

Пределы ,последовательности и ряды.

Меньшов Иван Сергеевич

15 декабря, 2021, Москва, Россия

Российский Университет Дружбы Народов

Цели и задачи

Цель лабораторной работы

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

Выполнение лабораторной работы

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

```
>> 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 1: Пределы код 01

Пределы. Оценка

Получим ответ. На следующей фигуре видно, что предел сходится к значению 2.71828.

```
>> f(n)
ans =

2.0000000000000000
2.593742460100002
2.704813829421529
2.716923932235520
2.718145926824356
2.718268237197528
2.718280469156428
2.718281693980372
2.718281786395798
2.718282030814509
```

Figure 2: Пределы код 02

Частичные суммы

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

```
>> format
>> n = [2:1:11]
n =
     2     3     4     5     6     7     8     9    10    11

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

1.2500e-01
4.6467e-02
4.1667e-02
2.6371e-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
0.2533
0.2619
0.2527
0.2086
0.2011
0.2012
0.2025
0.2036

>> plot(n,a,'o','n,s','+')
>> grid on
>> legend('terms','partial sums')
error: 'legend' undefined near line 1, column 1
>> legend('terms','partial sums')
```

Figure 3: Частичные суммы код 01

Частичные суммы

Построенные слагаемые и частичные суммы можно увидеть на следующем рисунке:

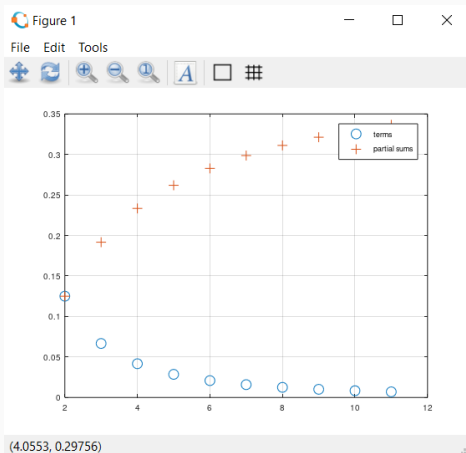


Figure 4: Частичные суммы код 02

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

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

Figure 5: Сумма ряда код 01

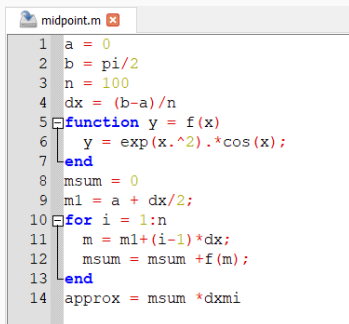
Численно посчитаем интеграл.

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

Figure 6: Вычисление интегралов код 01

Аппроксимирование суммами

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

A screenshot of a MATLAB script editor window titled 'midpoint.m'. The script contains 14 lines of code for numerical integration using the midpoint rule. The code is as follows:

```
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 msum = 0
9 m1 = a + dx/2;
10 for i = 1:n
11     m = m1 + (i-1)*dx;
12     msum = msum + f(m);
13 end
14 approx = msum * dx
```

Figure 7: Аппроксимирование суммами код 01

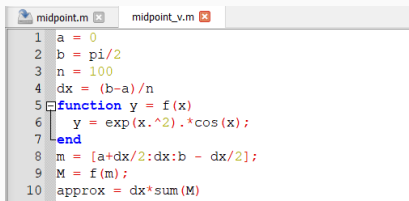
Запустим этот файл в командной строке.

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

Figure 8: Аппроксимирование суммами код 02

Аппроксимирование суммами

Теперь напомним векторизованный код, не требующий циклов. Для этого создадим вектор x-координат средних точек. Введём код в текстовый файл и назовём его `midpoint_v.m`.



```
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)
```

Figure 9: Аппроксимирование суммами код 03

Запустим этот файл в командной строке.

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

Figure 10: Аппроксимирование суммами код 04

Аппроксимирование суммами

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.

```
>> tic ; midpoint ; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
msum = 0
approx = 1.8758
Elapsed time is 0.00515604 seconds.
>> tic ; midpoint_v ; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.00274587 seconds.
```

Figure 11: Аппроксимирование суммами код 05

Выводы

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