

Отчёт по лабораторной работе №3

Компьютерный практикум по

статистическому анализу данных

Управляющие структуры

Выполнила: Коннова Татьяна Алексеевна,
НПИбд-01-22, 1132221814

Содержание

1 Цель работы	4
2 Выполнение лабораторной работы	5
2.1 Циклы while и for	5
2.2 Условные выражения	8
2.3 Функции	10
2.4 Сторонние библиотеки (пакеты) в Julia	12
2.5 Самостоятельная работа	14
3 Вывод	23
4 Список литературы. Библиография	24

Список иллюстраций

2.1 Примеры использования цикла while	6
2.2 Примеры использования цикла for	7
2.3 Пример использования цикла for для создания двумерного массива	8
2.4 Примеры использования условного выражения	9
2.5 Примеры способов написания функции	10
2.6 Сравнение результатов вывода	11
2.7 Примеры использования функций map() и broadcast()	12
2.8 Пример использования сторонних библиотек	13
2.9 Выполнение подпунктов задания №1	14
2.10 Выполнение подпунктов задания №1	15
2.11 Выполнение подпунктов задания №1	15
2.12 Выполнение подпунктов задания №1	15
2.13 Выполнение задания №2	16
2.14 Выполнение задания №3	16
2.15 Выполнение задания №4	16
2.16 Выполнение задания №5	17
2.17 Выполнение задания №6	17
2.18 Выполнение задания №7	18
2.19 Выполнение задания №7	18
2.20 Выполнение подпунктов задания №8	19
2.21 Выполнение подпунктов задания №8	19
2.22 Выполнение подпунктов задания №8	20
2.23 Выполнение задания №9	21
2.24 Выполнение подпунктов задания №10	21
2.25 Выполнение подпунктов задания №10	21
2.26 Выполнение задания №11	22

1 Цель работы

Основная цель работы — освоить применение циклов функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.

2 Выполнение лабораторной работы

2.1 Циклы while и for

Для различных операций, связанных с перебором индексируемых элементов структур данных, традиционно используются циклы while и for.

Синтаксис while

```
while <условие>
    <тело цикла>
end
```

Примеры использования цикла while (рис. 2.1):

Циклы while и for

```
[11]: # Формирование элементов массива:

# пока n<10 прибавить к n единицу и распечатать значение:
n = 0
while n < 10
    n += 1
    println(n)
end

1
2
3
4
5
6
7
8
9
10

[8]: # Демонстрация использования while при работе со строковыми элементами массива:
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]

i = 1
while i <= length(myfriends)
    friend = myfriends[i]
    println("Hi $friend, it's great to see you!")
    i += 1
end

Hi Ted, it's great to see you!
Hi Robyn, it's great to see you!
Hi Barney, it's great to see you!
Hi Lily, it's great to see you!
Hi Marshall, it's great to see you!
```

Рис. 2.1: Примеры использования цикла while

Такие же результаты можно получить при использовании цикла for.

Синтаксис for

```
for <переменная> in <диапазон>
    <тело цикла>
end
```

Примеры использования цикла for (рис. 2.2):

```
[9]: # Рассмотренные выше примеры, но с использованием цикла for:  
for n in 1:2:10  
    println(n)  
end  
  
1  
3  
5  
7  
9  
  
[10]: # Рассмотренные выше примеры, но с использованием цикла for:  
myfriends = ["Ted", "Robyn", "Barney", "Lily", "Marshall"]  
  
for friend in myfriends  
    println("Hi $friend, it's great to see you!")  
end  
  
Hi Ted, it's great to see you!  
Hi Robyn, it's great to see you!  
Hi Barney, it's great to see you!  
Hi Lily, it's great to see you!  
Hi Marshall, it's great to see you!
```

Рис. 2.2: Примеры использования цикла for

Пример использования цикла for для создания двумерного массива, в котором значение каждой записи является суммой индексов строки и столбца (рис. 2.3):

```
[14]: # Пример использования цикла for для создания двумерного массива,
# в котором значение каждой записи является суммой индексов строки и столбца:

# инициализация массива m x n из нулей:
m, n = 5, 5
A = fill(0, (m, n))

# формирование массива, в котором значение каждой записи
# является суммой индексов строки и столбца:
for i in 1:m
    for j in 1:n
        A[i, j] = i + j
    end
end
A
```

```
[14]: 5×5 Matrix{Int64}:
 2  3  4  5  6
 3  4  5  6  7
 4  5  6  7  8
 5  6  7  8  9
 6  7  8  9  10
```

```
[15]: # Другая реализация примера выше:

# инициализация массива m x n из нулей:
B = fill(0, (m, n))

for i in 1:m, j in 1:n
    B[i, j] = i + j
end
B
```

```
[15]: 5×5 Matrix{Int64}:
 2  3  4  5  6
 3  4  5  6  7
 4  5  6  7  8
 5  6  7  8  9
 6  7  8  9  10
```

```
[16]: # Ещё одна реализация этого же примера:

C = [i + j for i in 1:m, j in 1:n]
C
```

Рис. 2.3: Пример использования цикла for для создания двумерного массива

2.2 Условные выражения

Довольно часто при решении задач требуется проверить выполнение тех или иных условий. Для этого используют условные выражения.

Синтаксис условных выражений с ключевым словом:

```
if <условие 1>
    <действие 1>
elseif <условие 2>
    <действие 2>
else
    <действие 3>
end
```

Примеры использования условного выражения (рис. 2.4):

Условные выражения

```
[18]: # Пусть для заданного числа N требуется вывести слово «Fizz», если N делится на 3,
# «Buzz», если N делится на 5, и «FizzBuzz», если N делится на 3 и 5:

# используем `&&` для реализации операции "AND"
# операция % вычисляет остаток от деления
N = 100

if (N % 3 == 0) && (N % 5 == 0)
    println("FizzBuzz")
elseif N % 3 == 0
    println("Fizz")
elseif N % 5 == 0
    println("Buzz")
else
    println(N)
end

Buzz
```

```
[20]: # Пример использования тернарного оператора:

x = 5
y = 10

(x > y) ? x : y
```

```
[20]: 10
```

Рис. 2.4: Примеры использования условного выражения

2.3 Функции

Julia дает нам несколько разных способов написать функцию.

Примеры способов написания функции (рис. 2.5):

```
Функции

[23]: # Первый способ требует ключевых слов function и end:

function sayhi(name)
    println("Hi $name, it's great to see you!")
end
# функция возведения в квадрат:
function f(x)
    x^2
end

# Вызов функции осуществляется по её имени с указанием аргументов, например:
sayhi("C-3PO")
f(42)

Hi C-3PO, it's great to see you!
[23]: 1764

[27]: # В качестве альтернативы, можно объявить любую из выше определённых функций в одной строке:

sayhi2(name) = println("Hi $name, it's great to see you!")
f2(x) = x^2

sayhi("C-3PO")
f(42)

Hi C-3PO, it's great to see you!
[27]: 1764

[28]: # Наконец, можно объявить выше определённые функции как «анонимные»:

sayhi3 = name -> println("Hi $name, it's great to see you!")
f3 = x -> x^2

sayhi("C-3PO")
f(42)

Hi C-3PO, it's great to see you!
[28]: 1764
```

Рис. 2.5: Примеры способов написания функции

По соглашению в Julia функции, сопровождаемые восклицательным знаком, изменяют свое содержимое, а функции без восклицательного знака не делают этого (рис. 2.6):

```
[34]: # Сравнение результата применения sort и sort!:

# задаём массив v:
v = [3, 5, 2]
sort(v)
v

[34]: 3-element Vector{Int64}:
 3
 5
 2

[125]: sort!(v)
v

[125]: 3-element Vector{Int64}:
 2
 3
 5
```

Рис. 2.6: Сравнение результатов вывода

В Julia функция `map` является функцией высшего порядка, которая принимает функцию в качестве одного из своих входных аргументов и применяет эту функцию к каждому элементу структуры данных, которая ей передаётся также в качестве аргумента.

Функция `broadcast` — ещё одна функция высшего порядка в Julia, представляющая собой обобщение функции `map`. Функция `broadcast()` будет пытаться привести все объекты к общему измерению, `map()` будет напрямую применять данную функцию поэлементно.

Примеры использования функций `map()` и `broadcast()` (рис. 2.7):

```
[38]: # В Julia функция map является функцией высшего порядка, которая принимает функцию
# в качестве одного из своих входных аргументов и применяет эту функцию к каждому
# элементу структуры данных, которая ей передаётся также в качестве аргумента

f(x) = x^3
map(f, [1, 2, 3])

[38]: 3-element Vector{Int64}:
 1
 8
27

[39]: # Функция broadcast – ещё одна функция высшего порядка в Julia, представляющая собой обобщение функции map.
# Функция broadcast() будет пытаться привести все объекты к общему измерению, map() будет напрямую применять
# данную функцию поэлементно

f(x) = x^3
broadcast(f, [1, 2, 3])

[39]: 3-element Vector{Int64}:
 1
 8
27
```

Рис. 2.7: Примеры использования функций map() и broadcast()

2.4 Сторонние библиотеки (пакеты) в Julia

Julia имеет более 2000 зарегистрированных пакетов, что делает их огромной частью экосистемы Julia. Есть вызовы функций первого класса для других языков, обеспечивающие интерфейсы сторонних функций. Можно вызывать функции из Python или R, например, с помощью PyCall или Rcall.

С перечнем доступных в Julia пакетов можно ознакомиться на страницах следующих ресурсов: - <https://julialang.org/packages/> - <https://juliahub.com/ui/Home> - <https://juliaobserver.com/> - <https://github.com/svaksha/Julia.jl>

При первом использовании пакета в вашей текущей установке Julia вам необходимо использовать менеджер пакетов, чтобы явно его добавить:

```
import Pkg
Pkg.add("Example")
```

При каждом новом использовании Julia (например, в начале нового сеанса в REPL или открытии блокнота в первый раз) нужно загрузить пакет, используя ключевое слово using:

Например, добавим и загрузим пакет Colors:

```
Pkg.add("Colors")  
using Colors
```

Затем создадим палитру из 100 разных цветов:

```
palette = distinguishable_colors(100)
```

А затем определим матрицу 3×3 с элементами в форме случайного цвета из палитры, используя функцию rand:

```
rand(palette, 3, 3)
```

Пример использования сторонних библиотек (рис. 2.8):

```
using Colors  
  
# Затем создадим палитру из 100 разных цветов:  
palette = distinguishable_colors(100)  
  
# а затем определим матрицу  $3 \times 3$  с элементами в форме случайного цвета из палитры, используя функцию rand:  
rand(palette, 3, 3)
```

```
Resolving package versions...  
No Changes to `C:\Users\konno\.julia\environments\v1.11\Project.toml`  
No Changes to `C:\Users\konno\.julia\environments\v1.11\Manifest.toml`  
Precompiling project...  
10765.1 ms ✓ Reexport  
9147.6 ms ✓ Statistics  
3390.7 ms ✓ FixedPointNumbers  
2117.4 ms ✓ ColorTypes  
1416.1 ms ✓ ColorTypes → StyledStringsExt  
6388.3 ms ✓ Colors  
6 dependencies successfully precompiled in 38 seconds. 40 already precompiled.
```



Рис. 2.8: Пример использования сторонних библиотек

2.5 Самостоятельная работа

Выполнение задания №1 (рис. 2.9 - рис. 2.12):

№1. Используя циклы while и for:

- 1.1) выведите на экран целые числа от 1 до 100 и напечатайте их квадраты:

```
[47]: # while

n = 1
while n <= 100
    println("$n^2 = $(n^2)")
    n += 1
end

1^2 = 1
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
6^2 = 36
7^2 = 49
8^2 = 64
9^2 = 81
10^2 = 100
11^2 = 121
12^2 = 144
13^2 = 169
```

Рис. 2.9: Выполнение подпунктов задания №1

```
[48]: # for
for n in 1:100
    println("$n^2 = $(n^2)")
end

1^2 = 1
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
6^2 = 36
7^2 = 49
8^2 = 64
9^2 = 81
10^2 = 100
11^2 = 121
12^2 = 144
13^2 = 169
14^2 = 196
15^2 = 225
16^2 = 256
17^2 = 289
18^2 = 324
19^2 = 361
20^2 = 400
21^2 = 441
22^2 = 484
23^2 = 529
```

Рис. 2.10: Выполнение подпунктов задания №1

1.2) Создайте словарь squares, который будет содержать целые числа в качестве ключей и квадраты в качестве их пар-значений:

```
[51]: squares = Dict()
for n in 1:100
    squares[n] = n^2
end
println(squares)

Dict(Any, Any){5 => 25, 56 => 3136, 35 => 1225, 55 => 3025, 60 => 3600, 30 => 900, 32 => 1024, 6 => 36, 67 => 4489, 45 => 2025, 73 => 5329, 64 => 4096, 90 => 8100, 4 => 16, 13 => 169, 54 => 2916, 63 => 3969, 86 => 7396, 91 => 8281, 62 => 3844, 58 => 3364, 52 => 2704, 12 => 144, 28 => 784, 75 => 5625, 23 => 529, 92 => 8464, 41 => 1681, 43 => 1849, 11 => 121, 36 => 1296, 68 => 4624, 69 => 4761, 98 => 9604, 82 => 6724, 85 => 7225, 39 => 1521, 64 => 7056, 80 => 5929, 7' => 49, 25 => 625, 95 => 9025, 50 => 2500, 76 => 5776, 34 => 1156, 50 => 2500, 59 => 3481, 93 => 8649, 2 => 4, 10 => 100, 18 => 324, 26 => 676, 27 => 729, 42 => 1764, 67 => 7689, 100 => 10000, 79 => 6241, 16 => 256, 35 => 400, 81 => 6561, 19 => 361, 48 => 2401, 44 => 1936, 9 => 81, 31 => 961, 74 => 5476, 61 => 3721, 29 => 841, 94 => 8836, 46 => 2116, 57 => 3249, 78 => 4 900, 21 => 441, 38 => 1444, 88 => 7744, 78 => 6084, 72 => 5184, 24 => 576, 8 => 64, 17 => 289, 37 => 1369, 1 => 1, 53 => 2089, 22 => 484, 47 => 2289, 83 => 6089, 99 => 9801, 89 => 7921, 14 => 196, 3 => 9, 80 => 64 00, 96 => 9216, 51 => 2601, 33 => 1089, 48 => 1600, 40 => 2304, 15 => 225, 65 => 4225, 97 => 9409}
```

Рис. 2.11: Выполнение подпунктов задания №1

1.3) Создайте массив squares_arr, содержащий квадраты всех чисел от 1 до 100:

```
[53]: squares_arr = [n^2 for n in 1:100]
println(squares_arr)

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 184 9, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
```

Рис. 2.12: Выполнение подпунктов задания №1

Выполнение задания №2 (рис. 2.13):

№2. Напишите условный оператор, который печатает число, если число чётное, и строку «нечётное», если число нечётное. Перепишите код, используя тернарный оператор:

```
[55]: # Условный оператор
n = 42
if n % 2 == 0
    println(n)
else
    println("нечётное")
end
42

[56]: # Тернарный оператор
println(n % 2 == 0 ? n : "нечётное")
42
```

Рис. 2.13: Выполнение задания №2

Выполнение задания №3 (рис. 2.13):

- ▼ №3. Напишите функцию add_one, которая добавляет 1 к своему входу:

```
[57]: function add_one(x)
        return x + 1
    end
    println(add_one(5))
6
```

Рис. 2.14: Выполнение задания №3

Выполнение задания №4 (рис. 2.15):

▀ №4. Используйте map() или broadcast() для задания матрицы A, каждый элемент которой увеличивается на единицу по сравнению с предыдущим:

```
[126]: # map
A = reshape(1:9, 3, 3) # Пример матрицы
B = map(x -> x + 1, A)
println(B)
[2 5 8; 3 6 9; 4 7 10]

[59]: # broadcast
B = broadcast(x -> x + 1, A)
println(B)
[2 5 8; 3 6 9; 4 7 10]
```

Рис. 2.15: Выполнение задания №4

Выполнение задания №5 (рис. 2.16):

▼ №5. Задайте матрицу A следующего вида. Найдите A^3 . Замените третий столбец матрицы A на сумму второго и третьего столбцов: 1

```
[70]: # Определение матрицы
A = [1 1 3; 5 2 6; -2 -1 -3]

# Вычисление A в 3 степени
println(map(x -> x^3, A))

[1 1 27; 125 8 216; -8 -1 -27]

[64]: # Замена третьего столбца на сумму второго и третьего столбцов

A[:, 3] = A[:, 2] + A[:, 3]
println(A)

[1 1 4; 5 2 8; -2 -1 -4]
```

Рис. 2.16: Выполнение задания №5

Выполнение задания №6 (рис. 2.17):

№6. Создайте матрицу B с элементами $B_{i1} = 10$, $B_{i2} = -10$, $B_{i3} = 10$, $i = 1, 2, \dots, 15$. Вычислите матрицу $C = B^T B$:

```
[74]: # Создание матрицы B
B = repeat([10 -10 10], 15, 1)

[74]: 15x3 Matrix{Int64}:
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10
10 -10 10

[75]: # Вычисление C как B' * B
C = B' * B
println(C)

[1500 -1500 1500; -1500 1500 -1500; 1500 -1500 1500]
```

Рис. 2.17: Выполнение задания №6

Выполнение задания №7 (рис. 2.18 - рис. 2.19):

№7. Создайте матрицу Z размерности 6×6 , все элементы которой равны нулю, и матрицу E , все элементы которой равны 1. Используя цикл while или for и закономерности расположения элементов, создайте следующие матрицы размерности 6×6 :

```
[98]: # Функция для красивого вывода матриц
function print_matrix(mat)
    for row in eachrow(mat)
        println(row)
    end
    println()
end

# Создаем матрицу Z размерности 6х6, все элементы которой равны 0
Z = zeros(Int, 6, 6)

println("Матрица Z:")
print_matrix(Z)

# Матрица Z1
Z1 = copy(Z)
for i in 1:6
    for j in 1:6
        if abs(i - j) == 1
            Z1[i, j] = 1
        end
    end
end

# Матрица Z4
Z4 = copy(Z)
for i in 1:6
    for j in 1:6
        if (i + j) % 2 == 0
            Z4[i, j] = 1
        end
    end
end

# Выводим все матрицы
println("Матрица Z1:")
print_matrix(Z1)
println("Матрица Z4:")
print_matrix(Z4)
```

Рис. 2.18: Выполнение задания №7

```
Матрица Z:
[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]

Матрица Z1:
[0, 1, 0, 0, 0, 0]
[1, 0, 1, 0, 0, 0]
[0, 1, 0, 1, 0, 0]
[0, 0, 1, 0, 1, 0]
[0, 0, 0, 1, 0, 1]
[0, 0, 0, 0, 1, 0]

Матрица Z4:
[1, 0, 1, 0, 1, 0]
[0, 1, 0, 1, 0, 1]
[1, 0, 1, 0, 1, 0]
[0, 1, 0, 1, 0, 1]
[1, 0, 1, 0, 1, 0]
[0, 1, 0, 1, 0, 1]
```

Рис. 2.19: Выполнение задания №7

Выполнение задания №8 (рис. 2.20 - рис. 2.22):

▼ №8. В языке R есть функция outer(). Фактически, это матричное умножение с возможностью изменить применяемую операцию (например, заменить произведение на сложение или возведение в степень): ¶

8.1) Напишите свою функцию, аналогичную функции outer() языка R. Функция должна иметь следующий интерфейс: outer(x,y,operation):

```
[99]: function outer(x, y, operation)
        return [operation(xi, yj) for xi in x, yj in y]
    end

[99]: outer (generic function with 1 method)
```

Рис. 2.20: Выполнение подпунктов задания №8

8.2) Используя написанную вами функцию outer(), создайте матрицы следующей структуры:

```
[124]: # Матрица A1: сложение элементов
A1 = outer(0:4, 0:4, +)

# Матрица A2: возведение в степень (если 0^0, принимаем как 0)
function safe_pow(x, y)
    x == 0 && y == 0 ? 0 : x^y
end

# Матрица A2: с пропуском первого элемента в каждой строке
A2 = [j == 1 ? i : safe_pow(i, j) for i in 0:4, j in 1:5]

# Матрица A3: циклический сдвиг по модулю 5
A3 = outer(0:4, 0:4, (x, y) -> mod(x + y, 5))

# Матрица A4: циклический сдвиг по модулю 10
A4 = outer(0:9, 0:9, (x, y) -> mod(x + y, 10))

# Матрица A5: разница по модулю 9
A5 = outer(0:8, 0:8, (x, y) -> mod(x - y, 9))

# Функция для красивого вывода матриц
function print_matrix(name, mat)
    println("\nМатрица $name:")
    for row in eachrow(mat)
        println(row)
    end
end

# Печатаем все матрицы
print_matrix("A1", A1)
print_matrix("A2", A2)
print_matrix("A3", A3)
print_matrix("A4", A4)
print_matrix("A5", A5)
```

Рис. 2.21: Выполнение подпунктов задания №8

Матрица A1:

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4, 5]
[2, 3, 4, 5, 6]
[3, 4, 5, 6, 7]
[4, 5, 6, 7, 8]
```

Матрица A2:

```
[0, 0, 0, 0, 0]
[1, 1, 1, 1, 1]
[2, 4, 8, 16, 32]
[3, 9, 27, 81, 243]
[4, 16, 64, 256, 1024]
```

Матрица A3:

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4, 0]
[2, 3, 4, 0, 1]
[3, 4, 0, 1, 2]
[4, 0, 1, 2, 3]
```

Матрица A4:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
[2, 3, 4, 5, 6, 7, 8, 9, 0, 1]
[3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
[4, 5, 6, 7, 8, 9, 0, 1, 2, 3]
[5, 6, 7, 8, 9, 0, 1, 2, 3, 4]
[6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
[7, 8, 9, 0, 1, 2, 3, 4, 5, 6]
[8, 9, 0, 1, 2, 3, 4, 5, 6, 7]
[9, 0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Матрица A5:

```
[0, 8, 7, 6, 5, 4, 3, 2, 1]
[1, 0, 8, 7, 6, 5, 4, 3, 2]
[2, 1, 0, 8, 7, 6, 5, 4, 3]
[3, 2, 1, 0, 8, 7, 6, 5, 4]
[4, 3, 2, 1, 0, 8, 7, 6, 5]
[5, 4, 3, 2, 1, 0, 8, 7, 6]
[6, 5, 4, 3, 2, 1, 0, 8, 7]
[7, 6, 5, 4, 3, 2, 1, 0, 8]
[8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Рис. 2.22: Выполнение подпунктов задания №8

Выполнение задания №9 (рис. 2.23):

№9. Решите следующую систему линейных уравнений с 5 неизвестными:

```
[85]: A = [1 2 3 4 5;
          2 1 2 3 4;
          3 2 1 2 3;
          4 3 2 1 2;
          5 4 3 2 1];
b = [7, -1, -3, 5, -6];

x = A \ b # Решение системы
println(x)

[-3.916666666666667, 3.0000000000000013, 5.0, -9.500000000000002, 5.583333333333335]
```

Рис. 2.23: Выполнение задания №9

Выполнение задания №10 (рис. 2.24 - рис. 2.25):

№10. Создайте матрицу M размерности 6×10 , элементами которой являются целые числа, выбранные случайным образом с повторениями из совокупности $1, 2, \dots, 10$:

```
[103]: function print_matrix(name, mat)
    println("\nМатрица $name:")
    for row in eachrow(mat)
        println(row)
    end
end

M = rand(1:10, 6, 10)
print_matrix(M)

[4, 10, 8, 3, 4, 7, 7, 5, 1, 4]
[3, 10, 10, 4, 6, 2, 10, 6, 8, 1]
[5, 5, 4, 5, 4, 10, 4, 2, 6, 7]
[3, 2, 10, 7, 5, 3, 7, 4, 4, 1]
[7, 7, 3, 4, 4, 4, 8, 1, 7, 6]
[9, 10, 6, 9, 10, 3, 9, 1, 9, 10]
```

Рис. 2.24: Выполнение подпунктов задания №10

10.1) Найдите число элементов в каждой строке матрицы M , которые больше числа N (например, $N = 4$):

```
[104]: N = 4
greater_than_N = sum(M .> N, dims=2)
println(greater_than_N)

[5; 6; 6; 5; 5; 8;;]
```

10.2) Определите, в каких строках матрицы M число M (например, $M = 7$) встречается ровно 2 раза:

```
[105]: M_value = 7
rows_with_M_twice = findall(x -> count==(M_value), x) == 2, eachrow(M)
println(rows_with_M_twice)

[1, 4]
```

10.3) Определите все пары столбцов матрицы M , сумма элементов которых больше K (например, $K = 75$):

```
[106]: K = 75
col_pairs = []
for i in 1:size(M, 2)-1
    for j in i+1:size(M, 2)
        if sum(M[:,i] .* M[:,j]) > K
            push!(col_pairs, (i, j))
        end
    end
end
println(col_pairs)

Any[(1, 7), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 9), (3, 7), (3, 9), (4, 7), (5, 7), (6, 7), (7, 9)]
```

Рис. 2.25: Выполнение подпунктов задания №10

Выполнение задания №11 (рис. 2.26):

№11. Вычислите:

```
[92]: sum_1 = sum(i^4 * (3 + j) for i in 1:20 for j in 1:5)
println(sum_1)
```

21679980

```
[93]: sum_2 = sum(i^4 * (3 + i * j) for i in 1:20 for j in 1:5)
println(sum_2)
```

195839490

Рис. 2.26: Выполнение задания №11

3 Вывод

В ходе выполнения лабораторной работы было освоено применение циклов функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.

4 Список литературы. Библиография

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