

Лабораторная работа 6

Отчет по лабораторной работе 6

Хитяев Евгений Анатольевич НПИМд-02-21

Содержание

1	Цель работы	4
2	Теоретические сведения	5
3	Задание	6
4	Выполнение лабораторной работы	7
5	Выводы	15

List of Figures

4.1	Промежуточные вычисления для расчета предела	8
4.2	Искомый предел	9
4.3	Частичные суммы	10
4.4	Графическое представление результатов	11
4.5	Сумма ряда	11
4.6	Интегрирование функции	12
4.7	Содержание файла midpoint	12
4.8	Результаты вывода	13
4.9	Содержание файла midpoint_v	13
4.10	Вывод векторизованного кода программы	14
4.11	Сравнение полученных результатов	14

1 Цель работы

Научиться работать в Octave с пределами, последовательностями и рядами, а также научиться писать векторизованный программный код.

2 Теоретические сведения

Вся теоритическая часть по выполнению лабораторной работы была взята из инструкции по лабораторной работе №5 (“Лабораторная работа №6. Описание”) на сайте: <https://esystem.rudn.ru/course/view.php?id=12766>

3 Задание

Выполните работу и задокументируйте процесс выполнения.

4 Выполнение лабораторной работы

1. Пределы. Оценка

Определяем с помощью анонимной функции простую функцию. Создаём индексную переменную, возьмём степени 10, и оценим нашу функцию. Показано на Fig. 1.

```
a: C:\Users\Евгешок
Командное окно
For more information, visit ht
html

Read https://www.octave.org/bu
reports.
For information about changes
'.

>> 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
```

Figure 4.1: Промежуточные вычисления для расчета предела

Получим ответ. На Fig. 2 видно, что предел сходится к значению 2.71828.

```
>> f(n)
ans =

2.000000000000000000
2.593742460100002
2.704813829421529
2.716923932235520
2.718145926824356
2.718268237197528
2.718280469156428
2.718281693980372
2.718281786395798
2.718282030814509
```

Figure 4.2: Искомый предел

2. Частичные суммы

Определим индексный вектор, а затем вычислим члены. После чего введем последовательность частичных сумм, используя цикл. Показано на Fig .3

```

>> format
>> n = [2:1:11]';
>> a = 1 ./ (n .* (n+2))
a =

    1.2500e-01
    6.6667e-02
    4.1667e-02
    2.8571e-02
    2.0833e-02
    1.5873e-02
    1.2500e-02
    1.0101e-02
    8.3333e-03
    6.9930e-03

>> for i = 1:10
s (i) = sum (a(1:i));
end
>> s'
ans =

    0.1250
    0.1917
    0.2333
    0.2619
    0.2827
    0.2986
    0.3111
    0.3212
    0.3295
    0.3365

>> plot (n,a,'o',n,s,'+')
>> grid on
>> legend('terms','partial sums')

```

Figure 4.3: Частичные суммы

Построенные слагаемые и частичные суммы можно увидеть на Fig. 4.

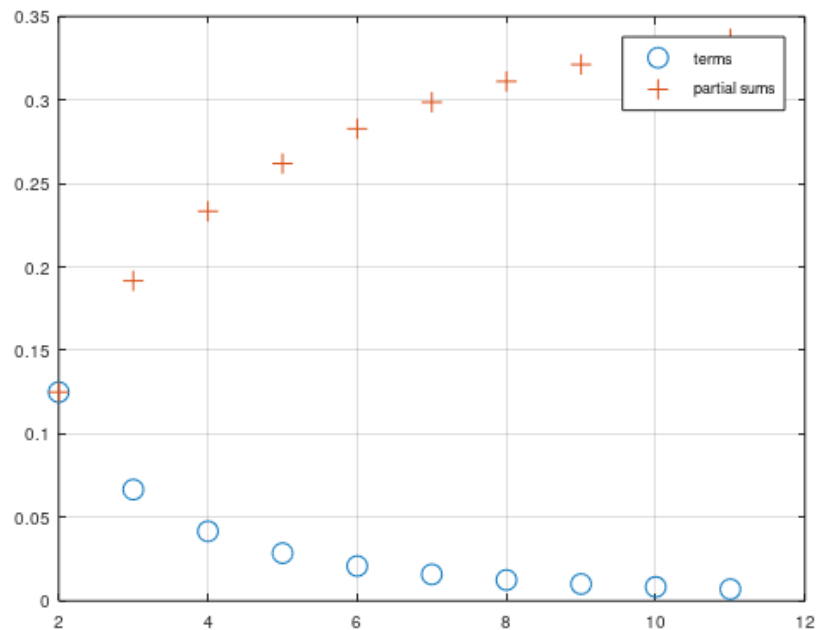


Figure 4.4: Графическое представление результатов

3. Сумма ряда

Найдём сумму первых 1000 членов гармонического ряда $1/n$. Действия показаны на Fig. 5.

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

Figure 4.5: Сумма ряда

4. Вычисление интегралов

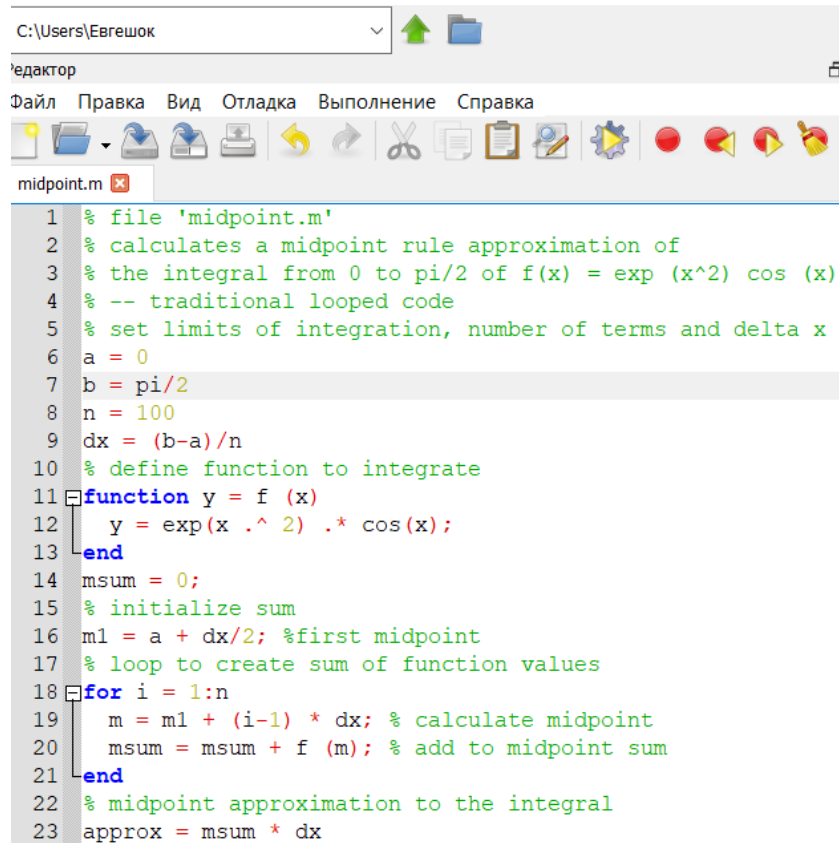
Численно посчитаем интеграл. См. Fig. 6.

```
>> function y = f(x)
y = exp (x .^ 2) .* cos(x);
end
>> quad ('f',0,pi/2)
ans = 1.8757
```

Figure 4.6: Интегрирование функции

5. Аппроксимирование суммами

Напишем скрипт для того, чтобы вычислить интеграл по правилу средней точки. Введём код в текстовый файл и назовём его `midpoint.m`. Скрипт показан на Fig. 7.



```
1 % file 'midpoint.m'
2 % calculates a midpoint rule approximation of
3 % the integral from 0 to pi/2 of f(x) = exp (x^2) cos (x)
4 % -- traditional looped code
5 % set limits of integration, number of terms and delta x
6 a = 0
7 b = pi/2
8 n = 100
9 dx = (b-a)/n
10 % define function to integrate
11 function y = f (x)
12     y = exp(x .^ 2) .* cos(x);
13 end
14 msum = 0;
15 % initialize sum
16 m1 = a + dx/2; %first midpoint
17 % loop to create sum of function values
18 for i = 1:n
19     m = m1 + (i-1) * dx; % calculate midpoint
20     msum = msum + f (m); % add to midpoint sum
21 end
22 % midpoint approximation to the integral
23 approx = msum * dx
```

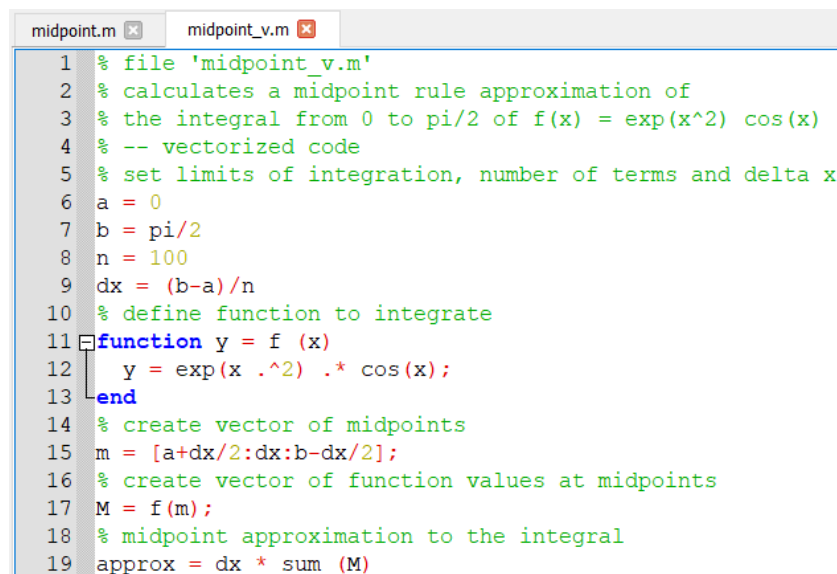
Figure 4.7: Содержание файла `midpoint`

Запустим этот файл в командной строке. Вывод см. на Fig. 8

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

Figure 4.8: Результаты вывода

Теперь напишем векторизованный код, не требующий циклов. Для этого создадим вектор x -координат средних точек. Показано на Fig. 9.



```
midpoint.m x midpoint_v.m x
1 % file 'midpoint_v.m'
2 % calculates a midpoint rule approximation of
3 % the integral from 0 to pi/2 of f(x) = exp(x^2) cos(x)
4 % -- vectorized code
5 % set limits of integration, number of terms and delta x
6 a = 0
7 b = pi/2
8 n = 100
9 dx = (b-a)/n
10 % define function to integrate
11 function y = f (x)
12     y = exp(x.^2) .* cos(x);
13 end
14 % create vector of midpoints
15 m = [a+dx/2:dx:b-dx/2];
16 % create vector of function values at midpoints
17 M = f(m);
18 % midpoint approximation to the integral
19 approx = dx * sum (M)
```

Figure 4.9: Содержание файла midpoint_v

Запустим этот файл в командной строке. Вывод см. на Fig. 10

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

Figure 4.10: Вывод векторизованного кода программы

Запустив оба кода, можно заметить, что ответы совпадают, однако векторизованный код считает быстрее, так как в нём не использованы циклы, которые значительно замедляют работу программы. Сравнение показано на Fig. 11.

```
>> tic; midpoint; toc  
a = 0  
b = 1.5708  
n = 100  
dx = 0.015708  
approx = 1.8758  
Elapsed time is 0.0387039 seconds.  
>> tic; midpoint_v; toc  
a = 0  
b = 1.5708  
n = 100  
dx = 0.015708  
approx = 1.8758  
Elapsed time is 0.010076 seconds.
```

Figure 4.11: Сравнение полученных результатов

5 Выводы

В ходе выполнения лабораторной работы я научился работать в Octave с пределами, последовательностями и рядами, а также научился писать векторизованный программный код. Более того, мне удалось определить, что векторизованный код работает существенно быстрее, чем код с циклами.