Лабораторная работа № 4. Линейная алгебра

4.1. Цель работы

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

4.2. Предварительные сведения

Рассмотрим на примерах решение базовых задач линейной алгебры.

4.2.1. Поэлементные операции над многомерными массивами

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

```
# Массив 4х3 со случайными целыми числами (от 1 до 20):
a = rand(1:20,(4,3))

# Поэлементная сумма:
sum(a)
# Поэлементная сумма по столбцам:
sum(a,dims=1)
# Поэлементная сумма по строкам:
sum(a,dims=2)

# Поэлементное произведение:
prod(a)
# Поэлементное произведение по столбцам:
prod(a,dims=1)
# Поэлементное произведение по строкам:
prod(a,dims=2)
```

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics:

```
# Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics

# Вычисление среднего значения массива:
mean(a)
# Среднее по столбцам:
mean(a,dims=1)
# Среднее по строкам:
mean(a,dims=2)
```

4.2.2. Транспонирование, след, ранг, определитель и инверсия матрицы

Для выполнения таких операций над матрицами, как транспонирование, диагонализация, определение следа, ранга, определителя матрицы и т.п. можно воспользоваться библиотекой (пакетом) `LinearAlgebra`:

```
# Подключение пакета LinearAlgebra:
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra
#  Массив 4x4 со случайными целыми числами (от 1 до 20):
b = rand(1:20,(4,4))

# Транспонирование:
transpose(b)
# След матрицы (сумма диагональных элементов):
tr(b)
# Извлечение диагональных элементов как массив:
diag(b)
# Ранг матрицы:
rank(b)
# Инверсия матрицы (определение обратной матрицы):
inv(b)
# Определитель матрицы:
det(b)
# Псевдобратная функция для прямоугольных матриц:
pinv(a)
```

4.2.3. Вычисление нормы векторов и матриц, повороты, вращения

Для вычисления нормы используется `LinearAlgebra.norm(x)`.

Евклидова норма:

$$\|\vec{X}\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2};$$

p-норма:

$$\|\vec{A}\|_p = \left(\sum_{i=1}^{n} |a_i|^p \right)^{1/p}.$$

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

# Вычисление евклидовой нормы:
norm(X)
# Вычисление p-нормы:
p = 1
norm(X,p)
```

Евклидово расстояние между двумя векторами $\vec{X}$ и $\vec{Y}$ определяется как $||\vec{X} - \vec{Y}||_2$.

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

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

Угол между двумя векторами $\vec{X}$ и $\vec{Y}$ определяется как $\cos^{-1}\dfrac{\vec{X}^T\vec{Y}}{||\vec{X}||_2||\vec{Y}||_2}$.

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

Вычисление нормы для двумерной матрицы:

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

# Вычисление Евклидовой нормы:
opnorm(d)
# Вычисление p-нормы:
p=1
opnorm(d,p)

# Поворот на 180 градусов:
rot180(d)
# Переворачивание строк:
reverse(d,dims=1)
# Переворачивание столбцов
reverse(d,dims=2)
#
```

4.2.4. Матричное умножение, единичная матрица, скалярное произведение

```
# Матрица 2х3 со случайными целыми значениями от 1 до 10:
A = rand(1:10,(2,3))
# Матрица 3х4 со случайными целыми значениями от 1 до 10:
B = rand(1:10,(3,4))

# Произведение матриц A и B:
A*B

# Единичная матрица 3х3:
Matrix{Int}(I, 3, 3)

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

# тоже скалярное произведение:
X'Y
```

4.2.5. Факторизация. Специальные матричные структуры

В математике факторизация (или разложение) объекта — его декомпозиция (например, числа, полинома или матрицы) в произведение других объектов или факторов, которые, будучи перемноженными, дают исходный объект.

Матрица может быть факторизована на произведение матриц специального вида для приложений, в которых эта форма удобна. К специальным видам матриц относят ортогональные, унитарные и треугольные матрицы.

LU-разложение — представление матрицы A в виде произведения двух матриц L и U, L — нижняя треугольная матрица, а U — верхняя треугольная матрица. LU-разложение существует только в том случае, когда матрица A обратима, а все её ведущие (угловые) главные миноры невырождены.

Обращение матрицы A эквивалентно решению линейной системы $AX = I$, где X — неизвестная матрица, I — единичная матрица. Решение X этой системы является обратной матрицей A^{-1}.

LUP-разложение — представление матрицы A в виде произведения $PA = LU$, где матрица L является нижнетреугольной с единицами на главной диагонали, U — верхнетреугольная общего вида матрица, P — матрица перестановок, получаемая из единичной матрицы путём перестановки строк или столбцов.

QR-разложение матрицы — представление матрицы в виде произведения унитарной (или ортогональной) матрицы Q и верхнетреугольной матрицы R. QR-разложение применяется для нахождения собственных векторов и собственных значений матрицы.

Q является ортогональной матрицей, если $Q^TQ = I$, где I — единичная матрица.

Спектральное разложение матрицы A — представление её в виде произведения $A = V\Lambda V^{-1}$, где V — матрица, столбцы которой являются собственными векторами матрицы A, Λ — диагональная матрица с соответствующими собственными значениями на главной диагонали, V^{-1} — матрица, обратная матрице V.

Рассмотрим несколько примеров. Для работы со специальными матричными структурами потребуется пакет `LinearAlgebra`.

Решение систем линейный алгебраических уравнений $Ax = b$:

```
# 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
```

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

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

В результате получим примерно следующее (так как элементы исходной матрицы заданы случайным образом):

```
LU{Float64,Array{Float64,2}}
L factor:
3×3 Array{Float64,2}:
 1.0       0.0       0.0
 0.385734  1.0       0.0
 0.559959  0.460677  1.0

U factor:
3×3 Array{Float64,2}:
 0.938848  0.437076  0.279009
 0.0       0.490475  0.444175
 0.0       0.0       0.555299
```

Различные части факторизации могут быть извлечены путём доступа к их специальным свойствам:

```
# Матрица перестановок:
Alu.P
# Вектор перестановок:
Alu.p
# Матрица L:
Alu.L
# Матрица U:
Alu.U
```

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

```
# Решение СЛАУ через матрицу A:
A\b
# Решение СЛАУ через объект факторизации:
Alu\b
```

Аналогично можно найти детерминант матрицы:

```
# Детерминант матрицы A:
det(A)

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

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

```
# QR-факторизация:
Aqr = qr(A)
```

В результате получим примерно следующее (так как элементы исходной матрицы заданы случайным образом):

```
LinearAlgebra.QRCompactWY{Float64,Array{Float64,2}}
Q factor:
3×3 LinearAlgebra.QRCompactWYQ{Float64,Array{Float64,2}}:
 -0.31898    0.861403  -0.395268
 -0.826942  -0.45672   -0.327985
 -0.463054   0.222243   0.858015
R factor:
3×3 Array{Float64,2}:
 -1.13532  -0.789624  -0.830966
 0.0        0.472712   0.551501
 0.0        0.0        0.476454
```

По аналогии с LU-факторизацией различные части QR-факторизации могут быть извлечены путём доступа к их специальным свойствам:

```
# Матрица Q:
Aqr.Q
# Матрица R:
Aqr.R

# Проверка, что матрица Q - ортогональная:
Aqr.Q'*Aqr.Q
```

Примеры собственной декомпозиции матрицы A:

```
#  Симметризация матрицы A:
Asym = A + A'
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(Asym)
# Собственные значения:
AsymEig.values
#Собственные векторы:
AsymEig.vectors

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

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

```
# Матрица 1000 х 1000:
n = 1000
A = randn(n,n)

# Симметризация матрицы:
Asym = A + A'
# Проверка, является ли матрица симметричной:
issymmetric(Asym)
```

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

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

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

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

Далее для оценки эффективности выполнения операций над матрицами большой размерности и специальной структуры воспользуемся пакетом `BenchmarkTools`:

```
import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@btime eigvals(Asym);

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

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

Далее рассмотрим примеры работы с разряженными матрицами большой размерности.

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

```
# 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)
```

При попытке задать подобную матрицу обычным способом и посчитать её собственные значения, вы скорее всего получите ошибку переполнения памяти:

```
B = Matrix(A)
```

```
OutOfMemoryError()
...
```

4.2.6. Общая линейная алгебра

Обычный способ добавить поддержку числовой линейной алгебры - это обернуть подпрограммы *BLAS* и *LAPACK*. Собственно, для матриц с элементами `Float32`,`Float64`, `Complex {Float32}` или `Complex {Float64}` разработчики Julia использовали такое же решение. Однако Julia также поддерживает общую линейную алгебру, что позволяет, например, работать с матрицами и векторами рациональных чисел.

Для задания рационального числа используется двойная косая черта:

```
1//2
```

В следующем примере показано, как можно решить систему линейных уравнений с рациональными элементами без преобразования в типы элементов с плавающей запятой (для избежания проблемы с переполнением используем `BigInt`):

```
# Матрица с рациональными элементами:
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

# LU-разложение:
lu(Arational)
```

4.3. Задание

1. Используя Jupyter Lab, повторите примеры из раздела 4.2.
2. Выполните задания для самостоятельной работы (раздел 4.4).

4.4. Задания для самостоятельного выполнения

4.4.1. Произведение векторов

1. Задайте вектор v. Умножьте вектор v скалярно сам на себя и сохраните результат в `dot_v`.
2. Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной `outer_v`.

4.4.2. Системы линейных уравнений

1. Решить СЛАУ с двумя неизвестными.
   a) $\begin{cases} x + y = 2, \\ x - y = 3. \end{cases}$
   b) $\begin{cases} x + y = 2, \\ 2x + 2y = 4. \end{cases}$
   c) $\begin{cases} x + y = 2, \\ 2x + 2y = 5. \end{cases}$
   d) $\begin{cases} x + y = 1, \\ 2x + 2y = 2, \\ 3x + 3y = 3. \end{cases}$
   e) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ x - y = 3. \end{cases}$
   f) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ 3x + 2y = 3. \end{cases}$
2. Решить СЛАУ с тремя неизвестными.
   a) $\begin{cases} x + y + z = 2, \\ x - y - 2z = 3. \end{cases}$
   b) $\begin{cases} x + y + z = 2, \\ 2x + 2y - 3z = 4, \\ 3x + y + z = 1. \end{cases}$
   c) $\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 1. \end{cases}$
   d) $\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 0. \end{cases}$

4.4.3. Операции с матрицами

1. Приведите приведённые ниже матрицы к диагональному виду
   a) $\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$

b) $\begin{pmatrix} 1 & -2 \\ -2 & 3 \end{pmatrix}$

c) $\begin{pmatrix} 1 & -2 & 0 \\ -2 & 1 & 2 \\ 0 & 2 & 0 \end{pmatrix}$

2. Вычислите

a) $\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}^{10}$

b) $\sqrt{\begin{pmatrix} 5 & -2 \\ -2 & 5 \end{pmatrix}}$

c) $\sqrt[3]{\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}}$

d) $\sqrt{\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}}$

3. Найдите собственные значения матрицы A, если

$$A = \begin{pmatrix} 140 & 97 & 74 & 168 & 131 \\ 97 & 106 & 89 & 131 & 36 \\ 74 & 89 & 152 & 144 & 71 \\ 168 & 131 & 144 & 54 & 142 \\ 131 & 36 & 71 & 142 & 36 \end{pmatrix}.$$

Создайте диагональную матрицу из собственных значений матрицы A. Создайте нижнедиагональную матрицу из матрица A. Оцените эффективность выполняемых операций.

4.4.4. Линейные модели экономики

Линейная модель экономики может быть записана как СЛАУ

$$x - Ax = y,$$

где элементы матрицы A и столбца y — неотрицательные числа. По своему смыслу в экономике элементы матрицы A и столбцов x, y не могут быть отрицательными числами.

1. Матрица A называется продуктивной, если решение x системы при любой неотрицательной правой части y имеет только неотрицательные элементы x_i. Используя это определение, проверьте, являются ли матрицы продуктивными.

a) $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

2. Критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все элементы матрица
$$(E - A)^{-1}$$
являются неотрицательными числами. Используя этот критерий, проверьте, являются ли матрицы продуктивными.
   a) $\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$
   b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$
   c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$
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.
   a) $\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$
   b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$
   c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$
   d) $\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0 & 0.1 & 0.2 \\ 0 & 0.1 & 0.3 \end{pmatrix}$

4.5. Содержание отчёта

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка задания работы.
3. Описание выполнения задания:
   – подробное пояснение выполняемых в соответствии с заданием действий;
   – скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
   – листинги (исходный код) программ и результаты его выполнения;
4. Выводы, согласованные с заданием работы.