

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО

ЛАБОРАТОРНОЙ РАБОТЕ №6

дисциплина: Научное программирование

Студентка: Голос Елизавета Сергеевна

Группа: НПМмд-02-20

Ст. билет № 1032202186

Цель работы

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

Ход работы

Пределы. Оценка

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

```
>> 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  
10000000000
```

Рис.1 Промежуточные вычисления для расчета предела

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

```
>> f(n)
ans =

2.0000000000000000
2.593742460100002
2.704813829421528
2.716923932235594
2.718145926824926
2.718268237192297
2.718280469095753
2.718281694132082
2.718281798347358
2.718282052011560
```

Рис.2 Искомый предел

Частичные суммы

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

```

>> 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 =

    0.1250000
    0.0666667
    0.0416667
    0.0285714
    0.0208333
    0.0158730
    0.0125000
    0.0101010
    0.0083333
    0.0069930

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

    0.12500
    0.19167
    0.23333
    0.26190
    0.28274
    0.29861
    0.31111
    0.32121
    0.32955
    0.33654

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

```

Рис.3 Частичные суммы

Построенные слагаемые и частичные суммы можно увидеть на рисунке 4.

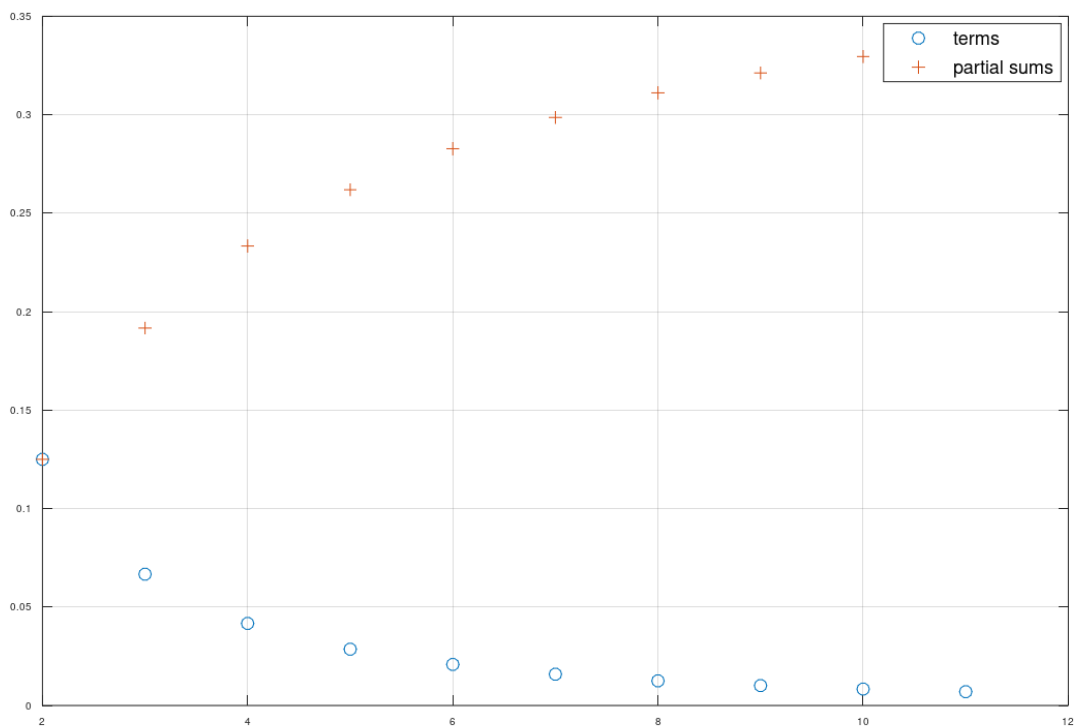


Рис.4 Графическое представление результатов.

Сумма ряда

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

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

Вычисление интегралов

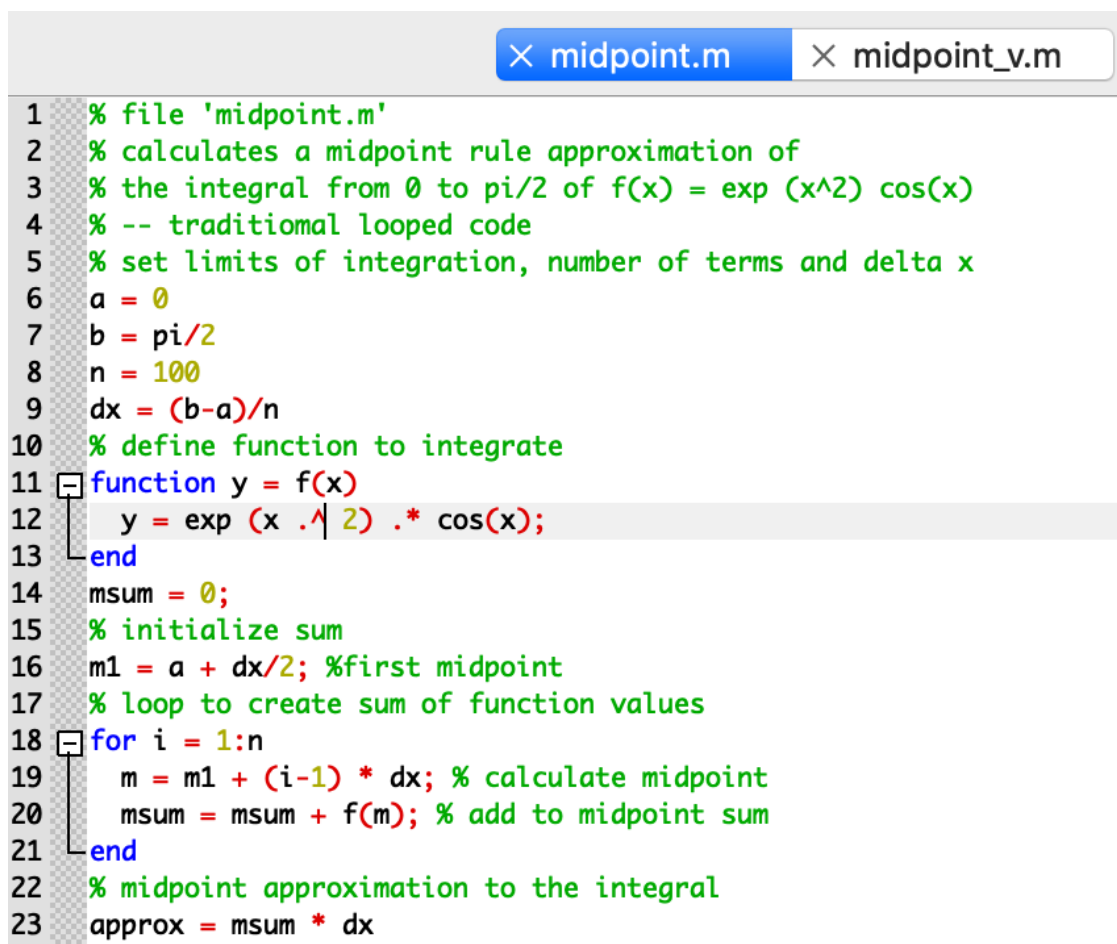
Численно посчитаем интеграл. См. рисунок 6.

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

Рис.6 Интегрирование функции

Аппроксимирование суммами

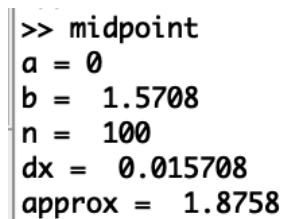
Напишем скрипт для того, чтобы вычислить интеграл по правилу средней точки. Введём код в текстовый файл и назовём его midpoint.m. Скрипт показан на рисунке 7.

A screenshot of a MATLAB editor window. At the top, there are two tabs: 'midpoint.m' (active) and 'midpoint_v.m'. The code in 'midpoint.m' is as follows:

```
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
```

Рис.7 Содержание файла midpoint

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

A screenshot of the MATLAB command window showing the output of the 'midpoint' function. The output is as follows:

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

Рис.8 Результаты вывода

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

Рис.9 Содержание файла midpoint_v

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

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

Рис.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 рисунке 11.

```
..  
>> tic; midpoint; toc  
a = 0  
b = 1.5708  
n = 100  
dx = 0.015708  
approx = 1.8758  
Elapsed time is 0.00766015 seconds.  
>> tic; midpoint_v; toc  
a = 0  
b = 1.5708  
n = 100  
dx = 0.015708  
approx = 1.8758  
Elapsed time is 0.000855923 seconds.
```

Рис.11 Сравнение полученных результатов

Вывод

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