
   10.5000
>> fprintf('Rezultat je %f.\n', x)
Rezultat je 10.500000.
>> ime='Milos'
ime =
Milos
>> fprintf('Ja se zovem %s.\n', ime)
Ja se zovem Milos.
```

Ako neposredno nakon `%d` ili `%f` dodamo još jedan broj, ili dva broja razdvojena tačkom, fiksiramo širinu ukupnog ispisa i, ako je naveden i drugi broj, maksimalan broj cifara iza decimalne tačke.

```
>> fprintf('Imam %6d din.\nImas %6d din.\n', 10, 1000)
Imam     10 dinara.
Imas   1000 dinara.
>> fprintf('%8.2f \n%8.2f \n%8.2f \n', pi, 444, 2/7)
    3.14
  444.00
    0.29
```

Logičke vrednosti i operatori

Logičke vrednosti, tačno i netačno, u Matlab-u se predstavljaju brojevima. Tako se 0 interpretira kao netačno, a 1 (a i svi ostali brojevi) kao tačno. U Matlab-u postoje i operatori koji odgovaraju standardnim matematičkim relacijama i kao rezultat daju logičku konstantu, npr. == (jednako), ~= (različito), > (veće), >= (veće ili jednako), < (manje), <= (manje ili jednako). Prilikom korišćenja svih navedenih relacija, i sa leve i sa desne strane relacije mora da se nalazi brojna vrednost. Važno je imati u vidu razliku između operatora dodele, =, i operatora za ispitivanje jednakosti, ==.

```
>> 1>2
ans =
     0
```

```
>> 3>2
ans =
     1
>> x=1
x =
     1
>> x==1
ans =
     1
>> x==2
ans =
     0
```

Sa obe strane pomenutih relacija umesto brojnih vrednosti mogu da se nalaze i matrice istih dimenzija. U tom slučaju, kao rezultat dobijamo matricu istih dimenzija koja sadrži jedinice i nule, a vrednost svakog elementa zavisi od toga da li su elementi na odgovarajućim pozicijama u prve dve matrice u relaciji ili ne.

```
>> [1 2 3]==[1 4 3]
ans =
     1     0     1
>> [1 2; 3 4]>[2 2;2 2]
ans =
     0     0
     1     1
>> [1 2; 3 4]~=[2 2;2 2]
ans =
     1     0
     1     1
```

Logički operatori *„i“, „ili“* i *„ne“* dobijaju se pomoću `&&`, `||` i `~`.

```
>> 1>=0 && x>0
ans =
     1
>> 1~=1 || x==x+1 || 1+1==4
ans =
     0
>> ~((x>=1 || x<-5) && x~=3)
ans =
     0
```

Deljivost jednog broja drugim proveravamo koristeći naredbu `mod` na sledeći način.

```
>> mod(17,5)==0
ans =
     0
>> mod(15,5)==0
ans =
     1
```

Zadaci:

31. Proveriti tačnost logičkog izraza $(((6>3)\wedge(2+1=3))\vee(4-2\neq 7))$.

32. Dodeliti proizvoljne vrednosti promenljivima x, y i z, a zatim proveriti da li je tačno da među njima promenljiva x nema ni najmanju ni najveću vrednost.
33. Neka je M matrica 4x4 čiji su elementi slučajni celi brojevi iz intervala [0,14]. Proveriti koji su elementi u M jednaki 10, koji su veći od 5, a koji su različiti od 2.

Funkcije

Sve do sada, za unošenje i izvršavanje naredbi koristili smo komandnu liniju. Takav pristup ograničava, jer naredbe izvršavamo jednu po jednu, onako kako ih unosimo. U narednom delu upoznaćemo se sa konceptom programa koji otvara veće mogućnosti za rešavanje matematičkih problema.

Programi u Matlab-u nazivaju se još i *m-datoteke*, zbog obavezne `.m` ekstenzije za datoteke u kojima se programi čuvaju. Postoji dva osnovna tipa programa – skript i funkcija. Glavna razlika leži u tome što kod funkcija imamo mogućnost prosleđivanja parametara (u funkciju i iz funkcije), dok kod skriptova to nije slučaj. Takođe, skriptovi dejstvuju na Matlab-ov workspace, dok se za funkcije prilikom izvršavanja kreira njihov sopstveni workspace. Mi ćemo se u narednom tekstu baviti isključivo funkcijama.

Unos funkcije, parametri, izvršavanje

Kreiranje funkcije se započinje iz komandne linije naredbom `edit`, a kao parametar se navodi buduće ime funkcije, koje sami biramo.

```
>> edit prva
```

Kao rezultat ove naredbe otvara se editor pomoću koga unosimo funkciju. Svaka funkcija počinje zaglavljem, a prva reč zaglavlja je uvek rezervisana reč `function`. Nakon nje navodimo izlazni parametar ili parametre – ako ih ima više navodimo ih kao elemente vektora. Zatim navodimo znak jednakosti, pa ime funkcije, koje mora da odgovara imenu koje smo naveli uz naredbu `edit` i imenu datoteke u kojoj ćemo posle sačuvati funkciju. Zaglavlje završavamo listom parametara (u zagradama) koji će biti prosleđeni funkciji. I izlazne i ulazne parametre moguće je izostaviti, s tim da u zaglavlju bez izlaznih parametara nema ni znaka jednakosti, a bez ulaznih parametara nema ni zagrada. U nastavku navodimo neka moguća zaglavlja funkcije čije je ime `prva`.

```
function prva

function prva(ocena)

function x=prva(p1,p2)

function [A B]=prva(p1,p2)
```

Nakon zaglavlja funkcije navodimo telo funkcije, odnosno jednu ili više naredbi, svaku od njih završavajući tačka zarezom. Kada unesemo celu funkciju, potrebno je da je sačuvamo u datoteci `ime_funkcije.m` (gde je `ime_funkcije` ime navedeno u zaglavlju funkcije), i nakon toga možemo da je pozovemo iz

komandne linije, navođenjem njenog imena, dakle `ime_funkcije`, navodeći u zagradama i parametre (u slučaju da funkcija ima ulazne parametre). Prilikom izvršavanja funkcije, naredbe iz tela funkcije izvršavaju se jedna po jedna, u redosledu u kom su navedene.

Kao prvi primer dajemo jednu veoma prostu funkciju.

```
function prva

disp('Zdravo!');
```

Prilikom izvršavanja, ona štampa reč `Zdravo!` u komandnom prozoru. Prazan red (drugi red funkcije) se ignoriše, i generalno govoreći, kod unosa funkcije, svi razmaci i prazni redovi (koji ne narušavaju integritet naredbi) se ignorišu.

U sledećoj funkciji imamo dva ulazna parametra, a u telu funkcije pojavljuje se još i promenljiva `x`.

```
function druga(a,b)

x=a+b;
fprintf('Zbir brojeva %d i %d je %d.\n',a,b,x);
```

Ova funkcija za uneta dva prirodna broja ispisuje ta dva broja i njihov zbir. Kao što smo već pomenuli, funkcija ima svoj sopstveni workspace, i ulazni parametri a i b i promenljiva x su potpuno nezavisni od eventualno postojećih promenljivih sa istim imenom u Matlab-ovom workspace-u, ili nekoj drugoj funkciji. Evo i par primera poziva ove funkcije iz komandne linije. Prilikom poziva funkcije, promenljive `a` i `b` automatski dobijaju vrednosti koje su prosleđene (u zagradama) prilikom poziva.

```
>> druga(2,3)
Zbir brojeva 2 i 3 je 5.
>> druga(7,12)
Zbir brojeva 7 i 12 je 19.
```

Sledeća funkcija pored ulaznog ima i izlazni parametar. Samim tim, poziv ove funkcije vraća vrednost koju parametar `rez` ima onog momenta kada se završi izvršavanje funkcije. U ovom slučaju, to je vrednost prirodnog logaritma od `x` uvećana za 1.

```
function rez=treca(x)

rez=log(x);
rez=rez+1;
```

Treba imati u vidu da funkcija vraća poslednju vrednost koja je dodeljena parametru `rez` pre završetka izvršavanje funkcije. Dakle, ukoliko je parametar `rez` imao više različitih vrednosti tokom izvršavanja funkcije, vraćena će biti samo poslednja vrednost tog parametra.

S obzirom da prethodna funkcija ima izlazni parametar, poziv funkcije ima vrednost. Ovakve funkcije treba razlikovati od funkcija koje nemaju izlazne parametre, i eventualno ispisuju nešto u komandnom prozoru.

```
>> treca(4)
ans =
    2.3863
>> x=treca(7)+1
x =
    3.9459
```

Komentari

U okviru funkcija možemo navoditi i komentare. Oni nemaju nikakav uticaj na izvršavanje, i koristimo ih da povećamo razumljivost i preglednost koda funkcije. Komentari se navode posle znaka za procenat. Dakle, sav tekst koji je naveden između znaka za procenat i kraja reda biće ignorisan prilikom izvršavanja funkcije.

Korišćenje komentara ilustrujemo sledećim primerom.

```
function rez=treca(x)
% funkcija za uneti broj x vraća prirodni logaritam tog
% broja, uvecan za 1

rez=log(x);    % u rez se smesta logaritam broja x
rez=rez+1;     % vrednost rez se uvecava za 1
```

If-grananje

If-grananje ima sledeću strukturu, gde umesto `...` navodimo nizove naredbi.

```
if logički_izraz          } if blok
    ...
elseif logički_izraz              } elseif blok
    ...
else
    ...    } else blok
end
```

Pritom se `if` blok pojavljuje tačno jednom, `else` blok može da se pojavi nijednom ili jednom, a `elseif` blok može da se ne pojavi, ali i da se ponovi više puta. Odmah nakon `if` i svakog pojavljivanja `elseif` navodi se logički izraz.

U slučaju da je logički izraz posle `if` tačan, izvršava se niz naredbi naveden u tom bloku i završava se izvršavanje celog grananja (uključujući i sve preostale blokove). Isto važi i za svaki naredni `elseif` blok. Ako nijedan od navedenih logičkih izraza nije tačan, onda se izvršava niz naredbi u `else` bloku (ako `else` blok postoji). Dakle, u svakom if-grananju izvršava se niz naredbi iz najviše jednog bloka (a ako znamo da blok `else` postoji, onda iz tačno jednog bloka).

Na primer, kada se izvrši sledeće

```
x=7;
if x<0
    x=-x;
end
```

vrednost promenljive `x` će biti 7, jer logički izraz `x<0` nije tačan u momentu izvršavanja. Ako se pak izvrši

```
x=-9;
if x<0
    x=-x;
end
```

vrednost promenljive `x` će biti 9, jer je logički izraz `x<0` tačan, pa će naredba u bloku `if` biti izvršena.

Ako pretpostavimo da se naredno if-grananje nalazi u okviru funkcije u kojoj promenljiva `x` već ima vrednost, ako je ta vrednost veća od 5 biće uvećana za 3, ako nije veća od 5, ali je veća ili jednaka od 2 biće uvećana za 2, a u ostalim slučajevima biće uvećana za 1.

```
if x>5
    x=x+3;
elseif x>=2
    x=x+2;
else
    x=x+1;
end
```

Sledeća funkcija za uneta dva prirodna broja vraća tačno ako je prvi parametar deljiv drugim, a inače vraća netačno.

```
function l=deljiv(a,b)

if mod(a,b)==0
    l=1;
else
    l=0;
end
```

S obzirom da je vrednost koju funkcija `deljiv` vraća logička, nakon što sačuvamo ovu funkciju, možemo je koristiti i u if-grananjima u drugim funkcijama, što ilustrujemo sledećim primerom.

```
function deli2(a,n)

if deljiv(a,2^n)
    fprintf('%d je %d puta deljiv sa dva.\n',a,n);
else
    fprintf('%d nije %d puta deljiv sa dva.\n',a,n);
end
```

Funkcija `deli2` štampa u komandnom prozoru poruku, u zavisnosti od toga da li parametar `a` možemo `n` puta podeliti sa 2 (odnosno, da li je deljiv sa 2^n).

```
>> deli2(24,3)
24 je 3 puta deljiv sa dva.
>> deli2(24,4)
```

```
24 nije 4 puta deljiv sa dva.
```

Zadaci:

34. Napisati funkciju koja za unet prirodan broj b vraća:
    a) b^2, ako je b deljiv sa 3,
    b) 3b, ako b nije deljiv sa 3, ali jeste sa 5, i
    c) 100, u svim ostalim slučajevima.
35. Napisati funkciju koja proverava da li je uneta godina prestupna i ispisuje odgovarajuću poruku – 'prestupna', odnosno 'nije prestupna'. Godina je prestupna ako je deljiva sa 400, ili ako je deljiva sa 4 a nije deljiva sa 100.

For-petlja

For-petlja ima sledeću strukturu, gde umesto `...` navodimo niz naredbi.

```
for promenljiva=vektor
    ...
end
```

Prilikom izvršavanja, promenljivoj `promenljiva` se najpre dodeljuje prvi element vektora `vektor`, a zatim se izvršava navedeni niz naredbi. Nakon toga, `promenljiva` dobija drugu vrednost iz `vektor`, i ponovo se izvršava navedeni niz naredbi. Ovaj proces se ponavlja sve do poslednjeg elementa iz `vektor`.

Prema tome, sledeća for-petlja

```
for i=[1 2 3]
    disp(i);
end
```

i ovaj niz naredbi

```
i=1;
disp(i);
i=2;
disp(i);
i=3;
disp(i);
```

su ekvivalentni – u oba slučaja će se u komandnom prozoru ispisati vrednosti 1, 2 i 3.

Niz naredbi naveden unutar for-petlje ne mora da zavisi od promenljive koja menja vrednost. Funkcija u sledećem primeru za dati prirodan broj *n* štampa *n* puta „Zdravo!“ u komandnom prozoru.

```
function stampa(n)

for i=1:n
    disp('Zdravo!');
end
```

Sledeća funkcija za dati prirodan broj *n* štampa sve prirodne brojeve manje ili jednake od *n* u opadajućem redosledu, zajedno sa njihovim kvadratima.

```
function stampa(n)

for i=n:-1:1
    fprintf('%d na kvadrat je %d.\n', i, i^2);
end
```

Jedan poziv funkcije `stampa` izgleda ovako.

```
>> stampa(6)
6 na kvadrat je 36.
5 na kvadrat je 25.
4 na kvadrat je 16.
3 na kvadrat je 9.
2 na kvadrat je 4.
1 na kvadrat je 1.
```

Naredna funkcija za dati vektor štampa dvostruke vrednosti svih elemenata tog vektora.

```
function dvostruke(v)

for a=v
    disp(2*a);
end
```

Evo i poziva ove funkcije iz komandne linije.

```
>> dvostruke([-2 3 -5 7])
    -4
     6
   -10
    14
```

Prethodnu funkciju smo mogli napisati i na sledeći način, prolazeći promenljivom `i` kroz sve *indekse* vektora `v`.

```
function dvostruke2(v)

n=size(v,2);
for i=1:n
    disp(2*v(i));
end
```

Zadaci:

36. Napisati funkciju koja vraća vektor koji sadrži elemente na parnim pozicijama datog vektora.
37. Napisati funkciju koja redom ispisuje treće stepene elemenata datog vektora.

While-petlja

While-petlja ima sledeću strukturu, gde umesto `...` navodimo niz naredbi.

```
while logički_izraz
    ...
end
```

Prilikom izvršavanja, u slučaju da je logički izraz tačan, izvršava se navedeni niz naredbi. Nakon toga, ponovo se proverava da li je logički izraz tačan, i ako jeste, niz naredbi se ponovo izvršava. Ovaj ciklus se ponavlja sve dotle dok prilikom provere logički izraz ne bude netačan. Primetimo da ukoliko je logički izraz netačan već kada while-petlja prvi put počne da se izvršava, niz naredbi unutar while-petlje se neće nijednom izvršiti.

Sledeća funkcija štampa sve stepene broja 3 koji su manji od unetog prirodnog broja *n*.

```
function step3(n)

s=1;
while s<n
    disp(s);
    s=s*3;
end
```

U svakom prolazu kroz petlju, vrednost promenljive `s` se štampa, a zatim se uvećava 3 puta. Na taj način `s` praktično uzima redom vrednosti stepena broja 3. Izvršavanje ove funkcije može da izgleda ovako.

```
>> step3(500)
     1
     3
     9
    27
    81
   243
```

Naredna funkcija za unet vektor sa bar jednim negativnim elementom štampa sve elemente tog vektora do prvog negativnog elementa.

```
function pre_neg(v)

i=1;
while v(i)>=0
    disp(v(i));
    i=i+1;
end
```

Efekat funkcije `pre_neg` vidimo na sledećem primeru.

```
>> pre_neg([2 3 -5 7])
     2
```

```
3
```

Videli smo da nam while-petlja, kao i for-petlja, omogućava da neki niz naredbi izvršimo više puta. Glavna razlika između te dve petlje leži u tome što se naredbe unutar for-petlje izvršavaju tačno određen broj puta – onoliko puta koliko ima elemenata u vektoru koji smo naveli, dok u while-petlji ne moramo unapred da navedemo koliko puta će se naredbe izvršavati, već samo kriterijum za završetak. Ovo treba imati u vidu kada se dvoumimo koju od ove dve petlje da iskoristimo.

Konkretno, problem iz prethodnog primera je tipičan problem za čije rešavanje je pogodnije koristiti while-petlju. Izvršavajući niz naredbi u while-petlji promenljiva `s` uzima redom vrednosti stepena broja 3. Pritom, ne znamo tačno koliko takvih vrednosti želimo da odštampamo, znamo samo da želimo da ih štampamo *sve dok* su manji od `n`.

Zadaci:

38. Napisati funkciju koja za date prirodne brojeve *a*, *b* i *c* vraća najmanji prirodan broj *n* takav da važi $an+b^n>c$.
39. Napisati funkciju koja za dat prirodan broj *a* ispisuje redom sve prirodne brojeve *n* za koje je 3^n manje od *an*.
40. Napisati funkciju koja za unet prirodan broj *n* pronalazi sve prirodne brojeve čiji kvadrati nisu veći od *n*. Vratiti tražene brojeve i njihove kvadrate u matrici u kojoj se brojevi nalaze u prvoj vrsti, a kvadrati tih brojeva u drugoj vrsti. Npr. za *n*=30, vraća se matrica
$$M=\begin{bmatrix}1 & 2 & 3 & 4 & 5\\ 1 & 4 & 9 & 16 & 25\end{bmatrix}.$$

Kombinovanje for-petlje i if-grananja

Do sada smo se upoznali sa tri strukture, if-grananjem, for-petljom i while-petljom. Njihovim kombinovanjem moguće je uhvatiti se u koštac sa nekim veoma komplikovanim problemima. Jednu kombinaciju ćemo koristiti posebno često – if-grananje unutar for-petlje.

Kao što smo već videli, for-petlja nam omogućava da promenljivoj redom dodelimo vrednosti elemenata datog vektora, tj. da promenljivom „prođemo“ kroz vektor. Na taj način, koristeći operatore za rad sa vektorima koje smo pominjali ranije, možemo npr. da promenljivoj redom dodelimo uzastopne brojeve između dva broja (što smo uradili u funkciji `stampa`), ili recimo uzastopne parne brojeve iz nekog intervala. Ali, ako je niz vrednosti koje želimo da dodelimo previše kompleksan, kreirati vektor koji sadrži te vrednosti nije lako. U takvim slučajevima često koristimo if-grananje unutar for-petlje.

Na primer, sledeća funkcija štampa sve prirodne brojeve manje ili jednake od unetog prirodnog broja *n*, za koje važi da su deljivi sa 5 i daju ostatak 1 pri deljenju sa 3.

```
function brojac=for_if(n)

for i=1:n
```

```
    if mod(i,5)==0 && mod(i,3)==1
        disp(i);
    end
end
```

U ovoj funkciji, for-petlju i if-grananje želimo da posmatramo zajedno, te ćemo ih izdvojiti iz funkcije i još jednom navesti.

```
for i=1:n
    if mod(i,5)==0 && mod(i,3)==1
        ...
    end
end
```

Tokom izvršavanja ove strukture, za svaku vrednost promenljive `i` između 1 i *n* postavlja se pitanje – da li je ta vrednost deljiva sa 5 i daje ostatak 1 pri deljenju sa 3? Samo ako je odgovor „da“ izvršavaju se naredbe unutar if-grananja, a u suprotnom se odmah prelazi na sledeću vrednost. Drugim rečima, naredbe koje se nalaze unutar if-grananja biće izvršene (redom) za sve prirodne brojeve između 1 i *n* za koje je ispunjen pomenuti uslov. Na ovaj način smo promenljivom `i` „prošli“ kroz sve takve brojeve, što bi samo korišćenjem for-petlje bilo nemoguće uraditi ovako efikasno (bez prethodnog kreiranja vektora koji sadrži pomenute brojeve).

Zadaci:

41. Ispisati sve elemente datog vektora *v* koji pri deljenju sa 2 i 3 daju isti ostatak.
42. Napisati funkciju koja vraća vektor koji se od unetog vektora *v* dobija tako što se parni elementi podele sa 2, a ostali elementi ostanu isti.

Brojanje, sabiranje i množenje

U programima ćemo često želeti da prebrojimo neke objekte, da saberemo neke brojeve, ili pak da ih pomnožimo. Ta tri problema su srodna, i rešavamo ih na sličan način.

Za brojanje ćemo koristiti promenljivu kojoj na početku dodeljujemo vrednost 0. Nakon toga, za svaki od objekata koje brojimo uvećavamo tu promenljivu za jedan. Na primer, sledeća funkcija za unet prirodan broj *n* vraća *broj* prirodnih brojeva manjih ili jednakih od *n* koji su deljivi sa 17.

```
function brojac=brojanje(n)

brojac=0;
for i=1:n
    if mod(i,17)==0
        brojac=brojac+1;
    end
end
```

U slučaju da želimo da saberemo više brojeva, koristimo promenljivu kojoj na početku dodeljujemo vrednost 0, a nakon toga svaki od brojeva koje sabiramo

dodamo na postojeću vrednost te promenljive. U sledećem primeru, funkcija za unet prirodni broj *n* vraća zbir kvadrata prvih *n* prirodnih brojeva.

```
function zbir=sabiranje(n)

zbir=0;
for i=1:n
    zbir=zbir+i^2;
end
```

Ako pak želimo da izračunamo proizvod više brojeva, koristimo promenljivu kojoj na početku dodelimo vrednost 1, a potom vrednost te promenljive ažuriramo množeći je svakim od pomenutih brojeva. Naredna funkcija za unet prirodni broj *n* ispisuje proizvod svih neparnih brojeva manjih ili jednakih od *n.*

```
function mnozenje(n)

proizvod=1;
for i=1:2:n
    proizvod=proizvod*i;
end
disp(proizvod);
```

Zadaci:

43. Napisati funkciju koja vraća broj elemenata datog vektora koji nisu deljivi sa 5
44. Napisati funkciju koja za unet vektor *v* vraća proizvod onih elemenata iz vektora *v* koji su veći od pozicije na kojoj se nalaze.
45. Napisati funkciju koja vraća aritmetičku sredinu elemenata na neparnim pozicijama datog vektora *v*.

Greške

Ko radi taj i greši – pre ili kasnije funkcija koju unesemo neće se ponašati onako kako želimo. Naravno, Matlab radi ono što mu kažemo, tako da krivica za učinjenu grešku uvek leži na nama – treba pronaći grešku i otkloniti je. Postoji dve suštinski različite vrste grešaka, sintaksne i semantičke greške.

Sa sintaksnim (odnosno gramatičkim) greškama smo se već sreli. U pitanju su greške zbog kojih Matlab nije u stanju da izvrši neki deo funkcije koju smo uneli, jer nije u skladu sa „pravilima pisanja“. Prostim jezikom govoreći, Matlab nas u tom slučaju ne razume.

Na primer, ako želimo da kreiramo funkciju koja ispisuje sve prirodne brojeve manje od *n,* i unesemo sledeće

```
function ispis(n)

for i=1:m
    disp(i);
end
```

prilikom izvršavanja će doći do greške – u komandnom prozoru će crvenim slovima biti ispisano sledeće.

```
>> ispis(5)
??? Undefined function or variable 'm'.

Error in ==> ispis at 3
for i=1:m
```

Matlab pre i posle operatora dvotačke očekuje brojnu vrednost, a s obzirom da promenljiva `m` nema vrednost, ne može se kreirati vektor pomoću `1:m`. Zato nam je javljeno postojanje sintaksne greške u trećem redu funkcije, sa kratkim objašnjenjem o kakvoj grešci se radi.

Naravno, ne preostaje nam ništa drugo nego da pređemo ponovo u editor, pogledamo našu funkciju i pokušamo da popravimo grešku – umesto `m` napisaćemo `n`.

```
function ispis(n)

for i=1:n
    disp(i);
end
```

Ako sada sačuvamo funkciju i pozovemo je npr. sa parametrom 4, dobijamo sledeće.

```
>> ispis(4)
     1
     2
     3
     4
```

Čini se da je sada sve u redu, jer Matlab više ne javlja grešku. Ali, pregledom ispisa utvrđujemo da naša funkcija, iako radi, i dalje ne radi ono što treba – treba da štampa prirodne brojeve **manje** od *n,* a ona štampa i broj *n.* Greške ovog tipa nazivaju se semantičke greške, a u slengu se zovu i bagovi (na engleskom, *bug*). Ponovnim pregledom funkcije u editoru lako uočavamo grešku, i popravljamo je – ispravna funkcija izgleda ovako.

```
function ispis(n)

for i=1:n-1
    disp(i);
end
```

Za razliku od sintaksnih grešaka, za koje prilikom izvršavanja odmah znamo da su prisutne jer nam Matlab to javlja, jedini način da utvrdimo semantičku grešku jeste da testiramo funkciju na primerima i posmatramo njen rad. U slučaju da uočimo nepravilnosti, vraćamo se u editor gde ćemo pokušati da otklonimo grešku. Nakon toga ponovo testiramo funkciju, sve dok nismo zadovoljni njenim radom. Nažalost, to što funkcija radi dobro za neke test primere nije garancija da će uvek raditi dobro, i taj fenomen muči sve programere – od početnika do profesionalaca.

Manipulacije vektorima i matricama

U prethodnom tekstu smo se već upoznali sa načinima za kreiranje vektora i matrica u Matlab-u, a pomenuli smo i neke osnovne operacije i naredbe koje dejstvuju na njih. U ovom delu nastavljamo u istom smeru, baveći se naprednijim tehnikama za rad sa vektorima i matricama.

For-petlje, matrice i vektori

Ako želimo da pristupimo većem broju elemenata vektora ili matrice, za to najčešće koristimo jednu ili dve for-petlje. Kao što smo ranije pomenuli, dimenzije vektora ili matrice očitavamo korišćenjem naredbe `size`.

Za „prolaz“ kroz elemente vektora potrebna nam je samo jedna for-petlja, u kojoj promenljivoj redom dodeljujemo vrednosti indeksa svih elemenata u vektoru, od 1 do ukupnog broja elemenata. Naredna funkcija za dati vektor `v` vraća vektor koji sadrži samo pozitivne elemente vektora `v`.

```
function w=pozitivni(v)

n=size(v,2);
w=[];
for i=1:n
    if v(i)>0
        w=[w v(i)];
    end
end
```

Ako for-petlju i if-grananje u ovoj funkciji posmatramo zajedno, ta struktura nam praktično omogućava da promenljivom `i` prođemo redom kroz indekse svih elemenata vektora `v` koji su veći od nule.

U sledećem primeru, funkcija za dati vektor `v` vraća vektor koji sadrži elemente vektora `v` koji su na parnim pozicijama.

```
function w=parne(v)

n=size(v,2);
for i=1:floor(n/2)
    w(i)=v(2*i);
end
```

Za prolaz kroz sve elemente matrice potrebne su nam dve for-petlje, i to tako da se jedna nalazi unutar druge. Naredna funkcija ispisuje sve elemente date matrice, jedan za drugim.

```
function ispis_m(A)

[n m]=size(A);
for i=1:n
    for j=1:m
        disp(A(i,j));
    end
```

```
end
```

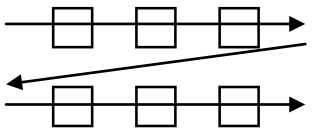
S obzirom da se for-petlja sa promenljivom `j` nalazi unutar for-petlje sa promenljivom `i`, za svaku vrednost promenljive `i` promenljiva `j` će uzeti redom sve vrednosti između 1 i `m`. Prema tome, ako prethodnu funkciju pozovemo sa matricom koja ima dve vrste i tri kolone, `n` će biti 2, `m` će biti 3, a blok

```
for i=1:n
    for j=1:m
        disp(A(i,j));
    end
end
```

se izvršava na potpuno isti način kao i naredni blok.

```
i=1;
j=1;
disp(A(i,j));
j=2;
disp(A(i,j));
j=3;
disp(A(i,j));
i=2;
j=1;
disp(A(i,j));
j=2;
disp(A(i,j));
j=3;
disp(A(i,j));
```

Drugim rečima, ako elemente matrice označimo kvadratima, kroz matricu „prolazimo“ na sledeći način, red po red.



Naredna funkcija za datu matricu *A* vraća matricu koja se od *A* dobija tako što se za jedan uvećaju elementi čiji je zbir indeksa vrste i kolone paran, dok ostali elementi ostaju isti.

```
function B=promeni(A)

n=size(A,1);
m=size(A,2);
for i=1:n
    for j=1:m
        if mod(i+j,2)==0
            B(i,j)=A(i,j)+1;
        else
            B(i,j)=A(i,j);
        end
```

```
    end
end
```

Alternativno, mogli smo na početku funkcije da matrici `B` dodelimo vrednost matrice `A` sa `B=A`, što bi nam omogućilo da izostavimo else blok if-grananja. Efekat prethodne funkcije vidimo na primeru.

```
>> promeni([1 2 3;4 5 6;7 8 9])
ans =
     2     2     4
     4     6     6
     8     8    10
```

Dve for-petlje koristimo i kada želimo da kreiramo matricu element po element, po datom pravilu. Sledeća funkcija za date prirodne brojeve *n* i *m* vraća matricu sa *n* vrsta i *m* kolona čiji su svi elementi jednaki proizvodu indeksa vrste i indeksa kolone u kojima se nalaze.

```
function A=proizvod(n,m)

for i=1:n
    for j=1:m
        A(i,j)=i*j;
    end
end
```

Naredna funkcija vraća kvadratnu matricu koja na sporednoj dijagonali sadrži dati vektor, a svi ostali elementi su nule.

```
function A=sporedna(v)

n=size(v,2);
A=zeros(n);
for i=1:n
    A(n-i+1,i)=v(i);
end
```

Za smeštanje elemenata datog vektora na sporednu dijagonalu, neophodno je odrediti indekse vrste i kolone pozicije u matrici na koju se smešta *i*-ti element – u pitanju je (*n-i+1*)-va vrsta i *i*-ta kolona. Do ovih formula se dolazi lako kada se pogledaju pozicije prvih nekoliko i poslednjih nekoliko elemenata vektora.

Treba primetiti da, iako nam je zadatak bio da kreiramo matricu, koristimo samo jednu for-petlju. Razlog leži u činjenici da smo najpre naredbom `zeros` u `A` smestili matricu traženih dimenzija koja sadrži sve nule, pa nam je nakon toga ostalo samo još da na sporednu dijagonalu postavimo elemente datog vektora. To znači da praktično samo još treba redom „proći“ kroz elemente vektora, a kao što smo videli, to radimo pomoću jedne for-petlje.

Daćemo još jedan primer, funkciju koja vraća kvadratnu matricu dimenzije za jedan veće od datog vektora, koja neposredno ispod glavne dijagonale sadrži dati vektor, sa obrnutim redosledom elemenata. Svi ostali elementi matrice ispod datog vektora su jedinice, a iznad datog vektora su nule.

S obzirom da je struktura matrice komplikovanija nego u prethodnom primeru, najlakše je formirati celu matricu element po element. Pritom, dodelu vrednosti elementima matrice vršimo na tri različita načina, zavisno od toga da li je vrednost koja se dodeljuje element vektora, jedinica ili nula.

```
function A=ispod(v)

n=size(v,2);
for i=1:n+1
    for j=1:n+1
        if i==j+1
            A(i,j)=v(n-j+1);
        elseif i>j+1
            A(i,j)=1;
        else
            A(i,j)=0;
        end
    end
end
```

U nastavku vidimo kako izgleda vraćena matrica za jedan poziv funkcije.

```
>> ispod([2 3 4 5])
ans =
     0     0     0     0     0
     5     0     0     0     0
     1     4     0     0     0
     1     1     3     0     0
     1     1     1     2     0
```

Sledeća funkcija vraća datu matricu, rotiranu za 90 stepeni u smeru kazaljke na satu. Rotiranje vršimo element po element, postavljajući svaki od elemenata na odgovarajuću poziciju u novoj matrici.

```
function B=rotacija(A)

[n m]=size(A);
for i=1:n
    for j=1:m
        B(j,n-i+1)=A(i,j);
    end
end
```

Isto ovo smo mogli uraditi i uzimajući redom vrste iz date matrice i stavljajući ih na odgovarajuću poziciju (tj. u odgovarajuću kolonu) nove matrice.

```
function B=rotacija2(A)

n=size(A,1);
for i=1:n
    B(:,i)=A(n-i+1,:)';
end
```

Pritom je umetanje kolona u matricu `B` iz prethodne funkcije moglo biti obavljeno i pomoću blok-matrica, bez eksplicitnog navođenja indeksa kolone u matrici B. Tako dobijamo i treću verziju iste funkcije.

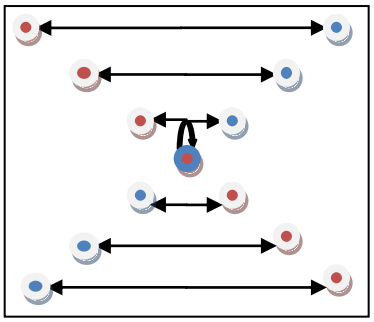
```
function B=rotacija3(A)

n=size(A,1);
B=[];
for i=1:n
    B=[A(i,:)' B];
end
```

Naravno, sve tri navedene verzije funkcije ispravno rešavaju pomenuti problem. I generalno, za sve probleme osim trivijalnih postoji mnoštvo različitih rešenja. Pritom, svako rešenje koje je semantički tačno smatramo dobrim, i ne želimo da izdvojimo jedno rešenje kao najbolje. Ocena toga koliko je rešenje „logično“ ili „lepo“ je individualna, i time se nećemo baviti. Može se diskutovati o tome koliko se koja funkcija brzo izvršava, ali ako je primenjujemo na podatke malog obima, brzina nije previše važan faktor.

Zadaci:

46. Napisati funkciju koja od uneta dva vektora *v* i *w* vraća vektor koji sadrži elemente sa parnih pozicija iz *v*, a posle njih elemente sa neparnih pozicija iz *w*.
47. Napisati funkciju koja za dati vektor *v* vraća vektor koji sadrži one elemente iz *v* koji su deljivi sa 3, ali čiji indeks nije deljiv sa 3.
48. Napisati funkciju koja za unet vektor *v* vraća vektor koji sadrži sve desne susede neparnih elementa iz *v*.
49. Napisati funkciju koja za uneta dva vektora *u* i *v* iste dužine vraća vektor koji se od vektora *u* dobija tako što svaki njegov negativni element zamenimo elementom iz *v* sa iste pozicije.
50. Napisati funkciju koja za datu matricu *M* vraća matricu koja se od *M* dobija tako što se nulama zamene elementi čiji je koren jednak zbiru indeksa vrste i kolone u kojima se nalazi, a ostali elementi ostanu isti.
51. Napisati funkciju koja vraća broj elemenata u datoj matrici *M* koji su deljivi ili sa 3 ili sa 5.
52. Napisati funkciju koja za unetu matricu *M* koja sadrži cele brojeve vraća zbir elemenata matrice koji su kvadrati nekog prirodnog broja, i proizvod onih elemenata iz *M* koji su deljivi indeksom kolone u kojoj se nalaze.
53. Napisati funkciju koja za datu matricu vraća skalarni proizvod vektora koji se nalaze u prvoj i poslednjoj vrsti te matrice.
54. Napisati funkciju koja vraća matricu koja se od unete kvadratne matrice *M* dobija tako što se elementi na glavnoj dijagonali množe odgovarajućim elementom sa sporedne dijagonale, a ostali elementi ostaju isti (v. sliku).



55. Napisati funkciju koja za unet prirodan broj n vraća matricu:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \\ \vdots & \vdots & \vdots \\ n & n^2 & n^3 \end{bmatrix}$$

56. Napisati funkciju koja za unet vektor $v=[v_1\ v_2\ v_3\ ...\ v_n]$ vraća kvadratnu matricu:

$$\begin{bmatrix} 5 & 5 & \cdots & 5 & v_n \\ 5 & 5 & \cdots & v_{n-1} & 5 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ v_1 & 5 & \cdots & 5 & 5 \end{bmatrix}$$

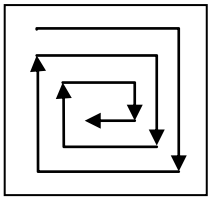
57. Napisati funkciju koja za unet vektor $v=[v_1\ v_2\ v_3\ ...\ v_n]$ vraća kvadratnu matricu:

$$\begin{bmatrix} v_n & 2 & 2 & \cdots & 2 \\ 3 & v_{n-1} & 2 & \cdots & 2 \\ 3 & 3 & v_{n-2} & \vdots & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 3 & 3 & 3 & \cdots & v_1 \end{bmatrix}$$

58. Napisati funkciju koja za unet vektor $v=[v_1\ v_2\ v_3\ ...\ v_n]$ vraća kvadratnu matricu:

$$\begin{bmatrix} v_1 & v_2 & v_3 & \cdots & v_{n-2} & v_{n-1} & v_n \\ 0 & v_2 & & & & v_{n-1} & 0 \\ 0 & & v_3 & & v_{n-2} & & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & & v_3 & & v_{n-2} & & 0 \\ 0 & v_2 & & & & v_{n-1} & 0 \\ v_1 & v_2 & v_3 & \cdots & v_{n-2} & v_{n-1} & v_n \end{bmatrix}$$

59. Napisati funkciju koja za unetu kvadratnu matricu M vraća zbir elemenata koji se nalaze na glavnoj dijagonali, u prvoj koloni ili u poslednjoj vrsti. Svaki element sabrati tačno jedanput!
60. Napisati funkciju koja ispisuje sve elemente unete kvadratne matrice M prolazeći kroz nju spiralno sleva na desno počevši od elemenata prve vrste (v. sliku).



Ekstremni elementi

U ovom delu bavimo se nalaženjem ekstremnog elementa, po nekom kriterijumu, u datom skupu. Da bismo to uradili, prolazimo redom kroz elemente skupa, sve vreme čuvajući informaciju o ekstremnom elementu skupa elemenata kroz koje smo do tada prošli.

Sledeća funkcija vraća najmanji element datog vektora.

```
function m=minimum(v)

n=size(v,2);
m=Inf;
for i=1:n
    if v(i)<m
        m=v(i);
    end
end
```

Svakim prolaskom kroz telo for-petlje posmatra se naredni član niza, i u slučaju da je on manji od svih prethodnih članova, njegova vrednost se smešta u promenljivu `m`. Kada se završi izvršavanje for-petlje, u promenljivoj `m` se nalazi najmanja vrednost niza `v`.

Pre izvršavanja for-petlje, promenljivoj `m` dodeljujemo vrednost `Inf`. Konstanta `Inf` u Matlab-u predstavlja vrednost „beskonačno“, tj. broj veći od svih (konačnih) brojeva. Samim tim, u prvom izvršavanju for-petlje, kada `i` ima vrednost 1, izraz `v(i)<m` je sigurno tačan, i tako obezbeđujemo da `m` dobije vrednost prvog elementa vektora `v`. Alternativno, umesto `m=Inf` smo mogli napisati i `m=v(1)`, modifikujući for-petlju tako da promenljiva `i` uzima vrednosti počevši od 2, umesto od 1.

Napominjemo da se minimalni i maksimalni element niza mogu dobiti i naredbama `min` i `max`.

Naredna funkcija vraća vrednost najvećeg i drugog po veličini elementa datog vektora (koji sadrži bar dva elementa).

```
function [m1 m2]=dva_najveca(v)

n=size(v,2);
m1=-Inf;
m2=-Inf;
for i=1:n
    if v(i)>m1
        m2=m1;
        m1=v(i);
    elseif v(i)>m2
        m2=v(i);
    end
end
```

Ažuriranje vrednosti promenljivih `m1` i `m2` se vrši u dva slučaja, ako je element koji posmatramo veći od do tada najvećeg elementa, ili ako to nije slučaj, ali je veći od elementa koji je do tada bio drugi po veličini.

Za dati vektor, *početni zbir* je zbir prvih nekoliko elemenata tog vektora. Sledeća funkcija vraća najveći početni zbir datog vektora.

```
function m=max_pocetni(v)

n=size(v,2);
m=-Inf;
for i=1:n
    s=sum(v(1:i));
    if s>m
        m=s;
    end
end
```

Zadaci:

61. Napisati funkciju koja za dati vektor *v* vraća najmanji od svih elemenata vektora (osim prvog i poslednjeg) koji imaju osobinu da su manji od razlike svojih susednih elemenata (razlike većeg i manjeg elementa).
62. Napisati funkciju koja za dati vektor (sa bar tri elementa) vraća isti taj vektor iz koga su uklonjena tri najmanja elementa.
63. Napisati funkciju koja za unetu matricu *A* koja sadrži prirodne brojeve vraća najveći element koji je deljiv sa 2, ali nije deljiv sa 3. Ukoliko nema takvog elementa u matrici, vratiti 0.

Sortiranje niza

Sortiranje niza brojeva je jedan od najvažnijih problema u teorijskom računarstvu. Za dati vektor, želimo da vratimo vektor koji sadrži iste vrednosti, ali uređene u neopadajućem poretku. Tako za vektor `[5 3 -1 2 7 3]` želimo da vraćeni vektor bude `[-1 2 3 3 5 7]`. Postoji mnogo različitih pristupa sortiranju, mi ćemo ovde navesti dva relativno prosta načina – tzv. *bubble-sort (sortiranje isplivavanjem)* i *insertion-sort (sortiranje umetanjem)*.

Kod **bubble-sort** algoritma, vrši se veći broj uzastopnih poređenja susednih elemenata u vektoru koji sortiramo. Konkretno, algoritam obavlja sortiranje vektora sa *n* elemenata u *n*-1 tzv. „prolaza“.

U prvom prolazu, najpre poredimo prva dva elementa vektora. U slučaju da je prvi veći od drugog, menjamo im mesta. Nakon toga, poredimo drugi i treći element, menjajući im mesta ako je drugi veći od trećeg. Isti proces ponavljamo i za treći i četvrti element, pa četvrti i peti, i redom sve parove susednih elemenata do kraja vektora, završavajući time prvi prolaz. Primetimo da se nakon prvog prolaza najveći element niza sigurno nalazi na poslednjoj poziciji u vektoru – on je kroz ovaj proces „isplivao“ na poslednju poziciju (otud potiče ime algoritma).

U drugom prolazu ponavljamo isti proces, redom poredeći susedne elemente vektora, s tim da završavamo prolaz jedan korak ranije – poređenjem elemenata na pozicijama *n*-2 i *n*-1 (imajući u vidu da se na poziciji *n* već nalazi najveći element vektora). Slično kao u prethodnom slučaju, nakon drugog prolaza element vektora koji je drugi po veličini „isplivao“ je na poziciju *n*-1.

Dalje nastavljamo sa prolazima, skraćujući svaki sledeći prolaz za jedno poređenje. Tako u *i*-tom prolazu poredimo elemente samo do pozicije *n*-*i*+1. Nakon (*n*-1)-og prolaza, vektor je sortiran.

U nastavku ćemo ilustrovati ovaj proces primerom, sortirajući vektor koji redom sadrži elemente 3, 9, 5, 6 i 1.

prvi prolaz:			
3 9 5 6 1	→	**3 9** 5 6 1	(3≤9, pa ne dolazi do zamene)
3 **9 5** 6 1	→	3 **5 9** 6 1	(9>5, pa dolazi do zamene)
3 5 **9 6** 1	→	3 5 **6 9** 1	(9>6, pa dolazi do zamene)
3 5 6 **9 1**	→	3 5 6 **1 9**	(9>1, pa dolazi do zamene)
drugi prolaz:			
3 5 6 1 9	→	**3 5** 6 1 9	
3 **5 6** 1 9	→	3 **5 6** 1 9	
3 5 **6 1** 9	→	3 5 **1 6** 9	
treći prolaz:			
3 5 1 6 9	→	**3 5** 1 6 9	
3 **5 1** 6 9	→	3 **1 5** 6 9	
četvrti prolaz			
3 1 5 6 9	→	**1 3** 5 6 9	

U nastavku sledi funkcija koja sortira vektor navedenim algoritmom.

```
function w=bubble(v)

n=size(v,2);
w=v;
for i=1:n-1
    for j=1:n-i
        if w(j)>w(j+1)
            x=w(j);
            w(j)=w(j+1);
            w(j+1)=x;
        end
    end
end
```

Kod **insertion-sort** algoritma, pored datog vektora *v* koristimo još jedan vektor *w*, koji je na početku prazan, a koji će, kada se algoritam izvrši, sadržati elemente datog vektora u traženom poretku. Elemente iz vektora *v* uzimamo jedan po jedan, svaki od njih umećući na odgovarajuće mesto u vektoru *w*, vodeći pritom računa da elementi u *w* uvek budu sortirani. Ovaj proces ćemo ilustrovati primerom.

v	*w* (pozicija za umetanje)	*w* (nakon umetanja)
5 3 9 4 0 7	ˇ	5
5 **3** 9 4 0 7	ˇ 5	3 5
5 3 **9** 4 0 7	3 5 ˇ	3 5 9
5 3 9 **4** 0 7	3 ˇ 5 9	3 4 5 9
5 3 9 4 **0** 7	ˇ 3 4 5 9	0 3 4 5 9
5 3 9 4 0 **7**	0 3 4 5 ˇ 9	0 3 4 5 7 9

U nastavku sledi funkcija koja sortira vektor navedenim algoritmom.

```
function w=insertion(v)

n=size(v,2);
w=[];
for i=1:n
    j=1;
    while j<=i-1 && v(i)>w(j)
        j=j+1;
    end
    w=[w(1:j-1) v(i) w(j:end)];
end
```

Za svaku vrednost promenljive `i`, element `v(i)` se u telu for-petlje umeće u vektor `w` na odgovarajuće mesto. Nakon izvršavanja bloka

```
j=1;
while j<=i-1 && v(i)>w(j)
    j=j+1;
end
```

promenljiva `j` sadrži poziciju u vektoru `w` pre koje treba ubaciti element `v(i)`. Nakon toga, koristeći standardno umetanje pomoću blok matrica, postavljamo `v(i)` na svoje mesto u vektoru `w`.

Treba imati u vidu da u nekim situacijama prilikom provere zadovoljenosti uslova `j<=i-1 && v(i)>w(j)` promenljiva `j` sadrži vrednost veću od dimenzije vektora `w`. Na primer, prilikom prvog prolaska kroz telo for-petlje `j` ima vrednost 1, a vektor `w` je prazan. Ako bismo u takvoj situaciji pokušali da pristupimo elementu `w(j)` došlo bi do greške. Ipak, do greške u našoj funkciji ne dolazi, jer se u Matlab-u tačnost konjunkcije dva iskaza proverava tako što se proveri tačnost prvog iskaza, i *samo ako je prvi iskaz tačan* proverava se i tačnost drugog iskaza. Ako je prvi iskaz netačan, cela konjunkcija je netačna bez obzira na tačnost drugog iskaza, pa se drugi iskaz ni ne posmatra. S obzirom da prilikom provere uslova `j<=i-1 && v(i)>w(j)` vektor `w` ima `i-1` elemenata, prvi iskaz u ovoj konjunkciji je tačan samo onda kada do navedene greške neće doći.

Na kraju, napominjemo da u Matlab-u postoji naredba `sort` koja služi za sortiranje niza, i ima isti efekat kao prethodne dve funkcije.

Zadaci:

64. Napisati funkciju koja dati vektor koji sadrži cele brojeve sortira nerastuće po ostatku pri deljenju sa 7.

65. Napisati funkciju koja dati vektor sa različitim celim brojevima uređuje testerasto – tako da je svaki element na parnoj poziciji veći od narednog, a svaki element na neparnoj poziciji manji od narednog (jedno testerasto uređenje vektora [1 2 3 4 6] je npr. 1<3>2<6>4; imati u vidu da testerasto uređenje vektora nije jedinstveno određeno).

Rekurzija

Rekurzija je tehnika programiranja koja je pogodna za rešavanje problema koje lako možemo svesti na jedan ili više problema koji su isti kao početni problem, ali su manjeg obima. U ovom procesu, ključni alat predstavljaju tzv. rekurzivni pozivi funkcije, odnosno pozivi (nove kopije) funkcije koja je u procesu izvršavanja. S obzirom da je u pitanju apstraktan koncept, sa njim ćemo se upoznavati postepeno, uz mnoštvo primera.

Prvi primer

Kod korišćenja rekurzije najpre je potrebno formulisati rešenje problema u rekurzivnom obliku, svodeći problem na prostiji analogan problem.

Počećemo jednostavnim primerom, koristeći rekurziju u funkciji koja za dat prirodan broj *n* ispisuje prvih *n* prirodnih brojeva u opadajućem redosledu. U slučaju da je *n*>1, problem se može rešiti tako što se ispiše broj *n,* a zatim se isti problem reši za broj *n*-1. Za n=1, problem se rešava tako što se jednostavno ispiše broj *n*. Ovu formulaciju koristimo da napišemo funkciju.

```
function stampa(n)

disp(n);
if n>1
    stampa(n-1);
end
```

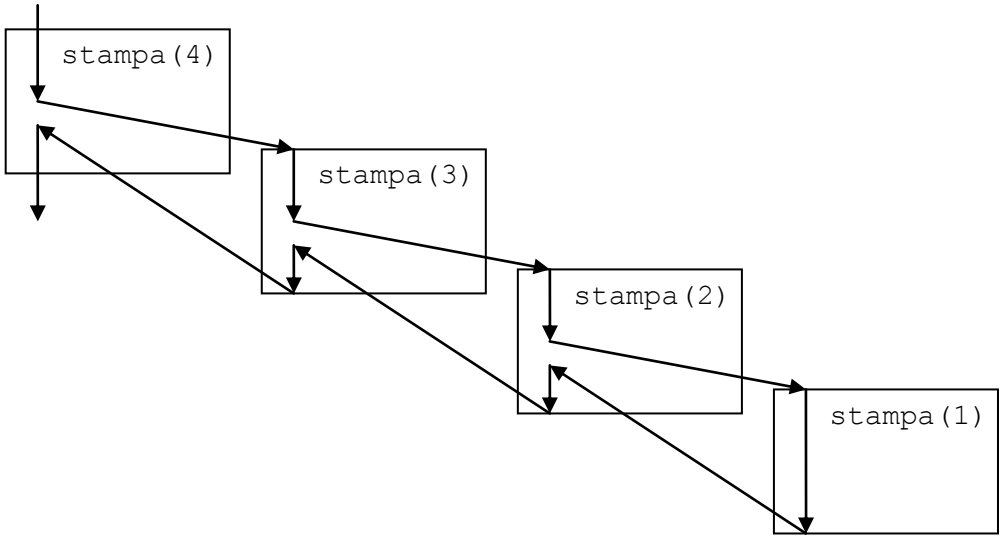
Primetimo najpre da se u telu funkcije `stampa` nalazi poziv iste te funkcije. Takav poziv nazivamo *rekurzivni poziv* funkcije. Iako na prvi pogled može da izgleda čudno, ovakav poziv je u Matlab-u dozvoljen, i štaviše – može da bude veoma koristan. Svaki novi rekurzivni poziv treba posmatrati kao izvršavanje nove kopije osnovne funkcije.

Kada je izvršimo, funkcija `stampa` radi upravo ono što želimo.

```
>> stampa(4)
     4
     3
     2
     1
```

Pritom, tokom izvršavanja funkcije `stampa(4)`, poziva se funkcija `stampa(3)`. Tokom njenog izvršavanja se poziva `stampa(2)`, a potom i

`stampa(1)`. Prilikom izvršavanja funkcije `stampa(1)` nema rekurzivnih poziva, jer prosleđeni parametar nije veći od jedan.



Treba imati u vidu da se tokom izvršavanja funkcije `stampa(1)` još nije završilo izvršavanje nijedne od tri funkcije koje su prethodno počele da se izvršavaju – izvršavanje svake od njih je zaustavljeno u momentu rekurzivnog poziva, i nastavlja se tek kada se završi izvršavanje rekurzivnog poziva. Prema tome, tek nakon završetka izvršavanja funkcije `stampa(1)` privodi se kraju izvršavanje funkcije `stampa(2)`, potom `stampa(3)`, i konačno i `stampa(4)`.

Kao što je već pomenuto, svaki od rekurzivnih poziva posmatramo kao zasebnu kopiju osnovne funkcije. Svi parametri u svakom od poziva su nezavisni od parametara u ostalim pozivima, iako imaju ista imena. Konkretno, u prethodnom primeru vrednost parametra *n* je drugačija u svakom od poziva funkcije.

Vraćanje vrednosti

Funkcije koje pozivamo rekurzivno mogu da vraćaju vrednost, što možemo da iskoristimo za postepeno izračunavanje.

Dvostruki faktorijel prirodnog broja *n* je proizvod svih prirodnih brojeva manjih ili jednakih *n* koji su iste parnosti kao *n*, i označava se sa *n*!!. Tako je 9!!=9·7 ·5·3·1= 945. Imajući u vidu rekurentnu zavisnost

$$n!! = \begin{cases} n \cdot (n-2)!!, & n > 2 \\ n, & n \le 2 \end{cases},$$

dvostruki faktorijel u Matlab-u možemo izračunati primenom rekurzije. Naredna funkcija za dati broj vraća njegov dvostruki faktorijel.

```
function df=dvostruki(n)

if n>2
    df=n*dvostruki(n-2);
```

```
else
    df=n;
end
```

Primetimo da u slučaju da prosleđeni parametar *n* nije veći od 2 ne dolazi do rekurzivnog poziva. Ova opcija naziva se *trivijalni slučaj*. Generalno gledano, takva opcija mora da postoji uvek kada koristimo rekurziju, da bismo izbegli kreiranje beskonačnog niza rekurzivnih poziva (koji se prilikom izvršavanja završava greškom, usled nedostatka memorije).

Fibonačijev niz je niz prirodnih brojeva čija su prva dva člana jedinice, a svaki sledeći je jednak zbiru prethodna dva. Prema tome, ako sa $f(n)$ označimo *n*-ti član niza, imamo

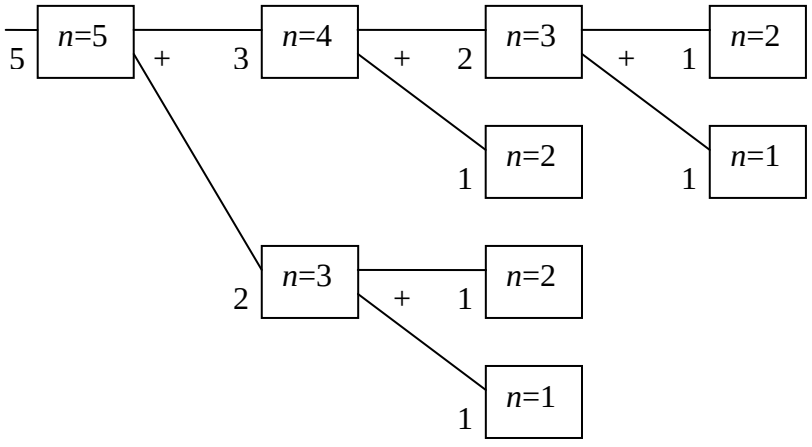
$$f(n)=\begin{cases} f(n-1)+f(n-2), & n>2 \\ 1, & n\le 2 \end{cases},$$

pa imajući ovo u vidu pišemo funkciju koja za dati prirodan broj *n* vraća *n*-ti član Fibonačijevog niza.

```
function f=fib(n)

if n>2
    f=fib(n-1)+fib(n-2);
else
    f=1;
end
```

S obzirom da se za svaki poziv funkcije (sa vrednošću parametra bar tri) dva puta rekurzivno poziva funkcija `fib`, dijagram poziva funkcija ima razgranatu strukturu. U nastavku ćemo prikazati proces izračunavanja petog člana niza. Svaki poziv funkcije označićemo vrednošću parametra *n* za koju je funkcija pozvana.



Zadaci:

66. Napisati funkciju koja za unet prirodan broj *n*, koristeći rekurziju vraća *n*-ti član niza zadatog formulom: $a_1 = 3;\ a_n = \frac{a_{n-1}}{2} + 2$.
67. Napisati funkciju koja za unet prirodan broj *n*, koristeći rekurziju vraća *n*-ti član niza zadatog formulom: $b_1 = 4;\ b_2 = 2;\ b_n = b_{n-1}^2 - b_{n-2} + 1$.
68. Napisati funkciju koja za unet prirodan broj *n*, koristeći rekurziju vraća *n*-ti Katalanov broj (*n*-ti Katalanov broj se računa po formuli $C_0 = 1;\ C_n = \frac{C_{n-1} \cdot (4n-2)}{n+1}$).

Još par primera

Sledeća funkcija za dat sortiran vektor *v* (u neopadajućem poretku) i prirodan broj *i* vraća tačno ako *i* pripada vektoru *v*, a inače vraća netačno. Pretraga se vrši tzv. *binarnim pretraživanjem,* gde se broj *i* poredi sa srednjim elementom vektora *v*, i zavisno od ishoda poređenja pretraživanje se u vektoru *v* rekurzivno nastavlja ili levo ili desno od srednjeg elementa. Ako je funkcija pozvana za prazan vektor, nema rekurzivnog poziva i vraća se netačno. S druge strane, ako je srednji element u vektoru baš jednak *i*, takođe nema rekurzivnog poziva i vraća se tačno.

```
function l=binarna(v,i)

n=size(v,2);
s=ceil(n/2);
if n==0
    l=0;
elseif v(s)==i
    l=1;
elseif v(s)>i
    l=binarna(v(1:s-1),i);
else
    l=binarna(v(s+1:n),i);
end
```

Kada koristimo rekurziju, ponekad je zgodno među ulazne parametre uvesti i pomoćni parametar koji se ne nalazi u opisu problema. Sledeća funkcija vraća niz koji sadrži elemente datog vektora u obrnutom redosledu. Obrnuti redosled elemenata vektora ćemo dobiti vršenjem zamene odgovarajućih elemenata, najpre prvog i poslednjeg, pa drugog i pretposlednjeg, itd. Pored vektora, funkciji prosleđujemo i parametar *i*, koji označava poziciju u vektoru za koju treba izvršiti sledeću zamenu. Zamene se vrše sve dok je parametar *i* manji od polovine dužine vektora.

```
function w=obrni(v,i)

n=size(v,2);
if i<=n/2
    x=v(i);
    v(i)=v(n-i+1);
    v(n-i+1)=x;
    w=obrni(v,i+1);
```

```
else
    w=v;
end
```

Kada pozivamo ovu funkciju, za vrednost parametra *i* stavljamo 1.

```
>> obrni([1 2 3 4 5],1)
ans =
     5     4     3     2     1
```

Zadaci:

69. Napisati funkciju koja za unet prirodan broj, vraća proizvod njegovih cifara.
70. Napisati funkciju koja za unet vektor *v* vraća broj njegovih neparnih elemenata.
71. Napisati funkciju koja vraća tačno ako je uneti vektor palindrom, a inače vraća netačno.

Rešavanje matematičkih problema

U narednom delu predstavićemo neke standardne tehnike koje se koriste za rešavanje konkretnih problema iz matematike i teorijskog računarstva.

Konjunkcije i disjunkcije većeg broja uslova

U nekim situacijama potrebno je proveriti da li je istovremeno zadovoljen veći broj uslova. Preciznije govoreći, u pitanju je provera da li je konjunkcija skupa iskaza tačna. Kao što je poznato, konjunkcija skupa iskaza je tačna samo ako su svi iskazi tačni. Probleme tog tipa rešavamo uvođenjem posebne promenljive kojoj najpre dajemo vrednost 1, odnosno „tačno“, a zatim prolazimo redom kroz pomenute iskaze, i svaki put kada naiđemo na netačan iskaz postavljamo vrednost promenljive na 0, odnosno „netačno“. Nakon prolaska kroz sve iskaze, promenljiva ima vrednost 1 ako i samo ako su svi iskazi bili tačni.

Sledeća funkcija vraća „tačno“ ako su svi elementi datog vektora parni brojevi, a inače vraća „netačno“.

```
function l=svi_parni(v)

n=size(v,2);
l=1;
for i=1:n
    if mod(v(i),2)~=0
        l=0;
    end
end
```

Na sličan način proveravamo i disjunkcije većeg broja iskaza, koristeći činjenicu da je disjunkcija iskaza zapravo ekvivalentna negaciji konjunkcije negacija istih iskaza,

$$x_1 \vee \ldots \vee x_n \equiv \neg(\neg x_1 \wedge \ldots \wedge \neg x_n).$$

Sledeća funkcija vraća „tačno“ ako je bar jedan element date matrice jednak nuli, a inače vraća „netačno“.

```
function l=jedan_nula(A)

[n m]=size(A);
l=0;
for i=1:n
    for j=1:m
        if A(i,j)==0
            l=1;
        end
    end
end
```

Evo još jednog primera. Naredna funkcija vraća „tačno“ ako je bar jedan element datog vektora jednak zbiru njemu susednih elemenata, a inače vraća „netačno“. Podrazumevaćemo da prvi i poslednji element niza imaju samo po jedan susedni element.

```
function l=zbir_suseda(v)

n=size(v,2);
l=0;
v=[0 v 0];
i=1;
while ~l && i<=n
    if v(i+1)==v(i)+v(i+2)
        l=1;
    end
    i=i+1;
end
```

Ovaj put umesto for-petlje koristimo while-petlju, što nam omogućava da završimo prolazak kroz vektor čim pronađemo jedan element koji zadovoljava dati uslov. To postižemo dodavanjem uslova `~l` uz `while`, pa čim promenljiva `l` dobije vrednost 1 prestaje dalje izvršavanje while-petlje.

Pre ulaska u while-petlju dodali smo nule na početak i na kraj vektora `v`, da ne bismo morali da posebno tretiramo prvi i poslednji element niza.

U nastavku dajemo i jedan kompleksniji primer. Sledeća funkcija za date vektore *v* i *w*, vraća „tačno“ ako vektor *v* sadrži vektor *w* kao podvektor (tj. ceo vektor *w* se nalazi negde unutar vektora *v*, na uzastopnim pozicijama), a inače vraća „netačno“.

```
function lg1=podvektor(v,w)

n=size(v,2);
m=size(w,2);
lg1=0;
```

```
for i=0:n-m
    lg2=1;
    for j=1:m
        if v(j+i)~=w(j)
            lg2=0;
        end
    end
    if lg2
        lg1=1;
    end
end
```

Ovaj put, uveli smo dve logičke promenljive, `lg1` i `lg2`. Promenljivu `lg2` koristimo za testiranje da li se vektor *w* pojavljuje u vektoru *v* na fiksiranoj poziciji. Konkretno, nakon izvršavanja bloka

```
lg2=1;
for j=1:m
    if v(j+i)~=w(j)
        lg2=0;
    end
end
```

promenljiva `lg2` ima vrednost jedan samo ako se vektor *w* pojavljuje u vektoru *v* neposredno nakon pozicije `i`. Ovaj blok se izvršava za svaku vrednost promenljive `i` između 0 i `n-m`, i ako nakon bilo koje od tih pozicija pronađemo vektor *w,* vrednost promenljive `lg1` se postavlja na 1.

Zadaci:

72. Napisati funkciju koja vraća „tačno“ ako je uneti vektor palindrom, a inače vraća „netačno“.
73. Napisati funkciju koja za dati vektor *v* vraća „tačno“ ako u njemu postoji element koji je veći od zbira svih ostalih elemenata tog vektora, a inače vraća „netačno“.
74. Napisati funkciju koja za unetu matricu *M* vraća „tačno“ ako je prvi element svake vrste manji od zbira svih elemenata te vrste, a inače vraća „netačno“.

Teorija brojeva

Naredni deo je posvećen nekim elementarnim problemima iz teorije brojeva, odnosno problemima koji se bave celim brojevima, sa posebnim osvrtom na deljivost, proste faktore, i sl. Kao što smo već videli, ostatak pri deljenju i deljivost brojeva proveravamo pomoću naredbe `mod`.

Funkcija koja sledi za dati prirodan broj *n* vraća najveći prirodan broj *k* takav da je *n* deljivo sa 2^k.

```
function k=stepen_2(n)

k=0;
while mod(n,2)==0
```

```
    k=k+1;
    n=n/2;
end
```

Sledeća funkcija vraća najveći zajednički delitelj (NZD) dva data prirodna broja. Algoritam koji koristimo je poznati Euklidov algoritam – rekurzivno veći od dva broja umanjujemo za vrednost manjeg, sve dok ne postanu jednaki. Kada postanu jednaki, jednaki su i traženom NZD.

```
function nzd=n_z_d(n,m)

while n~=m
    if n>m
        n=n-m;
    else
        m=m-n;
    end
end
nzd=n;
```

Naredna funkcija vraća „tačno“ ako je dati prirodan broj prost, a inače vraća netačno.

```
function l=prost(n)

l=1;
for i=2:n-1
    if mod(n,i)==0
        l=0;
    end
end
```

Ovu funkciju možemo napisati i u efikasnijoj formi, imajući u vidu da za svaki prirodan broj $n>2$ važi da je prost ako i samo ako nema delitelja između 2 i $\sqrt{n}$. Koristeći while-petlju, prekinućemo pretraživanje ovog intervala čim naiđemo na jednog delitelja broja n.

```
function l=prost_opt(n)

l=1;
i=2;
while l && i<=sqrt(n)
    if mod(n,i)==0
        l=0;
    end
    i=i+1;
end
```

Kao što je poznato, svaki prirodan broj n ima jedinstveni razvoj na proste faktore, $n = p_1^{s_1} \cdot p_2^{s_2} \cdots p_k^{s_k}$, gde su p_1, p_2, …, p_k različiti prosti brojevi, a s_1, s_2, …, s_k prirodni brojevi. Sledeća funkcija za dat prirodan broj n vraća upravo njegov

razvoj na proste faktore, u formi matrice koja ima dve vrste i *k* kolona, i koja u prvoj vrsti sadrži proste faktore broja *n*, a u drugoj njihove stepene.

```
function M=razvoj(n)

M=[];
for i=2:n
    s=0;
    while mod(n,i)==0
        n=n/i;
        s=s+1;
    end
    if s>0
        M=[M [i;s]];
    end
end
```

Treba primetiti da *n* pokušavamo da delimo svim brojevima između 2 i *n*, uključujući tu i složene brojeve. Ali, uzimajući u obzir da faktore tražimo redom prolazeći kroz brojeve od 2 do *n*, kao i da svaki put kada pronađemo faktor rekurzivno delimo *n* tim faktorom sve dok je to moguće, jasno je da će svaki pronađeni faktor biti prost broj. Napominjemo i da smo u okviru funkcije koristili blok naredbi vrlo sličan telu funkcije `stepen_2` koju smo dali ranije.

Sledi par primera izvršavanja prethodne funkcije.

```
>> razvoj(18)
ans =
     2     3
     1     2
>> razvoj(29)
ans =
    29
     1
>> razvoj(2070)
ans =
     2     3     5    23
     1     2     1     1
```

Eratostenovo sito je algoritam za nalaženje svih prostih brojeva manjih ili jednakih prirodnom broju *n*, prvi ga je opisao starogrčki matematičar Eratosten. Najpre napišemo redom sve brojeve od 2 do *n*. Počevši od prvog broja na spisku (broj dva), uklonimo sa spiska sve brojeve koji su deljivi tim brojem, a isto ponovimo i za svaki sledeći broj. Kada ovo uradimo za sve brojeve na spisku, algoritam je završen i brojevi koji nisu uklonjeni su svi prosti brojevi manji ili jednaki *n*.

Sledeća funkcija za dat prirodan broj *n* vraća vektor koji sadrži sve proste brojeve manje ili jednake *n*.

```
function v=eratosten(n)

v=2:n;
i=1;
s=n-1;
while i<s
```

```
    for j=s:-1:i+1
        if mod(v(j),v(i))==0
            v=[v(1:j-1) v(j+1:end)];
        end
    end
    i=i+1;
    s=size(v,2);
end
```

Kada u prethodnoj funkciji tražimo elemente u vektoru *v* koje treba ukloniti, to radimo od kraja vektora, kako pozicija elemenata koje još nismo pregledali ne bi bila izmenjena izbacivanjem prethodno pregledanih elemenata. Izvršavanje ove funkcije ilustruje sledeći primer.

```
>> eratosten(25)
ans =
     2     3     5     7    11    13    17    19    23
```

Za tri prirodna broja *a, b, c* kažemo da su Pitagorina trojka, ako važi $a^2 + b^2 = c^2$. Naredna funkcija štampa sve Pitagorine trojke brojeva manjih od datog prirodnog broja *n.* Pritom, vodimo računa da ista trojka (ili bilo koja njena permutacija) ne bude odštampana više puta.

Jasno je da u svakoj Pitagorinoj trojci važi $c>a$ i $c>b$. Da bismo eliminisali višestruki ispis permutacija iste trojke, dovoljno je da uvedemo restrikciju $a \leq b$. Treba imati u vidu da time i dalje nismo potpuno eliminisali ni jednu trojku.

Prema tome, želimo da prođemo kroz sve trojke *a, b, c* prirodnih brojeva takve da je $a \leq b < c$. To radimo modifikujući granice vektora u for-petljama.

```
function pitagora(n)

for a=1:n
    for b=a:n
        for c=b+1:n
            if a^2+b^2==c^2
                fprintf('%d %d %d\n',a,b,c);
            end
        end
    end
end
```

Savršenim brojem zovemo svaki prirodan broj koji je jednak zbiru svih svojih pravih delitelja (delitelja koji su manji od tog broja). Na primer, broj 28 je savršen, jer je 28=1+2+4+7+14. Naredna funkcija ispisuje sve savršene brojeve koji su manji ili jednaki datom broju *n.*

```
function savrseni(n)

for i=1:n
    s=0;
    for j=1:i/2
        if mod(i,j)==0
            s=s+j;
        end
```

```
    end
    if s==i
        disp(i);
    end
end
```

Primetimo da se u bloku

```
s=0;
for j=1:i/2
    if mod(i,j)==0
        s=s+j;
    end
end
```

kombinacijom for-petlje i if-grananja zapravo sabiraju svi pravi delitelji broja `i`, jer ni jedan pravi delitelj broja `i` nije veći od `i/2`, i taj zbir se smešta u promenljivu `s`.

Zadaci:

75. Napisati funkciju koja za datu matricu *M* koja sadrži prirodne brojeve vraća broj elemenata iz te matrice koji su kvadrati nekog prirodnog broja.
76. Napisati funkciju koja za uneta dva prirodna broja vraća „tačno“ ako su oni uzajamno prosti (nemaju zajedničkih delilaca), a inače vraća „netačno“.
77. Napisati funkciju koja za unet prirodan broj *n* vraća vektor u kome se nalaze svi prirodni brojevi *k* koji nisu veći od *n*, i koji imaju osobinu da im je proizvod pravih delilaca jednak k^2.
78. Napisati funkciju koja za date prirodne brojeve *n* i *m*, $n<m$, vraća vektor koji sadrži sve proste brojeve iz intervala [*n*, *m*].
79. Napisati funkciju koja vraća zbir prostih faktora unetog prirodnog broja *n*.
80. Napisati funkciju koja za unetu matricu *M* vraća broj njenih elemenata koji imaju tačno dva prosta faktora.

Cifre u zapisu broja

Ako posmatramo broj 3176, odmah nam je potpuno jasno da su cifre ovog broja 3, 1, 7 i 6. Kao što ćemo videti, proces izdvajanja cifara u Matlab-u nije trivijalan. Najpre, treba imati u vidu da je $3176 = 3\cdot 10^3 + 1\cdot 10^2 + 7\cdot 10^1 + 6\cdot 10^0$. Ako je dat prirodan broj *n*, njegovu cifru jedinica (prvu cifru zdesna) dobijamo pomoću `mod(n,10)`, dok za cifru stotina (treću cifru zdesna) moramo malo da se pomučimo – dobijamo je, na primer, na sledeći način.

```
>> n=3176
n =
        3176
>> floor(n/100)-10*floor(n/1000)
ans =
     1
```

Sledeća funkcija rešava ovaj problem odjednom za sve cifre, ona za dati prirodan broj vraća vektor koji sadrži redom sve cifre tog broja.

```
function v=cifre(n)

v=[];
while n>0
    v=[mod(n,10) v];
    n=floor(n/10);
end
```

Koristeći ovu funkciju možemo da rešimo mnoštvo problema u kojima je potrebno manipulisati ciframa broja. Recimo, najveću cifru broja *n* dobijamo jednostavno sa `max(cifre(n))`.

Funkcija koja sledi štampa sve četvorocifrene brojeve $\overline{abcd}$ takve da $\overline{abcd} = a + b^2 + c^3 + d^4$. Kada je izvršimo, štampaju se brojevi 1306, 1676 i 2427.

```
function cetvorocifreni

for i=1000:9999
    v=cifre(i);
    if i==v(1)+v(2)^2+v(3)^3+v(4)^4
        disp(i);
    end
end
```

Zadaci:

81. Napisati funkciju koja vraća „tačno“ ako je aritmetička sredina cifara datog prirodnog broja *n* jednaka broju njegovih cifara, a inače vraća „netačno“.
82. Napisati funkciju koja za dati prirodan broj vraća broj koji se zapisuje istim ciframa kojima je zapisan uneti broj, ali u obrnutom redosledu.
83. Napisati funkciju koja za unet prirodan broj *n* vraća vektor koji sadrži sve *n*-tocifrene brojeve čiji je zbir cifara jednak n^2.
84. Napisati funkciju koja vraća „tačno“ ako su sve cifre unetog broja prirodnog broja prosti brojevi, a inače vraća „netačno“ .
85. Napisati funkciju koja za unet vektor koji sadrži cele brojeve vraća broj elemenata koji imaju osobinu da su im prva i poslednja cifre jednake.
86. Napisati funkciju koja za datu matricu koja sadrži prirodne brojeve vraća broj njenih elemenata kojima je bar jedna cifra 0.

Brojni sistemi

Uobičajeni prikaz brojeva, ciframa od 0 do 9, nazivamo dekadni sistem, ili sistem sa bazom 10. Kao što smo već videli, svaka cifra u zapisu ima svoju „težinu“ – prva cifra zdesna je cifra jedinica, zatim sledi cifra desetica, pa cifra stotina, itd. Imajući to u vidu interpretiramo zapis broja, i tako je, na primer, $3176 = 3 \cdot 10^3 + 1 \cdot 10^2 + 7 \cdot 10^1 + 6 \cdot 10^0$.

Umesto baze 10, za bazu možemo uzeti i bilo koji drugi prirodan broj $b>1$. U brojnom sistemu sa bazom *b*, za zapis koristimo „cifre“ od 0 do b-1. Ako broj nije u dekadnom sistemu, navešćemo bazu *b* u indeksu broja da naglasimo u kom sistemu je

zapis. Tako, na primer, imamo $2143_5 = 2\cdot 5^3 + 1\cdot 5^2 + 4\cdot 5^1 + 3\cdot 5^0 = 298$, ili $1101_2 = 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 13$.

Kao što ćemo videti u nastavku, prevođenje broja iz jednog sistema u drugi se može iskoristiti za efikasno rešavanje mnoštva problema. Bez obzira koji brojni sistem je u pitanju, zapis broja ćemo čuvati u vektoru čiji elementi će redom biti cifre u tom zapisu. Sledeća funkcija vraća zapis datog broja *n* u sistemu sa bazom *b*.

```
function v=u_bazu(n,b)

v=[];
while n>0
    v=[mod(n,b) v];
    n=floor(n/b);
end
```

U obrnutom procesu, želimo da od zapisa broja u nekom brojnom sistemu dobijemo vrednost broja. Sledeća funkcija vraća broj *n* čiji se zapis nalazi u datom vektoru *v*, u bazi *b*.

```
function n=iz_baze(v,b)

s=size(v,2);
n=0;
for i=1:s
    n=b*n+v(i);
end
```

Zadaci:

87. Napisati funkciju koja za dati prirodni broj vraća zbir cifara tog broja zapisanog u bazi 3.
88. Napisati funkciju koja za dati prirodan broj vraća broj cifara tog broja zapisanog u bazi 6 koje su veće od 2.
89. Napisati funkciju koja vraća „tačno“ ako u zapisu datog prirodnog broja u bazi 2 ima više jedinica nego nula, a inače vraća „netačno“.
90. Napisati funkciju koja za dati prirodan broj *n* ispisuje prvih *n* prirodnih brojeva koji zapisani u bazi 7 imaju bar jednu cifru 1.

Varijacije sa ponavljanjem

Varijacije sa ponavljanjem skupa od *n* elemenata dužine *k* predstavljaju sve uređene *k*-torke elemenata iz tog skupa. Zbog jednostavnosti, pretpostavićemo da je pomenuti skup baš $\{0,1,\ldots,n-1\}$.

Za generisanje svih varijacija sa ponavljanjem koristićemo zapis broja u brojnom sistemu sa pažljivo odabranom bazom. Naime, ako uzmemo jednu proizvoljnu varijaciju sa ponavljanjem, zatim uklonimo sve nule koje se eventualno nalaze na početnim pozicijama (gledano sleva), i ono što ostane posmatramo kao zapis broja u sistemu sa bazom *n*, vidimo da je to broj sa najviše *k* cifara, dakle između 0 i $n^k - 1$. Nije teško proveriti da se na ovaj način zapravo uspostavlja

bijekcija između svih varijacija sa ponavljanjem skupa od n elemenata dužine k i skupa prirodnih brojeva $\{0,1,\ldots,n^k-1\}$. Dakle, da bismo prošli kroz sve varijacije sa ponavljanjem, treba samo da prođemo kroz sve brojeve iz pomenutog skupa, i da svaki zapišemo u brojnom sistemu sa bazom n, dodajući eventualno nule sleva da dobijemo zapis sa tačno k cifara.

Kao prvu demonstraciju ovog metoda, navodimo funkciju koja za date prirodne brojeve n i k štampa sve varijacije sa ponavljanjem skupa od n elemenata dužine k.

```
function var_pon(n,k)

for i=0:n^k-1
    v=[];
    broj=i;
    for j=1:k
        v=[mod(broj,n) v];
        broj=floor(broj/n);
    end
    disp(v);
end
```

Sledi primer izvršavanja ove funkcije.

```
>> var_pon(3,2)
     0     0
     0     1
     0     2
     1     0
     1     1
     1     2
     2     0
     2     1
     2     2
```

Treba primetiti da smo za nalaženje zapisa broja u sistemu sa bazom n ovde koristili for-petlju, dok smo u funkciji `u_bazu` koristili while-petlju. To je zato što u ovom slučaju želimo da dobijemo vektor sa tačno k elemenata, čak i ako u zapisu broja ima manje cifara.

Sledeća funkcija za dat prirodan broj k vraća broj binarnih nizova (nizova nula i jedinica) dužine k koji među svojim elementima imaju bar dve trećine jedinica. Zbir elemenata vektora koji sadrži samo nule i jedinice je zapravo jednak broju jedinica u tom vektoru, i tu činjenicu ćemo iskoristiti u ovoj funkciji.

```
function b=br_bin(k)

b=0;
for i=0:2^k-1
    v=[];
    broj=i;
    for j=1:k
        v=[mod(broj,2) v];
        broj=floor(broj/2);
    end
```

```
    if sum(v)>=2*k/3
        b=b+1;
    end
end
```

Naredna funkcija za dat prirodan broj *k* štampa sve *neopadajuće* nizove dužine *k* čiji su elementi brojevi od 1 do 5. Primetimo da je niz dužine *k* čiji su elementi brojevi od 1 do 5 zapravo isto što i varijacija s ponavljanjem skupa od 5 elemenata dužine *k*. Prema tome, možemo da adaptiramo niz naredbi iz prvog primera i prođemo kroz sve takve nizove. Naravno, da bismo dobili niz sa elementima od 1 do 5, niz sa elementima od 0 do 4 treba modifikovati uvećavanjem svih elemenata za jedan. Nakon toga, preostaje nam samo da za svaki niz proverimo da li je neopadajući, i da ga ispišemo ako jeste.

```
function neopadajuci_1_5(k)

for i=0:5^k-1
    v=[];
    broj=i;
    for j=1:k
        v=[mod(broj,5) v];
        broj=floor(broj/5);
    end
    v=v+1;

    l=1;
    s=1;
    while l && s<k
        if v(s)>v(s+1)
            l=0;
        end
        s=s+1;
    end

    if l
        disp(v);
    end
end
```

Iako na prvi pogled deluje komplikovano, blok spoljašnje for-petlje se sastoji iz tri relativno jednostavna dela. Naime, za svaku vrednost promenljive `i`, najpre se u vektor `v` smešta odgovarajući niz brojeva između 1 i 5. Zatim se proverava da li su elementi tog vektora uređeni u neopadajućem redosledu. Konačno, u slučaju da je odgovor na prethodno pitanje potvrdan, vektor `v` biva ispisan.

Zadaci:

91. Napisati funkciju koja vraća broj nizova elemenata {0,1,2} dužine *k* koji sadrže tačno 2 dvojke.
92. Napisati funkciju koja ispisuje varijacije od *n* elemenata *{0,1,2,...,n-1}* sa ponavljanjem dužine *k*, koje se sastoje od tačno 3 različite cifre.

93. Napisati funkciju koja ispisuje sve varijacije sa ponavljanjem dužine *k* skupa *{1,2,3,...,n}* u kojima se na parnim pozicijama nalaze neparni elementi.
94. Napisati funkciju koja ispisuje sve varijacije od *n* elemenata {1,2,3,...,*n*} sa ponavljanjem dužine *k*, kod kojih je zbir kubova elemenata veći od proizvoda kvadarata elemenata.
95. Napisati funkciju koja ispisuje sve varijacije od *n* elemenata *{1,2,3,...,n}* sa ponavljanjem dužine *k*, kod kojih nikoja dva susedna elementa nisu jednaka.

Partitivni skup

Partitivni skup skupa *A* je skup svih njegovih podskupova. Za prolazak kroz sve elemente partitivnog skupa, koristićemo vezu podskupova i binarnih nizova.

Neka je B proizvoljan podskup skupa A koji ima *n* elemenata. Ako svakom elementu skupa *A* pridružimo jedinicu ako taj element pripada skupu *B*, a nulu ako ne pripada, dobijamo niz nula i jedinica dužine *n*. Na primer, ako je *A*={1,2,3,4,5}, a *B*={2,5}, onda je odgovarajući binarni niz 0,1,0,0,1.

Nije teško proveriti da je ova veza između podskupova skupa *A* i binarnih nizova dužine *n* zapravo bijekcija, i na taj način generisanje podskupova možemo svesti na generisanje binarnih nizova, koje smo prethodno već savladali.

Naredna funkcija ispisuje sve podskupove skupa *A*, datog vektorom koji sadrži njegove elemente. Napominjemo da je pristup koji ovde koristimo veoma sličan generisanju binarnih nizova u funkciji `br_bin` – umesto binarnog vektora `v` formiramo vektor `podskup` koji sadrži elemente tekućeg podskupa.

```
function partitivni(skup)

k=size(skup,2);
for i=0:2^k-1
    podskup=[];
    broj=i;
    for j=1:k
        if mod(broj,2)==1
            podskup=[podskup skup(j)];
        end
        broj=floor(broj/2);
    end

    s=size(podskup,2);
    fprintf('{');
    for t=1:s
        fprintf('%d',podskup(t));
        if t<s
            fprintf(',');
        end
    end
    fprintf('}\n');
end
```

U drugom delu bloka for-petlje bavimo se ispisom koji je u skladu sa uobičajenom matematičkom notacijom. Napominjemo da smo umesto ovog

naprednog ispisa mogli jednostavno koristiti i naredbu `disp(podskup)`, s tim da u tom slučaju prazan skup ne bi bio ispisan, jer naredba `disp` ne daje nikakav ispis kada joj se kao parametar prosledi prazan vektor. Dajemo i primer izvršavanja prethodne funkcije.

```
>> partitivni([2 3 6])
{}
{2}
{3}
{2,3}
{6}
{2,6}
{3,6}
{2,3,6}
```

Prolaz kroz partitivni skup koji smo implementirali u prethodnoj funkciji omogućava nam da se bavimo i komplikovanijim problemima vezanim za podskupove. Sledeća funkcija za dati skup *A* vraća broj podskupova skupa *A* sa bar dva elementa čiji je zbir elemenata jednak nuli.

```
function b=zbir_elemenata(skup)

b=0;
k=size(skup,2);
for i=1:2^k-1
    podskup=[];
    broj=i;
    for j=1:k
        if mod(broj,2)==1
            podskup=[podskup skup(j)];
        end
        broj=floor(broj/2);
    end
    if size(podskup,2)>=2 && sum(podskup)==0
        b=b+1;
    end
end
```

Zadaci:

96. Napisati funkciju koja za unet skup ispisuje sve njegove podskupove koji sadrže bar polovinu ukupnog broja elemenata unetog skupa.
97. Napisati funkciju koja za unet skup ispisuje sve njegove podskupove u kojima je zbir elemenata veći od zbira elemenata početnog skupa.
98. Napisati funkciju koja vraća broj razbijanja datog skupa na 3 obeležena skupa, tako da svaki sadrži bar 2 elementa.

Permutacije

Permutacije skupa od *n* elemenata predstavljaju sve uređene *n*-torke elemenata tog skupa, takve da se svaki element skupa pojavljuje tačno jednom. Zbog jednostavnosti, pretpostavićemo da je pomenuti skup baš $\{1,2,\ldots,n\}$.

Prilikom generisanja svih permutacija zgodno je koristiti rekurziju. Naime, permutacije skupa $\{1,2,\ldots,n\}$ se mogu generisati tako što se najpre generišu permutacije skupa $\{1,2,\ldots,n-1\}$, a zatim se u svaku od njih ubaci element *n*, najpre na početak, zatim između prvog i drugog elementa, itd.

Sledeća funkcija štampa sve permutacije skupa od *n* elemenata. Pored parametra *n*, funkcija ima i ulazni parametar *v* koji će u inicijalnom pozivu biti prazan vektor, i koji tokom izvršavanja služi za prenos do tada formirane permutacije između poziva funkcije.

```
function permutacije(v,n)

s=size(v,2);
if n==0
    disp(v);
else
    for i=1:s+1
        permutacije([v(1:i-1) n v(i:s)],n-1);
    end
end
```

Prethodna funkcija se poziva na sledeći način.

```
>> permutacije([],3)
     1     2     3
     2     1     3
     2     3     1
     1     3     2
     3     1     2
     3     2     1
```

Kao što smo već pomenuli, funkciji se prosleđuje prazan vektor i broj elemenata skupa.

Koristeći funkciju `permutacije` kao osnovu, napisaćemo funkciju koja vraća *broj* permutacija dužine *n* takvih da im se nijedna dva susedna elementa ne razlikuju za jedan. Kao i kod prethodne funkcije, među ulaznim parametrima imamo i vektor (koji je prazan u inicijalnom pozivu). S obzirom da treba da vratimo broj permutacija, uvodimo i izlazni parametar.

```
function br=perm_br(v,n)

s=size(v,2);
if n==0
    br=1;
    for j=1:s-1
        if abs(v(j)-v(j+1))==1
            br=0;
```

```
        end
    end
else
    br=0;
    for i=1:s+1
        br=br+perm_br([v(1:i-1) n v(i:s)],n-1);
    end
end
```

U slučaju da je parametar *n* nula, proveravamo da li je uslov zadovoljen za vektor *v*, i ako jeste funkcija vraća vrednost jedan, a ako nije vraća nula. Ako *n* nije nula, funkcija se rekurzivno poziva za sva moguća produženja vektora *v*, a vraćena vrednost je zbir svih vraćenih vrednosti rekurzivnih poziva.

Za datu permutaciju π, *ciklus* je skup $\{a_1, a_2, \ldots, a_k\} \subseteq \{1,2,\ldots,n\}$ takav da se u π na poziciji a_1 nalazi elemenat a_2, na poziciji a_2 nalazi se a_3, i tako dalje. Konačno, na poziciji a_k nalazi se a_1. Za svaku permutaciju važi da su njeni ciklusi disjunktni, i da im je unija jednaka skupu $\{1,2,\ldots,n\}$. Na primer, ciklusi permutacije (3,4,1,5,2) su {1,3} i {2,4,5}, a permutacija (2,3,4,5,1) ima samo jedan ciklus, {1,2,3,4,5}.

Naredna funkcija štampa sve permutacije dužine *n* koje imaju samo jedan ciklus. Kao i kod prethodne dve funkcije, među ulaznim parametrima imamo i vektor (koji je prazan u inicijalnom pozivu).

```
function ciklus(v,n)

s=size(v,2);
if n==0
    c=v(1);
    k=1;
    while c~=1
        c=v(c);
        k=k+1;
    end
    if k==s
        disp(v);
    end
else
    for i=1:s+1
        ciklus([v(1:i-1) n v(i:s)],n-1);
    end
end
```

Na kraju, napominjemo i da se naredbom `perms` dobija matrica koja sadrži sve permutacije datog skupa.

Zadaci:

99. Napisati funkciju koja za unet skup ispisuje sve permutacije tog skupa.
100. Napisati funkciju koja ispisuje sve rasporede na šahovskoj tabli dimenzija *nxn* tako da se nikoja 2 topa ne napadaju.

Varijacije bez ponavljanja

Varijacije bez ponavljanja skupa od n elemenata dužine k predstavljaju sve uređene k-torke elemenata tog skupa, takve da se svaki element pojavljuje najviše jednom. Zbog jednostavnosti, pretpostavićemo da je pomenuti skup baš $\{1,2,\ldots,n\}$.

Generisanje svih varijacija bez ponavljanja može se obaviti slično generisanju svih permutacija. Glavna razlika leži u tome što kod permutacija svaki element skupa mora da se umetne u vektor, dok kod varijacija bez ponavljanja neke elemente skupa možemo i preskočiti. Sledeća funkcija štampa sve varijacije bez ponavljanja skupa od n elemenata dužine k.

```
function varBP(v,n,k)

s=size(v,2);
if k==0
    disp(v);
else
    for i=1:s+1
        varBP([v(1:i-1) n v(i:s)],n-1,k-1);
    end
    if n>k
        varBP(v,n-1,k);
    end
end
```

Prethodna funkcija se izvršava na sledeći način.

```
>> varBP([],3,2)
     2     3
     3     2
     1     3
     3     1
     1     2
     2     1
```

Zadaci:

101. Napisati funkciju koja za unete prirodne brojeve n i k ispisuje varijacije bez ponavljanja skupa *{1,2,3,...,n}* dužine k u kojima je najveći element neparan.
102. Napisati funkciju koja za unete prirodne brojeve n i k ispisuje varijacije bez ponavljanja skupa *{1,2,3,...,n}* dužine k u kojima su pojavljuju i 1 i k.

Kombinacije bez ponavljanja

Kombinacije bez ponavljanja skupa od n elemenata klase k predstavljaju svi podskupovi tog skupa koji imaju tačno k elemenata. Zbog jednostavnosti, pretpostavićemo da je pomenuti skup baš $\{1,2,\ldots,n\}$.

Prilikom generisanja svih kombinacija bez ponavljanja zgodno je koristiti rekurziju, a problem možemo rešiti slično kao i kod varijacija bez ponavljanja. S

obzirom da kod kombinacija redosled elemenata nije bitan, nove elemente u vektor uvek umećemo na početak. Zbog toga će svi vektori koje generišemo biti uređeni rastuće.

Sledeća funkcija štampa sve kombinacije bez ponavljanja skupa od *n* elemenata dužine *k*. Pored parametra *n*, funkcija ima i ulazni parametar *v* koji će u inicijalnom pozivu biti prazan vektor.

```
function kombBP1(v,n,k)

if k==0
    disp(v);
else
    kombBP1([n v],n-1,k-1);
    if n>k
        kombBP1(v,n-1,k);
    end
end
```

Prethodna funkcija izvršava se na sledeći način.

```
>> kombBP1([],5,3)
     3     4     5
     2     4     5
     1     4     5
     2     3     5
     1     3     5
     1     2     5
     2     3     4
     1     3     4
     1     2     4
     1     2     3
```

Podskupovi sa *k* elemenata se mogu generisati i na drugi način, tako što se najpre odabere najmanji element podskupa, označimo ga sa *m*, a nakon toga ostaje da se izabere podskup skupa $\{m+1,\ldots,n\}$ koji ima *k*-1 elemenata. Pritom, *m* može biti jedan od brojeva $1,2,\ldots,n-k+1$.

Sledeća funkcija štampa sve kombinacije bez ponavljanja skupa od *n* elemenata dužine *k*, odnosno radi isto što i funkcija `kombBP1`. Pored parametra *n* i parametra *v* koji će u inicijalnom pozivu biti prazan vektor, imamo i parametar *a* koji će u inicijalnom pozivu imati vrednost jedan. Ovaj parametar prenosi informaciju između rekurzivnih poziva funkcije o tome koji je najmanji element koji dolazi u obzir za sledeći element podskupa.

```
function kombBP2(v,a,n,k)

if k==0
    disp(v);
else
    for i=a:n-k+1
        kombBP2([v i],i+1,n,k-1);
    end
end
```

Prethodna funkcija izvršava se na sledeći način, a prosleđuju se četiri parametra – prazan vektor, jedinica, *n* i *k*. Treba primetiti da zbog drugačijeg načina generisanja kombinacija bez ponavljanja redosled ispisa kombinacija nije isti kao kod funkcije `kombBP1`.

```
>> kombBP2([],1,5,3)
     1     2     3
     1     2     4
     1     2     5
     1     3     4
     1     3     5
     1     4     5
     2     3     4
     2     3     5
     2     4     5
     3     4     5
```

Prethodnu funkciju lako možemo modifikovati da dobijemo prolaz kroz kombinacije bez ponavljanja *proizvoljnog* skupa. Sledeća funkcija za dat skup *S* i prirodan broj *k* vraća broj podskupova skupa *S* koji imaju *k* elemenata i svi elementi su istog znaka. Umesto broja *n*, funkciji prosleđujemo skup *S*.

```
function br=br_sk(v,a,S,k)

n=size(S,2);
if k==0
    if min(v)*max(v)>0
        br=1;
    else
        br=0;
    end
else
    br=0;
    for i=a:n-k+1
        br=br+br_sk([v S(i)],i+1,S,k-1);
    end
end
```

Napominjemo i da se naredbom `nchoosek` dobija matrica koja sadrži sve kombinacije bez ponavljanja datog skupa, dužine *k*.

Zadaci:

103. Napisati funkciju koja za unete prirodne brojeve *n* i *k*, ispisuje sve kombinacije bez ponavljanja dužine *k* elemenata iz skupa {1,2,3,...,*n*} kod kojih je zbir elemenata veći od kvadrata broja elemenata.
104. Napisati funkciju koja za unete prirodne brojeve *n* i *k* ispisuje sve kombinacije bez ponavljanja skupa *{1,2,3,...,n}* dužine *k* u kojima je broj parnih elemenata paran.

Kombinacije sa ponavljanjem

Multiskup je skup u kome se isti element može nalaziti i više puta. Kombinacije sa ponavljanjem skupa od n elemenata dužine k predstavljaju svi multiskupovi koji imaju tačno k elemenata i čiji elementi pripadaju datom skupu.

Primetimo da se skup svih kombinacija sa ponavljanjem skupa $\{1,2,\ldots,n\}$ dužine k bijektivno može preslikati na skup svih neopadajućih nizova dužine k čiji su elementi iz skupa $\{1,2,\ldots,n\}$.

Modifikacijom koda kojim smo generisali kombinacije bez ponavljanja možemo generisati i kombinacije sa ponavljanjem. Svaka od sledećih funkcija štampa sve kombinacije sa ponavljanjem skupa $\{1,2,\ldots,n\}$ dužine k.

```
function kombSP1(v,n,k)

if k==0
    disp(v);
else
    kombSP1([n v],n,k-1);
    if n>1
        kombSP1(v,n-1,k);
    end
end

function kombSP2(v,a,n,k)

if k==0
    disp(v);
else
    for i=a:n
        kombSP([v i],i,n,k-1);
    end
end
```

Kao i kod kombinacija bez ponavljanja, drugoj funkciji prosleđujemo dodatni parametar čija je vrednost 1. Ispod se može videti efekat ovih funkcija. Treba primetiti da je redosled kombinacija prilikom ispisa kod prve i druge funkcije različit.

```
>> kombSP1([],3,3)
     3     3     3
     2     3     3
     1     3     3
     2     2     3
     1     2     3
     1     1     3
     2     2     2
     1     2     2
     1     1     2
     1     1     1

>> kombSP2([],1,3,3)
     1     1     1
     1     1     2
     1     1     3
```

```
1     2     2
1     2     3
1     3     3
2     2     2
2     2     3
2     3     3
3     3     3
```

Zadaci:

105. Napisati funkciju koja za unete prirodne brojeve *n* i *k* ispisuje sve kombinacije sa ponavljanjem skupa *{1,2,3,..,n}* dužine *k* u kojima se najveći element javlja bar *k/2* puta.
106. Napisati funkciju koja za unete prirodne brojeve *n* i *k* ispisuje sve kombinacije sa ponavljanjem skupa *{1,2,3,..,n}* dužine *k* u kojima se ni jedan element ne pojavljuje više od 3 puta.

Varijacije sa ponavljanjem, još jednom

Već smo videli kako se generišu varijacije s ponavljanjem korišćenjem pretvaranja broja u brojni sistem sa drugom bazom. Sledeća funkcija štampa sve varijacije sa ponavljanjem skupa $\{1,2,\ldots,n\}$ dužine *k,* ovaj put korišćenjem rekurzije. Pored parametara *n* i *k,* funkciji treba proslediti i prazan vektor.

```
function varSP(v,n,k)

if k==0
    disp(v);
else
    for i=1:n
        varSP([v i],n,k-1);
    end
end
```

Zadaci:

107. Napisati funkciju koja za unete prirodne brojeve *n* i *k* vraća broj varijacija sa ponavljanjem skupa *{1,2,3,...,n}* dužine *k* u kojima su susedni elementi različiti.

Razbijanje broja u zbir

U ovom delu bavimo se problemom razbijanja broja *s* u zbir *l* nenegativnih celih brojeva, za date prirodne brojeve *s* i *l*. Na primer, 7 = 4 + 0 + 1 + 2 je jedno razbijanje broja 7 u zbir četiri broja. Mi želimo da generišemo sva razbijanja u zbir, i redosled sabiraka nam je bitan.

Sledeća funkcija za date prirodne brojeve *s* i *l* štampa sva razbijanja broja *s* u zbir *l* nenegativnih celih brojeva.

```
function razbijanje1(v,s,l)

if l==1
    disp([v s]);
else
    for i=0:s
        razbijanje1([v i],s-i,l-1);
    end
end
```

Sledi primer izvršavanja funkcije `razbijanje1`.

```
>> razbijanje1([],4,3)
     0     0     4
     0     1     3
     0     2     2
     0     3     1
     0     4     0
     1     0     3
     1     1     2
     1     2     1
     1     3     0
     2     0     2
     2     1     1
     2     2     0
     3     0     1
     3     1     0
     4     0     0
```

Za rešenje istog problema možemo iskoristiti i algoritam za generisanje svih kombinacija sa ponavljanjem. Neka je $\{a_1, a_2, \ldots, a_k\}$ jedna kombinacija sa ponavljanjem skupa $\{1,2,\ldots,n\}$ dužine *k*, tako da važi $a_1 \le a_2 \le \ldots \le a_k$. Tada je

$$(a_1 - 1) + (a_2 - a_1) + (a_3 - a_2) + \ldots + (a_k - a_{k-1}) + (n - a_k) = n - 1$$

razbijanje broja *n*–1 u zbir *k*+1 nenegativnih celih brojeva. Lako se proverava da je ovo preslikavanje kombinacija sa ponavljanjem u razbijanja u zbir bijektivno. Prema tome, da bismo generisali sva razbijanja broja *s* u zbir *l* nenegativnih celih brojeva, treba da prođemo kroz sve kombinacije sa ponavljanjem skupa $\{1,2,\ldots,s+1\}$ dužine *l*–1.

Sledeća funkcija za date prirodne brojeve *s* i *l* štampa sva razbijanja broja *s* u zbir *l* nenegativnih celih brojeva, tj. ova funkcija radi isto što i prethodna. U osnovi funkcije se nalazi algoritam iz funkcije `kombSP2`, modifikovan u skladu sa prethodnom analizom problema – umesto parametra *n* koristimo *s*+1, a umesto *k* koristimo *l*–1. Za razliku od prethodne funkcije, u ovom slučaju ćemo se malo više potruditi oko ispisa.

```
function razbijanje2(v,a,s,l)
```

```
if l==1
    v=[v s+1];
    fprintf('%d = %d ',s,v(1)-1);
    for j=2:size(v,2)
        fprintf('+ %d ',v(j)-v(j-1));
    end
    fprintf('\n');
else
    for i=a:s+1
        razbijanje([v i],i,s,l-1);
    end
end
```

Kao i kod funkcije `kombSP2`, prva dva prosleđena parametra prilikom poziva su prazan vektor i jedinica. Sledi primer izvršavanja funkcije `razbijanje2`.

```
>> razbijanje2([],1,4,3)
4 = 0 + 0 + 4
4 = 0 + 1 + 3
4 = 0 + 2 + 2
4 = 0 + 3 + 1
4 = 0 + 4 + 0
4 = 1 + 0 + 3
4 = 1 + 1 + 2
4 = 1 + 2 + 1
4 = 1 + 3 + 0
4 = 2 + 0 + 2
4 = 2 + 1 + 1
4 = 2 + 2 + 0
4 = 3 + 0 + 1
4 = 3 + 1 + 0
4 = 4 + 0 + 0
```

Zadaci:

108. Napisati funkciju koja za unete prirodne brojeve *n* i *k* štampa sva razbijanja broja *n* u zbir *k* brojeva, kod kojih se svaka dva susedna broja razlikuju bar za 2.
109. Napisati funkciju koja za unete prirodne brojeve *n* i *k* štampa sva razbijanja broja *n* u zbir *k* različitih brojeva.
110. Napisati funkciju koja za unete prirodne brojeve *n* i *k* štampa sva razbijanja broja *n* u proizvod *k* prirodnih brojeva.

Statističke ocene

Statistika je oblast koja se, uopšteno govoreći, bavi prikupljanjem i tumačenjem podataka, a mi ćemo se upoznati sa jednom od bazičnih metoda – tačkastom ocenom verovatnoće. Naime, ako u okviru eksperimenta posmatramo događaj koji se odigrava sa verovatnoćom koja nam nije poznata, tu verovatnoću možemo da odredimo približno, ponavljajući eksperiment više puta. Bez namere da

ulazimo dublje u matematičku analizu problema, reći ćemo još samo da što više puta ponovimo eksperiment, veće su šanse da naša procena bude „blizu“ stvarne vrednosti.

Na primer, bacamo kockicu za igru (koja na stranicama ima ispisane brojeve od jedan do šest), i zanima nas koja je verovatnoća da broj na kockici bude manji od tri. Ako bacimo kockicu sto puta, i u 36 bacanja broj na kockici bude manji od tri, procenjujemo pomenutu verovatnoću na 36/100=0,36. Inače, tačna vrednost verovatnoće je 1/3.

Ako za procenu koristimo računar, eksperiment možemo da ponovimo veći broj puta, što daje bolju ocenu. Da bismo simulirali eksperimente sa slučajnim ishodom u Matlab-u, koristimo naredbu `rand`. Konkretno, simulaciju bacanja kockice možemo obaviti pomoću izraza `ceil(6*rand)`, jer je njegova vrednost slučajno izabran broj iz skupa $\{1,2,\ldots,6\}$, i pritom nijedan od brojeva nije favorizovan.

Sledeća funkcija za dat prirodan broj *n* vraća procenu gorepomenute verovatnoće, ponavljajući eksperiment bacanja kocke *n* puta.

```
function p=ver_koc(n)

s=0;
for i=1:n
    if ceil(6*rand)<3
        s=s+1;
    end
end
p=s/n;
```

Kada pozovemo funkciju za razne vrednosti parametra *n,* vidimo da razlika između procene i tačne vrednosti verovatnoće varira. Naravno, čak i ako funkciju pozovemo sa istim parametrom *n,* vraćena vrednost može da bude drugačija.

```
>> ver_koc (100)
ans =
    0.3100
>> ver_koc (100)
ans =
    0.3500
>> ver_koc (1000)
ans =
    0.3270
>> ver_koc (100000)
ans =
    0.3346
>> ver_koc (1000000)
ans =
    0.3331
>> ver_koc (10000000)
ans =
    0.3333
```

Sledeća funkcija za dat prirodan broj *n* vraća procenu verovatnoće da na šest bačenih kockica za igru svi brojevi budu različiti, ponavljajući eksperiment *n* puta.

```
function p=ver_6(n)
```

```
s=0;
for i=1:n
    v=ceil(6*rand(1,6));
    w=zeros(1,6);
    for j=1:6
        w(v(j))=1;
    end
    if min(w)==1
        s=s+1;
    end
end
p=s/n;
```

U prethodnoj funkciji, za svako ponavljanje eksperimenta vektor koji sadrži brojeve na šest kockica kreiramo slično kao ranije – pomoću naredbi `ceil` i `rand`. Nakon toga, proveru da li su svi elementi vektora *v* različiti vršimo tako što u pomoćnom vektoru *w* beležimo pojavljivanje svakog od brojeva iz skupa {1,2,…,6} u vektoru *v*. Jasno, svi elementi vektora *v* su različiti ako i samo ako se svaki od elemenata skupa {1,2,…,6} u vektoru *v* pojavio bar jednom. Poslednji uslov će biti zadovoljen samo ako je svaki element vektora *w* jednak 1, pa je dovoljno proveriti da li je minimalni element tog vektora jednak 1.

Vrednost verovatnoće koju aproksimiramo je 0,0154… Kada pozovemo funkciju za razne vrednosti parametra *n*, vidimo da je razlika između procene i tačne vrednosti manja za veće *n*.

```
>> ver_6(100)
ans =
    0.0100
>> ver_6(1000)
ans =
    0.0170
>> ver_6(100000)
ans =
    0.0158
>> ver_6(1000000)
ans =
    0.0156
```

Pored bacanja kockice za igru, i bacanje novčića se lako simulira u Matlab-u. Na primer, izraz `floor(2*rand)` ima vrednost 0 ili 1, i obe vrednosti su isto verovatne, pa 0 možemo interpretirati kao „pismo“, a 1 kao „glavu“.

Sledeća funkcija za date prirodne brojeve *n* i *k* vraća procenu verovatnoće da se tokom bacanja novčića *k* puta pismo nije pojavilo tri puta za redom, ponavljajući eksperiment *n* puta.

```
function p=ver_k(n,k)

s=0;
for i=1:n
    v=floor(2*rand(1,k));
    b=1;
    for j=1:k-2
        if sum(v(j:j+2))==0
```

```
            b=0;
        end
    end
    s=s+b;
end
p=s/n;
```

Tačna verovatnoća za *k*=5 je 0,75, a ako pozovemo funkciju sa velikim prvim parametrom, dobijamo relativno dobru aproksimaciju.

```
>> ver_k(1000000,5)
ans =
    0.7497
```

Sledeća funkcija za date prirodne brojeve *n* i *k* vraća procenu verovatnoće da se tokom bacanja novčića *k* puta između svaka dva pisma glava pojavila bar dva puta, ponavljajući eksperiment *n* puta.

```
function p=ver_bar2(n,k)

s=0;
for i=1:n
    v=floor(2*rand(1,k));
    b=1;
    q=Inf;
    for j=1:k
        if v(j)==0
            if q<2
                b=0;
            end
            q=0;
        else
            q=q+1;
        end
    end
    s=s+b;
end
p=s/n;
```

Nakon što kreiramo vektor koji sadrži ishode bacanja novčića (vektor koji sadrži nule i jedinice), ostaje da proverimo da li se između svake dve nule u tom vektoru nalaze bar dve jedinice. Za tu proveru koristimo promenljivu *q*, kojom brojimo jedinice između susednih nula, i kada god naiđemo na nulu proveravamo da li je vrednost *q* bar dva. Pre početka prolaska kroz vektor promenljivoj *q* dodeljujemo vrednost beskonačno, jer kada naiđemo na prvu nulu uslov (koji proveravamo) je svakako još uvek zadovoljen, a *q* će u tom momentu i dalje imati vrednost beskonačno (bez obzira na eventualno prethodno uvećavanje), pa uslov if-grananja nije zadovoljen. Čim naiđemo na prvu nulu, promenljiva *q* dobija vrednost nula, i nakon toga je koristimo za brojanje jedinica na pomenuti način.

Tačna verovatnoća za *k*=4 je 0,375, a ako pozovemo funkciju sa velikim prvim parametrom, dobijamo relativno dobru aproksimaciju.

```
>> ver_bar2(1000000,4)
```

```
ans =
    0.3752
```

U ruletu, bacanjem kuglice bira se jedan broj iz skupa $\{0,1,2,\ldots,36\}$, pa taj izbor u Matlab-u možemo simulirati izrazom `floor(37*rand)`.

Sledeća funkcija za date prirodne brojeve *n* i *k* vraća procenu verovatnoće da nakon bacanja kuglice u ruletu *k* puta izabrani brojevi (ne neophodno u istom redosledu) čine aritmetičku progresiju, ponavljajući eksperiment *n* puta.

```
function p=ver_rulet(n,k)

s=0;
for i=1:n
    v=floor(37*rand(1,k));
    v=sort(v);
    b=1;
    for j=1:k-2
        if v(j+1)-v(j)~=v(j+2)-v(j+1)
            b=0;
        end
    end
    s=s+b;
end
p=s/n;
```

Proveru da li elementi vektora *v* čine aritmetičku progresiju vršimo tako što najpre sortiramo elemente u neopadajućem redosledu, a zatim proveravamo da li je razlika svaka dva susedna elementa ista.

S obzirom da tačnu vrednost verovatnoće koju aproksimiramo nije lako naći, važnost prethodne funkcije time je veća. Ako je pokrenemo npr. za $k=3$ i $n=10^7$, dobijamo približnu vrednost pomenute verovatnoće za $k=3$.

```
>> ver_rulet(10^7,3)
ans =
    0.0391
```

Zadaci:

111. Napisati funkciju koja za unet prirodan broj *n* nalazi verovatnoću da se prilikom bacanja 3 kockice paran broj pojavi bar 2 puta, ponavljajući eksperiment *n* puta.
112. Napisati funkciju koja za unete prirodne brojeve *n* i *k* vraća procenu verovatnoće da se tokom *k* bacanja kockice pojavi zbir *2k*, ponavljajući eksperiment *n* puta.
113. Napisati funkciju koja za unete prirodne brojeve *n* i *k* vraća procenu verovatnoće da se tokom *k* bacanja kockice pojavi broj 6 tačno *k/3* puta, ponavljajući eksperiment *n* puta.
114. Napisati funkciju koja za unete prirodne brojeve *n*, *k* i *m* vraća procenu verovatnoće da se među *k* slučajno odabranih brojeva iz skupa *{1,2,...,m}* pojavi neopadajući niz, ponavljajući eksperiment *n* puta.

115. Napisati funkciju koja za unete prirodne brojeve *n* i *k* vraća procenu verovatnoće da se tokom *k* bacanja novčića, pismo i glava naizmenično pojavljuju, ponavljajući eksperiment *n* puta.
116. Napisati funkciju koja za unete prirodne brojeve *n* i *k* vraća procenu verovatnoće da se tokom *k* bacanja kockica pojave najviše 3 različita broja, ponavljajući eksperiment *n* puta.
117. Napisati funkciju koja za unete prirodne brojeve *n* i *k* vraća procenu verovatnoće da se tokom igranja ruleta, gde kuglica može stati na jedan broj iz skupa {0,1,2,...,36}, prilikom *k* bacanja kuglice sve dobijene vrednosti budu kvadrati prirodnog broja, ponavljajući eksperiment *n* puta.

Rešenja zadataka

Promenljive i dodela

1.

```
>>xyz =
     4
>> abc=3
abc =
     3
>> xyz=xyz*2
xyz =
     8
>> abc=abc-xyz
abc =
    -5
```

Osnovne operacije i naredbe

2.

```
>> sin(pi/2)+cos(pi/6)-tan(pi/4)
ans =     0.8660
>> log2(log10(10000))+log(exp(4))
ans =      6
>> 5*pi+sqrt(20)+3*abs(-4)+floor(-2.2)+(2+6*i)*(2-6*i)
ans =    69.1801
```

Matrice

3.

```
>> v=22:2:40
v =
  Columns 1 through 9
    22    24    26    28    30    32    34    36    38
  Column 10
    40
```

4.

```
>> t=exp(1):exp(1):10*exp(1)
t =
  Columns 1 through 5
    2.7183    5.4366    8.1548   10.8731   13.5914
  Columns 6 through 10
   16.3097   19.0280   21.7463   24.4645   27.1828
```

5.

```
>> v=9:-1:-4
v =
  Columns 1 through 9
     9     8     7     6     5     4     3     2     1
  Columns 10 through 14
     0    -1    -2    -3    -4
>> v'
ans =
     9
     8
     7
     6
     5
     4
     3
     2
     1
     0
    -1
    -2
    -3
    -4
```

6.
```
>> A=[2 4 6; 3 9 0]
A =
2     4     6
3     9     0
```

7.
```
>>B=[1:2:9;pi:pi:5*pi;6:-1:2]
B =
  Columns 1 through 4
    1.0000    3.0000    5.0000    7.0000
    3.1416    6.2832    9.4248   12.5664
    6.0000    5.0000    4.0000    3.0000
  Column 5
    9.0000
   15.7080
    2.0000
```

Osnovne matrične operacije i naredbe

8.
```
>> B=4*eye(4)
B =
     4     0     0     0
     0     4     0     0
     0     0     4     0
```

```
     0     0     0     4
```

9.
```
>> B=-2*eye(4)+1
B =
    -1     1     1     1
     1    -1     1     1
     1     1    -1     1
     1     1     1    -1
```

10.
```
>> A=ones(2,3)
A =
     1     1     1
     1     1     1
>> B=[3 5 7; 2 1 3]
B =
     3     5     7
     2     1     3
>> (A+B)'
ans =
     4     3
     6     2
     8     4
```

11.
```
>> X=[3 5; 6 4; 7 8]
X =
     3     5
     6     4
     7     8
>> Y=[2 4 6; 8 2 4]
Y =
     2     4     6
     8     2     4
>> T=X*Y
T =
    46    22    38
    44    32    52
    78    44    74
>> log10(T)
ans =
    1.6628    1.3424    1.5798
    1.6435    1.5051    1.7160
    1.8921    1.6435    1.8692
>> K=3*ones(size(X,1),size(Y,2))
K =
     3     3     3
```

```
     3     3     3
     3     3     3
>> zeros(size(T))
ans =
     0     0     0
     0     0     0
     0     0     0
```

12.

```
>> A=eye(5)
A =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
>> B=6*ones(5)
B =
     6     6     6     6     6
     6     6     6     6     6
     6     6     6     6     6
     6     6     6     6     6
     6     6     6     6     6
>> C=diag([5 12 17 30 60])
C =
     5     0     0     0     0
     0    12     0     0     0
     0     0    17     0     0
     0     0     0    30     0
     0     0     0     0    60
a)>> R=A+B*C
R =
    31    72   102   180   360
    30    73   102   180   360
    30    72   103   180   360
    30    72   102   181   360
    30    72   102   180   361
b)>> det(R)
ans =
   745
c)>> rank(R)
ans =
     5
d)>> sum(R)
ans =
      151         361         511         901        1801
e)>> sum(sum(R))
ans =
```

```
        3725
```

Pristupanje delu matrice

13.

```
>> G=[1 2 3 4 5 6;5 6 7 8 9 10;9 10 11 12 13 14;13 14 15
16 17 18]
G =
     1     2     3     4     5     6
     5     6     7     8     9    10
     9    10    11    12    13    14
    13    14    15    16    17    18
>> G(:,[1 5])=[]
G =
     2     3     4     6
     6     7     8    10
    10    11    12    14
    14    15    16    18
```

14.

```
>> A=[11 21 31 41 51;12 22 32 42 52;13 23 33 43 53;14 24
34 44 54]
A =
    11    21    31    41    51
    12    22    32    42    52
    13    23    33    43    53
    14    24    34    44    54
>> N=A(1:2:end,[2 4])
N =
    21    41
    23    43
```

15.

```
>> A=[1 1 2 2 3 3 4 4; 5 6 6 7 7 8 8 9]
A =
     1     1     2     2     3     3     4     4
     5     6     6     7     7     8     8     9
>> A(:,[3 6])=A(:,[6 3])
A =
     1     1     3     2     3     2     4     4
     5     6     8     7     7     6     8     9
```

16.

```
>> X=[2 4 5 6; 3 2 1 7;8 7 6 5]
X =
     2     4     5     6
     3     2     1     7
```

```
     8     7     6     5
>> X(:,end:-1:1)
ans =
     6     5     4     2
     7     1     2     3
     5     6     7     8
```

Blok matrice

17.

```
>> A=[1 3; 2 5; 4 7]
A =
     1     3
     2     5
     4     7
>> A=[A [2;2;2]]
A =
     1     3     2
     2     5     2
     4     7     2
>> A=[A; [3 5 7]]
A =
     1     3     2
     2     5     2
     4     7     2
     3     5     7
```

18.

```
>>S=[1 3 5 7; 7 3 2 4;9 7 5 3; 1 2 1 2]
S =
     1     3     5     7
     7     3     2     4
     9     7     5     3
     1     2     1     2
>> B=[S -S zeros(size(S))]
B =
  Columns 1 through 9
     1     3     5     7    -1    -3    -5    -7     0
     7     3     2     4    -7    -3    -2    -4     0
     9     7     5     3    -9    -7    -5    -3     0
     1     2     1     2    -1    -2    -1    -2     0
  Columns 10 through 12
     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

19.

```
>> X=[1 2 3;4 5 6;9 8 7]
X =
     1     2     3
     4     5     6
     9     8     7
>> X=[X;X(1,:);ones(1,size(X,2))]
X =
     1     2     3
     4     5     6
     9     8     7
     1     2     3
     1     1     1
```

Slučajni brojevi:

20.

```
>> S=3*rand(3)
S =
    0.3336    0.7251    0.3959
    2.3408    1.2117    2.8262
    1.1692    0.2894    2.8684
```

21.

```
>> M=3+3*rand(3,4)
M =
    4.7256    4.0595    3.1291    5.1952
    3.1793    5.4636    3.5070    4.9432
    3.7043    3.0462    4.9473    4.3528
```

22.

```
>>T=fix(10*rand(3,5)+2)
T =
     7    11    10     2     4
    10     4     9    10     8
     3     4     3     3     4
>>T(:,[2 5])=[]
T =
     7    10     2
    10     9    10
     3     3     3
>>T([1 3],2)
ans =
    10
     3
```

23.

```
>> A=ceil(1000*rand(4))
A =
```

```
   353   203   199   932
   814   199    16   466
    10   604   747   419
   139   273   446   847
```

Skupovi

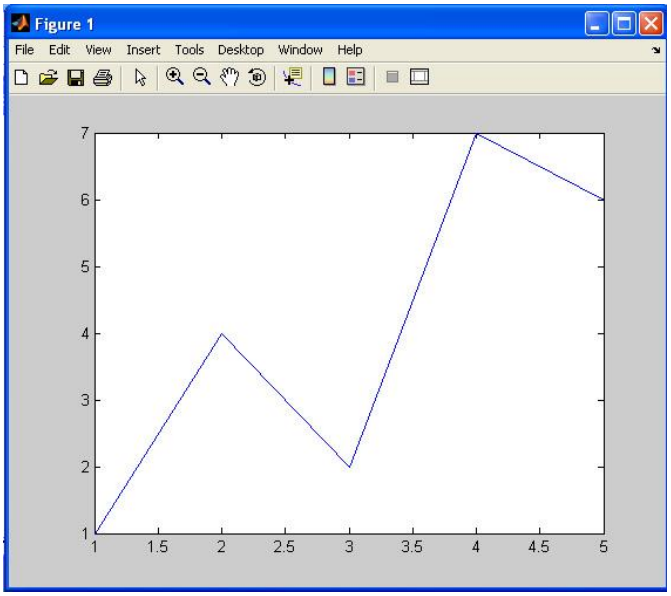
24.

```
>> S1=[1 6 3 4]
S1 =
     1     6     3     4
>> S2=[2 3 5 8]
S2 =
     2     3     5     8
>> S3=[2 5 7 9]
S3 =
     2     5     7     9
>> union(S1,S2)
ans =
     1     2     3     4     5     6     8
>> intersect(S2,S3)
ans =
     2     5
>> setdiff(S1,S2)
ans =
     1     4     6
>> setdiff(union(S1,S3),S2)
ans =
     1     4     6     7     9
>> setdiff(union(S2,S3),intersect(S2,S3))
ans =
     3     7     8     9
```

Grafički prikaz tačaka u ravni

25.

```
>> x=[1 4 2 7 6]
x =
     1     4     2     7     6
>> plot(1:5,x)
```



26.

```
>> x=0:0.1:1
x =
  Columns 1 through 5
         0    0.1000    0.2000    0.3000    0.4000
  Columns 6 through 10
    0.5000    0.6000    0.7000    0.8000    0.9000
  Column 11
    1.0000
>> plot(x,x.^2)
>> plot(x,x.^2,'g')
```

27.

```
>> t=0:0.2:1
t =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
>> plot(t,t.^2)
>> hold on
>> plot(t,t.^3,'y--')
>> hold off
```

28.

```
>>x=[0:0.1:2]
x =
  Columns 1 through 5
         0    0.1000    0.2000    0.3000    0.4000
  Columns 6 through 10
    0.5000    0.6000    0.7000    0.8000    0.9000
  Columns 11 through 15
    1.0000    1.1000    1.2000    1.3000    1.4000
  Columns 16 through 20
```

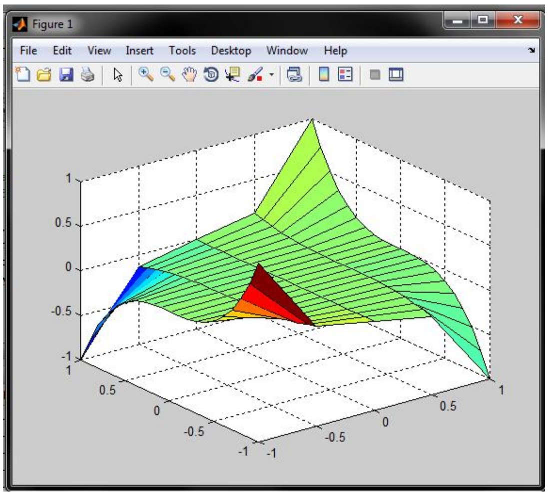
```
    1.5000    1.6000    1.7000    1.8000    1.9000
  Column 21
    2.0000
```

```
plot(x,x.^2,'gd:')
hold on
plot(x,x.^4,'ro--')
hold off
```

Grafički prikaz tačaka u prostoru

29.

```
>> [x y]=meshgrid(-1:0.5:1,-1:0.1:1);
>> surf(x,y,(x.*y).^3)
```

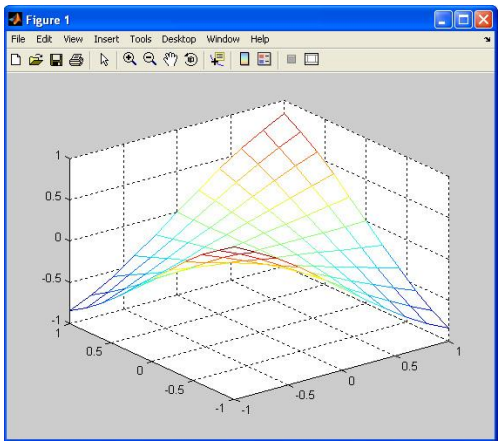


30.

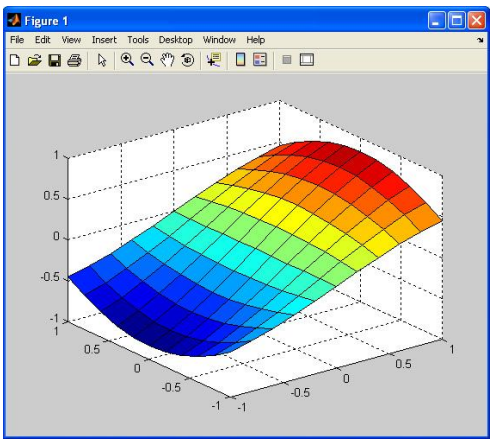
```
>> [x y]=meshgrid(-1:0.2:1)
x =
  Columns 1 through 5
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
   -1.0000   -0.8000   -0.6000   -0.4000   -0.2000
  Columns 6 through 10
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
```

```
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
         0    0.2000    0.4000    0.6000    0.8000
  Column 11
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
y =
  Columns 1 through 5
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000
   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000
   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000
   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000
         0         0         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.2000
    0.4000    0.4000    0.4000    0.4000    0.4000
    0.6000    0.6000    0.6000    0.6000    0.6000
    0.8000    0.8000    0.8000    0.8000    0.8000
    1.0000    1.0000    1.0000    1.0000    1.0000
  Columns 6 through 10
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000
   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000
   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000
   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000
         0         0         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.2000
    0.4000    0.4000    0.4000    0.4000    0.4000
    0.6000    0.6000    0.6000    0.6000    0.6000
    0.8000    0.8000    0.8000    0.8000    0.8000
    1.0000    1.0000    1.0000    1.0000    1.0000
  Column 11
   -1.0000
   -0.8000
   -0.6000
   -0.4000
```

```
   -0.2000
         0
    0.2000
    0.4000
    0.6000
    0.8000
    1.0000
>> mesh(x,y,sin(x.*y))
>> surf(x,y,sin(x).*cos(y))
```



mesh



surf

Logičke vrednosti i operatori

31.

```
>>((6<3)&&((2+1)==3))|||((4-2)~=7)
ans =
     1
```

32.

```
>> x=2
x =
     2
>> y=1
y =
     1
>> z=6
z =
     6
>> (x>y || x>z) && (x<y || x<z)
ans =
     1
```

33.

```
>> M=fix(15*rand(4))
```

```
M =

    14    13    12    13
     3    11     6    11
     9     6     9     2
     7     0    11     6

>> M==10
ans =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0

>> M>5
ans =
     1     1     1     1
     0     1     1     1
     1     1     1     0
     1     0     1     1

>> M~=2
ans =
     1     1     1     1
     1     1     1     1
     1     1     1     0
     1     1     1     1
```

If-grananje

34.
```
function rez = deljiv(b)

if mod(b,3)==0          % da li je deljiv sa 3?
    rez=b^2;
elseif mod(b,5)==0
    rez=3*b;
else
    rez=100;
end
```

35.
```
function prestupna(god)

if mod(god,400)==0
    disp('prestupna');
elseif mod(god,100)~=0 && mod(god,4)==0
    disp('prestupna');
else
    disp('nije prestupna');
```

```
end
```

For-petlja

36.
```
function w=parnePozicije(v)

n=size(v,2);
w=[];
for i=2:2:n
    w=[w v(i)];
end
```

37.
```
function treciStepen(v)

for a=v
    disp(a^3);
end
```

While-petlja

38.
```
function n=najmanji(a,b,c)

n=1;
while a*n+b^n<=c
    n=n+1;
end
```

39.
```
function stepen(a)

n=1;
s=3;
k=a;
while s<=k
    disp(n);
    n=n+1;
    s=s*3;
    k=k+a;
end
```

40.
```
function M = kvadrati(n)

M=[];
i=1;
while i^2<=n
    M=[M [i;i^2]];
    i=i+1;
end
```

Kombinovanje for-petlje i if-grananja

41.
```
function istiOstatak(v)

for a=v
    if mod(a,2)==mod(a,3)
        disp(a);
    end
end
```

42.
```
function w=podeli(v)

w=v;
for i=1:size(v,2)
    if mod(v(i),2)==0
        w(i)=v(i)/2;
    end
end
```

Brojanje, sabiranje i množenje

43.
```
function b = neDeljivi_a(v)

b=0;
for a=v
    if mod(a,5)~=0
        b=b+1;
    end
end
```

44.
```
function p=proizvod(v)

n=size(v,2);
p=1;
for i=1:n
    if v(i)>i
        p=p*v(i);
    end
end
```

45.
```
function rez=arSr(v)

br=ceil(n/2);
zbir=0;
n=size(v,2);
for i=1:2:n
```

```
    zbir=zbir+v(i);
end
rez=zbir/br;
```

For-petlje, matrice i vektori

46.
```
function u=noviVek(v,w)

u=[];
for i=2:2:size(v,2)
    u=[u v(i)];
end
for i=1:2:size(w,2)
    u=[u w(i)];
end
```

47.
```
function w=noviVek2(v)

w=[];
n=size(v,2);
for i=1:n
    if (mod(v(i),3)==0 ) && (mod(i,3)~=0)
        w=[w v(i)];
    end
end
```

48.
```
function w=susedi(v)

w=[];
n=size(v,2);
for i=1:n-1
    if mod(v(i),2)==1
        w=[w v(i+1)];
    end
end
```

49.
```
function u=zameni(u,v)

n=size(u,2);
for i=1:n
    if u(i)<0
        u(i)=v(i);
    end
end
```

50.
```
function A=zamena(M)
```

```
[n m]=size(M);
A=M;
for i=1:n
    for j=1:m
        if sqrt(M(i,j))== i+j
            A(i,j)=0;
        end
    end
end
```

51.
```
function b=deljiviSa3ili5(M)

b=0;
[n m]=size(M);
for i=1:n
    for j=1:m
        if mod(M(i,j),3)==0 || mod(M(i,j),5)==0
            b=b+1;
        end
    end
end
```

52.
```
function [z,p] = zbirProizv(M)

[n m]=size(M);
p=1;
z=0;
for i=1:n
    for j=1:m
        if sqrt(M(i,j))==fix(sqrt(M(i,j)))
            z=z+M(i,j);
        end
        if mod(M(i,j),j)==0
            p=p*M(i,j);
        end
    end
end
```

53.
```
function z=zbirMat

M=input('Unesite matricu!\n');
[n m]=size(M);
z=0;
for j=1:m
    z=z+M(1,j)*M(n,j);
end
```

54.
```
function A=mnozi(M)

A=M;
n=size(M,2);
for i=1:n
    A(i,i)=M(i,i)*M(i,n+1-i);
end
```

55.
```
function M=mat3(n)

for i=1:n
    for j=1:3
        M(i,j)=i^j;
    end
end
```

56.
```
function M = mat2(v)

n=size(v,2);
M=5*ones(n);
for i=1:n
    M(i,n-i+1)=v(n-i+1);
end
```

57.
```
function M= mat1(v)

n=size(v,2);
M=2*ones(n);
for i=1:n
    for j=1:i
        if i==j
            M(i,i)=v(n-i+1);
        else
            M(i,j)=3;
        end
    end
end
```

58.
```
function M=matOdVektora(v)

n=size(v,2);
M=diag(v);
M(1,:)=v;
M(n,:)=v;
for i=1:n
    M(i,n-i+1)=v(n-i+1);
```

```
end
```

59.
```
function  z = trougao(M)

n=size(M,1);
z=0;
if n>1
    for i=1:n-1
        z=z+M(i,i)+M(i+1,1)+M(n,i+1);
    end
elseif n==1
    z=M(1,1);
end
```

60.
```
function spirala(M)

n=size(M,1);
for t=1:ceil(n/2) %t označava koliko puta započinjemo
                  %kretanje sleva na desno
    for j=t:n+1-t
        fprintf('%d ',M(t,j));
    end
    for i=t+1:n+1-t
        fprintf('%d ',M(i,n+1-t));
    end
    for j=n-t:-1:t
        fprintf('%d ',M(n+1-t,j));
    end
    for i=n-t:-1:t+1
        fprintf('%d ',M(i,t));
    end
end
fprintf(' \n');
```

Ekstremni elementi

61.
```
function mn=minSusedi(v)

n=size(v,2);
mn=Inf;
for i=2:n-1
    if v(i)<abs(v(i+1)-v(i-1)) && v(i)<mn
        mn=v(i);
    end
end
```

62.
```
function w=izbaciNajmanja3(v)
```

```
w=v;
for k=1:3  %ponavljamo postupak izbacivanja 3 puta
    n=size(w,2);
    min=Inf;
    for i=1:n
        if w(i)<min
            ind=i;
            min=w(i);
        end
    end
    w(ind)=[];
end
```

63.
```
function mx=maxDeljiv(A)

[n m]=size(A);
mx=0;
for i=1:n
    for j=1:m
        if A(i,j)>mx && mod(A(i,j),2)==0 &&
                        mod(A(i,j),3)~=0
            mx=A(i,j);
        end
    end
end
```

Sortiranje niza

64.
```
function v=sortNerastuce(v)

n=size(v,2);
for i=1:n-1
    for j=n:-1:i+1
        if mod(v(j),7)>mod(v(j-1),7)
            %ako je ostatak pri deljenju v(j) sa 7
            %veci nego pri deljenju v(j-1) sa 7
            pom=v(j);
            v(j)=v(j-1);
            v(j-1)=pom;
        end
    end
end
```

65.
```
function v=sortirajTesterasto(v)

n=size(v,2);
```

```
mn=1;
%množilac koji menja znak elemenata i olakšava
%nam proveru uslova tako što uvek pitamo da li
%je mn*levi element manji od mn*desni element
for i=1:n-1
%dovoljno je da kroz sortiran niz prodjemo jedanput.
%Za dva posmatrana elementa ispitujemo da li ih treba
%zameniti ili ne. Neka su svi elementi zakljucno sa v(i)
%testerasto sortirani. Ako vazi da je v(i-1)<v(i),
%treba da bude v(i)>v(i+1). Ukoliko to nije tako,
%zamenom v(i) i v(i+1) nece se narusiti poredak elemenata
%v(1),...,v(i), jer ce i nova vrednost na mestu v(i) biti
%veca od v(i-1),a v(i) i v(i+1) ce biti pravilo uredjeni.

%provera da li se elementi nalaze u neodgovarajucem
%poretku
    if mn*v(i)>mn*v(i+1)  %ako da, zamenimo ih
       pom=v(i);
       v(i)=v(i+1);
       v(i+1)=pom;
   end
   mn=-mn;
end
```

Rekurzija

Vraćanje vrednosti

66.
```
function a=rek1(n)

if n==1
   a=3;
else
    a=rek1(n-1)/2+2;
end
```

67.
```
function b=rek2(n)

if n==1
    b=4;
elseif n==2
    b=2;
else
    b=rek2(n-1)^2-rek2(n-2)+1;
end
```

68.
```
function cn=katalan(n)
```

```
if n==0
    cn=1;
else
    cn=(4*(n-1)+2)*katalan(n-1)/(n+1);
end
```

Još par primera

69.
```
function p=proizv(n)

if n<=9
    p=n;
else
    p=mod(n,10)*proizv(floor(n/10));
end
```

70.
```
function b=brNep(v)

n=size(v,2);
if n==0
    b=0;
else
    if mod(v(n),2)==1
        b=1+brNep(v(1:n-1));
    else
        b=brNep(v(1:n-1));
    end
end
```

71.
```
function l=rekPalindrom(v)

n=size(v,2);
if n<=1
    l=1;
else
    l= (v(1)==v(n)) && rekPalindrom(v(2:n-1));
end
```

Rešavanje matematičkih problema

Konjunkcije i disjunkcije većeg broja uslova

72.
```
function l = palindrom(v)

l=1;
```

```
n=size(v,2);
for i=1:floor(n/2)
    if v(i)~=v(n+1-i)
        l=0;
    end
end
```

73.
```
function l=veciOdZbira(v)

l=0;
zbir=sum(v);
for a=v
    if a>zbir-a
        l=1;
    end
end
```

74.
```
function l=manjiZbirMatrica(M)

l=1;
n=size(M,1);     %interesuje nas broj vrsta, jer za svaku
                 %vrstu pojedinacno racunamo zbir.
i=1;
while i<=n && l
    if M(i,1)>=sum(M(i,:));
        l=0;
    end
    i=i+1;
end
```

Teorija brojeva

75.
```
function br=brojKvadrate(M)

br=0;
[n m]=size(M);
for i=1:n
    for j=1:m
        if sqrt(M(i,j))==round(sqrt(M(i,j)))
            br=br+1;
        end
    end
end
```

76.
```
function l=uzProsti(a,b)
```

```
m=min(a,b);
l=1;
for i=2:m
    if mod(a,i)==0 && mod(b,i)==0
        l=0;
    end
end
```

77.
```
function v=praviDel(n)

v=[];
for k=2:n
    p=1;
%najmani pravi delilac je 1, ali je on vec sadrzan u p.
    for i=2:k/2
        if mod(k,i)==0
            p=p*i;
        end
    end
    if p==k^2
        v=[v k];
    end
end
```

78.
```
function v=izmedju(n,m)

v=[];
for i=n:m
    prost=1;
    b=2;
%b je najmanji moguci delilac nekog broja razlicit od 1
    while prost && b<=sqrt(i)
        if mod(i,b)==0
            prost=0;
        end
        b=b+1;
    end
    if prost
        v=[v i];
    end
end
```

79.
```
function z=zbirProstFakt(n)

z=0;
for f=2:n
    b=0;
```

```
        while mod(n,f)==0
            b=b+1;
            n=n/f;
        end
        if b>0      %ako se prost faktor pojavljuje u broju
            z=z+f;        %dodamo u zbir taj faktor
        end
    end
```

80. `function br=dvaProstaFaktora(M)`

```
    br=0;
    [n m]=size(M);
    for i=1:n
        for j=1:m
%brojimo koliko svaki element M(i,j) ima prostih faktora.

            pom=M(i,j);   %kopiramo vrednost svakog elemeta
            f=2;          %najmanji prost faktor je 2
            brPro=0;      %broj prostih faktora je 0
            while pom>1
                if mod(pom,f)==0
                  brPro=brPro+1;
                end
                while mod(pom,f)==0
                    pom=pom/f;
                end
                f=f+1;
            end
            if brPro==2
                br=br+1;
            end
        end
    end
```

Cifre u zapisu broja

81. `function b=aritSred(n)`

```
    zc=0;                 %zbir cifara je 0;
    brCif=0;
    b=0;
    while n>0
        zc=zc+mod(n,10);  %uzimamo poslednju cifru i dodajemo
                          %je zbiru
```

```
    n=floor(n/10);    %uklanjamo poslednju cifru iz
broja.
    brCif=brCif+1;
end
ars=zc/brCif;         %racunamo aritmeticku sredinu.
if ars==brCif
    b=1;
end
```

82.
```
function b=unazad(n)

b=0;
while n>0
    cif=mod(n,10);
    b=b*10+cif;
    n=floor(n/10);
end
```

83.
```
function v=zbirCif(n)

v=[];
for i=10^(n-1):10^n-1
    z=0;
    b=i;
    while b>0
        cif=mod(b,10);
        z=z+cif;
        b=floor(b/10);
    end
    if z==n^2
        v=[v i];
    end
end
```

84.
```
function l=prosteCifre(n)

l=1;       %pretpostavimo da su sve cifre proste
while n>0
    cif=mod(n,10);
    d=2;
    prost=1;
    while d<=sqrt(cif) && prost
        if mod(cif,d)==0
            prost=0;
        end
        d=d+1;
    end;
```

```
    if ~prost
        l=0;
    end
    n=floor(n/10);
end
```

85.
```
function br=jednakaPrvaIposlCif(v)

br=0;
for i=1:size(v,2)
    n=v(i);
    posC=mod(n,10);
    n=floor(n,10);
    while n>=10
      n=floor(n/10);
    end
    pCif=n;
    if poslC==pCif
        br=br+1;
    end
end
```

86.
```
function br=nulaKaoCifra(M)

br=0;
[n m]=size(M);
for i=1:n
   for j=1:m
       a=M(i,j);
       lCif0=0;
%logička promenljiva koja označava da li broj
%ima nulu kao cifru. Na početku je false.
       while a>0
           cif=mod(a,10);
           if cif == 0
               lCif0=1;
           end
           a=floor(a/10);
       end
       if lCif0
           br=br+1;
       end
   end
end
```

Brojni sistemi

87.
```
function z=baza3(n)

v=[];
while n>0
    v=[mod(n,3) v];
    n=floor(n/3);
end
z=sum(v);
```

88.
```
function b=uBazi6(n)

b=0;
while n>0
    cif=mod(n,6);
    if cif>2
        b=b+1;
    end
    n=floor(n/6);
end
```

89.
```
function l=binarniJedinice(n)

v=[];
while n>0
    v=[mod(n,2) v];
    n=floor(n/2);
end

%proveravamo koliko ima jedinica u zapisu broja tako sto
%saberemo elemente vektora.
%Kako su u pitanju samo jedinice i nule,
%ako ima vise od jedinica nego nula,
%zbir ce biti veci od polovine
%ukupnog broja elemenata.
if sum(v)>size(v,2)/2
    l=1;
else
    l=0;
end
```

90.
```
function ispisi(n)

br=0;
k=1; %k prolazi kroz prirodne brojeve, pocevsi od 1.
while br<n
    pom=k; %kopiramo vrednost broja k, da se ne izgubi.
    v=[];
```

```
    %pretvaramo k u bazu 7.
    while pom>0
        v=[mod(pom,7) v];
        pom=floor(pom/7);
    end

%provera da li postoji cifra 1 u zapisu u bazi 7.
    l=0;
    for i=1:size(v,2)
        if v(i)==1
            l=1;
        end
    end
    if l
        disp(k);
        br=br+1;
    end
    k=k+1; %prelazak na sledeci prirodan broj.
end
```

Varijacije sa ponavljanjem

91.
```
function b=var1(k)

b=0;
for i=0:3^k-1
    v=[];
    broj=i;
    for j=1:k
        v=[mod(broj,3) v];
        broj=fix(broj/3);
    end
    br2=0;
    for j=1:k
        if v(j)==2
            br2=br2+1;
        end
    end
    if br2==2
        b=b+1;
    end
end
```

92.
```
function var2(n,k)

for i=0:n^k-1
    v=[];
```

```
    broj=i;
    for j=1:k
       v=[mod(broj,n) v];
       broj=fix(broj/n);
    end
    w=v;
    s=size(w,2);
    j=1;
    while j<s
        for t=s:-1:j+1
            if w(t)==w(j)
                w=[w(1:t-1) w(t+1:end)];
            end
        end
        j=j+1;
        s=size(w,2);
    end
    if size(w,2)==3
        disp(v);
    end
 end
```

93.
```
function var3(n,k)

for i=0:n^k-1
    v=[];
    broj=i;
    for j=1:k
        v=[mod(broj,n) v];
        broj=fix(broj/n);
    end
    v=v+1;
    l=1;
    s=2;
    while l && s<=k
        if mod(v(s),2)==0
            l=0;
        end
        s=s+2;
    end

    if l
        disp(v);
    end
end
```

94.
```
function var4(n,k)
```

```
for i=0:n^k-1
    v=[];
    broj=i;
    for j=1:k
        v=[mod(broj,n) v];
        broj=fix(broj/n);
    end
    v=v+1;
    if sum(v.^3)>prod(v.^2)
        disp(v);
    end
end
```

95.
```
function var5(n,k)

for i=0:n^k-1
    v=[];
    broj=i;
    for j=1:k
        v=[mod(broj,n) v];
        broj=fix(broj/n);
    end
    v=v+1;
    l=1;
    s=1;
    while l && s<k
        if v(s)==v(s+1)
            l=0;
        end
        s=s+1;
    end
    if l
        disp(v);
    end
end
```

Partitivni skup

96.
```
function partit1(skup)

k=size(skup,2);
for i=0:2^k-1
    v=[];
    broj=i;
    for j=1:k
        if mod(broj,2)==1
            v=[v skup(j)];
```

```
        end
        broj=floor(broj/2);
    end
    if size(v,2)>=k/2
        disp(v);
    end
end
```

97.
```
function partit2(skup)

k=size(skup,2);
for i=0:2^k-1
    v=[];
    broj=i;
    for j=1:k
        if mod(broj,2)==1
            v=[v skup(j)];
        end
        broj=fix(broj/2);
    end
    if sum(v)>sum(skup)
        disp(v);
    end
end
```

98.
```
function b=partit3(skup)

b=0;
n=size(skup,2);
for i=0:3^n-1
    s1=[];
    s2=[];
    s3=[];
    broj=i;
    for j=1:n
        if mod(broj,3)==0
            s1=[s1 skup(j)];
        elseif mod(broj,3)==1
            s2=[s2 skup(j)];
        else
            s3=[s3 skup(j)];
        end
        broj=fix(broj/3);
    end

    if size(s1,2)>=2 && size(s2,2)>=2 && size(s3,2)>=2
        b=b+1;
    end
```

```
end
```

Permutacije

99.
```
function perm(v,Sk)
n=size(v,2);
s=size(Sk,2);
if s==0
    disp(v);
else
    for i=1:n+1
        perm([v(1:i-1) Sk(s) v(i:n)],Sk(1:s-1));
    end
end
```

100.
```
function topovi(v,n)

s=size(v,2);
if n==0
    fprintf('{');
    for i=1:s
        fprintf('(%d,%d) ', i, v(i));
    end
    fprintf('}\n');
else
    for i=1:s+1
        topovi([v(1:i-1) n v(i:s)],n-1);
    end
end
```

Varijacije bez ponavljanja

101.
```
function varMaxEl(v,n,k)

s=size(v,2);
if k==0
    if mod(max(v),2)==1
        disp(v);
    end
else
    for i=1:s+1
        varMaxEl([v(1:i-1) n v(i:s)],n-1,k-1);
    end
    if n>k
        varMaxEl(v,n-1,k);
    end
end
```

102.
```
function varBP(v,n,k)

s=size(v,2);
if k==0
    i=1;
    l=0;
    while ~l && i<=s
        if v(i)==s  % s je jednako pocetnoj vrednosti
%parametra k
            l=1;
        end
        i=i+1;
    end
    if min(v)==1 && l
        disp(v);
    end
else
    for i=1:s+1
        varBP([v(1:i-1) n v(i:s)],n-1,k-1);
    end
    if n>k
        varBP(v,n-1,k);
    end
end
```

Kombinacije bez ponavljanja

103.
```
function komb(v,a,n,k)

s=size(v,2);
if k==0
  if sum(v)>s*s
      disp(v);
  end
else
  for i=a:n-k+1
      komb([v i],i+1,n,k-1);
  end
end
```

104.
```
function kombBezP(v,n,k)

if k==0
    b=0;
    for i=1:size(v,2)
        if mod(v(i),2)==0
```

```
            b=b+1;
        end
    end
    if b~=0 && mod(b,2)==0
        disp(v);
    end
 else
    if size(v,2)==0
        a=1;
    else
        a=v(end)+1;
    end
    for i=a:n-k+1
        kombBezP([v i],n,k-1);
    end
end
```

Kombinacije sa ponavljanjem

105.
```
function kSP(v,n,k)

s=size(v,2);
if k==0
    m=max(v);
    i=s;
    br=0;
    while i>0 && v(i)==m
        br=br+1;
        i=i-1;
    end
    if br>=s/2
        disp(v);
    end
else
    if s==0
        a=1;
    else
        a=v(end);
    end
    for i=a:n
        kSP([v i],n,k-1);
    end
end
```

106.
```
function kombSP(v,n,k)

s=size(v,2);
```

```
if k==0
    w=zeros(1,n);
    for j=1:s
        w(v(j))=w(v(j))+1;
    end
    if max(w)<=3
        disp(v);
    end
else
    if s==0
        a=1;
    else
        a=v(end);
    end
    for i=a:n
        kombSP([v i],n,k-1);
    end
end
```

Varijacije sa ponavljanjem

107.
```
function br=varP(v,n,k)

s=size(v,2);
if k==0
    l=1;
    i=1;
    br=0;
    while l && i<s
        if v(i)==v(i+1)
            l=0;
        end
        i=i+1;
    end
    if l
        br=1;
    end
else
    br=0;
    for i=1:n
        br=br+varP([v i], n, k-1);
    end
end
```

Razbijanje broja u zbir

108.
```
function razbijanja2(v,n,k)
```

```
if k==1
    v=[v n];
    l=1;
    for i=1:size(v,2)-1
      if abs(v(i)-v(i+1))<2
          l=0;
      end
    end
    if l
        disp(v);
    end
else
    for i=0:n
        razbijanja2([v i], n-i, k-1);
    end
end
```

109.
```
function razbijanja(v,n,k)

if k==1
    v=[v n];
    l=1;
    for i=1:size(v,2)-1
       for j=i+1:size(v,2)
           if v(i)==v(j)
               l=0;
           end
       end
    end
    if l
        disp(v);
    end
else
    for i=0:n
        razbijanja([v i], n-i, k-1);
    end
end
```

110.
```
function razbijanjaP(v,n,k)

if k==1
    v=[v n];
    disp(v);
else
    for i=1:n
        if mod(n,i)==0
            razbijanjaP([v i], n/i, k-1);
```

```
        end
    end
end
```

Statističke ocene

111.
```
function ver=z1(n)

br=0;
for i=1:n
    v=ceil(6*rand(1,3));
    tekb=0;
    for j=1:3
        if mod(v(j),2)==0
            tekb=tekb+1;
        end
    end
    if tekb>=2
        br=br+1;
    end
end
ver=br/n;
```

112.
```
function p=z2(n,k)

b=0;
for i=1:n
    v=ceil(6*rand(1,k));
    if sum(v)==2*k
        b=b+1;
    end
end
p=b/n;
```

113.
```
function p=z3(n,k)

b=0;
for i=1:n
    v=ceil(6*rand(1,k));
    bs=0;
    for j=1:k
        if v(j)==6
            bs=bs+1;
        end
    end
    if bs==k/3
        b=b+1;
```

```
    end
end
p=b/n;
```

114.
```
function p=z4(n,k,m)

b=0;
for i=1:n
    v=ceil(m*rand(1,k));
    l=1;
    j=1;
    while l && j<k
        if v(j)>v(j+1)
            l=0;
        end
        j=j+1;
    end
    if l
        b=b+1;
    end
end
p=b/n;
```

115.
```
function p=z5(n,k)

b=0;
for i=1:n
    v=floor(2*rand(1,k));
    l=1;
    j=1;
    while l && j<k
        if v(j)==v(j+1)
            l=0;
        end
        j=j+1;
    end
    if l
        b=b+1;
    end
end
p=b/n;
```

116.
```
function p=z6(n,k)

b=0;
for i=1:n
```

```
    v=ceil(6*rand(1,k));
    w=zeros(1,6);
    for j=1:k
        w(v(j))=w(v(j))+1;
    end
    el=0;
    for j=1:6
        if w(j)~=0
            el=el+1;
        end
    end
    if el<=3
        b=b+1;
    end
end
p=b/n;
```

117.
```
function p=z7(n,k)

b=0;
for i=1:n
    v=floor(37*rand(1,k));
    l=1;
    for j=1:k
        if v(j)==0 || sqrt(v(j))~=floor(sqrt(v(j)))
            l=0;
        end
    end
    if l
        b=b+1;
    end
end
p=b/n;
```