Отчёт по лабораторной работе №6

Сырцов Александр Юрьевич

Содержание

List of Tables

List of Figures

1 Цель работы

Научится находить частичные и полные суммы рядов, значения пределов и интегралов в Octave.

2 Задание

- Сделать отчёт по лабораторной работе в формате Markdown.
    - В качестве ответа предоставить отчёты в 3 форматах: pdf, docx и md (в архиве, поскольку он должен содержать скриншоты, Makefile и т.д.)

3 Выполнение лабораторной работы

3.1 Пределы, последовательности и ряды

1. Реализую функцию для оценки некоторого предела:

$$\lim_{n \to \infty} (1 + \frac{1}{n})^n$$

Функция реализована в функциональном стиле и представляет собой анонимное лямбда-выражение, присвоенное объекту `f`, которое в качестве своего терма принимает вектор значений `n`. Вместе с тем сразу задаём вектор `k` и меняем формат отображения чисел для следующего шага (рис. -fig. 3.1).

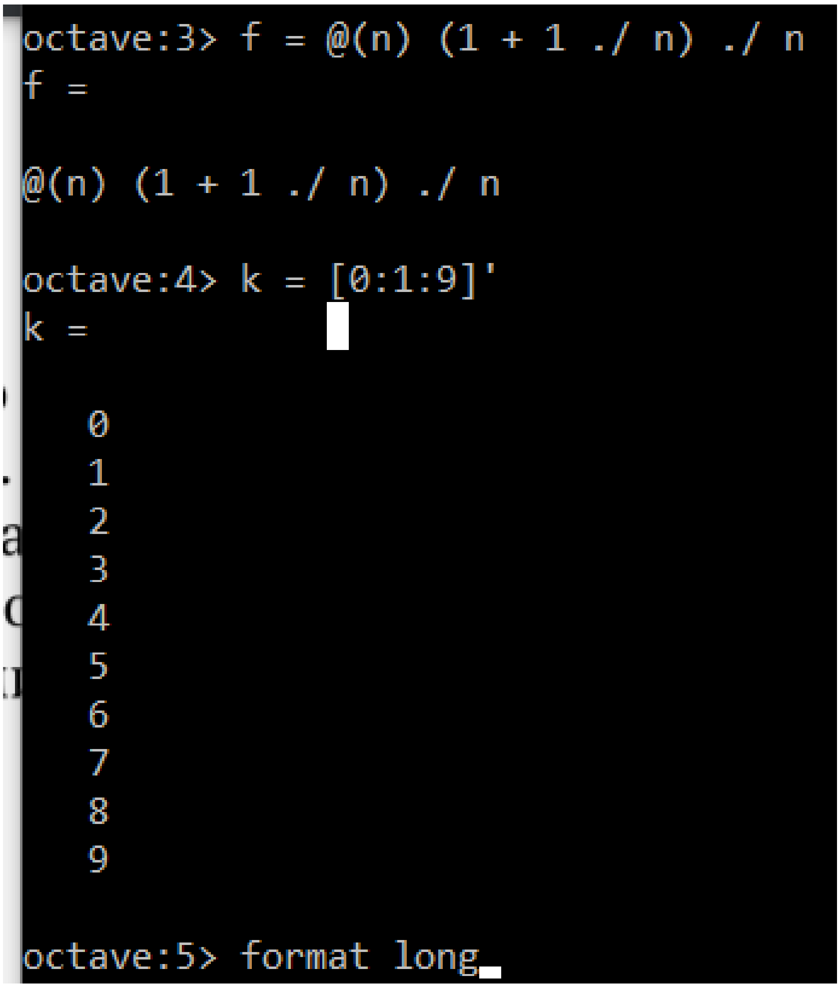


Figure 3.1: Анонимная функция и вектор *k*

Замечание 1: вектор `k` задан как транспонированный конвектор, что крайне удобно и быстро в сочетании с range-оператором.

Замечание 2: Реализованная функция чистая и является функцией первого класса, однако перестаёт быть анонимной после присвоения объекту `f`, так как он становится "allias" для изначального выражения.

2. Вектор индексов `k` определил собой количество итераций для вектора `n` (рис. -fig. 3.2), представляя собой степени от 0 до 9. Итоговый вектор значений – это вектор чисел от 1 до 1000000000. При его подстановке в функцию `f` мы постепенно приближаемся к окрестности нужного значения и по закономерности чисел можно сказать, к какому значению стремиться предел,

а именно к значению числа *e*, так как это второй замечательный предел (рис. -fig. 3.3).

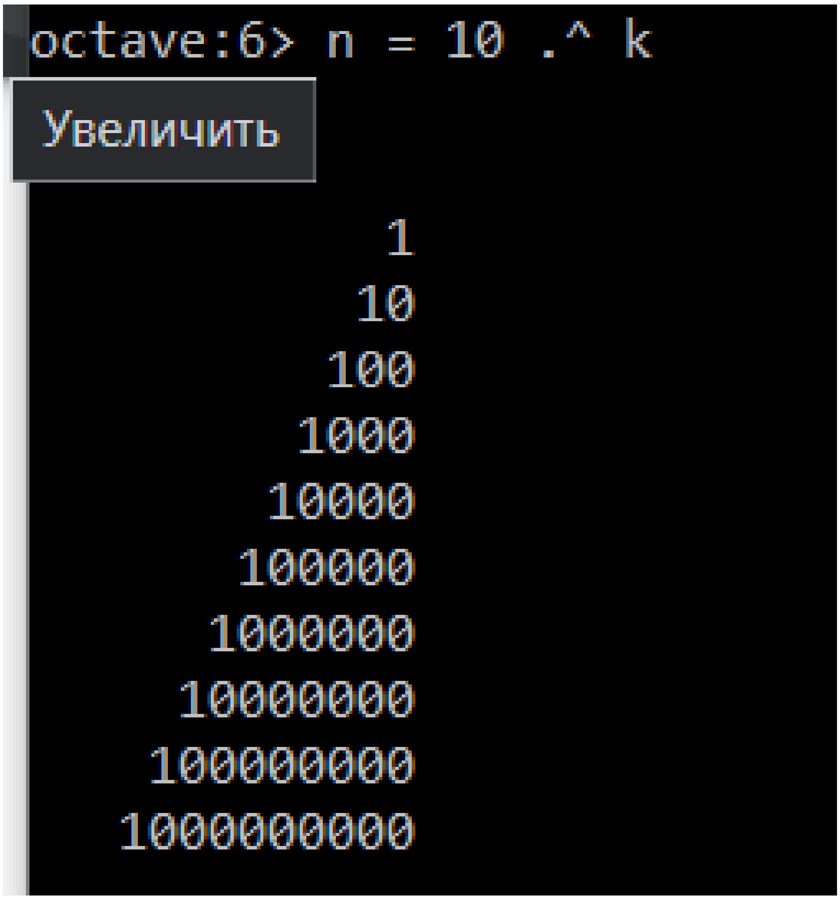


Figure 3.2: Инициалзация вектора значений

```
octave:7> f(n)
ans =

   2.00000000000000
   2.59374246010002
   2.70481382942153
   2.71692393223552
   2.71814592682436
   2.71826823719753
   2.71828046915643
   2.71828169398037
   2.71828178639580
   2.71828203081451
```

Figure 3.3: Подстановка значений в функцию

замечание: можно было не инициализировать отдельно k, а сразу подставить вектор для экономии памяти.

3.2 Частичные суммы

3. Необходимо рассчитать частичные суммы ряда от второго до одинадцатого элемента включительно:

$$\sum_{n=2}^{\infty} a_i$$

где

$$a_n = \frac{1}{n(n+2)}$$

Для этого я инициализирую вектор индексов и подставляю в n-й член ряда, получая десять элементов ряда (рис. -fig. 3.4).

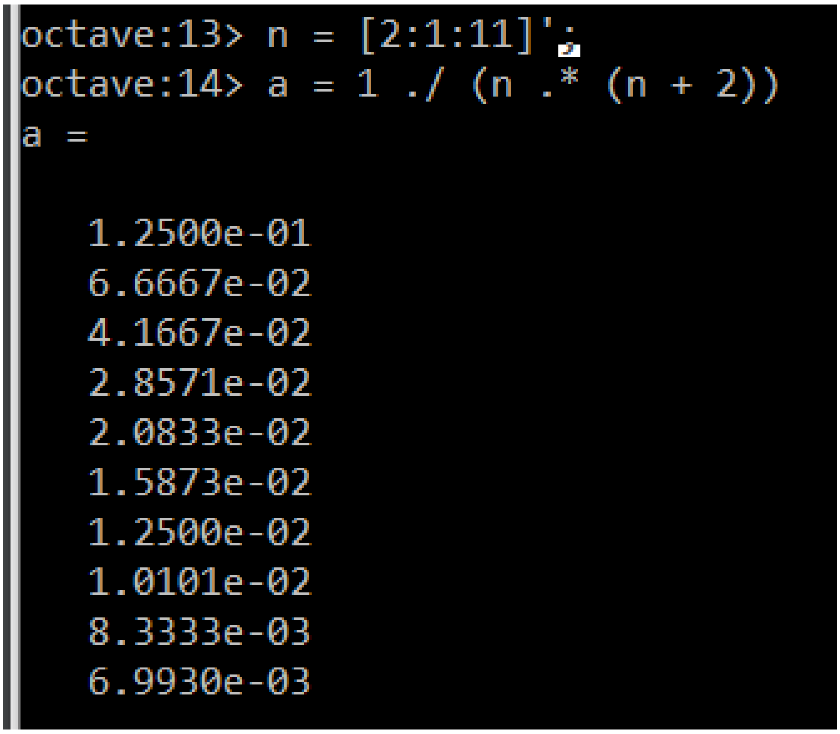


Figure 3.4: Значения для нахождения частичных сумм

4. Частичные суммы расчитываются, как суммы от одного элемента до другого, не покрывая исходный ряд. Следуя такой логике, реализую цикл от 1 до 10, где в конвектор частичных сумм записывается сумма от первого элемента вектора a до i-го с помощью функции `sum()`. В конце выодим все значения через транспонированный конвектор (рис. -fig. 3.5).

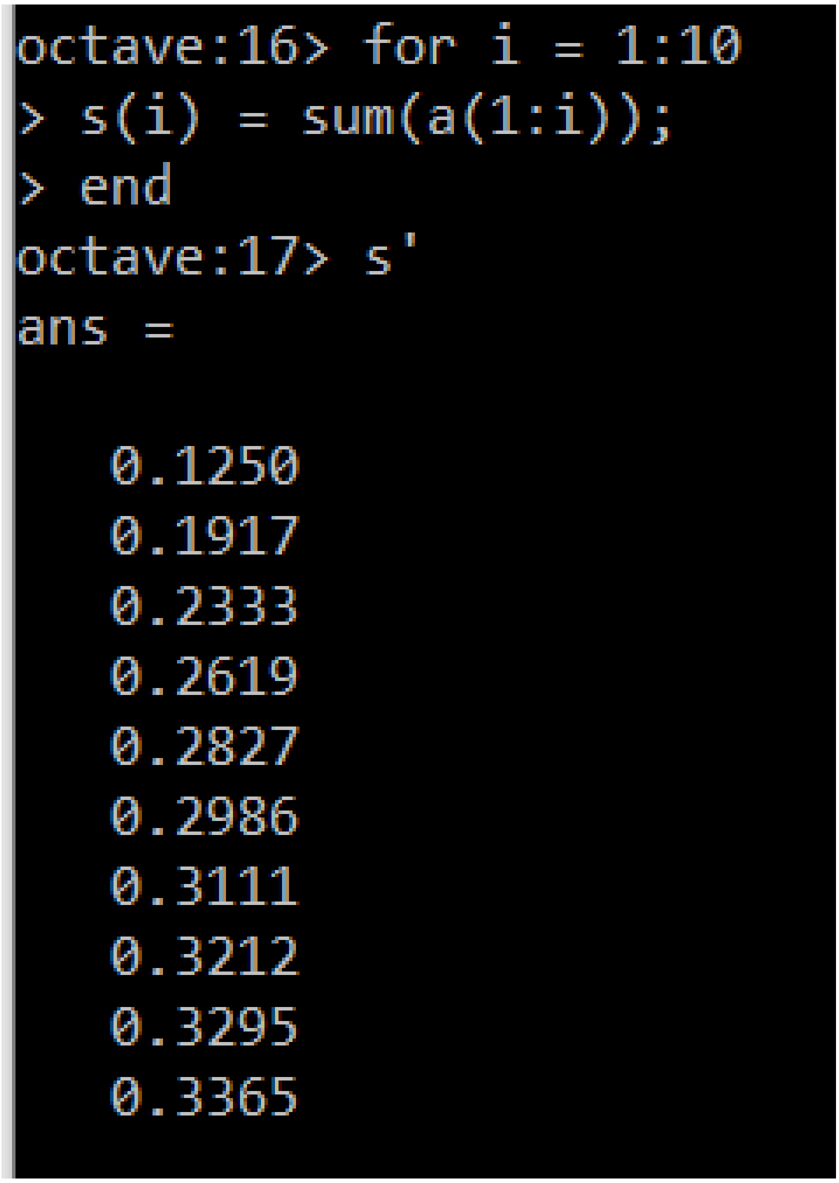


Figure 3.5: Частичные суммы

5. Визуализирую резултаты и очевидно получаю обратную корелляцию между значениями ряда и значением частичных сумм (рис. -fig. 3.6).

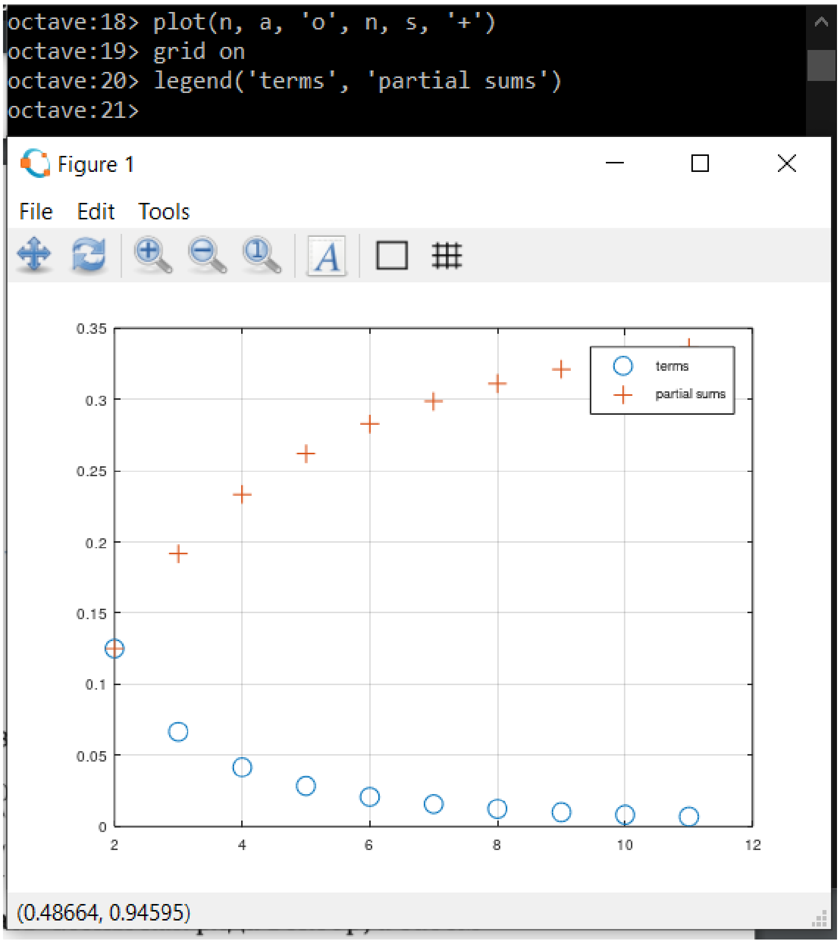


Figure 3.6: Граафик значений элементов ряда и частичных сумм ряда

3.3 Сумма ряда

6. Найду сумму элеметов горманического ряда:

$$\sum_{n=1}^{1000} \frac{1}{n}$$

Аналогично задав вектор индексов, находим элементы ряда и используем стандартную функцию `sum()` (рис. -fig. 3.7).

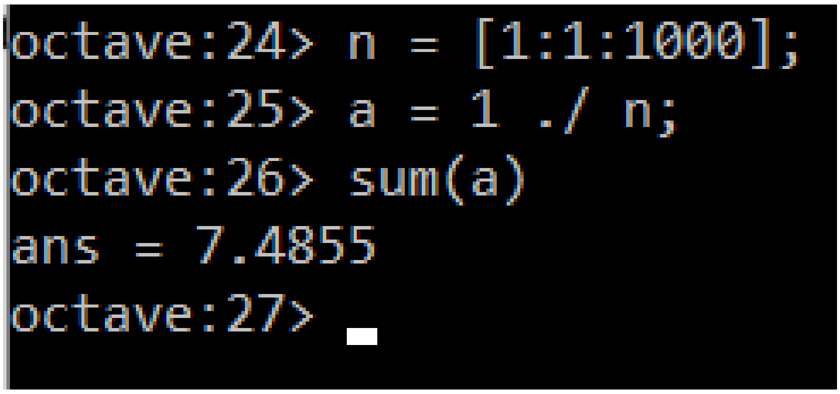


Figure 3.7: Сумма ряда

3.4 Численное интегрирование

7. Нахожу значение определённого интеграла:

$$\int_0^{\frac{\pi}{2}} e^{-x^2} \cos(x) \, dx$$

Для этого явно задаю функцию `f` и использую стандартную функцию `quad()` (рис. -fig. 3.8).

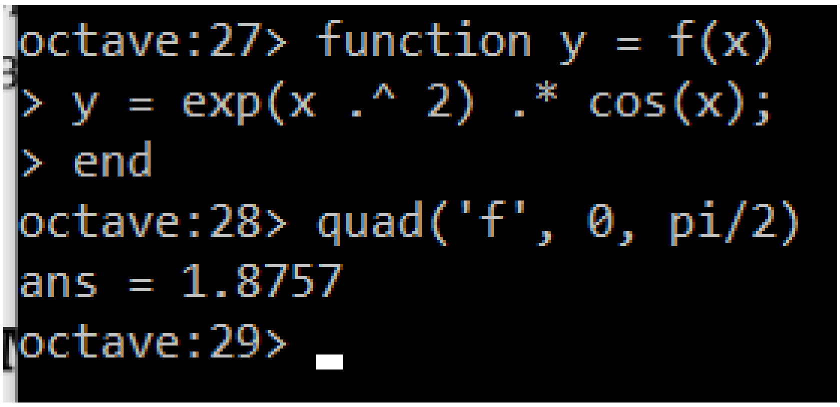


Figure 3.8: определённый интеграл функцией *quad()*

3.5 Аппроксимированние суммами

8. Рассчитаем тот же интеграл, но по правилу средней точки:

$$\int_0^{\frac{\pi}{2}} e^{-x^2} \cos(x)\, dx$$

Из комментариев к коду понятно, как работает правило и как работает код. Сначала реализация программы с использованием циклов `midpoint` (рис. -fig. 3.9).

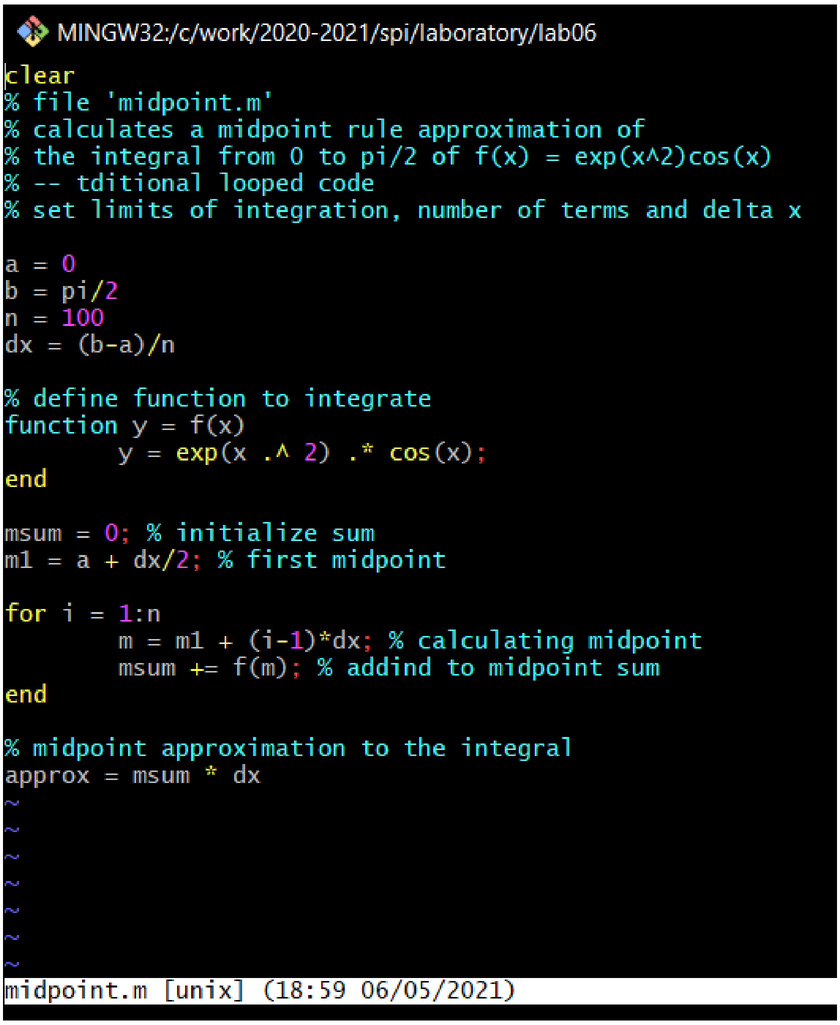
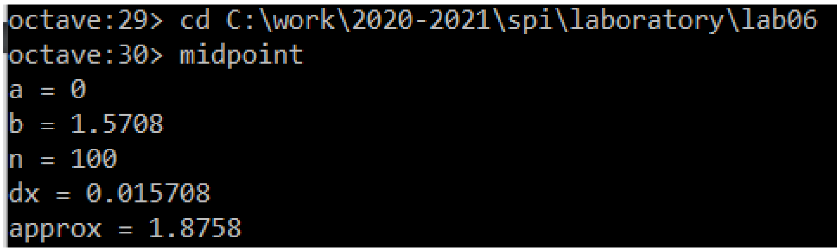


Figure 3.9: Реализация *midpoint*

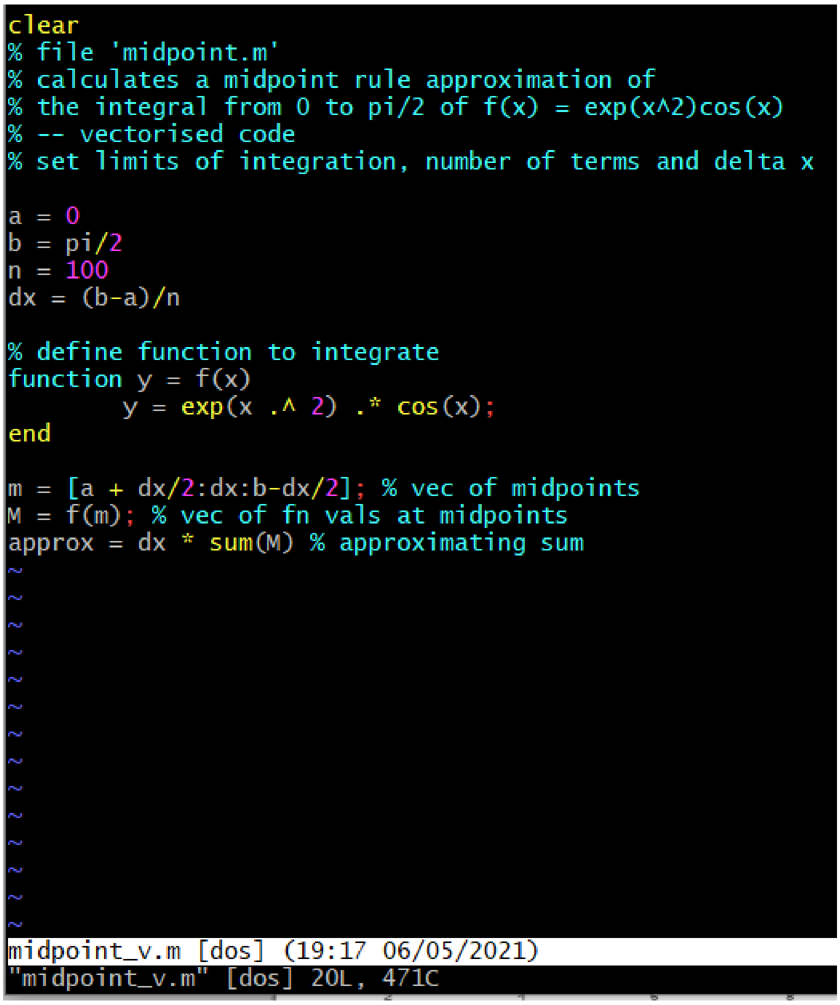
9. Запускаю код (рис. -fig. 3.10).



```
octave:29> cd C:\work\2020-2021\spi\laboratory\lab06
octave:30> midpoint
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
```

Figure 3.10: Запуск *midpoint*

10. Теперь аналогично реализую векторизованную программу `midpoint_v` (рис. -fig. 3.11) и запускаю (рис. -fig. 3.12).



```
clear
% file 'midpoint.m'
% calculates a midpoint rule approximation of
% the integral from 0 to pi/2 of f(x) = exp(x^2)cos(x)
% -- vectorised code
% set limits of integration, number of terms and delta x

a = 0
b = pi/2
n = 100
dx = (b-a)/n

% define function to integrate
function y = f(x)
        y = exp(x .^ 2) .* cos(x);
end

m = [a + dx/2:dx:b-dx/2]; % vec of midpoints
M = f(m); % vec of fn vals at midpoints
approx = dx * sum(M) % approximating sum
~
~
~
~
~
~
~
~
~
~
~
~
~
~
midpoint_v.m [dos] (19:17 06/05/2021)
"midpoint_v.m" [dos] 20L, 471C
```

Figure 3.11: Реализация *midpoint_v*

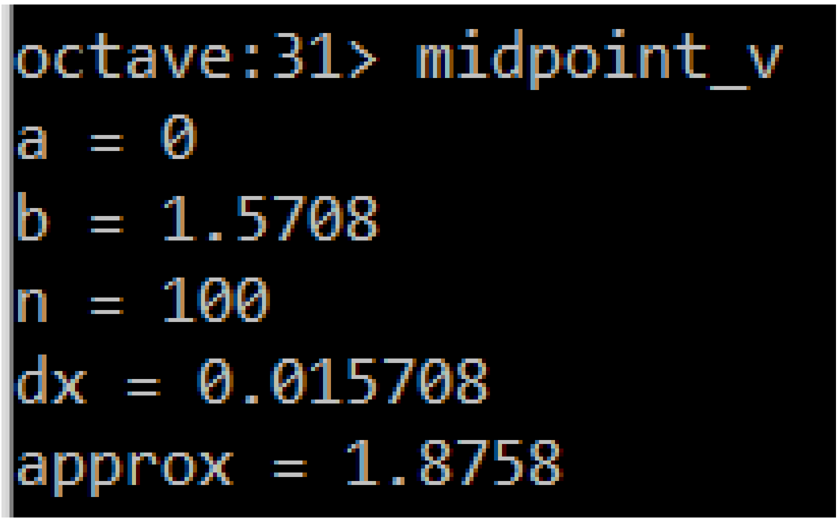


Figure 3.12: Запуск *midpoint_v*

11. Сравниваю результаты выполнения двух программ с помощью макросов (команд) `tic` и `toc`. Из результатов видно, что векторизванный код работает быстрее подобно тому, что мы наблюдали в одной из прошлых лабораторных (рис. -fig. 3.13).

```
octave:32> tic; midpoint; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.024925 seconds.
octave:33> tic; midpoint_v; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.012167 seconds.
```

Figure 3.13: Сравнение скорости исполнения кода

4 Выводы

Я успешно пополнил опыт вычисления приближённых и точных значений рядов, пределов, сумм и интегралов, научившись этому в языке программирования Octave.