

Gausova eliminacija

Tema prva dva predavanja ovog kursa je rešavanje sistema linearnih jednačina.

Postoje mnogi realni problemi koji se svode na rešavanje SLAJ.

Neki njih, kao što su rangiranje stranica na Internetu pomoću algoritma *PageRank* koji koristi *Google* ili određivanje pozicije pomoću *GPS*, pokazujemo u zavisnosti od vremena na kraju ovog ili sledećeg predavanja.

Algoritam Gausove eliminacije

Pre nego što stignemo do primena detaljno ćemo kroz kod objasniti kako funkcioniše algoritam Gausove eliminacije.

Gausova eliminacija je ustvari samo upošteenje načina na koji smo rešavali sisteme jednačina još od osnovne škole.

Recimo da imamo sledeći sistem:

$$\begin{aligned}4x_1 - 3x_2 + x_3 &= -8 \\ -2x_1 + x_2 - 3x_3 &= -4 \\ x_1 + x_2 + 2x_3 &= 3\end{aligned}$$

Prisetimo se na koji način smo ti pokušavali da rešimo sisteme jednačina u osnovnoj i srednjoj školi.

Pokušavali samo na neki način da sistem svedemo na jednu jednačinu sa jednom nepoznatom, pa da onda tu nepoznatu odredimo daljenjem i zamenimo je u neku od prethodnih jednačina da dobijemo drugu nepoznatu i tako redom.

Gausova eliminacija nije ništa drugo nego samo sistematizacija tog postupka.

Krećemo tako što nam je cilj da eliminišemo promenljivu x_1 iz druge jednačine.

U prvom koraku prvu jednačinu množimo sa $\frac{1}{4}$ i onda je dodamo drugoj:

$$\begin{aligned}4 \cdot \frac{1}{2}x_1 - 3 \cdot \frac{1}{2}x_2 + 1 \cdot \frac{1}{2}x_3 &= -8 \cdot \frac{1}{2} \\ -2x_1 + x_2 - 3x_3 &= -4\end{aligned}$$

Prvu jednačinu prepisujemo bez promene jer će nam u tom obliku biti potrebna u narednim koracima.

Na taj način ne menjamo rešenje jer množenje jednačine konstantom ne utiče na rešenje.

$$\begin{aligned}4x_1 - 3x_2 + x_3 &= -8 \\ 0 - \frac{1}{2}x_2 - \frac{5}{2}x_3 &= -8\end{aligned}$$

Uklonili smo x_1 iz druge jednačine, sada ga uklanjamo i iz treće.

U sledećem koraku prvu jednačinu množimo sa $-\frac{1}{4}$ i onda je dodamo trećoj:

$$\begin{aligned}4 \cdot -\frac{1}{4}x_1 - 3 \cdot -\frac{1}{4}x_2 + 1 \cdot -\frac{1}{4}x_3 &= -8 \cdot -\frac{1}{4} \\ x_1 - \frac{3}{4}x_2 + 2x_3 &= 3\end{aligned}$$

Prvu jednačinu prepisujemo bez promene jer će nam u tom obliku biti potrebna u narednim koracima.

$$\begin{aligned}4x_1 - 3x_2 + x_3 &= -8 \\ 0 - \frac{1}{2}x_2 - \frac{5}{4}x_3 &= 5 \\ 0 - 1 + 3x_3 &= 9\end{aligned}$$

Rezultat prethodnih operacija:

$$\begin{aligned}4x_1 - 3x_2 + x_3 &= -8 \\ 0 - \frac{1}{2}x_2 - \frac{5}{2}x_3 &= -8 \\ 0 - \frac{1}{4}x_2 + \frac{7}{4}x_3 &= 5\end{aligned}$$

Još uvek nemamo jednu jednačinu sa jednom nepoznatom, pa nastavljamo tako što koristimo drugu jednačinu da izbacimo x_2 iz treće.

Drvu jednačinu da pomnožimo sa $-2 \cdot \frac{1}{4} = -\frac{1}{2}$ i onda je dodamo trećoj:

$$\begin{aligned}4x_1 - 3x_2 + x_3 &= -8 \\ 0 - \frac{1}{2}x_2 - \frac{5}{2}x_3 - \frac{1}{2}x_2 - \frac{5}{2}x_3 &= -8 - \frac{1}{2} \\ 0 - \frac{1}{4}x_2 + \frac{7}{4}x_3 &= 5\end{aligned}$$

Drugu jednačinu prepisujemo bez množenja jer će nam u tom obliku biti potrebna u narednim koracima.

$$\begin{aligned}4x_1 - 3x_2 + x_3 &= -8 \\ 0 - \frac{1}{2}x_2 - \frac{5}{2}x_3 &= -8 \\ 0 - 0 + 3x_3 &= 9\end{aligned}$$

Sada smo sistem "konačno" dobili jednu jednačinu sa jednom nepoznatom.

$$\begin{aligned}4x_1 - 3x_2 + x_3 &= -8 \\ -\frac{1}{2}x_2 - \frac{5}{2}x_3 &= -8 \\ 3x_3 &= 9\end{aligned}$$

Upravo smo završili prvu fazu Gausove eliminacije koja se zove *eliminacija unapred*.

Sada vrlo lako možemo da izračunamo x_3 :

$$x_3 = \frac{9}{3} = 3$$

Zamenjujemo x_3 u drugu jednačinu i izračunavamo x_2 :

$$\begin{aligned}-\frac{1}{2}x_2 + \frac{5}{2} \cdot 3 &= -8 \\ -\frac{1}{2}x_2 &= -8 - \frac{15}{2} \\ -\frac{1}{2}x_2 &= -\frac{1}{2} \\ x_2 &= 1\end{aligned}$$

Kao poslednji korak zamenjujemo x_2 i x_3 u prvu jednačinu i izračunavamo x_1 :

$$\begin{aligned}4x_1 - 3 \cdot 1 + 3 &= -8 \\ 4x_1 &= -8 \\ x_1 &= -2\end{aligned}$$

Upravo smo završili drugu fazu Gausove eliminacije koja se zove *zamena unazad* i time rešili sistem.

Šta mislite da li je prethodni postupak primenljiv na sisteme veće od 3x3?

Haide da pokušamo da napišemo kod za metod koji smo koristili na prethodnom primeru pa da vidimo da li može da se primeni na bilo koji sistem.

Kod ćemo razdvojiti na manje celine.

Prva celina je množenje prve vrste nekom vrednosti p i dodavanje drugoj vrsti.

```
In [1]: A=[4,-3,1,-2,1,-3,1,-1,2]
b=[-8,-4,3]
A =
  4   -3    1
 -2    1   -3
  1   -1    2

b =
 -8
 -4
  3

In [2]: [n,m]=size(A);
p=m/4;
for j=1:n
    A(2,j)=A(2,j) + A(1,j)*p;
end
A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  1.00000   -1.00000    2.00000
```

Na koji način smo napisali prethodni kod:

- #### Imamo jednu promenljivu j koja je brojač za kolone.
- #### Menjamo sve elemente druge vrste, zato imamo $A(2,j)$.
- #### Na svaki element druge vrste dodajemo odgovarajući element prve vrste koji je u istoj koloni (koloni j) "znad njega".
- #### Pre nego što uradimo dodavanje, element prve vrste množimo sa p .

Na elemente treće vrste dodajemo elemente prve vrste pomnožene sa p .

```
In [8]: [n,m]=size(A);
p=m/4;
for j=1:n
    A(3,j)=A(3,j) + A(1,j)*p;
end
A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
 -1.00000    0.75000    1.75000

In [4]: A=[4,-3,1,-2,1,-3,1,-1,2]
[n,m]=size(A);
p=m/4;
for k=1:3
    for i = (k+1):3
        p = -A(1,k)/A(k,k);
        for j=i:n
            A(i,j)=A(i,j) + A(k,j)*p;
        end
    end
end
A =
  4   -3    1
 -2    1   -3
  1   -1    2

A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000   -0.25000    1.75000
```

Da li nam je potreban if ili možda postoji neki bolji način da odredimo p ?

```
In [20]: A=[4,-3,1,-2,1,-3,1,-1,2]
[n,m]=size(A);
for i = 2:3
    p = -A(1,i)/A(1,1);
    zaokružimo;
    A(i,j)=A(i,j) + A(1,j)*p;
end
end
A =
  4   -3    1
 -2    1   -3
  1   -1    2

A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000   -0.25000    1.75000
```

Upoštavamo sad kod tako da pored prve vrste množimo i drugu vrstu odgovarajim brojem i dodajemo na treću.

```
In [21]: A=[4,-3,1,-2,1,-3,1,-1,2]
[n,m]=size(A);
for k=1:2
    for i = (k+1):m
        p = -A(1,k)/A(k,k);
        for j=i:n
            A(i,j)=A(i,j) + A(k,j)*p;
        end
    end
end
A =
  4   -3    1
 -2    1   -3
  1   -1    2

p =
  0.50000
p =
 -0.25000
A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000   -0.25000    1.75000

p =
 -0.50000
A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000    0.00000    3.00000
```

Upoštavamo kod tako da možemo da radimo sa bilo kojom kvadratnom matricom $n \times n$.

Pošto je vektor b deo sistema dodajemo i njega u kod.

```
In [41]: A=[4,-3,1,-2,1,-3,1,-1,2]
b=[-8,-4,3]
[n,m]=size(A);
for k=1:n-1
    for i = (k+1):n
        p = -A(1,k)/A(k,k);
        for j=i:n
            A(i,j)=A(i,j) + A(k,j)*p;
        end
    end
end
A =
  4   -3    1
 -2    1   -3
  1   -1    2

b =
 -8
 -4
  3

A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000   -0.25000    1.75000

b =
 -8
 -8
  5

A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000    0.00000    3.00000

b =
 -8
 -8
  9
```

Sada ste naučili kako funkcioniše prva faza Gausove eliminacije koja se zove *eliminacija unapred*.

Cilj ove faze je ono što smo upravo uradili, a to je svođenje matrice A na gornju trougaonu.

Prelazimo na drugu (i poslednju) fazu koji se naziva *zamena unazad*.

Kao prvi korak odredićemo poslednje x , tj. x_3 .

Upošticećmo odmah kod i pišemo x_n jer je n broj kolona matrice A , tj. broj promenljivih.

```
In [46]: x=zeros(n,1);
x(n)=b(n)/A(n,n)
x =
  0
  0
  0
  3

In [47]: A =
  4   -3    1
 -2    1   -3
  1   -1    2

A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000    0.00000    3.00000

b =
 -8
 -8
  9

A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000    0.00000    3.00000

b =
 -8
 -8
  9

x =
 -2
  1
  3
```

Određujemo sada x_2 tako što zamenjujemo x_3 u drugu jednačinu.

$$\begin{aligned}-\frac{1}{2}x_2 + \frac{5}{2} \cdot 3 &= -8 \\ x_2 &= (-8 + \frac{15}{2}) \cdot -\frac{2}{1}\end{aligned}$$

```
In [48]: s=A(2,3)*x(3);
x(2)=(b(2)-s)/A(2,2)
x =
  0
  1
  3
```

Određujemo sada x_1 tako što zamenjujemo x_2 i x_3 u prvu jednačinu.

$$\begin{aligned}4x_1 - 3 \cdot 1 + 3 &= -8 \\ x_1 &= (-8 - (-3 \cdot 1 + 3)) \cdot \frac{1}{4}\end{aligned}$$

```
In [49]: s=A(1,2)*x(2)+A(1,3)*x(3);
x(1)=(b(1)-s)/A(1,1)
x =
 -2
  1
  3
```

Upoštavamo kod za x_1 .

```
In [50]: i=1;
s=0;
for j=(i+1):n
    s = s + A(i,j)*x(j);
end
x(i)=(b(i)-s)/A(i,i)
x =
 -2
  1
  3
```

Upoštavamo kod za sve promenljive.

```
In [52]: x=zeros(n,1);
for i=n:-1:1
    s=0;
    for j=(i+1):n
        s = s + A(i,j)*x(j);
    end
    x(i)=(b(i)-s)/A(i,i);
end
x =
 -2
  1
  3
```

Spajamo kod za eliminaciju unapred i zamenu unazad i pišemo funkciju za Gausovu eliminaciju.

```
In [53]: function x=gauss(A,b)
[n,m]=size(A,b)
for k=1:n-1
    for i = (k+1):n
        p = -A(1,k)/A(k,k);
        for j=i:n
            A(i,j)=A(i,j) + A(k,j)*p;
        end
    end
    b(i)=b(i) + b(k)*p;
end
end
A =
  4.00000   -3.00000    1.00000   -8.00000
  0.00000   -0.50000   -2.50000   -8.00000
  0.00000    0.00000    3.00000    9.00000

x =
 -2    1    3

In [54]: A=[4,-3,1,-2,1,-3,1,-1,2]
b=[-8,-4,3]
x=gauss(A,b)
A =
  4   -3    1
 -2    1   -3
  1   -1    2

b =
 -8   -4    3

A =
  4.00000   -3.00000    1.00000
  0.00000   -0.50000   -2.50000
  0.00000    0.00000    3.00000

b =
 -8   -8    9

x =
 -2
  1
  3
```

```
In [55]: A1=rand(5,8);
b1=rand(5,1);
x1=gauss(A1,b1)
A =
  0.88145    0.06545    0.66551    0.50140    0.42535    0.33813    0.35191    0.10188
  0.00000    0.58827    0.52406    0.33977    0.73756    0.56746    0.30932    0.87478
  0.00000    0.00000   -0.07908    0.17524   -0.24774    0.48218    0.19212    0.08764
  0.00000    0.00000    0.00000   -0.97014   -0.00940   -1.36006   -0.39232   -0.78129
  0.00000    0.00000    0.00000    0.00000   -0.45239    1.11872    0.06708    0.21936
 -0.00000    0.00000    0.00000    0.00000    0.00000    0.53327   -0.26236    0.87582
 -0.00000    0.00000    0.00000    0.00000    0.00000    0.00000   -1.36540    3.58156
  0.00000    0.00000    0.00000    0.00000   -0.00000    0.00000    0.00000   -0.64609

b =
  0.623916
  0.010216
  0.280552
 -0.843006
  0.734044
  1.283478
  5.086318
 -1.204327

x1 =
  0.511308
 -2.557122
  0.868970
 -0.981201
 -0.749261
 -0.081751
  1.164339
  1.864023
```

Pre nego što pređemo na sledeću celinu dat je malo optimizovani kod za Gausovu eliminaciju. Ovaj kod nije od značaja za vaše razumevanje algoritma već demonstrira kako možemo efikasno da koristimo vektorske operacije koje nude jezici kao što su MATLAB, Octave, Python itd. Kod nije deo obaveznog gradiva dat je samo informativno.

```
In [17]: function x = gauss_vect(A,b)
[n,n] = size(A);
Aaug = zeros(n,n);
Aaug(1:n,n)=A;
Aaug(:,n+1)=b;

for k=1:n-1
    for i=k+1:n
        m=-Aaug(i,k)/Aaug(k,k);
        Aaug(i,:)=Aaug(i,:)+m*Aaug(k,:);
    end
end
Aaug = zeros(1,n);
for i=n:-1:1
    x(i)=(Aaug(i,n)-dot(Aaug(i,1:n),x))/Aaug(i,i);
end
endfunction
```

```
In [18]: x = gauss_vect(A,b)
Aaug =
  4.00000   -3.00000    1.00000   -8.00000
  0.00000   -0.50000   -2.50000   -8.00000
  0.00000    0.00000    3.00000    9.00000

x =
 -2    1    3
```

Napomena: u ispisu matrice Aug, nakon pokretanja funkcije gauss, poslednja kolona je vektor b pa zato deluje kao da rezultat nije gornja trougaona matrica, tj. da poslednja vrsta ima 2 elementa.

Problemi sa originalnim (naivnim) algoritmom Gausove eliminacije

Pored očiglednog problema deljenja nulom do koga dolazi kada određujemo vrednost $m = \frac{a_{ij}}{a_{ii}}$, a vrednost $a_{ii} = 0$ što se u naivnoj GE ne proverava, postoje i malo suptilniji problemi o kojima ćemo se baviti u nastavku.

Ako pogledamo sledeći grafik funkcije $f(x) = \frac{1}{x}$, vidimo da kada su vrednosti x male, npr. manje od 1, za male promene x imamo velike promene $f'(x)$ dok za veće vrednosti x to ne važi. Ova osobina operacije deljenja nam je veoma značajna kao potencijalni izvor problema sa Gausovom eliminacijom.

```
In [56]: x=linspace(0.1,10,1000);
plot(x,1./x,"linewidth",10,"color",[1 0 0]);
xlabel("x");
ylabel("1/X");
```



```
In [20]: 1/0.0049
1/0.005
ans = 204.08
ans = 200
```

```
In [21]: 1/2300
1/2350
ans = 0.00043478
ans = 0.00042553
```

Povez ćemo sada operaciju deljenja i ograničen kapacitet računara za smeštanje brojeva i time pokazati šta su problemi sa postupkom GE.

Ograničen kapacitet računara za smeštanje brojeva može proizvesti gubitak informacija tokom računskih operacija. Na primer, ako pomnožimo sledeća dva broja koji mogu da se smeštaju na računar, rezultat je takav da ne može ceo biti smešten već moramo da ga zaokružimo.

12.123456789 * 14.123456789 = 171.225118092750190521

```
In [57]: format long
12.123456789 * 14.123456789
ans = 171.2251180927502
```

Sličnu situaciju imamo i sa deljenjem. Na primer ako delimo 56.985/101.53 dobijamo sledeći rezultat pomoću digitrona sa beskonačnom preciznošću:

0.56126268090909040457692723245119660603310351620210775140352605141133753570373288683147839077415542204274589641

Očigledno je da moramo da rezultat moramo da zaokružimo.

Gubitak informacija je naročito izražen kada su brojevi koji učestvuju u računskoj opreciji različitog reda veličine, kao u sledećem primeru.

1463.0345 + 0.000123456789 = 1463.034623456789

```
In [58]: 1463.0345 + 0.000123456789
ans = 1463.034623456789
```

Ako bi prethodni zbir zbog ograničenog kapaciteta računara morali da zaokružimo na recimo 9 cifara izgubili bi sve osim jedne cifre drugog sabirka, tj. drugi sabirak bi skoro bio beznačajan.

U Gausovoj eliminaciji prilikom svođenja na gornju trougaonu matricu ponavljamo sledeća dva koraka:

- $m = \frac{a_{ij}}{a_{ii}}$
- $a_{i,j} = a_{i,j} - m * a_{k,j}$

Do gubitka informacija može doći ako se u umanjenik i umanjiilac u koraku 2. razlikuju po redu veličine, što se može dogoditi ako je n jako mali ili jako veliki broj. Vrednost n može biti jako velika u slučaju da je n_1 jako mali broj, tj. blizu 0. Iz tog razloga trebalo bi da pokušamo da izbegnemo deljenje jako malim brojevima tokom Gausove eliminacije. To postižemo upotrebom pivotainga.

Pivoting

- #### Iz do sada navedenog može se zaključiti da se problem u koraku 2. može rešiti ili ublažiti ako pivot element (sa kojim se u postupku eliminacije unapred vrši deljenje) ima što veću moguću vrednost.
- #### Iz oblasti algebre znamo da promena rasporeda jednačina i promenljivih (premeštanje vrsta i kolona matrice sistema) ne menja rešenje sistema.
- #### Ovu činjenicu možemo da iskoristimo da bi prilikom eliminacije svake promenljive na mesto pivot element postavili najveći mogući element po apsolutnoj vrednosti.
- #### Postupak postavljanja najvećeg mogućeg elementa po apsolutnoj vrednosti na mesto pivot naziva se pivoting.
- #### Ako najveći mogući element tražimo u celoj matrici sistema, tačnije u delu koji nije anuliran u postupku eliminacije unapred, onda izvršavamo kompletno pivoting.
- #### Parcijalni pivoting predstavlja efikasniju varijantu pivotinga pri kojoj novi pivot element tražimo samo u koloni u kojoj se nalazi promenljiva koja se trenutno eliminiše. U praksi se najčešće koristi parcijalni pivoting jer je efikasniji uz prihvatljiv pad kvaliteta rešenja.

```
In [59]: [max,loc] = max([1,2,5,3])
max = 5
loc = 3
```



```
[60]: function x=gauss_with_pivoting(A,b)
[n,m]=size(A);
for k=1:n-1
    %pronalazimo i maksimalni element po apsolutnoj vrednosti u koloni k
    disp('Pre pivotinga')
    A
    [max_val,loc] = max(abs(A(k:n,k)));
    %loc se racuna apsolutno za deo kolone koji prosledujemo, a nama treba relativno u odnosu na celu k
    %loc sta bi bilo kada bi ostavili samo loc?
    loc = k+loc-1;
    temp = A(k,:);
    A(k,:) = A(loc,:);
    A(loc,:) = temp;
    temp = b(k);
    b(k) = b(loc);
    b(loc) = temp;
    disp('Posle pivotinga')
    A
    b
    for i = (k+1):n
        p = -A(i,k)/A(k,k);
        for j=i:n
            A(i,j)=A(i,j) + A(k,j)*p;
            end
        b(i)=b(i) + b(k)*p;
    end
end
A
b
x=zeros(n,1);
for i=n:-1:1
    s=0;
    for j=i:n
        s = s + A(i,j)*x(j);
    end
    x(i)=(b(i)-s)/A(i,i);
end
endfunction
```

```
In [62]: format short
A2=[5,1,2,3;-2,2,2,-3;0,1,1,4;6,2,2,4];
b2=[-1,-2,-1]';
x2 = gauss_with_pivoting(A2,b2)
```

```
Pre pivotinga
A =

     5     0     2     3
    -2     2     2    -3
     0     1     1     4
     6     2     2     4

b =

     1
    -1
     2
     1

Posle pivotinga
A =

     6     2     2     4
    -2     2     2    -3
     0     1     1     4
     5     0     2     3

b =

     1
    -1
     2
     1

Pre pivotinga
A =

     6.00000     2.00000     2.00000     4.00000
     0.00000     2.66667     2.66667    -1.66667
     0.00000     1.00000     2.00000     4.00000
     0.00000    -1.66667     0.33333    -0.33333

b =

     1.00000
    -0.66667
     2.00000
    -0.16667

Posle pivotinga
A =

     6.00000     2.00000     2.00000     4.00000
     0.00000     2.66667     2.66667    -1.66667
     0.00000     1.00000     2.00000    -1.37500
     0.00000    -1.66667     0.33333    -0.33333

b =

     1.00000
    -0.66667
     2.25000
    -0.25000

Pre pivotinga
A =

     6.00000     2.00000     2.00000     4.00000
     0.00000     2.66667     2.66667    -1.66667
     0.00000     0.00000     2.00000     4.62500
     0.00000     0.00000     2.00000    -1.37500

b =

     1.00000
    -0.66667
     2.25000
    -0.25000

Posle pivotinga
A =

     6.00000     2.00000     2.00000     4.00000
     0.00000     2.66667     2.66667    -1.66667
     0.00000     0.00000     2.00000     4.62500
     0.00000     0.00000     0.00000     4.62500

b =

     1.00000
    -0.66667
    -0.15541
     0.20946
     0.48649

x2 =

    -0.17568
    -0.15541
     0.20946
     0.48649
```

Ako pogledamo poslednju matricu pred pivoting za naš primer vidimo da bi bez pivotinga u tom slučaju imali deljenje nulom.

Kao deo informativnog gradiva dat je i optizovan kod uz pivoting.

```
In [27]: function x = gauss_vect_with_pivoting(A,b)
[n,m] = size(A);
Aaug = zeros(n,m);
Aaug(1:n,1:n)=A;
Aaug(:,n+1)=b;

for k=1:n-1
    %pronalazimo maksimalni element po apsolutnoj vrednosti u koloni k
    disp('Pre pivotinga')
    Aaug
    [max_val,loc] = max(abs(Aaug(k:n,k)));
    %loc se racuna apsolutno za deo kolone koji prosledujemo, a nama treba relativno u odnosu na celu
    %kolonu. Sta bi bilo kada bi ostavili samo loc?
    loc = k+loc-1;
    temp = Aaug(k,:);
    Aaug(k,:) = Aaug(loc,:);
    Aaug(loc,:) = temp;
    disp('Posle pivotinga')
    Aaug
    for i=k+1:n
        m=-Aaug(i,k)/Aaug(k,k);
        Aaug(i,:)=Aaug(i,:)+m*Aaug(k,:);
    end
end

x=zeros(1,n);
for i=n:-1:1
    x(i)=(Aaug(i,n+1)-dot(Aaug(i,1:n),x))/Aaug(i,i);
end
endfunction
```

Da li Gausova eliminacija može da rezultuje pogrešnim rešenjem?

- ##### Sam postupak (algoritam) Gausove eliminacije ne sadrži grešku, ali GE može vratiti pogrešno rešenje ako vrednosti A ili b odstupaju od onih koje su originalno zadate.
- ##### Zašto bi bilo odstupanje u vrednostima?
- ##### Jedan od banalnih razloga može biti da je neko prilikom unosa podataka pogrešno.
- ##### Drugi, mnogo realniji i češći razlog je da su vrednosti A i b takve da ne mogu da se potpuno tačno reprezentuju na računaru koji radi sa ograničenom preciznošću.

Logično pitanje je na koji način možemo da proverimo da li je GE vratio pogrešno rešenje.

- ##### Prvi odgovor bio bi da jednostavno ubacimo dobijeno rešenje x u sistem i proverimo jednakost, tj. da pomnožimo x sa A i proverimo da li važi $Ax = b$.

U nastavku ćemo pokazati da ostatak $r = Ax - b$ nije pouzdana mera tačnosti kada brojeve reprezentujemo sa ograničenom preciznošću. Kod primera u nastavku računske operacije se izvršavaju na računaru koji može da smešti samo tri značajne cifre broja. To je namerno urađeno da bi se ilustrovalo problem sa nepouzdanošću ostataka r . Savremeni računari mogu da smešte 16 značajnih cifara upotrebom tipa *double*, odnosno i njihov kapacitet je isto ograničen samo sa većim brojem cifara.

```
In [63]: function x=gauss_lp(A,b)
[n,m]=size(A);

for k=1:n-1
    for j=(k+1):n
        f_full_prec=-A(i,k)/A(k,k) %ispis dodet da bi se ilustrovalo gubitak informacija zbog zaokru
        f_round_to_sig_fig=-A(i,k)/A(k,k),3)
        for i=j:k:n
            axf_full_prec=f*A(k,j) %ispis dodet da bi se ilustrovalo gubitak informacija zbog zaokru
            A(i,j)=round_to_sig_fig(A(i,j)+round_to_sig_fig(f*A(k,j),3),3)
        end
    end
    bxf_full_prec=f*b(k) %ispis dodet da bi se ilustrovalo gubitak informacija zbog zaokruživanja
    b(b)=round_to_sig_fig(b(b)+round_to_sig_fig(f*b(k),3),3)
end
end
A
for i=n:-1:1
    s=0;
    for j=i+1:n
        s = round_to_sig_fig(s + round_to_sig_fig(A(i,j)*x(j),3),3);
    end
    x(i)=round_to_sig_fig(b(i)-s)/A(i,i),3);
end
endfunction

function r_val=round_to_sig_fig(val, sig_figs)
temp = mat2str(val, sig_figs);
r_val = eval(temp);
endfunction
```

```
In [64]: A=[0.641,0.242;0.321,0.122]
b=[0.683,0.444]

A =

     0.64100     0.24200
     0.32100     0.12200

b =

     0.68300
     0.44400
```

```
In [65]: x_low_pr = gauss_lp(A,b)

f_full_prec = -0.50078
f = -0.50100
axf_full_prec = -0.32114
A =

     0.64100     0.24200
     0.00000     0.12200

axf_full_prec = -0.12124
A =

     0.64100     0.24200
     0.00000     0.00100

bxf_full_prec = -0.44238
b =

     0.8830000
     0.0020000

A =

     0.64100     0.24200
     0.00000     0.00100

x_low_pr =

     0.62200     2.00000
```

```
In [66]: residual_lp=round_to_sig_fig(round_to_sig_fig(A*x_low_pr', 3)-b,3)
residual_full_prec=Ax_low_pr'-b

residual_lp =

     0
     0

residual_full_prec =

    -0.00029800
    -0.00033800
```

Ako radimo u ograničenoj preciznosti od 3 značajne cifre, ostatak je 0 što ukazuje na to da je naše rešenje tačno. Međutim tačno rešenje se već na prvoj značajnoj cifri razlikuje od našeg rešenja. Kao što ćete videti u nastavku.

```
In [67]: x_full_prec = gauss_with_pivoting(A,b)

Pre pivotinga
A =

     0.64100     0.24200
     0.32100     0.12200

b =

     0.88300
     0.44400

Posle pivotinga
A =

     0.64100     0.24200
     0.32100     0.12200

b =

     0.88300
     0.44400

A =

     0.64100     0.24200
     0.00000     0.00081

b =

     0.8830000
     0.0018112

x_full_prec =

     0.53462
     2.23269
```

```
In [9]: x_full_prec - x_low_pr'

ans =

    -0.087385
     0.232692
```

Prethodni primer pokazuje nepouzdanost ostataka za određivanje greške. Zato ćemo u nastavku predavanja pokazati na koji način možemo da procenimo grešku. Ne moramo znati tačnu grešku ako smo uspeali da je procenimo na malu vrednost tj. nije nam važno koliko je tačno greška mala, već da je mala. Za procenu nam je od velikog značaja kondicioni broj matrice.

Kondicioni broj funkcije

- ##### Kondicioni broj funkcije $f(x)$ meri koliko promene ulaza x utiču na promene izlaza y .
- ##### Veliki kondicioni broj znači da male promene ulaza daju velike na promene izlaza.
- ##### Tada kažemo da je funkcija loše uslovljena.
- ##### Ako ste ikad malo pomerili slavinu pod tušem, a voda je od hladne prešla na jako vrelu – to je loše uslovljena funkcija!

Kondicioni broj matrice

- ##### U kontekstu rešavanja sistema linearnih jednačina kondicioni matrice A meri koliko male promene vektora b ili matrice A utiču na rešenje x .
- ##### Kao što smo već pomenuli male promene mogu biti posledica lošeg unosa podataka ili ograničenog kapaciteta računara za smeštanje brojeva.
- ##### Za izračunavanje kondicionog broja matrice značajan nam je koncept norme matrice koji ćemo ukratko objasniti u nastavku.

Norma vektora i matrice

- ##### Norma matrice je relana vrednost koju dodeljujemo matrici.
- ##### Postoje različite norme matrice koje se razlikuju po načinu na koji na osnovu matrice izračunavamo jednu relanu vrednosti.
- ##### Detalji vezani za normu matrice predmet su linearne algebre, dok ćemo na ovom kursu pokazati samo neke od poznatih načina za određivanje norme matrice.

Maksimum zbira apsolutnih vrednosti vrsta matrice:

$$\|A\|_{\infty} = \max_i \sum_{j=1}^n |A_{ij}|$$

```
In [68]: B=[1,2;-1,-3.9999]
norm(B, inf)

B =

     1.0000     2.0000
    -1.0000    -3.9999

ans =

     4.9999
```

Maksimum zbira apsolutnih vrednosti kolona matrice:

$$\|A\|_1 = \max_j \sum_{i=1}^n |A_{ij}|$$

```
In [69]: B=[1,2;-1,-3.9999]
norm(B,1)

B =

     1.0000     2.0000
    -1.0000    -3.9999

ans =

     5.9999
```

Sada kada smo objasnili normu matrice, daćemo definiciju kondicionog broja matrice:

$$\text{cond}(A) = \|A\| \|A^{-1}\|$$

```
In [70]: cond(B)

ans =

     3.3698
```

Kondicioni broj matrice i Gausova eliminacija

Kondicioni broj matrice i vektor b

- ##### Ako u sistemu $Ax = b$, b nije tačno (zbog grešaka zaokruživanja npr.), koliko će se x razlikovati od tačnog rešenja?
- ##### Ako umesto b imamo $b + \Delta b$ onda važi sledeća nejednakost:

$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta b\|}{\|b\|}$$

- ##### Dakle, relativna (procentualna) greška rešenja ograničena je ne samo sa relativnom greškom vektora b već i sa kondicionim brojem matrice A .
- ##### Kondicioni broj se na neki način može smatrati "pojačivačem" greške vektora.
- ##### Ako kondicioni broj matrice A veliki postoji mogućnost da će male promene vektora b rezultovati velikim promenama rešenja x , kao što ćete videti na sledećem primeru.
- ##### Termin "postoji mogućnost" upotrebljen je namerno jer je relativna greška rešenja samo ograničena pomoću prethodne formule (upotrebljen je znak nejednakosti, a ne znak jednakosti).

Kondicioni broj matrice i matrica A

- ##### Ako umesto A imamo neku matricu E koja se razlikuje od A (zbog grešaka zaokruživanja npr.) onda važi sledeća nejednakost:

$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|E\|}{\|A\|}$$

Primer rešavanja loše uslovljenog sistema pomoću Gausove eliminacije

$$\begin{aligned} -x_1 + 2x_2 &= 3 \\ -x_1 + 2.1x_2 &= 3 \end{aligned}$$

Grafik loše uslovljenog sistema. Da li primećujete nešto specifično na njemu?

```
In [71]: figure inline -w 600 -h 600
x1= linspace(-10,5,100);
x2=(3-x1)/2;
plot(x1,x2,'linewidth',10,'color',[1 0 0])
hold on;
x2=(3-x1)/2.1;
plot(x1,x2,'linewidth',10,'color',[0 0 1])

xlabel('x1');
ylabel('x2');
axis([-10,10,-4,5]);
hold off;
```



```
In [72]: A=[2,-1;2,-1];
A =

     2.0000    -1.0000
     2.1000    -1.0000

In [73]: cond(A)
log10(cond(A))

ans =

    104.09
     2.0174

In [74]: A=[2,-1;2,-1];
b1=[3,3];
b1=gauss(A)
x1=gauss_with_pivoting(A,b1);

Pre pivotinga
A =

     2.0000    -1.0000
     2.1000    -1.0000

b =

     3     3

Posle pivotinga
A =

     2.1000    -1.0000
     2.0000    -1.0000

b =

     3     3

A =

     2.10000    -1.00000
     0.00000    -0.04762

b =

     3.00000     0.14286
```

Menjamo vrednost druge komponente vektora b

```
In [76]: A=[2,-1;2,-1];
b2=[3,3.1];
x2=gauss_with_pivoting(A,b2)

Pre pivotinga
A =

     2.0000    -1.0000
     2.1000    -1.0000

b =

     3.0000     3.1000

Posle pivotinga
A =

     2.1000    -1.0000
     2.0000    -1.0000

b =

     3.1000     3.0000

A =

     2.10000    -1.00000
     0.00000    -0.04762

b =

     3.10000     0.047619

x2 =

     1.00000
    -1.00000
```

```
In [206]: A
b1
b2
x1
x2

A =

     2.0000    -1.0000
     2.1000    -1.0000

b1 =

     3     3

b2 =

     3.0000     3.1000

x1 =

    -2.1147e-15
    -3.0000e+00

x2 =

     1.00000
    -1.00000
```

Izračunavamo gornju granicu relativne greške rešenja

```
In [77]: cond(A)*norm(b1-b2)/norm(b1)

ans =

     2.4534

Pošto znamo rešenje, izračunavamo relativnu grešku rešenja
```

```
In [78]: norm(x1-x2)/norm(x1)

ans =

     0.74536

Relativna greška je preko 75%, ali je opet drastično manja nego gornje ograničenje od 245%.
```

Red veličine kondicionog broja i relativna greška

Postoji tvrdjenje (koje nećemo dokazivati) pomoću koga možemo da povežemo relativnu grešku i red veličine kondicionog broja:

Ako imamo dve vrednosti x i \tilde{x} , tada se one poklapaju u bar $m - 1$ značajnih cifara ako važi:

$$\frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \leq 5 \cdot 10^{-m}$$

To znači da bi rešenje dobijeno pomoću GE imalo bar jednu značajnu cifru od tačnog rešenja, relativna greška mora da bude manja od $5 \cdot 10^{-2} = 0.05$.

```
In [79]: cond(A)
log10(cond(A))

ans =

    104.09
     2.0174

Za naš primer vidimo da je kondicioni broj reda veličine takvog da odgovara približno broju 10 na stepen 2, što znači da naše rešenje je potencijalno netačno da se poklapa sa tačnim rešenjem ni u jednoj cifri.
```

Dakle, možemo da vidimo da nam vrednost $\log_{10}(\text{cond}(A))$ može poslužiti kao mera broja cifara tačnog rešenja koje možemo da izgubimo ako imamo greške u podacima prilikom rešavanja nekog sistema.

```
In [80]: m=1; %proveravamo da li imamo bar jednu istu cifru sta tačnim rešenjem
(norm(x1-x2)/norm(x1))<5*10^-m
ans =

     0

Pogledajmo sada jedan primer dobro uslovljenog sistema, tj. sistema kod koga matrica  $A$  ima malu vrednosti kondicionog broja.
```

$$\begin{aligned} 2x_1 - x_2 &= 3 \\ -2x_1 - x_2 &= 3 \end{aligned}$$

```
In [81]: figure inline -w 600 -h 600
x1= linspace(-10,5,100);
x2=(-3-x1);
plot(x1,x2,'linewidth',10,'color',[1 0 0])
hold on;
x2=(-3-x1);
plot(x1,x2,'linewidth',10,'color',[0 0 1])

xlabel('x1');
ylabel('x2');
axis([-10,10,-25,20]);
hold off;
```



```
In [82]: A=[2,-1;-2,-1];
cond(A)
log10(cond(A))

A =

     2     -1
    -2     -1

ans =

     2.0000
     0.30103
```

```
[83]: b1=[3,3];
      b2=[3,3.1];
      x1=gauss_with_pivoting(A,b1);
      x2=gauss_with_pivoting(A,b2);
```

Pre pivotinga

A =

2	-1
-2	-1

b =

3	3
---	---

Posle pivotinga

A =

2	-1
-2	-1

b =

3	3
---	---

A =

2	-1
0	-2

b =

3	6
---	---

Pre pivotinga

A =

2	-1
-2	-1

b =

3.0000	3.1000
--------	--------

Posle pivotinga

A =

2	-1
-2	-1

b =

3.0000	3.1000
--------	--------

A =

2	-1
0	-2

b =

3.0000	6.1000
--------	--------

```
In [84]: A
        b1
        b2
        x1
        x2

A =

     2     -1
    -2     -1

b1 =

     3     3

b2 =

     3.0000     3.1000

x1 =

     0
    -3

x2 =

    -0.025000
    -3.050000
```

```
In [85]: cond(A)*norm(b1-b2)/norm(b1)

ans = 0.047140
```

```
In [86]: norm(x1-x2)/norm(x1)

ans = 0.018634
```

```
In [87]: log10(cond(A))

ans = 0.30103
```

```
In [88]: m=2;
        [norm(x1-x2)/norm(x1)] < 5*10^-m

ans = 1
```

Ponekad je velika vrednost kondicionog broja rezultat veoma različitih opsega u kojima se nalaze vrednosti matrice sistema. U takvim slučajevima skaliranje vrednosti na isti opseg može da smanji kondicioni broj.

```
In [89]: A=[0.6,135.4;22.0,-17000]
        cond(A)

A =

     0.60000     135.40000
    22.00000   -17000.00000

ans = 21930.58677
```

```
In [90]: A1=[0.6/135.4,135.4/135.4;22.0/17000,-17000/17000]
        cond(A1)

A1 =

     0.0044313     1.0000000
     0.0012941    -1.0000000

ans = 349.32
```