

# Пределы, последовательности и ряды

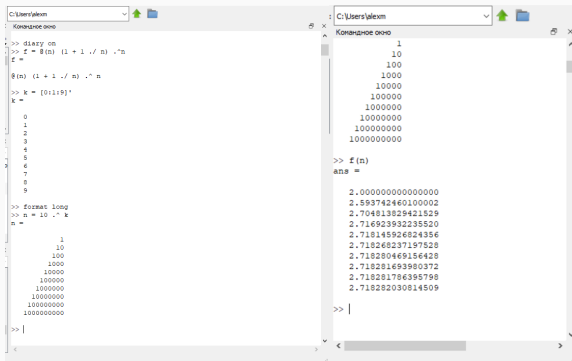
---

Милёхин Александр НПМмд-02-21

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.



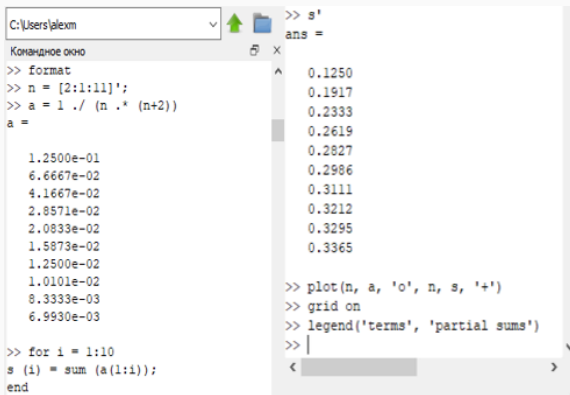
```
>> diary on
>> f = @(n) (1 + 1 ./ n) .* n
f =
    @(n) (1 + 1 ./ n) .* n
>> k = [0:1:9]';
k =
     0
     1
     2
     3
     4
     5
     6
     7
     8
     9
>> format long
>> n = 10 .* k
n =
     0
    10
   100
  1000
 10000
100000
1000000
10000000
100000000
1000000000
>> |

C:\Users\alexm
Колонное окно
1
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
>> f(n)
ans =
2.000000000000000
2.593742460100002
2.704813829421529
2.716923932235520
2.718145926824356
2.718268237197528
2.718280469156428
2.718281693980372
2.718281786395798
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.



The image shows a MATLAB Command Window with the following code and output:

```
C:\Users\alexm
Командное окно
>> 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

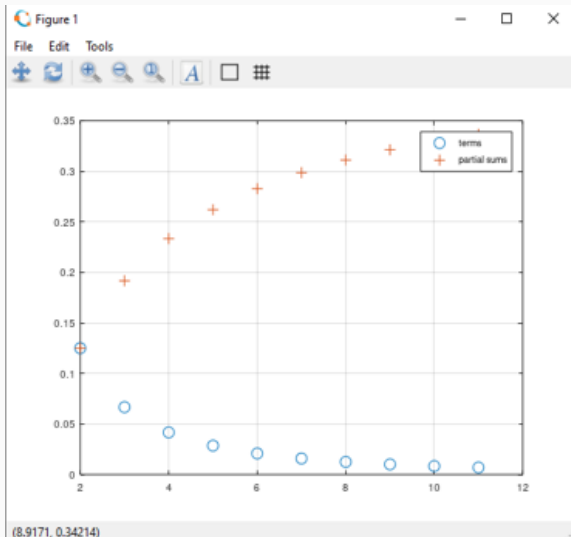
>> s'
ans =
    0.1250
    0.1917
    0.2333
    0.2619
    0.2827
    0.2986
    0.3111
    0.3212
    0.3295
    0.3365

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

**Figure 2:** Частичные суммы. Выполнение команд

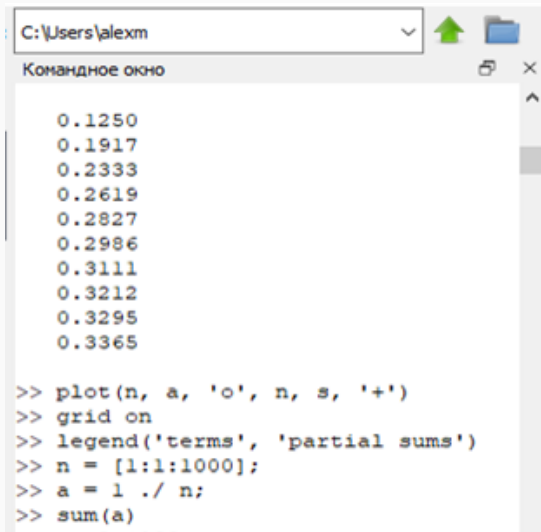
# Частичные суммы

Построенные слагаемые и частичные суммы представлены ниже ниже.



# Сумма ряда

Найдём сумму первых 1000 членов гармонического ряда  $1/n$ .



The screenshot shows a Windows Command Prompt window titled "Командное окно" (Command Window) with the address bar set to "C:\Users\alexm". The window displays the output of MATLAB commands. The first part shows a list of numerical values, which are the partial sums of the harmonic series for the first 10 terms. The second part shows MATLAB commands being executed, including plotting the terms and partial sums, and calculating the sum of the first 1000 terms.

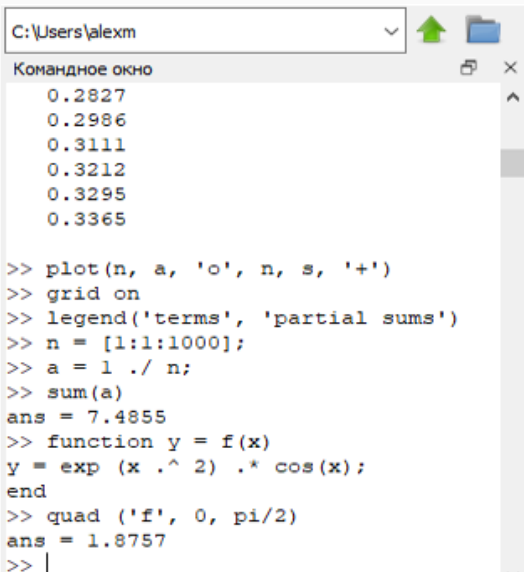
```
C:\Users\alexm
Командное окно

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

# Вычисление интегралов

Численно посчитаем интеграл.



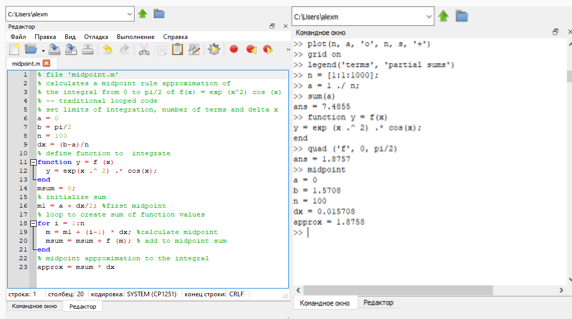
The screenshot shows a MATLAB Command Window with the following content:

```
C:\Users\alexm
Командное окно
0.2827
0.2986
0.3111
0.3212
0.3295
0.3365

>> plot(n, a, 'o', n, s, '+')
>> grid on
>> legend('terms', 'partial sums')
>> n = [1:1:1000];
>> a = 1 ./ n;
>> sum(a)
ans = 7.4855
>> function y = f(x)
y = exp(x.^2) .* cos(x);
end
>> quad('f', 0, pi/2)
ans = 1.8757
>> |
```

# 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.



The image shows a MATLAB environment with two windows. The left window is the 'Редактор' (Editor) showing a script named `midpoint.m`. The script calculates the integral of  $f(x) = \exp(x^2) \cos(x)$  from  $a=0$  to  $b=\pi/2$  using the midpoint rule with  $n=100$  subintervals. The right window is the 'Командное окно' (Command Window) showing the execution of the script, which includes plotting the function and the partial sums, and displaying the numerical results for the integral approximation.

```
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 % initialize sum
15 m1 = a + dx/2; %first midpoint
16 % loop to create sum of function values
17 for i = 1:n
18     m = m1 + (i-1) * dx; %calculate midpoint
19     sum = sum + f(m); % add to midpoint sum
20 end
21 % midpoint approximation to the integral
22 approx = sum * dx
```

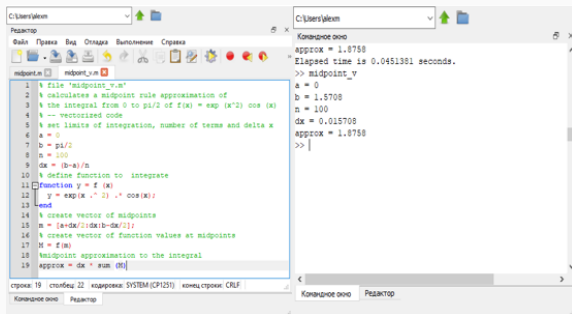
```
>> plot(n, a, 'o', n, b, '+')
>> grid on
>> legend('terms', 'partial sums')
>> n = [1:1000];
>> a = 1 ./ n;
>> sum(a)
ans = 7.4855
>> function y = f(x)
y = exp(x.^2) .* cos(x);
end
>> quad('f', 0, pi/2)
ans = 1.8757
>> midpoint
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
>>
```

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.



The screenshot shows a MATLAB environment with two windows. The 'Редактор' (Editor) window on the left displays a script named 'midpoint\_v.m'. The script is a vectorized implementation of the midpoint rule for integration. It defines a function  $y = f(x) = \exp(x^2) \cos(x)$  and calculates the integral from  $a=0$  to  $b=\pi/2$  using  $n=100$  points. The code uses vectorized operations to create a vector of x-coordinates, evaluate the function at those points, and sum the results to approximate the integral. The 'Командное окно' (Command Window) on the right shows the execution output, which includes the elapsed time (0.0451381 seconds) and the final approximation value of 1.8758.

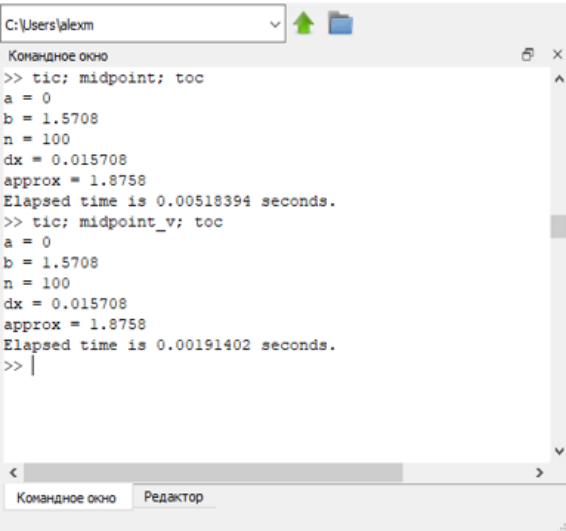
```
1 % file 'midpoint_v.m'
2 % calculates a midpoint rule approximation of
3 % the integral from 0 to pi/2 of f(x) = exp (x^2) cos (x)
4 % -- vectorized code
5 % set limits of integration, number of terms and delta x
6 a = 0
7 b = pi/2
8 n = 100
9 dx = (b-a)/n
10 % define function to integrate
11 function 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)
```

```
approx = 1.8758
Elapsed time is 0.0451381 seconds.
>> 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.



The screenshot shows a MATLAB Command Window titled 'Командное окно' (Command Window) with the path 'C:\Users\alexm'. It displays two blocks of MATLAB code, each preceded by a timing command 'tic' and followed by 'toc'. The first block calculates the midpoint approximation for a function on the interval [0, 1.5708] with 100 subintervals, resulting in an elapsed time of 0.00518394 seconds. The second block performs the same calculation but with a variable 'v' in the function name, resulting in a significantly shorter elapsed time of 0.00191402 seconds. The window includes standard icons for file operations and window management, and a taskbar at the bottom with buttons for 'Командное окно' and 'Редактор'.

```
C:\Users\alexm
Командное окно
>> tic; midpoint; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.00518394 seconds.
>> tic; midpoint_v; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.00191402 seconds.
>> |
```

Я научился работать в Octave с пределами, последовательностями и рядами, а также научился писать векторизованный программный код. Я определил, что векторизованный код работает существенно быстрее, чем код с циклами.

Спасибо за внимание