

Отчёт по лабораторной работе №6

Пределы, последовательности и ряды

Виктория Михайловна Шутенко, НФИбд-03-19

Содержание

Цель работы	3
Выполнение лабораторной работы	4
Предел	4
Частичные суммы	6
Сумма ряда	8
Численное Интегрирование Вычисление интегралов	9
Аппроксимирование суммами	10
Выводы	13

Цель работы

Приобрести практические навыки работы с пределами, последовательностями и рядами в Octave.

Выполнение лабораторной работы

Предел

1. В 1 пункте нужно было определить функцию. Я использовала приведенный в методичке способ анонимной функции. Так я смогла определить простую функцию.

$$>> f = @(n)(1 + 1./n).^n$$

- где @ - входная переменная
2. Далее я создала индексную переменную, состоящую из целых чисел от 0 до 9. (Рис. 01):

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

3. Синтаксис[0: 1: 9]вектор строки начинается с нуля и увеличивается с шагом от одного до девяти. Я заметила, что использовалась операция транспонирования для того, чтобы результат было легче читать как вектор-столбцы. Далее я взяла степени 10 которые будут входными значениями, а затем оценила их:

- Потом нарисовала точки на графике:

$$>> formatlong$$

$$>> n = 10.^k$$

`>> f(n)`

`>> format`

4. В итоге я получила то, что предел сходится к конечному значению, которое составляет приблизительно 2,718... Подобные методы могут быть использованы для численного исследования последовательностей и рядов (Рис. 01)



```
Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

[octave:1> diary on
[octave:2> f = @(n)(1+1 ./n) .^n
f =

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

[octave:3> k = [0:1:9]'
k =

    0
    1
    2
    3
    4
    5
    6
    7
    8
    9

[octave:4> format long
[octave:5> n = 10 .^ k
n =

    1
   10
  100
 1000
10000
100000
1000000
10000000
100000000
1000000000

[octave:6> f(n)
ans =

 2.000000000000000
 2.593742460100002
 2.704813829421528
 2.716923932235594
 2.718145926824926
 2.718268237192297
 2.718280469095753
 2.718281694132082
 2.718281798347358
 2.718282052011560

[octave:7> format
[octave:8>
```

Рис. 0.1: Метод анонимной функции; индексная переменная, состоящая из целых чисел; взятие степеней 10, которые будут входными

Частичные суммы

1. Далее я работала с заданной в методичке суммой ряда.
2. Для начало я определила индекс вектора n от двух до 11 а затем вычислила его члены (Рис. 02):

```
>> n = [2 : 1 : 11]';
```

```
>> a = 1./(n.* (n + 2))
```

3. Для того чтобы узнать частную сумму нужно написать `sum(a)`. А чтобы получить последовательность частных сумм, то нужно использовать цикл. Я использовала цикл `for` с индексом i от 1 до 10. Для каждого i получила частичную сумму последовательности a_n от первого слагаемого до i слагаемого. На выходе получается 10-й элементный вектор из этих частных сумм.

```
>> for i = 1 : 10
```

```
    s(i) = sum(a(1 : i));
```

```
    end
```

```
>> s'
```

4. Потом я построила слагаемые и частичные суммы для n и получила следующий граф (Рис. 03).

```
>> plot(n, a, 'o', n, s, '+' )
```

```
>> grid on
```

```
>> legend('terms', 'partialsums')
```

```

[octave:8> n = [2:1:11]';
[octave:9> 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

[octave:10> for i = 1:10
[> s(i) = sum (a(1:i));
[> end
[octave:11> s'
ans =

    0.1250
    0.1917
    0.2333
    0.2619
    0.2827
    0.2986
    0.3111
    0.3212
    0.3295
    0.3365

[octave:12> plot(n,a,'o',n,s,'+')
octave:13> FALLBACK (log once): Fallback to SW vertex for line stipple
FALLBACK (log once): Fallback to SW vertex processing, m_disable_code: 2000
FALLBACK (log once): Fallback to SW vertex processing in drawCore, m_disable_code: 2000

[octave:13> grid on
[octave:14> legend('terms','partial sums')
[octave:15>

```

Рис. 0.2: Построение графика суммы ряда

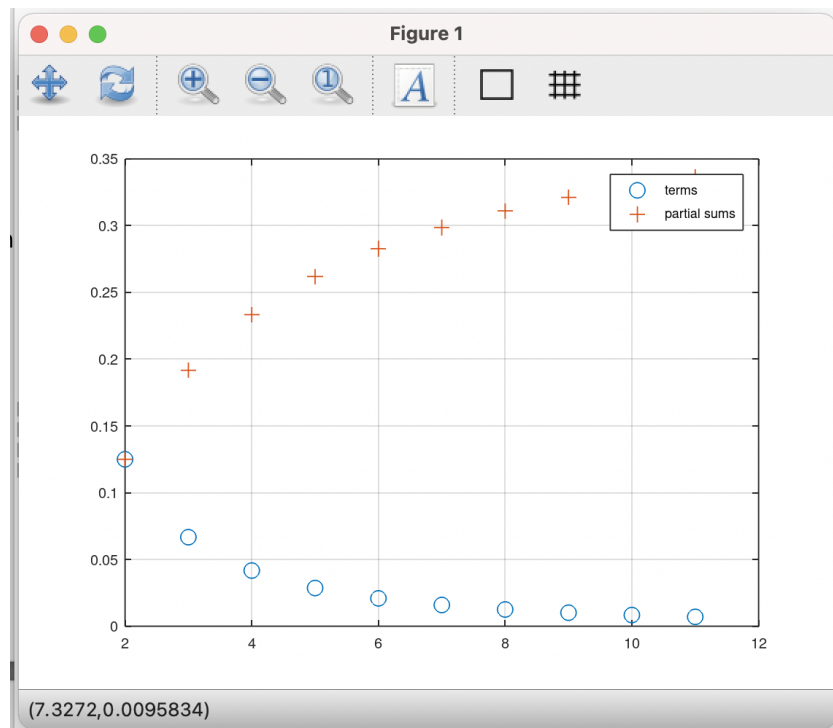


Рис. 0.3: График суммы ряда

Сумма ряда

1. Далее я искала сумму первых 1000 членов гармонического ряда. Для этого сгенерировала члены ряда, как вектор, и взяла их сумму (Рис.04).

```
>> n = [1 : 1 : 1000];
```

```
>> a = 1./n;
```

```
>> sum(a)
```



```
[octave:15> print graph1.png -dpng
]
[octave:16> print('graph1.pdf','-dpdf')
]
[octave:17> n = [1:1:1000];
]
[octave:18> a = 1 ./ n;
]
[octave:19> sum (a)
]
ans = 7.4855
octave:20> ]
```

Рис. 0.4: Генерирование членов ряда, как вектор и взятие их суммы

Численное Интегрирование Вычисление интеграллов

1. Для вычисления интеграллов я использовала команду `quad`.
2. Для начало я определила функцию. Я заметила, что функция $\exp(x)$ используется для e^2 (Рис. 05).

```
>> function y = f(x)
```

```
y = exp(x.^2) .* cos(x);
```

```
>> end
```

```
>> quad('f',0,pi/2)
```

```
[octave:20> function y = f(x)
]
[> y = exp (x .^ 2) .* cos (x);
]
[> end
]
[octave:21> quad ('f',0,pi/2)
]
ans = 1.8757
octave:22> ]
```

Рис. 0.5: Вычисление интеграллов

3. Позднее я разобралась с пунктом использования анонимной функции.(Рис. 06)

$$f = @(x)(\exp(x.^2)).* \cos(x)$$

```
[octave:36> f = @(x)(exp(x.^2)) .* cos(x)
f =

@(x) (exp (x .^ 2)) .* cos (x)

[octave:37> quad(f,0,pi/2)
ans = 1.8757
```

Рис. 0.6: Использование анонимных функций

Аппроксимирование суммами

1. Я написала скрипт для того чтобы вычислить интеграл. По правилу средней точки для n равного 100. Стратегия заключалась в использовании цикла, который добавляет значение функции к промежуточной сумме с каждой итерации. В конце сумма умножается на дельта x
2. Я заранее подготовила два файла: `midpoint` и `midpoint_v` используя VSC(Рис. 08, 010).
3. Запустила `midpoint` (Рис. 07)

```
>> midpoint
```

4. Запустила `midpoint_v` (Рис. 09)

```
>> midpoint_v
```

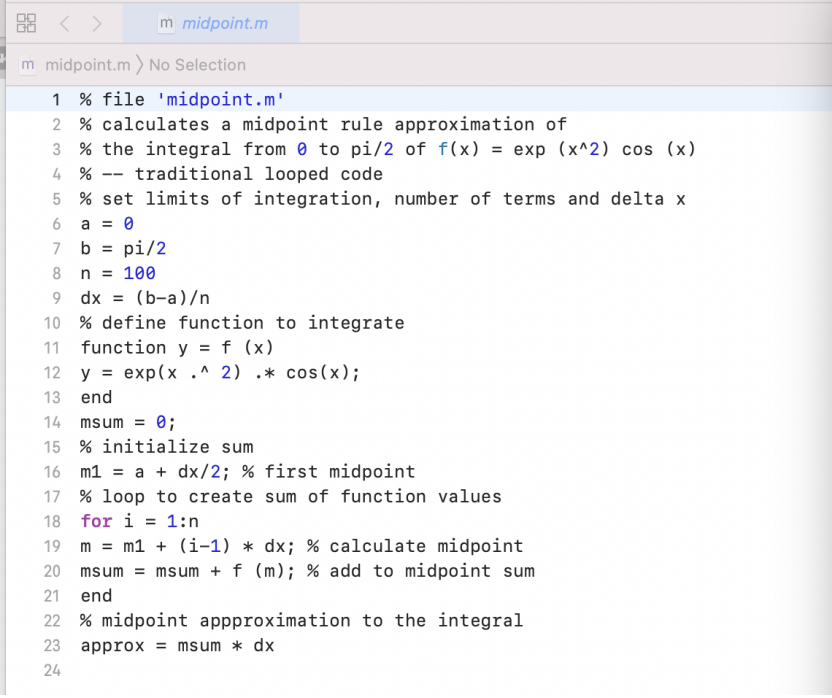
5. Наконец, я сравнила результаты и поняла, что первый способ быстрее (Рис. 09)

```
>> tic; midpoint; toc
```

```
>> tic; midpoint_v; toc
```

```
[octave:22> midpoint
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
```

Рис. 0.7: Запуск midpoint



```
m midpoint.m
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
24
```

Рис. 0.8: Файл midpoint

```

[octave:23> midpoint_v
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
[octave:24> tic; midpoint; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.00528002 seconds.
[octave:25> tic; midpoint_v; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.00101805 seconds.
octave:26> ]

```

Рис. 0.9: Запуск midpoint_v

```

midpoint_v.m
midpoint_v.m > No Selection
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 % -- 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 % 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 = msum * dx
20

```

Рис. 0.10: Файл midpoint_v

Выводы

В ходе выполнения лабораторной работы я приобрела практические навыки работы с пределами, последовательностями и рядами в Octave.