Zadatak 1.

Napišite funkcijsku *m*–datoteku *kut.m* koja sadrži jedino funkciju *kut* čije su ulazne varijable vektori $\vec{a}$ i $\vec{b}$ zapisani u obliku jednoretčanih matrica *A* i *B*. Funkcija treba ispisati mjeru kuta (u radijanima i u stupnjevima) kojega zatvaraju vektori $\vec{a}$ i $\vec{b}$. Što se dobiva za $\vec{a}=\vec{i}-2\cdot\vec{j}+3\cdot\vec{k}$ i $\vec{b}=7\cdot\vec{i}+5\cdot\vec{j}+\vec{k}$?

Zadatak 2.

Izračunajte zbroj svih elemenata matrice $A\in M_{4,6}(\mathbb{R})$ čiji su elementi definirani pravilom $a_{ij}=\left\lceil \log_2\left(i^j+j^i\right)\right\rceil$, za sve dopustive (i,j).

Zadatak 3.

Napišite funkcijsku *m*–datoteku *z3.m* koja sadrži jedino funkciju *z3* čije su ulazne varijable realna matrica *A*, te $a,b\in\mathbb{R}$ takvi da je $a\le b$. (Nije potrebno provjeravati valjanost uvjeta $a\le b$.) Funkcija treba ispisati ukupan broj svih elemenata matrice *A* koji se nalaze u segmentu $[a,b]$, te najmanji i najveći od njih. Što se dobiva za $A=\begin{bmatrix}2.7 & 2.71 & 2.72\\ 3.1 & 3.14 & 3.15\end{bmatrix}$, $a=e$ i $b=\pi$?

Zadatak 4.

Napišite funkcijsku *m*-datoteku *z4*.m koja sadrži jedino funkciju *z4* čija je jedina ulazna varijabla realna matrica *A*. Funkcija treba ispisati ukupan broj svih elemenata matrice *A* koji su potpuni kvadrati cijelih brojeva, te njihovu aritmetičku sredinu. Što se dobiva za $A=\begin{bmatrix}-1 & 0 & 1\\ 2 & 4 & 9\end{bmatrix}$?

Zadatak 5.

Niz Fibonaccijevih brojeva $(F_n)_{n\in\mathbb{N}}$ definiran je rekurzivno s $\begin{cases}F_1=F_2=1,\\ F_n=F_{n-1}+F_{n-2},\ \text{za } n\ge 3.\end{cases}$ Napišite funkcijsku datoteku *z5*.m čiji je koja sadrži jedino funkciju *z5* čiji je jedini argument $x\in\mathbb{R}$. Funkcija treba ispisati najmanji Fibonaccijev broj strogo veći od *x* i njegovu poziciju u Fibonaccijevu nizu. Što se dobiva za $x=\pi^2$?

Zadatak 6.

Napišite *m*-datoteku koja omogućuje upis realnih brojeva s tipkovnice u jednoretčanu matricu *A* sve dok se prvi put ne upiše strogo negativan realan broj. (Pretpostavimo da ne upisujemo slova ili neki drugi znak s tipkovnice.) Sve elemente dobivene matrice treba uzlazno sortirati (bez ispisa), pa ispisati koliko je među njima prirodnih brojeva. Što se dobiva ako se s tipkovnice unese niz brojeva 1, ln 2, log 5, $\log_2 512$, π, *e*, -1? (*Napomena*: 0 nije prirodan broj.)

Zadatak 7. *Harmonijski niz* definiran je pravilom $a_n = \frac{1}{n},\ \forall n \in \mathbb{N}$. Napišite funkcijsku datoteku *z*7.m koja sadrži jedino funkciju *z*7 čiji je jedini argument $x \in \mathbb{R}$. Funkcija treba vratiti najveći $m \in \mathbb{N}$ takav da je $\sum_{n=1}^{m} a_n \le x$. Što se dobiva za $x = 10$?

Zadatak 8.

Za svaki $n \in \mathbb{N}$ označimo s b_n zbroj svih znamenaka u dekadskom zapisu broja *n*. Izračunajte zbroj prvih 100 članova niza $(b_n)_{n\in\mathbb{N}}$.

Rezultati zadataka:

1. 1.5708; 90.
2. 131.
3. 3; 2.72; 3.14.
4. 4; 3.5.
5. 13; 7.
6. A = [0.6931, 0.6990, 1.0000, 2.7183, 3.1416, 9.0000]. U matrici A ima ukupno 2 prirodna broja.
7. 12 366.
8. 901.

Komentari/objašnjenja programskih kodova

kut.m

U ovom zadatku samostalno stvaramo funkciju u MATLAB-u. Dosad smo koristili ugrađene funkcije, te ih pozivali navođenjem njihova imena i svih potrebnih ulaznih argumenata. No, obično smo i izlazne varijable označavali posebnim imenom (npr. `rezultat`, `rjesenje` i sl.) Dakle, svaka funkcija je zapravo bila jednoznačno određena analogno kao i funkcije koje smo sretali u matematici: imenom izlazne varijable, imenom funkcije i popisom svih ulaznih varijabli.

Pogledajmo kako je stvorena funkcija `kut`. Prvi redak koda započet je rezerviranom riječju `function`. Ona označava da će `m`-datoteka koju ćemo stvoriti biti MATLAB-ova funkcija. Nakon te riječi slijedi zadavanje imena izlazne varijable (`y`), imena funkcije (`kut`) i popisa svih ulaznih varijabli (`a`, `b`). U ovom slučaju imamo dvije ulazne varijable: to će biti dvije matrice koje reprezentiraju vektore. Radi jednostavnosti, pretpostavili smo da su matrice jednoretčane.

- *Pitanja za razmišljanje*: Jesmo li mogli pretpostaviti da su matrice jednostupčane ili da je jedna od matrica jednoretčana, a druga jednostupčana? Ako jesmo, kako bi u tom slučaju izgledao analitički račun (tj. račun „ručno na papiru“ uz pomoć olovke i kalkulatora)?

Kut između dvaju vektora $\vec{a}$ i $\vec{b}$ je $\varphi \in [0, \pi]$ definiran pravilom:

$$\varphi = \arccos\left(\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|} \right).$$

Da bismo mogli odrediti taj kut, moramo odrediti svaki element u okrugloj zagradi. Počnimo od brojnika. Skalarni umnožak vektora (zapisanih kao jednoretčane matrice) određuje ugrađena funkcija `dot`. Njezini su argumenti dvije jednoretčane matrice čiji skalarni umnožak treba odrediti.

Prijeđimo na nazivnik. U nazivniku se nalazi umnožak duljina vektora $\vec{a}$ i $\vec{b}$. Duljinu nekoga vektora (zapisanoga u obliku jednoretčane matrice) određuje ugrađena funkcija `norm`. Ona ima jedinstveni argument, a to je, prirodno, vektor čiju duljinu treba odrediti.

Nakon što smo, koristeći ugrađene funkcije `dot` i `norm`, odredili svaki element brojnika, odnosno nazivnika gornjega izraza, preostaje primijeniti ugrađene funkcije `acos` i `acosd`. Uzimajući ranije određene vrijednosti skalarnoga umnoška i duljina vektora, prva će funkcija će vratiti kut između zadanih vektora iskazan u radijanima, dok će druga vratiti taj kut iskazan u stupnjevima.

Oprez: Slika funkcije arccos je segment $[0, \pi]$, pa je prirodna mjera traženoga kuta radijan. Međutim, imajući u vidu geometrijsku interpretaciju, mjera može biti iskazana i u stupnjevima.

U predzadnjem retku koda ispisujemo dobivene rezultate u obliku rečenice. U tu svrhu koristimo ugrađenu funkciju `fprintf`. Njezina osnovna svrha je ispis podataka u tekstualnu datoteku, ali može se iskoristiti i za ispis rezultata u komandni prozor. Opcija `\n` omogućuje prelazak u novi redak, a time i bolju preglednost ispisanoga rezultata. Za ispis vrijednosti varijabli `y1` i `y2` koristili smo opciju `%d` koja se može koristiti i više puta u okviru ove funkcije. Prvo navođenje %d rezultirat će ispisom vrijednosti varijable `y1`, a drugo navođenje `%d` ispisom vrijednosti varijable y2. Koje vrijednosti varijabli treba ispisati i kojim redoslijedom, navodimo nakon što zatvorimo znak jednostrukoga navodnika. Iza toga znaka moramo staviti zarez, pa potom navesti redoslijed vrijednosti koje želimo ispisati. Ako tih vrijednosti ima više – kao u ovom slučaju – međusobno ih odvajamo znakovima zareza.

Oprez: Nakon što zatvorimo znak jednostrukoga navodnika, moramo navesti toliko vrijednosti varijabli koliko smo puta naveli opciju `%d` unutar znakova jednostrukih navodnika. Ako unutar znakova jednostrukoga navodnika npr. četiri puta napišemo `%d`, a nakon zatvaranja znaka jednostrukoga navodnika navedemo samo dvije vrijednosti, MATLAB će javiti grešku.

Zaključno uočimo da **svaka** funkcija u MATLAB-u **mora** završiti s ugrađenom funkcijom `end`. Ona označava da je funkcija završena i omogućuje daljnje izvršavanje naredbi u programu ili radnom prozoru. Dobivenu funkciju u radnom prozoru pozivamo potpuno analogno kao i sve ostale MATLAB-ove ugrađene funkcije.

z2.m

U ovom je zadatku strategija sljedeća: najprije odredimo svaki element matrice A koristeći pravilo zadavanja elemenata matrice navedeno u zadatku, a potom zbrojimo sve te elemente. Zadavanje matrice A možemo izvršiti na dva načina: zasebnim izračunom i unosom svakoga pojedinoga elementa, te pomoću *for*-petlje. Prvi je način mukotrpan i prespor već za matrice reda 2, pa ćemo primijeniti drugi način.

For-petlja je programska struktura koja, grubo i pojednostavljeno, izvršava određeni skup naredbi onoliko puta koliko vrijednosti može poprimiti njezin brojač. Svaka takva petlja ima sljedeći oblik:

```
for varijabla=pocetna:korak:krajnja
…
end
```

Varijabla je varijabla koja će u prvom izvršavanju petlje poprimiti vrijednost *pocetna*. Nakon što se izvrši cjelokupni skup naredbi unutar petlje, vrijednost varijable *varijabla* povećat će se za vrijednost varijable *korak*. Ako dobivena vrijednost prelazi vrijednost varijable *krajnja*, cjelokupni skup naredbi unutar petlje više se neće izvršiti. U suprotnom, te će se naredbe izvršiti još jednom, pa će se vrijednost varijable *varijabla* potom ponovno povećati za vrijednost varijable *korak*. Na taj način petlja će se izvršavati sve dok vrijednost varijable *varijabla* ne premaši vrijednost varijable *krajnja*.

Napomenimo da se varijabla *korak* smije izostaviti ako i samo ako je vrijednost te varijable jednaka 1. U suprotnom, tu varijablu ne smijemo izostaviti.

- *Pitanje za razmišljanje*: Pretpostavimo da su *pocetna*, *korak* i *krajnja* unaprijed zadane vrijednosti takve da je *pocetna* ≤ *krajnja*. Koliko će se točno puta izvršiti gornja petlja?

Unutar jedne petlje, kao dio skupa naredbi, mogu se pojavljivati različite strukture, pa čak i druge petlje. Pritom moramo paziti da **nigdje** u dotičnim strukturama **ne koristimo** varijablu *varijabla*. Ona je deklarirana u prvom retku petlje, poprima točno određene vrijednosti u svakom pojedinom koraku i te vrijednosti ne možemo mijenjati drugim naredbama unutar petlje.

Pogledajmo na koji smo način definirali matricu A. U prvom retku definirali smo *for*-petlju čija je varijabla označena s `i`. Njezina početna vrijednost i korak jednaki su 1, pa, prema ranijoj napomeni, korak zbog toga ne mora nužno biti naveden. Krajnja

vrijednost varijable `i` jednaka je 4. Što će se dogoditi izvršavanjem ove petlje? Ona će se izvršiti ukupno četiri puta. U prvom izvršavanju vrijednost varijable `i` bit će jednaka 1, u drugom 2, u trećem 3, a u posljednjem, četvrtom, izvršavanju 4. Što označava varijabla `i`? Kao i u analognoj situaciji u *Matematici* 1, ona označava broj retka matrice. Kratko i grubo kažemo: „Prva petlja prolazi po svim retcima matrice A.“

Unutar prve petlje smjestili smo drugu petlju. Njezina varijabla je označena s `j`. Njezina početna vrijednost i korak jednaki su 1, pa je, analogno kao i maloprije, korak zbog toga izostavljen. Krajnja vrijednost varijable `j` jednaka je 6. Što će se dogoditi izvršavanjem ove petlje? Ona će se izvršiti ukupno $4 \cdot 6 = 24$ puta. U prvom izvršavanju vrijednost varijable `j` bit će 1, u drugom 2, u trećem 3, u četvrtom 4, u petom 5, u šestom 6, no, u sedmom izvršavanju vrijednost varijable `j` bit će ponovno jednaka 1. Zašto? Naime, kad je $i = 1$ druga petlja će se izvršiti ukupno šest puta. Ona će potom „stati“ i program će se vratiti na prvu petlju. Vrijednost varijable i povećat će se na 2, pa će se i za tu vrijednost druga petlja izvršiti još šest puta. Izvršavanje obiju petlji prestat će tek kad se unutrašnja petlja (po varijabli j) izvrši šesti put za vrijednost $i = 4$.

U trećem retku navedeno je pravilo zadavanja elemenata matrice A. Uočimo da umjesto standardne matematičke oznake a_{ij} ovdje piše `A(i,j)`. To i nije čudno jer, kako otprije znamo, element matrice A dohvaćamo tako da najprije napišemo ime matrice, otvorimo okruglu zagradu, napišemo broj retka u kojemu se nalazi taj element, potom napišemo broj stupca u kojemu se nalazi taj element (**ne smijemo** zamijeniti redoslijed, pa najprije napisati broj stupca, a potom broj retka!) i potom zatvorimo zagradu. Dakle, s `A(i,j)` definirat ćemo element matrice A koji se nalazi na presjeku i-toga retka i j-toga stupca. Taj element definiran je formulom u zadatku, pa tu formulu upisujemo s desne strane znaka jednakosti. Ovdje znak = treba interpretirati kao „dobiva vrijednost“.

Što zapravo radimo ovim petljama? Grubo rečeno, popunjavamo matricu A po retcima. Potpuno je svejedno popunjavamo li je po retcima ili stupcima, pa u ovakvim slučajevima možemo samostalno birati način popunjavanja matrice.

- *Pitanje za razmišljanje*: Preradite kod tako da matrica A bude popunjena po stupcima.

Nakon što s ugrađenom funkcijom `end` zatvorimo **svaku petlju posebno**, pa izvršavanjem obiju petlji popunimo matricu A, preostaje zbrojiti sve njezine elemente. U tu svrhu koristimo funkciju `sum`. Pritom moramo dodati i opciju `'all'` jer matrica A nije ni jednoretčana, ni jednostupčana.

z3.m

U ovom zadatku funkcija ima tri ulazne varijable različitih tipova. A je matrica, dok su $a,b\in\mathbb{R}$. Napomenimo da ćemo prilikom pozivanja funkcije iz radnoga prozora morati paziti na redoslijed navođenja ulaznih varijabli. Dakle, najprije ćemo morati navesti matricu, pa donju granicu segmenta i, naposljetku, gornju granicu segmenta.

Strategija rješavanja problema je sljedeća. „Proći“ ćemo „kroz“ matricu A, tj. za svaki njezin element utvrdit ćemo pripada li segmentu $[a,b]$. Ako pripada, prepisat ćemo ga u novu matricu B. Ta će matrica biti jednoretčana jer ne znamo postoji li uopće element matrice A koji pripada segmentu $[a,\ b]$ i, ako postoji, koliko je takvih elemenata. Zbog toga je najjednostavnije prepisati sve takve elemente u jednoretčanu matricu.

Međutim, ne možemo unaprijed znati postoji li uopće barem jedan element matrice A koji pripada segmentu $[a,b]$. Zbog toga vrijednost brojača tih elemenata (varijablu k) u drugomu retku koda inicijaliziramo na nulu. Ako ne postoji nijedan element matrice A koji pripada segmentu $[a,b]$, vrijednost varijable k ostat će nepromijenjena, tj. jednaka nuli. Nadalje, istu varijablu iskoristit ćemo i u još jednu svrhu: matrica u koju ćemo „prepisati“ sve elemente matrice A koji pripadaju segmentu $[a,b]$bit će jednoretčana, pa će varijabla k ujedno označavati i broj stupaca te matrice. Jasno je da je za *svaku* jednoretčanu matricu ukupan broj njezinih elemenata identički jednak broju njezinih stupaca.

Pri „prolasku kroz“ matricu željeli bismo primijeniti strategiju iz zadatka 2. Međutim, tu se javlja problem. Mi na početku ne znamo koliko redaka, odnosno stupaca ima matrica A. Zadatak kaže da je A bilo kakva realna matrica, ali ne kaže ništa o broju njezinih redaka/stupaca. Zbog toga ćemo u *for*-petljama koristiti ugrađenu funkciju `size`. Ta funkcija ima nekoliko značenja, a ovdje nam je posebno korisno sljedeće: ako je argument funkcije matrica, onda funkcija može vratiti broj redaka ili broj stupaca te matrice. Da bismo to postigli, uz matricu A kao argument funkcije `size` dodajemo broj 1 ako želimo odrediti broj redaka matrice A, odnosno broj 2 ako želimo odrediti broj stupaca matrice A.

Dakle, u trećem i četvrtom retku koda pokrećemo dvije *for*-petlje kojima je svrha „proći kroz“ cijelu matricu A i ispitati svaki njezin element. U petom retku definiramo „ispitivanje“. U MATLAB-u ne postoji ugrađena logička funkcija koja bi provjeravala pripada li neki broj nekom intervalu, pa „ispitivanje“ provodimo prema definiciji segmenta:

$$\left(x \in [a,b]\right) \Leftrightarrow \left(a \le x \le b\right).$$

Zbog toga za svaki element matrice A provjeravamo vrijede li **istovremeno** relacije gornje relacije. Pritom **ne možemo** jednostavno napisati `a<=A(i,j)<=b` jer MATLAB odjednom može provjeriti valjanost samo jedne relacije. To je razlog zbog kojega moramo koristiti logički operator AND kojega zapisujemo kao &&. Što će MATLAB napraviti kad „naleti“ na taj operator? Provjerit će valjanost logičkoga uvjeta s lijeve strane znakova &&, potom će provjeriti valjanost logičkoga uvjeta s desne strane znakova && i na temelju dobivenih istinitosti utvrditi je li istinit logički uvjet dobiven primjenom operatora AND. Podsjetimo: $A \cdot B$ je istinito ako i samo ako su istiniti i A i B, što u ovom slučaju i želimo.

Time je zapravo obavljen najvažniji dio posla. Ako je logički uvjet u petom retku ispunjen, onda se vrijednost brojača k, a time i ukupan broj stupaca matrice B, poveća za 1. Time se odmah omogućuje da MATLAB pripadni element matrice A prepiše na poziciju $b_{1,k}$ (tj. na presjek prvoga retka i k-toga stupca). Ako taj logički uvjet nije ispunjen, vrijednost brojača k ostat će nepromijenjena.

Uočimo da smo u okviru *for*-petlji primijenili strukturu *if – else*. Ta struktura općenito kaže: ako vrijedi neki uvjet, izvrši jedan niz naredbi, a ako ne vrijedi, izvrši drugi niz naredbi. Ovdje nismo imali drugi niz naredbi – ako neki element matrice ne pripada zadanom segmentu, jednostavno ga „pustimo na miru“ i s njim ne radimo ništa. Neovisno o postojanju dijela *else*, **svaka** struktura koja počinje uvjetnom naredbom `if` mora završiti s `end` (analogno kao i *for*-petlja).

Nakon što smo za svaki element matrice A provjerili valjanost logičkoga uvjeta u petom retku koda i prepisali sve elemente matrice A koji zadovoljavaju taj uvjet u matricu B, preostaje ispisati ukupno tri podatka.

Prvi podatak je ukupan broj elemenata matrice B. Tu moramo biti oprezni jer, kako smo rekli, unaprijed ne možemo znati hoće li ta matrica uopće postojati. Zbog toga ćemo koristiti još jednu *if – else* strukturu. Provjeravajući vrijednost brojača k najprije ćemo utvrditi postoji li matrica B. Ako je $k = 0$, onda nismo našli nijedan element matrice A koji pripada segmentu $[a,b]$, pa u 12. retku koda ispisujemo odgovarajuću poruku koristeći funkciju `fprintf`. U suprotnom, vrijednost brojača k je jednaka ili veća od 1, pa u retcima 14. – 16. - ponovno koristeći funkciju `fprintf` – ispisujemo zaključnu vrijednost toga brojača, odnosno ukupan broj svih elemenata matrice A koji pripadaju segmentu $[a,b]$.

Najmanji element matrice B odredimo primjenjujući ugrađenu funkciju `min`, a najveći primjenjujući ugrađenu funkciju `max`. Pri pozivima obiju funkcija bitno je da je B **jednoretčana** matrica. Da je B bila matrica tipa (m,n) , gdje su $m,n\in\mathbb{N}:m,n\geq 2$, za određivanje najmanjega, odnosno najvećega elementa matrice B morali bismo napisati `min(B,[],'all')` ili `min(min(B))`, odnosno `max(B,[],'all')` ili `max(max(B))`.

Zaključno, primijetimo da u kodu nigdje nismo koristili izlaznu varijablu y deklariranu u prvom retku koda. MATLAB će nas porukom upozoriti na to prilikom poziva funkcije u glavnom programu. Umjesto y mogli smo upisati `[x y z]`, pa u kodu deklarirati x kao ukupan broj elemenata matrice B, y kao najmanji element te matrice, a z kao najveći element te matrice. Međutim, time bismo izgubili na jasnoći funkcije jer bez otvaranja koda ne bismo znali što nam označava koja od tih triju varijabli. Dakle, nauštrb jasnoće funkcije „žrtvovali“ smo deklariranje nepotrebne izlazne varijable.

- *Pitanje za razmišljanje*: Preradite kod tako da „prolaz kroz matricu“ A bude po stupcima, a ne po retcima.

- *Pitanje za razmišljanje*: Kako bi izgledao logički uvjet u petom retku koda da je umjesto segmenta $[a,b]$ bio zadan interval $\langle a,b\rangle$? Odgovorite na analogno pitanje za interval $\langle a,b]$, odnosno interval $[a,b\rangle$.

- *Pitanje za razmišljanje*: Je li moguće međusobno zamijeniti 6. i 7. redak koda tako da se u matricu B najprije „prepiše“ element matrice A koji pripada segmentu $[a,b]$, a potom poveća vrijednost brojača k? Ako jest, preradite kod tako da provedete navedenu zamjenu i (ponovno) dobijete točan rezultat. Ako nije, objasnite svoj odgovor.

z4.m

U ovom je kodu zapravo jedini problem kako ispitati je li *bilo koji* realan broj kvadrat nekoga cijeloga broja. Problem nema smisla ako je taj realan broj strogo negativan. Zbog toga ćemo za svaki element matrice A morati provjeriti je li nenegativan ili nije. Ako je neki element strogo negativan, pustit ćemo ga na miru i ništa nećemo raditi s njim. Međutim, ako je taj element (označimo ga s x) nenegativan, onda će on biti potpun kvadrat nekoga cijeloga broja ako i samo ako vrijedi barem jedna od jednakosti:

$$\sqrt{x} = \left\lfloor \sqrt{x} \right\rfloor \text{ ili } \sqrt{x} = \left\lceil \sqrt{x} \right\rceil .$$

Naime, u vježbi 2. smo rekli da su funkcije „najmanje cijelo“ i „najveće cijelo“ identitete na skupu cijelih brojeva, tj. one sve cijele brojeve „puštaju na miru“, a sve ostale zaokružuju na prvi veći/manji cijeli broj. Zbog toga ćemo u rješavanju ovoga zadatka iskoristiti upravo jednu od njih. Dakle, za svaki element matrice A provjerit ćemo vrijede li **istovremeno** relacije $a_{ij} \geq 0$ i $\sqrt{a_{ij}} = \left\lceil \sqrt{a_{ij}} \right\rceil$. Ako vrijede, taj element ćemo prepisati u (jednoretčanu) matricu B jer je riječ o potpunom kvadratu cijeloga broja. Ako ne vrijedi barem jedna od njih, onda ćemo taj element pustiti na miru i zanemariti u daljnjim retcima koda.

Analogno kao i u prethodnom zadatku, ne možemo unaprijed znati postoji li ijedan element matrice A koji je potpuni kvadrat cijeloga broja. Zbog toga ćemo u drugomu retku koda inicijalizirati brojač takvih elemenata (ponovno označen s k) i postaviti njegovu vrijednost na nulu. Ako ne postoji nijedan element matrice A koji je potpuni kvadrat nekoga cijeloga broja, vrijednost brojača k ostat će nepromijenjena, tj. jednaka nuli. Nadalje, ponovno analogno kao i u prethodnom zadatku, sve elemente matrice A koji su potpuni kvadrati cijelih brojeva prepisat ćemo u *jednoretčanu* matricu B, pa će varijabla k ujedno biti i ukupan broj stupaca matrice B.

U retcima 3. – 10. implementiramo upravo opisani algoritam. „Prolazimo kroz matricu“ A redak po redak, ispitujemo svaki element te matrice i elemente koji su potpuni kvadrati prepisujemo u jednoretčanu matricu B. „Prepisivanje“ radimo tako da povećamo vrijednost brojača k za jedan, čime „otvaramo“ novu praznu poziciju u jednoretčanoj matrici B u koju ćemo potom prepisati ispitani element koji je potpun kvadrat nekoga cijeloga broja.

Naposljetku, susrećemo se s istim problemom kao i u prethodnom zadatku: što ako u matricu B nije prepisan nijedan element matrice A, tj. ako nijedan element matrice A nije potpun kvadrat? Rješenje toga problema je potpuno analogno onome u

prethodnom zadatku, pa ga ovdje nećemo posebno komentirati. Reći ćemo tek da smo aritmetičku sredinu mogli odrediti pomoću opcije `mean(B)` jer je B jednoretčana matrica.

- *Pitanja za razmišljanje*: Kako bi trebalo preraditi kod da se odredi ukupan broj svih elemenata matrice A koji su potpuni kubovi nekih cijelih brojeva? Kako bi trebalo preraditi kod da se odredi ukupan broj svih elemenata matrice A koji su potencije broja 2? Općenito, ako zadamo prirodan broj $k > 1$, kako treba preraditi kod da se odredi ukupan broj svih elemenata matrice A koji su potencije broja k? (*Uputa*: U svim odgovorima koristite odgovarajuću logaritamsku funkciju.)

- *Pitanja za razmišljanje*: Kako bi trebalo preraditi kod da se odredi ukupan broj svih elemenata matrice A koji su parni prirodni brojevi? Kako bi trebalo preraditi kod da se odredi ukupan broj elemenata matrice A koji su neparni prirodni brojevi? Kako bi trebalo preraditi kod da se utvrdi ukupan broj elemenata matrice A koji su djeljivi unaprijed zadanim prirodnim brojem n? (Ne treba provjeravati je li $n \in \mathbb{N}$.) Odgovorite na analogno pitanje za broj elemenata matrice A koji nisu djeljivi s n. (*Uputa*: U svim odgovorima koristite ugrađenu funkciju `mod`.)

- *Pitanje za razmišljanje*: Je li moguće zamijeniti redoslijed 6. i 7. retka koda tako da u matricu B najprije bude prepisan element matrice A koji je potpun kvadrat nekoga cijeloga broja, a potom vrijednost brojača bude povećana za jedan? Ako jest, preradite kod tako da provedete navedenu zamjenu i dobijete (ponovno) točan rezultat. Ako nije, objasnite svoj odgovor.

z5.m

Najprije primijetimo da je Fibonaccijev niz neopadajući – štoviše, podniz kojega dobijemo kad izbacimo prvi ili drugi član Fibonaccijeva niza je strogo rastući. Elemente toga niza pohranit ćemo u jednoretčanu matricu *F*. Počevši od elementa na presjeku prvoga retka i trećega stupca, a u skladu s rekurzivnom relacijom kojom je definiran niz, svaki element matrice *F* izračunat ćemo kao zbroj dvaju neposredno prethodnih elemenata te matrice.

Razradimo algoritam pomoću kojega ćemo riješiti problem. Prema uvjetu zadatka, x je **bilo koji** realan broj. Ako je $x<1$, onda zadatak ima dva rješenja: to su prva dva člana niza koja su međusobno jednaka i jednaka 1. U suprotnom, zadatak ima jedinstveno rješenje.

Slijedom navedenoga, najprije provjerimo je li $x<1$. Ako jest, onda MATLAB treba ispisati poruku da su rješenja zadatka F_1 i F_2. Ako nije, onda treba popunjavati matricu *F* sljedećim članovima Fibonaccijeva niza i za svakoga novoga člana provjeriti je li strogo veći od *x*. Ako jest, treba zaustaviti popunjavanje matrice *F*. U suprotnom, treba nastaviti popunjavanje te matrice. Dakle, popunjavanje matrice *F* se odvija **sve dok** su njezini elementi jednaki ili manji od *x*, a prestaje kad taj uvjet više ne bude ispunjen.

Kako izvesti popunjavanje matrice *F*? Uvedimo varijablu *k* koja će nam označavati poziciju posljednjega upisanoga elementa. (Podsjetimo: matrica *F* je jednoretčana, pa je svaki njezin element jednoznačno određen zadavanjem broja stupca u kojemu se nalazi.) Koristimo *while*-strukturu koja, pojednostavljeno, kaže: izvršavajte neki skup naredbi sve dok je zadovoljen logički uvjet naveden iza riječi `while`. S obzirom da smo slučaj $x<1$ eliminirali koristeći *if*-strukturu u retcima 4. i 5., izvršenje redaka 6. – 12. dogodit će se ako i samo ako je $x\geq 1$. Dakle, „smjestimo“ se na poziciju 2 (tj. u drugi stupac), pa sve dok ne premašimo *x* najprije se prebacimo na prvu sljedeću poziciju (povećavajući vrijednost brojača *k* za jedan), te potom na tu (praznu) poziciju upišimo element izračunat pomoću zadane rekurzivne relacije. Na taj način ćemo u svakom „prolazu“ kroz *while*-strukturu znati poziciju posljednjega upisanoga elementa, a samim time i taj element.

Kad se skup naredbi u *while*-strukturi prestane izvršavati zbog prestanka valjanosti logičkoga uvjeta, vrijednost varijable *k* bit će jednaka broju stupca u kojemu se nalazi posljednji upisani element. No, taj element je upravo traženi element jer se skup naredbi u *while*-strukturi prestao izvršavati čim je u matricu *F* upisan prvi element strogo veći od *x*. Zbog toga su taj element i njegova pozicija rješenja zadatka.

- *Pitanja za razmišljanje*: Možemo li u 7. retku koda inicijalizirati $k = 3$? Ako možemo, preradite kod tako da ispisuje ispravne rezultate. Ako ne možemo, objasnite svoj odgovor.

- *Pitanja za razmišljanje*: Preradite kod tako da ispisuje:

- najmanji Fibonaccijev broj jednak ili veći od x i njegovu poziciju u nizu;
- najveći Fibonaccijev broj jednak ili manji od x i njegovu poziciju u nizu;
- najveći Fibonaccijev broj strogo manji od x i njegovu poziciju u nizu.

  Ako traženi Fibonaccijev broj ne postoji, funkcija treba ispisati odgovarajuću poruku.

z6.m

U ovom zadatku pokazat ćemo kako s tipkovnice unositi elemente u jednoretčanu matricu, te uzlazno sortirati elemente dobivene matrice. U prvom retku deklariramo varijablu i čija će vrijednost biti jednaka poziciji (zapravo, broju stupca) posljednjega upisanoga elementa u matrici A. U komentarima prethodnih zadataka analizirali smo slučaj u kojemu matrica A ne sadrži nijedan element, pa zbog toga početnu vrijednost varijable i treba postaviti na nulu.

Prelazimo na unos brojeva. Taj unos započinje tako da korisniku najprije ispišemo poruku „Upišite prvi broj:“. To činimo koristeći ugrađenu funkciju `input` pomoću koje omogućujemo unos prvoga elementa matrice i taj element pohranimo u varijablu `x`. Daljnji postupak je jednostavan: provjerimo je li x nenegativan realan broj, pa, ako jest, pohranimo ga na prvu poziciju u matrici A i omogućimo unos sljedećega realnoga broja. (Prema pretpostavci, unos slova i drugih znakova s tipkovnice nije moguć.) Postupak ponavljamo sve dok prvi put ne upišemo strogo negativan realan broj. Primijetimo da će taj postupak funkcionirati i za brojeve poput π, e, $\ln 2$ i dr.

Nakon što smo upisali sve željene elemente matrice A, nastavljamo sa zadatkom. Budući da unaprijed ne znamo postoji li uopće matrica A, moramo koristiti `if-else` strukturu kako bismo mogli razlikovati dva slučaja. Ako je $i > 0$, to znači da je u matricu A upisan barem jedan element, pa možemo provesti sve daljnje zahtjeve.

Jednoretčanu matricu uzlazno sortiramo koristeći ugrađenu funkciju `sort`. Pri ulaznom sortiranju svih elemenata matrice A dovoljno je napisati `sort(A)`, dok bismo pri silaznom sortiranju tih elemenata morali napisati `sort(A,'descend')`.

Potpuno analogno kao i u rješenju zadatka 4., provjeru je li neki element matrice A prirodan broj obavljamo koristeći funkciju „najmanje cijelo“. Međutim, moguće je da neki element matrice A bude jednak nuli, a nula nije prirodan broj. Zbog toga osim uvjeta `A(j)==ceil(A(j)`, moramo postaviti i uvjet `A(j)>0`. Ta dva uvjeta povezana logičkim veznikom AND dat će ispravan kriterij za provjeru pripadnosti svakoga elementa matrice A skupu prirodnih brojeva. Napomenimo da smo, umjesto funkcije `ceil`, mogli koristiti i funkciju „najveće cijelo“, odnosno `floor`.

Oprez: Varijablu koja označava poziciju elementa za kojega provjeravamo je li prirodan broj **nismo** mogli označiti s i jer smo tu varijablu već iskoristili prilikom definiranja *for*-petlje. Zbog toga je ta „pozicijska“ varijabla označena s j.

Primijetimo da smo u 11. retku koristili i ugrađenu funkciju `length`. Ako je A jednoretčana matrica, onda `length(A)` vraća ukupan broj stupaca matrice A. Naravno, umjesto `length(A)` mogli smo koristiti i funkciju `size` u obliku `size(A,2)`.

U retcima 8. – 15., dakle, provjerili smo postoji li uopće matrica A, te, ako postoji, uzlazno smo sortirali sve njezine elemente, ispisali dobiveni rezultat (uočite da je u 9. retku matrica dobivena uzlaznim sortiranjem svih elemenata matrice A također označena s A, što je potpuno korektno) i prebrojali koliko ukupno prirodnih brojeva ima u matrici A (sortiranoj ili nesortiranoj, svejedno je). U 17. retku naveli smo poruku koja će biti ispisana ako i samo ako nijedan element matrice A nije prirodan broj, dok smo u 22. retku naveli poruku koja će biti ispisana ako i samo ako je prvi učitani broj strogo negativan. Time smo obuhvatili sve moguće slučajeve i dovršili rješenje zadatka.

- *Pitanja za razmišljanje*: Je li upisane brojeve moguće zapisati u obliku jednostupčane matrice A? Ako jest, preradite kod tako da ispisuje ispravne rezultate. Ako nije, obrazložite svoj odgovor.

- *Pitanja za razmišljanje*: Kako bi glasio 14. redak koda da smo trebali prebrojati sve elemente matrice A koji nisu prirodni brojevi? Kako bi glasio taj redak da smo trebali prebrojati sve elemente matrice A koji su jednaki ili veći od 10? Kako bi glasio taj redak da smo trebali prebrojati sve elemente matrice A koji su potpuni kvadrati nekih cijelih brojeva? U svakom od tih slučajeva preradite dobiveni kod tako da ispisuje ispravne rezultate.

- *Pitanje za razmišljanje*: Preinačite kod tako da dodatno svi elementi matrice A koji nisu prirodni brojevi budu izbrisani, da svi elementi matrice A koji su parni prirodni brojevi budu podijeljeni sa 2 i da svaki od preostalih elemenata matrice A bude zamijenjen prirodnim brojem koji je jednak ili manji od drugoga korijena dotičnoga elementa.

z7.m

I u ovom zadatku moramo razmotriti dva slučaja. Prvi i jednostavniji je slučaj $x<1$. U tom slučaju zadatak nema rješenja Naime, harmonijski niz je strogo padajući niz čiji je prvi član jednak 1, pa je, za svaki $m\in\mathbb{N}$, zbroj prvih m članova toga niza sigurno jednak ili veći od 1. Dakle, potrebno je ispisati poruku da traženi broj ne postoji, što je učinjeno u trećemu retku koda.

Složeniji slučaj je $x\geq 1$. Njega razmatramo u retcima 5. – 11. Da bismo razjasnili daljnje korake, ispišimo prvih nekoliko članova harmonijskoga niza: $1,\frac{1}{2},\frac{1}{3},\frac{1}{4},...$ i riješimo problem za relativno male strogo pozitivne vrijednosti x:

- Za $x=1$ funkcija treba ispisati 1 jer treba uzeti samo prvi član niza.
- Za $x=1.4$ funkcija također treba ispisati 1 jer treba uzeti samo prvi član niza.
- Za $x=1.5$ funkcija treba ispisati 2 jer je $1+\frac{1}{2}=\frac{3}{2}=1.5$

Razradimo strategiju rješavanja za $x\geq 1$. Deklariramo varijablu n i postavimo njezinu početnu vrijednost na nulu. Potom deklariramo varijablu *zbroj* i također inicijaliziramo njezinu početnu vrijednost na 0. Ona će u svakom koraku označavati zbroj točno n članova harmonijskoga niza. Na početku nismo uzeli nijedan član niza, pa su zbog toga i vrijednost varijable n i vrijednost varijable *zbroj* jednake nuli.

Članove harmonijskoga niza zbrajamo sve dok ne premašimo x i to je implementirano u retcima 7. – 10. U svakom koraku najprije povećamo vrijednost varijable n za 1 (tako „uzimamo“ n-ti član harmonijskoga niza), a potom povećavamo vrijednost varijable *zbroj* za „uzeti“ n-ti član (definiran pravilom $a_n=\frac{1}{n}$). Dakle, nakon prvoga „prolaza“ kroz *while*-strukturu vrijednost varijable n bit će jednaka 1, dok će vrijednost varijable *zbroj* biti jednaka $0+\frac{1}{1}=0+1=1.$ Nakon drugoga „prolaza“ kroz istu strukturu, vrijednost varijable n bit će jednaka 2, dok će vrijednost varijable *zbroj* biti jednaka $1+\frac{1}{2}=\frac{3}{2}$. Nakon trećega „prolaza“ kroz istu strukturu, vrijednost varijable n bit će jednaka 3, dok će vrijednost varijable *zbroj* biti jednaka $\frac{3}{2}+\frac{1}{3}=\frac{11}{6}$ itd.

11. redak koda je najzbunjujućiji dio cijeloga rješenja. Zašto je rješenje zadatka za jedan manje od posljednje vrijednosti varijable n nastale posljednjim izvršavanjem

naredbi koje tvore *while*-strukturu? Razjasnimo ovaj prividni paradoks na primjeru $x=1.$

Već smo komentirali da, nakon prvoga prolaza kroz dotičnu strukturu, varijable n i *zbroj* imaju međusobno jednake vrijednosti (i te su vrijednosti jednake 1). Sada se vraćamo na logički uvjet zadan u 7. retku. Vrijednost varijable *zbroj* je 1 i za tu vrijednost vrijedi $1 \le 1 = x.$ Zbog toga drugi put prolazimo kroz *while*-strukturu. Rekli smo da, nakon drugoga prolaza kroz dotičnu strukturu, varijable n i *zbroj* imaju vrijednosti redom 2 i $\frac{3}{2}$. Ponovno se vraćamo na logički uvjet zadan u 7. retku. Vrijednost varijable *zbroj* je $\frac{3}{2}$ i za tu vrijednost **ne** vrijedi logički uvjet $\frac{3}{2} \le 1 = x.$ Zbog toga su naši prolazi kroz *while*-strukturu završeni, pa prelazimo na izvršavanje naredbi poslije te strukture.

Vidimo da smo kroz *while*-strukturu prošli točno jednom previše. Drugim riječima, trebalo je samo jednom proći kroz tu strukturu. Posljednja *pohranjena* vrijednost varijable n (nakon drugoga prolaza kroz *while*-strukturu) jednaka je 2. Da bismo dobili ispravno rješenje, tj. zaključili da treba samo jednom proći kroz *while*-strukturu i uzeti samo 1. član niza, posljednju *pohranjenu* vrijednost varijable n treba smanjiti za 1. To je učinjeno u 11. retku koda.

- *Pitanje za razmišljanje*: Jesmo li u 5. retku mogli inicijalizirati $n=1$? Ako jesmo, preradite kod tako da ispisuje ispravan rezultat. Ako nismo, objasnite svoj odgovor.

- *Pitanja za razmišljanje*: Preradite programski kod tako da ispisuje točno rješenje ako se na početku postavi uvjet $\sum_{n=1}^{m} a_n < x$.

z8.m.

U ovom zadatku najprije moramo razriješiti sljedeći problem: kako dobiti svaku pojedinu znamenku prirodnoga broja n? Osnovnu ideju izložit ćemo na jednostavnom primjeru.

Neka je $n = 1234$. Dijeljenjem s 10 dobijemo količnik 123 i ostatak 4. Dijeljenjem broja 123 s 10 dobijemo količnik 12 i ostatak 3. Dijeljenjem broja 12 s 10 dobijemo količnik 1 i ostatak 2. Dijeljenjem broja 1 s 10 dobijemo količnik 0 i ostatak 1. Niz ostataka pri ovim dijeljenjima je 4, 3, 2, 1 i to su sve znamenke zadanoga broja (ispisane obrnutim redoslijedom).

Zbog toga je osnovna ideja: podijelimo zadani broj sa 10 i zapišimo ostatak. Ako je dobiveni količnik veći od 0, podijelimo ga s 10 i zapišimo novi ostatak. Ponavljamo postupak sve dok posljednji dobiveni količnik ne bude jednak 0. Tada zapišemo ostatak i prekinemo postupak.

Kako ovaj postupak implementirati u MATLAB-u? Ostatak pri dijeljenju prirodnoga broja n brojem 10 dobijemo izravnom primjenom ugrađene funkcije `mod`. No, kako dobiti količnik? Ne možemo jednostavno napisati da je taj količnik jednak $\frac{n}{10}$ jer vrijednost toga razlomka ne mora biti prirodan broj. Uostalom, pogledajmo gornji primjer: $\frac{1234}{10} = 123.4$, a mi smo daljnji postupak nastavili s brojem 123, a ne sa 123.4. Što onda učiniti? Vrlo jednostavno: količnik dvaju prirodnih brojeva dobijemo korištenjem funkcije „najveće cijelo“, odnosno funkcije `floor`. Jasno je da je $\left\lfloor \frac{1234}{10} \right\rfloor = \lfloor 123.4 \rfloor = 123$, $\left\lfloor \frac{123}{10} \right\rfloor = \lfloor 12.3 \rfloor = 12$ itd. Tako ćemo, dakle, količnike određivati pomoću funkcije `floor`, a ostatke pri dijeljenju – koji će zapravo biti znamenke – pomoću funkcije `mod`. U kodu ćemo koristiti i relaciju `~=` koju čitamo „biti različit od“.

Pogledajmo kako izgleda programski kod. U prvom retku inicijaliziramo varijablu *rezultat* i postavimo njezinu vrijednost na 0. Vrijednost te varijable nakon izvršenja cijeloga koda bit će rješenje zadatka, pa otuda i ime varijabli. Potom za svaki prirodan broj u segmentu [1, 100] odredimo zbroj njegovih znamenaka. U tu svrhu koristimo *for*-petlju. Primijetimo da ćemo unutar te petlje mijenjati vrijednost varijable n koju koristimo u sintaksi *for*-petlje. Ta promjena ovdje nije problematična. Naime, nakon što se u jednom prolazu kroz *for*-petlju izvrše sve naredbe unutar nje, vrijednost varijable n se povećava za 1 u odnosu na vrijednost na početku prolaza, a ne na

posljednju vrijednost dobivenu izvršenjem svih naredbi unutar petlje. Drugim riječima, *for*-petlja će se izvršiti točno 100 puta: prvi put za $n = 1$, drugi put za $n = 2$, …, te 100. put za $n = 100$.

Pri svakom prolazu kroz *for*-petlju gore opisanim algoritmom računamo zbroj znamenaka broja n (uzimajući za n najprije 1, pa 2, pa 3 itd.). Taj zbroj u 4. retku koda dodajemo vrijednosti varijabli *rezultat* mijenjajući njezinu vrijednost. Naime, mi zapravo računamo zbroj zbrojeva znamenaka svih prirodnih brojeva iz segmenta $[1,100]$, a taj je zbroj, zbog asocijativnosti zbrajanja cijelih brojeva, očito jednak

$$S = 1+2+3+4+5+6+7+8+9+(1+0)+(1+1)+(1+2)+\ldots+(9+8)+(9+9)+(1+0+0) =$$
$$= 1+2+3+4+5+6+7+8+9+1+0+1+1+1+2+\ldots+9+8+9+9+1+0+0.$$

Nakon prvoga prolaza kroz *for*-petlju vrijednost varijable *rezultat* bit će jednaka zbroju znamenaka prvih 1 prirodnih brojeva, tj. b_1. Nakon drugoga prolaza kroz tu petlju vrijednost varijable *rezultat* bit će jednaka zbroju *svih* znamenaka prvih dvaju prirodnih brojeva, tj. b_1+b_2 itd. Nakon 100. prolaza vrijednost varijable *rezultat* bit će jednaka $b_1+b_2+\ldots+b_{100}$, odnosno rješenju zadatka.

- *Pitanje za razmišljanje*: Preradite navedeni kod tako da ispisuje zbrojeve umnožaka svih znamenaka, tj. za slučaj kad je b_n jednak umnošku svih znamenaka u dekadskom zapisu broja n. Potom preradite kod za slučaj kad je b_n jednak zbroju kvadrata svih znamenaka u dekadskom zapisu broja n.

- *Pitanje za razmišljanje*: Može li se navedeni zadatak riješiti korištenjem *while*-strukture umjesto *for*-petlje? Ako može, preradite kod tako da ispisuje ispravno rješenje. Ako ne može, objasnite svoj odgovor.