Оглавление

[**Глава 1. Введение** 3](#_Toc200456927)

[**1.1. Актуальность темы** 3](#_Toc200456928)

[**1.2. Цели и задачи исследования** 3](#_Toc200456929)

[**1.3. Ключевые особенности подхода** 4](#_Toc200456930)

[**1.4. Предпосылки и исходные данные для работы** 5](#_Toc200456931)

[**Глава 2. Конструктивные особенности и принцип работы подпорных стен уголкового типа** 6](#_Toc200456932)

[**2.1. Классификация подпорных стен** 6](#_Toc200456933)

[**1.3. Принцип работы и нагрузки** 8](#_Toc200456934)

[**Глава 3. Сравнительный анализ методов автоматизированного проектирования подпорных стен** 10](#_Toc200456935)

[**3.1. Введение и постановка задачи** 10](#_Toc200456936)

[**3.2. Методология исследования** 10](#_Toc200456937)

[**3.3. Сравнительный анализ методов** 10](#_Toc200456938)

[3.3.1. Чистый код на Python 10](#_Toc200456939)

[3.3.2. Чистый код на VBA 12](#_Toc200456940)

[3.3.3. Grasshopper + экспорт в AutoCAD 13](#_Toc200456941)

[**3.4. Количественное сравнение методов** 15](#_Toc200456942)

[**3.5. Выводы** 15](#_Toc200456943)

[**Глава 4. Структурирование исходных данных** 18](#_Toc200456944)

[**4.1. Создание Excel – файла исходных данных** 18](#_Toc200456945)

[4.1.1. Параметризация сечения подпорной стенки 18](#_Toc200456946)

[4.1.2. Подсчет объемов работ 20](#_Toc200456947)

[**4.2. Формирование ведомости объемов работ** 24](#_Toc200456948)

[**Глава 5. Программная реализация** 26](#_Toc200456949)

[**5.1. Выгрузка исходных данных** 26](#_Toc200456950)

[**5.2. Отрисовка опалубочного чертежа подпорной стенки в Autocad** 27](#_Toc200456951)

[5.2.1. Описание интерфейса и функционала 28](#_Toc200456952)

[5.2.2. Отрисовка видов и обозначений 30](#_Toc200456953)

[**5.3. Создание BIM - модели в Tekla Struktures** 33](#_Toc200456954)

[5.3.1. Отрисовка секции подпорной стены 33](#_Toc200456955)

[5.3.1. Корректировка секции подпорной стены 35](#_Toc200456956)

[**Глава 6. Заключение** 36](#_Toc200456957)

[**6.1. Итоги работы** 36](#_Toc200456958)

[**6.2. Перспективное развитие программы и ее дальнейшее применение** 37](#_Toc200456959)

[**Список литературы:** 39](#_Toc200456960)

[**Приложения:** 40](#_Toc200456961)

[**Приложение 1. Общий вид исходных данных - файл Data** 40](#_Toc200456962)

[**Приложение 2. Ведомость объемов работ - файл Data** 41](#_Toc200456963)

[**Приложение 3.** **Общий вид вкладки Р1.1** 42](#_Toc200456964)

[**Приложение 4.** **Общий вид вкладки ВОР в Excel файле ВОР\_ПСТ** 43](#_Toc200456965)

[**Приложение 5. Топология всех видов опалубочных чертежей** 44](#_Toc200456966)

[**Приложение 6.** **Макет чертежа** 45](#_Toc200456967)

[**Приложение 7. Результат работы модуля AutocadWork.py** 46](#_Toc200456968)

[**Приложение 8. Результат работы модуля TeklaWork.py** 47](#_Toc200456969)

[**Приложение 9. Файл Stena.py** 48](#_Toc200456970)

[**Приложение 10. Файл Classes.py** 49](#_Toc200456971)

[**Приложение 11. Файл Functions.py** 52](#_Toc200456972)

[**Приложение 12. Файл AutocadWork.py** 54](#_Toc200456973)

[**Приложение 13. Файл TeklaWork.py** 63](#_Toc200456974)

# **Глава 1. Введение**

## **1.1. Актуальность темы**

В современном строительстве возрастает спрос на автоматизацию проектирования, особенно при работе с железобетонными конструкциями. Подпорные стены уголкового типа широко применяются в гражданском и промышленном строительстве, однако их проектирование и последующий выпуск документации - остается трудоемким процессом. Традиционные методы создания опалубочных чертежей и BIM-моделей требуют значительных временных затрат и подвержены ошибкам из-за ручной отрисовки и заполнения данных спецификаций. Автоматизация этих процессов позволит сократить сроки проектирования и минимизировать ошибки, вызванные человеческим фактором.

На сегодняшний день существуют решения для автоматизации проектирования железобетонных конструкций, однако специализированных инструментов для подпорных стен уголкового типа недостаточно. Большинство существующих BIM-библиотек и скриптов (например, для Revit или Tekla Structures) предлагают лишь базовые элементы, не учитывающие всех особенностей железобетонных конструкций.

## **1.2. Цели и задачи исследования**

Цель работы – разработка алгоритма и программного инструмента для автоматизированного создания опалубочных чертежей и BIM-моделей подпорных стен уголкового типа по заранее определенным параметрам. Эти параметры предполагается заполнять в табличном формате в программе Excel. Так же на основе данных этих таблиц в дальнейшем будут сформированы Ведомости объемов работ и все спецификации, выводимые на листы чертежей.

Задачи исследования:

1. Анализ конструктивных особенностей подпорных стен уголкового типа;
2. Исследование существующих методов автоматизации в BIM-проектировании;
3. Разработка параметрической модели подпорной стены с возможностью адаптации под различные геометрические особенности;
4. Создание алгоритма генерации опалубочных чертежей и BIM-модели;
5. Реализация скрипта (на Python без использования Grasshopper) для автоматизации процесса;
6. Тестирование разработанного решения на реальных проектах;

Цели исследования:

1. Сокращение времени проектирования;
2. Унификация процесса моделирования подпорных стен;
3. Снижение количества ошибок и несоответствий между разными разделами документации (между ведомостью объемов работ и спецификациями на листах);

Упрощение взаимодействия между конструкторами и BIM-проектировщиками.

## **1.3. Ключевые особенности подхода**

1. Отказ от Grasshopper в пользу прямого программирования на Python обусловлен тем, что это повышает гибкость, увеличивает скорость обработки данных и снижает зависимость от дополнительного программного обеспечения.
2. Использование Excel как интуитивного интерфейса для инженеров (ввод параметров, табличные расчеты).
3. Сквозная автоматизация (от расчетов до BIM-модели) без ручных операций.

## **1.4. Предпосылки и исходные данные для работы**

Существует два основных этапа проектирования сооружения: стадия Проектной документации и стадия Рабочей Документации. Чтобы понять, чем отличается рабочая документация от проектной, стоит более подробно остановиться на ключевых особенностях каждой из них.

Стадия Проект - предназначена для прохождения экспертизы и защиты проектных решений. Проект строительства объекта включает функциональные, архитектурные, технические, инженерные решения. Все они отражаются в графическом и текстовом форматах. Если говорить о том, что включает проектная документация и рабочая документация, отличия состоят в детализации. Проектная документация включает более широкий спектр информации об объекте, но прорабатывается с невысокой степенью детализации, главное она должна пройти государственную или негосударственную экспертизу на соответствие строительным нормам и обеспечению безопасности. После чего на ее основе разрабатывается рабочая документация.

Стадия разработки Рабочей документации является основной и неотъемлемой частью всей документации по строительному объекту. Содержит информацию, установленную законодательством и позволяющую строительным бригадам осуществлять производственный процесс. Комплект включает в себя рабочие чертежи и схемы, описание решений, спецификацию материалов и изделий. Существует ряд требований, которых следует придерживаться, занимаясь подготовкой рабочей документации.

Текстовая часть содержит сведения в отношении объекта капитального строительства, описание принятых технических и иных решений, пояснения, ссылки на нормативные и (или) технические документы, используемые при подготовке проектной документации и результаты расчетов, обосновывающие принятые решения. Графическая часть отображает принятые технические и иные решения и выполняется в виде чертежей, схем, планов и других документов в графической форме.

В данной работе я буду рассматривать раздел разработки Рабочей документации для комплекта железобетонных элементов КЖ-ПСТ.

# **Глава 2. Конструктивные особенности и принцип работы подпорных стен уголкового типа**

## **2.1. Классификация подпорных стен**

Подпорные стены применяются в гражданском и промышленном строительстве для удержания грунтовых масс от обрушения. В зависимости от конструкции и способа восприятия нагрузок их можно разделить на:

* Массивные – работают за счет собственного веса (бетон, бутовая кладка).

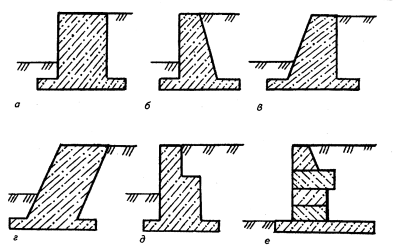


Рис.1 – Массивные подпорные стены

* Тонкостенные (уголковые) – состоят из вертикальной плиты (консоли) и горизонтального фундаментного выступа (анкерной плиты), образуя Г-образную конструкцию.

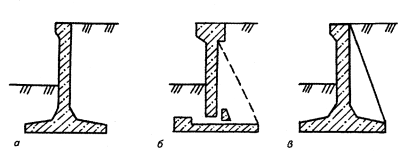


Рис.2 – Тонкостенные подпорные стены

* Анкерные – дополнительно укрепляются тяжами или сваями.
* Гибкие (шпунтовые) – из металлического или железобетонного шпунта.

Преимущества уголковых стен:

* Меньший расход материала по сравнению с массивными.
* Возможность устройства на слабых грунтах за счет распределения нагрузки через фундаментную плиту.
* Удобство армирования и изготовления.

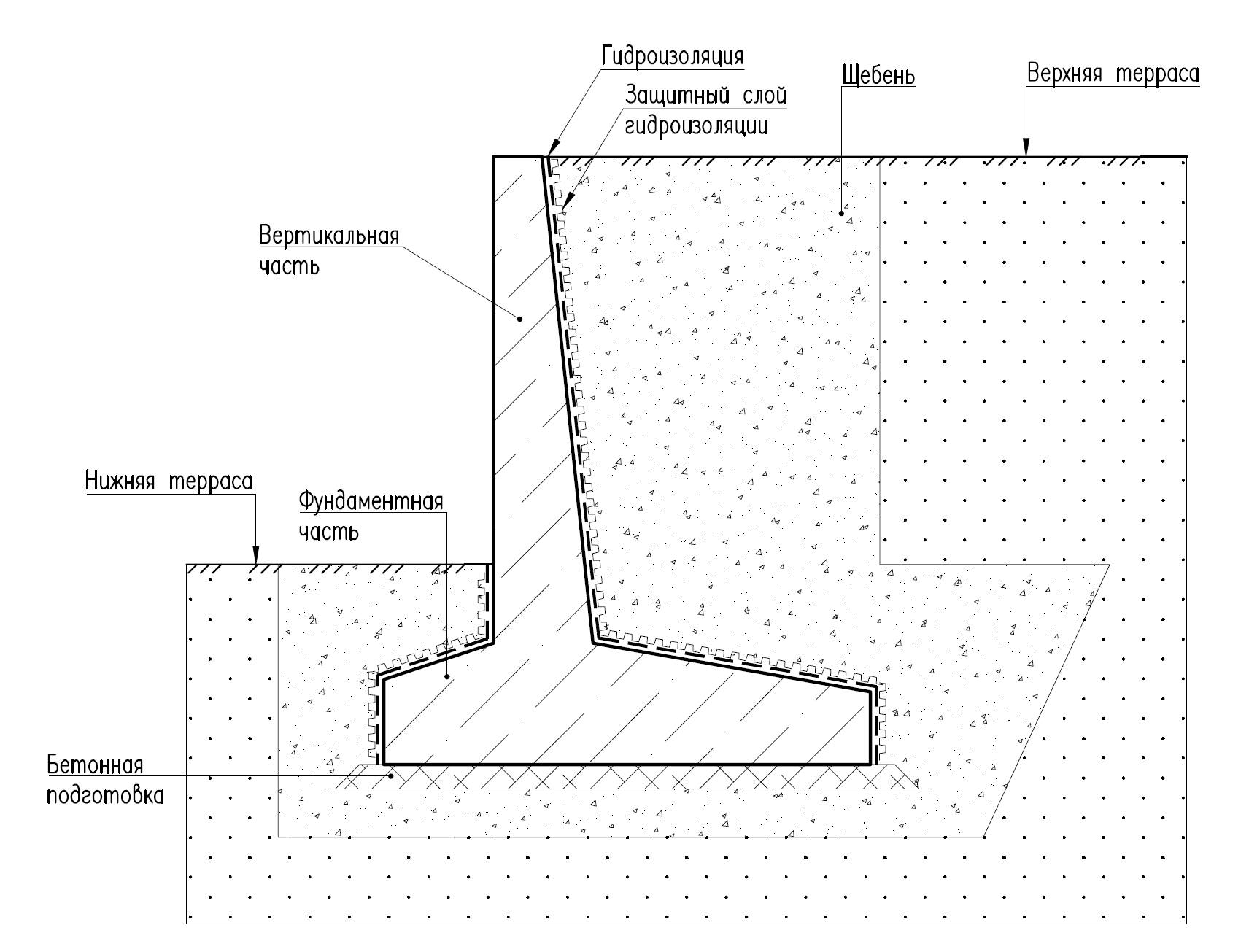
**2.2. Конструкция уголковой подпорной стены**

Рис.3 – Схема устройства подпорной стенки

* Вертикальная часть (стена) – воспринимает давление грунта.
* Фундаментная часть (ростверк) – обеспечивает устойчивость, предотвращает опрокидывание.
* Пятка – выступающая часть плиты со стороны обратной засыпки.
* Носок – часть плиты с лицевой стороны.
* Зуб – выступающая снизу часть фундамента плиты (не обязательная часть)
* Свайное основание (не обязательная часть)

## **1.3. Принцип работы и нагрузки**

Подпорная стена уголкового типа работает как жесткая железобетонная конструкция, удерживающая грунт за счет собственной устойчивости и взаимодействия с основанием. Основные воздействия (СП 43.13330.2018):

1. Восприятие давления грунта

Грунт, расположенный за стеной, стремится сместиться под действием силы тяжести, создавая горизонтальное давление на вертикальную часть стены (консоль). Это давление:

* Возрастает с глубиной – максимальное значение у основания.
* Зависит от типа грунта (песок, глина, суглинок) и его плотности.
* Рассчитывается по теории Кулона или Ренкина, учитывая угол внутреннего трения грунта и его сцепление.

2. Устойчивость против опрокидывания

Стена должна не только выдерживать давление грунта, но и не опрокинуться под его действием. Это обеспечивается:

* Собственным весом конструкции – бетон и арматура создают стабилизирующий момент.
* Весом грунта на фундаментной плите – особенно на ее "пятке" (части, выступающей в сторону засыпки).
* Длиной фундаментной плиты – чем она больше, тем выше сопротивление опрокидыванию.

Формула проверки на опрокидывание:

где:

​ – удерживающий момент (от веса стены и грунта),

– опрокидывающий момент (от давления грунта),

*k* – коэффициент запаса (обычно 1.2–1.5).

3. Устойчивость против сдвига

Давление грунта может вызвать горизонтальное смещение стены. Чтобы этого не произошло:

* Сила трения между основанием и фундаментной плитой должна быть больше горизонтальной составляющей давления.
* В слабых грунтах иногда применяют шпонки или анкеры для дополнительного закрепления.

Формула проверки на сдвиг:

где:

​– сила трения (зависит от веса конструкции и коэффициента трения),

– горизонтальное давление грунта.

*k* – коэффициент запаса (обычно 1.2–1.5).

# **Глава 3. Сравнительный анализ методов автоматизированного проектирования подпорных стен**

## **3.1. Введение и постановка задачи**

В современной практике проектирования подпорных стен особую актуальность приобретают методы автоматизированного создания чертежей и 3D-моделей. В данной главе проводится сравнительный анализ трех ключевых подходов:

1. Чистый код на Python с использованием API AutoCAD
2. Программирование на VBA во встроенной среде AutoCAD
3. Визуальное программирование в Grasshopper (с экспортом в AutoCAD)

Цель исследования - выявить оптимальный метод для различных сценариев проектирования подпорных стен на основе комплексной оценки по техническим и эргономическим критериям.

## **3.2. Методология исследования**

Для объективного сравнения применялись следующие методы:

* Функциональное тестирование (создание типовых элементов подпорных стен)
* Хронометраж выполнения операций
* Анализ сложности реализации
* Оценка точности и надежности результатов
* Исследование возможностей параметризации
* Базовые параметры для сравнения:
* Время разработки решения
* Быстродействие при выполнении
* Гибкость и адаптируемость
* Требования к квалификации пользователя
* Интеграционные возможности

## **3.3. Сравнительный анализ методов**

### 3.3.1. Чистый код на Python

Преимущества:

* Высокая гибкость и контроль над процессом проектирования

Можно реализовать любую логику, даже сложные алгоритмы, которые трудно или невозможно сделать в Grasshopper или VBA.

* Доступ к современным библиотекам (numpy для расчетов, pandas для работы с данными и т.д.)
* Возможность сложной параметризации и оптимизации конструкций
* Поддержка Объектно-Ориентированного Программирования для создания сложных иерархических моделей
* Кроссплатформенность и перспективность развития
* Масштабируемость

Легко адаптировать код для больших проектов. Легче подключаться к базам данных, внешним сервисам, API Revit/Dynamo и, например, с расчётными программами.

* Повторное использование

Код можно организовать в библиотеки и использовать в разных проектах.

* Версионность и документация

Код можно комментировать, хранить в Git, отслеживать изменения.

Недостатки:

* Требуется дополнительная настройка среды выполнения
* Более высокий порог входа для программирования

Для эффективной работы и возможности дополнения программы требуется знание программирования, понимание BIM-логики.

* Необходимость глубокого знания API AutoCAD
* Относительно медленное выполнение при использовании COM-интерфейса
* Сложность отладки и меньшая наглядность

Визуализация алгоритмов в чистом коде сложнее, чем в Grasshopper. Ошибки не всегда очевидны, особенно для новичков. В следствие этого написание кода занимает больше времени, чем визуальное моделирование в Grasshopper.

Оптимальная область применения:

* Комплексные проекты с нестандартными решениями
* Системы с интеграцией внешних расчетных модулей
* Долгосрочные проекты, требующие дальнейшего развития

### 3.3.2. Чистый код на VBA

Преимущества:

* Полная интеграция с AutoCAD

Встроенный редактор VBA позволяет получить быстрый доступ к API без дополнительных настроек.

* Высокая скорость выполнения скриптов

Так как код выполняется внутри AutoCAD без внешних вызовов и подключений – время выполнения программы существенно сокращается.

* Простота развертывания

Доступ ко всем функциям AutoCAD API без установки дополнительных библиотек

* Достаточно низкий порог входа для базовых задач
* Поддержка старых версий AutoCAD

Где-то Python может не работать, а VBA поддерживает все версии.

Недостатки:

* Устаревшая среда разработки и язык программирования

Microsoft постепенно отказывается от VBA и прекращает его поддержку. Это так же приводит к уменьшению количества людей в сообществе и невозможности обсуждать работу своей программы и получать помощь от более уверенных пользователей.

* Ограниченные возможности работы с внешними данными

Нет доступа к внешним библиотекам – Только то, что есть в AutoCAD и Windows.

* Отсутствие современных парадигм программирования

Нет удобных структур данных как в Python.

* Проблемы с поддержкой в новых версиях AutoCAD
* Сложность сопровождения сложных проектов
* Сложность отладки –

Примитивный редактор кода, не реализовано удобное управление процессом установки, настройки и обновления различных компонентов программного обеспечения.

Оптимальная область применения:

* Быстрое решение стандартных задач
* Простые макросы для рутинных операций
* Проекты с жесткими требованиями к скорости выполнения

### 3.3.3. Grasshopper + экспорт в AutoCAD

Преимущества:

* Доступность

Низкий порог входа. Подходит для инженеров и архитекторов без навыков программирования.

* Наглядность

Визуальные связи между компонентами упрощают понимание логики - интуитивно понятная среда разработки.

* Интерактивность

Параметры можно менять в ходе разработки, сразу видя изменения

* Быстрое прототипирование и итерации

Можно быстро собрать алгоритм без написания кода и понимания сложного синтаксиса.

* Широкие возможности параметрического моделирования
* Большое сообщество

Большое количество готовых плагинов (например, Karamba для расчётов, Human UI для интерфейсов)

Недостатки:

* Ограниченная производительность при сложных операциях
* Проблемы с организацией больших проектов

Большие графы становятся запутанными. Трудно организовать сложную логику, как в коде.

* Зависимость от сторонних плагинов

Некоторые функции требуют установки дополнений.

* Необходимость дополнительных операций экспорта
* Ограниченная гибкость

Сложные алгоритмы могут требовать кастомных компонентов или скриптов (Python/C#).

Оптимальная область применения:

* Концептуальное проектирование и исследования
* Параметрические модели средней сложности
* Быстрая визуализация идей и вариантов

## **3.4. Количественное сравнение методов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **Python** | **VBA** | **Grasshopper** |
| **Время разработки** | Высокое | Низкое | Среднее |
| **Скорость выполнения** | Средняя | Высокая | Низкая |
| **Гибкость** | Очень высокая | Низкая | Высокая |
| **Параметризация** | Полная | Ограниченная | Полная |
| **Поддержка сообщества** | Очень большая | Малая | Большая |
| **Перспективы развития** | Отличные | Плохие | Хорошие |

## **3.5. Выводы**

Проведенный анализ позволяет сформулировать следующие заключения:

1. Для сложных инженерных задач с требованиями к точности расчетов и интеграцией с внешними системами оптимальным выбором является Python, несмотря на более высокие начальные затраты.
2. Для рутинных операций в среде AutoCAD, особенно при работе с устаревшими версиями, VBA сохраняет свою актуальность благодаря простоте и скорости выполнения.
3. Для концептуального проектирования и параметрических исследований Grasshopper предоставляет наиболее эффективный рабочий процесс, хотя и имеет ограничения при работе с большими моделями.

Перспективы дальнейших исследований:

* Разработка гибридных подходов (например, Grasshopper + Python)
* Автоматизация проверки соответствия нормативным требования
* Интеграция с системами конечно-элементного анализа
* Оптимизация геометрии подпорных стен с использованием машинного обучения

Выбор конкретного метода должен основываться на требованиях проекта, доступных ресурсах и квалификации команды. В перспективе наблюдается четкая тенденция в сторону использования Python и визуального программирования, в то время как значение VBA постепенно снижается.

В рамках настоящей магистерской диссертации для автоматизации проектирования подпорных стен был выбран метод программирования на Python с использованием библиотеки PyAutocad и API Tekla Structures. Данный выбор обусловлен следующими ключевыми факторами:

1. Соответствие целям и задачам исследования

Проект требовал реализации сложной параметризации геометрии подпорных стен c возможностью последующего расширения функционала и созданием чертежей и моделей в двух разных САП-комплексах одновременно.

Python, в отличие от VBA и Grasshopper, предоставил полный контроль над всеми аспектами проектирования и системный подход к решению инженерных задач.

2. Технические преимущества

Выбор Python обусловлен его уникальными техническими характеристиками:

1. Возможность реализации ООП-подхода, что критически важно для создания:

* Иерархических классов элементов стены
* Системы наследования типов конструкций
* Инкапсуляции параметров класса стены (Wall)

1. Интеграция с внешними САПР через .NET API, обеспечивающая:

* Высокую производительность
* Прямой доступ к внутренним объектам AutoCAD
* Минимальные накладные расходы

В данном пункте имеется в виду, что для работы этого приложения не требуется покупать какие-то сторонние приложения, как в случае с Grasshopper, что существенно упрощает работу с ним.

1. Научная новизна и перспективность

В дальнейшем развитии приложения, Python позволяет реализовать инновационные аспекты исследования:

* Алгоритмы оптимизации формы стены на основе методов машинного обучения
* Автоматическую генерацию расчетной документации
* Параметрическую адаптацию к изменяющимся условиям

Эти возможности недостижимы при использовании VBA и требуют сложных обходных решений в Grasshopper.

Заключение:

Выбор Python в качестве основного инструмента автоматизации доказал свою эффективность на всех этапах исследования:

* На этапе разработки - благодаря богатому инструментарию и современным практикам программирования
* В процессе тестирования - за счет надежности и предсказуемости результатов
* При внедрении - благодаря гибкости и адаптируемости решения

Этот подход не только полностью удовлетворил требования текущего проекта, но и создал надежный фундамент для дальнейших исследований в области автоматизированного проектирования инженерных сооружений. Реализованное решение демонстрирует преимущества системного программирования перед традиционными методами в САПР и открывает новые возможности для цифровой трансформации в проектировании.

# **Глава 4. Структурирование исходных данных**

Необходимо сразу внести ясность – в данной программе не будут фигурировать проверки и расчеты подпорных стен, так как она создается в предпосылке работы на том этапе проектирования, когда их результаты уже получены и учтены. Расчеты конструкций и получение ключевых производятся в программных комплексах, основанных на методе конечных элементов. Результаты обрабатываются в Excel.

## **4.1. Создание Excel – файла исходных данных**

Первым шагом стала разработка Excel файла – и структурирование всех необходимых исходных данных и параметризация поперечного сечения железобетонной монолитной подпорной стены уголкового типа. Для демонстрации работы программы и всех вспомогательных файлов проекта, было выбрано искусственное сооружение – подпорная стена ИССО 1.1.1, состоящая из трех отдельных секций С1.1, С1.2 и С1.3. Каждая секция имеет одинаковую конфигурацию, но различные параметры.

### 4.1.1. Параметризация сечения подпорной стенки

Параметризация позволяет быстро изменять геометрию стены под конкретные условия, оптимизируя её по несущей способности и внешнему виду.

Поперечное сечение уголковой подпорной стены состоит из двух основных элементов:

1. Вертикальной стеновой части (консоль) – воспринимает давление грунта;
2. Горизонтальной части – фундамент (ростверк) – обеспечивает устойчивость и предотвращает опрокидывание.

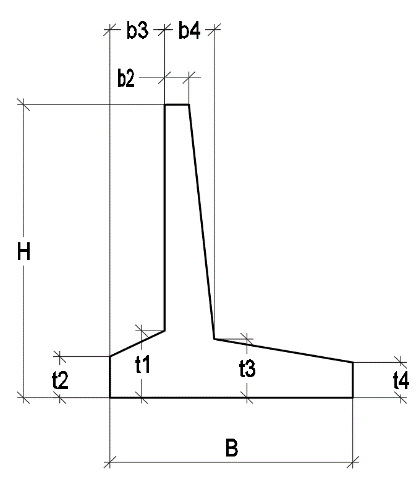


Рис. 4 – Параметризированное сечение железобетонной подпорной стены уголкового типа, где:

H // height end, height start – общая высота (различная для начала и конца подпорной стены);

B // foundation\_width – ширина фундамента подпорной стены;

b2 // top\_wall\_width – ширина стены наверху подпорной стены;

b3 // edge\_distance – расстояние от стены до границы подпорной стены;

b4 // bottom\_wall\_width – ширина стены на границе с ростверком;

t1 // t1 – толщина перекрытия 1 у стены;

t2 // t2 – толщина перекрытия 1 у насыпи;

t3 // t3 – толщина перекрытия 2 у стены;

t4 // t4 – толщина перекрытия 2 у насыпи;

В таблице Excel файла Data все параметры и необходимые характеристики выглядят вот так:

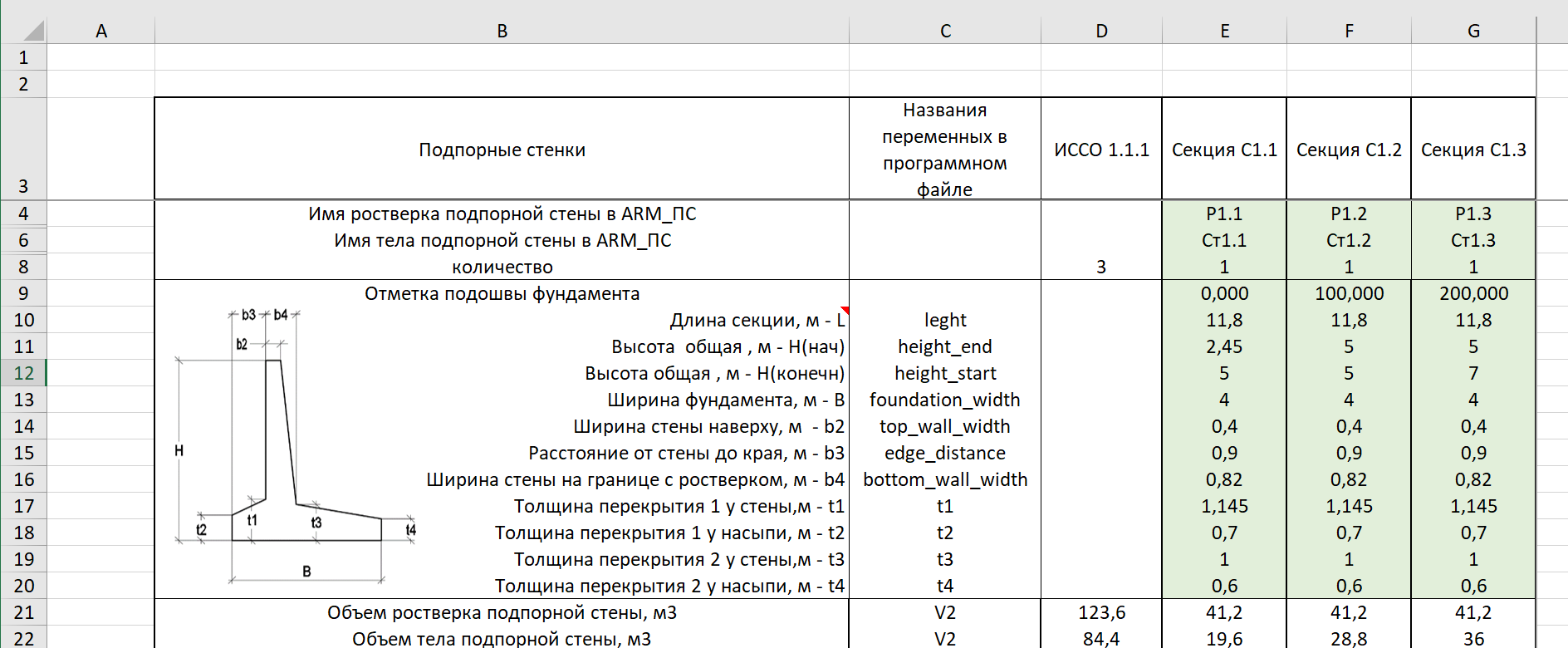


Рис.5 – Параметры подпорной стены и ее секций в файле Data

Отмеченное зеленым цветом подлежит заполнению – вручную, белым –вычисляется автоматически.

### 4.1.2. Подсчет объемов работ

Так же на этой же вкладке файла Data производится заполнение некоторых параметров и вычисление объемов для заполнения таблиц спецификаций, выводимых моей программой на лист опалубочного, чертежа и включаемых в отдельную Ведомость объемов работ. Не все параметры из этого перечня будут напрямую задействованы в данной работе, но они необходимы для общего понимания конструкции и состава ИССО, а также заполнения Ведомостей объемов работ и возможности дальнейшего развития программы.

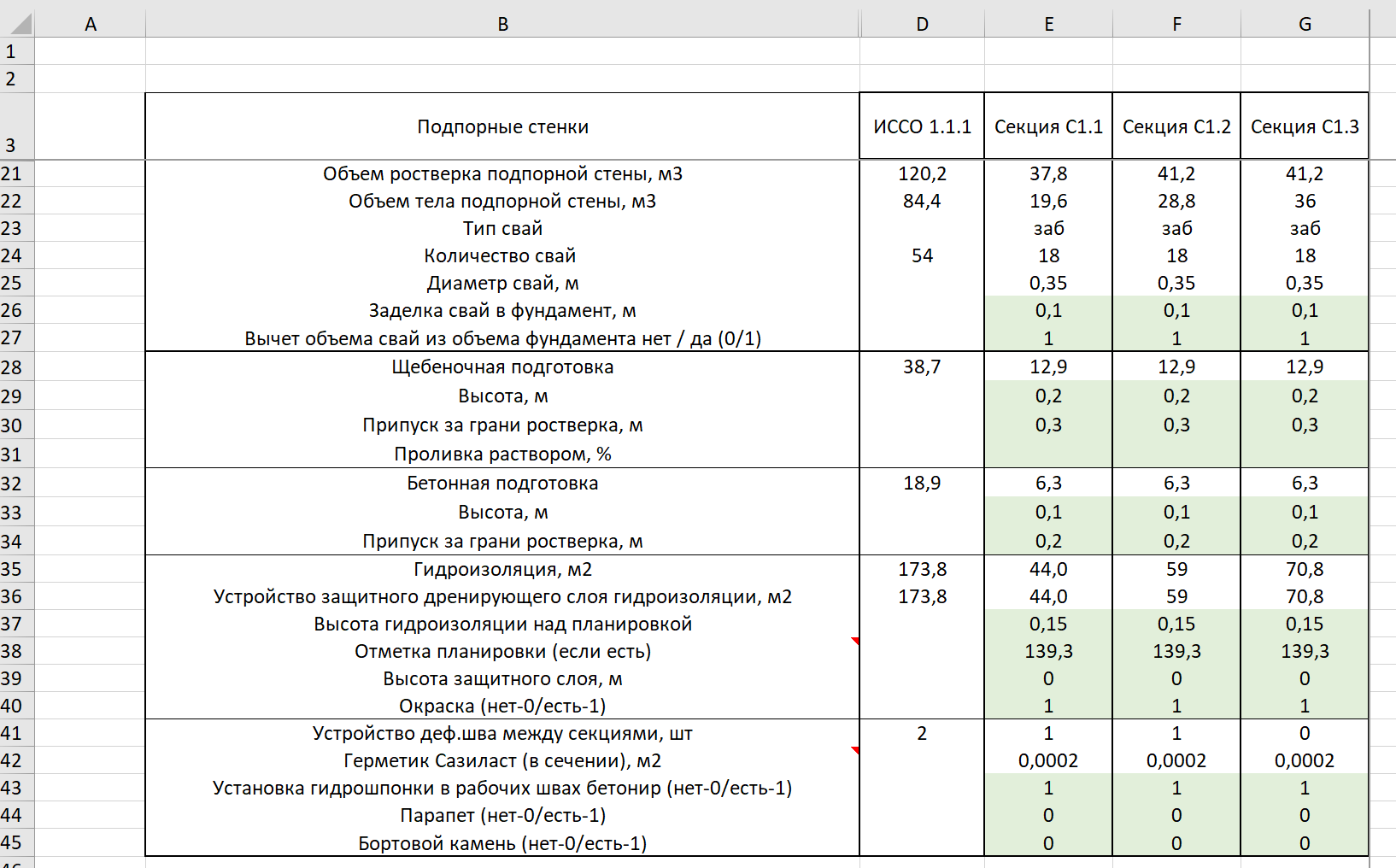


Рис.6 – Объемы работ подпорной стены и ее секций в файле Data

Объемы и параметры столбца с названием ИССО1.1.1 – содержат в себе суммирование объемов по всем секциям. Разделение объемов бетона и объемов арматуры вертикальной и фундаментных частей подпорной стены обусловлены исключительно производственной необходимостью, так как в некоторых случаях они могут иметь различную марку и характеристики бетона.

В файле Data предусмотрена связка с еще одним Excel файлом который производит расчет количества арматуры подпорной стены – ARM\_ПСТ. Все основные функции работы этого файла были уже написаны на момент разработки программы и в этой работе описываться не будут.

Мной в него вносились лишь незначительные корректировки и прописывались формулы для расчета длин стержней. Так же туда отправлялись необходимые параметры из файла Data для расчетов длин арматурных стержней по фундаментной и вертикальной части каждой секции. Все необходимые значения подгружаются автоматически при заполнении ячеек с названием конкретной секции с помощью ссылок функции ГПР.

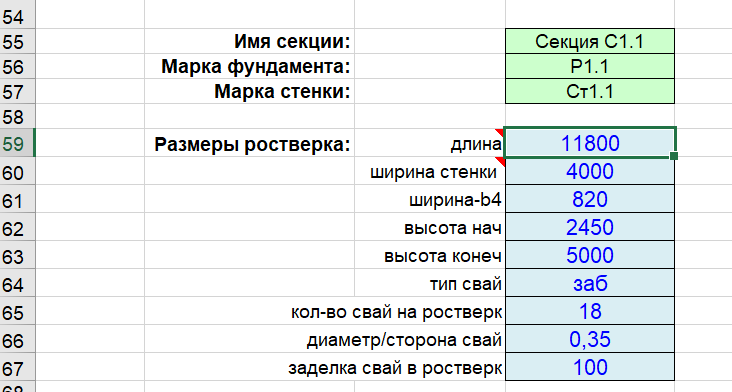


Рис. 7 – Заполнение исходных данных файла ARM\_ПСТ

Зеленым цветом отмечены ячейки, заполняемые вручную, синим – автоматически.

Заполнение происходит с помощью функции ГПР:

=ГПР($D$55;[DATA\_.xlsm]\_ПС!$E$3:$Z$25;8)\*1000

где $D$55 – ячейка, содержащее в себе название секции, по которой производится поиск столбца (секции) в файле Data, и подбор нужной строки (в данном случае 8 по счету) с необходимым параметром.

Эта связка необходима для полноценного заполнения всех необходимых объемов для Ведомости объемов работ и дальнейшего усовершенствования и развития моей программы.

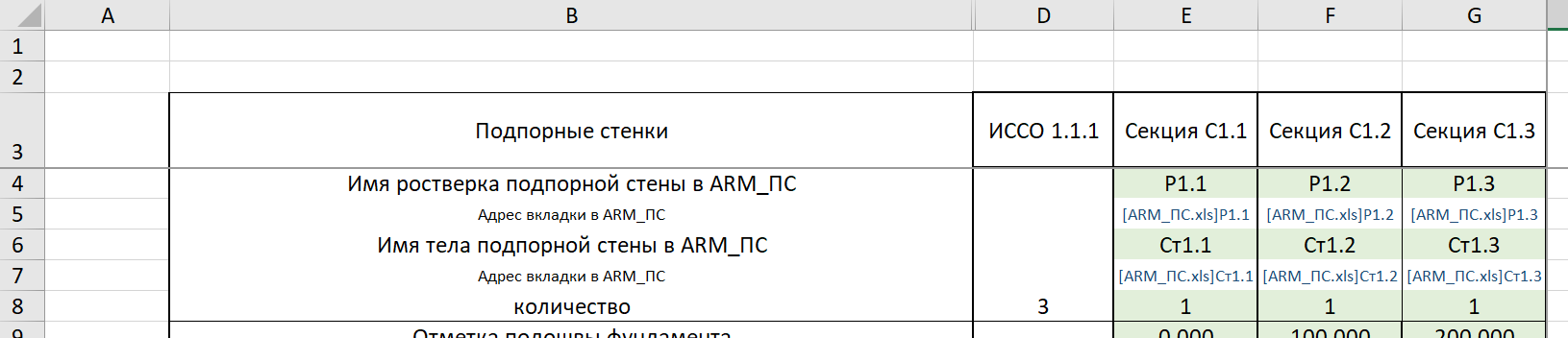


Рис.8 – Ссылки на файлы с расчетом армирования отдельных компонентов подпорной стены.

Результатом работы Excel файла ARM\_ПСТ является расчет массы всех стержней арматуры фундаментной и вертикальной частей каждой секции подпорной стены по отдельности. В файле Data они распределяются по диаметрам и классам и приводятся на странице ВОР\_ПСТ.

ГПР($D$56;'C:\Users\Ponka\Desktop\Диплом\Diplom2\[DATA\_.xlsm]\_ПС'!$E$3:$Z$25;11)\*1000

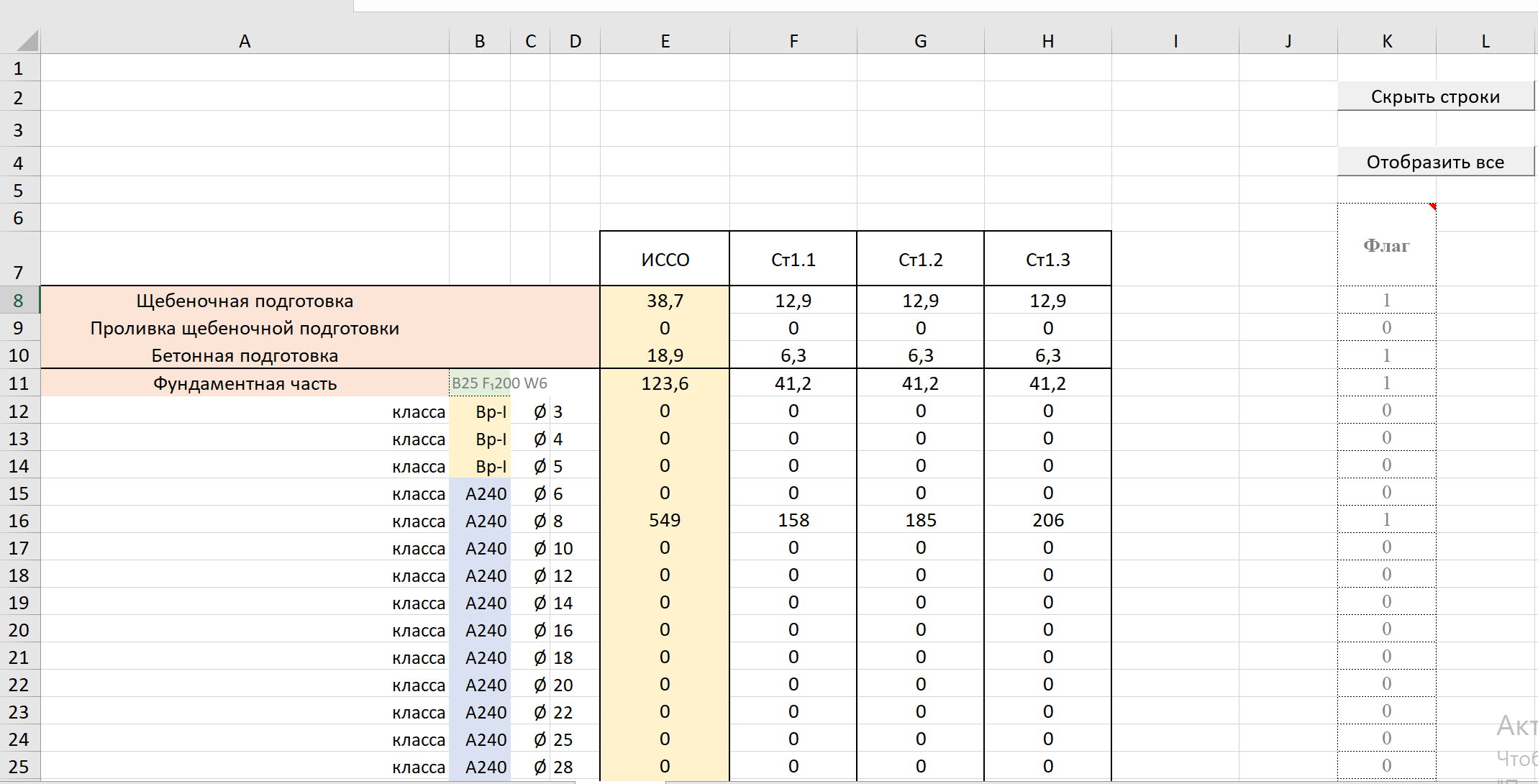


Рис. 9 – Фрагмент Таблицы объемов ВОР\_ПС из файла Data

Флаг и кнопки «скрыть строки» и «показать строки» скрывают нулевые объемы с сохранением единого шаблона заполнения, что позволяет копировать файлы E из объекта в объект и не перебивать искомые адреса ячеек. На этой логике построено все взаимодействие как между отдельными файлами Excel между собой, так и с программным кодом.

Ссылка на количество арматуры каждого диаметра и класса задается с помощью:

=ДВССЫЛ(АДРЕС(9;3;;;\_ПС!E$5))

где ПС!E$5 - ячейка в которой содержится ссылка на необходимый лист в файле ARM\_ПС, а адрес ячейки, содержащий необходимый объем всегда одинаковый благодаря структуре расчетного листа каждого элемента.

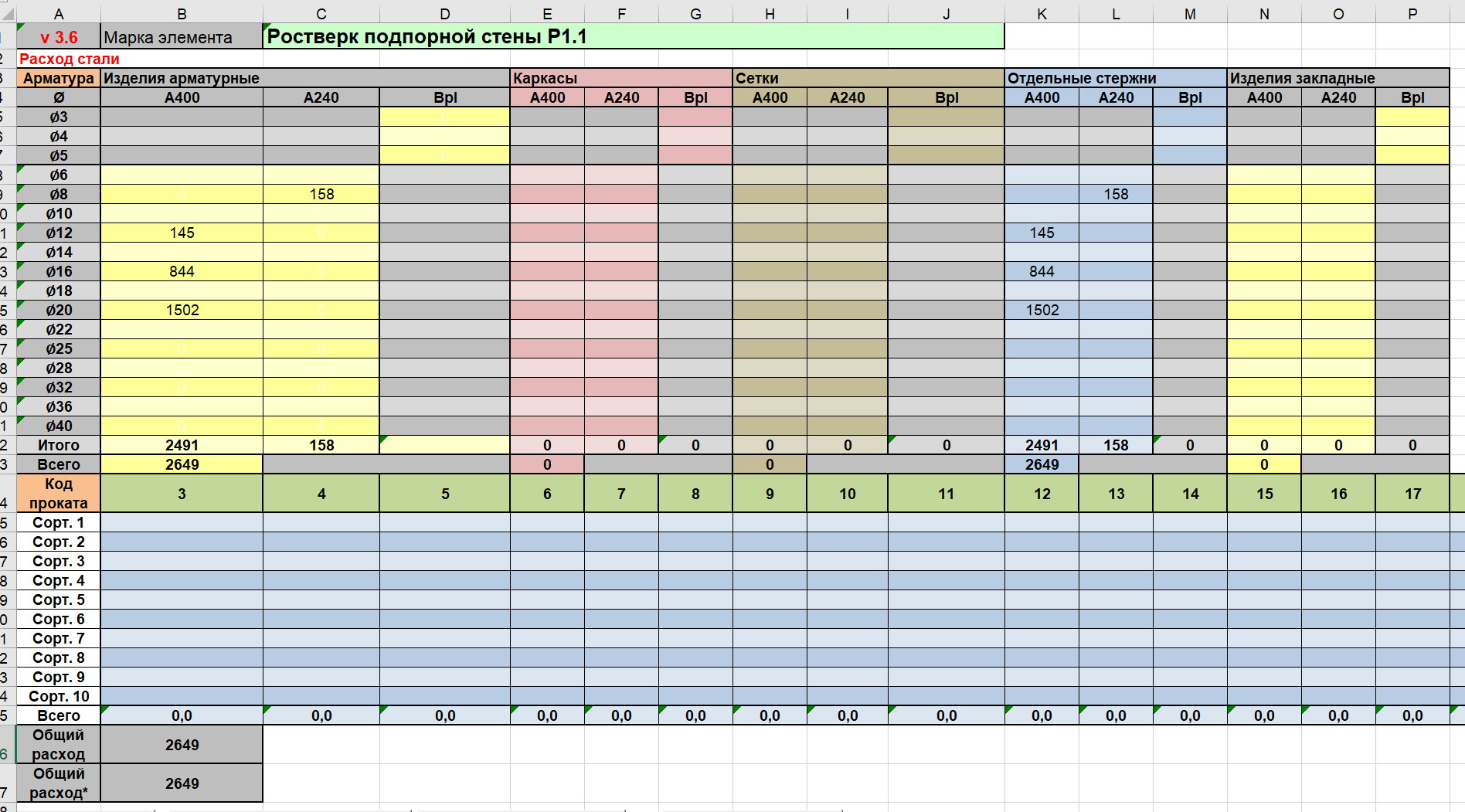


Рис.10 – Фрагмент расчетного листа\* объемов арматуры ростверка Р1.1 секции подпорной стены в файле ARM\_ПСТ

\* - не разрабатывалось в рамках магистерской диссертации.

## **4.2. Формирование ведомости объемов работ**

После заполнения и подсчета всех объемов в файле Data, можно приступить к формированию одного из исходных продуктов – Ведомости объемов работ (далее ВОР). Это отдельный файл Excel , который не содержит в себе ни одной расчетной формулы, а только лишь ссылки на файл Data , в котором уже есть все необходимые исходные данные и объемы для заполнения ВОР. Это было сделано специально, для того чтобы между файлами, на основе которых производится построение и заполнение основных чертежей, и файлами Ведомостей объемов работ не допускалось никаких разночтений.

Файл ВОР не был включен в состав Data отдельным листом, а вынесен как самостоятельный документ, исходя из производственной необходимости.

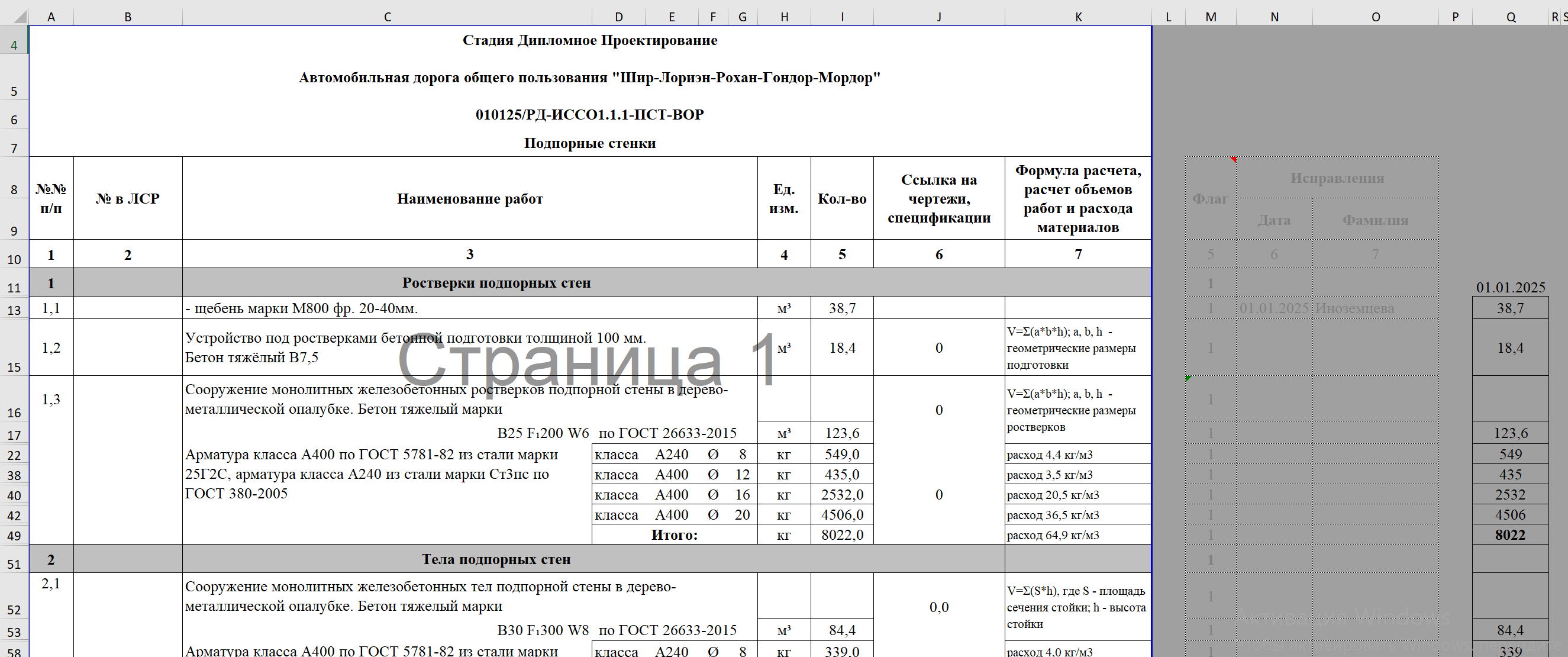


Рис.11 – Фрагмент листа из файла ВОР\_ПС

Как можно заменить, в этом файле так же предусмотрен протокол скрытия нулевых объемов. Так же можно обратить внимание на задублированные объемы работ в столбце «Q». Это было сделано для того, чтобы можно было отслеживать внесенные изменения по датам. Они не выводятся на печать и необходимы только для контроля версий. Если какой-то объем в какой-то момент времени будет отличаться от предыдущей версии, он подсветится красным цветом с помощью условного форматирования для того, чтобы обратить на это внимание пользователя и исключить вероятность случайных изменений.

# **Глава 5. Программная реализация**

Программа делится на несколько модулей:

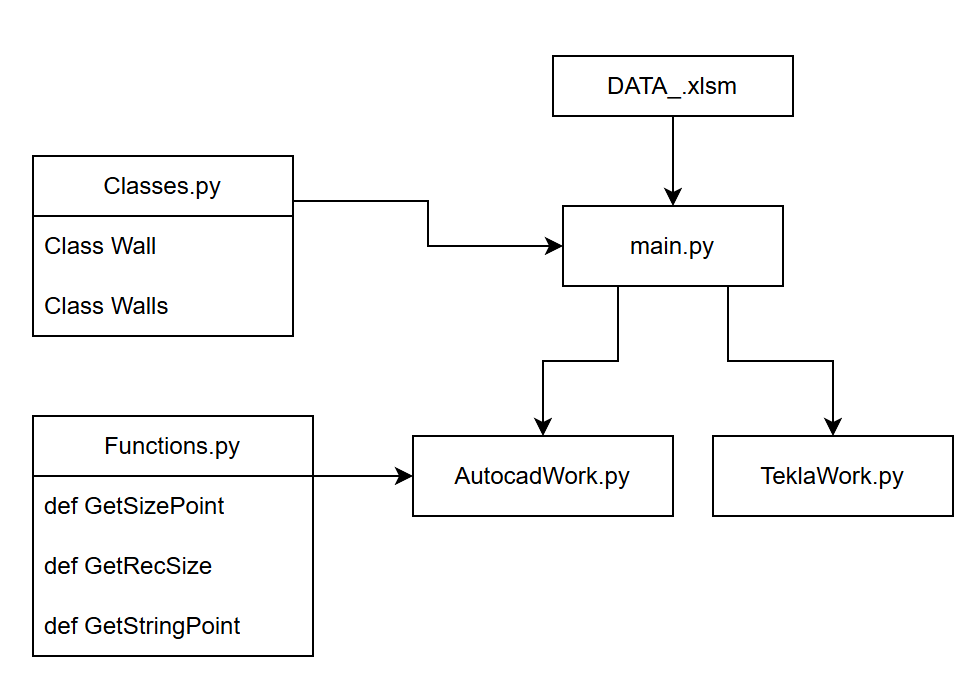


Рис.12– Схема взаимодействия модулей программы.

## **5.1. Выгрузка исходных данных**

Первым делом необходимо выгрузить данные для отрисовки опалубки секции подпорной стены. Для этого будет использована Python библиотека для работы с Excel – Openpyxl. Исходным является файл DATA\_.xlsm, для успешной выгрузки данных необходимо, чтобы файл находился в одной папке с исполняемым файлом и имел расширение .xlsm (из-за особенностей работы библиотеки).

Алгоритм считывания данных заключается в том, что после открытия файла исходных данных происходит последовательный проход по ячейкам начиная со стартовой и данные записываются в объект класса Wall. Число строк в столбце постоянно и каждая переменная имеет свой постоянный адрес. Число столбцов варьируется в зависимости от количества секций подпорной стены, и программа продолжает считывать их, пока не наткнется на пустой столбец.

Параллельно при вводе данных производится конвертация единиц всех линейных размеров из метров в миллиметры.

Каждая секция является объектом класса Wall, который включает в себя информацию из всех ячеек, представленных на Рис.2, описывающих положение и характеристики конкретной секции.

Программа работает со списком объектов класса Wall. Этот список хранится в классе Walls.

Листинг 1. Сlass Walls

|  |
| --- |
| class Walls(object):  name: str # Название сооружения  total\_count: int # Колличество секций в сооружении  sections = [] # Список классов(Wall) параметров секции  sections\_coors = [] # Список списков координат секций |

## **5.2. Отрисовка опалубочного чертежа подпорной стенки в Autocad**

После выгрузки необходимой информации начинается этап отрисовки чертежа в Autocad. Для работы с Autocad была использована библиотека PyAutocad. Для начала работы модуля необходимо открыть чертеж Autocad и зайти в пространство модели, иначе программа выдаст ошибку. Первым шагом пользователю необходимо задать точку вставки - incert\_point. Программа считает координаты, введенные пользователем и запишет только координату Х, а в качестве координаты У - будет использован параметр //foundation\_base. Это связано с необходимостью выстраивать опалубочный в Балтийской системе высот и получать все дальнейшие отметки в корректном формате.

### 5.2.1. Описание интерфейса и функционала

Построение видов производится исходя из топологии, она показывает какие точки должны соединиться между собой в отрезки и для всех секций одинаковая. Список координат для каждой секции тоже повторяется, а вот значения координат всегда разные.

Следующим этапом является заполнение списка координат для каждого вида опалубочного чертежа секции подпорной стенки. Для каждой секции записывается отдельный список координат с учетом заданных расстояний между видами и отсчитывается от нуля (см. Приложение ).

В качестве примера разберем фрагмент программы, который рассчитывает координаты вида 1-1

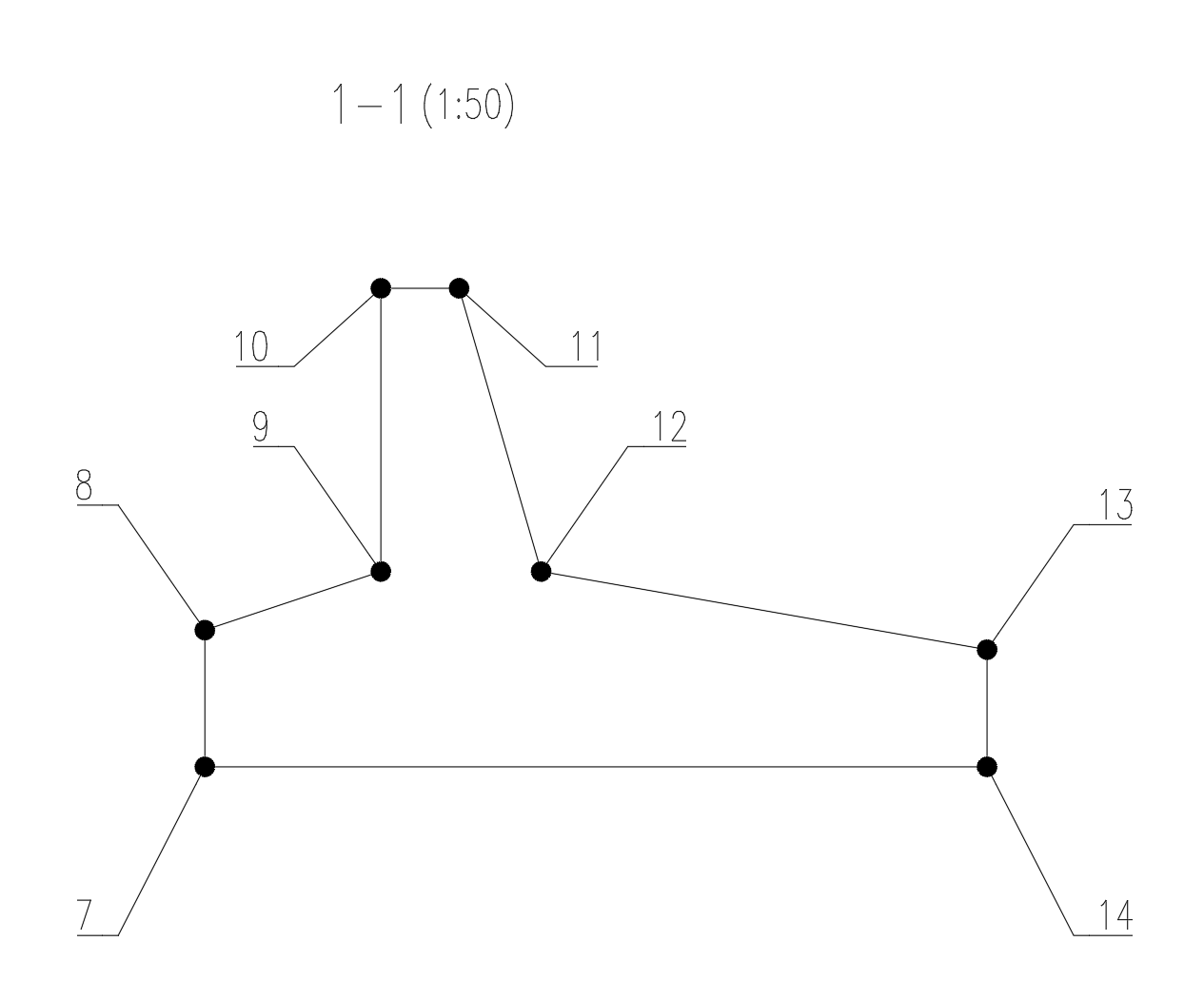


Рис.13 – Схема расстановки координат топологии вида 1-1

|  |
| --- |
| #Вид 1-1  #Задаем точку начала координат чтобы каждый вид считать от 0.0 а не прибавлять все расстояния в каждой точке  start\_coordinates = APoint(incert\_point.x + input\_data.leght + View\_l1, incert\_point.y)  SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y)) #7  SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y + input\_data.t2)) #8  SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance , start\_coordinates.y + input\_data.t1)) #9  SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance , start\_coordinates.y + input\_data.height\_start)) #10  SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance + input\_data.top\_wall\_width, start\_coordinates.y + input\_data.height\_start)) #11  SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance + input\_data.bottom\_wall\_width, start\_coordinates.y + input\_data.t3)) #12  SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.foundation\_width, start\_coordinates.y + input\_data.t4)) #13  SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.foundation\_width, start\_coordinates.y)) #14 |

Листинг 2. Расчет координат вида 1-1

Для каждого вида задается своя точка start\_coordinates. Это необходимо для того чтобы следующие координаты вида можно было отсчитывать не от общего нуля incert\_point, а от своего локального – левой нижней точки вида. Каждая точка start\_coordinates учитывает расстояние между видами внутри модели. Это расстояние задается с помощью переменных внутри кода программы.

Листинг 3. Переменные, описывающие расстояния между видами.

|  |
| --- |
| Line\_distance = 85000 #расстояние между двух видовых рамок секций  View\_l1 = 5000 #расстояние между фасадом и сечением 1-1 (по горизонтали)  View\_l2 = 5000 #расстояние сечением 1-1 и сечением 2-2 (по горизонтали)  View\_l3 = 10000 #расстояние между фасадом и планом (по вертикали) |

Далее производится заполнение массива топологии списками, содержащими в себе начало и конец отрезка.

Листинг 4. Заполнение массива топологии вида 1-1

|  |
| --- |
| Topology = []  #Вид 1-1  Topology.append([7,8])  Topology.append([8,9])  Topology.append([9,10])  Topology.append([10,11])  Topology.append([11,12])  Topology.append([12,13])  Topology.append([13,14])  Topology.append([14,7]) |

После составления списка координат и топологии начинается отрисовка видов с простановкой необходимых размеров и текста.

Каждый элемент чертежа должен располагаться на своем слое. Это необходимо для соблюдения стиля печати каждого элемента (контурные линии выводятся толще, чем размерные и т.д.). Чтобы слой мог быть выбран – он должен уже существовать в модели.

Для переключения слоя используется переменная:

acad.doc.ActiveLayer = acad.doc.Layers.Item ("Contur")

### 5.2.2. Отрисовка видов и обозначений

Отрисовка фронтального, профильного и поперечного сечений с учетом масштаба делится на несколько этапов:

1. Отрисовка контура видов. Производится в слое “Contur”

Отрисовка производится отрезками с помощью метода:

acad.model.AddLine (start\_point, end\_point)

метод принимает начальную и конечную точку отрезка, которые являются объектами класса APoint – класс библиотеки PyAutocad.

1. Простановка размеров и отметок уровня. Производится в слое “Size”

Отрисовка размерный линий производится в двух масштабах (1:100; 1:50), соответственно в двух размерных стилях 'LIN100' и 'LIN50'. Для корректного вывода я создала функцию GetSizePoint. Она принимает две координаты – начало и конец размерной линии, направление размера (задается через угол в радианах) и местоположение размерной линии (задается длиной). Результатом работы функции является вывод трех координат, необходимых для простановки размера.

Отрисовка размерных линий производится с помощью метода AddDimRotated, который принимает три координаты и угол направления размера. Это необходимо для более точного позиционирования размера в режиме линейной отрисовки. Метод, принимающий только три точки – отрисовывает параллельный размер.

Простановка отметок уровня производится вставкой в необходимые точки, заранее созданного в автокаде аннотативного блока отметки 'Otmetka'. Поскольку блок аннотативный – менять его масштаб в зависимости от вида нельзя. Он вставляется в чертеж только в масштабе 1:1. Далее уже производится настройка списка его аннотативных масштабов уже внутри автокада, если требуется. Поэтому при вставке, если список аннотативных масштабов не настроен, блок отметки может оказаться очень маленьким и незаметным на чертеже. Вставка блока производится с помощью метода: acad.model.InsertBlock().

1. Создание таблицы ведомости объемов работ. Производится в слое “T\_Border”

Первым этапом создается сама таблица. Стоит отметить что первая строка таблицы создается сразу объединенной с помощью метода: acad.model.AddTable().

Позиционируется таблица в правом верхнем углу ограничивающей рамки в пространстве модели. Далее пользуемся методами: tabtable.SetColumnWidth (0, 14500), table.SetRowHeight (0, 1500) для редактирования размеров строк и столбцов, поскольку при создании таблицы – задать эти параметры нельзя. По тому же принципу производится заполнение таблицы и выставление стиля и высоты текста. Для заполнения таблицы применяется метод:

tabtable.SetText (1, 1 , 'Ед.изм.').

Далее циклом проходимся по всей таблице для того, чтобы отредактировать высоту и стиль текста.

Листинг 5. Редактирование высоты текста и стиля в таблице

|  |
| --- |
| table\_text\_height=350  for row in range(table.Rows):  for col in range(table.Columns):  table.SetCellTextStyle(row, col, "RS0.7")  table.SetCellAlignment(row, col,5)  table.SetCellTextHeight(row, col, table\_text\_height) |

1. Простановка названий видов. Производится в слое “Text\_B”

Названия видов выводятся просто многострочным текстом на заданном расстоянии от вида с помощью метода: acad.model.AddMText(). Параметры текста так же задаются отдельно.

Листинг 6. Редактирование высоты текста и стиля в многострочном тексте

|  |
| --- |
| acad.doc.ActiveLayer = acad.doc.Layers.Item("Text\_B")  text\_style = acad.doc.TextStyles.Item("RS0.7")  acad.doc.ActiveTextStyle = text\_style  to\_view\_dis = 2500 # Расстояние от вида до его текста |

1. Отрисовка рамки – границы всех чертежей конкретной секции. Производится в слое “Notes”, не выводящимся на печать.

Рамка рисуется отрезками, но для того чтобы их отрисовать, сначала необходимо рассчитать все необходимые размеры и координаты. Для этого я создала функцию “ GetRecSize ”, которая принимает размеры отступов границ рамки от видов и максимальные размеры самих видов, после чего высчитывает размеры для отрезков.

Листинг 6. Функция расчета параметров рамки

|  |
| --- |
| def GetRecSize(incert\_point, max\_width, max\_height):  vertical\_otstyp = 15000  horizontal\_otstyp\_left = 10000  horizontal\_otstyp\_right = horizontal\_otstyp\_left \* 3  H = max\_height + vertical\_otstyp \* 2  B = max\_width + horizontal\_otstyp\_left + horizontal\_otstyp\_right  Point = APoint(incert\_point.x - horizontal\_otstyp\_left, incert\_point.y - max\_height - vertical\_otstyp )  rez = [Point, H, B]  return rez |

Дополнительно стоит отметить, что в программа учитывает переменную высоту подпорной стенки при отрисовки видов. Если у секции постоянная высота, то программа не отрисует вид 2-2 потому что в нем нет необходимости.

На этом заканчивается работа с Autocad. Далее программа завершает свою работу.

## **5.3. Создание BIM - модели в Tekla Struktures**

Для того чтобы начать работу с Tekla Structures, необходимо инициализировать повторный вызов программы.

Первым этапом работы модуля TeklaWork.py является подключение библиотек. Для корректной работы программы файлы библиотеки API Tekla: Tekla.Structures.dll, Tekla.Structures.Drawing.dll, Tekla.Structures.Model.dll- должны находиться в папке вместе с остальными файлами программы. Для подключения библиотек, первым делом необходимо получить полный путь к файлу проекта. Для этого из библиотеки pathlib импортируется класс Path из которого используется метод Path.cwd().

После чего вручную прописывается “адресс” каждого dll-файла: clr.AddReference(str(tekla\_path)+ "/Tekla.Structures.dll"). Далее производится импорт необходимых элементов модели из библиотек таких как: модель, балка, точка, положение и т.д.

### 5.3.1. Отрисовка секции подпорной стены

Функция, отвечающая за отрисовку в Tekla, называется InputIKomponent(). Она принимает объект класса Walls - список секций подпорной стенки. Для начала отрисовки компонентов необходимо сначала подключиться к пространству модели. В программе предусмотрена проверка на наличие Tekla на компьютере пользователя и успешное подключение к модели.

Листинг 7. Подключение к пространству модели в Tekla Structures

|  |
| --- |
| def InputIKomponent(input\_datas):  model = Model()  if not model.GetConnectionStatus():  print("Не удалось подключиться к модели Tekla")  exit() |

Следующим этапом необходимо циклом пройтись по каждой секции и создать в Tekla секцию подпорной стены. Для этого необходимо создать элемент Beam с заданным стандартным профилем RCRW. Профиль можно изменить, отредактировав каждый параметр по отдельности в строке описания профиля, где каждая цифра обозначает определенный размер поперечного сечения:

RCRW5000\*4000-400\*900-820-1145\*700-1000\*600

При вставке балки, в программе предусмотрена проверка на успешную вставку. Результат проверки выводится пользователю на экран.

Для позиционирования балки в пространстве модели, используются две координаты – начальная и конечная по центру балки. Они заданы в Excel – файле Data и являются координатами локальной системы координат. Это связано с ограниченностью пространства модели в Tekla тремя километрами. Если превысить это допустимое значения для того, чтобы позиционировать балку в Мировой системе координат, как мы это делаем в Autocad при создании общего плана сооружения – в Tekla будут возникать ошибки. Чтобы правильно позиционировать балку в Мировой системе координат, в свойствах проекта Tekla необходима задать параметры базовой точки, которая является началом локальных координат пространства модели.

Положение балки по глубине задается отдельным параметром.

Листинг 8. Установка положения балки по глубине в позицию "Спереди"

|  |
| --- |
| position = getattr(Position.DepthEnum, 'FRONT')  beam.Position.Depth = position |

При вставке, балка создается с постоянной высотой, что не всегда соответствует действительности. Поэтому необходимо задать обрезку балки по высоте. Для этого необходимо вставить еще одну балку большего сечения, которая будет выступать обрезающим объектом, а в последствии антителом.

### 5.3.1. Корректировка секции подпорной стены

Поскольку подпорная стенка задается через балку – после создания она всегда имеет постоянную высоту. В программной реализации ей присваивается максимальная высота из двух возможных. Для корректного отображения секции – ее необходимо обрезать. Tekla Structures предлагает несколько способов обрезки: через секущую плоскость, фаску и через вычитание одного объекта из другого. Остановимся на последнем варианте, так как он самый простой в реализации и самый удобный по мнению команды BIM – проектировщиков.

Для реализации обрезки секции подпорной стенки методом вычитания объекта –необходимо задать этот самый обрезающий объект. В данном случае это будет тоже балка, но с упрощенным вариантом сечения –квадрат. Ему присваиваются размеры, превышающие габариты подпорной стены. Для обрезающей балки задаются так же две точки по средней линии - начальная и конечная. По Х и У они совпадают с координатами самой стенки, а отметка высоты задается через начальную и конечную высоту секции.

После создания обрезающей балки и заполнения ее параметров –производится вычитание.

Листинг 9. Обрезка секции подпорной стенки

|  |
| --- |
| boolean\_part = BooleanPart()  boolean\_part.Father = beam  boolean\_part.SetOperativePart(cut\_beam)  boolean\_part.Type = BooleanPart.BooleanTypeEnum.BOOLEAN\_CUT  boolean\_part.Insert()  model.CommitChanges()  model.GetModelObjectSelector().GetAllObjects().SelectInstances = False  cut\_beam.Delete()  model.CommitChanges(); |

После обрезки, секущую балку необходимо стереть.

# **Глава 6. Заключение**

## **6.1. Итоги работы**

В ходе выполнения магистерской диссертации на тему «Автоматизация проектирования железобетонных подпорных стен уголкового типа» была достигнута поставленная цель – разработана программная система, позволяющая ускорить и стандартизировать процесс проектирования подпорных стен с минимальным участием инженера – проектировщика.

Ключевые результаты работы:

1. Анализ существующих решений

* Изучены нормативные требования (СП, ГОСТ ) и методы расчёта подпорных стен.
* Проведён критический обзор современных САПР (Grasshopper, Tekla), выявлены их ограничения в части автоматизации чертежей.

1. Разработка методики автоматизированного проектирования

* Обработаны результаты расчёта устойчивости, прочности и армирования уголковых стен.
* Систематизировано хранение и обмен исходными данными между программными комплексами

1. Создание программного обеспечения

Реализовано приложение, интегрирующее Excel → AutoCAD→ Tekla:

* Excel – ввод исходных данных (высота стены, нагрузки, марка бетона, армирование).
* Tekla – генерация 3D-модели для визуализации и проверки коллизий.
* AutoCAD – автоматическая отрисовка опалубочных чертежей с размерами и спецификациями.

## **6.2. Перспективное развитие программы и ее дальнейшее применение**

Эта программа разработана для автоматизации процесса выпуска рабочей документации по комплектам монолитных железобетонных подпорных стен уголкового типа на базе проектного института Гипростроймост.

В первую очередь она предназначена для сокращения времени на работу с рутинными задачами, которые циклично повторяются в больших количествах и практически не претерпевают изменений. В частности, это отрисовка опалубочных чертежей. Опалубочные чертежи — это детализированные схемы, по которым изготавливается опалубка для бетонирования подпорной стены. Они должны точно отражать геометрию конструкции и ее положение в пространстве, расположение закладных элементов и технологические особенности бетонирования.

Подпорные стены часто имеют типизированное поперечное сечение лишь с небольшими изменениями (высота, длина, отметка подошвы фундамента, армирование). Ручное черчение занимает много времени и повышает риск ошибок, связанных с не одновременным внесением изменений в различные составные части проекта. Можно легко поменять геометрию секции в чертеже Autocad, но не изменить эти данные в ведомости объемов работ и наоборот. В таком случае, данные на разных листах проекта будут отличаться друг от друга и придется затрачивать дополнительные усилия на перепроверку и установление достоверных значений.

Таким образом - основным посылом создания этой программы является желание не только увеличить скорость разработки документации, но стремление добиться соответствия между собой всей информации, представленной в проекте. Так же разработка всех последующих схожий проектов на основе одной программы позволит добиться полного единообразия оформления чертежей по всему проектному институту и не допускать разночтений в пределах одного объекта строительства.

В дальнейшем я вижу развитие разработанной программы в добавлении отрисовки не только опалубочных, но и арматурных чертежей. Вся информация, необходимая для определения размера и позиционирования каждого стержня внутри контуров подпорной стены, уже содержится в данном проекте. Конфигурацию они имеют практически идентичную от стенки к стенке и могут нуждаться лишь в незначительной корректировке вручную. На данном этапе разработки данная функция не рассматривалась ввиду большей детализации арматурного чертежа по сравнению с опалубочным, что влечет за собой большую трудоемкость разработки.

Еще одним этапом доработки я вижу возможность добавления в проект возможности позиционирования и отрисовки свайного поля. Да, подпорные стенки тоже могут быть со свайным фундаментом. В чем заключается практический интерес автоматизации? В возможности автоматической расстановки свай с соблюдением необходимых требований СП 22.13330.2016 – «Основания зданий и сооружений» и учетом необходимых расчетных характеристик.

Так же я бы хотела в будущем развить эту программу и добавить в нее отрисовку ростверков и опор, так как они тоже имеют практически идентичную конфигурацию и в одном отдельном сооружении их может быть довольно большое количество.

Так же хотелось бы добавить в программу возможность работы с динамическими блоками и напрямую обращаться и записывать их атрибуты. Это позволит заметно расширить функционал и добавить отрисовку таких вещей как: обозначение вида сечения, мультивыноски и прочие динамические блоки.

# **Список литературы:**

1. Программирование на Python в примерах и задачах. Васильев А. Н, 2022.
2. Python для Excel. Феликс Зумштейн. Перевод СПб: БХВ-Петербург, 2023.
3. Grasshopper. Учебник для начинающих. Основы. Редакция V3.2. Перевод Д. Булка, 2022. – https://free-stl.ru/manual/GrasshopperPrimer\_V3-3\_RU-low.pdf
4. Расчет подпорных стен. Методические указания к курсовой работе по дисциплине "Транспортные сооружения".Щуко С.А. , Вихрев А.В.. Владимир 2002г.
5. СП 22.13330.2016. Свод правил. Основания зданий и сооружений. Актуализированная редакция СНиП 2.02.01-83
6. СП 381.1325800.2018 «Сооружения подпорные. Правила проектирования»
7. Основания и фундаменты. Кириллов В.С. 1980

# **Приложения:**

## **Приложение 1. Общий вид исходных данных - файл Data**

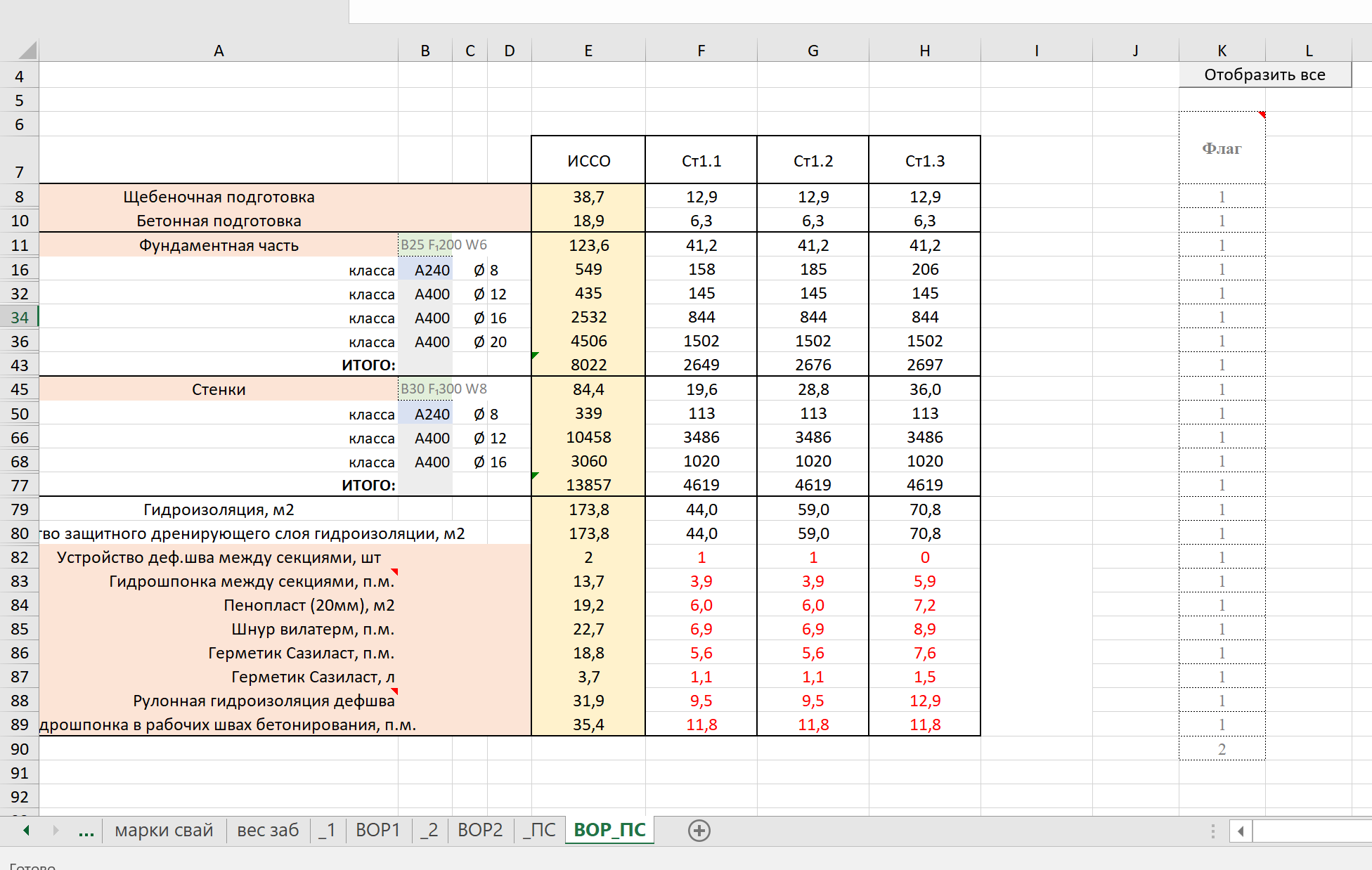
Общий вид вкладки \_ПС в Excel файле Data со всеми исходными данными\* и параметрами искусственного сооружения ИССО1.1.1 и его составляющих.



\* - зеленые ячейки заполняются вручную, белые – вычисляются автоматически.

## **Приложение 2. Ведомость объемов работ - файл Data**

Общий вид вкладки ВОР\_ПС в Excel файле Data со всеми расчетными и ссылочными значениями объемов.



## **Приложение 3.** **Общий вид вкладки Р1.1**

Общий вид вкладки Р1.1 в Excel файле\* ARM\_ПСТ - расчетного листа объемов арматуры ростверка Р1.1 секции подпорной стены.



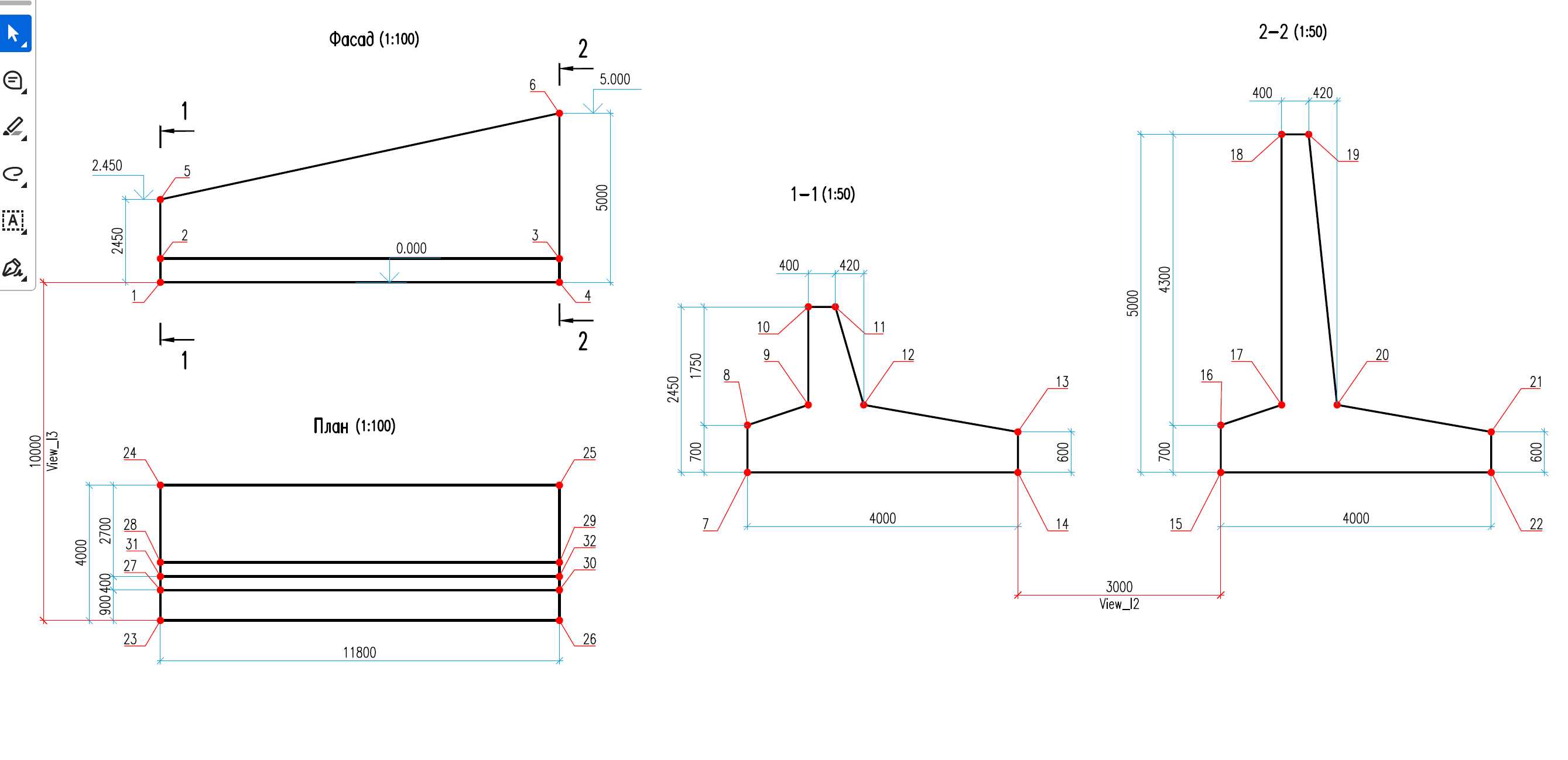
\* - не разрабатывалось в рамках магистерской диссертации, но послужило вдохновением для ее создания.

## **Приложение 4.** **Общий вид вкладки ВОР в Excel файле ВОР\_ПСТ**

Является готовым документом, подлежащим обязательному приложению в состав томов рабочей документации.

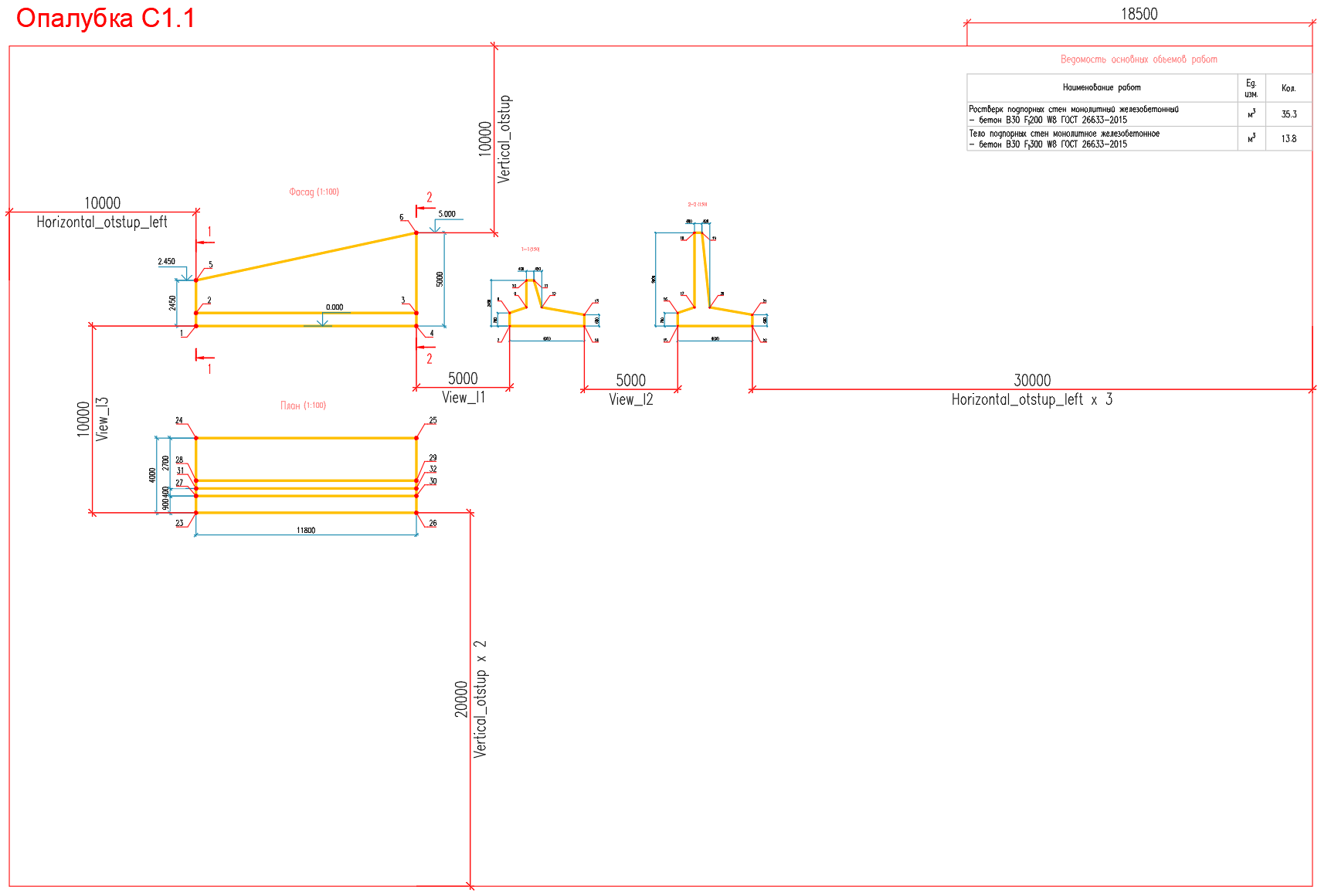


## **Приложение 5. Топология всех видов опалубочных чертежей**



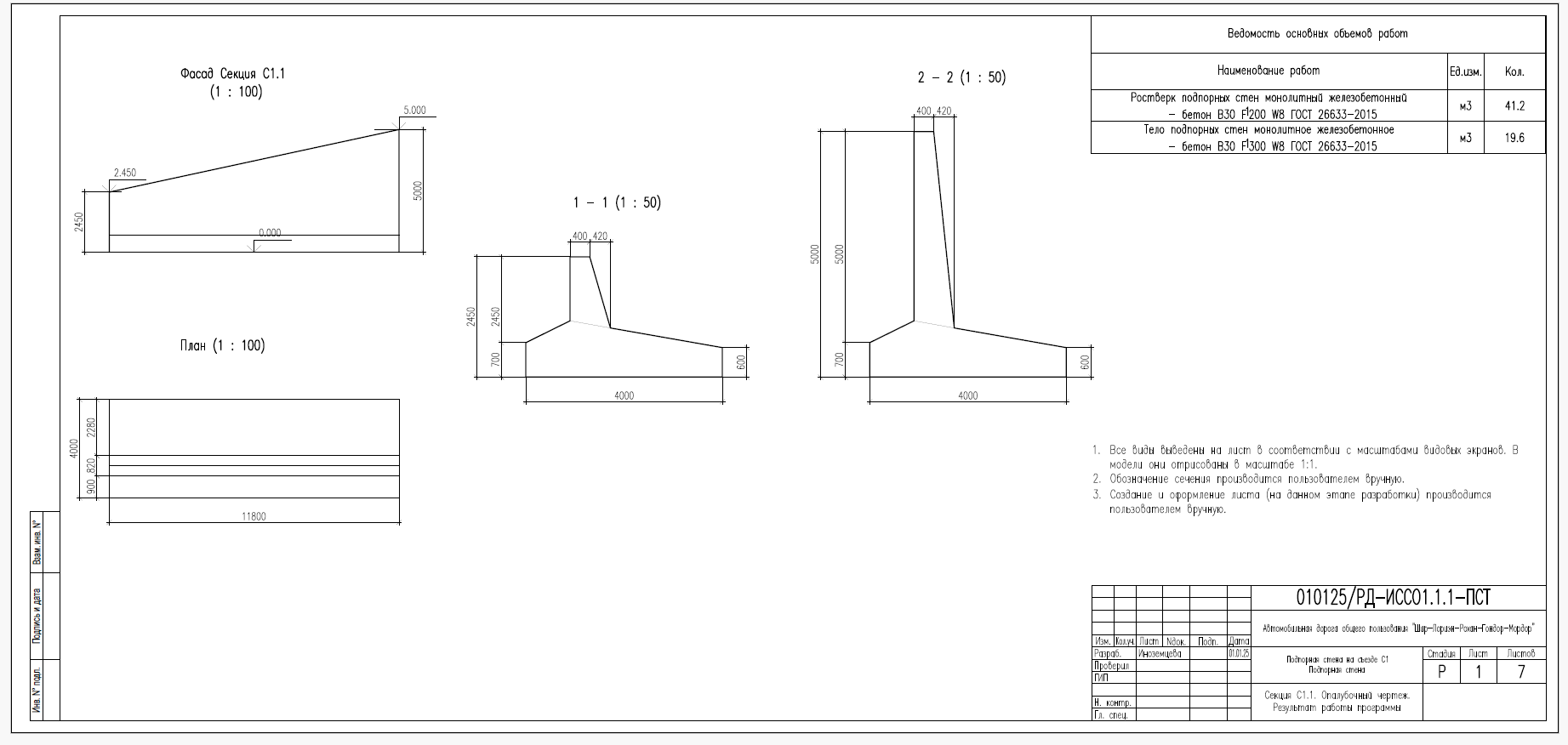
## **Приложение 6.** **Макет чертежа**

Общий вид макета чертежа в пространстве модели с нанесением всех вспомогательных размеров и отметок.



## **Приложение 7. Результат работы модуля AutocadWork.py**

Общий вид готового чертежа в пространстве листа



## **Приложение 8. Результат работы модуля TeklaWork.py**

## **Приложение 9. Файл Stena.py**

from Classes import Wall,Walls

from pathlib import Path # Библиотека для указания пути к текущей папке

import Functions

import openpyxl

import AutocadWork

# Описание файла и поожения данных для выгрузки в питон

file\_Name=str(Path.cwd())+"/Excel\_Autocad\_project/DATA\_.xlsm" #"C:/Users/Ponka/Desktop/Диплом/Diplom2/DATA\_.xlsm"

sheet\_Name="\_ПС"

read\_row=3 #Стартовая строка

read\_column=5 #Стартовый столбец считывания данных по секции

workbook = openpyxl.load\_workbook(file\_Name, data\_only=True )

sheet=workbook[sheet\_Name]

input\_datas=Walls()

input\_datas.name=sheet.cell(row=read\_row,column=read\_column-1)

input\_datas.total\_count=0

print("Начало считывания данных с Ексель")

while sheet.cell(row=read\_row,column=read\_column).value != None:#"None"

input\_data = Wall()

input\_data.SetData(sheet,read\_column)

input\_datas.total\_count += 1

input\_datas.sections.append(input\_data)

read\_column+=1

print("Отрисовать подпорные стенки?\n1. В Autocad \n2. В Tekla Structures")

choice=int(input())

if choice == 1:

AutocadWork.DrawAutocad(input\_datas)

if choice == 2:

try:

import TeklaWork

TeklaWork.InputIKomponent(input\_datas)

except:

print("На компьютере отсутствует Текла или в папке нет одного из его модулей")

input("Нажмите Enter для завершения программы")

## **Приложение 10. Файл Classes.py**

import openpyxl

class Coor(object):

#pass

X:float

Y:float

def \_\_init\_\_(self):

self.X=0

self.Y=0

def \_\_init\_\_(self,x,y):

self.X=x

self.Y=y

def Print(self):

print(str(self.X)+", " + str(self.Y))

class Wall(object):

name: str # Имя стенки

name\_wall:str

name\_founding:str

count:int

foundation\_base:float # Отметка подошвы фундамента

leght: float # Длина секции, м - L

height\_end: float # Высота в конце стенки - H\_кон

height\_start: float # Высота в начале стенки - H\_нач

foundation\_width: float #Ширина ф-та - В

top\_wall\_width: float #ширина стены наерху - в2

edge\_distance: float #расстояние от стены до края - в3

bottom\_wall\_width: float #ширина стены внизу - в4

t1: float #толщина перекрытия 1 у стены

t2: float #толщина перекрытия 1 у насыпи

t3: float #толщина перекрытия 2 у стены

t4: float #олщина перекрытия 2 у насыпи

V1: float #объем ростверка подпорной стены, м3

V2: float #объем стеновой части подпорной стены, м3

tek\_start\_x:float

tek\_start\_y:float

tek\_end\_x:float

tek\_end\_y:float

def SetData(self, sheet,read\_column):#filename, sheetName

# workbook = openpyxl.load\_workbook(filename, data\_only=True )

# sheet=workbook[sheetName]

read\_row=3

#read\_column=5

self.name=sheet.cell(row=read\_row,column=read\_column).value#имя секции

self.name\_founding=sheet.cell(row=read\_row+1,column=read\_column).value

self.name\_wall=sheet.cell(row=read\_row+3,column=read\_column).value

self.count=sheet.cell(row=read\_row+5,column=read\_column).value

self.foundation\_base=sheet.cell(row=read\_row+6,column=read\_column).value

self.leght=sheet.cell(row=read\_row+7,column=read\_column).value\*1000

self.height\_start=sheet.cell(row=read\_row+8,column=read\_column).value\*1000

self.height\_end=sheet.cell(row=read\_row+9,column=read\_column).value\*1000

self.foundation\_width=sheet.cell(row=read\_row+10,column=read\_column).value\*1000

self.top\_wall\_width=sheet.cell(row=read\_row+11,column=read\_column).value\*1000

self.edge\_distance=sheet.cell(row=read\_row+12,column=read\_column).value\*1000

self.bottom\_wall\_width=sheet.cell(row=read\_row+13,column=read\_column).value\*1000

self.t1=sheet.cell(row=read\_row+14,column=read\_column).value\*1000

self.t2=sheet.cell(row=read\_row+15,column=read\_column).value\*1000

self.t3=sheet.cell(row=read\_row+16,column=read\_column).value\*1000

self.t4=sheet.cell(row=read\_row+17,column=read\_column).value\*1000

self.V1=sheet.cell(row=read\_row+18,column=read\_column).value

self.V2=sheet.cell(row=read\_row+19,column=read\_column).value

#

self.tek\_start\_x = (sheet.cell(row=read\_row+25,column=read\_column).value+sheet.cell(row=read\_row+27,column=read\_column).value)/2

self.tek\_start\_y = (sheet.cell(row=read\_row+26,column=read\_column).value+sheet.cell(row=read\_row+28,column=read\_column).value)/2

self.tek\_end\_x = (sheet.cell(row=read\_row+29,column=read\_column).value+sheet.cell(row=read\_row+31,column=read\_column).value)/2

self.tek\_end\_y = (sheet.cell(row=read\_row+30,column=read\_column).value+sheet.cell(row=read\_row+32,column=read\_column).value)/2

def ShowAll(self):

print("Имя секции: "+self.name)

print("Имя ростверка: "+self.name\_founding)

print("Имя стенки: "+self.name\_wall)

print("Отметка подошвы: "+str(self.foundation\_base))

print("Колличество секций: "+str(self.count))

print("Длина: "+str(self.leght))

print("Общая высота (нач): "+str(self.height\_start))

print("Общая высота (кон): "+str(self.height\_end))

print("Ширина фундамента: "+str(self.foundation\_width))

print("Ширина стены сверху : "+str(self.top\_wall\_width))

print("Расстояние от стены до края: "+str(self.edge\_distance))

print("Ширина стены внизу: "+str(self.bottom\_wall\_width))

print("Толщина перекрытия 1 у стены: "+str(self.t1))

print("Толщина перекрытия 1 у насыпи: "+str(self.t2))

print("Толщина перекрытия 2 у стены: "+str(self.t3))

print("Толщина перекрытия 2 у насыпи: "+str(self.t4))

print("Объем ростверка подпорной стены, м3: "+str(self.V1))

print("Объем тела подпорной стены, м3: "+str(self.V2))

class Walls(object):

name: str # Название сооружения

total\_count: int # Колличество секций в сооружении

sections=[] # Список классов(Wall) параметров секции

sections\_coors=[] # Список списков координат секций

## **Приложение 11. Файл Functions.py**

import math

from pyautocad import APoint

#Функция для работы с размерами.

# PointST - APoint начальной точки размера

# PointEND - APoint конечной точки размера

# TypeSizeLine - Int тип - размера (горизонтальный 0 / вертикальный 1)

# PositionSizeLine - Float отступ - положение размерной линии (<0 - размер идет влево или вниз, >0 вправо или вверх)

def GetSizePoint (PointST, PointEND, TypeSizeLine, PositionSizeLine):

rez = APoint(0,0)

# TypeSizeLine - угол в радианах - указатель поаворота размера

# PositionSizeLine - расстояние - местоположение размерной линии

if TypeSizeLine == 0 or TypeSizeLine == math.pi:

rez.x = (PointEND.x - PointST.x)/2

if TypeSizeLine == 0:

rez.y = min(PointST.y, PointEND.y) - PositionSizeLine

else: rez.y = max(PointST.y, PointEND.y) + PositionSizeLine

if TypeSizeLine == math.pi/2 or TypeSizeLine == 3\*math.pi/2:

rez.y = (PointEND.y - PointST.y)/2

if TypeSizeLine == math.pi/2:

rez.x = min(PointST.x, PointEND.x) - PositionSizeLine

else: rez.x = max(PointST.x, PointEND.x) + PositionSizeLine

return rez

# Функция расчета параметров рамки

def GetRecSize(incert\_point, max\_width, max\_height):

vertical\_otstyp = 15000

horizontal\_otstyp\_left = 10000

horizontal\_otstyp\_right = horizontal\_otstyp\_left \* 3

H = max\_height + vertical\_otstyp \* 2

B = max\_width + horizontal\_otstyp\_left + horizontal\_otstyp\_right

Point = APoint(incert\_point.x - horizontal\_otstyp\_left, incert\_point.y - max\_height - vertical\_otstyp )

rez = [Point, H, B]

return rez

#

def ConvertAdress(input\_adress):

print("Convert adress")

def GetStringPoint(point1, point2,up\_dist):

rez=APoint(0,0)

rez.x=point1.x+(point2.x-point1.x)/4

rez.y=max(point1.y,point2.y)+up\_dist

return rez

## **Приложение 12. Файл AutocadWork.py**

try:

from pyautocad import Autocad, APoint

from Classes import Wall,Walls

import math

import Functions

def DrawAutocad(input\_datas):

# --\\ОТРИСОВКА\\--

acad= Autocad(create\_if\_not\_exists=False)

print("Необходимо ввести точку вставки, пожалуйста перейдите в AutoCad")

start\_point = acad.doc.Utility.GetPoint(APoint(0, 0), "Введите точку вставки: ")# ввод от пользователя точки вставки картинки в автокад

Line\_distance = 85000 #расстояние между двух видовых рамок секций

View\_l1 = 5000 #расстояние между фасадом и сечением 1-1 (по горизонтали)

View\_l2 = 3000 #расстояние сечением 1-1 и сечением 2-2 (по горизонтали)

View\_l3 = 10000 #расстояние между фасадом и планом (по вертикали)

i=0

for input\_data in input\_datas.sections:

incert\_point=APoint(start\_point[0]+Line\_distance\*i,input\_data.foundation\_base)#Точка вставки - левый нижний угол фасада подпорной стены

i+=1

SECTION\_Coor = [] #список всех координат видов в одной секции стенки

SECTION\_Coor.append(APoint(0,0)) #0 - фиктивная точка 00 для начала нумерации нормальных точек с единицы

SECTION\_Coor.append(incert\_point)#1

#Фасад

SECTION\_Coor.append(APoint(incert\_point.x, incert\_point.y + input\_data.t2))#2

SECTION\_Coor.append(APoint(SECTION\_Coor[2].x + input\_data.leght, SECTION\_Coor[2].y))#3

SECTION\_Coor.append(APoint(SECTION\_Coor[1].x + input\_data.leght, SECTION\_Coor[1].y))#4

SECTION\_Coor.append(APoint(incert\_point.x, incert\_point.y + input\_data.height\_start))#5

SECTION\_Coor.append(APoint(incert\_point.x + input\_data.leght, incert\_point.y + input\_data.height\_end))#6

#Вид 1-1

#Задаем точку НК чтобы каждый вид считать от 0.0 а не прибавлять все расстояния в каждой точке

start\_coordinates = APoint(incert\_point.x + input\_data.leght + View\_l1, incert\_point.y)

print("start coordinate: " + str(start\_coordinates.x) +" " + str(start\_coordinates.y))

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y))#7

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y + input\_data.t2))#8

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance , start\_coordinates.y + input\_data.t1))#9

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance , start\_coordinates.y + input\_data.height\_start))#10

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance + input\_data.top\_wall\_width, start\_coordinates.y + input\_data.height\_start))#11

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance + input\_data.bottom\_wall\_width, start\_coordinates.y + input\_data.t3))#12

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.foundation\_width, start\_coordinates.y + input\_data.t4))#13

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.foundation\_width, start\_coordinates.y))#14

#Вид 2-2

#Задаем точку НК чтобы каждый вид считать от 0.0 а не прибавлять все расстояния в каждой точке

start\_coordinates = APoint(incert\_point.x + input\_data.leght + View\_l1+ View\_l2+ input\_data.foundation\_width, incert\_point.y)

#print("start coordinate: " + str(start\_coordinates.x) +" " + str(start\_coordinates.y))

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y))#15

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y + input\_data.t2))#16

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance , start\_coordinates.y + input\_data.t1))#17

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance , start\_coordinates.y + input\_data.height\_end))#18

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance + input\_data.top\_wall\_width, start\_coordinates.y + input\_data.height\_end))#19

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.edge\_distance + input\_data.bottom\_wall\_width, start\_coordinates.y + input\_data.t3))#20

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.foundation\_width, start\_coordinates.y + input\_data.t4))#21

SECTION\_Coor.append(APoint(start\_coordinates.x + input\_data.foundation\_width, start\_coordinates.y))#22

#План

start\_coordinates = APoint(incert\_point.x, incert\_point.y - View\_l3)

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y))#23

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y + input\_data.foundation\_width))#24

SECTION\_Coor.append(APoint(SECTION\_Coor[24].x+input\_data.leght, SECTION\_Coor[24].y))#25

SECTION\_Coor.append(APoint(SECTION\_Coor[23].x+input\_data.leght, SECTION\_Coor[23].y))#26

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y+input\_data.edge\_distance))#27

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y+input\_data.edge\_distance+input\_data.bottom\_wall\_width))#28

SECTION\_Coor.append(APoint(SECTION\_Coor[28].x + input\_data.leght, SECTION\_Coor[28].y))#29

SECTION\_Coor.append(APoint(SECTION\_Coor[27].x + input\_data.leght, SECTION\_Coor[27].y))#30

SECTION\_Coor.append(APoint(start\_coordinates.x, start\_coordinates.y+input\_data.edge\_distance+input\_data.top\_wall\_width))#31

SECTION\_Coor.append(APoint(SECTION\_Coor[31].x+input\_data.leght, SECTION\_Coor[31].y))#32

input\_datas.sections\_coors.append(SECTION\_Coor)

#Массив топологии видов стенок

#Отрисовка ведется отрезками.Заполняем список мини-спиками - в которых записано начало и конец отрезка

Topology=[]

#Фасад

Topology.append([1,4])

Topology.append([2,3])

Topology.append([5,6])

Topology.append([1,5])

Topology.append([4,6])

#Вид 1-1

Topology.append([7,8])

Topology.append([8,9])

Topology.append([9,10])

Topology.append([10,11])

Topology.append([11,12])

Topology.append([12,13])

Topology.append([13,14])

Topology.append([14,7])

#Линии границы ф-та и стены на 1-1

Topology.append([9,12])

#План

Topology.append([23,24])

Topology.append([24,25])

Topology.append([25,26])

Topology.append([26,23])

Topology.append([28,29])

Topology.append([27,30])

Topology.append([31,32])

#Вид 2-2

Topology.append([15,16])

Topology.append([16,17])

Topology.append([17,18])

Topology.append([18,19])

Topology.append([19,20])

Topology.append([20,21])

Topology.append([21,22])

Topology.append([22,15])

#Линии границы ф-та и стены на 2-2

#Topology.append([17,20])

# Создание маcсива топологии для отрисовки размерных линий

# Все размеры создаются строго слева направо или снизу вверх!!!

# [[x,y,Тип размера, Величина отступа]]

SizeTopology=[]

#Размеры видов в масштабе 1:100

SizeStyle1 = 'LIN100'

SizeTopology.append([1, 5, math.pi/2, 1000,SizeStyle1])

SizeTopology.append([4, 6, 3\*math.pi/2, 1000,SizeStyle1])

SizeTopology.append([23, 26, 0, 1000,SizeStyle1])

SizeTopology.append([23, 24, math.pi/2, 1200,SizeStyle1])

SizeTopology.append([23, 27, math.pi/2, 500,SizeStyle1])

SizeTopology.append([27, 28, math.pi/2, 500,SizeStyle1])

SizeTopology.append([28, 24, math.pi/2, 500,SizeStyle1])

#Размеры сечений в масштабе 1:50

SizeStyle2 = 'LIN50'

SizeTopology.append([7, 10, math.pi/2, 1000,SizeStyle2])

SizeTopology.append([7, 14, 0, 500,SizeStyle2])

SizeTopology.append([8, 10, math.pi/2, 500,SizeStyle2])

SizeTopology.append([7, 8, math.pi/2, 500,SizeStyle2])

SizeTopology.append([10, 11, math.pi, 300,SizeStyle2])

SizeTopology.append([11, 12, math.pi, 300,SizeStyle2])

SizeTopology.append([14, 13,math.pi\*1.5, 500,SizeStyle2])

# Размерные линии вида 2-2

SizeTopology.append([15, 18, math.pi/2, 1000,SizeStyle2])

SizeTopology.append([16, 18, math.pi/2, 500,SizeStyle2])

SizeTopology.append([15, 16, math.pi/2, 500,SizeStyle2])

SizeTopology.append([15, 22, 0, 500,SizeStyle2])

SizeTopology.append([18, 19, math.pi, 300,SizeStyle2])

SizeTopology.append([19, 20, math.pi, 300,SizeStyle2])

SizeTopology.append([22, 21,math.pi\*1.5, 500,SizeStyle2])

#Отрисовка видов

index = 0 # Индекс для обращения к параметрам секции

for SECTION\_Coor in input\_datas.sections\_coors:

acad.doc.ActiveLayer = acad.doc.Layers.Item("Notes")

rec\_size=Functions.GetRecSize(SECTION\_Coor[1],(SECTION\_Coor[22].x-SECTION\_Coor[1].x),(SECTION\_Coor[6].y-SECTION\_Coor[26].y))

#command = f"\_-RECTANG \_non {rec\_size[0].x},{rec\_size[0].y} \_non {rec\_size[0].x + rec\_size[2]},{rec\_size[0].y + rec\_size[1]}\n"

#acad.doc.SendCommand(command)

points = [

APoint(rec\_size[0].x, rec\_size[0].y),

APoint(rec\_size[0].x + rec\_size[2], rec\_size[0].y),

APoint(rec\_size[0].x + rec\_size[2], rec\_size[0].y + rec\_size[1]),

APoint(rec\_size[0].x, rec\_size[0].y + rec\_size[1]),

APoint(rec\_size[0].x, rec\_size[0].y) # замыкаем прямоугольник

]

for it in range(1,len(points)):

acad.model.AddLine(points[it-1],points[it]) # Создание полилинии (прямоугольника)

acad.doc.ActiveLayer = acad.doc.Layers.Item("Contur")#установка слоя для отрисовки

i=0

for item in Topology:

start\_point=APoint(SECTION\_Coor[item[0]].x,SECTION\_Coor[item[0]].y)

end\_point=APoint(SECTION\_Coor[item[1]].x,SECTION\_Coor[item[1]].y)

acad.model.AddLine(start\_point,end\_point)

#Если высота подпорной стенки постоянная, то вид 2-2 не отрисовываем

if input\_datas.sections[index].height\_end == input\_datas.sections[index].height\_start and i == 20:

break

i+=1

#Отрисовка размерных линий

acad.doc.ActiveLayer = acad.doc.Layers.Item("Size") #установка слоя для отрисовки

i=0

for item in SizeTopology:

acad.doc.ActiveDimStyle = acad.doc.DimStyles.Item(item[4]) # Выставление необходимого размерного стиля

start\_point = APoint(SECTION\_Coor[item[0]].x, SECTION\_Coor[item[0]].y )

end\_point = APoint(SECTION\_Coor[item[1]].x, SECTION\_Coor[item[1]].y )

dim\_position = Functions.GetSizePoint(start\_point,end\_point,item[2],item[3])

dim\_obj = acad.model.AddDimRotated(start\_point, end\_point, dim\_position,item[2])

#Если высота подпорной стенки постоянная, то вид 2-2 не отрисовываем

if input\_datas.sections[index].height\_end == input\_datas.sections[index].height\_start and i == 13:

break

i+=1

# Вставка блока отметки

acad.model.InsertBlock(

APoint(SECTION\_Coor[1].x+(SECTION\_Coor[4].x- SECTION\_Coor[1].x)/2 , input\_datas.sections[index].foundation\_base), #Точка вставки - SECTION\_Coor[4].x)/2

'Otmetka', #Имя блока

1, #Масштаб по Х

1, #Y

1, #Z

0) #Угол поворота

acad.model.InsertBlock(

SECTION\_Coor[6],

'Otmetka',

1,

1,

1,

0)

acad.model.InsertBlock(

SECTION\_Coor[5],

'Otmetka',

1,

1,

1,

0)

#Вывод названий видов

acad.doc.ActiveLayer = acad.doc.Layers.Item("Text\_B")

text\_style = acad.doc.TextStyles.Item("RS0.7")

acad.doc.ActiveTextStyle=text\_style

to\_view\_dis = 2500 # Расстояние от вида до его текста

text\_fas=acad.model.AddMText(Functions.GetStringPoint(SECTION\_Coor[5],SECTION\_Coor[6],to\_view\_dis),0,"Фасад " + input\_datas.sections[index].name +"\n (1 : 100)")

text\_fas.Height = 400

text\_fas.AttachmentPoint = 5#ACAttachmentPoint.MiddleCenter

Выравнивание текста по центру

text\_plan=acad.model.AddMText(Functions.GetStringPoint(SECTION\_Coor[24],SECTION\_Coor[25],to\_view\_dis),0,"План (1 : 100)")

text\_plan.Height=400

text\_1\_1=acad.model.AddMText(Functions.GetStringPoint(SECTION\_Coor[10],SECTION\_Coor[11],to\_view\_dis/2),0,"1 - 1 (1 : 50)")

text\_1\_1.Height=200

#Если высота подпорной стенки постоянная, то вид 2-2 не отрисовываем

if input\_datas.sections[index].height\_end != input\_datas.sections[index].height\_start:

text\_2\_2=acad.model.AddMText(Functions.GetStringPoint(SECTION\_Coor[18],SECTION\_Coor[19],to\_view\_dis/2),0,"2 - 2 (1 : 50)")

text\_2\_2.Height=200

# Создание таблицы ведомости объемов работ

acad.doc.ActiveLayer = acad.doc.Layers.Item("T\_Border") #установка слоя для отрисовки

# Apoint (точка вставки, кол-во строк, кол-во столбцов, высота строки, ширина столбца) - таблица автоматически создается с объединенной 1-й строкой -типа название таблицы

table = acad.model.AddTable(APoint(points[2].x - 18500, points[2].y),4,3,1000,1000)

#table.SetTextHeight(5) #высота текста 3 - задается для всей таблицы

# Задать сразу правильные размеры столбцов и строк - нельзя. Поэтому изменяем их после создания таблицы другим методом:#

table.SetColumnWidth (0, 14500) # 1й столбец ширина = 145000

table.SetColumnWidth (1, 1500)

table.SetColumnWidth (2, 2500)

table.SetRowHeight (0, 1500)

table.SetRowHeight (1, 1500)

table.SetRowHeight (2, 1300)

table.SetRowHeight (3, 1300)

#Заполнение текстом

table.SetText (0, 0 , 'Ведомость основных объемов работ') # 1 строка 1 столбец " заголовок"

table.SetText (1, 0 , 'Наименование работ')

table.SetText (2, 0 , 'Ростверк подпорных стен монолитный железобетонный \n - бетон В30 F{\\S1^;}200 W8 ГОСТ 26633-2015')

table.SetText (3, 0 , 'Тело подпорных стен монолитное железобетонное \n - бетон В30 F{\\S1^;}300 W8 ГОСТ 26633-2015')

table.SetText (1, 1 , 'Ед.изм.')

table.SetText (1, 2 , 'Кол.')

table.SetText (2, 1 , 'м3')

table.SetText (3, 1 , 'м3')

table.SetText (2, 2, round(input\_datas.sections[index].V1 , 1))

table.SetText (3, 2, round(input\_datas.sections[index].V2 , 1))

# Редактирование высоты и стиля текста в таблице

table\_text\_height=350

for row in range(table.Rows):

for col in range(table.Columns):

table.SetCellTextStyle(row, col, "RS0.7")

table.SetCellAlignment(row, col,5)

table.SetCellTextHeight(row, col, table\_text\_height)

index+=1 #Итератор смены секции

except Exception as e:

print(e)

## **Приложение 13. Файл TeklaWork.py**

try:

#Подключение бибилиотек Tekla

from pathlib import Path # Библиотека для указания пути к текущей папке

import clr

from Classes import Wall,Walls

# tekla\_path = r"C:/Program Files/Tekla Structures/2020.0/nt/bin/plugins/" # Укажите ваш путь

tekla\_path = Path.cwd() # Узнаем путь к папке, где лежит приложение

clr.AddReference(str(tekla\_path) + "/Tekla.Structures.dll") # отвечает за работу геометрии в текле

clr.AddReference(str(tekla\_path) + "/Tekla.Structures.Model.dll") # отвечает за работу с балками и прочими компонентами

clr.AddReference(str(tekla\_path) + "/Tekla.Structures.Drawing.dll") # отвечает за работу с чертежом

from Tekla.Structures.Model import Model, Beam,Position,BooleanPart

from Tekla.Structures.Geometry3d import Point

from Tekla.Structures.Model import \*

def InputIKomponent(input\_datas):

model = Model()

if not model.GetConnectionStatus():

print("Не удалось подключиться к модели Tekla")

exit()

for wall in input\_datas.sections:

beam = Beam()

beam.StartPoint = Point(wall.tek\_start\_x, wall.tek\_start\_y, wall.foundation\_base)# Стартовая точка подпорной стенки

beam.EndPoint = Point(wall.tek\_end\_x, wall.tek\_end\_y, wall.foundation\_base)# Конечная точка подпорной стенки

beam.Profile.ProfileString = f"RCRW{max(wall.height\_start,wall.height\_end)}\*{wall.foundation\_width}-{wall.top\_wall\_width}\*{wall.edge\_distance}-{wall.bottom\_wall\_width}-{wall.t1}\*{wall.t2}-{wall.t3}\*{wall.t4}"

# Материал Подпорной стенки

beam.Material.MaterialString = "Сoncrete"

beam.Class="3" # Задает цвет конструкциции (3 - зеленый)

#Установка положения балки по глубине в позицию "Спереди"

position = getattr(Position.DepthEnum, 'FRONT')

beam.Position.Depth = position

if beam.Insert():# Вставляем объект и проверяем: создался или нет

print(wall.name\_wall + " успешно создана")

model.CommitChanges()

else:

print("Ошибка при создании " + wall.name\_wall)

return None

cut\_beam=Beam()

cut\_beam.StartPoint = Point(wall.tek\_start\_x, wall.tek\_start\_y, wall.foundation\_base+wall.height\_start)# Стартовая точка подпорной стенки

cut\_beam.EndPoint = Point(wall.tek\_end\_x, wall.tek\_end\_y, wall.foundation\_base+wall.height\_end)# Конечная точка подпорной стенки

cut\_beam.Profile.ProfileString = f"{wall.foundation\_width\*2}\*{wall.foundation\_width\*2}"

cut\_beam.Class=BooleanPart.BooleanOperativeClassName

cut\_beam.Position.Depth = position

cut\_beam.Insert()

boolean\_part = BooleanPart()

boolean\_part.Father = beam

boolean\_part.SetOperativePart(cut\_beam)

boolean\_part.Type = BooleanPart.BooleanTypeEnum.BOOLEAN\_CUT

boolean\_part.Insert()

model.CommitChanges()

model.GetModelObjectSelector().GetAllObjects().SelectInstances = False

cut\_beam.Delete()

model.CommitChanges();

except Exception as e:

print(e)