

# Презентация по лабораторной работе №6.

---

Хитяев Евгений Анатольевич, НПИМд-02-21

16 декабря 2021

РУДН, Москва, Россия

## **Лабораторная работа №6.**

---

## Лабораторная работа №6.

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

## Пределы. Оценка

Определяем с помощью анонимной функции простую функцию. Создаём индексную переменную, возьмём степени 10, и оценим нашу функцию (см. скриншот)

<pre>&gt;&gt; diary on &gt;&gt; f = @(n) (1 + 1 ./ n) .^ n f =  @(n) (1 + 1 ./ n) .^ n  &gt;&gt; k = [0:1:9]' &gt;&gt; format long &gt;&gt; n = 10 .^ k n =</pre>	<pre>&gt;&gt; f(n) ans =</pre>
1	2.0000000000000000
10	2.593742460100002
100	2.704813829421529
1000	2.716923932235520
10000	2.718145926824356
100000	2.718268237197528
1000000	2.718280469156428
10000000	2.718281693980372
100000000	2.718281786395798
1000000000	2.718282030814509

**Figure 1:** Пределы. Оценка. Выполнение команд

## Частичные суммы

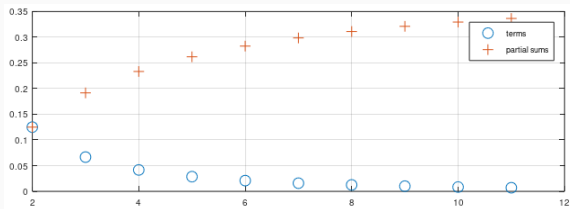
Определим индексный вектор, а затем вычислим члены. После чего введем последовательность частичных сумм, используя цикл. Показано на скриншоте.

```
>> 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 2:** Частичные суммы. Выполнение команд

# Частичные суммы

Построенные слагаемые и частичные суммы можно увидеть на скриншоте ниже.



**Figure 3:** Построение слагаемых и частичных сумм

## Сумма ряда

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

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

Figure 4: Сумма ряда

Численно посчитаем интеграл. Вычисления отображены на скриншоте.

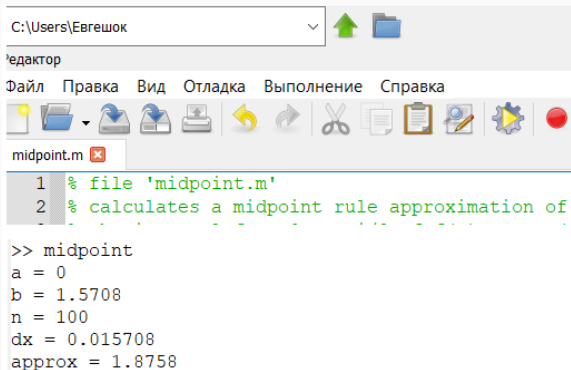
```
>> function y = f(x)
y = exp (x .^ 2) .* cos(x);
end
>> quad ('f',0,pi/2)
ans = 1.8757
```

**Figure 5:** Интегрирование функции



# Аппроксимирование суммами

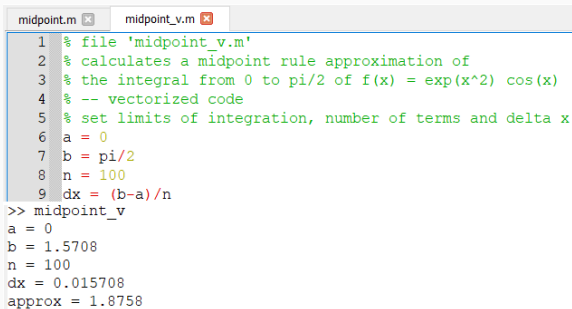
Напишем скрипт для того, чтобы вычислить интеграл по правилу средней точки. Введём код в текстовый файл и назовём его `midpoint.m`. Запустим этот файл в командной строке. Выполненные действия представлены ниже.



**Figure 6:** Вычисление интеграла по правилу средней точки

# Аппроксимирование суммами

Теперь напишем векторизованный код, не требующий циклов. Для этого создадим вектор x-координат средних точек. Запустим этот файл в командной строке. Действия представлены на скриншоте.



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

**Figure 7:** Векторизованный код программы

## Аппроксимирование суммами

Запустим оба кода. Сравнение показано на ниже.

```
>> 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 8:** Сравнение полученных результатов

- 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.