

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«Московский государственный индустриальный университет»  
(ФГБОУ ВПО «МГИУ»)  
КАФЕДРА ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

## КУРСОВАЯ РАБОТА

по дисциплине «Методы хранения и обработки информации»  
на тему «Вычисление значений выражений, содержащих только  
записанные в десятичной системе счисления натуральные числа,  
абсолютная величина которых не превосходит 3999. Вычисление  
мощности множества точек пересечения границы выпуклой оболочки с  
замкнутым единичным кругом с центром в начале координат.  
Нахождение суммы длин проекций полностью невидимых рёбер  
полиэдра, центр которых находится на расстоянии строго меньше  
единицы от плоскости  $x = 2$ »

Группа

141131

Студент

Д.А. Хотелов

Руководитель работы  
к.ф.-м.н., доцент

Е.А. Роганов

Москва 2015

## Аннотация

Работа посвящена модификации проектов «Стековый копилятор формул», «Выпуклая оболочка» и «Изображение проекции полиэдра». В первом из этих проектов вычислялось значение выражений, содержащих только записанные в десятичной системе счисления натуральные числа, абсолютная величина которых не превосходит 3999. Во втором проекте вычислялась мощность множества точек пересечения границы выпуклой оболочки с замкнутым единичным кругом с центром в начале координат. В третьем проекте находилась сумма длин проекций полностью невидимых рёбер, центр которых находится на расстоянии строго меньше единицы от плоскости  $x = 2$ .

## Содержание

1.	Введение . . . . .	3
2.	Модификация проекта «Стековый копилятор формул» . . . . .	3
3.	Модификация проекта «Выпуклая оболочка» . . . . .	5
4.	Модификация проекта «Изображение проекции полиэдра» . . . . .	9
5.	Приложение . . . . .	12

# 1. Введение

В данной курсовой работе рассматриваются модификации трёх эталонных проектов «Стековый компилятор формул», «Выпуклая оболочка» и «Изображение проекции полиэдра», реализованных на объектно-ориентированном языке программирования высокого уровня Ruby.

Проект «Стековый компилятор формул»[1] представляет собой программную реализацию некоторой функции  $\tau$ , действующей из множества цепочек одного языка  $L_1$  (в рассматриваемом случае это язык арифметических формул) в множество цепочек другого  $L_2$  (язык программ стекового калькулятора) таким образом, что  $\forall \omega \in L_1$  семантика цепочек  $\omega$  и  $\tau(\omega) \in L_2$  совпадает. Говоря другими словами, компилятор реализует перевод с одного языка на другой с сохранением смысла.

Проект «Выпуклая оболочка»[2] решает задачу индуктивного перевычисления выпуклой оболочки последовательно поступающих точек плоскости и таких её характеристик, как периметр и площадь.

Проект «Изображение проекции полиэдра»[3] выполняет построение изображения полиэдра с учётом удаления невидимых линий.

Целями работы являются:

- Проект «Стековый компилятор формул» требуется модифицировать так, чтобы вычислялись значения выражений, содержащих только записанные в десятичной системе счисления натуральные числа, абсолютная величина которых не превосходит 3999.
- В проект «Выпуклая оболочка» добавить вычисление мощности множества точек пересечения границы выпуклой оболочки с замкнутым единичным кругом с центром в начале координат.
- Модифицировать проект «Изображение проекции полиэдра» таким образом, чтобы определялась и печаталась следующая характеристика полиэдра: сумма длин проекций полностью невидимых рёбер, центр которых находится на расстоянии строго меньше единицы от плоскости  $x = 2$ .

Для того чтобы выполнить полученные задания, необходимо было изучить особенности языка Ruby, подробно разобрать каждый эталонный проект и применить полученные знания в области информатики, компьютерной математики и аналитической геометрии на плоскости и в пространстве.

## 2. Модификация проекта «Стековый компилятор формул»

### Постановка задачи

Модифицируйте эталонный проект таким образом, чтобы вычислялись значения выражений, содержащих только записанные в десятичной системе счисления натуральные числа, абсолютная величина которых не превосходит 3999. Результат должен быть выдан в десятичной системе счисления.

После запуска файла `calc.rb` пользователю предлагается ввести арифметическое выражение, затем происходит вывод результата вычисленного выражения. Допустим для выражения  $45 * 2 - 10$  должен выводиться ответ 80. В эталонном проекте можно вводить лишь цифры от 0 до 9.

## Решение задачи и модификация кода

Чтобы выполнить поставленную задачу, необходимо знать где обрабатывается каждый символ и как обработать число:  $10 < n < 5000$ .

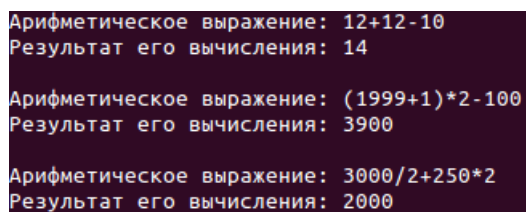
Обработка каждого символа осуществляется в файле `compf.rb` в методе `compile` класса `Compf`. Необходимо уметь обрабатывать символ, длина которого больше одного. Кроме этого следует учитывать проверку каждого символа в файле `calc.rb` в методе `check_symbol` класса `Calc`. Здесь модификация наиболее проста: необходимо допускать длину цифр больше 1 и не допускать значения самих цифр больше 3999. Эта проверка реализована следующим образом:

```
def check_symbol(c)
  raise "Недопустимый символ '#{c}'" if c !~ /[0-9]+/ || c.to_i > 3999
end
```

Сама идея обработки символов заключается в том, чтобы перед обработкой очередного символа увеличивать длину цепочки на число(если это число), пока не встретится любой другой символ. Затем обрабатывается последовательность этих чисел, а именно преобразование из строкового представления в числовое и добавление этого числа в стек для подсчёта итогового результата, и затем обрабатывается только что прибывший символ. Реализация метода `compile` класса `Compf`:

```
def compile(str)
  @data.clear
  s1 = ""
  "(#{str})".each_char do |c|
    if c =~ /[0-9]/
      s1 += c
    else
      process_symbol(s1) if s1 != ""
      s1 = ""
    end
    process_symbol(c) if s1 == ""
  end
  @data.join(' ')
end
```

Пример работы данной реализации(рис. 1):



```
Арифметическое выражение: 12+12-10
Результат его вычисления: 14

Арифметическое выражение: (1999+1)*2-100
Результат его вычисления: 3998

Арифметическое выражение: 3000/2+250*2
Результат его вычисления: 2000
```

Рис. 1.

### 3. Модификация проекта «Выпуклая оболочка»

#### Постановка задачи

Модифицируйте эталонный проект таким образом, чтобы вычислялась мощность множества точек пересечения границы выпуклой оболочки с замкнутым единичным кругом с центром в начале координат.

После запуска программа предлагает пользователю ввести последовательно координаты вершин выпуклой оболочки. Введённая точка индуктивно добавляется в выпуклую оболочку. Нам же необходимо вместе со значениями периметра и площади выпуклой оболочки выводить мощность множества точек пересечения границы выпуклой оболочки с замкнутым единичным кругом с центром в начале координат.

Допустим для точек  $A(-1, -1)$ ,  $B(1, 1)$  и  $C(1, -1)$  программа выдаст в качестве ответа `infinity`, потому что отрезок  $AB$  пересекает единичный круг в бесконечном множестве точек, но при добавлении точки  $D(-1, 1)$  множество перевычисляется, и в качестве ответа выдаётся `4`, потому что квадрат  $ABCD$  описывает окружность и пересекается с ней в четырёх точках.

#### Решение и модификация кода

Для решения данной задачи нам необходимо находить расстояние от центра круга до стороны выпуклой оболочки. Это задание сводится к нахождению расстояния от точки до отрезка (точка нам задана  $O(0, 0)$ , а отрезком является сторона выпуклой оболочки).

Допустим нам дан отрезок  $AB$  и точка  $O$ . Расстоянием от точки  $O$  до отрезка  $AB$  является отрезок  $OA$ ,  $OB$  или высота из точки  $O$  до  $AB$  (если эта высота падает на  $AB$ ). Высота падает на  $AB$ , если  $\angle OAB$  и  $\angle OBA$  являются острыми или один из них прямой, другой острый. Нам даны лишь координаты этих точек. С помощью метода `dist` класса `R2Point` можно найти расстояние всех трёх сторон. Затем по теореме косинусов находим  $\cos \angle A$  и  $\cos \angle B$ :

$$OB^2 = AO^2 + AB^2 - 2 \cdot AO \cdot AB \cos \angle A \Rightarrow \cos \angle A = \frac{AO^2 + AB^2 - OB^2}{2 \cdot AO \cdot AB}$$

Аналогично находим  $\cos \angle B$ . Затем определяем – является ли угол острым или прямым. Если выполняется условие:  $0 \leq \cos \alpha \leq 1$ , то угол острый или прямой. Саму высоту мы находим из площади треугольника:  $S = \frac{1}{2} \cdot AB \cdot h \Rightarrow h = \frac{2 \cdot S}{AB}$ . Сторона  $AB$  известна, площадь можно найти с помощью метода `R2Point.area` класса `R2Point`.

Теперь можно найти расстояние от точки до стороны выпуклой оболочки, как наименьшее значение длины среди сторон  $AO$ ,  $OB$  и  $h$ .

После этого можно найти множество точек пересечения отрезка с замкнутым единичным кругом в начале координат. Если расстояние меньше единицы, то множеством точек пересечения будет являться `infinity` (рис. 2):

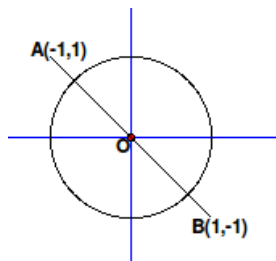


Рис. 2.

Если расстояние равно единицы, то множеством точек пересечения будет являться 1 (рис. 3):

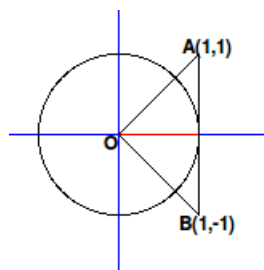


Рис. 3.

Если расстояние больше единицы, то множеством точек пересечения будет являться 0 (рис. 4):

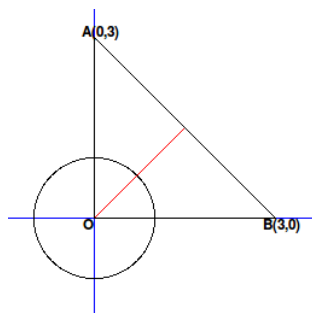


Рис. 4.

Метод `R2Point.intersect`, который принимает две точки в качестве аргумента и выдаёт множество точек пересечения этого отрезка с единичным кругом описан в классе `R2Point` и реализован следующим образом:

```
def R2Point.intersect(p1, p2)
    zero = R2Point.new(0,0)
    ao = p1.dist(zero)
    bo = p2.dist(zero)
    ab = p1.dist(p2)
```

```

cosA = (ao**2+ab**2-bo**2) / (2*ao*ab)
cosB = (bo**2+ab**2-ao**2) / (2*bo*ab)
h = nil
if (0..1).include?(cosA) && (0..1).include?(cosB)
  h = 2.0*R2Point.area(p1, p2, zero).abs/ab
end
a = [ao, bo]
a << h if h != nil
r = a.min
if r < 1
  "infinity"
elsif r == 1
  1
else
  0
end
end
end

```

Теперь можно находить множество точек пересечения стороны выпуклой оболочки с кругом. Рассмотрим это вычисление для всей выпуклой оболочки.

Если выпуклой оболочкой является точка, то множество точек пересечения с кругом будет сама эта точка, либо множество будет равняться нулю. Для отрезка мы уже знаем как найти это множество, а многоугольник рассмотрим подробно.

Так как ответом пересечения стороны выпуклой оболочки и круга может быть число или `infinity`, то целесообразно хранить сумму этих чисел(`@n_point`), количество `infinity`(`@n_infinity`) и итоговый ответ(`@n_points`) в разных переменных. Для того, чтобы менять значения этих переменных в зависимости от удаления или добавления ребра выпуклой оболочки, написан метод `oper_with_n_p`, принимающий на вход две точки для отрезка и ещё один аргумент, позволяющий обрабатывать ситуацию, когда ребро выпуклой оболочки удаляется или добавляется. Этот метод находится в секции `private` для того, чтобы он был виден только в классе `Polygon`. Листинг данного метода:

```

def oper_with_n_p(a, b, op = "add")
  case R2Point.intersect(a, b)
  when 1
    op == "add" ? @n_point += 1 : @n_point -= 1
  when "infinity"
    op == "add" ? @n_infinity += 1 : @n_infinity -= 1
  end
  @n_points = (@n_infinity > 0 ? "infinity" : @n_point)
end

```

Теперь с помощью этого метода можно индуктивно вычислять множество точек пересечения замкнутого круга с центром в начале координат.

При добавлении новой точки необходимо учитывать удаление освещенных рёбер, соответственно мощность множества точек пересечения необходимо пересчитывать. Чтобы сделать это индуктивно, мы используем дек, в который помещаем значение точек – вершин выпуклой оболочки.

Сам процесс индуктивного перевычисления мощности множества точек пересечения можно представить в таблице(табл. 1), которая приведена ниже:

Удаляем ребро, соединяющее начало и конец дека, если оно освещено нашей точкой:	<code>oper_with_n_p(@points.last,@points.first,"del")</code>
Удаляем освещенные ребра из начала дека:	<code>oper_with_n_p(p, @points.first, "del")</code>
Удаляем освещенные ребра из конца дека:	<code>oper_with_n_p(p, @points.last, "del")</code>
Обрабатываем два добавленных ребра:	<code>oper_with_n_p(t, @points.first)</code> <code>oper_with_n_p(t, @points.last)</code>

Таблица 1.

Пример работы программы представлен на рис. 5. Добавляются точки ABC и результат: **infinity**. Затем добавляется точка D(рис. 6), множество точек пересчитывается и результатом будет число 4 – четыре точки пересечения выпуклой оболочки с кругом.

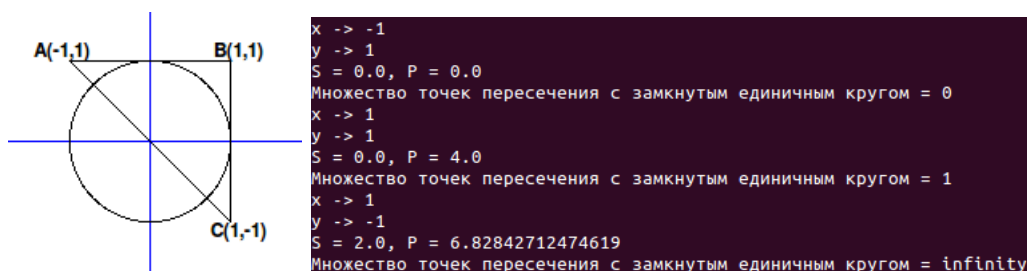


Рис. 5.

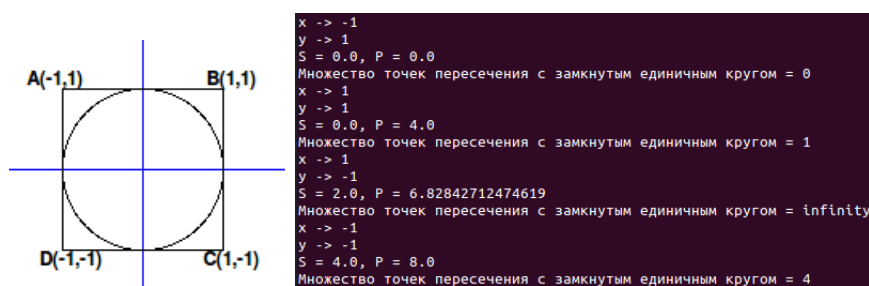


Рис. 6.



## 4. Модификация проекта «Изображение проекции полиэдра»

### Постановка задачи

Модифицируйте эталонный проект таким образом, чтобы определялась и печаталась следующая характеристика полиэдра: сумма длин проекций полностью невидимых рёбер, центр которых находится на расстоянии строго меньше 1 от плоскости  $x = 2$ .

При выполнении задания должна использоваться следующая трактовка величин, задающих полиэдр в geom - файле: реальные координаты вершин полиэдра вычисляются с учетом вращений пространства, задаваемых углами Эйлера, и коэффициента гомотетии.

Ребро будем называть полностью невидимым, если оно полностью затемнено и нет ни одного отрезка в переменной @gaps класса Edge. В переменной @gaps хранятся только незатемнённые отрезки от грани.

### Решение задачи

Для решения поставленной задачи нам необходимо находить длину проекции невидимого ребра. Так как ребро задаётся двумя точками в пространстве, то проекциями этих точек на плоскость  $XY$  будут соответственные точки с координатами  $(x; y)$  без учёта  $z$ -координаты. Поэтому длину ребра находим через две точки  $A(x_1; y_1)$  и  $B(x_2; y_2)$  по формуле:

$$l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Так же нам необходим центр ребра, но здесь надо учитывать то, что нам нужна лишь  $x$ -координата, потому что мы определяем расстояние от этого центра до плоскости  $x = 2$ . Для этого нам достаточно найти центр ребра только по  $x$ -координате. Нужный центр ребра вычисляем по формуле:

$$\frac{(x_1 + x_2)}{2}$$

Остаётся определить условие того, что центр ребра находится на расстоянии строго меньше единицы от плоскости  $x = 2$ . Центр ребра должен быть больше единицы, но меньше трёх.

### Модификация кода

Метод `some_len`, который находит длину полностью невидимого ребра, центр которого находится на расстоянии строго меньше единицы от плоскости  $x = 2$ , мы определили в файле `shadow/polyedr.rb` в классе `Edge`. Листинг данного метода:

```

def some_len
  s = 0
  if @gaps.empty?
    k = (@beg.x + @fin.x)/2.0
    s = ((@beg.x - @fin.x) ** 2 + (@beg.y - @fin.y) ** 2) ** 0.5 if k < 3 && k > 1
  end
  s
end

```

Но нам нужна сумма всех таких рёбер полиэдра. Для этого мы определили метод `some_sum_len_edges` в классе `Polyedr`. В данном методе мы обходим все рёбра и складываем нужные нам с помощью метода `some_len`. Листинг метода `some_sum_len_edges`:

```

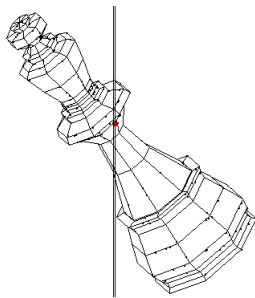
def some_sum_len_edges
  s = 0
  edges.each{|e| s += e.some_len}
  s
end

```

Затем остаётся лишь вывести эту сумму, но она должна выводиться после учёта затемнения рёбер.

Данную сумму выводим в файле `run_polyedr.rb`: `puts poliedr.some_sum_len_edges`.

Пример работы данной реализации представлен на рис. 7 (Для наглядности нарисуем плоскости  $x = 1$  и  $x = 3$ , а также центры рёбер, попадающие строго в этот диапазон):



```

Начало работы с полиэдром 'king'
Сумма длин проекций полностью невидимых рёбер,
центр которых находится на расстоянии строго меньше 1 от плоскости x = 2:
12.535071065896597
Изображение полиэдра 'king' заняло 20.672304318 сек.

```

Рис. 7.

## Список литературы и интернет-ресурсов

- [1] [http://edu1.msiu.ru/f/7275-material\\_ici\\_toc.zip/index.html](http://edu1.msiu.ru/f/7275-material_ici_toc.zip/index.html) — Описание проекта «Стековый компилятор формул».
- [2] [http://edu1.msiu.ru/f/7561-material\\_ici\\_toc.zip/index.html](http://edu1.msiu.ru/f/7561-material_ici_toc.zip/index.html),  
[http://edu1.msiu.ru/f/7591-material\\_ici\\_toc.zip/index.html](http://edu1.msiu.ru/f/7591-material_ici_toc.zip/index.html) — Описание проекта «Выпуклая оболочка».
- [3] [http://edu1.msiu.ru/f/7780-material\\_ici\\_toc.zip/index.html](http://edu1.msiu.ru/f/7780-material_ici_toc.zip/index.html),  
[http://edu1.msiu.ru/f/7811-material\\_ici\\_toc.zip/index.html](http://edu1.msiu.ru/f/7811-material_ici_toc.zip/index.html),

[http://edu1.msiu.ru/f/7863-material\\_ici\\_toc.zip/index.html](http://edu1.msiu.ru/f/7863-material_ici_toc.zip/index.html) — Описание проекта «Изображение проекции полиэдра».

- [4] <http://ru.wikipedia.org/wiki/Ruby> — Википедия (свободная энциклопедия) о языке Ruby.
- [5] С.М. Львовский. *Набор и вёрстка в системе L<sup>A</sup>T<sub>E</sub>X*, 3-е изд., испр. и доп. — М., МЦНМО, 2003. Доступны исходные тексты этой книги.
- [6] D. E. Knuth. *The T<sub>E</sub>Xbook*. — Addison-Wesley, 1984. Русский перевод: Дональд Е. Кнут. *Все про T<sub>E</sub>X*. — Протвино, РДТ<sub>E</sub>X, 1993.

## 5. Приложение

### Изменения, внесённые в эталонный проект «Выпуклая оболочка»

Изменения в файле `convex.rb`

```
7a7
> def n_points; 0 end
24a25,27
> def n_points
>   (@p.x**2 + @p.y ** 2)**(0.5) <= 1 ? 1 : 0
> end
35c38,41
< def add(r)
---
> def n_points
>   R2Point.intersect(@p, @q)
> end
> def add(r)
47c53
< attr_reader :points, :perimeter, :area
---
> attr_reader :points, :perimeter, :area, :n_points
60a67,72
>   @n_infinity = 0 #сколько раз отрезки, пересекающие круг дают в ответе "infinity"
>   @n_point = 0 #в скольких точках пересекаются все отрезки, исключая "infinity"
>   @n_points = 0 #множество точек пересечения с кругом
>   oper_with_n_p(a, b)
>   oper_with_n_p(c, b)
>   oper_with_n_p(a, c)
78c89
<
---
>   oper_with_n_p(@points.last, @points.first, "del")
95a95
>   oper_with_n_p(p, @points.first, "del")
92c104,105
< @area += R2Point.area(t, p, @points.last).abs
---
>   @area += R2Point.area(t, p, @points.last).abs
>   oper_with_n_p(p, @points.last, "del")
98a112,113
>   oper_with_n_p(t, @points.first)
>   oper_with_n_p(t, @points.last)
102a117,126
> private
> def oper_with_n_p(a, b, op = "add")
>   case R2Point.intersect(a, b)
>   when 1
>     op == "add" ? @n_point += 1 : @n_point -= 1
>   when "infinity"
>     op == "add" ? @n_infinity += 1 : @n_infinity -= 1
>   end
>   @n_points = (@n_infinity > 0 ? "infinity" : @n_point)
> end
```

Изменения в файле `r2point.rb`

```
41a41,64
> #нахождение количества точек пересечения отрезка с единичной окружностью
> def R2Point.intersect(p1, p2)
>   zero = R2Point.new(0,0)
>   ao = p1.dist(zero)
>   bo = p2.dist(zero)
>   ab = p1.dist(p2)
>   cosA = (ao**2+ab**2-bo**2) / (2*ao*ab)
>   cosB = (bo**2+ab**2-ao**2) / (2*bo*ab)
>   h = nil
```

```

> if (0..1).include?(cosA) && (0..1).include?(cosB)
>   h = 2.0*R2Point.area(p1, p2, zero).abs/ab
> end
> a = [ao, bo]
> a << h if h != nil
> r = a.min
> if r < 1
>   "infinity"
> elsif r == 1
>   1
> else
>   0
> end
> end

```

## Изменения в файле tk\_drawer.rb

```

17a22,27
> #рисование единичной окружности
> def TkDrawer.draw_new_point
>   size = SCALE
>   x = SIZE/2
>   TkCval.new(CANVAS, x + size, x + size, x - size, x - size)
> end

```

## Изменения в файле run\_tkconvex.rb

```

7a9
> TkDrawer.draw_new_point
11a14
> TkDrawer.draw_new_point
12a16
> puts "Множество точек пересечения с замкнутым единичным кругом = #{fig.n_points}"

```

## Изменения в файле convex\_спес.rb

```

29a31,34
> it "Количество точек пересечения нульугольника с единичной окружностью в центре координат = 0" do
>   expect(fig.n_points).to be 0
> end
>
34a40
> let(:fig1) { Point.new(R2Point.new(3,3)) }
59a66,73
> it "Количество точек пересечения точки (0, 0) с единичной окружностью в центре координат = 1" do
>   expect(fig.n_points).to be 1
> end
>
> it "Количество точек пересечения точки (3, 3) с единичной окружностью в центре координат = 0" do
>   expect(fig1.n_points).to be 0
> end
>
99a114,129
> describe Segment do
>   let(:fig) { Segment.new(R2Point.new(0.0,0.0), R2Point.new(1.0,0.0)) }
>   let(:fig1) { Segment.new(R2Point.new(2.0,0.0), R2Point.new(2.0,3.0)) }
>   let(:fig2) { Segment.new(R2Point.new(-1.0,1.0), R2Point.new(2.0,1.0)) }
>   context "нахождение количества точек пересечения отрезка с единичной окружностью"
>   it "Для отрезка (0,0) (1,0) = infinity" do
>     expect(fig.n_points).to eq "infinity"
>   end
>   it "Для отрезка (2,0) (2,3) = 0" do
>     expect(fig1.n_points).to be 0
>   end
>   it "Для отрезка (-1,1) (2, 1) = 1" do
>     expect(fig2.n_points).to be 1
>   end
> end
188a219,265
> describe Polygon do

```

```

>
> let(:fig) do
>   a = R2Point.new(0.0,0.0)
>   b = R2Point.new(1.0,0.0)
>   c = R2Point.new(0.0,1.0)
>   fig = Polygon.new(a,b,c)
> end
> let(:fig1) do
>   a = R2Point.new(-1.0,-1.0)
>   b = R2Point.new(1.0,1.0)
>   c = R2Point.new(-1.0,1.0)
>   d = R2Point.new(1.0,-1.0)
>   fig1 = Polygon.new(a,b,c)
>   fig1.add(d)
> end
> let(:fig2) do
>   a = R2Point.new(-2.0,-2.0)
>   b = R2Point.new(2.0,2.0)
>   c = R2Point.new(2.0,-2.0)
>   d = R2Point.new(-2.0,2.0)
>   fig2 = Polygon.new(a,b,c)
>   fig2.add(d)
> end
> context "нахождение количества точек пересечения выпуклой оболочки с единичной окружностью:" do
>
>   it "Для точек (0,0) (1,0) (0,1) = infinity" do
>     expect(fig.n_points).to eq "infinity"
>   end
>   it "Для точек (0,3) (0,-3) (1,0) = infinity" do
>     fig.add(R2Point.new(0.0,3.0))
>     fig.add(R2Point.new(0.0,-3.0))
>     expect(fig.n_points).to eq "infinity"
>   end
>   it "Для точек (0,0) (1,0) (0,1) (1,1) (-1,-1) = infinity" do
>     fig.add(R2Point.new(1.0,1.0))
>     fig.add(R2Point.new(-1.0,-1.0))
>     expect(fig.n_points).to eq "infinity"
>   end
>   it "Для точек (-1,-1) (1,1) (-1,1) (1,-1) = 4" do
>     expect(fig1.n_points).to be 4
>   end
>   it "Для точек (2,2) (2,-2) (-2,2) (-2,-2) = 0" do
>     expect(fig2.n_points).to be 0
>   end
> end
> end

```