Programsko rješenje osnovnih algoritama linearne algebre u programskom jeziku C

Jukić, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

PROGRAMSKO RJEŠENJE OSNOVNIH ALGORITAMA LINEARNE ALGEBRE U PROGRAMSKOM JEZIKU C

Završni rad

Matej Jukić

Osijek, 2019.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 24.09.2019.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Matej Jukić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3783, 27.09.2019.
OIB studenta:	65940738935
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Naslov završnog rada:	Programsko rješenje osnovnih algoritama linearne algebre u programskom jeziku C
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	24.09.2019.
Datum potvrde ocjene Odbora:	04.03.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 26.05.2020.

Ime i prezime studenta:	Matej Jukić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3783, 27.09.2019.
Turnitin podudaranje [%]:	20

Ovom izjavom izjavljujem da je rad pod nazivom**: Programsko rješenje osnovnih algoritama linearne algebre u programskom jeziku C**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Rudec

i sumentora Izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Linearna algebra je grana matematike koja izučava vektore, matrice i linearne operatore. Budući da rad s matricama zahtjeva puno vremena i lako se pogriješi prilikom računanja zbog ljudske nesmotrenosti, pokazalo se praktičnim koristiti računala i programska rješenja kako bi se olakšalo i ubrzalo obavljanje različitih operacija nad njima i smanjila mogućnost pogreške. Zbog složenosti pojedinih algoritama pokazalo se dobrim koristiti programski jezik C koji je brz i omogućava upravljanje memorijom. Kratki pretpregled:

1.1. Zadatak završnog rada

Prikazati osnovne algoritma linearne algebre I. u programskom jeziku C i napraviti program koji će obavljati osnovne operacije nad matricama i vektorima.

Drugo poglavlje sadrži osnovne informacije o programskom jeziku C

Treće poglavlje će obrađivati vektore i osnovne operacije s vektorima kao što su zbrajanje vektora, množenje vektora skalarom, skalarni umnožak, vektorski umnožak i mješoviti umnožak.

Četvrto poglavlje se bavi matricama i osnovnim operacijama nad matricama kao što su množenje matrice skalarom, zbrajanje, množenje i transponiranje matrica, traženje determinante matrice, adjungiranje i invertiranje matrice i rangom matrice.

Posljednje poglavlje će biti zaključak i sadržavat će osvrt na prethodno napisana poglavlja.

2. PROGRAMSKI JEZIK C

C je viši programski jezik kojeg je osmislio Dennis Ritchie 1970-ih kao alat za programiranje računalnih operacijskih sustava, točnije UNIX-a. C je s vremenom doživio niz promjena i ima više inačica s vlastitim standardima kao što su: K&R C, ISO C / ANSI C, C99 i C11.

Uskoro je našao primjenu i u ostalim područjima, osim izrade operacijskih sustava, pa se tako počeo upotrebljavati za izradu tekst procesora, računalnih igara i sl. Iako je C jezik široke primjene, trenutno mu popularnost opada jer zahtijeva od programera poznavanje upravljanja memorijom računala, dok većina novijih jezika ima ugrađeno automatsko upravljanje memorijom.
Ipak, taj nedostatak je ujedno i C-ova prednost jer mu skoro pa izravan pristup memoriji omogućava brže izvođenje koda programske podrške.
C u posljednje vrijeme ponovo pronalazi uporabu u programiranju mikrokontrolera i interneta objekata baš zbog iznad navedenih svojstava.

C sadrži sve osnovne tipove podataka kao što su: cjelobrojni(int), znakovni (char), realni (float) i realne dvostruke točnosti (double). Također se programerima pruža mogućnost da stvaraju vlastite tipove podataka kao što su strukture ili da stvaraju polja istovrsnih tipova podataka. C sadrži sve osnovne naredbe za kontrolu toka kao što su: if/else, for, while, do/while i switch. Svaki program u C-u obavezno mora sadržavati jednu main() funkciju kako bi radio. Osim nje može sadržavati i neodređen broj drugih funkcija. C nema ugrađene funkcije kao neki programski jezici, nego ih mora uvoziti iz biblioteka pomoću pretprocesorske naredbe #include. Neke od poznatijih biblioteka su stdio.h, stdlib.h, string.h, math.h itd. [1][2]

3. VEKTORI

3.1. O vektorima

Vektor je usmjerena dužina. Može ga se odrediti s dvije točke T_1 i T_2, gdje je T_1 početna točka ili hvatište vektora, a T_2 je krajnja točka i označava se kao $\overrightarrow{T_1T_2}$.
Vektore se može označiti i malim tiskanim slovima [3], zato se može uvesti

$\overrightarrow{T_1T_2} = \vec{a}$,
gdje su

$$T_1(x_{1,1},\ x_{1,2}, x_{1,3}, \dots, x_{1,n}),$$
$$T_2(x_{2,1},\ x_{2,2}, x_{2,3}, \dots, x_{2,n}),$$

$\vec{a} = (x_{2,1} - x_{1,1})\overrightarrow{x_1} + (x_{2,2} - x_{1,2})\overrightarrow{x_2} + (x_{2,3} - x_{1,3})\overrightarrow{x_3} + \cdots + (x_{2,n} - x_{1,n})\overrightarrow{x_n}$.
[3]

Za potrebe završnog rada bilo bi poželjno definirati vektore u trodimenzionalnom vektorskom prostoru V^3 s 3 međusobno okomite osi: $Ox, Oy\ i\ Oz$ i njima pripadajućim, linearno nezavisnim, okomitim, jediničnim vektorima $\vec{\imath}, \vec{\jmath}, \vec{k}$. [4]

Za vektore se kaže da su međusobno nezavisni ako se ne mogu prikazati kao linearna kombinacija preostalih vektora, odnosno relacija

$\alpha\overrightarrow{x_1} + \beta\overrightarrow{x_2} + \gamma\overrightarrow{x_2} + \cdots + \omega\overrightarrow{x_n} = 0$, vrijedi samo

ako su $\alpha = \beta = \gamma = \cdots = \omega = 0$ [3]

Svaki vektor iz vektorskog prostora V^3 je moguće prikazati kao zbroj jediničnih vektora $\vec{\imath}, \vec{\jmath}, \vec{k}$.

Sada se točke T_1 i T_2 može prikazati
kao

$$T_1(x_1,\ y_1, z_1),$$

$$T_2(x_2,\ y_2, z_2),$$

Točkama T_1 i T_2 odgovaraju radij-vektori $\overrightarrow{OT_1}$ i $\overrightarrow{OT_2}$.

$\overrightarrow{OT_1} = (x_1 - 0)\vec{\imath} + (y_1 - 0)\vec{\jmath} + (z_1 - 0)\vec{k} = x_1\vec{\imath} + y_1\vec{\jmath} + z_1\vec{k}$ i

$\overrightarrow{OT_2} = (x_2 - 0)\vec{\imath} + (y_2 - 0)\vec{\jmath} + (z_2 - 0)\vec{k} = x_2\vec{\imath} + y_2\vec{\jmath} + z_2\vec{k}$,

Vektor $\vec{a}$ se može prikazati kao zbroj vektora $\overrightarrow{T_1O}$ i $\overrightarrow{OT_2}$:

$$\vec{a} = \overrightarrow{T_1O} + \overrightarrow{OT_2} = \overrightarrow{OT_2} - \overrightarrow{OT_1} = x_2\vec{\imath} + y_2\vec{\jmath} + z_2\vec{k} - x_1\vec{\imath} + y_1\vec{\jmath} + z_1\vec{k} =$$

$(x_2 - x_1)\vec{\imath} + (y_2 - y_1)\vec{\jmath} + (z_2 - z_1)\vec{k}$ (*)

Neka su:

$a_x = x_2 - x_1,$

$a_y = y_2 - y_1,$

$a_z = z_2 - z_1,$

Uvrštavanjem a_x, a_y, a_z u (*) dobije se:

$$\vec{a} = a_x\vec{\imath} + a_y\vec{\jmath} + a_z\vec{k}$$

Svaki vektor ima 3 osobine koje ga opisuju, a to su duljina (modul), smjer i orijentacija. [3]

Modul vektora iskazujemo kao

$$|\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Smjer govori na kojem pravcu nositelju se nalazi duljina.

Orijentaciju se određuje za vektore istog smjera - kolinearne vektore. Za vektore se kaže da su kolinearni ako se mogu iskazati kao skalarni umnožak jedan drugoga. [3] Dva kolinearna vektora mogu biti iste i suprotne orijentacije. Dva vektora imaju istu orijentaciju ako važi $\vec{a} = \lambda\vec{b}$,

a suprotnu ako je $\vec{a} = -\lambda\vec{b}$,

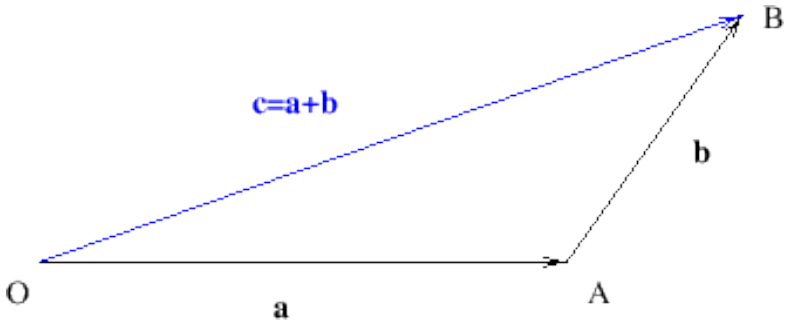
gdje je $\lambda>0$, $\lambda \epsilon R$.

U programu je vektor realiziran strukturom

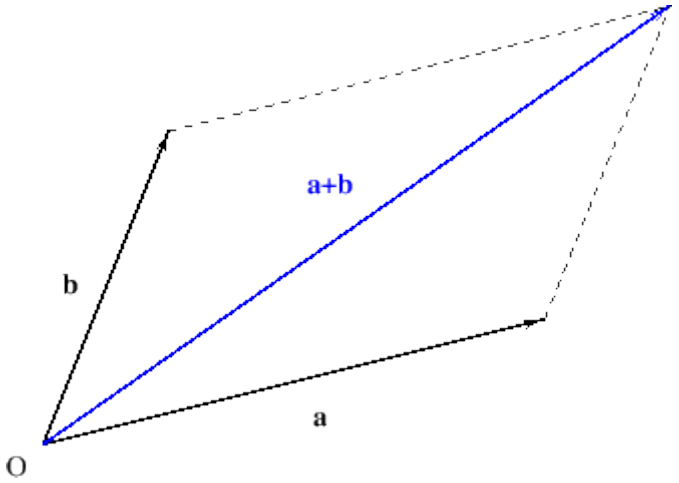
3.2. Zbrajanje vektora

3.2.1. O zbrajanju vektora

Vektori se zbrajaju koristeći pravilo trokuta i pravilo paralelograma.



Sl.3.1. *Zbrajanje vektora pravilom trokuta*



Sl.3.2. *Zbrajanje vektora pravilom paralelograma*

Svaki vektor u trodimenzionalnom pravokutnom vektorskom prostoru može se prikazati kao zbroj tri linearno nezavisna jedinična vektora $\vec{\imath}, \vec{\jmath}, \vec{k}$, koji odgovaraju trima međusobno okomitim osima: $Ox, Oy\ i\ Oz$. [4]

Neka su:

$$\vec{a} = a_x\vec{\imath} + a_y\vec{\jmath} + a_z\vec{k},$$

$$\vec{b} = b_x\vec{\imath} + b_y\vec{\jmath} + b_z\vec{k}$$

$$\vec{c} = \vec{a} + \vec{b} \quad .$$

Slijedi da je:

$\vec{c} = a_x\vec{\imath} + a_y\vec{j} + a_z\vec{k} + b_x\vec{\imath} + b_y\vec{j} + b_z\vec{k} = (a_x + b_x)\vec{\imath} + (a_y + b_y)\vec{j} + (a_z + b_z)\vec{k}$.

Odnosno zbroj vektora je jednak zbroju njihovih odgovarajućih jediničnih vektora.

3.2.2. Svojstva zbrajanja

1.Grupoidnost:

Rezultat zbrajanja 2 vektora je vektor

$$\vec{a} + \vec{b} = \vec{c}$$

2. Asocijativnost

Za vektore $\vec{a}$, $\vec{b}$, $\vec{c}$ $\in V^3$ vrijedi:

$$(\vec{a} + \vec{b}) + \vec{c} = \vec{a} + (\vec{b} + \vec{c})$$

3. Neutralni element

Za proizvoljni vektor $\vec{a} \in V^3$ postoji neutralni element $\vec{0}$ (nul-vektor) za koji vrijedi

$$\vec{a} + \vec{0} = \vec{a}$$

4. Inverzni element

Za proizvoljni vektor $\vec{a} \in V^3$ postoji inverzni element $-\vec{a}$ za koji vrijedi

$$\vec{a} + (-\vec{a}) = 0$$

5. Komutativnost

Za vektore $\vec{a}, \vec{b} \in V^3$ vrijedi

$\vec{a} + \vec{b} = \vec{b} + \vec{a}$ [3]

3.2.3 Programsko rješenje zbrajanja i vremenska složenost algoritma

```
struct_vektor* zbroji_vektore(struct_vektor* V1, struct_vektor* V2)
{
   struct_vektor* V=alokacija_vektora();
   for(int i=0; i<n; i++)
/*polje s 3 elementa za pohranjivanje skalara koji množe kanonsku bazu prostora V^3*/
   {
      V->vektor[i]=V1->vektor[i]+V2->vektor[i];
```

```
/*Zbrajamo odgovarajuće elemente vektora,
Zbroj skalara koji stoje uz i⃗ sprema se u prvi element polja, one koji stoje uz j⃗ u drugi, a uz k⃗ u treći*/
    }

    return V;
}
```

Vremenska složenost ovog algoritma je *O(n)*, gdje je *n* broj vektora baze vektorskog prostora.

3.3. Množenje vektora skalarom

3.3.1. Teorijska podloga

Za proizvoljni vektor $\overrightarrow{a} \in V^3$ i $\lambda \in \mathbb{R}$ definira se vektor $\vec{b}$ za koji vrijedi

$$\vec{b} = \lambda\overrightarrow{a}$$

Jer je $\vec{a} = a_x\vec{\imath} + a_y\vec{\jmath} + a_z\vec{k}$ i $\vec{b} = b_x\vec{\imath} + b_y\vec{\jmath} + b_z\vec{k}$, uvrštavanjem se dobije

$$b_x\vec{\imath} + b_y\vec{\jmath} + b_z\vec{k} = \lambda a_x\vec{\imath} + \lambda a_y\vec{\jmath} + \lambda a_z\vec{k}$$

Zbog linearne nezavisnosti $\vec{\imath}, \vec{\jmath}, \vec{k}$ vrijedi:

$b_x = \lambda a_x$, $b_y = \lambda a_y$ i $b_z = \lambda a_z$

3.3.2. Svojstva množenja skalarom

1.Distributivnost vektora [3]

Za $\vec{a}, \vec{b} \in V^3$ i $\lambda \in \mathbb{R}$ vrijedi

$\lambda(\vec{a} + \vec{b}) = \lambda\vec{a} + \lambda\vec{b}$

2. Distributivnost skalara

Za $\vec{a} \in V^3$ i $\lambda, \mu \in R$ vrijedi

$(\lambda + \mu)\vec{a} = \lambda\vec{a} + \mu\vec{a}$

3. Kvaziasocijativnost

Za $\vec{a} \in V^3$ i $\lambda, \mu \in R$ vrijedi

$(\lambda\mu)\vec{a} = \lambda(\mu\vec{a})$

3.3.3. Programsko rješenje množenja skalarom i vremenska složenost algoritma

```
void pomnozi_skalarom(float a, struct_vektor* V){
 for (int i=0; i<n; i++){
    V->vektor[i]*=a;
  }
}
```

Vremenska složenost ovog algoritma je *O(n)*, gdje je *n* broj vektora baze vektorskog prostora.

3.4. Skalarni umnožak vektora

3.4.1. O skalarnom umnošku vektora

Neka postoje 2 vektora $\vec{a}, \vec{b} \in V^3$ koje množimo skalarno, njihov umnožak će biti realni broj $m \in R$ koji je jednak umnošku modula navedenih vektora i kosinusu kuta između njih. [4]

$m = \vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| cos\sphericalangle(\vec{a}\vec{b})$

Neka su $\vec{a} = a_x\vec{\imath} + a_y\vec{\jmath} + a_z\vec{k}$ i $\vec{b} = b_x\vec{\imath} + b_y\vec{\jmath} + b_z\vec{k}$,

Iz toga slijedi:

$$\vec{a} \cdot \vec{b} = \vec{a} = (a_x\vec{\imath} + a_y\vec{\jmath} + a_z\vec{k}) \cdot (b_x\vec{\imath} + b_y\vec{\jmath} + b_z\vec{k}) =$$

$$= a_x\vec{\imath} \cdot b_x\vec{\imath} + a_x\vec{\imath} \cdot b_y\vec{\jmath} \cdot + a_x\vec{\imath} \cdot b_z\vec{k}$$

$$+ a_y\vec{\jmath} \cdot b_x\vec{\imath} + a_y\vec{\jmath} \cdot b_y\vec{\jmath} + a_y\vec{\jmath} \cdot b_z\vec{k}$$

$$+a_z\vec{k}\cdot b_x\vec{\imath}+a_z\vec{k}\cdot b_y\vec{\jmath}+a_z\vec{k}\cdot b_z\vec{k}.$$

Budući da su jedinični vektori $\vec{\imath},\vec{\jmath},\vec{k}$ baze vektorskog prostora, međusobno su linearno nezavisni i okomiti pa za njih vrijedi [3]:

$$\vec{\imath}\cdot\vec{\imath}=|\vec{\imath}|\cdot|\vec{\imath}|cos\sphericalangle(\vec{\imath}\vec{\imath})=|\vec{\imath}|\cdot|\vec{\imath}|cos\sphericalangle(0°)=|\vec{\imath}|^2=1,$$

$$\vec{\imath}\cdot\vec{\jmath}=|\vec{\imath}|\cdot|\vec{\jmath}|cos\sphericalangle(\vec{\imath}\vec{\jmath})=|\vec{\imath}|\cdot|\vec{\jmath}|cos\sphericalangle(90°)=0,$$

$$\vec{\imath}\cdot\vec{k}=|\vec{\imath}|\cdot\left|\vec{k}\right|cos\sphericalangle\left(\vec{\imath}\vec{k}\right)=|\vec{\imath}|\cdot|\vec{\jmath}|cos\sphericalangle(90°)=0,$$

$$\vec{\jmath}\cdot\vec{\jmath}=|\vec{\jmath}|\cdot|\vec{\jmath}|cos\sphericalangle(\vec{\jmath}\vec{\jmath})=|\vec{\jmath}|\cdot|\vec{\jmath}|cos\sphericalangle(0°)=|\vec{\jmath}|^2=1,$$

$$\vec{\jmath}\cdot\vec{k}=|\vec{\jmath}|\cdot\left|\vec{k}\right|cos\sphericalangle\left(\vec{\jmath}\vec{k}\right)=|\vec{\jmath}|\cdot\left|\vec{k}\right|cos\sphericalangle(90°)=0,$$

$$\vec{k}\cdot\vec{k}=\left|\vec{k}\right|\cdot\left|\vec{k}\right|cos\sphericalangle\left(\vec{k}\vec{k}\right)=\left|\vec{k}\right|\cdot\left|\vec{k}\right|cos\sphericalangle(0°)=\left|\vec{k}\right|^2=1$$

Iz ovoga konačno slijedi

$$\vec{a}\cdot\vec{b}=a_x\cdot b_x\cdot|\vec{\imath}|^2+a_y\cdot b_y\cdot|\vec{\jmath}|^2+a_z\cdot b_z\cdot|k|^2=a_x\cdot b_x+a_y\cdot b_y+a_z\cdot b_z$$

3.4.2. Svojstva skalarnog umnoška vektora

1. Pozitivnost [4]

$\vec{a}\cdot\vec{a}\geq 0,\ \vec{a}\cdot\vec{a}=0$ ako i samo ako je $\vec{a}=\vec{0}$

2.Komutativnost

Za umnožak dva vektora vrijedi:

$$\vec{a}\cdot\vec{b}=\vec{b}\cdot\vec{a}$$

3.Distributivnost

Za bilo koja 3 vektora vrijedi:

$$\vec{a}\cdot\left(\vec{b}+\vec{c}\right)=\vec{a}\cdot\vec{b}+\vec{a}\cdot\vec{c}$$

4.Homogenost

Za $\vec{a},\vec{b}\in V^3$ i $\lambda\in\mathbb{R}$ vrijedi

$\lambda\left(\vec{a}\cdot\vec{b}\right)=(\lambda\vec{a})\cdot\vec{b}$

3.4.3. Programsko rješenje skalarnog umnoška vektora i vremenska složenost algoritma

```
float skalarni_umnozak(struct_vektor* V1, struct_vektor* V2, int n){
```

```
    float umnozak=0;
    for(int i=0; i<n; i++){
        umnozak+=V2->vektor[i]*V1->vektor[i];
/*umnošku pridodajemo umnožak skalara odgovarajućih jediničnih vektora+/
    }
    return umnozak;
}
```

Vremenska složenost algoritma skalarnog umnoška dvaju vektora je O(n), gdje je n broj vektora baze vektorskog prostora.

3.5.Vektorski umnožak

3.5.1. O vektorskom umnošku

Imamo vektore $\vec{a}, \vec{b}, \vec{c} \in V^3$

Gdje je vektorski umnožak dva vektora $\vec{a} \times \vec{b}$ vektor $\vec{c}$, koji je okomit na vektore $\vec{a}$ i $\vec{b}$ i čiji je modul jednak površini paralelograma konstruiranog na ta dva vektora. [4]

$$\vec{c} = \vec{a} \times \vec{b}$$
$$|\vec{c}| = |\vec{a}| \cdot |\vec{b}| sin\sphericalangle(\vec{a}\vec{b})$$

Neka su $\vec{a} = a_x\vec{\imath} + a_y\vec{\jmath} + a_z\vec{k}$ i $\vec{b} = b_x\vec{\imath} + b_y\vec{\jmath} + b_z\vec{k}$,

Iz toga slijedi:

$$\vec{a} \times \vec{b} = (a_x\vec{\imath} + a_y\vec{\jmath} + a_z\vec{k}) \times (b_x\vec{\imath} + b_y\vec{\jmath} + b_z\vec{k}) = a_x\vec{\imath} \times b_x\vec{\imath} + a_x\vec{\imath} \times b_y\vec{\jmath} + a_x\vec{\imath} \times b_z\vec{k} + a_y\vec{\jmath} \times b_x\vec{\imath} + a_y\vec{\jmath} \times b_y\vec{\jmath} + a_y\vec{\jmath} \times b_z\vec{k} + a_z\vec{k} \times b_x\vec{\imath} + a_z\vec{k} \times b_y\vec{\jmath} + a_z\vec{k} \times b_z\vec{k}.$$

Budući da su jedinični vektori $\vec{\imath}, \vec{\jmath}, \vec{k}$ baze vektorskog prostora, međusobno su linearno nezavisni i okomiti pa za njih vrijedi:

$$\vec{\imath} \times \vec{\imath} = |\vec{\imath}| \cdot |\vec{\imath}| sin\sphericalangle(\vec{\imath}\vec{\imath}) = |\vec{\imath}| \cdot |\vec{\imath}| sin\sphericalangle(0°) = 0,$$

$$\vec{\imath} \times \vec{\jmath} = |\vec{\imath}| \cdot |\vec{\jmath}| sin\sphericalangle(\vec{\imath}\vec{\jmath}) \cdot \vec{k} = |\vec{\imath}| \cdot |\vec{\jmath}| sin\sphericalangle(90°) \cdot \vec{k} = \vec{k},$$

$$\vec{\imath} \times \vec{k} = |\vec{\imath}| \cdot |\vec{k}| sin\sphericalangle(\vec{\imath}\vec{k}) \cdot \vec{\jmath} = |\vec{\imath}| \cdot |\vec{k}| sin\sphericalangle(-90°) \cdot \vec{\jmath} = -\vec{\jmath},$$

$$\vec{\jmath} \times \vec{\jmath} = |\vec{\jmath}| \cdot |\vec{\jmath}| sin\sphericalangle(\vec{\jmath}\vec{\jmath}) = |\vec{\jmath}| \cdot |\vec{\jmath}| sin\sphericalangle(0°) = 0,$$

$$\vec{j} \times \vec{k} = |\vec{j}| \cdot |\vec{k}| sin\sphericalangle(\vec{j}\vec{k}) \cdot \vec{\imath} = |\vec{j}| \cdot |\vec{k}| sin\sphericalangle(90°) \cdot \vec{\imath} = \cdot \vec{\imath},$$

$$\vec{k} \times \vec{k} = |\vec{k}| \cdot |\vec{k}| sin\sphericalangle(\vec{k}\vec{k}) = |\vec{k}| \cdot |\vec{k}| sin\sphericalangle(0°) = 0$$

Uvrštavanjem u prethodni izraz i sređivanjem dobijemo:

$\vec{c} = \vec{a} \times \vec{b} = a_x\vec{\imath} \times b_x\vec{\imath} + a_x\vec{\imath} \times b_y\vec{j} + a_x\vec{\imath} \times b_z\vec{k} + a_y\vec{j} \times b_x\vec{\imath} + a_y\vec{j} \times b_y\vec{j} + a_y\vec{j} \times b_z\vec{k} + a_z\vec{k} \times b_x\vec{\imath} + a_z\vec{k} \times b_y\vec{j} + a_z\vec{k} \times b_z\vec{k} = 0 + a_xb_y\,\vec{k} - a_xb_z\vec{j} - a_yb_x\vec{k} + 0 + a_yb_z\,\vec{\imath} + a_zb_x\,\vec{j} - a_zb_y\vec{\imath} + 0$

Konačno [3] :

$$\vec{c} = \vec{a} \times \vec{b} = (a_yb_z - a_zb_y)\,\vec{\imath} + (a_zb_x - a_xb_z)\,\vec{j} + (a_xb_y - a_yb_x)\,\vec{k}$$

3.5.2. Svojstva vektorskog umnoška

1.Antikomutativnost [4]
Za $\vec{a}, \vec{b} \in V^3$ vrijedi

$\vec{a} \times \vec{b} = -(\vec{b} \times \vec{a})$.

2.Distributivnost
Za vektore $\vec{a}, \vec{b}, \vec{c} \in V^3$ vrijedi

$\vec{a} \times (\vec{b} + \vec{c}) = \vec{a} \times \vec{b} + \vec{a} \times \vec{c}$.

3. Homogenost

Za $\vec{a}, \vec{b} \in V^3$ i $\lambda \in \mathbb{R}$ vrijedi

$\lambda(\vec{a} \times \vec{b}) = \lambda\vec{a} \times \vec{b}$

3.5.3. Programsko rješenje vektorskog umnoška i vremenska složenost algoritma

```
struct_vektor* vektorski_produkt(struct_vektor* V1, struct_vektor* V2){
struct_vektor* V=alokacija_vektora();
V->vektor[0]=(V1->vektor[4]*V2->vektor[3])-(V1->vektor[3]*V2->vektor[4]);
V->vektor[4]=-((V1->vektor[0]*V2->vektor[3])-(V1->vektor[3]*V2->vektor[0]));
V->vektor[3]=(V1->vektor[0]*V2->vektor[4])-(V1->vektor[4]*V2->vektor[0]);
```

return V;

}

Vremenska složenost algoritma je *O(n)*=1 jer ne postoji nikakva promjenjiva varijabla od koje zavisi vremenska složenost algoritma

3.6. Mješoviti umnožak

3.6.1. O mješovitom umnošku triju vektora

Neka su vektori $\vec{a}$, $\vec{b}$, $\vec{c} \in V^3$.

Mješoviti umnožak vektora $\vec{a}, \vec{b}, \vec{c}$ je skalar $m \in \mathbb{R}$, za koji vrijedi

$$m = (\vec{a} \times \vec{b}) \cdot \vec{c}$$

Vrijednost skalara m jednaka je obujmu paralelepipeda kojega zatvaraju vektori $\vec{a}, \vec{b}, \vec{c}$.

Može se zamijetiti da se mješoviti produkt sastoji od vektorskog i skalarnog umnoška, tj.

$(\vec{a} \times \vec{b}) = \vec{d}$ iz čega slijedi $\vec{d} \cdot \vec{c} = m$. [4]

O vektorskom i skalarnom umnošku je pisano u prethodnim potpoglavljima, zato nije potrebno nešto naročito reći o mješovitom produktu.

3.6.2. Programsko rješenje mješovitog umnoška i vremenska složenost algoritma

U programskom rješenju bi bilo najbolje poslužiti se gotovim rješenjima iz prethodnih pootpoglavlja.

Programsko rješenje:

```
float mjesoviti_produkt(struct_vektor* V1, struct_vektor* V2, struct_vektor* V3){
return skalarni_umnozak(vektorski_produkt(V1, V2), V3, 3);
}
```

Možemo zaključiti da programsko rješenje ima vremensku složenost *O(n)=1* jer nema nijednu varijablu.

4. MATRICE

4.1.O matricama

4.1.1. Što je matrica?

Matrica je svaka pravokutna tablica realnih ili kompleksnih brojeva. Ako matrica sadrži m redaka i n stupaca, kažemo da je tipa $m \times n$ i zapisujemo ju u obliku

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

ili kraće **A**=(a_{ij}). Element a_{ij} naziva se **opći element** matrice **A**. To je realan ili kompleksan broj.

Za kvadratnu matricu s n redaka i stupaca kaže se da je reda n. Za kvadratnu matricu se kaže da je gornja trokutasta ako su joj svi elementi ispod dijagonale jednaki nuli. donja trokutasta ako su elementi iznad dijagonale jednaki nuli, dijagonalna ako su svi elementi osim onih na dijagonali jednaki nuli, skalarna ako su svi elementi dijagonalne matrice jednaki. Za dijagonalnu matricu kojoj su dijagonalni elementi jednaki jedinici kažemo da je jedinična matrica i označavamo ju s **I**. Matricu kojoj su svi elementi jednaki nuli nazivamo nul matricom i označavamo s **0.**

4.1.2. Realizacija matrica u programskom rješenju

U programskom rješenju matrica je deklarirana kao struktura koja sadrži dvije cjelobrojne varijable, koje određuju broj stupaca i redaka matrice, dva pokazivača na strukturu matrice kako bi se mogla napraviti povezana lista koja bi služila za pohranu matrica i dvostruki pokazivač kako bih mogao alocirati 2D polje koje sadrži matrične vrijednosti.

```
typedef struct struct_matrica{
   int br_redaka;
   int br_stupaca;
   float** matrica;
   struct struct_matrica *iduca_matrica;
   struct struct_matrica *prethodna_matrica;}
```

struct_matrica;

4.2. Zbrajanje matrica

4.2.1. O zbrajanju matrica

Neka je zadan sustav jednadžbi

$$\begin{aligned} y_1 &= a_{11}x_{11} + a_{12}x_{12} + a_{13}x_{13} + \cdots + a_{1n}x_{1n} \\ y_2 &= a_{21}x_{21} + a_{22}x_{22} + a_{23}x_{23} + \cdots + a_{2n}x_{2n} \\ y_3 &= a_{31}x_{31} + a_{32}x_{32} + a_{33}x_{33} + \cdots + a_{3n}x_{3n} \\ &\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ y_m &= a_{m1}x_{m1} + a_{m2}x_{m2} + a_{m3}x_{m3} + \cdots + a_{mn}x_{mn} \end{aligned}$$

i

$$\begin{aligned} z_1 &= b_{11}x_{11} + b_{13}x_{12} + b_{13}x_{13} + \cdots + b_{1n}x_{1n} \\ z_2 &= b_{21}x_{21} + b_{22}x_{22} + b_{23}x_{23} + \cdots + b_{2n}x_{2n} \\ z_3 &= b_{31}x_{31} + b_{32}x_{32} + b_{33}x_{33} + \cdots + b_{3n}x_{3n} \\ &\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ z_m &= b_{m1}x_{m1} + b_{m2}x_{m2} + b_{m3}x_{m3} + \cdots + b_{mn}x_{mn} \end{aligned}$$

Neka je

$$y = y_1 + y_2 + y_3 + \ldots + y_m$$
$$z = z_1 + z_2 + z_3 + \ldots + z_m$$
$$w = w_1 + w_2 + w_3 + \ldots + w_m$$

i neka je $w=y+z$, tada vrijedi

$$\begin{aligned} w_1 &= (a_{11} + b_{11})x_{11} + (a_{12} + b_{12})x_{12} + (a_{13} + b_{13})x_{13} + \cdots + (a_{1n} + b_{1n})x_{1n} \\ w_2 &= (a_{21} + b_{21})x_{21} + (a_{22} + b_{22})x_{22} + (a_{23} + b_{23})x_{23} + \cdots + (a_{2n} + b_{2n})x_{2n} \\ w_3 &= (a_{31} + b_{31})x_{31} + (a_{32} + b_{32})x_{32} + (a_{33} + b_{33})x_{33} + \cdots + (a_{3n} + b_{3n})x_{3n} \\ &\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ w_m &= (a_{m1} + b_{m1})x_{m1} + (a_{m2} + b_{m2})x_{m2} + (a_{m3} + b_{m3})x_{m3} + \cdots (a_{mn} + b_{mn})x_{mn} \end{aligned}$$

Uvedimo odgovarajuću matricu **W** za w.

$$\mathbf{W} = \begin{bmatrix} (a_{11} + b_{11}) & \cdots & (a_{1n} + b_{1n}) \\ \vdots & \ddots & \vdots \\ (a_{m1} + b_{m1}) & \cdots & (a_{mn} + b_{mn}) \end{bmatrix}$$

Iz ovoga se može zaključiti da je zbroj dviju matrica jednak zbroju njihovih elemenata koji se nalaze u istom stupcu i istom retku.

4.2.2. Svojstva zbrajanja matrica

Svojstva zbrajanja matrica su:

1. Komutativnost

   M+N=N+M

2. Asocijativnost

   (M+N)+K=M+(N+K)

   gdje su **M**, **N**, **K** matrice istih dimenzija.

4.2.3. Programsko rješenje za zbrajanje matrica i vremenska složenost algoritma

Programsko rješenje prvo provjerava jesu li dvije matrice istog tipa, a zatim zbraja njihove odgovarajuće elemente. Vremenska složenost algoritma je $O(n^2)$ jer algoritam ima 2 for petlje.

```
struct_matrica* zbroji_matrice(struct_matrica *matrica1, struct_matrica *matrica2)
{
  if (((matrica1->br_redaka)!=(matrica2->br_redaka))||((matrica2->br_stupaca)!=(matrica2->br_stupaca)))
  {
    printf("Nije moguce zbrojiti matrice s razlicitim brojem stupaca ili redaka");
    return NULL;
  }
  else
  {
    struct_matrica *matrica_zbroja;
    matrica_zbroja=alokacija_matrice(matrica1->br_redaka, matrica1->br_stupaca);
    int i, j;
```

```
    for(i=0; i<(matrica1->br_redaka); i++)
    {
       for (j=0; j<(matrica1->br_stupaca); j++)
       {
          matrica_zbroja->matrica[i][j]=matrica1->matrica[i][j]+matrica2->matrica[i][j];
       }
    }
    return matrica_zbroja;
}
```

4.3. Množenje matrica skalarom

4.3.1. O množenju matrice skalarom

Neka je zadana matrica $\mathbf{M}_{ij} = \begin{bmatrix} a_{11} & \cdots & a_{1j} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ij} \end{bmatrix}$ i skalar $\lambda \in \mathbb{R}$.

Množenje matrice skalarom definiramo kao

$$\lambda\mathbf{M}_{ij} = \begin{bmatrix} \lambda a_{11} & \cdots & \lambda a_{1j} \\ \vdots & \ddots & \vdots \\ \lambda a_{i1} & \cdots & \lambda a_{ij} \end{bmatrix}.$$

4.3.2. Svojstva množenja matrice skalarom

1.Distributivnost matrica

$$\lambda(\mathbf{M} + \mathbf{N}) = \lambda\mathbf{M} + \lambda\mathbf{N}$$

2. Distributivnost skalara

$$(\lambda + \mu)\mathbf{M} = \lambda\mathbf{M} + \mu\mathbf{M}$$

3. Kvaziasocijativnost

$$(\lambda\mu)\mathbf{M} = \lambda(\mu\mathbf{M})$$

Gdje su $\lambda, \mu \epsilon \mathbb{C}$ i $\mathbf{M}$, $\mathbf{N}$ su matrice istog tipa

4.3.3. Programsko rješenje množenja skalarom i vremenska složenost algoritma

```
void pomnozi(float skalar, struct_matrica* matrica1){
int i, j;
```

```
for (i=0; i<(matrica1->br_redaka); i++){
   for(j=0; j<(matrica1->br_stupaca); j++){
      matrica1->matrica[i][j]*=skalar;   }}}
```

Programsko rješenje sadrži dvije „for" petlje pomoću kojih skalar množi svaki element u 2D-polju.

Vremenska složenost je $O(n^2)$.

4.4.Transponiranje matrica

4.4.1.O transponiranju matrica

Neka postoji matrica $\mathbf{M}_{ij} = \begin{bmatrix} a_{11} & \cdots & a_{1j} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ij} \end{bmatrix}$, tada za nju postoji matrica$\mathbf{M}_{ji}^{T}$, za koju vrijedi

$\mathbf{M}_{ji}^{T} = \begin{bmatrix} a_{11} & \cdots & a_{i1} \\ \vdots & \ddots & \vdots \\ a_{1j} & \cdots & a_{ij} \end{bmatrix}$ gdje su $i, j \in \mathbb{Z}$

Za transponiranje kvadratnih matrica se može reći da je to zrcaljenje matrice u odnosu na njenu glavnu dijagonalu jer se elementi na glavnoj dijagonali preslikavaju na glavnu dijagonalu transponirane matrice.

4.4.2. Programsko rješenje i vremenska složenost algoritma

Funkcija sadrži 2 „for“ petlje pomoću kojih se prolazi kroz matricu i njene elemente preslikava u transponiranu matricu. Vremenska složenost je $O(n^2)$.

```
struct_matrica* transponiraj(struct_matrica* matrica1){
int i, j;
struct_matrica* transponirana_matrica;
transponirana_matrica=alokacija_matrice(matrica1->br_stupaca, matrica1->br_redaka);
for(i=0; i<(matrica1->br_redaka); i++){
   for(j=0; j<(matrica1->br_stupaca); j++){
      transponirana_matrica->matrica[j][i]=matrica1->matrica[i][j];
```

```
    }
}
return transponirana_matrica;}
```

4.5. Množenje matrica

4.5.1. O množenju matrica

Množenje matrica je definirano za dvije matrice samo ako je broj stupaca lijeve matrice jednak broju redaka desne matrice.
Za takve matrice kažemo da su ulančane. [3]
Neka su $\mathbf{M}_{ij}$ i $\mathbf{N}_{jk}$ dvije ulančane matrice,

$$\mathbf{M_{ij}} = \begin{bmatrix} a_{11} & \cdots & a_{1j} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ij} \end{bmatrix}$$

$$\mathbf{N_{jk}} = \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{j1} & \cdots & b_{jk} \end{bmatrix}$$

Neka je matrica $\mathbf{K}_{ik}$ njihov umnožak.

$$\mathbf{K}_{ik} = \mathbf{M}_{ij}\mathbf{N}_{jk} =$$

$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{11}b_{11} + \cdots + a_{1j}b_{j1} & \cdots & a_{11}b_{1k} + a_{12}b_{2k} + a_{13}b_{3k} + \cdots + a_{1j}b_{jk} \\ \vdots & \ddots & \vdots \\ a_{i1}b_{11} + a_{i2}b_{21} + a_{i1}b_{11} + \cdots + a_{ij}b_{j1} & \cdots & a_{i1}b_{1k} + a_{i2}b_{2k} + a_{i3}b_{3k} + \cdots + a_{ij}b_{jk} \end{bmatrix}$$

4.5.2. Svojstva množenja matrica

1. Asocijativnost

$$\left(\mathbf{M}_{ij}\mathbf{N}_{jk}\right)\mathbf{K}_{kl} = \mathbf{M}_{ij}(\mathbf{N}_{jk}\mathbf{K}_{kl})$$

2. Distributivnost zdesna

$$\left(\mathbf{M}_{ij} + \mathbf{N}_{ij}\right)\mathbf{K}_{jk} = \mathbf{M}_{ij}\mathbf{K}_{jk} + \mathbf{N}_{ij}\mathbf{K}_{jk}$$

3. Distributivnost slijeva

$$\mathbf{K}_{jk}(\mathbf{M}_{kl} + \mathbf{N}_{kl}) = \mathbf{K}_{jk}\mathbf{M}_{kl} + \mathbf{K}_{jk}\mathbf{N}_{kl}$$

4.5.3. Programsko rješenje množenja matrica i vremenska složenost algoritma

Vremenska složenost algoritma je $O(n^3)$ jer sadrži 3 ugniježđene „for“ petlje.

```
struct_matrica* pomnozi_matrice(struct_matrica* matrica1, struct_matrica* matrica2)
{
   if((matrica1->br_stupaca)!=(matrica2->br_redaka))
   {
      printf("mnozenje nije moguće");
      return NULL;
   }
   else
   {
      int i,j,k;
      float temp=0;//služi za zbrajanje umnoška pojedinih elemenata
      struct_matrica* matrica_umnoska;
      matrica_umnoska=alokacija_matrice(matrica1->br_redaka, matrica2->br_stupaca);
      for (i=0; i<(matrica1->br_redaka); i++){
               for(j=0; j<(matrica2->br_stupaca); j++){
            for(k=0; k<(matrica2->br_redaka); k++){
                     temp+=((matrica1->matrica[i][k])*(matrica2->matrica[k][j])); }
```

```
            matrica_umnoska->matrica[i][j]=temp;
            temp=0;
        }
    }
    return matrica_umnoska;}}
```

4.6.Determinanta

4.6.1.O determinantama

[4] Determinanta realne kvadratne matrice je funkcija det: $\mathbf{M}_n \to \mathbb{R}$. Determinantu matrice **A** označavamo s det **A** ili |**A**| ili $\begin{vmatrix} a_{11} & \dots & a_{1n} \\ a_{n1} & \dots & a_{nn} \end{vmatrix}$

Definiramo ju induktivno po redu *n* matrice.

Za *n*=1 i $\mathbf{A} = [a_{11}]$ je det **A** = a_{11}.

Za n=2 i $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$,

det **A**= $\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} := \boldsymbol{a_{11}a_{22} - a_{21}a_{12}}$.

Za matrice višeg reda determinanta se definira (i može računati) razvojem po bilo kojem retku ili stupcu, na način

$$\det \mathbf{A} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}.$$

Općenito se determinanta reda n dobije pomoću determinanti reda n-1 tako da se zbroje (ili oduzmu) produkti elemenata nekog retka ili stupca s determinantama matrica koje se dobiju uklanjanjem retka i stupca u kojem se taj element nalazi.“

Iz iznad navedenog se može zaključiti da se svaka determinanta reda *n* dobije zbrojem za red nižih , *n* determinanti *n*-1. Tj. determinanta se računa rekurzivno dok se ne dođe do determinante matrice reda 1.

Iz toga slijedi da je vremenska složenost algoritma za računanje determinanti

$$n \cdot (n-1) \cdot (n-2) \dots \big(n - (n-1)\big) = n! \,.$$

4.6.2.Svojstva determinanti

1. Determinanta matrice je jednaka determinanti transponirane matrice

$\det \mathbf{M} = \det \mathbf{M}^T$

2. Determinanta mijenja predznak sa svakom zamjenom svoja dva retka ili stupca.

3. Zajednički faktor nekog stupca ili retka se može izvući kao skalar koji množi determinantu.

4. Determinanta je jednaka nuli kada su svi elementi nekog retka ili stupca matrice jednaki nuli.

5. Determinanta matrice je jednaka nuli ako matrica sadrži dva jednaka ili proporcionalna stupca

6. Determinanta matrice ostaje ista ako se nekom retku ili stupcu matrice pribroji neki drugi redak ili stupac pomnožen skalarom.

7. Determinanta trokutaste matrice je jednaka umnošku elemenata matrice na dijagonali

8. Binet-Cauchyjev teorem: „Umnožak determinanti dviju matrica jednak je determinanti umnoška dviju matrica. Matrice **M** i **N** su istoga reda. $\det \mathbf{M} \cdot \det \mathbf{N} = \det (\mathbf{M} \cdot \mathbf{N})$ [4]

4.6.3. Programsko rješenje determinante i vremenska složenost algoritma

Determinanta se u programskom rješenju računa pomoću rekurzivne funkcije i vremenska složenost je $O(n!\, n^2)$ gdje je n red matrice.

Programsko rješenje:

```
float determinanta_matrice_rekurzija(float** matrica, int red_matrice)
{
   float determinanta=0;
   float predznak=1;
   float **pmatrica;

   if(red_matrice==1)
   {
      return matrica[0][0];
   }
```

```
    else
    {
        int i;
        for(i=0; i<red_matrice; i++) {
            pmatrica=podmatrica(matrica, red_matrice, 0, i);
            determinanta+=predznak*(matrica[0][i]*determinanta_matrice_rekurzija(pmatrica,
red_matrice-1));
            predznak=-1*predznak;
        }
        return determinanta;
    }
}
```

4.7.Adjungirana matrica

4.7.1.O adjungiranoj matrici

Adjungirana matrica **A*** matrice **A** je matrica kod koje na mjestu *i, j* stoji algebarski komplement elementa a_{ji}. Tj. $\mathbf{A}^* \coloneqq (-\mathbf{1})^{\boldsymbol{i+j}}\mathbf{M}_{\boldsymbol{ji}}$, gdje je $\mathbf{M}_{\boldsymbol{ji}}$ determinanta elementa podmatrice elementa a_{ji}.

4.7.2 Programsko rješenje adjungirane matrice i vremenska složenost algoritma

Budući da adjungirana matrica treba naći determinantu podmatrice za svaki element matrice, njena vremenska složenost je $O(n^2 * (n-1)!) = O(n * n!)$

```
struct_matrica* adjungiraj(struct_matrica* matrica1)
{
    int i, j;
    float s=-1;
    struct_matrica* adjungirana_matrica;
    adjungirana_matrica=alokacija_matrice(matrica1->br_redaka, matrica1->br_stupaca);
```

```
    for (i=0; i<(matrica1->br_redaka); i++)
    {
        for (j=0; j<(matrica1->br_stupaca); j++)
        {
            adjungirana_matrica->matrica[i][j]=pow(s,(i+j))*determinanta_matrice_rekurzija(podmatrica(matrica1->matrica, matrica1->br_redaka, i, j), matrica1->br_redaka-1);
        }
    }

    adjungirana_matrica=transponiraj(adjungirana_matrica);

    return adjungirana_matrica;
}
```

4.8. Inverzna matrica

4.8.1. O inverznoj matrici

Neko je **A** kvadratna matrica reda *n* i neka postoji matrica **X** takva da je **XA**=**AX**=**I**. Tada se može reći da je **X** inverzna matrica matrice **A i** obilježava se s $\mathbf{A}^{-1}$. [3]
Inverzna matrica se može naći Gaussovim postupkom ili pomoću Cramerovog pravila. Cramerovo pravilo je zgodno za traženje inverza matrice 2. ili 3. reda, dok je za matrice viših redova preporučljivo koristiti Gaussov postupak.

Cramerovo pravilo glasi $\mathbf{A}^{-1} = \frac{1}{\det \mathbf{A}} \cdot \mathrm{adj}\mathbf{A}$ [3]

Gaussov postupak se provodi pomoću proširene matrice oblika [$\mathbf{A}|\mathbf{I}$] gdje se pomoću elementarnih transformacija dođe do proširene matrice [$\mathbf{I}|\mathbf{A}^{-1}$]. [4]

Programska rješenja za determinantu i adjungiranu matricu poznata su iz prethodnih potpoglavlja.

4.8.2. Programsko rješenje Cramerovog pravila i vremenska složenost algoritma

Budući da Cramerovo pravilo glasi $\mathbf{A}^{-1} = \frac{1}{\det\mathbf{A}} \cdot \text{adj}\mathbf{A}$, mogu se iskoristiti programska rješenja iz prethodnih potpoglavlja pa se tako dobije:

```
struct_matrica* inverz(struct_matrica* matrica1)
{
   float a=*(determinanta_matrice(matrica1));
   if(a==0) // Ukoliko je determinanta različita od 0, matrica nema inverz
   {
      printf("nema inverz");
      return NULL;
   }
   struct_matrica* invertirana_matrica=adjungiraj(matrica1);
   pomnozi((1.00/(a)), invertirana_matrica); //dijelimo članove adjungirane matrice s determinantom
   return invertirana_matrica;
}
```

Pri čemu je vremenska složenost algoritma jednaka vremenskoj složenosti algoritma za računanje adjungirane matrice $O(n * n!)$.

4.8.3. Programsko rješenje Gaussovog postupka i vremenska složenost algoritma

Gaussov postupak ima vremensku složenost algoritma $O(n^3)$ jer unutar sebe sadrži tri ugniježđene „for“ petlje. Za matrice reda $n \geq 3$ je brži od Cramerovog pravila.

Programsko rješenje:

```
struct_matrica* inverz_Gauss(struct_matrica* M){
   int nema_inverz=0; /Logička oznaka kako bi došlo do prekida programa ukoliko nema inverz/
   float mnozitelj; /*Služi za pohranu vrijednosti kojom se množe elementi matrice*/
struct_matrica* inverzM=alokacija_matrice(M->br_redaka, M->br_stupaca);
for (int i=0; i<inverzM->br_redaka; i++){
   for (int j=0; j<inverzM->br_stupaca; j++){
         if(i==j) inverzM->matrica[i][j]=1.00;
         else inverzM->matrica[i][j]=0.00;
   }
}/*Deklariramo inverznu matricu, alociramo memoriju za nju i popunjavamo ju kao jediničnu matricu*/
for (int i=0; i<M->br_redaka; i++){
/* „For“ petlja za dijagonalne elemente matrice*/
   if(M->matrica[i][i]==0){
/*dio koda koji vrši zamjenu 2 retka ukoliko je dijagonalni element jednak nuli.*/
         for(int k=i+1; k<M->br_redaka; k++){
            if(M->matrica[k][i]){
         /*dio koda koji provjerava postoji li redak s kojim je moguće izvršiti zamjenu*/
               zamijeni_retke(M->matrica, i, k);
               zamijeni_retke(inverzM->matrica, i, k);
               nema_inverz=0;
               break;
            }
         else nema_inverz=1; /*ukoliko ne postoji redak s kojim s može izvršiti zamjena, matrica nema inverz*/
```

```
            }
        }
        if(nema_inverz) break; /*Izvođenje koda se prekida ukoliko matrica nema inverz*/

if((M->matrica[i][i])){/*Provjerava je li element na glavnoj dijagonali različit od nule*/
            if(M->matrica[i][i]!=1.00){/*Provjerava je li element na glavnoj dijagonali različit od 1*/
                float djeljitelj=M->matrica[i][i];
            for(int j=0; j<M->br_redaka; j++){
             M->matrica[i][j]=M->matrica[i][j]/djeljitelj;
             inverzM->matrica[i][j]=inverzM->matrica[i][j]/djeljitelj;
/*Dijeli sve elemente u retku s vrijednošću na glavnoj dijagonali kako bi se dobilo element jednak 1 na glavnoj dijagonali*/
                        }
            }
for (int k=0; k<(M->br_redaka);k++){ /*Dio koda koji poništava sve elemente u stupcu osim onog koji se nalazi na glavnoj dijagonali*/
             if(k!=i){
            mnozitelj=M->matrica[k][i];
             for (int j=0; j<(M->br_stupaca); j++){
                    M->matrica[k][j]-=mnozitelj*(M->matrica[i][j]);
                    inverzM->matrica[k][j]-=mnozitelj*(inverzM->matrica[i][j]);
             }
        }
}
            }
```

```
}
if(nema_inverz)
   return NULL;
};
ispis_matrice(inverzM);
return inverzM; /*vraća inverznu matricu*/
}
```

4.9. Rang matrice

4.9.1. O rangu matrice

Primjenom elementarnih transformacija zamjene redaka, množenja retka skalarom različitim od 0 i dodavanja retka nekom drugom retku matricu se može svesti na jedinstveni reducirani oblik. [4]

Tu podrazumijevamo sljedeći oblik matrice:
Prvi ne-nul element svakog retka iznosi 1. Svi elementi u stupcu tog stožernog elementa jednaki su nuli.
Svi retci koji sadrže samo nul elemente nalaze se iza onih redaka koji sadrže barem jedan ne-nul element.
Svaki naredni stožer (gledajući po retcima) nalazi se desno (u retku s većim indeksom) od prethodnog stožera: ako stožer u retku i_1 leži u stupcu j_1, a stožer u retku $i_2 > i_1$ leži u stupcu j_2, tada je $j_2 > j_1$.

[4]

Rang matrice je broj ne-nul redaka u reduciranom obliku matrice. Označavamo ga obično slovom r i pišemo r(**A**)=*r*. Ako je matrica **A** tipa $m \times n$, tada je uvijek $r(\boldsymbol{A}) \leq \min\{m, n\}$. Kvadratna matrica koja ima rang jednak svome redu je u reduciranoj formi jedinična matrica.

4.9.2. Programsko rješenje ranga matrice i vremenska složenost algoritma

Vremenska složenost algoritma za rang matrice je $O(n^3)$ jer sadrži 3 ugniježđene „for“ petlje.

Programsko rješenje:

```
int rang_matrice(struct_matrica *M){
   int rang_n; /*varijabla u koju se sprema maksimalni mogući rang*/
   int rang; /*varijabla u koju se sprema stvarni rang*/
   float mnozitelj;
if ((M->br_redaka)<(M->br_stupaca)){
        rang_n=M->br_redaka; /*Najveći mogući rang matrice je jednak najmanjoj dimenziji matrice r(A) ≤ min {m,n}*/
}
else rang_n=M->br_stupaca;
rang=rang_n; /*Inicijaliziramo rang matrice*/
for (int i=0; i<rang_n; i++){ /*Petlja pomoću koje se provjerava glavna dijagonala matrice*/
if (M->matrica[i][i]){/*Ako je element na glavnoj dijagonali matrice različit od nule, njim se poništavaju svi elementi u stupcu ispod njega
   for (int k=i+1; k<(M->br_redaka);k++){
      mnozitelj=(M->matrica[k][i])/(M->matrica[i][i]);
      for (int j=0; j<(M->br_stupaca); j++){
            M->matrica[k][j]-=mnozitelj*(M->matrica[i][j]);
      }
   }
}
else {  /*Slučaj kada je element na glavnoj dijagonali jednak nuli.*/
      int nula_element=1; /*postavlja se varijabla koja pamti jesu li svi elementi stupca nula*/
   for (int k=i+1; k<(M->br_redaka); k++){/*Vrši se provjera postoji li redak ispod trenutnog elementa matrice na glavnoj dijagonali koji ima ne-nul element na odgovarajućem mjestu.*/
      if (M->matrica[k][i]){ /*ukoliko postoji, poziva se funkcija za zamjenu redaka*/
```

```
            zamijeni_retke(M->matrica, i, k);
            printf("\n prolaz kroz potpetlju br. %d \n", i+1);
            nula_element=0; /*Vrijednost varijable se mijenja jer postoji ne-nul element*/
            break; /*Prekida izvođenje petlje jer je pronađen redak za zamjenu*/}     }
    if(nula_element) rang--;} /*Ukoliko nije pronađen ne-nul element, rang matrice se umanjuje*/
printf("\n");
ispis_matrice(M);}
return rang;}
```

5. ZAKLJUČAK

Cilj završnoga rada bio je izraditi programsko rješenje za osnovne algoritme linearne algebre u programskom jeziku C.

U svakom je poglavlju najprije pružena matematička teorijska podloga o svakome algoritmu kako bi se dobio uvid u to što rade, a zatim su izložena rješenja u programskom jeziku C. Razlog zašto je izabran programski jezik C je njegova jednostavnost prilikom obrade podataka male veličine, brzina prilikom izvršavanja algoritama visoke vremenske složenosti i jer programsko rješenja nije zahtijevalo uređeno korisničko sučelje. Potrebno je zamijetiti da su određena programska rješenja možda mogla biti odrađena i s manjom složenošću, ali mi je cilj bio izraditi programska rješenja koja odgovaraju teoriji koja je pružena u knjigama linearne algebre. Programsko rješenje se može koristiti kao konzolna aplikacija za izvršavanje osnovnih operacija linearne algebre nad vektorima i matricama. U aplikaciji nije moguće ulančavanje više operacija, nego se unesu 2 matrice ili 2 vektora nakon čega se izračunaju rezultati operacija između njih.

LITERATURA

[1] Encyclopaedia Britannica, C computer programming language https://www.britannica.com/technology/C-computer-programming-language

[2] Jurak, M., Programski jezik C, 2003. https://web.math.pmf.unizg.hr/~singer/Prog_Add/c.pdf

[3] Mitrinović, D.S., Mihailović, D., Vasić, P.M., Linearna algebra Polinomi Analitička geometrija, Građevinska knjiga, Beograd 1985.

[4] Elezović N., Aglić A., *Linearna algebra zbirka zadataka*, Element, Zagreb, 2003.

SAŽETAK

Ovim radom su obrađeni osnovni algoritmi linearne algebre I u programskom jeziku C. Rad započinje osnovnim informacijama o programskom jeziku C,
zatim se pruža teoretska podloga o vektorima, nakon čega slijedi teorijska dio o osnovnim operacijama s vektorima popraćen odgovarajućim programskim rješenjima.
Nakon vektora na red dolaze matrice, za koje se također iznosi teorijska podloga, a nakon toga slijede pojedinačno osnovne operacije s matricama i njima pripadajuća programska rješenja u programskom jeziku C.

Ključne riječi: linearna algebra, osnovni algoritmi, programski jezik C, vektori, matrice

ABSTRACT

Basic algorithms of linear algebra in programming language C

This paper deals with basic linear algebra algorithms in C programming language.

The paper begins with basic information about the C programming language,

then a theoretical background on the vectors is provided, followed by a theoretical section on basic vector followed by the appropriate software solutions.

The vectors are followed by matrices, for which the theoretical background is also given, followed individually by basic matrix operations and their corresponding programming solutions in the C programming language.

Keywords: Linear algebra, basic algorithms, programming language C, vectors, matrix

ŽIVOTOPIS

Matej Jukić rođen je 23. prosinca 1996. u Žepču. Pohađao je OŠ „Fra Grga Martić“ u Ozimici i Lugu od 2003. do 2011. i Osnovnu glazbenu školu „Katarina Kosača Kotromanić“ u Žepču od 2009. do 2014. Nakon završene osnovne škole pohađao je opću gimnaziju u Katoličkom školskom centru „Don Bosco“ Žepče, gdje je maturirao 2015. godine i iste godine upisao preddiplomski smjer računarstva na Elektrotehničkom fakultetu Osijek (današnji Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek).

PRILOZI

CD:

1. Završni rad „Osnovni algoritmi linearne algebre u programskom jeziku C.docx“
2. Završni rad „Osnovni algoritmi linearne algebre u programskom jeziku C.pdf“
3. Izvorni kod programskog rješenja