Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 6

Название: Вычисления с заданной точностью в Ruby

Дисциплина: Языки Интернет-программирования

Студент ИУ6-33Б _____ И.А. Нуруллаев
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ _____
(Подпись, дата) (И.О. Фамилия)

Москва, 2022

Часть 1

Решить задачу, организовав итерационный цикл. Вычислить сумму ряда $S = \sum_{k=1}^{\infty} \frac{1}{k(k+1)(k+2)(k+3)}$ с точностью $\xi = 10^{-2}, 10^{-3}$. Точное значение: $\frac{1}{3 \cdot 3!}$. Определить, как изменяется число итераций при изменении точности.

Решение:

main.rb

```ruby
def calc_sum(accuracy)
  sum = 0.0
  step = 0.0
  k = 1

  loop do
    step = 1.0 / (k * (k + 1) * (k + 2) * (k + 3))
    sum += step
    k += 1
    break if step < accuracy
  end

  [sum, k]
end

```

user.rb

```ruby
require_relative 'main'

result1 = calc_sum(0.01)
result2 = calc_sum(0.001)

puts("Вычисление с точностью 0.01 дало результат #{result1[0]}
, потребовалось итераций - #{result1[1]}")
puts("Вычисление с точностью 0.001 дало результат #{result2[0]}
, потребовалось итераций - #{result2[1]}")

```

test.rb

```ruby
require 'minitest/autorun'
require_relative 'main'

# Sum tester
class Test < Minitest::Test
  def test_first
    result = calc_sum 0.01
    assert_in_delta 1.0 / (3 * ((1..3).inject(:*) || 1)), result[0], 0.01, 'Неправильный результат'
  end

  def test_second
    result = calc_sum 0.001
    assert_in_delta 1.0 / (3 * ((1..3).inject(:*) || 1)), result[0], 0.001, 'Неправильный результат'
  end
end

```

Результат выполнения программ:

```
PS C:\Users\might\Desktop\WPL_bmstu\Lab6\Part 1> ruby .\user.rb
Вычисление с точностью 0.01 дало результат 0.0499999999999999996, потребовалось итераций - 3
Вычисление с точностью 0.001 дало результат 0.05456349206349206, потребовалось итераций - 6
PS C:\Users\might\Desktop\WPL_bmstu\Lab6\Part 1> ruby .\test.rb
Run options: --seed 37755

# Running:

..

Finished in 0.014950s, 133.7828 runs/s, 133.7828 assertions/s.
2 runs, 2 assertions, 0 failures, 0 errors, 0 skips
```

Часть 2

Решить предыдущее задание с помощью Enumerable или Enumerator

Решение:

main.rb

```ruby
def calc_sum(accuracy)
  series(accuracy).take_while { |prev_step| prev_step.abs > accuracy }.last[2]
end

def series(accuracy)
  Enumerator.new do |yielder|
    sum = 0.0
    step = 0.0
    prev_step = 2 * accuracy
    k = 0

    loop do
      k += 1
      step = 1.0 / (k * (k + 1) * (k + 2) * (k + 3))
      sum += step
      yielder.yield prev_step, step, sum
      prev_step = step
    end
  end
end

```

user.rb

```ruby
require_relative 'main'

puts("Вычисление с точностью 0.01 дало результат #{calc_sum(0.01)}")
puts("Вычисление с точностью 0.001 дало результат #{calc_sum(0.001)}")

```

test.rb

```ruby
require 'minitest/autorun'
require_relative 'main'

# Sum tester
class Test < Minitest::Test
  def test_first
    result = calc_sum 0.01
    assert_in_delta 1.0 / (3 * ((1..3).inject(:*) || 1)), result, 0.01, 'Неправильный результат'
  end

  def test_second
    result = calc_sum 0.001
    assert_in_delta 1.0 / (3 * ((1..3).inject(:*) || 1)), result, 0.001, 'Неправильный результат'
  end
end

```

Результат выполнения программ:

```
PS C:\Users\might\Desktop\WPL_bmstu\Lab6\Part 2> ruby .\user.rb
Вычисление с точностью 0.01 дало результат 0.0499999999999999996
Вычисление с точностью 0.001 дало результат 0.054563492063492063
PS C:\Users\might\Desktop\WPL_bmstu\Lab6\Part 2> ruby .\test.rb
Run options: --seed 57044

# Running:

..

Finished in 0.011318s, 176.7019 runs/s, 176.7019 assertions/s.
2 runs, 2 assertions, 0 failures, 0 errors, 0 skips
```

Часть 3

Составить метод trap для вычисления определенного интеграла по формуле трапеций
$\int_a^b f(x)\,dx \approx \frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)$, где $f(x)$ подынтегральная функция, $[a,b]$ - интервал интегрирования, n - число отрезков разбиения. В основной программе использовать метод trap для вычисления интегралов: $\int_{-1}^{4} (x+\cos x)\,dx$ и $\int_0^2 \frac{tg(x+1)}{x+1}\,dx$.

Реализовать вызов метода двумя способами: в виде передаваемого lambda-выражения и в виде блока.

main.rb

```ruby
def trap(x_a, x_b, n_divisions, f_lambda = nil, &f_block)
  func = f_lambda || f_block
  range = (x_b - x_a).to_f
  dx = range / n_divisions
  sum = (x_a...x_b).step(dx).reduce do |acc, x|
    acc + func.call(x)
  end
  dx * (((func.call(x_a) + func.call(x_b)) / 2) + sum)
end

```

test.rb

```ruby
require 'minitest/autorun'
require_relative 'main'

# Integral testing
class Test < Minitest::Test
  def test_f1_lambda
    f1 = ->(x) { x + Math.cos(x) }
    assert_in_epsilon 7.58467, trap(-1, 4, 10_000, f1), 1e-3
  end

  def test_f1_block
    assert_in_epsilon 7.58467, trap(-1, 4, 10_000) { |x| x + Math.cos(x) }, 1e-3
  end

  # для второй функции интеграл расходящийся
end

```

user.rb

```ruby
require_relative 'main'

f1 = ->(x) { x + Math.cos(x) }
puts("Результат вычисления интеграла первой функции - #{trap(-1, 4, 10_000, f1)}")
puts("Результат вычисления интеграла второй функции - #{trap(0, 2, 10_000) { |x|
Math.tan(x + 1) / (x + 1) }}")

```

Результат выполнения программ:

```
PS C:\Users\might\Desktop\WPL_bmstu\Lab6\Part 3> ruby .\user.rb
Результат вычисления интеграла первой функции - 7.584168487736007
Результат вычисления интеграла второй функции - -34.408570243914426
PS C:\Users\might\Desktop\WPL_bmstu\Lab6\Part 3> ruby .\test.rb
Run options: --seed 64333

# Running:

..

Finished in 0.014908s, 134.1589 runs/s, 134.1589 assertions/s.
2 runs, 2 assertions, 0 failures, 0 errors, 0 skips
```

Проверка кода при помощи *rubocop*

```
PS C:\Users\might\Desktop\WPL_bmstu\Lab6> rubocop
Inspecting 9 files
.........

9 files inspected, no offenses detected
```

Итоговый код данной лабораторной работы доступен по ссылке:
https://github.com/tenessinum/WPL_bmstu/tree/main/Lab6