In [1]:
```
# Массив 4x3 со случайными целыми числами (от 1 до 20):
a = rand(1:20,(4,3))
```

Out[1]:
```
4×3 Array{Int64,2}:
 20  14   5
 12  13  14
 17  19  19
 12   5   8
```

In [2]:
```
# Поэлементная сумма:
sum(a)
```

Out[2]:
```
158
```

In [3]:
```
# Поэлементная сумма по столбцам:
sum(a,dims=1)
```

Out[3]:
```
1×3 Array{Int64,2}:
 61  51  46
```

In [4]:
```
# Поэлементная сумма по строкам:
sum(a,dims=2)
```

Out[4]:
```
4×1 Array{Int64,2}:
 39
 39
 55
 25
```

In [5]:
```
# Поэлементное произведение:
prod(a)
```

Out[5]:
```
9006955776000
```

In [6]:
```
# Поэлементное произведение по столбцам:
prod(a,dims=1)
```

Out[6]:
```
1×3 Array{Int64,2}:
 48960  17290  10640
```

In [7]:
```
# Поэлементное произведение по строкам:
prod(a,dims=2)
```

Out[7]:
```
4×1 Array{Int64,2}:
 1400
 2184
 6137
  480
```

In [8]:
```
# Подключение пакета Statistics:
using Statistics
```

In [9]:
```
# Вычисление среднего значения массива:
mean(a)
```

Out[9]:
```
13.166666666666666
```

In [10]:
```
# Среднее по столбцам:
mean(a,dims=1)
```

Out[10]:
```
1×3 Array{Float64,2}:
 15.25  12.75  11.5
```

In [11]:
```
# Среднее по строкам:
mean(a,dims=2)
```

Out[11]:
```
4×1 Array{Float64,2}:
 13.0
 13.0
 18.333333333333332
  8.333333333333334
```

In [12]:
```
# Подключение пакета LinearAlgebra:
using LinearAlgebra
```

In [13]:
```
# Массив 4x4 со случайными целыми числами (от 1 до 20):
b = rand(1:20,(4,4))
```

Out[13]:
```
4×4 Array{Int64,2}:
 16  16  17  11
 15  17  10  14
 19  16   7  18
  4  14  15  12
```

In [14]:
```
# Транспонирование:
transpose(b)
```

Out[14]:
```
4×4 Transpose{Int64,Array{Int64,2}}:
 16  15  19   4
 16  17  16  14
 17  10   7  15
 11  14  18  12
```

In [15]:
```
# След матрицы (сумма диагональных элементов):
tr(b)
```

Out[15]:
```
52
```

In [16]:
```
# Извлечение диагональных элементов как массив:
diag(b)
```

Out[16]:
```
4-element Array{Int64,1}:
 16
 17
  7
 12
```

In [17]:
```
# Ранг матрицы:
rank(b)
```

Out[17]:
```
4
```

In [18]:
```
# Инверсия матрицы (определение обратной матрицы):
inv(b)
```

Out[18]:
```
4×4 Array{Float64,2}:
  0.0814069  -0.0558786   0.0441072  -0.0755921
 -0.106652    0.397249   -0.214579   -0.0438236
  0.10268    -0.175011    0.0416962   0.047511
 -0.0310594  -0.226067    0.18352     0.100269
```

In [19]:
```
# Определитель матрицы:
det(b)
```

Out[19]:
```
-7051.000000000001
```

In [20]:
```
# Псевдобратная функция для прямоугольных матриц:
pinv(a)
```

Out[20]:
```
3×4 Array{Float64,2}:
  0.0320428  -0.0173001  -0.0385881   0.101895
  0.0544927   0.0110326   0.050597   -0.173533
 -0.0814197   0.0289178   0.0193037   0.0794348
```

In [21]:
```
# Создание вектора X:
X = [2, 4, -5]
# Вычисление евклидовой нормы:
norm(X)
```

Out[21]:
```
6.708203932499369
```

In [22]:
```
# Вычисление p-нормы:
p = 1
norm(X,p)
```

Out[22]:
```
11.0
```

In [23]:
```
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
norm(X-Y)
```

Out[23]:
```
9.486832980505138
```

In [24]:
```
# Проверка по базовому определению:
sqrt(sum((X-Y).^2))
```

Out[24]:
```
9.486832980505138
```

In [25]:
```
# Угол между двумя векторами:
acos((transpose(X)*Y)/(norm(X)*norm(Y)))
```

Out[25]:
```
2.4404307889469252
```

In [26]:
```
# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
# Вычисление Евклидовой нормы:
opnorm(d)
```

Out[26]:
```
7.147682841795258
```

In [27]:
```
# Вычисление p-нормы:
p=1
opnorm(d,p)
```

Out[27]:
```
8.0
```

In [28]:
```
# Поворот на 180 градусов:
rot180(d)
```

Out[28]:
```
3×3 Array{Int64,2}:
 0   1  -2
 3   2  -1
 2  -4   5
```

In [29]:
```
# Переворачивание строк:
reverse(d,dims=1)
```

Out[29]:
```
3×3 Array{Int64,2}:
 -2   1  0
 -1   2  3
  5  -4  2
```

In [30]:
```
# Переворачивание столбцов
reverse(d,dims=2)
```

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

In [31]:
```
# Матрица 2x3 со случайными целыми значениями от 1 до 10:
A = rand(1:10,(2,3))
# Матрица 3x4 со случайными целыми значениями от 1 до 10:
B = rand(1:10,(3,4))
# Произведение матриц A и B:
A*B
```

Out[31]:
```
2×4 Array{Int64,2}:
 73  129  146  96
 36   84   80  85
```

In [32]:
```
# Единичная матрица 3x3:
Matrix{Int}(I, 3, 3)
```

Out[32]:
```
3×3 Array{Int64,2}:
 1  0  0
 0  1  0
 0  0  1
```

In [33]:
```
# Скалярное произведение векторов X и Y:
```

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

Out[33]:
```
-17
```

In [34]:
```
# тоже скалярное произведение:
X'Y
```

Out[34]:
```
-17
```

In [35]:
```
# Задаём квадратную матрицу 3x3 со случайными значениями:
A = rand(3, 3)
# Задаём единичный вектор:
x = fill(1.0, 3)
# Задаём вектор b:
b = A*x
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
A\b
```

Out[35]:
```
3-element Array{Float64,1}:
 1.0000000000000002
 1.0000000000000002
 0.9999999999999999
```

In [36]:
```
# LU-факторизация:
Alu = lu(A)
```

Out[36]:
```
LU{Float64,Array{Float64,2}}
L factor:
3×3 Array{Float64,2}:
 1.0         0.0       0.0
 0.0215489   1.0       0.0
 0.755189   -0.410268  1.0
U factor:
3×3 Array{Float64,2}:
 0.745778  0.399112  0.556115
 0.0       0.562748  0.442716
 0.0       0.0       0.463535
```

In [37]:
```
# Матрица перестановок:
Alu.P
```

Out[37]:
```
3×3 Array{Float64,2}:
 0.0  1.0  0.0
 1.0  0.0  0.0
 0.0  0.0  1.0
```

In [38]:
```
# Вектор перестановок:
Alu.p
```

Out[38]:
```
3-element Array{Int64,1}:
 2
 1
 3
```

In [39]:
```
# Матрица L:
```

```
Alu.L
```

Out[39]:
```
3×3 Array{Float64,2}:
 1.0         0.0       0.0
 0.0215489   1.0       0.0
 0.755189   -0.410268  1.0
```

In [40]:
```
# Матрица U:
Alu.U
```

Out[40]:
```
3×3 Array{Float64,2}:
 0.745778  0.399112  0.556115
 0.0       0.562748  0.442716
 0.0       0.0       0.463535
```

In [41]:
```
# Решение СЛАУ через матрицу A:
A\b
# Решение СЛАУ через объект факторизации:
Alu\b
```

Out[41]:
```
3-element Array{Float64,1}:
 1.0000000000000002
 1.0000000000000002
 0.9999999999999999
```

In [42]:
```
# Детерминант матрицы A:
det(A)
# Детерминант матрицы A через объект факторизации:
det(Alu)
```

Out[42]:
```
-0.19453870808593457
```

In [43]:
```
# QR-факторизация:
Aqr = qr(A)
```

Out[43]:
```
LinearAlgebra.QRCompactWY{Float64,Array{Float64,2}}
Q factor:
3×3 LinearAlgebra.QRCompactWYQ{Float64,Array{Float64,2}}:
 -0.0171936   0.950597   0.309951
 -0.79789     0.173774  -0.577212
 -0.602558   -0.257231   0.755484
R factor:
3×3 Array{Float64,2}:
 -0.934688  -0.370767  -0.874456
  0.0        0.594335   0.34833
  0.0        0.0        0.350193
```

In [44]:
```
# Матрица Q:
Aqr.Q
```

Out[44]:
```
3×3 LinearAlgebra.QRCompactWYQ{Float64,Array{Float64,2}}:
 -0.0171936   0.950597   0.309951
 -0.79789     0.173774  -0.577212
 -0.602558   -0.257231   0.755484
```

In [45]:
```
# Матрица R:
Aqr.R
```

Out[45]:
```
3×3 Array{Float64,2}:
 -0.934688  -0.370767  -0.874456
  0.0        0.594335   0.34833
  0.0        0.0        0.350193
```

In [46]:
```
# Проверка, что матрица Q - ортогональная:
Aqr.Q'*Aqr.Q
```

Out[46]:
```
3×3 Array{Float64,2}:
  1.0          5.55112e-17  -5.55112e-17
  5.55112e-17  1.0           1.38778e-16
 -5.55112e-17  1.38778e-16   1.0
```

In [47]:
```
# Симметризация матрицы A:
Asym = A + A'
```

Out[47]:
```
3×3 Array{Float64,2}:
 0.0321414  1.31713   1.0179
 1.31713    0.798223  0.626642
 1.0179     0.626642  1.40375
```

In [48]:
```
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(Asym)
```

Out[48]:
```
Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
values:
3-element Array{Float64,1}:
 -1.0514690842795993
  0.5398757000023702
  2.745706611055038
vectors:
3×3 Array{Float64,2}:
 -0.828599  -0.219744  -0.514914
  0.518478  -0.648172  -0.557722
  0.211196   0.7291    -0.651007
```

In [49]:
```
# Собственные значения:
AsymEig.values
```

Out[49]:
```
3-element Array{Float64,1}:
 -1.0514690842795993
  0.5398757000023702
  2.745706611055038
```

In [50]:
```
#Собственные векторы:
AsymEig.vectors
```

Out[50]:
```
3×3 Array{Float64,2}:
 -0.828599  -0.219744  -0.514914
  0.518478  -0.648172  -0.557722
  0.211196   0.7291    -0.651007
```

In [51]:
```
# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym
```

Out[51]:
```
3×3 Array{Float64,2}:
  1.0          -4.96998e-16  6.41848e-17
 -2.10942e-15   1.0          4.21885e-15
  1.55431e-15   1.44329e-15  1.0
```

In [52]:
```
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
# Симметризация матрицы:
Asym = A + A'
# Проверка, является ли матрица симметричной:
issymmetric(Asym)
```

Out[52]:
```
true
```

In [53]:
```
# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)
```

Out[53]:
```
false
```

In [54]:
```
# Явно указываем, что матрица является симметричной:
Asym_explicit = Symmetric(Asym_noisy)
```

Out[54]:
```
1000×1000 Symmetric{Float64,Array{Float64,2}}:
  2.54109    -1.66412    -1.36648    …  -1.21036    -3.32486    -2.31201
 -1.66412    -0.556662   -1.10184        1.35204    -0.941845    1.07017
 -1.36648    -1.10184     0.738523       0.0710064   1.23134     0.321599
 -2.06341     1.27252     2.15785        3.09151    -2.79786    -1.56803
  0.0759214  -1.59261    -2.43744        1.81205     1.73542     0.209564
 -0.648086    2.64896    -0.894978   …  -0.97634     1.71194     0.890063
 -2.00471     0.214989   -0.0890087      0.424532   -1.25779     1.41538
 -0.682275   -0.129745    0.297798       1.96514     1.28319     0.441039
  2.87796     2.64717     1.06876        2.20326    -2.65283    -1.67481
 -1.70872     0.1125      0.214889       1.65068    -1.16746    -0.773745
 -1.57843     1.07353     0.528109   …  -0.10094     2.05464     0.327137
  1.17833    -1.5881      0.753564      -0.0167435   0.112799   -0.104341
 -1.58094    -2.40645    -0.94734        1.47916     0.574298   -0.814481
  ⋮                                   ⋱
 -0.558015   -0.62611     1.81584       -1.67451    -1.65487     0.0502965
 -0.528791   -0.701032    1.0775         1.50585    -1.74712     1.9705
  0.614954   -0.700055   -0.326399   …   0.03108    -0.082049   -0.0323835
 -0.88154     0.929196    1.75481       -0.176258    0.275194    0.716849
  3.41611     1.0575      0.996495      -0.399122   -0.832901   -1.54032
  0.568058    0.754153    1.40639        0.266481    0.638609   -2.03444
 -1.77952     0.262096    1.66668       -0.0606716   0.0669151   1.48225
 -1.66134    -1.57498     0.305506   …  -1.55174    -0.100931   -2.04096
  2.13899     1.04025    -2.02098        1.35944     0.612215   -3.20208
 -1.21036     1.35204     0.0710064      1.51728     0.265085   -0.289733
 -3.32486    -0.941845    1.23134        0.265085   -0.867382    0.763197
 -2.31201     1.07017     0.321599      -0.289733    0.763197   -4.2135
```

In [55]:
```
using BenchmarkTools
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@btime eigvals(Asym);
```

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

In [56]:
```
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы:
@btime eigvals(Asym_noisy);
```

```
  1.350 s (13 allocations: 7.92 MiB)
```

In [57]:
```
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы,
# для которой явно указано, что она симметричная:
@btime eigvals(Asym_explicit);
```

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

In [58]:
```
# Трёхдиагональная матрица 1000000 x 1000000:
n = 1000000;
A = SymTridiagonal(randn(n), randn(n-1))
# Оценка эффективности выполнения операции по нахождению
# собственных значений:
@btime eigmax(A)
```

```
  720.857 ms (38 allocations: 183.11 MiB)
```

Out[58]:
```
6.337207988066645
```

In [59]:
```
B = Matrix(A)
```

```
OutOfMemoryError()

Stacktrace:
 [1] Array at .\boot.jl:408 [inlined]
 [2] Array at .\boot.jl:416 [inlined]
 [3] zeros at .\array.jl:525 [inlined]
 [4] zeros at .\array.jl:521 [inlined]
 [5] Array{Float64,2}(::SymTridiagonal{Float64,Array{Float64,1}}) at D:\buildbot\w
orker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\tridiag.j
l:127
 [6] Array{T,2} where T(::SymTridiagonal{Float64,Array{Float64,1}}) at D:\buildbot
\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\tridiag.
jl:141
 [7] top-level scope at In[59]:1
 [8] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
 [9] execute_code(::String, ::String) at C:\Users\Admin\.julia\packages\IJulia\rWZ
9e\src\execute_request.jl:27
 [10] execute_request(::ZMQ.Socket, ::IJulia.Msg) at C:\Users\Admin\.julia\package
s\IJulia\rWZ9e\src\execute_request.jl:86
 [11] #invokelatest#1 at .\essentials.jl:710 [inlined]
 [12] invokelatest at .\essentials.jl:709 [inlined]
 [13] eventloop(::ZMQ.Socket) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\e
ventloop.jl:8
 [14] (::IJulia.var"#15#18")() at .\task.jl:356
```

In [60]:
```
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
# Единичный вектор:
x = fill(1, 3)
# Задаём вектор b:
b = Arational*x
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
Arational\b
```

Out[60]:
```
3-element Array{Rational{BigInt},1}:
 1//1
 1//1
 1//1
```

In [61]:
```
# LU-разложение:
lu(Arational)
```

Out[61]:
```
LU{Rational{BigInt},Array{Rational{BigInt},2}}
L factor:
3×3 Array{Rational{BigInt},2}:
 1//1  0//1   0//1
 1//2  1//1   0//1
 5//8  9//20  1//1
U factor:
3×3 Array{Rational{BigInt},2}:
 4//5  3//10    3//5
 0//1  1//4     1//5
 0//1  0//1   -13//200
```

In [ ]:
```

```

In [62]:
```
using LinearAlgebra
# 1. Задайте вектор v. Умножьте вектор v скалярно сам на себя и сохраните результа
v=[3, 4 ,5]
dot_v=dot(v,v)
dot_v
```

Out[62]:
```
50
```

In [63]:
```
#Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной
outer_v=cross(v,v)
outer_v
```

Out[63]:
```
3-element Array{Int64,1}:
 0
 0
 0
```

In [64]:
```
#2 Решить СЛАУ с двумя неизвестными.
A=[1 1 ; 1 -1]
b=[2 ; 3]
A\b
```

Out[64]:
```
2-element Array{Float64,1}:
  2.5
 -0.5
```

In [65]:
```
A=[1 1 ; 2 2]
b=[2 ; 4]
lu(A)
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:21 [inlined]
 [3] lu!(::Array{Float64,2}, ::Val{true}; check::Bool) at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:85
 [4] #lu#136 at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:273 [inlined]
 [5] lu at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:272 [inlined] (repeats 2 times)
 [6] top-level scope at In[65]:3
 [7] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
 [8] execute_code(::String, ::String) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:27
 [9] execute_request(::ZMQ.Socket, ::IJulia.Msg) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:86
 [10] #invokelatest#1 at .\essentials.jl:710 [inlined]
 [11] invokelatest at .\essentials.jl:709 [inlined]
 [12] eventloop(::ZMQ.Socket) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\eventloop.jl:8
 [13] (::IJulia.var"#15#18")() at .\task.jl:356
```

In [66]:

```
using NLsolve

function f!(F, v)
            x = v[1]
            y = v[2]
            F[1] = x + y - 2
            F[2] = 2*x + 2*y - 4
    end
res=nlsolve(f!, [0.0; 0.0])
res.zero
```

Out[66]:

```
2-element Array{Float64,1}:
 0.9132041931152343
 1.0867958068847656
```

In [67]:

```
A=[1 1 ; 2 2]
b=[2 ; 5]
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:21 [inlined]
 [3] lu!(::Array{Float64,2}, ::Val{true}; check::Bool) at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:85
 [4] #lu#136 at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:273 [inlined]
 [5] lu at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:272 [inlined] (repeats 2 times)
 [6] \(::Array{Int64,2}, ::Array{Int64,1}) at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\generic.jl:1116
 [7] top-level scope at In[67]:3
 [8] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
 [9] execute_code(::String, ::String) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:27
 [10] execute_request(::ZMQ.Socket, ::IJulia.Msg) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:86
 [11] #invokelatest#1 at .\essentials.jl:710 [inlined]
 [12] invokelatest at .\essentials.jl:709 [inlined]
 [13] eventloop(::ZMQ.Socket) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\eventloop.jl:8
 [14] (::IJulia.var"#15#18")() at .\task.jl:356
```

In [68]:

```
using NLsolve

function f!(F, v)
          x = v[1]
          y = v[2]
          F[1] = x + y - 2
          F[2] = 2*x + 2*y - 5
    end
res=nlsolve(f!, [0.0; 0.0])
res.zero
```

Out[68]:

```
2-element Array{Float64,1}:
 -7.629159860605175
 10.029159860627605
```

In [69]:

```
A=[1 1 ; 2 2 ; 1 -1]
b=[2 ; 1 ; 3]
A\b
```

Out[69]:

```
2-element Array{Float64,1}:
  1.9000000000000001
 -1.0999999999999996
```

In [70]:

```
A=[1 1 ; 2 1 ; 3 2]
b=[2 ; 1 ; 3]
A\b
```

Out[70]:

```
2-element Array{Float64,1}:
 -0.9999999999999989
  2.9999999999999982
```

In [71]:

```
#решение СЛАУ с тремя неизместными
```

In [72]:
```
A=[1 1 1 ; 1 -1 -2]
b=[2 ; 3]
A\b
```

Out[72]:
```
3-element Array{Float64,1}:
  2.2142857142857144
  0.35714285714285704
 -0.5714285714285712
```

In [73]:
```
A=[1 1 1 ; 2 2 -3 ; 3 1 1]
b=[2 ; 4 ; 1]
A\b
```

Out[73]:
```
3-element Array{Float64,1}:
 -0.5
  2.5
  0.0
```

In [74]:
```
A=[1 1 1 ; 1 1 2 ; 2 2 3]
b=[1 ; 0 ; 1]
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:21 [inlined]
 [3] lu!(::Array{Float64,2}, ::Val{true}; check::Bool) at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:85
 [4] #lu#136 at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:273 [inlined]
 [5] lu at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:272 [inlined] (repeats 2 times)
 [6] \(::Array{Int64,2}, ::Array{Int64,1}) at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\generic.jl:1116
 [7] top-level scope at In[74]:3
 [8] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
 [9] execute_code(::String, ::String) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:27
 [10] execute_request(::ZMQ.Socket, ::IJulia.Msg) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:86
 [11] #invokelatest#1 at .\essentials.jl:710 [inlined]
 [12] invokelatest at .\essentials.jl:709 [inlined]
 [13] eventloop(::ZMQ.Socket) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\eventloop.jl:8
 [14] (::IJulia.var"#15#18")() at .\task.jl:356
```

In [75]:
```
using NLsolve

function f!(F, v)
    x = v[1]
    y = v[2]
    z = v[3]
    F[1] = x + y  + z - 1
    F[2] = x + y + 2*z
    F[3] = 2*x + 2*y + 3*z -1
end
res=nlsolve(f!, [0.0; 0.0; 0.0])
res.zero
```

Out[75]:
```
3-element Array{Float64,1}:
  2.1541665449153977
 -0.154166544912257
 -1.0000000000019942
```

In [76]:
```
A=[1 1 1 ; 1 1 2 ; 2 2 3]
b=[1 ; 0 ; 0]
A\b
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\factorization.jl:21 [inlined]
 [3] lu!(::Array{Float64,2}, ::Val{true}; check::Bool) at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:85
 [4] #lu#136 at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:273 [inlined]
 [5] lu at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\lu.jl:272 [inlined] (repeats 2 times)
 [6] \(::Array{Int64,2}, ::Array{Int64,1}) at D:\buildbot\worker\package_win64\build\usr\share\julia\stdlib\v1.5\LinearAlgebra\src\generic.jl:1116
 [7] top-level scope at In[76]:3
 [8] include_string(::Function, ::Module, ::String, ::String) at .\loading.jl:1091
 [9] execute_code(::String, ::String) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:27
 [10] execute_request(::ZMQ.Socket, ::IJulia.Msg) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\execute_request.jl:86
 [11] #invokelatest#1 at .\essentials.jl:710 [inlined]
 [12] invokelatest at .\essentials.jl:709 [inlined]
 [13] eventloop(::ZMQ.Socket) at C:\Users\Admin\.julia\packages\IJulia\rWZ9e\src\eventloop.jl:8
 [14] (::IJulia.var"#15#18")() at .\task.jl:356
```

In [77]:
```
function f!(F, v)
    x = v[1]
    y = v[2]
    z = v[3]
    F[1] = x + y  + z - 1
    F[2] = x + y + 2*z
    F[3] = 2*x + 2*y + 3*z
end
res=nlsolve(f!, [0.0; 0.0; 0.0])
res.zero
```

Out[77]:
```
3-element Array{Float64,1}:
 -1.3210360630766114
  2.987702729750822
 -0.9999999999975434
```

In [78]:
```
#Приведите приведённые ниже матрицы к диагональному виду
```

In [79]:
```
A=[ 1 -2 ; -2 1 ]
Alu = lu(A)
```

Out[79]:
```
LU{Float64,Array{Float64,2}}
L factor:
2×2 Array{Float64,2}:
  1.0  0.0
 -0.5  1.0
U factor:
2×2 Array{Float64,2}:
 -2.0   1.0
  0.0  -1.5
```

In [80]:
```
A=[ 1 -2 ; -2 3 ]
Alu = lu(A)
```

Out[80]:
```
LU{Float64,Array{Float64,2}}
L factor:
2×2 Array{Float64,2}:
  1.0  0.0
 -0.5  1.0
U factor:
2×2 Array{Float64,2}:
 -2.0   3.0
  0.0  -0.5
```

In [81]:
```
A=[ 1 -2 0 ; -2 1 2 ; 0 2 0]
Alu = lu(A)
```

Out[81]:
```
LU{Float64,Array{Float64,2}}
L factor:
3×3 Array{Float64,2}:
  1.0   0.0   0.0
 -0.0   1.0   0.0
 -0.5  -0.75  1.0
U factor:
3×3 Array{Float64,2}:
 -2.0  1.0  2.0
  0.0  2.0  0.0
  0.0  0.0  1.0
```

In [82]:
```
################
```

In [83]:
```
A = [1 -2 ; -2 1]
A^10
```

Out[83]:
```
2×2 Array{Int64,2}:
  29525  -29524
 -29524   29525
```

In [84]:
```
B= [5 -2; -2 5]
B^(1/2)
```

Out[84]:
```
2×2 Symmetric{Float64,Array{Float64,2}}:
  2.1889   -0.45685
 -0.45685   2.1889
```

In [85]:
```
C = [1 -2 ; -2 1]
C^(1/3)
```

Out[85]:
```
2×2 Symmetric{Complex{Float64},Array{Complex{Float64},2}}:
  0.971125+0.433013im  -0.471125+0.433013im
 -0.471125+0.433013im   0.971125+0.433013im
```

In [86]:
```
D = [1 2 ; 2 3]
D^(1/2)
```

Out[86]:
```
2×2 Symmetric{Complex{Float64},Array{Complex{Float64},2}}:
 0.568864+0.351578im  0.920442-0.217287im
 0.920442-0.217287im   1.48931+0.134291im
```

In [87]:
```
using BenchmarkTools
```

In [88]:
```
A=[140 97 74 168 131 ; 97 106 89 131 36 ; 74 89 152 144 71 ; 168 131 144 54 142 ;
@btime eig_A = Diagonal(eigvals(A))
```

Out[88]:
```
  4.329 μs (11 allocations: 2.81 KiB)
5×5 Diagonal{Float64,Array{Float64,1}}:
 -128.493     ·        ·        ·         ·
    ·     -55.8878    ·        ·         ·
    ·        ·      42.7522    ·         ·
    ·        ·        ·      87.1611     ·
    ·        ·        ·        ·      542.468
```

In [89]:
```
@btime Alu=lu(A)
```

Out[89]:
```
  887.037 ns (3 allocations: 448 bytes)
LU{Float64,Array{Float64,2}}
L factor:
5×5 Array{Float64,2}:
 1.0        0.0        0.0       0.0       0.0
 0.779762   1.0        0.0       0.0       0.0
 0.440476  -0.47314    1.0       0.0       0.0
 0.833333   0.183929  -0.556312  1.0       0.0
 0.577381  -0.459012  -0.189658  0.897068  1.0
U factor:
5×5 Array{Float64,2}:
 168.0  131.0     144.0      54.0     142.0
   0.0  -66.1488  -41.2857   99.8929  -74.7262
   0.0    0.0      69.0375  167.478   -26.9035
   0.0    0.0       0.0     197.797    11.4442
   0.0    0.0       0.0       0.0     -95.657
```

In [90]:
```
E=[1 1 ; 1 1]
A=[1 2 ; 3 4]
b=[3 ; 3]
(E-A)\b
```

Out[90]:
```
2-element Array{Float64,1}:
  3.0
 -3.0
```

In [91]:
```
E=[1 1 ; 1 1]
A=[1 2 ; 3 4]
b=[3 ; 3]
(E-(1/2)*A)\b
```

Out[91]:
```
2-element Array{Float64,1}:
  6.0
 -6.0
```

In [92]:
```
E=[1 1 ; 1 1]
A=[1 2 ; 3 4]
b=[3 ; 3]
(E-(1/10)*A)\b
```

Out[92]:
```
2-element Array{Float64,1}:
  29.999999999999957
 -29.99999999999995
```

In [93]:
```
E = [1 0 ; 0 1]
A =  [ 1 2 ; 3 1]
inv(E-A)
```

Out[93]:
```
2×2 Array{Float64,2}:
 -0.0  -0.333333
 -0.5   0.0
```

In [94]:
```
inv(E-(1/2)*A)
```

Out[94]:
```
2×2 Array{Float64,2}:
 -0.4  -0.8
 -1.2  -0.4
```

In [95]:
```
inv(E-(1/10)*A)
```

Out[95]:
```
2×2 Array{Float64,2}:
 1.2  0.266667
 0.4  1.2
```

In [96]:
```
eigvals(A)
```

Out[96]:
```
2-element Array{Float64,1}:
 -1.4494897427831779
  3.4494897427831783
```

In [97]:
```
eigvals(0.5*A)
```

Out[97]:
```
2-element Array{Float64,1}:
 -0.7247448713915892
  1.724744871391589
```

In [98]:
```
eigvals(0.1*A)
```

Out[98]:
```
2-element Array{Float64,1}:
 -0.14494897427831785
  0.34494897427831783
```

In [99]:
```
A = [0.1 0.2 0.3 ; 0 0.1 0.2 ; 0 0.1 0.3]
eigvals(A)
```

Out[99]:
```
3-element Array{Float64,1}:
 0.02679491924311228
 0.1
 0.37320508075688774
```

In [ ]: