

# **Лабораторная работа №6**

**Научное программирование**

Таубер Кирилл Олегович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>5</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>4</b>	<b>Вывод</b>	<b>13</b>

## Список иллюстраций

3.1	Оценка выражения под знаком предела . . . . .	7
3.2	Оценка выражения под знаком предела . . . . .	8
3.3	Частичные суммы . . . . .	9
3.4	Сумма ряда . . . . .	9
3.5	Вычисление интеграла . . . . .	10
3.6	Аппроксимирование суммами . . . . .	10
3.7	Аппроксимирование суммами . . . . .	11
3.8	Аппроксимирование суммами - векторизованный код . . . . .	11
3.9	Аппроксимирование суммами - векторизованный код . . . . .	11
3.10	Сравнение кодов . . . . .	12

# 1 Цель работы

Изучить в Octave методы расчета пределов, частичных сумм, суммы ряда, а также методы вычисления интегралов и аппроксимирования суммами.

## 2 Теоретическое введение

**Анонимная функция** - особый вид функций, которые объявляются в месте использования и не получают уникального идентификатора для доступа к ним. Обычно при создании анонимные функции либо вызываются напрямую, либо ссылка на функцию присваивается переменной, с помощью которой затем можно косвенно вызывать данную функцию.

В Octave анонимные функции определяются с помощью синтаксиса `@(argument-list) expression`. Любые переменные, которые не найдены в списке аргументов, наследованы от объема включения. Анонимные функции полезны для создания простых функций без имени от выражений или для обертывания вызовов к другим функциям для адаптации их к использованию функциями как `quad`, которая применяется при вычислении интегралов.

Более подробно см. в [`@Octave_1:bash`] и [`@Octave_2:bash`].

### 3 Выполнение лабораторной работы

Рассмотрим предел:

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

Оценим данное выражение. Для этого определим функцию с помощью метода анонимной функции. Создадим также индексную переменную, состоящую из целых чисел от 0 до 9. Возьмем степени 10, которые будут входными значениями, а затем оценим  $f(n)$  (рис. fig. 3.1) и (рис. fig. 3.2).

Имя

 diary

```
>> diary on
>> f=@(n) (1+1./n).^n
f =

@(n) (1 + 1 ./ n) .^ n

>> k=[0:1:9]'
k =

     0
     1
     2
     3
     4
     5
     6
     7
     8
     9

>> format long
>> n=10.^k
n =

         1
        10
       100
      1000
     10000
    100000
   1000000
  10000000
 100000000
1000000000
```

Рис. 3.1: Оценка выражения под знаком предела

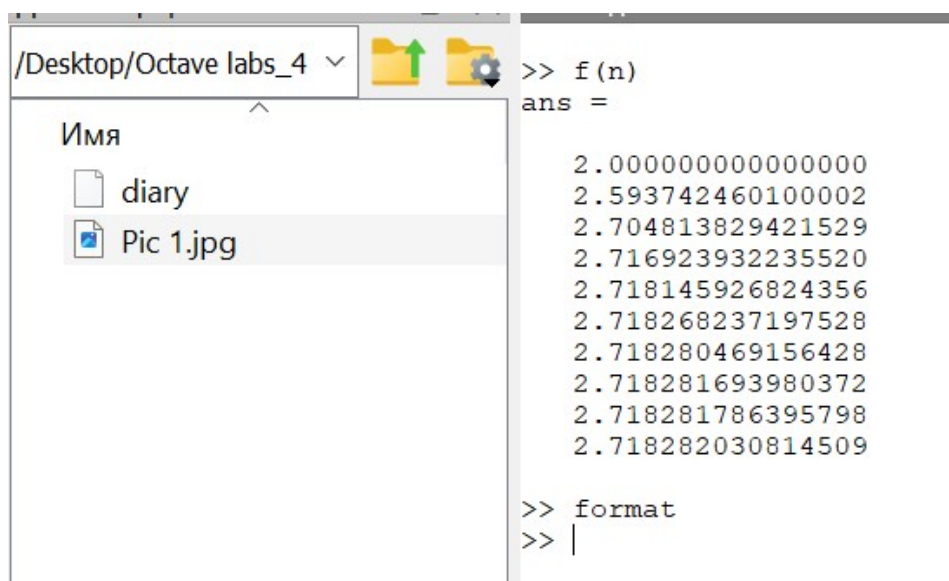


Рис. 3.2: Оценка выражения под знаком предела

Полученный результат близок к теоретическому значению предела -  $e$ .

Пусть  $\sum_{n=2}^{\infty} a_n$  - ряд,  $n$ -й член равен  $a_n = \frac{1}{n(n+2)}$ . Определим индексный вектор  $n$  от 2 до 11 и вычислим члены. Для получения последовательности частичных сумм используем цикл и функцию `sum(a)`. На выходе получаем 10-элементный вектор частичных сумм. Строим слагаемые и частичные суммы для  $2 \leq n \leq 11$  на графике (рис. fig. 3.3).



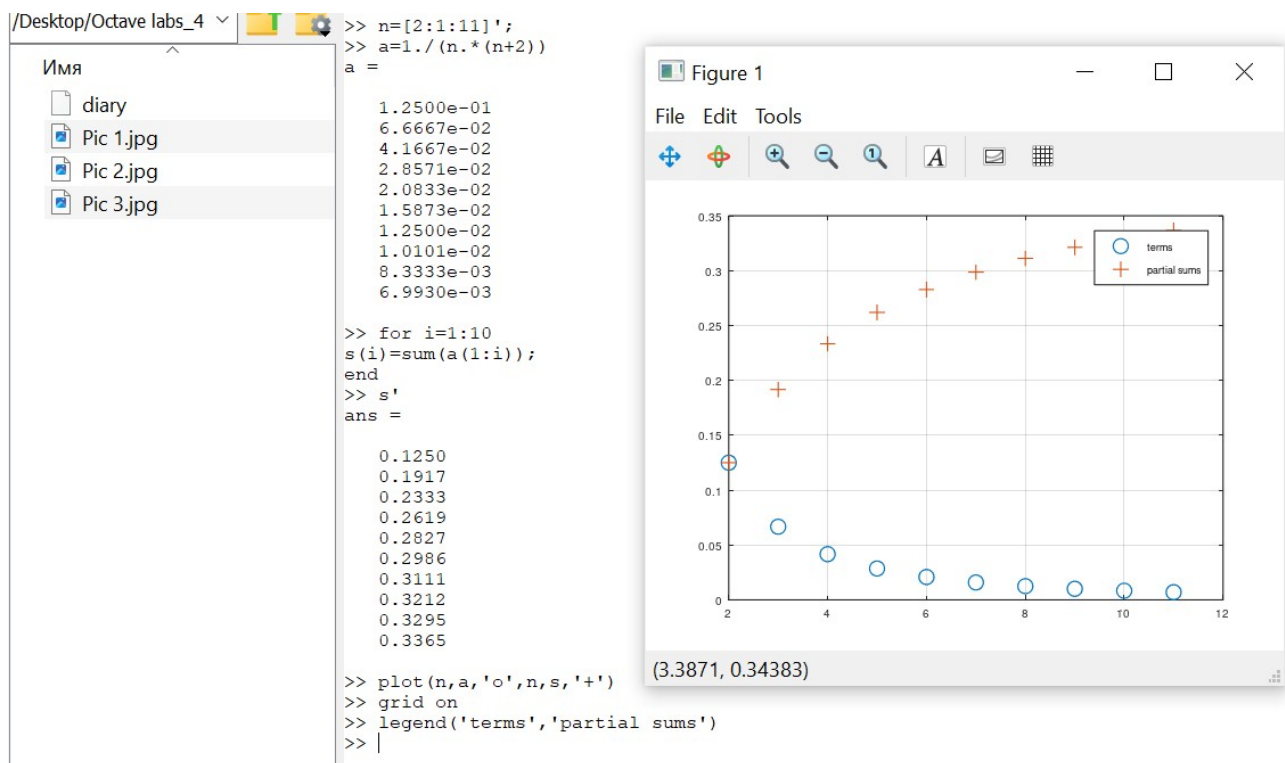


Рис. 3.3: Частичные суммы

Найдем сумму первых 1000 членов гармонического ряда (рис. fig. 3.4):

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

```

>> n=[1:1:1000];
>> a=1./n;
>> sum(a)
ans = 7.4855
>> |

```

Рис. 3.4: Сумма ряда

Вычислим интеграл (рис. fig. 3.5):

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

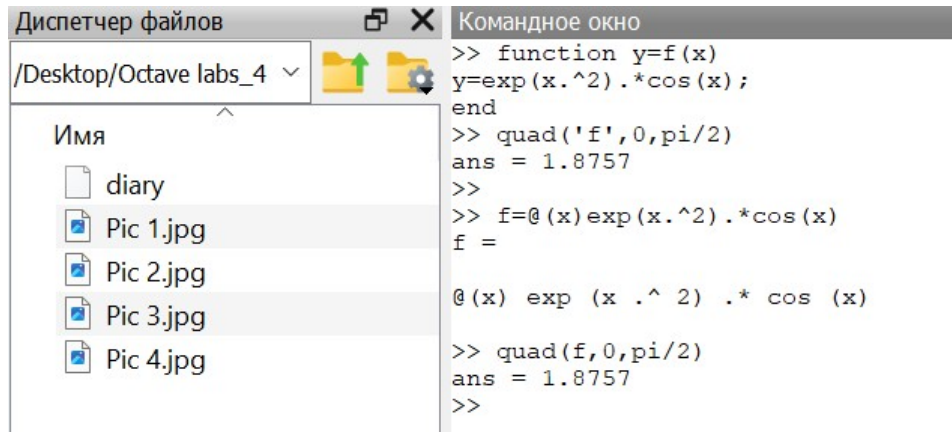


Рис. 3.5: Вычисление интеграла

Пишем код, чтобы вычислить указанный ранее интеграл по правилу средней точки для  $n=100$ : используем цикл, который добавляет значение функции к промежуточной сумме с каждой итерацией, а в конце сумму умножаем на  $\Delta x$  (рис. fig. 3.6) и (рис. fig. 3.7).

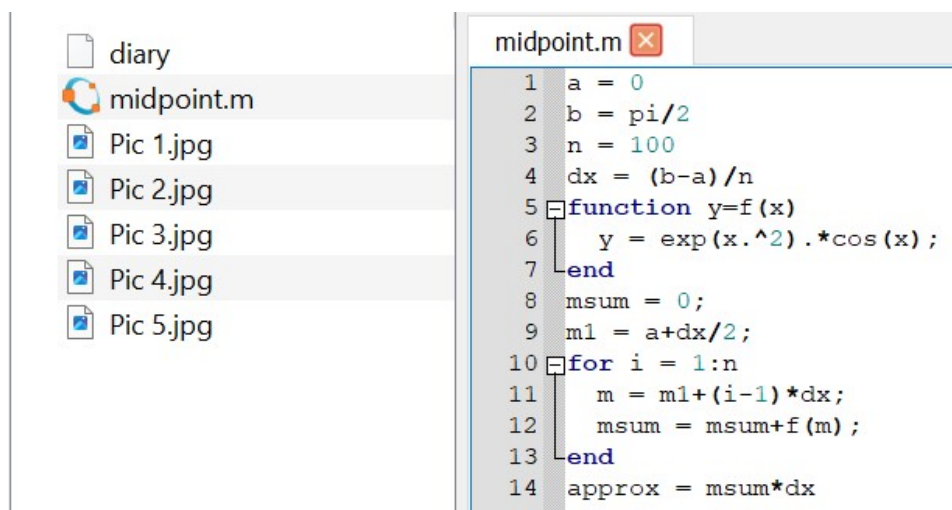


Рис. 3.6: Аппроксимирование суммами

```
>> midpoint
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
>>
```

Рис. 3.7: Аппроксимирование суммами

Напишем векторизованный код. Создадим вектор  $x$ -координат средних точек, далее оцениваем  $f$  по этому вектору средней точки, чтобы получить вектор значений функции. Аппроксимация средней точки - это сумма компонент вектора, умноженная на  $\Delta x$  (рис. fig. 3.8) и (рис. fig. 3.9).

```
1 a = 0
2 b = pi/2
3 n = 100
4 dx = (b-a)/n
5 function y=f(x)
6     y = exp(x.^2).*cos(x);
7 end
8 m = [a+dx/2:dx:b-dx/2];
9 M = f(m);
10 approx = dx*sum(M)
11
```

Рис. 3.8: Аппроксимирование суммами - векторизованный код

```
>> midpoint_v
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
>>
```

Рис. 3.9: Аппроксимирование суммами - векторизованный код

Полученные результаты совпадают с предыдущими. Сравним время выполнения для каждой реализации. Получили, что векторизованный код более эффективен по времени, чем традиционный (рис. fig. 3.10).










Имя	
 diary	>> tic; midpoint; toc
 midpoint_v.m	a = 0
 midpoint.m	b = 1.5708
 Pic 1.jpg	n = 100
 Pic 2.jpg	dx = 0.015708
 Pic 3.jpg	approx = 1.8758
 Pic 4.jpg	Elapsed time is 0.00568295 seconds.
 Pic 5.jpg	>> tic; midpoint_v; toc
 Pic 6.jpg	a = 0
	b = 1.5708
	n = 100
	dx = 0.015708
	approx = 1.8758
	Elapsed time is 0.00294685 seconds.
	>>

Рис. 3.10: Сравнение кодов

## 4 Вывод

В ходе выполнения данной лабораторной работы я изучил в Octave методы расчета пределов, частичных сумм, суммы ряда, а также методы вычисления интегралов и аппроксимирования суммами.