Содержание

Введение

Математические методы впервые были применены к изучению заболеваний в 1760 году Даниэлем Бернулли [2], который использовал их для оценки эффективности различных методов вакцинации от оспы.

В 1840 году Уильям Фарр успешно описал данные по смертности от оспы в Англии и Уэльсе за период с 1837 по 1839 год, используя нормальное распределение. Этот метод был развит Джоном Браунли, который в своей статье «Статистический подход к иммунной защите: теория эпидемий» (1906) сравнил ряды эпидемиологических данных на основе распределения Пирсона.

Хамер и Росс, используя математическое описание распространения заболеваний, смогли решить задачи по выяснению механизмов регулярного повторения эпидемии кори и установлению связи между количеством комаров и возникновением малярии. Их работы, наряду с исследованиями Росса и Хадсона, Сопера, а также Кермака и Маккендрика (1927), стали основой для дальнейших исследований в области математического моделирования эпидемий.

В этих работах впервые был применен «закон действующих масс», согласно которому количество вновь инфицированных прямо пропорционально произведению числа восприимчивых и инфицированных особей. Модель Кермака и Маккендрика положила начало широкому использованию детерминированных SIR-моделей («Susceptible — Infected — Recovered»), в которых с помощью систем дифференциальных или разностных уравнений описывается динамика групп восприимчивых, инфицированных и выздоровевших.

Рассматриваемые в данной работе модели могут спрогнозировать, как прогрессируют инфекционные заболевания. Модели используют базовые допущения или собранные статистические данные для нахождения параметров для различных инфекционных заболеваний, используя их для расчета эффекта различных вмешательств, таких как ввод карантина. [1]

Программная реализация рассматриваемых математических моделей позволяет провести численный анализ различных сценариев, влияние карантинных мер и других факторов. В данной работе рассматриваются и реализуются основные детерминированные модели развития эпидемий: SI, SIR, SIRS, SEIR, SIQR, MSEIR и M-модель. Эти модели отличаются уровнем детализации и применяются в зависимости от характеристик конкретного заболевания и целей анализа.

1 Математическое моделирование инфекционных заболеваний

В данном разделе дается понятие математической модели. Рассматриваются конкретные модели для прогнозирования развития эпидемиологической ситуации.

1.1 Типы эпидемических моделей

1.1.1 Стохастические

Стохастическая модель — это инструмент для оценки распределения вероятностей потенциальных результатов с учётом случайных изменений одного или нескольких входных параметров с течением времени. [1]

В начале XX века ученые-эпидемиологи пришли к выводу, что для описания распространения инфекционных заболеваний в ограниченных популяциях (например, семьях) применительно к болезням, таким как корь, более целесообразным является вероятностный подход, нежели детерминированный.

В 1926 году Маккендрик сформулировал стохастический вариант модели SIR с непрерывным временем для вывода уравнений, описывающих продолжительность эпидемии, применительно к гриппу и малярии. Однако работа Маккендрика не получила широкого признания.

Значительный вклад в развитие моделей с вероятностным описанием внесла работа Гринвуда 1931 года, посвященная изучению вспышки кори. Также важную роль сыграла модель Рида и Фроста, разработанная ими в период с 1928 по 1930 годы, но опубликованная лишь в 1952 году.

В моделях Гринвуда и Рида-Фроста описание количества инфицированных на каждом временном интервале привело к использованию цепи биномиальных распределений, что обусловило появление термина «цепочечно-биномиальные модели».

1.1.2 Детерминированные

Детерминированная модель (противоположна модели стохастической) — математическая модель, параметры и переменные которой зависят друг от друга функционально, т.е. не подчинены случайным колебаниям процесса, в связи с чем характер системы в любое время полностью определяется первоначально выбранными условиями.

При работе с большими группами населения, как в случае с туберкулёзом, часто используются детерминированные или компартментальные математические модели. В детерминированной модели люди в популяции распределяются по разным подгруппам или компартментам, каждый из которых представляет собой определённую стадию эпидемии.

Скорости перехода из одного класса в другой математически выражаются как производные, поэтому модель формулируется с помощью дифференциальных уравнений [18]. При построении таких моделей необходимо предполагать, что численность населения в группе дифференцируема по времени и что процесс эпидемии является детерминированным. Другими словами, изменения численности населения в группе можно рассчитать, используя только историю, которая использовалась для разработки модели.

1.2 Детерминистские популяционные модели

В популяционных моделях, применяемых для анализа динамики населения в определенном регионе, последнее представляется как совокупность групп. Формирование этих групп обусловлено различным статусом индивидов относительно рассматриваемого заболевания (например, восприимчивые к заболеванию, инфицированные, больные на различных стадиях болезни, лица в состоянии ремиссии).

Важно отметить, что внутри каждой группы индивиды считаются идентичными по своим характеристикам. Изменения численности групп во времени обусловлены рядом следующих процессов:

- переходы индивидов из одной группы в другую вследствие инфицирования, развития заболевания, выявления и лечения заболевших индивидов;
- пополнение групп за счёт иммиграции и рождения индивидов;
- убыль в результате естественной смертности индивидуумов, гибели из–заболезни и эмиграции в другие регионы.

Среди моделей, используемых для описания динамики распространения инфекционных заболеваний в популяциях, наиболее распространены модели SIR. В этих моделях население подразделяется на три группы:

- S (Susceptible) - восприимчивые к заболеванию;
- I (Infected) - инфицированные;
- R (Recovered/Removed) - переболевшие или удалённые из популяции.

Передача инфекции происходит от инфицированных к восприимчивым. Индивиды, переболевшие болезнью, приобретают иммунитет и не могут быть инфицированы повторно.

Математически модели SIR описываются системами дифференциальных уравнений [18] (для непрерывного времени) или разностными уравнениями (для дискретного времени). Эти уравнения отражают изменение численности каждой группы с течением времени.

1.2.1 Классическая SIR-модель

SIR-модель - одна из самых базовых и популярных моделей. Она делит популяцию на три группы:

- S (Susceptible) - восприимчивые к заболеванию;
- I (Infected) - инфицированные;
- R (Recovered/Removed) - переболевшие или удалённые из популяции.

Схема модели SIR показана рисунке 1.

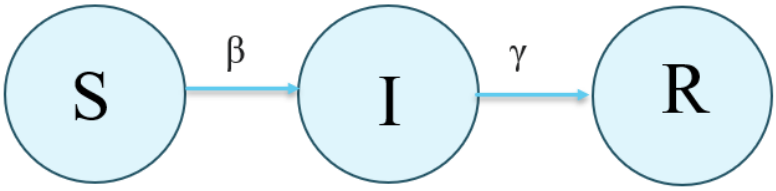


Рисунок 1 – SIR-модель

Данная модель была предложена в 1927 году Уильямом Огильви Кермаком и Эндерсоном Грейем МакКендриком. Эта модель стала классическим фундаментом математической эпидемиологии, на основе которой были разработаны и предложены модифицированные модели, рассматривающие дополнительные группы популяции. [2]

Модель описывает изменение численности этих групп во времени с помощью системы дифференциальных уравнений [18]. Например, стандартной SIR-модели соответствует система (1).

$$\begin{cases} \frac{dS}{dt} = -\beta * S * I \\ \frac{dI}{dt} = \beta * S * I - \gamma * I, \\ \frac{dR}{dt} = \gamma * I \end{cases} \quad (1)$$

где β – скорость передачи инфекции;

γ – скорость выздоровления.

1.2.2 SI-модель

SI-модель является упрощенной версией классической SIR-модели, где отсутствует стадия выздоровления, то есть все, кто заражается, остаются инфицированными. Схема данной модели показана на рисунке 2.

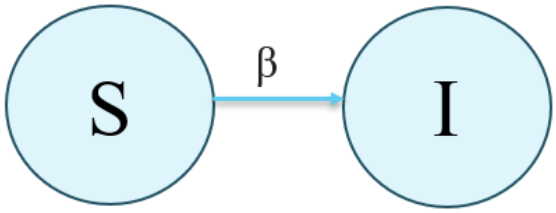


Рисунок 2 – SI-модель

Появилась данная модель в начале XX века. Основана на работах У. Х. Хаммерса (1920) и Нильса Кристиана Бергера (1932), но как часть общего подхода к эпидемиологическому моделированию. Она может использоваться для описания хронических инфекций, таких как ВИЧ. Данной модели соответствует система (2).

$$\begin{cases} \frac{dS}{dt} = -\beta * S * I \\ \frac{dI}{dt} = \beta * S * I \end{cases}, \quad (2)$$

где β – скорость передачи инфекции;

γ – скорость выздоровления.

1.2.3 SIRS-модель

Разработана как расширение SIR-модели для инфекций, при которых иммунитет не является постоянным (например, грипп). Схема данной модели показана на рисунке 3.

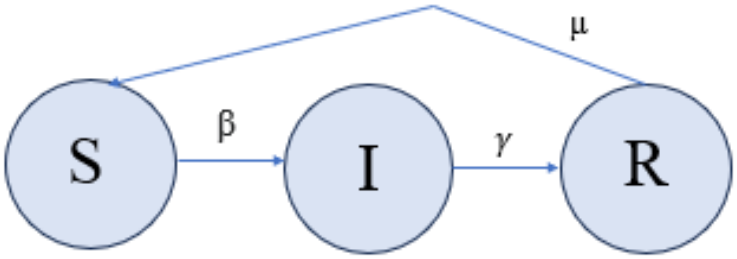


Рисунок 3 – SIRS-модель

Применялась активно в 1950–1970-х годах. В этой модели индивиды, выздоровевшие и перешедшие в группу R, могут потерять иммунитет и снова стать восприимчивыми. Данной модели соответствует система (3).

$$\begin{cases} \frac{dS}{dt} = -\beta * S * I + \delta * R \\ \frac{dI}{dt} = \beta * S * I - \gamma * I \\ \frac{dR}{dt} = \gamma * I - \delta * R \end{cases}, \quad (3)$$

где β – скорость передачи инфекции;

γ – скорость выздоровления;

δ – скорость потери иммунитета.

1.2.4 SIQR-модель

SIQR-модель разработана в конце XX — начале XXI века на фоне необходимости учитывать карантинные меры в условиях эпидемий. Схема данной модели представлена на рисунке 4.

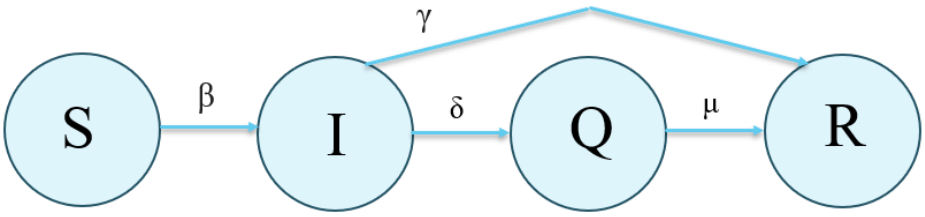


Рисунок 4 – SIQR-модель

Наиболее эффективна при моделировании таких инфекций как COVID-19, SARS, туберкулез, холера и другие заболевания, при которых карантин влияет на распространение. Данной модели соответствует система (4).

$$\begin{cases} \frac{dS}{dt} = -\beta * S * I \\ \frac{dI}{dt} = \beta * S * I - \gamma * I - \delta * I \\ \frac{dQ}{dt} = \delta * I - \mu * Q \\ \frac{dR}{dt} = \gamma * I + \mu * Q \end{cases}, \quad (4)$$

где β – скорость передачи инфекции;

γ – скорость выздоровления;

δ – скорость изоляции инфицированных людей (помещения на карантин);

μ – скорость выздоровления людей, находящихся на карантине.

1.2.5 SEIR-модель

Эта модель включает дополнительное состояние – E (Exposed), или «латентную» фазу, когда человек инфицирован, но еще не заразен. Схема данной модели представлена на рисунке 5.

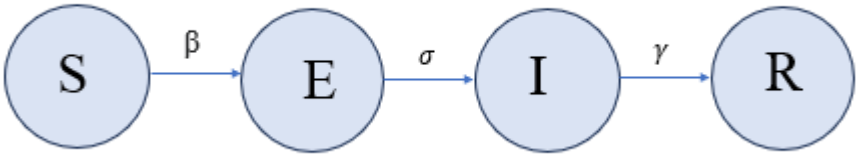


Рисунок 5 – SEIR-модель

Модель SEIR полезна для описания инфекций с инкубационным периодом, таких как COVID-19. Предложена в 1930-х–1950-х годах как усовершенствование модели SIR. Активно применялась во второй половине XX века для анализа заболеваний с инкубационным периодом. Данной модели соответствует система (5).

$$\begin{cases} \frac{dS}{dt} = -\beta * S * I \\ \frac{dE}{dt} = \beta * S * I - \sigma * E \\ \frac{dI}{dt} = \sigma * E - \gamma * I \\ \frac{dR}{dt} = \gamma * I \end{cases}, \quad (5)$$

где β – скорость передачи инфекции;

γ – скорость выздоровления;

σ – обратная величина инкубационного периода.

1.2.6 MSEIR-модель

Данная модель является расширение SEIR-модели, в которой добавляется группа M (Maternal immunity), представляющая новорожденных с пассивным иммунитетом, получивших антитела от матери. Схема данной модели показана на рисунке 6.

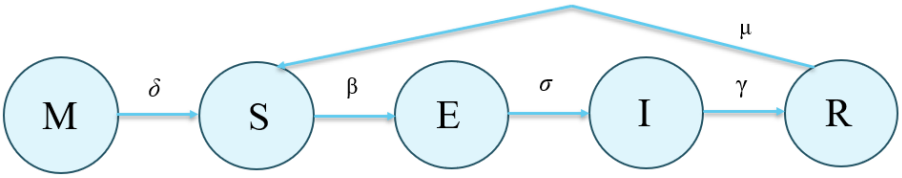


Рисунок 6 – MSEIR-модель

MSEIR-модель наиболее эффективна при моделировании таких инфекций как корь, краснуха, коклюш и другие детские заболевания, при которых материнский иммунитет играет важную роль в первые месяцы жизни младенца. Данной модели соответствует система (6).

$$\begin{cases} \frac{dM}{dt} = \mu * N - \delta * M - \mu * M \\ \frac{dS}{dt} = \delta * M - \beta * S * I - \mu * S \\ \frac{dE}{dt} = \beta * S * I - \sigma * E - \mu * E \\ \frac{dI}{dt} = \sigma * E - \gamma * I - \mu * I \\ \frac{dR}{dt} = \gamma * I - \mu * R \end{cases}, \quad (6)$$

где β – скорость передачи инфекции;

γ – скорость выздоровления;

σ – скорость перехода из инкубационного периода в инфекционный;

μ – естественная смертность/рождаемость;

δ – скорость потери материнского иммунитета;

N – общее количество населения.

1.3 Постановка задачи

В связи со стремительным распространением инфекционных заболеваний и повышенной мобильности населения приобретает особую актуальность разработка и анализ математических моделей, отражающих динамику эпидемических процессов. Данные модели не только позволяют прогнозировать развитие эпидемиологической ситуации, но и оценивать эффективность различных мер общественного здравоохранения – таких как вакцинация, карантинные мероприятия и ограничения на передвижение.

Целью данной работы является разработка программной реализации классических и модифицированных моделей математической эпидемиологии последующим численным решением соответствующих систем дифференциальных уравнений и сравнительным анализом полученных результатов.

Для достижения поставленной цели решены следующие задачи:

1. Проведен теоретический анализ существующих моделей распространения инфекционных заболеваний.

2. Сформулированы математические модели (в виде систем обыкновенных дифференциальных уравнений) для каждой рассматриваемой схемы.

3. Реализовано программное решение моделей с использованием численного метода Рунге-Кутты 4-го порядка и метода Эйлера.

4. Разработан пользовательский интерфейс, обеспечивающий ввод параметров и визуализацию результатов моделирования.

5. Проведен сравнительный анализ поведения различных моделей при идентичных начальных условиях.

6. Исследованы влияние ключевых параметров (скорость заражения, инкубационный период, миграция и т.п.) на динамику эпидемии.

Таким образом, данная работа представляет собой синтез теоретических и практических аспектов, направленных на создание инструмента для анализа эпидемических ситуаций и поддержки принятия решений в сфере общественного здравоохранения.

1.4 Выводы по главе

В данной главе рассмотрены типы математических моделей, проведен теоретический обзор классической SIR-модели и её модификаций. Данный обзор позволяет алгоритмически сконструировать программное средство для моделирования развития эпидемиологической ситуации. Поскольку представленные в данной главе системы дифференциальных уравнений нельзя решить аналитически, то необходимо прибегнуть к использованию численных методов, таких как методы Эйлера и Рунге-Кутты 4-го порядка.

2 Алгоритмическое конструирование программного средства

В данном разделе приводятся схемы программного средства и используемых численных методов для решения СДУ.

2.1 Структура программного средства

Для удобного моделирования развития эпидемиологической ситуации необходимо разработать графический интерфейс, в котором можно выбрать одну или несколько моделей, ввести для каждой начальные данные и реализовать вывод результата в удобном для пользователя формате и должно включать в себя численные методы для решения СДУ. Описанная логика программного средства показана на рисунке 8.

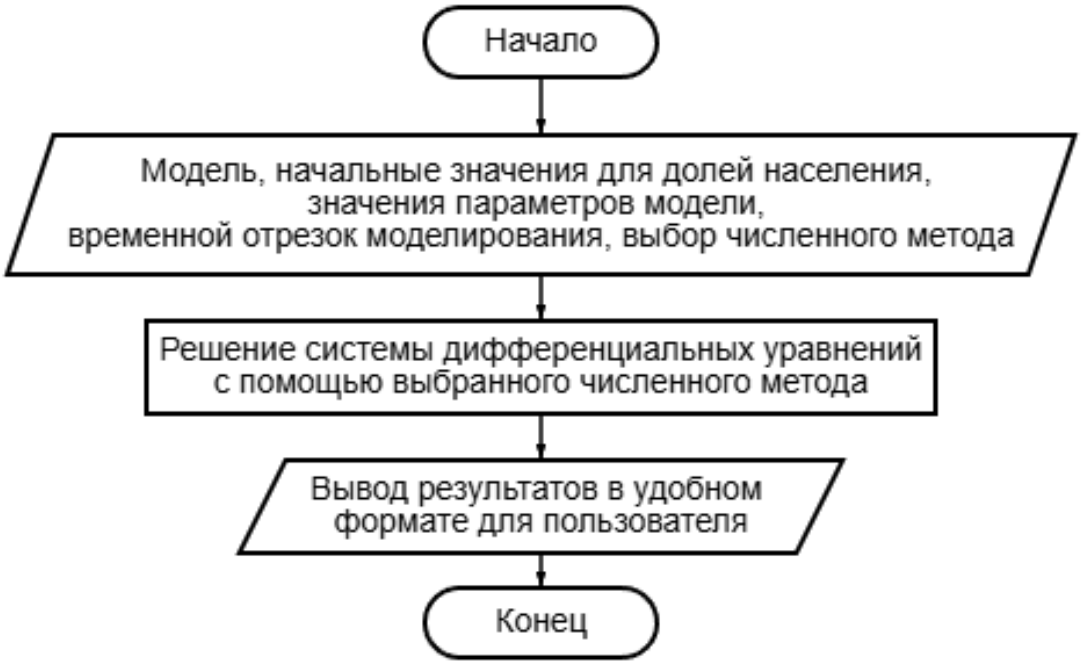


Рисунок 8 – Схема программного средства

2.1.1 Метод Эйлера

Для метода Эйлера необходимы следующие входные данные: математическая модель, временной диапазон, начальные значения, параметры модели. Реализация метода Эйлера под данную задачу представлена на рисунке 9.

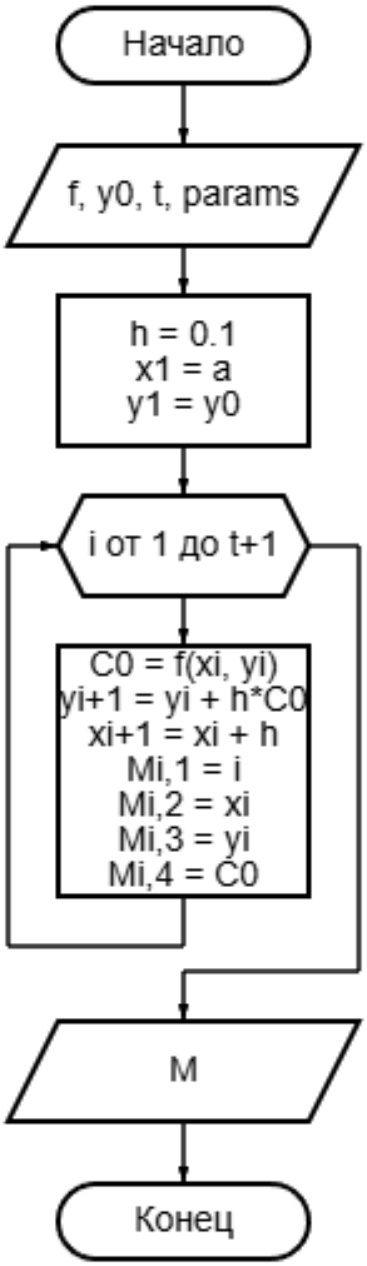


Рисунок 9 – Схема метода Эйлера

2.1.1 Метод Рунге-Кутта 4-го порядка

Для более точного численного метода Рунге-Кутта 4-го порядка необходимы следующие входные данные: выбранная модель, начальные значения, временной отрезок, параметры модели [17]. Реализация метода Рунге-Кутта 4-го порядка под данную задачу представлена на рисунке 10.

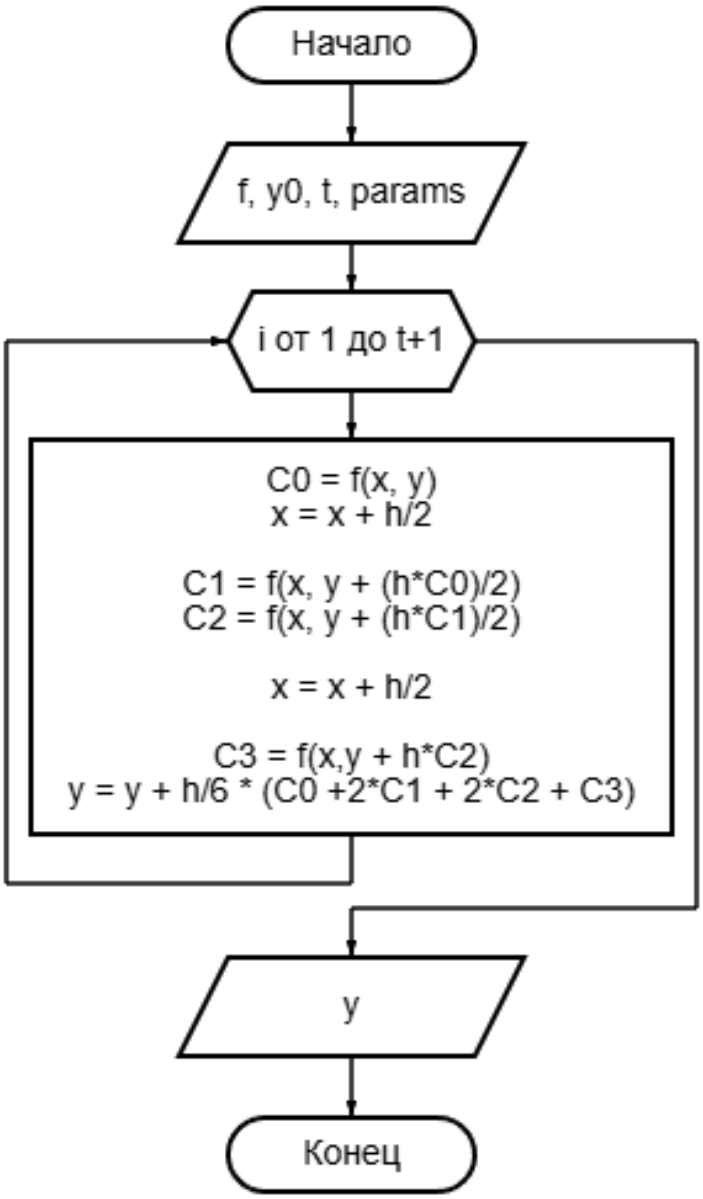


Рисунок 10 – Схема метода Рунге-Кутта 4-го порядка

2.2 Выводы по главе

В данной главе рассмотрен общая схема программного средства, а также схемы численных методов, используемых для решения систем дифференциальных уравнений. Данные схемы используются для дальнейшего решения поставленной задачи.

3 Программное конструирование

В данном разделе обоснованы выбор языка программирования, используемый для реализации программы, и используемых дополнительных библиотек. Описаны основные классы и методы для выполнения поставленной задачи.

3.1 Выбор языка программирования и среды разработки

Для разработки программного средства был взят за основу язык программирования Python [13] версии 3.13.1. Были выделены следующие достоинства выбранного языка: динамическая типизация данных, автоматическое управление памятью, читаемость кода, минималистичность синтаксиса языка, богатая стандартная библиотека и большое количество дополнительных библиотек. [3]

Средой разработки был выбран Visual Studio Code, т.к. включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса и средства для факторинга. Распространяется бесплатно и позиционируется как «лёгкий» редактор кода. [4]

3.2 Основные классы программного средства

Для реализации программного средства была выбрана объектно-ориентированная парадигма. В процессе программного конструирования были выделены следующие классы: класс численных методов NumericalMethods, класс математических моделей EpidemicModels и главный класс программного средства EpidemicApp.

NumericalMethods представляет собой класс, содержащий численные методы Эйлера и Рунге-Кутта 4-го порядка для решения систем дифференциальных уравнений. Данный класс содержит методы, приведенные в таблице 1.

Таблица 1 – Методы класса NumericalMethods

Название метода	Функциональное назначение метода
euler_method	метод для нахождения решения системы дифференциальных уравнений с помощью метода Эйлера
runge_kutta_4	метод для нахождения решения системы дифференциальных уравнений с помощью метода Рунге-Кутта 4-го порядка

Класс EpidemicModels содержит методы, реализующие математические модели развития эпидемиологических ситуаций, представленные в данной работе. Содержит методы, приведенные в таблице 2.

Таблица 2 – Методы класса EpidemicModels

Название метода	Функциональное назначение метода
sir_model	метод, реализующий логику SIR-модели
si_model	метод, реализующий логику SI-модели
sirs_model	метод, реализующий логику SIRS-модели
siqr_model	метод, реализующий логику SIQR-модели
seir_model	метод, реализующий логику SEIR-модели
mseir_model	метод, реализующий логику MSEIR-модели
multi_stage_model	метод, реализующий логику M-модели
run_si_model	метод, запускающий SI-модель на указанном диапазоне времени
run_sir_model	метод, запускающий SIR-модель на указанном диапазоне времени
run_sirs_model	метод, запускающий SIRS-модель на указанном диапазоне времени
run_seir_model	метод, запускающий SEIR-модель на указанном диапазоне времени
run_mseir_model	метод, запускающий MSEIR-модель на указанном диапазоне времени
run_siqr_model	метод, запускающий SIQR-модель на указанном диапазоне времени
run_m_model	метод, запускающий M-модель на указанном диапазоне времени

Класс EpidemicApp содержит методы, реализующие логику работы всей программы, а также загрузку и данных из файла в программу и экспорт результатов в Excel файл. Методы данного класса создают главное окно программного средства, осуществляют ввод данных и запуск моделирования. Содержит методы, приведенные в таблице 3.

Таблица 3 – Методы класса EpidemicApp

Название метода	Функциональное назначение метода
create_widgets	метод создания основного интерфейса
create_models_tab	метод создания вкладки выбора моделей
create_params_tab	метод создания вкладки для ввода параметров и начальных условий
create_data_tab	метод создания вкладки для загрузки и экспорта данных
create_graphs	метод настройки графиков для отображения результатов
create_validate_func	метод для валидации вводимых данных
set_default_values	метод, устанавливающий значения по умолчанию
update_model_selection	метод, ограничивающий и проверяющие количество выбранных моделей
run_simulation	метод, запускающий прогнозирование на основе выбранных моделей
load_csv_data	метод, загружающие начальные данные из CSV файла
process_csv_data	метод, обрабатывающий загруженные данные
export_results	метод, экспортирующий результаты моделирование в Excel файл(ы)

Полный листинг программы приведён в Приложении Б. UML-диаграмма классов программного средства приведено в Приложении А.

3.3 Выводы по главе

В данной главе обоснован выбор языка и среды программирования для реализации программного средства. Разработаны основные классы и методы программы.

4 Тестирование программного средства

В данном разделе описаны и протестированы контрольные примеры с подробным описанием входных данных и подтверждающими работу программного средства изображениями. Приводится инструкция по использованию программного средства. Проведен анализ результатов моделирования.

4.1 Инструкция по использованию программного средства

Разработанное программное средство предназначено для моделирования развития эпидемиологической ситуации с помощью представленных в данной работе математических моделей: SIR, SI, SIRS, SIQR, SEIR, MSEIR и M-модель. Для начала работы с программой пользователю необходимо выбрать 1-4 модели, выбрать начальную и конечную дату моделирования, а также ввести начальные значения долей населения и значение параметров выбранных моделей. Интерфейс пользователя поддерживает вариант, при котором пользователь может загрузить в программу файл со своими данными.

Запуск моделирования осуществляется через нажатие кнопки «Запустить моделирование». После нажатия в окне программного средства появятся графики выбранных моделей на указанном отрезке времени. Пользователь может сохранить результат моделирования в формате Excel файла, нажав на кнопку «Экспорт в файл». Создаться архив, в котором будут сохранены по отдельности результаты каждой модели. В сохраненных Excel файлах будет 4 листа:

— «Параметры» - лист, в котором сохранены введенные пользователем значения для параметров модели;

— «Начальные условия» - лист, в котором сохранены введенные пользователем начальные значения для долей населения;

— «Решение» - лист, в котором сохранено по дням решение математической модели с помощью численного метода;

— «График» - лист, содержащий результирующий график моделирования развития эпидемиологической ситуации.

Начальное состояние интерфейс программного средства представлен на рисунке 11.

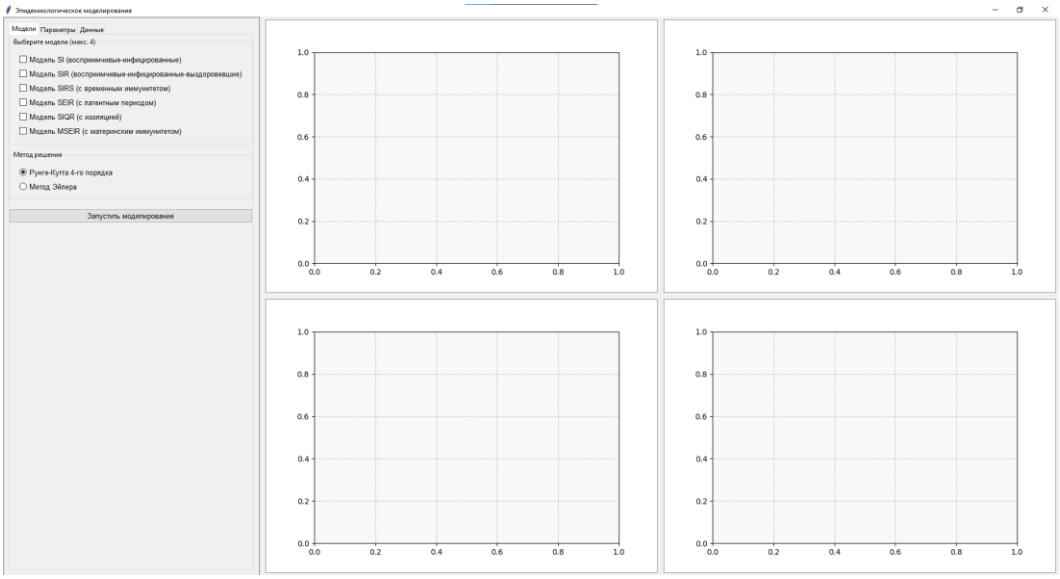


Рисунок 11 – Начальное состояние

Конечное состояние интерфейса программного средства показано на рисунке 12.

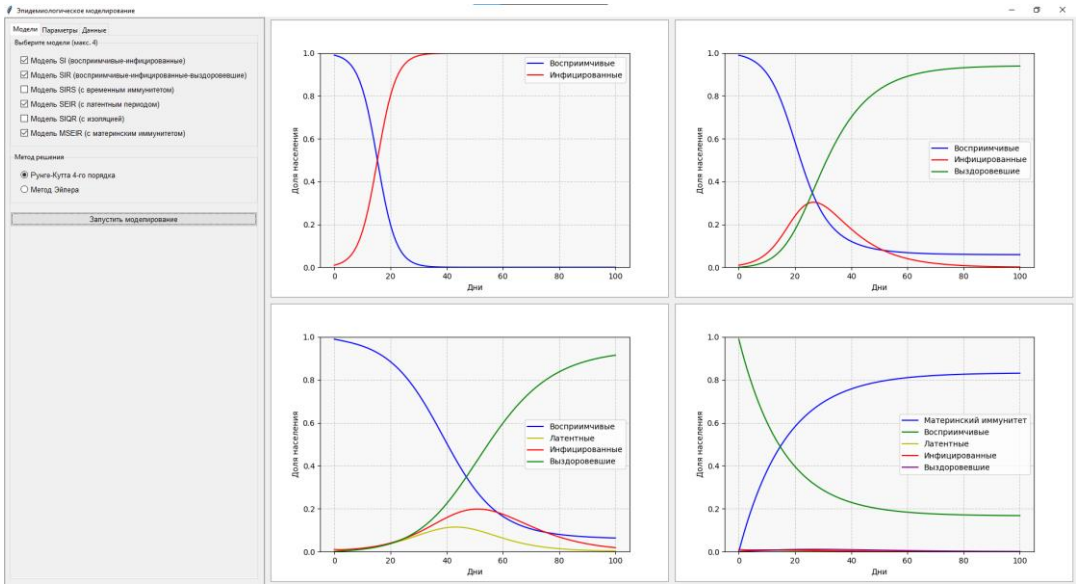


Рисунок 12 – Конечное состояние

4.2 Описание загрузки данных из файла

Для загрузки заранее составленных данных для моделирования необходимо перейти на вкладку «Данные» и нажать на кнопку «Загрузить данные из CSV», что показано на рисунке 13.

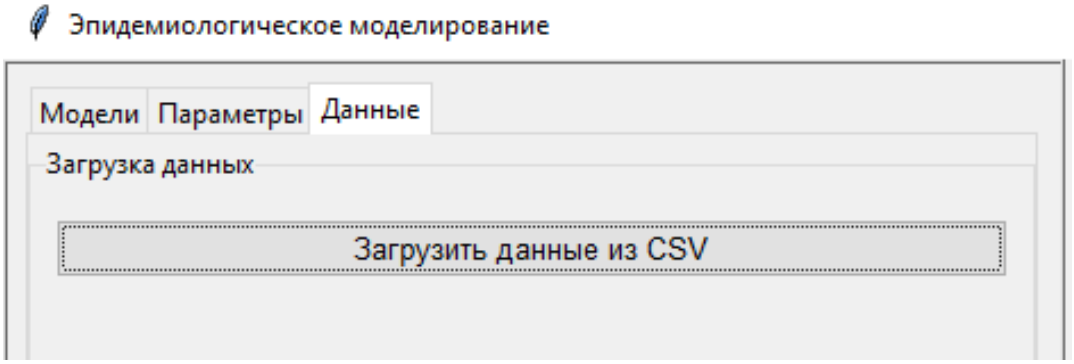


Рисунок 13 – Раздел загрузки данных из файла

После этого программа попытается выгрузить необходимые данные из файла: даты, количество населения, восприимчивые, инфицированные и выздоровевшие. Если данные из файла не были успешно выгружены, то на экран пользователя будет выведено уведомление об ошибке с указанием отсутствующего столбца в файле, представленное на рисунке 14.

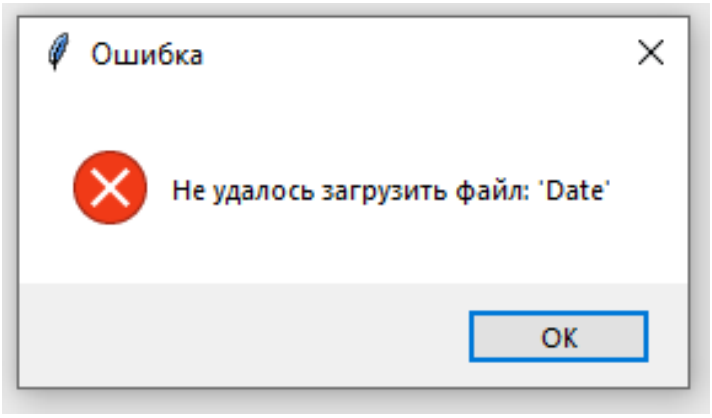


Рисунок 14 – Ошибка загрузки данных

Если данные загружены успешно, то программное средство выведен на экран пользователя окно выбора страны, если в файле находятся несколько стран, а также диапазон дат, информация за которые будет считаться начальными данными для моделирования. Данное окно показано на рисунке 15.

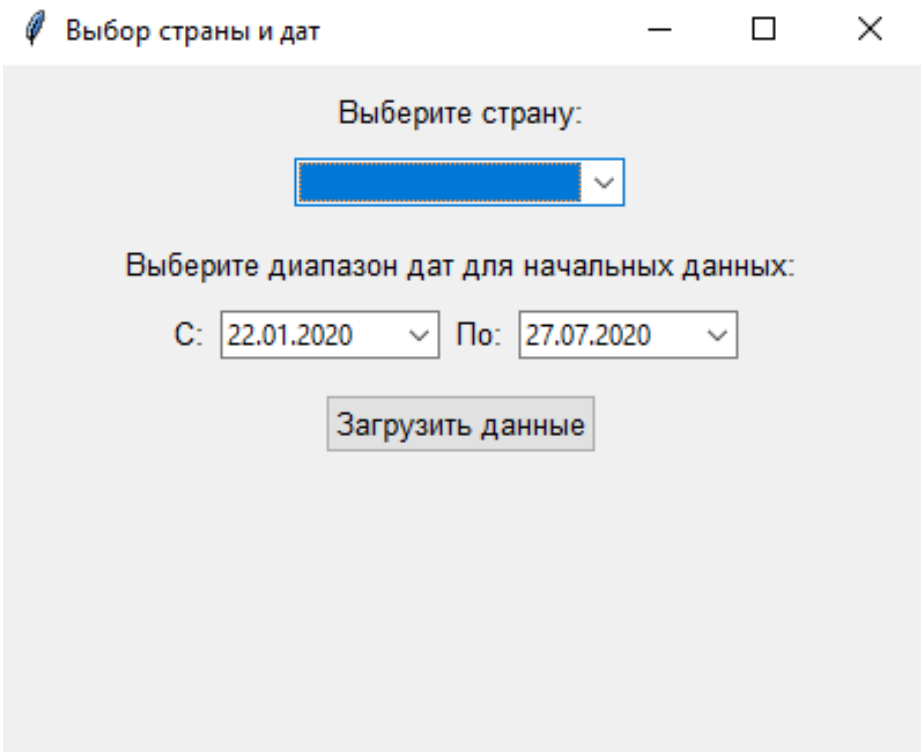


Рисунок 15 – Окно выбора страны и дат

После нажатия на кнопку «Загрузить данные» на экране пользователя появиться уведомление об успехе операции с указанием страны и периода, выбранных пользователем. Данное уведомление показано на рисунке 16.

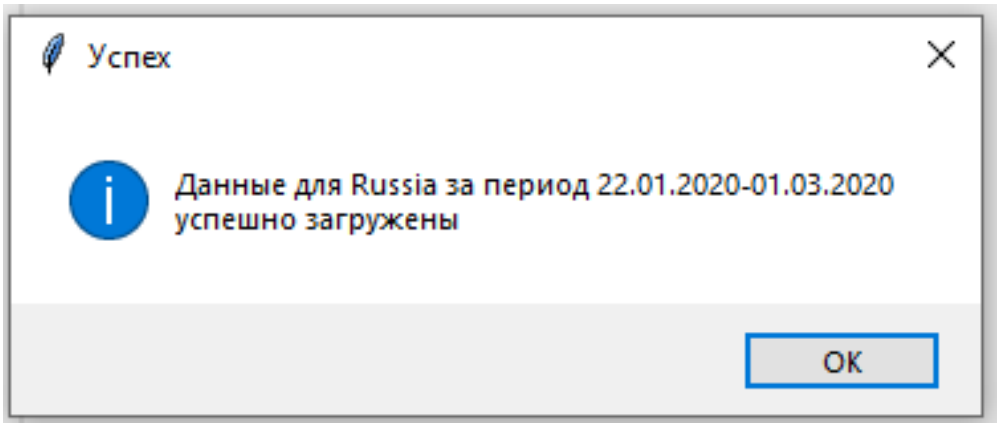


Рисунок 16 – Уведомление об успехе загрузки данных

Загруженные данные по долям населения нормализуются в диапазоне от нуля до единицы и распределяются на S (восприимчивых), I (инфицированных) и R (выздоровевших). Данное распределение показано на рисунке 17.

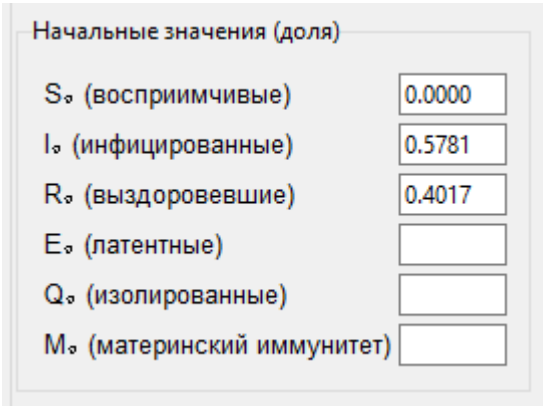


Рисунок 17 – Распределение долей населения

Дата начала прогнозирования автоматически определяется как конечная дата выбора начальных данных из файла. Конечная дата прогнозирования отличается от начальной на 3 месяца, что показано на рисунке 18.

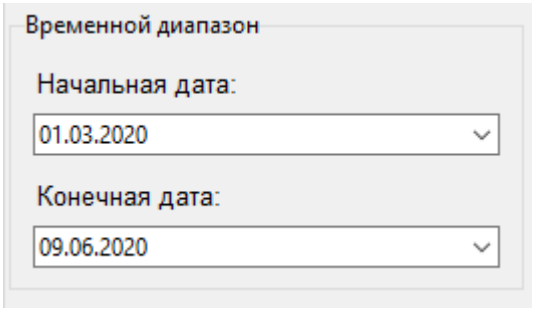


Рисунок 18 – Временной диапазон прогнозирования

Пользователь может заранее или после загрузки данных из CSV файла выбрать начальную и конечную дату прогнозирования, что показано на рисунках 19 и 20.

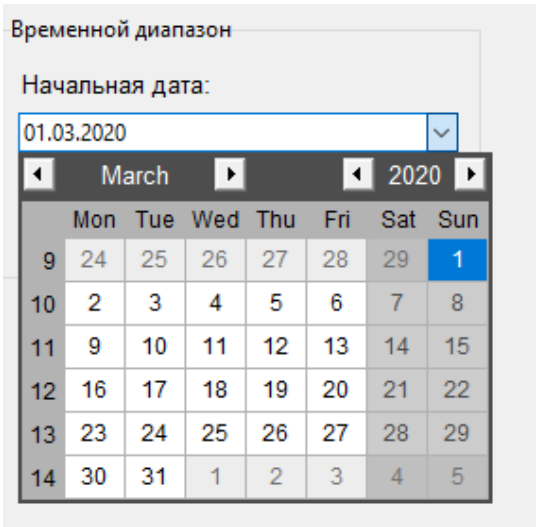


Рисунок 19 – Выбор начальной даты

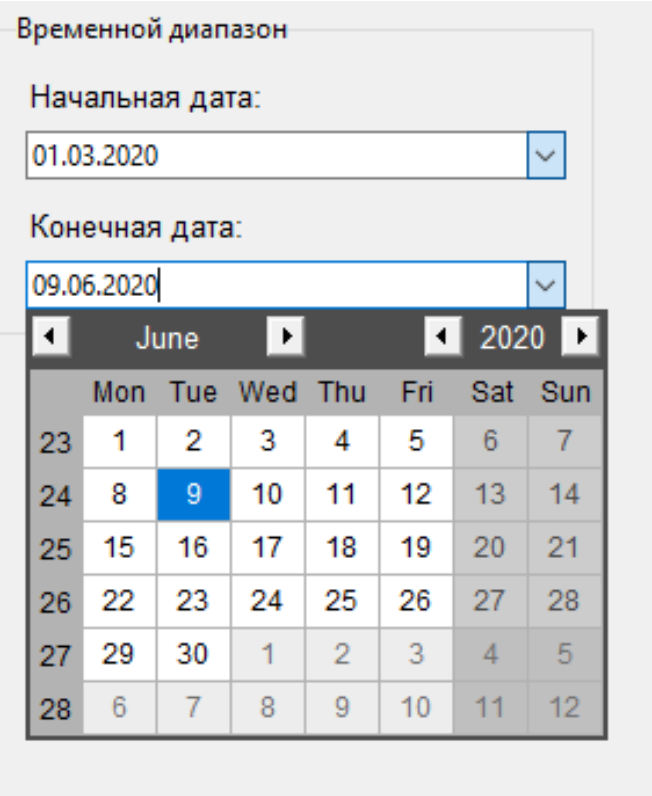


Рисунок 20 – Выбор конечной даты

Пользователь может заранее или после загрузки данных из CSV файла ввести значения параметров модели, либо оставить значения по умолчанию, что показаны на рисунке 21.

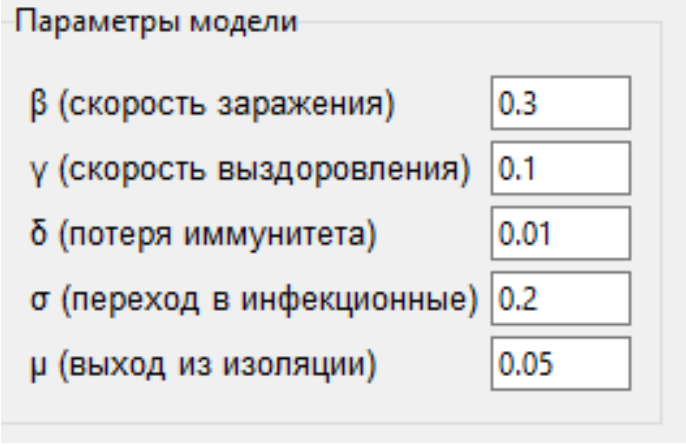


Рисунок 21 – Значения параметров по умолчанию

Пользователь может заранее или после загрузки данных из CSV файла выбрать от одной до четырех математических моделей. Если этот выбор превышает ограничение, то на экран будет выведено сообщение о превышении лимита, что показано на рисунке 22.

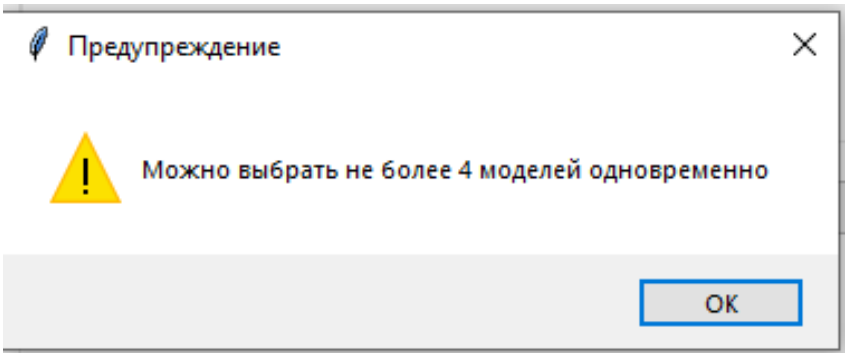


Рисунок 22 – Предупреждение об превышении лимита

Пользователь также может заранее или после выгрузки данных выбрать численный метод: Рунге-Кутта 4-го порядка или Эйлера. Чтобы начать моделирование пользователю необходимо нажать на кнопку «Запустить моделирование».

4.3 Тестирование математических моделей

Поскольку представленные в данной работе модели отличаются друг от друга как параметрами, так и начальными условиями, то каждую из них необходимо тестировать отдельно.

4.3.1 SIR-модель

SIR-модель состоит из трех начальных значений (восприимчивые, инфицированные и выздоровевшие) и двух параметров (скорость заражения и скорость выздоровления). Для первого контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $I_0 = 0.01$, $R_0 = 0.0$, $\beta = 0.3$, $\gamma = 0.1$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 23.

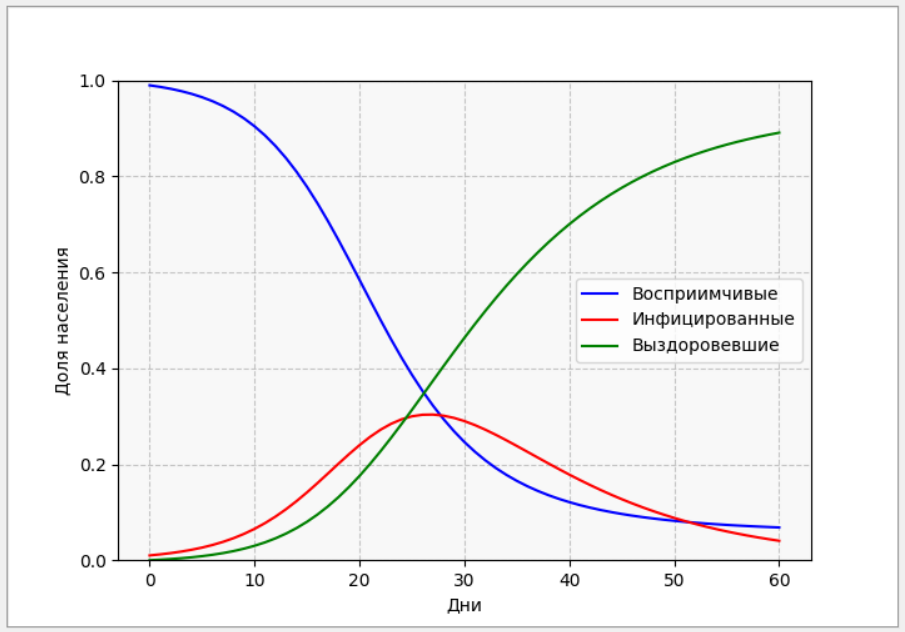


Рисунок 23 – Результат SIR-модели для первого контрольного примера с помощью метода Рунге-Кутта [17] 4-го порядка

Для второго контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $I_0 = 0.01$, $R_0 = 0.0$, $\beta = 0.3$, $\gamma = 0.5$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 24.

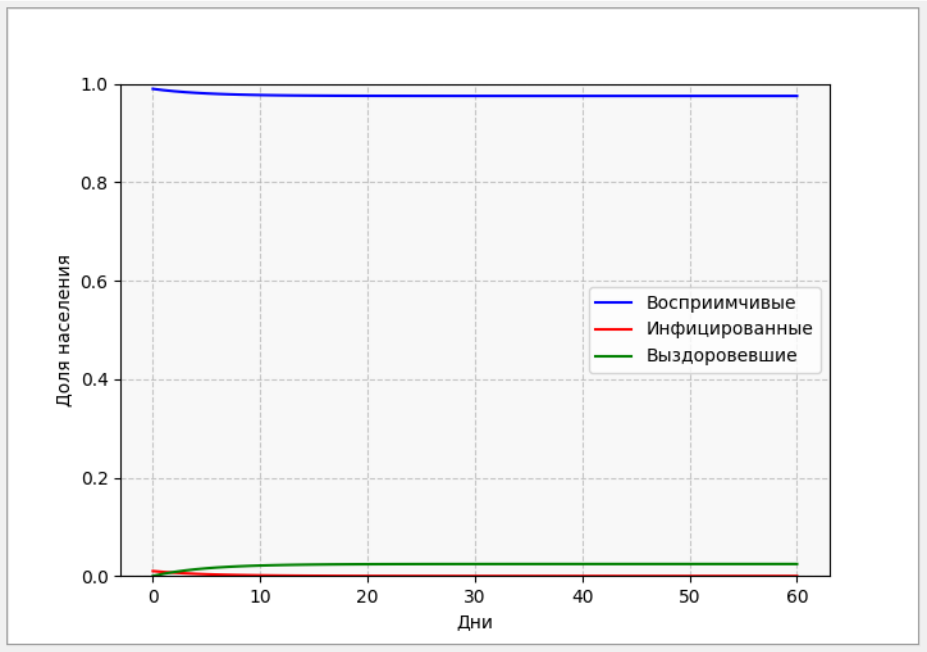


Рисунок 24 – Результат SIR-модели для второго контрольного примера с помощью метода Рунге-Кутта 4-го порядка

На графике видно, что инфицированные выздоравливают быстрее, а так как начальное количество данной доли население мало, то эпидемия заканчивается в первые же дни своего появления. Увеличим долю инфицированных до 0.25, а восприимчивых уменьшим до 0.75, оставив остальные параметры без изменения. Результат прогнозирования с измененными данными показан на рисунке 25.

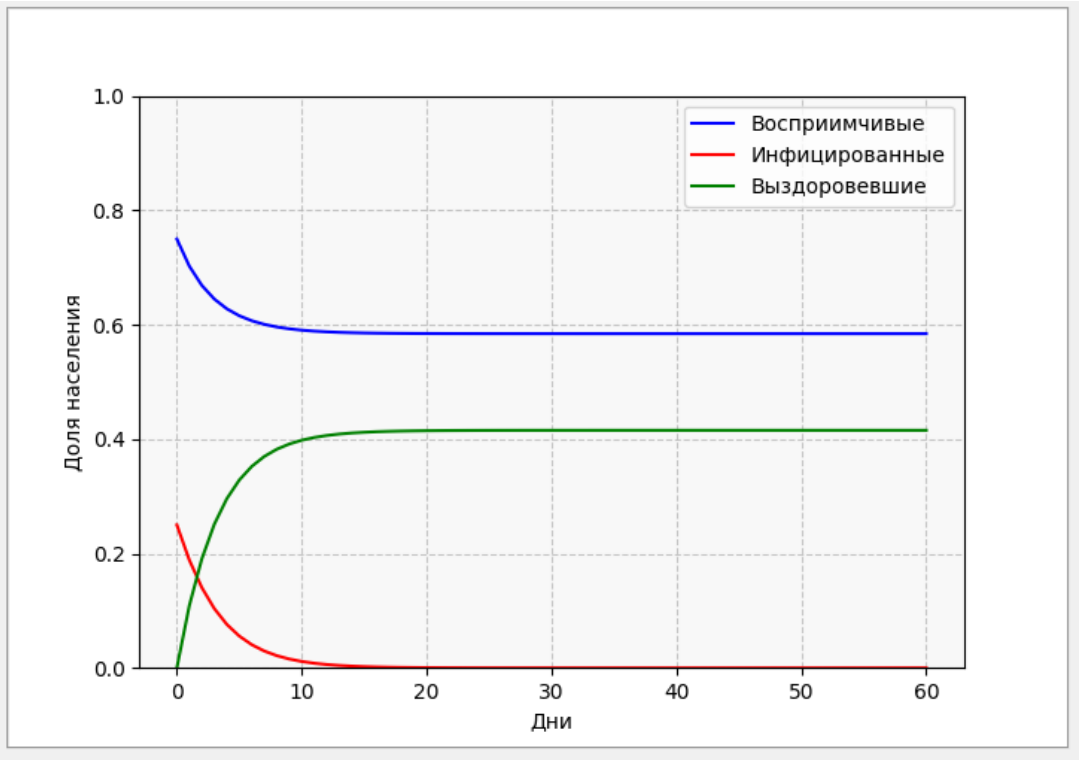


Рисунок 25 – Результат прогнозирования с измененными начальными данными для второго контрольного примера

На графике можно увидеть, что несмотря на увеличение количества восприимчивых, эпидемия все равно заканчивается в первый десяток дней. Пересечение графиков выздоровевших и инфицированных примерно похоже на пересечение этих же графиков в предыдущем варианте второго контрольного примера.

4.3.2 SI-модель

SI-модель состоит из двух начальных условий (восприимчивые и инфицированные) и одного параметра (скорость заражения). Для первого

контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $I_0 = 0.01$, $\beta = 0.3$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 26.

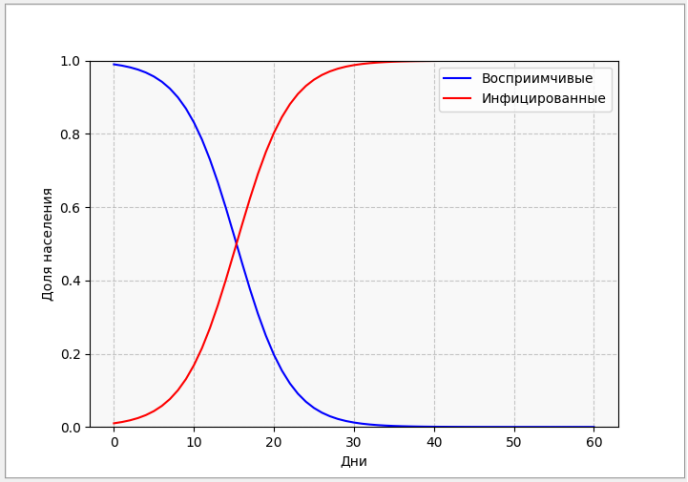


Рисунок 26 – Результат SI-модели для первого контрольного примера с помощью метода Рунге-Кутта [17] 4-го порядка

Для второго контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $I_0 = 0.01$, $\beta = 0.6$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Эйлера. Результат прогнозирования показан на рисунке 27.

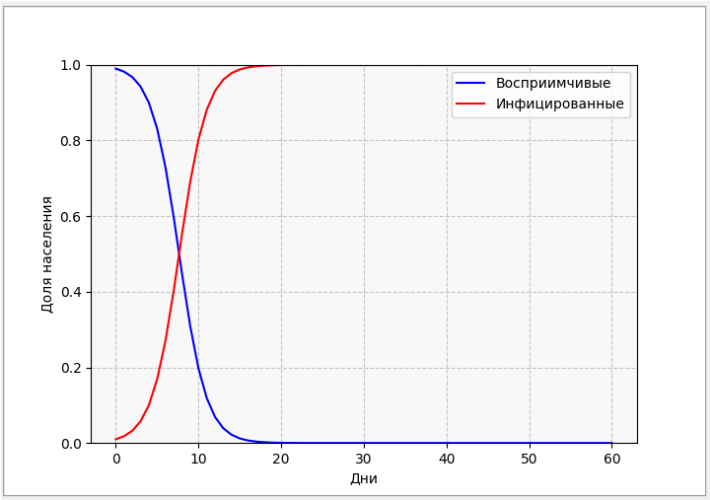


Рисунок 27 - Результат SI-модели для второго контрольного примера с помощью метода Рунге-Кутта 4-го порядка

На графике видно, что с увеличением скорости заражения рост инфицированных ускоряется. Количество инфицированных и восприимчивых к болезни наступает в первый десяток дней, когда в предыдущем примере это происходило только во второй десяток.

4.3.3 SIRS-модель

SIRS-модель состоит из трех начальных значений (восприимчивые, инфицированные и выздоровевшие) и трех параметров (скорость заражения, скорость выздоровления и скорость потери иммунитета). Для первого контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $I_0 = 0.01$, $R_0 = 0.0$, $\beta = 0.3$, $\gamma = 0.1$, $\delta = 0.01$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 28.

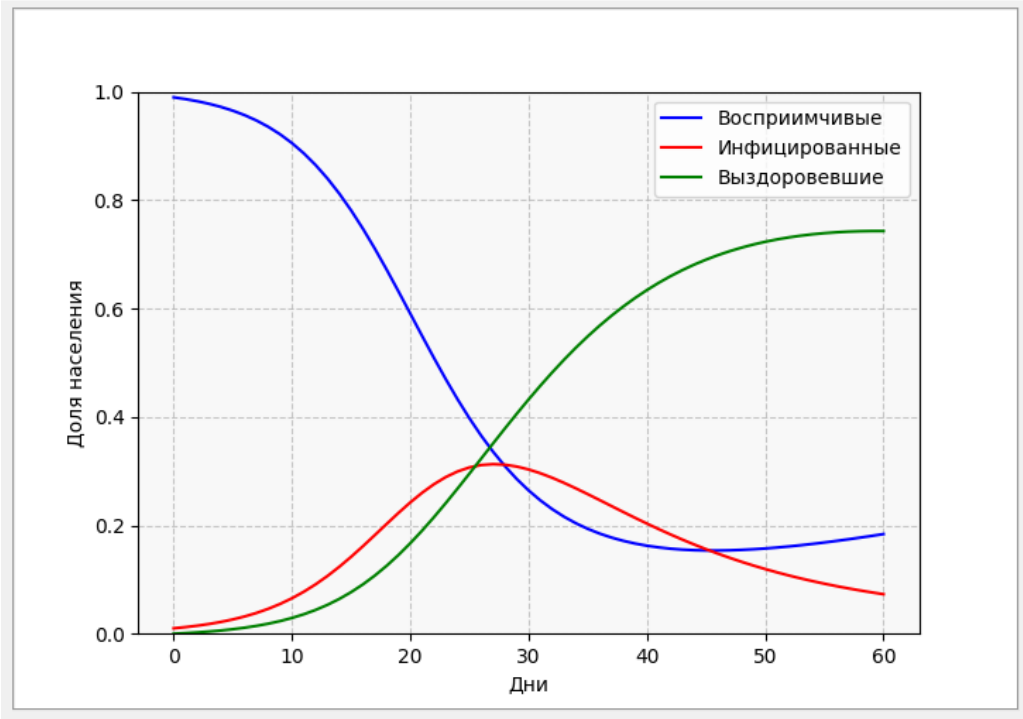


Рисунок 28 – Результат SIRS-модели для первого контрольного примера с помощью метода Рунге-Кутта 4-го порядка

Для второго контрольного примера возьмем следующие начальные данные: $S_0 = 0.99, I_0 = 0.01, R_0 = 0.0, \beta = 0.3, \gamma = 0.1, \delta = 0.5$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 29.

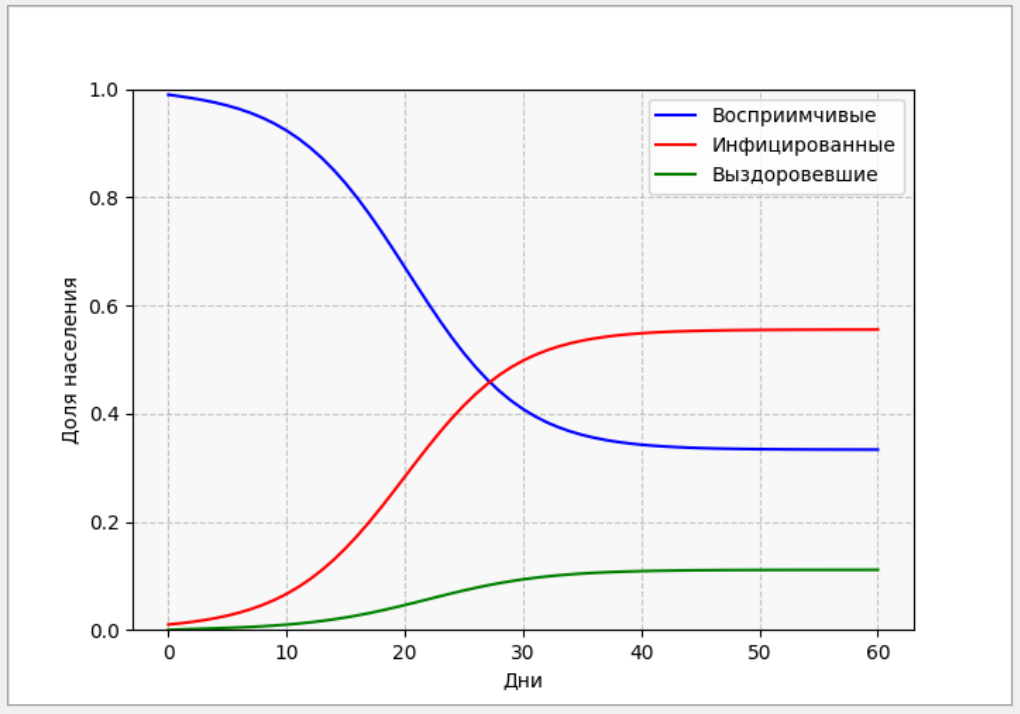


Рисунок 29 – Результат SIRS-модели для второго контрольного примера с помощью метода Рунге-Кутта 4-го порядка

На графике видно, что увеличение параметра скорости потери иммунитета изменяет динамику течения эпидемии. Выздоровевшая доля населения быстрее теряет иммунитет, а значит быстрее заболевает повторно. Если в первом контрольном примере инфицированных к 60 дню было около 0.1, то во втором контрольном примере их почти 0.6 и их количество практически не изменяется.

4.3.4 SIQR-модель

SIQR-модель состоит из четырех начальных значений (восприимчивые, инфицированные, изолированные и выздоровевшие) и четырех параметров

(скорость заражения, скорость выздоровления, скорость перехода на карантин и скорость выздоровления изолированных). Для первого контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $I_0 = 0.01$, $Q_0 = 0.0$, $R_0 = 0.0$, $\beta = 0.3$, $\gamma = 0.1$, $\delta = 0.05$, $\mu = 0.05$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 30.

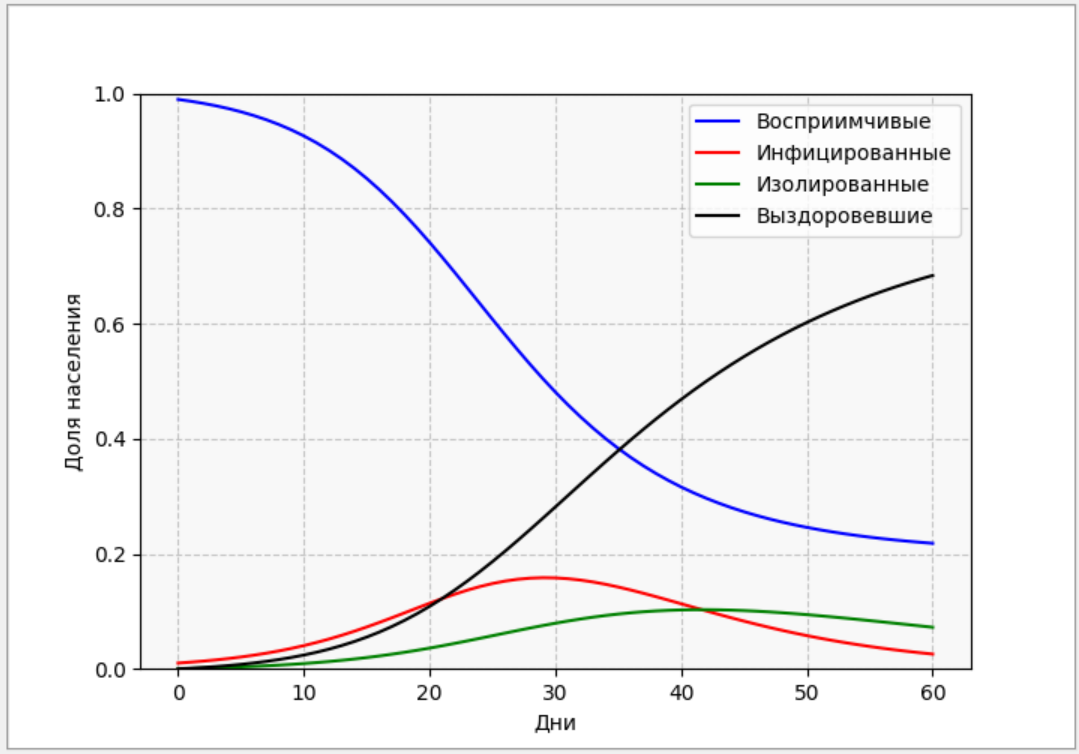


Рисунок 30 – Результат SIQR-модели для первого контрольного примера с помощью метода Рунге-Кутта 4-го порядка

Для второго контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $I_0 = 0.01$, $Q_0 = 0.0$, $R_0 = 0.0$, $\beta = 0.3$, $\gamma = 0.1$, $\delta = 0.1$, $\mu = 0.05$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 31.

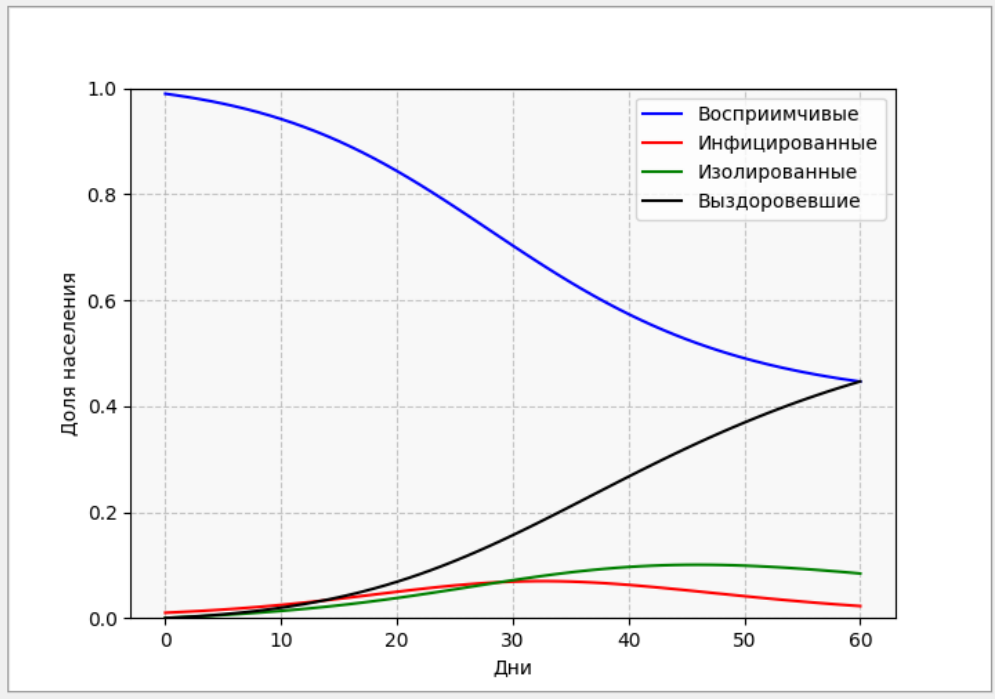


Рисунок 31 – Результат SIQR-модели для второго контрольного примера с помощью метода Рунге-Кутта 4-го порядка

На графике видно, что увеличение параметра скорости изоляции инфицированных сильно влияет на течение эпидемии. В первом случае количество инфицированных начинало снижаться с увеличением роста изолированных, но их пик наступил позже и находится в пределах 0.1 доли населения, когда пик инфицированных был примерно 0.7-0.8. Во втором случае наблюдается ситуация, когда изолированных намного больше, чем инфицированных. Количество выздоровевших и восприимчивых совпадает ровно на 60 день, а рост инфицированных снижается до 0.05-0.1.

4.3.5 SEIR-модель

SEIR-модель состоит из четырех начальных значений (восприимчивые, инфицированные, латентные и выздоровевшие) и трех параметров (скорость заражения, скорость выздоровления и скорость выхода из латентного периода). Для первого контрольного примера возьмем следующие начальные данные: $S_0 = 0.99,\ E_0 = 0.0,\ I_0 = 0.0,\ R_0 = 0.0,\ \beta = 0.3,\ \gamma = 0.1,\ \sigma = 0.2$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что

составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 32.

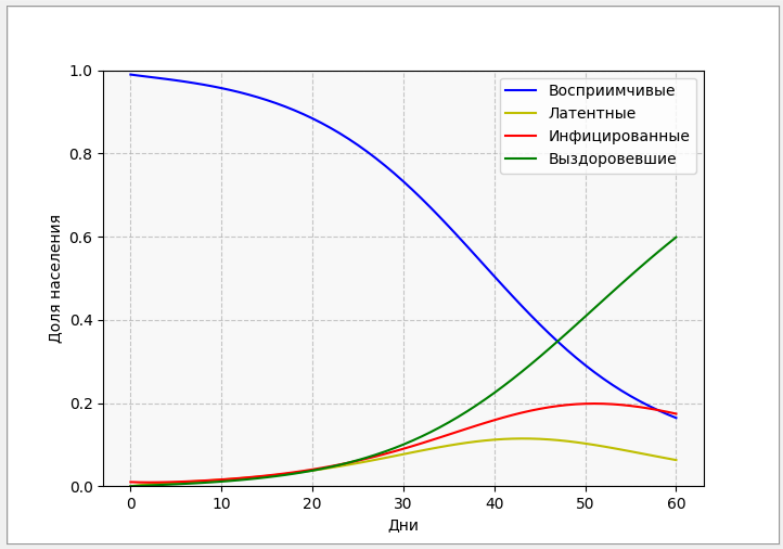


Рисунок 32 – Результат SEIR-модели для первого контрольного примера с помощью метода Рунге-Кутта 4-го порядка

Для второго контрольного примера возьмем следующие начальные данные: $S_0 = 0.99$, $E_0 = 0.0$, $I_0 = 0.01$, $R_0 = 0.0$, $\beta = 0.3$, $\gamma = 0.1$, $\sigma = 0.4$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 33.

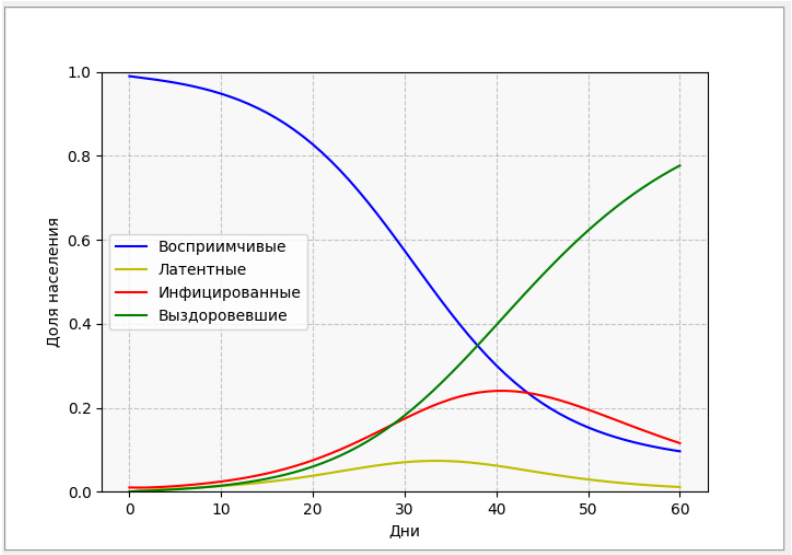


Рисунок 33 – Результат SEIR-модели для второго контрольного примера с помощью метода Рунге-Кутта 4-го порядка

На графике видно, что увеличение параметра выхода из латентного состояния способствует резкому росту количества инфицированных. Пик эпидемии наступает раньше, чем в первом случае. Количество инфицированных также становится большем. В первом случае в пике их было не больше 0.2, когда во втором случае их пик приходится на 0.25-0.3

4.3.6 MSEIR-модель

MSEIR-модель состоит из пяти начальных значений (младенцы с материнским иммунитетом, восприимчивые, инфицированные, латентные и выздоровевшие) и пяти параметров (скорость заражения, скорость выздоровления, скорость выхода из латентного периода, естественная смертность/рождаемость и скорость потери материнского иммунитета). Для первого контрольного примера возьмем следующие начальные данные: $M_0 = 0.3, S_0 = 0.69, E_0 = 0.0, I_0 = 0.01, R_0 = 0.0, \beta = 0.5, \gamma = 0.14, \sigma = 0.2, \delta = 0.1, \mu = 0.01$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Рунге-Кутта 4-го порядка. Результат прогнозирования показан на рисунке 34.

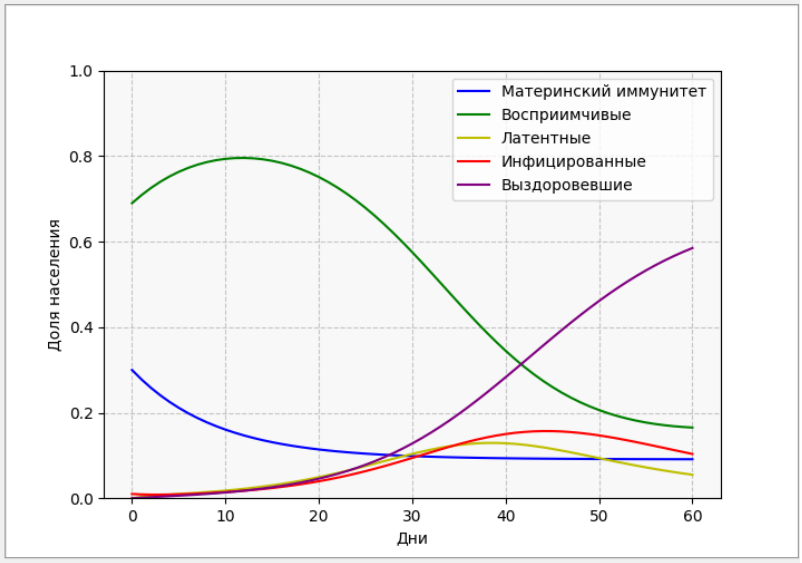


Рисунок 34 – Результат MSEIR-модели для первого контрольного примера с помощью метода Рунге-Кутта 4-го порядка

Для второго контрольного примера возьмем следующие начальные данные: $M_0 = 0.3$, $S_0 = 0.69$, $E_0 = 0.0$, $I_0 = 0.01$, $R_0 = 0.0$, $\beta = 0.5$, $\gamma = 0.14$, $\sigma = 0.2$, $\delta = 0.01$, $\mu = 0.01$. Начальная дата моделирования 15.05.2025, конечная дата моделирования 14.07.2025, что составляет 60 дней. СДУ решается с помощью численного метода Эйлера. Результат прогнозирования показан на рисунке 35.

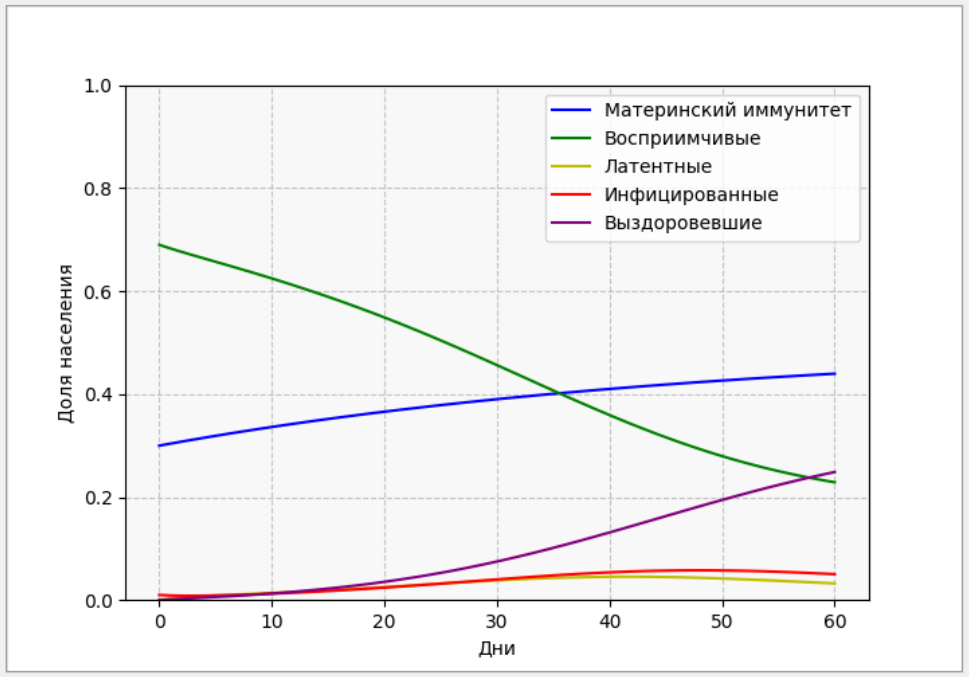


Рисунок 35 – Результат MSEIR-модели для второго контрольного примера с помощью метода Рунге-Кутта 4-го порядка

На графике видно, что уменьшение параметра потери материнского иммунитета снижает пик инфицированных, а также способствует сдерживанию эпидемии, так как к 60 дню количество заболевших снижается.

4.4 Анализ результатов моделирования

На основе представленных в данной работе математических моделей и их тестировании можно выделить следующие особенности:

— Модель SIR подходит для моделирования заболеваний, иммунитет для которых постоянен и сохраняется на всю жизнь. К таким заболеваниям относится ветряная оспа;

— Модель SI подходит для моделирования заболеваний, у которых отсутствует этап выздоровления. К таким заболеваниям относится гепатит B и C, а также ВИЧ/СПИД инфекции;

— Модель SIRS позволяет прогнозировать развитие эпидемиологической ситуации для более продолжительного срока, так как она учитывает потерю иммунитета у населения. Данная модель наиболее подходит для моделирования распространения ротавирусных и респираторных вирусов;

— Модель SEIR позволяет моделировать инфекции, у которых есть латентный (инкубационный) период. К таким инфекциям относится COVID-19;

— Модель SIQR подходит для моделирования развития эпидемиологической ситуации, при которой необходимо проанализировать воздействие карантинных мер. Данная модель наиболее полезна при моделировании развития эпидемии COVID-19 и других респираторных инфекций;

— Модель MSEIR подходит для моделирования распространения инфекций наиболее опасных для новорожденных. К таким заболеваниям относится корь.

Подводя итог анализа моделей, стоит отметить, что для каждой инфекции необходимо выбирать наиболее подходящий инструментарий, позволяющий представить полную картину возможного развития эпидемиологической ситуации. В таблице 4 приведены инфекции и модели, наиболее подходящие для их моделирования.

Таблица 4 – Инфекции и модели подходящие для их моделирования

Название модели	Наиболее подходящие инфекции для моделирования
SIR	Грипп, ветряная оспа, корь, эпидемический паротит
SI	ВИЧ, гепатит B/C и другие хронические вирусные инфекции
SIRS	Риновирусы, ротавирусные инфекции, стрептококк группы A
SIQR	COVID-19, туберкулез, Эбола, другие инфекции с жесткими карантинными мерами
SEIR	COVID-19, туберкулез, корь, ветряная оспа
MSEIR	Краснуха, корь, ветряная оспа, инфекции с пассивным материнским иммунитетом

4.5 Выводы по главе

В данной главе протестировано программное средство и реализованные математические модели на различных начальных данных. На основе анализа представленных в данной работе моделей выделены инфекции наиболее подходящие для моделирования развития эпидемиологической ситуации.

Заключение

В ходе работы создано программное средство для математического моделирования динамики эпидемиологической ситуации. Разработанная система предоставляет возможность проведения численных экспериментов с использованием различных параметров модели, анализа распространения инфекции и оценки эффективности мер по ее контролю.

Результаты тестирования подтвердили корректность работы программы и ее пригодность для решения актуальных задач в области эпидемиологии. Созданный инструмент может быть использован как для научных исследований, так и в качестве вспомогательного средства для принятия решений в сфере управления при возникновении реальной эпидемической угрозы.

Программное средство позволяет расширить свои функциональные возможности, в том числе внедрение более сложных моделей, учет пространственных факторов и автоматическая калибровка параметров по реальным эпидемиологическим данным.

Перечень используемых информационных источников

1. Кермак У., МакКендрик А. Математическая теория эпидемий. — 1927.
2. Бернулли Д. Essai d'une nouvelle analyse de la mortalité causée par la petite vérole. — 1760.
3. Diekmann O., Heesterbeek J.A.P. Mathematical Epidemiology of Infectious Diseases. — Wiley, 2000. ISBN 978-0-471-49841-3.
4. Hethcote H.W. The Mathematics of Infectious Diseases. — SIAM Review, 2000. DOI: 10.1137/S0036144500371907.
5. Brauer F., Castillo-Chavez C., Feng Z. Mathematical Models in Epidemiology. — Springer, 2019. ISBN 978-1-4939-9827-3.
6. Ross R. The Prevention of Malaria. — John Murray, 1911.
7. Anderson R.M., May R.M. Infectious Diseases of Humans: Dynamics and Control. — Oxford University Press, 1991. ISBN 978-0-19-854040-3.
8. Keeling M.J., Rohani P. Modeling Infectious Diseases in Humans and Animals. — Princeton University Press, 2008. ISBN 978-0-691-11617-4.
9. Edelstein-Keshet L. Mathematical Models in Biology. — SIAM, 2005. ISBN 978-0-89871-554-5.
10. Murray J.D. Mathematical Biology. — Springer, 2002. ISBN 978-0-387-95223-1.
11. Allen L.J.S. An Introduction to Mathematical Biology. — Pearson Education, 2007. ISBN 978-0-13-045591-1.
12. Python Software Foundation. Python 3.13 Documentation. — https://docs.python.org/3.13/
13. Лутц М. Изучаем Python. — СПб: Символ-Плюс, 2021. ISBN 978-5-6046440-6-4.
14. Гудман Б. Python и анализ данных. — М.: Диалектика, 2022. ISBN 978-5-8459-2531-2.

15. Бутлер М. Программирование на Python: от простого к сложному. — СПб: Питер, 2020. ISBN 978-5-4461-0925-0.

16. Сиденко С. Методы численного анализа. — М.: Физматлит, 2018. ISBN 978-5-9221-1813-9.

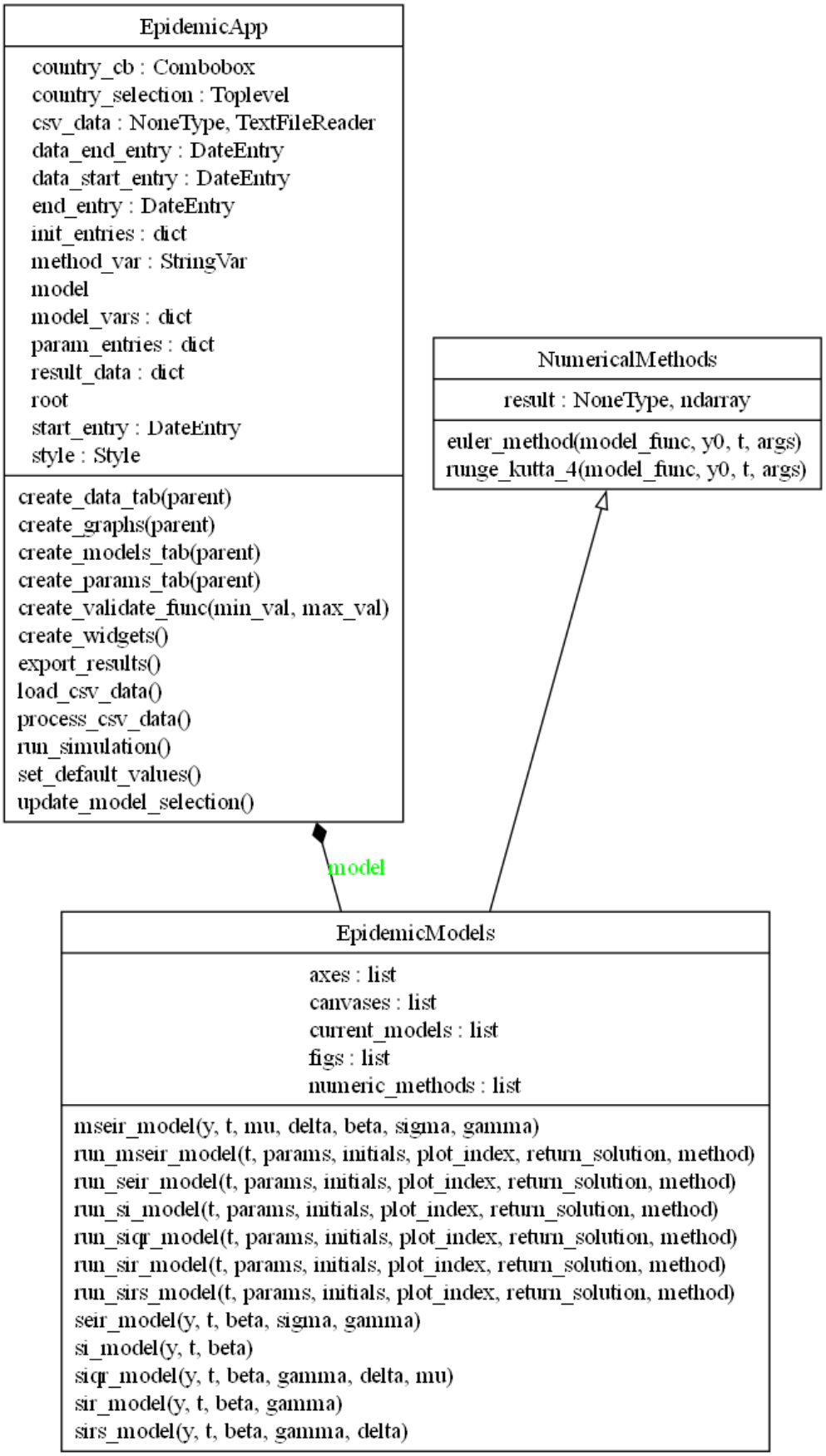
17. Кутта М. Введение в численные методы. — М.: Наука, 1987.

18. Guckenheimer J., Holmes P. Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields. — Springer, 2002. ISBN 978-0-387-90819-1.

19. Воронин А.А. Математические модели распространения эпидемий. — М.: МГУ, 2019. ISBN 978-5-19-011784-6.

20. Арсеньев В.В. Математические методы в биологии. — М.: URSS, 2015. ISBN 978-5-9710-2073-0.

Приложение А UML-диаграмма программного средства



Приложение Б Исходный код программного средства

Листинг 1 – Исходный код программного средства

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
from tkcalendar import DateEntry
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np
import pandas as pd
import os
import zipfile
from io import BytesIO
import re

class NumericalMethods:
    def __init__(self):
        self.result = None

    def euler_method(self, model_func, y0, t, args):
        y = np.zeros((len(t), len(y0)))
        y[0] = y0

        for i in range(1, len(t)):
            dt = t[i] - t[i-1]
            dy = model_func(y[i-1], t[i-1], *args)
            y[i] = y[i-1] + dy * dt
        self.result = y.T

    def runge_kutta_4(self, model_func, y0, t, args):
        y = np.zeros((len(t), len(y0)))
        y[0] = y0

        for i in range(1, len(t)):
```

```
            dt = t[i] - t[i-1]
            h = dt

            k1 = model_func(y[i-1], t[i-1], *args)
            k2 = model_func(y[i-1] + 0.5*h*k1, t[i-1] + 0.5*h,
*args)
            k3 = model_func(y[i-1] + 0.5*h*k2, t[i-1] + 0.5*h,
*args)
            k4 = model_func(y[i-1] + h*k3, t[i-1] + h, *args)

            y[i] = y[i-1] + (h/6.0)*(k1 + 2*k2 + 2*k3 + k4)

        self.result = y.T

class EpidemicModels(NumericalMethods):
    def __init__(self):
        self.current_models = []
        self.numeric_methods = []
        self.axes = []
        self.canvases = []
        self.figs = []

    def sir_model(self, y, t, beta, gamma):
        S, I, R = y
        dSdt = -beta * S * I
        dIdt = beta * S * I - gamma * I
        dRdt = gamma * I
        return np.array([dSdt, dIdt, dRdt])

    def si_model(self, y, t, beta):
        S, I = y
        dSdt = -beta * S * I
        dIdt = beta * S * I
        return np.array([dSdt, dIdt])
```

```
    def sirs_model(self, y, t, beta, gamma, delta):
        S, I, R = y
        dSdt = -beta * S * I + delta * R
        dIdt = beta * S * I - gamma * I
        dRdt = gamma * I - delta * R
        return np.array([dSdt, dIdt, dRdt])

    def siqr_model(self, y, t, beta, gamma, delta, mu):
        S, I, Q, R = y
        dSdt = -beta * S * I
        dIdt = beta * S * I - gamma * I - delta * I
        dQdt = delta * I - mu * Q
        dRdt = gamma * I + mu * Q
        return np.array([dSdt, dIdt, dQdt, dRdt])

    def seir_model(self, y, t, beta, sigma, gamma):
        S, E, I, R = y
        dSdt = -beta * S * I
        dEdt = beta * S * I - sigma * E
        dIdt = sigma * E - gamma * I
        dRdt = gamma * I
        return np.array([dSdt, dEdt, dIdt, dRdt])

    def mseir_model(self, y, t, mu, delta, beta, sigma, gamma):
        M, S, E, I, R = y
        N = M + S + E + I + R
        dMdt = mu * N - delta * M - mu * M
        dSdt = delta * M - beta * S * I / N - mu * S
        dEdt = beta * S * I / N - sigma * E - mu * E
        dIdt = sigma * E - gamma * I - mu * I
        dRdt = gamma * I - mu * R
        return np.array([dMdt, dSdt, dEdt, dIdt, dRdt])

    def run_si_model(self, t, params, initials, plot_index,
return_solution=False, method="runge_kutta"):
```

```
        y0 = [initials["S0"], initials["I0"]]

        if method == "runge_kutta":
            self.runge_kutta_4(self.si_model, y0, t,
(params["beta"],))
        else:
            self.euler_method(self.si_model, y0, t,
(params["beta"],))

        S, I = self.result

        ax = self.axes[plot_index]
        ax.clear()
        ax.plot(t, S, 'b', label='Восприимчивые')
        ax.plot(t, I, 'r', label='Инфицированные')
        ax.set_ylim(0, 1)
        ax.set_xlabel('Дни')
        ax.set_ylabel('Доля населения')
        ax.grid(True)
        ax.legend()
        self.canvases[plot_index].draw()

        if return_solution:
            return {"S": S, "I": I}

    def run_sir_model(self, t, params, initials, plot_index,
return_solution=False, method="runge_kutta"):
        y0 = [initials["S0"], initials["I0"], initials["R0"]]

        if method == "runge_kutta":
            self.runge_kutta_4(self.sir_model, y0, t,
(params["beta"], params["gamma"]))
        else:
            self.euler_method(self.sir_model, y0, t,
(params["beta"], params["gamma"]))
```

```
        S, I, R = self.result

        ax = self.axes[plot_index]
        ax.clear()
        ax.plot(t, S, 'b', label='Восприимчивые')
        ax.plot(t, I, 'r', label='Инфицированные')
        ax.plot(t, R, 'g', label='Выздоровевшие')
        ax.set_ylim(0, 1)
        ax.set_xlabel('Дни')
        ax.set_ylabel('Доля населения')
        ax.grid(True)
        ax.legend()
        self.canvases[plot_index].draw()

        if return_solution:
            return {"S": S, "I": I, "R": R}

    def run_sirs_model(self, t, params, initials, plot_index,
return_solution=False, method="runge_kutta"):
        y0 = [initials["S0"], initials["I0"], initials["R0"]]

        if method == "runge_kutta":
            self.runge_kutta_4(self.sirs_model, y0, t,
(params["beta"], params["gamma"], params["delta"]))
        else:
            self.euler_method(self.sirs_model, y0, t,
(params["beta"], params["gamma"], params["delta"]))

        S, I, R = self.result

        ax = self.axes[plot_index]
        ax.clear()
        ax.plot(t, S, 'b', label='Восприимчивые')
        ax.plot(t, I, 'r', label='Инфицированные')
```

```
        ax.plot(t, R, 'g', label='Выздоровевшие')
        ax.set_ylim(0, 1)
        ax.set_xlabel('Дни')
        ax.set_ylabel('Доля населения')
        ax.grid(True)
        ax.legend()
        self.canvases[plot_index].draw()

        if return_solution:
            return {"S": S, "I": I, "R": R}

    def run_seir_model(self, t, params, initials, plot_index,
return_solution=False, method="runge_kutta"):
        y0 = [initials["S0"], initials["E0"], initials["I0"],
initials["R0"]]

        if method == "runge_kutta":
            self.runge_kutta_4(self.seir_model, y0, t,
(params["beta"], params["sigma"], params["gamma"]))
        else:
            self.euler_method(self.seir_model, y0, t,
(params["beta"], params["sigma"], params["gamma"]))

        S, E, I, R = self.result

        ax = self.axes[plot_index]
        ax.clear()
        ax.plot(t, S, 'b', label='Восприимчивые')
        ax.plot(t, E, 'y', label='Латентные')
        ax.plot(t, I, 'r', label='Инфицированные')
        ax.plot(t, R, 'g', label='Выздоровевшие')
        ax.set_ylim(0, 1)
        ax.set_xlabel('Дни')
        ax.set_ylabel('Доля населения')
        ax.grid(True)
```

```
        ax.legend()
        self.canvases[plot_index].draw()

        if return_solution:
            return {"S": S, "E": E, "I": I, "R": R}

    def run_mseir_model(self, t, params, initials, plot_index,
return_solution=False, method="runge_kutta"):
        y0 = [initials["M0"], initials["S0"], initials["E0"],
initials["I0"], initials["R0"]]

        if method == "runge_kutta":
            self.runge_kutta_4(self.mseir_model, y0, t,
                        (params["mu"], params["delta"],
params["beta"], params["sigma"], params["gamma"]))
        else:
            self.euler_method(self.mseir_model, y0, t,
                        (params["mu"], params["delta"],
params["beta"], params["sigma"], params["gamma"]))

        M, S, E, I, R = self.result

        ax = self.axes[plot_index]
        ax.clear()
        ax.plot(t, M, 'b', label='Материнский иммунитет')
        ax.plot(t, S, 'g', label='Восприимчивые')
        ax.plot(t, E, 'y', label='Латентные')
        ax.plot(t, I, 'r', label='Инфицированные')
        ax.plot(t, R, 'purple', label='Выздоровевшие')
        ax.set_ylim(0, 1)
        ax.set_xlabel('Дни')
        ax.set_ylabel('Доля населения')
        ax.grid(True)
        ax.legend()
        self.canvases[plot_index].draw()
```

```
        if return_solution:
            return {"M": M, "S": S, "E": E, "I": I, "R": R}

    def run_siqr_model(self, t, params, initials, plot_index,
return_solution=False, method="runge_kutta"):
        y0 = [initials["S0"], initials["I0"], initials["Q0"],
initials["R0"]]

        if method == "runge_kutta":
            self.runge_kutta_4(self.siqr_model, y0, t,
                        (params["beta"], params["gamma"],
params["delta"], params["mu"]))
        else:
            self.euler_method(self.siqr_model, y0, t,
                        (params["beta"], params["gamma"],
params["delta"], params["mu"]))

        S, I, Q, R = self.result

        ax = self.axes[plot_index]
        ax.clear()
        ax.plot(t, S, 'b', label='Восприимчивые')
        ax.plot(t, I, 'r', label='Инфицированные')
        ax.plot(t, Q, 'g', label='Изолированные')
        ax.plot(t, R, 'k', label='Выздоровевшие')
        ax.set_ylim(0, 1)
        ax.set_xlabel('Дни')
        ax.set_ylabel('Доля населения')
        ax.grid(True)
        ax.legend()
        self.canvases[plot_index].draw()

        if return_solution:
            return {"S": S, "I": I, "Q": Q, "R": R}
```

```
class EpidemicApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Эпидемиологическое моделирование")
        self.root.geometry("1200x800")
        self.root.minsize(1000, 700)

        self.style = ttk.Style()
        self.style.configure('TFrame', background='#f0f0f0')
        self.style.configure('TLabel', background='#f0f0f0',
font=('Arial', 10))
        self.style.configure('TButton', font=('Arial', 10))
        self.style.configure('TCheckbutton', background='#f0f0f0',
font=('Arial', 10))
        self.style.configure('TRadiobutton', background='#f0f0f0',
font=('Arial', 10))

        self.model = EpidemicModels()
        self.result_data = {}

        self.create_widgets()
        self.set_default_values()

    def create_widgets(self):
        main_paned = ttk.PanedWindow(self.root,
orient=tk.HORIZONTAL)
        main_paned.pack(fill=tk.BOTH, expand=True)

        control_frame = ttk.Frame(main_paned, width=350,
relief=tk.RIDGE, padding=10)
        main_paned.add(control_frame, weight=0)

        graph_frame = ttk.Frame(main_paned)
        main_paned.add(graph_frame, weight=1)
```

```
        control_notebook = ttk.Notebook(control_frame)
        control_notebook.pack(fill=tk.BOTH, expand=True)

        models_tab = ttk.Frame(control_notebook)
        control_notebook.add(models_tab, text="Модели")
        self.create_models_tab(models_tab)

        params_tab = ttk.Frame(control_notebook)
        control_notebook.add(params_tab, text="Параметры")
        self.create_params_tab(params_tab)

        data_tab = ttk.Frame(control_notebook)
        control_notebook.add(data_tab, text="Данные")
        self.create_data_tab(data_tab)

        self.create_graphs(graph_frame)

    def create_models_tab(self, parent):
        models_group = ttk.LabelFrame(parent, text="Выберите
модели (макс. 4)", padding=10)
        models_group.pack(fill=tk.BOTH, pady=5)

        self.model_vars = {}
        models = [
            ('SI', 'Модель SI (восприимчивые-инфицированные)'),
            ('SIR', 'Модель SIR (восприимчивые-инфицированные-
выздоровевшие)'),
            ('SIRS', 'Модель SIRS (с временным иммунитетом)'),
            ('SEIR', 'Модель SEIR (с латентным периодом)'),
            ('SIQR', 'Модель SIQR (с изоляцией)'),
            ('MSEIR', 'Модель MSEIR (с материнским иммунитетом)')
        ]

        for model_code, model_desc in models:
```

```
            var = tk.BooleanVar()
            cb = ttk.Checkbutton(models_group, text=model_desc,
variable=var,

command=self.update_model_selection)
            cb.pack(anchor='w', padx=5, pady=2)
            self.model_vars[model_code] = var

        method_group = ttk.LabelFrame(parent, text="Метод
решения", padding=10)
        method_group.pack(fill=tk.BOTH, pady=5)

        self.method_var = tk.StringVar(value="runge_kutta")
        ttk.Radiobutton(method_group, text="Рунге-Кутта 4-го
порядка",
                        variable=self.method_var,
value="runge_kutta").pack(anchor='w', padx=5, pady=2)
        ttk.Radiobutton(method_group, text="Метод Эйлера",
                        variable=self.method_var,
value="euler").pack(anchor='w', padx=5, pady=2)

        ttk.Button(parent, text="Запустить моделирование",
                   command=self.run_simulation).pack(fill=tk.X,
pady=10)

    def create_params_tab(self, parent):
        canvas = tk.Canvas(parent)
        scrollbar = ttk.Scrollbar(parent, orient="vertical",
command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(
                scrollregion=canvas.bbox("all")
```

```
            )
        )

        canvas.create_window((0, 0), window=scrollable_frame,
anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        params_group = ttk.LabelFrame(scrollable_frame,
text="Параметры модели", padding=10)
        params_group.pack(fill=tk.X, pady=5)

        self.param_entries = {}
        params = [
            ("beta", "β (скорость заражения)", 0.0, 1.0),
            ("gamma", "γ (скорость выздоровления)", 0.0, 1.0),
            ("delta", "δ (потеря иммунитета)", 0.0, 1.0),
            ("sigma", "σ (переход в инфекционные)", 0.0, 1.0),
            ("mu", "μ (выход из изоляции)", 0.0, 1.0)
        ]

        for param_code, param_desc, min_val, max_val in params:
            row = ttk.Frame(params_group)
            row.pack(fill=tk.X, pady=2)

            ttk.Label(row, text=param_desc,
width=25).pack(side=tk.LEFT)
            entry = ttk.Entry(row, width=8)
            entry.pack(side=tk.RIGHT)
            self.param_entries[param_code] = entry

        initials_group = ttk.LabelFrame(scrollable_frame,
text="Начальные значения (доля)", padding=10)
```

```
        initials_group.pack(fill=tk.X, pady=5)

        self.init_entries = {}
        initials = [
            ("S0", "S₀ (восприимчивые)", 0.0, 1.0),
            ("I0", "I₀ (инфицированные)", 0.0, 1.0),
            ("R0", "R₀ (выздоровевшие)", 0.0, 1.0),
            ("E0", "E₀ (латентные)", 0.0, 1.0),
            ("Q0", "Q₀ (изолированные)", 0.0, 1.0),
            ("M0", "M₀ (материнский иммунитет)", 0.0, 1.0)
        ]

        for init_code, init_desc, min_val, max_val in initials:
            row = ttk.Frame(initials_group)
            row.pack(fill=tk.X, pady=2)

            ttk.Label(row, text=init_desc,
width=25).pack(side=tk.LEFT)
            entry = ttk.Entry(row, width=8)
            entry.pack(side=tk.RIGHT)
            self.init_entries[init_code] = entry

        time_group = ttk.LabelFrame(scrollable_frame,
text="Временной диапазон", padding=10)
        time_group.pack(fill=tk.X, pady=5)

        ttk.Label(time_group, text="Начальная
дата:").pack(anchor='w', pady=(0, 5))
        self.start_entry = DateEntry(time_group,
date_pattern='dd.mm.yyyy')
        self.start_entry.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(time_group, text="Конечная
дата:").pack(anchor='w', pady=(0, 5))
```

```
        self.end_entry = DateEntry(time_group,
date_pattern='dd.mm.yyyy')
        self.end_entry.pack(fill=tk.X)

    def create_data_tab(self, parent):
        load_group = ttk.LabelFrame(parent, text="Загрузка
данных", padding=10)
        load_group.pack(fill=tk.BOTH, pady=5, expand=True)

        ttk.Button(load_group, text="Загрузить данные из CSV",
                   command=self.load_csv_data).pack(fill=tk.X,
pady=5)

        export_group = ttk.LabelFrame(parent, text="Экспорт
результатов", padding=10)
        export_group.pack(fill=tk.BOTH, pady=5)

        ttk.Button(export_group, text="Экспорт в Excel и ZIP",
                   command=self.export_results).pack(fill=tk.X,
pady=5)

    def create_graphs(self, parent):
        self.model.axes = []
        self.model.canvases = []
        self.model.figs = []

        for i in range(4):
            fig = plt.Figure(figsize=(6, 4), dpi=100)
            ax = fig.add_subplot(111)

            ax.grid(True, linestyle='--', alpha=0.7)
            ax.set_facecolor('#f8f8f8')

            canvas = FigureCanvasTkAgg(fig, master=parent)
            canvas_widget = canvas.get_tk_widget()
```

```
            canvas_widget.grid(row=i//2, column=i%2, padx=5,
pady=5, sticky='nsew')
            canvas_widget.config(borderwidth=2, relief=tk.GROOVE)

            parent.grid_rowconfigure(i//2, weight=1)
            parent.grid_columnconfigure(i%2, weight=1)

            self.model.figs.append(fig)
            self.model.axes.append(ax)
            self.model.canvases.append(canvas)

    def create_validate_func(self, min_val, max_val):
        def validate(value):
            if value == "":
                return True
            try:
                num = float(value)
                return min_val <= num <= max_val
            except ValueError:
                return False
        return validate

    def set_default_values(self):
        self.param_entries["beta"].insert(0, "0.3")
        self.param_entries["gamma"].insert(0, "0.1")
        self.param_entries["delta"].insert(0, "0.01")
        self.param_entries["sigma"].insert(0, "0.2")
        self.param_entries["mu"].insert(0, "0.05")

        self.init_entries["S0"].insert(0, "0.99")
        self.init_entries["I0"].insert(0, "0.01")
        self.init_entries["R0"].insert(0, "0.0")
        self.init_entries["E0"].insert(0, "0.0")
        self.init_entries["Q0"].insert(0, "0.0")
        self.init_entries["M0"].insert(0, "0.0")
```

```
        today = datetime.now()
        self.start_entry.set_date(today)
        self.end_entry.set_date(today + timedelta(days=100))

    def update_model_selection(self):
        selected = sum(var.get() for var in
self.model_vars.values())

        if selected > 4:
            for model_code, var in
reversed(self.model_vars.items()):
                if var.get():
                    var.set(False)
                    messagebox.showwarning("Предупреждение",
"Можно выбрать не более 4 моделей одновременно")
                    break

    def run_simulation(self):
        selected_models = [name for name, var in
self.model_vars.items() if var.get()]
        if not selected_models:
            messagebox.showwarning("Ошибка", "Выберите хотя бы
одну модель")
            return

        try:
            params = {k: float(e.get()) if e.get() else 0.0 for k,
e in self.param_entries.items()}
            initials = {k: float(e.get()) if e.get() else 0.0 for
k, e in self.init_entries.items()}
        except ValueError:
            messagebox.showerror("Ошибка", "Некорректные значения
параметров")
            return
```

```
        if self.end_entry.get_date() <=
self.start_entry.get_date():
            messagebox.showerror("Ошибка", "Конечная дата должна
быть позже начальной")
            return

        self.result_data.clear()
        days = (self.end_entry.get_date() -
self.start_entry.get_date()).days
        t = np.linspace(0, days, days + 1)
        method = self.method_var.get()

        for ax in self.model.axes:
            ax.clear()

        for i, model_code in enumerate(selected_models):
            if i >= 4:
                break

            if model_code == "SI":
                sol = self.model.run_si_model(t, params, initials,
i, return_solution=True, method=method)
            elif model_code == "SIR":
                sol = self.model.run_sir_model(t, params,
initials, i, return_solution=True, method=method)
            elif model_code == "SIRS":
                sol = self.model.run_sirs_model(t, params,
initials, i, return_solution=True, method=method)
            elif model_code == "SEIR":
                sol = self.model.run_seir_model(t, params,
initials, i, return_solution=True, method=method)
            elif model_code == "SIQR":
                sol = self.model.run_siqr_model(t, params,
initials, i, return_solution=True, method=method)
```

```
            elif model_code == "MSEIR":
                sol = self.model.run_mseir_model(t, params,
initials, i, return_solution=True, method=method)

            if sol:
                self.result_data[model_code] = pd.DataFrame(sol,
index=t)

        for canvas in self.model.canvases:
            canvas.draw()

    def load_csv_data(self):
        path = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
        if not path:
            return

        try:
            df = pd.read_csv(path)
            df["Date"] = pd.to_datetime(df["Date"])
            df.sort_values("Date", inplace=True)
            self.csv_data = df

            countries =
sorted(df["Country/Region"].dropna().unique())

            self.country_selection = tk.Toplevel(self.root)
            self.country_selection.title("Выбор страны и дат")
            self.country_selection.geometry("400x300")

            ttk.Label(self.country_selection, text="Выберите
страну:").pack(pady=(10, 5))
            self.country_cb = ttk.Combobox(self.country_selection,
values=countries, state="readonly")
            self.country_cb.pack(pady=5)
```

```
            ttk.Label(self.country_selection, text="Выберите
диапазон дат для начальных данных:").pack(pady=(10, 5))

            date_frame = ttk.Frame(self.country_selection)
            date_frame.pack(pady=5)

            ttk.Label(date_frame, text="С:").pack(side=tk.LEFT)
            self.data_start_entry = DateEntry(date_frame,
date_pattern='dd.mm.yyyy')
            self.data_start_entry.pack(side=tk.LEFT, padx=5)

            ttk.Label(date_frame, text="По:").pack(side=tk.LEFT)
            self.data_end_entry = DateEntry(date_frame,
date_pattern='dd.mm.yyyy')
            self.data_end_entry.pack(side=tk.LEFT, padx=5)

            min_date = df["Date"].min().to_pydatetime()
            max_date = df["Date"].max().to_pydatetime()
            self.data_start_entry.set_date(min_date)
            self.data_end_entry.set_date(max_date)

            ttk.Button(self.country_selection, text="Загрузить
данные",
                       command=self.process_csv_data).pack(pady=10)
        except Exception as e:
            messagebox.showerror("Ошибка", f"Не удалось загрузить
файл: {str(e)}")

    def process_csv_data(self):
        country = self.country_cb.get()
        if not country:
            messagebox.showwarning("Ошибка", "Выберите страну")
            return
```

```
        try:
            start_date = self.data_start_entry.get_date()
            end_date = self.data_end_entry.get_date()

            if start_date > end_date:
                messagebox.showerror("Ошибка", "Начальная дата не
может быть позже конечной")
                return

            df_country = self.csv_data[
                (self.csv_data["Country/Region"] == country) &
                (self.csv_data["Date"] >=
pd.to_datetime(start_date)) &
                (self.csv_data["Date"] <=
pd.to_datetime(end_date))
            ].copy()

            if df_country.empty:
                messagebox.showerror("Ошибка", "Нет данных для
выбранного диапазона дат")
                return

            latest = df_country.iloc[-1]
            total = latest["Confirmed"] + latest["Recovered"] +
latest["Deaths"]
            if total == 0:
                total = 1

            S0 = 1 - (latest["Confirmed"] + latest["Recovered"] +
latest["Deaths"]) / total
            I0 = latest["Confirmed"] / total
            R0 = latest["Recovered"] / total

            for entry in self.init_entries.values():
                entry.delete(0, tk.END)
```

```
            self.init_entries["S0"].insert(0, f"{S0:.4f}")
            self.init_entries["I0"].insert(0, f"{I0:.4f}")
            self.init_entries["R0"].insert(0, f"{R0:.4f}")

            self.start_entry.set_date(end_date)
            self.end_entry.set_date(end_date +
timedelta(days=100))

            self.country_selection.destroy()
            messagebox.showinfo("Успех", f"Данные для {country} за
период {start_date.strftime('%d.%m.%Y')}-
{end_date.strftime('%d.%m.%Y')} успешно загружены")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Не удалось обработать
данные: {str(e)}")

    def export_results(self):
        if not self.result_data:
            messagebox.showwarning("Нет данных", "Сначала
выполните моделирование")
            return

        save_path = filedialog.asksaveasfilename(
            defaultextension=".zip",
            filetypes=[("ZIP files", "*.zip")],
            initialfile="epidemic_results.zip"
        )

        if not save_path:
            return

        try:
            mem_zip = BytesIO()
```

```
            with zipfile.ZipFile(mem_zip, mode='w',
compression=zipfile.ZIP_DEFLATED) as zf:
                for model_name, df in self.result_data.items():
                    excel_buffer = BytesIO()
                    with pd.ExcelWriter(excel_buffer,
engine='xlsxwriter') as writer:
                        initials_df = pd.DataFrame({
                            'Параметр': ['S₀ (восприимчивые)', 'I₀
(инфицированные)', 'R₀ (выздоровевшие)',
                                         'E₀ (латентные)', 'Q₀
(изолированные)', 'M₀ (материнский иммунитет)'],
                            'Значение':
[self.init_entries["S0"].get(), self.init_entries["I0"].get(),

self.init_entries["R0"].get(), self.init_entries["E0"].get(),

self.init_entries["Q0"].get(), self.init_entries["M0"].get()]
                        })
                        initials_df.to_excel(writer,
sheet_name='Начальные данные', index=False)

                        params_df = pd.DataFrame({
                            'Параметр': ['β (скорость заражения)',
'γ (скорость выздоровления)',
                                         'δ (потеря иммунитета)',
'σ (переход в инфекционные)',
                                         'μ (выход из изоляции)'],
                            'Значение':
[self.param_entries["beta"].get(),
self.param_entries["gamma"].get(),

self.param_entries["delta"].get(),
self.param_entries["sigma"].get(),

self.param_entries["mu"].get()]
```

```
                        })
                        params_df.to_excel(writer,
sheet_name='Параметры', index=False)

                        method_info = pd.DataFrame({
                            'Информация': ['Метод решения:',
'Выбранная модель:'],
                            'Значение':
[self.method_var.get().replace("runge_kutta", "Рунге-Кутта 4-го
порядка").replace("euler", "Метод Эйлера"),
                                                model_name]
                        })

                        method_info.to_excel(writer,
sheet_name='Решение', startrow=0, index=False, header=False)

                        df.to_excel(writer, sheet_name='Решение',
startrow=3, index=True)

                        worksheet = writer.sheets['Решение']
                        worksheet.write(3, 0, 'Дни')

                        graph_sheet =
writer.book.add_worksheet('График')
                        chart = writer.book.add_chart({'type':
'line'})

                        max_row = len(df) + 4
                        categories =
f"='Решение'!$A$5:$A${max_row}"

                        for i, col in enumerate(df.columns, 1):
                            chart.add_series({
                                'name':
f"='Решение'!${chr(66+i)}$4",
```

```
                                'categories': categories,
                                'values':
f"='Решение'!${chr(66+i)}$5:${chr(66+i)}${max_row}",
                            })

                        chart.set_x_axis({'name': 'Дни'})
                        chart.set_y_axis({'name': 'Доля
населения'})
                        chart.set_title({'name': f'Модель
{model_name} ({method_info.iloc[0,1]})'})

                        graph_sheet.insert_chart('B2', chart,
{'x_scale': 2, 'y_scale': 2})

                    zf.writestr(f"{model_name}.xlsx",
excel_buffer.getvalue())

            with open(save_path, "wb") as f:
                f.write(mem_zip.getvalue())

            messagebox.showinfo("Экспорт завершён", f"Файлы
успешно сохранены в архив:\n{save_path}")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Не удалось
экспортировать данные: {str(e)}")

if __name__ == "__main__":
    root = tk.Tk()
    app = EpidemicApp(root)
    root.mainloop()
```