Лабораторная работа №1

Имитационное моделирование

Чистов Даниил Максимович

Содержание

1. Цель работы

Целью данной работы является создание рабочего пространства для последующих работ, получение практических навыков с Julia и git.

2. Задание

- Создать рабочий каталог для всего курса.
- Создать рабочее пространство для программ в рамках 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: чистый код, jupyter notebook, документацию в формате Quarto.
- Выполнить код из jupyter notebook.
- Интегрировать документацию в формате Quarto в отчёт.

3. Выполнение лабораторной работы

3.1 Создание рабочего катологa для всего курса

Требуется создать рабочие каталоги на двух платформах - GitVerse и GitHub. Я воспользовался шаблоном курса, создал репозитории на GitVerse (рис. 1), аналогично на GitHub

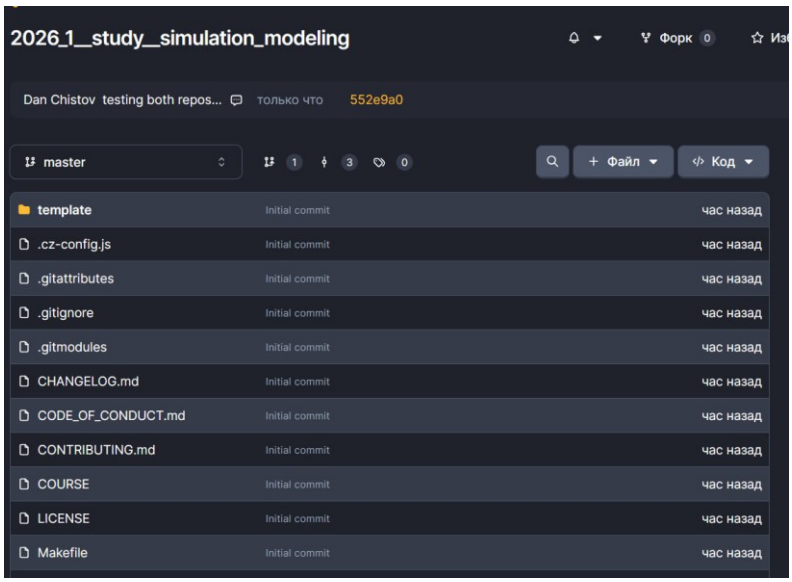


Рисунок 1: Созданный репозиторий на платформе GitVerse

Теперь перехожу в папку курса и инициализирую его командой make prepare (рис. 2).

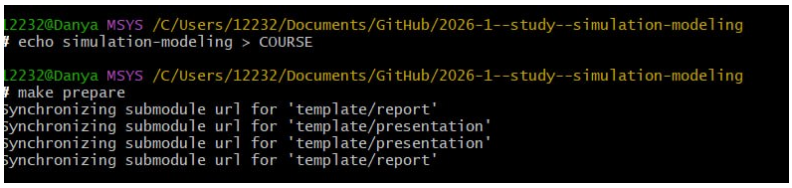


Рисунок 2: Инициализация курса в консоли

Загружаю изменения на сервер (рис. 3).

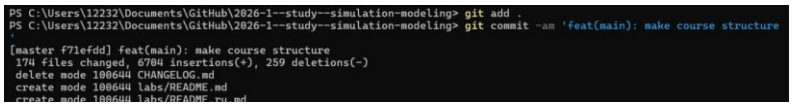


Рисунок 3: Загрузка изменений на сервер с помощью git

Отправляю изменения на обе платформы - GitHub (origin), GitVerse (simmod) (рис. 4).

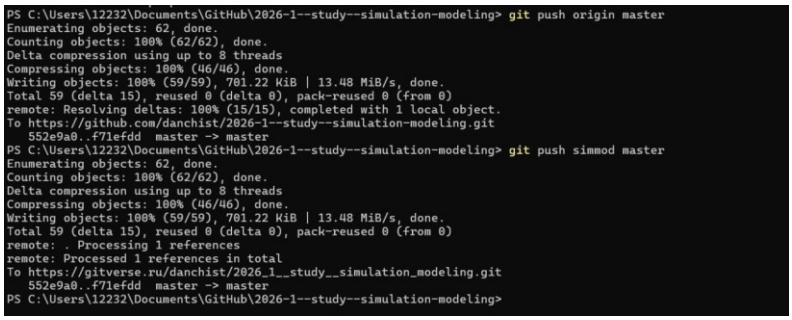


Рисунок 4: Отправление изменений на GitHub и GitVerse

Изменения успешно внесены, рабочий католог курса создан (рис. 6).

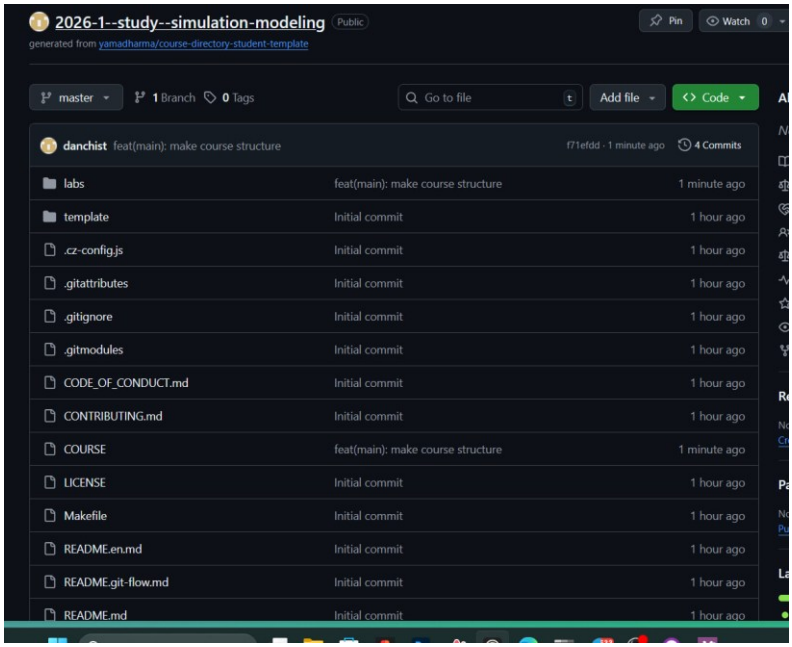


Рисунок 5: Рабочий католог курса на платформе GitHub

По заданию курс требует git flow для работы. Инициализирую его, выбираю v, как префикс для новых версий (рис. 6).

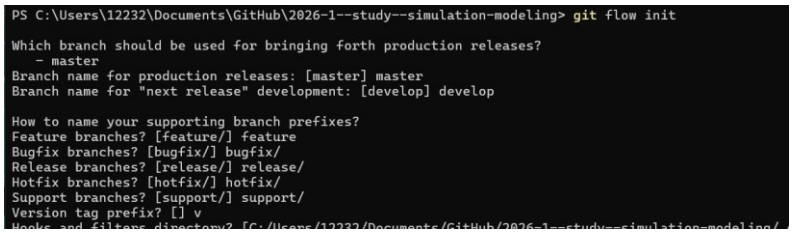


Рисунок 6: Инициализация git flow

После инициализации git flow появилось разделение на ветки develop и master, нужно внести изменения на обе платформы GitHub и GitVerse. Командой git branch проверяю, что я нахожусь на ветке develop. После чего отправляю изменения на платформы (рис. 7).

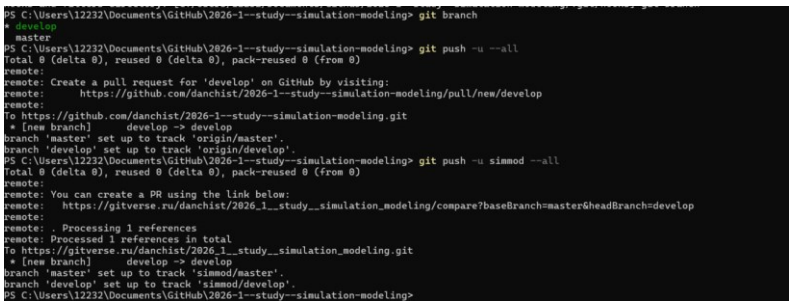


Рисунок 7: Отправление изменений курса на платформы

Инициализирую первый релиз - версия 1.0.0, затем командой standard-changelog –first-release создаю журнал изменений, добавляю изменения в git, создаю коммит (рис. 8).

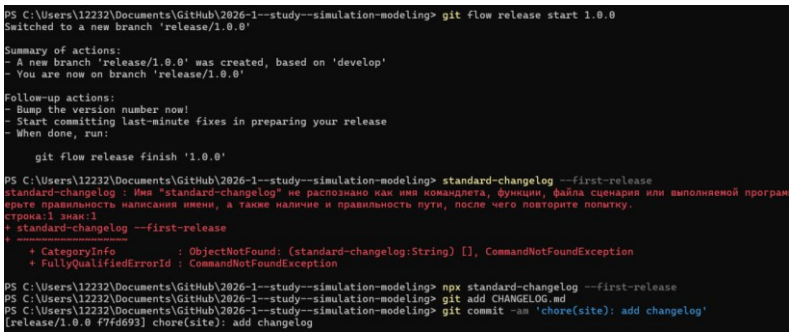


Рисунок 8: Инициализация первого релиза

Отправляю изменения на обе платформы, включая созданный тэг (рис. 9).

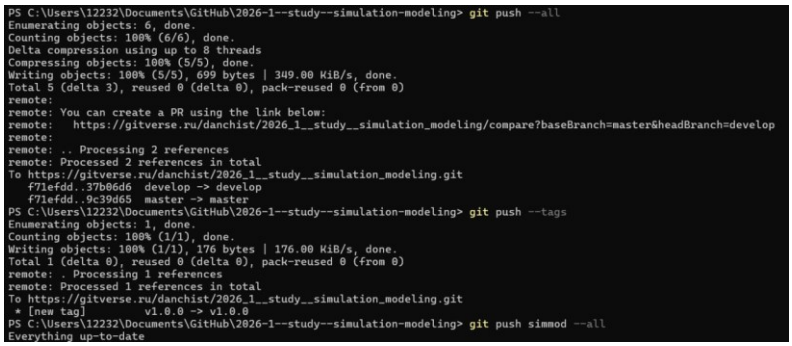


Рисунок 9: Изменения отправлены на GitHub и GitVerse

На GitVerse самостоятельно создаю первый релиз (рис. 10).

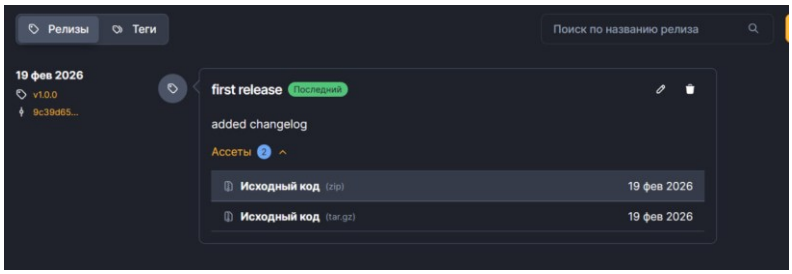


Рисунок 10: Первый релиз на GitVerse

На GitHub создаю релиз командами release create и прикрепляю журнал изменений (рис. 11).

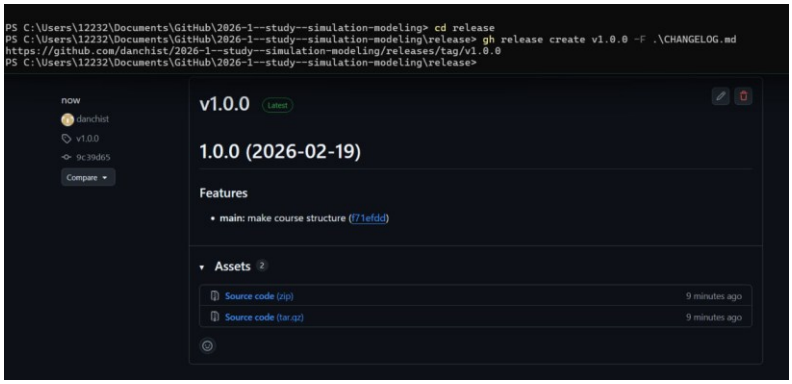


Рисунок 11: Первый релиз на GitHub

3.2 Создание проекта DrWatson для лабораторных работ

Перехожу в каталог lab01, первый раз запускаю Julia, после чего командами using Pkg, Pkg.add(“DrWatson”) загружаю пакет DrWatson. Затем инициализирую проект (рис. 12).

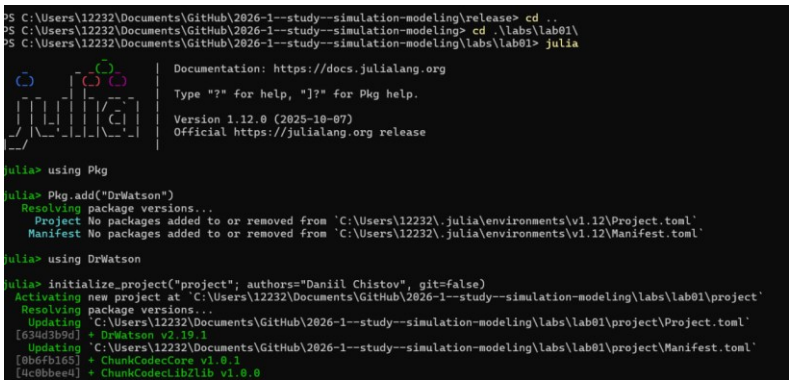


Рисунок 12: Инициализация проекта с помощью DrWatson

Для дальнейшей работы требуется загрузить большее количество пакетов. Вместо того, чтобы делать это самостоятельно, воспользуюсь скриптом, который загрузит все требуемые пакеты. Ниже программный код данного скрипта (рис. 13).

.gitignore | .gitignore | Makefile | add_packages.jl

```
##!/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")
```

Рисунок 13: Программный код для загрузки требуемых пакетов

Запускаю скрипт, пакеты загружены успешно (рис. 14).

```
PS C:\Users\12232\Documents\GitHub\2026-1--study--simulation-modeling\labs\lab01\project> julia .\add_packages.jl
  Activating project at `C:\Users\12232\Documents\GitHub\2026-1--study--simulation-modeling\labs\lab01\project`
Установка базовых пакетов...
   Resolving package versions...
   Installed SciMLLogging ── v1.9.0
   Installed CSV ──────────── v0.10.16
   Installed Plots ────────── v1.41.6
   Installed JumpProcesses ─ v9.22.0
  Installing artifacts ──────────────────────────── 1/1
    Updating `C:\Users\12232\Documents\GitHub\2026-1--study--simulation-modeling\labs\lab01\project\Project.toml`
  [6e4b80f9] + BenchmarkTools v1.6.3
  [336ed68f] + CSV v0.10.16
  [a93c6f00] + DataFrames v1.8.1
  [0c46a032] + DifferentialEquations v7.17.0
  [7073ff75] + IJulia v1.34.3
  [033835bb] + JLD2 v0.6.3
  [98b081ad] + Literate v2.21.0
  [91a5bcdd] + Plots v1.41.6
  [d7167be5] + Quarto v1.0.0
    Updating `C:\Users\12232\Documents\GitHub\2026-1--study--simulation-modeling\labs\lab01\project\Manifest.toml`
  [47edcb42] + ADTypes v1.21.0
```

Рисунок 14: Загрузка требуемых пакетов

Теперь требуется проверить загруженные пакеты следующим скриптом. Перед этим, в папке scripts создаю файл, вставляю требуемый программный код проверки пакетов (рис. 15).

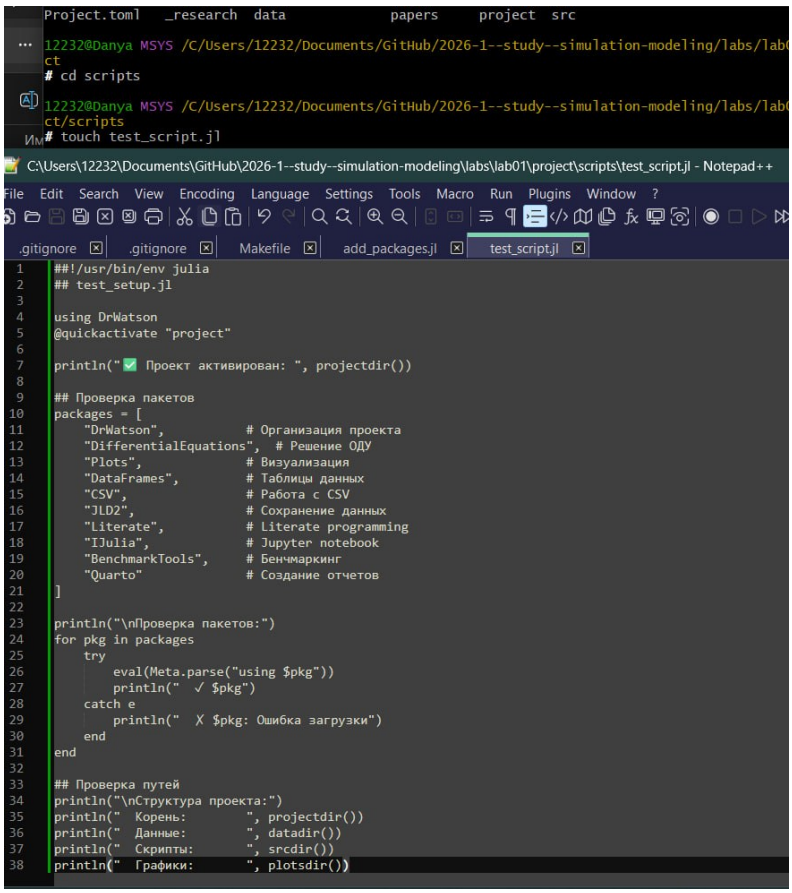


Рисунок 15: Создание скрипта для проверки пакетов

Запускаю скрипт. Все пакеты проверены. Можно приступать к работе с ними (рис. 16).

Рисунок 16: Пакеты успешно проверены

3.3 Работа с моделью экспоненциального роста

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.

В лабораторной работе требуется реализовать данную модель с помощью Julia.

Создаю скрипт 01_exponential_growth.jl, данный скрипт получит на вход один набор параметров для модели экспоненциального роста, получит решение, добавит их в таблицу, а также нарисует график (рис. 17).

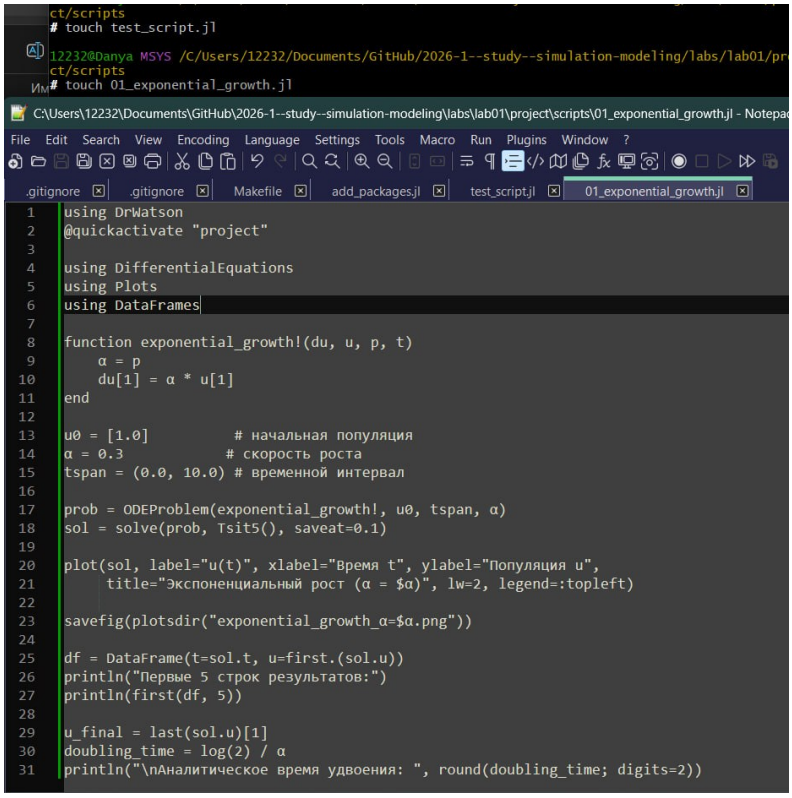


Рисунок 17: Первая версия скрипта 01_exponential_growth.jl

Запускаю скрипт, работа выполнено успешно, получены результаты, а также график (рис. 18).

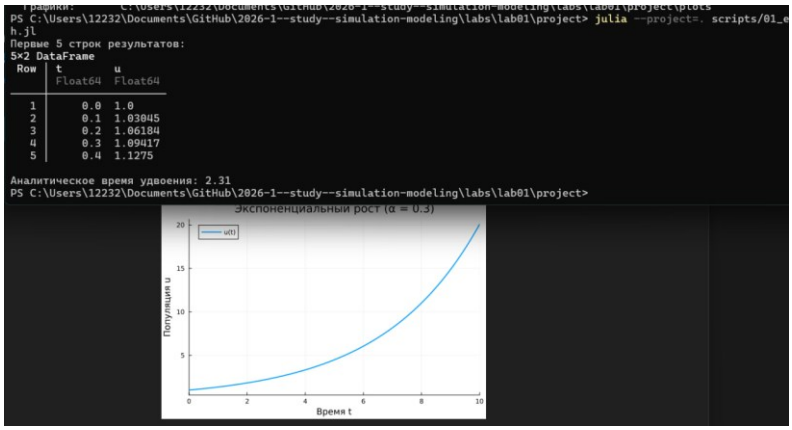


Рисунок 18: Результат работы 01_exponential_growth.jl

3.4 Преобразование кода в литературный вид

Теперь требуется преобразовать программный код данного скрипта в литературный вид.

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

Преобразую код 01_exponential_growth.jl в литературный вид (рис. 19).

Makefile | add_packages.jl | 01_exponential_growth.jl

```
# # 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:
# $\frac{du}{dt} = α u, \quad u(0) = u_0$

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"))
```

Рисунок 19: Литературный вид 01_exponential_growth.jl

Т. к. сама суть исполняемого кода не была изменена, результат выплонения программы такой же. Рисунок также был создан (рис. 20).

```
PS C:\Users\12232\Documents\GitHub\2026-1--study--simulation-modeling\labs\lab01\project> julia --project=. scripts/01_exponential_growth.jl
Первые 5 строк результатов:
5×2 DataFrame
 Row │ t        u
     │ Float64  Float64
─────┼──────────────────
   1 │     0.0  1.0
   2 │     0.1  1.03045
   3 │     0.2  1.06184
   4 │     0.3  1.09417
   5 │     0.4  1.1275

Аналитическое время удвоения: 2.31
PS C:\Users\12232\Documents\GitHub\2026-1--study--simulation-modeling\labs\lab01\project>
```

Рисунок 20: Результат 01_exponential_growth.jl в литературном виде

Теперь можно воспользоваться особенностями такого вида программирования. По заданию требуется создать скрипт tangle.jl, который на вход будет получать код в литературном виде, а на выходе будет производить три файла - чистый код, jupyter notebook, документацию в формате Quarto (рис. 21).

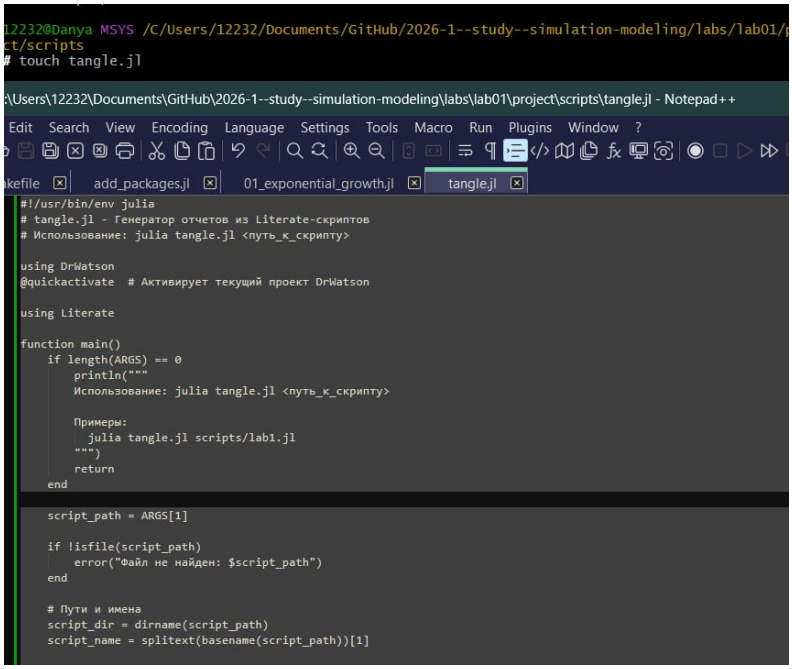


Рисунок 21: Создание tangle.jl

Запускаю tangle.jl, все три файла успешно созданы (рис. 22).

Рисунок 22: Результат работы tangle.jl

Проверим созданный Jupyter notebook. Файл открывается и успешно работает (рис. 23).

Экспоненциальный рост

Цель: Исследовать решение уравнения $du/dt = \alpha u$.

Инициализация проекта и загрузка пакетов

In [ ]:
```
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: $\frac{du}{dt} = \alpha u, \quad u(0) = u_0$

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

Первый запуск с параметрами по умолчанию

Зададим начальные параметры:

In [ ]:
```
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

Рисунок 23: Проверка 01_exponential_growth в виде Jupyter notebook

Теперь требуется внести код 01_exponential_growth в данный отчёт. Перед этим в файл _quarto.yml добавляю код, включающий поддержку Julia (рис. 24).

```yaml
project:
  title: "report"
  output-dir: "_output"

standalone: true
self-contained: true

## Julia support
engine: julia
julia:
  exeflags: ["--project=../project"]

## Generic options
```

Рисунок 24: Изменения в _quarto.yml

Затем вношу изменения в preamble.tex, также добавляя поддержку Julia (рис. 25).

Рисунок 25: Изменения в preamble.tex

Код успешно добавлены, вы можете прочитать его ниже:

3.5 Отчёт 01_exponential_growth

4. Экспоненциальный рост

Цель: Исследовать решение уравнения $du/dt = \alpha u$.

4.1 Инициализация проекта и загрузка пакетов

```
using DrWatson
@quickactivate "../project"

using Plots
default(fmt = :png)
gr(fmt = :png)

using DifferentialEquations
using DataFrames
using JLD2

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

```
┌ Warning: DrWatson could not find find a project file by recursively checking
given `dir` and its parents. Returning `nothing` instead.
│   (given dir: C:\Users\12232)
└ @ DrWatson
C:\Users\12232\.julia\packages\DrWatson\2QF5p\src\project_setup.jl:84

"C:\\Users\\12232\\Documents\\GitHub\\2026-1--study--simulation-
modeling\\labs\\lab01\\project\\data\\"
```

4.2 Определение модели

Уравнение экспоненциального роста: $\frac{du}{dt} = \alpha u, \quad u(0) = u_0$

```
function exponential_growth!(du, u, p, t)
    α = p
```

```
    du[1] = α * u[1]
end

exponential_growth! (generic function with 1 method)
```

4.3 Первый запуск с параметрами по умолчанию

Зададим начальные параметры:

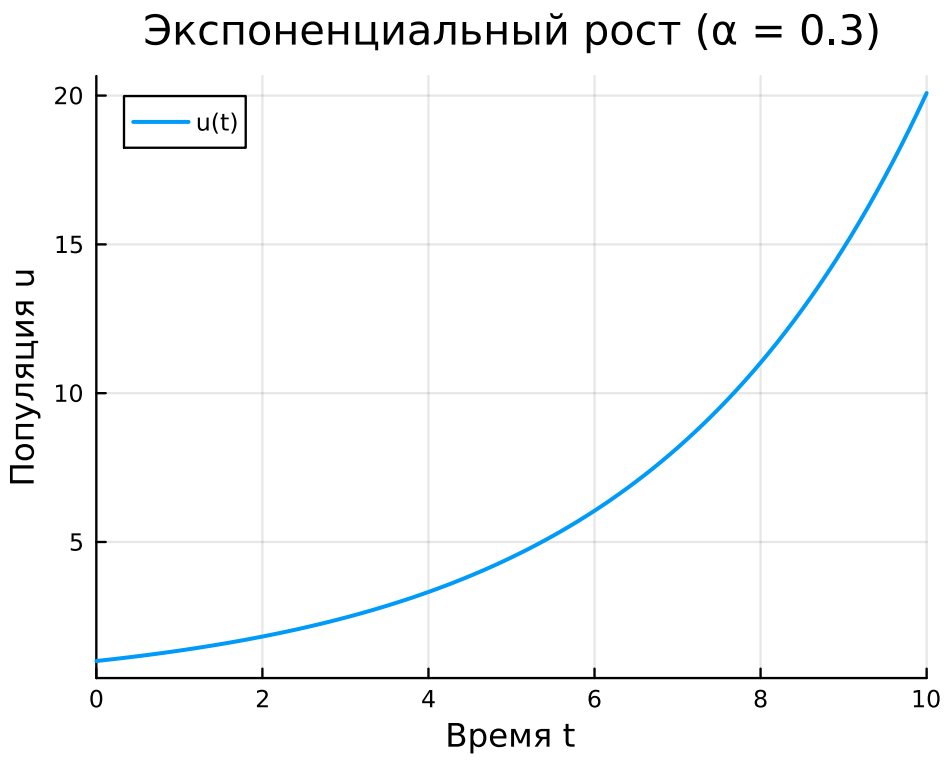
```
u0 = [1.0]          # начальная популяция
α = 0.3            # скорость роста
tspan = (0.0, 10.0) # временной интервал

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

retcode: Success
Interpolation: 1st order linear
t: 101-element Vector{Float64}:
  0.0
  0.1
  0.2
  0.3
  0.4
  0.5
  0.6
  0.7
  0.8
  0.9
  1.0
  1.1
  1.2
  ⋮
  8.9
  9.0
  9.1
  9.2
  9.3
  9.4
  9.5
  9.6
  9.7
  9.8
  9.9
 10.0
u: 101-element Vector{Vector{Float64}}:
 [1.0]
 [1.030454533950446]
 [1.0618365551529674]
 [1.0941743028794098]
 [1.1274968605386673]
 [1.1618342327450653]
 [1.1972173476990624]
```

```
[1.233678057187257]
[1.2712491879500347]
[1.3099646073864084]
[1.3498590188273822]
[1.390968273354968]
[1.4333293964993763]
⋮
[14.439892233074872]
[14.87962750503581]
[15.33275596650523]
[15.79968870415309]
[16.280848966976613]
[16.776672166300525]
[17.287605875776965]
[17.814109831385352]
[18.35665593143248]
[18.91572823655283]
[19.491822969707854]
[20.08544851618676]
```

4.4 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"))
```

```
"C:\\Users\\12232\\Documents\\GitHub\\2026-1--study--simulation-
modeling\\labs\\lab01\\project\\plots\\exponential_growth_α=0.3.png"
```

4.5 Анализ результатов

Создадим таблицу с данными:

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

```
Первые 5 строк результатов:
5×2 DataFrame
 Row │ t        u
     │ Float64  Float64
─────┼──────────────────
   1 │     0.0  1.0
   2 │     0.1  1.03045
   3 │     0.2  1.06184
   4 │     0.3  1.09417
   5 │     0.4  1.1275
```

Вычислим удвоение популяции:

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

```
Аналитическое время удвоения: 2.31
```

4.6 Сохранение всех результатов

```
@save datadir(script_name, "all_results.jld2") df
```

4.7 Реализация модели с параметрами

Теперь требуется создать новый код, вместо того, чтобы подавать один набор данных, как в 01_exponential_growth.jl, изменю программу так, чтобы подавалось несколько наборов данных. Код уже преобразован в литературный вид (рис. 26).

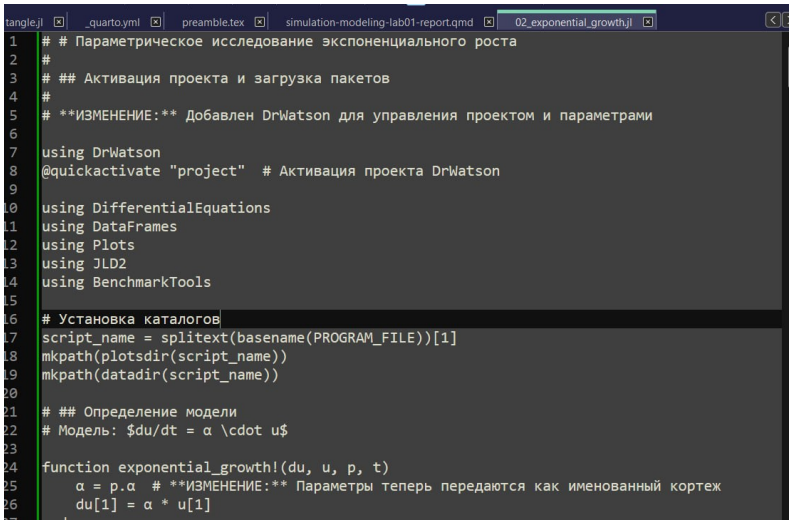


Рисунок 26: Создание 02_exponential_growth.jl

Запускаю программу, код работает успешно (рис. 27).

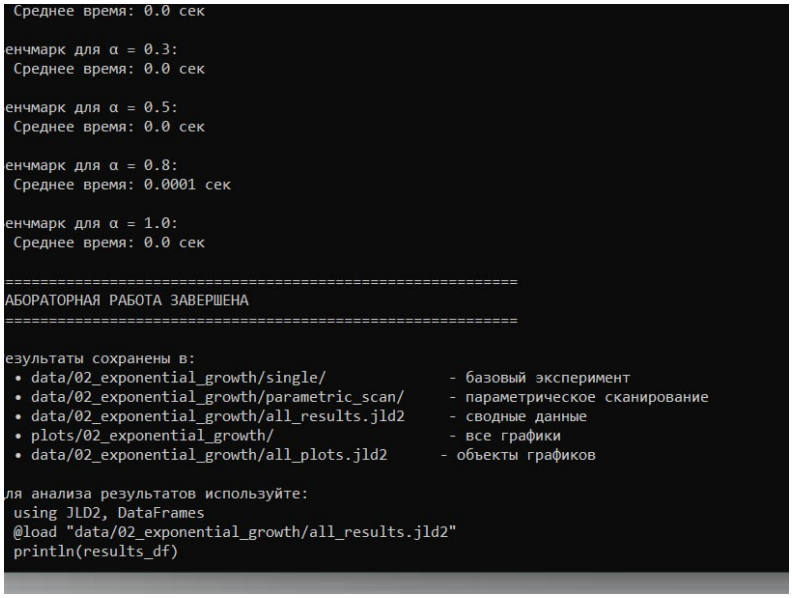
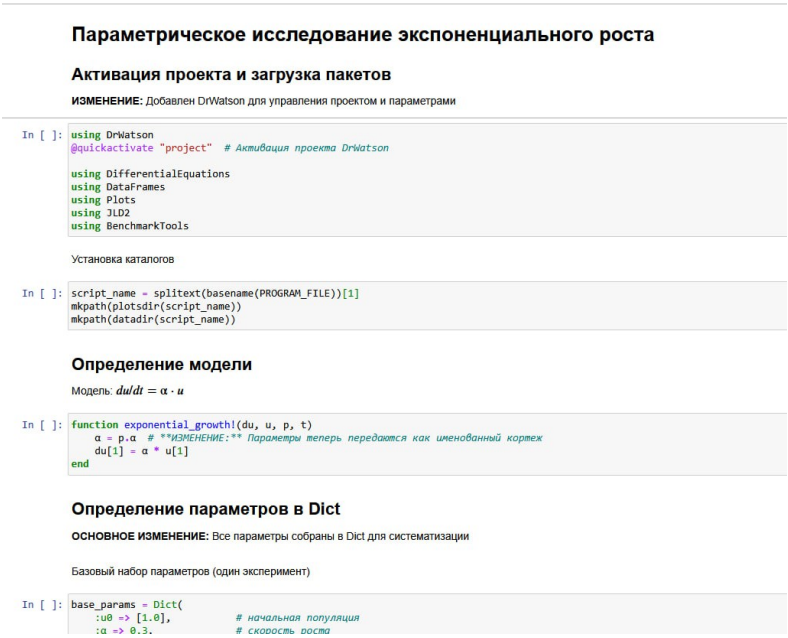


Рисунок 27: Результат работы 02_exponential_growth.jl

Программой tangle.jl создаю три файла из 02_exponential_growth.jl, среди которых Jupyter notebook. Он успешно работает (рис. 28).



Параметрическое исследование экспоненциального роста

Активация проекта и загрузка пакетов

ИЗМЕНЕНИЕ: Добавлен DrWatson для управления проектом и параметрами

In [ ]:
```
using DrWatson
@quickactivate "project"  # Активация проекта DrWatson

using DifferentialEquations
using DataFrames
using Plots
using JLD2
using BenchmarkTools
```

Установка каталогов

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

Определение модели

Модель: $du/dt = \alpha \cdot u$

In [ ]:
```
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

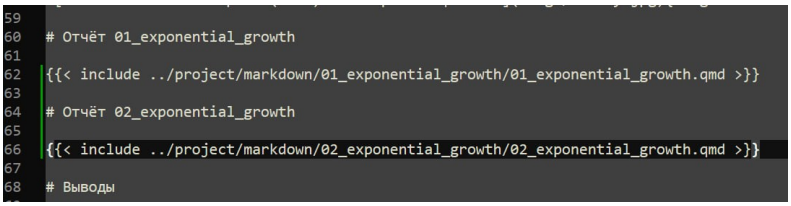
ОСНОВНОЕ ИЗМЕНЕНИЕ: Все параметры собраны в Dict для систематизации

Базовый набор параметров (один эксперимент)

In [ ]:
```
base_params = Dict(
    :u0 => [1.0],           # начальная популяция
    :α => 0.3,              # скорость роста
```

Рисунок 28: 02_exponential_growth.jl в виде Jupyter notebook

Добавлю эту программу в данный отчёт (рис. 29).



```
59
60  # Отчёт 01_exponential_growth
61
62  {{< include ../project/markdown/01_exponential_growth/01_exponential_growth.qmd >}}
63
64  # Отчёт 02_exponential_growth
65
66  {{< include ../project/markdown/02_exponential_growth/02_exponential_growth.qmd >}}
67
68  # Выводы
```

Рисунок 29: 02_exponential_growth.jl внутри отчёта

Код успешно интегрирован в отчёт, его вы можете просмотреть ниже:

5. Отчёт 02_exponential_growth

6. Параметрическое исследование экспоненциального роста

6.1 Активация проекта и загрузка пакетов

ИЗМЕНЕНИЕ: Добавлен DrWatson для управления проектом и параметрами

```
using DrWatson
@quickactivate "../project" # Активация проекта DrWatson

using Plots
default(fmt = :png)
```

```
gr(fmt = :png)

using DifferentialEquations
using DataFrames
using JLD2
using BenchmarkTools
```

```
┌ Warning: DrWatson could not find find a project file by recursively checking
given `dir` and its parents. Returning `nothing` instead.
│ (given dir: C:\Users\12232)
└ @ DrWatson
C:\Users\12232\.julia\packages\DrWatson\2QF5p\src\project_setup.jl:84
```

Установка каталогов

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

```
"C:\\Users\\12232\\Documents\\GitHub\\2026-1--study--simulation-
modeling\\labs\\lab01\\project\\data\\"
```

6.2 Определение модели

Модель: $du/dt = \alpha \cdot u$

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

```
exponential_growth! (generic function with 1 method)
```

6.3 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
```

```
Базовые параметры эксперимента:
  α = 0.3
  u0 = [1.0]
  saveat = 0.1
  solver = Tsit5{typeof(OrdinaryDiffEqCore.trivial_limiter!),
typeof(OrdinaryDiffEqCore.trivial_limiter!),
Static.False}(OrdinaryDiffEqCore.trivial_limiter!,
OrdinaryDiffEqCore.trivial_limiter!, static(false))
  experiment_name = base_experiment
  tspan = (0.0, 10.0)
```

6.4 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

run_single_experiment (generic function with 1 method)
```

6.5 Запуск базового эксперимента

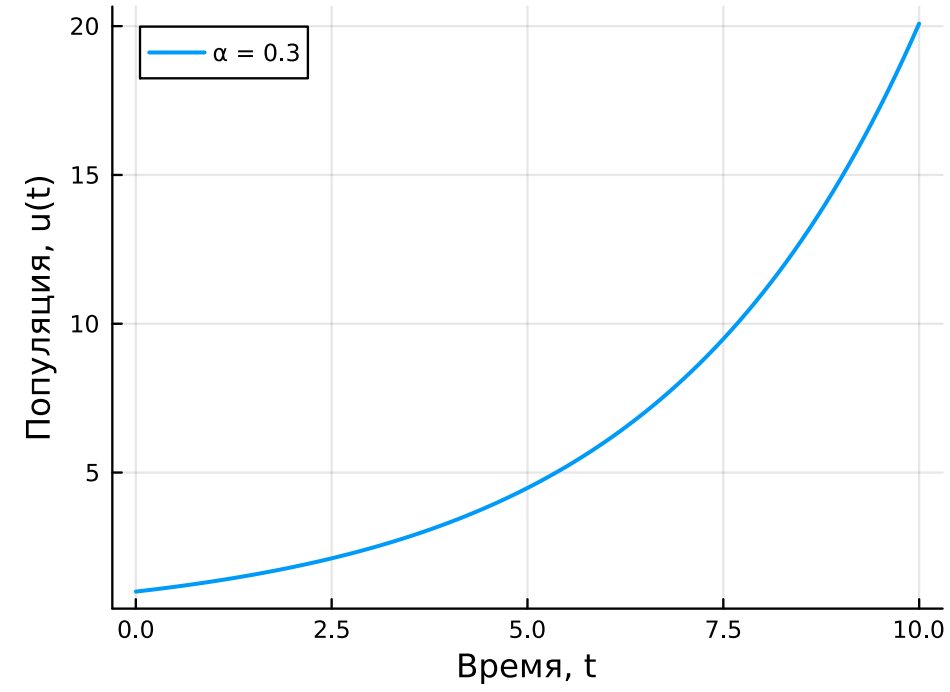
ИЗМЕНЕНИЕ: Используем produce_or_load для автоматического кэширования

```
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)
```

```
Результаты базового эксперимента:
  Финальная популяция: 20.08544851618676
  Время удвоения: 2.31
  Файл результатов: C:\Users\12232\Documents\GitHub\2026-1--study--simulation-
modeling\labs\lab01\project\data\single\exp_growth_experiment_name=base_experime
nt_saveat=0.1_α=0.3.jld2
```

6.6 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"))
```

```
"C:\\Users\\12232\\Documents\\GitHub\\2026-1--study--simulation-
modeling\\labs\\lab01\\project\\plots\\single_experiment.png"
```

6.7 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"]
)
```

```
Dict{Symbol, Vector} with 6 entries:
  :α               => [0.1, 0.3, 0.5, 0.8, 1.0]
  :u0              => [[1.0]]
  :saveat          => [0.1]
  :solver          => [Tsit5{typeof(trivial_limiter!), typeof(trivial_limiter!)…
  :experiment_name => ["parametric_scan"]
  :tspan           => [(0.0, 10.0)]
```

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)
```

```


============================================================
ПАРАМЕТРИЧЕСКОЕ СКАНИРОВАНИЕ
Всего комбинаций параметров: 5
Исследуемые значения α: [0.1, 0.3, 0.5, 0.8, 1.0]
============================================================
```

6.8 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

Прогресс: 1/5 | α = 0.1
Прогресс: 2/5 | α = 0.3
Прогресс: 3/5 | α = 0.5
Прогресс: 4/5 | α = 0.8
Прогресс: 5/5 | α = 1.0
```

6.9 Анализ и визуализация результатов сканирования

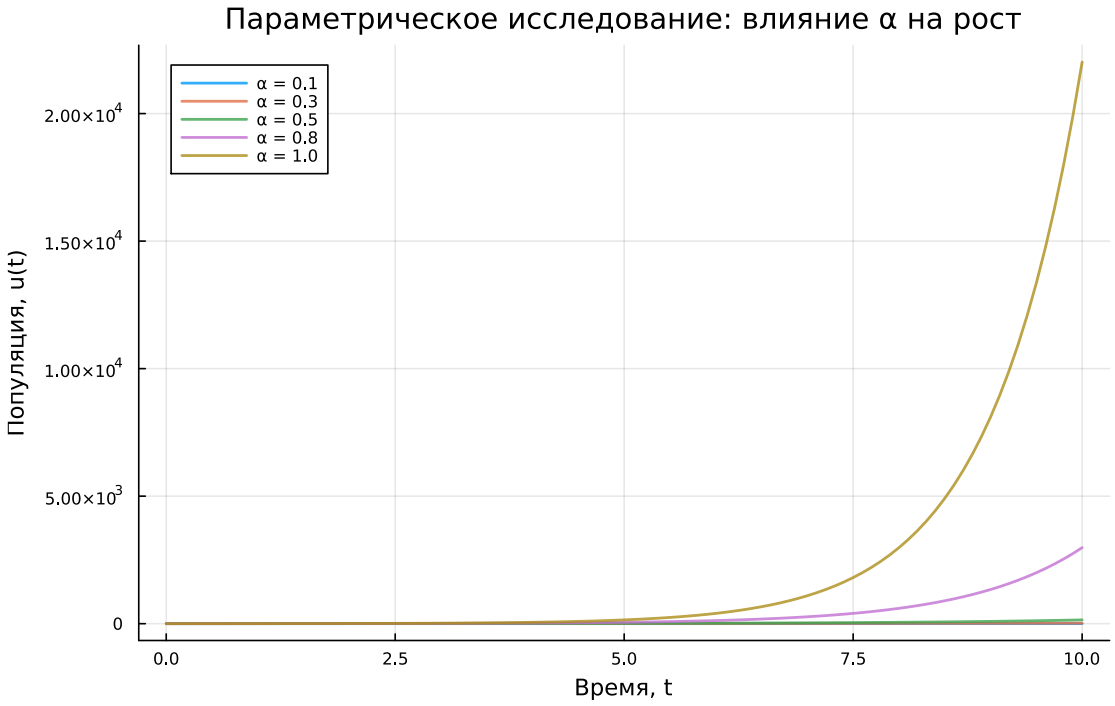
НОВАЯ СЕКЦИЯ: Сравнительный анализ всех экспериментов

Сводная таблица результатов

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


Сводная таблица результатов:
5×3 DataFrame
 Row │ α        final_population  doubling_time
     │ Float64  Float64           Float64
─────┼──────────────────────────────────────────
   1 │     0.1           2.71828       6.93147
   2 │     0.3          20.0854        2.31049
   3 │     0.5         148.409         1.38629
   4 │     0.8        2980.57          0.866434
   5 │     1.0       22021.0           0.693147
```

Сравнительный график всех траекторий

```julia
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
)
```



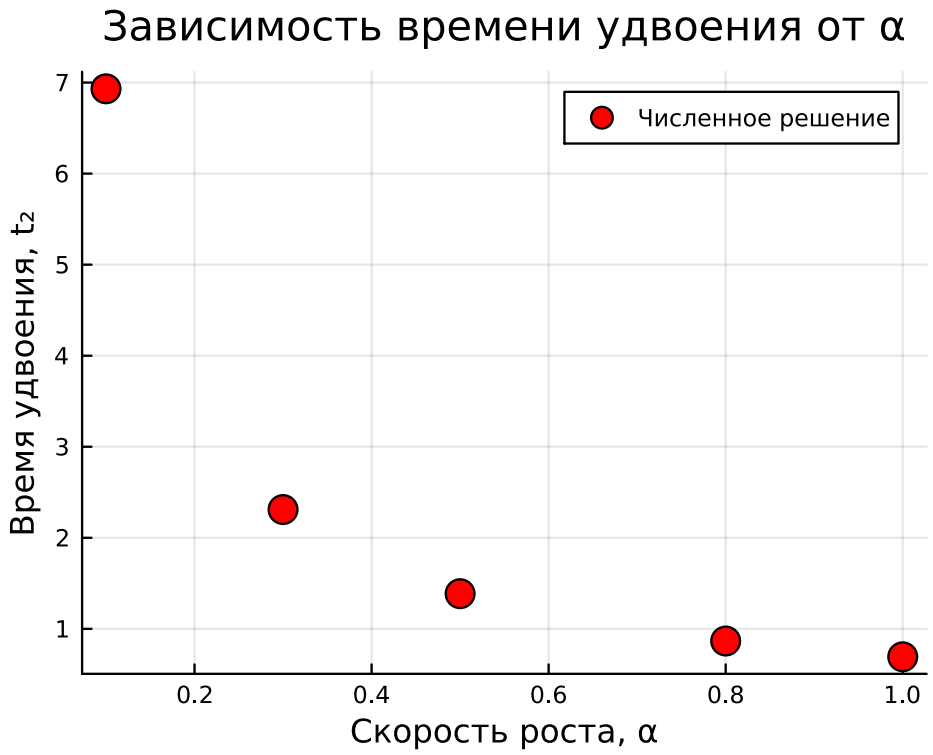
Сохраним график в папку plots

```julia
savefig(plotsdir(script_name, "parametric_scan_comparison.png"))
```

```
"C:\\Users\\12232\\Documents\\GitHub\\2026-1--study--simulation-
modeling\\labs\\lab01\\project\\plots\\parametric_scan_comparison.png"
```

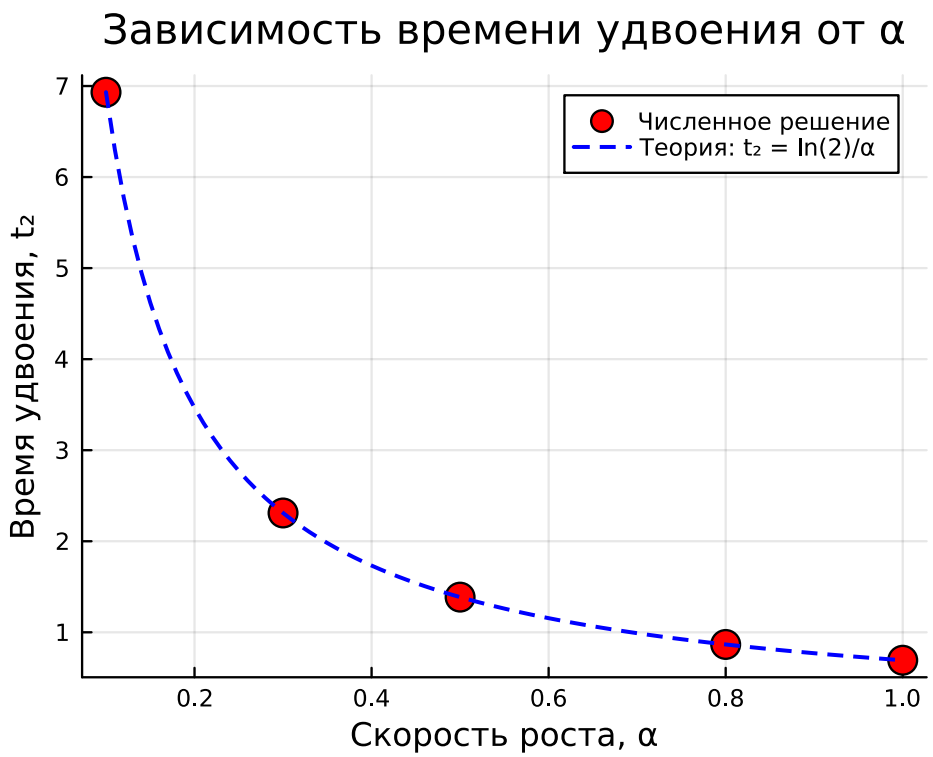
График зависимости времени удвоения от α

```
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: $t_2 = \ln(2)/\alpha$

```
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"))
```

```
"C:\\Users\\12232\\Documents\\GitHub\\2026-1--study--simulation-
modeling\\labs\\lab01\\project\\plots\\doubling_time_vs_alpha.png"
```

6.10 Бенчмаркинг с разными параметрами

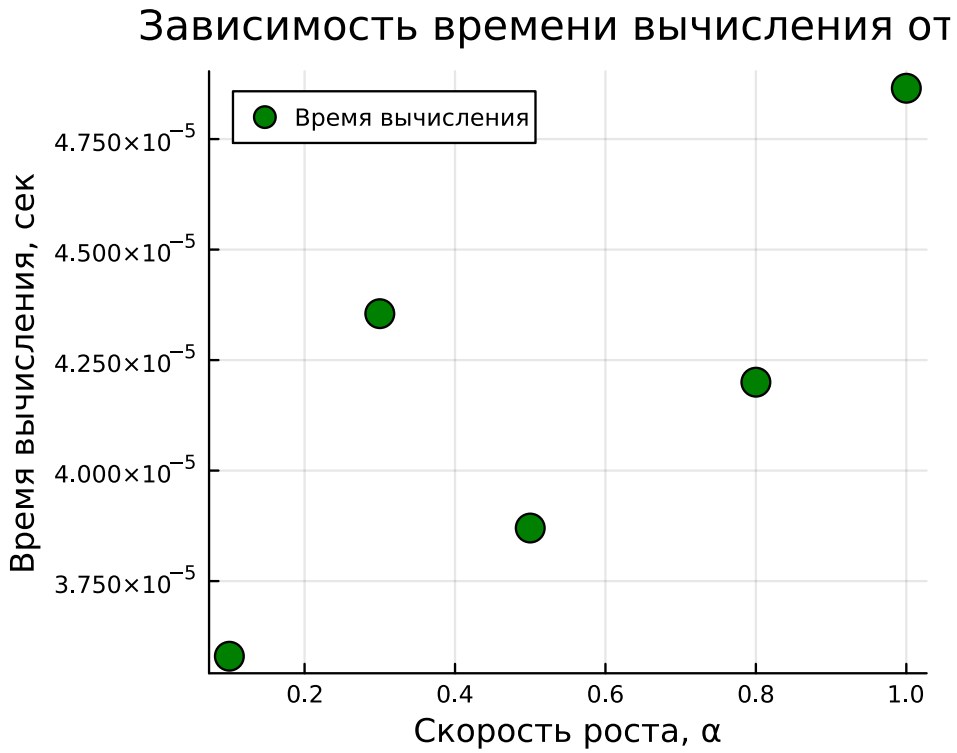
ИЗМЕНЕНИЕ: Бенчмаркинг для разных значений α

```
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


============================================================
Бенчмаркинг для разных значений α
============================================================

Бенчмарк для α = 0.1:
  Среднее время: 0.0 сек

Бенчмарк для α = 0.3:
  Среднее время: 0.0 сек

Бенчмарк для α = 0.5:
  Среднее время: 0.0 сек

Бенчмарк для α = 0.8:
  Среднее время: 0.0 сек

Бенчмарк для α = 1.0:
  Среднее время: 0.0 сек
```

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"))
```

```
"C:\\Users\\12232\\Documents\\GitHub\\2026-1--study--simulation-
modeling\\labs\\lab01\\project\\plots\\computation_time_vs_alpha.png"
```

6.11 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)")


============================================================
ЛАБОРАТОРНАЯ РАБОТА ЗАВЕРШЕНА
============================================================

Результаты сохранены в:
  • data//single/              - базовый эксперимент
  • data//parametric_scan/     - параметрическое сканирование
  • data//all_results.jld2     - сводные данные
  • plots//                    - все графики
  • data//all_plots.jld2      - объекты графиков

Для анализа результатов используйте:
  using JLD2, DataFrames
  @load "data//all_results.jld2"
  println(results_df)
```

7. Выводы

В результате выполнения данной лабораторной работы, я вспомнил основные методы работы с системой Git и разными платформами, а также получил практические навыки работы с Julia.

8. Список литературы

Лабораторная работа №1