

Лабораторная работа №3

Введение в Octave

Сырцов Александр Юрьевич

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	21

List of Tables

1 Цель работы

Познакомиться с языком Octave, изучить основные инструменты работы с векторами и матрицами.

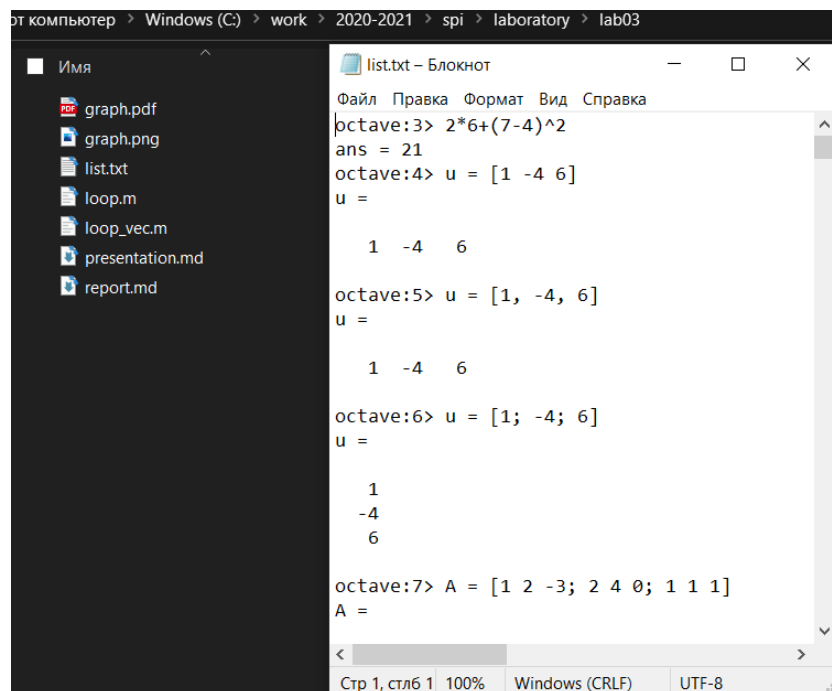
2 Задание

Подготовить отчёт.

3 Выполнение лабораторной работы

1. Включаю листинг команд и записываю всё в файл `list.txt`. Сам файл и его содержимое (рис. -fig. 3.1)

diary `list.txt`



The image shows a Windows file explorer window on the left with a dark theme, displaying a directory structure: `Имя`, `graph.pdf`, `graph.png`, `list.txt`, `loop.m`, `loop_vec.m`, `presentation.md`, and `report.md`. The right pane shows the contents of `list.txt` in a text editor. The text in the editor is as follows:

```
octave:3> 2*6+(7-4)^2
ans = 21
octave:4> u = [1 -4 6]
u =

    1   -4    6

octave:5> u = [1, -4, 6]
u =

    1   -4    6

octave:6> u = [1; -4; 6]
u =

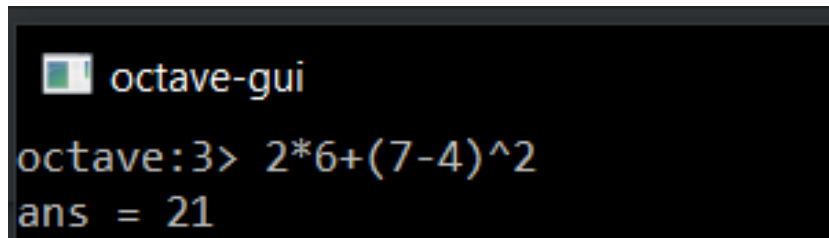
    1
   -4
    6

octave:7> A = [1 2 -3; 2 4 0; 1 1 1]
A =
```

The status bar at the bottom of the text editor indicates: `Стр 1, столб 1`, `100%`, `Windows (CRLF)`, and `UTF-8`.

Figure 3.1: 1

2. Использую Octave для вычисления математического выражения (рис. - fig. 3.2)

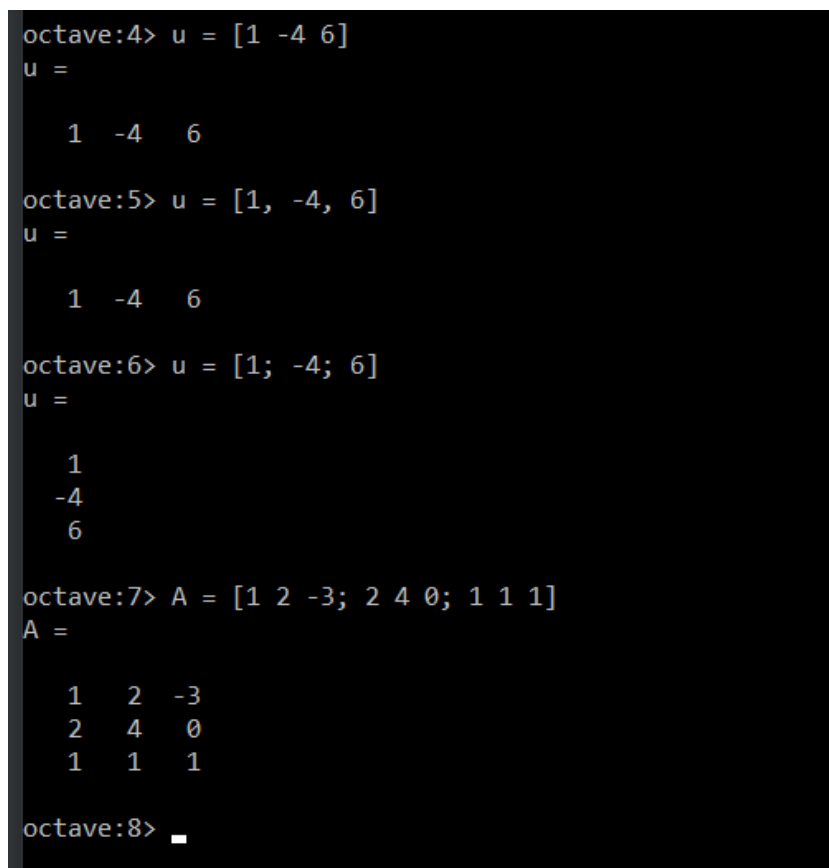


```
octave-gui
octave:3> 2*6+(7-4)^2
ans = 21
```

Figure 3.2: 2

3. Задаю конвектор u , или вектор-строку, двумя способами (используя стандартный синтаксис и синтаксис с запятыми). Также создаю вектор с теми же значениями, переопределяя u , и матрицу A (рис. -fig. 3.3)

уже сейчас можно сказать, что Octave обладает ленивой динамической типизацией.



```
octave:4> u = [1 -4 6]
u =
    1  -4   6

octave:5> u = [1, -4, 6]
u =
    1  -4   6

octave:6> u = [1; -4; 6]
u =
     1
    -4
     6

octave:7> A = [1 2 -3; 2 4 0; 1 1 1]
A =
     1     2    -3
     2     4     0
     1     1     1

octave:8> _
```

Figure 3.3: 3

1. Задав два вектора u и v , нашёл значение выражения $2v + 3u$, затем скалярное (в ответе число), векторное произведение (в ответе вектор) двух векторов и норму вектора u (рис. -fig. 3.4)

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

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

```
octave:9> 2*v + 3*u
ans =

     7
    -10
     16

octave:10> dot(u, v)
ans = -8
octave:11> dot(v, u)
ans = -8
octave:12> cross(u, v)
ans =

    -2
     13
     9

octave:13> norm(u)
ans = 7.2801
octave:14>  
```

Figure 3.4: 4

5. Вычисляю проекцию вектора u на v , перед этим переопределив их (рис. -fig. 3.5)

Стандартный инструмент для решения не задан, поэтому пользуемся выведе-

нием формулы проекции через формулу скалярного произведения.

```
octave:16> u = [3 5]
u =
    3    5

octave:17> v = [7 2]
v =
    7    2

octave:18> proj = dot(u, v)/(norm(v))^2 * v
proj =
    4.0943    1.1698
```

Figure 3.5: 5

6. Задаём матрицы A и B (рис. -fig. 3.6)

```
octave:19> A = [1 2 -3; 2 4 0; 1 1 1]
A =
    1    2   -3
    2    4    0
    1    1    1

octave:20> 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
```

Figure 3.6: 6

7. Вычислил их произведение, произведение с транспонированной матрицей B и выражение вида $2A - 4I$, I - единичная матрица (рис. -fig. 3.7)

```
octave-gui
octave:21> A*B
ans =

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

octave:22> B' * A
ans =

     2     3    -2
    -3    -5    -7
    -5   -10    -9
    16    32   -12

octave:23> 2*A - 4 * eye(3)
ans =

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

Figure 3.7: 7

Пример единичной матрицы 5x5 (рис. -fig. 3.8)

```
octave:24> eye(5)
ans =

Diagonal Matrix

    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1
```

Figure 3.8: 8

8. высчитаем ранг матрицы A (рис. -fig. 3.8)

```
octave:28> rank(A)
ans = 3
```

Figure 3.9: 9

9. Построим график, задав все значения x через range-функцию `linspace()` и определим нашу переменную y со всеми значениями во всех точках x. Чтобы получить график, используем функцию `plot()`, которая выводит непрерывную линию по умолчанию (рис. -fig. 3.10)

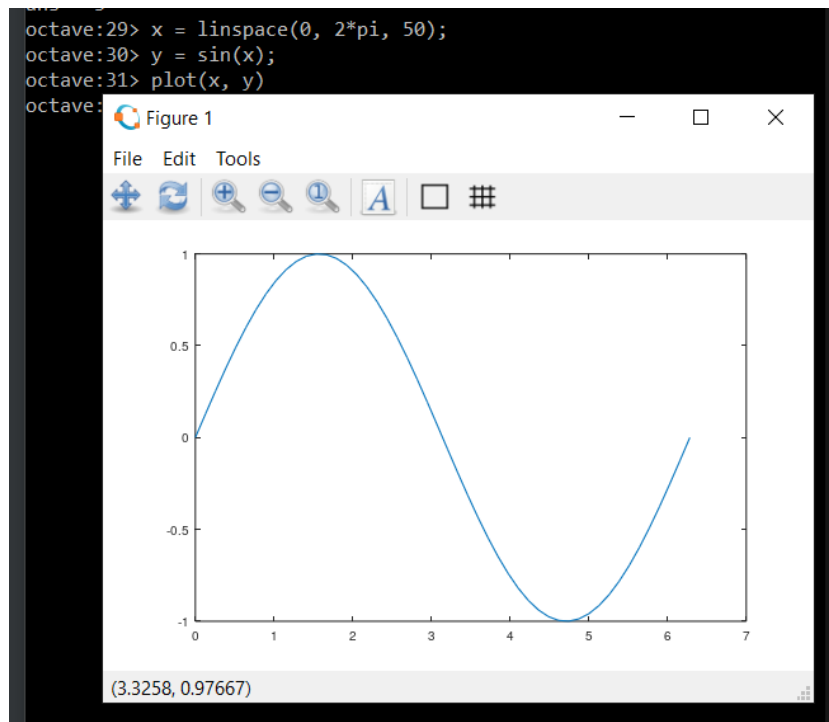


Figure 3.10: 10

10. Улучшим внешний вид графика, определив параметр цвета 'r' (можно было задать его через параметр 'r-', явно задав отображение в виде линии) и параметр ширины линии. Кроме того функцией `axis()` подраниваем значения осей, то есть меняем масштаб изображения, включаем сетку, добавляем подписи к осям и легенду (рис. -fig. 3.11)

Функция `plot()` принимает множественные параметры, поэтому, считывая параметр 'linewidth', считывает следующий за ним параметр, как значение ширины линии. По сути это своеобразная замена инструмента ключевых параметров.

```
% предварительно отчистим фигуру
clf
```

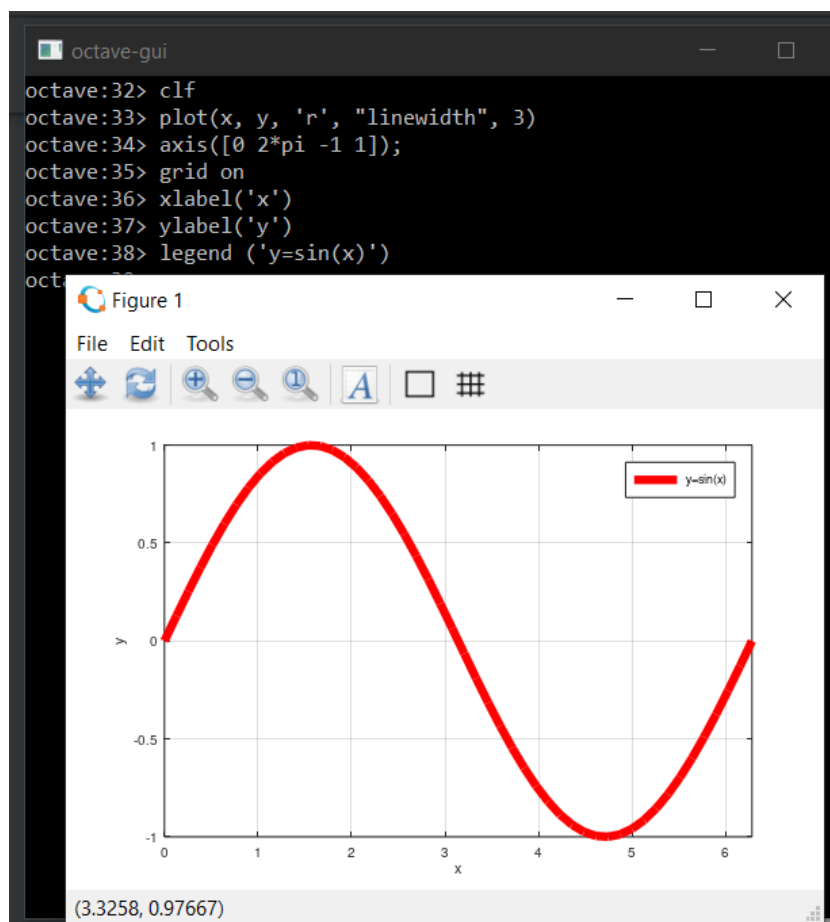


Figure 3.11: 11

11. Теперь изображаем две диаграммы: точки и прямую линейной регрессии, задав нужные значения (рис. -fig. 3.12)

```
octave-gui
octave:39> clear
octave:40> clf
octave:41> x = [1 2 3 4]
x =
    1    2    3    4
octave:42> y = [1 2 5 4]
y =
    1    2    5    4
octave:43> plot (x , y , 'o')
octave:44> hold on
octave:45> plot (x, 1.2*x)
octave:46> grid on
octave:47> axis ([0 5 0 6])
octave:48> legend ('data points' , 'regressionline')
octave:49>
```

Figure 3.12: 12

% аналогично можно было

% отобразить две диаграммы на одной фигуре так

```
plot(x, y, 'o', x, 1.2 * x, '-')
```

Итоговое изображение выглядит так (рис. -fig. 3.13)

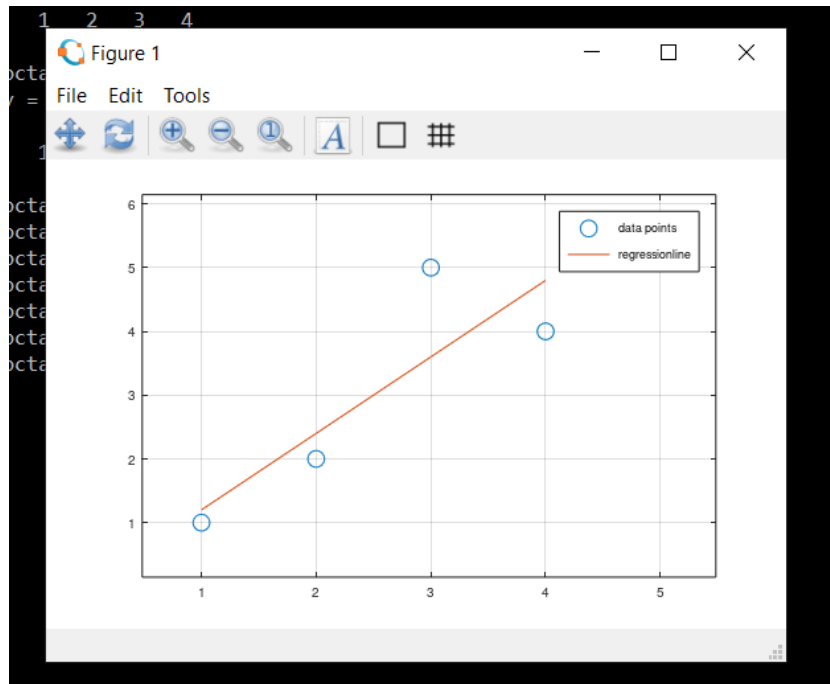


Figure 3.13: 13

12. Изобразим третий график, который потом сохраним двумя способами. для начала зададим x (рис. -fig. 3.14), далее отобразим график и сохраним в формате png (рис. -fig. 3.14)

```
octave:49> clear
octave:50> clf
octave:51> x = linspace(-10, 10, 100);
```

Figure 3.14: 14

```
octave:53> plot (x, x.^2.*sin(x))
octave:54> print graph.png -dpng
```

Figure 3.15: 15

% второй способ отличается тем,
 % что вместо макроса,
 % мы используем print в качестве функции

```
% конкретно тут сохраняем в формате pdf
print('graph2.pdf', '-dpdf')
```

Сам график выглядит так (рис. -fig. 3.16)

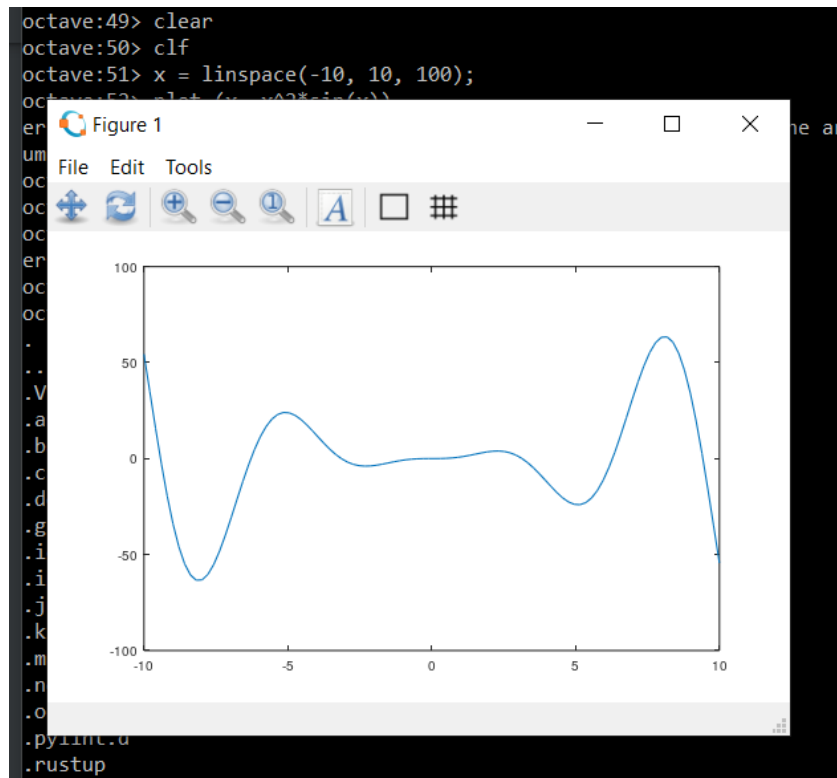


Figure 3.16: 16

То, что всё было сохранено, можно увидеть выше (рис. -fig. 3.1)

13. Сравним работу двух алгоритмов вычисления суммы элементов последовательности от 1 до 1000000: через range-цикл `for` с итератором и векторную форму (то есть через векторы и арифметические операторы с точкой)

Код с циклом `for` (рис. -fig. 3.17)

```
MINGW32:/c/work/2020-2021/spi/laboratory/lab03
1 clear
2 tic
3 s = 0;
4
5 for n = 1:100000
6     s += 1/n^2;
7 end
8 toc
```

Figure 3.17: 17

Код с векторами (рис. -fig. 3.17)

```
MINGW32:/c/work/2020-2021/spi/laboratory/lab03
1 clear
2 tic
3 n = 1:100000;
4 s = sum(1./n.^2);
5 toc
```

Figure 3.18: 18

Результаты оказались неожиданными – векторная форма отработала намного быстрее (рис. -fig. 3.19)

```
octave:1> cd C:\work\2020-2021\spi\laboratory\lab03
octave:2> loop
Elapsed time is 0.756483 seconds.
octave:3> loop_vec
Elapsed time is 0.00608397 seconds.
octave:4>
```

Figure 3.19: 19

Скорее всего, дело в реализации инструментов языка. По аналогии с Python, где цикл `for` реализован через язык C, поэтому работает быстрее, чем более нативный цикл `while`, можно предположить, что в Octave присутствует реализация на языках более низкого уровня абстракции.

4 Выводы

Мне удалось освоить начальные навыки владения языком Octave, с помощью его инструментов произвести операции с векторами, конвекторами, матрицами: создать и сохранить в разных форматах диаграммы и понять некоторые принципы его работы.