NA05 Gaussova eliminacija

Ivan Slapničar

23. listopada 2018.

1 Gaussova eliminacija

1.1 Općenito

Sustav $Ax = b$ se rješava u tri koraka (*bez pivotiranja*):

1. $A = LU$ (LU rastav, $O(\frac{2}{3}n^3)$ operacija),
2. $Ly = b$ (donje trokutrasti sustav, n^2 operacija),
3. $Ux = y$ (gornje torkutasti sustav, n^2 operacija).

S pivotiranjem vrijedi

1. $PA = LU$,
2. $Ly = P^T b$,
3. $Ux = y$.

1.2 LU rastav

```
In [1]: function mylu{T}(A1::Array{T}) # Strang, str. 100
            A=deepcopy(A1)
            n,m=size(A)
            # Ovo prihvaća brojeve i blok-matrice
            U=map(Float64,[zero(A[1,1]) for i=1:n, j=1:n])
            L=map(Float64,[zero(A[1,1]) for i=1:n, j=1:n])
            for k=1:n
                L[k,k]=one(A[1,1])
                for i=k+1:n
                    L[i,k]=A[i,k]/A[k,k]
                    for j=k+1:n
                        A[i,j]=A[i,j]-L[i,k]*A[k,j]
                    end
                end
                for j=k:n
                    U[k,j]=A[k,j]
```

```
            end
        end
        L,U
    end
```

Out[1]: mylu (generic function with 1 method)

In [2]:
```
srand(123)
A=rand(6,6)
```

Out[2]:
```
6×6 Array{Float64,2}:
 0.768448  0.586022   0.865412  0.582318  0.20923   0.48
 0.940515  0.0521332  0.617492  0.255981  0.918165  0.790201
 0.673959  0.26864    0.285698  0.70586   0.614255  0.356221
 0.395453  0.108871   0.463847  0.291978  0.802665  0.900925
 0.313244  0.163666   0.275819  0.281066  0.555668  0.529253
 0.662555  0.473017   0.446568  0.792931  0.940782  0.031831
```

In [3]:
```
L,U=mylu(A);
```

In [4]:
```
L
```

Out[4]:
```
6×6 Array{Float64,2}:
 1.0       0.0         0.0         0.0        0.0     0.0
 1.22392   1.0         0.0         0.0        0.0     0.0
 0.877039  0.368849    1.0         0.0        0.0     0.0
 0.514613  0.289733   -0.471902    1.0        0.0     0.0
 0.407632  0.113088    0.0869892   0.215088   1.0     0.0
 0.862199  0.0484897   0.896224   -0.0434479  2.3274  1.0
```

In [5]:
```
U
```

Out[5]:
```
6×6 Array{Float64,2}:
 0.768448   0.586022   0.865412   0.582318  0.20923    0.48
 0.0       -0.665108  -0.441699  -0.456726  0.662085   0.202722
 0.0        0.0       -0.310382   0.363608  0.186542  -0.139531
 0.0        0.0        0.0        0.296226  0.591194   0.52933
 0.0        0.0        0.0        0.0       0.25212    0.20895
 0.0        0.0        0.0        0.0       0.0       -0.730114
```

In [6]:
```
L*U-A
```

Out[6]:
```
6×6 Array{Float64,2}:
 0.0  0.0   0.0          0.0  0.0  0.0
 0.0  0.0   0.0          0.0  0.0  0.0
 0.0  0.0  -5.55112e-17  0.0  0.0  0.0
 0.0  0.0   0.0          0.0  0.0  0.0
 0.0  0.0   0.0          0.0  0.0  0.0
 0.0  0.0   0.0          0.0  0.0  3.46945e-17
```

1.3 Trokutasti sustavi

In [7]:
```
function myU{T}(U::Array{T},b1::Array{T})
    b=deepcopy(b1)
    n=length(b)
    for i=n:-1:1
       for j=n:-1:i+1
            b[i]=b[i]-U[i,j]*b[j]
       end
        b[i]=b[i]/U[i,i]
    end
    b
end

function myL{T}(L::Array{T},b1::Array{T})
    b=deepcopy(b1)
    n=length(b)
    for i=1:n
        for j=1:i-1
            b[i]=b[i]-L[i,j]*b[j]
        end
        b[i]=b[i]/L[i,i]
    end
    b
end
```

Out[7]:
```
myL (generic function with 1 method)
```

In [8]:
```
b=rand(6)
```

Out[8]:
```
6-element Array{Float64,1}:
 0.900681
 0.940299
 0.621379
 0.348173
 0.570613
 0.203997
```

In [9]:
```
# Riješimo sustav koristeći ugrađenu funkciju
x=A\b
```

Out[9]:
```
6-element Array{Float64,1}:
  2.72418
  4.63586
 -3.5114
 -2.63338
 -0.195864
  1.46631
```

```
In [10]: # Riješimo sustav koristeći naše funkcije
         y=myL(L,b)

Out[10]: 6-element Array{Float64,1}:
           0.900681
          -0.162059
          -0.108779
          -0.119709
           0.257004
          -1.07057
```

```
In [11]: x1=myU(U,y)

Out[11]: 6-element Array{Float64,1}:
           2.72418
           4.63586
          -3.5114
          -2.63338
          -0.195864
           1.46631
```

```
In [12]: # Usporedimo rješenja
         x-x1

Out[12]: 6-element Array{Float64,1}:
          -8.88178e-16
          -3.55271e-15
           1.77636e-15
           1.77636e-15
          -6.38378e-16
          -2.22045e-16
```

1.4 Brzina

Program `mylu()` je spor. Između ostalog, alocira nepotrebno tri matrice i ne računa s blok matricama.

Program se može preformulirati tako da su i L i U spremljene u polje A, pri čemu se dijagonala od L ne sprema jer su svi elementi jednaki 1 (vidi Introduction to Linear Algebra, str. 100):

```
In [13]: function mylu1{T}(A1::Array{T}) # Strang, str. 100
             A=deepcopy(A1)
             n,m=size(A)
             for k=1:n-1
                 rho=k+1:n
                 A[rho,k]=A[rho,k]/A[k,k]
                 A[rho,rho]=A[rho,rho]-A[rho,k]*A[k,rho]'
```

```
        end
        A
    end
```

Out[13]: mylu1 (generic function with 1 method)

In [14]: mylu1(A)

```
Out[14]: 6×6 Array{Float64,2}:
 0.768448   0.586022    0.865412    0.582318   0.20923    0.48
 1.22392   -0.665108   -0.441699   -0.456726   0.662085   0.202722
 0.877039   0.368849   -0.310382    0.363608   0.186542  -0.139531
 0.514613   0.289733   -0.471902    0.296226   0.591194   0.52933
 0.407632   0.113088    0.0869892   0.215088   0.25212    0.20895
 0.862199   0.0484897   0.896224   -0.0434479  2.3274    -0.730114
```

In [15]: L

```
Out[15]: 6×6 Array{Float64,2}:
 1.0       0.0         0.0         0.0        0.0     0.0
 1.22392   1.0         0.0         0.0        0.0     0.0
 0.877039  0.368849    1.0         0.0        0.0     0.0
 0.514613  0.289733   -0.471902    1.0        0.0     0.0
 0.407632  0.113088    0.0869892   0.215088   1.0     0.0
 0.862199  0.0484897   0.896224   -0.0434479  2.3274  1.0
```

In [16]: U

```
Out[16]: 6×6 Array{Float64,2}:
 0.768448   0.586022   0.865412   0.582318  0.20923    0.48
 0.0       -0.665108  -0.441699  -0.456726  0.662085   0.202722
 0.0        0.0       -0.310382   0.363608  0.186542  -0.139531
 0.0        0.0        0.0        0.296226  0.591194   0.52933
 0.0        0.0        0.0        0.0       0.25212    0.20895
 0.0        0.0        0.0        0.0       0.0       -0.730114
```

Usporedimo brzine LAPACK-ovog programa `lu()` i našeg naivnog programa `mylu()`na većoj dimenziji.

Izvedite program par puta radi točnijeg mjerenja brzine.

```
In [17]: n=512
         A=rand(n,n);
```

```
In [19]: @time lu(A);
```

```
  0.038275 seconds (21 allocations: 6.009 MiB)
```

```
In [21]: @time mylu1(A);
```

```
  1.124749 seconds (9.07 k allocations: 1.003 GiB, 4.20% gc time)
```

1.4.1 Blok varijanta

`mylu()` i `mylu1()` su nekoliko desetaka puta sporiji od `lu()`.

Preradimo `mylu1()` za rad s blokovima (nemamo ugrađeno pivotiranje!)

```
In [22]: function mylu2{T}(A1::Array{T}) # Strang, page 100
             A=deepcopy(A1)
             n,m=size(A)
             for k=1:n-1
                 for rho=k+1:n
                     A[rho,k]=A[rho,k]/A[k,k]
                     for l=k+1:n
                         A[rho,l]=A[rho,l]-A[rho,k]*A[k,l]
                     end
                 end
             end
             A
         end
```

```
Out[22]: mylu2 (generic function with 1 method)
```

Napravimo prvo mali test:

```
In [23]: # Probajte k,l=32,16 i k,l=64,8
         k,l=2,4
         Ab=[rand(k,k) for i=1:l, j=1:l];
```

```
In [24]: A0=mylu2(Ab)
```

```
Out[24]: 4×4 Array{Array{Float64,2},2}:
          [0.19178 0.0976698; 0.234544 0.627093]  …  [0.295925 0.164099; 0.942843 0.830942]
          [5.80254 -0.70825; 3.25685 -0.331261]      [-0.339473 -0.265039; -0.232242 0.107999]
          [3.23452 -0.450395; 2.34086 0.764213]      [0.0241067 1.07794; -0.633465 0.531221]
          [0.838168 0.600568; 0.876648 0.608517]     [-0.124081 0.60866; -0.886185 0.597327]
```

```
In [25]: # Provjera
         U=triu(A0)
         L=tril(A0)
         for i=1:maximum(size(L))
             L[i,i]=eye(L[1,1])
         end
```

```
In [26]: Rezidual=L*U-Ab
```

```
Out[26]: 4×4 Array{Array{Float64,2},2}:
          [0.0 0.0; 0.0 0.0]                       ...  [0.0 0.0; 0.0 0.0]
          [0.0 -1.11022e-16; 0.0 -2.77556e-17]     [0.0 0.0; 0.0 0.0]
          [0.0 0.0; 0.0 0.0]                       [0.0 0.0; 0.0 0.0]
          [0.0 -5.55112e-17; 0.0 -5.55112e-17]     [0.0 0.0; 0.0 0.0]
```

```
In [27]: # pretvaranje blok matrice u obicnu
         unblock(A) = mapreduce(identity, hcat, [mapreduce(identity, vcat, A[:,i])
                 for i = 1:size(A,2)])
```

```
Out[27]: unblock (generic function with 1 method)
```

```
In [28]: vecnorm(unblock(Rezidual))
```

```
Out[28]: 3.444376352465766e-16
```

Probajmo veće dimenzije ($n = k \cdot l$).

```
In [29]: k,l=32,16
         Ab=[rand(k,k) for i=1:l, j=1:l];
```

```
In [31]: @time mylu2(Ab);
```

```
  0.076786 seconds (3.95 k allocations: 26.553 MiB, 8.32% gc time)
```

Vidimo da je `mylu2()` gotovo jednako brz kao `lu()`, uz napomenu da `mylu2()` nema ugrađeno pivotiranje. Program još uvijek nije optimalan jer alocira previše memorije.

1.5 Pivotiranje

Standardne implementacije uvijek računaju Gaussovu eliminaciju s *parcijalnim pivotiranjem*:

> u svakom koraku se retci pivotiranju tako da pivotni element ima najveću apsolutnu vrijednost u danom stupcu. Na taj način je
>
> $$|L_{ij}| \leq 1,$$
>
> što u praksi dovoljno spriječava rast elemenata.

```
In [32]: A=[0.00003 1;2 3]
```

```
Out[32]: 2×2 Array{Float64,2}:
          3.0e-5  1.0
          2.0     3.0

In [33]: L,U=mylu(A)
         L

Out[33]: 2×2 Array{Float64,2}:
              1.0  0.0
          66666.7  1.0

In [34]: U

Out[34]: 2×2 Array{Float64,2}:
          3.0e-5       1.0
          0.0     -66663.7

In [35]: # s pivoritranjem
         P=[0 1;1 0]
         L,U=mylu(P*A)
         L

Out[35]: 2×2 Array{Float64,2}:
          1.0     0.0
          1.5e-5  1.0

In [36]: U

Out[36]: 2×2 Array{Float64,2}:
          2.0  3.0
          0.0  0.999955

In [37]: L*U-P*A

Out[37]: 2×2 Array{Float64,2}:
          0.0  0.0
          0.0  0.0

In [38]: # Slučajna matrica, standardna funkcija lu()
         srand(248)
         A=rand(5,5)
         L,U,P=lu(A);

In [39]: P
```

```
Out[39]: 5-element Array{Int64,1}:
          3
          4
          5
          2
          1
```

```
In [40]: L
```

```
Out[40]: 5×5 Array{Float64,2}:
          1.0        0.0         0.0       0.0       0.0
          0.0820339  1.0         0.0       0.0       0.0
          0.437335   0.0959387   1.0       0.0       0.0
          0.928548   0.0219166   0.150299  1.0       0.0
          0.386654   0.74805    -0.562755  0.803388  1.0
```

```
In [41]: U
```

```
Out[41]: 5×5 Array{Float64,2}:
          0.740619  0.105456  0.288167  0.0134151   0.810915
          0.0       0.939039  0.427963  0.871155    0.820178
          0.0       0.0       0.697849  0.776953    0.204272
          0.0       0.0       0.0       0.649058   -0.21677
          0.0       0.0       0.0       0.0         0.183107
```

```
In [42]: L*U-A[P,:]
```

```
Out[42]: 5×5 Array{Float64,2}:
          0.0  0.0          0.0          0.0  0.0
          0.0  0.0          0.0          0.0  0.0
          0.0  0.0          0.0          0.0  0.0
          0.0  0.0          0.0          0.0  0.0
          0.0  1.11022e-16  5.55112e-17  0.0  0.0
```

1.5.1 Potpuno pivotiranje

Sljedeći program računa Gaussovu eliminaciju s *potpunim pivotiranjem* - u svakom koraku se retci i stupci zamijene takoda se na pivotnu poziciju dovede element koji ima najveću apsolutnu vrijednost u trenutnoj podmatrici.

```
In [43]: function gecp{T}(A1::Array{T})
             # Gaussova eliminacija s potpunim pivotiranjem
             # Izlaz: Pr*L*U*Pc'=A ili Pr'*A*Pc=L*U
             A=deepcopy(A1)
             n,m=size(A)
             Pr=eye(n,n)
```

```
    Pc=eye(n,n)
    D=zeros(n)
    for i=1:n-1
        am1,im1=findmax(abs.(A[i:n,i:n]),1)
        am,JJ=findmax(am1)
        II=mod(im1[JJ],(n-i+1))
        if II==0
            II=n-i+1
        end
        imax=II+i-1
        jmax=JJ+i-1
        #  zamijena redaka
        if (imax != i)
            temp = Pr[:,i]
            Pr[:,i] = Pr[:,imax]
            Pr[:,imax] = temp
            temp = A[i,:]
            A[i,:] = A[imax,:]
            A[imax,:] = temp
        end
        # zamijena stupaca
        if (jmax != i)
            temp = Pc[:,i]
            Pc[:,i] = Pc[:,jmax]
            Pc[:,jmax] = temp
            temp = A[:,i]
            A[:,i] = A[:,jmax]
            A[:,jmax] = temp
        end
        # eliminacija
        D[i]=A[i,i]
        A[i+1:n,i] = A[i+1:n,i]/D[i]
        A[i+1:n,i+1:n] = A[i+1:n,i+1:n] - A[i+1:n,i]*A[i,i+1:n]'
        A[i,i+1:n]=A[i,i+1:n]/D[i]
    end
    D[n]=A[n,n]
    L=eye(n,n)+tril(A,-1)
    U=eye(n,n)+triu(A,1)
    U=diagm(D)*U
    L,U,Pr,Pc
end
```

```
Out[43]: gecp (generic function with 1 method)
```

```
In [44]: n=5
         A=rand(n,n)
         b=rand(n)
```

```
Out[44]: 5-element Array{Float64,1}:
          0.966598
          0.432393
          0.299164
          0.750221
          0.155147
```

```
In [45]: L,U,Pr,Pc=gecp(A);
```

```
In [46]: Pr
```

```
Out[46]: 5×5 Array{Float64,2}:
          0.0  0.0  1.0  0.0  0.0
          1.0  0.0  0.0  0.0  0.0
          0.0  0.0  0.0  1.0  0.0
          0.0  1.0  0.0  0.0  0.0
          0.0  0.0  0.0  0.0  1.0
```

```
In [47]: Pr*L*U*Pc'-A
```

```
Out[47]: 5×5 Array{Float64,2}:
          0.0  0.0  0.0  0.0  0.0
          0.0  0.0  0.0  0.0  0.0
          0.0  0.0  0.0  0.0  0.0
          0.0  0.0  0.0  0.0  0.0
          0.0  0.0  0.0  0.0  0.0
```

```
In [48]: y=myL(L,Pr'*b)
```

```
Out[48]: 5-element Array{Float64,1}:
           0.432393
           0.417654
           0.751194
          -0.275206
          -0.237007
```

```
In [49]: z=myU(U,y)
```

```
Out[49]: 5-element Array{Float64,1}:
           1.66403
           2.28921
          -0.219506
          -0.459733
          -5.36971
```

```
In [50]: x=Pc*z
```

```
Out[50]: 5-element Array{Float64,1}:
           1.66403
           2.28921
          -5.36971
          -0.459733
          -0.219506
```

```
In [51]: A*x-b
```

```
Out[51]: 5-element Array{Float64,1}:
          -4.44089e-16
          -3.88578e-16
          -2.22045e-16
          -3.33067e-16
          -3.88578e-16
```

1.6 Točnost

Neka je zadan sustav $Ax = b$, pri čemu je matrica A regularna.

Da bi primijenili koncepte iz bilježnice NA04 Pogreska unatrag_i stabilni_algoritmi, potrebno je:

1. napraviti teoriju smetnje za dani problem
2. analizirati pogreške algoritma (Gaussove eliminacije)

1.6.1 Teorija smetnje

Neka je

$$(A + \delta A)\hat{x} = (b + \delta b)$$

za neki $\hat{x} = x + \delta x$.

Želimo ocijeniti

$$\frac{\|\hat{x} - x\|}{\|x\|} \equiv \frac{\|\delta x\|}{\|x\|}.$$

Uvedimo oznake (npr. prema Matrix Computations, poglavlje 2.6.2)

$$\delta A = \varepsilon F, \quad \delta b = \varepsilon f, \qquad \hat{x} = x(\varepsilon),$$

čime smo dobili jednodimenzionalni problem

$$(A + \varepsilon F)\, x(\varepsilon) = b + \varepsilon f.$$

za neke (nepoznate) matricu F i vektor f.

Deriviranje po ε daje

$$Fx(\varepsilon) + (A + \varepsilon F)\, x(\varepsilon) = f.$$

Uvrštavanje $\varepsilon = 0$ daje

$$Fx + A\dot{x}(0) = f,$$

odnosno

$$\dot{x}(0) = A^{-1}(f - Fx).$$

Taylorov razvoj oko $\varepsilon = 0$ glasi

$$x(\varepsilon) = x(0) + \varepsilon\dot{x}(0) + O(\varepsilon^2),$$

odnosno, uz zanemarivanje člana $O(\varepsilon^2)$,

$$\hat{x} - x = \varepsilon A^{-1}(f - Fx) = A^{-1}(\varepsilon f + \varepsilon Fx) = A^{-1}(\delta b + \delta A x).$$

Svojstva norme povlače

$$\|\hat{x} - x\| \leq \|A^{-1}\|(\|\delta b\| + \|\delta A\| \cdot \|x\|).$$

Konačno, zbog $\|b\| \leq \|A\|\|x\|$, imamo

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \|A\| \cdot \|A^{-1}\| \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right). \tag{1}$$

Broj

$$\kappa(A) \equiv \|A\| \cdot \|A^{-1}\|$$

je **uvjetovanost** (**kondicija**) matrice A i kazuje nam

> koliko se relativno uvećaju relativne promjene u polaznim podacima (matrici A i vektoru b).

Pogledajmo primjer iz Numeričke matematike, str. 42:

```
In [52]: A= [0.234 0.458; 0.383 0.750]
```

```
Out[52]: 2×2 Array{Float64,2}:
          0.234  0.458
          0.383  0.75
```

```
In [53]: b=[0.224;0.367]
```

```
Out[53]: 2-element Array{Float64,1}:
          0.224
          0.367

In [54]: x=A\b

Out[54]: 2-element Array{Float64,1}:
          -1.0
           1.0

In [55]: δb=[0.00009; 0.000005]
         x1=A\(b+δb)

Out[55]: 2-element Array{Float64,1}:
          -0.241744
           0.612791

In [56]: cond(A), norm(δb)/norm(b), norm(x1-x)/norm(x)

Out[56]: (11322.197586092605, 0.0002096449170953002, 0.6020311134825742)

In [57]: δA=[-0.001 0;0 0]
         x2=(A+δA)\b

Out[57]: 2-element Array{Float64,1}:
          0.129518
          0.423193

In [58]: cond(A), norm(δA)/norm(A), norm(x2-x)/norm(x)

Out[58]: (11322.197586092605, 0.0010134105230118603, 0.896804787832142)
```

1.6.2 Pogreška Gaussove eliminacije

Prema Matrix Computations, poglavlje 3.3, za izračunate faktore $\hat L$ i $\hat U$ vrijedi

$$\hat L \cdot \hat U = A + \delta A$$

gdje je (nejednakost se čita po elementima matrica, ε je sada točnost stroja)

$$|\delta A| \leq 3(n-1)\varepsilon(|A| + |\hat L| \cdot |\hat U|) + O(\varepsilon^2).$$

Zanemarivanje člana $O(\varepsilon^2)$ i prelazak na normu daju

$$\|\delta A\| \approx\leq O(n)\varepsilon(\|A\| + \|\hat L\| \cdot \|\hat U\|),$$

pa je

$$\frac{\|\delta A\|}{\|A\|} \leq O(n)\varepsilon\left(1+\frac{\|\hat{L}\|\cdot\|\hat{U}\|}{\|A\|}\right).$$

Ukoliko se Gaussova eliminacija radi s pivotiranjem, tada će najvjerojatnije zadnji kvocijent također biti malen (≈ 1). Također, pogreška kod rješavanja trokutastih sustava nije veća od navedene pa uvrštavanjem u (1) slijedi da za relativnu pogrešku izračunatog rješenja vrijedi

$$\frac{\|\hat{x}-x\|}{\|x\|} \leq \kappa(A)O(n\varepsilon).$$

Zaključimo:

> *Ukoliko je kondicija matrice velika, rješenje može biti netočno.*

In [59]:
```
n=10
v=rand(n)
```

Out[59]:
```
10-element Array{Float64,1}:
 0.661435
 0.486509
 0.417289
 0.388693
 0.596575
 0.0759266
 0.486095
 0.469004
 0.528265
 0.755251
```

In [60]:
```
# Vandermonmdeove matrice imaju veliku kondiciju.
V=Array{Float64}(n,n)
for i=1:n
    V[:,i]=v.^(i-1)
end
V=V'
```

Out[60]:
```
10×10 Array{Float64,2}:
 1.0        1.0         1.0          …  1.0         1.0         1.0
 0.661435   0.486509    0.417289        0.469004    0.528265    0.755251
 0.437496   0.236691    0.17413         0.219965    0.279064    0.570403
 0.289375   0.115152    0.0726625       0.103164    0.14742     0.430797
 0.191403   0.0560226   0.0303213       0.0483845   0.0778767   0.32536
 0.126601   0.0272555   0.0126527    …  0.0226925   0.0411395   0.245728
 0.0837381  0.01326     0.00527984      0.0106429   0.0217326   0.185586
 0.0553873  0.00645113  0.00220322      0.00499156  0.0114806   0.140164
 0.0366351  0.00313853  0.000919379     0.00234106  0.00606477  0.105859
 0.0242318  0.00152692  0.000383647     0.00109797  0.00320381  0.0799502
```

In [61]: b_v=rand(n)

Out[61]:
```
10-element Array{Float64,1}:
 0.331049
 0.85607
 0.713867
 0.395595
 0.203756
 0.662775
 0.453411
 0.230215
 0.311921
 0.310941
```

In [62]: x_v=V\b_v

Out[62]:
```
10-element Array{Float64,1}:
    -1.95061e6
     1.24432e11
    -1.60949e8
     3.42069e7
     1.98874e7
 -2127.92
    -1.26768e11
     2.77062e9
    -3.25708e8
 50186.2
```

In [63]:
```
cond(V)
```

Out[63]:
```
1.804836065696248e13
```

In [64]: Vb=map(BigFloat,V)
bb=map(BigFloat,b_v)
xb=Vb\bb;

In [65]: map(Float64,norm(xb-x_v)/norm(xb))

Out[65]:
```
2.5620705632299447e-5
```

1.6.3 Umjetno loša kondicija

In [66]:
```
A=[1 1; 1 2]
b=[1;3]
x=A\b
@show x,cond(A)
```

```
A1=[1e-4 1e-4;1 2]
b1=[1e-4;3]
x1=A1\b1
x,cond(A1),x-x1
```

```
(x, cond(A)) = ([-1.0, 2.0], 6.854101966249685)
```

```
Out[66]: ([-1.0, 2.0], 50000.00017991671, [8.88178e-16, -4.44089e-16])
```

1.6.4 Procjena kondicije

Računanje kondicije prema definiciji $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ zahtijeva računanje matrice inverzne matrice, za što je potrebno $O(n^3)$ operacija. To je isti red veličine operacija koji je potreban za rješavanje zadanog sustava. Prilikom rješavanja sustava na raspolaganju su nam trokutasti faktori L i U, što se može iskoristiti kako bi se kondicija približno izračunala u $O(n^2)$ operacija. Detalji se nalaze u Section **??**. LAPACK rutina dtrcon.f računa približnu kondiciju trokutaste matrice.

Izračunajmo približnu kondiciju Vandermondeove matrice iz prethodnog primjera.

```
In [67]: ?LAPACK.trcon!
```

```
Out[67]:
```

```
trcon!(norm, uplo, diag, A)
```

Finds the reciprocal condition number of (upper if `uplo = U`, lower if `uplo = L`) triangular matrix `A`. If `diag = N`, `A` has non-unit diagonal elements. If `diag = U`, all diagonal elements of `A` are one. If `norm = I`, the condition number is found in the infinity norm. If `norm = O` or `1`, the condition number is found in the one norm.

```
In [68]: L,U=lu(V);
```

```
In [69]: cond(V,1),cond(L,1),cond(U,1)
```

```
Out[69]: (1.6503607594791473e13, 37.48683913429581, 6.680561720694167e12)
```

```
In [70]: 1./LAPACK.trcon!('O','L','U',L),1./LAPACK.trcon!('O','U','N',U)
```

```
Out[70]: (37.48683913429582, 6.680561720694167e12)
```

1.7 Rezidual

Izračunato rješenje $\hat{x}$ sustava $Ax = b$ je točno rješenje nekog sličnog sustava (vidi Afternotes on Numerical Analysis, str. 128):

$$(A+\delta A)\,\hat x = b. \tag{1}$$

Rezidual (ili **ostatak**) definiramo kao

$$r=b-A\hat x.$$

Tada je

$$0=b-(A+\delta A)\,\hat x = r-\delta A\,\hat x$$

pa je

$$\|r\|=\|\delta A\,\hat x\|\leq \|\delta A\|\cdot\|\hat x\|,$$

odnosno

$$\frac{\|\delta A\|}{\|A\|}\geq \frac{\|r\|}{\|A\|\cdot\|\hat x\|}.$$

Dakle,

> ako *relativni rezidual*
>
> $$\frac{r}{\|A\|\cdot\|\hat x\|}$$
>
> ima veliku normu, tada *rješenje nije izračunato stabilno.*

S druge strane, ako relativni rezidual ima malu normu, tada je *rješenje izračunato stabilno.* Naime, za

$$\delta A=\frac{r\hat x^T}{\|\hat x\|^2}$$

vrijedi (1):

$$b-(A+\delta A)\hat x=(b-A\hat x)-\delta A\hat x = r-\frac{r\hat x^T\hat x}{\|\hat x\|^2}=r-\frac{r\|\hat x^T\hat x\|}{\|\hat x\|^2}=r-r=0.$$

Također vrijedi

$$\frac{\|\delta A\|}{\|A\|}\leq\frac{\|r\|\,\|\hat x\|}{\|A\|\cdot\|\hat x\|^2}=\frac{\|r\|}{\|A\|\cdot\|\hat x\|}.$$

Izračunajmo reziduale za prethodni primjer dimenzije 2:

```
In [71]: r=b-A*x
```

```
Out[71]: 2-element Array{Float64,1}:
          0.0
          0.0
```

```
In [72]: norm(r)/(norm(A)*norm(x))
```

```
Out[72]: 0.0
```

```
In [73]: r1=b1-A1*x1
```

```
Out[73]: 2-element Array{Float64,1}:
          4.06576e-20
          0.0
```

```
In [74]: norm(r1)/(norm(A1)*norm(x1))
```

```
Out[74]: 8.131516279004551e-21
```

Izračunajmo rezidual za Vandermondeov sustav:

```
In [75]: rᵥ=bᵥ-V*xᵥ
```

```
Out[75]: 10-element Array{Float64,1}:
          -3.83296e-6
          -2.75583e-6
          -3.56353e-6
           7.61436e-7
          -1.17092e-6
          -1.06218e-6
          -9.18339e-7
           2.51501e-7
          -4.45021e-7
          -2.52872e-7
```

```
In [76]: norm(rᵥ)/(norm(V)*norm(xᵥ))
```

```
Out[76]: 9.542483730083024e-18
```

Zaključujemo da je rješenje x_v izračunato stabilno, odnosno s vrlo malom pogreškom unatrag u početnim podatcima. To još uvijek ne znači da je rješenje relativno vrlo točno.