lab4

December 6, 2024

0.1 № 4.

0.1.1

[1]:
```
#     4x3                   (  1   20):
a = rand(1:20,(4,3))
```

[1]:
```
4×3 Matrix{Int64}:
 17  10  17
  1   3  11
 11  10  12
  8   9   4
```

[2]:
```
#           :
sum(a)
```

[2]:
```
113
```

[3]:
```
#                  :
sum(a,dims=1)
```

[3]:
```
1×3 Matrix{Int64}:
 37  32  44
```

[4]:
```
#                 :
sum(a,dims=2)
```

[4]:
```
4×1 Matrix{Int64}:
 44
 15
 33
 21
```

[5]:
```
#              :
prod(a)
```

[5]:
```
36255859200
```

[6]:
```
#                       :
prod(a,dims=1)
```

[6]:
```
1×3 Matrix{Int64}:
 1496  2700  8976
```

[7]:
```
#                      :
prod(a,dims=2)
```

[7]:
```
4×1 Matrix{Int64}:
 2890
   33
 1320
  288
```

[8]:
```
import Pkg
Pkg.add("Statistics")
```

```
    Updating registry at
`C:\Users\User\.julia\registries\General.toml`
   Resolving package versions…
    Updating
`C:\Users\User\.julia\environments\v1.10\Project.toml`
  [10745b16] + Statistics v1.10.0
  No Changes to
`C:\Users\User\.julia\environments\v1.10\Manifest.toml`
```

[9]:
```
using Statistics
#                      :
mean(a)
```

[9]:
```
9.416666666666666
```

[10]:
```
#            :
mean(a,dims=1)
```

[10]:
```
1×3 Matrix{Float64}:
 9.25  8.0  11.0
```

[11]:
```
#           :
mean(a,dims=2)
```

[11]:
```
4×1 Matrix{Float64}:
 14.666666666666666
  5.0
 11.0
  7.0
```

0.1.2 , , ,

[12]:
```julia
#            LinearAlgebra:
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra
```

```
   Resolving package versions…
    Updating
`C:\Users\User\.julia\environments\v1.10\Project.toml`
  [37e2e46d] + LinearAlgebra
  No Changes to
`C:\Users\User\.julia\environments\v1.10\Manifest.toml`
```

[13]:
```julia
#     4x4                    (  1   20):
b = rand(1:20,(4,4))
```

[13]:
```
4×4 Matrix{Int64}:
  7  13  19   6
 16  15  14  12
  1  12  11  14
 19   4   2  12
```

[14]:
```julia
#          :
transpose(b)
```

[14]:
```
4×4 transpose(::Matrix{Int64}) with eltype Int64:
  7  16   1  19
 13  15  12   4
 19  14  11   2
  6  12  14  12
```

[15]:
```julia
#         (                ):
tr(b)
```

[15]:
```
45
```

[16]:
```julia
#                          :
diag(b)
```

[16]:
```
4-element Vector{Int64}:
  7
 15
 11
 12
```

[17]:
```julia
#        :
rank(b)
```

[17]: 4

[18]:
```
#           (                ):
inv(b)
```

[18]:
```
4×4 Matrix{Float64}:
  0.00578035   0.0289017  -0.0520231    0.0289017
 -0.179716     0.283237   -0.0189175   -0.171308
  0.166185    -0.169075    0.00433526   0.0809249
  0.0230557   -0.111994    0.0879532    0.0811876
```

[19]:
```
#            :
det(b)
```

[19]:
```
15224.000000000002
```

[20]:
```
#                            :
pinv(a)
```

[20]:
```
3×4 Matrix{Float64}:
  0.108197   -0.114931   -0.0327538   -0.0455152
 -0.117346    0.0514163   0.0703955    0.14614
  0.0156561   0.0804726   0.00337855  -0.0479738
```

0.1.3 , ,

[21]:
```
#            X:
X = [2, 4, -5]
```

[21]:
```
3-element Vector{Int64}:
  2
  4
 -5
```

[22]:
```
#                :
norm(X)
```

[22]:
```
6.708203932499369
```

[23]:
```
#        p-   :
p = 1
norm(X,p)
```

[23]:
```
11.0
```

[24]:
```
#                         X   Y:
X = [2, 4, -5]
Y = [1,-1,3]
```

```
norm(X-Y)
```

```
[24]: 9.486832980505138
```

```
[25]: #                    :
      sqrt(sum((X-Y).^2))
```

```
[25]: 9.486832980505138
```

```
[26]: #                 :
      acos((transpose(X)*Y)/(norm(X)*norm(Y)))
```

```
[26]: 2.4404307889469252
```

```
[27]: #          :
      d = [5 -4 2 ; -1 2 3; -2 1 0]
```

```
[27]: 3×3 Matrix{Int64}:
        5  -4  2
       -1   2  3
       -2   1  0
```

```
[28]: #                 :
      opnorm(d)
```

```
[28]: 7.147682841795258
```

```
[29]: #        p-   :
      p=1
      opnorm(d,p)
```

```
[29]: 8.0
```

```
[30]: #        180      :
      rot180(d)
```

```
[30]: 3×3 Matrix{Int64}:
       0   1  -2
       3   2  -1
       2  -4   5
```

```
[31]: #              :
      reverse(d,dims=1)
```

```
[31]: 3×3 Matrix{Int64}:
       -2   1  0
       -1   2  3
        5  -4  2
```

[32]:
```
#
reverse(d,dims=2)
```

[32]:
```
3×3 Matrix{Int64}:
 2  -4   5
 3   2  -1
 0   1  -2
```

0.1.4 , ,

[34]:
```
#     2x3                     1   10:
A = rand(1:10,(2,3))
#     3x4                     1   10:
B = rand(1:10,(3,4))

print(A)
print()
print(B)
```

```
[4 6 8; 7 4 4][4 9 10 7; 5 5 8 4; 5 10 10 8]
```

[35]:
```
#             A   B:
A*B
```

[35]:
```
2×4 Matrix{Int64}:
 86  146  168  116
 68  123  142   97
```

[36]:
```
#             3x3:
Matrix{Int}(I, 3, 3)
```

[36]:
```
3×3 Matrix{Int64}:
 1  0  0
 0  1  0
 0  0  1
```

[37]:
```
#                    X   Y:
X = [2, 4, -5]
Y = [1,-1,3]
dot(X,Y)
```

[37]:
```
-17
```

[38]:
```
#                 :
X'Y
```

[38]:
```
-17
```

0.1.5 .

[72]:
```
#                 3x3               :
A = rand(3, 3)
```

[72]:
```
3×3 Matrix{Float64}:
 0.323776  0.766627  0.346568
 0.82158   0.154756  0.212603
 0.089916  0.231533  0.572396
```

[73]:
```
#             :
x = fill(1.0, 3)
```

[73]:
```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

[74]:
```
#          b:
b = A*x
```

[74]:
```
3-element Vector{Float64}:
 1.4369700554357259
 1.1889391623864485
 0.8938450054376931
```

[75]:
```
#                                  \
# (      ,    x -            ):
A\b
```

[75]:
```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

[76]:
```
# LU-       :
Alu = lu(A)
```

[76]:
```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3×3 Matrix{Float64}:
 1.0       0.0       0.0
 0.394089  1.0       0.0
 0.109443  0.304115  1.0
U factor:
3×3 Matrix{Float64}:
 0.82158  0.154756  0.212603
 0.0      0.705639  0.262783
 0.0       0.0        0.469212
```

[79]:
```
#                  A:
A\b
```

[79]:
```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

[80]:
```
#                        :
Alu\b
```

[80]:
```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

[81]:
```
#             A:
det(A)
```

[81]:
```
-0.2720205154988499
```

[82]:
```
#             A                :
det(Alu)
```

[82]:
```
-0.2720205154988499
```

[83]:
```
# QR-       :
Aqr = qr(A)
```

[83]:
```
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64},
Matrix{Float64}}
R factor:
3×3 Matrix{Float64}:
 -0.887643  -0.446326  -0.381176
  0.0         0.682691   0.382295
  0.0         0.0        0.44889
```

[84]:
```
#      Q:
Aqr.Q
```

[84]:
```
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
```

[65]:
```
#      R:
Aqr.R
```

[65]:
```
3×3 Matrix{Float64}:
 -0.884374  -0.455329  -0.264604
  0.0       -0.792468  -0.521473
  0.0        0.0        0.462976
```

[66]:
```
#     ,        Q -        :
Aqr.Q'*Aqr.Q
```

[66]:
```
3×3 Matrix{Float64}:
  1.0          -2.77556e-17  -1.38778e-17
 -5.55112e-17   1.0           0.0
  0.0           0.0           1.0
```

[67]:
```
#              A:
Asym = A + A'
```

[67]:
```
3×3 Matrix{Float64}:
 1.74614   0.464061  0.211887
 0.464061  1.7004    0.570514
 0.211887  0.570514  0.929872
```

[68]:
```
#                            :
AsymEig = eigen(Asym)
```

[68]:
```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 0.6257132275305365
 1.3504140615067672
 2.4002822908619654
vectors:
3×3 Matrix{Float64}:
  0.0328487   0.788192  -0.614552
 -0.479667   -0.52701   -0.701555
  0.876835   -0.317825  -0.360758
```

[69]:
```
#            :
AsymEig.values
```

[69]:
```
3-element Vector{Float64}:
 0.6257132275305365
 1.3504140615067672
 2.4002822908619654
```

[70]:
```
#          :
AsymEig.vectors
```

```
[70]: 3×3 Matrix{Float64}:
        0.0328487   0.788192  -0.614552
       -0.479667   -0.52701   -0.701555
        0.876835   -0.317825  -0.360758
```

```
[71]: #      ,                   :
      inv(AsymEig)*Asym
```

```
[71]: 3×3 Matrix{Float64}:
        1.0          -4.51028e-17   4.09395e-16
       -6.93889e-17   1.0          -1.11022e-16
        9.99201e-16   1.11022e-16   1.0
```

```
[85]: #      1000   1000:
      n = 1000
      A = randn(n,n)
```

```
[85]: 1000×1000 Matrix{Float64}:
        0.945564   -0.4136      0.288745   …  -0.0576756  -1.29635   -1.55455
       -0.733987   -0.0813476  -1.8088         0.00406874 -0.146778  -0.543025
        0.416595    0.215597    1.72953        2.10461     2.27       0.493712
       -1.17069    -1.28196     0.172427      -0.281696    0.113201   0.402691
       -0.534953   -0.454402   -0.0904224      0.737858    0.374288   1.03684
       -0.730627    0.79689     0.552524   …  -0.205875    1.10018   -0.537194
       -0.0978003  -0.556665    0.477047      -1.486      -0.628359  -0.915956
        0.562794   -1.07607    -0.526134       0.217834    1.1874    -0.744079
        0.585691    0.901834    0.0827245      1.53281     0.17808   -0.185429
       -0.546395    0.0415964  -0.736421       0.566128   -0.838803   0.172493
        0.413408    0.344019   -0.731707   …  -0.139578    1.23046   -0.406932
       -0.559737   -0.124365   -0.153766       2.37222     1.8644     0.234218
        2.64962    -0.339176    1.10631        1.39374     0.525698  -0.482248
       ⋮                                   ⋱
        1.15559    -1.46166    -1.49758       -0.585228   -1.92706   -0.161073
       -1.14017     0.672569    1.92851       -0.064009    0.101601   0.308163
        0.23013     1.43359     1.09088    …   1.47976     2.01193   -1.21324
       -0.456641   -0.34784    -1.69538        0.216645   -0.678083   0.237115
        0.557833    1.06935     0.690091      -0.104601   -0.676934   2.56679
       -0.0322185  -0.415773    0.477601      -1.52515     0.624246  -0.660277
       -0.810073    0.906215    0.130511       1.3033     -0.350364  -0.445778
       -1.39737    -0.0498869  -1.29807    …  -0.125225   -1.98868   -0.57905
       -0.489871   -1.03055    -0.382777      -0.408173    0.166898   2.47756
        2.38396     0.108788    0.98079       -1.67812    -1.18574    2.19278
        0.407764    0.613379    0.673965      -0.99283    -0.817734  -0.661687
       -1.19675    -0.097588    0.516592       0.694347    1.05551    0.146226
```

```
[86]: #             :
      Asym = A + A'
```

```
[86]: 1000×1000 Matrix{Float64}:
        1.89113    -1.14759     0.705339   …   2.32628    -0.888584  -2.7513
       -1.14759    -0.162695   -1.59321        0.112857    0.466601  -0.640613
        0.705339   -1.59321     3.45905        3.0854      2.94397    1.0103
       -1.95993    -1.60121    -0.690125      -0.234089    1.40913    0.396142
       -1.10852     1.63509    -0.371592       1.63391     1.63206    1.65072
       -0.598721    0.935986    0.537715   …  -0.363744    1.30787    0.429677
        1.16067    -0.255707    0.216914      -2.40255    -2.46799   -1.46619
       -0.434411   -0.607108    0.816192       1.19282     1.27309   -0.270205
        0.412727    0.722902    0.424145       2.62941    -0.288722  -0.113701
       -0.434874   -0.659047   -1.59224        0.0749583   0.163534   0.777786
        1.96235     0.363711   -1.27095    …  -0.482602    0.352057  -0.0786477
       -0.814709   -0.445876   -1.83821        1.94458     2.5976     1.03824
        2.16479    -2.58858     2.87103        0.668372   -0.158715   0.53989
        ⋮                                  ⋱                          
        0.0138219  -1.43221     0.141513       1.13619    -2.71519   -0.368563
       -1.18133    -0.0518368   1.23705        0.923544   -0.336829   0.0998745
        0.789905    3.15444     1.84744    …   3.2034      4.25491   -1.54858
        0.0580671   0.419394   -2.65152       -0.614025   -0.492288  -0.157775
        1.01359    -0.496089    1.09417       -0.0253229   0.173054   2.17623
        2.10788    -0.433621    0.264345      -0.018247    0.350143   0.316145
        0.757072   -1.23472     0.593982       1.89946     1.68289   -1.29059
        0.917399   -0.740756   -2.87718    …  -1.5719     -1.71603   -1.20678
        0.10315    -0.956864   -0.613048       0.528737    0.177616   3.96403
        2.32628     0.112857    3.0854        -3.35623    -2.17857    2.88713
       -0.888584    0.466601    2.94397       -2.17857    -1.63547    0.393826
       -2.7513     -0.640613    1.0103         2.88713     0.393826   0.292451
```

```
[87]: #     ,                          :
      issymmetric(Asym)
```

```
[87]: true
```

```
[88]: #          :
      Asym_noisy = copy(Asym)
      Asym_noisy[1,2] += 5eps()
```

```
[88]: -1.147586931664726
```

```
[89]: #     ,                          :
      issymmetric(Asym_noisy)
```

```
[89]: false
```

```
[90]: #          ,                         :
      Asym_explicit = Symmetric(Asym_noisy)
```

```
[90]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:
        1.89113    -1.14759     0.705339  …   2.32628    -0.888584  -2.7513
       -1.14759    -0.162695   -1.59321       0.112857    0.466601  -0.640613
        0.705339   -1.59321     3.45905       3.0854      2.94397    1.0103
       -1.95993    -1.60121    -0.690125     -0.234089    1.40913    0.396142
       -1.10852     1.63509    -0.371592      1.63391     1.63206    1.65072
       -0.598721    0.935986    0.537715  …  -0.363744    1.30787    0.429677
        1.16067    -0.255707    0.216914     -2.40255    -2.46799   -1.46619
       -0.434411   -0.607108    0.816192      1.19282     1.27309   -0.270205
        0.412727    0.722902    0.424145      2.62941    -0.288722  -0.113701
       -0.434874   -0.659047   -1.59224       0.0749583   0.163534   0.777786
        1.96235     0.363711   -1.27095   …  -0.482602    0.352057  -0.0786477
       -0.814709   -0.445876   -1.83821       1.94458     2.5976     1.03824
        2.16479    -2.58858     2.87103       0.668372   -0.158715   0.53989

        0.0138219  -1.43221     0.141513      1.13619    -2.71519   -0.368563
       -1.18133    -0.0518368   1.23705       0.923544   -0.336829   0.0998745
        0.789905    3.15444     1.84744   …   3.2034      4.25491   -1.54858
        0.0580671   0.419394   -2.65152      -0.614025   -0.492288  -0.157775
        1.01359    -0.496089    1.09417      -0.0253229   0.173054   2.17623
        2.10788    -0.433621    0.264345     -0.018247    0.350143   0.316145
        0.757072   -1.23472     0.593982      1.89946     1.68289   -1.29059
        0.917399   -0.740756   -2.87718   …  -1.5719     -1.71603   -1.20678
        0.10315    -0.956864   -0.613048      0.528737    0.177616   3.96403
        2.32628     0.112857    3.0854       -3.35623    -2.17857    2.88713
       -0.888584    0.466601    2.94397      -2.17857    -1.63547    0.393826
       -2.7513     -0.640613    1.0103        2.88713     0.393826   0.292451
```

```
[91]: import Pkg
      Pkg.add("BenchmarkTools")
      using BenchmarkTools
```

```
   Resolving package versions…
   Installed BenchmarkTools  v1.5.0
    Updating
`C:\Users\User\.julia\environments\v1.10\Project.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
    Updating
`C:\Users\User\.julia\environments\v1.10\Manifest.toml`
  [6e4b80f9] ↑ BenchmarkTools v1.4.0 ⇒ v1.5.0
Precompiling project…
    BenchmarkTools
    MathOptInterface
    Optim
    DiffEqNoiseProcess
    StochasticDiffEq
    DifferentialEquations
  6 dependencies successfully precompiled in 144 seconds. 322 already
```

```
precompiled.
```

[92]:
```
#
#                              :
@btime eigvals(Asym)
```

```
  140.190 ms (11 allocations: 7.99 MiB)
```

[92]:
```
1000-element Vector{Float64}:
 -89.90745697267798
 -89.28462914513722
 -87.7618794507008
 -86.82276458104195
 -86.17545419337942
 -85.50230407876174
 -85.06852580328531
 -84.70714389011076
 -84.5030609471845
 -84.18620811687069
 -83.58445180670451
 -83.05188255007515
 -82.88233698029482

  82.50731589150371
  83.0077687026647
  83.13600831985721
  83.28420721839369
  84.08896605208413
  84.6386358303351
  85.29488028420883
  85.65940396658235
  86.54358105900604
  87.12252883775672
  87.33499654466465
  87.5091798420538
```

[96]:
```
#
#                           :
@btime eigvals(Asym_noisy)
```

```
  736.767 ms (14 allocations: 7.93 MiB)
```

[96]:
```
1000-element Vector{Float64}:
 -89.90745697267911
 -89.28462914513783
 -87.76187945070113
 -86.8227645810423
 -86.17545419337884
 -85.50230407876121
 -85.06852580328572
 -84.70714389011033
 -84.50306094718434
 -84.18620811687012
 -83.58445180670492
 -83.05188255007515
 -82.8823369802946
   ⋮
  82.50731589150357
  83.00776870266446
  83.1360083198572
  83.28420721839342
  84.08896605208409
  84.63863583033428
  85.29488028420847
  85.65940396658272
  86.54358105900539
  87.12252883775666
  87.33499654466515
  87.50917984205434
```

[97]:
```
#
#                                   ,
#                           ,                 :
@btime eigvals(Asym_explicit)
```

```
  153.165 ms (11 allocations: 7.99 MiB)
```

[97]:
```
1000-element Vector{Float64}:
 -89.90745697267839
 -89.28462914513725
 -87.76187945070029
 -86.82276458104262
 -86.1754541933794
 -85.50230407876178
 -85.06852580328525
 -84.70714389011067
 -84.50306094718444
 -84.18620811687055
 -83.5844518067044
 -83.05188255007535
 -82.88233698029536
   ⋮
  82.5073158915038
  83.00776870266455
  83.13600831985687
  83.2842072183936
  84.08896605208412
 84.6386358303352
 85.29488028420896
 85.65940396658252
 86.54358105900596
 87.12252883775663
 87.33499654466463
 87.50917984205351
```

[100]:
```julia
#                 1000000   1000000:
n = 1000000
A = SymTridiagonal(randn(n), randn(n-1))
```

[100]:
```
1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
 0.540287   0.7954                          …
 0.7954     1.1026   -1.52548
          -1.52548   0.197984  1.57573
                     1.57573   0.214912
                               1.4176
                                          …



                                          …





                                          …





                                          …
                                             -0.169001
                                              0.546604   0.357314
                                              0.357314  -0.4732    0.106754
                                                         0.106754  0.213239
```

[101]:
```julia
#
#              :
@btime eigmax(A)
```

```
  524.792 ms (17 allocations: 183.11 MiB)
```

[101]:
```
6.275049715720039
```

```
[108]: B = Matrix(A)
```

```
OutOfMemoryError()

Stacktrace:
 [1] Array
   @ .\boot.jl:479 [inlined]
 [2] Matrix{Float64}(M::SymTridiagonal{Float64, Vector{Float64}})
   @ LinearAlgebra C:\Users\User\AppData\Local\Programs\Julia-1.10.
↪0\share\julia\stdlib\v1.10\LinearAlgebra\src\tridiag.jl:127
 [3] (Matrix)(M::SymTridiagonal{Float64, Vector{Float64}})
   @ LinearAlgebra C:\Users\User\AppData\Local\Programs\Julia-1.10.
↪0\share\julia\stdlib\v1.10\LinearAlgebra\src\tridiag.jl:138
 [4] top-level scope
   @ In[108]:1
```

0.1.6

```
[102]: #                     :
       Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
```

```
[102]: 3×3 Matrix{Rational{BigInt}}:
        1//10  1//2  2//5
        3//10  4//5  7//10
         1     3//5  4//5
```

```
[103]: #           :
       x = fill(1, 3)
       #          b:
       b = Arational*x
```

```
[103]: 3-element Vector{Rational{BigInt}}:
          1
         9//5
        12//5
```

```
[104]: #                                  \
       # (      ,    x -           ):
       Arational\b
```

```
[104]: 3-element Vector{Rational{BigInt}}:
        1
        1
        1
```

```
[105]: # LU-      :
       lu(Arational)
```

```
[105]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
       L factor:
       3×3 Matrix{Rational{BigInt}}:
         1       0      0
        3//10    1      0
        1//10  22//31   1
       U factor:
       3×3 Matrix{Rational{BigInt}}:
        1   3//5    4//5
        0  31//50  23//50
        0    0     -1//155
```

0.2

0.2.1

1. v. v dot_v.

```
[106]: v = rand(1:100, 3); display(v)
       dot_v = v'v
```

```
3-element Vector{Int64}:
 16
 69
 25
```

```
[106]: 5642
```

2. v ( ), outer_v.

```
[107]: outer_v = v*v'
```

```
[107]: 3×3 Matrix{Int64}:
         256  1104   400
        1104  4761  1725
         400  1725   625
```

0.2.2

1. .

```
[109]: function LinearDep(mtrx::Matrix, vec::Vector)
           # returns isSolvable::Bool, ind::Vector{Int64} --
           A = hcat(mtrx, vec)
           Ac = copy(mtrx); bc = copy(vec)
           s1 = size(A)[1]; s2 = size(A)[2]-1
           t = [false for i in 1:size(A)[1]]
           poss_j = collect(2:s2)
           for i in 1:s2
               for j in i+1:s1
```

```julia
            mbool = true
            temp = A[j, :]./A[i, :]
            if length(unique(temp[1:s2])) == 1
                if temp[s2+1] == temp[1]
                    t[j] = true
                else
                    return false, []
                end
            end
            tii = i
            if Ac[i, i] == 0
                tii = sortperm(abs.(Ac[i, :]))[s2]
                if Ac[i, tii] == 0
                    mbool = false
                end
            end
            if mbool
                c = -Ac[j, tii] / Ac[i, tii]
                if isequal(Ac[j, :].+(c*Ac[i, :]), zeros(Float64, s2))
                    if bc[j] + c*bc[i] != 0
                        return false, []
                    else
                        t[j] = true
                        Ac[j, :] = Ac[j, :].+(c*Ac[i, :])
                        bc[j] += c*bc[i]
                    end
                else
                    Ac[j, :].+= (c*Ac[i, :])
                    bc[j] += c*bc[i]
                end
            end
        end
    end
    for i in 1:s1
        if isequal(Ac[i, :], zeros(Float64, s2))
            t[i] = true
        end
    end
    answ = deleteat!(collect(1:s1), t)
    if length(answ) >= s2
        return true, answ
    else
        return false, [pi]
    end
end

function SLAU_solver(A::Matrix, b::Vector)
```

```
    if ndims(A) != 2 || size(A)[1] != length(b)
        println("              !")
        return
    end
    s1 = size(A)[1]; s2 = size(A)[2]
    if s1 == s2 && det(A) != 0
        return A\b
    elseif s1 < s2
        println("         ,         ")
        return
    else # s1 > s2 || (s1 == s2 && det(A) == 0)
        isSolvable, indNonLinear = LinearDep(A, b)
        if !isSolvable && isequal(indNonLinear, [])
            println("      ")
            return
        elseif !isSolvable && isequal(indNonLinear, [pi])
            println("                ")
            return
        else
            length(indNonLinear) > s2 ? indNonLinear = indNonLinear[1:s2] :
            return A[indNonLinear, :]\b[indNonLinear]
        end
    end
end
A = Float64[1 2 3; 1/3 2 1; 2 3 6; 3 4 5]
b = Float64[1, 1, 4, 5]
SLAU_solver(A, b)
```

a) ( )

$$\begin{cases} x + y = 2, \\ x - y = 3. \end{cases}$$

[110]:
```
A = Float64[1 1; 1 -1]
b = Float64[2, 3]
SLAU_solver(A, b)
```

[110]:
```
2-element Vector{Float64}:
  2.5
 -0.5
```

b) ( )

$$\begin{cases} x + y = 2, \\ 2x + 2y = 4. \end{cases}$$

[111]:
```julia
A = Float64[1 1; 2 2]
b = Float64[2, 4]
SLAU_solver(A, b)
```

c) (,)

$$\begin{cases} x + y = 2, \\ 2x + 2y = 5. \end{cases}$$

[113]:
```julia
A = Float64[1 1; 2 2]
b = Float64[2, 5]
SLAU_solver(A, b)
```

d) ( )

$$\begin{cases} x + y = 1, \\ 2x + 2y = 2, \\ 3x + 3y = 3. \end{cases}$$

[115]:
```julia
A = Float64[1 1; 2 2; 3 3]
b = Float64[1, 2, 3]
SLAU_solver(A, b)
```

e)

$$\begin{cases} x + y = 2, \\ 2x + y = 1, \\ x - y = 3. \end{cases}$$

[116]:
```julia
A = Float64[1 1; 2 1; 1 -1]
b = Float64[2, 1, 3]
SLAU_solver(A, b)
```

f)

$$\begin{cases} x + y = 2, \\ 2x + y = 1, \\ 3x + 2y = 3. \end{cases}$$

[118]:
```
A = Float64[1 1; 2 1; 3 2]
b = Float64[2, 1, 3]
SLAU_solver(A, b)
```

[118]:
```
2-element Vector{Float64}:
 -1.0
  3.0
```

2.

a)

$$\begin{cases} x + y + z = 2, \\ x - y - 2z = 3. \end{cases}$$

[121]:
```
A = Float64[1 1 1; 1 -1 -2]
b = Float64[2, 3]
SLAU_solver(A, b)
```

,

b)

$$\begin{cases} x + y + z = 2, \\ 2x + 2y - 3z = 4, \\ 3x + y + z = 1. \end{cases}$$

[123]:
```
A = Float64[1 1 1; 2 2 -3; 3 1 1]
b = Float64[2, 4, 1]
SLAU_solver(A, b)
```

[123]:
```
3-element Vector{Float64}:
 -0.5
  2.5
  0.0
```

c)

$$\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 1. \end{cases}$$

[124]:
```
A = Float64[1 1 1; 1 1 2; 2 2 3]
b = Float64[1, 0, 1]
SLAU_solver(A, b)
```

d)

$$\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 0. \end{cases}$$

```
[125]: A = Float64[1 1 1; 1 1 2; 2 2 3]
       b = Float64[1, 0, 0]
       SLAU_solver(A, b)
```

0.2.3

1.

```
[130]: function to_Diagonal(mtrx)
           s = size(mtrx)[1]
           ordDown = vcat([[if i == s; [i, j-1] else [i, j] end for j in i+1:s] for i␣
        ↪in 1:s]...)
           ordUP = vcat([[if i == s; [j-1, i] else [j, i] end for j in i+1:s] for i in␣
        ↪s:-1:1]...)
           answ = [[] for _ in 1:s]
           for i in ordDown
               if mtrx[i[2], i[1]] != 0
                   k = mtrx[i[2], i[1]] / mtrx[i[1], i[1]]
                   answ[i[2]] = [mtrx[i[2], j] - mtrx[i[1], j] * k for j in 1:s]
               end
           end
           for i in ordUP
               if mtrx[i[2], i[1]] != 0
                   k = mtrx[i[2], i[1]] / mtrx[i[1], i[1]]
                   answ[i[2]] = [mtrx[i[2], j] - mtrx[i[1], j] * k for j in 1:s]
               end
           end
           return copy(hcat(answ...)')
       end
```

```
[130]: to_Diagonal (generic function with 1 method)
```

```
[132]: # a)
       A = Float64[1 -2; -2 1]
       to_Diagonal(A)
```

```
[132]: 2×2 Matrix{Float64}:
        -3.0   0.0
         0.0  -3.0
```

```
[133]: # b)
       A = Float64[1 -2; -2 3]
       to_Diagonal(A)
```

```
[133]: 2×2 Matrix{Float64}:
        -0.333333   0.0
         0.0       -1.0
```

```
[134]: # c)
       A = Float64[1 -2 0; -2 1 2; 0 2 0]
       to_Diagonal(A)
```

```
[134]: 3×3 Matrix{Float64}:
         -3.0    0.0    4.0
        NaN     -Inf   NaN
          4.0    0.0   -4.0
```

2.

```
[135]: function mtrx_Function(A::Matrix, op)
           X = eigvecs(A)
           lamb = diagm(eigvals(A))
           lambfunc = [op(l) for l in lamb]
           answ = X^(-1)*lambfunc*X
           return answ
       end
```

```
[135]: mtrx_Function (generic function with 1 method)
```

```
[137]: # a)
       A = [1 -2; -2 1]; display(A^10)
       mtrx_Function(A, x -> x^10)
```

```
2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
```

```
[137]: 2×2 Matrix{Float64}:
         29525.0  -29524.0
        -29524.0   29525.0
```

```
[139]: # b)
       A = [5 -2; -2 5]
       mtrx_Function(A, x -> sqrt(x))
```

```
[139]: 2×2 Matrix{Float64}:
         2.1889   -0.45685
        -0.45685   2.1889
```

```
[140]: # c)
       A = [1 -2; -2 1]
       mtrx_Function(A, x -> cbrt(x))
```

```
[140]: 2×2 Matrix{Float64}:
         0.221125  -1.22112
        -1.22112    0.221125
```

```
[141]: # d)
       A = ComplexF64[1 2; 3 4]
       mtrx_Function(A, x -> sqrt(x))
```

```
[141]: 2×2 Matrix{ComplexF64}:
        0.553689+0.464394im  -0.889962+0.234276im
        -1.09755+0.288922im    1.76413+0.145754im
```

3. ,

```
[142]: A = [140 97 74 168 131; 97 106 89 131 36; 74 89 152 144 71; 168 131 144 52 142;␣
        ↪131 36 71 142 36]
       @btime eigvals(A)
```

```
  2.167  s (10 allocations: 2.59 KiB)
```

```
[142]: 5-element Vector{Float64}:
        -129.84037845927043
         -56.008181312078634
          42.75068638743729
          87.15844501190598
         541.9394283720058
```

. .

.

```
[143]: @btime diagm(eigvals(A))
```

```
  2.200  s (11 allocations: 2.84 KiB)
```

```
[143]: 5×5 Matrix{Float64}:
        -129.84     0.0       0.0       0.0       0.0
           0.0    -56.0082    0.0       0.0       0.0
           0.0      0.0      42.7507    0.0       0.0
           0.0      0.0       0.0      87.1584    0.0
           0.0      0.0       0.0       0.0     541.939
```

```
[144]: @btime blA = Bidiagonal(A, :L)
```

```
  313.248 ns (3 allocations: 224 bytes)
```

```
[144]: 5×5 Bidiagonal{Int64, Vector{Int64}}:
        140
         97   106
              89   152
                  144    52
                       142   36
```

0.2.4

1.   ,   .
                              ,   ,   .

```
[148]: function economicModel(M, y)
           x = (Diagonal(fill(1, 2)) - M)^(-1) * y
           return x
       end
```

```
[148]: economicModel (generic function with 1 method)
```

```
[149]: # a)
       A = [1 2; 3 4]
       Y = [2; 1]
       X = economicModel(A,Y); display(X)
       if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
           println("       ")
       else
           println("      ")
       end
```

```
2-element Vector{Float64}:
  0.6666666666666667
 -1.0
```

```
[150]: # b)
       A = [1 2; 3 4]*0.5
       Y = [2; 1]
       X = economicModel(A,Y); display(X)
       if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
           println("       ")
       else
           println("      ")
       end
```

```
2-element Vector{Float64}:
  0.5
 -1.75
```

```
[155]: # c)
       A = [1 2; 3 4]*0.1
       Y = [2; 1]
       X = economicModel(A,Y); display(X)
       if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
           println("        ")
       else
           println("       ")
       end
```

```
2-element Vector{Float64}:
 2.9166666666666665
 3.125
```

2. :   ,

```
[154]: function OnesModel(M)
           x = (Diagonal(fill(1, size(M,1))) - M)^(-1)
           return x
       end
```

```
[154]: OnesModel (generic function with 1 method)
```

```
[156]: # a)
       A = [1 2; 3 1]
       X = OnesModel(A); display(X)
       if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
           println("        ")
       else
           println("       ")
       end
```

```
2×2 Matrix{Float64}:
 -0.0  -0.333333
 -0.5   0.0
```

```
[157]: # b)
       A = [1 2; 3 1]*0.5
       X = OnesModel(A); display(X)
       if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
           println("        ")
       else
           println("       ")
       end
```

```
2×2 Matrix{Float64}:
 -0.4  -0.8
 -1.2  -0.4
```

[158]:
```
# c)
A = [1 2; 3 1]*0.1
X = OnesModel(A); display(X)
if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
    println("        ")
else
    println("       ")
end
```

```
2×2 Matrix{Float64}:
 1.2  0.266667
 0.4  1.2
```

3. :   ,
   1.   ,   ,   .

[160]:
```
function UnitarCheck(M)
    #=
        ,                                1,                    .
        :
                                          .
                                                 .
        A              ,     A^(-1)= A
        :    A^(-1) = A*,    A* -                 A,    A -       .
    =#
    x = 0
    try if conj(transpose(M)) == M^(-1) x += 1 end
    catch e
        return "        . $(-1)"
    end
    if size(M, 1) == size(M, 2) x += 1 end
    if abs(det(M)) == 1 x += 1 end

    if x == 3 return "       . $(x)" else "        . $(x)" end
end

eigenvalues(M) = eigen(M).values
```

[160]:
```
eigenvalues (generic function with 1 method)
```

[161]:
```
# a)
A = [1 2; 3 1]
X = eigenvalues(A); display(X)
```

```
if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
    println("       ")
else
    println("      ")
end
```

```
2-element Vector{Float64}:
 -1.4494897427831779
  3.4494897427831783
```

[162]:
```
# b)
A = [1 2; 3 1]*0.5
X = eigenvalues(A); display(X)
if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
    println("       ")
else
    println("      ")
end
```

```
2-element Vector{Float64}:
 -0.7247448713915892
  1.724744871391589
```

[163]:
```
# c)
A = [1 2; 3 1]*0.1
X = eigenvalues(A); display(X)
if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
    println("       ")
else
    println("      ")
end
```

```
2-element Vector{Float64}:
 -0.14494897427831785
  0.34494897427831783
```

[164]:
```
# d)
A = [0.1 0.2 0.3; 0.0 0.1 0.2; 0.0 0.1 0.3]
X = eigenvalues(A); display(X)
if mapreduce(z -> if z < 0 1 else 0 end, +, X) > 0
    println("       ")
else
    println("      ")
end
```

```
3-element Vector{Float64}:
 0.02679491924311228
 0.1
 0.37320508075688774
```

[ ]:

```
```