Лабораторная работа №6

Научное программирование

Николаев Дмитрий Иванович, НПМмд-02-24

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Цель данной лабораторной работы заключается в исследовании пределов последовательностей, суммирования рядов, численного интегрирования и аппроксимации интегралов с использованием методов программирования на языках **Octave** и **Julia**.

2 Теоретическое введение

- **Пределы последовательностей**: Предел последовательности — это значение, к которому стремятся элементы последовательности при стремлении индекса к бесконечности. Например, известный предел

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e.$$

- **Ряды и частичные суммы**: Ряд — это бесконечная сумма элементов последовательности, частичная сумма — это сумма конечного количества членов ряда.

- **Численное интегрирование**: Методы численного интегрирования включают различные техники для аппроксимации значений интегралов, такие как правило средней точки, трапеций и правило Симпсона.

Методы, используемые в данной лабораторной работе, основываются на численных алгоритмах, которые широко применяются для решения задач, где аналитические методы являются сложными или невозможными.

3 Выполнение лабораторной работы

Следуя указаниям из [1], выполним лабораторную работу на Octave и Julia.

3.1 Octave

3.1.1 Предел последовательности

Вычислим предел последовательности

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

для различных значений n ([3.1]).

```
>> 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

>> |
```

Рис. 3.1: Предел последовательности на Octave

3.1.2 Частичные суммы ряда

Рассмотрим ряд

$$a_n = \frac{1}{n(n+2)}$$

и найдём его элементы ([3.2]), частичные суммы ([3.3]) и построим их графики ([3.4]).

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

   1.250000000000000e-01
   6.666666666666667e-02
   4.166666666666666e-02
   2.857142857142857e-02
   2.083333333333333e-02
   1.587301587301587e-02
   1.250000000000000e-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
```

Рис. 3.2: Элементы последовательности на Octave

```
>> 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')
```

Рис. 3.3: Частичные суммы на Octave

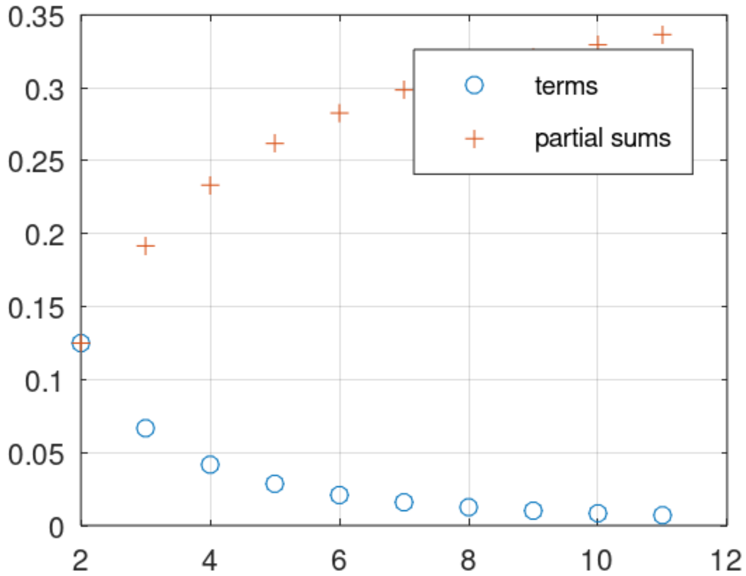


Рис. 3.4: Графики элементов последовательности и её частичные суммы на Octave

3.1.3 Гармонический ряд

Найдем сумму первых 1000 членов гармонического ряда ([3.5]).

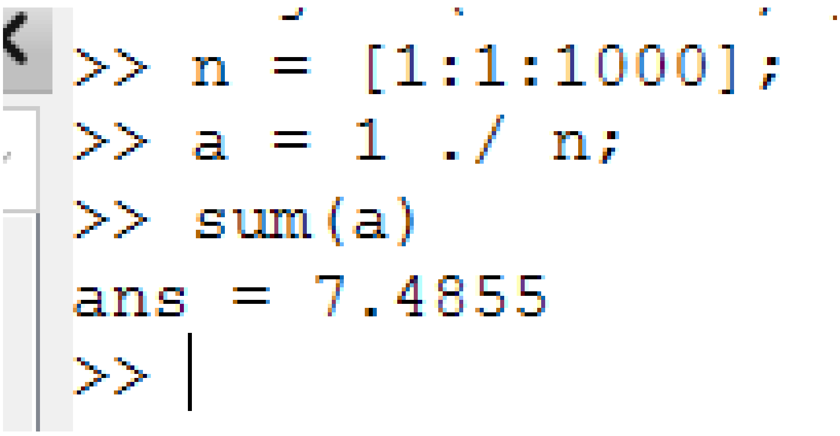


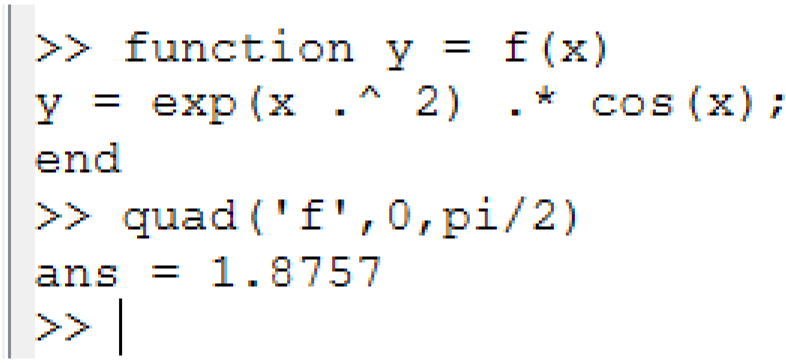
Рис. 3.5: Сумма первых 1000 членов гармонического ряда на Octave

3.1.4 Численное интегрирование

Вычислим интеграл

$$\int_0^{\frac{\pi}{2}} e^{x^2} \cos(x)\, dx$$

с помощью встроенной функции quad ([3.6]).



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

Рис. 3.6: Вычисление интеграла с помощью встроенной функции на Octave

3.1.5 Аппроксимирование методом средней точки

Вычислим интеграл из предыдущего подраздела по правилу средней точки для $n = 100$ ([3.7,3.8]).

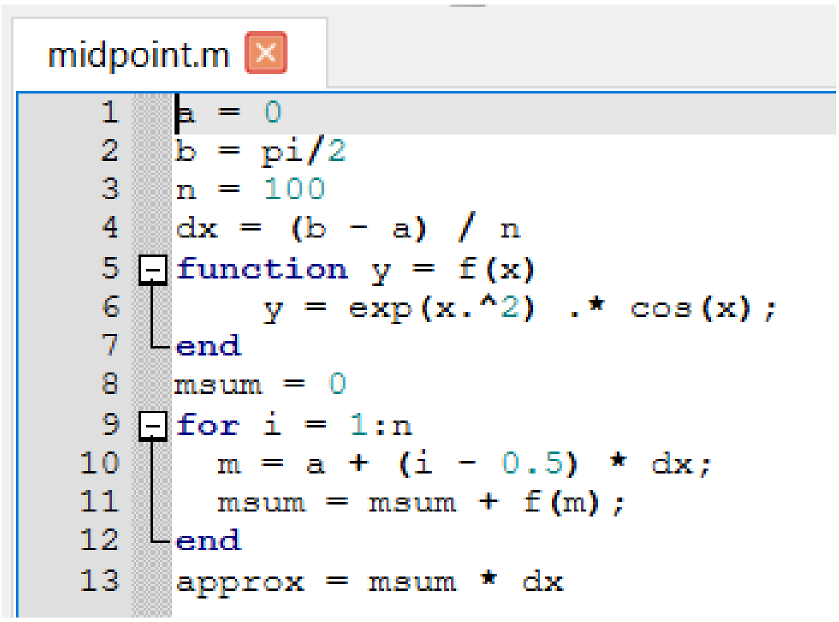


Рис. 3.7: Программа вычисления интеграла с помощью циклов на Octave

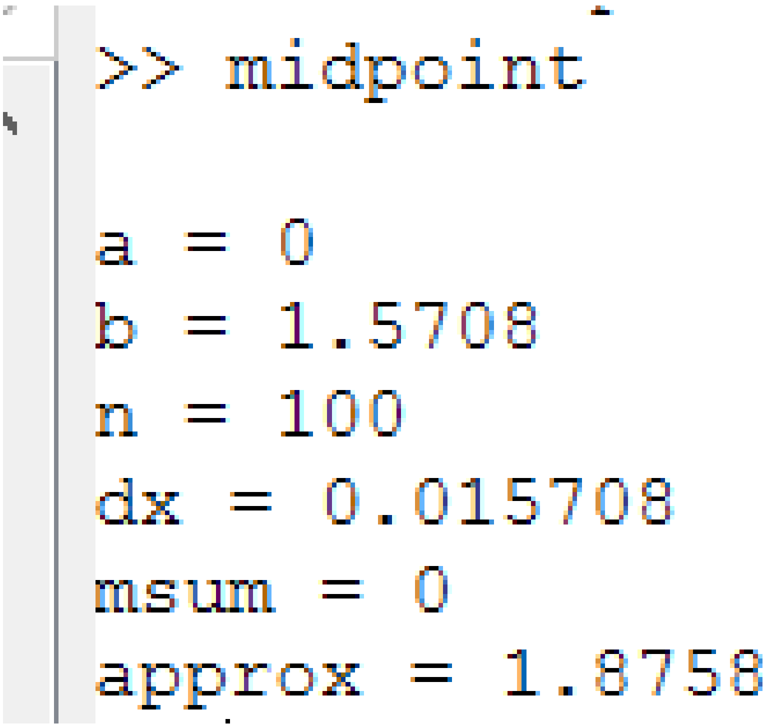


Рис. 3.8: Результат вычисления интеграла с помощью циклов на Octave

3.1.6 Векторизованное вычисление методом средней точки

Теперь вычислим интеграл по правилу средней точки с помощью векторизированных операций для $n = 100$ ([3.9,3.10]).

midpoint_v.m

```
a = 0
b = pi/2
n = 100
dx = (b-a)/n
function y = f(x)
      y = exp(x.^2) .* cos(x);
end
m = [a+dx/2:dx:b-dx/2];
M = f(m);
approx = dx * sum(M)
```

Рис. 3.9: Программа вычисления интеграла с помощью векторизированных операций на Octave

```
>> midpoint_v

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

Рис. 3.10: Результат вычисления интеграла с помощью векторизированных операций на Octave

3.1.7 Сравнение времени выполнения

Сравним время выполнения традиционного и векторизованного кода ([3.11]). Как видим код на основе векторизированных операций выполняется примерно в 2 раза быстрее.

```
>> 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.
>>
```

Рис. 3.11: Сравнение времени вычисления интеграла с помощью циклов и векторизированных операций на Octave

3.2 Julia

Реализуем вычисление предела, членов ряда, частичных сумм и вычисление интеграла с помощью пакета `QuadGK` ([3.12]). После чего построим график членов ряда и его частичных сумм ([3.13]). Далее реализуем вычисление интеграла методом средней точки с помощью циклов и векторизированных операций ([3.14]), в результате мы не наблюдаем существенного ускорения во времени вычисления интеграла ([3.15]).

```julia
using QuadGK
using BenchmarkTools
using Plots
# Вычисление предела
f(n) = (1 .+ 1 ./ n) .^ n
k = 0:9
n = 10 .^ k
println("Последовательное вычисление предела:\n", f(n))

# Вычисление частичных сумм
n = 2:11
a = 1 ./ (n .* (n .+ 2))
s = [sum(a[1:i]) for i in 1:length(a)]
println("Массив из частичных сумм:\n", s)

fig1 = plot(n, a, label = "Элементы ряда", color=:red, title = "Сумма ряда")
plot!(n, s, label = "Частичные суммы", color=:blue)
savefig(fig1, "fig2.png")

# Сумма первых 1000 членов гармонического ряда
n = 1:1000
a = 1 ./ n
println("Тысячное гармоническое число = ", sum(a))

# Численное интегрирование
f(x) = exp(x^2) * cos(x)
println("Численное вычисление интеграла с помощью пакета QuadGK: ", quadgk(f, 0, pi/2))
```

Рис. 3.12: Вычисление предела, членов ряда, частичных сумм и интегралов на Julia

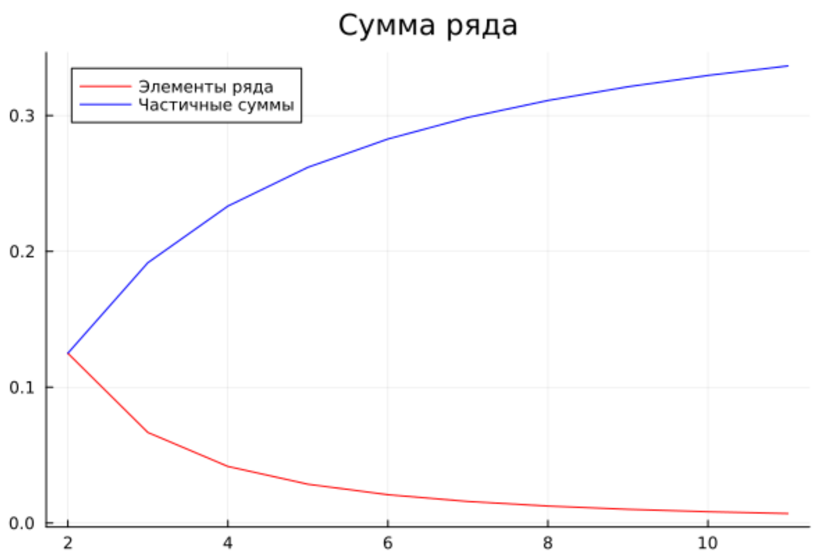
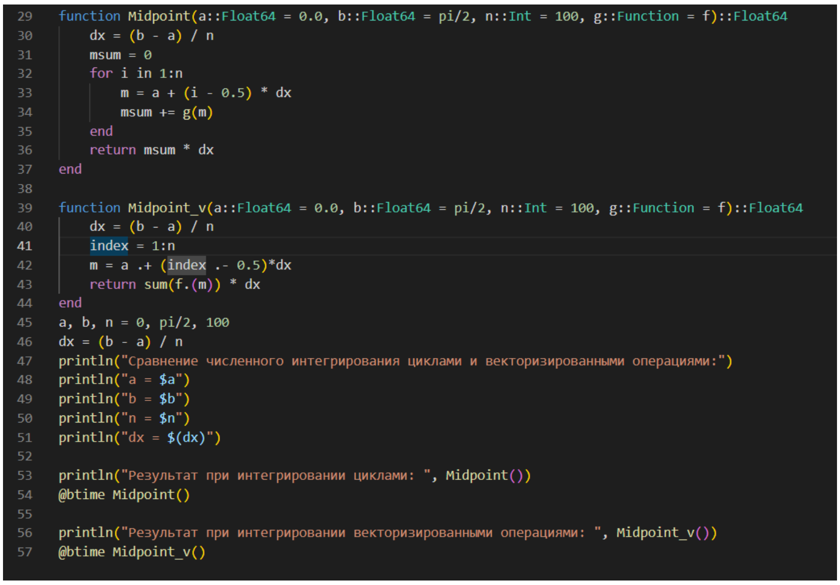


Рис. 3.13: Графики элементов последовательности и её частичные суммы на Julia



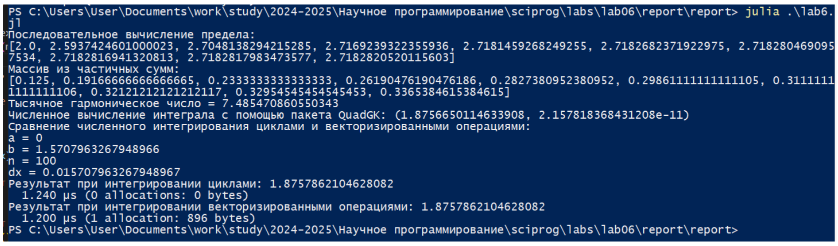
```julia
function Midpoint(a::Float64 = 0.0, b::Float64 = pi/2, n::Int = 100, g::Function = f)::Float64
    dx = (b - a) / n
    msum = 0
    for i in 1:n
        m = a + (i - 0.5) * dx
        msum += g(m)
    end
    return msum * dx
end

function Midpoint_v(a::Float64 = 0.0, b::Float64 = pi/2, n::Int = 100, g::Function = f)::Float64
    dx = (b - a) / n
    index = 1:n
    m = a .+ (index .- 0.5)*dx
    return sum(f.(m)) * dx
end
a, b, n = 0, pi/2, 100
dx = (b - a) / n
println("Сравнение численного интегрирования циклами и векторизированными операциями:")
println("a = $a")
println("b = $b")
println("n = $n")
println("dx = $(dx)")

println("Результат при интегрировании циклами: ", Midpoint())
@btime Midpoint()

println("Результат при интегрировании векторизированными операциями: ", Midpoint_v())
@btime Midpoint_v()
```

Рис. 3.14: Программа вычисления интеграла с помощью циклов и векторизированных операций на Julia



```
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, 2.718268237192297 5, 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 векторизированными 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 векторизированными 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. 3.15: Сравнение времени вычисления интеграла с помощью циклов и векторизированных операций на Julia

4 Выводы

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

Список литературы

1. Кулябов Д. С. Лабораторная работа №6. Пределы, последовательности и ряды [Электронный ресурс]. RUDN, 2024. URL: https://esystem.rudn.ru/pluginfile.php/2372908/mod_resource/content/2/README.pdf.