

Презентация по лабораторной работе №4

Компьютерный практикум по статистическому анализу данных

Еюбоглу Тимур

25 октября 2025 г.

Российский университет дружбы народов, Москва, Россия

- Еюбоглу Тимур
- Студент группы НПИбд-01-22
- Студ. билет 1032224357
- Российский университет дружбы народов имени Патриса Лумумбы

Цель лабораторной работы

- Изучить возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Выполнение лабораторной работы

Поэлементные операции над многомерными массивами

Для матрицы 4×3 рассмотрим поэлементные операции сложения и произведения её элементов:

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

```
[1]: 4x3 Matrix{Int64}:
 17   6  18
 18  16  14
  5  19  18
  6   1   5
```

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

```
[2]: 143
```

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

```
[3]: 1x3 Matrix{Int64}:
 46  42  55
```

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

```
[4]: 4x1 Matrix{Int64}:
 41
 48
 42
 12
```

```
[6]: # Поэлементное произведение:
prod(a)
```

```
[6]: 379761177600
```

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

```
[7]: 1x3 Matrix{Int64}:
 9180  1824  22680
```

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

```
[8]: 4x1 Matrix{Int64}:
 1836
 4032
 4032
 1710
```

Поэлементные операции над многомерными массивами

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics:

```
[10]: # Подключение пакета Statistics:  
using Statistics
```

```
# Вычисление среднего значения массива:  
mean(a)
```

```
[10]: 11.916666666666666
```

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

```
[11]: 1x3 Matrix{Float64}:  
11.5 10.5 13.75
```

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

```
[12]: 4x1 Matrix{Float64}:  
13.66666666666666  
16.0  
14.0  
4.0
```

Транспонирование, след, ранг, определитель и инверсия матрицы

▼ 2. Транспонирование, след, ранг, определитель и инверсия матрицы

```
[14]: # Подключение пакета LinearAlgebra:  
using LinearAlgebra  
  
# Массив 4x4 со случайными целыми числами (от 1 до 20):  
b = rand(1:20,(4,4))
```

```
[14]: 4x4 Matrix{Int64}:  
      2  20  16  4  
      4  11  10  3  
     10   3  14  6  
     19   8   5  14
```

```
[15]: # Транспонирование:  
transpose(b)
```

```
[15]: 4x4 transpose(::Matrix{Int64}) with eltype Int64:  
      2   4  10  19  
     20  11   3   8  
     16  10  14   5  
      4   3   6  14
```

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

```
[16]: 41
```

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

```
[17]: 4-element Vector{Int64}:  
      2  
     11  
     14  
     14
```

Транспонирование, след, ранг, определитель и инверсия матрицы

```
[18]: # Ранг матрицы:  
rank(b)
```

```
[18]: 4
```

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

```
[19]: 4x4 Matrix{Float64}:  
-0.342422 0.650064 -0.068699 -0.0120223  
-0.052383 0.203521 -0.0888793 0.00944611  
0.0517389 -0.0944611 0.0918849 -0.0339201  
0.47617 -0.964792 0.111207 0.0944611
```

```
[20]: # Определитель матрицы:  
det(b)
```

```
[20]: -4657.999999999995
```

```
[21]: # Псевдобротная функция для прямоугольных матриц:  
pinv(a)
```

```
[21]: 3x4 Matrix{Float64}:  
0.00319438 0.0652316 -0.0564106 0.00893  
-0.0643184 0.0614736 0.0222671 -0.0207411  
0.0684769 -0.0828166 0.0473269 0.0149927
```

Вычисление нормы векторов и матриц, повороты, вращения

3. Вычисление нормы векторов и матриц, повороты, вращения

```
[22]: # Создание вектора X:  
X = [2, 4, -5]
```

```
[22]: 3-element Vector{Int64}:  
      2  
      4  
     -5
```

```
[23]: # Вычисление евклидовой нормы:  
norm(X)
```

```
[23]: 6.708203932499369
```

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

```
[24]: 11.0
```

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

```
[25]: 9.486832980505138
```

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

```
[26]: 9.486832980505138
```

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

```
[27]: 2.4404307889469252
```

Вычисление нормы векторов и матриц, повороты, вращения

Вычисление нормы для двумерной матрицы:

```
[28]: # Создание матрицы:  
d = [5 -4 2 ; -1 2 3; -2 1 0]
```

```
[28]: 3x3 Matrix{Int64}:  
      5  -4   2  
     -1   2   3  
     -2   1   0
```

```
[29]: # Вычисление Евклидовой нормы:  
opnorm(d)
```

```
[29]: 7.147682841795258
```

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

```
[30]: 8.0
```

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

```
[31]: 3x3 Matrix{Int64}:  
      0   1   -2  
      3   2   -1  
      2   -4   5
```

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

```
[32]: 3x3 Matrix{Int64}:  
      -2   1   0  
      -1   2   3  
      5  -4   2
```

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

```
[33]: 3x3 Matrix{Int64}:  
      2  -4   5  
      3   2  -1  
      0   1  -2
```

Матричное умножение, единичная матрица, скалярное произведение

4. Матричное умножение, единичная матрица, скалярное произведение

```
[34]: # Матрица 2x3 со случайными целыми значениями от 1 до 10:  
A = rand(1:10,(2,3))
```

```
[34]: 2x3 Matrix{Int64}:  
 2 1 1  
 8 7 10
```

```
[35]: # Матрица 3x4 со случайными целыми значениями от 1 до 10:  
B = rand(1:10,(3,4))
```

```
[35]: 3x4 Matrix{Int64}:  
 3 4 10 8  
 2 6 3 9  
 1 1 2 2
```

```
[36]: # Произведение матриц A и B:  
A*B
```

```
[36]: 2x4 Matrix{Int64}:  
 9 15 25 27  
 48 84 121 147
```

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

```
[37]: 3x3 Matrix{Int64}:  
 1 0 0  
 0 1 0  
 0 0 1
```

```
[38]: # Скалярное произведение векторов X и Y:  
X = [2, 4, -5]  
Y = [1, -1, 3]  
dot(X,Y)
```

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

```
[39]: # тоже скалярное произведение:  
X'*Y
```

```
[39]: -17
```

Факторизация. Специальные матричные структуры

5. Факторизация. Специальные матричные структуры

```
[40]: # Задаём квадратную матрицу 3x3 со случайными значениями:  
A = rand(3, 3)
```

```
[40]: 3x3 Matrix{Float64}:  
0.882431 0.943103 0.605134  
0.640795 0.0451344 0.635614  
0.690356 0.246666 0.592887
```

```
[41]: # Задаём единичный вектор:  
x = fill(1.0, 3)
```

```
[41]: 3-element Vector{Float64}:  
1.0  
1.0  
1.0
```

```
[42]: # Задаём вектор b:  
b = A*x
```

```
[42]: 3-element Vector{Float64}:  
2.430669297689036  
1.3215433484125134  
1.5299083315262747
```

```
[43]: # Решение исходного уравнения получаем с помощью функции \  
# (убеждаемся, что x - единичный вектор):  
A\b
```

```
[43]: 3-element Vector{Float64}:  
1.0000000000000069  
0.9999999999999984  
0.9999999999999929
```

Факторизация. Специальные матричные структуры

Julia позволяет вычислять LU-факторизацию и определяет составной тип факторизации для его хранения:

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

```
[44]: LU{Float64, Matrix{Float64}, Vector{Int64}}  
L factor:  
3x3 Matrix{Float64}:  
1.0      0.0      0.0  
0.726169  1.0      0.0  
0.782334  0.7677769  1.0  
U factor:  
3x3 Matrix{Float64}:  
0.882431  0.943103  0.605134  
0.0       -0.639719  0.196184  
0.0       0.0       -0.0311543
```

```
[45]: # Матрица перестановок:  
Alu.P
```

```
[45]: 3x3 Matrix{Float64}:  
1.0  0.0  0.0  
0.0  1.0  0.0  
0.0  0.0  1.0
```

```
[46]: # Вектор перестановок:  
Alu.p
```

```
[46]: 3-element Vector{Int64}:  
1  
2  
3
```

```
[47]: # Матрица L:  
Alu.L
```

```
[47]: 3x3 Matrix{Float64}:  
1.0      0.0      0.0  
0.726169  1.0      0.0  
0.782334  0.7677769  1.0
```

```
[48]: # Матрица U:  
Alu.U
```

```
[48]: 3x3 Matrix{Float64}:  
0.882431  0.943103  0.605134  
0.0       -0.639719  0.196184  
0.0       0.0       -0.0311543
```

Факторизация. Специальные матричные структуры

Исходная система уравнений $Ax = b$ может быть решена или с использованием исходной матрицы, или с использованием объекта факторизации:

```
[49]: # Решение СЛАУ через матрицу A:  
A\b
```

```
[49]: 3-element Vector{Float64}:  
1.000000000000069  
0.999999999999984  
0.999999999999929
```

```
[50]: # Решение СЛАУ через объект факторизации:  
Alu\b
```

```
[50]: 3-element Vector{Float64}:  
1.000000000000069  
0.999999999999984  
0.999999999999929
```

```
[51]: # Детерминант матрицы A:  
det(A)
```

```
[51]: 0.017586842847094095
```

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

```
[52]: 0.017586842847094095
```

Рис. 10: Пример решения с использованием исходной матрицы и с использованием объекта факторизации

Факторизация. Специальные матричные структуры

Julia позволяет вычислять QR-факторизацию и определяет составной тип факторизации для его хранения:

```
[53]: # QR-факторизация:  
Aqr = qr(A)  
  
[53]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}  
Q factor: 3x3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}  
R factor:  
3x3 Matrix{Float64}:  
 -1.2907  -0.79913  -1.04641  
  0.0       0.560104  -0.161714  
  0.0       0.0        -0.0243274  
  
[54]: # Матрица Q:  
Aqr.Q  
  
[54]: 3x3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}  
  
[55]: # Матрица R:  
Aqr.R  
  
[55]: 3x3 Matrix{Float64}:  
 -1.2907  -0.79913  -1.04641  
  0.0       0.560104  -0.161714  
  0.0       0.0        -0.0243274  
  
[56]: # Проверка, что матрица Q - ортогональная:  
Aqr.Q'*Aqr.Q  
  
[56]: 3x3 Matrix{Float64}:  
 1.0       0.0        -5.55112e-17  
 1.66533e-16 1.0        0.0  
 0.0       2.22045e-16 1.0
```

Факторизация. Специальные матричные структуры

Примеры собственной декомпозиции матрицы A:

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

```
[57]: 3x3 Matrix{Float64}:  
1.76406 1.5839 1.29549  
1.5839 0.0902688 0.86228  
1.29549 0.86228 1.18577
```

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

```
[58]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}  
values:  
3-element Vector{Float64}:  
-0.8695583384409882  
0.21485179692981338  
3.6956119617767946  
vectors:  
3x3 Matrix{Float64}:  
0.491179 -0.488132 -0.721436  
-0.868753 -0.214306 -0.446476  
0.0633308 0.84605 -0.529329
```

```
[59]: # Собственные значения:  
AsymEig.values
```

```
[59]: 3-element Vector{Float64}:  
-0.8695583384409882  
0.21485179692981338  
3.6956119617767946
```

```
[60]: #Собственные векторы:  
AsymEig.vectors
```

```
[60]: 3x3 Matrix{Float64}:  
0.491179 -0.488132 -0.721436  
-0.868753 -0.214306 -0.446476  
0.0633308 0.84605 -0.529329
```

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

```
[61]: 3x3 Matrix{Float64}:  
1.0 -1.9984e-15 -3.10862e-15  
-9.99281e-16 1.0 -1.55431e-15  
4.44089e-15 3.10862e-15 1.0
```

Факторизация. Специальные матричные структуры

Далее рассмотрим примеры работы с матрицами большой размерности и специальной структуры:

[62]: # Матрица 1000 x 1000:

n = 1000

A = randn(n,n)

[62]: 1000x1000 Matrix{float64}:

```
0.189086 -1.53028  0.486299  - 0.758935 -1.11152 -1.37552  
-0.634277  0.649888 -0.172438  1.684446  0.8801  -1.04439  
0.840243  1.68924  -0.337911  0.323935 -0.338181 -0.888466  
0.2523898 0.182762  1.86293  0.506897 -0.87474  1.61224  
-0.450202  0.44922  -0.202952  0.485268  1.1124  -0.104841  
-1.277788  0.39112  -1.17964  -1.91543  0.117764  0.658176  
0.6317412 0.858475  1.93548   0.731247  0.985846  1.591376  
0.866632  0.0937344 -1.23936  -0.119384  0.274476 -0.605794  
-2.35817  -1.51724  0.324026  0.0553818 -1.11942  -0.51897  
0.1693733 0.173795  0.1214443 -0.897702  0.111827  1.517389  
-0.2685557 -1.9483  0.703383  -0.844949 -0.105927 -0.973783  
-3.14089  -0.4618  1.08879  -1.73238  -0.0458123 0.732366  
1.051388  1.43149  1.16767  -0.430051  -1.23154 -1.20881  
:  
1.85675  -0.092866  1.04126  -0.296344  0.000803  0.234219  
-1.74748  1.2909  0.822812  1.17699  -1.12317  0.792247  
-1.12883  -0.179724  -1.36866  -0.855854  0.410456  1.17483  
-0.8146883 0.366683  0.253385  0.805336  1.68599  -0.0144126  
-0.4813201 1.10081  -1.17539  1.12272  0.76954  -0.0144126  
-0.804277183 0.300819  0.154727  0.548881  0.7953676  1.459  
0.00476204  1.34054  -0.1833044  -1.21564  1.00342  1.00884  
-0.9587609  0.14018  0.734342  -1.21923  0.32895  0.036475  
-0.379791  0.674633  -0.542285  1.10646  0.242543  0.36251  
-0.234844  -0.382717  -0.0809377  0.909537  0.434977  0.258415  
0.26623  -0.773272  0.971496  2.22529  0.714399  -0.825674  
0.1495106  1.97052  1.15124  -0.568779  -1.11365  1.33837
```

[63]: # Симметризация матрицы:

Asym = A + A'

[63]: 1000x1000 Matrix{float64}:

```
0.370171 -2.14455  1.33154  -0.524091 -0.905286 -1.23041  
-2.14455  3.29978  1.43678  -2.00718  0.0318261 0.92613  
1.33154  1.43678  3.29978  0.0318261 0.222572 0.222548  
1.03099  2.318458  1.37213  0.531503  0.88543  0.222548  
-0.757724 0.238888 -0.00214946 0.169433  2.01213  0.372268  
-2.44661  1.72085  0.1855228  -2.37462  0.247145  1.1086  
-0.688267 -0.496106  2.9525  0.70481  0.245884  2.77085  
2.13138  -1.09553  -2.30285  0.978318  -0.347693 -1.17925  
-1.03651  -2.6548  -1.11806  0.585998  -0.469279  0.184027  
3.31587  -0.652932 -0.179783  -0.167379  -0.519292  0.147241  
0.651795  2.31253  2.20225  -0.661855  -2.516  0.863578  
-3.80064  -0.196885  0.760797  1.48799  0.21873  1.2333  
1.1911  1.73385  1.13667  0.779874  -0.269164 -1.2405  
:  
-0.432956 -0.99338  1.22839  1.01865  1.89091 -0.172598  
-2.78779  0.147219  1.22839  0.639331  1.217119  1.217119  
-2.28779  -0.447219  0.587223  -0.575938  1.140896  1.45391  
0.486158  1.4372  -0.221826  2.51225  1.62576  0.579377  
-0.8268229 -0.0172396  -1.173816  2.54136  1.93185  0.669446  
0.364596  1.69168  0.3648008  0.661939  1.9942  -3.66392  
1.19641  -0.911763  0.176437  -0.533748  -2.92989  -0.797762  
0.217283  -1.22359  0.666128  -0.0614411  -0.618577  -1.63402  
0.0385428  -0.150321  -0.375111  -0.975737  1.05767  0.0268845  
0.5264091  2.06718  0.4046672  1.821911  1.79032  -0.310364  
-0.905286  0.0328261  0.633225  1.79032  1.4288  1.03932  
-1.23041  0.92613  1.06277  0.310364  -1.93932  2.67675
```

[64]: # проверка, является ли матрица симметричной:

issymmetric(Asym)

Факторизация. Специальные матричные структуры

Пример добавления шума в симметричную матрицу (матрица уже не будет симметричной):

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

```
[66]: -2.1445519133285655
```

```
[67]: # Проверка, является ли матрица симметричной:  
issymmetric(Asym_noisy)
```

```
[67]: false
```

Рис. 14: Пример добавления шума в симметричную матрицу

Факторизация. Специальные матричные структуры

▼ В Julia можно объявить структуру матрица явно, например, используя Diagonal, Triangular, Symmetric, Hermitian, Tridiagonal и SymTridiagonal:

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

```
[68]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:  
 0.378171 -2.14455 1.33154 ... 0.524091 -0.905286 -1.23041  
 -2.14455 1.29978 1.43678 2.06718 0.0328261 0.92613  
 1.33154 1.43678 -0.675183 -0.404472 0.633225 1.06277  
 1.03899 -0.316458 1.37217 -0.931503 -1.68343 -0.222548  
 -0.797724 0.288688 -0.00214946 0.609433 2.01135 0.372868  
 -2.44661 1.72085 0.105528 ... -2.37402 0.247145 1.3086  
 -0.608267 -0.0960106 2.9525 -0.70481 0.245804 2.73805  
 2.1318 -1.00553 -2.30285 0.978318 -0.347693 -1.17926  
 -1.81651 -2.6546 -1.11806 0.585098 -0.469279 0.184027  
 3.31587 -0.652932 -0.179783 -0.167379 -0.513926 0.147241  
 0.651795 -2.31253 2.20225 ... -0.661855 -2.516 0.863578  
 -3.80064 -0.106805 0.760797 -1.48799 0.210751 1.2835  
 1.1911 1.73589 1.11657 0.779074 -0.269166 -1.24056  
 :  
 -0.432956 -0.99338 1.22839 1.01865 -1.89691 -0.172508  
 -2.70779 -0.13743 1.24084 0.637531 -1.12259 1.66219  
 -2.89039 0.447319 -0.982212 ... -0.537538 0.140869 1.45391  
 -0.406158 1.4372 -0.212816 2.51235 1.62976 0.579377  
 -0.820229 -0.0172196 -1.17387 2.54116 -1.03185 0.969346  
 0.364596 1.69348 0.268008 0.661939 -1.9942 -2.66392  
 1.19641 -0.911753 0.176437 -0.533748 -2.92098 -0.707762  
 0.217823 -1.22359 0.666128 ... 0.0614431 -0.618577 -1.63402  
 0.0385428 -0.150121 -0.37511 -0.975737 1.05767 0.0268845  
 0.524091 2.06718 -0.404472 -1.81911 1.79832 -0.310364  
 -0.905286 0.0328261 0.633225 1.79032 1.4288 -1.93932  
 -1.23041 0.92613 1.06277 -0.310364 -1.93932 2.67675
```

Рис. 15: Пример явного объявления структуры матрицы

Факторизация. Специальные матричные структуры

```
[66]: import Pkg;
Pkg.add("BenchmarkTools")

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

Resolving package versions...
No Changes to `C:\Users\timur\.julia\environments\v1.11\Project.toml`
No Changes to `C:\Users\timur\.julia\environments\v1.11\Manifest.toml`
47.976 ms (21 allocations: 7.99 MiB)

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

398.842 ms (27 allocations: 7.93 MiB)

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

48.687 ms (21 allocations: 7.99 MiB)
```

Рис. 16: Использование пакета BenchmarkTools

Факторизация. Специальные матричные структуры

- Далее рассмотрим примеры работы с разреженными матрицами большой размерности.

Использование типов `Tridiagonal` и `SymTridiagonal` для хранения трёхдиагональных матриц позволяет работать с потенциально очень большими трёхдиагональными матрицами:

```
[88]: # Трёхдиагональная матрица 1000000 x 1000000
n = 1000000;
A = SymTridiagonal(randn(n), randn(n-1))
```

```
[88]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}
```

```
[87]: # Оценка эффективности выполнения операции по нахождению  
# собственных значений:  
@btime eigmax(A)
```

484.820 ms (17 allocations: 183.11 MiB)

```
[87]: 7.228144545523525
```

Общая линейная алгебра

6. Общая линейная алгебра

```
[81]: # Матрица с рациональными элементами:  
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
```

```
[81]: 3x3 Matrix{Rational{BigInt}}:  
 3//5  7//10  7//10  
 7//10  9//10  1  
 2//5   7//10  3//10
```

```
[83]: # Единичный вектор:  
x = fill(1, 3)
```

```
[83]: 3-element Vector{Int64}:  
 1  
 1  
 1
```

```
[84]: # Задаём вектор b:  
b = Arational*x
```

```
[84]: 3-element Vector{Rational{BigInt}}:  
 2  
 13//5  
 7//5
```

```
[85]: # Решение исходного уравнения получаем с помощью функции \  
# (убеждаемся, что x - единичный вектор):  
Arational\b
```

```
[85]: 3-element Vector{Rational{BigInt}}:  
 1  
 1  
 1
```

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

```
[86]: LU{Rational{BigInt}}, Matrix{Rational{BigInt}}, Vector{Int64}
```

```
L factor:
```

```
3x3 Matrix{Rational{BigInt}}:
```

```
 1   0   0  
 4//7  1   0  
 6//7  -5//13  1
```

```
U factor:
```

```
3x3 Matrix{Rational{BigInt}}:
```

```
 7//10  9//10  1  
 0     13//70  -19//70  
 0     0       -17//65
```

Самостоятельная работа

▼ Произведение векторов

1) Задайте вектор v . Умножьте вектор v скалярно сам на себя и сохраните результат в `dot_v`:

```
[93]: using LinearAlgebra  
  
# Задаем вектор v  
v = [1, 2, 3]  
  
# Скалярное произведение  
dot_v = dot(v, v)
```

```
[93]: 14
```

2) Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной `outer_v`:

```
[94]: # Матричное (внешнее) произведение  
outer_v = v * v'
```

```
[94]: 3x3 Matrix{Int64}:  
1 2 3  
2 4 6  
3 6 9
```

Рис. 19: Решение задания “Произведение векторов”

Самостоятельная работа

Системы линейных уравнений

1) Решить СЛАУ с двумя неизвестными:

```
[95]: using LinearAlgebra

# Система а
A1 = [1; 1; 1 -1]
b1 = [2; 3]
x1 = A1 \ b1
println("Система а: x = $x1")

# Система б
A2 = [1; 1; 2 2]
b2 = [2; 4]
try
    x2 = A2 \ b2
    println("Система б: x = $x2")
catch e
    println("Система б: Нет решения (линейно зависимая система)")
end

# Система с
A3 = [1; 1; 2 2]
b3 = [2; 5]
try
    x3 = A3 \ b3
    println("Система с: x = $x3")
catch e
    println("Система с: Нет решения (несовместная система)")
end

# Система д
A4 = [1; 1; 2 2; 3 3]
b4 = [1; 2; 3]
try
    x4 = A4 \ b4
    println("Система д: x = $x4")
catch e
    println("Система д: Нет решения (линейно зависимая система)")
end

# Система е
A5 = [1; 1; 2 1; 1 -1]
b5 = [2; 1; 3]
try
    x5 = A5 \ b5
    println("Система е: x = $x5")
catch e
    println("Система е: Нет решения (несовместная система)")
end

# Система ф
A6 = [1; 1; 2 1; 3 2]
b6 = [2; 1; 3]
try
    x6 = A6 \ b6
    println("Система ф: x = $x6")
catch e
    println("Система ф: Нет решения (сингулярная матрица или неподходящее решение)")
end
```

Система а: x = [2.5, -0.5]
Система б: Нет решения (линейно зависимая система)
Система с: Нет решения (несовместная система)
Система д: x = [0.4999999999999999, 0.3]
Система е: x = [1.58888888888884, -0.999999999999997]

Самостоятельная работа

2) Решить СЛАУ с тремя неизвестными:

```
[96]: using LinearAlgebra

# Система а
A1 = [1 1 1; 1 -1 -2]
b1 = [2; 3]
try
    x1 = A1 \ b1
    println("Система а: x = $x1")
catch e
    println("Система а: Нет решения (недостаточно уравнений для определения решения)")
end

# Система б
A2 = [1 1 1; 2 2 -3; 3 1 1]
b2 = [2; 4; 1]
try
    x2 = A2 \ b2
    println("Система б: x = $x2")
catch e
    println("Система б: Нет решения")
end

# Система с
A3 = [1 1 1; 1 1 2; 2 2 3]
b3 = [1; 0; 1]
try
    x3 = A3 \ b3
    println("Система с: x = $x3")
catch e
    println("Система с: Нет решения (сингулярная матрица или неопределённое решение)")
end

# Система д
A4 = [1 1 1; 1 1 2; 2 2 3]
b4 = [1; 0; 0]
try
    x4 = A4 \ b4
    println("Система д: x = $x4")
catch e
    println("Система д: Нет решения (сингулярная матрица или неопределённое решение)")
end
```

Система а: x = [2.2142857142857144, 0.35714285714285704, -0.5714285714285712]

Система б: x = [-0.5, 2.5, 0.0]

Система с: Нет решения (сингулярная матрица или неопределённое решение)

Система д: Нет решения (сингулярная матрица или неопределённое решение)

Самостоятельная работа

▼ Операции с матрицами

1) Приведите приведённые ниже матрицы к диагональному виду:

```
[100]: # a)

A = [1 -2; -2 1]

eigen_A = eigen(A) # Собственные значения и векторы
diag_matrix = Diagonal(eigen_A.values) # Диагональная матрица
```

```
[100]: 2x2 Diagonal{Float64, Vector{Float64}}:
-1.0
·
· 3.0
```

```
[98]: # b)

B = [1 -2; -2 3]

eigen_B = eigen(B) # Собственные значения и векторы
diag_matrix = Diagonal(eigen_B.values) # Диагональная матрица
```

```
[98]: 2x2 Diagonal{Float64, Vector{Float64}}:
-0.236068
·
· 4.23607
```

```
[99]: # c)

C = [1 -2 0; -2 1 2; 0 2 0]

eigen_C = eigen(C) # Собственные значения и векторы
diag_matrix = Diagonal(eigen_C.values) # Диагональная матрица
```

```
[99]: 3x3 Diagonal{Float64, Vector{Float64}}:
-2.14134
·
· 0.515138
·
· 2.6662
```

Самостоятельная работа

▼ 2) Вычислите:

```
[109]: # Исходная матрица (a)
A = [1 -2;
      -2 1]

# Собственные значения и векторы
eigen_decomp = eigen(A)
P = eigen_decomp.vectors # Матрица собственных векторов
D = Diagonal(eigen_decomp.values) # Диагональная матрица собственных значений

# Возведим диагональную матрицу в 10-ю степень
D_10 = D.^10

# Вычисляем A^10
A_10 = P * D_10 * inv(P)

println("Матрица A^10:")
println(A_10)

Матрица A^10:
[29525.0 -29524.0; -29524.0 29525.0]
```

Рис. 23: Решение задания “Операции с матрицами”

Самостоятельная работа

```
[108]: # Исходная матрица (A)
A = [5 -2;
      -2 5]

# Собственные значения и векторы
eigen_decomp = eigen(A)
eigenvalues = eigen_decomp.values
eigenvectors = eigen_decomp.vectors

# Проверяем, что собственные значения неотрицательные
if all(eigenvalues .>= 0)
    # Диагональная матрица с квадратными корнями собственных значений
    sqrt_D = Diagonal(sqrt.(eigenvalues))

    # Квадратный корень матрицы
    sqrt_A = eigenvectors * sqrt_D * inv(eigenvectors)

    println("Исходная матрица A:")
    println(A)

    println("\nКвадратный корень матрицы sqrt(A):")
    println(sqrt_A)

    # Проверка, что sqrt(A)^2 = A
    println("\nПроверка: sqrt(A)^2:")
    println(sqrt_A * sqrt_A)
else
    println("Матрица A имеет отрицательные собственные значения, квадратный корень не определён.")
end
```

Исходная матрица A:
[5 -2; -2 5]

Квадратный корень матрицы sqrt(A):
[2.188901059316734 -0.45685025174785676; -0.45685025174785676 2.188901059316734]

Проверка: sqrt(A)^2:
[5.000000000000000 -2.0000000000000001; -2.0000000000000001 5.0000000000000002]

Самостоятельная работа

```
[107]: # Исходная матрица (c)
A = [1 -2;
      -2 1]

# Собственные значения и векторы
eigen_decomp = eigen(A)
eigenvalues = eigen_decomp.values
eigenvectors = eigen_decomp.vectors

# Преобразуем собственные значения в комплексные для вычисления кубического корня
complex_eigenvalues = Complex.(eigenvalues)
cube_root_D = Diagonal(complex_eigenvalues .^ (1/3))

# Кубический корень матрицы
cube_root_A = eigenvectors * cube_root_D * inv(eigenvectors)

println("Исходная матрица A:")
println(A)

println("\nКубический корень матрицы 3V(A):")
println(cube_root_A)

# Проверка: (3V(A))^3 = A
println("\nПроверка: (3V(A))^3:")
println(cube_root_A * cube_root_A * cube_root_A)

Исходная матрица A:
[1 -2; -2 1]

Кубический корень матрицы 3V(A):
ComplexF64[0.971124785153704 + 0.4330127018922193im -0.47112478515370404 + 0.4330127018922193im; -0.47112478515370404 + 0.4330127018922193im 0.971124785153704 + 0.4330127018922193im]

Проверка: (3V(A))^3:
ComplexF64[0.9999999999999991 + 0.0im -1.9999999999999991 + 5.551115123125783e-17im; -1.9999999999999991 + 5.551115123125783e-17im 0.9999999999999991 + 0.0im]
```

Рис. 25: Решение задания “Операции с матрицами”

Самостоятельная работа

3) Найдите собственные значения матрицы A . Создайте диагональную матрицу из собственных значений матрицы A . Создайте нижнедиагональную матрицу из матрицы A . Оцените эффективность выполняемых операций:

```
[110]: # Исходная матрица A
A = [
    148  97  74 168 131;
    97 186  89 131  36;
    74  89 152 144  71;
   168 131 144  54 142;
   131  36  71 142  36
]

# 1. Нахождение собственных значений и векторов
@btime eigen_decomp = eigen(A)
eigenvalues = eigen_decomp.values
eigenvectors = eigen_decomp.vectors

println("Собственные значения матрицы A:")
println(eigenvalues)

# 2. Создание диагональной матрицы из собственных значений
# Применение создания переменной и вывод без использования @btime
diag_matrix = Diagonal(eigenvalues)

println("\nДиагональная матрица из собственных значений:")
println(Matrix(diag_matrix)) # Преобразуем в стандартный массив для вывода

# 3. Создание нижнедиагональной матрицы из A
lower_triangular = LowerTriangular(A)

println("\nНижнедиагональная матрица из A:")
println(Matrix(lower_triangular))

# 4. Оценка эффективности
println("\nЭффективность выполнения операций:")
@btime eigen(A)
@btime Diagonal(eigenvalues)
@btime lowerTriangular(A)

5.433 μs (11 allocations: 3.00 KiB)
Собственные значения матрицы A:
[-1.0, 3.0]

Диагональная матрица из собственных значений:
[-1.0 0.0; 0.0 3.0]

Нижнедиагональная матрица из A:
[148 0 0 0; 97 186 0 0; 74 89 152 0; 168 131 144 54; 131 36 71 142 36]

Эффективность выполнения операций:
5.483 μs (11 allocations: 3.00 KiB)
193.239 ns (1 allocation: 16 bytes)
177.997 ns (1 allocation: 16 bytes)

[110]: 5×5 LowerTriangular{Int64, Matrix{Int64}}:
 148   .   .   .   .
   97  186   .   .   .
   74   89  152   .   .
  168  131  144   54   .
  131   36   71  142   .
```

Самостоятельная работа

Линейные модели экономики

- ✓ 1) Матрица A называется продуктивной, если решение x системы при любой неотрицательной правой части u имеет только неотрицательные элементы x_i . Используя этоопределение, проверьте, являются ли матрицы продуктивными;
- 2) Критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все элементы матрицы $(E - A)^{-1}$ являются неотрицательными числами. Используя этот критерий, проверьте, являются ли матрицы продуктивными;
- 3) Спектральный критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все её собственные значения по модулю меньше 1. Используя этот критерий, проверьте, являются ли матрицы продуктивными.

```
[112]: # Матрица
A1 = [1 2; 3 4]
A2 = -(1/2)*A1
A3 = -(1/10)*A1
A4 = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]

# Функция проверки через  $(E - A)^{-1}$ 
function check_productivity_via_inverse(A)
    E = I(size(A, 1))
    B = E - A
    try
        B_inv = inv(B)
        all(B_inv .>= 0)
    catch
        false
    end
end

# Функция проверки спектрального критерия
function check_productivity_via_spectrum(A)
    eigenvalues = eigvals(A)
    all(abs.(eigenvalues) .< 1)
end

# Проверка продуктивности для всех матриц
matrices = [A1, A2, A3, A4]
for (i, A) in enumerate(matrices)
    println("Matrix $i:")
    println()
    println("Via $(E - A)^{-1}: ", check_productivity_via_inverse(A))
    println("Via spectrum: ", check_productivity_via_spectrum(A))
    println("...")#30
end

Matrix A1:
[1.0 2.0; 3.0 4.0]
Via (E - A)^{-1}: false
Via spectrum: false
-----
Matrix A2:
[0.5 1.0; 1.5 2.0]
Via (E - A)^{-1}: false
Via spectrum: false
-----
Matrix A3:
[0.1 0.2; 0.3000000000000004 0.4]
Via (E - A)^{-1}: true
Via spectrum: true
-----
Matrix A4:
[0.1 0.2 0.3; 0.0 0.1 0.2; 0.0 0.1 0.3]
Via (E - A)^{-1}: true
Via spectrum: true
```

Вывод

- В ходе выполнения лабораторной работы были изучены возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Список литературы. Библиография

Список литературы. Библиография

[1] Julia Documentation: <https://docs.julialang.org/en/v1/>

Вывод

- В ходе выполнения лабораторной работы познакомились с инструментом для измерения пропускной способности сети в режиме реального времени — iPerf3, а также получили навыки проведения воспроизводимого эксперимента по измерению пропускной способности моделируемой сети в среде Mininet

Список литературы. Библиография

Список литературы. Библиография

[1] Mininet: <https://mininet.org/>