**Санкт–Петербургский государственный университет**

***ТРОФИМОВ Кирилл Игоревич***

# Выпускная квалификационная работа

## *Сравнительное тестирование численных методов решения задачи Коши для систем обыкновенных дифференциальных уравнений*

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2020 «Прикладная математика, фундаментальная информатика и программирование»

Научный руководитель:
профессор, кафедра информационных систем,
д.ф. - м.н. Олемской Игорь Владимирович

Рецензент:
профессор, д.т.н. Ширунов Гурий Николаевич

Санкт-Петербург
2024 г.

# Содержание

# Введение

Задача Коши является одной из фундаментальных проблем в теории обыкновенных дифференциальных уравнений находя широкое применение в различных областях науки. Решение этой задачи зачастую требует использования численных методов, поскольку аналитические решения сложных систем редко бывают доступны. Поэтому разработка и сравнение численных методов решения задачи Коши являются актуальными в области вычислительной математики.

Сравнительное тестирование численных методов включает в себя проверку их эффективности и надежности в применении к различным видам задач. Это позволяет определить степень их пригодности для решения реальных, схожих с тестируемыми, систем.

Главным объектом исследования является сравнение явных одношаговых методов типа Рунге-Кутта разных классов для решения задачи Коши.

# Постановка задачи

Для проведения тестирования мне была предоставлена разработанная Олемским И.В. схема прямого интегрирования систем дифференциальных уравнений второго порядка. Передо мной встала цель провести сравнение полученного метода с аналогами. Были выделены следующие задачи:

1. Реализация алгоритмического ядра для проведения тестирования методов в равных условиях
2. Реализация схемы Олемского И. В.
3. Реализация остальных методов
4. Проведение сравнительного тестирования
5. Анализ полученных результатов

# Глава 1. Обзор литературы

## 1.1 Задача Коши

Задача Коши для обыкновенного дифференциального уравнения первого порядка имеет вид:

$$\frac{dy}{dt} = f(t, y(t)), \;\; t \in (t_0; T], \tag{1.1}$$

$$y(t_0) = y_0, \tag{1.2}$$

где $y : [t_0; T] \to R, \;\; f : (t_0; T) \times R \to R, \;\; y_0 \in R$

## 1.2 Явные одношаговые методы

Пусть требуется найти решение задачи Коши (1.1),(1.2) на отрезке $[t_0, T]$. Рассмотрим разбиение отрезка точками:

$$t_0 < t_1 < t_2 < ... < t_N = T. \tag{1.3}$$

Полученный набор точек называется сеткой, а точки $t_n$ - узлами сетки.

Одношаговые методы - это методы, которые последовательно дают приближения $y_n$ к значениям точного решения $y(t_n)$ в каждом узле $t_n$ сетки на основе известного приближения $y_{n-1}$ к решению в предыдущем узле $t_{n-1}$.

В общем виде они имеют представление:

$$y_{n+1} = F(f, t_{n+1}, t_n, y_{n+1}, y_n) \tag{1.4}$$

В случае, когда правая часть (1.4) не зависит от искомой функции $y_{n+1}$, мы говорим о явных одношаговых методах вида:

$$y_{n+1} = F(f, t_{n+1}, t_n, y_n)$$

## 1.3 Метод рядов Тейлора

Не всегда у поставленной задачи Коши (1.1),(1.2) имеется аналитическое решение, т.е. уравнение не интегрируется в явном виде. Поэтому появилась необходимость использовать методы, которые позволяют получать приближенное решение задачи. Одним из таких методов является метод рядов Тейлора.

Пусть требуется найти решение задачи (1.1),(1.2) на отрезке $[t_0, T]$. Рассмотрим разбиение отрезка точками (1.3).

Предположим, что правая часть $f(t, y(t))$ дифференциального уравнения (1.1) имеет непрерывные частные производные до порядка $s$. Тогда искомое решение $y(t)$ имеет непрерывные производные до $s+1$ - ого порядка включительно.

Тогда точное значение решения в узле $t_{i+1}$, при условии наличия точного решения в точке $t_i$, записывается по формуле Тейлора в виде:

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + ... + \frac{h^s}{s!}y^{(s)}(t_i) + \frac{h^{s+1}}{(s+1)!}y^{(s+1)}(\xi), \quad (1.5)$$

где

$$h = t_{i+1} - t_i, \quad \xi \in (t_i, t_{i+1}).$$

Если ограничиться первыми $s+1$ членами, а также заменить точное значение $y(t_i)$ его приближением $y_i$ , то получим приближенную формулу вида:

$$y_{i+1} = y_i + hy'_i + \frac{h^2}{2}y''_i + ... + \frac{h^s}{s!}y_i^{(s)}, \quad (1.6)$$

Где производные, которые входят в правую часть формулы (1.6), могут быть фактически найдены:

$$y'_i = f(t_i, y_i),$$

$$y''_i = \{f'_x + ff'_y\}|_{t_i},$$

$$y'''_i = \{f''_{xx} + 2ff''_{xy} + f^2f''_{yy} + (f'_x + ff'_y)f'_y\}|_{t_i}.$$

...

С увеличением порядка выражения для производных становятся все более громоздкими, что требует большого объема вычислений. Это является существенным недостатком данного метода [1].

## 1.4 Явные методы типа Рунге-Кутта

В начале 20 века Карл Рунге и Мартин Кутта предложили подход к вычислению приближенного решения $y(t)$ задачи (1.1),(1.2) в узле $t+h$ как линейную комбинацию корректирующих функций $k_i(h)$.

Явный одношаговый $s-$этапный метод Рунге-Кутты имеет вид:

$$y(t+h) \approx y_{t+h} = y_t + \sum_{i=1}^{s} b_i k_i(h), \tag{1.7}$$

Где корректирующие функции $k_i(h)$ представлены следующим образом:

$$k_1(h) = hf(t, y_t),$$

$$k_2(h) = hf(t + c_2 h, y_t + a_{21} k_1(h)),$$

...

$$k_s(h) = hf(t + c_s h, y_t + \sum_{j=1}^{s-1} a_{sj} k_j(h)). \tag{1.8}$$

Параметры метода $a_{ij},\ b_i,\ c_i$ определяются так, чтобы разложение (1.7) по степеням $h$ совпадало с разложением (1.5) до наиболее высокой степени $h$ при произвольном шаге $h$ и произвольной правой части $f(t, y)$.

Явные методы типа Рунге-Кутта отлично согласуются с задачей (1.1),(1.2) поскольку:

1. Для получения приближенного решения достаточно иметь начальное значение.

2. Чтобы начать вычисления, не требуется предварительного знания глобального поведения системы или специальных условий.

3. Есть возможность легко изменить шаг интегрирования без дополнительных потерь.

# Глава 2. Методы интегрирования

## 2.1 Интегрирование с переменным шагом

Интегрирование с переменным шагом в методах Рунге-Кутта позволяет более гибко подстраивать шаг интегрирования к изменяющейся природе функции или изменениям точности, которые требуются в разных областях решения. Использование данного приема обеспечивает баланс между точностью результата и вычислительной эффективностью.

Для успешного регулирования длины шага, в первую очередь, необходимо оценивать величину локальной погрешности, которая для явного метода Рунге-Кутта порядка $p$ имеет оценку:

$$error_{local} = y(t+h) - y_{t+h} \approx C_{p+1}h^{p+1}, \tag{2.1}$$

где $C_{p+1}$ - некоторая константа, которая зависит от коэффициентов метода.

## 2.2 Оценка погрешности при помощи комбинации независимых формул

Данный метод основан на построении двух формул вида (1.7), имеющих различный порядок точности. Рассмотрим комбинацию методов с порядками $p$ и $q$, где $p > q$, $s \geq r$:

$$y^p_{t+h} = y_t + \sum_{i=1}^{s} b_i k_i(h), \tag{2.2}$$

где

$$k_1(h) = hf(t, y_t),$$

...

$$k_s(h) = hf(t + c_s h, y_t + \sum_{j=1}^{s-1} a_{sj} k_j(h)),$$

и

$$y^q_{t+h} = y_t + \sum_{i=1}^{r} \overline{b_i k_i}(h), \tag{2.3}$$

где

$$\overline{k_1}(h) = hf(t, y_t),$$

...

$$\overline{k_r}(h) = hf(t + \overline{c}_r h, y_t + \sum_{j=1}^{r-1} \overline{a}_{rj} \overline{k_j}(h)).$$

Локальные погрешности для каждой из рассмотренных формул (2.2),(2.3) имеют вид:

$$error^p_{local} = y(t+h) - y^p_{t+h} = O(h^{p+1}), \tag{2.4}$$

$$error^q_{local} = y(t+h) - y^q_{t+h} = O(h^{q+1}) \tag{2.5}$$

Из равенств (2.4) и (2.5) следует оценка локальной погрешности метода меньшего порядка (2.3):

$$error^q_{local} \approx y^p_{t+h} - y^q_{t+h} + O(h^{p+1}) \tag{2.6}$$

Оставив в (2.6) только члены главного порядка, получим оценку, требующую $s + r - 1$ вычислений $f(t, y)$:

$$error^q_{local} \approx y^p_{t+h} - y^q_{t+h} \tag{2.7}$$

## 2.3 Вложенные методы

Рассмотрим комбинацию независимых формул (2.2), (2.3) где $p > q$, $s = r$, а коэффициэнты $a$, $c$ имеют вид:

$$a_{ij} = \overline{a}_{ij}, \quad c_i = \overline{c}_i, \tag{2.8}$$

$$i = 1, 2, ..., s, \quad j = 1, ..., i,$$

соответственно корректирующие функции для обеих формул равны между собой:

$$k_i(h) = \overline{k}_i(h), \quad i = 1, 2, ..., s, \tag{2.9}$$

Приближения к решению строятся используя два различных набора весовых коэффициентов $b_i$ и $\overline{b}_i$ (поэтому и считается, что один метод "вложен"в

другой):

$$y^p_{t+h} = y_t + \sum_{i=1}^{s} b_i k_i(h), \tag{2.10}$$

$$y^q_{t+h} = y_t + \sum_{i=1}^{s} \overline{b}_i k_i(h). \tag{2.11}$$

Тогда локальная погрешность (2.7) для вложенного метода (2.10),(2.11) имеет вид:

$$error^q_{local} \approx y^p_{t+h} - y^q_{t+h} = \sum_{i=1}^{s} w_i k_i(h) \tag{2.12}$$

где

$$w_i = \overline{b}_i - b_i, \;\; i = 1, 2, ..., s.$$

# Глава 3. Тестируемые методы

## 3.1 Метод Фельберга

В основе данного метода стоит идея вложенных методов. Фельберг вывел много формул различных порядков. Они представляют интерес к рассмотрению, поскольку в качестве искомого приближения к решению используется приближение меньшего порядка.

Для тестирования была выбрана расчетная схема Фельберга $RKF6(7)10$[2] здесь:

$-RKF-$ метод типа Рунге-Кутта Фельберга,

$-6-$ порядок точности искомого приближения,

$-7-$ порядок точности метода 'оценщика',

$-10-$ число этапов метода (количество обращений к процедуре вычисления правой части системы).

«Фельберг, чтобы сделать свои методы оптимальными, пытался минимизировать коэффициенты погрешности для результата низшего порядка $y^q_{t+h}$. Как следствие этого возникает опасность, что разность $y^q_{t+h} - y^p_{t+h}$, состоящая главным образом из минимизированных коэффициентов погрещности, недооценивает локальную погрешность».[3]

Также, локальная погрешность обычно имеет мало общего с глобальной,

так как обычно свойства устойчивости или неустойчивости решаемой задачи Коши неизвестны.

## 3.2 Метод Дорманда-Принца

Появляется очевидный вопрос: почему нельзя использовать приближение более высокого порядка в качестве искомого и начального значения на следующем шаге? Несмотря на то, что тогда "оценка погрешности (2.12)" в данном случае будет оценивать последние учтенные контрольные члены с $q+1$-го до $p$-го в разложении приближенного решения, появились методы, в основу которых легла эта идея.

Первым, кто рассмотрел данную идею, был математик В. А. Егоров. На базе существующего классического 4-х этапного метода Рунге-Кутта 4 порядка, из тех-же корректирующих функций, он построил приближение 2 порядка. Разность полученных приближений дает оценку двух последних учтенных контрольных членов, что позволяет адаптировать величину шага для последующих вычислений.

В настоящее время, одними из наиболее известных методов, использующих данный подход, являются методы Дорманда-Принца. Именно они используются в современных математических ПО (MATLAB, библиотека Sci-Py для Python), для решения систем обыкновенных дифференциальных уравнений, в качестве стандартных к применению.

Для тестирования была выбрана рассчетная схема Дорманда-Принца $DoPri5(4)7F$[4] здесь:

$-DoPri-$ метод типа Рунге-Кутта Дорманда-Принца,

$-5-$ порядок точности искомого приближения,

$-4-$ порядок точности метода 'оценщика',

$-7-$ число этапов метода,

$-F-$ используется технология $FSAL$ (First Same As Last - последнее вычисление правой части системы $f(t,y)$ на текущем шаге является первым на следующем. Соответственно, в среднем, на каждом шаге для метода $DoPri5(4)7F$ требуется не 7 вычислений правой части, а 6).

## 3.3 Структурный метод И. В. Олемского

Рассмотрим распространение задачи Коши (1.1),(1.2) на системы обыкновенных дифференциальных уравнений:

$$\frac{dy_i}{dt} = f_i(t, y_0, ..., y_n), \quad i = 1, ..., n, \tag{3.1}$$

$$y_i(t_0) = y_{i0}, \quad i = 1, ..., n, \tag{3.2}$$

$$t \in [t_0, T] \subset R, \;\; y_i : [t_0, T] \to R, \;\; f_i : [t_0, T] \times R^{n+1} \to R, \;\; i = 1, ..., n. \tag{3.3}$$

Будем говорить, что к задаче (3.1),(3.2),(3.3) применим Структурный метод, если система (3.1) имеет вид:

$$\frac{dy_0}{dt} = f_0(t, y_0, ..., y_n), \tag{3.4}$$

$$\frac{dy_i}{dt} = f_i(t, y_0, ..., y_{i-1}, y_{l+1}, ..., y_n), \;\; i = 1, 2, ..., l, \tag{3.5}$$

$$\frac{dy_j}{dt} = f_j(t, y_0, ..., y_{j-1}), \;\; j = l+1, ..., n. \tag{3.6}$$

где

$$l \in \{0\} \cup N, \;\; n \in \{0\} \cup N, \;\; l \le n, \;\; t \in [t_0, T] \in R,$$

$$y_s : [t_0, T] \to R^{r_s}, s = 0, 1, ..., n,$$

$$f_0 : [t_0, T] \times R^r \to R^{r_0}, \;\; \sum_{s=0}^{n} r_s = r,$$

$$f_i : [t_0, T] \times R^{r - \sum_{g=i}^{l} r_g} \to R^{r_i}, \;\; i = 1, 2, ..., l,$$

$$f_j : [t_0, T] \times R^{r - \sum_{g=j}^{n} r_g} \to R^{r_j}, \;\; j = l+1, ..., n$$

«Две группы уравнений (3.5),(3.6) структурно тождественны. Каждое уравнение одной из групп уравнений (1.7),(1.8) занимает определенное место в последовательности уравнений своей группы. Его правая часть не зависит от искомых функций, поведение которых описывается этим и всеми последующими уравнениями этой же группы. Уравнение (3.4) - представитель общей группы, в которую вошли все уравнения, не имеющие структурных особенно-

стей указанного выше типа. Причем в рассматриваемой системе может отсутствовать как общая группа уравнений (3.4) ($r_0 = 0$), так и группа уравнений (3.5) ($l = 0, n \in N$)».[5]

Мне была предоставлена И. В. Олемским схема прямого интегрирования систем дифференциальных уравнений второго порядка. Для каждой группы уравенний (3.4),(3.5),(3.6) предлагается использование своей расчетной схемы. Для тестирования была выбрана комбинация, применяемая к системам, которые состоят из структурно выделенных групп (3.5),(3.6). Будем называть ее $RKS6(4)7F$.

Благодаря использованию структурных особенностей систем, метод $RKS6(4)7F$ позволяет находить приближения к решению 6 порядка точности всего за 7 вычислений правой части системы. В дополнение, поскольку используется технология $FSAL$, общие затраты метода на шаге в среднем составляют 6 обращений к процедуре вычисления правой части.

# Глава 4. Сравнительное тестирование методов

Для проведения тестирования было разработано алгоритмическое ядро на базе языка программирования Python с 16-значной разрядной сеткой. Для объективной оценки методов был взят алгоритм автоматического выбора шага из реализации одного из представленных методов в среде MATLAB (ode45-реализация метода $DoPri5(4)7F$) вида:

$$h_{hew} = h * min(facmax, max(facmin, fac * (\frac{tol}{error})^{\frac{1}{p+1}}), \qquad (4.1)$$

где

$facmax = 5, facmin = 0.2-$ параметры, ограничивающие возможность слишком быстрого увеличения или уменьшения шага,

$fac = 0.8-$ гарантирующий множитель, обеспечивающий приемлимый показатель погрешности в ближайшее время с высокой вероятностью,

$p-$порядок метода,

$h-$ шаг, используемый для вычисления текущего приближения,

$error-$ величина оценки погрешность,

$tol-$ допустимое значение погрешности.

Основными критериями оценки являлись надежность и трудоемкость. Для визуализации результатов использовалась библиотека matplotlib для Python.

## 4.1 Модели для тестирования

Для иллюстрации и сравнения работы методов были выбраны две задачи, каждая из которых является системой из двух дифференциальных уравнений второго порядка.

Обе представленные системы обладают необходимыми структурными особенностями для применения $RKS6(4)7F$ метода.

### 4.1.1 Орбита Аренсторфа

Плоская ограниченная задача трех тел из небесной механики. Рассматриваются два тела с массами $1-\mu$ и $\mu$, участвующие в совместном круговом движении в некоторой плоскости, и движущееся вблизи них в той же плоскости третье тело пренебрежимо малой массы.

Уравнения имеют вид:

$$\begin{cases} x'' = x + 2y' - \mu' \dfrac{x+\mu}{D_1} - \mu \dfrac{x-\mu'}{D_2}, \\ y'' = y - 2x' - \mu' \dfrac{y}{D_1} - \mu \dfrac{y}{D_2}, \end{cases} \tag{4.2}$$

где:

$$\mu = 0.012277471, \;\; \mu' = 1 - \mu,$$

$$D_1 = ((x+\mu)^2 + y^2)^{\frac{3}{2}}, \;\; D_2 = ((x-\mu')^2 + y^2)^{\frac{3}{2}},$$

с начальными условиями:

$$x(0) = 0.994,$$
$$x'(0) = 0,$$
$$y(0) = 0,$$
$$y'(0) = -2.00158510637908252240537862224,$$

и периодом:

$$T = 17.065216560157962558891720624 9.$$

### 4.1.2 ”Модельная” задача

Данная система в с сравнении с орбитой Аренсторфа имеет более простую, линейную правую часть, а также нам известно её точное (аналитическое) решение.

Система имеет вид:

$$\begin{cases} x'' = 3y' + 2x, \\ y'' = -3x' + 2y, \end{cases} \tag{4.3}$$

с начальными условиями:

$$x(0) = y'(0) = 1,$$
$$x'(0) = y(0) = 0,$$

и периодом:

$$T = 2\pi.$$

Аналитическое решение (4.3) можно представить в виде:

$$\begin{aligned} x(t) &= C_1 cos(t) + C_2 sin(t) + C_3 cos(2t) + C_4 sin(2t), \\ y(t) &= -C_1 sin(t) + C_2 cos(t) - C_3 sin(2t) + C_4 cos(2t), \\ x'(t) &= -C_1 sin(t) + C_2 cos(t) - 2C_3 sin(2t) + 2C_4 cos(2t), \\ y'(t) &= -C_1 cos(t) - C_2 sin(t) - 2C_3 cos(2t) - 2C_4 sin(2t). \end{aligned}$$

где коэффициенты $C_1, C_2, C_3, C_4$ зависят от начальных условий задачи Коши.

## 4.2 Результаты тестирования

### 4.2.1 Эффективность

Для различных методов требуется разный объем вычислительных затрат для нахождения численного решения. В рамках представленных задач были проведены рассчеты при различных ограничениях на допустимую погрешность *tol*.

Графики приведены в двойной логарифмической шкале. По оси абсцисс располагается число обращений метода к процедуре вычисления правой части системы, а по оси ординат - величина глобальной погрешности. Чем выше и правее расположена кривая на графике, тем эффективнее метод (то есть тем ниже количество вычислений (трудоемкость) при той же величине глобальной погрешности).

Несмотря на отсутствие аналитического решения для одной из рассматриваемых задач, величину глобальной погрешности мы можем получить отталкиваясь от периодичности системы. Таким образом, для обеих моделей, глобальная погрешность решения на интервале $(0, T)$ имеет вид:

$$error_{global} = ||y_0 - y_T||$$

Где:

$y_0$ — начальное условие задачи Коши,
$y_T$ — численное решение, полученное в точке $t = T$.

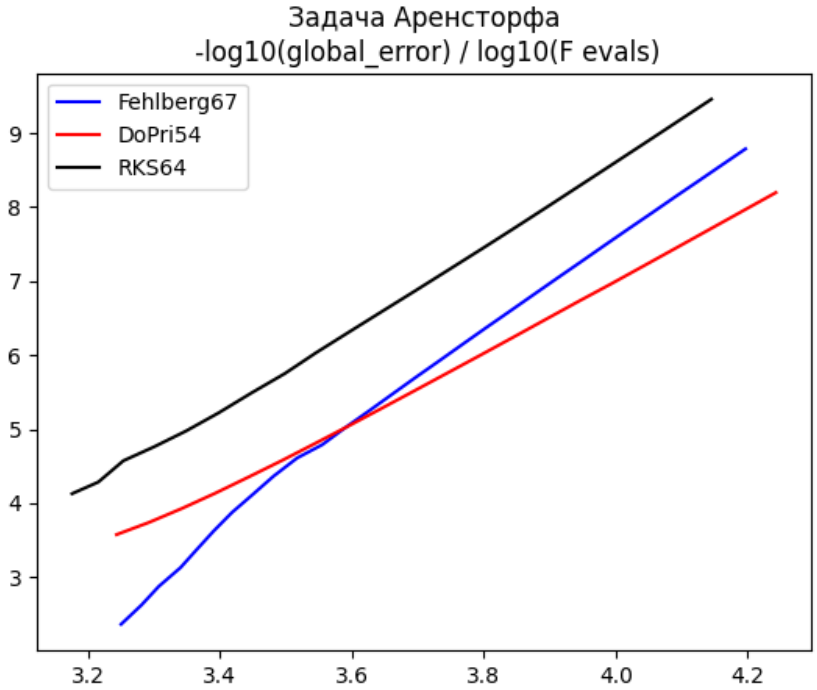


**Рис. 1:** Зависимость глобальной погрешности от трудоемкости для задачи Аренсторфа

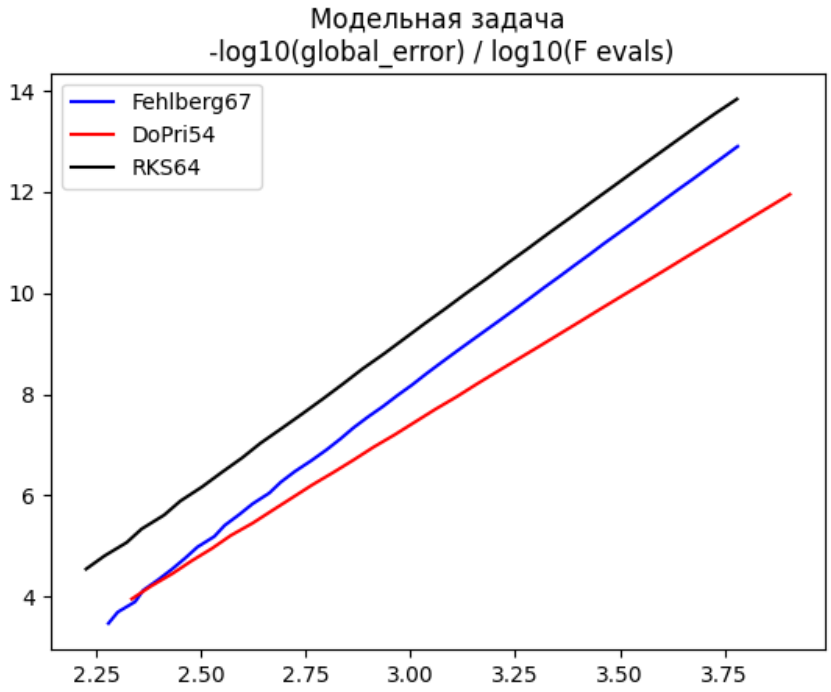


**Рис. 2:** Зависимость глобальной погрешности от трудоемкости для модельной задачи

### 4.2.2 Надежность

С помощью точного решения системы (4.3), на каждом шаге интегрирования, помимо оценки погрешности $error$, можно вычислять истинную локальную погрешность.

$$error_{local}^{true} = y(t+h) - y_{t+h}.$$

Проверяя надежность метода, необходимо рассмотреть влияние входного параметра $tol$ (допустимая погрешность) на динамику отношения истинной локальной погрешности к оценке (обозначим $E = \frac{error_{local}^{true}}{error}$) для обеспечения равномерного распределения погрешности на величину длины шага.

Были проведены вычисления для разных значений $tol$ : $1e-3,\ 1e-5,\ 1e-7$. По оси абсцисс расположен интервал интегрирования $(0, 2\pi)$, а по оси ординат величина $E$ (Рис. 3, 5, 7) или длина шага $h$ (Рис. 4, 6, 8).

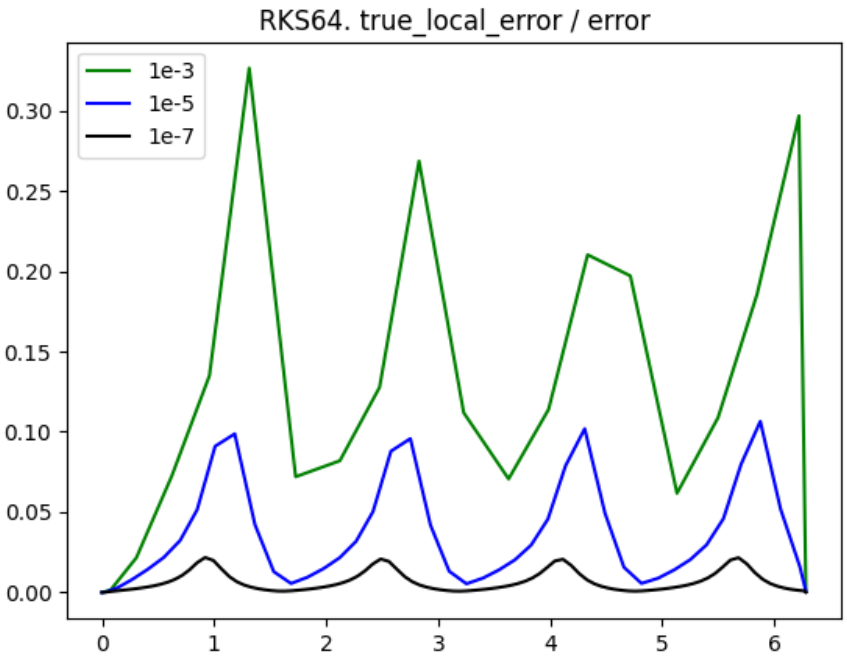


**Рис. 3:** Отношение истинной локальной погрешности к оценке для метода $RKS6(4)7F$

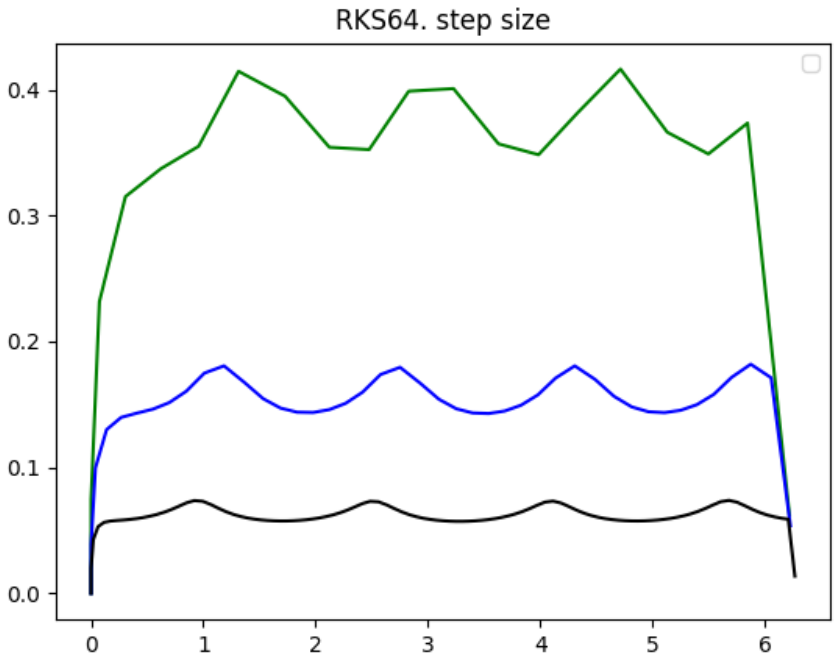


**Рис. 4:** Величина шага для метода $RKS6(4)7F$

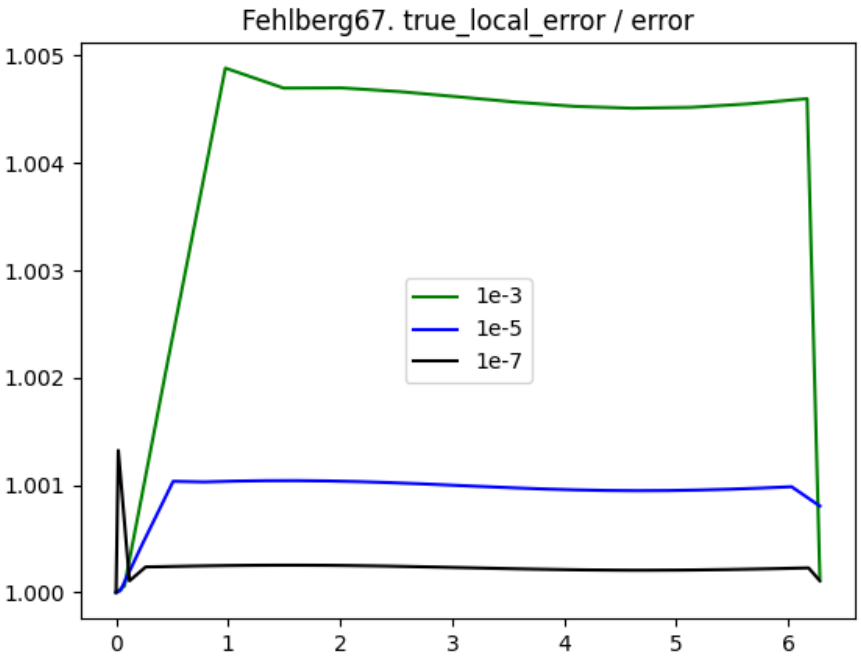


**Рис. 5:** Отношение истинной локальной погрешности к оценке для метода $RKF6(7)10$

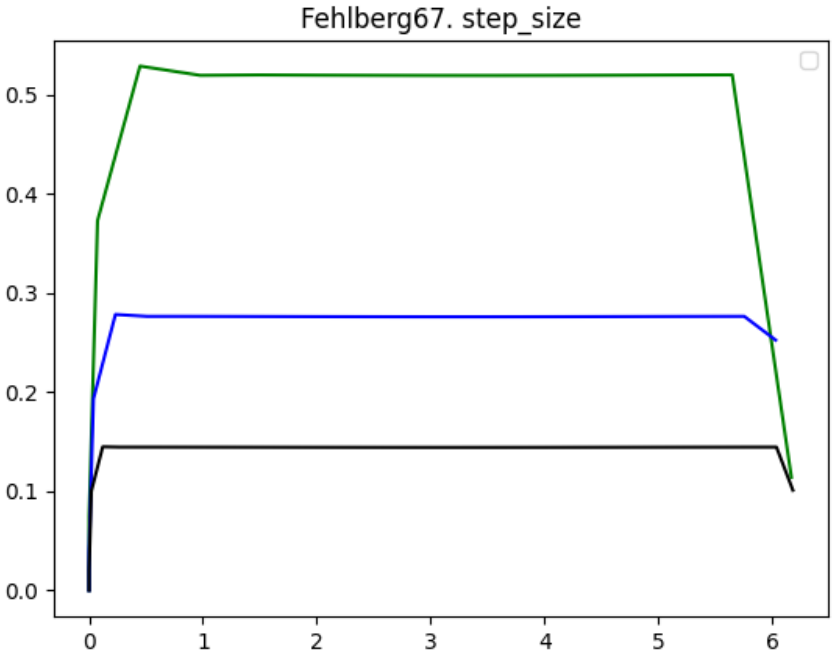


**Рис. 6:** Величина шага для метода $RKF6(7)10$

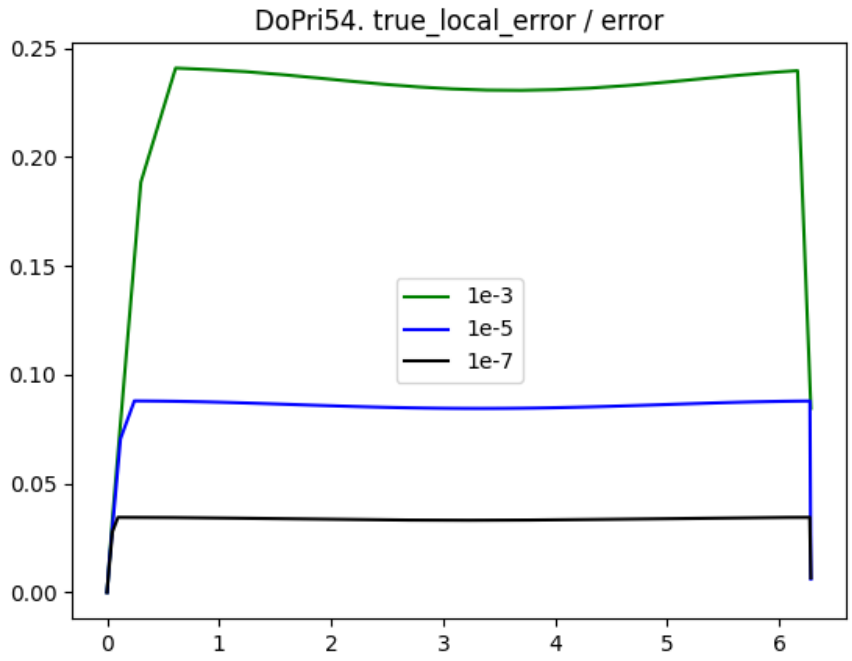


**Рис. 7:** Отношение истинной локальной погрешности к оценке для метода $DoPri5(4)7F$

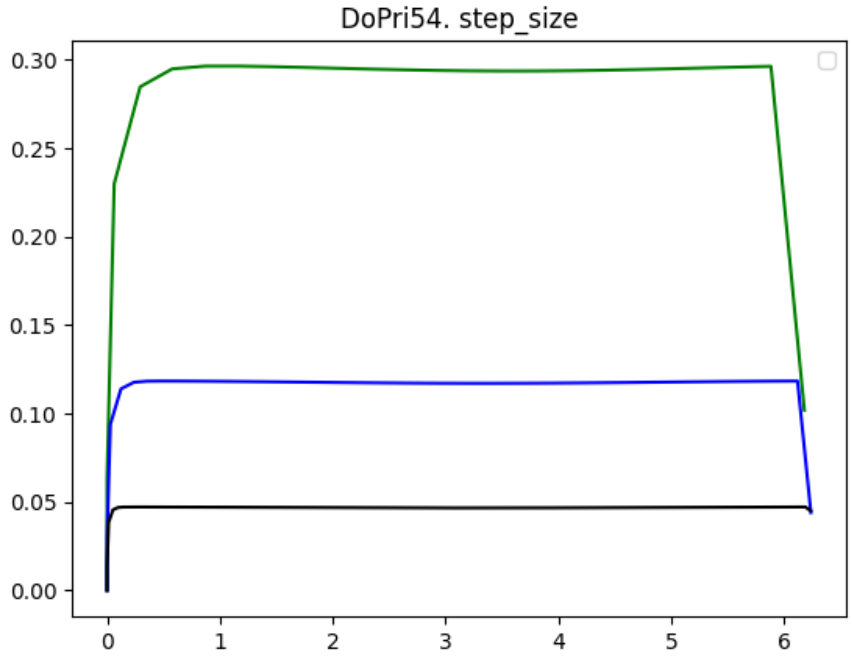


**Рис. 8:** Величина шага для метода $DoPri5(4)7F$

# Выводы

В первую очередь важно отметить, что рассмотренные методы подтвердили заявленный теоретический порядок точности. Поскольку прямые, изображенные на графике зависимости логарифма погрешности от логарифма трудоемскости (Рис. 1, 2), имеют угол наклона равный $\mathrm{arctg}(p)$, где $p-$порядок метода.

Результаты сравнительного тестирования показывают, что наиболее эффективным методом, в рамках рассмотренных задач, оказался $RKS6(4)$, поскольку при равном порядке глобальной погрешности его вычислительные затраты (число обращений метода к процедуре вычисления правой части) существенно меньше чем у $RKF6(7)10$ и $DoPri5(4)7F$. А при равных затратах на вычисление правой части метод $RKS6(4)$ показывает лучшие результаты по величине глобальной погрешности.

Каждый из тестируемых методов показал себя как надежный, поскольку динамика отношения истинной локальной погрешности к оценке имеет один и тот-же вид при различных ограничениях на допустимую погрешность $tol$ (Рис. 3, 5, 7). Также о надежности методов говорит адекватное влияние оценки погрешности на изменение длины шага (Рис. 4, 6, 8) (на участках, где истинная локальная погрешность увеличивается, следует закономерное уменьшение шага интегрирования и наоборот).

# Заключение

В ходе работы было проведено сравенние явных одношаговых методов Рунге-Кутта для решения обыкновенных дифференциальных уравнений на двух задачах.

1. Реализовано алгоритмическое ядро (с использованием языка программирования Python с 16-значной разрядной сеткой) для проведения сравнительного тестирования в равных условиях.

2. На базе единого алгоритмического ядра реализованы рассчетные схемы методов $RKF6(7)10$, $DoPri5(4)7F$, $RKS6(4)7F$ с алгоритмом автоматического выбора шага.

3. Проведено тестирование рассмотренных методов на различных моделях.

4. Проведен численный анализ результатов сравнения методов по критериям эффективности и надежности.

Результаты тестирования показали, что наиболее эффективным среди рассматриваемых методов оказался метод И. В. Олемского $RKS6(4)7F$. Также в рамках оценки надежности, каждый метод обеспечивает равномерное распределение погрешности на единицу величины шага.

# Список литературы


[1] Арушанян О.Б., Залеткин С.Ф. Численное решение обыкновенных дифференциальных уравнений на Фортране. М.: Изд-во МГУ, 1990.

[2] Fehlberg, E. Classical fifth-, sixth-, seventh-, and eighth-order Runge–Kutta formulas with stepsize control / E. Fehlberg NASA Technical Report 287. — 1968.

[3] Хайрер, Э. Решение обыкновенных дифференциальных уравнений. Нежёсткие задачи / Пер. с англ. И. А. Кульчицкой, С. С. Филиппова под ред. С. С. Филиппова. / Э. Хайрер, С. Нёрсетт, Г. Ваннер. — М.: Мир, 1990. — 512 с

[4] Dormand, J. R. A Family of Embedded Runge–Kutta Formulae / J. R. Dormand, P. J. Prince / Journal of Computational and Applied Mathematics. — 1980. — Mar. — Vol. 6. — P. 19—26.

[5] И. В. Олемской, Структурный подход к задаче конструирования явных одношаговых методов, Ж. вычисл. матем. и матем. физ., 2003, том 43, номер 7, 961–974

[6] Олемской, И. В. Численный метод интегрирования систем обыкновенных дифференциальных уравнений / И. В. Олемской / Математические методы анализа управляемых процессов. — 1986. — С. 157—160

# Приложение

Интегрирование на промежутке $[t\_start, t\_end]$, с начальным шагом $h\_start$, с ограничением на допустимую погрешность $tol$, с начальными условиями задачи Коши $values$, системы $f$, методом $method$, с порядком $p$, с числом этапов $s$, и параметрами алгоритма автоматического выбора шага: $facmax = 5.0, facmin = 0.2, fac = 0.8$.

```
def Calc_1_1(t_start, t_end, h_start, tol, values, f,
    method, p, s, facmax=5.0, facmin=0.2, fac=0.8):
    x_prev, dx_prev, y_prev, dy_prev = (x for x in
        values)
    error = 0
    result = {t_start: (x_prev, dx_prev, y_prev,
        dy_prev, error)}
    current_t = t_start
    steps = 1

    while t_end - current_t > 0:
        temp = method(current_t, x_prev, y_prev,
            dx_prev, dy_prev, h, f[0], f[1])
        error = temp[4]
        steps += 1

        if error < tol:
            current_t += h
            result[current_t] = temp
            x_prev, dx_prev, y_prev, dy_prev =
                (temp[i] for i in range(0, 4))
        h = h * min(facmax, max(facmin, fac * ((tol /
            error) ** (1 / (p + 1)))))

        if t_end - current_t - h < 0:
            h = t_end - current_t
```

```
21              temp = method(current_t, x_prev, y_prev,
                    dx_prev, dy_prev, h, f[0], f[1])
                error = temp[4]
23              steps += 1

25              if error < tol:
                    current_t += h
27                  result[current_t] = temp
                    break
29              else:
                    h = h / 2
31
    return result, steps
```

Интегрирование по интервалу допустимой погрешности $[tol\_start, tol\_end]$, с начальными условиями задачи Коши $x\_0$, системы $equations$, на промежутке $interval$, методом $method$, с порядком $p$, с числом этапов $s$, и параметрами алгоритма автоматического выбора шага: $facmax = 5.0, facmin = 0.2, fac = 0.8$.

```
def Calc_other_method_by_tol_interval(tol_start,
    tol_end, x_0, equations, interval, method, p, s,
    fac=0.8, x_end=x_0):

    tols = [10 ** (-i) for i in
      [-math.log10(tol_start) + x * 0.25 for x in
      range(0, 1 + 4 * int(-math.log10(tol_end) +


    global_errors, steps = [], []

    for current_tol in tols:
        current_result, current_steps = \
            Calc_1_1(t_start=interval[0],
                     t_end=interval[1],
                     h_start='auto',
                     tol=current_tol,
                     values=x_0,
                     f=equations,
                     method=method,
                     p=p,
                     s=s,
                     facmax=5.0,
                     facmin=0.2,
                     fac=fac)
        current_new_x_0 = current_result[interval[1]]
        steps.append((current_steps,
          len(current_result)))
```

```
24            global_errors.append(norma2([current_new_x_0[i]
                - x_end[i] for i in range(0, 4)]))

26        result = {
              'tols': tols,
28            'steps': steps,
              'global_errors': global_errors
30        }

32        return result
```

Пример реализации одного из методов - $DoPri5(4)7F$.

```
def DoPriRK54(t, x_prev, y_prev, dx_prev, dy_prev, h,
    fdx, fdy):
    fx = lambda t, x, y, dx, dy: dx
    fy = lambda t, x, y, dx, dy: dy

    kx = dict()
    ky = dict()
    kdx = dict()
    kdy = dict()

    values = (x_prev, y_prev, dx_prev, dy_prev)

    k = [kx, ky, kdx, kdy]
    f = [fx, fy, fdx, fdy]

    for i in range(0, 4):
        k[i][0] = f[i](t, *values)

    for i in range(0, 4):
        temp_var = tuple(values[j] +
                         h * (1 / 5) * k[j][0] for j
                           in range(0, 4))
        k[i][1] = f[i](t + (1 / 5) * h, *temp_var)

    for i in range(0, 4):
        temp_var = tuple(values[j] +
                         h * ((3 / 40) * k[j][0] +
                              (9 / 40) * k[j][1]) for
                                j in range(0, 4))
        k[i][2] = f[i](t + (3 / 10) * h, *temp_var)

    for i in range(0, 4):
```

```
        temp_var = tuple(values[j] +
                         h * ((44 / 45) * k[j][0] +
                              (-56 / 15) * k[j][1] +
                              (32 / 9) * k[j][2]) for
                                j in range(0, 4))
        k[i][3] = f[i](t + (4 / 5) * h, *temp_var)

    for i in range(0, 4):
        temp_var = tuple(values[j] +
                         h * ((19372 / 6561) *
                           k[j][0] +
                              (-25360 / 2187) *
                                k[j][1] +
                              (64448 / 6561) *
                                k[j][2] +
                              (-212 / 729) * k[j][3])
                                for j in range(0, 4))
        k[i][4] = f[i](t + (8 / 9) * h, *temp_var)

    for i in range(0, 4):
        temp_var = tuple(values[j] +
                         h * ((9017 / 3168) * k[j][0]
                           +
                              (-355 / 33) * k[j][1] +
                              (46732 / 5247) *
                                k[j][2] +
                              (49 / 176) * k[j][3] +
                              (-5103 / 18656) *
                                k[j][4]) for j in
                                range(0, 4))
        k[i][5] = f[i](t + h, *temp_var)

    for i in range(0, 4):
```

```
54          temp_var = tuple(values[j] +
                             h * ((35 / 384) * k[j][0] +
56                                (500 / 1113) * k[j][2] +
                                  (125 / 192) * k[j][3] +
58                                (-2187 / 6784) *
                                    k[j][4] +
                                  (11 / 84) * k[j][5])
                                    for j in range(0, 4))
60          k[i][6] = f[i](t + h, *temp_var)

62      x_next, y_next, dx_next, \
        dy_next = (values[i] +
64                 h * ((35 / 384) * k[i][0] +
                        (500 / 1113) * k[i][2] +
66                      (125 / 192) * k[i][3] +
                        (-2187 / 6784) * k[i][4] +
68                      (11 / 84) * k[i][5]) for i in
                          range(4))

70      x1_next, y1_next, dx1_next, \
        dy1_next = (values[i] +
72                  h * ((5179 / 57600) * k[i][0] +
                         (7571 / 16695) * k[i][2] +
74                       (393 / 640) * k[i][3] +
                         (-92097 / 339200) * k[i][4] +
76                       (187 / 2100) * k[i][5] +
                         (1 / 40) * k[i][6]) for i in
                           range(4))
78
        errx, erry, errdx, \
80      errdy = (h * ((35 / 384 - 5179 / 57600) * k[i][0]
          +
                      (500 / 1113 - 7571 / 16695) *
```

```
                k[i][2] +
82              (125 / 192 - 393 / 640) * k[i][3] +
                (-2187 / 6784 + 92097 / 339200) *
                k[i][4] +
84              (11 / 84 - 187 / 2100) * k[i][5] +
                (-1 / 40) * k[i][6]) for i in
                range(4))
86
    error = (errx ** 2 + erry ** 2 + errdx ** 2 +
      errdy ** 2) ** (1 / 2)
88  return x_next, dx_next, y_next, dy_next, error
```