

Лабораторная работа №6

Научное программирование

Николаев Дмитрий Иванович, НПМмд-02-24

5 октября 2024

Российский университет дружбы народов имени Патриса Лумумбы, Москва, Россия

Прагматика выполнения

- Повышение навыков владения Octave;
- Повышение навыков владения Julia;
- Применение полученных знаний на практике в дальнейшем.

Цели

- Исследование пределов последовательностей, суммирования рядов, численного интегрирования и аппроксимации интегралов с использованием методов программирования на языках **Octave** и **Julia**.

Задачи

1. Вычисление пределов последовательностей.
2. Нахождение частичных сумм рядов.
3. Численное интегрирование функций.
4. Аппроксимирование интегралов методом средней точки.
5. Сравнение производительности традиционного и векторизованного кода.

Выполнение работы

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

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

>> k = [0:1:9]';
>> n = 10 .^ k;
>> f(n)
ans =

    2.0000
    2.5937
    2.7048
    2.7169
    2.7181
    2.7183
    2.7183
    2.7183
    2.7183
    2.7183

>> format long
>> f(n)
ans =

    2.000000000000000
    2.593742460100002
    2.704813829421529
    2.716923932235520
    2.718145926824356
    2.718268237197528
    2.718280469156428
    2.718281693980372
    2.718281786395798
    2.718282030814509
```

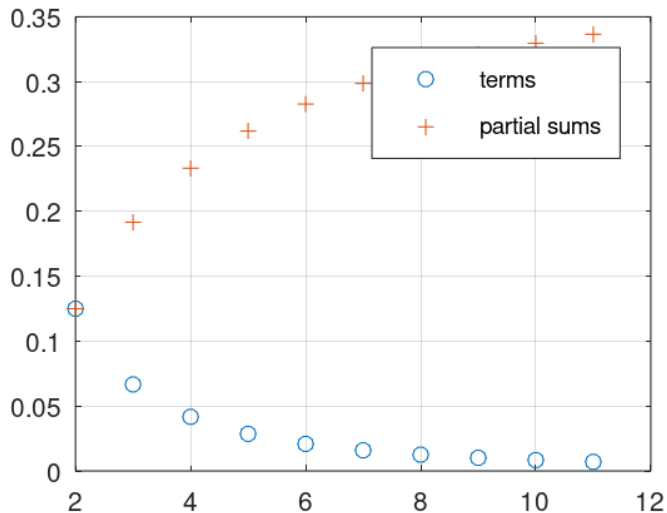
```
>> n = [2:1:11]';  
>> a = 1./ (n .* (n+2))  
a =  
  
1.2500000000000000e-01  
6.666666666666667e-02  
4.166666666666666e-02  
2.857142857142857e-02  
2.083333333333333e-02  
1.587301587301587e-02  
1.2500000000000000e-02  
1.010101010101010e-02  
8.333333333333333e-03  
6.993006993006993e-03  
  
>> format  
>> 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')
```

Остатки. Графики элементов последовательности и её частичные суммы

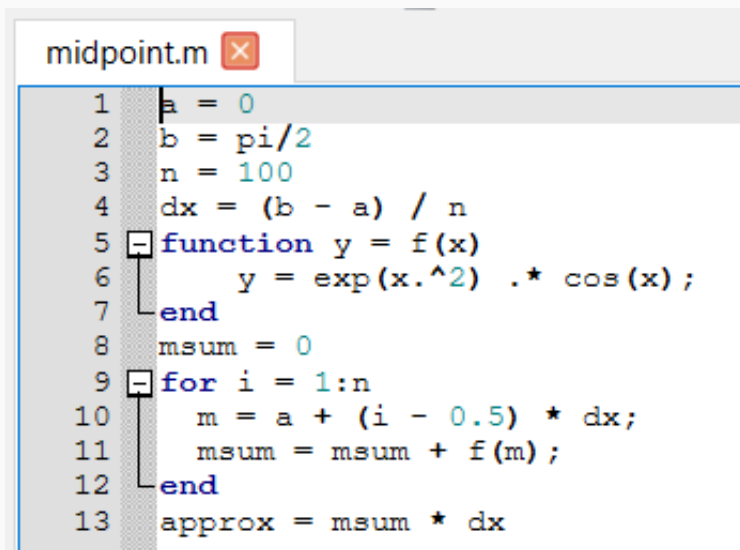


```
< >> n = [1:1:1000];  
>> a = 1 ./ n;  
>> sum(a)  
ans = 7.4855  
>> |
```

Рис. 5: Сумма первых 1000 членов гармонического ряда на Octave

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

Рис. 6: Вычисление интеграла с помощью встроенной функции на Octave



The image shows a screenshot of an Octave script editor window titled "midpoint.m". The script contains the following code:

```
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 for i = 1:n
10     m = a + (i - 0.5) * dx;
11     msum = msum + f(m);
12 end
13 approx = msum * dx
```

The code defines a function `f(x) = exp(x.^2) .* cos(x)` and uses a `for` loop to approximate the integral of `f(x)` from `a = 0` to `b = pi/2` using the midpoint rule with `n = 100` subintervals. The result is stored in `approx`.

```
>> midpoint
```

```
a = 0
```

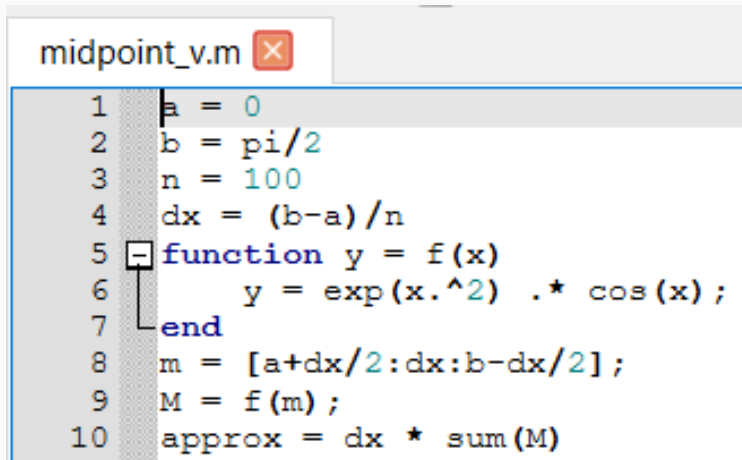
```
b = 1.5708
```

```
n = 100
```

```
dx = 0.015708
```

```
msum = 0
```

```
approx = 1.8758
```

```
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)
```

Рис. 9: Программа вычисления интеграла с помощью векторизованных операций на Octave

```
>> midpoint_v
```

```
a = 0
```

```
b = 1.5708
```

```
n = 100
```

```
dx = 0.015708
```

```
approx = 1.8758
```

Octave. Сравнение времени вычисления интеграла с помощью циклов и векторизованных операций

```
approx = 1.8758
>> tic; midpoint; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
msum = 0
approx = 1.8758
Elapsed time is 0.00432611 seconds.
>> tic; midpoint_v; toc
a = 0
b = 1.5708
n = 100
dx = 0.015708
approx = 1.8758
Elapsed time is 0.00213408 seconds
```

Julia. Вычисление предела, членов ряда, частичных сумм и интегралов

```
1 using QuadGK
2 using BenchmarkTools
3 using Plots
4 # Вычисление предела
5 f(n) = (1 .+ 1 ./ n) .^ n
6 k = 0:9
7 n = 10 .^ k
8 println("Последовательное вычисление предела:\n", f(n))
9
10 # Вычисление частичных сумм
11 n = 2:11
12 a = 1 ./ (n .* (n .+ 2))
13 s = [sum(a[1:i]) for i in 1:length(a)]
14 println("Массив из частичных сумм:\n", s)
15
16 fig1 = plot(n, a, label = "Элементы ряда", color=:red, title = "Сумма ряда")
17 plot!(n, s, label = "Частичные суммы", color=:blue)
18 savefig(fig1, "fig2.png")
19
20 # Сумма первых 1000 членов гармонического ряда
21 n = 1:1000
22 a = 1 ./ n
23 println("Тысячное гармоническое число = ", sum(a))
24
25 # Численное интегрирование
26 f(x) = exp(x^2) * cos(x)
27 println("Численное вычисление интеграла с помощью пакета QuadGK: ", quadgk(f, 0, pi/2))
28
```

Рис. 12: Вычисление предела, членов ряда, частичных сумм и интегралов на Julia

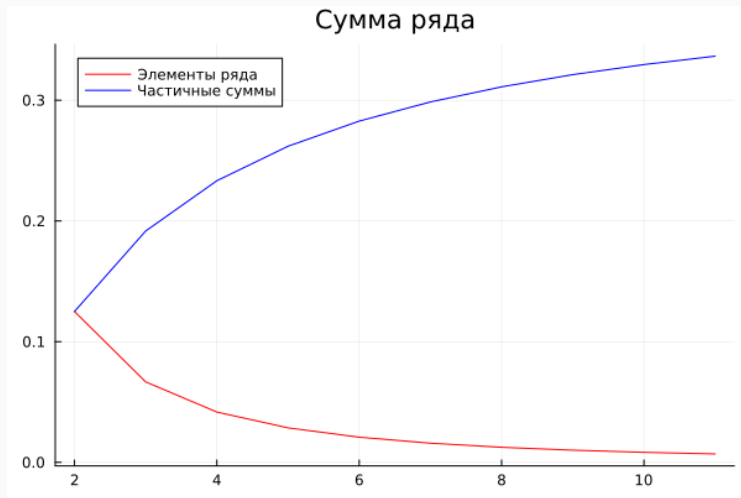


Рис. 13: Графики элементов последовательности и её частичные суммы на Julia

Julia. Вычисления интеграла с помощью циклов и векторизованных операций

```
29 function Midpoint(a::Float64 = 0.0, b::Float64 = pi/2, n::Int = 100, g::Function = f)::Float64
30     dx = (b - a) / n
31     msum = 0
32     for i in 1:n
33         m = a + (i - 0.5) * dx
34         msum += g(m)
35     end
36     return msum * dx
37 end
38
39 function Midpoint_v(a::Float64 = 0.0, b::Float64 = pi/2, n::Int = 100, g::Function = f)::Float64
40     dx = (b - a) / n
41     index = 1:n
42     m = a .+ (index .- 0.5)*dx
43     return sum(f.(m)) * dx
44 end
45 a, b, n = 0, pi/2, 100
46 dx = (b - a) / n
47 println("Сравнение численного интегрирования циклами и векторизованными операциями:")
48 println("a = $a")
49 println("b = $b")
50 println("n = $n")
51 println("dx = $(dx)")
52
53 println("Результат при интегрировании циклами: ", Midpoint())
54 @btime Midpoint()
55
56 println("Результат при интегрировании векторизованными операциями: ", Midpoint_v())
57 @btime Midpoint_v()
```

Рис. 14: Программа вычисления интеграла с помощью циклов и векторизованных операций на

Julia. Сравнение времени вычисления интеграла с помощью циклов и векторизованных операций

```
PS C:\Users\User\Documents\work\study\2024-2025\Научное программирование\sciprog\labs\lab06\report\report> julia .\lab6.jl
Последовательное вычисление предела:
[2.0, 2.5937424601000023, 2.7048138294215285, 2.7169239322355936, 2.7181459268249255, 2.7182682371922975, 2.7182804690957534, 2.7182816941320813, 2.7182817983473577, 2.7182820520115603]
Массив из частичных сумм:
[0.125, 0.19166666666666665, 0.23333333333333333, 0.26190476190476186, 0.2827380952380952, 0.29861111111111105, 0.31111111111111106, 0.32121212121212117, 0.32954545454545453, 0.3365384615384615]
Тысячное гармоническое число = 7.485470860550343
Численное вычисление интеграла с помощью пакета Quadgk: (1.8756650114633908, 2.157818368431208e-11)
Сравнение численного интегрирования циклами и векторизованными операциями:
a = 0
b = 1.5707963267948966
n = 100
dx = 0.015707963267948967
Результат при интегрировании циклами: 1.8757862104628082
1.240 μs (0 allocations: 0 bytes)
Результат при интегрировании векторизованными операциями: 1.8757862104628082
1.200 μs (1 allocation: 896 bytes)
PS C:\Users\User\Documents\work\study\2024-2025\Научное программирование\sciprog\labs\lab06\report\report>
```

Рис. 15: Сравнение времени вычисления интеграла с помощью циклов и векторизованных операций на Julia

Результаты

По результатам работы, я реализовал вычисление пределов последовательностей, суммирование рядов, численное интегрирование и аппроксимацию интегралов с использованием циклов и векторизированных операций на языках Octave и Julia.