**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра вычислительной математики**

**СТАБИЛИЗИРОВАННЫЕ ДВУХШАГОВЫЕ МЕТОДЫ РУНГЕ—КУТТЫ ВТОРОГО ПОРЯДКА**

Курсовая работа

Карпушевич Анастасии Владимировны

студентки 3 курса,

специальность «прикладная математика»

Научный руководитель:

кандидат физ.-мат. наук,

доцент Б.В. Фалейчик

Минск, 2024

Реферат

Курсовой проект, 29 с., 3 источника, 5 рис., 8 таблиц, 2 прил.

СТАБИЛИЗИРОВАННЫЕ ДВУХШАГОВЫЕ МЕТОДЫ РУНГЕ—КУТТЫ ВТОРОГО ПОРЯДКА

*Объект исследования* – стабилизированные двухшаговые методы Рунге-Кутты второго порядка.

*Цель работы* – исследовать на практике преимущества стабилизированных методов Рунге-Кутты второго порядка по сравнению с классическими методами при решении жестких задач.

*Методы исследования* – методы Рунге-Кутты и Эйлера.

*Результаты работы:* проведен вычислительный эксперимент, сравнивающий эффективность и точность различных численных методов; написана программа, реализующая методы Рунге-Кутты и Эйлера для решения жестких дифференциальных уравнений; проведено сравнение методов по погрешности и трудоемкости, для подтверждения преимущества стабилизированных методов Рунге-Кутты второго порядка.

Рэферат

Курсавы праект, 29 с., 3 крыніцы, 5 мал., 8 табліц, 2 дадаткі.

СТАБІЛІЗАВАНЫЯ ДВУХШАГОВЫЕ МЕТАДЫ РУНГЕ-КУТТА ДРУГОГА ПАРАДКУ

*Аб'ект даследавання* – стабілізаваныя двухшаговые метады Рунге-Кутта другога парадку.

*Мэта работы* – даследаваць на практыцы перавагі стабілізаваных метадаў Рунге-Кутта другога парадку ў параўнанні з класічнымі метадамі пры вырашэнні жорсткіх задач.

*Метады даследавання* – метады Рунге-Кутта і Эйлера.

*Вынікі работы:* праведзены вылічальны эксперымент, які параўноўвае эфектыўнасць і дакладнасць розных лікавых метадаў; напісана праграма, якая рэалізуе метады Рунге-Кутта і Эйлера для вырашэння жорсткіх дыферэнцыяльных раўнанняў; праведзена параўнанне метадаў па хібнасці і працаёмкасці, для пацверджання перавагі стабілізаваных метадаў Рунге-Кутта другога парадку.

Summary

Course project, 29 p., 3 sources, 5 fig., 8 tables, 2 applications.

STABILIZED TWO—STEP RUNGE-KUTTA METHODS OF THE SECOND ORDER

*Object of research* – stabilized two-step Runge-Kutta methods of the second order.

*Purpose of the work* – to investigate in practice the advantages of second-order stabilized Runge-Kutta methods in comparison with classical methods for solving hard problems.

*Methods of research* – Runge-Kutta and Euler methods.

*Results of the work*: a computational experiment was conducted comparing the efficiency and accuracy of various numerical methods; a program was written that implements the Runge-Kutta and Euler methods for solving rigid differential equations; a comparison of methods in error and labor intensity was carried out to confirm the advantages of stabilized second-order Runge-Kutta methods.

Оглавление

[Введение 5](#_Toc166675661)

[1. Классические явыные методы Рунге-Кутты 6](#_Toc166675662)

[1.1 Общий вид методов 6](#_Toc166675663)

[1.2 Вывод условий порядка 7](#_Toc166675664)

[1.3 Абсолютная устойчивость 8](#_Toc166675665)

[2. Стабилизированные двушаговые методы Рунге-Кутты 9](#_Toc166675666)

[2.1 Стабилизированные одношаговые методы Рунге-Кутты 9](#_Toc166675667)

[2.2 Стабилизированные двушаговые методы Рунге-Кутты 9](#_Toc166675668)

[3. Вычислительный эксперимент 11](#_Toc166675669)

[3.1 Организация вычислительного эксперимента 11](#_Toc166675670)

[3.2 Вывод вычислительного эксперимента 20](#_Toc166675671)

[Заключение 22](#_Toc166675672)

[Список использованной литературы 23](#_Toc166675673)

[Приложения 24](#_Toc166675674)

# Введение

В условиях современного мира, где многие технические и научные задачи требуют численного решения, методы Рунге-Кутты играют важную роль. От простых обыкновенных дифференциальных уравнений (ОДУ) до сложных задач моделирования физических, биологических или инженерных процессов, эти методы остаются одним из самых популярных инструментов в арсенале инженеров и ученых. Однако, несмотря на широкое использование классических методов Рунге-Кутты, они имеют определенные ограничения, особенно при работе с жесткими системами.

Для решения таких систем часто требуется уменьшать шаг интегрирования, что может привести к увеличению времени расчетов и снижению эффективности. В этом контексте стабилизированные методы Рунге-Кутты, включая двухшаговые методы второго порядка, становятся очень востребованными. Они позволяют решать жесткие системы с большей стабильностью и при больших шагах, что делает их привлекательным выбором для широкого спектра применений.

Целью данной курсовой работы является изучение стабилизированных двухшаговых методов Рунге-Кутты второго порядка, оценка их эффективности и демонстрация их преимуществ по сравнению с классическими методами. Для достижения этой цели мы рассмотрим общие принципы работы классических методов Рунге-Кутты. Также будет проведен вычислительный эксперимент, чтобы подтвердить теоретические предположения и сравнить эффективность стабилизированных методов с классическими.

Таким образом, в рамках данной работы мы изучим структуру стабилизированных методов Рунге-Кутты, оценим их устойчивость, а также продемонстрируем, как эти методы могут быть применены к решению реальных задач, требующих численных расчетов.

## Классические явыные методы Рунге-Кутты

### Общий вид методов

Пусть s — это целое положительное число («число стадий», или «этапов») и a21, a31, a32, …, as1, as2, as,s-1, …, b1, …, bs, c2, …, cs — вещественные коэффиценты. Тогда метод

Называется s-стадийным (s-этапным) явным методом Рунге-Кутты для задачи

.

Обычно коэффиценты ci удовлетворяют условиям

или, короче,

Смысл этих условий в том, что все точки, в которых вычисляется f, являются приближениями первого порядка к решению. Эти условия сильно упрощают вывод условий, определяющих порядок аппроксимации для методов высокого порядка.

Явные методы являются наиболее простыми в реализации методами типа Рунге-Кутты: вспомогательные значения вычисляются последовательно, каждое последующее из них явно выражается через уже найденные.

### Вывод условий порядка

Классическим показателем точности методов численного интегрирования ОДУ является порядок метода. Это понятие связано с разложением точного и приближённого решений в ряд Тейлора.

Для начала рассматривается y в скалярном случае и предполагается, что оно достаточно гладкое. Затем вводятся следующие обозначения:

Разложение Тейлора для в точке имеет вид

Тот факт, что y удовлетворяет изначальной системе ОДУ, позволяет вычислить неизвестные коэффициенты для любого k. Действительно, мы имеем

и так далее. Таким образом, мы имеем теоретическую возможность точно вычислить разложение Тейлора для в точке .

Теперь зафиксируем начальное условие , рассмотрим произвольный одношаговый метод Φ и η — функцию этого метода. Традиционный способ оценки точности такого приближения заключается в сравнении разложений Тейлора для η(x) c разложением для в точке . Чем больше членов в этих разложениях совпадают, тем выше порядок метода.[1]

Условия третьего порядка методов Рунге-Кутта:

### Абсолютная устойчивость

Абсолютную устойчивость можно рассматривать как минимальное свойство, которое должно выполняться для любого метода интегрирования.

Она часто применяется для предсказания устойчивого поведения метода при решении нелинейной задачи общего вида.

Рассмотрим скалярное уравнение:

Любой метод Рунге-Кутты, применяемый к решению данного уравнения записывается в виде:

,

где — полином или рациональная функция с вещественными коэффициентами. Функцию R называют функцией устойчивости метода.

Метод называется абсолютно устойчивым для , если для этого z выполняется неравенство

Из него следует, что

для любой пары , удовлетворяющей условию [2]

## Стабилизированные двушаговые методы Рунге-Кутты

Стабилизированные методы Рунге-Кутты были разработаны, чтобы решить проблему устойчивости при решении жестких обыкновенных дифференциальных уравнений (ОДУ).

### Стабилизированные двушаговые методы Рунге-Кутты

Стабилизированные двухшаговые методы Рунге-Кутты — это обобщение классических методов Рунге-Кутты, в котором используются два шага для вычисления нового значения решения. Первый шаг используется для оценки промежуточного значения решения, а второй шаг — для вычисления окончательного значения.

#### Общий вид

,

,

,

,

Где

где h — шаг интегрирования, s — количество стадий.

Условия порядка:

#### Построение стабилизированных методов

Для анализа устойчивости двухшаговых методов используется характеристическое уравнение, которое описывает поведение погрешностей в методе. Уравнение имеет вид:

Оптимизация длины области устойчивости подразумевает выбор таких параметров метода, которые максимизируют интервал устойчивости, т.е. диапазон значений шага h, при которых метод остаётся устойчивым.

Под демпфированием численного метода решения задачи Коши понимается такая модификация его области устойчивости, которая обеспечивает ненулевую ширину этой области вдоль всего интервала устойчивости.

## Вычислительный эксперимент

### Организация вычислительного эксперимента

1. Фиксируем λ и конечную точку X;
2. Выбираем метод;
3. Задаем последовательность сеток (т.е последовательность )
4. Для каждого j вычисляем погрешность и трудоемкость (число вычислений функции )
5. Составить таблицы и построить графики и

Методы, которые будут использоваться в процессе вычислительного эксперимента:

1. Метод Эйлера
2. Одношаговый метод Рунге-Кутты второго порядка
3. Одношаговый метод Рунге-Кутты четвертого порядка (1)
4. Двушаговый метод Рунге-Кутты второго порядка

Рассмотрим задачу

Фиксируем :

Выбираем конечную точку X: ,

**Метод Эйлера:**

Если мы вычислим: , то мы мы найдем производную y' в начальной точке. Для достаточно малой Δx, мы можем предположить значение y как

.

Или кратко: .

А для общего случая:

Мы продолжаем вычислять следующие значения y используя это выражения до тех пор пока мы не достигнем точки x .

Задаем последовательность сеток:

Были взяты 10 различных шагов:

0.1, 0.08, 0.039, 0.01, 0.009, 0.0075, 0.005, 0.002, 0.0005, 0.0002.

Функция устойчивости для метода Эйлера:

Решаем уравнение

Это дает решение z = -2. Таким образом, интервал устойчивости:

Для нашего уравнения, где

**Одношаговый метод Рунге-Кутты второго порядка:**

Алгоритм метода Рунге-Кутта второго порядка можно разделить на следующие этапы:

,

.

Где h — размер шага.

,

Задаем последовательность сеток:

Были взяты 10 различных шагов:

0.1, 0.08, 0.039, 0.01, 0.009, 0.0075, 0.005, 0.002, 0.0005, 0.0002.

Функция устойчивости для одношагового метода Рунге-Кутты 2-го порядка:

Решаем уравнение

Это дает корни z = 0, z = −2, z = -1 ± i. Наибольший по модулю вещественный корень здесь z = −2. Таким образом, интервал устойчивости:

Для нашего уравнения, где

**Одношаговый метод Рунге-Кутты четвертого порядка:**

Алгоритм метода Рунге-Кутта четвертого порядка можно разделить на следующие этапы:

,

.

Где h — размер шага.

Задаем последовательность сеток:

Были взяты 10 различных шагов:

0.1, 0.08, 0.039, 0.01, 0.009, 0.0075, 0.005, 0.002, 0.0005, 0.0002.

Функция устойчивости для одношагового метода Рунге-Кутты 4-го порядка:

Решаем уравнение

Таким образом, интервал устойчивости:

Для нашего уравнения, где

**Двушаговый метод Рунге-Кутты второго порядка:**

,

,

,

,

Где

[3]

Коэффиценты, используемые для вычисления при :

Задаем последовательность сеток:

Были взяты 10 различных шагов:

0.1, 0.08, 0.039, 0.01, 0.009, 0.0075, 0.005, 0.002, 0.0005, 0.0002. (3)

Интервал устойчивости:

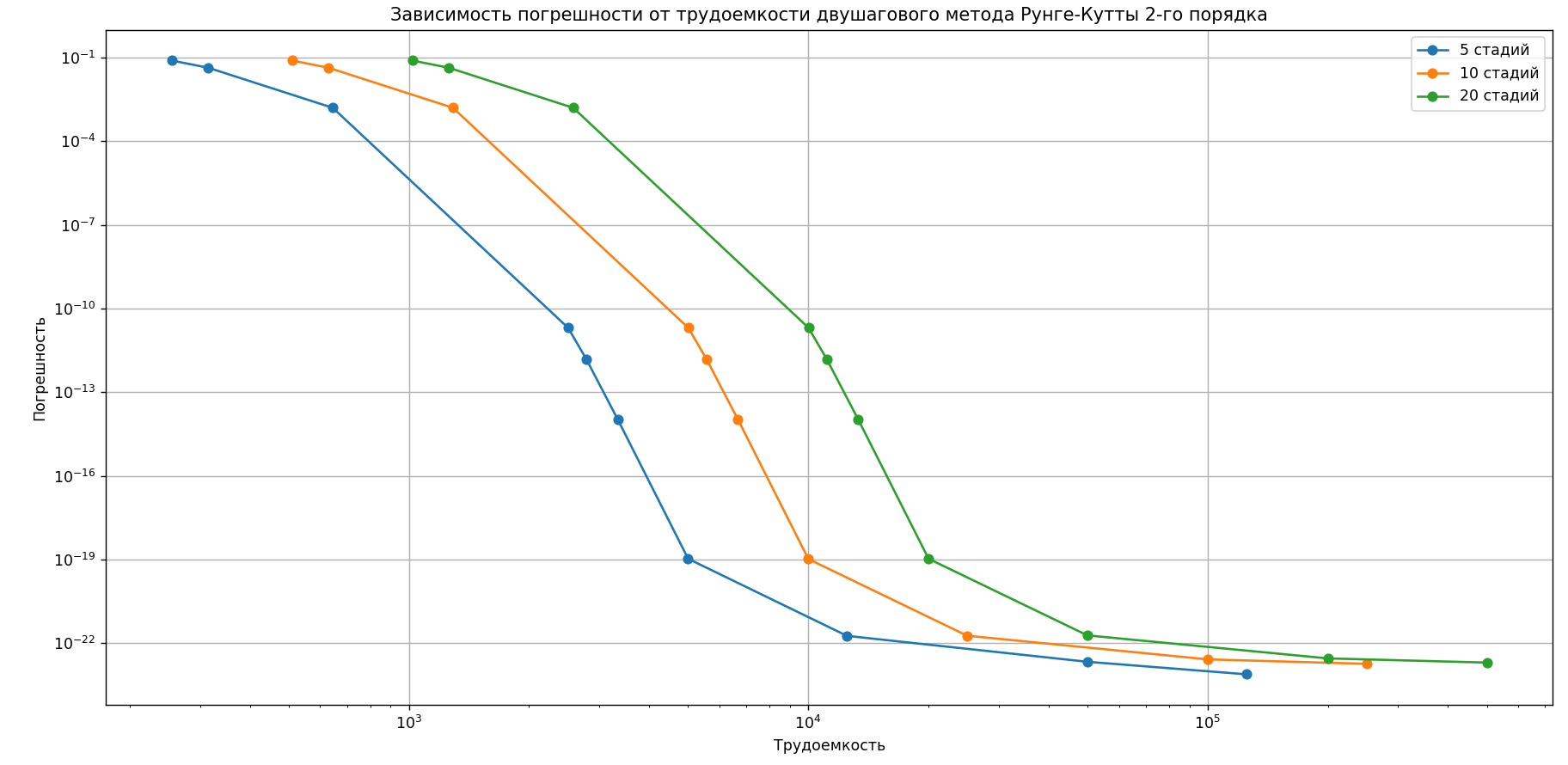


Рисунок 3.1 – вычислительный эксперимент на базе задачи (2) и шагах (3) для двушагового метода Рунге-Кутты 2-ого порядка при s = 5, 10, 20 и

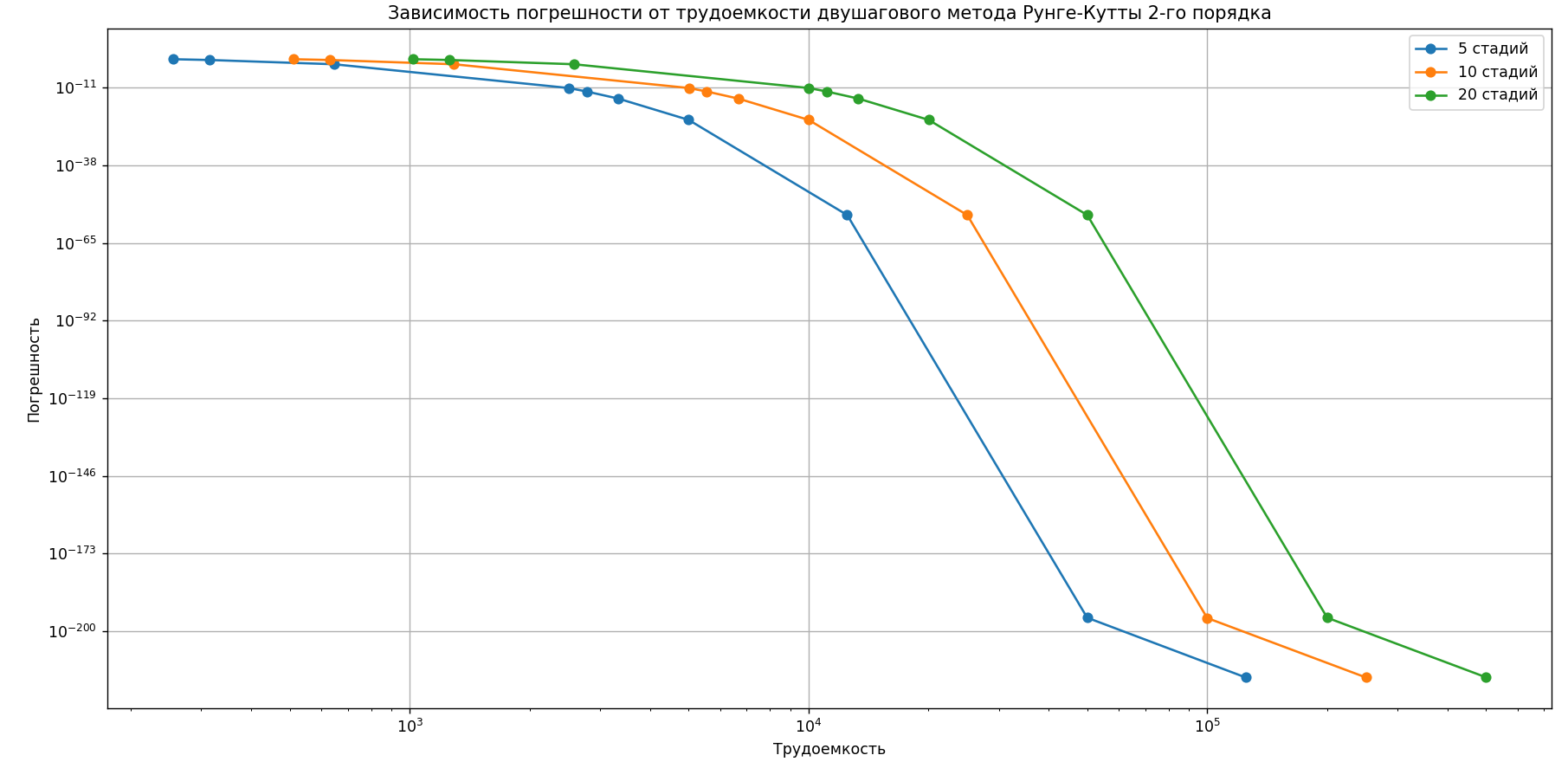


Рисунок 3.2 – вычислительный эксперимент на базе задачи (2) и шагах (3) для двушагового метода Рунге-Кутты 2-ого порядка при s = 5, 10, 20 и

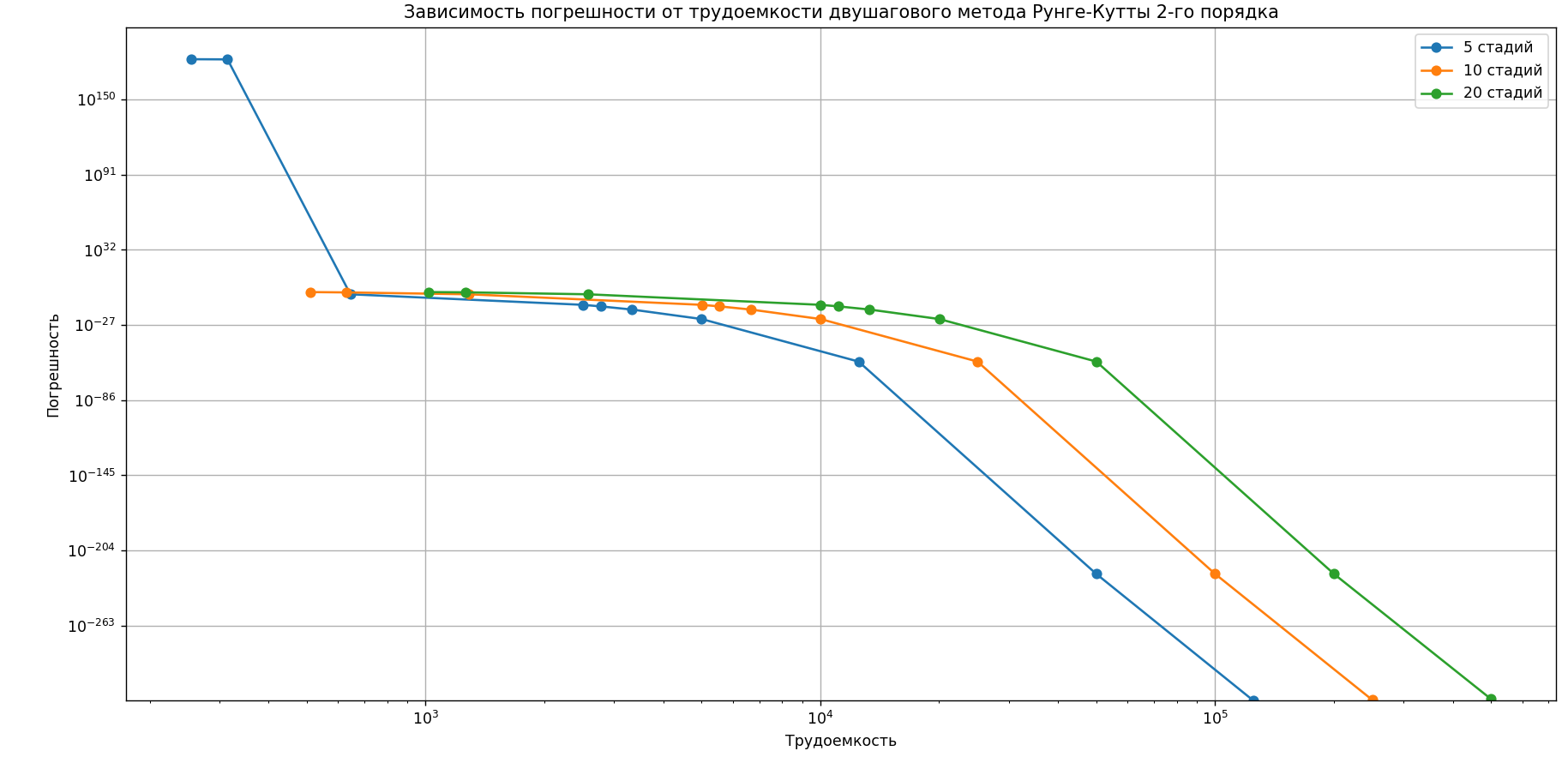


Рисунок 3.3 – вычислительный эксперимент на базе задачи (2) и шагах (3) для двушагового метода Рунге-Кутты 2-ого порядка при s = 5, 10, 20 и

Сравнение полученных результатов:

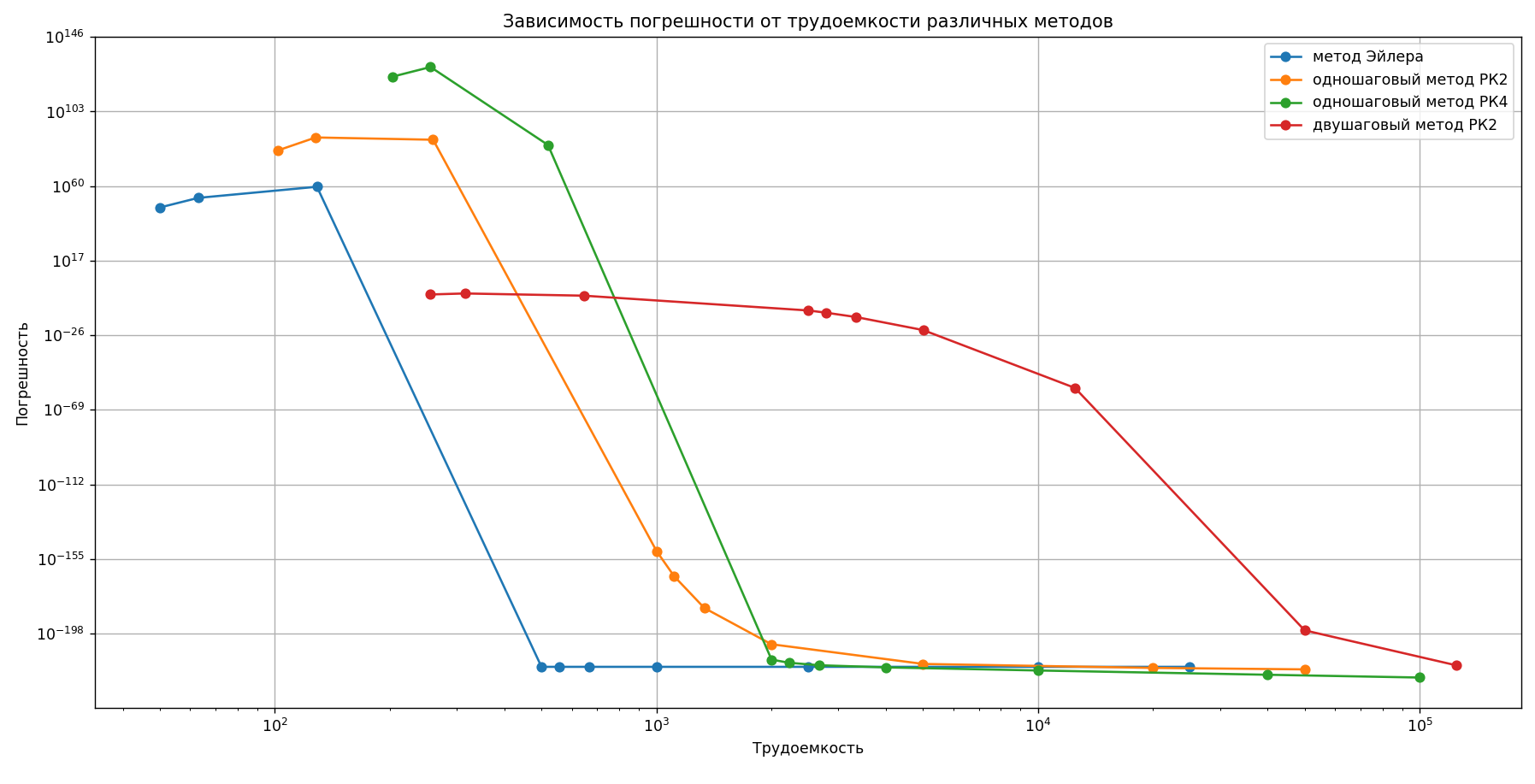


Рисунок 3.4 – вычислительный эксперимент на базе задачи (2) и шагах (3) для методов (0) при

Метод был реализован на языке Python. Текст программы приведен в приложении А.

Данные вычислительного эксперимента:

1. Метод Эйлера

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 5.15377521e+047 | 50 |
| 0.08 | 1.74251498e+053 | 63 |
| 0.039 | 4.46007055e+059 | 129 |
| 0.01 | 7.12457641e-218 | 500 |
| 0.009 | 7.12457641e-218 | 556 |
| 0.0075 | 7.12457641e-218 | 667 |
| 0.005 | 7.12457641e-218 | 1000 |
| 0.002 | 7.12457641e-218 | 2500 |
| 0.0005 | 7.12455919e-218 | 10000 |
| 0.0002 | 7.07971299e-218 | 25000 |

1. Одношаговый метод Рунге-Кутты 2-ого порядка

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 4.35705293e+080 | 102 |
| 0.08 | 1.17549435e+088 | 128 |
| 0.039 | 5.75694369e+086 | 260 |
| 0.01 | 3.05493636e-151 | 1002 |
| 0.009 | 1.07153469e-165 | 1114 |
| 0.0075 | 5.94832953e-184 | 1336 |
| 0.005 | 7.58607870e-205 | 2002 |
| 0.002 | 3.35352068e-216 | 5002 |
| 0.0005 | 1.72042256e-218 | 20002 |
| 0.0002 | 2.45201754e-219 | 50002 |

1. Одношаговый метод Рунге-Кутты 4-ого порядка

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 1.56548694e+123 | 204 |
| 0.08 | 4.90370754e+128 | 256 |
| 0.039 | 5.45790647e+083 | 520 |
| 0.01 | 1.03658284e-213 | 2004 |
| 0.009 | 1.57982551e-215 | 2228 |
| 0.0075 | 5.86636475e-217 | 2672 |
| 0.005 | 3.46056486e-218 | 4004 |
| 0.002 | 5.63549336e-220 | 10004 |
| 0.0005 | 1.93437230e-222 | 40004 |
| 0.0002 | 4.82956362e-224 | 100004 |

1. Двушаговый метод Рунге-Кутты 2-ого порядка

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 3.45109171e-003 | 255 |
| 0.08 | 1.34286978e-002 | 315 |
| 0.039 | 7.18816178e-004 | 645 |
| 0.01 | 2.06313772e-012 | 2505 |
| 0.009 | 1.15475317e-013 | 2780 |
| 0.0075 | 3.49757654e-016 | 3335 |
| 0.005 | 1.02677644e-023 | 5005 |
| 0.002 | 4.55937318e-057 | 12505 |
| 0.0005 | 7.92424812e-197 | 50005 |
| 0.0002 | 5.00515542e-217 | 125005 |

При для достаточно больших шагов (h = 0.1, 0.08, 0.039 в данном случае) погрешность при вычислении нестабилизированными методами (методом Эйлера и одношаговым методом Рунге-Кутты 4-ого порядка) получается очень большой. В сравнении с нестабилизированными методами, даже на достаточно больших шагах, двушаговый метод Рунге-Кутты 2-ого порядка показывает лучшие результаты. Поэтому, несмотря на то, что трудоемкость у данного метода выше, чем у предыдущих двух, его использование целесообразнее для достижения точных результатов на достаточно больших шагах.

Рассмотрим задачу:

Общий вид данной задачи:

Решение данной задачи:

Фиксируем :

Выбираем конечную точку X: ,

Задаем последовательность сеток:

Были взяты 10 различных шагов:

0.1, 0.08, 0.039, 0.01, 0.009, 0.0075, 0.005, 0.002, 0.0005, 0.0002.

Сравнение полученных результатов:

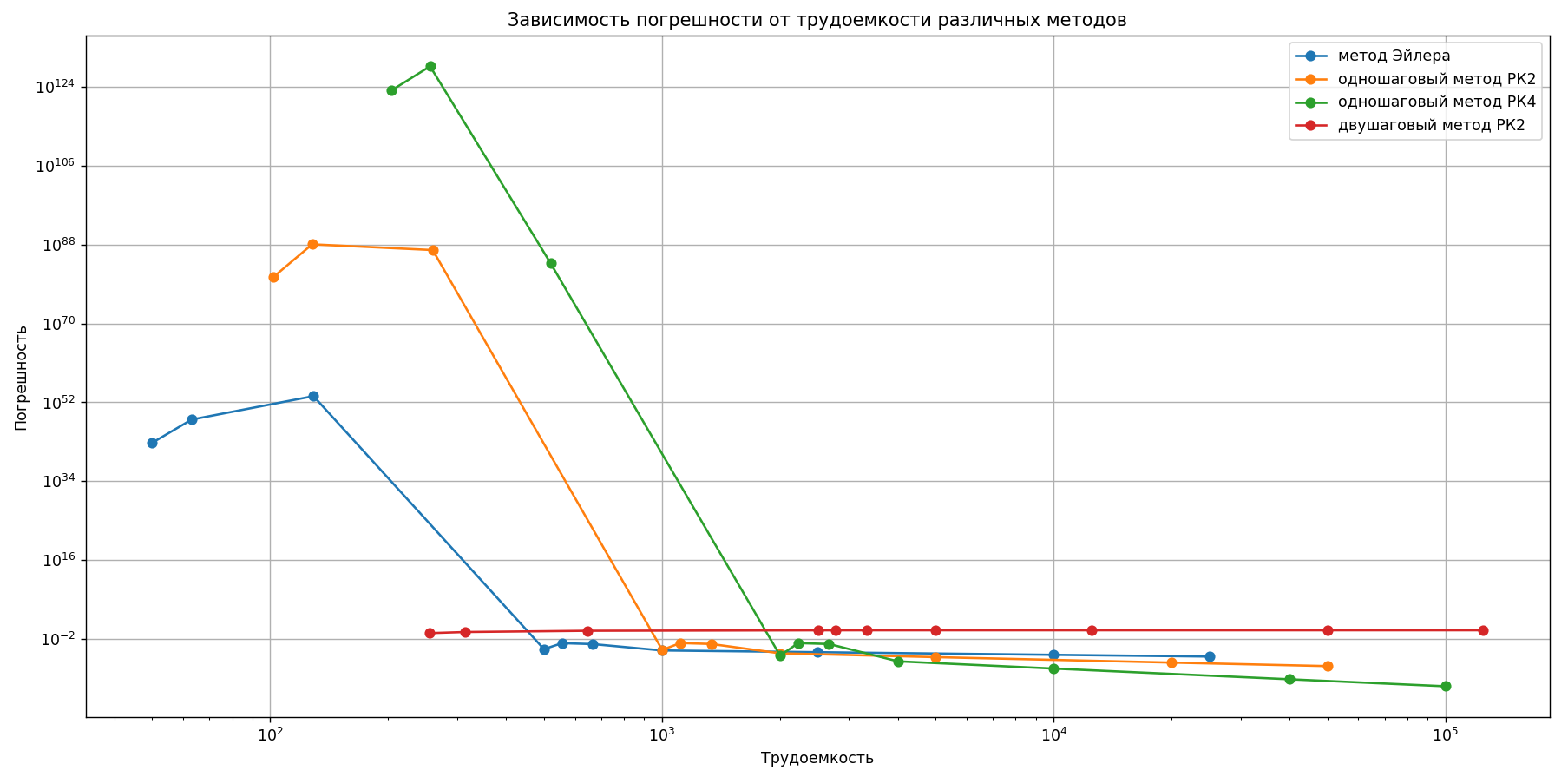
****

Рисунок 3.5 – вычислительный эксперимент на базе задачи (4) и шагах (3) для методов (0) при

Метод был реализован на языке Python. Текст программы приведен в приложении Б.

Данные вычислительного эксперимента:

1. Метод Эйлера

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 6.01600053e+42 | 50 |
| 0.08 | 1.16226678e+48 | 63 |
| 0.039 | 2.61014061e+53 | 129 |
| 0.01 | 4.80395682e-05 | 500 |
| 0.009 | 1.09912772e-03 | 556 |
| 0.0075 | 6.76139471e-04 | 667 |
| 0.005 | 2.40307518e-05 | 1000 |
| 0.002 | 9.61491352e-06 | 2500 |
| 0.0005 | 2.40405362e-06 | 10000 |
| 0.0002 | 9.61647422e-07 | 25000 |

1. Одношаговый метод Рунге-Кутты 2-ого порядка

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 4.35700807e+80 | 102 |
| 0.08 | 1.17548553e+88 | 128 |
| 0.039 | 5.75691422e+86 | 260 |
| 0.01 | 3.21240072e-05 | 1002 |
| 0.009 | 1.16593546e-03 | 1114 |
| 0.0075 | 7.26583604e-04 | 1336 |
| 0.005 | 5.34592789e-06 | 2002 |
| 0.002 | 7.12479337e-07 | 5002 |
| 0.0005 | 4.10984889e-08 | 20002 |
| 0.0002 | 6.47596154e-09 | 50002 |

1. Одношаговый метод Рунге-Кутты 4-ого порядка

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 1.56547550e+123 | 204 |
| 0.08 | 4.90368091e+128 | 256 |
| 0.039 | 5.45787517e+083 | 520 |
| 0.01 | 1.60321032e-006 | 2004 |
| 0.009 | 1.14332005e-003 | 2228 |
| 0.0075 | 7.12603027e-004 | 2672 |
| 0.005 | 7.95740638e-008 | 4004 |
| 0.002 | 1.76706272e-009 | 10004 |
| 0.0005 | 6.41287023e-012 | 40004 |
| 0.0002 | 1.61648472e-013 | 100004 |

1. Двушаговый метод Рунге-Кутты 2-ого порядка

|  |  |  |
| --- | --- | --- |
| Шаг | Погрешность | Трудоемкость |
| 0.1 | 0.21246593 | 255 |
| 0.08 | 0.38482918 | 315 |
| 0.039 | 0.73661664 | 645 |
| 0.01 | 0.94054656 | 2505 |
| 0.009 | 0.94540773 | 2780 |
| 0.0075 | 0.94995197 | 3335 |
| 0.005 | 0.9543182 | 5005 |
| 0.002 | 0.95819125 | 12505 |
| 0.0005 | 0.95887867 | 50005 |
| 0.0002 | 0.95891699 | 125005 |

### Вывод вычислительного эксперимента

Результаты показали, что:

Метод Эйлера:

* Показал огромные погрешности при больших шагах (например, h = 0.1 и h = 0.08), демонстрируя явную неустойчивость.
* Для очень малых шагов (h ≤ 0.01), погрешности уменьшались до приемлемых значений, но при этом значительно возрастала трудоемкость.

Одношаговый метод Рунге-Кутты 2-го порядка:

* Также показал большие погрешности при больших шагах, даже выше, чем у метода Эйлера.
* С уменьшением шага погрешности уменьшались, но трудоемкость существенно возрастала, превышая трудоемкость метода Эйлера.

Одношаговый метод Рунге-Кутты 4-го порядка:

* При больших шагах погрешности оставались высокими, но при уменьшении шага до h ≤ 0.01 метод показывал значительное уменьшение погрешностей.
* Однако, трудоемкость была значительно выше, чем у методов 2-го порядка.

Двухшаговый метод Рунге-Кутты 2-го порядка:

* Показал существенно лучшие результаты по сравнению с классическими методами.
* При больших шагах (h = 0.1, h = 0.08) погрешности были намного меньше, чем у остальных методов.
* Несмотря на увеличенную трудоемкость, метод обеспечивал высокую точность даже при более крупных шагах.

Стабилизированные методы демонстрировали значительно меньшие погрешности при больших шагах по сравнению с классическими методами Рунге-Кутты.

# Заключение

Проведенные вычислительные эксперименты подтвердили теоретические ожидания относительно стабилизированных двухшаговых методов Рунге-Кутты 2-го порядка.

Основные выводы:

Устойчивость: Стабилизированные двухшаговые методы значительно улучшили устойчивость решений при больших шагах по сравнению с классическими методами, что особенно важно при решении жестких задач.

Точность: Эти методы продемонстрировали меньшие погрешности при различных значениях шага, обеспечивая более точные результаты.

Трудоемкость: Хотя стабилизированные методы оказались более трудоемкими, их использование оправдано благодаря высокой точности и устойчивости, особенно при решении жестких задач, где классические методы часто терпят неудачу.

Таким образом, стабилизированные двухшаговые методы Рунге-Кутты 2-го порядка являются предпочтительным выбором для решения жестких дифференциальных уравнений, где необходима высокая точность и устойчивость. Проведенные эксперименты демонстрируют их преимущества и подтверждают целесообразность их использования в практических задачах.

# Список использованной литературы

1. Фалейчик Б. В. «Одношаговые методы численного решения задачи Коши : учеб.- метод. Пособие» / Б. В. Фалейчик. — Минск : БГУ, 2010.— 42 с.
2. Деккер К., Вервер Я., «Устойчивость методов Рунге-Кутты для жестких нелинейных дифференциальных уравнений», М.: Мир, 1988. 334 с.
3. Moisa A. V., Faleichik B. V. «Second order stabilized two-step Runge-Kutta methods», Journal of Computational and Applied Mathematics, 2023.— 13 p.

# Приложения

Приложение А

Код программы на языке Python

import numpy as np

import matplotlib.pyplot as plt

l = -100

x0 = 0

x\_end = 5

h = 0.01

hs = [0.1, 0.08, 0.039, 0.01, 0.009, 0.0075, 0.005, 0.002, 0.0005, 0.0002]

y0 = 1

def y\_al(x):

return np.exp(l \* x)

def f(x, y):

return l \* y

def eiler(f, y0, x0, h):

errors = np.zeros(len(h))

f\_evals = np.zeros(len(h))

for k in range(len(h)):

x\_values = np.arange(x0, x\_end + h[k], h[k])

y\_values = np.zeros\_like(x\_values)

y\_values[0] = y0

for i in range(1, len(x\_values)):

dy = f(x\_values[i - 1], y\_values[i - 1]) \* h[k]

y\_values[i] = y\_values[i - 1] + dy

f\_evals[k] += 1

errors[k] = abs(y\_al(x\_end) - y\_values[len(y\_values)-1])

return errors, f\_evals

def rk1\_2\_step(f, hs):

errors = np.zeros(len(hs))

f\_evals = np.zeros(len(hs))

for k in range(len(hs)):

x\_values = np.arange(x0, x\_end + hs[k], hs[k])

y\_values = np.zeros\_like(x\_values)

y\_values[0] = 1

for i in range(1, len(x\_values)):

k1 = f(x\_values[i - 1], y\_values[i - 1])

k2 = f(x\_values[i - 1] + 2 / 3 \* hs[k], y\_values[i - 1] + 2 / 3 \* k1 \* hs[k])

y\_values[i] = y\_values[i - 1] + (hs[k] / 4) \* (k1 + 3 \* k2)

f\_evals[k] = len(x\_values) \* 2

errors[k] = abs(y\_al(x\_end) - y\_values[len(y\_values) - 1])

return errors, f\_evals

def rk4\_step(f, hs):

errors = np.zeros(len(hs))

f\_evals = np.zeros(len(hs))

for k in range(len(hs)):

x\_values = np.arange(x0, x\_end + hs[k], hs[k])

y\_values = np.zeros\_like(x\_values)

y\_values[0] = 1

for i in range(1, len(x\_values)):

k1 = hs[k] \* f(x\_values[i - 1], y\_values[i - 1])

k2 = hs[k] \* f(x\_values[i - 1] + 0.5 \* hs[k], y\_values[i - 1] + 0.5 \* k1)

k3 = hs[k] \* f(x\_values[i - 1] + 0.5 \* hs[k], y\_values[i - 1] + 0.5 \* k2)

k4 = hs[k] \* f(x\_values[i - 1] + hs[k], y\_values[i - 1] + k3)

y\_values[i] = y\_values[i - 1] + (1 / 6) \* (k1 + 2 \* k2 + 2 \* k3 + k4)

f\_evals[k] = len(x\_values) \* 4

errors[k] = abs(y\_al(x\_end) - y\_values[len(y\_values) - 1])

return errors, f\_evals

def two\_step\_rk2(f, y0, x0, h, s):

if s == 5:

a\_tilde = 19.991085619464535

a = 0.950022296412323

b = 0.04997770358767691

m = [1.9918588786954916, 1.9838492426656018, 1.9760315849167438, 1.9684604922450784]

m\_tilde = [0.04203714921461939, 0.08373206889818684, 0.08339536663324355, 0.08306673458794599, 0.08274846743558949]

c = [18.991085619464535, 19.033122768679153, 19.158549757260907, 19.365346371620134, 19.65025313347653]

if s == 10:

a\_tilde = 19.99120407390261353

a = 0.950022

b = 0.0499781

m = [1.99795, 1.99592, 1.99389, 1.99188, 1.98989, 1.98793, 1.98599, 1.98408, 1.98221]

m\_tilde = [0.0105172, 0.0210129, 0.0209915, 0.0209702, 0.020949, 0.0209281, 0.0209074, 0.0208871, 0.020867, 0.0208473]

c = [18.9912, 19.0017, 19.0333, 19.0856, 19.1586, 19.252, 19.3654, 19.4983, 19.6502, 19.8206]

if s == 20:

a\_tilde = 19.991085619464535

a = 0.950022

b = 0.0499781

m = [1.99949, 1.99898, 1.99847, 1.99796, 1.99745, 1.99694, 1.99643, 1.99593, 1.99543, 1.99493, 1.99444, 1.99394, 1.99345, 1.99297, 1.99249, 1.99201, 1.99153, 1.99106, 1.9906]

m\_tilde = [0.0026298, 0.00525825, 0.0052569, 0.00525556, 0.00525422, 0.00525288, 0.00525155, 0.00525022, 0.0052489, 0.00524758, 0.00524627, 0.00524496, 0.00524367, 0.00524238, 0.0052411, 0.00523984, 0.00523858, 0.00523733, 0.00523609, 0.00523487]

c = [18.9913, 18.9939, 19.0018, 19.0149, 19.0333, 19.0569, 19.0857, 19.1196, 19.1587, 19.2028, 19.252, 19.3062, 19.3654, 19.4294, 19.4983, 19.5719, 19.6502, 19.7331, 19.8206, 19.9125]

errors = np.zeros(len(h))

f\_evals = np.zeros(len(h))

for k in range(len(h)):

y1 = np.exp(l \* h[k])

v = np.zeros(s + 1)

n\_steps = int((x\_end - x0) / h[k]) + 1

y = np.zeros(n\_steps)

y[0] = y0

v[0] = y0

y[1] = y1

for i in range(1, n\_steps - 1):

v[0] = a\_tilde \* y[i] + (1 - a\_tilde) \* y[i - 1]

v[1] = v[0] + h[k] \* m\_tilde[0] \* f(x0 + h[k] \* i + c[0] \* h[k], v[0])

for j in range (2, s + 1):

v[j] = m[j - 2] \* v[j - 1] + (1 - m[j - 2]) \* v[j - 2] + h[k] \* m\_tilde[j - 1] \* f(x0 + h[k] \* i + c[j - 1] \* h[k], v[j - 1])

y[i + 1] = a \* y[i] + b \* v[s]

f\_evals[k] = s \* n\_steps

y\_al = np.exp(l \* (x\_end))

errors[k] = abs(y\_al - y[n\_steps - 1])

return errors, f\_evals

error\_values\_eiler, workload\_values\_eiler = eiler(f, y0, x0, hs)

error\_values\_rk1\_2, workload\_values\_rk1\_2 = rk1\_2\_step(f, hs)

error\_values\_rk4, workload\_values\_rk4 = rk4\_step(f, hs)

error\_values\_rk2, workload\_values\_rk2 = two\_step\_rk2(f, y0, x0, hs, 5)

plt.figure(figsize=(8, 6))

plt.grid(True)

plt.xscale('log')

plt.yscale('log')

plt.xlabel("Трудоемкость")

plt.ylabel("Погрешность")

plt.title("Зависимость погрешности от трудоемкости различных методов")

plt.plot(workload\_values\_eiler, error\_values\_eiler, 'o-', label='метод Эйлера')

plt.plot(workload\_values\_rk1\_2, error\_values\_rk1\_2, 'o-', label='одношаговый метод РК2')

plt.plot(workload\_values\_rk4, error\_values\_rk4, 'o-', label='одношаговый метод РК4')

plt.plot(workload\_values\_rk2, error\_values\_rk2, 'o-', label='двушаговый метод РК2')

plt.tight\_layout()

plt.legend()

plt.show()

Приложение Б

Код программы на языке Python

import numpy as np

import math

import matplotlib.pyplot as plt

l = -100

x0 = 0

x\_end = 5

h = 0.01

hs = [0.1, 0.08, 0.039, 0.01, 0.009, 0.0075, 0.005, 0.002, 0.0005, 0.0002]

y0 = 0

def y\_al(x):

return math.sin(x)

def f(x, y):

return l \* (y - math.sin(x)) + math.cos(x)

def eiler(f, y0, x0, h):

errors = np.zeros(len(h))

f\_evals = np.zeros(len(h))

for k in range(len(h)):

x\_values = np.arange(x0, x\_end + h[k], h[k])

y\_values = np.zeros\_like(x\_values)

y\_values[0] = y0

for i in range(1, len(x\_values)):

dy = f(x\_values[i - 1], y\_values[i - 1]) \* h[k]

y\_values[i] = y\_values[i - 1] + dy

f\_evals[k] += 1

errors[k] = abs(y\_al(x\_end) - y\_values[len(y\_values)-1])

return errors, f\_evals

def rk1\_2\_step(f, hs):

errors = np.zeros(len(hs))

f\_evals = np.zeros(len(hs))

for k in range(len(hs)):

x\_values = np.arange(x0, x\_end + hs[k], hs[k])

y\_values = np.zeros\_like(x\_values)

y\_values[0] = 1

for i in range(1, len(x\_values)):

k1 = f(x\_values[i - 1], y\_values[i - 1])

k2 = f(x\_values[i - 1] + 2 / 3 \* hs[k], y\_values[i - 1] + 2 / 3 \* k1 \* hs[k])

y\_values[i] = y\_values[i - 1] + (hs[k] / 4) \* (k1 + 3 \* k2)

f\_evals[k] = len(x\_values) \* 2

errors[k] = abs(y\_al(x\_end) - y\_values[len(y\_values) - 1])

return errors, f\_evals

def rk4\_step(f, hs):

errors = np.zeros(len(hs))

f\_evals = np.zeros(len(hs))

for k in range(len(hs)):

x\_values = np.arange(x0, x\_end + hs[k], hs[k])

y\_values = np.zeros\_like(x\_values)

y\_values[0] = 1

for i in range(1, len(x\_values)):

k1 = hs[k] \* f(x\_values[i - 1], y\_values[i - 1])

k2 = hs[k] \* f(x\_values[i - 1] + 0.5 \* hs[k], y\_values[i - 1] + 0.5 \* k1)

k3 = hs[k] \* f(x\_values[i - 1] + 0.5 \* hs[k], y\_values[i - 1] + 0.5 \* k2)

k4 = hs[k] \* f(x\_values[i - 1] + hs[k], y\_values[i - 1] + k3)

y\_values[i] = y\_values[i - 1] + (1 / 6) \* (k1 + 2 \* k2 + 2 \* k3 + k4)

f\_evals[k] = len(x\_values) \* 4

errors[k] = abs(y\_al(x\_end) - y\_values[len(y\_values) - 1])

return errors, f\_evals

def two\_step\_rk2(f, y0, x0, h, s):

if s == 5:

a\_tilde = 19.991085619464535

a = 0.950022296412323

b = 0.04997770358767691

m = [1.9918588786954916, 1.9838492426656018, 1.9760315849167438, 1.9684604922450784]

m\_tilde = [0.04203714921461939, 0.08373206889818684, 0.08339536663324355, 0.08306673458794599, 0.08274846743558949]

c = [18.991085619464535, 19.033122768679153, 19.158549757260907, 19.365346371620134, 19.65025313347653]

if s == 10:

a\_tilde = 19.99120407390261353

a = 0.950022

b = 0.0499781

m = [1.99795, 1.99592, 1.99389, 1.99188, 1.98989, 1.98793, 1.98599, 1.98408, 1.98221]

m\_tilde = [0.0105172, 0.0210129, 0.0209915, 0.0209702, 0.020949, 0.0209281, 0.0209074, 0.0208871, 0.020867, 0.0208473]

c = [18.9912, 19.0017, 19.0333, 19.0856, 19.1586, 19.252, 19.3654, 19.4983, 19.6502, 19.8206]

if s == 20:

a\_tilde = 19.991085619464535

a = 0.950022

b = 0.0499781

m = [1.99949, 1.99898, 1.99847, 1.99796, 1.99745, 1.99694, 1.99643, 1.99593, 1.99543, 1.99493, 1.99444, 1.99394, 1.99345, 1.99297, 1.99249, 1.99201, 1.99153, 1.99106, 1.9906]

m\_tilde = [0.0026298, 0.00525825, 0.0052569, 0.00525556, 0.00525422, 0.00525288, 0.00525155, 0.00525022, 0.0052489, 0.00524758, 0.00524627, 0.00524496, 0.00524367, 0.00524238, 0.0052411, 0.00523984, 0.00523858, 0.00523733, 0.00523609, 0.00523487]

c = [18.9913, 18.9939, 19.0018, 19.0149, 19.0333, 19.0569, 19.0857, 19.1196, 19.1587, 19.2028, 19.252, 19.3062, 19.3654, 19.4294, 19.4983, 19.5719, 19.6502, 19.7331, 19.8206, 19.9125]

errors = np.zeros(len(h))

f\_evals = np.zeros(len(h))

for k in range(len(h)):

y1 = math.sin(h[k])

v = np.zeros(s + 1)

n\_steps = int((x\_end - x0) / h[k]) + 1

y = np.zeros(n\_steps)

y[0] = y0

y[1] = y1

for i in range(1, n\_steps - 1):

v[0] = a\_tilde \* y[i] + (1 - a\_tilde) \* y[i - 1]

v[1] = v[0] + h[k] \* m\_tilde[0] \* f(x0 + h[k] \* i + c[0] \* h[k], v[0])

for j in range (2, s + 1):

v[j] = m[j - 2] \* v[j - 1] + (1 - m[j - 2]) \* v[j - 2] + h[k] \* m\_tilde[j - 1] \* f(x0 + h[k] \* i + c[j - 1] \* h[k], v[j - 1])

y[i + 1] = a \* y[i] + b \* v[s]

f\_evals[k] = s \* n\_steps

y\_al = np.exp(l \* (x\_end))

errors[k] = abs(y\_al - y[n\_steps - 1])

return errors, f\_evals

error\_values\_eiler, workload\_values\_eiler = eiler(f, y0, x0, hs)

error\_values\_rk1\_2, workload\_values\_rk1\_2 = rk1\_2\_step(f, hs)

error\_values\_rk4, workload\_values\_rk4 = rk4\_step(f, hs)

error\_values\_rk2, workload\_values\_rk2 = two\_step\_rk2(f, y0, x0, hs, 5)

plt.figure(figsize=(8, 6))

plt.grid(True)

plt.xscale('log')

plt.yscale('log')

plt.xlabel("Трудоемкость")

plt.ylabel("Погрешность")

plt.title("Зависимость погрешности от трудоемкости различных методов")

plt.plot(workload\_values\_eiler, error\_values\_eiler, 'o-', label='метод Эйлера')

plt.plot(workload\_values\_rk1\_2, error\_values\_rk1\_2, 'o-', label='одношаговый метод РК2')

plt.plot(workload\_values\_rk4, error\_values\_rk4, 'o-', label='одношаговый метод РК4')

plt.plot(workload\_values\_rk2, error\_values\_rk2, 'o-', label='двушаговый метод РК2')

plt.tight\_layout()

plt.legend()

plt.show()