УНИВЕРСИТЕТ ИТМО

ФГАОУ ВО «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Лабораторная работа №2

Вариант 670

Лабушев Тимофей

Группа P3402

Санкт-Петербург

2020

Задание

Провести интеграционное тестирование программы, осуществляющей вычисление системы функций:

$$\begin{cases} \left(\dfrac{\dfrac{(sin(x) \cdot sin(x))^2}{cot(x)}}{\dfrac{sec(x)}{\dfrac{sin(x)}{tan(x)}} + (sin(x) - sec(x))} \right)^2, & \text{if } x \leqslant 0 \\ \dfrac{log_5(x)}{log_2(x)} - log_2(x) - log_3(x) - log_{10}^3(x) + (log_{10}(x) \cdot log_{10}(x)), & \text{if } x > 0 \end{cases}$$

1. Все составляющие систему функции (как тригонометрические, так и логарифмические) должны быть выражены через базовые (тригонометрическая — синус; логарифмическая — натуральный логарифм)

2. Структура приложения, тестируемого в рамках лабораторной работы, должна выглядеть следующим образом:

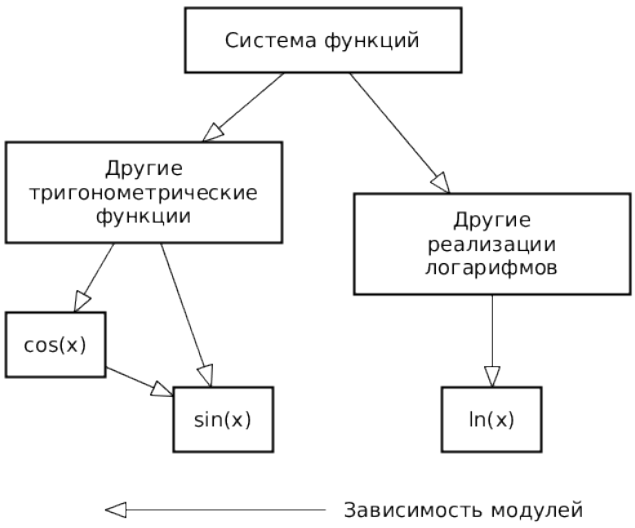

Рисунок 1. Структура тестируемого приложения

3. Обе базовые функции ($sin(x)$ 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.

UML-диаграмма классов разработанного приложения

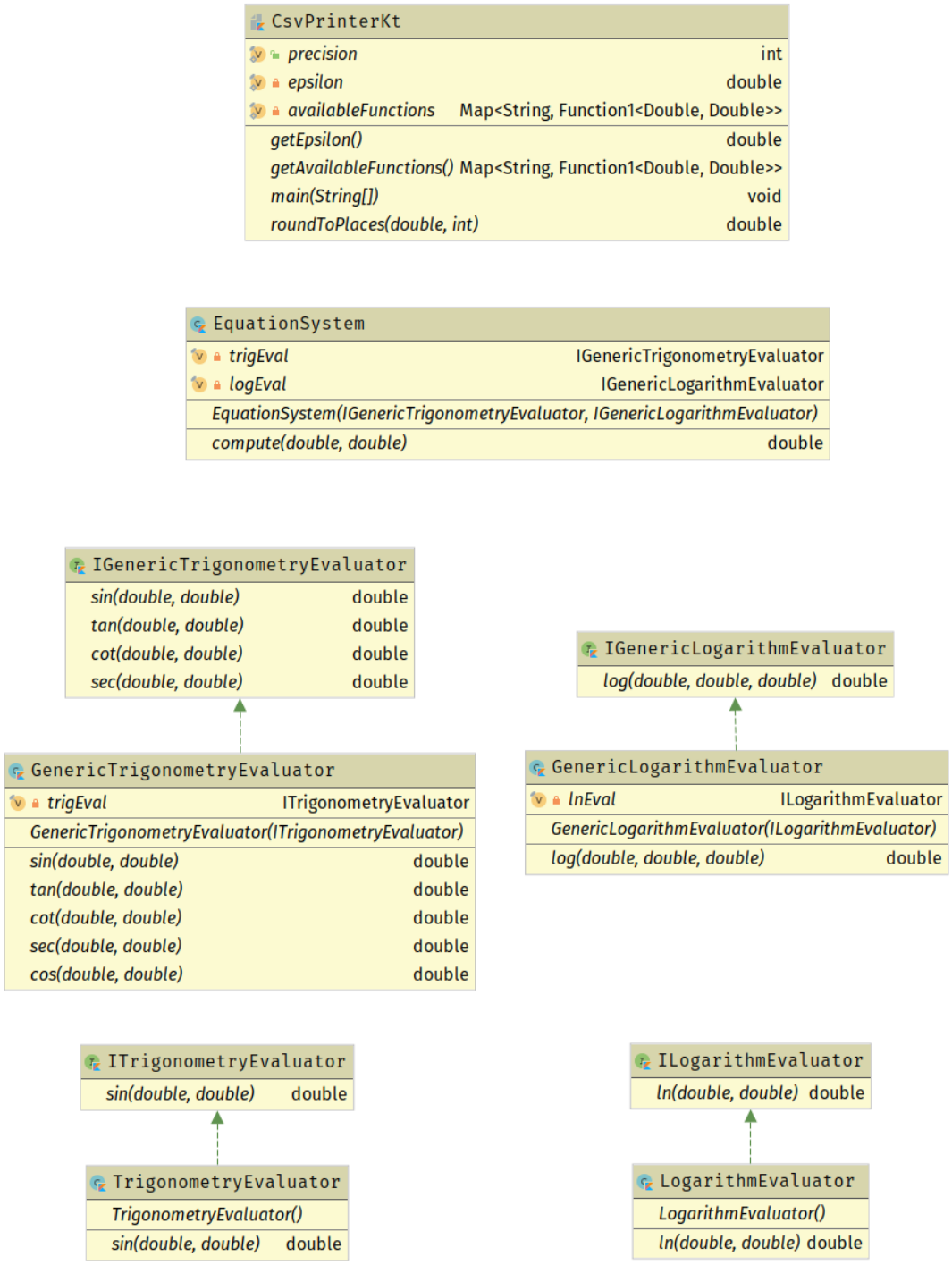

Рисунок 2. UML-диаграмма классов разработанного приложения

Описание тестового покрытия

При разработке была выбрана стратегия интеграции *сверху вниз*. Сначала был реализован модуль системы функций. Для него были написаны автоматизированные модульные тесты с заглушками тригонометрических и логарифмических функций. Затем реализован модуль тригонометрических функций и протестирован с заглушкой модуля аппроксимации синуса, и так далее.

Интеграция приложения производилась по одному модулю, т.е. при наличии нескольких зависимостей создавались интеграционные тесты для каждой из них по отдельности, и только затем для полной интеграции. Это позволило быстрее обнаружать ошибки и регрессии: если проблема связана только с одним модулем, то тесты других пройдут, что явно укажет на источник.

Параллельно разрабатывался модуль вывода значений в CSV файл. Для него была выбрана стратегия интеграции *end-to-end*, поскольку вывод каждой функции был связан с определенным модулем, и по завершении его разработки можно было сразу приступать к реализации сценария вывода его результатов.

Графики функций, используемые в процессе интеграции

Для тригонометрических функций использовался диапазон значений от $-\frac{7}{6}\pi$ до $\frac{7}{6}\pi$:

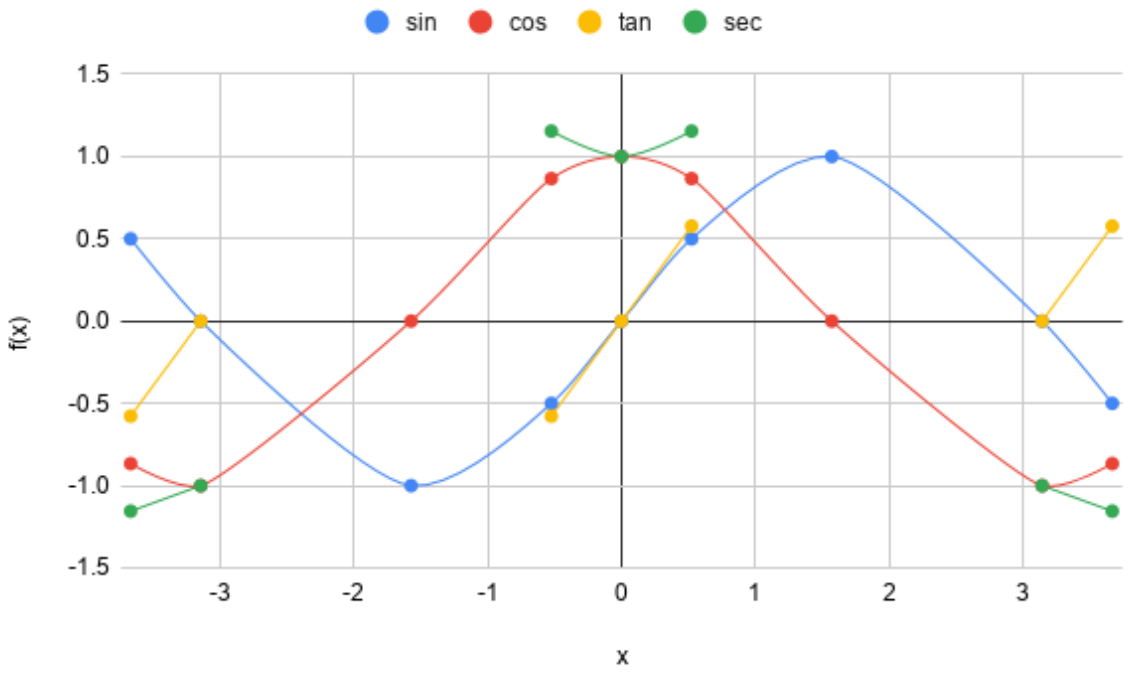

Рисунок 3. График тестируемых тригонометрических функций

Для логарифма проверялись значения $x < 1$ и $x \geqslant 1$:

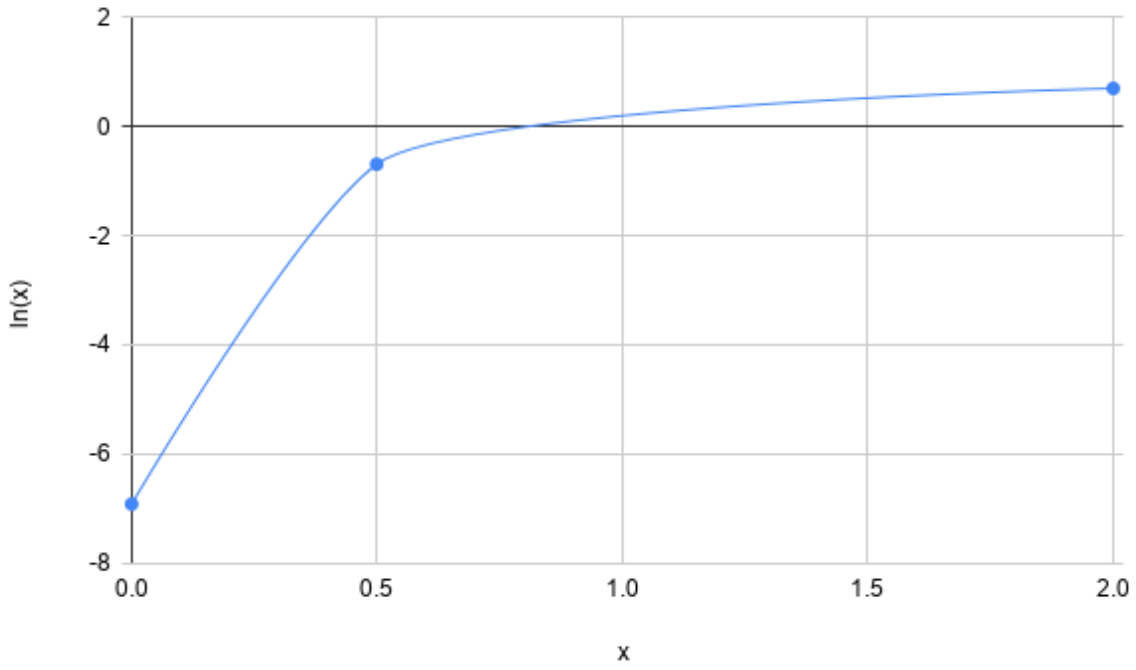

Рисунок 4. График тестируемой логарифмической функции

Для системы функций были взяты значения $x < 0$ и $x \geqslant 0$. В левой части выбирались точки экстремума функции, в правой рассматривались промежутки, где значение логарифма меньше и больше 0.

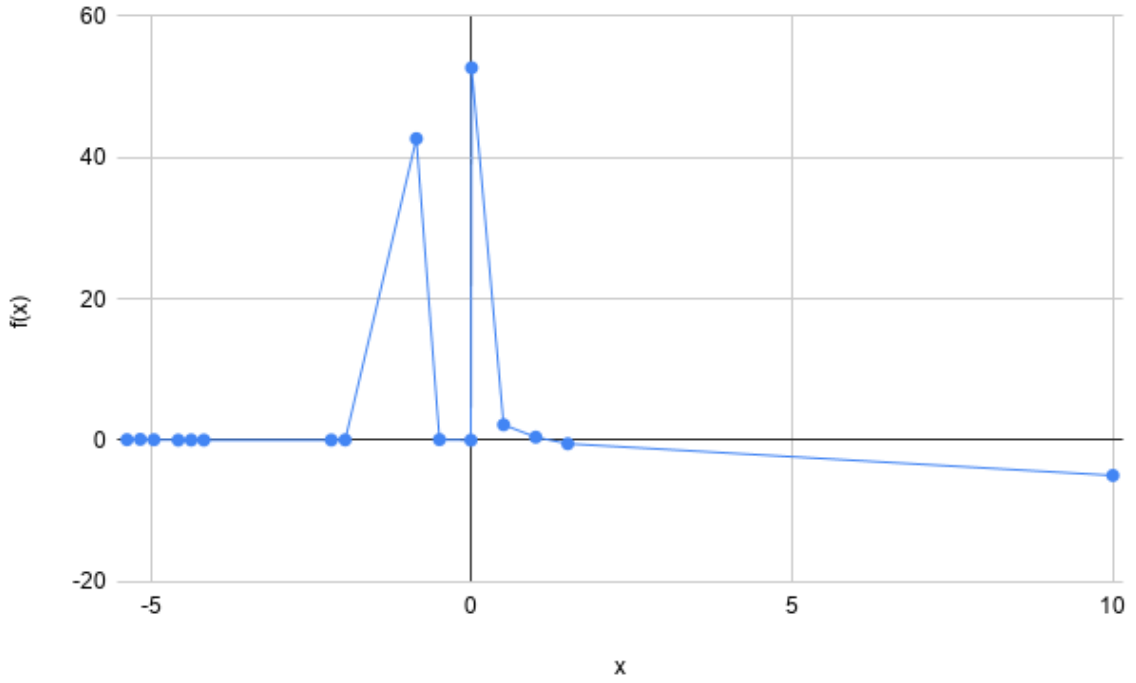

Рисунок 5. График тестируемой системы функций

Исходный код

`https://github.com/timlathy/itmo-fourth-year/tree/master/Software-Testing-7th-Term/Lab2`

Выводы

В ходе выполнения работы было рассмотрено интеграционное тестирование приложения подходом *сверху вниз* при помощи фреймворка JUnit 5 (модуль Jupiter). При написании тестов было изучено использование заглушек с применением библиотеки Mockito, а также загрузка табличных значений для параметризованных тестов из внешних CSV файлов при помощи аннотации `@CsvFileSource`.