**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Операционные системы и среды»

на тему: **«Распределенное компонентное приложение подготовки данных для обработки и отображения поверхности»**

Исполнитель: студент гр. ИП-31

Федосов О.В.

Руководитель: преподаватель Косинов Г.П.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2020

СОДЕРЖАНИЕ

Введение

[1 Обзор существующих методов решения задачи](#_Toc7027793)

[1.1 Решение интегралов](#_Toc7027793)

[1.2 Выбор языка программирования для разработки приложения](#_Toc7027792)

[1.3 Построение поверхности](#_Toc7027792)

[1.4 Инструменты для распараллеливония вычислений на потоки](#_Toc7027792)

[2 Алгоритмический анализ](#_Toc7027795)

[2.1 Постановка задачи](#_Toc7027792)

[2.2 Построения функциональной модели](#_Toc7027795)

[3 Разработка программного кода](#_Toc7027796)

[3.1 Описания классов](#_Toc7027792)

[3.2 Разработка графического интерфейса](#_Toc7027796)

[4 Тестирование и верификация](#_Toc7027792)

[Заключение](#_Toc7027792)

[Список использованных источников](#_Toc7027792)

[Приложения А - Листинг сценария для математического пакета](#_Toc7027792)

[Приложения B – Диаграмма компонентов](#_Toc7027792)

[Приложения C - Листинг програмного кода](#_Toc7027792)

ВВЕДЕНИЕ

1. **ОБЗОР МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ**

## **1.1 Решение интегралов**

Интеграл — одно из важнейших понятий математического анализа, которое возникает при решении задач о нахождении площади под кривой, пройденного пути при неравномерном движении, массы неоднородного тела, и тому подобных, а также в задаче о восстановлении функции по её производной (неопределённый интеграл).

Двойные интегралы – это обобщение понятия определённого интеграла для функции двух переменных, заданной как

Двойной интеграл записывается как показано на формуле (1.1):

|  |  |
| --- | --- |
|  | (1.1) |

Здесь D – плоская фигура, ограниченная линиями, выражения которых (равенства) даны в задании вычисления двойного интеграла. Слева и справа – равенствами, в которых слева переменная x, а сверху и снизу – равенствами, в которых слева переменная y. Это место и далее – одно из важнейших для понимания техники вычисления двойного интеграла.

Вычислить двойной интеграл – значит найти число, равное площади упомянутой фигуры D.

Метод трапеций – метод численного интегрирования функции одной переменной, заключающийся в замене на каждом элементарном отрезке подынтегральной функции на многочлен первой степени, то есть линейную функцию. Площадь под графиком функции аппроксимируется прямоугольными трапециями. Алгебраический порядок точности равен 1.

Если отрезок является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле (1.2):

|  |  |
| --- | --- |
|  | (1.2) |

Это простое применение формулы для площади трапеции — произведение полусуммы оснований, которыми в данном случае являются значения функции в крайних точках отрезка, на высоту (длину отрезка интегрирования). Погрешность аппроксимации можно оценить через максимум второй производной (1.3):

|  |  |
| --- | --- |
|  | (1.3) |

Аппроксимация функции линейной зависимостью при интегрировании методом трапеций показана на рисунке 1.1.

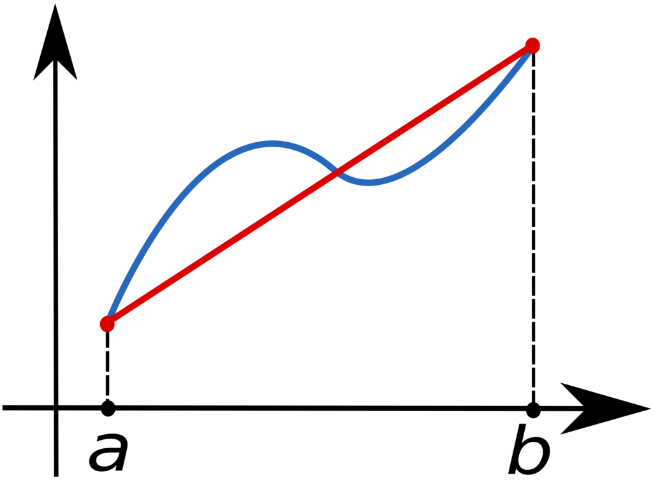


Рисунок 1.1 – Аппроксимация функции линейной зависимостью при интегрировании методом трапеций

* 1. **Выбор языка программирования для разработки** приложения

**1.2.1** Выбор язык программирования.

В процессе анализа поставленной задачи было решено разработать приложение, реализующее все требования к курсовой работе. Для разработки приложения использовался язык программирования Python. Пользовательский интерфейс был создан с использование фреймворка Django.

**1.2.2** Python.

Python— высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Django — свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Проект поддерживается организацией Django Software Foundation.

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других Один из основных принципов фреймворка — DRY

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

* 1. **Построение поверхности**

**1.3.1** Поверхность.

Поверхность в геометрии и топологии — двумерное топологическое многообразие. Наиболее известными примерами поверхностей являются границы геометрических тел в обычном трёхмерном евклидовом пространстве. С другой стороны, существуют поверхности (например, бутылка Клейна), которые нельзя вложить в трёхмерное евклидово пространство без привлечения сингулярности или самопересечения.

Концепция поверхности применяется в физике, инженерном деле, компьютерной графике и прочих областях при изучении физических объектов. Например, анализ аэродинамических качеств самолёта базируется на обтекании потоком воздуха его поверхности.

**1.3.2** Построение графиков в Python при помощи Matplotlib.

Визуализация данных — это большая часть работы специалистов в области data science. На ранних стадиях развития проекта часто необходимо выполнять разведочный анализ данных (РАД, Exploratory data analysis (EDA)), чтобы выявить закономерности, которые обнаруживают данные. Визуализация данных помогает представить большие и сложные наборы данных в простом и наглядном виде. На этапе окончания проекта важно суметь отчитаться о его результатах так, чтобы даже непрофессионалам, не обладающим техническими знаниями, всё стало ясно и понятно.

Matplotlib — это популярная библиотека для визуализации данных, написанная на языке Python. Хоть пользоваться ей очень просто, настройка данных, параметров, графиков и отрисовки для каждого нового проекта — занятие нудное и утомительное. В курсовом проекте необходимо будет построить поверхность и диаграмму.

Для построения поверхности используйте функцию plot\_surface(). Пример работы показан на рисунке 1.2.

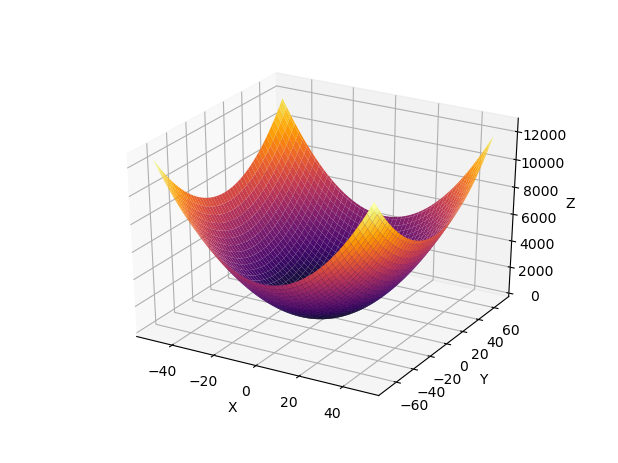


Рисунок 2.1 – Поверхность с использование библиотеки Matplotlib

* 1. **Инструменты для распараллеливания вычислений на потоки** 
     1. Параллелизм в Python.

### Многопроцессорная обработка – это использование двух или более процессорных блоков в одной компьютерной системе. Это лучший способ получить полный потенциал от нашего оборудования, используя полное количество процессорных ядер, доступных в нашей компьютерной системе.

### Многопоточная обработка – это способность ЦП управлять использованием операционной системы путем одновременного выполнения нескольких потоков. Основная идея многопоточности заключается в достижении параллелизма путем разделения процесса на несколько потоков.

При работе с параллельными приложениями в Python есть ограничение, называемое GIL (Global Interpreter Lock). GIL никогда не позволяет нам использовать несколько ядер CPU, и поэтому мы можем сказать, что в Python нет настоящих потоков. GIL – мьютекс – блокировка взаимного исключения, которая делает вещи безопасными. Другими словами, мы можем сказать, что GIL препятствует параллельному выполнению кода Python несколькими потоками. Блокировка может удерживаться только одним потоком за раз, и если мы хотим выполнить поток, он должен сначала получить блокировку.

Используя многопроцессорность, мы можем эффективно обойти ограничение, вызванное GIL:

1. Используя многопроцессорность, мы используем возможности нескольких процессов и, следовательно, мы используем несколько экземпляров GIL.
2. В связи с этим нет ограничений на выполнение байт-кода одного потока в наших программах одновременно.

Используя многопроцессорность, мы используем возможности нескольких процессов и, следовательно, мы используем несколько экземпляров GIL. В связи с этим нет ограничений на выполнение байт-кода одного потока в наших программах одновременно.

**1.4.2.** Использование параллелизма в курсовом проекте.

Для распараллеливания в курсовом проекте будет использоваться библиотека multiprocessing. Библиотека предоставляет возможность создание нескольких процессов одновременно, что позволяет выполнять вычисление параллельно. В данной курсовой работе будет использоваться Spawn.

Spawn означает начать что-то новое. Следовательно, порождение процесса означает создание нового процесса родительским процессом. Родительский процесс продолжает свое выполнение асинхронно или ожидает, пока дочерний процесс не завершит свое выполнение. Для создание нового процесса необходимо выполнить следующие шаги:

1. Импорт многопроцессорного модуля;
2. Создание объекта процесса;
3. Запуск процесса с помощью вызова метода *start();*
4. Дождитесь, пока процесс завершит свою работу, и выйдите, вызвав метод *join().*
5. РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ МОДЕЛИ ПРИЛОЖЕНИЯ

**2.1 Постановка задачи**

Необходимо разработать распределенное компонентное приложение подготовки данных для обработки и отображения поверхности.

Методом численного интегрирования выбрать метод Трапеции.

Вид поверхности:

Для данной поверхности 𝑋𝑛, 𝑋𝑘, 𝑌𝑛, 𝑌𝑘 принять -50, 50, -70, 70 соответственно.

Значения коэффициентов:

A = 3, B = 1, C = 5, D = -5.

В программе предусмотреть возможность отображения поверхности, расчет площади заданной поверхности в заданных пределах, анализ влияния распараллеливания выполнения расчета площади на время.

Проверить результат нахождения площади и построения поверхности в математическом пакете, используя стандартные функции, и сравнить результаты.

* 1. **Построение функциональной модели**

Для выполнения поставленной задачи, приложение должно реализовывать следующий функционал:

1. расчет площади заданной поверхности заданным численным методом, согласно варианту;
2. построение графика поверхности;
3. анализ влияния распараллеливания вычисления площади поверхности заданным численным методом на время вычисления и относительную погрешность относительно аналитического значения.

Структурная схема приложения показана на рисунке 2.1.

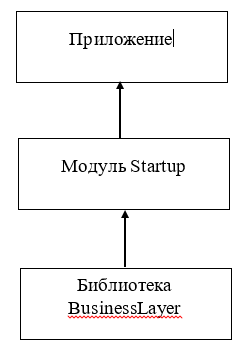


Рисунок 2.1 – Структурная схема приложения

Модули комплексного приложения для решения задачи:

1. главное приложение (SurfaceTreatment);
2. библиотека (BusinessLayer);
3. модуль (Startup).

Библиотека BusinessLayer выполняет основную бизнес-логику. Она состоит из двух классов:

1. TrapezoidMethod: используется для вычисления двойного интеграла, распараллеливания и построения поверхности и диаграммы;
2. FileHelper: используется для записи данных в файл и чтения из файла.

Модуль Startup реализует весь пользовательский интерфейс и основной функционал нахождение и построения поверхности. В Startup подключен модуль BusinessLayer для работы с поверхностью.

Фреймворк Django позволяет создать приложение из одно или несколько вспомогательных приложений (модулей). Таким образом можно легко изменить приложение, подключив другой модуль или убрать один из модулей приложения (если их несколько).

Приложение будет использовать один модуль – Startup.

1. разработка Компонентов

**3.1 Описания классов**

Методы класса TrapezoidMethod показаны в таблице 3.1. Так как Python не является строго типизированным язык программирования, тип входящих параметров и тип возвращаемых значений не указывается.

Таблица 3.1 – Описание TrapezoidMethod

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *setParams* | Установка значений для параметров A, B, C, D |
| *setIntervals* | Установка значений для интервалов Xн, Xк, Yн, Yк |
| *execute* | Нахождение площади поверхности |
| *executeAnalysis* | Нахождения площади поверхности для анализа |
| *calcFormY* | Нахождения интеграла по Y |
| *calcFormX* | Нахождение интеграла по X |
| *getFunc* | Получение результат функции площади поверхности |
| *getFuncForY* | Получение производной по Y |
| *getFuncForX* | Получение производной по X |
| *getFuncForWrite* | Получение функции для записи в файл |
| *getMatrix* | Получение матрицы Z |
| *draw* | Отрисовка поверхности |
| *drawAnalysis* | Отрисовка диаграммы |
| *writeFile* | Запись в файл частичного результата (при использовании нескольких процессов) |

Параметры метода *setParams* показаны в таблице 3.2.

Таблица 3.2 – Описание параметров метода *setParams*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *a* | Параметр функции А |
| *b* | Параметр функции B |
| *c* | Параметр функции C |
| *d* | Параметр функции D |

Параметры метода *setIntervals* показаны в таблице 3.3.

Таблица 3.3 – Описание параметров метода *setIntervals*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *xs* | Начальное значение интервала Х |
| *xf* | Конечное значение интервала X |
| *ys* | Начальное значение интервала Y |
| *yf* | Конечное значение интервала Y |

Параметры метода *execute* показаны в таблице 3.4.

Таблица 3.4 – Описание параметров метода *execute*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *n* | Количество делений |
| *processNumber* | Количество процессов |

Параметры метода *executeAnalysis* показаны в таблице 3.5.

Таблица 3.5 – Описание параметров метода *executeAnalysis*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *n* | Количество делений |
| *processNumber* | Количество процессов |

Параметры метода *calcFormY* показаны в таблице 3.6.

Таблица 3.6 – Описание параметров метода *calcFormY*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *x* | Список точек для X |
| *ys* | Начальное значение Ys |
| *hy* | h для интеграла по Y |
| *hx* | h для интеграла по X |
| *n* | Количество делений |
| *num* | Переменная, в которой будет сохранен результат вычисления |

Параметры метода *calcFormX* показаны в таблице 3.7.

Таблица 3.7 – Описание параметров метода *calcFormX*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *x* | Список точек для X |
| *y* | Значение Y |
| *hx* | h для интеграла по X |

Параметры метода *getFunc* показаны в таблице 3.8.

Таблица 3.8 – Описание параметров метода *getFunc*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *x* | Список точек для X |
| *y* | Значение Y |

Параметры метода *getFuncForY* показаны в таблице 3.9.

Таблица 3.9 – Описание параметров метода *getFuncForY*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *y* | Значение Y |

Параметры метода *getFuncForX* показаны в таблице 3.10.

Таблица 3.10 – Описание параметров метода *getFuncForX*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *x* | Значение X |

Параметры метода *getFuncForWrite* показаны в таблице 3.11.

Таблица 3.11 – Описание параметров метода *getFuncForWrite*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *x* | Значение X |
| *y* | Значение Y |

Параметры метода *draw* показаны в таблице 3.12.

Таблица 3.12 – Описание параметров метода *draw*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *x* | Значение X |
| *y* | Значение Y |
| *z* | Значение Z |

Параметры метода *drawAnalysis* показаны в таблице 3.13.

Таблица 3.13 – Описание параметров метода *drawAnalysis*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *times* | Время |
| *procNumbers* | Количество процессов |

Параметры метода *writeFile* показаны в таблице 3.14.

Таблица 3.14 – Описание параметров метода *writeFile*

|  |  |
| --- | --- |
| Наименование методов | Описание |
| *result* | Результат для записи в файл |

* 1. **Разработка графического интерфейса**

Стартовая страница приложения показан на рисунке 3.1:

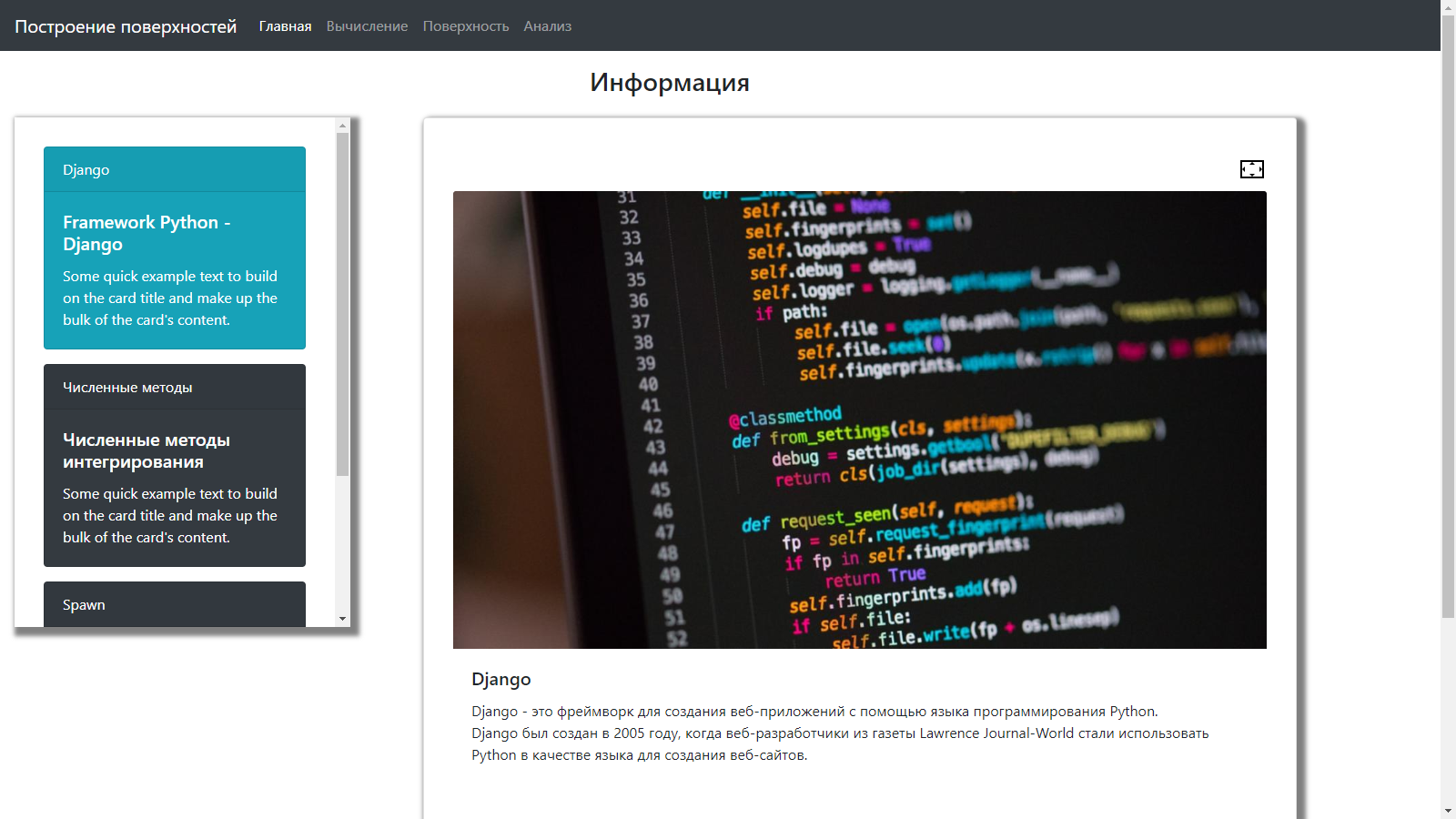


Рисунок 3.1 – Стартовая страница приложения

На стартовой странице отображена вся необходимая информация о курсовом проекте: описание фреймворка Django, численного метода, распараллеливания и др.

Для нахождения площади поверхности необходимо перейти на страницу Вычисление. Страница показана на рисунке 3.2.

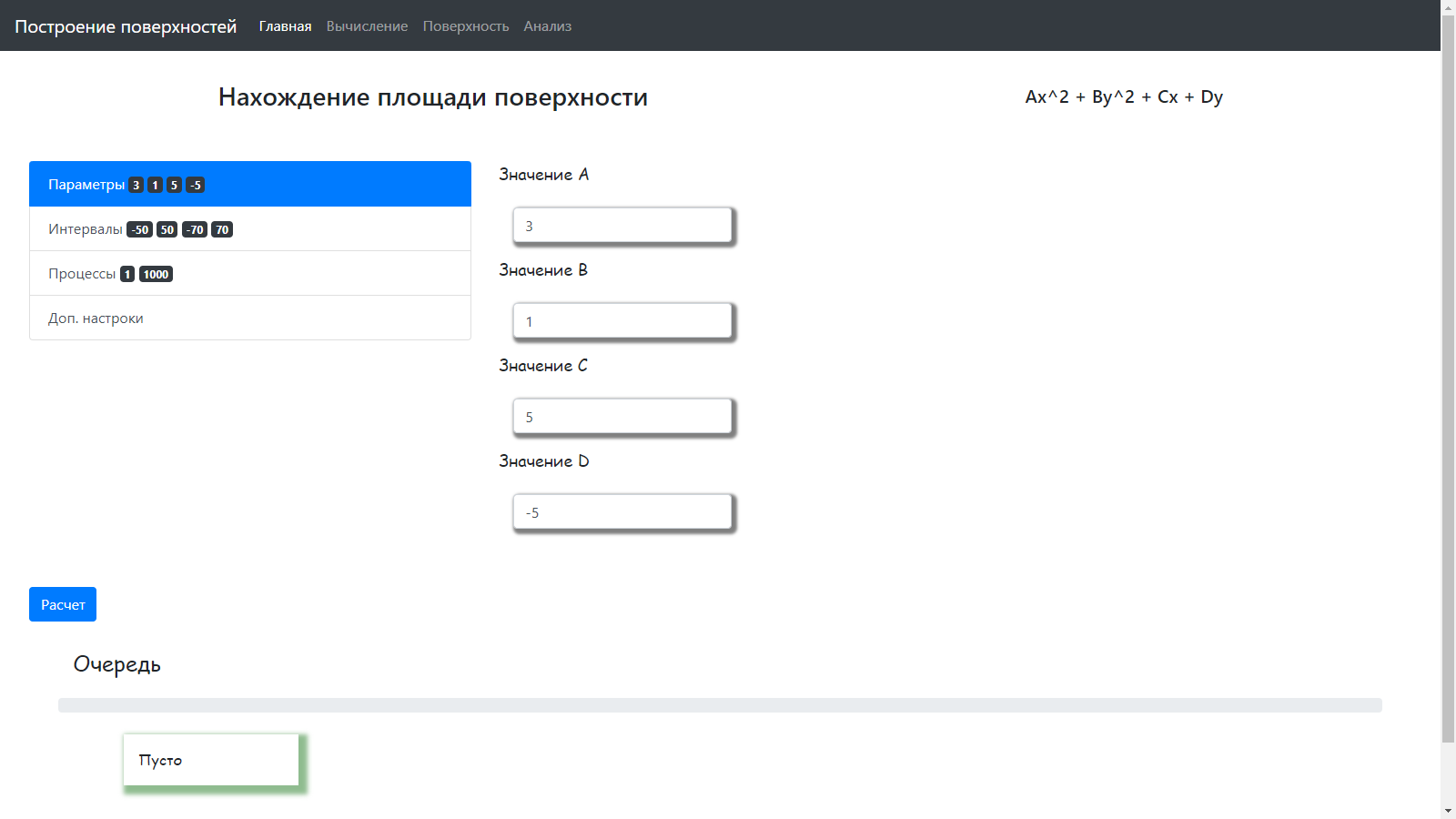


Рисунок 3.2 – Страница нахождения площади поверхности

На странице Вычисление показан функция для вычисления, ввод данных для расчета и очередь. Ввод данных разбит на несколько блоков:

1. Параметры: в данном блоке указываются параметры A, B, C, D для функции;
2. Интервалы: в данном блоке указываются интервалы для интеграла: начальное значение X, Y и конечное значение X, Y;
3. Процессы: в данном блоке указывается количество процессов (для параллельного вычисления) и количество делений;
4. Дополнительные настройки: в данном блоке указывается, необходимо ли сохранять данные для X, Y, Z, можно указать имена файлов для сохранения и отобразить точное вычисление.

Результат вычисления отображается в разделе Очередь. Пример вычисления показан на рисунке 3.3.

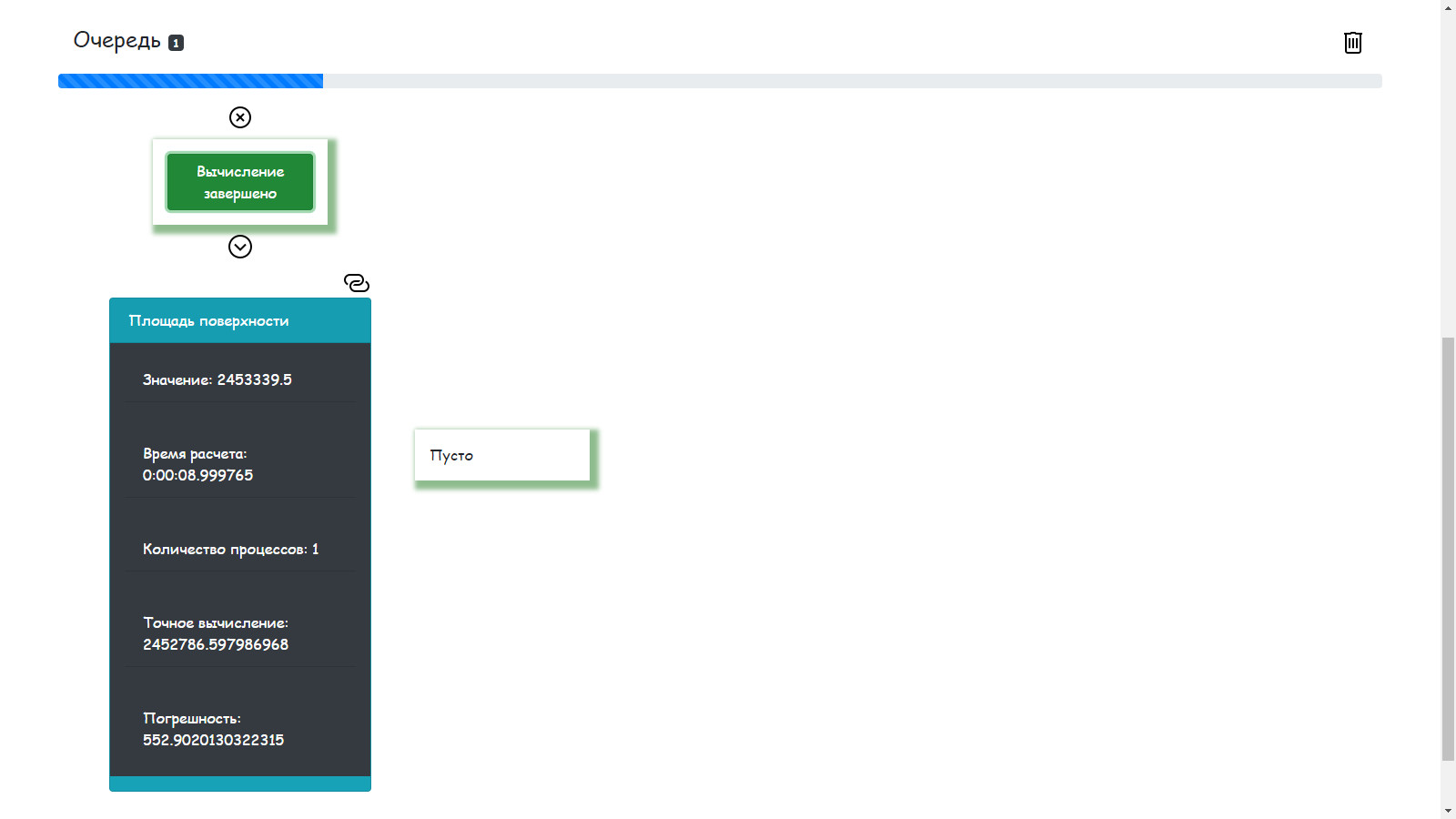


Рисунок 3.3 –Вывод результат вычисления

На странице Поверхность можно построить поверхность в Python. Страница показана на рисунке 3.4.

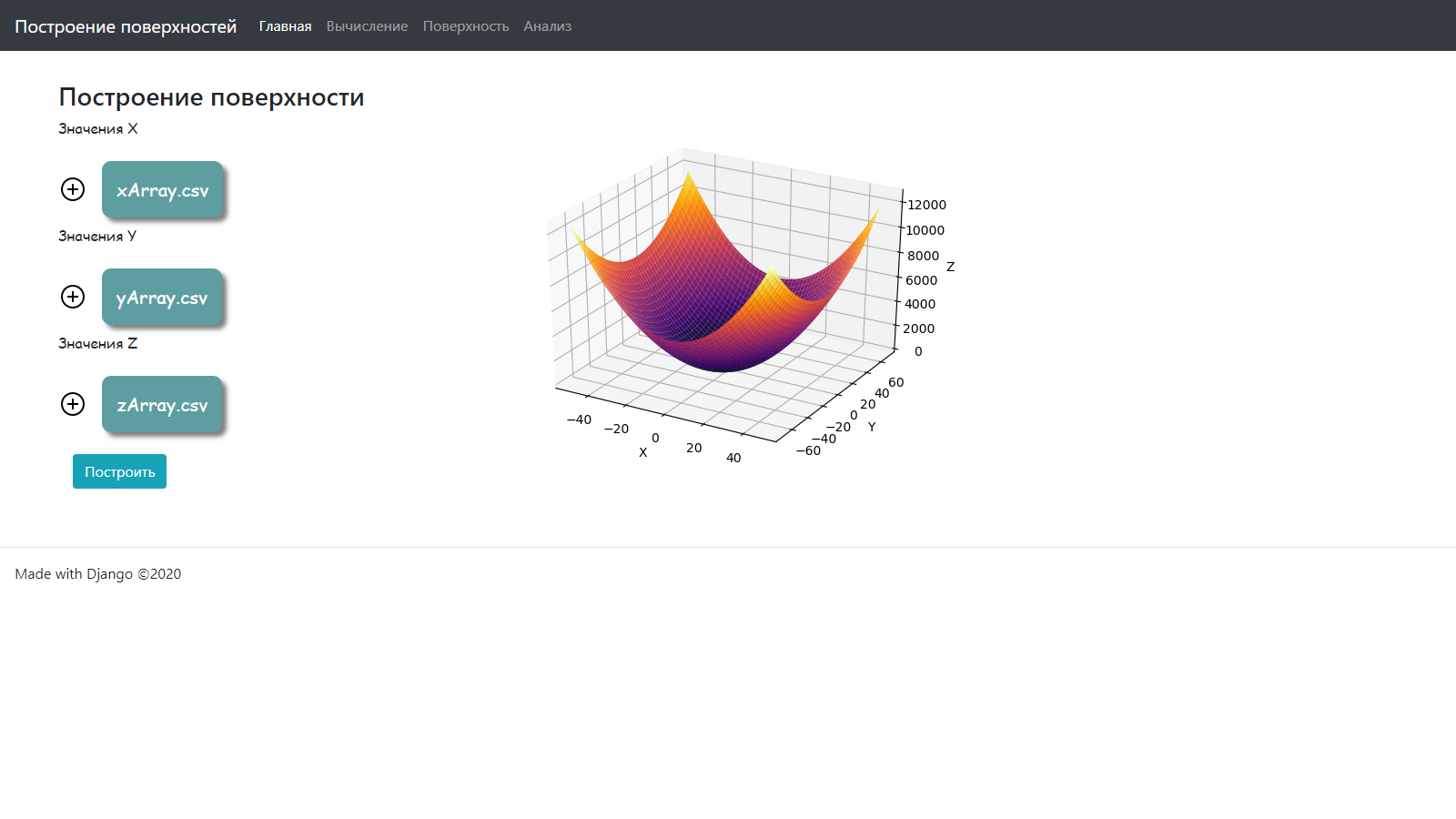


Рисунок 3.4 – Страница построения поверхности

После построения отображается поверхность в 3D-пространстве. Поверхность показана на рисунке 3.5.

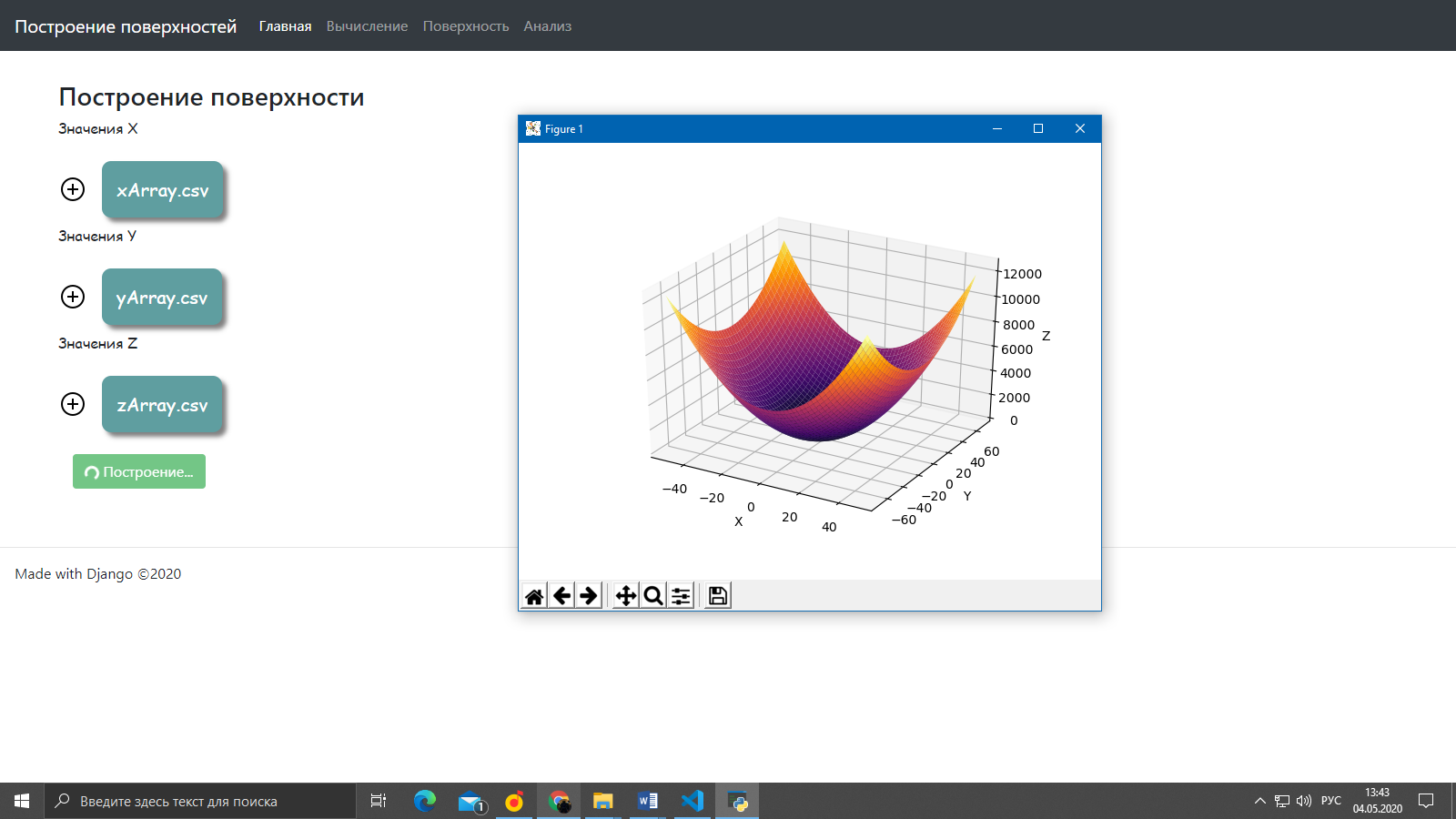


Рисунок 3.5 – Поверхность

На странице Анализ можно провести анализ зависимости время выполнения от количества запущенных процессов для вычисления. Страница показана на рисунке 3.6.

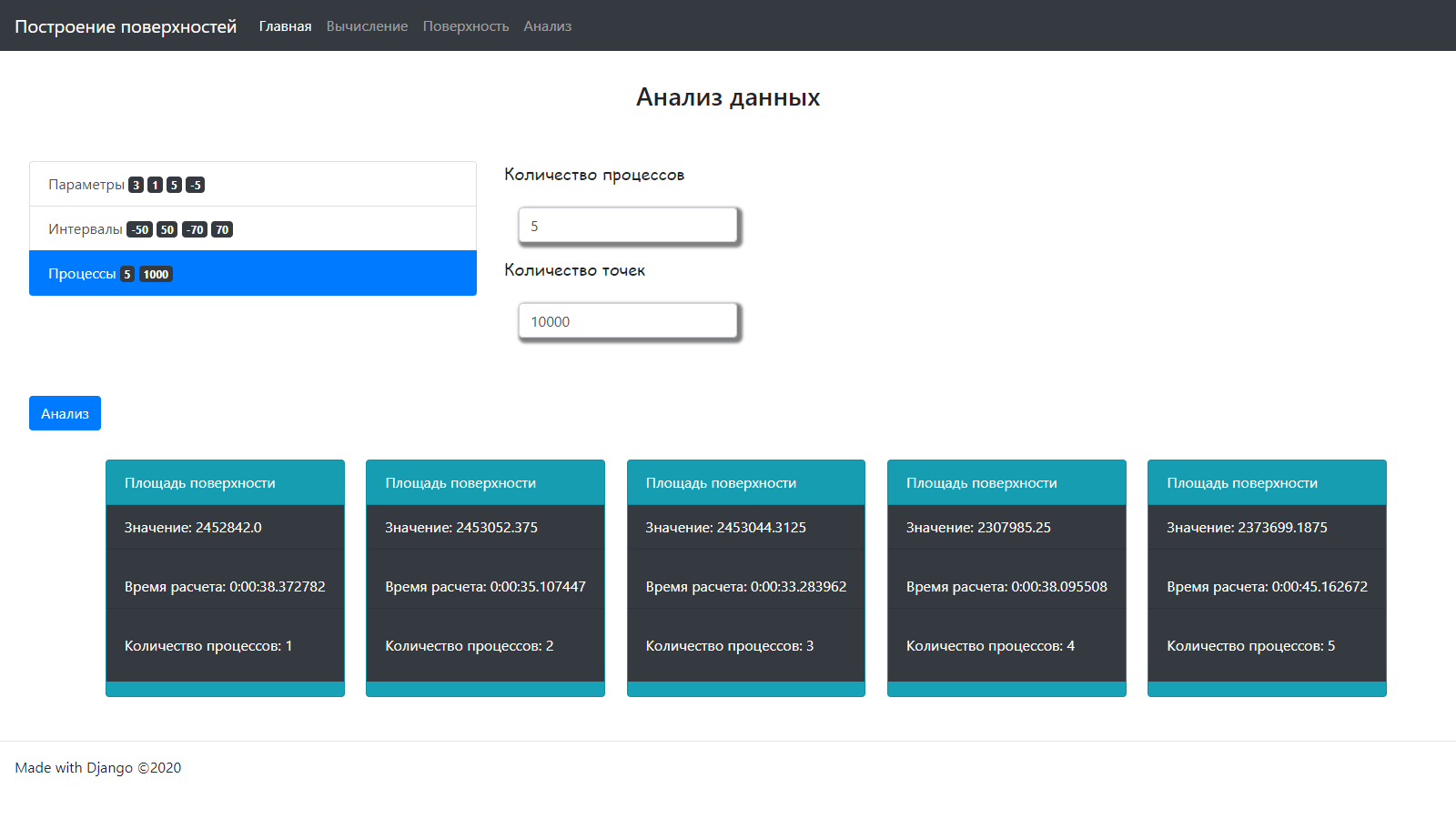


Рисунок 3.6 – Страница анализа

После проведения анализа отображается диаграмма зависимости время выполнения от количества процессов. Диаграмма показана на рисунке 3.7.

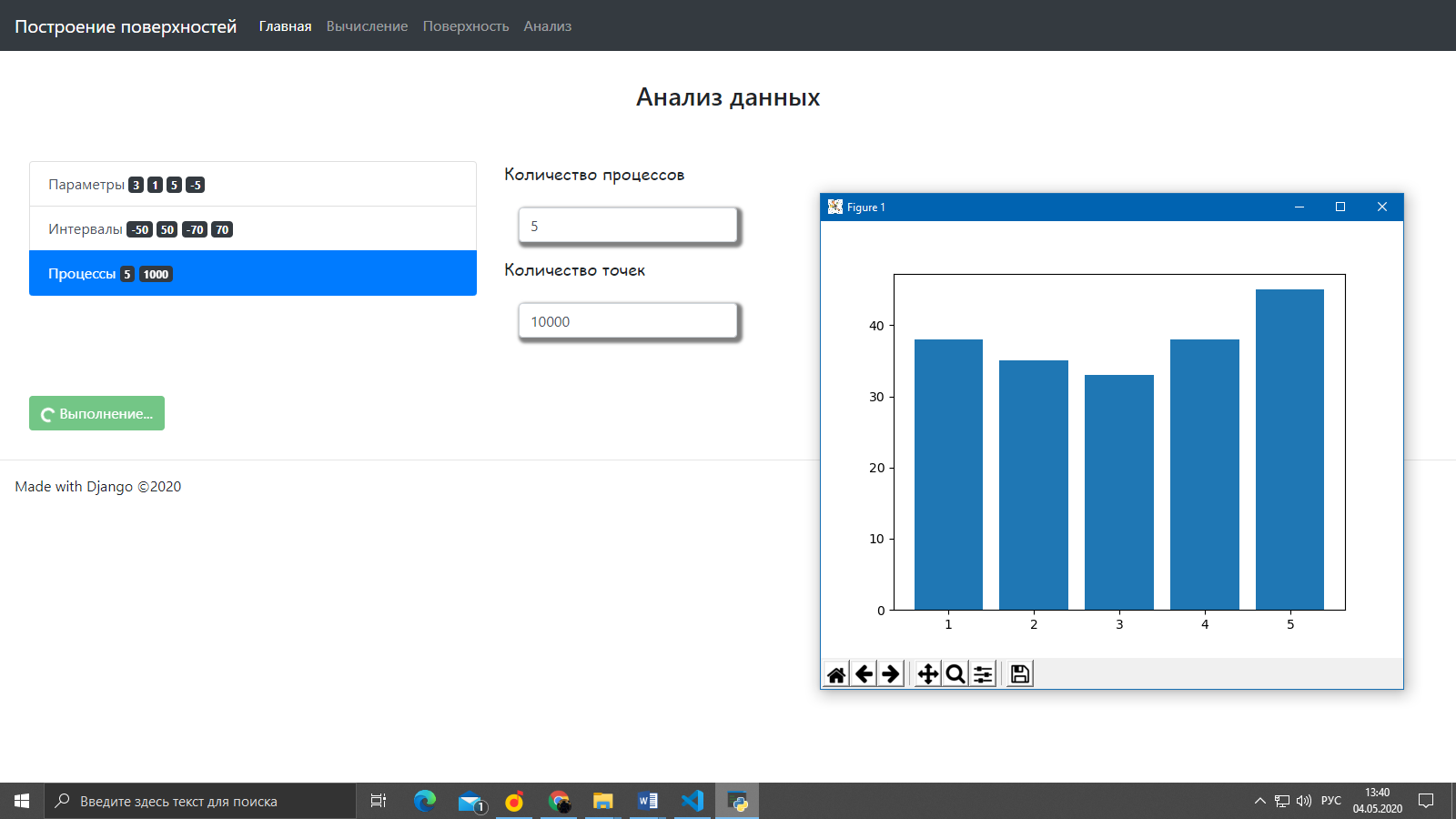


Рисунок 3.7 – Диаграмма

1. **ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ**

**4.1 Тестирование**

***4.1.1*** Тестирование графического интерфейса.

Для тестирование графического интерфейса необходимо составить чек-лист. В таблице 4.1 приведен список тестов по проверке графического интерфейса.

Таблица 4.1**–** Чек-лист тестирования графического интерфейса

|  |  |  |
| --- | --- | --- |
| Номер теста | Описание | Ожидаемый результат |
| 1.1 | Увеличить браузер на весь экран | Страница отобразилась в увеличенном размере |
| 1.2 | Уменьшить браузер до минимального допустимого размера | Страница отобразилась в уменьшенном виде |
| 1.3 | Поле с указанием количество процессов оставить пустым | Вычисление площади не возможно |
| 1.4 | Ввести отрицательное число для поле ввода количества делении | Вычисление площади не возможно |
| 1.5 | Не указывать файл с данными при построении поверхности | Построение поверхности не возможно |

Проверим масштабируемость приложения, уменьшив окно браузера до минимального допустимого размера. На рисунке 4.1 изображен результат выполнения теста.

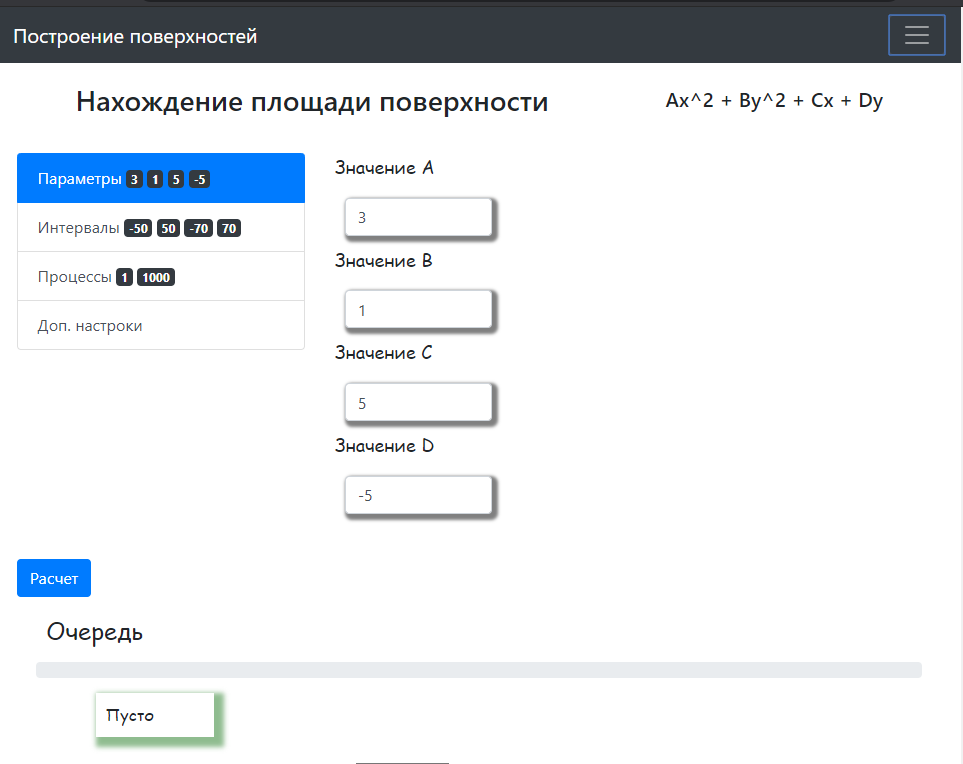


Рисунок 4.1 – Результат выполнения теста 1.1

Теперь проверим масштабируемость приложения увеличив окно браузера до максимального размера. На рисунке 4.2 изображен результат выполнения теста.

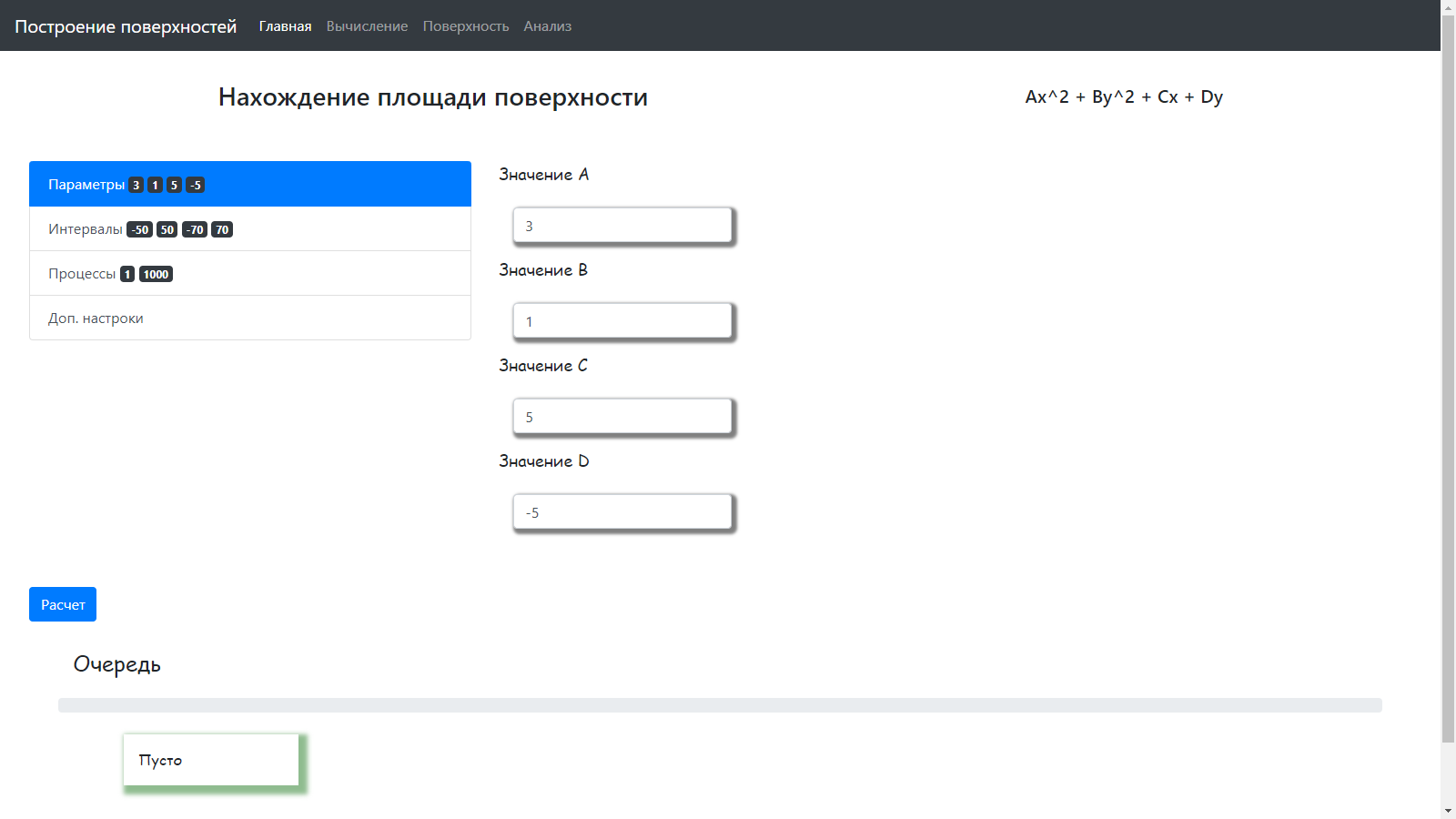


Рисунок 4.2 – Результат выполнения теста 1.2

Если поле для ввода количества процессов оставить пустым, то вычисление будет невозможно и появится предупреждение. На рисунке 4.3 показано, что кнопка вычисления заблокирована если поле оставить пустым.

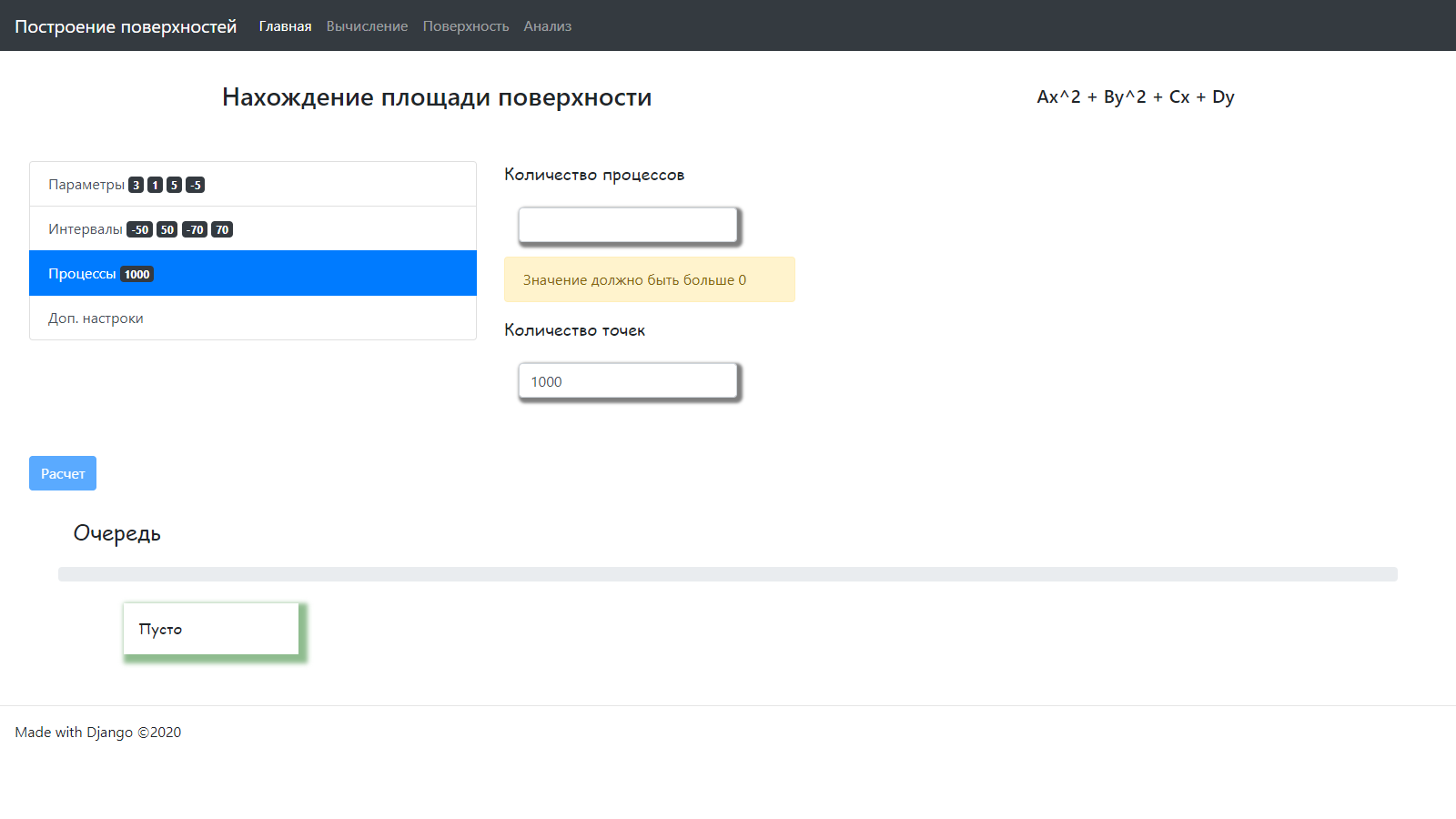


Рисунок 4.3 – Результат выполнения теста 1.3

На рисунке 4.4 показано, что кнопка для вычисления заблокируется, если указать отрицательное значение для поля количество делений.

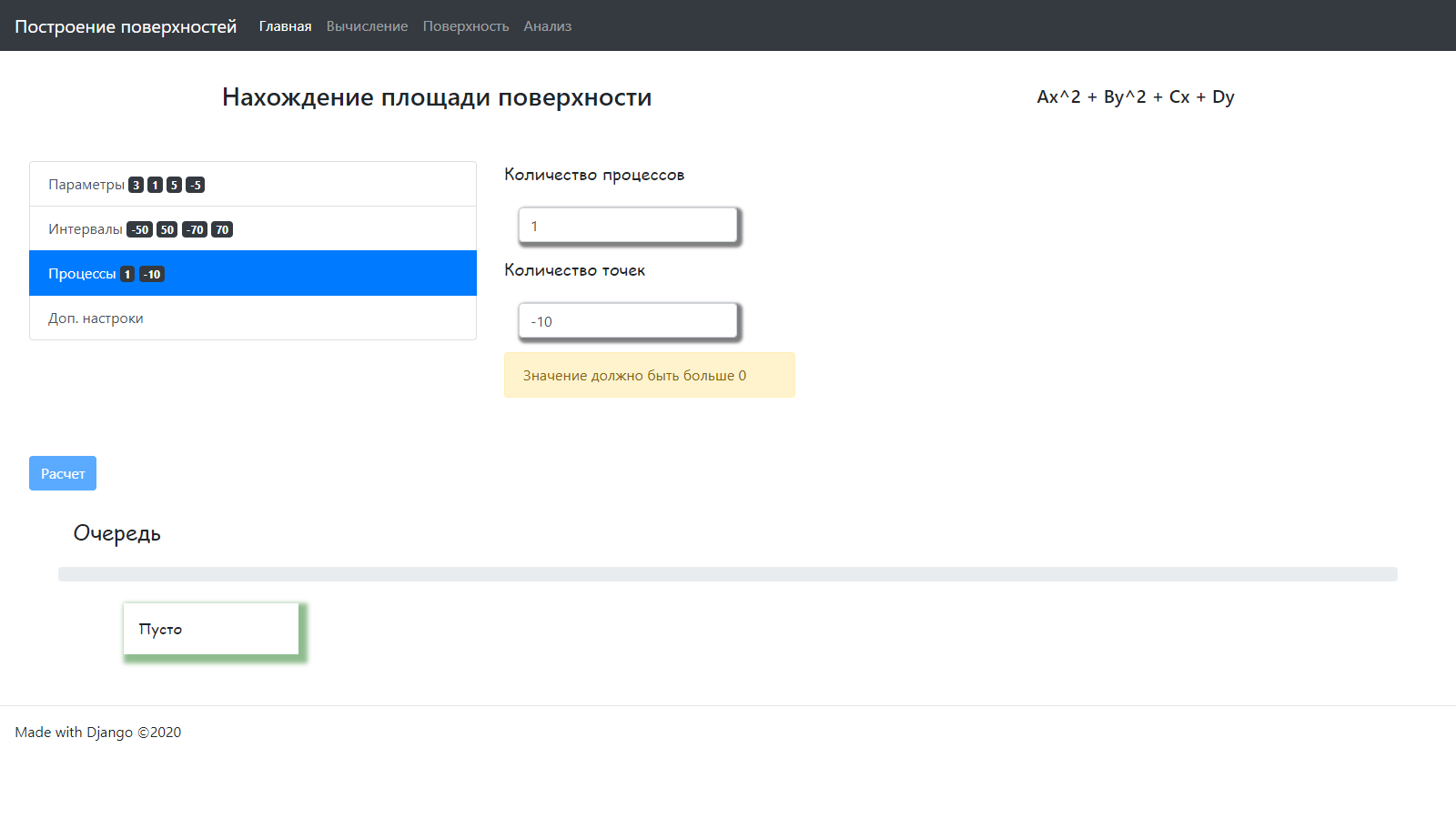


Рисунок 4.4 – Результат проверки теста 1.4

На рисунке 4.5 показано, что при попытке построить поверхность, не указав хотя бы один из файлов с данными, кнопка построения заблокируется и появится предупреждение.

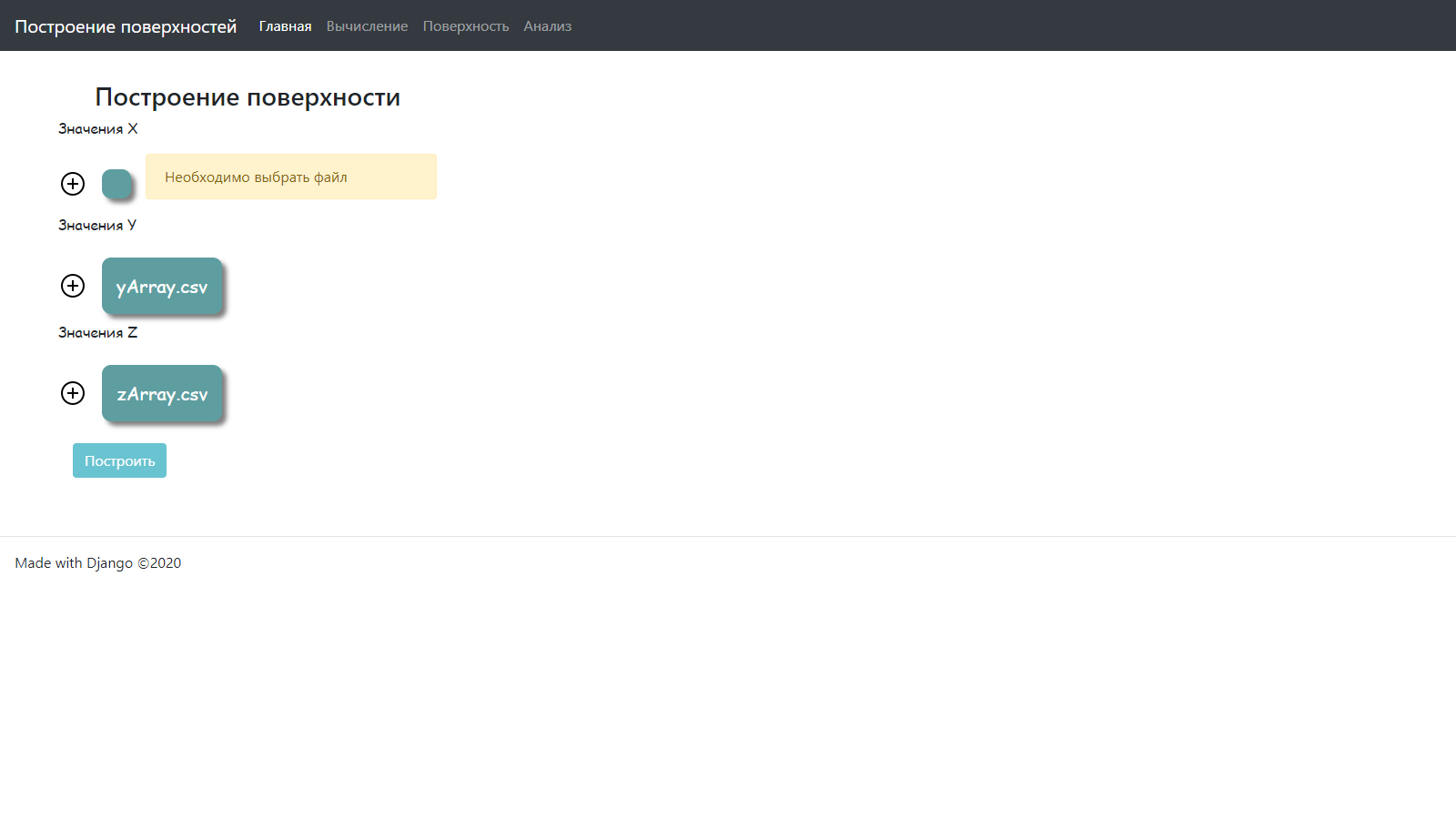


Рисунок 4.5 – Результат выполнения теста 1.5

В таблице 4.2 приведен результат тестирования.

Таблица 4.2 **–** Результат выполнения тестов из чек-листа

|  |  |
| --- | --- |
| Результат выполнения | Номер теста |
| Выполнено успешно | 1.1 |
| Выполнено успешно | 1.2 |
| Выполнено успешно | 1.3 |
| Выполнено успешно | 1.4 |
| Выполнено успешно | 1.5 |

* + 1. Тестирование программы.

Тестирование программы было проведено при помощи созданных Unit тестов. Чек-лист тестирование позитивных тестов показан в таблице 4.3.

Согласно [8], Unit-тестирование — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Таблица 4.3 – Чек лист тестирование позитивных тестов

|  |  |  |
| --- | --- | --- |
| Номер теста | Описание | Ожидаемый результат |
| 2.1 | Проверка правильного построения графика при вводе корректных данных | 1.График построен верно |
| 2.2 | Проверка правильного преобразование введённой функции в обратную польскую запись | 1.Функция правильно преобразовалась в обратную польскую запись |

Чек-лист тестирование негативных тестов показан в таблице 4.4.

Таблица 4.4 – Чек лист тестирование негативных тестов

|  |  |  |
| --- | --- | --- |
| Номер теста | Описание | Ожидаемый результат |
| 3.1 | Проверка построения графика если были введены не корректные данные | 1.Кнопка «Получить» заблокирована |
| 3.2 | Проверка построения графика, если функция введена верна, но интервалы функции введены не верно | 1.График не построился |
| 2.3 | Проверка преобразование функции в обратную польскую запись, если функция введена не корректна | 1.При преобразовании была возвращена ошибка преобразования |

Результат тестирования Unit тестов показан на рисунке 4.8.

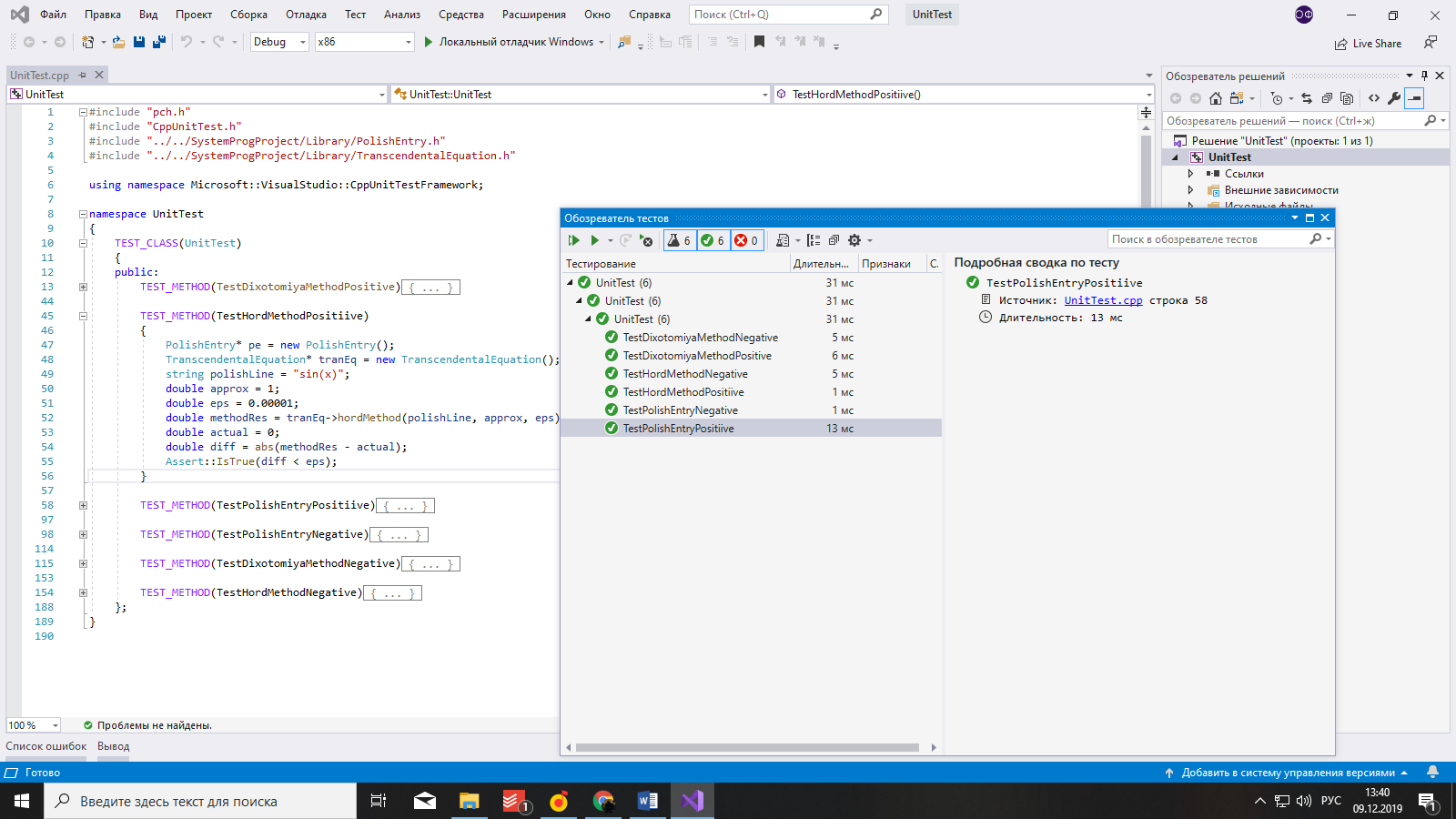


Рисунок 4.8 – Результат Unit тестирования

* 1. **Верификация и анализ полученных данных**

Программа, написанная в результате выполнения курсового проекта, строит график функции из задания при помощи получения результатов через преобразование функции к обратной польской записи и нахождение значения через математический сопроцессор.

При запуске программы появится окно с пустой сеткой. На рисунке 4.9 показан вид окна при запуске программы.

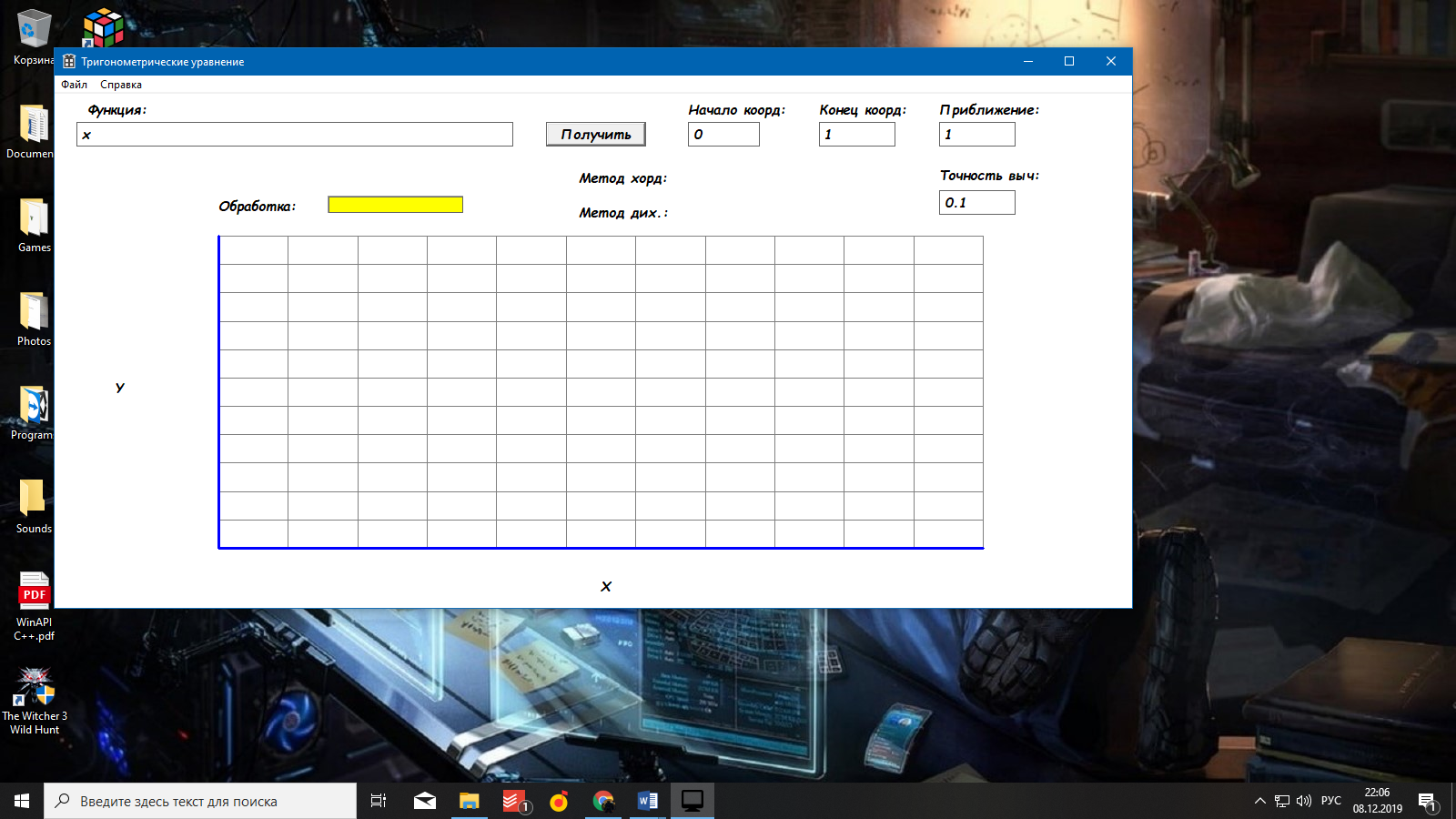


Рисунок 4.9 – Вид главного окна при запуске приложения

В программе имеется возможность изменения промежутков отображения графика.

С помощью Scilab происходит проверка найденных корней. На рисунке 4.10 показан график и найденные корни, которые были получены в приложении.

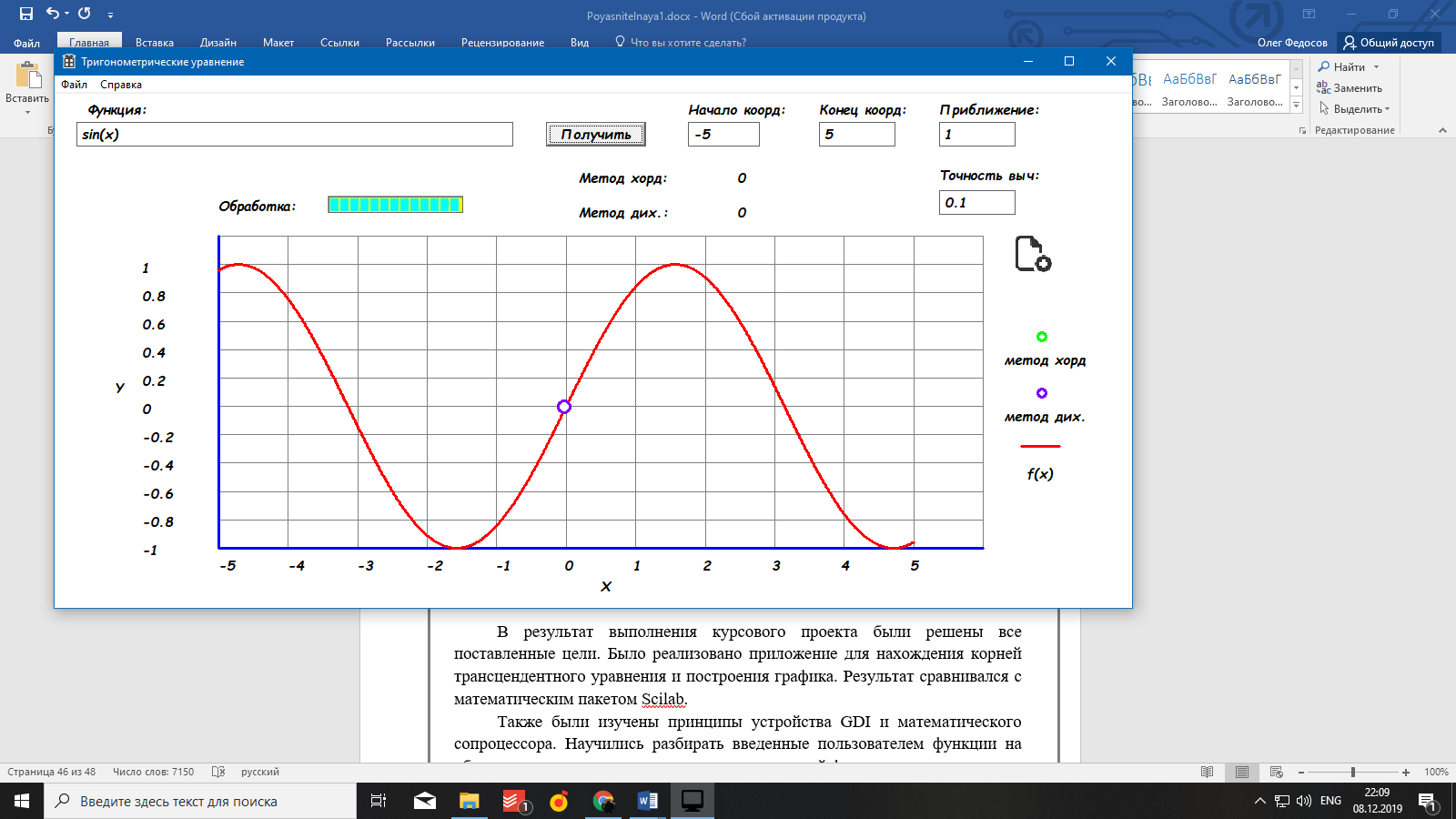


Рисунок 4.10 **–** Результат нахождения корней

Сравним результат с результатом из Scilab. Результат, полученный в Scilab показан на рисунке 4.11.

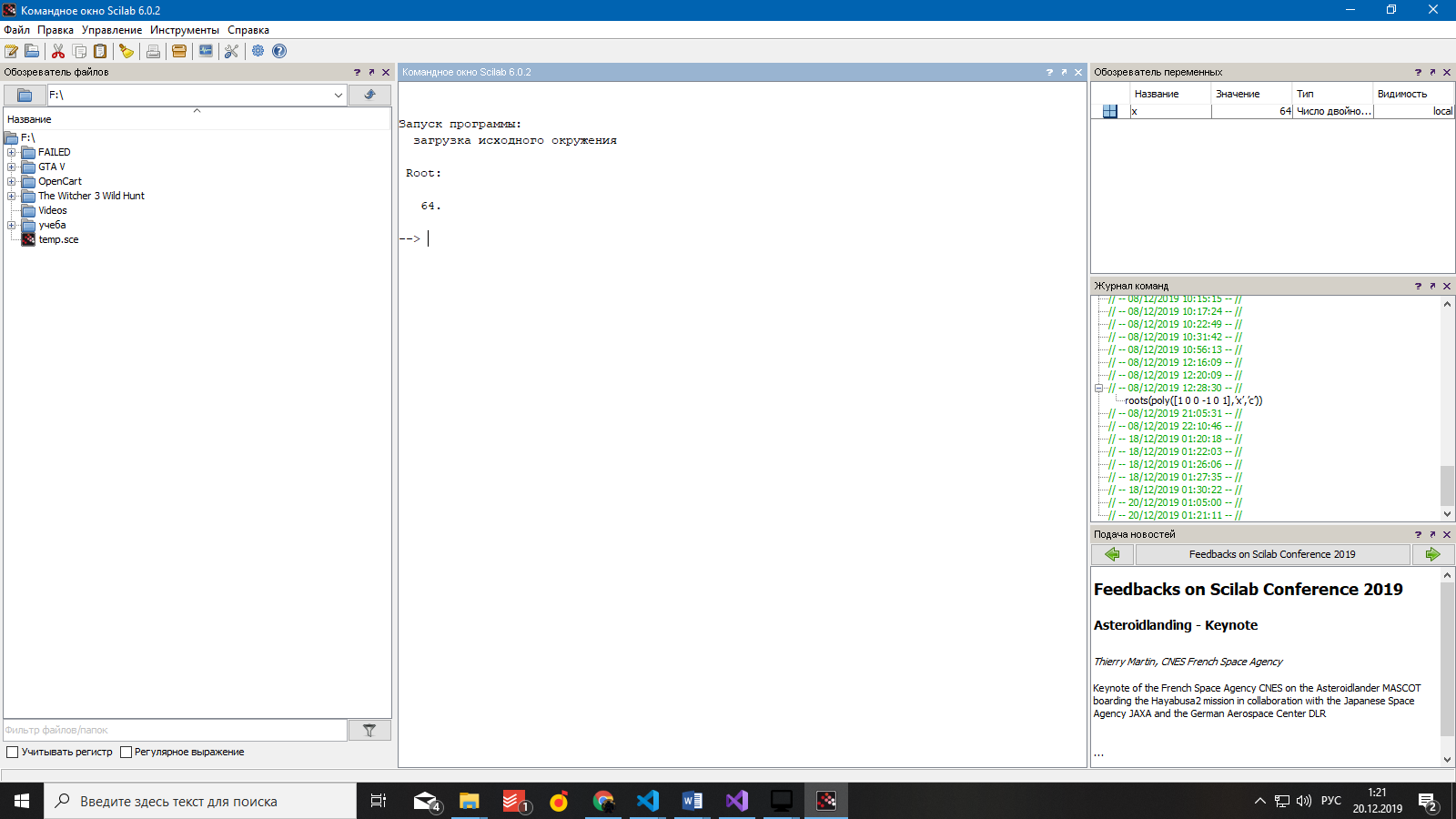


Рисунок 4.11 – Результат нахождения корней в Scilab

На рисунке 4.11 показаны корни двух методов. Первый корень, метод хорд, показывает лишь один корень - ближайший к начальному приближению. Метод дихотомии находит все возможные корни на указанном интервале.

Корни, найденные разработанной программой, совпадают с корнями. Найденными в Scilab.

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Калиткин, Н.Н. Численные методы / Н.Н. Калиткин. – СПб.:BHV, 2014. – 202 с.
2. Холл, Дж. Современные численные методы решения интегралов/ Дж. Холл, Дж. Уамл. — М.: Мир, 2007. – 312 с.
3. Кудрявцев, Е. М. Справочник по Mathcad / Е. М. Кудрявцев. – М.:ДМK Пресс. – 2005. – 111 с.
4. Бизли. Python. Подробный справочник / В.И.Юров. – Спб: Питер, 2014.– 624 с.