Имитационное моделирование. Практикум

А. В. Королькова Кулябов Д. С.

Содержание

Введение **5**
Прагматика построения курса 5

1 Подготовка стенда **7**
1.1 Программная инженерия 7
1.2 Использование git 11
1.3 Литературное программирование 27
1.4 Julia. Пакет DrWatson 30
1.5 Язык Markdown 34
1.6 Соглашение об именовании Denote 49
1.7 Настройка git 53
1.8 Рабочее пространство лабораторной работы 56
1.9 Создание проекта DrWatson для лабораторных работ 65
1.10 Модель экспоненциального роста 70
1.11 Задание 85

2 Основные модели **87**
2.1 Модель SIR 87
2.2 Модель Лотки–Вольтерры 100
2.3 Задание 113

3 Агентное моделирование **115**
3.1 Агентный подход 115
3.2 Daisyworld 118
3.3 Задание 134

4 Реализация основных моделей в агентном подходе **137**
4.1 Эпидемиологическая модели SIR 137
4.2 Задание 163

5 Аппарат сетей Петри **165**
5.1 Задание 165

6 Реализация основных моделей в подходе сетей Петри **167**
6.1 Задание 167

7 Дискретно-событийное моделирование **169**
7.1 Задание 169

8 Реализация основных моделей в дискретно-событийном подходе **171**
8.1 Задание 171

Введение

Современный мир, с его сложными технологическими, экономическими и социальными системами, все реже поддаётся анализу с помощью лишь аналитических методов и интуиции. Возникает потребность в инструментах, позволяющих не просто изучать, но и «проигрывать» будущее, проверять гипотезы и принимать обоснованные решения в условиях неопределённости. Таким универсальным и мощным инструментом является имитационное моделирование.

Имитационное моделирование — это экспериментальный метод познания, основанный на построении компьютерной модели системы, адекватно отражающей ее ключевые свойства и логику поведения во времени. В отличие от аналитической модели, имитационная не стремится дать единственный ответ в формульном виде. Вместо этого она позволяет наблюдать динамику системы, проводить эксперименты в виртуальной среде, варьируя входные параметры и анализируя результаты.

Однако теоретическое изучение принципов имитационного моделирования, без погружения в практику создания и экспериментирования с моделями, остаётся неполным. Знание становится подлинным умением только тогда, когда оно проходит проверку в лаборатории — виртуальной лаборатории моделирования.

Целью данной книги является формирование у читателя целостного практического навыка — от постановки задачи и разработки концептуальной модели до реализации, проведения экспериментов и анализа результатов. Каждая лабораторная работа представляет собой законченный проект, нацеленный на освоение конкретных методов и технологий.

Прагматика построения курса

Предлагается набор моделей, имеющих хорошо известное классическое решение. В данном разделе будут приведены примеры этих моделей. Это будут сквозные

задачи.

Предлагается демонстрировать применение на основе следующих сквозных задач:

— модель SIR (дифференциальные уравнения);
— модель хищник–жертва (дифференциальные уравнения).

Далее каждый подход имитационного моделирования демонстрируется на основе специфической для данного метода задачи, а также на основе сквозной практической задачи. На каждый раздел имитационного моделирования выделяется две лабораторных работы. Одна лабораторная работа на специфические задачи, одна лабораторная работа на сквозные задачи.

Предлагается использовать один язык программирования — Julia. Для каждого метода имитационного моделирования используется своя библиотека.

1

Подготовка стенда

1.1 Программная инженерия

1.1.1 Семантическое версионирование

Семантический подход в версионированию программного обеспечения.

1.1.1.1 Краткое описание семантического версионирования

- Семантическое версионирование описывается в манифесте семантического версионирования.
- Кратко его можно описать следующим образом:
  - Версия задаётся в виде кортежа `МАЖОРНАЯ_ВЕРСИЯ.МИНОРНАЯ_ВЕРСИЯ.ПАТЧ`.
  - Номер версии следует увеличивать:
    - МАЖОРНУЮ версию, когда сделаны обратно несовместимые изменения API.
    - МИНОРНУЮ версию, когда вы добавляете новую функциональность, не нарушая обратной совместимости.
    - ПАТЧ-версию, когда вы делаете обратно совместимые исправления.
  - Дополнительные обозначения для предрелизных и билд-метаданных возможны как дополнения к МАЖОРНАЯ.МИНОРНАЯ.ПАТЧ формату.

1.1.1.2 Принципы

— Коммиты должны иметь стандартизованный вид.
— В семантическое версионирование применяется вместе с *общепринятыми коммитами*.

1.1.2 Общепринятые коммиты

Использование спецификации *Conventional Commits*.

1.1.2.1 Описание

Спецификация Conventional Commits:

— Соглашение о том, как нужно писать сообщения commit'ов.
— Совместимо с SemVer.
— Регламентирует структуру и основные типы коммитов.

1.1.2.1.1 Структура коммита

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Или, по-русски:

```
<тип>(<область>): <описание изменения>
<пустая линия>
[необязательное тело]
<пустая линия>
[необязательный нижний колонтитул]
```

— Заголовок является обязательным.
— Любая строка сообщения о фиксации не может быть длиннее 100 символов.

- Тема (subject) содержит краткое описание изменения.
  - Используйте повелительное наклонение в настоящем времени: «изменить» («change» not «changed» nor «changes»).
  - Не используйте заглавную первую букву.
  - Не ставьте точку в конце.
- Тело (body) должно включать мотивацию к изменению и противопоставлять это предыдущему поведению.
  - Как и в теме, используйте повелительное наклонение в настоящем времени.
- Нижний колонтитул (footer) должен содержать любую информацию о критических изменениях.
  - Следует использовать для указания внешних ссылок, контекста коммита или другой мета информации.
  - Также содержит ссылку на issue (например, на github), который закрывает эта фиксация.
  - Критические изменения должны начинаться со слова `BREAKING CHANGE:` с пробела или двух символов новой строки. Затем для этого используется остальная часть сообщения фиксации.

1.1.2.1.2 Базовые типы коммитов

- `fix:` — коммит типа fix исправляет ошибку (bug) в вашем коде (он соответствует PATCH в SemVer).
- `feat:` — коммит типа feat добавляет новую функцию (feature) в ваш код (он соответствует MINOR в SemVer).
- `BREAKING CHANGE:` — коммит, который содержит текст `BREAKING CHANGE:` в начале своего не обязательного тела сообщения (body) или в подвале (footer), добавляет изменения, нарушающие обратную совместимость вашего API (он соответствует MAJOR в SemVer). BREAKING CHANGE может быть частью коммита любого типа.
- `revert:` — если фиксация отменяет предыдущую фиксацию. Начинается с `revert:`, за которым следует заголовок отменённой фиксации. В теле должно быть написано: Это отменяет фиксацию `<hash>` (это SHA-хэш отменяемой фиксации).
- Другое: коммиты с типами, которые отличаются от `fix:` и `feat:`, также разрешены. Например, `@commitlint/config-conventional` (основанный на The Angular convention) рекомендует: chore:, docs:, style:, refactor:, perf:, test:, и другие.

1.1.2.1.3 Соглашения The Angular convention

Одно из популярных соглашений о поддержке исходных кодов — конвенция Angular (The Angular convention).

1.1.2.1.3.1 Типы коммитов The Angular convention

Конвенция Angular (The Angular convention) требует следующие типы коммитов:

- `build:` — изменения, влияющие на систему сборки или внешние зависимости (примеры областей (scope): gulp, broccoli, npm).
- `ci:` — изменения в файлах конфигурации и скриптах CI (примеры областей: Travis, Circle, BrowserStack, SauceLabs).
- `docs:` — изменения только в документации.
- `feat:` — новая функция.
- `fix:` — исправление ошибок.
- `perf:` — изменение кода, улучшающее производительность.
- `refactor:` — Изменение кода, которое не исправляет ошибку и не добавляет функции (рефакторинг кода).
- `style:` — изменения, не влияющие на смысл кода (пробелы, форматирование, отсутствие точек с запятой и т. д.).
- `test:` — добавление недостающих тестов или исправление существующих тестов.

1.1.2.1.3.2 Области действия (scope)

Областью действия должно быть имя затронутого пакета npm (как его воспринимает человек, читающий журнал изменений, созданный из сообщений фиксации).

Есть несколько исключений из правила «использовать имя пакета»:

- `packaging` — используется для изменений, которые изменяют структуру пакета, например, изменения общедоступного пути.
- `changelog` — используется для обновления примечаний к выпуску в `CHANGELOG.md`.
- отсутствует область действия — полезно для изменений стиля, тестирования и рефакторинга, которые выполняются во всех пакетах (например, style: добавить отсутствующие точки с запятой).

1.2 Использование git

1.2.1 Первоначальна настройка git

1.2.1.1 Системы контроля версий. Общие понятия

— *Системы контроля версий (Version Control System, VCS)* применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.
— В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельта-компрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.
— Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом.
— Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.
— В отличие от классических, в распределённых системах контроля версий цен-

тральный репозиторий не является обязательным.
- Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

1.2.1.2 Примеры использования git

- Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды `git` с различными опциями.
- Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

1. Основные команды git
   - Перечислим наиболее часто используемые команды `git`.
   - Создание основного дерева репозитория:

     ```
     git init
     ```

   - Получение обновлений (изменений) текущего дерева из центрального репозитория:

     ```
     git pull
     ```

   - Отправка всех произведённых изменений локального дерева в центральный репозиторий:

     ```
     git push
     ```

   - Просмотр списка изменённых файлов в текущей директории:

     ```
     git status
     ```

   - Просмотр текущих изменений:

     ```
     git diff
     ```

   - Сохранение текущих изменений:
     - Добавить все изменённые и/или созданные файлы и/или каталоги:

       ```
       git add .
       ```

     - Добавить конкретные изменённые и/или созданные файлы и/или каталоги:

```
git add имена_файлов
```

  — Удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):

```
git rm имена_файлов
```

— Сохранение добавленных изменений:
  — Сохранить все добавленные изменения и все изменённые файлы:

```
git commit -am 'Описание коммита'
```

  — Сохранить добавленные изменения с внесением комментария через встроенный редактор:

```
git commit
```

  — Создание новой ветки, базирующейся на текущей:

```
git checkout -b имя_ветки
```

  — Переключение на некоторую ветку:

```
git checkout имя_ветки
```

    — При переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой.
  — Отправка изменений конкретной ветки в центральный репозиторий:

```
git push origin имя_ветки
```

  — Слияние ветки с текущим деревом:

```
git merge --no-ff имя_ветки
```

— Удаление ветки:
  — удаление локальной уже слитой с основным деревом ветки:

```
git branch -d имя_ветки
```

  — принудительное удаление локальной ветки:

```
git branch -D имя_ветки
```

  — удаление ветки с центрального репозитория:

```
git push origin :имя_ветки
```

2. Стандартные процедуры работы при наличии центрального репозитория
   — Работа пользователя со своей веткой начинается с проверки и получения

изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений):

```
git checkout master
git pull
git checkout -b имя_ветки
```

— Затем можно вносить изменения в локальном дереве и/или ветке.
— После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории. Для этого необходимо проверить, какие файлы изменились к текущему моменту:

```
git status
```

— При необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.
— Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов:

```
git diff
```

— Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями:

```
git add …
git rm …
```

— Если нужно сохранить все изменения в текущем каталоге, то используем:

```
git add .
```

— Затем сохраняем изменения, поясняя, что было сделано:

```
git commit -am "Some commit message"
```

— Отправляем изменения в центральный репозиторий:

```
git push origin имя_ветки
```

или

```
git push
```

3. Работа с локальным репозиторием
   — Создадим локальный репозиторий.

— Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

```
git config --global user.name "Имя Фамилия"
git config --global  user.email "work@mail"
```

— Настроим utf-8 в выводе сообщений `git`:

```
git config --global quotepath false
```

— Для инициализации локального репозитория, расположенного, например, в каталоге `~/tutorial`, необходимо ввести в командной строке:

```
cd
mkdir tutorial
cd tutorial
git init
```

— После это в каталоге `tutorial` появится каталог `.git`, в котором будет храниться история изменений.
— Создадим тестовый текстовый файл `hello.txt` и добавим его в локальный репозиторий:

```
echo 'hello world' > hello.txt
git add hello.txt
git commit -am 'Новый файл'
```

— Воспользуемся командой `status` для просмотра изменений в рабочем каталоге, сделанных с момента последней ревизии:

```
git status
```

— Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл `.gitignore` с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов:

```
curl -L -s https://www.gitignore.io/api/list
```

— Затем скачать шаблон, например, для C и C++

```
curl -L -s https://www.gitignore.io/api/c >> .gitignore
curl -L -s https://www.gitignore.io/api/c++ >> .gitignore
```

4. Работа с сервером репозиториев

— Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

```
ssh-keygen -C "Имя Фамилия <work@mail>"
```

— Ключи сохраняться в каталоге `~/.ssh/`.
— Существует несколько доступных серверов репозиториев с возможностью бесплатного размещения данных. Например, https://github.com/.
— Для работы с ним необходимо сначала завести на сайте https://github.com / учётную запись. Затем необходимо загрузить сгенерённый нами ранее открытый ключ.
— Для этого зайти на сайт https://github.com/ под своей учётной записью и перейти в меню *GitHub setting*.
— После этого выбрать в боковом меню *GitHub setting>SSH-ключи* и нажать кнопку *Добавить ключ*. Скопировав из локальной консоли ключ в буфер обмена:

```
cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

— Вставляем ключ в появившееся на сайте поле.
— После этого можно создать на сайте репозиторий, выбрав в меню , дать ему название и сделать общедоступным (публичным).
— Для загрузки репозитория из локального каталога на сервер выполняем следующие команды:

```
git remote add origin
  ssh://git@github.com/<username>/<reponame>.git
git push -u origin master
```

— Далее на локальном компьютере можно выполнять стандартные процедуры для работы с `git` при наличии центрального репозитория.

1.2.1.3 Базовая настройка git

1. Первичная настройка параметров git
   — Зададим имя и email владельца репозитория:

```
git config --global user.name "Name Surname"
git config --global user.email "work@mail"
```

   — Настроим utf-8 в выводе сообщений git:

```
git config --global core.quotepath false
```

   — Настройте верификацию и подписание коммитов git.

- Зададим имя начальной ветки (будем называть её `master`):

```
git config --global init.defaultBranch master
```

2. Учёт переносов строк
   - В разных операционных системах приняты разные символы для перевода строк:
     - Windows: `\r\n` (`CR` и `LF`);
     - Unix: `\n` (`LF`);
     - Mac: `\r` (`CR`).
   - Посмотреть значения переносов строк в репозитории можно командой:

```
git ls-files --eol
```

   1. Параметр `autocrlf`
      - Настройка `core.autocrlf` предназначена для того, чтобы в главном репозитории все переводы строк текстовых файлах были одинаковы.
      - Настройка `core.autocrlf` с параметрами `true` и `input` делает все переводы строк текстовых файлов в главном репозитории одинаковыми.
        - `core.autocrlf true`: конвертация `CRLF`→`LF` при коммите и обратно `LF`→`CRLF` при выгрузке кода из репозитория на файловую систему (обычно используется в Windows).
        - `core.autocrlf input`: конвертация `CRLF`→`LF` только при коммитах (используются в MacOS/Linux).
      - Варианты конвертации

core.autocrlf	false	input	true
git commit	LF → LF	LF → LF	LF → CRLF
	CR → CR	CR → CR	CR → CR
	CRLF → CRLF	CRLF → LF	CRLF → CRLF
git checkout	LF → LF	LF → LF	LF → CRLF
	CR → CR	CR → CR	CR → CR
	CRLF → CRLF	CRLF → CRLF	CRLF → CRLF

      Варианты конвертации для разных значений параметра core.autocrlf
      - Установка параметра:
        - Для Windows

```
git config --global core.autocrlf true
```

        - Для Linux

```
git config --global core.autocrlf input
```

2. Параметр `safecrlf`
   - Настройка `core.safecrlf` предназначена для проверки, является ли окончаний строк обратимым для текущей настройки `core.autocrlf`.
     - `core.safecrlf true`: запрещается необратимое преобразование `lf<->crlf`. Полезно, когда существуют бинарные файлы, похожие на текстовые файлы.
     - `core.safecrlf warn`: печать предупреждения, но коммиты с необратимым переходом принимаются.
   - Установка параметра:

```
git config --global core.safecrlf warn
```

1.2.1.4 Создание ключа ssh

1. Общая информация
   1. Алгоритмы шифрования ssh
      1. Аутентификация
         В SSH поддерживается четыре алгоритма аутентификации по открытым ключам:
         - DSA:
           - размер ключей DSA не может превышать 1024, его следует отключить;
         - RSA:
           - следует создавать ключ большого размера: 4096 бит;
         - ECDSA:
           - ECDSA завязан на технологиях NIST, его следует отключить;
         - Ed25519:
           - используется пока не везде.
      2. Симметричные шифры
         - Из 15 поддерживаемых в SSH алгоритмов симметричного шифрования, безопасными можно считать:
           - `chacha20-poly1305`;
           - `aes*-ctr`;
           - `aes*-gcm`.
         - Шифры `3des-cbc` и `arcfour` потенциально уязвимы в силу использования DES и RC4.
         - Шифр `cast128-cbc` применяет слишком короткий размер блока (64 бит).
      3. Обмен ключами
         - Применяемые в SSH методы обмена ключей DH (Diffie-Hellman) и ECDH (Elliptic Curve Diffie-Hellman) можно считать безопасными.

    - Из 8 поддерживаемых в SSH протоколов обмена ключами вызывают подозрения три, основанные на рекомендациях NIST:
      - `ecdh-sha2-nistp256`;
      - `ecdh-sha2-nistp384`;
      - `ecdh-sha2-nistp521`.
    - Не стоит использовать протоколы, основанные на *SHA1*.
2. Файлы ssh-ключей
    - По умолчанию пользовательские ssh-ключи сохраняются в каталоге `~/.ssh` в домашнем каталоге пользователя.
    - Убедитесь, что у вас ещё нет ключа.
    - Файлы закрытых ключей имеют названия типа `id_<алгоритм>` (например, `id_dsa`, `id_rsa`).
      - По умолчанию закрытые ключи имеют имена:

        ```
        id_dsa
        id_ecdsa
        id_ed25519
        id_rsa
        ```

    - Открытые ключи имеют дополнительные расширения `.pub`.
      - По умолчанию публичные ключи имеют имена:

        ```
        id_dsa.pub
        id_ecdsa.pub
        id_ed25519.pub
        id_rsa.pub
        ```

    - При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав `Enter`.
    - Сменить пароль на ключ можно с помощью команды:

      ```
      ssh-keygen -p
      ```

2. Создание ключа ssh
    - Ключ ssh создаётся командой:

      ```
      ssh-keygen -t <алгоритм>
      ```

    - Создайте ключи:
      - по алгоритму *rsa* с ключём размером 4096 бит:

        ```
        ssh-keygen -t rsa -b 4096
        ```

      - по алгоритму *ed25519*:

        ```
        ssh-keygen -t ed25519
        ```

- При создании ключа команда попросит ввести любую ключевую фразу для более надёжной защиты вашего пароля. Можно пропустить этот этап, нажав `Enter`.
- Сменить пароль на ключ можно с помощью команды:

```
ssh-keygen -p
```

3. Добавление SSH-ключа в учётную запись GitHub
   - Скопируйте созданный SSH-ключ в буфер обмена командой (для X11):

```
xclip -i < ~/.ssh/id_ed25519.pub
```

   - Скопируйте созданный SSH-ключ в буфер обмена командой (для Wayland):

```
wl-copy < ~/.ssh/id_ed25519.pub
```

   - Откройте настройки своего аккаунта на GitHub и перейдем в раздел `SSH and GPC keys`.
   - Нажмите кнопку `ew SSH key`.
   - Добавьте в поле `Title` название этого ключа, например, `ed25519@hostname`.
   - Вставьте из буфера обмена в поле `Key` ключ.
   - Нажмите кнопку `Add SSH key`.

1.2.1.5 Верификация коммитов

1. Проверка коммитов в Git
   - GitHub и GitLab будут показывать значок *Verified* рядом с вашими новыми коммитами.

   1. Режим бдительности (vigilant mode)
      - На GitHub есть настройка vigilant mode.
      - Все неподписанные коммиты будут явно помечены как *Unverified*.
      - Включается это в настройках в разделе *SSH and GPG keys*. Установите метку на *Flag unsigned commits as unverified*.

2. Верификация коммитов с помощью *PGP*
   - Как настроить PGP-подпись коммитов с помощью `gpg`.

   1. Общая информация
      - Коммиты имеют следующие свойства:
        - author (автор) — контрибьютор, выполнивший работу (указывается для справки);

    - committer (коммитер) — пользователь, который закоммитил изменения.
  - Эти свойства можно переопределить при совершении коммита.
  - Авторство коммита можно подделать.
  - В git есть функция подписи коммитов.
  - Для подписывания коммитов используется технология PGP.
  - Подпись коммита позволяет удостовериться в том, кто является коммитером. Авторство не проверяется.
2. Создание ключа
  - Генерируем ключ

```
gpg --full-generate-key
```

  - Из предложенных опций выбираем:
    - тип *RSA and RSA*;
    - размер 4096;
    - выберите срок действия; значение по умолчанию — 0 (срок действия не истекает никогда).
  - GPG запросит личную информацию, которая сохранится в ключе:
    - Имя (не менее 5 символов).
    - Адрес электронной почты.
      - При вводе email убедитесь, что он соответствует адресу, используемому на GitHub.
    - Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.
3. Экспорт ключа
  - Выводим список ключей и копируем отпечаток приватного ключа:

```
gpg --list-secret-keys --keyid-format LONG
```

  - Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа.
  - Формат строки:

```
sec   Алгоритм/Отпечаток_ключа Дата_создания [Флаги] [Годен_до]
      ID_ключа
```

  - Экспортируем ключ в формате ASCII по его отпечатку:

```
gpg --armor --export <PGP Fingerprint>
```

4. Добавление PGP ключа в GitHub
  - Копируем ключ и добавляем его в настройках профиля на GitHub (или GitLab).
  - Скопируйте ваш сгенерированный PGP ключ в буфер обмена:

```
gpg --armor --export <PGP Fingerprint> | xclip -sel clip
```

- Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопку *New GPG key* и вставьте полученный ключ в поле ввода.

5. Подписывание коммитов git
   - Подпись коммитов при работе через терминал:

```
git commit -a -S -m 'your commit message'
```

   - Флаг `-S` означает создание подписанного коммита. При этом может потребоваться ввод кодовой фразы, заданной при генерации GPG-ключа.
6. Настройка автоматических подписей коммитов git
   - Используя введёный email, укажите Git применять его при подписи коммитов:

```
git config --global user.signingkey <PGP Fingerprint>
git config --global commit.gpgsign true
git config --global gpg.program $(which gpg)
```

1.2.2 Рабочий процесс Gitflow

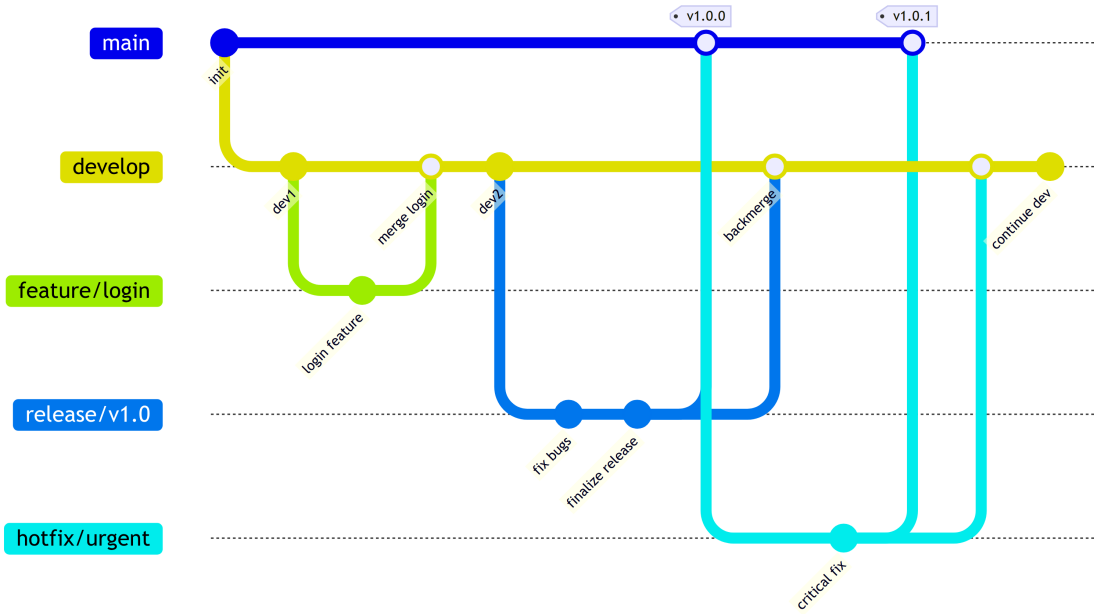
Рабочий процесс *Gitflow Workflow*. Будем описывать его с использованием пакета `git-flow`.

1.2.2.1 Общая информация

- Репозиторий: https://github.com/petervanderdoes/gitflow-avh/
- Описание модели ветвления: https://nvie.com/posts/a-successful-git-branching-model/
- Gitflow Workflow опубликована и популяризована Винсентом Дриссеном из компании vie.
- Gitflow Workflow предполагает выстраивание строгой модели ветвления с учётом выпуска проекта.
- Данная модель отлично подходит для организации рабочего процесса на основе релизов.
- Работа по модели Gitflow включает создание отдельной ветки для исправлений ошибок в рабочей среде.
- Последовательность действий при работе по модели Gitflow:
  - Из ветки `master` создаётся ветка `develop`.
  - Из ветки `develop` создаётся ветка `release`.
  - Из ветки `develop` создаются ветки `feature`.

— Когда работа над веткой `feature` завершена, она сливается с веткой `develop`.
— Когда работа над веткой релиза `release` завершена, она сливается в ветки `develop` и `master`.
— Если в `master` обнаружена проблема, из `master` создаётся ветка `hotfix`.
— Когда работа над веткой исправления `hotfix` завершена, она сливается в ветки `develop` и `master`.

1.2.2.2 Пример



Ключевые моменты:

— Сначала создаются основные ветки (main/master и develop)
— Feature ветки создаются от develop и туда же мержатся
— Release ветки создаются от develop и мержатся в main (с тегом) и обратно в develop
— Hotfix ветки создаются от main и мержатся в main (с тегом) и в develop

1.2.2.3 Установка программного обеспечения

— Для Windows используется пакетный менеджер Chocolatey.
— Git-flow входит в состав пакета git.

```
choco install git
```

- Для MacOS используется пакетный менеджер Homebrew.

```
brew install git-flow
```

- Linux
  - Gentoo

    ```
    emerge dev-vcs/git-flow
    ```

  - Ubuntu

    ```
    apt-get install git-flow
    ```

  - Centos
    - Первоначально нужно установить репозиторий *epel* (https://fedoraproject.org/wiki/EPEL):

      ```
      dnf install epel-release
      ```

    - Затем, собственно, установить git-flow:

      ```
      dnf install gitflow
      ```

  - Fedora
    - Это программное обеспечение удалено из основного репозитория.
    - Можно установить вручную или из коллекции репозиториев *Copr*.
    - Установка из коллекции репозиториев *Copr* (https://copr.fedorainfracloud.org/coprs/elegos/gitflow/):

      ```
      # Enable the copr repository
      dnf copr enable elegos/gitflow
      # Install gitflow
      dnf install gitflow
      ```

    - Установка вручную:

      ```
      cd /tmp
      wget --no-check-certificate -q https://raw.github.com/peterv
       ↪ anderdoes/gitflow/develop/contrib/gitflow-installer.sh
      chmod +x gitflow-installer.sh
      sudo ./gitflow-installer.sh install stable
      ```

1.2.2.4 Поддержка завершения команды

1.2.2.4.1 git-flow-completion

— Репозиторий: https://github.com/bobthecow/git-flow-completion
— Поддержка bash, zsh, fish.

1.2.2.4.2 oh-my-fish/plugin-git-flow

— Репозиторий: https://github.com/oh-my-fish/plugin-git-flow
— Завершение для fish в рамках фреймворка *Oh My Fish*.

1.2.2.5 Процесс работы с Gitflow

1.2.2.5.1 Основные ветки (master) и ветки разработки (develop)

Для фиксации истории проекта в рамках этого процесса вместо одной ветки `master` используются две ветки. В ветке `master` хранится официальная история релиза, а ветка `develop` предназначена для объединения всех функций. Кроме того, для удобства рекомендуется присваивать всем коммитам в ветке `master` номер версии.

При использовании библиотеки расширений `git-flow` нужно инициализировать структуру в существующем репозитории:

```
git flow init
```

Для github параметр `Version tag prefix` следует установить в `v`.

После этого проверьте, на какой ветке Вы находитесь:

```
git branch
```

1.2.2.5.2 Функциональные ветки (feature)

- Под каждую новую функцию должна быть отведена собственная ветка, которую можно отправлять в центральный репозиторий для создания резервной копии или совместной работы команды. Ветки `feature` создаются не на основе `master`, а на основе `develop`. Когда работа над функцией завершается, соответствующая ветка сливается обратно с веткой `develop`. Функции не следует отправлять напрямую в ветку `master`.
- Как правило, ветки `feature` создаются на основе последней ветки `develop`.

1. Создание функциональной ветки
   Создадим новую функциональную ветку:

```
git flow feature start feature_branch
```

   - Далее работаем как обычно.
2. Окончание работы с функциональной веткой
   - По завершении работы над функцией следует объединить ветку `feature_branch` с `develop`:

```
git flow feature finish feature_branch
```

1.2.2.5.3 Ветки выпуска (release)

- Когда в ветке `develop` оказывается достаточно функций для выпуска, из ветки develop создаётся ветка `release`. Создание этой ветки запускает следующий цикл выпуска, и с этого момента новые функции добавить больше нельзя — допускается лишь отладка, создание документации и решение других задач. Когда подготовка релиза завершается, ветка `release` сливается с `master` и ей присваивается номер версии. После нужно выполнить слияние с веткой `develop`, в которой с момента создания ветки релиза могли возникнуть изменения.
- Благодаря тому, что для подготовки выпусков используется специальная ветка, одна команда может дорабатывать текущий выпуск, в то время как другая команда продолжает работу над функциями для следующего.
- Создать новую ветку release можно с помощью следующей команды:

```
git flow release start 1.0.0
```

- Для завершения работы на ветке `release` используются следующие команды:

```
git flow release finish 1.0.0
```

1.2.2.5.4 Ветки исправления (hotfix)

— Ветки поддержки или ветки `hotfix` используются для быстрого внесения исправлений в рабочие релизы. Они создаются от ветки `master`. Это единственная ветка, которая должна быть создана непосредственно от master. Как только исправление завершено, ветку следует объединить с `master` и `develop`. Ветка `master` должна быть помечена обновленным номером версии.
— Наличие специальной ветки для исправления ошибок позволяет команде решать проблемы, не прерывая остальную часть рабочего процесса и не ожидая следующего цикла релиза.
— Ветку `hotfix` можно создать с помощью следующих команд:

```
git flow hotfix start hotfix_branch
```

— По завершении работы ветка `hotfix` объединяется с `master` и `develop`:

```
git flow hotfix finish hotfix_branch
```

1.3 Литературное программирование

Литературное (грамотное) программирование — это подход, приоритизирующий понятность программы для человека, а не её исполнение компьютером. В экосистеме Julia он реализуется через несколько инструментов. Предлагается использовать `Literate.jl`.

1.3.1 Суть литературного программирования

Идея, предложенная Дональдом Кнутом [1; 6; 9], меняет традиционный взгляд: программа создаётся не как последовательность инструкций для машины, а как литературное эссе, объясняющее логику решения задачи.

Литературное программирование подходит не всем и не для всех задач, но может стать мощным инструментом для создания сложных, ясных и поддерживаемых программ.

- Идеи, а не синтаксис.
  - Автор представляет программу как сеть взаимосвязанных идей, располагая фрагменты кода в логическом для человеческого понимания порядке, а не в диктуемом языком программирования.
- Два процесса. Для работы с таким исходником нужны две утилиты:
  - `weave` — «плетёт» (weave) документ для чтения (PDF, HTML).
  - `tangle` — «запутывает» (tangle) из фрагментов исполняемый исходный код.
- Не просто комментарии.
  - Это не просто подробные комментарии (вроде JavaDoc). Это методология проектирования, где документация первична, а код в неё встраивается.

1.3.2 Реализация в Julia: основные инструменты

В Julia есть несколько пакетов для литературного программирования:

Инструмент	Ключевая задача	Основные форматы вывода	Особенность
Literate.jl	Преобразование скриптов `.jl` в документы.	Markdown, Jupyter Notebook (`.ipynb`), скрипт `.jl`.	Простая интеграция с Quatro для документации пакетов.
Weave.jl	Генерация научных отчётов с исполняемым кодом.	LaTeX/PDF, HTML, Jupyter Notebook, MS Word.	Богатые возможности оформления научных публикаций.
Jupyter / IJulia	Интерактивные вычисления и прототипирование.	`.ipynb` (Jupyter Notebook).	Интерактивность, но менее пригоден для автоматизации.

- `Literate.jl` отлично подходит для автоматизации создания документации, а также для написания статей и учебников прямо в `.jl`-файлах.
- `Weave.jl` больше заточен под академические отчёты и публикации, требующие строгого форматирования (LaTeX/PDF).
- `Jupyter` лучше подходит для интерактивного исследования данных, но управление версиями `.ipynb`-файлов и их автоматизация сложнее.

`Literate.jl` — наиболее универсальный и простой для старта инструмент. Его базовый синтаксис интуитивен: - Строки, начинающиеся с #, интерпретируются как Markdown. - Все остальные строки — это код на Julia. - Пакет умеет выполнять код и вставлять результаты (текст, графики) в итоговый документ.

1.3.3 Пример с Literate.jl

Допустим, у вас есть файл `linear_regression.jl`:

```
# # Простая линейная регрессия
# Сгенерируем синтетические данные и подберём модель.
using LinearAlgebra, Plots

# ## Генерация данных
n = 100
x = rand(n)
# Целевая переменная с шумом:
y = 2.5 * x .+ 1.2 .+ 0.3 * randn(n)

# scatter(x, y, label="данные", legend=:topleft)

# ## Решение методом наименьших квадратов
# Создаём матрицу плана с колонкой единиц:
X = [ones(n) x]
β = X \ y  # Находим коэффициенты
println("Найденные коэффициенты: β₀ = $(round(β[1], digits=3)), β₁ =
↪  $(round(β[2], digits=3))")
```

Чтобы преобразовать это в документ, выполните в Julia:

```
using Literate
# Создать Markdown-файл в папке `docs`
Literate.markdown("linear_regression.jl", "docs")
# Или сразу создать Jupyter Notebook
Literate.notebook("linear_regression.jl", "docs")
```

1.4 Julia. Пакет DrWatson

1.4.1 Общая информация

— DrWatson.jl — фреймворк для организации научных проектов в Julia.
— Репозиторий: https://github.com/JuliaDynamics/DrWatson.jl
— Документация: https://juliadynamics.github.io/DrWatson.jl

1.4.2 Установка и инициализация

— Установка в REPL:

```julia
# Установка
using Pkg
Pkg.add("DrWatson")

# Создание проекта
using DrWatson
initialize_project("MyProject"; authors=["Ваше Имя"], git=true)
```

— Создаваемая структура:

```
MyProject/
├── Project.toml
├── Manifest.toml
├── data/
│   ├── sims/       # Результаты симуляций
│   ├── exp_raw/    # Сырые экспериментальные данные
│   └── exp_pro/    Обработанные данные
├── scripts/        # Скрипты для анализа
├── src/            # Исходный код проекта
└── _research/      # Черновики и временные файлы
```

— Активирует окружение проекта, загружает зависимости:

```julia
@quickactivate :MyProject  # Автоматически находит Project.toml
```

1.4.3 Пути

— Стандартные пути:

```
datadir()        # → "data"
plotsdir()       # → "plots"
projectdir()     # → Корень проекта
scriptdir("analysis.jl") → "scripts/analysis.jl"
```

1.4.4 Сохранение результатов

```
using DrWatson: savename, @dict, safesave

params = @dict(α=0.1, β=2.0)  # Автоматическое именование
results = Dict(:data ⇒ rand(10))

## Сохранение с уникальным именем
filename = savename(params, "jld2")  # → "α=0.1_β=2.0.jld2"
safesave(datadir("sims", filename), results)
```

1.4.5 Загрузка данных

```
using DrWatson: load, @strdict

## Поиск файлов по параметрам
filter = @strdict(α=0.1)
file = getlatest(filter, datadir("sims"))  # Последний файл с α=0.1
data = load(file)["data"]
```

1.4.6 Воспроизводимость

```
gitdescribe()  # Возвращает текущий коммит Git → "v0.1.0-5-gf7a2d"
tag!(filename) # Добавляет хеш коммита к имени файла
```

1.4.7 Пакетная обработка

```julia
using BSON

## Генерация параметров
all_params = Dict(:α => [0.1, 0.5], :β => 1:5)

## Запуск симуляций
for params in dict_list(all_params)
    result = simulate(params)
    safesave(datadir("sims", savename(params, "bson")), result)
end
```

1.4.8 Анализ результатов

```julia
using DataFrames

## Сбор всех данных в DataFrame
df = collect_results(datadir("sims"))
select!(df, :α, :β, :result)
```

1.4.9 Интеграция с Literate.jl

```julia
## scripts/generate_report.jl
using Literate

Literate.markdown("scripts/analysis.jl", "docs"; flavor =
 ↪  Literate.QuartoFlavor())
```

1.4.10 Makefile

— Сделаем Makefile:

```makefile
.PHONY: report clean
```

```
report:
    julia --project=. scripts/run_simulations.jl
    julia --project=. scripts/generate_report.jl
    quarto render docs/report.qmd

clean:
    rm -rf data/sims/*
    rm -rf plots/*
```

1.4.11 Практики

- Структура кода
  - Храните многократно используемые функции в `src/` как модуль:

```
# src/MyProject.jl
module MyProject
export simulate

function simulate(params)
  # ...
end
end
```

- Версионирование данных
  - Используйте Git LFS для больших файлов в `data/`
- Шаблоны проектов
  - Создайте шаблон через PkgTemplates.jl:

```
using PkgTemplates
t = Template(;
  dir="~/Projects",
  julia=v"1.9",
  plugins=[DrWatsonPlugin()]
)
t("MyNewProject")
```

- Отладка
  - Используйте `produce_or_load` для кэширования результатов:

```
produce_or_load(params, datadir("cache")) do p
  compute_expensive_operation(p)
```

```
end
```

1.4.12 Пример рабочего процесса

- `initialize_project()` → Создает структуру
- `quickactivate()` → Активирует окружение
- Редактируем скрипты в `scripts/`
- `savename()` + `safesave()` → Сохраняем результаты
- `collect_results()` → Анализируем данные
- Literate.jl → Генерируем отчёт
- Quarto → Рендерим финальный документ

1.5 Язык Markdown

1.5.1 Синтаксис языка Markdown

Общая информация по языку Markdown.

1.5.1.1 Базовый синтаксис Markdown

- Чтобы создать заголовок, используйте знак (#), например:

```
### This is heading 1
#### This is heading 2
##### This is heading 3
###### This is heading 4
```

- Чтобы задать для текста полужирное начертание, заключите его в двойные звездочки:

```
This text is **bold**.
```

This text is **bold**.

— Чтобы задать для текста курсивное начертание, заключите его в одинарные звездочки:

```
This text is *italic*.
```

This text is *italic*.

— Чтобы задать для текста полужирное и курсивное начертание, заключите его в тройные звездочки:

```
This is text is both ***bold and italic***.
```

This is text is both ***bold and italic***.

Блоки цитирования создаются с помощью символа >:

```
> The drought had lasted now for ten million years, and the reign of
 ↪ the terrible lizards had long since ended. Here on the Equator,
 ↪ in the continent which would one day be known as Africa, the
 ↪ battle for existence had reached a new climax of ferocity, and
 ↪ the victor was not yet in sight. In this barren and desiccated
 ↪ land, only the small or the swift or the fierce could flourish,
 ↪ or even hope to survive.
```

Неупорядоченный (маркированный) список можно отформатировать с помощью звездочек или тире:

```
- List item 1
- List item 2
- List item 3
```

Чтобы вложить один список в другой, добавьте отступ для элементов дочернего списка:

```
- List item 1
  - List item A
```

```
  - List item B
- List item 2
```

Упорядоченный список можно отформатировать с помощью соответствующих цифр:

```
1. First instruction
1. Second instruction
1. Third instruction
```

Чтобы вложить один список в другой, добавьте отступ для элементов дочернего списка:

```
1. First instruction
   1. Sub-instruction
   1. Sub-instruction
1. Second instruction
```

Синтаксис Markdown для встроенной ссылки состоит из части `[link text]`, представляющей текст гиперссылки, и части `(file-name.md)` – URL-адреса или имени файла, на который дается ссылка:

```
[link text](file-name.md)
```

Markdown поддерживает как встраивание фрагментов кода в предложение, так и их размещение между предложениями в виде отдельных огражденных блоков. Огражденные блоки кода — это простой способ выделить синтаксис для фрагментов кода. Общий формат огражденных блоков кода:

```
''' language
your code goes in here
'''
```

Верхние и нижние индексы:

H_2O

записывается как

```
H~2~O
```

2^{10}

записывается как

```
2^10^
```

1.5.2 Математика в Markdown

Markdown сам по себе не обрабатывает формулы, но он легко интегрируется с системой вёрстки математических выражений LaTeX.

1.5.2.1 Основные принципы

Формулы записываются с помощью синтаксиса LaTeX внутри специальных разделителей:

- `$ ... $` — для формул внутри текста (инлайновых).
  - Пример: `Скорость вычисляется по формуле $v = \frac{s}{t}$.`
  - Результат: Скорость вычисляется по формуле $v = \frac{s}{t}$.
- `$$ ... $$` — для выключенных формул (блочных), которые отображаются на отдельной строке, по центру.
  - Пример:

    ```
    Второй закон Ньютона:
    $$ \vec{F} = m \vec{a} $$
    ```

  - Результат: Второй закон Ньютона:

$$\vec{F} = m\vec{a}$$

Внутритекстовые формулы делаются аналогично формулам LaTeX. Например, формула $\sin^2(x) + \cos^2(x) = 1$ запишется как

```
$\sin^2 (x) + \cos^2 (x) = 1$
```

Выключные формулы:

$$\sin^2(x) + \cos^2(x) = 1 \tag{1.1}$$

со ссылкой в тексте «Смотри формулу (1.1).» записывается как

```
$$
\sin^2 (x) + \cos^2 (x) = 1
$$ {#eq-sin2-cos2}

Смотри формулу (-@eq-sin2-cos2).
```

1.5.2.2 Основные элементы синтаксиса

Элемент	Синтаксис LaTeX	Пример (внутри $ или $$)	Результат
Индексы	`x^2`, `a_n`	`$E = mc^2$`, `$x_{i+1}$`	$E = mc^2$, x_{i+1}
Дроби	`\frac{a}{b}`	`$\frac{\Delta y}{\Delta x}$`	$\frac{\Delta y}{\Delta x}$
Корни	`\sqrt{x}`, `\sqrt[n]{x}`	`$\sqrt{2}`, `$\sqrt[3]{8}$`	$\sqrt{2}$, $\sqrt[3]{8}$
Греческие буквы	`\alpha`, `\beta`, `\Omega`, `\Gamma`	`$\alpha, \beta, \gamma, \pi$`	$\alpha, \beta, \gamma, \pi$
Скобки (авторазмер)	`\left( ... \right)`	`$\left( \frac{1}{x} \right)$`	$\left(\frac{1}{x}\right)$
Интегралы, суммы	`\int`, `\sum`, `\prod`	`$\int_a^b f(x)dx$`, `$\sum_{i=1}^n i$`	$\int_a^b f(x)dx$, $\sum_{i=1}^n i$
Пределы	`\lim_{x \to a}`	`$\lim_{x \to 0} \frac{\sin x}{x}$`	$\lim_{x\to 0} \frac{\sin x}{x}$
Векторы	`\vec{a}`, `\overline{AB}`	`$\vec{F}$`, `$\overline{v}$`	$\vec{F}$, $\overline{v}$
Матрицы	`\begin{matrix} ... \end{matrix}` (см. ниже)		

1.5.2.3 3. Примеры сложных конструкций

— Матрица (в блочном режиме $$):

```
$$
A = \begin{pmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{pmatrix}
$$
```

$$
A = \begin{pmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{pmatrix}
$$

— Система уравнений:

```
$$
\begin{cases}
x + 2y = 5 \\
3x - y = 1
\end{cases}
$$
```

$$
\begin{cases}
x + 2y = 5 \\
3x - y = 1
\end{cases}
$$

— Определённый интеграл с пределом:

```
$$
I = \int_0^{\pi} \sin^2 x \, dx = \frac{\pi}{2}
$$
```

$$
I = \int_0^{\pi} \sin^2 x \, dx = \frac{\pi}{2}
$$

Формула с производной:

```
$$
f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}
$$
```

$$
f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}
$$

1.5.2.4 Практические советы

- Стиль и читаемость: используйте блочные формулы ($$) для важных, ключевых уравнений. Инлайновые ($) — для упоминаний в тексте.
- Комментарии в коде: В сыром Markdown-файле рядом со сложной формулой можно ставить комментарий:

```
$$ \Delta = b^2 - 4ac $$ <!-- Дискриминант квадратного уравнения -->
```

- Пробелы и выравнивание: Внутри формул можно использовать `\,` (тонкую шпацию), `\:` (среднюю шпацию), `\;` (большую шпацию), `\quad` (квадратную шпацию), `\qquad` (двойную квадратную шпацию) для ручной расстановки отступов, чтобы формулы выглядели аккуратнее.

1.5.3 Язык markdown. Вариант Quarto

1.5.3.1 Общая информация

- Quarto основан на Pandoc и использует его вариацию markdown в качестве базового синтаксиса документа.
- Pandoc markdown — это расширенная и слегка переработанная версия синтаксиса Markdown Джона Грубера.

1.5.3.2 Форматирование текста

- Markdown:

```
*курсив*, **полужирный**, ***полужирный курсив***
```

— *курсив* , **полужирный** , ***полужирный курсив***
— Markdown:

```
верхний индекс^2^ нижний индекс~2~
```

— верхний индекс2 / нижний индекс2
— Markdown:

```
~~зачеркивание~~
```

— ~~зачеркивание~~
— Markdown:

```
`verbatim code`
```

— `verbatim code`

1.5.3.3 Заголовки

— Markdown:

```
# Заголовок 1

## Заголовок 2

### Заголовок 3

#### Заголовок 4

##### Заголовок 5

###### Заголовок 6
```

1.5.3.4 Ссылки и изображения

— Markdown:

```
<https://quarto.org>

[Quarto](https://quarto.org)

![Caption](elephant.png)

[![Caption](elephant.png)](https://quarto.org)

[![Caption](elephant.png "An elephant")](https://quarto.org)

[![](elephant.png){fig-alt="Alt text"}](https://quarto.org)
```

1.5.3.5 Списки

- Списки в Quarto требуют целой пустой строки над списком.
- В противном случае список не будет отображаться в виде списка, а будет отображаться как обычный текст вдоль одной строки.
- Markdown:

```
* unordered list
    + sub-item 1
    + sub-item 2
        - sub-sub-item 1
```

- Вывод:
  - unordered list
    - sub-item 1
    - sub-item 2
      - sub-sub-item 1
- Markdown:

```
*   item 2

    Continued (indent 4 spaces)
```

- Вывод:
  - item 2
    Continued (indent 4 spaces)
- Markdown:

```
1. ordered list
2. item 2
    i) sub-item 1
         A.  sub-sub-item 1
```

— Вывод:

1. ordered list
2. item 2
    1. sub-item 1
        1. sub-sub-item 1

— Markdown:

```
- [ ] Task 1
- [x] Task 2
```

— Вывод:

☐ Task 1
☒ Task 2

1.5.3.6 Сноски

— Поддерживается нумерация и форматирование сносок[1]:

```
Here is a footnote reference,[^1] and another.[^longnote]

[^1]: Here is the footnote.

[^longnote]: Here's one with multiple blocks.

    Subsequent paragraphs are indented to show that they
belong to the previous footnote.

        { some.code }

    The whole paragraph can be indented, or just the first
    line.  In this way, multi-paragraph footnotes work like
    multi-paragraph list items.
```

[1] Вот такая вот сноска.

```
This paragraph won't be part of the note, because it
isn't indented.
```

— Можно писать сноски в виде отдельных абзацев:

```
Here is an inline note.^[Inlines notes are easier to write, since
↪ you don't have to pick an identifier and move down to type the
↪ note.]
```

— Идентификаторы сносок должны быть уникальными в пределах документа.
— В книгах Quarto главы объединяются в один документ для определённых форматов (включая PDF, DOCX и EPUB), поэтому идентификаторы сносок должны быть уникальными для всех глав.

1.5.3.7 Таблицы

— Markdown:

```
| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |
```

— Вывод:

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

1.5.3.8 Исходный код

— Следует использовать ““ ““ для разграничения блоков исходного кода:

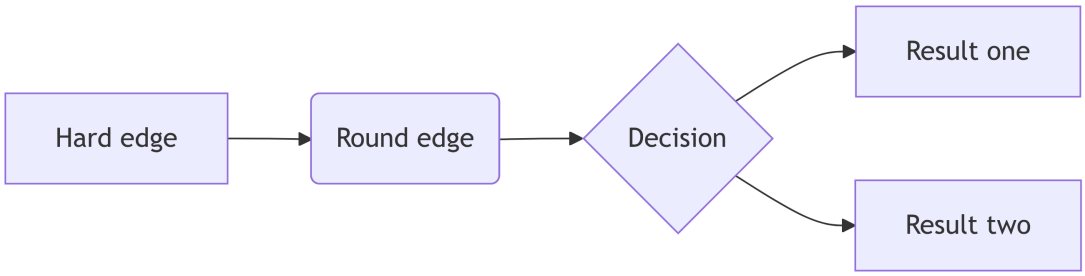
```
```
code
```
```

— Можно указать язык для подсветки синтаксиса блоков кода:

```
```python
1 + 1
```
```

— Если ваш язык не поддерживается, вы можете использовать язык `default`, чтобы получить оформление по умолчанию:

```
```default
code
```
```

— Эквивалентом краткой формы является длинная форма, которая использует язык как класс (например, `.python`) внутри фигурных скобок:

```
```{.python}
1 + 1
```
```

— Длинная форма позволяет добавлять атрибуты к блоку:

```
```{.python filename="run.py"}
code
```
```

1.5.3.9 Аннотации кода

— Аннотации ячеек кода включают в себя:
  — Завершение каждой аннотированной строки комментарием и номером аннотации в угловых скобках (например, `# <2>`). Используйте одно и то же число, чтобы охватить аннотацию на нескольких строках.
  — Добавление упорядоченного списка сразу после ячейки кода, где каждый элемент соответствует номеру аннотации в коде.

```
```r
x <- 1:10  # <1>
y <- x^2   # <2>
z <- x^3   # <2>
```
```

```
1. Create a vector `x`, and then,
2. compute `y` and `z`
```

— Возможные позиции аннотаций для вывода HTML:
  — `below` (по умолчанию): По умолчанию аннотации отображаются под ячейкой кода.
  — `hover` : Аннотации отображаются при наведении мыши на маркер строки.
  — `select` : Аннотации появляются при нажатии на маркер и могут быть удалены повторным нажатием.
— Чтобы изменить положение аннотации по умолчанию, используйте поле метаданных `code-annotations` в заголовке документа:

```
---
code-annotations: hover
---
```

1.5.3.10 Необработанный контент

— Необработанный контент может быть включён напрямую без разбора Quarto, используя атрибут `raw`:

```
```{=html}
<iframe src="https://quarto.org/" width="500"
 ↪  height="400"></iframe>
```
```

— Для вывода PDF используйте необработанный блок LaTeX:

```
```{=latex}
\renewcommand*{\labelitemi}{\textgreater}
```
```

— Можно включить необработанный контент в строку:

```
Here's some raw inline HTML: `<a>html</a>`{=html}
```

1.5.3.11 Уравнения

— Используйте разделители $ для строчной математики и разделители $$ выделенной математики.
— Markdown:

```
inline math: $E = mc^{2}$
```

— Вывод: inline math: $E = mc^2$
— Markdown:

```
display math:

$$E = mc^{2}$$
```

— Вывод: display math:

$$E = mc^2$$

— Если вы хотите определить пользовательские макросы TeX, включите их в разделители $$, заключенные в блок `.hidden`:

```
::: {.hidden}
$$
 \def\RR{{\bf R}}
 \def\bold#1{{\bf #1}}
$$
:::
```

Для MathJax (применяется по умолчанию) можно использовать команды `\def`, `\newcommand`, `\renewcommand`, `\newenvironment`, `\renewenvironment`, `\let`.

1.5.3.12 Диаграммы

— Quarto имеет встроенную поддержку для встраивания диаграммы Mermaid и Graphviz:

```
```{mermaid}
```
```

```
flowchart LR
    A[Hard edge] --> B(Round edge)
    B --> C{Decision}
    C --> D[Result one]
    C --> E[Result two]
```



1.5.3.13 Разрывы страниц

— Шорткод `pagebreak` позволяет вставлять в документ собственный разрыв страницы (например, в LaTeX это будет `\newpage`, в MS Word — собственный разрыв страницы docx, в HTML — директива CSS `page-break-after: always`).
— Собственные разрывы страниц поддерживаются для HTML, LaTeX, Context, MS Word, Open Document, ePub (для других форматов используется символ перевода страницы).

```
page 1

{{< pagebreak >}}

page 2
```

1.5.3.14 Порядок атрибутов

— `div` и `span` в pandoc могут иметь любую комбинацию идентификаторов, классов и атрибутов ключ-значение.
— Чтобы pandoc распознал их, они должны быть предоставлены в определённом порядке: идентификаторы, классы, а затем атрибуты ключ-значение.
— Любой из них может быть опущен, но должен следовать этому порядку, если они предоставлены.

— Например, следующее является допустимым:

```
[This is good]{#id .class key1="val1" key2="val2"}
```

— Следующее pandoc не распознает:

```
[This does *not* work!]{.class key="val" #id}
```

1.5.3.15 Сочетания клавиш

— Шорткод `kbd` можно использовать для описания сочетаний клавиш в документации. В форматах Javascript он попытается определить операционную систему формата и отобразит правильное сочетание клавиш. В форматах печати он выведет информацию о сочетаниях клавиш для всех операционных систем.

```
To print, press {{< kbd Shift-Ctrl-P >}}.
To open an existing new project, press {{< kbd mac=Shift-Command-O
 ↪  win=Shift-Control-O linux=Shift-Ctrl-L >}}.
```

To print, press `Shift-Ctrl-P`. To open an existing new project, press `Shift-Control-O` (windows), `Shift-Ctrl-L` (linux), `Shift-Command-O` (mac).

1.6 Соглашение об именовании Denote

— Приведены принципы именования Denote.
— Для справки приведена базовая схема именования Denote.
— Далее приведена схема именования, применяемая при создании каталогов и имён файлов.

1.6.1 Базовая схема именования

— Каталог с файлами содержит просто список файлов, желательно избегать подкаталоги.
— Шаблон имени файла:

```
DATE==SIGNATURE--TITLE__KEYWORDS.EXTENSION
```

- Поле `DATE` : дата в формате год-месяц-день, за которой следует с большой буквы `T` (для «времени») текущее время в обозначение час-минута-секунда.
  - Примерно так: `20220531T091625`.
  - `DATE` служит уникальным идентификатором каждого файла.
- Поле `SIGNATURE` : строка буквенно-цифровых символов.
  - Не имеют чётко определённой цели.
  - Определяется пользователем.
  - Варианты заполнения:
    - Разделение видео на части : `part1` и `part2`.
    - Приоритет : `a`, `b`, `c`.
    - Для установления последовательные отношения между файлами (например, 1, 1a, 1b, 1b1, 1b2, ...).
- Поле `TITLE` : заголовок заметки, предоставленный пользователем.
  - По умолчанию он автоматически записывается строчными буквами, через дефис.
- Поле `KEYWORDS` состоит из одной или нескольких записей, разделённых значком подчёркивание.
  - Каждое ключевое слово строка, предоставленная пользователем в соответствующем приглашении, которая в целом описывает содержание записи.
  - Каждое из ключевых слов представляет собой одно слово.
- `EXTENSION` : тип файла.
- Примеры:

```
20220610T043241--initial-thoughts-on-the-zettelkasten-
method__notetaking.org
20220610T062201--define-custom-org-hyperlink-type__denote_emacs_package.md
20220610T162327--on-hierarchy-and-taxis__notetaking_philosophy.txt
```

- Различные разделители полей, а именно `--` и `__` дают эффективный способ поиска.
- Можно использовать следующие перестановки частей имён файлов:

```
DATE.EXT
DATE--TITLE.EXT
DATE__KEYWORDS.EXT
DATE==SIGNATURE.EXT
DATE==SIGNATURE--TITLE.EXT
DATE==SIGNATURE--TITLE__KEYWORDS.EXT
DATE==SIGNATURE__KEYWORDS.EXT
```

- По умолчанию имена файлов имеют три поля и два набора разделителей между ними:

```
DATE--TITLE__KEYWORDS.EXTENSION
```

— Когда подпись присутствует:

```
DATE==SIGNATURE--TITLE__KEYWORDS.EXTENSION
```

1.6.2 Концептуальные вариации применения

— Идеология Denote совместима с разными методологиями.

Концепция	Как применяется в Denote
Плоское хранение (Flat-file)	Все файлы можно хранить в одной папке, а для навигации полагаться на уникальные имена и ключевые слова. Это основной и рекомендуемый подход.
Zettelkasten и Folgezettel	Поле `SIGNATURE` подходит для создания сложных цепочек заметок (например, `20231001==1a`, `20231002==1a1`, `20231003==1a2`).
Проектное управление	Ключевые слова можно использовать как теги проектов (например, `__project_alpha`), а в заголовке указывать конкретную задачу.
Мультимедийный архив	Схему можно применять не только к текстовым заметкам, но и к любым другим файлам. Имя файла становится универсальным описательным якорем.

— При создании вариаций стоит придерживаться двух ключевых принципов Denote, которые обеспечивают её эффективность:
  — Уникальность: идентификатор на основе timestamp (`DATE`) должен оставаться неизменным якорем файла.
  — Машиночитаемость: чёткие разделители (`--`, `__`, `==`) нужно сохранять, чтобы система и скрипты могли легко парсить имена.
— Denote есть каркас, а не жёсткий стандарт.
— Вы можете убрать необязательные поля, изменить порядок или использовать подпись для своих целей, сохраняя при этом главные преимущества системы — понятность, уникальность и независимость от инструментов.

1.6.3 Применяемая схема

1.6.3.1 Основная идея

- Использовать каталоги для грубой семантической и хронологической группировки.
- Полное имя файла Denote — универсальный, уникальный идентификатор и описание.
- Идея категорий взята из Johnny Decimal.
- Плоская и гибкая категоризация: файл может принадлежать к нескольким равнозначным категориям без жёсткой иерархии.
- Машиночитаемость: знак `=` есть уникальный разделитель, который легко выделить скриптом или настроить поиск.
- Визуальная ясность: категории чётко сгруппированы в начале имени между `==` и `--`

1.6.3.2 Варианты меток даты для поля `DATE`

- Вы можете выбирать формат даты в зависимости от смысла содержимого, сохраняя сортировку.

Формат даты в имени файла	Когда использовать	Пример имени файла
`2026` (год)	Для документов, тематика которых актуальна целый год: годовые отчёты, планы, резюме.	`2026--годовой-отчет__финансы.pdf`
`2026-1` (полугодие)	Для целей, планов или результатов, сгруппированных по полугодиям.	`2026-1--планы-на-семестр__стратегия.md`
`2026-02-06` (конкретная дата)	Для заметок, событий и документов, привязанных к конкретному дню (встречи, дневниковые записи).	`2026-02-06--встреча-с-командой__проект_митинг.org`

1.6.3.3 Поле SIGNATURE

- В поле `SIGNATURE` (после `==`) указывается набор равноправных категорий, разделённых знаком `=`.

— Это создаёт плоский, но структурированный список категорий прямо в имени файла.

1.6.3.4 Общий формат

```
ДАТА==КАТЕГОРИЯ1=КАТЕГОРИЯ2=КАТЕГОРИЯ3--ЗАГОЛОВОК__КЛЮЧЕВЫЕ_СЛОВА
```

— Примеры:
  — `2026-02-10==проект=вебсайт=дизайн--обзор-макетов__ui_клиент_встреча.fig`
  — `2026-1==сфера=здоровье=питание--полезные-рецепты__еда_идеи.md`
  — `2025-12-24==архив=путешествия=япония=фото--поездка-в-токио__отпуск.jpg`

1.7 Настройка git

1.7.1 Установка git

— Установим *git*:

```
sudo dnf -y install git
```

1.7.2 Установка gh

— Установим интерфейс командной строки к github:

```
sudo dnf -y install gh
```

1.7.3 Базовая настройка git

— Зададим имя и email владельца репозитория:

```
git config --global user.name "Name Surname"
git config --global user.email "work@mail"
```

— Настроим utf-8 в выводе сообщений git:

```
git config --global core.quotepath false
```

— Настройте верификацию и подписание коммитов git.
— Зададим имя начальной ветки (будем называть её `master`):

```
git config --global init.defaultBranch master
```

— Параметр `autocrlf`:

```
git config --global core.autocrlf input
```

— Параметр `safecrlf`:

```
git config --global core.safecrlf warn
```

1.7.4 Настройка gitverse

— Создайте учётную запись на https://gitverse.ru.
— Заполните основные данные на https://gitverse.ru.

1.7.5 Настройка github

— Создайте учётную запись на https://github.com.
— Заполните основные данные на https://github.com.

1.7.6 Настройка gh

— Для начала необходимо авторизоваться

```
gh auth login
```

— Утилита задаст несколько наводящих вопросов.
— Авторизоваться можно через броузер.

1.7.7 Создайте ключ SSH

— Создадим ключ для доступа к github:

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519-git -C
 ↪  "your_email@example.com"
```

— Добавляем ключ в агент:

```
ssh-add ~/.ssh/id_ed25519-git
```

— Добавьте ключ в учётную запись gitverse через web-интерфейс.
— Добавьте ключ в учётную запись github:

```
gh ssh-key add ~/.ssh/id_ed25519-git.pub --title
 ↪  your_email@example.com
gh ssh-key add ~/.ssh/id_ed25519-git.pub --title
 ↪  your_email@example.com --type signing
```

1.7.8 Создайте ключи *pgp*

— Генерируем ключ

```
gpg --full-generate-key
```

— Из предложенных опций выбираем:
  — тип *RSA and RSA*;
  — размер 4096;
  — выберите срок действия; значение по умолчанию — 0 (срок действия не истекает никогда).
— GPG запросит личную информацию, которая сохранится в ключе:
  — Имя (не менее 5 символов).
  — Адрес электронной почты.
    — При вводе email убедитесь, что он соответствует адресу, используемому на GtVerse и GitHub.
  — Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.

1.7.9 Добавление PGP ключа на хостинг

— Выводим список ключей и копируем отпечаток приватного ключа:

```
gpg --list-secret-keys --keyid-format LONG
```

— Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа.
— Формат строки:

```
sec   Алгоритм/Отпечаток_ключа Дата_создания [Флаги] [Годен_до]
      ID_ключа
```

— Скопируйте ваш сгенерированный PGP ключ в буфер обмена:

```
gpg --armor --export <PGP Fingerprint> | wl-copy
```

— Перейдите в настройки Gitverse (https://gitverse.ru/settings/keys), нажмите на кнопку *Добавить GPG* и вставьте полученный ключ в поле ввода.
— Перейдите в настройки GitHub (https://github.com/settings/keys), нажмите на кнопку *New GPG key* и вставьте полученный ключ в поле ввода.

1.7.10 Настройка автоматических подписей коммитов git

— Используя введёный email, укажите Git применять его при подписи коммитов:

```
git config --global user.signingkey <PGP Fingerprint>
git config --global commit.gpgsign true
git config --global gpg.program $(which gpg)
```

1.8 Рабочее пространство лабораторной работы

При выполнении лабораторной работы следует придерживаться структуры рабочего пространства.

1.8.1 Основные идеи

- Стандартные соглашения об именах
- Стандартное соглашение для путей к файлам
- Стандартная настройка курса внутри шаблона курса

1.8.2 Используемые стандарты и программные продукты

- Стандарт Git Flow (раздел 1.2.2).
- Стандарт Семантическое версионирование (раздел 1.1.1).
- Стандарт Общепринятые коммиты (раздел 1.1.2).

1.8.3 Дополнительное программное обеспечение

1.8.3.1 Средства разработки

1. Fedora
   - Установите средства разработки:

     ```
     sudo dnf -y group install development-tools
     ```

1.8.3.2 Quarto

1. Установка
   1. Windows
      - Chocolatey:

        ```
        choco install quarto
        ```
   2. Linux
      2. Gentoo
         - Gentoo, репозиторий karma:

           ```
           emerge quarto
           ```
      3. Arch
         - Arch linux:

```
pacman -S quarto-cli-bin
```

- Manjaro linux:

```
pamac install quarto-cli-bin
```

4. Fedora
   - Установка из CORP:

```
sudo dnf -y copr enable iucar/rstudio
sudo dnf -y install quarto
sudo dnf -y install libxcrypt-compat
```

1.8.3.3 Общепринятые коммиты

1. Установка Node.js
   - На Node.js базируется программное обеспечение для семантического версионирования и общепринятых коммитов.
   - Для управления пакетами лучше использовать `pnpm`, но можно и `yarn`.
   - Gentoo
     - Node.js:

```
emerge nodejs
emerge yarn
```

     - pnpm ставим из оверлея `guru`:

```
eselect repository enable guru
emerge --sync guru
emerge pnpm-bin
```

   - Ubuntu

```
apt-get install nodejs
apt-get install yarn
apt-get install pnpm
```

   - Fedora

```
sudo dnf -y install nodejs
sudo dnf -y install yarn pnpm
```

   - Windows
     - Chocolatey:

```
choco install nodejs
choco install yarn
choco install pnpm
```

- MacOS

```
brew install node
```

2. Настройка Node.js
   Для работы с Node.js добавим каталог с исполняемыми файлами, устанавливаемыми пакетным менеджером, в переменную `PATH`.
   - Linux
     - `pnpm`
       - Запустите:

```
pnpm setup
```

       - Перелогиньтесь, или выполните:

```
source ~/.bashrc
```

     - `yarn`
       - В файле `~/.bashrc` добавьте к переменной `PATH`:

```
PATH=~/.yarn/bin:$PATH
```

3. Установка git-flow
   - Linux
     - Gentoo

```
emerge dev-vcs/git-flow
```

     - Ubuntu

```
apt-get install git-flow
```

     - Fedora
       - Устанавливается из COPR:

```
sudo dnf -y copr enable elegos/gitflow
sudo dnf install gitflow
```

   - Windows Git-flow входит в состав пакета git.

```
choco install git
```

   - MacOS

```
brew install git-flow
```

4. Общепринятые коммиты
   1. commitizen
      - Данная программа используется для помощи в форматировании коммитов.
        - pnpm:

          ```
          pnpm add -g commitizen
          ```

        - yarn:

          ```
          yarn global add commitizen
          ```

      - При этом устанавливается скрипт `git-cz`, который мы и будем использовать для коммитов.
   2. cz-customizable
      - Данная программа позволяет более глубоко настроить форматирование коммитов.
        - pnpm:

          ```
          pnpm add -g cz-customizable
          ```

   3. standard-version
      - Данная программа автоматизирует изменение номера версии.
        - pnpm:

          ```
          pnpm add -g standard-version
          ```

        - yarn:

          ```
          yarn global add standard-version
          ```

1.8.4 Общие правила

- Для именования каталогов и файлов будем использовать соглашение Denote (см. 1.6).
- Рабочее пространство по предмету располагается в следующей иерархии:

```
~/work/study/
└── <учебное полугодие>/
    └── <учебное полугодие>==study-<код предмета>/
```

— Например, для 2025-2026 учебного года (второй семестр) и предмета «Имитационное моделирование» (код предмета `simulation-modeling`) структура каталогов примет следующий вид:

```
~/work/study/
└── 2026-1/
    └── 2026-1==study-simulation-modeling/
```

— Название проекта на хостинге git имеет вид (из-за ограничений на символы):

```
<учебный год>-study-<код предмета>
```

— Например, для 2025-2026 учебного года и предмета «Имитационное моделирование» (код предмета `simulation-modeling`) название проекта примет следующий вид:

```
2026-1-study-simulation-modeling
```

— Каталог для лабораторных работ имеет вид `labs`.
— Каталоги для лабораторных работ имеют вид `lab<номер>`, например: `lab01`, `lab02` и т.д.
— Каталог для групповых проектов имеет вид `group-project`.
— Каталог для персональных проектов имеет вид `personal-project`.
— Каталог для внешнего курса имеет вид `external-course`.
— Если проектов несколько, то они нумеруются подобно лабораторным работам.
— Этапы проекта обозначаются как `stage<номер>`.

1.8.5 Шаблон для рабочего пространства

— Репозиторий:
  — https://gitverse.ru/dharma/course-directory-student-template
  — https://github.com/yamadharma/course-directory-student-template.

1.8.5.1 Сознание репозитория курса на основе шаблона

— Репозиторий на основе шаблона можно создать либо вручную, через web-интерфейс, либо с помощью утилит `gh` для github.

1.8.5.1.1 Создание вручную

- Сделать свой репозиторий на основе шаблона можно и вручную: https://docs.github.com/ru/repositories/creating-and-managing-repositories/creating-a-repository-from-a-template.
- Авторизуйтесь на https://gitverse.ru.
- Перейдите в репозиторий https://gitverse.ru/dharma/course-directory-student-template.
- Найдите кнопку (ссылку) «Использовать как шаблон» https://gitverse.ru/new?template_id=195300.
- После нажатия откроется форма для создания нового репозитория.
- Укажите название для вашего нового проекта: `2026-1--study--simulation-modeling`
- Укажите описание для вашего проекта.
- Выберите уровень доступа (публичный).
- Нажмите кнопку для подтверждения создания.
- Клонируйте репозиторий:

```
mkdir -p ~/work/study/2026-1/2026-1==study-simulation-modeling
cd ~/work/study/2026-1/2026-1==study-simulation-modeling
git clone ssh://git@gitverse.ru:2222/<owner>/2026-1-study-simulation
↪ -modeling.git
```

1.8.5.2 Структура шаблона

- Посмотреть доступные цели `make`:

  ```
  make help
  ```

- Посмотреть список доступных курсов:

  ```
  make list
  ```

- При создании структуры название курса берётся из следующих мест:
  - название курса находится в файле `COURSE`;
  - каталог курса называется как аббревиатура курса.

1.8.5.3 Настройка каталога курса

— Перейдите в каталог курса:

```
cd ~/work/study/2026-1/2026-1==study-simulation-modeling/2026-1-st
↪ udy-simulation-modeling
```

— Инициализируйте курс:

```
echo simulation-modeling > COURSE
make prepare
```

— Отправьте файлы на сервер:

```
git add .
git commit -am 'feat(main): make course structure'
git push
```

1.8.5.4 Скопируйте на github

— Создайте на github репозиторий `2026-1-study-simulation-modeling`
— Запуште репозиторий на github:

```
git remote add github
↪ git@github.com:<owner>/2026-1-study-simulation-modeling.git
git push github master
```

1.8.5.5 Использование git flow

— Будем использовать для работы git flow (см. раздел 1.2.2).

1. Конфигурация git-flow
   — Инициализируем git-flow

   ```
   git flow init
   ```

   Префикс для ярлыков установим в `v`.

— Проверьте, что Вы на ветке `develop`:

```
git branch
```

— Загрузите весь репозиторий в хранилище:

```
git push -u --all
```

— Создадим релиз с версией 1.0.0

```
git flow release start 1.0.0
```

— Создадим журнал изменений

```
standard-changelog --first-release
```

— Добавим журнал изменений в индекс

```
git add CHANGELOG.md
git commit -am 'chore(site): add changelog'
```

— Зальём релизную ветку в основную ветку

```
git flow release finish 1.0.0
```

— Отправим данные на github

```
git push --all
git push --tags
```

— Скопируем CHANGELOG.md в каталог `release`:

```
mkdir -p ../release
cp CHANGELOG.md ../release
```

— Создадим релиз.
— На gitverse сделайте это вручную.
— Для github будем использовать утилиты работы с github:

```
gh release create v1.0.0 -F ../release/CHANGELOG.md
```

1.8.5.6 Rutube

Рабочее пространство для лабораторной работы

Смотреть видео на Rutube

1.8.5.7 VKVideo

Рабочее пространство для лабораторной работы

Смотреть видео на VK

1.9 Создание проекта DrWatson для лабораторных

— Каталог DrWatson делается в рамках лабораторной работы:

```
cd labs/lab01
```

1.9.1 Создание каталога проекта DrWatson

1.9.1.1 Вариант A: Через REPL Julia

1. Откройте терминал и запустите Julia:

   ```
   julia
   ```

2. В REPL выполните:

```
using Pkg
Pkg.add("DrWatson")
using DrWatson
initialize_project("project"; authors="Ваше Имя", git=false)
```

3. Перейдите в созданный каталог:

```
cd("project")
```

1.9.1.2 Вариант Б: Через скрипт

Создайте файл `setup_project.jl`:

```
##!/usr/bin/env julia
## setup_project.jl

using Pkg
Pkg.add("DrWatson")
using DrWatson

## Создание проекта
project_name = "project"
initialize_project(project_name; authors="Ваше Имя", git=false)

println("✅ Проект создан: ", project_name)
println("📂 Перейдите в директорию: cd ", project_name)
```

Запустите:

```
julia setup_project.jl
cd project
```

1.9.2 Добавление необходимых пакетов

1.9.2.1 Вариант A: Установка скриптом

Создайте файл `add_packages.jl` в корне проекта:

```julia
##!/usr/bin/env julia
## add_packages.jl

using Pkg
Pkg.activate(".")  # Активируем текущий проект

## ОСНОВНЫЕ ПАКЕТЫ ДЛЯ РАБОТЫ
packages = [
    "DrWatson",           # Организация проекта
    "DifferentialEquations",  # Решение ОДУ
    "Plots",              # Визуализация
    "DataFrames",         # Таблицы данных
    "CSV",                # Работа с CSV
    "JLD2",               # Сохранение данных
    "Literate",           # Literate programming
    "IJulia",             # Jupyter notebook
    "BenchmarkTools",     # Бенчмаркинг
    "Quarto"              # Создание отчетов
]

println("Установка базовых пакетов...")
Pkg.add(packages)

println("\n✅ Все пакеты установлены!")
println("Для проверки: using DrWatson, DifferentialEquations, Plots")
```

Запустите:

```
julia add_packages.jl
```

1.9.2.2 Вариант Б: Поэтапная установка в REPL

1. Активируйте проект:

```
using Pkg
Pkg.activate(".")
```

2. Установите основные пакеты:

```
Pkg.add("DrWatson")
Pkg.add("DifferentialEquations")
Pkg.add("Plots")
Pkg.add("DataFrames")
Pkg.add("Literate")
Pkg.add("CSV")
Pkg.add("JLD2")
Pkg.add("IJulia")
Pkg.add("BenchmarkTools")
Pkg.add("Quarto")
```

1.9.3 Проверка установки

Создайте тестовый скрипт `scripts/test_setup.jl`:

```
##!/usr/bin/env julia
## test_setup.jl

using DrWatson
@quickactivate "project"

println("✅ Проект активирован: ", projectdir())

## Проверка пакетов
packages = [
    "DrWatson",            # Организация проекта
    "DifferentialEquations",  # Решение ОДУ
    "Plots",               # Визуализация
    "DataFrames",          # Таблицы данных
    "CSV",                 # Работа с CSV
    "JLD2",                # Сохранение данных
```

```
    "Literate",           # Literate programming
    "IJulia",             # Jupyter notebook
    "BenchmarkTools",     # Бенчмаркинг
    "Quarto"              # Создание отчетов
]

println("\nПроверка пакетов:")
for pkg in packages
    try
        eval(Meta.parse("using $pkg"))
        println("  ✓ $pkg")
    catch e
        println("  × $pkg: Ошибка загрузки")
    end
end

## Проверка путей
println("\nСтруктура проекта:")
println("  Корень:        ", projectdir())
println("  Данные:        ", datadir())
println("  Скрипты:       ", srcdir())
println("  Графики:       ", plotsdir())
```

Запустите:

```
julia --project=. scripts/test_setup.jl
```

1.9.4 Итоговая структура проекта

```
lab01
├── presentation
├── project/
│   ├── Project.toml          # Зависимости проекта
│   ├── Manifest.toml         # Точные версии
│   ├── src/                  # Исходный код пакета
│   ├── data/                 # Данные
│   ├── scripts/              # Скрипты
│   ├── plots/                # Графики
```

```
│   ├── papers/              # Статьи
│   ├── docs/                # Документация
│   └── test/               # Тесты
└── report
```

1.10 Модель экспоненциального роста

В качестве примера выполнения работы используем модель экспоненциального роста.

1.10.1 Описание модели

Экспоненциальный рост — это процесс увеличения величины, при котором скорость роста в каждый момент времени пропорциональна текущему значению этой величины. Чем больше система, тем быстрее она растет.

Простая аналогия: проценты на проценты в банке или снежный ком, который, катясь, увеличивается тем быстрее, чем больше его размер.

1.10.1.1 Дифференциальное уравнение

$$\frac{du}{dt} = \alpha u.$$

Здесь:

- u — текущее значение растущей величины (численность популяции, капитал, количество зараженных и т.д.).
- t — время.
- du/dt — скорость роста (производная по времени).
- α — константа роста (мальтузианский параметр). Это ключевой параметр модели. Если $\alpha > 0$ — рост, если $\alpha < 0$ — экспоненциальное затухание.

Смысл уравнения: Скорость изменения (du/dt) прямо зависит от текущего размера (u).

1.10.1.2 Решение

Решение этого дифференциального уравнения дает известную формулу:

$$u(t) = u_0 e^{\alpha t}.$$

1.10.1.3 Ключевые характеристики

- Удвоение за постоянное время.
  - Время, за которое величина удваивается (время удвоения T_2), постоянно и не зависит от текущего размера.
  - $T_2 = ln(2)/u \approx 0.693/u$.

1.10.1.4 Где применяется модель

- Биология: Рост популяции бактерий в неограниченной среде с избытком ресурсов.
- Финансы: Сложный процент по вкладам.
- Эпидемиология: Начальная фаза распространения инфекции, когда иммунитета в популяции почти нет.
- Демография: Рост населения Земли в определённые исторические периоды.
- Физика: Цепная ядерная реакция (деление ядер).
- Информационные технологии: Закон Мура (рост вычислительной мощности), рост трафика в сетях.

1.10.1.5 Ограничения

Экспоненциальный рост — идеализированная модель. В реальности он не может продолжаться бесконечно из-за ограниченности ресурсов (пищи, пространства, капитала и т.д.). После некоторого времени рост обычно замедляется и переходит в логистический рост (S-образная кривая).

1.10.2 Реализация модели

1.10.2.1 Скрипт

— Создайте следующий скрипт (scripts/01_exponential_growth.jl):

```
using DrWatson
@quickactivate "project"

using DifferentialEquations
using Plots
using DataFrames

function exponential_growth!(du, u, p, t)
    α = p
    du[1] = α * u[1]
end

u0 = [1.0]          # начальная популяция
α = 0.3            # скорость роста
tspan = (0.0, 10.0) # временной интервал

prob = ODEProblem(exponential_growth!, u0, tspan, α)
sol = solve(prob, Tsit5(), saveat=0.1)

plot(sol, label="u(t)", xlabel="Время t", ylabel="Популяция u",
     title="Экспоненциальный рост (α = $α)", lw=2, legend=:topleft)

savefig(plotsdir("exponential_growth_α=$α.png"))

df = DataFrame(t=sol.t, u=first.(sol.u))
println("Первые 5 строк результатов:")
println(first(df, 5))

u_final = last(sol.u)[1]
doubling_time = log(2) / α
println("\nАналитическое время удвоения: ", round(doubling_time;
 ↪  digits=2))
```

1.10.2.2 Выполнение

— Создайте скрипт `scripts/01_exponential_growth.lj`.
— Выполните скрипт:

```
julia --project=. scripts/01_exponential_growth.jl
```

— Посмотрите результирующие графики в каталоге `plots/`.

1.10.3 Литературная реализация модели

— Добавьте в файл описание в духе литературного программирования (см. раздел 1.3).

1.10.3.1 Программный код

— Измените файл `scripts/01_exponential_growth.lj`:

```
# # Экспоненциальный рост
# **Цель:** Исследовать решение уравнения du/dt = αu.
#
# ## Инициализация проекта и загрузка пакетов
using DrWatson
@quickactivate "project"

using DifferentialEquations
using Plots
using DataFrames
using JLD2

script_name = splitext(basename(PROGRAM_FILE))[1]
mkpath(plotsdir(script_name))
mkpath(datadir(script_name))

# ## Определение модели
# Уравнение экспоненциального роста:
# du/dt = αu,   u(0) = u_0
```

```
function exponential_growth!(du, u, p, t)
    α = p
    du[1] = α * u[1]
end

# ## Первый запуск с параметрами по умолчанию
# Зададим начальные параметры:
u0 = [1.0]          # начальная популяция
α = 0.3            # скорость роста
tspan = (0.0, 10.0) # временной интервал

prob = ODEProblem(exponential_growth!, u0, tspan, α)
sol = solve(prob, Tsit5(), saveat=0.1)

# ## Визуализация результатов
# Построим график решения:
plot(sol, label="u(t)", xlabel="Время t", ylabel="Популяция u",
     title="Экспоненциальный рост (α = $α)", lw=2, legend=:topleft)

# Сохраним график в папку plots
savefig(plotsdir(script_name, "exponential_growth_α=$α.png"))

# ## Анализ результатов
# Создадим таблицу с данными:
df = DataFrame(t=sol.t, u=first.(sol.u))
println("Первые 5 строк результатов:")
println(first(df, 5))

# Вычислим удвоение популяции:
u_final = last(sol.u)[1]
doubling_time = log(2) / α
println("\nАналитическое время удвоения: ", round(doubling_time;
 ↪  digits=2))

# ## Сохранение всех результатов
@save datadir(script_name, "all_results.jld2") df
```

1.10.3.2 Выполнение

— Выполним программный код:

```
julia --project=. scripts/01_exponential_growth.jl
```

— Проверьте полученные данные и графики.

1.10.3.3 Создание производных форматов

— Создадим скрипт для генерации производных форматов (`scripts/tangle.jl`):

```julia
#!/usr/bin/env julia
# tangle.jl - Генератор отчетов из Literate-скриптов
# Использование: julia tangle.jl <путь_к_скрипту>

using DrWatson
@quickactivate  # Активирует текущий проект DrWatson

using Literate

function main()
    if length(ARGS) == 0
        println("""
        Использование: julia tangle.jl <путь_к_скрипту>

        Примеры:
          julia tangle.jl scripts/lab1.jl
        """)
        return
    end

    script_path = ARGS[1]

    if !isfile(script_path)
        error("Файл не найден: $script_path")
    end

    # Пути и имена
    script_dir = dirname(script_path)
    script_name = splitext(basename(script_path))[1]

    println("Генерация из: $script_path")
```

```
    # Чистый скрипт (без комментариев)
    scripts_dir = scriptsdir(script_name)
    Literate.script(script_path, scripts_dir; credit=false)
    println("  ✓ Чистый скрипт: $(scripts_dir)/$(script_name).jl")

    # Quarto-документ
    quarto_dir = projectdir("markdown", script_name);
    Literate.markdown(script_path, quarto_dir;
                      flavor=Literate.QuartoFlavor(),
                      name=script_name, credit=false)

    println("  ✓ Quarto: $(quarto_dir)/$(script_name).qmd")

    # Jupyter notebook
    notebooks_dir = projectdir("notebooks", script_name)
    Literate.notebook(script_path, notebooks_dir, name=script_name;
     ↪  execute=false, credit=false)
    println("  ✓ Notebook: $(notebooks_dir)/$(script_name).ipynb")

    println("\nГотово! Все файлы созданы.")
end

# Запуск
if abspath(PROGRAM_FILE) == @__FILE__
    main()
end
```

— Выполните программу:

```
julia --project=. scripts/02_exponential_growth.jl
```

— Создайте производные форматы:

```
julia --project=. scripts/tangle.jl scripts/01_exponential_growth.jl
```

1.10.3.4 Jupyter notebook

Выполните Jupyter-ноутбук `notebooks/01_exponential_growth/01_exponential_growth.ipynb`.

1.10.3.5 Документирование в отчёте

— В каталоге отчёта в файл `_quarto.yml` включите поддержку кода `julia`. Добавьте после описания проекта:

```
engine: julia
julia:
  exeflags: ["--project=../project"]
```

— В преамбуле `preamble.tex` подключите пакет `juliamono`.
— В файле отчёта после описания выполнения лабораторной работы подключите файл описания программы:

```
{{< include ../project/markdown/01_exponential_growth/01_exponential
↪ _growth.qmd >}}
```

— Скомпилируйте отчёт.

1.10.4 Реализация модели с параметрами

— Исследование не ограничивается одним значением параметров.
— Изменим программу так, чтобы она принимала набор параметров.

1.10.4.1 Программный код

— Перенесём программу в файл `scripts/02_exponential_growth.lj`:

```
# # Параметрическое исследование экспоненциального роста
#
# ## Активация проекта и загрузка пакетов
#
# **ИЗМЕНЕНИЕ:** Добавлен DrWatson для управления проектом и
↪ параметрами

using DrWatson
```

```
@quickactivate "project"  # Активация проекта DrWatson

using DifferentialEquations
using DataFrames
using Plots
using JLD2
using BenchmarkTools

# Установка каталогов
script_name = splitext(basename(PROGRAM_FILE))[1]
mkpath(plotsdir(script_name))
mkpath(datadir(script_name))

# ## Определение модели
# Модель: du/dt = α · u

function exponential_growth!(du, u, p, t)
    α = p.α  # **ИЗМЕНЕНИЕ:** Параметры теперь передаются как
     ↪  именованный кортеж
    du[1] = α * u[1]
end

# ## Определение параметров в Dict
# **ОСНОВНОЕ ИЗМЕНЕНИЕ:** Все параметры собраны в Dict для
 ↪  систематизации

# Базовый набор параметров (один эксперимент)
base_params = Dict(
    :u0 => [1.0],           # начальная популяция
    :α => 0.3,              # скорость роста
    :tspan => (0.0, 10.0),  # интервал времени
    :solver => Tsit5(),     # метод решения
    :saveat => 0.1,         # шаг сохранения результатов
    :experiment_name => "base_experiment"
)

println("Базовые параметры эксперимента:")
for (key, value) in base_params
    println("  $key = $value")
end

# ## Функция-обертка для запуска одного эксперимента
# **ИСПРАВЛЕНИЕ:** Возвращаем Dict со строковыми ключами
```

```
function run_single_experiment(params::Dict)
    @unpack u0, α, tspan, solver, saveat = params
    prob = ODEProblem(exponential_growth!, u0, tspan, (α=α,)) #
     ↪  Создаем и решаем задачу
    sol = solve(prob, solver; saveat=saveat)
    final_population = last(sol.u)[1] # Анализ результатов
    doubling_time = log(2) / α
    return Dict(
        "solution" => sol,
        "time_points" => sol.t,
        "population_values" => first.(sol.u),
        "final_population" => final_population,
        "doubling_time" => doubling_time,
        "parameters" => params  # Сохраняем исходные параметры
    ) # Используем строки как ключи для совместимости с DrWatson
end

# ## Запуск базового эксперимента
# **ИЗМЕНЕНИЕ:** Используем produce_or_load для автоматического
 ↪  кэширования

data, path = produce_or_load(
    datadir(script_name, "single"),      # Папка для сохранения
    base_params,            # Параметры эксперимента
    run_single_experiment,  # Функция для выполнения
    prefix = "exp_growth",  # Префикс имени файла
    tag = false,            # Не добавлять git-тег
    verbose = true
)

println("\nРезультаты базового эксперимента:")
println("  Финальная популяция: ", data["final_population"])
println("  Время удвоения: ", round(data["doubling_time"]; digits=2))
println("  Файл результатов: ", path)

# ## Визуализация базового эксперимента

p1 = plot(data["time_points"], data["population_values"],
    label="α = $(base_params[:α])",
    xlabel="Время, t",
    ylabel="Популяция, u(t)",
    title="Экспоненциальный рост (базовый эксперимент)",
    lw=2,
    legend=:topleft,
```

```
    grid=true
)

# Сохраним график в папку plots
savefig(plotsdir(script_name, "single_experiment.png"))

# ## Параметрическое сканирование
# **НОВАЯ СЕКЦИЯ:** Исследование влияния параметра α

# Сетка параметров для сканирования
param_grid = Dict(
    :u0 => [[1.0]],           # фиксируем начальное условие
    :α => [0.1, 0.3, 0.5, 0.8, 1.0],  # исследуемые значения скорости
     ↪  роста
    :tspan => [(0.0, 10.0)],  # фиксируем интервал времени
    :solver => [Tsit5()],     # фиксируем метод решения
    :saveat => [0.1],         # фиксируем шаг сохранения
    :experiment_name => ["parametric_scan"]
)

# Генерация всех комбинаций параметров
all_params = dict_list(param_grid)

println("\n" * "="^60)
println("ПАРАМЕТРИЧЕСКОЕ СКАНИРОВАНИЕ")
println("Всего комбинаций параметров: ", length(all_params))
println("Исследуемые значения α: ", param_grid[:α])
println("="^60)

# ## Запуск всех экспериментов и собор результатов
# **НОВАЯ СЕКЦИЯ:** Автоматический запуск и сохранение всех вариантов

all_results = []
all_dfs = []

for (i, params) in enumerate(all_params)
    println("Прогресс: $i/$(length(all_params)) | α = $(params[:α])")

    data, path = produce_or_load(
        datadir(script_name, "parametric_scan"),  # Данные
        params,                       # Текущий набор параметров
        run_single_experiment,        # Функция для выполнения
        prefix = "scan",              # Префикс имени файла
        tag = false,
```

```
        verbose = false              # Не выводить подробности для
         ↪  каждого запуска
    ) # Автоматическое сохранение/загрузка каждого эксперимента

    result_summary = merge(
        params,
        Dict(
            :final_population => data["final_population"],
            :doubling_time => data["doubling_time"],
            :filepath => path  # Путь к сохраненным данным
        )
    ) # Сохраняем сводные результаты (используем символы для
     ↪  параметров, но данные из data - строки)

    push!(all_results, result_summary)

    df = DataFrame(
        t = data["time_points"],
        u = data["population_values"],
        α = fill(params[:α], length(data["time_points"]))
    ) # Сохраняем полные данные для визуализации
    push!(all_dfs, df)
end

# ## Анализ и визуализация результатов сканирования
# **НОВАЯ СЕКЦИЯ:** Сравнительный анализ всех экспериментов

# Сводная таблица результатов
results_df = DataFrame(all_results)
println("\nСводная таблица результатов:")
println(results_df[!, [:α, :final_population, :doubling_time]])

# Сравнительный график всех траекторий
p2 = plot(size=(800, 500), dpi=150)

for params in all_params

    data, _ = produce_or_load(
        datadir(script_name, "parametric_scan"),
        params,
        run_single_experiment,
        prefix = "scan"
    ) # Загружаем данные (они уже есть на диске)
```

```
    plot!(p2, data["time_points"], data["population_values"],
        label="α = $(params[:α])",
        lw=2,
        alpha=0.8
    )
end

plot!(p2,
    xlabel="Время, t",
    ylabel="Популяция, u(t)",
    title="Параметрическое исследование: влияние α на рост",
    legend=:topleft,
    grid=true
)

# Сохраним график в папку plots
savefig(plotsdir(script_name, "parametric_scan_comparison.png"))

# График зависимости времени удвоения от α
p3 = plot(results_df.α, results_df.doubling_time,
    seriestype=:scatter,
    label="Численное решение",
    xlabel="Скорость роста, α",
    ylabel="Время удвоения, t₂",
    title="Зависимость времени удвоения от α",
    markersize=8,
    markercolor=:red,
    legend=:topright
)

# Теоретическая кривая: t₂ = ln(2)/α
α_range = 0.1:0.01:1.0

plot!(p3, α_range, log(2) ./ α_range,
    label="Теория: t₂ = ln(2)/α",
    lw=2,
    linestyle=:dash,
    linecolor=:blue
)

# Сохраним график в папку plots
savefig(plotsdir(script_name, "doubling_time_vs_alpha.png"))

# ## Бенчмаркинг с разными параметрами
```

```
# **ИЗМЕНЕНИЕ:** Бенчмаркинг для разных значений α

println("\n" * "="^60)
println("Бенчмаркинг для разных значений α")
println("="^60)

benchmark_results = []
for α_value in param_grid[:α]

    bench_params = Dict(
        :u0 => [1.0],
        :α => α_value,
        :tspan => (0.0, 10.0),
        :solver => Tsit5(),
        :saveat => 0.1
    ) # Подготавливаем параметры для бенчмарка

    function benchmark_run() # Функция для бенчмарка
        prob = ODEProblem(exponential_growth!,
                          bench_params[:u0],
                          bench_params[:tspan],
                          (α=bench_params[:α],))
        return solve(prob, bench_params[:solver];
                     saveat=bench_params[:saveat])
    end

    println("\nБенчмарк для α = $α_value:")
    b = @benchmark $benchmark_run() samples=100 evals=1 # Запуск
     ↪ бенчмарка
    push!(benchmark_results, (α=α_value, time=median(b).time/1e9))  #
     ↪ время в секундах

    println("  Среднее время: ", round(median(b).time/1e9;
     ↪ digits=4), " сек")
end

# График зависимости времени вычисления от α
bench_df = DataFrame(benchmark_results)
p4 = plot(bench_df.α, bench_df.time,
    seriestype=:scatter,
    label="Время вычисления",
    xlabel="Скорость роста, α",
    ylabel="Время вычисления, сек",
    title="Зависимость времени вычисления от α",
```

```
    markersize=8,
    markercolor=:green,
    legend=:topleft
)

# Сохраним график в папку plots
savefig(plotsdir(script_name, "computation_time_vs_alpha.png"))

# ## Сохранение всех результатов
# **НОВАЯ СЕКЦИЯ:** Сохранение сводных данных для последующего
 ↪ анализа
@save datadir(script_name, "all_results.jld2") base_params
 ↪ param_grid all_params results_df bench_df

@save datadir(script_name, "all_plots.jld2") p1 p2 p3 p4

println("\n" * "="^60)
println("ЛАБОРАТОРНАЯ РАБОТА ЗАВЕРШЕНА")
println("="^60)
println("\nРезультаты сохранены в:")
println("  • data/$(script_name)/single/              - базовый
 ↪ эксперимент")
println("  • data/$(script_name)/parametric_scan/     -
 ↪ параметрическое сканирование")
println("  • data/$(script_name)/all_results.jld2     - сводные
 ↪ данные")
println("  • plots/$(script_name)/                    - все графики")
println("  • data/$(script_name)/all_plots.jld2      - объекты
 ↪ графиков")
println("\nДля анализа результатов используйте:")
println("  using JLD2, DataFrames")
println("  @load \"data/$(script_name)/all_results.jld2\"")
println("  println(results_df)")
```

— Выполните программу:

```
julia --project=. scripts/02_exponential_growth.jl
```

— Создайте производные форматы:

```
julia --project=. scripts/tangle.jl scripts/02_exponential_growth.jl
```

1.10.4.2 Jupyter notebook

Выполните Jupyter-ноутбук `notebooks/02_exponential_growth/02_exponential_growth.ipynb`.

1.10.4.3 Документирование в отчёте

— В каталоге отчёта в файл `_quarto.yml` включите поддержку кода `julia`. Добавьте после описания проекта:

```
engine: julia
julia:
  exeflags: ["--project=../project"]
```

— В преамбуле `preamble.tex` подключите пакет `juliamono`.
— В файле отчёта после описания выполнения лабораторной работы подключите файл описания программы:

```
{{< include ../project/markdown/02_exponential_growth/02_exponential
↪ _growth.qmd >}}
```

— Скомпилируйте отчёт.

1.11 Задание

— Создать рабочий каталог для всего курса.
— Создать рабочее пространство для программ в рамках лабораторной работы.
— Выполнить все задания по тексту лабораторной работы.
— Установить необходимые пакеты.
— Выполнить предложенный код.
— Преобразовать код в литературный стиль.
— Сгенерировать из литературного кода:
  — чистый код;

  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.

2

Основные модели

2.1 Модель SIR

Модель SIR есть классическая и фундаментальная математическая модель эпидемиологии, описывающая распространение инфекционного заболевания в закрытой популяции [4; 5].

2.1.1 Базовая модель

2.1.1.1 Ключевая идея

Модель SIR делит всю популяцию на три взаимосвязанные группы (компартменты), что отражено в её названии:

— S — Susceptible (Восприимчивые): люди, которые не болели, не имеют иммунитета и могут заразиться.
— I — Infectious (Инфицированные/Заразные): люди, которые в данный момент больны и могут передавать инфекцию.
— R — Recovered (Выздоровевшие/Удаленные): люди, которые переболели и приобрели иммунитет (или умерли). Они больше не участвуют в процессе передачи.

Основная цель модели: не предсказать судьбу конкретного человека, а понять

общую динамику эпидемии — будет ли она разрастаться, как быстро, сколько людей в итоге переболеет, как влияют карантинные меры.

2.1.1.2 Ключевые допущения

Модель строится на ряде допущений:

- Закрытая популяция: Общее число людей $N = S + I + R$ постоянно (не учитываются рождения, смерти от других причин, миграция).
- Однородное смешение: Каждый восприимчивый имеет равные шансы встретиться с каждым заразным.
- Постоянные параметры: Параметры заразности β и выздоровления γ не меняются со временем (на деле они зависят от поведения людей, сезона, появления штаммов).
- Отсутствие латентного периода: После заражения человек сразу становится заразным.
- Пожизненный иммунитет: Переболев, человек приобретает стойкий иммунитет и больше не может заразиться.

2.1.1.3 Математическое описание

Динамика модели описывается тремя уравнениями, которые показывают, как меняется число людей в каждой группе во времени.

1. Уравнение для восприимчивых (S):

$$\frac{dS}{dt} = -\beta IS/N,$$

   - dS/dt — скорость изменения числа восприимчивых;
   - β — коэффициент заражения (infection rate). Показывает, сколько в среднем человек заражает других в единицу времени, если все вокруг восприимчивы;
   - $-\beta IS/N$ — число новых случаев в единицу времени. Пропорционально и числу заразных (I), и доле восприимчивых (S/N). Знак минус показывает, что число восприимчивых уменьшается.
2. Уравнение для заразных (I):

$$\frac{dI}{dt} = \beta IS/N - \gamma I,$$

   - dI/dt — скорость изменения числа заразных;

- $\beta IS/N$ — приток новых больных (те, кто перешёл из S);
- γ — коэффициент выздоровления (recovery rate). Обратная величина $1/\gamma$ — это средняя продолжительность болезни (в тех же единицах времени, что и t);
- γI — отток людей, которые выздоровели (перешли в R). Например, если $1/\gamma = 10$ дней, то ежедневно выздоравливает примерно 1/10 часть всех больных.

3. Уравнение для выздоровевших (R):

$$\frac{dR}{dt} = \gamma I,$$

- dR/dt — скорость изменения числа выздоровевших;
- γI — поток людей, выздоравливающих из группы I.

2.1.1.4 Ключевые параметры и показатели

- Базовое репродуктивное число R_0 :

$$R_0 = \beta/\gamma.$$

  Среднее число людей, которых заражает один больной за все время своей болезни, в полностью восприимчивой популяции (когда $S \approx N$).

  - $R_0 > 1$: эпидемия будет расти (каждый больной заражает больше одного человека).
  - $R_0 < 1$: вспышка затухнет.
  - Пример: $R_0 = 3$ означает, что без мер один больной заразит в среднем троих.
- Эффективное репродуктивное число R_e:

$$R_e = R_0(S/N).$$

  Учитывает, что со временем восприимчивых становится меньше. Эпидемия начинает сходить на нет, когда R_e падает ниже 1 не из-за принятых мер, а из-за исчерпания пула восприимчивых.
- Порог коллективного иммунитета (Herd Immunity Threshold). Чтобы эпидемия не могла начаться, доля иммунных (либо переболевших, либо вакцинированных) должна превысить:

$$1 - 1/R_0.$$

  - Для $R_0 = 3$ порог = $1 - 1/3 \approx 67\%$. Это означает, что если иммунитет имеет более 67% популяции, вспышки не произойдёт.

2.1.1.5 Типичная динамика

На графике решения системы уравнений видна характерная картина.

- Начало: почти все в группе S. Появляется несколько I.
- Рост: число I начинает расти экспоненциально, S — быстро падает.
- Пик эпидемии: кривая I достигает максимума. Это происходит в момент, когда $R_e = 1$ (т.е. $S/N = 1/R_0$).
- Спад: после пика число I уменьшается, так как восприимчивых S осталось мало. Рост R замедляется.
- Завершение: Эпидемия затухает, I стремится к нулю. Часть популяции так и осталась в S (никогда не заразилась). Кривая R выходит на плато — это итоговое число переболевших.

2.1.1.6 Ограничения модели SIR

- Не учитываются демографические процессы (рождения, смерти, миграцию) — подходит для коротких эпидемий.
- Нет латентного периода (модель SEIR).
- Нет учёта возрастной структуры или разной степени активности людей.
- Предполагает однородность популяции и постоянство параметров.
- Не учитывает бессимптомных носителей (их можно частично включить в I).

2.1.1.7 Практическое применение

- Обучения основам эпидемиологии.
- Быстрой оценки масштаба угрозы по R_0.
- Прогнозирования нагрузки на систему здравоохранения (пиковое значение I — это необходимое число коек).
- Оценки эффекта мер: карантин, изоляция, маски снижают параметр β. Быстрая диагностика и лечение увеличивают γ (уменьшают время заразности).

2.1.1.8 Популярные модификации

- SIRS: Иммунитет временный, выздоровевшие снова становятся восприимчивыми (как при гриппе).
- SEIR: Добавлена группа E (Exposed) — инфицированные, но ещё не заразные.

— SIR с демографией: Добавлены рождение (приток в S) и смертность (отток из всех групп).
— Стратифицированные модели: Популяция делится на группы по возрасту, активности, географически.

2.1.1.9 Пример анализа

Допустим:

— N = 1 000 000 чел.
— β = 0.4 (один больной заражает 0.4 человека в день, если все восприимчивы).
— $1/\gamma$ = 10 дней (болеют 10 дней) => $\gamma = 0.1$.
— $R_0 = \beta/\gamma = 0.4/0.1 = 4$ (очень заразная болезнь).
— Вначале: $I_0 = 10$, $S_0 = 999990$, $R_0 = 0$.

Выводы из модели:

— Начальный рост будет очень быстрым.
— Пик эпидемии наступит, когда $S/N = 1/R_0 = 1/4$, то есть когда восприимчивых останется 25% (250 тыс. человек).
— Порог коллективного иммунитета: $1 - 1/4 = 75\%$. Именно столько в итоге переболеет к концу эпидемии (~750 тыс. человек).

2.1.2 Трёхпараметрическая модель SIR

2.1.2.1 Декомпозиция заразности

В классической двухпараметрической модели (β, γ) параметр β (коэффициент заражения) является составным. Он скрывает в себе два процесса:

— Контакт между людьми (поведенческий, управляемый фактор).
— Передачу инфекции при контакте (биологический фактор).

Трёхпараметрическая модель делает это разделение явным:

— c — среднее число контактов (достаточно тесных для передачи инфекции) одного человека в единицу времени.

- β — вероятность передачи инфекции при одном контакте между заразным и восприимчивым (безразмерная величина, от 0 до 1).
- γ — скорость выздоровления (доля инфицированных, выздоравливающих в единицу времени). Как и раньше, $1/\gamma$ — средняя продолжительность заразного периода.

Итоговый параметр силы заражения теперь выражается как произведение: $c \cdot \beta$.

2.1.2.2 Математическое описание

Поток новых заражений (dS/dt) теперь формируется так:

- У одного заразного (I) в день происходит c контактов.
- Доля восприимчивых среди всех контактов равна S/N. Следовательно, число релевантных контактов с восприимчивыми для всех заразных равно $cI(S/N)$.
- Но не каждый такой контакт приводит к заражению. Вероятность этого события — β.

Итоговая система уравнений:

- Уравнение для восприимчивых (S):

$$\frac{dS}{dt} = -(\beta c)IS/N.$$

  Скорость убывания S пропорциональна произведению: вероятность заражения (β) × скорость контактов (c) × доля восприимчивых (S/N) × число заразных (I).
- Уравнение для заразных (I):

$$\frac{dI}{dt} = (\beta c)IS/N - \gamma I.$$

  Прирост из S минус отток в R.
- Уравнение для выздоровевших (R):

$$dR/dt = \gamma I.$$

2.1.2.3 Базовое репродуктивное число R_0

Формула для R_0:

$$R_0 = (c\beta)/\gamma.$$

Смысл:

- $c\beta$ — это среднее число людей, которых заражает один больной за один день, если все вокруг восприимчивы.
- $1/\gamma$ — это количество дней, которые человек в среднем заразен.

R_0 = (число заражений в день) × (число заразных дней).

2.1.2.4 Пример расчета

- $c = 20$ контактов в день.
- $\beta = 0.05$ (5% вероятность передачи при контакте).
- $1/\gamma = 5$ дней (болеют 5 дней), следовательно $\gamma = 0.2$.
- $R_0 = (20 \cdot 0.05)/0.2 = (1)/0.2 = 5$.

Вывод: один больной в полностью восприимчивой популяции заразит в среднем 5 человек.

2.1.2.5 Динамика модели

Динамика кривых S, I, R качественно не меняется (такой же S-образный рост R и колокол I), но количественные результаты сильно зависят от параметров.

- До вмешательств: высокие c и β, стандартное γ → высокий R_0 → быстрый рост, высокий пик, много переболевших.
- Введение карантина ($c\downarrow$): R_0 падает ниже 1 ($R_e < 1$), кривая I начинает снижаться, пик сглаживается. Итоговое число переболевших ($R(\infty)$) может быть значительно меньше.
- Снятие карантина ($c\uparrow$): если снять меры слишком рано (когда S ещё велико), R_e снова станет больше 1, и произойдет вторая волна.

2.1.2.6 Ограничения

- Однородность популяции (все контактируют одинаково, что неверно: есть супер-распространители).
- Постоянство параметров во времени и для всех людей.
- Отсутствие инкубационного периода.
- Закрытая популяция.

2.1.2.7 Практическая ценность трёхпараметрической формы

- Планирование мер: позволяет оценить эффективность разных стратегий. Например, снизить c на 50% может быть дешевле, чем снизить β на 50%.
- Обратный расчёт: по наблюдаемой динамике эпидемии можно оценить не только R_0, но и приблизительно понять, за счёт чего он снизился (за счёт падения контактов c или за счёт мер защиты β).
- Наглядность для неспециалистов: легче объяснить политикам и населению, что нужно меньше контактировать ($c\downarrow$) и носить маски ($\beta\downarrow$).

2.1.3 Программный код

- Как основа для дальнейшей работы предлагается следующий программный код.
- Файл `scripts/sir_ode.jl`.

```
using DrWatson
@quickactivate "project"

using DifferentialEquations
using SimpleDiffEq
using Tables
using DataFrames
using StatsPlots
using LaTeXStrings  # Для красивого отображения формул на графиках
using Plots
using BenchmarkTools

script_name = splitext(basename(PROGRAM_FILE))[1]
mkpath(plotsdir(script_name))
mkpath(datadir(script_name))

function sir_ode!(du, u, p, t)
    (S, I, R) = u
    (β, c, γ) = p
    N = S + I + R
    @inbounds begin
        du[1] = -β * c * I / N * S
        du[2] = β * c * I / N * S - γ * I
        du[3] = γ * I
    end
```

```
    nothing
end

# Параметры модели
δt = 0.1
tmax = 40.0
tspan = (0.0, tmax)
u0 = [990.0, 10.0, 0.0]  # S, I, R
p = [0.05, 10.0, 0.25]   # β, c, γ

# Расчет базового репродуктивного числа
R0 = (p[2] * p[1]) / p[3]  # R₀ = (c * β) / γ

# Создание и решение задачи
prob_ode = ODEProblem(sir_ode!, u0, tspan, p)
sol_ode = solve(prob_ode, dt = δt)

# Подготовка данных в DataFrame
df_ode = DataFrame(Tables.table(sol_ode'))
rename!(df_ode, ["S", "I", "R"])
df_ode[!, :t] = sol_ode.t
df_ode[!, :N] = df_ode.S + df_ode.I + df_ode.R  # Общая численность
 ↪  популяции

# Вывод параметров модели
println("Параметры модели SIR:")
println("β (вероятность заражения) = ", p[1])
println("c (среднее число контактов) = ", p[2])
println("γ (скорость выздоровления) = ", p[3])
println("R₀ = c * β / γ = ", round(R0, digits=3))
println("Средняя продолжительность болезни = ", round(1/p[3],
 ↪  digits=2), " дней")
println("Начальные условия: S₀ = ", u0[1], ", I₀ = ", u0[2], ", R₀ =
 ↪  ", u0[3])

# 1. ОСНОВНОЙ ГРАФИК: динамика всех трех групп
plt1 = @df df_ode plot(:t,
    [:S :I :R],
    label=[L"S(t)" L"I(t)" L"R(t)"],
    xlabel="Время, дни",
    ylabel="Количество людей",
    title="Модель SIR: Динамика эпидемии",
    linewidth=2,
    legend=:right,
```

```
    grid=true,
    size=(800, 500))

# Добавление аннотаций с параметрами
annotate!(plt1, maximum(df_ode.t) * 0.7, maximum(df_ode.N) * 0.8,
    text("Параметры:\nβ = $(p[1])\nc = $(p[2])\nγ = $(p[3])\nR₀ =
 ↪  $(round(R0, digits=2))",
    8, :left))

# График только инфицированных (I)
plt2 = @df df_ode plot(:t, :I,
    label=L"I(t)",
    xlabel="Время, дни",
    ylabel="Количество инфицированных",
    title="Динамика числа зараженных",
    color=:red,
    linewidth=2,
    fill=(0, 0.3, :red),
    grid=true,
    size=(800, 400))

# Отметка пика эпидемии
peak_idx = argmax(df_ode.I)
peak_time = df_ode.t[peak_idx]
peak_value = df_ode.I[peak_idx]
vline!(plt2, [peak_time], color=:black, linestyle=:dash,
 ↪  label=false, linewidth=1)
annotate!(plt2, peak_time, peak_value * 1.05,
    text("Пик: $(round(peak_value, digits=1)) на $(round(peak_time,
 ↪  digits=1)) день",
    8, :top))

# График в логарифмическом масштабе (для анализа экспоненциального
 ↪  роста)
plt3 = @df df_ode plot(:t, :I,
    label=L"I(t)",
    xlabel="Время, дни",
    ylabel="Количество инфицированных (лог. масштаб)",
    title="Экспоненциальный рост (лог. шкала)",
    yscale=:log10,
    color=:red,
    linewidth=2,
    grid=true,
    size=(800, 400))
```

```
# График долей населения (в процентах)
plt4 = @df df_ode plot(:t,
    [:S :I :R] ./ df_ode.N .* 100,
    label=[L"S(t)/N" L"I(t)/N" L"R(t)/N"],
    xlabel="Время, дни",
    ylabel="Доля популяции, %",
    title="Динамика эпидемии (в процентах)",
    linewidth=2,
    legend=:right,
    grid=true,
    size=(800, 500))

# Горизонтальная линия для порога коллективного иммунитета
if R0 > 1
    herd_immunity_threshold = (1 - 1/R0) * 100
    hline!(plt4, [herd_immunity_threshold], color=:purple,
     ↪  linestyle=:dash,
        label="Порог коллективного иммунитета
         ↪  ($(round(herd_immunity_threshold, digits=1))%)",
        linewidth=1.5)
end

# Фазовый портрет (I vs S)
plt5 = plot(df_ode.S, df_ode.I,
    label="Фазовая траектория",
    xlabel=L"S(t)",
    ylabel=L"I(t)",
    title="Фазовый портрет SIR модели",
    color=:blue,
    linewidth=2,
    grid=true,
    size=(800, 500),
    legend=:topright)

# Добавление стрелок направления
for i in 1:50:length(df_ode.S)-1
    plot!(plt5, [df_ode.S[i], df_ode.S[i+1]], [df_ode.I[i],
     ↪  df_ode.I[i+1]],
        arrow=:closed, color=:blue, alpha=0.5, label=false)
end

# График Rₑ - эффективного репродуктивного числа
df_ode[!, :Re] = R0 .* df_ode.S ./ df_ode.N
```

```
plt6 = @df df_ode plot(:t, :Re,
    label=L"R_e(t)",
    xlabel="Время, дни",
    ylabel=L"R_e",
    title="Динамика эффективного репродуктивного числа",
    color=:green,
    linewidth=2,
    grid=true,
    size=(800, 400))

# Горизонтальная линия на уровне 1
hline!(plt6, [1.0], color=:red, linestyle=:dash, label="Порог
 ↪  эпидемии (Rₑ=1)", linewidth=1.5)

# Отметка момента, когда Rₑ становится < 1
cross_idx = findfirst(x -> x < 1, df_ode.Re)
if !isnothing(cross_idx) && cross_idx > 1
    cross_time = df_ode.t[cross_idx]
    vline!(plt6, [cross_time], color=:black, linestyle=:dash,
     ↪  label=false, linewidth=1)
    annotate!(plt6, cross_time, 1.2,
        text("Rₑ<1 с $(round(cross_time, digits=1)) дня", 8, :left))
end

# Компактный график всех кривых в одной панели
plt7 = plot(layout=(2, 3), size=(1200, 800))

# Верхний ряд
plot!(plt7[1], df_ode.t, df_ode.S, label=L"S(t)", color=1,
 ↪  linewidth=2, title="Восприимчивые")
plot!(plt7[2], df_ode.t, df_ode.I, label=L"I(t)", color=2,
 ↪  linewidth=2, title="Зараженные")
plot!(plt7[3], df_ode.t, df_ode.R, label=L"R(t)", color=3,
 ↪  linewidth=2, title="Выздоровевшие")

# Нижний ряд
plot!(plt7[4], df_ode.t, df_ode.I, label=L"I(t)", color=2,
 ↪  linewidth=2,
    yscale=:log10, title="Лог. масштаб")
plot!(plt7[5], df_ode.S, df_ode.I, label=false, color=4, linewidth=2,
    title="Фазовый портрет", xlabel=L"S", ylabel=L"I")
plot!(plt7[6], df_ode.t, df_ode.Re, label=L"R_e", color=:green,
 ↪  linewidth=2,
    title=L"R_e(t)", hline=[1.0], linestyle=:dash, linecolor=:red)
```

```
# Сохранение графиков

savefig(plt1, plotsdir(script_name, "sir_main.png"))
savefig(plt2, plotsdir(script_name, "sir_infected.png"))
savefig(plt3, plotsdir(script_name, "sir_log_scale.png"))
savefig(plt4, plotsdir(script_name, "sir_percentages.png"))
savefig(plt5, plotsdir(script_name, "sir_phase_portrait.png"))
savefig(plt6, plotsdir(script_name, "sir_effective_R.png"))
savefig(plt7, plotsdir(script_name, "sir_panel.png"))

# Бенчмарк для оценки производительности
println("\nБенчмарк решения:")
@benchmark solve(prob_ode, dt = δt)

# Дополнительный анализ
println("\n=== АНАЛИЗ РЕЗУЛЬТАТОВ ===")
println("Общая численность популяции (контроль): N = ",
 ↪  round(df_ode.N[1], digits=1))
println("Пиковое число зараженных: I_max = ", round(peak_value,
 ↪  digits=1))
println("Время достижения пика: t_peak = ", round(peak_time,
 ↪  digits=1), " дней")
println("Итоговое число переболевших: R(∞) = ", round(df_ode.R[end],
 ↪  digits=1))
println("Доля переболевших: ", round(df_ode.R[end]/df_ode.N[1]*100,
 ↪  digits=1), "%")

if R0 > 1
    println("\nТеоретический анализ:")
    println("  - Порог коллективного иммунитета: ",
     ↪  round((1-1/R0)*100, digits=1), "%")
    println("  - Теоретический пик при S/N = 1/R₀ = ", round(1/R0,
     ↪  digits=3))
end
```

2.2 Модель Лотки–Вольтерры

2.2.1 Ключевая идея и исторический контекст

Модель Лотки-Вольтерры — это фундаментальная математическая модель в экологии, описывающая динамику взаимодействия двух видов: хищников и жертв. Она была независимо предложена в 1920-х годах:

— Альфредом Лоткой (1925) для химических реакций [7; 8].
— Витторио Вольтеррой (1926) для объяснения колебаний улова рыбы в Адриатическом море [10; 12].

Модель демонстрирует, как даже простая система взаимодействий может порождать сложные колебательные режимы, объясняя циклические изменения численности в природных экосистемах.

2.2.2 Основные допущения модели

Модель строится на следующих упрощающих предположениях:

— Закрытая система: популяции изолированы, нет миграции.
— Неограниченные ресурсы для жертв: в отсутствие хищников жертвы растут экспоненциально.
— Линейная функциональная реакция: вероятность встречи хищника и жертвы пропорциональна произведению их численностей.
— Постоянные параметры: коэффициенты взаимодействия не меняются во времени.
— Отсутствие внутривидовой конкуренции: нет конкуренции за ресурсы внутри вида.
— Хищники питаются только жертвами: нет альтернативных источников пищи.
— Отсутствие временных задержек: все процессы происходят мгновенно.

2.2.3 Математическая формализация

Система состоит из двух дифференциальных уравнений:

2.2.3.1 Уравнение для популяции жертв (x)

$$\frac{dx}{dt} = \alpha x - \beta xy,$$

где: - $\alpha > 0$ — коэффициент естественного прироста жертв (рождаемость); - $\beta > 0$ — коэффициент поедания жертв хищниками; - xy — вероятность встречи хищника и жертвы (закон массового действия).

Смысл: Численность жертв увеличивается за счёт размножения (αx) и уменьшается за счёт поедания хищниками (βxy).

2.2.3.2 Уравнение для популяции хищников (y)

$$\frac{dy}{dt} = \delta xy - \gamma y,$$

где: - $\delta > 0$ — коэффициент конверсии (эффективность превращения съеденных жертв в потомство хищников), - $\gamma > 0$ — коэффициент естественной смертности хищников.

Смысл: Численность хищников увеличивается за счёт питания жертвами (δxy) и уменьшается за счёт естественной смертности (γy).

2.2.4 Ключевые параметры и их экологическая интерпретаци

Параметр	Обозначение	Экологический смысл	Единицы измерения	Типичные значения
α	Скорость размножения жертв	Среднее число потомков на одну жертву в единицу времени	1/время	0.1-1.0 день$^{-1}$
β	Скорость поедания	Доля жертв, съедаемых одним хищником в единицу времени	1/(хищник×время)	0.01-0.1 день$^{-1}$
δ	Эффективность конверсии	Количество новых хищников, рождающихся от одной съеденной жертвы	хищник/жертва	0.01-0.1

Параметр	Обозначение	Экологический смысл	Единицы измерения	Типичные значения
γ	Смертность хищников	Доля хищников, умирающих в единицу времени	1/время	0.1-0.5 день$^{-1}$

2.2.5 Анализ системы: стационарные точки и их устойчивость

2.2.5.1 Стационарные точки (положения равновесия)

- Тривиальное равновесие (0, 0): обе популяции вымирают.
  - Всегда неустойчиво (седло).
- Равновесие только жертв (x*, 0): хищники вымерли.
  - x* = бесконечность (в модели без ограничений).
  - В реалистичных моделях с ёмкостью среды: $x* = K$.
- Нетривиальное равновесие (x, y): сосуществование видов.
  - $x* = \gamma/\delta$ (равновесие жертв).
  - $y* = \alpha/\beta$ (равновесие хищников).
  - Центр (нейтрально устойчивая точка): в линейном приближении.
  - Устойчивый фокус/неустойчивый фокус: при модификациях модели.

2.2.5.2 Интерпретация стационарных точек

- $x* = \gamma/\delta$: чем выше смертность хищников (γ) или ниже эффективность конверсии (δ), тем больше равновесная численность жертв.
- $y* = \alpha/\beta$: чем выше рождаемость жертв (α) или ниже эффективность охоты (β), тем больше равновесная численность хищников.

2.2.6 Динамика системы и фазовый портрет

2.2.6.1 Характерные черты решения

- Колебательный характер: популяции испытывают циклические колебания.
- Сдвиг фаз: пик численности хищников отстаёт от пика численности жертв.

  - Типичный сдвиг: 1/4 периода (90°).
- Консервативная система: в классической модели сохраняется величина:

$$V(x, y) = \delta x - \gamma lnx + \beta y - \alpha lny = const.$$

- Замкнутые траектории: На фазовой плоскости (x-y) получаются замкнутые кривые вокруг точки равновесия.

2.2.6.2 Типичная последовательность событий за один цикл

Рост количества жертв → Рост количества хищников (лаг) → Падение количества жертв → Падение хищников количества (лаг) → ...

2.2.7 Изоклины и направления движения

2.2.7.1 Изоклина жертв (dx/dt = 0)

Горизонтальная линия:

$$y = \alpha/\beta = const.$$

- Выше изоклины: $dx/dt < 0$ (жертвы убывают).
- Ниже изоклины: $dx/dt > 0$ (жертвы растут).

2.2.7.2 Изоклина хищников (dy/dt = 0)

Вертикальная линия:

$$x = \gamma/\delta = const.$$

- Правее изоклины: dy/dt > 0 (хищники растут)
- Левее изоклины: dy/dt < 0 (хищники убывают)

2.2.7.3 Разбиение фазовой плоскости на 4 квадранта

- Северо-Восток: $x > \gamma/\delta$, $y > \alpha/\beta \to x \downarrow, y \uparrow$.
- Северо-Запад: $x < \gamma/\delta$, $y > \alpha/\beta \to x \downarrow, y \downarrow$.
- Юго-Запад: $x < \gamma/\delta$, $y < \alpha/\beta \to x \uparrow, y \downarrow$.
- Юго-Восток: $x > \gamma/\delta$, $y < \alpha/\beta \to x \uparrow, y \uparrow$.

2.2.8 8. Период колебаний и амплитуда

2.2.8.1 Период малых колебаний (вблизи равновесия)

$$T \approx \frac{2\pi}{\sqrt{\alpha\gamma}}.$$

- Период зависит только от параметров α и γ.
- Не зависит от параметров взаимодействия β и δ.
- Чем выше рождаемость жертв (α) или смертность хищников (γ), тем короче период.

2.2.8.2 Амплитуда колебаний

- Зависит от начальных условий
- В классической модели: амплитуда сохраняется
- В модифицированных моделях: может меняться

2.2.9 Ограничения модели

- Слишком идеализирована: игнорирует многие реальные факторы.
- Нейтральная устойчивость: в классическом виде не соответствует реальным системам.
- Отсутствие адаптации: параметры постоянны, нет эволюции.
- Чувствительность к начальным условиям: в стохастических версиях.
- В природе редко наблюдаются чистые циклы без затухания/нарастания.
- Влияние факторов среды (климат, погода) не учитывается.
- Пространственная структура популяций игнорируется.

2.2.10 Программный код

- Как основа для дальнейшей работы предлагается следующий программный код.
- Файл `scripts/lv_ode.jl`.

```
using DrWatson
```

```julia
@quickactivate "project"

using DifferentialEquations
using DataFrames
using StatsPlots
using LaTeXStrings
using Plots
using Statistics
using FFTW

script_name = splitext(basename(PROGRAM_FILE))[1]
mkpath(plotsdir(script_name))
mkpath(datadir(script_name))

# Описание модели Лотки-Вольтерры
"""
Модель Лотки-Вольтерры (хищник-жертва)

Система уравнений:
  dx/dt = αx - βxy  # Изменение популяции жертв
  dy/dt = δxy - γy  # Изменение популяции хищников

Где:
  x - популяция жертв (например, зайцы)
  y - популяция хищников (например, лисы)
  α - естественный прирост жертв (в отсутствие хищников)
  β - коэффициент поедания жертв хищниками
  δ - коэффициент прироста хищников за счет поедания жертв
  γ - естественная смертность хищников (в отсутствие жертв)
"""

function lotka_volterra!(du, u, p, t)
    x, y = u          # x - жертвы, y - хищники
    α, β, δ, γ = p    # параметры модели

    @inbounds begin
    du[1] = α*x - β*x*y       # уравнение для жертв
    du[2] = δ*x*y - γ*y       # уравнение для хищников
    end
    nothing
end

# Параметры модели и начальные условия
# Классические параметры из литературы
```

```
p_lv = [0.1,   # α: скорость размножения жертв
    0.02,  # β: скорость поедания жертв хищниками
    0.01,  # δ: коэффициент конверсии пищи (жертв) в хищников
    0.3]   # γ: смертность хищников

# Начальные условия: [жертвы, хищники]
u0_lv = [40.0, 9.0]  # начальная популяция

# Временные параметры
tspan_lv = (0.0, 200.0)  # длительность симуляции
dt_lv = 0.01              # шаг интегрирования

# Создание и решение задачи
prob_lv = ODEProblem(lotka_volterra!, u0_lv, tspan_lv, p_lv)
sol_lv = solve(prob_lv,
               dt = dt_lv,
               Tsit5(),           # Метод 5-го порядка
               reltol=1e-8,       # Относительная точность
               abstol=1e-10,      # Абсолютная точность
               saveat=0.1,        # Сохраняем каждые 0.1 единицы
                ↪  времени
               dense=true         # Включаем плотный вывод для
                ↪  интерполяции
)

# Подготовка данных
df_lv = DataFrame()
df_lv[!, :t] = sol_lv.t
df_lv[!, :prey] = [u[1] for u in sol_lv.u]  # жертвы
df_lv[!, :predator] = [u[2] for u in sol_lv.u]  # хищники

# Рассчет производных для анализа
df_lv[!, :dprey_dt] = p_lv[1] .* df_lv.prey .- p_lv[2] .* df_lv.prey
 ↪  .* df_lv.predator
df_lv[!, :dpredator_dt] = p_lv[3] .* df_lv.prey .* df_lv.predator .-
 ↪  p_lv[4] .* df_lv.predator

# Вывод информации о модели
println("="^60)
println("Модель Лотки-Вольтерры (хищник-жертва)")
println("="^60)
println("\nПараметры модели:")
println("α (скорость размножения жертв) = ", p_lv[1])
println("β (скорость поедания жертв) = ", p_lv[2])
```

```
println("δ (коэффициент конверсии) = ", p_lv[3])
println("γ (смертность хищников) = ", p_lv[4])
println("\nНачальные условия:")
println("Жертвы (x₀) = ", u0_lv[1])
println("Хищники (y₀) = ", u0_lv[2])

# Стационарные точки (нулевые изоклины)
x_star = p_lv[4] / p_lv[3]  # стационарная точка для жертв
y_star = p_lv[1] / p_lv[2]  # стационарная точка для хищников
println("\nСтационарные точки (положения равновесия):")
println("x* = γ/δ = ", round(x_star, digits=3))
println("y* = α/β = ", round(y_star, digits=3))

# Построение графиков

# График 1: Динамика популяций во времени
plt1 = plot(df_lv.t, [df_lv.prey df_lv.predator],
    label=[L"Жертвы (x)" L"Хищники (y)"],
    xlabel="Время",
    ylabel="Популяция",
    title="Модель Лотки-Вольтерры: Динамика популяций",
    linewidth=2,
    legend=:topright,
    grid=true,
    size=(900, 500),
    color=[:green :red])

# Добавление стационарных уровней
hline!(plt1, [x_star], color=:green, linestyle=:dash, alpha=0.5,
 ↪  label="x* (равновесие жертв)")
hline!(plt1, [y_star], color=:red, linestyle=:dash, alpha=0.5,
 ↪  label="y* (равновесие хищников)")

# График 2: Фазовый портрет (хищники vs жертвы)
plt2 = plot(df_lv.prey, df_lv.predator,
    label="Фазовая траектория",
    xlabel="Популяция жертв (x)",
    ylabel="Популяция хищников (y)",
    title="Фазовый портрет системы",
    color=:blue,
    linewidth=1.5,
    grid=true,
    size=(800, 600),
    legend=:topright)
```

```
# Добавление стрелок направления на фазовом портрете
step = 50  # шаг для отображения стрелок
for i in 1:step:length(df_lv.prey)-step
    plot!(plt2, [df_lv.prey[i], df_lv.prey[i+step]],
      [df_lv.predator[i], df_lv.predator[i+step]],
      arrow=:closed, color=:blue, alpha=0.3, label=false)
end

# Добавление стационарной точки
scatter!(plt2, [x_star], [y_star],
    color=:black, markersize=8, label="Стационарная точка (x*, y*)")

# Изоклины (нулевого роста)
x_range = LinRange(0, maximum(df_lv.prey)*1.1, 100)
y_nullcline = p_lv[1] ./ (p_lv[2] .* x_range)  # y-изоклина (dy/dt =
 ↪  0)
plot!(plt2, x_range, y_nullcline,
    color=:red, linestyle=:dash, linewidth=1.5, label="Изоклина
     ↪  хищников (dy/dt=0)")

y_range = LinRange(0, maximum(df_lv.predator)*1.1, 100)
x_nullcline = p_lv[4] ./ (p_lv[3] .* ones(length(y_range)))  #
 ↪  x-изоклина (dx/dt = 0)
plot!(plt2, x_nullcline, y_range,
    color=:green, linestyle=:dash, linewidth=1.5, label="Изоклина
     ↪  жертв (dx/dt=0)")

# График 3: Производные (скорости изменения)
plt3 = plot(df_lv.t, [df_lv.dprey_dt df_lv.dpredator_dt],
    label=[L"dx/dt" L"dy/dt"],
    xlabel="Время",
    ylabel="Скорость изменения",
    title="Производные популяций",
    linewidth=1.5,
    legend=:topright,
    grid=true,
    size=(900, 400),
    color=[:green :red])

hline!(plt3, [0], color=:black, linestyle=:solid, alpha=0.3,
 ↪  label=false)

# График 4: Относительные изменения (в %)
```

```
df_lv[!, :prey_pct_change] = df_lv.dprey_dt ./ df_lv.prey .* 100
df_lv[!, :predator_pct_change] = df_lv.dpredator_dt ./
 ↪  df_lv.predator .* 100

plt4 = plot(df_lv.t, [df_lv.prey_pct_change
 ↪  df_lv.predator_pct_change],
    label=[L"dx/dt / x (\%)" L"dy/dt / y (\%)"],
    xlabel="Время",
    ylabel="Относительное изменение, %",
    title="Относительные темпы роста",
    linewidth=1.5,
    legend=:topright,
    grid=true,
    size=(900, 400),
    color=[:green :red])

# График 5: Спектральный анализ (быстрое преобразование Фурье)
function compute_fft(signal, dt)
    n = length(signal)
    # Используем rfft для вещественных сигналов (возвращает только
     ↪  положительные частоты)
    spectrum = abs.(rfft(signal))
    # Соответствующие частоты для rfft
    freq = rfftfreq(n, 1/dt)
    return freq, spectrum
end

# Вычисление периодов колебаний
freq_prey, spectrum_prey = compute_fft(df_lv.prey .-
 ↪  mean(df_lv.prey), dt_lv)
freq_predator, spectrum_predator = compute_fft(df_lv.predator .-
 ↪  mean(df_lv.predator), dt_lv)

plt5 = plot(freq_prey, [spectrum_prey spectrum_predator],
    label=[L"Жертвы (x)" L"Хищники (y)"],
    xlabel="Частота",
    ylabel="Амплитуда",
    title="Спектральный анализ (Фурье)",
    linewidth=1.5,
    xscale=:log10,
    yscale=:log10,
    legend=:topright,
    grid=true,
    size=(800, 400),
```

```
    color=[:green :red])

# Нахождение доминирующих частот
if length(spectrum_prey) > 0
    idx_prey = argmax(spectrum_prey[2:end]) + 1  # пропускаем нулевую
     ↪  частоту
    dominant_freq_prey = freq_prey[idx_prey]
    period_prey = 1/dominant_freq_prey
    println("\nДоминирующая частота колебаний жертв: ",
     ↪  round(dominant_freq_prey, digits=4), " Гц")
    println("Период колебаний жертв: ", round(period_prey,
     ↪  digits=2), " единиц времени")
end

# График 6: Компактная панель всех графиков
plt6 = plot(layout=(3, 2), size=(1200, 900))

plot!(plt6[1], df_lv.t, df_lv.prey, label=L"x(t)", color=:green,
 ↪  linewidth=2,
      title="Популяция жертв", grid=true)
plot!(plt6[2], df_lv.t, df_lv.predator, label=L"y(t)", color=:red,
 ↪  linewidth=2,
      title="Популяция хищников", grid=true)
plot!(plt6[3], df_lv.prey, df_lv.predator, label=false, color=:blue,
 ↪  linewidth=1.5,
      title="Фазовый портрет", xlabel=L"x", ylabel=L"y", grid=true)
scatter!(plt6[3], [x_star], [y_star], color=:black, markersize=5,
 ↪  label="(x*, y*)")
plot!(plt6[4], df_lv.t, [df_lv.dprey_dt df_lv.dpredator_dt],
      label=[L"dx/dt" L"dy/dt"], color=[:green :red], linewidth=1.5,
      title="Скорости изменения", grid=true, legend=:topright)
plot!(plt6[5], freq_prey, spectrum_prey, label=L"x", color=:green,
 ↪  linewidth=1.5,
      title="Спектр жертв", xscale=:log10, yscale=:log10, grid=true)
plot!(plt6[6], df_lv.t, [df_lv.prey_pct_change
 ↪  df_lv.predator_pct_change],
      label=[L"dx/x" L"dy/y"], color=[:green :red], linewidth=1.5,
      title="Относительные изменения", grid=true, legend=:topright)

# Анализ результатов
println("\n" * "="^60)
println("Анализ результатов")
println("="^60)
```

```
println("\nОсновные статистики:")
println("Жертвы: min = ", round(minimum(df_lv.prey), digits=2),
        ", max = ", round(maximum(df_lv.prey), digits=2),
        ", mean = ", round(mean(df_lv.prey), digits=2))
println("Хищники: min = ", round(minimum(df_lv.predator), digits=2),
        ", max = ", round(maximum(df_lv.predator), digits=2),
        ", mean = ", round(mean(df_lv.predator), digits=2))

# Упрощенный анализ колебаний без поиска максимумов
# Вместо сложного анализа сдвига фаз, просто посчитаем основные
 ↪  характеристики

# Находим время первого пика жертв (простой алгоритм)
function find_first_peak(signal, time)
    for i in 2:length(signal)-1
        if signal[i] > signal[i-1] && signal[i] > signal[i+1]
            return time[i], signal[i]
        end
    end
    return NaN, NaN
end

peak_time_prey, peak_value_prey = find_first_peak(df_lv.prey,
 ↪  df_lv.t)
peak_time_predator, peak_value_predator =
 ↪  find_first_peak(df_lv.predator, df_lv.t)

if !isnan(peak_time_prey) && !isnan(peak_time_predator)
    phase_shift = peak_time_predator - peak_time_prey
    println("\nАнализ колебаний:")
    println("Первый пик жертв: время = ", round(peak_time_prey,
     ↪  digits=2),
            ", значение = ", round(peak_value_prey, digits=2))
    println("Первый пик хищников: время = ",
     ↪  round(peak_time_predator, digits=2),
            ", значение = ", round(peak_value_predator, digits=2))
    println("Сдвиг фаз (хищники отстают): ", round(phase_shift,
     ↪  digits=2))
end

# Сохранение графиков

savefig(plt1, plotsdir(script_name, "lv_dynamics.png"))
savefig(plt2, plotsdir(script_name, "lv_phase_portrait.png"))
```

```
savefig(plt3, plotsdir(script_name, "lv_derivatives.png"))
savefig(plt4, plotsdir(script_name, "lv_relative_changes.png"))
savefig(plt5, plotsdir(script_name, "lv_spectrum.png"))
savefig(plt6, plotsdir(script_name, "lv_panel.png"))

# Дополнительный анализ: чувствительность к параметрам
println("\n\n" * "="^60)
println("Анализ чувствительности")
println("="^60)

# Функция для анализа влияния параметров
function analyze_parameter_sensitivity(param_index, values,
 ↪  param_name)
    println("\nАнализ чувствительности к параметру: ", param_name)
    results = []

    for val in values
    p_test = copy(p_lv)
    p_test[param_index] = val
    prob_test = ODEProblem(lotka_volterra!, u0_lv, tspan_lv, p_test)
    sol_test = solve(prob_test, dt = dt_lv)
    prey_end = sol_test.u[end][1]
    predator_end = sol_test.u[end][2]

    push!(results, (param=val, prey=prey_end, predator=predator_end))
    println("  $(param_name)=$(val): жертвы=$(round(prey_end,2)),
     ↪  хищники=$(round(predator_end,2))")
    end
    return results
end

# Анализ чувствительности к ключевым параметрам
if false  # Установите в true для выполнения анализа
    println("\n1. Влияние скорости размножения жертв (α):")
    analyze_parameter_sensitivity(1, [0.05, 0.1, 0.2, 0.3], "α")

    println("\n2. Влияние смертности хищников (γ):")
    analyze_parameter_sensitivity(4, [0.1, 0.3, 0.5, 0.7], "γ")
end

println("\nМоделирование завершено успешно!")
```

2.3 Задание

- Создать рабочий каталог для кода.
- Установить необходимые пакеты.
- Выполнить предложенный код.
- Преобразовать код в литературный стиль.
- Сгенерировать из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.

3

Агентное моделирование

3.1 Агентный подход

Агентный подход к имитационному моделированию (Agent-Based Modeling, ABM) — это метод исследования сложных систем, в котором поведение системы возникает из взаимодействия множества автономных сущностей, называемых агентами. Вместо того чтобы описывать систему глобальными уравнениями, мы моделируем каждую индивидуальную единицу и правила её поведения, а затем наблюдаем, какие коллективные паттерны появляются снизу вверх. Этот подход особенно полезен, когда поведение системы трудно предсказать из-за нелинейностей, гетерогенности участников или адаптивных стратегий.

3.1.1 Ключевые компоненты агентной модели

Любая агентная модель включает три основных элемента:

- Агенты — это активные сущности. Они обладают:
  - *Свойствами (атрибутами)*: возраст, цвет, запас ресурсов, координаты и т.п.
  - *Правилами поведения*: как агент реагирует на изменения среды и действия других агентов (например, перемещение, размножение, потребление).
  - *Целями* (не обязательно): в некоторых моделях агенты стремятся максимизировать свою выгоду или выжить.
  - *Способностью к обучению или адаптации* (в продвинутых моделях).

- Среда — это пространство, в котором существуют агенты. Она может быть:
  - *Дискретной* (клеточная сетка, как в Daisyworld).
  - *Непрерывной* (двумерное или трёхмерное пространство).
  - *Сетевой* (граф социальных связей).
  - *Абстрактной* (например, рынок без явных координат).

Среда также может иметь свои свойства (температура, ресурсы) и динамику (диффузия, обновление).

- Взаимодействия — правила, определяющие, как агенты влияют друг на друга и на среду. Они могут быть:
  - *Локальными* (только с соседями).
  - *Глобальными* (все агенты взаимодействуют со всеми).
  - *Через среду* (агенты изменяют среду, а среда влияет на агентов).

3.1.2 Основные принципы агентного моделирования

- Эмерджентность: глобальное поведение системы не закладывается явно, а возникает из локальных взаимодействий. Это позволяет открывать неожиданные закономерности.
- Автономия: агенты действуют независимо, на основе своей внутренней логики.
- Гетерогенность: агенты могут различаться по своим характеристикам и правилам, что отражает реальное разнообразие.
- Локальность: чаще всего агенты обладают информацией только о своём ближайшем окружении.

3.1.3 Преимущества агентного подхода

- Гибкость: можно моделировать системы любой сложности, включая социальные, биологические, экономические.
- Учёт индивидуальных различий: вместо «среднего пациента» или «среднего потребителя» моделируются конкретные индивиды.
- Адаптивность: агенты могут менять своё поведение в ответ на изменения среды.
- Наглядность: динамику модели можно визуализировать, наблюдая за агентами в пространстве.
- Исследование механизмов: позволяет проверять, как микро-правила приводят к макро-явлениям.

3.1.4 Области применения

Агентное моделирование используется в экологии (популяционная динамика, модели вроде Daisyworld), экономике (рынки, поведение потребителей), социологии (распространение мнений, сегрегация), эпидемиологии (распространение инфекций), транспорте (движение пешеходов и автомобилей), политологии и многих других областях.

3.1.5 Инструменты для агентного моделирования

В Julia основным инструментом является пакет Agents.jl [3], который предоставляет гибкую инфраструктуру для создания ABM, включая различные типы пространств, визуализацию и сбор данных. В других языках популярны NetLogo (специализированный язык), Mesa (Python), AnyLogic и др.

3.1.6 Этапы построения агентной модели

— Концептуализация: определить цель модели, выделить ключевых агентов, их атрибуты и правила, описать среду и взаимодействия.
— Спецификация: формально описать модель (можно использовать псевдокод или диаграммы).
— Реализация: написать код на выбранном языке.
— Верификация: проверить, что программа правильно реализует задуманные правила (нет ошибок).
— Валидация: сравнить результаты модели с реальными данными или ожидаемым поведением, чтобы убедиться в адекватности модели.
— Анализ: провести эксперименты, варьируя параметры, изучить чувствительность, выявить закономерности.
— Интерпретация: связать полученные результаты с реальным миром или теоретическими концепциями.

3.2 Daisyworld

3.2.1 Модель Daisyworld

Модель Daisyworld, предложенная Джеймсом Лавлоком и Эндрю Уотсоном, иллюстрирует гипотезу Геи [2; 11]. Гипотеза Геи рассматривает планету как единую, саморегулирующуюся систему, включающую как живые, так и неживые части.

В мире маргариток произрастают чёрные и белые маргаритки. Их альбедо различается: чёрные маргаритки поглощают свет и тепло, нагревая окружающую среду; белые маргаритки делают наоборот. Маргаритки могут размножаться только в определённом температурном диапазоне, а это значит, что слишком много (или слишком мало) тепла от солнца и/или окружающей среды в конечном итоге остановит их размножение.

Поверхность Геи нагревается солнцем, но растущие на ней маргаритки поглощают или отражают свет звёзд, изменяя локальную температуру.

Если температура становится слишком высокой или слишком низкой, маргаритки не захотят размножаться. Пока температура благоприятна, маргаритки конкурируют за землю и пытаются дать начало новому растению в местах, расположенных неподалёку от них.

Когда климат слишком холодный, черным маргариткам необходимо размножаться, чтобы повысить температуру, и наоборот — когда климат слишком теплый, необходимо производить больше белых маргариток, чтобы охладить климат.

Взаимодействие живых и неживых аспектов этого мира находит равновесие в широком диапазоне параметров, хотя при достаточно сильном внешнем воздействии маргаритки не смогут регулировать температуру планеты и в конечном итоге вымрут.

3.2.2 Описание модели Daisyworld

В модели Daisyworld агентами являются чёрные и белые маргаритки. Они живут на клеточной сетке (среда).

Свойства агентов: вид и возраст.

Правила:

— Маргаритки изменяют локальную температуру за счёт разного альбедо.
— Температура влияет на вероятность размножения (чем ближе к оптимуму, тем выше шанс заселить соседнюю пустую клетку).
— Агенты стареют и умирают после определённого возраста.
— Среда (температура) диффундирует между клетками.

3.2.2.1 Агенты

Маргаритки двух типов — чёрные (`black`) и белые (`white`). Каждая маргаритка занимает одну клетку сетки и имеет возраст (количество шагов с момента появления).

3.2.2.2 Пространство

Двумерная квадратная сетка (например, 30×30) с периодическими границами.

3.2.2.3 Параметры

— `luminosity` – солнечная постоянная (может изменяться со временем).
— `albedo_black`, `albedo_white` – альбедо чёрных и белых маргариток.
— `surface_albedo` — альбедо пустой почвы.
— `max_age` — максимальный возраст маргаритки.

3.2.2.4 Динамика

— Для каждой клетки рассчитывается локальная температура.
— Каждая маргаритка может погибнуть с вероятностью, зависящей от температуры. Если температура выходит за допустимый диапазон, вероятность смерти повышается.
— Если клетка пуста, на неё может попасть семя от соседней маргаритки. Вероятность успешного прорастания зависит от температуры клетки. Если вероятность превышает случайное число, на пустой клетке появляется новая маргаритка того же цвета, что и родительская.

3.2.3 Реализация на Agents.jl

Создадим файл `src/daisyworld.jl`. Здесь мы определим тип агента и функции шага модели.

```
using Agents
import StatsBase
using Random

@agent struct Daisy(GridAgent{2})
    breed::Symbol
    age::Int
    albedo::Float64 # 0-1 fraction
end

function update_surface_temperature!(pos, model)
    absorbed_luminosity = if isempty(pos, model) # no daisy
        (1 - model.surface_albedo) * model.solar_luminosity
    else
        daisy = model[id_in_position(pos, model)]
        (1 - daisy.albedo) * model.solar_luminosity
    end
    local_heating = absorbed_luminosity > 0 ? 72 *
     ↪  log(absorbed_luminosity) + 80 : 80
    model.temperature[pos...] = (model.temperature[pos...] +
     ↪  local_heating) / 2
end

function diffuse_temperature!(pos, model)
    ratio = model.ratio # diffusion ratio
    npos = nearby_positions(pos, model)
    model.temperature[pos...] =
        (1 - ratio) * model.temperature[pos...] +
        sum(model.temperature[p...] for p in npos) * 0.125 * ratio
end

function propagate!(pos, model)
    isempty(pos, model) && return
    daisy = model[id_in_position(pos, model)]
    temperature = model.temperature[pos...]
    seed_threshold = (0.1457 * temperature - 0.0032 * temperature^2)
     ↪  - 0.6443
    if rand(abmrng(model)) < seed_threshold
```

```
        empty_near_pos = random_nearby_position(pos, model, 1, npos
         ↪  -> isempty(npos, model))
        if !isnothing(empty_near_pos)
            add_agent!(empty_near_pos, model, daisy.breed, 0,
             ↪  daisy.albedo)
        end
    end
end

function daisy_step!(agent::Daisy, model)
    agent.age += 1
    agent.age ≥ model.max_age && remove_agent!(agent, model)
end

function daisyworld_step!(model)
    for p in positions(model)
        update_surface_temperature!(p, model)
        diffuse_temperature!(p, model)
        propagate!(p, model)
    end
    model.tick = model.tick + 1
    solar_activity!(model)
end

function solar_activity!(model)
    if model.scenario == :ramp
        if model.tick > 200 && model.tick ≤ 400
            model.solar_luminosity += model.solar_change
        end
        if model.tick > 500 && model.tick ≤ 750
            model.solar_luminosity -= model.solar_change / 2
        end
    elseif model.scenario == :change
        model.solar_luminosity += model.solar_change
    end
end

using Random

function daisyworld(;
    griddims = (30, 30),
    max_age = 25,
    init_white = 0.2, # % cover of the world surface of white breed
    init_black = 0.2, # % cover of the world surface of black breed
```

```
    albedo_white = 0.75,
    albedo_black = 0.25,
    surface_albedo = 0.4,
    solar_change = 0.005,
    solar_luminosity = 1.0, # initial luminosity
    scenario = :default,
    seed = 165,
)

    rng = MersenneTwister(seed)
    space = GridSpaceSingle(griddims)
    properties = (;max_age, surface_albedo, solar_luminosity,
     ↪  solar_change, scenario,
        tick = 0, ratio = 0.5, temperature = zeros(griddims)
    )
    properties = Dict(k⇒v for (k,v) in pairs(properties))

    model = StandardABM(Daisy, space; properties, rng, agent_step! =
     ↪  daisy_step!, model_step! = daisyworld_step!)

    grid = collect(positions(model))
    num_positions = prod(griddims)
    white_positions =
        StatsBase.sample(grid, Int(init_white * num_positions);
         ↪  replace = false)
    for wp in white_positions
        add_agent!(wp, Daisy, model, :white, rand(abmrng(model),
         ↪  0:max_age), albedo_white)
    end
    allowed = setdiff(grid, white_positions)
    black_positions =
        StatsBase.sample(allowed, Int(init_black * num_positions);
         ↪  replace = false)
    for bp in black_positions
        add_agent!(bp, Daisy, model, :black, rand(abmrng(model),
         ↪  0:max_age), albedo_black)
    end

    for p in positions(model)
        update_surface_temperature!(p, model)
    end

    return model
end
```

3.2.4 Базовая визуализация

Сделаем базовую визуализацию (рис. 3.1). Построим тепловую карту, которая будет представлять собой температуру поверхности модели. Маргаритки будут отображаться черно-белыми в соответствии с их видом.

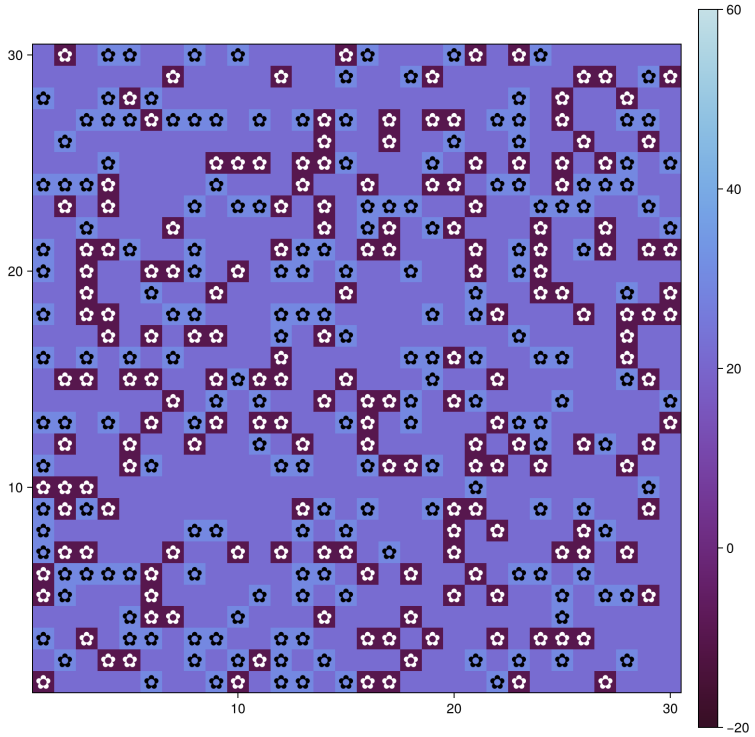


Рисунок 3.1. Пример визуализации

- Создайте файл `scripts/daisyworld.jl`.
- Запустите его.
- Проанализируйте результат.
- Преобразуйте код в литературный стиль.
- Сгенерируйте из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполните код из jupyter notebook.
- Интегрируйте документацию в формате Quarto в отчёт.

```
using DrWatson
@quickactivate "project"
using Agents
using DataFrames
using Plots


include(srcdir("daisyworld.jl"))

using CairoMakie
model = daisyworld()

daisycolor(a::Daisy) = a.breed

plotkwargs = (
    agent_color=daisycolor, agent_size = 20, agent_marker = '✿',
    heatarray = :temperature,
    heatkwargs = (colorrange = (-20, 60),),
)
plt1, _ = abmplot(model; plotkwargs...)

step!(model, 5)
plt2, _ = abmplot(model; heatarray = model.temperature,
 ↪  plotkwargs...)

step!(model, 40)
plt3, _ = abmplot(model; heatarray = model.temperature,
 ↪  plotkwargs...)


save(plotsdir("daisy_step001.png"), plt1)
save(plotsdir("daisy_step005.png"), plt2)
save(plotsdir("daisy_step040.png"), plt3)
```

3.2.5 Анимация модели

Создадим визуализацию не отдельных моментов, а видео эволюции модели.

— Создайте файл `scripts/daisyworld-animate.jl`.
— Запустите его.

— Проанализируйте результат.
— Поместите результирующий файл на видеохостинги.

```
using DrWatson
@quickactivate "project"
using Agents
using DataFrames
using Plots

include(srcdir("daisyworld.jl"))

using CairoMakie
model = daisyworld()

daisycolor(a::Daisy) = a.breed

plotkwargs = (
    agent_color=daisycolor, agent_size = 20, agent_marker = '✿',
    heatarray = :temperature,
    heatkwargs = (colorrange = (-20, 60),),
)

abmvideo(
    plotsdir("simulation.mp4"),
    model;
    title = "Daisy World",
    frames = 60,
    plotkwargs...,
)
```

3.2.6 Динамика числа маргариток

Построим график изменения числа маргариток в зависимости от модельного времени (рис. 3.2).

— Создайте файл `scripts/daisyworld-count.jl`.
— Запустите его.
— Проанализируйте результат.
— Преобразуйте код в литературный стиль.
— Сгенерируйте из литературного кода:

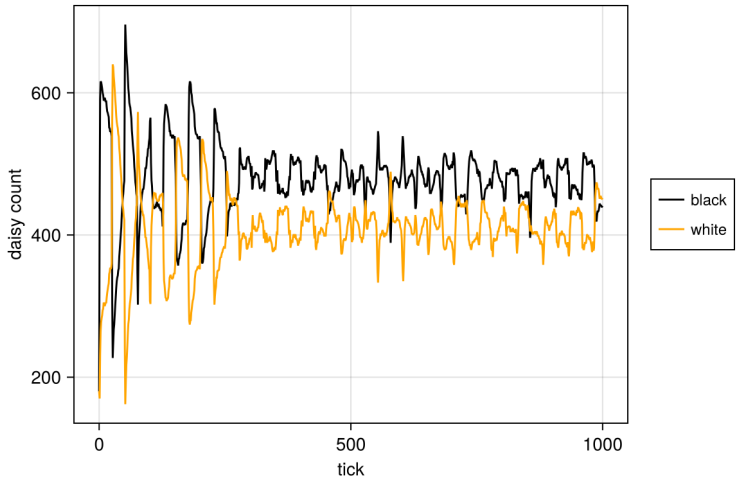


Рисунок 3.2. График изменения числа маргариток

  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.

- Выполните код из jupyter notebook.
- Интегрируйте документацию в формате Quarto в отчёт.

```
using DrWatson
@quickactivate "project"
using Agents
using DataFrames
using Plots


include(srcdir("daisyworld.jl"))

using CairoMakie

black(a) = a.breed == :black
white(a) = a.breed == :white
adata = [(black, count), (white, count)]

model = daisyworld(; solar_luminosity = 1.0)

agent_df, model_df = run!(model, 1000; adata)
figure = Figure(size = (600, 400));
```

```
ax = figure[1, 1] = Axis(figure, xlabel = "tick", ylabel = "daisy
 ↪  count")
blackl = lines!(ax, agent_df[!, :time], agent_df[!, :count_black],
 ↪  color = :black)
whitel = lines!(ax, agent_df[!, :time], agent_df[!, :count_white],
 ↪  color = :orange)
Legend(figure[1, 2], [blackl, whitel], ["black", "white"], labelsize
 ↪  = 12)
# figure

save(plotsdir("daisy_count.png"), figure)
```

3.2.7 Динамика модели

Построим комплексный график изменения числа маргариток, температуры, альбедо в зависимости от модельного времени (рис. 3.3).

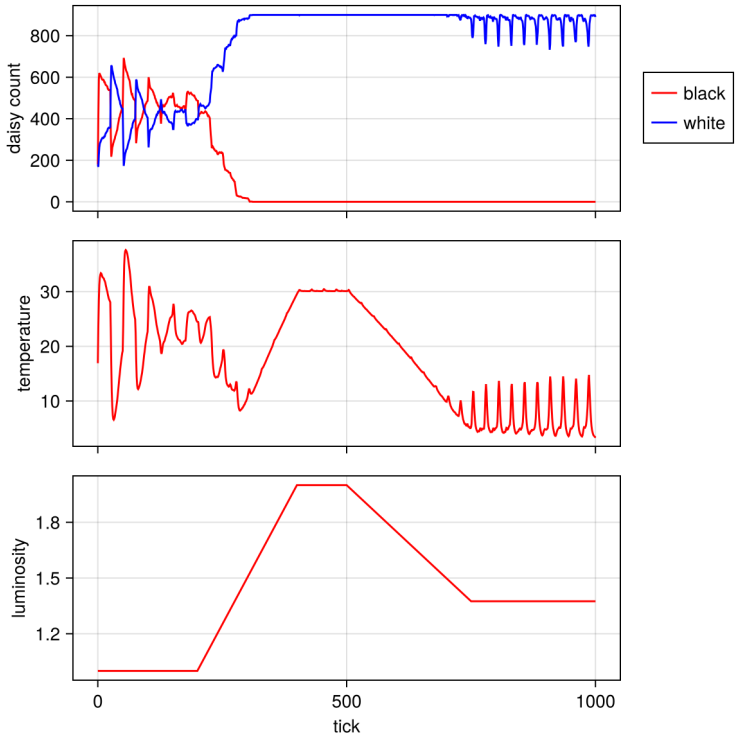


Рисунок 3.3. Динамика модели

— Создайте файл `scripts/daisyworld-luminosity.jl`.

- Запустите его.
- Проанализируйте результат.
- Преобразуйте код в литературный стиль.
- Сгенерируйте из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполните код из jupyter notebook.
- Интегрируйте документацию в формате Quarto в отчёт.

```
using DrWatson
@quickactivate "project"
using Agents
using DataFrames
using Plots


include(srcdir("daisyworld.jl"))

using CairoMakie

black(a) = a.breed == :black
white(a) = a.breed == :white
adata = [(black, count), (white, count)]

model = daisyworld(solar_luminosity = 1.0, scenario = :ramp)

temperature(model) = StatsBase.mean(model.temperature)
mdata = [temperature, :solar_luminosity]

agent_df, model_df = run!(model, 1000; adata = adata, mdata = mdata)

figure = CairoMakie.Figure(size = (600, 600));
ax1 = figure[1, 1] = Axis(figure, ylabel = "daisy count")
blackl = lines!(ax1, agent_df[!, :time], agent_df[!, :count_black],
 ↪  color = :red)
whitel = lines!(ax1, agent_df[!, :time], agent_df[!, :count_white],
 ↪  color = :blue)
figure[1, 2] = Legend(figure, [blackl, whitel], ["black", "white"])

ax2 = figure[2, 1] = Axis(figure, ylabel = "temperature")
ax3 = figure[3, 1] = Axis(figure, xlabel = "tick", ylabel =
 ↪  "luminosity")
```

```
lines!(ax2, model_df[!, :time], model_df[!, :temperature], color =
 ↪  :red)
lines!(ax3, model_df[!, :time], model_df[!, :solar_luminosity],
 ↪  color = :red)
for ax in (ax1, ax2); ax.xticklabelsvisible = false; end
figure


save(plotsdir("daisy_luminosity.png"), figure)
```

3.2.8 Базовая визуализация (параметры)

Расширим базовую визуализацию за счёт параметров.

- Создайте файл `scripts/daisyworld__param.jl`.
- Запустите его.
- Проанализируйте результат.
- Преобразуйте код в литературный стиль.
- Сгенерируйте из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполните код из jupyter notebook.
- Интегрируйте документацию в формате Quarto в отчёт.

```
using DrWatson
@quickactivate "project"
using Agents
using DataFrames
using Plots
using CairoMakie

include(srcdir("daisyworld.jl"))

## Параметры эксперимента
param_dict = Dict(
    :griddims => (30, 30),
    :max_age => [25, 40],
    :init_white => [0.2, 0.8],
```

```
    :init_black => 0.2,
    :albedo_white => 0.75,
    :albedo_black => 0.25,
    :surface_albedo => 0.4,
    :solar_change => 0.005,
    :solar_luminosity => 1.0,
    :scenario => :default,
    :seed => 165,
)

## Создаём список всех комбинаций
params_list = dict_list(param_dict)

for params in params_list

    model = daisyworld(;params...)

    daisycolor(a::Daisy) = a.breed

    plotkwargs = (
        agent_color=daisycolor,
        agent_size = 20,
        agent_marker = '☆',
        heatarray = :temperature,
        heatkwargs = (colorrange = (-20, 60),),
    )
    plt1, _ = abmplot(model; plotkwargs...)

    step!(model, 5)
    plt2, _ = abmplot(model; heatarray = model.temperature,
     ↪  plotkwargs...)

    step!(model, 40)
    plt3, _ = abmplot(model; heatarray = model.temperature,
     ↪  plotkwargs...)

    plt1_name = savename("daisyworld",params) * "_step01" * ".png"
    plt2_name = savename("daisyworld",params) * "_step05" * ".png"
    plt3_name = savename("daisyworld",params) * "_step40" * ".png"

    save(plotsdir(plt1_name), plt1)
    save(plotsdir(plt2_name), plt2)
    save(plotsdir(plt3_name), plt3)
```

```
end
```

3.2.9 Динамика числа маргариток (параметры)

Построим график изменения числа маргариток в зависимости от модельного времени с разными параметрами модели.

- Создайте файл `scripts/daisyworld-count__param.jl`.
- Запустите его.
- Проанализируйте результат.
- Преобразуйте код в литературный стиль.
- Сгенерируйте из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполните код из jupyter notebook.
- Интегрируйте документацию в формате Quarto в отчёт.

```
using DrWatson
@quickactivate "project"
using Agents
using DataFrames
using Plots
using CairoMakie

include(srcdir("daisyworld.jl"))


black(a) = a.breed == :black
white(a) = a.breed == :white
adata = [(black, count), (white, count)]

## Параметры эксперимента
param_dict = Dict(
    :griddims => (30, 30),
    :max_age => [25, 40],
    :init_white => [0.2, 0.8],
    :init_black => 0.2,
    :albedo_white => 0.75,
```

```
    :albedo_black => 0.25,
    :surface_albedo => 0.4,
    :solar_change => 0.005,
    :solar_luminosity => 1.0,
    :scenario => :default,
    :seed => 165,
)

## Создаём список всех комбинаций
params_list = dict_list(param_dict)

for params in params_list

    model = daisyworld(;params...)

    agent_df, model_df = run!(model, 1000; adata)
    figure = Figure(size = (600, 400));
    ax = figure[1, 1] = Axis(figure, xlabel = "tick", ylabel = "daisy
     ↪  count")
    blackl = lines!(ax, agent_df[!, :time], agent_df[!,
     ↪  :count_black], color = :black)
    whitel = lines!(ax, agent_df[!, :time], agent_df[!,
     ↪  :count_white], color = :orange)
    Legend(figure[1, 2], [blackl, whitel], ["black", "white"],
     ↪  labelsize = 12)

    plt_name = savename("daisy-count",params) * ".png"

    save(plotsdir(plt_name), figure)

end
```

3.2.10 Динамика модели (параметры)

Построим комплексный график изменения числа маргариток, температуры, альбедо в зависимости от модельного времени с разными параметрами модели.

— Создайте файл `scripts/daisyworld-luminosity__param.jl`.
— Запустите его.
— Проанализируйте результат.

- Преобразуйте код в литературный стиль.
- Сгенерируйте из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполните код из jupyter notebook.
- Интегрируйте документацию в формате Quarto в отчёт.

```
using DrWatson
@quickactivate "project"
using Agents
using DataFrames
using Plots
using CairoMakie

include(srcdir("daisyworld.jl"))


black(a) = a.breed == :black
white(a) = a.breed == :white
adata = [(black, count), (white, count)]

## Параметры эксперимента
param_dict = Dict(
    :griddims => (30, 30),
    :max_age => [25, 40],
    :init_white => [0.2, 0.8],
    :init_black => 0.2,
    :albedo_white => 0.75,
    :albedo_black => 0.25,
    :surface_albedo => 0.4,
    :solar_change => 0.005,
    :solar_luminosity => 1.0,
    :scenario => :ramp,
    :seed => 165,
)

## Создаём список всех комбинаций
params_list = dict_list(param_dict)

for params in params_list

    model = daisyworld(;params...)
```

```
    temperature(model) = StatsBase.mean(model.temperature)
    mdata = [temperature, :solar_luminosity]

    agent_df, model_df = run!(model, 1000; adata = adata, mdata =
     ↪  mdata)

    figure = CairoMakie.Figure(size = (600, 600));
    ax1 = figure[1, 1] = Axis(figure, ylabel = "daisy count")
    blackl = lines!(ax1, agent_df[!, :time], agent_df[!,
     ↪  :count_black], color = :red)
    whitel = lines!(ax1, agent_df[!, :time], agent_df[!,
     ↪  :count_white], color = :blue)
    figure[1, 2] = Legend(figure, [blackl, whitel], ["black",
     ↪  "white"])

    ax2 = figure[2, 1] = Axis(figure, ylabel = "temperature")
    ax3 = figure[3, 1] = Axis(figure, xlabel = "tick", ylabel =
     ↪  "luminosity")
    lines!(ax2, model_df[!, :time], model_df[!, :temperature], color
     ↪  = :red)
    lines!(ax3, model_df[!, :time], model_df[!, :solar_luminosity],
     ↪  color = :red)
    for ax in (ax1, ax2); ax.xticklabelsvisible = false; end


    plt_name = savename("daisy-luminosity",params) * ".png"
    save(plotsdir(plt_name), figure)

end
```

3.3 Задание

- Создать рабочий каталог для кода.
- Установить необходимые пакеты.
- Выполнить предложенный код.
- Преобразовать код в литературный стиль.
- Сгенерировать из литературного кода:
  - чистый код;
  - jupyter notebook;

  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.
- Результирующие файлы не удаляйте, выложите на git.

4

Реализация основных моделей в агентном подходе

4.1 Эпидемиологическая модели SIR

Создадим агентную модель распространения инфекционного заболевания на основе классической компартментальной модели SIR (Susceptible-Infectious-Recovered). Модель будет реализована с использованием пакета `Agents.jl`. В отличие от классической модели на дифференциальных уравнениях, агентный подход позволит учесть индивидуальные характеристики, пространственную структуру и стохастичность процессов.

4.1.1 Теоретическое введение

4.1.1.1 Классическая модель SIR

Модель SIR, предложенная Кермаком и Маккендриком в 1927 году, описывает динамику эпидемии в популяции, разделённой на три группы:

— S (Susceptible) — восприимчивые к заболеванию индивиды;
— I (Infectious) — инфицированные, способные заражать восприимчивых;
— R (Recovered) — выздоровевшие (или умершие), получившие иммунитет и более не участвующие в распространении.

Классическая модель описывается системой обыкновенных дифференциальных уравнений:

$$dS/dt = -\beta SI/N,$$
$$dI/dt = \beta SI/N - \gamma I,$$
$$dR/dt = \gamma I.$$

где β — коэффициент передачи инфекции, γ — скорость выздоровления.

4.1.1.2 Ограничения классического подхода

Несмотря на широкое применение, модель на ОДУ имеет ряд ограничений:

- Однородность популяции — все индивиды считаются одинаковыми.
- Отсутствие пространственной структуры — предполагается полное перемешивание.
- Детерминированность — не учитываются случайные флуктуации.
- Непрерывность — количество людей рассматривается как непрерывная величина.

4.1.1.3 Преимущества агентного подхода

Агентное моделирование позволяет преодолеть эти ограничения:

- Каждый индивид моделируется отдельно с уникальными характеристиками.
- Взаимодействия происходят локально в пространстве или социальной сети.
- Процессы носят стохастический характер.
- Можно учитывать гетерогенность контактов, мобильность, меры контроля.

4.1.2 Описание модели

4.1.2.1 Агенты

Каждый агент — человек, обладающий следующими свойствами:

- `status :: Symbol` — состояние (:S, :I, :R);
- `days_infected :: Int` — количество дней с момента заражения (для инфицированных).

4.1.2.2 Пространство состояний

Агенты размещаются на графе, где узлы представляют локации (города, районы), а рёбра — возможные перемещения между ними. Такой подход позволяет моделировать метапопуляционную динамику и миграцию.

4.1.2.3 Параметры модели

- `Ns` — вектор численности населения по локациям;
- `β_und` — вероятность передачи для невыявленных случаев;
- `β_det` — вероятность передачи для выявленных случаев;
- `infection_period` — длительность заболевания (дней);
- `detection_time` — время до выявления заболевания;
- `death_rate` — вероятность летального исхода;
- `reinfection_probability` — вероятность повторного заражения;
- `migration_rates` — матрица вероятностей миграции между локациями.

4.1.2.4 Динамика

На каждом шаге (соответствует одному дню) для каждого агента выполняются следующие процессы :

- Миграция — агент может переместиться в другую локацию с вероятностью, заданной матрицей `migration_rates`.
- Передача инфекции — если агент инфицирован, он может заразить восприимчивых агентов в той же локации. Количество заражённых за день определяется пуассоновским процессом с интенсивностью, зависящей от статуса выявления (β_und или β_det).
- Обновление счётчика — для инфицированных увеличивается `days_infected`.
- Выздоровление или смерть — если `days_infected` достиг `infection_period`, агент либо выздоравливает (переходит в R), либо умирает (удаляется из модели) с вероятностью `death_rate`. Выздоровевшие могут быть повторно инфицированы с вероятностью `reinfection_probability`.

4.1.3 Код модели

- `src/sir_model.jl`

```
using Agents, Random
using Agents.Graphs
using StatsBase: sample, Weights
using Distributions: Poisson

using DrWatson: @dict

# Определение типа агента
@agent struct Person(GraphAgent)
    days_infected::Int  # количество дней с момента заражения
    status::Symbol      # :S, :I, :R
end

# Функция инициализации модели
function initialize_sir(;
    Ns = [1000, 1000, 1000],        # численность по трём городам
    migration_rates = nothing,       # матрица миграции
    β_und = [0.5, 0.5, 0.5],        # передача невыявленными
    β_det = [0.05, 0.05, 0.05],      # передача выявленными
    infection_period = 14,
    detection_time = 7,
    death_rate = 0.02,
    reinfection_probability = 0.1,
    Is = [0, 0, 1],                  # начальное количество
     ↪  инфицированных
    seed = 42,
    n_steps = 100,
)
    rng = Xoshiro(seed)
    C = length(Ns)  # количество городов

    # Если матрица миграции не задана, создаём реалистичную
    if migration_rates === nothing
        migration_rates = zeros(C, C)
        for i = 1:C
            for j = 1:C
                # Вероятность миграции пропорциональна размеру
                 ↪  целевого города
                migration_rates[i, j] = (Ns[i] + Ns[j]) / Ns[i]
            end
        end
        # Нормализация
        for i = 1:C
```

```
        migration_rates[i, :] ./= sum(migration_rates[i, :])
    end
end

properties = @dict(
    Ns,
    β_und,
    β_det,
    migration_rates,
    infection_period,
    detection_time,
    death_rate,
    reinfection_probability,
    C
)

# Пространство — полный граф (все города связаны)
space = GraphSpace(complete_graph(C))
model = StandardABM(Person, space; properties, rng,
 ↪  (agent_step!) = (sir_agent_step!))

# Заполняем города агентами
for city = 1:C
    for _ = 1:Ns[city]
        add_agent!(city, model, 0, :S)
    end
end

# Инфицируем начальных носителей
for city = 1:C
    if Is[city] > 0
        # Получаем список агентов в городе
        city_agents = ids_in_position(city, model)
        # Выбираем случайных для инфицирования
        infected_ids = sample(rng, city_agents, Is[city];
         ↪  replace = false)
        for id in infected_ids
            agent = model[id]
            agent.status = :I
            agent.days_infected = 1
        end
    end
end
```

```
    return model
end

# Шаг агента
function sir_agent_step!(agent, model)
    # Миграция
    migrate!(agent, model)

    # Передача инфекции (только если агент инфицирован)
    if agent.status == :I
        transmit!(agent, model)
    end

    # Обновление счётчика дней для инфицированных
    if agent.status == :I
        agent.days_infected += 1
    end

    # Выздоровление или смерть
    recover_or_die!(agent, model)
end

# Миграция между городами
function migrate!(agent, model)
    current_city = agent.pos
    # Выбираем целевой город согласно вероятностям
    probs = model.migration_rates[current_city, :]
    target = sample(abmrng(model), 1:model.C, Weights(probs))

    if target ≠ current_city
        move_agent!(agent, target, model)
    end
end

# Передача инфекции
function transmit!(agent, model)
    # Определяем интенсивность заражения в зависимости от стадии
    rate = if agent.days_infected < model.detection_time
        model.β_und[agent.pos]  # невыявленный — высокая заразность
    else
        model.β_det[agent.pos]  # выявленный — низкая заразность
    end

    # Количество заражённых за день — пуассоновский процесс
```

```
    n_infections = rand(abmrng(model), Poisson(rate))
    n_infections == 0 && return nothing

    # Получаем всех агентов в той же локации, кроме самого заразного
    neighbors = [a for a in agents_in_position(agent.pos, model) if
     ↪  a.id != agent.id]

    # Перемешиваем для случайного порядка
    shuffle!(abmrng(model), neighbors)

    for contact in neighbors
        # Проверяем возможность заражения
        if contact.status == :S
            contact.status = :I
            contact.days_infected = 1
            n_infections -= 1
            n_infections == 0 && return nothing
        elseif contact.status == :R && rand(abmrng(model)) ≤
         ↪  model.reinfection_probability
            contact.status = :I
            contact.days_infected = 1
            n_infections -= 1
            n_infections == 0 && return nothing
        end
    end
end

# Выздоровление или смерть
function recover_or_die!(agent, model)
    if agent.status == :I && agent.days_infected ≥
     ↪  model.infection_period
        if rand(abmrng(model)) ≤ model.death_rate
            remove_agent!(agent, model)  # смерть (исправлено)
        else
            agent.status = :R  # выздоровление
            agent.days_infected = 0
        end
    end
end

# Вспомогательные функции для сбора данных
infected_count(model) = count(a.status == :I for a in
 ↪  allagents(model))
```

```
recovered_count(model) = count(a.status == :R for a in
 ↪  allagents(model))
susceptible_count(model) = count(a.status == :S for a in
 ↪  allagents(model))
total_count(model) = nagents(model)
```

4.1.4 Базовый эксперимент

— `scripts/sir_run_basic.jl`

4.1.4.1 Что делает

Запускает один эксперимент с фиксированными параметрами (по умолчанию) и сохраняет динамику численности агентов. Это служит для проверки работоспособности модели и получения базового понимания эпидемического процесса.

4.1.4.2 Входные параметры

Параметры заданы в коде в виде словаря `params`:

— `Ns = [1000, 1000, 1000]` — численность населения в трёх городах.
— `β_und = [0.5, 0.5, 0.5]` — интенсивность заражения невыявленными больными.
— `β_det = [0.05, 0.05, 0.05]` — интенсивность заражения выявленными больными.
— `infection_period = 14` — длительность болезни (дней).
— `detection_time = 7` — время до выявления после заражения.
— `death_rate = 0.02` — вероятность смерти при завершении болезни.
— `reinfection_probability = 0.1` — вероятность повторного заражения.
— `Is = [0, 0, 1]` — начальное количество инфицированных в каждом городе (только в третьем).
— `seed = 42` — зерно генератора случайных чисел.
— `n_steps = 100` — количество дней симуляции.

4.1.4.3 Что делает внутри

- Инициализирует модель с помощью `initialize_sir(; params...)`.
- Вручную выполняет цикл на `n_steps` шагов, применяя на каждом шаге функцию `sir_agent_step!` ко всем живым агентам (без использования `step!` для совместимости).
- После каждого шага собирает статистику: количество восприимчивых, инфицированных, выздоровевших и общую численность (с учётом умерших).
- Сохраняет данные в массивы, затем создаёт DataFrame и сохраняет его в файлы `.jld2` (через `wsave`) в папке `data/`.
- Строит график динамики и сохраняет его в `plots/sir_basic_dynamics.png`.

4.1.4.4 Результаты

- Файлы данных:
  - `data/sir_basic_agent.jld2` — DataFrame с колонками `time`, `susceptible`, `infected`, `recovered`.
  - `data/sir_basic_model.jld2` — DataFrame с колонками `time`, `total`.
- График: `plots/sir_basic_dynamics.png` — четыре линии: S(t), I(t), R(t) и общая численность (пунктир) (рис. 4.1). Позволяет визуально оценить пик эпидемии, скорость распространения и влияние смертности.

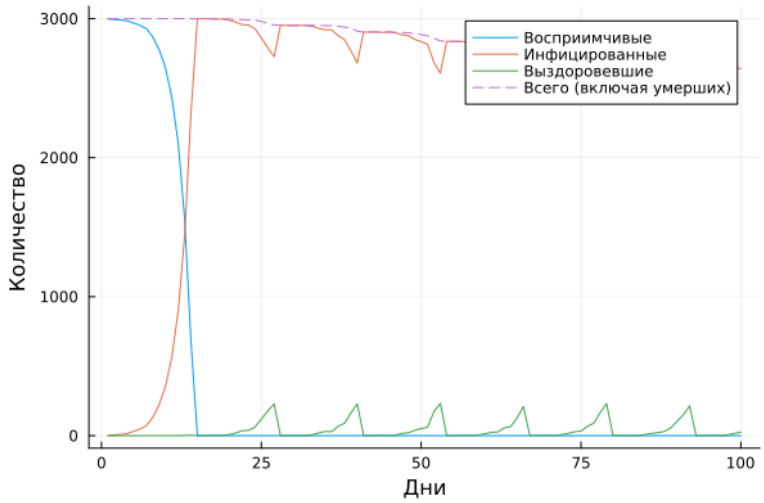


Рисунок 4.1. График динамики SIR

```
using DrWatson
@quickactivate "project"
```

```julia
using Agents, DataFrames, Plots
using JLD2

include(srcdir("sir_model.jl"))

# Параметры эксперимента
params = Dict(
    :Ns => [1000, 1000, 1000],
    :β_und => [0.5, 0.5, 0.5],
    :β_det => [0.05, 0.05, 0.05],
    :infection_period => 14,
    :detection_time => 7,
    :death_rate => 0.02,
    :reinfection_probability => 0.1,
    :Is => [0, 0, 1],
    :seed => 42,
    :n_steps => 100,
)

# Инициализация модели
model = initialize_sir(; params...)

# Подготовка массивов для хранения данных
times = Int[]
S_vals = Int[]
I_vals = Int[]
R_vals = Int[]
total_vals = Int[]

# Запуск симуляции вручную
for step = 1:params[:n_steps]
    Agents.step!(model, 1)

    push!(times, step)
    push!(S_vals, susceptible_count(model))
    push!(I_vals, infected_count(model))
    push!(R_vals, recovered_count(model))
    push!(total_vals, total_count(model))
end

# Создаём DataFrame для удобства (опционально)
agent_df =
    DataFrame(time = times, susceptible = S_vals, infected = I_vals,
     ↪  recovered = R_vals)
```

```
model_df = DataFrame(time = times, total = total_vals)

# Визуализация
plot(
    agent_df.time,
    agent_df.susceptible,
    label = "Восприимчивые",
    xlabel = "Дни",
    ylabel = "Количество",
)
plot!(agent_df.time, agent_df.infected, label = "Инфицированные")
plot!(agent_df.time, agent_df.recovered, label = "Выздоровевшие")
plot!(agent_df.time, model_df.total, label = "Всего (включая
 ↪  умерших)", linestyle = :dash)
savefig(plotsdir("sir_basic_dynamics.png"))

# Сохранение данных
@save datadir("sir_basic_agent.jld2") agent_df
@save datadir("sir_basic_model.jld2") model_df
```

4.1.5 Сканирование коэффициента заразности

— `scripts/sir_scan_beta.jl`

4.1.5.1 Что делает

Исследует, как изменение базовой заразности (β_und и пропорционально β_det) влияет на эпидемические показатели: пик заболеваемости, долю переболевших, число умерших. Выполняется параметрическое сканирование с несколькими повторными прогонами для учёта стохастичности.

4.1.5.2 Входные параметры

— Диапазон β_und: от 0.1 до 1.0 с шагом 0.1.
— Для каждого значения β_und делается 3 прогона с разными seed (42, 43, 44).
— Остальные параметры фиксированы (аналогично базовому скрипту).

4.1.5.3 Что делает внутри

- Для каждой комбинации (β, seed) создаётся словарь параметров.
- Запускается функция `run_experiment(p)`, которая:
  - инициализирует модель с данными параметрами;
  - вручную выполняет симуляцию на 100 шагов;
  - отслеживает пиковую долю инфицированных, финальную долю инфицированных и выздоровевших, общее число умерших.
- Все результаты собираются в список, преобразуются в DataFrame и сохраняются в CSV-файл.
- Затем данные усредняются по повторным прогонам для каждого значения β и строятся графики.

4.1.5.4 Результаты

- CSV-файл: `data/beta_scan_all.csv` — все прогоны с колонками: `β_und`, `seed`, `peak`, `final_inf`, `final_rec`, `deaths`.
- График: `plots/beta_scan.png` (рис. 4.2) показывает зависимость от β:
  - средняя пиковая доля инфицированных;
  - средняя конечная доля инфицированных;
  - средняя доля умерших (нормированная на численность).

График позволяет найти пороговое значение β, при котором возникает эпидемия, и оценить, как увеличивается нагрузка на систему с ростом заразности.

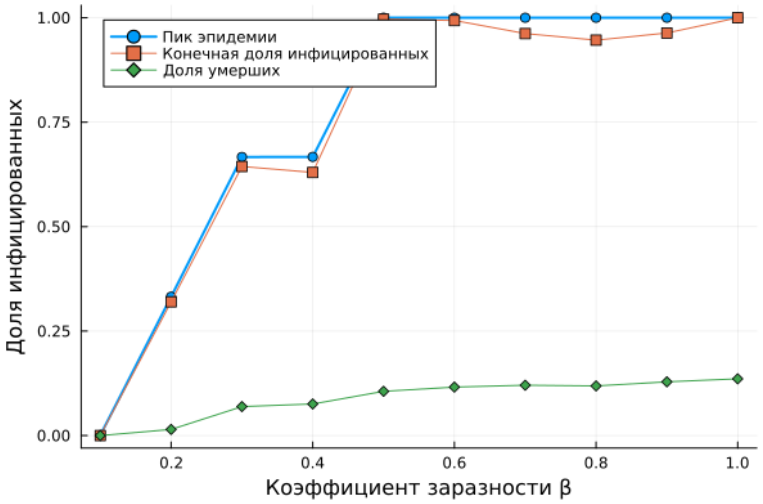


Рисунок 4.2. График динамики SIR

```julia
# scripts/scan_beta.jl
using DrWatson
@quickactivate "project"

using Agents, DataFrames, Plots, CSV, Random

include(srcdir("sir_model.jl"))

# Функция для запуска одного эксперимента и возврата метрик
function run_experiment(p)
    # Создаём β_und и β_det на основе скалярного beta
    beta = p[:beta]
    β_und = fill(beta, 3)
    β_det = fill(beta/10, 3)
    # Передаём в модель
    model = initialize_sir(;
        Ns = p[:Ns],
        β_und = β_und,
        β_det = β_det,
        infection_period = p[:infection_period],
        detection_time = p[:detection_time],
        death_rate = p[:death_rate],
        reinfection_probability = p[:reinfection_probability],
        Is = p[:Is],
        seed = p[:seed],
        n_steps = p[:n_steps],  # этот параметр не используется в
         ↪ initialize_sir, но может быть нужен для цикла
    )

    infected_fraction(model) =
        count(a.status == :I for a in allagents(model)) /
         ↪ nagents(model)
    peak_infected = 0.0

    for step = 1:p[:n_steps]
        # Ручной шаг (безопасный)
        agent_ids = collect(allids(model))
        for id in agent_ids
            agent = try
                model[id]
            catch
                nothing
            end
```

```
            if agent !== nothing
                sir_agent_step!(agent, model)
            end
        end

        frac = infected_fraction(model)
        if frac > peak_infected
            peak_infected = frac
        end
    end

    final_infected = infected_fraction(model)
    final_recovered = count(a.status == :R for a in
     ↪  allagents(model)) / nagents(model)
    total_deaths = sum(p[:Ns]) - nagents(model)

    return (
        peak = peak_infected,
        final_inf = final_infected,
        final_rec = final_recovered,
        deaths = total_deaths,
    )
end

# Диапазон значений beta (скаляр)
beta_range = 0.1:0.1:1.0
seeds = [42, 43, 44]

# Создаём список параметров
params_list = []
for b in beta_range
    for s in seeds
        push!(
            params_list,
            Dict(
                :beta => b,  # скалярное значение
                :Ns => [1000, 1000, 1000],
                :infection_period => 14,
                :detection_time => 7,
                :death_rate => 0.02,
                :reinfection_probability => 0.1,
                :Is => [0, 0, 1],
                :seed => s,
                :n_steps => 100,
```

```
            ),
        )
    end
end

# Запуск экспериментов
results = []
for params in params_list
    data = run_experiment(params)
    push!(results, merge(params, Dict(pairs(data))))
    println("Завершён эксперимент с beta = $(params[:beta]), seed =
     ↪  $(params[:seed])")
end

# Сохраняем все прогоны
df = DataFrame(results)
CSV.write(datadir("beta_scan_all.csv"), df)

# Усреднение по повторам
using Statistics
grouped = combine(
    groupby(df, [:beta]),
    :peak => mean => :mean_peak,
    :final_inf => mean => :mean_final_inf,
    :deaths => mean => :mean_deaths,
)

# График
plot(
    grouped.beta,
    grouped.mean_peak,
    label = "Пик эпидемии",
    xlabel = "Коэффициент заразности β",
    ylabel = "Доля инфицированных",
    marker = :circle,
    linewidth = 2,
)
plot!(
    grouped.beta,
    grouped.mean_final_inf,
    label = "Конечная доля инфицированных",
    marker = :square,
)
```

```
plot!(grouped.beta, grouped.mean_deaths ./ 3000, label = "Доля
 ↪  умерших", marker = :diamond)
savefig(plotsdir("beta_scan.png"))

println("Результаты сохранены в data/beta_scan_all.csv и
 ↪  plots/beta_scan.png")
```

4.1.5.5 Исследование эффекта миграции

— `scripts/sir_migration_effect.jl`

4.1.5.6 Что делает

Исследует, как интенсивность перемещения людей между городами влияет на скорость распространения эпидемии (время достижения пика) и масштаб пика. Инфекция начинается только в одном городе, остальные изначально здоровы.

4.1.5.7 Входные параметры

— Интенсивность миграции `migration_intensity` от 0 до 0.5 с шагом 0.1.
— Для каждого значения интенсивности создаётся матрица миграции: вероятность остаться в городе = `1 - intensity`, вероятность переехать в любой другой город = `intensity/(C-1)`.
— Каждое значение интенсивности прогоняется с тремя разными `seed` (42, 43, 44).
— Остальные параметры фиксированы (включая `β_und` = 0.5, `β_det` = 0.05, начальная инфекция только в городе 1).

4.1.5.8 Что делает внутри

— Для каждой комбинации (intensity, seed) создаётся словарь параметров с соответствующей матрицей миграции.
— Запускается функция `peak_time(p)`, которая:
  — инициализирует модель;
  — выполняет симуляцию на 150 шагов;

  - запоминает шаг, на котором доля инфицированных была максимальной (время пика), и значение пика.
- Результаты сохраняются в CSV, затем усредняются по повторам для каждого значения интенсивности.
- Строится график зависимости времени пика и величины пика от интенсивности миграции.

4.1.5.9 Результаты

- CSV-файл: `data/migration_scan_all.csv`. Все прогоны с колонками: `migration_intensity`, `seed`, `peak_time`, `peak_value`.
- График: `plots/migration_effect.png` (рис. 4.3). Два подграфика (или один с двумя осями Y), показывающие:
  - время достижения пика (в днях) vs интенсивность миграции;
  - пиковую численность инфицированных vs интенсивность миграции.

График демонстрирует, как ускорение обмена людьми между городами приводит к более раннему и более высокому пику эпидемии.

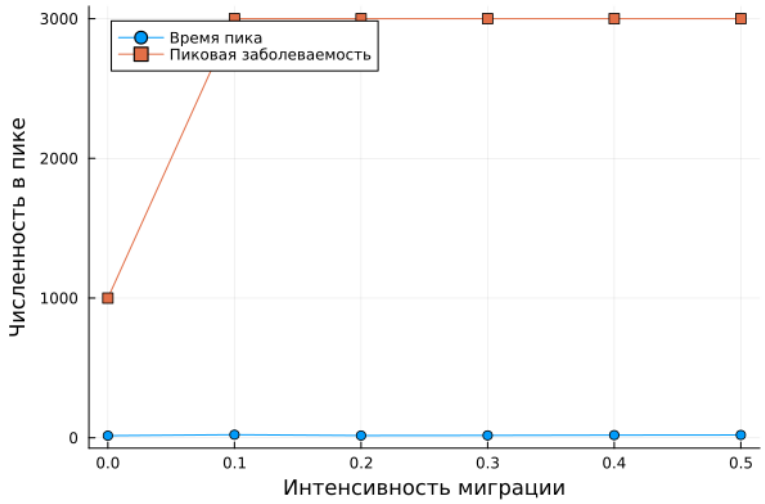


Рисунок 4.3. График динамики миграции

```
using DrWatson
@quickactivate "project"
using Agents, DataFrames, Plots, CSV, Random

include(srcdir("sir_model.jl"))
```

```
# Функция для создания матрицы миграции с заданной интенсивностью
function create_migration_matrix(C, intensity)
    M = ones(C, C) .* intensity ./ (C-1)
    for i = 1:C
        M[i, i] = 1 - intensity
    end
    return M
end

# Функция для измерения времени достижения пика
function peak_time(p)
    # Создаём матрицу миграции на основе интенсивности
    migration_rates = create_migration_matrix(p[:C],
     ↪  p[:migration_intensity])
    model = initialize_sir(;
        Ns = p[:Ns],
        β_und = p[:β_und],
        β_det = p[:β_det],
        infection_period = p[:infection_period],
        detection_time = p[:detection_time],
        death_rate = p[:death_rate],
        reinfection_probability = p[:reinfection_probability],
        Is = p[:Is],
        seed = p[:seed],
        migration_rates = migration_rates,
    )

    infected_frac(model) = count(a.status == :I for a in
     ↪  allagents(model)) / nagents(model)
    peak = 0.0
    peak_step = 0

    for step = 1:p[:n_steps]
        # Ручной шаг
        agent_ids = collect(allids(model))
        for id in agent_ids
            agent = try
                model[id]
            catch
                nothing
            end
            if agent !== nothing
                sir_agent_step!(agent, model)
            end
```

```
        end

        frac = infected_frac(model)
        if frac > peak
            peak = frac
            peak_step = step
        end
    end

    return (peak_time = peak_step, peak_value = peak)
end

# Сканирование интенсивности миграции
migration_intensities = 0.0:0.1:0.5
seeds = [42, 43, 44]

params_list = []
for mig in migration_intensities
    for s in seeds
        push!(
            params_list,
            Dict(
                :migration_intensity => mig,  # скаляр
                :C => 3,
                :Ns => [1000, 1000, 1000],
                :β_und => [0.5, 0.5, 0.5],
                :β_det => [0.05, 0.05, 0.05],
                :infection_period => 14,
                :detection_time => 7,
                :death_rate => 0.02,
                :reinfection_probability => 0.1,
                :Is => [1, 0, 0],
                :seed => s,
                :n_steps => 150,
            ),
        )
    end
end

# Запуск
results = []
for params in params_list
    data = peak_time(params)
    push!(results, merge(params, Dict(pairs(data))))
```

```
    println(
        "Завершён эксперимент с migration_intensity =
         ↪ $(params[:migration_intensity]), seed =
         ↪ $(params[:seed])",
    )
end

# Сохраняем все прогоны
df = DataFrame(results)
CSV.write(datadir("migration_scan_all.csv"), df)

# Усреднение по повторам
using Statistics
grouped = combine(
    groupby(df, [:migration_intensity]),
    :peak_time => mean => :mean_peak_time,
    :peak_value => mean => :mean_peak_value,
)

# Визуализация
plot(
    grouped.migration_intensity,
    grouped.mean_peak_time,
    marker = :circle,
    xlabel = "Интенсивность миграции",
    ylabel = "Время до пика (дни)",
    label = "Время пика",
)
plot!(
    grouped.migration_intensity,
    grouped.mean_peak_value .* 3000,
    marker = :square,
    xlabel = "Интенсивность миграции",
    ylabel = "Численность в пике",
    label = "Пиковая заболеваемость",
)
savefig(plotsdir("migration_effect.png"))

println("Результаты сохранены в data/migration_scan_all.csv и
 ↪ plots/migration_effect.png")
```

4.1.6 Многокритериальная оптимизация параметров

— `scripts/sir_optimize_parameters.jl`

4.1.6.1 Что делает

Ищет оптимальные комбинации параметров, минимизирующие одновременно два критерия: пиковую заболеваемость и долю умерших. Использует эволюционный алгоритм (Borg MOEA) из пакета `BlackBoxOptim`.

4.1.6.2 Входные параметры (переменные)

— β_und — в диапазоне [0.1, 1.0].
— `detection_time` — в диапазоне [3, 14] (целые).
— `death_rate` — в диапазоне [0.01, 0.1].

4.1.6.3 Что делает внутри

— Определяет целевую функцию `cost_multi(x)`, которая для заданных параметров запускает несколько повторных симуляций (5) и возвращает среднюю долю пиковой заболеваемости и среднюю долю умерших.
— Запускает оптимизатор с методом `:borg_moea`, максимизирующим (или минимизирующим) два критерия.
— Результат оптимизации — набор точек на Парето-фронте.

4.1.6.4 Результаты

— Вывод в консоль: оптимальные параметры и соответствующие им значения критериев.
— Сохранение в JLD2-файл: `data/optimization_result.jld2` — содержит словарь с ключами `"best"` (вектор параметров) и `"fitness"` (вектор критериев).

Этот скрипт позволяет найти компромиссные стратегии управления (например, раннее выявление снижает пик, но не влияет на смертность).

```
using DrWatson
@quickactivate "project"

using BlackBoxOptim, Random, Statistics

include(srcdir("sir_model.jl"))

# Целевая функция: минимизируем пиковую заболеваемость и смертность
function cost_multi(x)
    # x[1]: β_und, x[2]: death_rate, x[3]: detection_time
    model = initialize_sir(;
        Ns = [1000, 1000, 1000],
        β_und = fill(x[1], 3),
        β_det = fill(x[1]/10, 3),
        infection_period = 14,
        detection_time = round(Int, x[2]),
        death_rate = x[3],
        reinfection_probability = 0.1,
        Is = [0, 0, 1],
        seed = 42,
        n_steps = 100,
    )

    infected_frac(model) = count(a.status == :I for a in
     ↪  allagents(model)) / nagents(model)
    dead_count(model) = 3000 - nagents(model)

    peak_infected = 0.0

    # Запускаем с несколькими повторами
    replicates = 5
    peak_vals = Float64[]
    dead_vals = Int[]

    for rep = 1:replicates
        model = initialize_sir(;
            Ns = [1000, 1000, 1000],
            β_und = fill(x[1], 3),
            β_det = fill(x[1]/10, 3),
            infection_period = 14,
            detection_time = round(Int, x[2]),
            death_rate = x[3],
            reinfection_probability = 0.1,
```

```
            Is = [0, 0, 1],
            seed = 42 + rep,
            n_steps = 100,
        )

        for step = 1:100
            Agents.step!(model, 1)
            frac = infected_frac(model)
            if frac > peak_infected
                peak_infected = frac
            end
        end

        push!(peak_vals, peak_infected)
        push!(dead_vals, dead_count(model))
    end

    return (mean(peak_vals), mean(dead_vals) / 3000)  # доля умерших
end

# Запуск оптимизации
result = bboptimize(
    cost_multi,
    Method = :borg_moea,
    FitnessScheme = ParetoFitnessScheme{2}(is_minimizing = true),
    SearchRange = [
        (0.1, 1.0),     # β_und
        (3.0, 14.0),    # detection_time
        (0.01, 0.1),    # death_rate
    ],
    NumDimensions = 3,
    MaxTime = 120,  # 2 минуты
    TraceMode = :compact,
)

best = best_candidate(result)
fitness = best_fitness(result)

println("Оптимальные параметры:")
println("β_und = $(best[1])")
println("Время выявления = $(round(Int, best[2])) дней")
println("Смертность = $(best[3])")
println("Достигнутые показатели:")
println("Пик заболеваемости: $(fitness[1])")
```

```
println("Доля умерших: $(fitness[2])")

save(datadir("optimization_result.jld2"), Dict("best" ⇒ best,
 ↪  "fitness" ⇒ fitness))
```

4.1.7 Сводная визуализация результатов

— `scripts/sir_visualize_dynamics.jl`

4.1.8 Что делает

Этот скрипт загружает результаты параметрического сканирования (файл `data/beta_scan_all.csv`, созданный `scan_beta.jl`) и строит единый составной график, объединяющий три панели:

— пик эпидемии и конечная доля инфицированных;
— число умерших;
— доля выздоровевших.

График сохраняется как `plots/comprehensive_analysis.png`. Скрипт предназначен для итогового отчёта и не требует повторного запуска симуляций.

4.1.9 Входные данные (что получает)

— CSV-файл `data/beta_scan_all.csv`, содержащий все прогоны сканирования β (колонки: `β_und`, `peak`, `final_inf`, `deaths`, `final_rec` и др.).
— Если файл не найден, скрипт выдаст предупреждение.

4.1.10 Что делает внутри

— Загружает CSV с помощью `CSV.read` и преобразует в DataFrame.
— При необходимости усредняет данные по повторным прогонам (группировка по `β_und`). В приведённой реализации используется прямое построение без усреднения, но можно добавить.
— Создаёт фигуру с тремя вертикальными подграфиками:

  - Верхний: зависимость пиковой и конечной доли инфицированных от β.
  - Средний: зависимость числа умерших от β.
  - Нижний: зависимость доли выздоровевших от β.
- Настраивает подписи осей, легенду и сохраняет рисунок в `plots/comprehensive_analysis.png`.

4.1.11 Результаты

- График: `plots/comprehensive_analysis.png` (рис. 4.4). Три панели, позволяющие одновременно оценить:
  - порог возникновения эпидемии (рост пика);
  - нелинейное увеличение смертности;
  - насыщение доли переболевших.
- В консоль может выводиться дополнительная статистика (например, значения β, при которых пик превышает 5%).

В отличие от `scan_beta.jl`, этот скрипт не выполняет симуляции, а только визуализирует уже полученные данные. Это позволяет быстро перестраивать графики при изменении оформления, не тратя время на повторные расчёты.

```
using DrWatson
@quickactivate "project"
using Agents, DataFrames, Plots, CSV

include(srcdir("sir_model.jl"))

# Загружаем результаты сканирования
df = CSV.read(datadir("beta_scan_all.csv"), DataFrame)

# Создаём составной график
p1 = plot(df.beta, df.peak, label = "Пик", xlabel = "β", ylabel =
 ↪  "Доля инфицированных")
plot!(p1, df.beta, df.final_inf, label = "Конечная")

p2 = plot(df.beta, df.deaths, xlabel = "β", ylabel = "Число умерших")

p3 = plot(df.beta, df.final_rec, xlabel = "β", ylabel = "Доля
 ↪  выздоровевших")

plot(p1, p2, p3, layout = (3, 1), size = (800, 900))
```

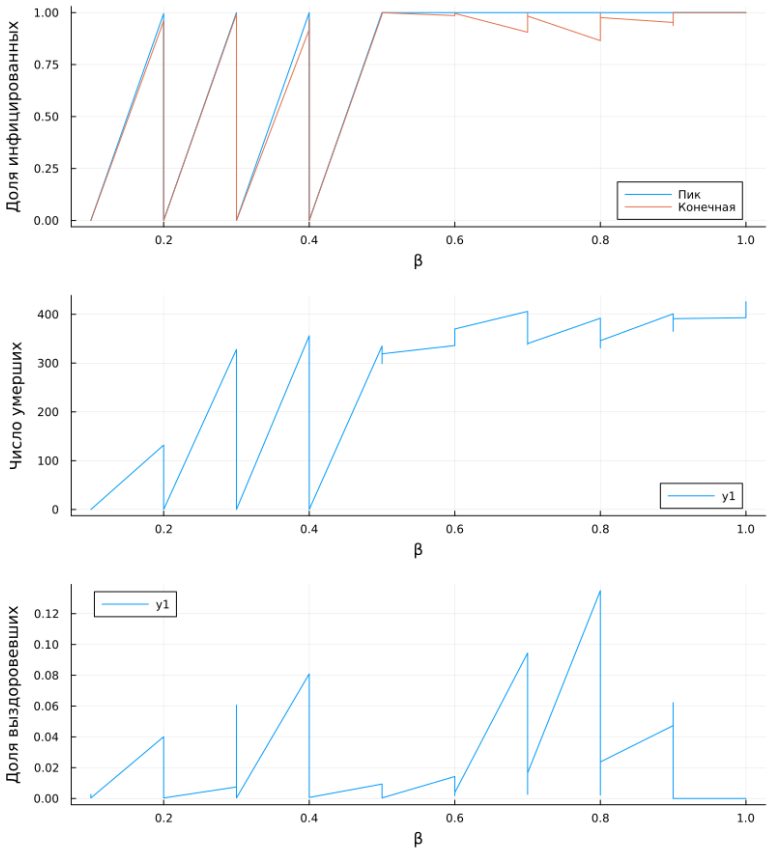


Рисунок 4.4. График динамики миграции

```
savefig(plotsdir("comprehensive_analysis.png"))
```

4.1.12 Дополнительные задания

1. Базовый уровень
   - Запустите модель с параметрами по умолчанию.
   - Постройте график динамики численности S, I, R.
   - Определите базовое репродуктивное число R_0 по формуле $R_0 = \beta/\gamma$, где $\gamma = 1/\text{infection_period}$.
   - Сравните с наблюдаемой динамикой.
2. Исследование порога
   - Найдите минимальное значение β, при котором возникает эпидемия (пик $I > 5\%$ популяции).
   - Сравните с теоретическим порогом $R_0 = 1$.
3. Эффект гетерогенности
   - Задайте разные значения β для разных городов.
   - Как это влияет на общую динамику?
   - Постройте графики для каждого города отдельно.
4. Миграция
   - Исследуйте влияние интенсивности миграции на скорость распространения инфекции из одного города в другие.
   - При какой интенсивности время до пика минимально?
5. Карантинные меры
   - Модифицируйте модель, добавив возможность закрытия города (обнуление миграции из него) при превышении порога заболеваемости.
   - Оцените эффективность такой меры.
6. Оптимизация
   - Используя оптимизацию, найдите параметры, которые минимизируют общее число умерших при сохранении пика заболеваемости ниже 30%.

4.2 Задание

- Создать рабочий каталог для кода.
- Установить необходимые пакеты.
- Выполнить предложенный код.
- Преобразовать код в литературный стиль.

- Сгенерировать из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.

5

Аппарат сетей Петри

5.1 Задание

- Создать рабочий каталог для кода.
- Установить необходимые пакеты.
- Выполнить предложенный код.
- Преобразовать код в литературный стиль.
- Сгенерировать из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.

6

Реализация основных моделей в подходе сетей Петри

6.1 Задание

- Создать рабочий каталог для кода.
- Установить необходимые пакеты.
- Выполнить предложенный код.
- Преобразовать код в литературный стиль.
- Сгенерировать из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.

7

Дискретно-событийное моделирование

7.1 Задание

- Создать рабочий каталог для кода.
- Установить необходимые пакеты.
- Выполнить предложенный код.
- Преобразовать код в литературный стиль.
- Сгенерировать из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.

8

Реализация основных моделей в дискретно-событийном подходе

8.1 Задание

- Создать рабочий каталог для кода.
- Установить необходимые пакеты.
- Выполнить предложенный код.
- Преобразовать код в литературный стиль.
- Сгенерировать из литературного кода:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.
- Добавить в код в литературном стиле вычисление для набора параметров.
- Сгенерировать из литературного кода с параметрами:
  - чистый код;
  - jupyter notebook;
  - документацию в формате Quarto.
- Выполнить код из jupyter notebook с параметрами.
- Интегрировать документацию с параметрами в формате Quarto в отчёт.

Список литературы

1. A Multi-Language Computing Environment for Literate Programming and Reproducible Research / E. Schulte [et al.] // Journal of Statistical Software. — 2012. — Vol. 46, no. 3. — ISSN 1548-7660. — DOI: 10.18637/jss.v046.i03.
2. Daisyworld: A review / A. J. Wood [et al.] // Reviews of Geophysics. — 2008. — Jan. — Vol. 46, no. 1. — ISSN 1944-9208. — DOI: 10.1029/2006rg000217.
3. *Datseris G.*, *Vahdati A. R.*, *DuBois T. C.* Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity // SIMULATION. — 2022. — Jan. — P. 003754972110688. — DOI: 10.1177/00375497211068820.
4. *Hethcote H. W.* The Mathematics of Infectious Diseases // SIAM Review. — 2000. — Jan. — Vol. 42, no. 4. — P. 599–653. — ISSN 1095-7200. — DOI: 10.1137/s0036144500371907.
5. *Kermack W. O.*, *McKendrick A. G.* A Contribution to the Mathematical Theory of Epidemics // Proceedings of the Royal Society of London. Series A Containing Papers of a Mathematical and Physical Character. — 1927. — Авг. — Т. 115, № 772. — С. 700—721. — ISSN 2053-9150. — DOI: 10.1098/rspa.1927.0118.
6. *Knuth D. E.* Literate Programming // The Computer Journal. — 1984. — Feb. — Vol. 27, no. 2. — P. 97–111. — ISSN 1460-2067. — DOI: 10.1093/comjnl/27.2.97.
7. *Lotka A. J.* Contribution to the Theory of Periodic Reaction // The Journal of Physical Chemistry A. — 1910. — Vol. 14, no. 3. — P. 271–274. — DOI: 10.1021/j150111a004.
8. *Lotka A. J.* Elements of Physical Biology. — Baltimore : Williams, Wilkins Company, 1925. — 435 p. — URL: https://archive.org/details/elementsofphysic017171mbp.
9. The Story in the Notebook / M. B. Kery [et al.] // Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. — ACM, 04/2018. — P. 1–11. — DOI: 10.1145/3173574.3173748.
10. *Volterra V.* Variations and fluctuations of the number of individuals in animal species living together // Journal du Conseil permanent International pour l Exploration de la Mer. — 1928. — Vol. 3, no. 1. — P. 3–51.

11. *Watson A. J.*, *Lovelock J. E.* Biological homeostasis of the global environment: the parable of Daisyworld // Tellus B: Chemical and Physical Meteorology. — 1983. — Jan. — Vol. 35, no. 4. — P. 284. — ISSN 0280-6509. — DOI: 10.3402/tellusb.v35i4.14616.

12. *Вольтерра В.* Математическая теория борьбы за существование : пер. с фр. — М. : Наука, 1976.. — 288 с. — Пер. по изд.: *Volterra V.* Leçons sur la Théorie mathématique de la lutte pour la vie. — Paris : Gauthiers-Villars, 1931. — ISBN 2876470667. — URL : http://www.amazon.com/lecons-theorie-math-lutte-pour/dp/2876470667%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D2876470667.