Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Тестирование программного обеспечения **Лабораторная работа 1**

Вариант 39185

**Выполнили:**Козлов Виктор Николаевич

Мишанин Никита Алексеевич

**Группа:** P33111

**Преподаватель:**Яркеев Александр Сергеевич

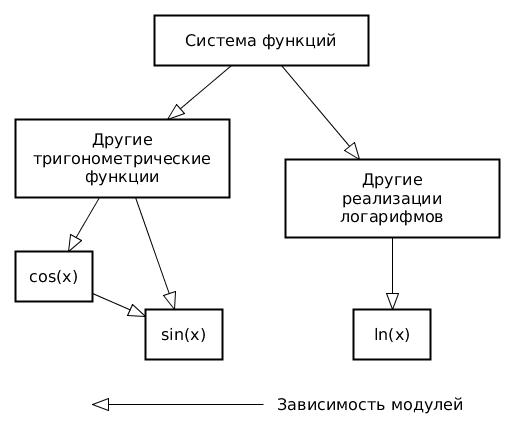
2022 г.

Санкт-Петербург

# Задание

## Изображение выглядит как текст Автоматически созданное описание

**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 сsv файл вида «X, Результаты модуля (X)», позволяющее произвольно менять шаг наращивания Х. Разделитель в файле csv можно использовать произвольный.

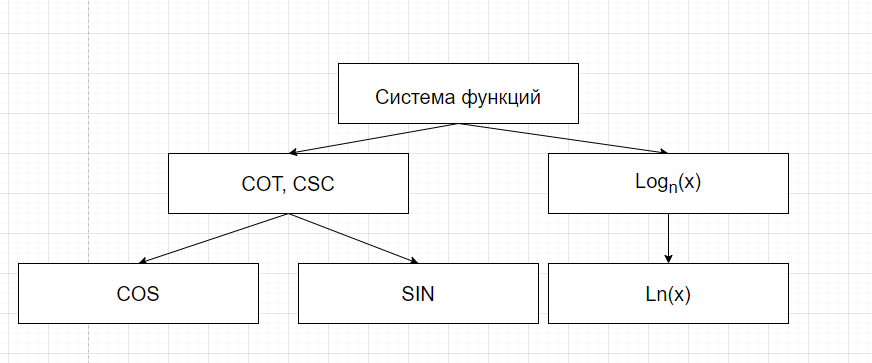
**Порядок выполнения работы:**

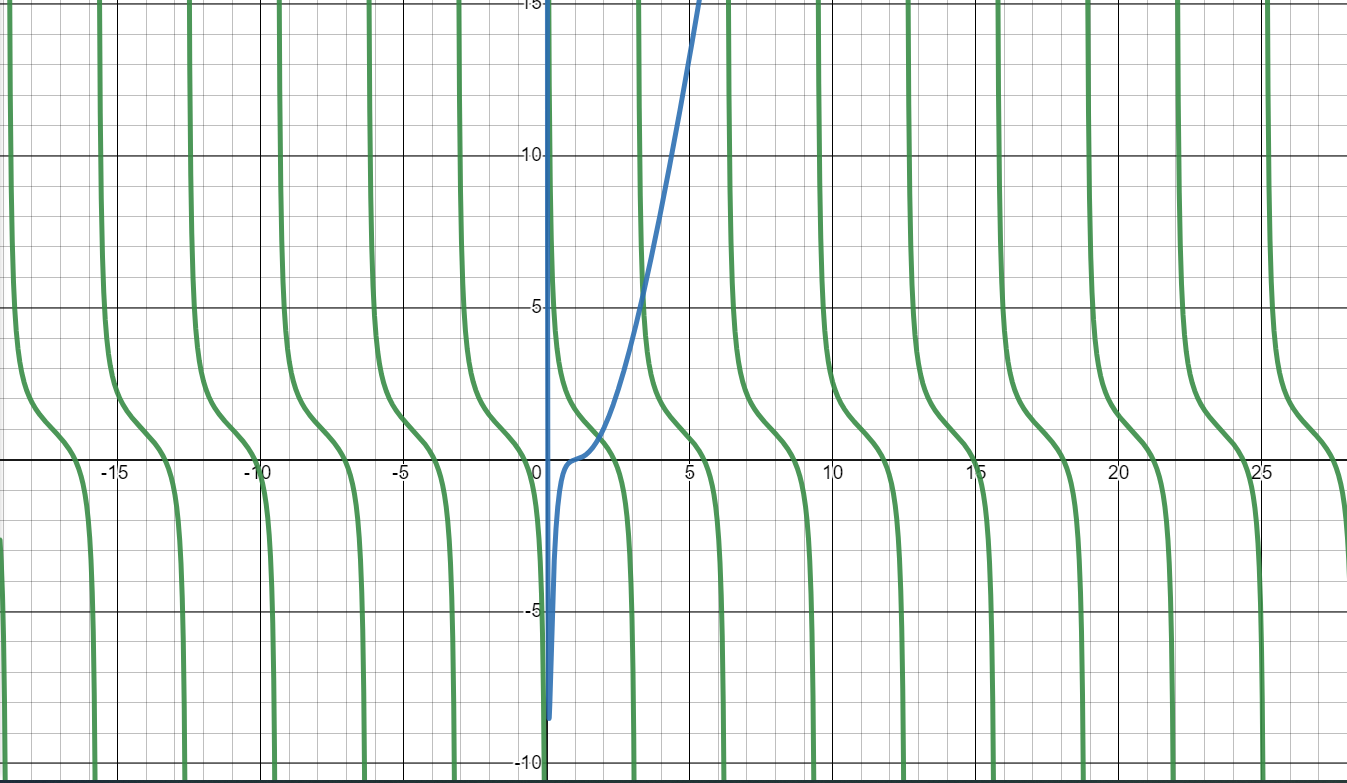
1. Разработать приложение, руководствуясь приведёнными выше правилами.
2. С помощью JUNIT4 разработать тестовое покрытие системы функций, проведя анализ эквивалентности и учитывая особенности системы функций. Для анализа особенностей системы функций и составляющих ее частей можно использовать сайт <https://www.wolframalpha.com/>.
3. Собрать приложение, состоящее из заглушек. Провести интеграцию приложения по 1 модулю, с обоснованием стратегии интеграции, проведением интеграционных тестов и контролем тестового покрытия системы функций.

**Отчёт по работе должен содержать:**

1. Текст задания, систему функций.
2. UML-диаграмму классов разработанного приложения.
3. Описание тестового покрытия с обоснованием его выбора.
4. Графики, построенные csv-выгрузкам, полученным в процессе интеграции приложения.
5. Выводы по работе.

Система функций:

  
  
UML-диаграмма классов  
Изображение выглядит как текст, электроника, цепь

Автоматически созданное описание  
  
  
  
  
Описание тестового покрытия с обоснованием его выбора  
  
  
  
  
Так как выбор функции зависит от x, то мы проверяем точку x=0, а также некоторое количество точек x>0 и x<0.

При x>0 выделим точку в промежутке (0, 0.83], так как там график убывает, так же одну точку от (0.83, 1), в x = 1 будет Nan, так как из-за высчитывания LogN возникает деление на 0. А так же в промежутке (0, +inf).

При x<0 функция повторяется. Мы можем захватить отрезок [-2pi; 0] в котором отражена вся повторяющаяся часть функции.  
  
Исходный код:  
  
<https://github.com/vitos4chitos/TPO2>  
  
  
  
Вывод:  
  
В ходе выполнения лабораторной работы, мы написали систему функций и протестировали её с помощью интеграционного тестирования с использованием заглушек и библиотеки Mockito. Главная сложность при проведении интеграционного тестирования – подбор возвращаемых значений у вызываемых модулей.  Это требует глубокого понимания кода и внушительного количества времени, с которым мы взаимодействуем, даже несмотря на то, что по факту сами настраиваем зависимые модули в виде заглушек.

# 

# Выполнение

1)

public class FunctionDecomposition {  
  
 private double results = 0.;  
  
 public Double tryFunctionDecomposition(Double x, final Long BOUND) {  
 for (long k = 0; k <= BOUND; k++) {  
 double a = (power(k) \* power(x, 2 \* k)) / ((fac(2 \* k)));  
 results += a;  
 }  
 return results;  
 }  
  
 public Double power(Double n, Long degree) {  
 Double result = 1.;  
 for (int i = 1; i <= degree; i++)  
 result \*= n;  
 return result;  
 }  
  
 public Long power(Long degree) {  
 long result = 1L;  
 for (int i = 1; i <= degree; i++)  
 result \*= -1;  
 return result;  
 }  
  
 public Long fac(Long arg) {  
 if (arg == 1 || arg == 0)  
 return 1L;  
 else  
 return fac(arg - 1) \* arg;  
 }  
  
 public Boolean checkInputArgs(String x, String k) {  
 try {  
 double x\_check = Double.*parseDouble*(x);  
 long k\_check = Long.*parseLong*(k);  
  
 } catch (ClassCastException | NumberFormatException e) {  
 return false;  
 }  
 return true;  
 }  
}

import org.junit.jupiter.api.Test;  
  
import static java.lang.Math.*PI*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class FunctionDecompositionTest {  
  
 private final funсtion.FunctionDecomposition moduleFunction = new funсtion.FunctionDecomposition();  
  
 @Test  
 void testModulePowerForOne() {  
 *assertEquals*(-1, moduleFunction.power(5L));  
 *assertEquals*(1, moduleFunction.power(4L));  
 *assertEquals*(1, moduleFunction.power(0L));  
 }  
  
 @Test  
 void testModulePowerForNumbers() {  
 *assertEquals*(8., moduleFunction.power(2., 3L));  
 *assertEquals*(81., moduleFunction.power(3., 4L));  
 *assertEquals*(1., moduleFunction.power(1., 0L));  
 *assertEquals*(0., moduleFunction.power(0., 5L));  
 *assertEquals*(33\_232\_930\_569\_601., moduleFunction.power(7., 16L));  
 }  
  
 @Test  
 void testModuleFactorial() {  
 *assertEquals*(120L, moduleFunction.fac(5L));  
 *assertEquals*(1L, moduleFunction.fac(1L));  
 *assertEquals*(3\_628\_800L, moduleFunction.fac(10L));  
 *assertEquals*(2\_432\_902\_008\_176\_640\_000L, moduleFunction.fac(20L));  
 }  
  
 @Test  
 void testModuleInputArgs() {  
 *assertEquals*(false, moduleFunction.checkInputArgs("3523 gf-232!", "3"));  
 *assertEquals*(false, moduleFunction.checkInputArgs("3", "-=1111"));  
 *assertEquals*(false, moduleFunction.checkInputArgs("1`=23-", "--1"));  
  
 String boundLong = String.*valueOf*(Long.*MAX\_VALUE*);  
 String boundDouble = String.*valueOf*(Double.*MAX\_VALUE*);  
 *assertEquals*(true, moduleFunction.checkInputArgs(boundDouble, boundLong));  
  
 *assertEquals*(true, moduleFunction.checkInputArgs("0", "9999999"));  
 *assertEquals*(true, moduleFunction.checkInputArgs("-999999", "0"));  
 }  
  
 @Test  
 void testModuleDecompositionFunctionThreeDegrees() {  
 final double x = 3. \* (*PI* / 180);  
 double results = moduleFunction.tryFunctionDecomposition(x, 3L);  
 double cos\_x = Math.*cos*(x);  
 String result = String.*format*("%5f",results);  
 String cos\_x\_r = String.*format*("%5f",cos\_x);  
 *assertEquals*(cos\_x\_r, result);  
 }  
  
 @Test  
 void testModuleDecompositionFunctionNinetyDegrees() {  
 final double x = 90. \* (*PI* / 180);  
 double results = moduleFunction.tryFunctionDecomposition(x, 6L);  
 double cos\_x = Math.*cos*(x);  
 String result = String.*format*("%5f",results);  
 String cos\_x\_r = String.*format*("%5f",cos\_x);  
 *assertEquals*(cos\_x\_r, result);  
 }  
  
 @Test  
 void testModuleDecompositionFunctionNinetyZeroDegrees() {  
 final double x = 0. \* (*PI* / 180);  
 double results = moduleFunction.tryFunctionDecomposition(x, 7L);  
 double cos\_x = Math.*cos*(x);  
 String result = String.*format*("%5f",results);  
 String cos\_x\_r = String.*format*("%5f",cos\_x);  
 *assertEquals*(cos\_x\_r, result);  
 }  
  
 @Test  
 void testModuleDecompositionFunctionNinetyFortyFiveDegrees() {  
 final double x = 45. \* (*PI* / 180);  
 double results = moduleFunction.tryFunctionDecomposition(x, 5L);  
 double cos\_x = Math.*cos*(x);  
 String result = String.*format*("%5f",results);  
 String cos\_x\_r = String.*format*("%5f",cos\_x);  
 *assertEquals*(cos\_x\_r, result);  
 }  
  
 @Test  
 void testModuleDecompositionThirtyDegrees() {  
 final double x = 30. \* (*PI* / 180);  
 double results = moduleFunction.tryFunctionDecomposition(x, 3L);  
 double cos\_x = Math.*cos*(x);  
 String result = String.*format*("%5f",results);  
 String cos\_x\_r = String.*format*("%5f",cos\_x);  
 *assertEquals*(cos\_x\_r, result);  
 }  
}

Написанный модуль реализует алгоритм разложения тригонометрической функции в ряд Тейлора до передаваемого аргумента. Модуль использует несколько важных функций для реализации степени, факториала и проверки входных значений. Тестирование модуля осуществлено посредствам JUNIT5, для каждой функции модуля и для основного модуля разработан комплекс тестов, который полностью покрывает код.

Изображение выглядит как текст

Автоматически созданное описание

2)

import java.util.Arrays;  
  
public class HashMap {  
 private int FREE = Integer.*MIN\_VALUE*;  
 private int size;  
 private int[] keys;  
 private int[] values;  
  
 /\* Конструктор \*/  
 public HashMap(int size) {  
 this.size = size;  
 keys = new int [this.size];  
 values = new int [this.size];  
 Arrays.*fill*(keys, FREE);  
 }  
  
 /\* Добавляет пару в множество \*/  
 public void put(int x, int y) {  
 for (int i = index(hash(x)); ; i++) {  
 if (i == size) i = 0;  
 if (keys[i] == FREE)  
 keys[i] = x;  
 if (keys[i] == x) {  
 values[i] = y;  
 return;  
 }  
 }  
 }  
 /\* Извлекает значение \*/  
 public int get(int x) {  
 for (int i = index(hash(x)); ; i++) {  
 if (i == size) i = 0;  
 if (keys[i] == FREE) throw new RuntimeException("No such key!");  
 if (keys[i] == x) return values[i];  
 }  
 }

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class HashMapTest {  
  
 @Test  
 void testHashMapSize(){  
 HashMap hm = new HashMap(3);  
 *assertEquals*(hm.getSize(), 3);  
 }  
  
 @Test  
 void testHashMapAdd(){  
 HashMap hm = new HashMap(3);  
 hm.put(1, 1);  
 *assertTrue*(hm.containsKey(1));  
 *assertEquals*(hm.get(1), 1);  
 *assertFalse*(hm.containsKey(2));  
 }  
  
 @Test  
 void testHashMapDelete(){  
 HashMap hm = new HashMap(3);  
 hm.put(1, 1);  
 hm.deleteKey(1);  
 *assertFalse*(hm.containsKey(1));  
 }  
  
 @Test  
 void testHashMapReAdd(){  
 HashMap hm = new HashMap(3);  
 hm.put(1, 1);  
 hm.put(1, 2);  
 *assertTrue*(hm.containsKey(1));  
 *assertEquals*(hm.get(1), 2);  
 }  
  
  
}

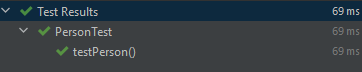
## Изображение выглядит как текст Автоматически созданное описание

## Unit-тестирование объектов

Изображение выглядит как текст, парковка, снимок экрана

Автоматически созданное описание

import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class PersonTest {  
 @Test  
 public void testPerson() {  
 Person person = new Person("Василий", "Иванов", Feelings.*Comfortable*);  
 *assertEquals*(person.getName(), "Василий");  
 *assertEquals*(person.getSurname(), "Иванов");  
 *assertEquals*(person.getFeelings(), Feelings.*Comfortable*);  
 }  
}



import org.junit.jupiter.api.Test;  
  
import java.util.ArrayList;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class SpaceShipTest {  
 @Test  
 public void testOwn\_free\_willSpaceShip() {  
 ArrayList<Person> crew = new ArrayList<Person>();  
 crew.add(new Person("Василий", "Иванов", Feelings.*Comfortable*));  
 crew.add(new Person("Иван", "Васильев", Feelings.*Uncomfortable*));  
 SpaceShip spaceShip = new SpaceShip("Золотое Сердце", crew, Principle.*Own\_free\_will*);  
 *assertEquals*(spaceShip.getName(), "Золотое Сердце");  
 *assertEquals*(spaceShip.getCrew(), crew);  
 *assertEquals*(spaceShip.getPrinciple(), Principle.*Own\_free\_will*);  
 *assertEquals*(spaceShip.getLaws(), Laws.*Human*);  
 }  
  
 @Test  
 public void testCoincidenceSpaceShip() {  
 ArrayList<Person> crew = new ArrayList<Person>();  
 crew.add(new Person("Василий", "Иванов", Feelings.*Comfortable*));  
 crew.add(new Person("Иван", "Васильев", Feelings.*Uncomfortable*));  
 SpaceShip spaceShip = new SpaceShip("Золотое Сердце", crew, Principle.*Coincidence*);  
 *assertEquals*(spaceShip.getName(), "Золотое Сердце");  
 *assertEquals*(spaceShip.getCrew(), crew);  
 *assertEquals*(spaceShip.getPrinciple(), Principle.*Coincidence*);  
 *assertEquals*(spaceShip.getLaws(), Laws.*Human*);  
 }  
  
 @Test  
 public void testPhysicalSpaceShip() {  
 ArrayList<Person> crew = new ArrayList<Person>();  
 crew.add(new Person("Василий", "Иванов", Feelings.*Comfortable*));  
 crew.add(new Person("Иван", "Васильев", Feelings.*Uncomfortable*));  
 SpaceShip spaceShip = new SpaceShip("Золотое Сердце", crew, Principle.*Physical*);  
 *assertEquals*(spaceShip.getName(), "Золотое Сердце");  
 *assertEquals*(spaceShip.getCrew().get(0).getFeelings(), Feelings.*Uncomfortable*);  
 *assertEquals*(spaceShip.getCrew().get(1).getFeelings(), Feelings.*Uncomfortable*);  
 *assertEquals*(spaceShip.getPrinciple(), Principle.*Physical*);  
 *assertEquals*(spaceShip.getLaws(), Laws.*Molecular*);  
 }  
  
 @Test  
 public void testUncomfortableOwn\_free\_willSpaceShip(){  
 ArrayList<Person> crew = new ArrayList<Person>();  
 crew.add(new Person("Василий", "Иванов", Feelings.*Uncomfortable*));  
 crew.add(new Person("Иван", "Васильев", Feelings.*Uncomfortable*));  
 SpaceShip spaceShip = new SpaceShip("Золотое Сердце", crew, Principle.*Own\_free\_will*);  
 *assertEquals*(spaceShip.getName(), "Золотое Сердце");  
 *assertEquals*(spaceShip.getCrew(), crew);  
 *assertEquals*(spaceShip.getPrinciple(), Principle.*Own\_free\_will*);  
 *assertEquals*(spaceShip.getLaws(), Laws.*Molecular*);  
 }  
  
 @Test  
 public void testUncomfortableCoincidenceSpaceShip(){  
 ArrayList<Person> crew = new ArrayList<Person>();  
 crew.add(new Person("Василий", "Иванов", Feelings.*Uncomfortable*));  
 crew.add(new Person("Иван", "Васильев", Feelings.*Uncomfortable*));  
 SpaceShip spaceShip = new SpaceShip("Золотое Сердце", crew, Principle.*Coincidence*);  
 *assertEquals*(spaceShip.getName(), "Золотое Сердце");  
 *assertEquals*(spaceShip.getCrew(), crew);  
 *assertEquals*(spaceShip.getPrinciple(), Principle.*Coincidence*);  
 *assertEquals*(spaceShip.getLaws(), Laws.*Molecular*);  
 }  
  
 @Test  
 public void testUncomfortableSpaceShip() {  
 ArrayList<Person> crew = new ArrayList<Person>();  
 crew.add(new Person("Василий", "Иванов", Feelings.*Uncomfortable*));  
 crew.add(new Person("Иван", "Васильев", Feelings.*Uncomfortable*));  
 SpaceShip spaceShip = new SpaceShip("Золотое Сердце", crew, Principle.*Physical*);  
 *assertEquals*(spaceShip.getName(), "Золотое Сердце");  
 *assertEquals*(spaceShip.getCrew().get(0).getFeelings(), Feelings.*Uncomfortable*);  
 *assertEquals*(spaceShip.getCrew().get(1).getFeelings(), Feelings.*Uncomfortable*);  
 *assertEquals*(spaceShip.getPrinciple(), Principle.*Physical*);  
 *assertEquals*(spaceShip.getLaws(), Laws.*Molecular*);  
 }  
  
}

Изображение выглядит как текст, снимок экрана, экран

Автоматически созданное описание

# Вывод

В процессе выполнения лабораторной работы мы научился писал юнит-тесты (метод черного ящика) для разработанных классов. Сложность заключается в необходимости проявить гибкость мышления при проверке ожидаемого поведения, т.е. придумать альтернативный способ достижения результата, либо вручную формировать как исходные, так и ожидаемые данные для сравнения. Важно отметить, что достижение 100%-го покрытия очень сложно, поэтому необходимо проверять лишь «избранные» входные данные.