SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

**Iva Kustura**

ITERATIVNE METODE ZA RJEŠAVANJE SUSTAVA LINEARNIH JEDNADŽBI

pROJEKT

Varaždin, 2024.

**1. Uvod**

Iterativne metode za numeričko rješavanje sustava linearnih jednadžbi koriste početne pretpostavke i iterativno poboljšavaju približno rješenje dok ne postignu zadovoljavajuću točnost. Ove metode posebno su korisne za velike i rijetke matrice gdje bi direktne metode bile neefikasne ili nepraktične. Njihova prednost je što ne zahtjevaju puno memorije jer ne pohranju cijele matrice u memoriji. Efikasnije su za određene tipove matrica jer smanjuju broj potrebnih operacija u usporedbi s direktnim metodama. Iterativne metode su podijeljene na dvije skupine : stacionarne i nestacionarne. U skupinu stacionarnih metoda pripadaju Jacobijeva metoda, Gauss-Seidelova, SOR (eng. Successive Over-Relaxtion) metoda. Nestacionarne metode obuhvaćaju metodu konjugiranih gradijenata, GMRES ( eng. Generalized Minimal Residual Method ) i BiCGSTAB (eng. Bi-Conjugate Gradient Stabilized Method). U nastavku ću detaljnije objasniti razliku između Jacobijeve i Gaus-Seidelove metode te u programskom jeziku Python implementirati navedene metode. Programski kod ću tesirati na primjeru rijetke matrice i usporediti dobivena rješenja s rješenjima dobivenim LU faktorizacijom. [1]

**2. Jacobijeva metoda**

Jacobijeva metoda je stacionarna iterativna metoda za rješavanje sustava linearnih jednadžbi koja koristi dijagonalne elemente matrice za iteracije. Jednostavna je za implementaciju, a posebno je korisna za dijagonalno dominantne matrice. Iterativni postupak temelji se na rješavanju svake jednandžbe sustava linearnih jednadžbi za jednu varijablu , dok se pretpostavlja da su ostale varijable fiksirane na vrijednost iz prethodne iteracije.

U prvom koraku postavlja se početno rješenje . Često se bira . Drugi korak je iteracija gdje je :

* Svaki i-ti element vektora x :
* nova vrijednost i-te komponente vektora rješenja u (k+1)-toj iteraciji, a vrijednost j-te komponente iz prethodne (k-te) iteracije. [1]

Iteracije se zaustavljaju kada je norma razlike između vektora rješenja u dvije uzastopne iteracije manja od zadane tolerancije ili kada je postignut maksimalni broj iteracija.

Prednost ove metode je jednostavna implementacija i razumijevanje. Izračun svake komponente može biti izvršen neovisno od ostalih komponenti unutar iste iteracije te je zbog toga pogodna za paralelizaciju i izvođenje na računalima koja podržavaju paralelnu obradu. Zbog toga ju često nazivamo „Metoda istovremenih promjena“ jer nije važan redoslijed izvođenja jednadžbi. Nedostatak je relativno spora konvergencija, posebno za sustave koji nisu dijagonalno dominantni. Može zahtijevati veliki broj iteracija za postizanje zadovoljavajuće točnosti.

**2.1. Progamska implementacija**

U nastavku se nalazi implementacija Jacobijeve metode. A predstavlja matricu koeficijenata sustava linearnih jednadžbi, a vektor konstanti je *b.* Početni vektor rješenja je *x0.* Kriterij zaustavljanja je varijabla *tol*. Maksimalni broj iteracija definiran je u varijabli *max\_iterations*. Prva petlja prolazi maksimalni broj iteracija koji je definiran u metodi. Sljedeća petlja prolazi n puta, odnosno broj redaka u matrici. Za izračun nove vrijednosti vrijedi

.

Najprije se definira varijabla *sum\_* za svaku iteraciju s vrijednošću *b[i]* koja je konstanta s desne strane jednadžbe i-tog reda sustava . Sljedeća petlja iterira kroz sve elemente reda *i* u matrici *A.* Provjeravamo jednakost *i != j* kako bismo preskočili elemente koji se nalaze na dijagonali, odnosno elemente *A[i][i].* Ako se element ne nalazi na dijagonali, oduzima se njegova vrijednost pomnožena s *x[j]* od varijable *sum\_.* Nakon što prođemo sve elemente reda *x\_new[i]* dobivamo dijeljenjem varijable *sum\_* s elementom dijagonale. Kada se izvrše obje petlja računa se *max\_diff* odnosno maksimalna apsolutna razlika između novih i starih vrijednosti. U slučaju da je njezina vrijednosst manja od zadane tolerancije, iteracije se zaustavljaju i vraća se rješenje jer je to jedan od kriterija zaustavljanja. Inače, iteracije traju maksimalan broj iteracija definiran u parametru funkcije *jacobi().* Rješenje vraća broj iteracija i trenutnu vrijedost polja *x\_new*.

def jacobi(A, b, x0, tol=1e-10, max\_iterations=1000):

n = len(A)

x = x0[:]

x\_new = x0[:]

for k in range(max\_iterations):

for i in range(n):

sum\_ = b[i]

for j in range(n):

if i != j:

sum\_ -= A[i][j] \* x[j]

x\_new[i] = sum\_ / A[i][i]

max\_diff = max(abs(x\_new[i] - x[i]) for i in range(n))

if max\_diff < tol:

return x\_new, k

x = x\_new[:]

return x, k

A = [

[4, -1, 0, 0],

[-1, 4, -1, 0],

[0, -1, 4, -1],

[0, 0, -1, 3]

]

b = [15, 10, 10, 10]

x0 = [0, 0, 0, 0]

x, iterations = jacobi(A, b, x0)

print("Rješenje:", x)

print("Broj iteracija:", iterations)

Izvođenjem navedenog programskog koda dobivamo rješenje :

*Rješenje:*

*[4.999999999979709, 4.999999999966491, 4.999999999962085,4.999999999970897]*

*Broj iteracija: 29 .*

Na istom primjeru matrice, izvršit ću Gaus-Sieldovu metodu koja bi u usporebi s Jacobijevom trebala imati manji broj iteracija.

**3. Gaus-Seidelova metoda**

Gaus-Seidelova metoda je stacionarna iterativna metoda rješavanja linearnih jednadžbi koja u odnosu na Jacobijevu metodu ima manji broj iteracija jer koristi prethodno izračunate aproksimacije vektora u računanju sljedećih komponenti vektora. Slično kao i Jacobijeva metoda, Gauss-Seidelova metoda iterativno se približava rješenju, ali koristi unaprijeđene vrijednosti varijabli čim postanu dostupne u istoj iteraciji. Konvergiranje je brže pa se iz tog razloga dolazi do rješenja korištenjem manjeg broja iteracija. [2]

Za i-ti redak sustava linearnih jednadžbi vrijedi :

Ako je vrijedi , tada možemo ovaj izraz preurediti i računati vrijednost :

Koriste se nove vrijednosti za sve , odnosno već ažurirane vrijednosti unutar iste iteracije. Za sve se koriste stare vrijednosti iz prethodne iteracije jer još nisu izračunate nove vrijednosti. Na ovaj način se povećava efikasnost zbog manjeg broja iteracija. Osim toga, manji je broj memorijskih zahtjeva jer metoda zahtjeva manje prostora za pohranu. U nekim slučajevima se događa da je potreban veliki broj iteracija, primjerice kada nije zadovoljen uvjet dijagonalne dominantnosti (vrijednost dijagonalnog elementa u retku veća ili jednaka sumi apsolutnih vrijednosti svih ostalih elemenata u retku). Još jedan nedostatak je što ovisi o ažuriranim vrijednostima pa metoda nije lako paralelirizana. Spora konvergencija se događa ako matrica nije pozitivno definirana ili nije dijagonalno dominantna.

**3.1. Programska implementacija**

Programska implementacija Gauss-Seidelove metode najprije stvara kopiju početnog vektora rješenja koja se ažurira kroz iteracije kako bismo dobili konačno rješenje *x*. Glavna petlja iterira maksimalni broj iteracija, a varijabla *x\_old* označava kopiju trenutnog rješenja prije ažuriranja *x*. Varijabla *sum* predstavlja sumu s desne strane jednadžbe. Unutarnja petlja iterira kroz varijable kako bi se izračunao . Navedeni umnožak se oduzima od sume u svakoj iteraciji, a nova vrijednost se dobiva djeljenjem sume i dijagonalnog elementa (). Završetkom dvije unutrašnje petlje, provjerava se maksimalna razlika između novih i starih vrijednosti varijabli . U slučaju da je maksimalna razlika manja od zadane tolerancije, metoda se zaustavlja jer je rješenje dovoljno precizno. Vraća se vektor rješenja i broj iteracija. Ako se zadana preciznost ne zadovolji prije izvršenja maksimalnog broja iteracija, tada se vraća rješenje nakon što se dosegne maksimalni broj iteracija.

def gauss\_seidel(A, b, x0, tol=1e-10, max\_iterations=1000):

    n = len(A)

    x = x0[:]

    for k in range(max\_iterations):

        x\_old = x[:]

        for i in range(n):

            sum\_ = b[i]

            for j in range(n):

                if i != j:

                    sum\_ -= A[i][j] \* x[j]

            x[i] = sum\_ / A[i][i]

        max\_diff = max(abs(x[i] - x\_old[i]) for i in range(n))

        if max\_diff < tol:

            return x, k

    return x, k

A = [

    [4, -1, 0, 0],

    [-1, 4, -1, 0],

    [0, -1, 4, -1],

    [0, 0, -1, 3]

]

b = [15, 10, 10, 10]

x0 = [0, 0, 0, 0]

x, iterations = gauss\_seidel(A, b, x0)

print("Rješenje:", x)

print("Broj iteracija:", iterations)

Zbog usporedbe efikasnosti metoda, korištena je ista matrica *A* i *b* kao u prethodnoj metodi.Korištenjem Gauss-Sedelove metode, dobiveni rezultat je :

*Rješenje:*

*[4.999999999995111, 4.9999999999964935, 4.999999999998362, 4.999999999999454]*

*Broj iteracija: 16*

Možemo uočiti kako je broj iteracija u ovoj metodi gotovo dvostruko manji jer metoda koristi dostupne vrijednosti iz prethodne iteracije u svakoj novoj iteraciji. Zaključak je da je Gauss-Seidelova metoda na ovakvim jednostavnim matricama efikasnija jer prolaskom kroz manji broj iteracija vraća zadovoljavajući rezultat.

**4. Metoda LU faktorizacije**

LU faktorizacija je metoda koja se koristi za rješavanje sustava linearnih jednadžbi razlažući matricu A na umnožak dviju matrica – donje trokutasta i gornje trokutasta matrica. Metoda *foward\_substitution* rješava sustav , a metoda *backward\_substitution* rješava sustav . Složenost ove LU faktorizacije je općenito gdje je *n* dimenzija kvadratne matrice *A*. Metoda LU faktorizacije ima veću složenost od prethodno navedenih metoda, pa bi prema tome trebala vraćati preciznija rješenja. [3]

def forward\_substitution(L, b):

    n = len(L)

    y = [0.0] \* n

    for i in range(n):

        sum\_ = sum(L[i][j] \* y[j] for j in range(i))

        y[i] = (b[i] - sum\_) / L[i][i]

    return y

def backward\_substitution(U, y):

    n = len(U)

    x = [0.0] \* n

    for i in range(n - 1, -1, -1):

        sum\_ = sum(U[i][j] \* x[j] for j in range(i + 1, n))

        x[i] = (y[i] - sum\_) / U[i][i]

    return x

def lu\_solve(A, b):

    L, U = lu\_decomposition(A)

    y = forward\_substitution(L, b)

    x = backward\_substitution(U, y)

    return x

A = [

    [4, -1, 0, 0],

    [-1, 4, -1, 0],

    [0, -1, 4, -1],

    [0, 0, -1, 3]

]

b = [15, 10, 10, 10]

x = lu\_solve(A, b)

print("Rješenje:", x)

Metodom LU faktorizacije dobivamo sljedeće rješenje :

*Rješenje: [5.0, 5.0, 4.999999999999999, 4.999999999999999]*

Možemo primjetiti da su prve dvije varijable izračunate u potpunosti točno, dok je približno rješenje druge dvije varijable jako blizu potpuno točnog rješenja ( točno rješenje iznosi [*5.0, 5.0, 5.0, 5.0*] ). U ovoj metodi nisu korištene iteracije jer je naglasak na složenosti algoritma, a ne na pojedinačnim iteracijama kao u prethodnim metodama.

**5. Usporedba dobivenih rješenja i testiranje na primjeru rijetke matrice**

Na primjeru jednostavne matrice možemo zaključiti da je efikasnija Gauss-Seidelova metoda koja brže konvergira i tako dolazi na brži način do rješenja. Što se tiče točnosti, budući da je rezultat [ 5.0, 5.0, 5.0, 5.0 ], preciznija je ona metoda čiji su rezultati bliže toj vrijednosti. Za svaku varijablu iz ovog primjera, preciznije rezultate ( odnosno bliže 5.0) ima Jacobijeva metoda. Primjerice, prva varijabla :

Ipak, najpreciznija metoda je u ovom slučaju metoda LU faktorizacije jer za dvije varijable daje u potpunosti točno rješenje. Međutim, ona se razlikuje po tome što nije iterativna metoda pa rješenja računa na drugi način.

Provedeno testiranje na primjeru rijetke matrice (80x80) dalo je pozitivne i vrlo slične rezultate u sve tri metode. Najprije je definirana matrica *A* i *b* generiranjem nasumičnih brojeva, a zatim je vraćen rezultat iz svih matrica. Budući da je matrica relativno velika, uzet ćemo prvih nekoliko rješenja :

*Jacobijeva metoda –*

*Rješenje: [3.66008454, 4.64079022, 4.90356586, 4.97397518, 4.9928409, 4.99789583…*

*Gauss-Seidelova metoda –*

*Rješenje: [3.66019755 4.64094085 4.90372901 4.97414248 4.99300958 4.99806496 …*

*LU faktorizacija –*

*Rješenje: [3.6602540378443864, 4.641016151377546, 4.903810567665797, 4.974226119285642, 4.993093909476771, 4.998149518621444…*

Dobivena rješenja se razlikuju nakon treće ili četvrte decimale što možemo smatrati zadovoljavajućim. Moramo uzeti u obzir da postoji mogućnost da generirana matrica nije pogodna za izračun iterativnim metodama te da one svakako računaju približnu vrijednost jer se nakon zadovoljene preciznosti zaustavljaju.

Budući da je rješenje dobiveno LU faktorizacijom točnije, usporedit ćemo nekoliko vrijednosti iz rezultata s rezultatima dobivenim Jacobijevom i Gauss-Seidelovom metodom. Tablica prikazuje razliku između pojedinog rezultata dobivenog LU faktorizacijom i Jacobijevom odnosno Gauss-Seidelovom metodom. Možemo zaključiti da veću preciznost ima Gauss-Seidelova metoda u svakom slučaju. Obzirom da su precizniji rezultati bili i na primjeru jednostavnije matrice, Gauss-Seidelova metoda je bolji izbor u ovakvom slučaju gdje nije potrebna paralelizacija.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Jacobijeva metoda | 0,000169183844386 | 0,000225931377546 | 0,000244707665797 | 0,000250939285642 |
| Gauss-Seidelova | **0,000056487844386** | **0,000075301377546** | **0,000081557665797** | **0,000083639285642** |

Tablica 1. Prikaz rezultata

**6. Zaključak**

Gauss-Seidelova metoda je vrlo slična Jacobijevoj, ali u navedenim slučajevima daje bolje rješenje kroz manje iteracija. Jacobijeva metoda sporije konvergira jer se ne koriste podatci iz prethodnih iteracija pa je zbog toga prikladna u primjeni paralelizacije jer su varijable međusobno neovisne. Takav slučaj nije testiran, pa je zbog toga Gauss-Seidelova metoda bolji izbor. S druge strane, paralelizaciju je teško primijeniti koristeći metodu Gauss-Seidelove metode. Što se tiče metode LU faktorizacije, nakon izračuna dekompozicije rješenje se dobiva brzo. LU faktorizacija nije iterativna metoda te je u ovom slučaju korištena kako bismo mogli usporediti Jacobijevu i Gauss-Seidelovu metodu. Što se tiče izbora metode, trebali bismo uzeti u obzir specifičnosti problema kao što su veličina matrice, uvjeti dijagonalne dominantnosti, potreba za brzim rješenjem i slično. Temeljem definiranih specifičnosti lakše je odlučiti koja će metoda biti bolji izbor. Obje se mogu izvršavati nad velikim matricama, razlika je u preciznosti i načinu izvedbe.

**7. Literatura**

[1] I. Lučić, „Iterativne metode“, diplomski rad, Sveučilište J.J. Strossmayera, Odjel za matematiku,Osijek, 2012.. Preuzeto 14.6. s :

<https://www.mathos.unios.hr/~mdjumic/uploads/diplomski/LUč04.pdf>

[2] B. Bošnjak, „Iterativne metode za rješavanje linearnih sustava“, završni rad, Sveučilište J.J. Strossmayera, Odjel za matematiku,Osijek, 2020.. Preuzeto 14.6. s :

<https://www.mathos.unios.hr/~mdjumic/uploads/diplomski/BOŠ21.pdf>

[3] Z.Bujanović, „Gaussove eliminacije i LU faktorizacija“, (bez dat.), Preuzeto 14.6. s :

<https://web.math.pmf.unizg.hr/nastava/unm/skripta/html/02a_Gaussove_eliminacije_i_LU.html>