|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 **09.03.03 Прикладная информатика.**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 6 |

Название: Программирование на языке Ruby

**Дисциплина:** Языки интернет-программирования

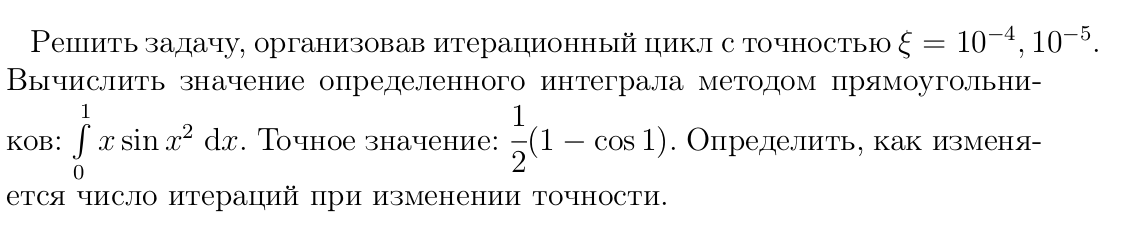
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-35Б |  |  | Л.Э. Барсегян |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  |  |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2022

Вариант № 5.02

**Часть 1.**

Задание:



**Код программы.**

**Код файла integral.rb**

# frozen\_string\_literal: true

# module Integral

module Integral

@n = 1

@sum = 0

@exact\_result = (1 - Math.cos(1)) / 2

def self.sum

@sum

end

def self.calc\_rects(left)

k = 1

while k <= @n

x = left + (@h \* (k - 1)) + (@h / 2)

y = x \* Math.sin(x\*\*2)

@sum += @h \* y

k += 1

end

end

def self.print\_sum(left, right, epsilon)

puts "Левый предел left = #{left}"

puts "Правый предел right = #{right}"

puts "Точность epsilon = #{epsilon}"

puts "Точный результат: #{@exact\_result}"

puts "Полученная сумма с точностью #{epsilon}: #{@sum}"

puts "Число итераций (т.е. число прямоугольников n): #{@n - 1}\n\n"

end

def self.integrate(left, right, epsilon)

return if epsilon.zero? || left == right

while (@exact\_result - @sum).abs > epsilon.to\_f

@sum = 0

@h = (right - left).to\_f / @n

calc\_rects(left)

@n += 1

end

print\_sum(left, right, epsilon)

@sum

end

end

**Код файла main.rb**

# frozen\_string\_literal: true

require\_relative 'integral'

Integral.integrate(0, 1, 10\*\*-4)

Integral.integrate(0, 1, 10\*\*-5)

**Код файла integral\_spec.rb**

# frozen\_string\_literal: true

require\_relative 'integral'

RSpec.describe Integral do

describe '#integrate' do

context 'epsilon == 0' do

it 'returns nil' do

a = rand(0..10)

b = rand(11..50)

expect(Integral.integrate(a, b, 0)).to eq(nil)

end

end

context 'a == b' do

it 'returns nil' do

a = rand(0..50)

e = rand(0.0..1.0)

expect(Integral.integrate(a, a, e)).to eq(nil)

end

end

context 'when everything is fine' do

it 'returns correct result' do

a = rand(0.0..5.0)

b = rand(5.1..10.0)

e = rand(0.0..1.0)

epsilon = (Integral.integrate(a, b, e) - (1 - Math.cos(1)) / 2)

expect(epsilon.abs).to be < e

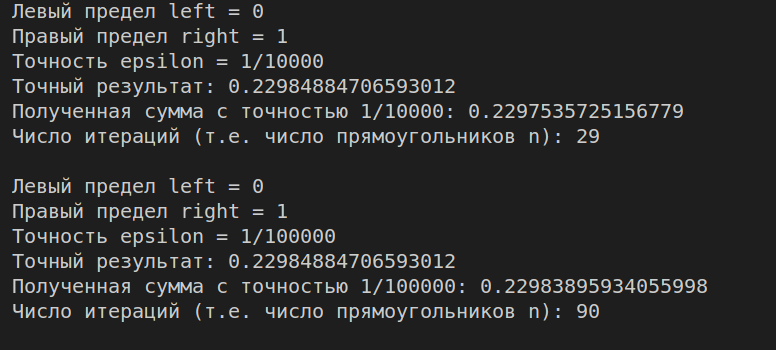
end

end

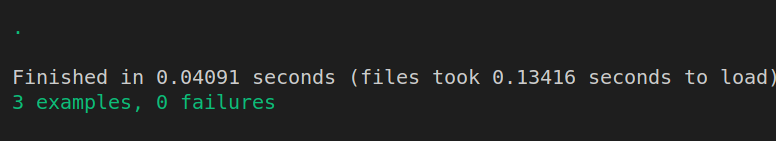
end

end

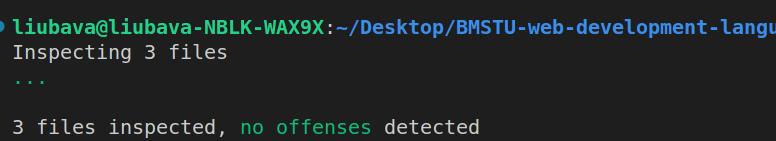
**Результаты выполнения программы**

****

**Результат тестирования на основе RSpec**

****

**Проверка на соответствие стилю программирования на Ruby при помощи rubocop**

****

**Часть 2.**

**Задание:**

****

**Код файла main.rb**

# frozen\_string\_literal: true

require\_relative 'integral'

Integral.integrate(0, 1, 10\*\*-4)

Integral.integrate(0, 1, 10\*\*-5)

**Код файла integral.rb**

**# frozen\_string\_literal: true**

# module Integral

module Integral

@exact\_result = (1 - Math.cos(1)) / 2

def self.h(right, left)

(right - left).to\_f / @n

end

def self.y(x\_value)

x\_value \* Math.sin(x\_value\*\*2)

end

def self.sum\_n(left, right)

@n.times do |k|

x = left + (h(right, left) \* k) + (h(right, left) / 2)

@sum += h(right, left) \* y(x)

k + 1

end

end

def self.calc\_rects(left, right)

Enumerator.new do |yielder|

loop do

@sum = 0

sum\_n(left, right)

yielder << @sum

@n += 1

end

end

end

def self.integrate(left, right, epsilon)

return if epsilon.zero? || left >= right

@n = 1

calc\_rects(left, right).take\_while { |sum| (@exact\_result - sum).abs > epsilon }

print\_sum(left, right, epsilon)

@sum

end

def self.print\_sum(left, right, epsilon)

puts "Левый предел left = #{left}"

puts "Правый предел right = #{right}"

puts "Точность epsilon = #{epsilon}"

puts "Точный результат: #{@exact\_result}"

puts "Полученная сумма с точностью #{epsilon}: #{@sum}"

puts "Число итераций (т.е. число прямоугольников n): #{@n}\n\n"

end

end

**Код файла integral\_spec.rb**

# frozen\_string\_literal: true

require\_relative 'integral'

RSpec.describe Integral do

include Integral

describe '#integrate' do

context 'epsilon == 0' do

it 'returns nil' do

a = rand(0..10)

b = rand(11..50)

expect(Integral.integrate(a, b, 0)).to eq(nil)

end

end

context 'a == b' do

it 'returns nil' do

a = rand(0..50)

e = rand(0.0..1.0)

expect(Integral.integrate(a, a, e)).to eq(nil)

end

end

context 'when everything is fine' do

it 'returns correct result' do

e = rand(0.0..0.01)

epsilon = (Integral.integrate(0, 1, e) - (1 - Math.cos(1)) / 2)

expect(epsilon.abs).to be < e

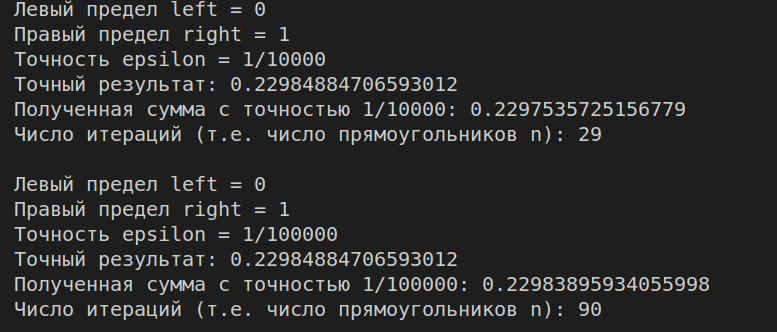
end

end

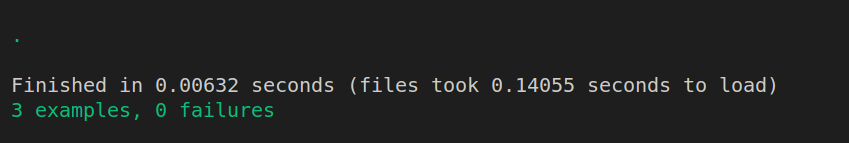
end

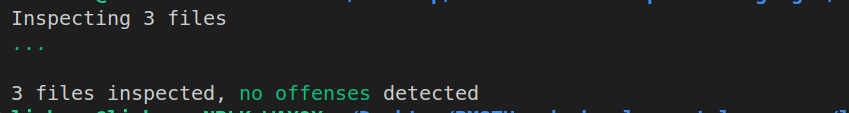
end

**Результаты выполнения программы**

****

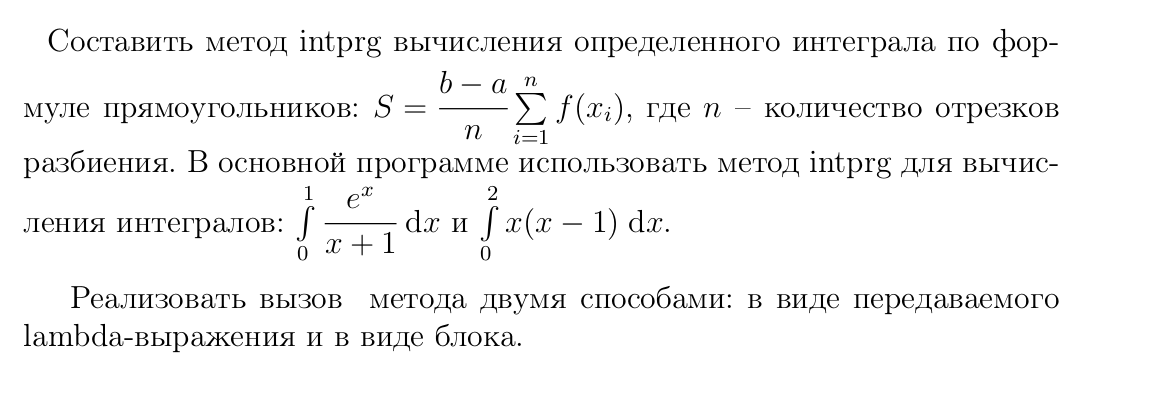
**Результат тестирования на основе Rspec**

**Про****верка на соответствие стилю программирования на Ruby при помощи rubocop**

****

**Часть 3.**

**Задание:**

**Выполнение программы с помощью блока yield.**

**Код файла main\_yield.rb**

# frozen\_string\_literal: true

require\_relative 'integral\_yield'

Integral.intprg(0, 1, 400) { |x| Math.exp(x) / (x + 1) }

Integral.intprg(0, 2, 500) { |x| x \* (x - 1) }

**Код файла integral\_yield.rb**

# frozen\_string\_literal: true

# module Integral

module Integral

def self.calc\_rects

@sum = 0

@n.times do |k|

delta\_x = (@right - @left).to\_f / @n

x = @left + (delta\_x \* k) + (delta\_x / 2)

y = yield x

@sum += delta\_x \* y

k + 1

end

@sum

end

def self.intprg(left, right, rects\_num, &block)

return if rects\_num.zero? || left >= right

@n = rects\_num

@left = left

@right = right

calc\_rects(&block)

print\_sum

@sum

end

def self.print\_sum

puts "Левый предел left = #{@left}"

puts "Правый предел right = #{@right}"

puts "Число итераций (т.е. число прямоугольников n): #{@n}"

puts "Полученная сумма: #{@sum}\n\n"

end

end

**Код файла integral\_yield\_spec.rb**

# frozen\_string\_literal: true

require\_relative 'integral\_yield'

RSpec.describe Integral do

describe '#intprg' do

context 'epsilon == 0' do

it 'returns nil' do

expect(Integral.intprg(30, 400, 0) { |x| Math.exp(x) / (x + 1) }).to eq(nil)

end

end

context 'a == b' do

it 'returns nil' do

a = rand(0..50)

expect(Integral.intprg(a, a, 500) { |x| Math.exp(x) / (x + 1) }).to eq(nil)

end

end

context 'when everything is fine' do

it 'returns correct result' do

epsilon = (Integral.intprg(0, 1, rand(20..10\_000)) { |x| Math.exp(x) / (x + 1) } - 1.1253860828001119)

expect(epsilon.abs).to be < 0.1

epsilon = (Integral.intprg(0, 2, rand(20..10\_000)) { |x| x \* (x - 1) } - 2 / 3.to\_f)

expect(epsilon.abs).to be < 0.1

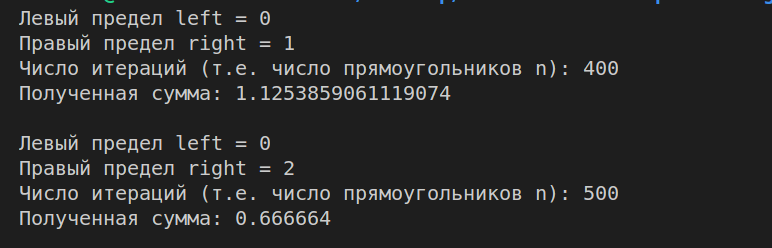
end

end

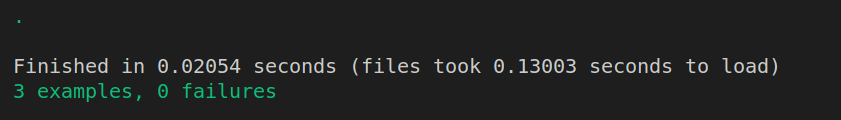
end

end

**Результаты выполнения программы**



**Результат тестирования на основе Rspec**



**Выполнение программы с помощью передаваемого метода lambda.**

**Код файла integral\_lambda.rb**

**# frozen\_string\_literal: true**

# module Integral

module Integral

def self.calc\_rects(lambda)

@sum = 0

@n.times do |k|

delta\_x = (@right - @left).to\_f / @n

x = @left + (delta\_x \* k) + (delta\_x / 2)

y = lambda.call(x)

@sum += delta\_x \* y

k + 1

end

@sum

end

def self.intprg(left, right, rects\_num, lambda)

return if rects\_num.zero? || left >= right

@n = rects\_num

@left = left

@right = right

calc\_rects(lambda)

print\_sum

@sum

end

def self.print\_sum

puts "Левый предел left = #{@left}"

puts "Правый предел right = #{@right}"

puts "Число итераций (т.е. число прямоугольников n): #{@n}"

puts "Полученная сумма: #{@sum}\n\n"

end

end

**Код файла main\_lambda.rb**

**# frozen\_string\_literal: true**

require\_relative 'integral\_lambda'

func1 = ->(x) { Math.exp(x) / (x + 1) }

Integral.intprg(0, 1, 10\_000, func1)

func2 = ->(x) { x \* (x - 1) }

Integral.intprg(0, 2, 10\_000, func2)

**Код файла integral\_lambda\_spec.rb**

**# frozen\_string\_literal: true**

require\_relative 'integral\_lambda'

RSpec.describe Integral do

include Integral

describe '#intprg' do

context 'epsilon == 0' do

it 'returns nil' do

func1 = ->(x) { Math.exp(x) / (x + 1) }

expect(Integral.intprg(rand(0..10), rand(11..50), 0, func1)).to eq(nil)

end

end

context 'a == b' do

it 'returns nil' do

a = rand(0..50)

func1 = ->(x) { Math.exp(x) / (x + 1) }

expect(Integral.intprg(a, a, rand(1..10\_000), func1)).to eq(nil)

end

end

context 'when everything is fine' do

it 'returns correct result' do

func1 = ->(x) { Math.exp(x) / (x + 1) }

expect((Integral.intprg(0, 1, rand(20..10\_000), func1) - 1.1253860828001119).abs).to be < 0.1

func2 = ->(x) { x \* (x - 1) }

expect((Integral.intprg(0, 2, rand(20..10\_000), func2) - 2 / 3.to\_f).abs).to be < 0.1

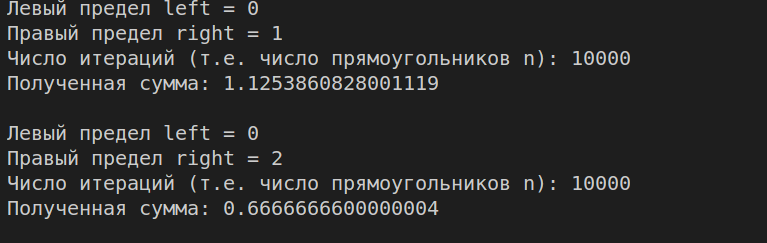
end

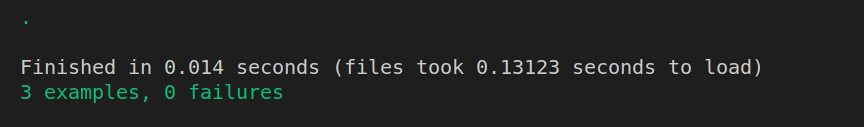
end

end

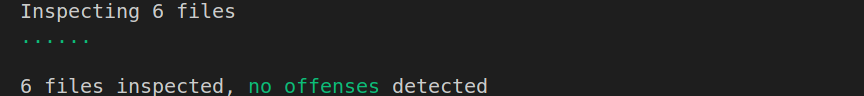
end

**Результаты выполнения программы:**

**Результат тестирования на основе Rspec**



**Проверка всех файлов на соответствие стилю программирования на Ruby при помощи rubocop**

**Вывод:** ознакомилась с функциональным стилем Ruby, попробовала улучить код с помощью Enumerator.