Отчёт по лабораторной работе №3

Дисциплна: Научное программирование

Живцова Анна, 1132249547

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

- Изучить основы языков научного программирования Octave и Julia
- Выполнить практические примеры для закрепления синтаксиса
- Проверить эффективность векторных вычислений

2 Задание

- Изучить типы переменных
- Выполнить примеры операций из линейной алгебры
- Освоить процедуру рисования графиков
- На примере сравнить скорость выполнения программ, записанных через цикл и векторные операции

3 Теоретическое введение

Octave – язык научного программирования ысокого уровня. Задуманный изначально как программное пособие для проектирования химического реактора и названный в честь профессора химии Октава Левеншпиля, преподававшего автору математического пакета, Octave призван был заменить студентам Техасского Университета сложный в отладке Fortran. Версия 1.0 вышла в свет 17 февраля 1994 г.

GNU Octave — свободная программная система для математических вычислений, использующая совместимый с MATLAB язык высокого уровня. Предоставляет интерактивный командный интерфейс для решения линейных и нелинейных математических задач, а также проведения других численных экспериментов. Кроме того, Octave можно использовать для пакетной обработки. Язык Octave оперирует арифметикой вещественных и комплексных скаляров, векторов и матриц, имеет расширения для решения линейных алгебраических задач, нахождения корней систем нелинейных алгебраических уравнений, работы с полиномами, решения различных дифференциальных уравнений, интегрирования систем дифференциальных и дифференциально-алгебраических уравнений первого порядка, интегрирования функций на конечных и бесконечных интервалах. Система написана на C++ с использованием стандартной библиотеки шаблонов [1].

Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с син-

таксисом других математических языков (например, MATLAB и Octave), однако имеет некоторые существенные отличия. Julia написан на Си, C++ и Scheme. Имеет встроенную поддержку многопоточности и распределённых вычислений, реализованные в том числе в стандартных конструкциях.

Язык является динамическим, при этом поддерживает JIT-компиляцию (JIT-компилятор на основе LLVM входит в стандартный комплект), благодаря чему, по утверждению авторов языка, приложения, полностью написанные на языке (без использование низкоуровневых библиотек и векторных операций) практически не уступают в производительности приложениям, написанным на статически компилируемых языках, таких как Си или C++. Большая часть стандартной библиотеки языка написана на нём же [2].

4 Выполнение лабораторной работы

1. Типы данных и операции. Инициализировала переменные типа вектор-строка, вектор-столбец, матрица (см рис. 4.1 и 4.2). Выполнила арифметические операции (сложение, вычитание, умножение, возведение в степень), а также операции линейной алгебры: сложение векторов, умножения на скаляр, скалярное умножение, векторное умножение, матричное умножение, трансппонирование, сложениие матриц и обращение матриц. Нашла проекцию вектора на вектор, норму вектора, а также определитель, ранг и собственные значения матрицы (см рис. 4.3, 4.5, 4.4, 4.6).

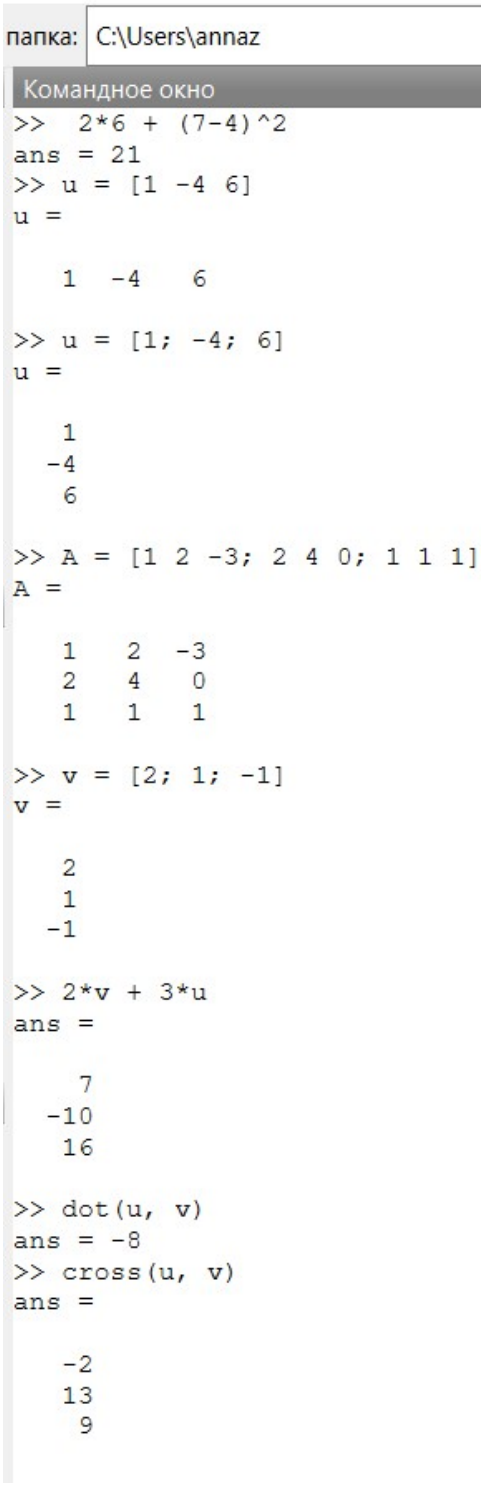


Рис. 4.1: Типы данных. Octave

```
2*6 + (7-4)^2
```

```
21
```

```
u = [1 -4 6]
```

```
1×3 Matrix{Int64}:
 1  -4  6
```

```
u = [1; -4; 6]
```

```
3-element Vector{Int64}:
  1
 -4
  6
```

```
A = [1 2 -3; 2 4 0; 1 1 1]
```

```
3×3 Matrix{Int64}:
 1  2  -3
 2  4   0
 1  1   1
```

```
u = [1; -4; 6]
```

```
3-element Vector{Int64}:
  1
 -4
  6
```

```
v = [2; 1; -1]
```

```
3-element Vector{Int64}:
  2
  1
 -1
```

```
2*v + 3*u
```

```
3-element Vector{Int64}:
   7
 -10
  16
```

```
using LinearAlgebra
```

```
dot(u, v)
```

```
-8
```

Рис. 4.2: Типы данных. Julia

```
>> norm(u)
ans = 7.2801
>> u = [3 5]
u =

   3   5

>>  v = [7 2]
v =

   7   2

>> proj = dot(u, v)/(norm(v))^2 * v
proj =

   4.0943   1.1698

>> B = [1 2 3 4; 0 -2 -4 6; 1 -1 0 0]
B =

   1   2   3   4
   0  -2  -4   6
   1  -1   0   0

>> A * B
ans =

   -2    1   -5   16
    2   -4  -10   32
    2   -1   -1   10

>> B' * A
ans =

    2    3   -2
   -3   -5   -7
   -5  -10   -9
```

Рис. 4.3: Операции линейной алгебры. Octave

```
cross(u, v)
```

```
3-element Vector{Int64}:
 -2
 13
  9
```

```
norm(u)
```

```
7.280109889280518
```

```
u = [3 5]
v = [7 2]
dot(u, v)/(norm(v))^2 * v
```

```
1×2 Matrix{Float64}:
 4.09434  1.16981
```

```
B = [1 2 3 4; 0 -2 -4 6; 1 -1 0 0];
```

```
A * B
```

```
3×4 Matrix{Int64}:
 -2   1   -5  16
  2  -4  -10  32
  2  -1   -1  10
```

```
B' * A
```

```
4×3 Matrix{Int64}:
  2    3   -2
 -3   -5   -7
 -5  -10   -9
 16   32  -12
```

Рис. 4.4: Операции линейной алгебры. Julia

```
>> 2 * A - 4 * eye(3)
ans =

  -2   4  -6
   4   4   0
   2   2  -2

>> det(A)
ans = 6
>> inv (A)
ans =

   0.6667  -0.8333   2.0000
  -0.3333   0.6667  -1.0000
  -0.3333   0.1667        0

>> eig (A)
ans =

   4.5251 +      0i
   0.7374 + 0.8844i
   0.7374 - 0.8844i

>> rank (A)
```

Рис. 4.5: Операции линейной алгебры (2). Octave

```
2 * A - 4*I
```

```
3×3 Matrix{Int64}:
 -2  4  -6
  4  4   0
  2  2  -2
```

```
det(A)
```

```
6.0
```

```
inv(A)
```

```
3×3 Matrix{Float64}:
  0.666667  -0.833333   2.0
 -0.333333   0.666667  -1.0
 -0.333333   0.166667   0.0
```

```
eigvals(A)
```

```
3-element Vector{ComplexF64}:
 0.7374488725928396 - 0.8843675977506604im
 0.7374488725928396 + 0.8843675977506604im
  4.525102254814321 + 0.0im
```

```
rank(A)
```

```
3
```

Рис. 4.6: Операции линейной алгебры (2). Julia

2. Рисование графиков. Освоила функцию рисования графиков и способы настройки внешнего вида графиков: подпись осей, совмещение нескольких графиков на одном рисунке, установка легенды, сетки, толщины линий, цвета линий, названия рисунка, диапазона осей (см рис. 4.7, 4.9, 4.11, 4.8, 4.10, 4.12).

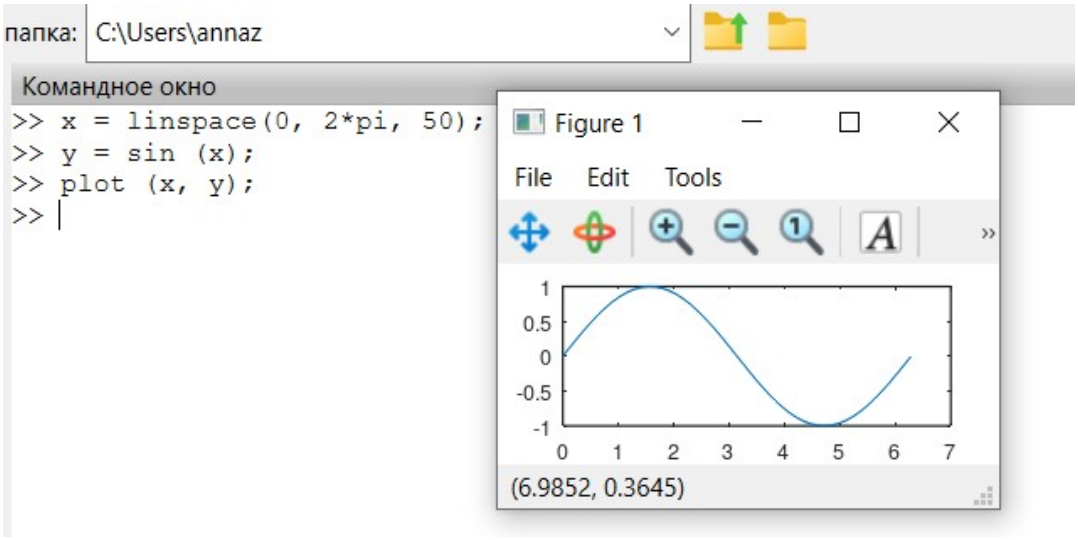


Рис. 4.7: Базовый вид графика. Octave

```
x = range(0,stop=2*pi,length=50);
```

```
y = sin.(x);
```

```
using Plots
```

```
plot(x, y)
```

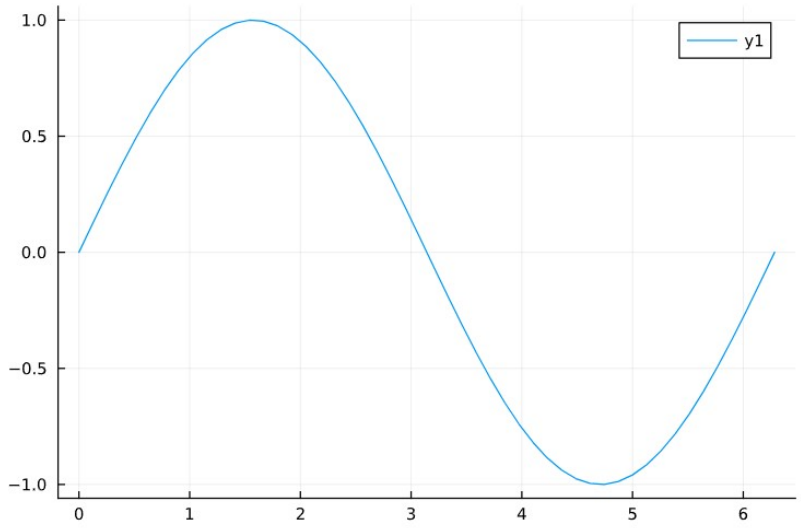


Рис. 4.8: Базовый вид графика. Julia

```
>> plot (x, y, 'r' , 'linewidth', 3)
>> axis([0 2*pi -1 1]);
>> grid on
>> xlabel ('x');
>> ylabel('y');
>>  title ('Sine graph');
>> legend ('y=sin(x)');
>>
```

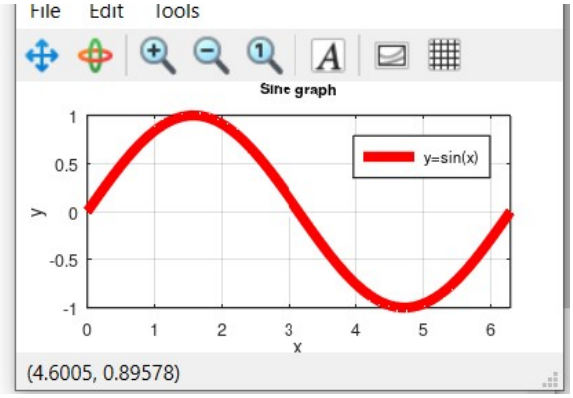


Рис. 4.9: График с настроенным внешним видом. Octave

```
plot(x, y, linewidth=3, lc=:red, minorgrid=true, label="y=sin(x)")
xlabel!("x")
ylabel!("y")
title!("Sine graph")
```

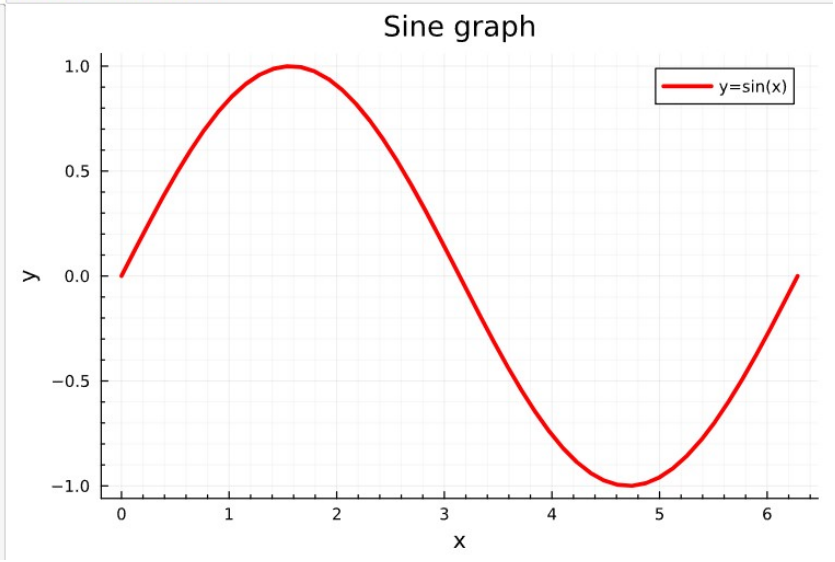


Рис. 4.10: График с настроенным внешним видом. Julia

```
>> x = [1 2 3 4];
>> y = [1 2 5 4];
>> plot (x , y , 'o')
>> hold on
>> plot (x, 1.2*x)
>> grid on;
>>  axis ([0 5 0 6]);
>> legend ('data points' , 'regress
>>
```

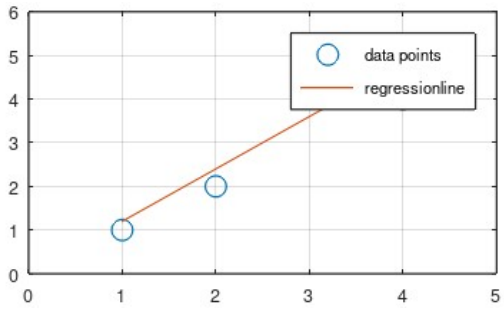


Рис. 4.11: Два графика на одномрисунке. Octave

```
x = [1 2 3 4];
y = [1 2 5 4];
scatter(x' , y', label="data points")
plot!(x', 1.2*x', label="regressionline")
```

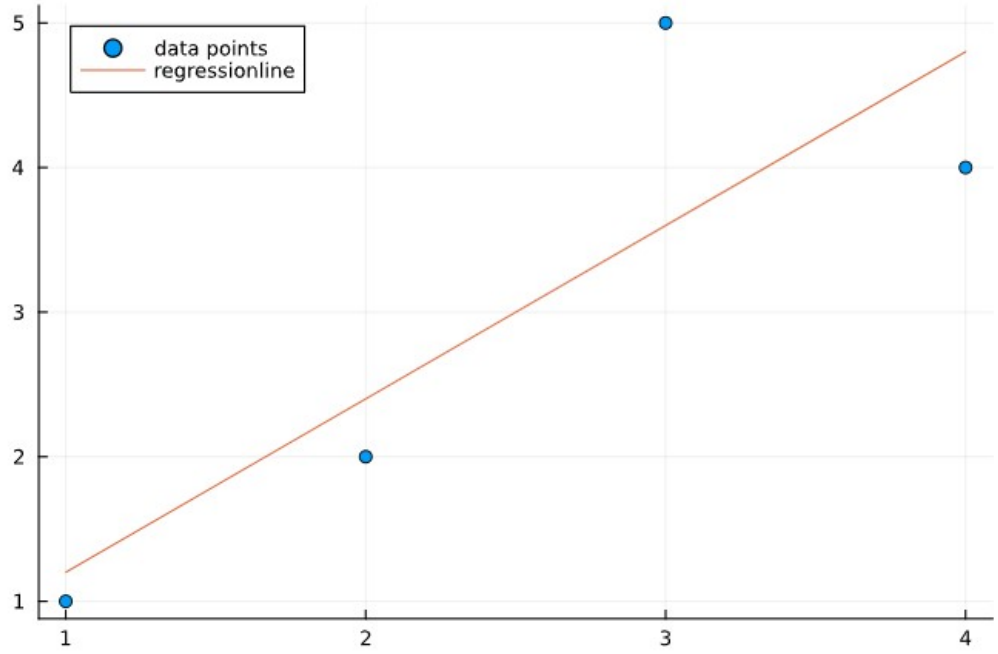


Рис. 4.12: Два графика на одномрисунке. Julia

3. Поэлемеентные операции. Изучила синтаксис поэлементных операций (см. рис 4.13 и 4.14). Проверила скорость выполнения программы при использовании цикла и поэлементных операций в векторах (см. рис 4.16 и 4.15).

```
>> x = linspace(-10, 10, 100);
>> plot (x, x^2*sin(x))
error: for x^y, only square matrix argu
>> plot (x, x.^2.*sin(x))
>> print graph2.png -dpng
>> print('graph2.pdf','-dpdf')
>>
```

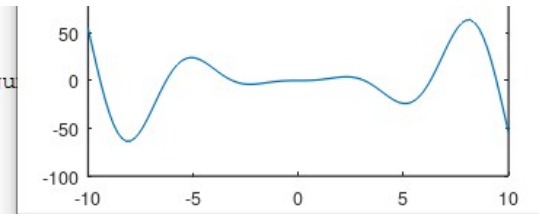


Рис. 4.13: Пример применения поэлементных операций. Octave

```
x = range(-10,stop=10,length=100);
plot(x, (x.^2).*sin.(x))
```

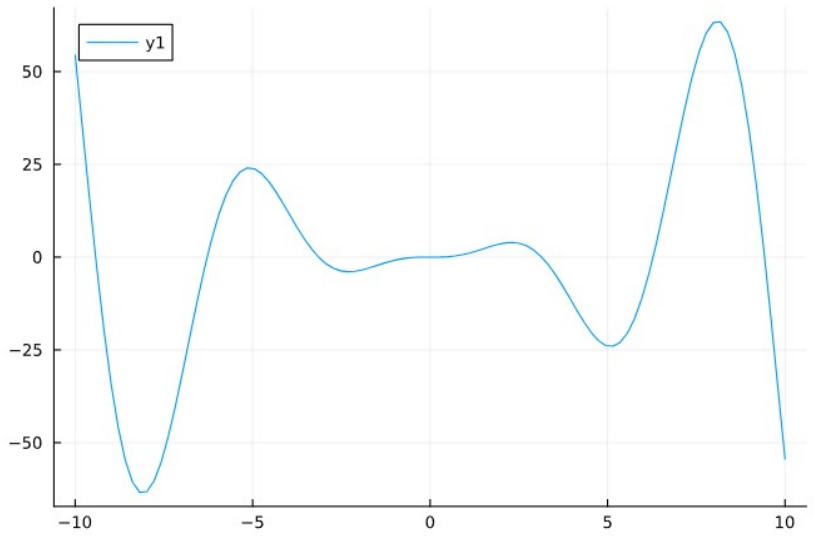


Рис. 4.14: Пример применения поэлементных операций. Julia

```
using BenchmarkTools
```

```
function loop()
    s = 0;
    for n = 1:100000
        s = s + 1/n^2;
    end
    s
end
```

```
loop (generic function with 1 method)
```

```
@btime loop()
```

```
  312.300 μs (0 allocations: 0 bytes)

1.6449240668982423
```

```
function ve()
    n = 1:100000;
    s = sum(1 ./ (n.^2));
    s
end
```

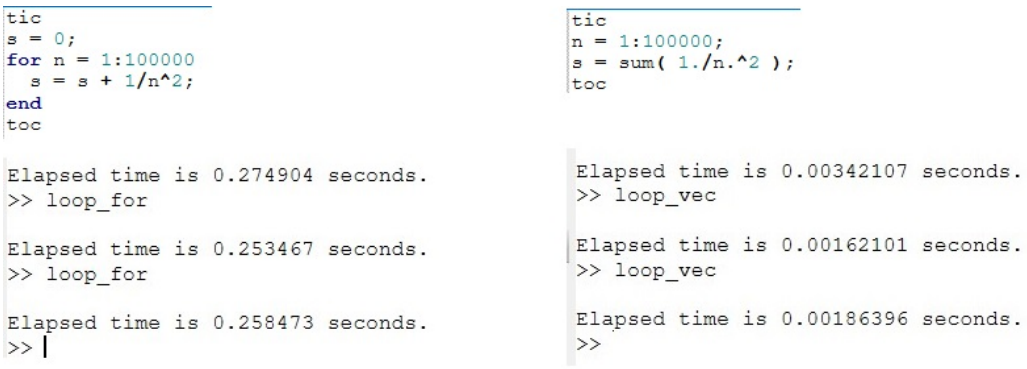
```
ve (generic function with 1 method)
```

```
@btime ve()
```

```
  137.300 μs (4 allocations: 781.31 KiB)

1.644924066898228
```

Рис. 4.15: Сравнение времени выполнения программы в зависимости от формы записи. Julia



```
tic
s = 0;
for n = 1:100000
  s = s + 1/n^2;
end
toc

Elapsed time is 0.274904 seconds.
>> loop_for

Elapsed time is 0.253467 seconds.
>> loop_for

Elapsed time is 0.258473 seconds.
>>
```

```
tic
n = 1:100000;
s = sum( 1./n.^2 );
toc

Elapsed time is 0.00342107 seconds.
>> loop_vec

Elapsed time is 0.00162101 seconds.
>> loop_vec

Elapsed time is 0.00186396 seconds.
>>
```

Рис. 4.16: Сравнение времени выполнения программы в зависимости от формы записи. Octave

5 Выводы

В данной работе я познакомилась с языками научного программирования Octave и Julia. Изучила некоторые типы данных, арифметические операции (сложение, вычитание, умножение, возведение в степень), а также операции линейной алгебры: сложение векторов, умножения на скаляр, скалярное умножение, векторное умножение, матричное умножение, трансппонирование, сложение матриц и обращение матриц. Нашла проекцию вектора на вектор, норму вектора, а также определитель, ранг и собственные значения матрицы. Освоила процедуру рисования и настройки внешнего вида (цвета, легенды, подписи осей, названия рисунка, толщины линий, сетки, масштаба) графиков, а также на примере сравнила скорость выполнения программ, записанных через цикл и векторные операции. Поэлементные операции с векторами показывают лучшую производительность.

6 Список литературы

1. GNU Octave documentation. The Octave Project Developers, 2024.
2. Julia documentation. Julia Programming Language., 2024.