Gausova eliminacija

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

Postoje mnogi realni problemi koji se svode ne rešavanje SLAJ.

Neke njih, kao što su rangiranje stranica na internetu pomoću algoritma *PageRank* koji koristi *Google* ili određivanje pozicije pomoću GPS, pokazaćemo 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 uopštenje načina na koji smo rešavali sisteme jedančina 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 mi 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 deljenjem 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 $\frac{2}{4} = \frac{1}{2}$ 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 konstanom 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:

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

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

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

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.

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

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

Drugu jednačinu prepisujemo bez množenja sa $-\frac{1}{2}$ 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*?

Hajde 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=2/4;
for j=1:n %imamo n elemenata u svakoj vrsti
    A(2,j)=A(2,j) + A(1,j)*p;
end
A
```

```
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) "iznad 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=-1/4;
for j=1:n
    A(3,j)=A(3,j) + A(1,j)*p;
end
A
```

```
A =

   4.00000  -3.00000   1.00000
   0.00000  -0.50000  -2.50000
  -1.00000   0.75000   2.75000
```

Spajamo prethodna dva isečka koda u jedan.

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

```
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(i,1)/A(1,1);
    for j=1:n
        A(i,j)=A(i,j) + A(1,j)*p;
    end
end
A
```

```
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 odgovarajućim 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 %koristimo prvu i drugu vrstu da eliminišemo x1 i x2
    for i = (k+1):3 %vrsta k koja se koristi za eliminaciju dodaje se na sve vrste posle nje do kraja,
 tj. petlja ide od k+1
        p = -A(i,k)/A(k,k);
        for j=1:n
            A(i,j)=A(i,j) + A(k,j)*p;
        end
    end
    A
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 *nxn*.

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 %koristimo sve vrste za radimo eliminaciju osim poslednje zato petlja ide do n-1
    for i = (k+1):n %menjamo vrednosti 3 na n da bi omogućili rad sa proizvoljnim matricama
        p = -A(i,k)/A(k,k);
        for j=1:n
            A(i,j)=A(i,j) + A(k,j)*p;
        end
        b(i)=b(i) + b(k)*p;
    end
    A
    b
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 svojđenje matrice A na gornju trougaonu.

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

Kao prvi korak odredićemo poslednje x, tj. x_3.

Upoštićemo 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

x =

   0
   0
   3
```

In [47]:
```
A
b
```

```
A =

   4.00000  -3.00000   1.00000
   0.00000  -0.50000  -2.50000
   0.00000   0.00000   3.00000

b =

  -8
  -8
   9
```

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 promenjlive.

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
```

```
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);
for k=1:n-1
    for i = (k+1):n
        p = -A(i,k)/A(k,k);
        for j=1: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 [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(8,8);
b1=rand(8,1);
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   0.48278   0.18212   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
   0.280652
  -0.843006
   0.734044
   1.283478
   5.086318
  -1.204327

x1 =

   0.511908
  -2.557122
   0.868970
  -0.981201
  -0.748261
  -0.081751
   1.164339
   1.864023
```

Pre nego što pređemo na sledeću celinu dat je malo optimizovaniji 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)
    [m,n] = size(A);
    Aaug = zeros(n,n);
    Aaug(1:n,1: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
    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
```

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
```

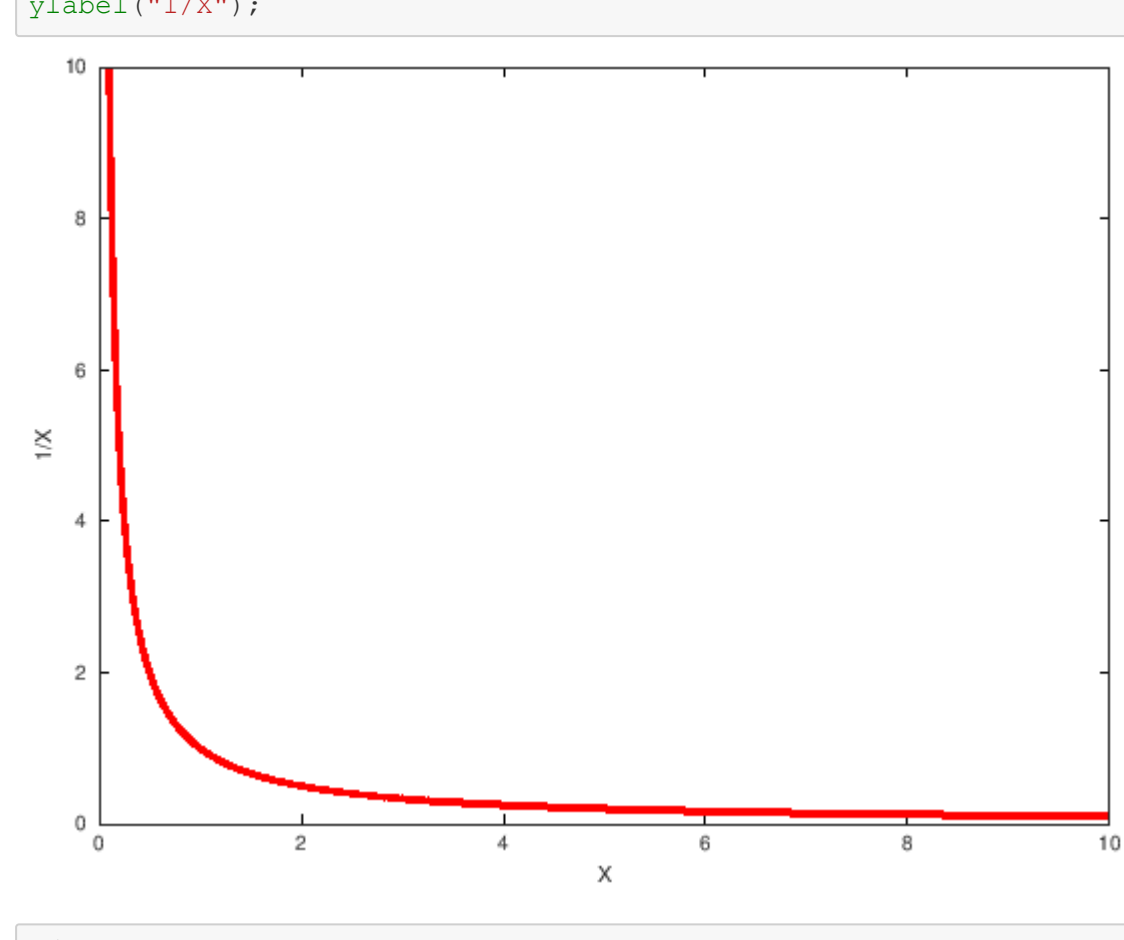
Napomena: u ispisu matrice Aug, nakon pokretanja funkcje gauss, poslednja kolona je vektor b pa zato deluje kao da rezultat nije gornja trougaona matrica, tj. da poslednja vrsta ima 2 elementa.

Problemi sa orignialnim (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 navinoj 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
```

Povezaćemo 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 smeste 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:

0.561262680980990840145769723234511966906333103516202107751403526051413375357037328868314783807741554220427459864

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:

1. $m = \frac{a_{i,j}}{a_{i,i}}$

2. $a_{i,j} = a_{i,j} - m * a_{k,j}$

Do gubitka informacija može doći ako se umanjenik i umanjilac u koraku 2. razlikuju po redu veličine, što se može dogoditi ako je m jako mali ili jako veliki broj. Vrednost m može biti jako velika u slučaju da je $a_{i,i}$ 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 pivotinga.

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 elementa postavli najveći mogući element po apsolutnoj vrednosti.
- #### Postupak postavljanja najvećeg mogućeg elementa po aspsulutnoj vrednosti na mesto pivota naziva se pivoting.
- #### Ako najveći mogući element tražimo u celoj matrici sistema, tačnije u delu koji nije anuliran u postupku elimancije unapred, onda izvršavamo kompletan pivoting.
- #### Pracijalni 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 najćešće koristi parcijalni pivoting jer je efikasniji uz prihvatljiv pad kvaliteta rešenja.

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

```
max =  5
loc =  3
```

In [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
```