

Лабораторная работа № 4

Линейная алгебра

Герра Гарсия Паола Валентина

Российский университет дружбы народов, Москва, Россия

Информация

- Герра Гарсия Паола Валентина
- студентка
- Российский университет дружбы народов
- 1032225472@pfur.ru

Цель работы

Основной целью работы является изучение возможностей специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Задание

1. Используя JupyterLab, повторите примеры.
2. Выполните задания для самостоятельной работы.

Выполнение лабораторной работы

The screenshot shows a Jupyter Notebook interface with the title "jupyter Lab4_статистическая_аналитика Last Checkpoint 1 hour 10 mins (modified)". The notebook contains the following code:

```
In [1]: # генерация случайных чисел от 1 до 20
a = rand(1,20,20)

In [2]: del Matrix[a]
a
5 18 17
5 18 17
5 18 17
5 18 17

In [3]: # вспомогательные функции
def(a):
    print("Функция имеет не холдинг")
    print("Функция имеет не строку")
    print("Функция имеет строку")
    print("Функция имеет холдинг")

In [4]: del Matrix[a]
a
30
30
30
30

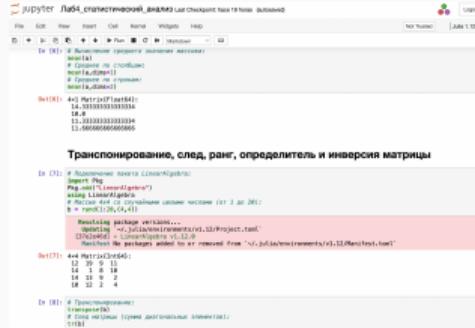
In [5]: # вспомогательное выражение
proto()
# вспомогательное выражение не с холдинг
proto,a,b,c,d
# вспомогательное выражение по строкам
proto,a,b,c,d

In [6]: proto
30
30
30
30

In [7]: # вспомогательные строки Statistica
import stat
stat
```

Рис. 1: Поэлементные операции над многомерными массивами

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
In [1]: # Делаем матрицы
import numpy as np
using LinearAlgebra
# вводим количество строк и столбцов
m = 3
n = 4
# создаем матрицы
matrix1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
matrix2 = np.array([[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]])

In [2]: # Сумма матриц
matrix1 + matrix2
Out[2]:
array([[14, 16, 18, 19],
       [16, 18, 20, 21],
       [20, 22, 23, 24]])

In [3]: # Транспонирование матрицы
matrix1.T
Out[3]:
array([[1, 5, 9],
       [2, 6, 10],
       [3, 7, 11],
       [4, 8, 12]])

In [4]: # Ранг матрицы
rank(matrix1)
Out[4]: 3

In [5]: # Определитель матрицы
det(matrix1)
Out[5]: -44.0

In [6]: # Инверсия матрицы
inv(matrix1)
Out[6]:
array([[ 0.22222222,  0.44444444,  0.66666667,  0.88888889],
       [-0.11111111,  0.11111111,  0.22222222,  0.33333333],
       [-0.33333333,  0.33333333,  0.55555556,  0.77777778],
       [ 0.11111111, -0.11111111, -0.22222222, -0.33333333]])
```

The code defines two 3x4 matrices, adds them, transposes matrix1, calculates its rank, determinant, and inverse.

Рис. 2: Поэлементные операции над многомерными массивами

Выполнение лабораторной работы

The screenshot shows a Jupyter Notebook interface with several code cells and their outputs:

- In [9]:** # Создание вектора X:
X = np.array([2, -1, 3])
Вычисление скалярной нормы:
norm(X)
Вычисление p-нормы:
p = 1
norm(X,p)
- Out[9]:** 11.0
- In [10]:** # Расстояние между двумя векторами X и Y:
Y = np.array([-1, 4, -3])
norm(X-Y)
Проверка по базису определение:
sqrt((X-Y).T @ X-Y)
- Out[10]:** 5.4669329695138
- In [11]:** # Ирон между двумя векторами:
dot((X.T @ Y) / (norm(X)*norm(Y)))
Вычисление норм для диагональной матрицы:
Создание матрицы:
D = np.diag([4, 2, 1, -1, 2, 3, -2, 1, 0])
- Out[11]:** 3x3 Matrix(D164):
5 -4 2
-1 3 0
-2 1 0
- In [12]:** # Вычисление Евклидовой нормы:
dot(D @ D)
Вычисление p-нормы:
p3
dot(D @ D,p)
Ротация на 100 градусов:
rot100(D)
Переопределение строк:

Рис. 3: Транспонирование,след,ранг,определитель инверсия матрицы

Выполнение лабораторной работы

```
Матричное умножение, единичная матрица, скалярное произведение

In [14]: # Матрица 2x2 со случайными целыми значениями от 1 до 10:
A = rand(1,10,[2,2])
# Матрица 3x3 со случайными целыми значениями от 1 до 10:
B = rand(1,10,[3,4])
# Произведение матриц A и B:
A*B

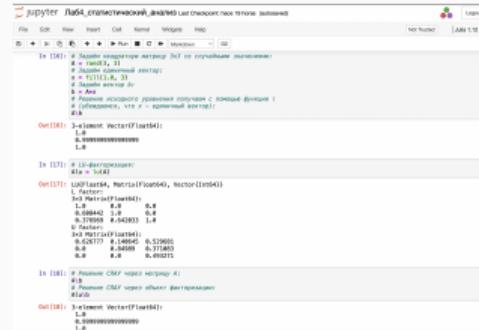
Out[14]: 3x6 Matrix[Int64]:
165 332 154 111
187 76 64 88

In [15]: # Единичная матрица 3x3:
Matrix{Int64}(I, 3, 3)
# Скалярное произведение векторов X и Y:
X = [1, 4, -5]
Y = [-2, 3, -1]
dot(X,Y)
# Еще скалярное произведение:
X*Y

Out[15]: -17
```

Рис. 4: Матричное умножение, единичная матрица, скалярное произведение

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook cell with the following code:

```
In [10]: # Positive CMF через метод QR
A = rand(3, 2)
B = A.T @ A
A = A / norm(A)
B = B / norm(B)
# Positive CMF через факторизацию
A1, B1 = qr(A)

Out[10]: (array([[-0.57735027, -0.81649658],
       [-0.81649658,  0.57735027],
       [-0.4472136,  0.4472136]]), array([[ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136]]))

In [11]: # Positive CMF через метод LU
A = rand(3, 2)
B = A.T @ A
A = A / norm(A)
B = B / norm(B)
# Positive CMF через факторизацию
L, U = lu(A)

Out[11]: (LUFactorMat, Matrix(FromMat), Vector(FromMat))
L = Matrix(FromMat)
U = Matrix(FromMat)
U.Factor()
L = L.LUmatrix()
U = U.LUmatrix()
U.Factor()

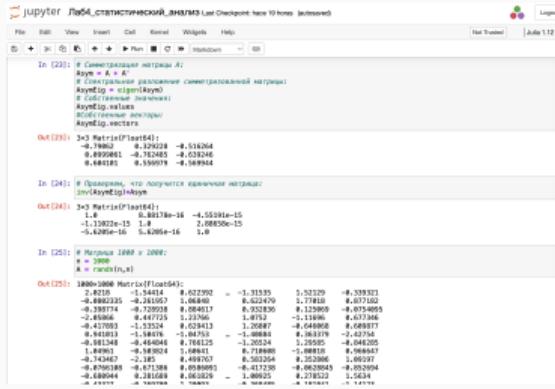
Out[11]: (array([[ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136]]), array([[ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136]]))

In [12]: # Positive CMF через метод Cholesky
A = rand(3, 2)
B = A.T @ A
A = A / norm(A)
B = B / norm(B)
# Positive CMF через факторизацию
K1, K2 = chol(A)

Out[12]: (array([[-0.57735027, -0.81649658],
       [-0.81649658,  0.57735027],
       [-0.4472136,  0.4472136]]), array([[ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136],
       [ 0.4472136,  0.4472136]]))
```

Рис. 5: Факторизация.Специальные матричные структуры

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook window titled "jupyter Lab4_статистический_анализ Last Checkpoint: hace 19 horas (autosaved)". The notebook has three cells:

- In [23]:** # Генерация матрицы A:

```
кнопка A + AT
# Вычисление разложения симметризованной матрицы
кнопка D + квадратные скобки
```
- In [24]:** # Транспонирование, что получится одиничная матрица:

```
print(keyMatrix)
keyMatrix
```

Out[24]:

3x3	3x3	3x3
1.0	0.88178e-18	-4.55391e-15
-1.13922e-15	1.0	2.08850e-15
-5.42856e-16	5.42856e-16	1.0
- In [25]:** # Матрица 1000 x 1000:

```
# вводим n,x
# n = random(n,x)
```

Out[25]:

1000x1000 Matrix{Float64}:
2.8729 -0.488235 -0.389774 -2.14806 -0.417093 -1.153524 0.941813 -0.183333 1.84961 -0.742105 -0.8790104 -0.680944
1.54102 0.622392 0.729518 0.984617 0.239413 -1.20987 -1.04753 -1.00884 0.183333 5.686418 0.7198688 -0.088108 0.986437 -0.417258 -0.062853 0.278522
-3.31535 -0.31535 -0.958135 0.358135 -1.25986 -0.125986 -0.644968 0.363376 -2.42734 -0.208333 -0.7198688 -0.088108 0.986437 -0.417258 -0.062853 0.278522 1.5634

Рис. 6: Факторизация.Специальные матричные структуры

Выполнение лабораторной работы

The screenshot shows a Jupyter Notebook interface with three code cells:

- In [29]:** A code cell containing Python imports and a call to `eigvals` for a symmetric matrix.
- In [30]:** A code cell containing a comment about matrix factorization and calling `eigvals` again.
- In [31]:** A code cell containing a comment about matrix factorization and calling `eigvals` again.

The notebook header indicates it is titled "Lab4_статистический_шифр" and has a last checkpoint from 19 hours ago. The kernel is set to "Julia 1.12".

```
import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметрической матрицы
@time eigvals(Asym2)

Resolving package versions...
Installed Compat - v4.18.1
Installed BenchmarkTools - v1.3.3
Updating .julia/environments/v1.12/Project.toml
  [4e4d8ff3] + BenchmarkTools v1.3.3
  [4ecdd993] + Compat v4.18.1
  [38c2dca8] + LinearAlgebra v1.11.0
  [9abed945] + Profile v1.11.0
Precompiling packages...
Profile: 0.5 ms
  5272.5 ms ✓ Compat - CompatLinearAlgebraExt
  4795.0 ms ✓ Profile
  4795.0 ms ✓ BenchmarkTools
4 dependencies successfully precompiled in 7 seconds. 50 already precompiled.

07.421 ms (18 allocations: 0.06 MiB)

In [30]: # Оценка эффективности выполнения операции по нахождению
# собственных значений симметрической матрицы
@time eigvals(Symm_toSym)
```

```
882.553 ms (27 allocations: 7.99 MiB)

In [31]: # Оценка эффективности выполнения операции по нахождению
# собственных значений симметрической матрицы,
# для которой явно указано, что она симметрическая
@time eigvals(Symm_explicit)
```

```
65.258 ms (18 allocations: 0.06 MiB)
```

Рис. 7: Факторизация.Специальные матричные структуры

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook interface with the title "jupyter Лаба_статистический_анализ Last Checkpoint: hace 19 hours (auto saved)". The notebook has three cells:

- In [34]:** A comment block starting with `# Матрица с рациональными элементами:`. It defines a matrix `A = Matrix(Rational(BigInteger))` with dimensions `(rand(1):10, 3, 311/18)`. It also defines a vector `x = f11111, 3` and a rational number `b = Rational(18)`.
- Out [34]:** The output is a 3-element vector of type `Rational(BigInteger)`:
 $\begin{aligned} & 12/5 \\ & 3/2 \\ & 9/5 \end{aligned}$
- In [35]:** A comment block stating "# Решение исходного уравнения получаем с помощью функции l." followed by `Arational1.b`.
- Out [35]:** The output is a 3-element vector of type `Rational(BigInteger)`:
 $\begin{aligned} & 1 \\ & 1 \\ & 1 \end{aligned}$
- In [36]:** A comment block stating "# LU-разложение:" followed by `lu(Arational1)`.
- Out [36]:** The output consists of three parts:
 - `L, Factor:` A 3x3 matrix of type `Rational(BigInteger)`:
 $\begin{matrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 5/7 & 5/7 & 1 \end{matrix}$
 - `U:` A 3x3 matrix of type `Rational(BigInteger)`:
 $\begin{matrix} 4/5 & 9/10 & 7/10 \\ 0 & -1/5 & -1/2 \\ 0 & 0 & -3/2 \end{matrix}$

Рис. 8: Общая линейная алгебра

Задания для самостоятельного выполнения

Задания для самостоятельного выполнения

```
In [38]: using LinearAlgebra  
  
println("→4.4.1 Producto de vectores")  
  
#1 vector y operaciones  
v = [1, 2, 3, 4, 5]  
  
#producto scalar  
dot_v = dot(v, v)  
println("1. Producto escalar v.v = $dot_v")  
  
→4.4.1 Producto de vectores  
1. Producto escalar v.v = 55
```

Рис. 9: Произведение векторов

Задания для самостоятельного выполнения

```
In [39]: #producto exterior  
outer_v = v * v'  
println("\n2. Producto exterior:")  
display(outer_v)
```

2. Producto exterior:

5×5 Matrix{Int64}:

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

Рис. 10: Произведение векторов

Задания для самостоятельного выполнения

The screenshot shows a Jupyter Notebook cell with the following Python code:

```
#!/usr/bin/env python3
# coding: utf-8
# This program solves linear systems
# using Gaussian elimination method
# Author: S. V. Mikhalev
# Date: 2019-09-12
# Version: 1.0
# License: MIT
# GitHub: https://github.com/mikhalevsv/linear-equation-solver

from numpy import array, linalg
from sympy import Matrix, solve
from math import gcd

def solve_linear_system(A, b):
    """Solve linear system Ax = b using Gaussian elimination method.
    A - matrix of coefficients
    b - vector of constants
    Returns solution vector x or None if system is inconsistent.
    """
    n = len(b)
    # Create augmented matrix [A|b]
    augmented_matrix = array([list(row) + [b[i]] for i, row in enumerate(A)])
    # Perform Gaussian elimination
    for i in range(n):
        # Make the diagonal element non-zero
        if augmented_matrix[i][i] == 0:
            for j in range(i+1, n):
                if augmented_matrix[j][i] != 0:
                    augmented_matrix[i], augmented_matrix[j] = augmented_matrix[j], augmented_matrix[i]
                    break
        else:
            # Divide the entire row by the diagonal element
            augmented_matrix[i] /= augmented_matrix[i][i]
        # Eliminate other elements in the column
        for j in range(i+1, n):
            if augmented_matrix[j][i] != 0:
                augmented_matrix[j] -= augmented_matrix[i] * augmented_matrix[j][i]
    # Back-substitution
    x = []
    for i in range(n-1, -1, -1):
        x.append(augmented_matrix[i][n] / augmented_matrix[i][i])
        for j in range(i-1, -1, -1):
            augmented_matrix[j][n] -= augmented_matrix[i][n] * augmented_matrix[j][i]
    return x

# Test cases
A1 = [[1, 2, 3], [2, 1, 3], [3, 3, 1]]
b1 = [1, 2, 3]
print(solve_linear_system(A1, b1))

A2 = [[1, 2, 3], [2, 1, 3], [3, 3, 1]]
b2 = [1, 2, 4]
print(solve_linear_system(A2, b2))

A3 = [[1, 2, 3], [2, 1, 3], [3, 3, 1]]
b3 = [1, 2, 5]
print(solve_linear_system(A3, b3))
```

Рис. 11: Системы линейных уравнений

Задания для самостоятельного выполнения

The screenshot shows a Java application window titled "Линейные уравнения". It displays three examples of linear equations:

- Example 1: $\begin{cases} x_1 + 2x_2 = 0 \\ x_1 - 3x_2 = 1 \end{cases}$
Method: прямой подстановкой (также бывает и методом исключения)
Answer: $x_1 = 0, x_2 = -0,5$
- Example 2: $\begin{cases} x_1 + 2x_2 = 0 \\ x_1 + 3x_2 = 1 \end{cases}$
Method: прямой подстановкой (также бывает и методом исключения)
Answer: $x_1 = 1, x_2 = -0,5$
- Example 3: $\begin{cases} x_1 + 2x_2 = 0 \\ x_1 + 3x_2 = 1 \end{cases}$
Method: прямой подстановкой
Answer: $x_1 = 0, x_2 = 0,5$

Рис. 12: Систем линейных уравнений

Задания для самостоятельного выполнения



Рис. 13: Систем лінійних рівнянь

Задания для самостоятельного выполнения



The screenshot shows a Jupyter Notebook interface with three code cells. The first cell contains:

```
In [1]: M = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
M * M
print("Коэффициент детерминации x = x * x * x")
print("Коэффициент детерминации y = y * y * y")
print("Коэффициент детерминации z = z * z * z")
```

The second cell contains:

```
In [2]: A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("A * B = ", A * B)
print("B * A = ", B * A)
print("A * A = ", A * A)
print("B * B = ", B * B)
```

The third cell contains:

```
In [3]: M = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
M * M
print("Матрица M")
print("Матрица M")
print("Матрица M")
print("Матрица M")
```

Рис. 14: Операции с матрицами

Задания для самостоятельного выполнения

The screenshot shows a Jupyter Notebook interface with three code cells and their outputs:

```
In [78]: b1 = sqrt(5 -2z -2z^3);  
2x3 Matrix(ComplexF64):  
 0.3889 -0.4569  
 -0.4569 -2.1889  
In [79]: M1_chrt = M1^(1/3)  
println("n\n")  
display(round.(M1_chrt, digits=4))  
0:  
0:  
2x2 Matrix(ComplexF64):  
 0.9711+0.433im -0.4711+0.433im  
 -0.4711+0.433im 0.4711+0.433im  
In [80]: m0  
m0 = (1 2; 2 3)  
try  
    M4_sqrt = sqrt(M4)  
    println("n(sqrt([1 2; 2 3])) no tiene raiz real definida positiva")  
    display(round.(M4_sqrt, digits=4))  
catch e  
    println("n(sqrt([1 2; 2 3])) no tiene raiz real definida positiva")  
    M4_sqrt = sqrt.(M4, atol=1e-10)  
    println("raiz compleja approximada")  
    display(round.(M4_sqrt_complex, digits=4))  
end  
  
di = sqrt([1 2; 2 3]);  
2x2 Matrix(ComplexF64):  
 0.5684+0.3516im 0.3284-0.2373im  
 0.3284-0.2373im 1.4895+0.1345im
```

Рис. 15: Операции с матрицами

Задания для самостоятельного выполнения

The screenshot shows a Jupyter Notebook interface with three code cells and their outputs.

In [83]:

```
# matriz triangular inferior
L_A = tril(A)
println("matriz triangular inferior")
display(L_A)
```

Output:

```
matriz triangular inferior
5x5 Matrix{Int64}:
 148   8   0   0   0
  97  106   0   0   0
   74   89  152   0   0
  168  131  144  54   0
  131   36   71  142  36
```

In [84]:

```
# matriz triangular superior
U_A = triu(A)
println("matriz triangular superior")
display(U_A)
```

Output:

```
matriz triangular superior
5x5 Matrix{Int64}:
 148   97   74  168  131
   0  106   89  131  36
   0   0  152  144   71
   0   0   0  54  142
   0   0   0   0  36
```

In [85]:

```
#eficiencia
println("Evaluacion de eficiencia:")
println("Tamaño de A: $(size(A))")
@time eigen(A);
println("tiempo para descomposición espectral")
```

Output:

```
Evaluacion de eficiencia:
Tamaño de A: (5, 5)
@time eigen(A);
  0.0000000000000002 seconds (2.000 CPU calls) ⚡
```

Рис. 16: Операции с матрицами

Задания для самостоятельного выполнения

The screenshot shows a Jupyter Notebook window titled "jupyter Lab4_статистический_анализ Last Checkpoint: hace 10 horas (autosaved)". The notebook contains the following code:

```
In [100]: function es_productiva_inversa(A)
n = size(A, 1)
E = Matrix{Float64}(I, n, n)

try
    inversa = Inv(E-A)
    return all(inversa .≈ -1e-10)
catch
    return false
end
end

Out[100]: es_productiva_inversa (generic function with 1 method)

In [101]: function es_productiva_spectral(A)
vals = eigenvals(A)
return all(abs.(vals) .> 1e-10)
end

Out[101]: es_productiva_spectral (generic function with 1 method)

In [106]: #1. Matrices del ejercicio 4.4.4.2
using LinAlg

# Matrices a evaluar
matrices = [
    (1, 2, 3, 4), "A = [[1; 2; 3; 4]]",
    (0.5, 1, 2, 3, 4), "A = 0.5*[[1; 2; 3; 4]]",
    (0.3, 1, 2, 3, 4), "A = 0.3*[[1; 2; 3; 4]]"
]

for (k, nombre) in matrices_economia
    println("definición: $(es_productiva_definicion(A))")
    println("(-A)^{-1} no negativo: $(es_productiva_inversa(A)))")
    println("Criterio spectral: $(es_productiva_spectral(A)))")
end
```

Рис. 17: Линейные модели экономики

Задания для самостоятельного выполнения

```

jupyter La64_STATISTICHESKIJ_16-18.xls Last Checkpoint: hace 19 horas (automated)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Julia 1.12

In [113]: # 4. Comparación de criterios
println("Vista. Resumen comparativo:")
println("Matriz A: $(definicion(A)))
println("Matriz B: $(definicion(B)))
println("C: $(definicion(C)))")
println("D: $(definicion(D)))")

todas_matrices = vcat(matrices_accesos, matrices_2)
nombres_unicos = []
matrices_unicas = []

# Eliminar duplicados
for A, nombre in todas_matrices
    if !ismember(nombre, nombres_unicos)
        push!(nombres_unicos, nombre)
        push!(matrices_unicas, A)
    end
end

# Crear matriz Xd
push!(nombres_unicos, "A_3x3")
push!(matrices_unicas, [0.1 0.2 0.3; 0.4 0.1 0.2; 0.1 0.3])

for A, nombre in zip(matrices_unicas, nombres_unicos)
    def = es_productivo_definicion(A, 98)
    inv_pos = es_productivo_inverso(A)
    imp = es_productivo_spectral(A)

    # Aclarar nombre para tabla
    nombre_corto = length(nombre) > 12 ? nombre[1:12] : "...".* nombre
    println("$(if ispalindrome(nombre_corto), 151, 152) $(imp) $(def) $(inv_pos) $(imp))")
end

4. Resumen comparativo:

```

Рис. 18: Линейные модели экономики

Выводы

В процессе выполнения данной лабораторной работы я изучила возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Список литературы

1. JuliaLang[Электронный ресурс].2024JuliaLang.orgcontributors.URL:<https://julialang.org/>(дата обращения:11.10.2024).
2. Julia 1.11 Documentation [Электронный ресурс]. 2024 JuliaLang.orgcontributors. URL: <https://docs.julialang.org/en/v1/> (дата обращения:11.10.2024)