Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Тестирование программного обеспечения

Лабораторная работа 2

Вариант 5129

Выполнили:
Марков Петр Денисович
Кривоносов Егор Дмитриевич

Группа: P33111

Преподаватель:
Яркеев Александр Сергеевич

2022 г.

Санкт-Петербург

Задание

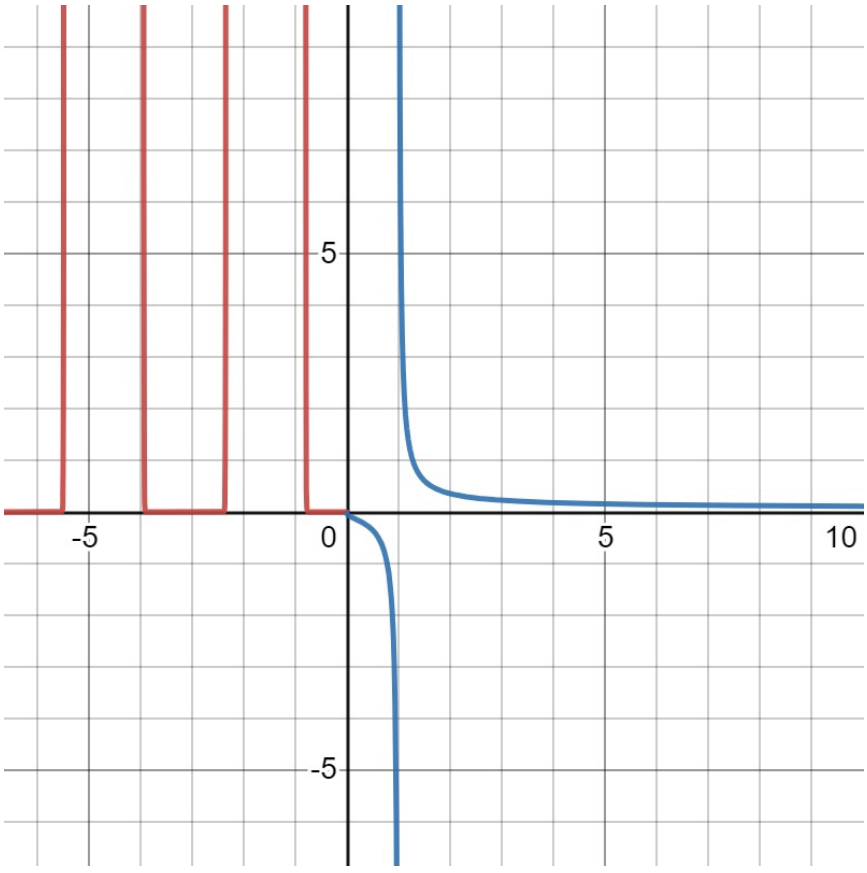
Провести интеграционное тестирование программы, осуществляющей вычисление системы функций (в соответствии с вариантом).

Вариант 5129

Введите вариант: 5129

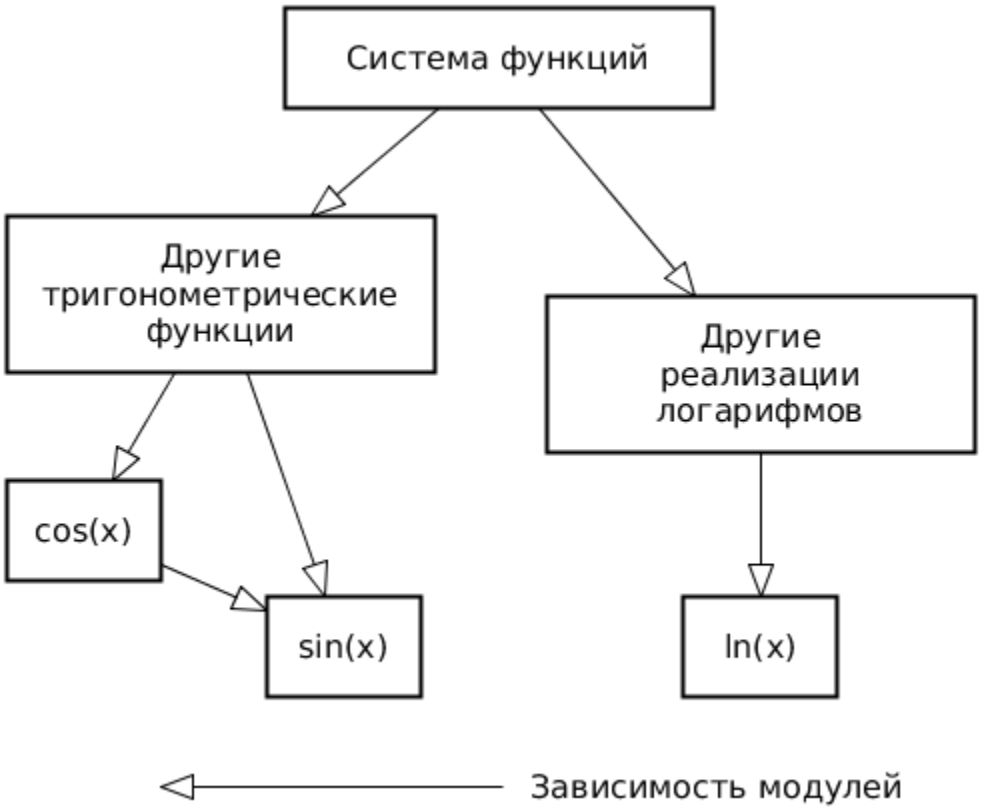
$$\begin{cases} \left(\left(\left(\left(\left(\tan(x)^2\right)^3\right)^3\right)^2\right)^3\right) & \text{if } x \le 0 \\ \left(\frac{\left(\left(\frac{\log_3(x)\cdot\log_{10}(x)}{\ln(x)}\right)+\log_5(x)\right)-\log_3(x)}{\ln(x)\cdot\log_{10}(x)}\right) & \text{if } x > 0 \end{cases}$$

```
x <= 0 : (((((tan(x) ^ 2) ^ 3) ^ 3) ^ 2) ^ 3)
x > 0 : (((((log_3(x) * log_10(x)) / ln(x)) + log_5(x)) - log_3(x)) / (ln(x) *
log_10(x)))
```



Правила выполнения работы:

1. Все составляющие систему функции (как тригонометрические, так и логарифмические) должны быть выражены через базовые (тригонометрическая зависит от варианта; логарифмическая - натуральный логарифм).
2. Структура приложения, тестируемого в рамках лабораторной работы, должна выглядеть следующим образом (пример приведён для базовой тригонометрической функции sin(x)):

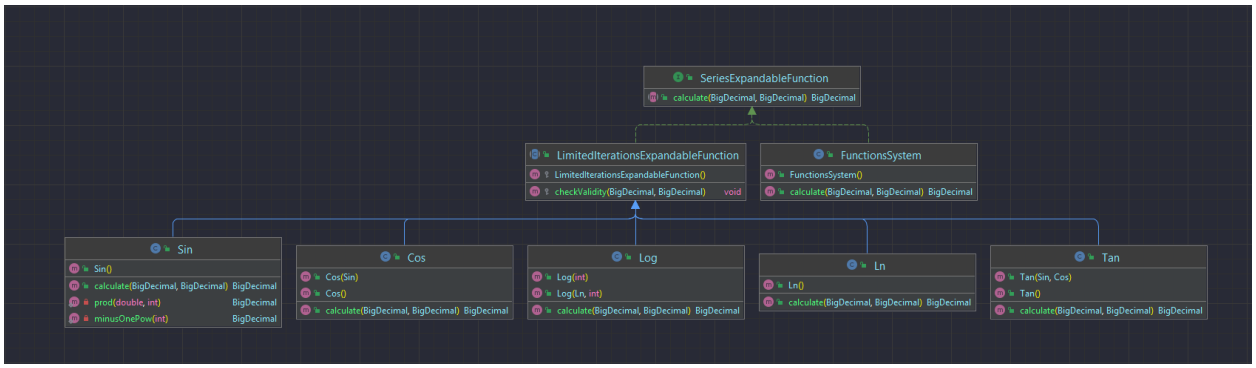


3. Обе "базовые" функции (в примере выше - sin(x) и ln(x)) должны быть реализованы при помощи разложения в ряд с задаваемой погрешностью. Использовать тригонометрические / логарифмические преобразования для упрощения функций ЗАПРЕЩЕНО.
4. Для КАЖДОГО модуля должны быть реализованы табличные заглушки. При этом, необходимо найти область допустимых значений функций, и, при необходимости, определить взаимозависимые точки в модулях.
5. Разработанное приложение должно позволять выводить значения, выдаваемое любым модулем системы, в csv файл вида «X, Результаты модуля (X)», позволяющее произвольно менять шаг наращивания X. Разделитель в файле csv можно использовать произвольный.

Выполнение

Исходный код: https://github.com/RedGry/TPO-LAB-2

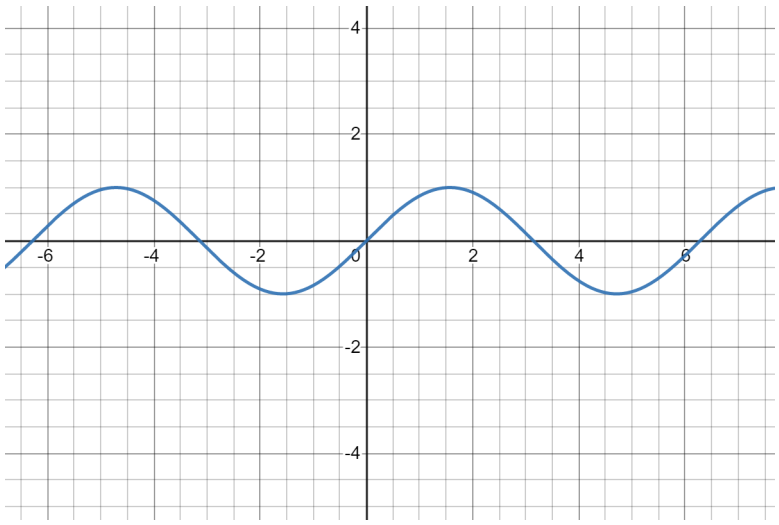
UML



Модульное тестирование

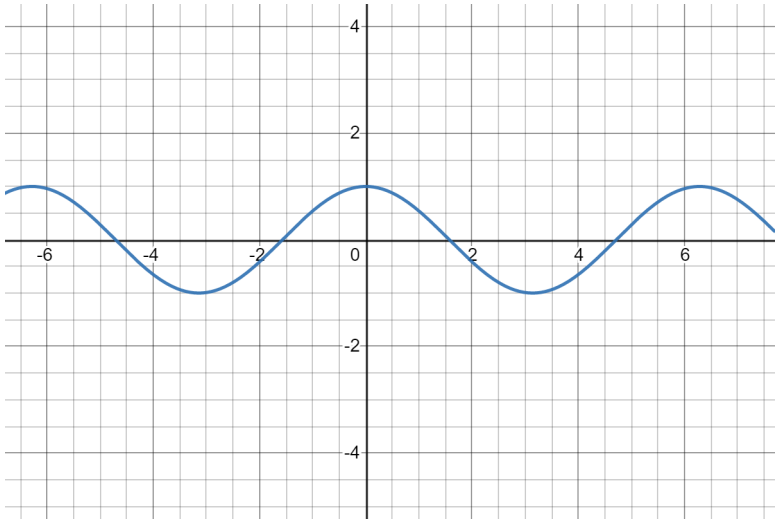
Для модульный тестов были выбраны крайние точки периодичности функций, точки, где функция меняет знак на противоположный, а по точке с каждой стороны внутри периода функций, а также, если есть места, где она неопределённа, взято случайное значение из этого промежутка (функция должна вернуть Exception или null).

sin(x)



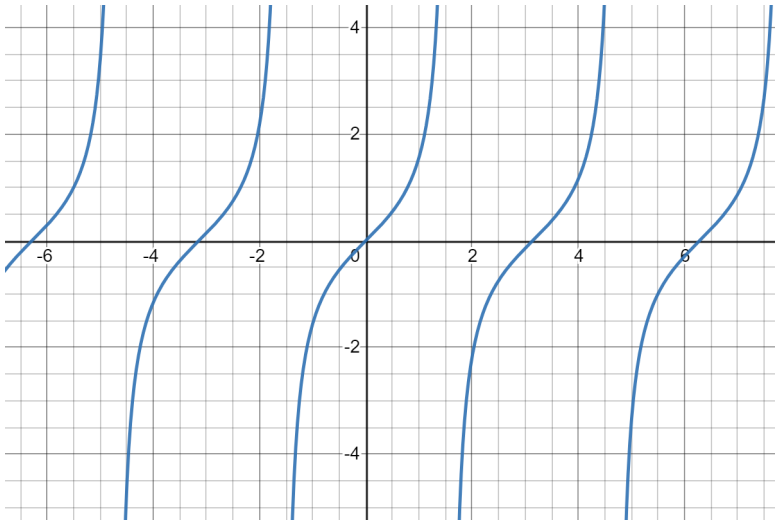
X	Y
0	0
1.57…	1
1	0.8415
-113	0.0972

cos(x)

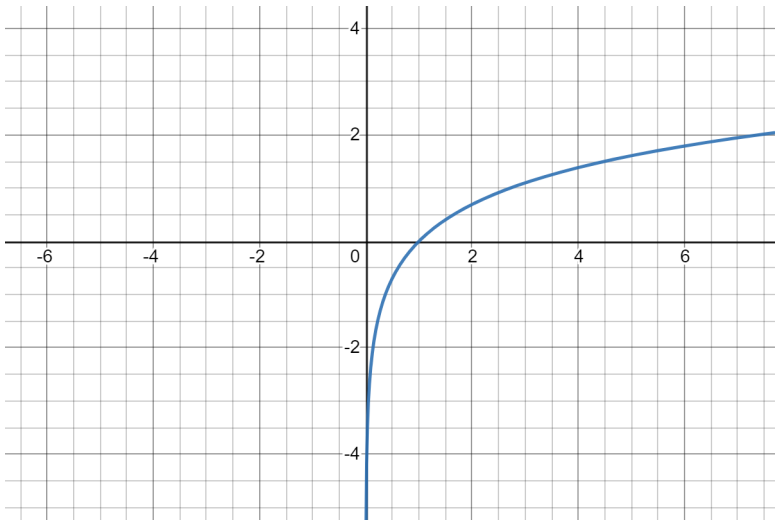
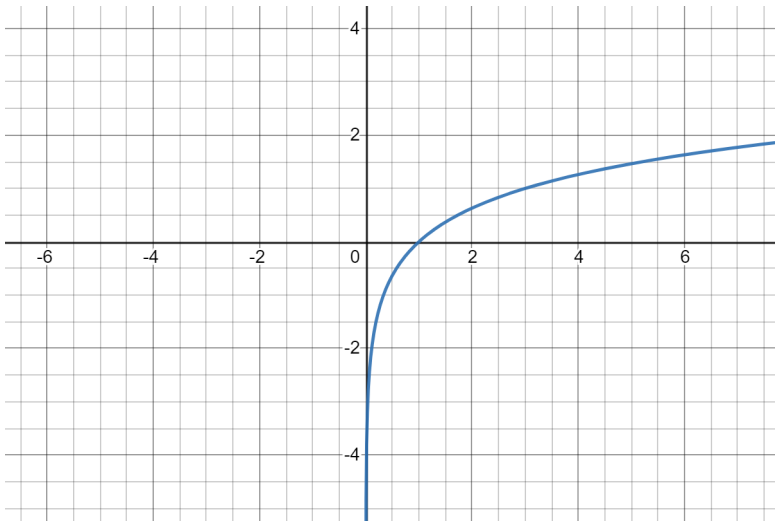
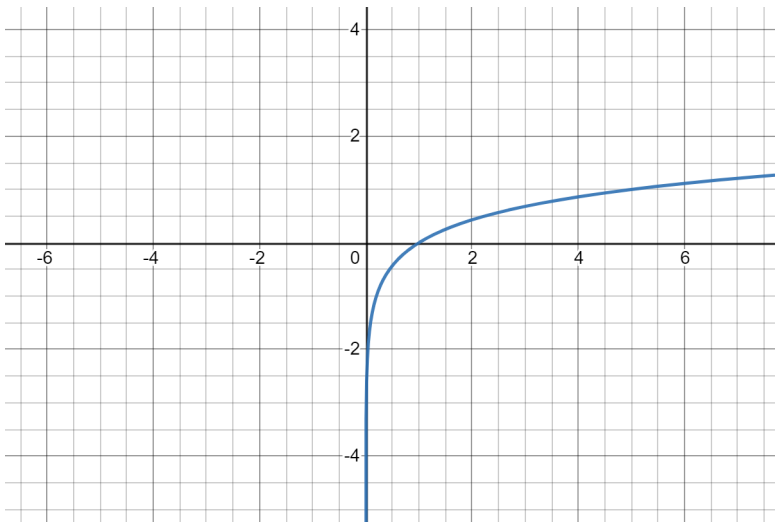


X	Y
5	0.283662
0	1
1.57…	0
1	0.5403
-543	-0.8797

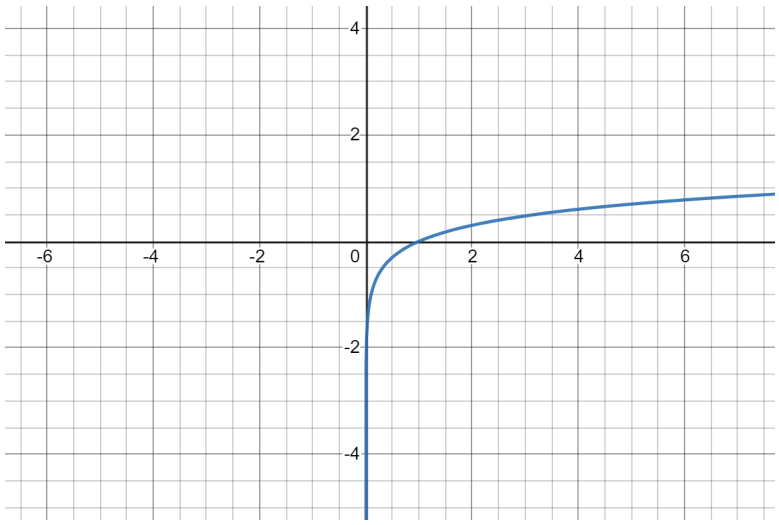
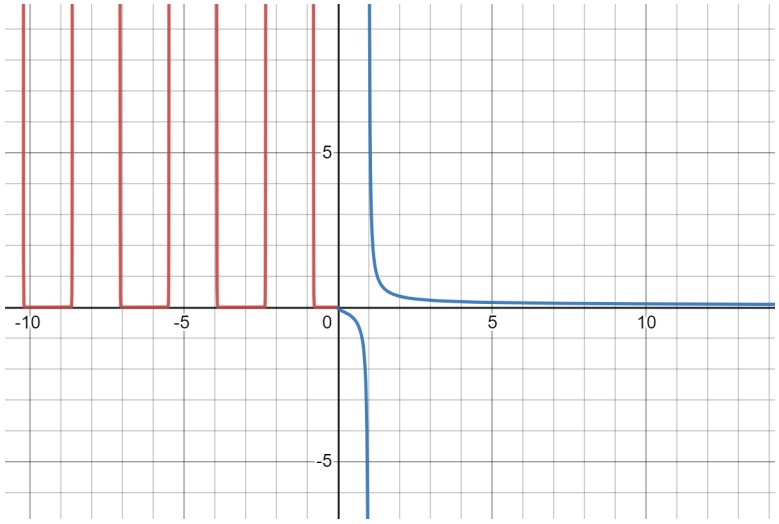
tan(x)



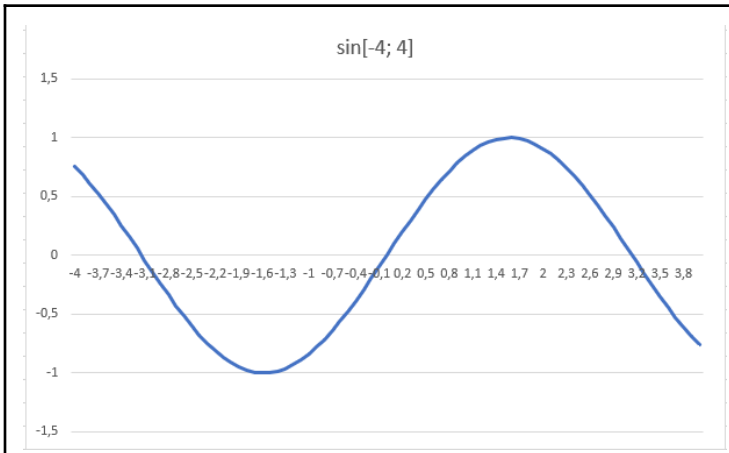
X	Y
5	-3.3805
1	1.5575
134	-1.9101
0	0

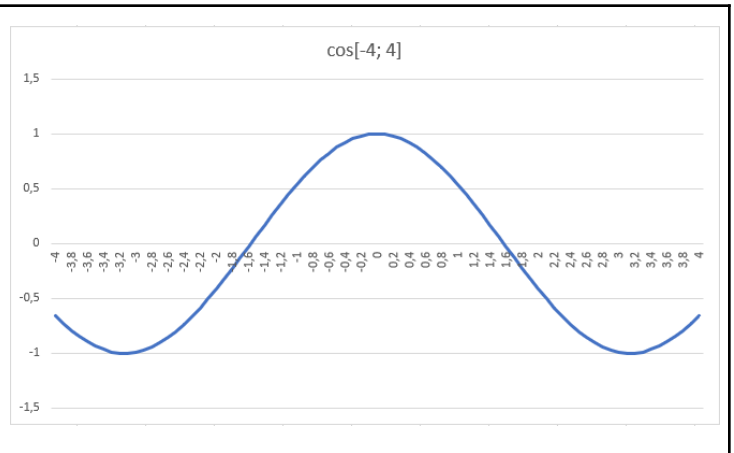
ln(x)		
log3(x)		
log5(x)		

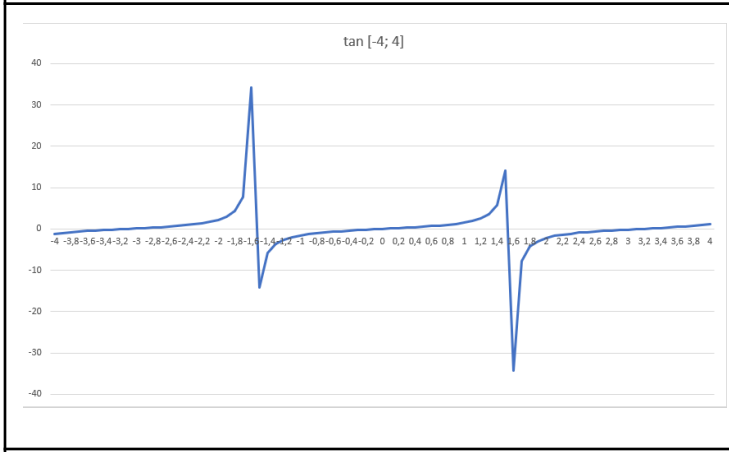
X	Y
1	0
4	1.38629

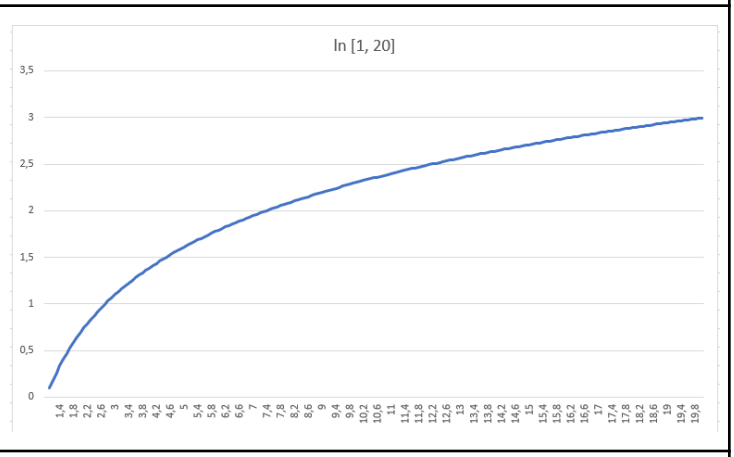
log10(x)		
system		<table><tr><td>X</td><td>Y</td></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>NULL</td></tr><tr><td>5</td><td>0.15223487</td></tr><tr><td>-5</td><td>1349425697703…</td></tr></table>

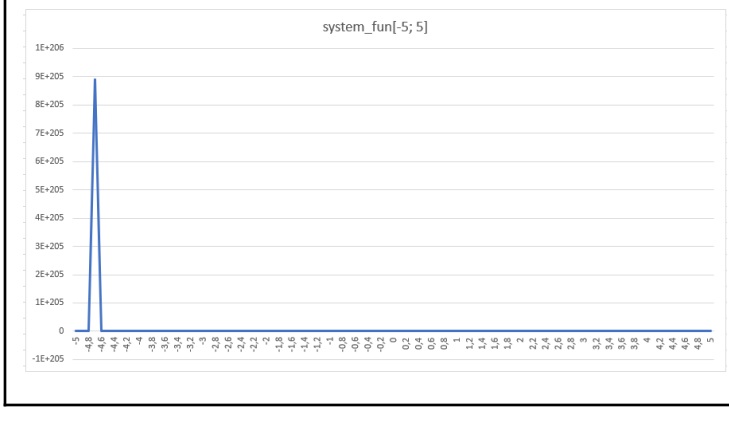
Графики (построены по csv выгрузкам)

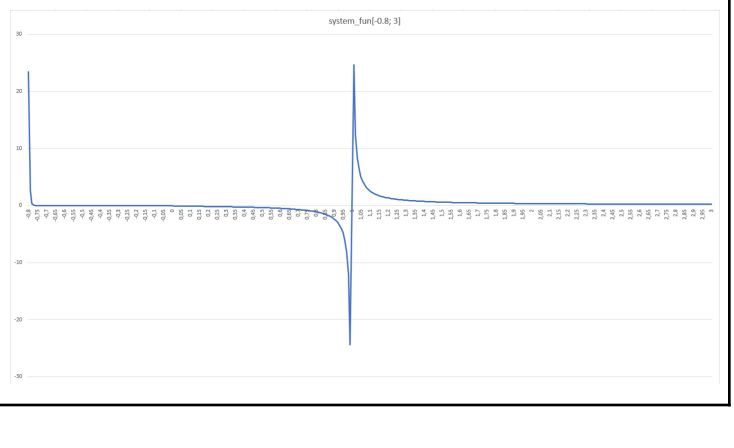












Вывод

Главная сложность при проведении интеграционного тестирования – подбор возвращаемых значений у вызываемых модулей, ведь только тогда зависимые модули смогут сгенерировать правильное значение. Это требует глубокого понимания кода, с

которым мы взаимодействуем, даже несмотря на то, что по факту сами пишем зависимые модули в виде заглушек. Это также порождает вторую проблему – написание большого количество кода для тестов (помочь с этим могут специальные фреймворки, в частности Mockito, если мы говорим про jvm).