ПРОГРАММИРОВАНИЕ, 2021, № 1, с. 25–34

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 004.422.8

ИСПОЛЬЗОВАНИЕ ШАБЛОНИЗАТОРА КАК ИНСТРУМЕНТАРИЯ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

© 2021 г. М. Н. Геворкян[a,*], А. В. Королькова[a,], Д. С. Кулябов[a,b,***]**

[a] *Кафедра прикладной информатики и теории вероятностей, Российский университет дружбы народов, ул. Миклухо-Маклая, д. 6, Москва, Россия, 117198*

[b] *Лаборатория информационных технологий, Объединенный институт ядерных исследований, ул. Жолио-Кюри 6, Дубна, Московская область, Россия, 141980*

**E-mail: gevorkyan-mn@rudn.ru*

***E-mail: korolkova-av@rudn.ru*

****E-mail: kulyabov-ds@rudn.ru*

Поступила в редакцию 15.07.2020 г.
После доработки 07.08.2020 г.
Принята к публикации 12.09.2020 г.

В исследовательских задачах, требующих применения численных методов решения систем обыкновенных дифференциальных уравнений, часто возникает необходимость выбора наиболее эффективного и оптимального для конкретной задачи численного метода. В частности, для решения задачи Коши, сформулированной для системы обыкновенных дифференциальных уравнений, применяются методы Рунге–Кутты (явные или неявные, с управлением шагом сетки или без и т.д.). При этом приходится перебирать множество реализаций численного метода, подбирать коэффициенты или другие параметры численной схемы. В данной статье предложено описание разработанной авторами библиотеки и скриптов автоматизации генерации функций программного кода на языке Julia для набора численных схем методов Рунге–Кутты. При этом для символьных манипуляций использовано программное средство подстановки по шаблону. Предлагаемый подход к автоматизации генерации программного кода позволяет вносить изменения не в каждую подлежащую сравнению функцию по отдельности, а использовать для редактирования единый шаблон, что с одной стороны дает универсальность в реализации численной схемы, а с другой позволяет свести к минимуму число ошибок в процессе внесения изменений в сравниваемые реализации численного метода. Рассмотрены методы Рунге–Кутты без управления шагом, вложенные методы с управлением шагом и методы Розенброка также с управлением шагом. Полученные автоматически с помощью разработанной библиотеки программные коды численных схем протестированы при численном решении нескольких известных задач.

DOI: 10.31857/S0132347421010052

1. ВВЕДЕНИЕ

Методы Рунге–Кутты являются основными численными методами для решения нежестких систем обыкновенных дифференциальных уравнений. Известны численные схемы высоких порядков (10 и выше) с управлением шагом и плотной выдачей. Для многих языков программирования разработаны библиотеки, реализующие наиболее эффективные из построенных численных схем. Наиболее отлаженные коды были созданы для языка Fortran 77 Э. Хайрером (Ernst Hairer) и его коллегами, они описаны в книгах [1, 2] и доступны для скачивания на сайте [3]. Эти программы реализуют методы `DOPRI5` [4] и `DOPRI853` [5], которые обеспечивают управление шагом и плотную выдачу. Второй из этих методов также поддерживает переключение порядка метода между 8 и 5 для обеспечения большего быстродействия при решении гладких задач.

Подпрограммы `DOPRI5` и `DOPRI853` очень хорошо оптимизированы и доступны для использования во многих библиотеках и математических пакетах, таких как Matlab [6], Octave [7], SciPy [8], SciLab [9], Boost C++ [10] и т.д. Однако часто для учебных и исследовательских целей возникает необходимость перебора нескольких методов Рунге–Кутты для выбора оптимального для конкретной задачи [11–13]. Кроме того, эффективность той или иной численной схемы также существенно зависит от решаемой задачи.

Чтобы можно было достоверно сравнить разные численные схемы между собой, необходимо иметь универсальный код, который реализует вложенные методы для любого набора коэффи-

циентов. При этом все реализации должны быть унифицированы и отличаться лишь набором коэффициентов.

С целью решения поставленной задачи нами разработана библиотека для языка программирования Julia [14–16], описание которой и приводится в данной статье. Библиотека состоит из двух частей: вычислительной части, реализующей алгоритмы типа Рунге–Кутты, и конструирующей части, создающей с помощью символьных вычислений конкретный вариант алгоритма Рунге–Кутты. Отличительной особенностью конструирующей части является автоматическая генерация кода функций из нескольких готовых шаблонов. Первоначально рассматривался вариант реализации конструирующей части с помощью универсальной системы компьютерной алгебры, например, с использованием SymPy [17]. Однако в ходе реализации выяснилось, что нам нужно от системы компьютерной алгебры в основном операции сопоставления по образцу. В результате было принято решение использовать более легковесное средство символьных манипуляций. Фактически, для символьных манипуляций мы используем не развитую систему компьютерной алгебры, а программное средство подстановки по шаблону. Такой подход хотя и несколько экзотичен, но оправдан большей производительностью сгенерированных функций. Для генерации кода использован язык Python [18] и шаблонизатор Jinja2 [19].

1.1. Структура статьи

В разделе 2 работы дается краткое описание вложенных методов Рунге–Кутты и стратегии управления шагом. Раздел 3 посвящен описанию разработанной библиотеки для языка программирования Julia. В разделе 4 приведены результаты тестирования сгенерированных автоматически с помощью разработанной библиотеке функций для численного решения задач для систем обыкновенных дифференциальных уравнений. В частности в этом разделе подробно рассмотрено решение ограниченной задачи трех тел (задачи по вычислению орбит Аренсторфа).

2. МЕТОДЫ РУНГЕ–КУТТЫ

Методы Рунге–Кутты применяются к задаче Коши, сформулированной для системы обыкновенных дифференциальных уравнений (ОДУ). Теория методов Рунге–Кутты хорошо известна и обстоятельно изложена в книгах [1, 2, 20]. Поэтому в данном разделе мы приведем лишь основные формулы и кратко остановимся на используемой в разработанной программной библиотеке стратегии управления шагом.

2.1. Постановка задачи

Пусть даны две гладкие функции: неизвестная функция $y^\alpha(t)\colon [t_0, T] \to \mathbb{R}^N$ и известная функция $f^\alpha(t, y^\beta(t))\colon \mathbb{R}\times\mathbb{R}^N \to \mathbb{R}^N$, где $[t_0, T] \in \mathbb{R}$, $\alpha = 1, \ldots, N$. Пусть также известно значение функции в начальный момент времени $y_0^\alpha = y^\alpha(t_0)$. Тогда задача Коши для системы из N обыкновенных дифференциальных уравнений формулируется следующим образом:

$$\begin{cases} \dot y^\alpha(t) = f^\alpha(t, y^\beta(t)), \\ y^\alpha(t_0) = y_0^\alpha, \quad \alpha,\beta = 1,\ldots,N. \end{cases} \tag{1}$$

Систему (1) можно записать покомпонентно:

$$\begin{cases} \dot y^1(t) = f^1(t, y^1(t),\ldots, y^N(t)), \\ \vdots \\ \dot y^N(t) = f^N(t, y^1(t),\ldots, y^N(t)), \\ y^\alpha(t_0) = y_0^\alpha. \end{cases}$$

Зададим на отрезке $[t_0, T]$ сетку из набора точек $t_0 < t_1 < t_2 < \ldots < t_k < \ldots < t_N = T$ с шагом сетки $h_{k+1} = t_{k+1} - t_k$. Каждой точке сетки по некоторому правилу, называемому *численной схемой*, ставится в соответствие некоторая величина y_k^α, которая должна с необходимой точностью аппроксимировать решение системы ОДУ в точках сетки, т.е. $y_k^\alpha \approx y^\alpha(t_k)$. Для оценки погрешности аппроксимации используют норму $\|y^\alpha(t_k) - y_k^\alpha\|$.

2.2. Явные вложенные методы Рунге–Кутты

Погрешность метода Рунге–Кутты может оцениваться и другим образом. Идея состоит в том, что кроме основного решения y_m^α в точке рассматривается также вспомогательное решение $\hat y_m^\alpha$, выполненное с помощью метода Рунге–Кутты смежного порядка. Разность этих решений может служить оценкой локальной погрешности метода меньшего порядка. Полученная оценка локальной погрешности помогает выбрать переменный шаг интегрирования. Методы, использующие данный способ оценки локальной погрешности, называются *вложенными* методами Рунге–Кутты [1]. При этом метод y_m^α называется *основным*, а метод $\hat y_m^\alpha$ – *вложенным*.

Явный вложенный численный метод Рунге–Кутты для задачи Коши (2.1) задается следующими формулами:

$$k^{1\alpha} = f^{\alpha}(t_m, y_m^{\beta}),$$
$$k^{i\alpha} = f^{\alpha}\left(t_m + c^i h_m, y_m^{\alpha} + h\sum_{j=1}^{i-1} a_j^i k^{j\beta}\right),$$
$$i = 2, \ldots, s,$$

$$y_{m+1}^{\alpha} = y_m^{\alpha} + h_m(b_1 k^{1\alpha} + b_2 k^{2\alpha} + \ldots + b_{s-1} k^{s-1,\alpha} + b_s k^{s\alpha}),$$
$$\hat{y}_{m+1}^{\alpha} = y_m^{\alpha} + h_m(\hat{b}_1 k^{1\alpha} + \hat{b}_2 k^{2\alpha} + \ldots + \hat{b}_{s-1} k^{s-1,\alpha} + \hat{b}_s k^{s\alpha}),$$

где N – число уравнений в системе ОДУ, s – число *стадий* численного метода. Латинские индексы $i, j, l = 1, \ldots, s$ относятся к численной схеме, греческие α, β – к системе ОДУ, а индексом m обозначен номер шага.

Порядок аппроксимации решения сеточными функциям y_m^{α} и $\hat{y}_m^{\alpha}$ различен, что позволяет на каждом шаге вычислять погрешность аппроксимации, поэтому при указании порядка вложенного метода используют обозначение $p(\hat{p})$, где p и $\hat{p}$ – порядки основного и вложенного метода Рунге–Кутты соответственно.

Величины $c^i, a_j^i, b_j, \hat{b}_j$ полностью определяют численную схему и называются *коэффициентами метода*, их принято группировать в виде *таблицы Бутчера*, названной так в честь Дж.Ч. Бутчера (John C. Butcher):

0	0	0	…	0	0
c^2	a_1^2	0	…	0	0
c^3	a_1^3	a_2^3	…	0	0
⋮	⋮	⋮	⋱	⋮	⋮
c^{s-1}	a_1^{s-1}	a_2^{s-1}	…	0	0
c^s	a_1^s	a_2^s	…	a_{s-1}^s	0
	b_1	b_2	…	b_{s-1}	b_s
	$\hat{b}_1$	$\hat{b}_2$	…	$\hat{b}_{s-1}$	$\hat{b}_s$

Нахождение коэффициентов метода Рунге–Кутты для порядков выше 4 является отдельной сложной задачей, которой мы здесь не будем касаться и которая подробно раскрыта в книге [1]. Заметим только, что на коэффициенты c^i и a_j^i также принято налагать дополнительные условия следующего вида: $c^i = a_1^i + \ldots + a_s^i$.

Явный метод Рунге–Кутты наиболее прост для реализации в программном виде, так как на каждом шаге метода необходимо последовательно вычислять элементы двухмерного массива `k[1:s,1:N]` в цикле по первому измерению `i=1:s`. Элемент `k[1,1:N]` требует только значений t_m, y_m, вычисленных на предыдущем шаге, элемент `k[2,1:N]` требует `k[1,1:N]`, элемент `k[3,1:N]` требует `k[1,1:N]` и `k[2,1:N]` и т.д. Заметим, что индексация элементов массива в языке Julia начинается с 1, также как и в языке Fortran.

К настоящему времени найдены коэффициенты для множества вложенных методов Рунге–Кутты вплоть до 10-го порядка точности. Наиболее часто используются методы, найденные Дж.Р. Дорманом в соавторстве с П.Дж. Принцем (J.R. Dorman, P.J. Prince) [4, 5, 21], Э. Фельбергом (E. Fehlberg) [22, 23] и Дж.Р. Кэшом в соавторстве с А.Х. Карпом (J.R. Cash, A.H. Karp) [24].

2.3. Стратегии управления длиной шага

Существуют различные стратегии управления величиной шага h в зависимости от локальной погрешности. Выбор той или иной стратегии чаще всего диктуется характером решаемой задачи. Здесь мы изложим классический алгоритм, предложенный в книге Э. Хайрера [2], который основан на идеях из теории управления и хорошо работает для большинства нежестких задач.

Пусть y_m^{α} и $\hat{y}_m^{\alpha}$ – два численных решения разного порядка аппроксимации, вычисленные на шаге m работы алгоритма. Потребуем, чтобы $|y_m^{\alpha} - \hat{y}_m^{\alpha}| \le sc^{\alpha}$, где величина желаемой локальной погрешности sc^{α} (scale) вычисляется по формуле

$$sc^{\alpha} = A_{tol} + \max(|y_m^{\alpha}|, |\hat{y}_m^{\alpha}|)R_{tol},$$

где A_{tol} и R_{tol} – желаемая величина абсолютной и относительной погрешности соответственно. Допустимая ошибка на шаге m находится как среднее квадратичное:

$$E_m = \sqrt{\frac{1}{N}\sum_{\alpha=1}^{N}\left(\frac{y_m^{\alpha} - y_m^{\alpha}}{sc^{\alpha}}\right)^2}.$$

Новая величина шага вычисляется по формуле

$$h_{m+1} = h_m / \max(f_{\min}, \min(f_{\max}, E_m^a E_{m-1}^{-b}/f_s)),$$

где E_{m-1} – величина ошибки, вычисленная на предыдущем шаге. Фактор f_s обычно равен 0.9 или 0.8 и предназначен для предотвращения слишком резкого увеличения величины шага. Показатели степени a и b подбираются исходя из задачи. В книге [2] рекомендуются следующие универсальные значения $a = 0.7/p - 0.75b$ и $b = 0.4/p$, где p – порядок аппроксимации используемого метода. Факторы $f_{\min}$ и $f_{\max}$ позволяют ограничить границы изменения размера шага и также зависят от решаемой задачи. На практике хорошо зарекомендовали себя значения $f_{\min} = 0.1$ и $f_{\max} = 5.0$.

В случае $E_m < 1$ вычисленные значения y_m^{α} считаются удовлетворительными и метод переходит к следующему шагу вычислений. Если же $E_m > 1$, то вычисления считаются неудовлетворительны-

ми и текущий шаг повторяется с новым значением величины h_m

$$h_m \leftarrow h_m / \min(f_{\max}, E_m^a / f_s).$$

В качестве начального значения y_m^α на следующем шаге $m + 1$ можно использовать как непосредственно y_m^α, так и значение вложенного метода $\hat{y}_m^\alpha$.

3. ОПИСАНИЕ ПРОГРАММЫ

3.1. Мотивация использования генерации кода

Универсальная реализация явного вложенного метода Рунге–Кутты подразумевает, что программе в качестве параметров должны передаваться массивы коэффициентов из таблиц Бутчера, которые она будет использовать для проведения расчетов. Очевидным способом хранения данных коэффициентов является использование массивов. Однако матрица a_j^i является нижне-диагональной и хранение ее в виде массива $s \times s$ приводит к тому, что более чем половина выделенной для массива памяти тратится на хранение нулей. Коэффициенты c^i, b_j и $\hat{b}_j$ также часто имеют в своем составе нули, хранение которых неразумно.

В связи с этим большинство кодов, реализующих явные вложенные методы Рунге–Кутты, используют для хранения коэффициентов набор именованных констант, а не массивы. Это вызвано также и тем, что операции со скалярными величинами в большинстве языков проводятся быстрее, чем операции с массивами.

В языке Julia стандартные массивы являются динамическими, поэтому накладные расходы на хранение двумерного массива $s \times s$ больше, чем на расходы на хранение $s \times s$ именованных констант.

При сохранении требования универсальности создаваемого кода и вместе с тем желание увеличить скорости вычислений и уменьшить расход памяти, привели нас к решению использовать автоматическую генерацию кода по одному шаблону для каждого отдельного метода.

Кроме выигрыша в производительности, автоматическая генерации кода позволяет добавлять или изменять не каждую функцию по отдельности, а все функции совокупно путем редактирования одного лишь шаблона. Это позволяет как уменьшить количество ошибок, так и генерировать различные варианты функций для разных целей.

3.2. Описание реализации генератора методов Рунге–Кутты

В качестве языка для генерации кода нами был выбран язык Python, так как он поддерживает разнообразные средства для работы с текстом. Также в стандартной библиотеке Python присутствует тип данных `Fraction`, который позволяет задать коэффициенты Рунге–Кутты в виде рациональных дробей, а потом уже преобразовывать их в вещественный вид с нужным порядком точности.

Кроме самого языка Python была использована библиотека для обработки шаблонов Jinja2 [19]. Данный шаблонизатор (template engine) разрабатывался изначально для генерации HTML-страниц, однако он обладает очень гибким синтаксисом и может использоваться как универсальное средство для генерации текстовых файлов любого вида, в том числе и исходных кодов на любых языках программирования. Кроме Jinja2 мы использовали библиотеку `numpy` [8] для работы с массивами коэффициентов.

Шаблоны для генерации функций хранятся в файлах `rk_method.jl` и `erk_method.jl`, которые представляют собой исходный код на языке Julia с инструкциями Jinja2. Шаблонизатор позволяет поместить всю логику генерации кода в шаблон и передавать извне только данные о методе.

Информация о численных схемах хранится в виде JSON-файла, где каждый метод представляет собой JSON-объект примерно следующего вида:

```
{ "name": "Название (одним словом)",
"description": "Краткое описаниие" ,
"stage": s,
"order": p,
"extrapolation_order": p_hat,
"a": [["0" , "0" , "0" ], ["1" , "0" ,
↪ "0" ], ["1/4" , "1/4" , "0" ]],
"b": ["1/2" , "1/2" , "0" ],
"b_hat": ["1/6" , "1/6" , "2/3" ],
"c": ["0" , "1" , "1/2" ]}
```

Название JSON-объекта используется в качестве имени сгенерированной в последствии функции. Массивы `a, b, b_hat, c` могут быть как числового, так и строкового типа. Если коэффициенты метода заданы в виде рациональных дробей, то можно указать их в виде `"m/n"`, затем они будут преобразованы в объект Fraction стандартной библиотеки Python, а в теле сгенерированной функции будут представлены десятичной дробью двойной точности с 17 значащими знаками.

Список объектов указанного выше вида последовательно обрабатывается скриптами `erk_generator.py` и `rk_generator.py`. Скрипты, используя переданную им информацию о мето-

дах, генерируют для каждого метода код нескольких функций языка Julia.

Скрипты позволяют сгенерировать функции для 16 вложенных методов Рунге–Кутты. Методы низкого порядка взяты из книги Хайрера [1]. Из методов пятого порядка и выше заданы коэффициенты из работ [4, 5, 21] и [22, 23]. Метод из работы [24] реализован отдельно, так как предусматривает переключение порядка точности и сложный алгоритм управления шагом. Для добавления своих методов следует задать JSON-объект указанного выше формата и запустить скрипты вновь. Для добавленных методов будут сгенерированы соответствующие функции.

3.3. Описание сгенерированных функций

Разработанная нами библиотека имеет стандартную для модулей языка Julia структуру. Весь исходный код находится в каталоге `src`. В подкаталоге `src/generated` располагаются файлы с автоматически генерируемыми функциями, которые, в свою очередь, с помощью директивы `include` включаются в главный файл модуля `src/RungeKutta.jl`. Код, ответственный за управление шагом, находится в файле `StepControl.jl` и является общим для всех методов.

Для каждого из вложенных методов Рунге–Кутты генерируются три функции, две из которых имеют следующий вид:

```
ERK(func, A_tol, R_tol, x_0, t_start,
↪ t_stop) -> (T, X)
ERK(func, A_tol, R_tol, x_0, t_start,
↪ t_stop, last) -> (t, x)
```

Функции имеют одинаковое имя, а список их аргументов различается лишь последним аргументом. Благодаря поддержке языком Julia множественной диспетчеризации (перегрузка функций), компилятор сам определяет в каком случае какую из реализаций вызывать.

В указанных выше функциях:

• `func::Function` – правая часть системы ОДУ $\dot{x}^{\alpha}(t) = f^{\alpha}(t, x^{\beta})$: `func(t, x)`, где `t::Float64` – время, `x::Vector{Float64}` – значение функции $x(t)$; аргумент `t` должен всегда присутствовать в вызове функции, даже если система автономная и явно от времени не зависит;

• аргументы `A_tol` и `R_tol` должны иметь тип `Float64` и обозначают абсолютную и относительную точности методы соответственно;

• `x_0::Vector{Float64}` – начальное значение функции $x_0^{\alpha} = x^{\alpha}(t_0)$;

• `t_start` и `t_stop` имеют тип `Float64` и обозначают начальную и конечную точки промежутка интегрирования;

• если указан аргумент `last::Bool`, то будут возвращены последняя точка промежутка интегрирования t_n и вектор $x_n^{\alpha} \approx x^{\alpha}(t_n)$, в противном случае возвращаются массивы `T::Vector(Float64}` и `Matrix{Float64}`.

Для методов Рунге–Кутты без управления шагом генерируются аналогичные функции, единственное отличие которых заключается в отсутствии аргументов `A_tol` и `R_tol`. Вместо них следует передавать аргумент `h`, который задает размер шага, используемого для вычислений.

Для реализации алгоритма управления шагом генерируется еще одна функция:

```
ERK_info(func, A_tol, R_tol, x_0,
↪ t_start, t_stop) -> (accepted_t,
↪ accepted_h, rejected_t, rejected_h)
```

Эта функция возвращает следующие массивы:

• `accepted_t`: все точки сетки, в которых погрешность вычислений была признана удовлетворительной;

• `accepted_h`: все принятые размеры шагов;

• `rejected_t`: все точки сетки, в которых погрешность вычислений была признана неудовлетворительной;

• `rejected_h`: все отбракованные размеры шага;

• `errors`: значения локальной ошибки E_n.

Все возвращаемые значения имеют тип `Vector{Float64}`.

4. ТЕСТИРОВАНИЕ СГЕНЕРИРОВАННЫХ ПРОГРАММНЫХ КОДОВ ЧИСЛЕННЫХ СХЕМ

Для тестирования созданных методов были использованы три системы дифференциальных уравнений, рассмотренные в [2]. Первая система – уравнения ван дер Поля:

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} - \mu(1 - x^2)\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0,$$

$$\begin{cases} \dot{x}_1(t) = x_2(t), \\ \dot{x}_2(t) = \mu(1 - x_1^2(t))x_2(t) - x_1(t), \end{cases}$$

заданные со следующими начальными значениями:

$$\mu = 1, \quad \mathbf{x} = (0, \sqrt{3})^T, \quad 0 \le t \le 12,$$

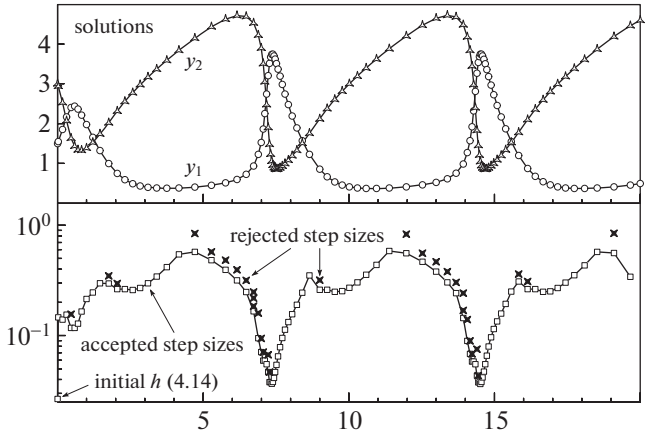

Рис. 1. Решение системы (2) и отброшенные и принятые шаги (рисунок из [1, с. 170, рис. 4.1]).

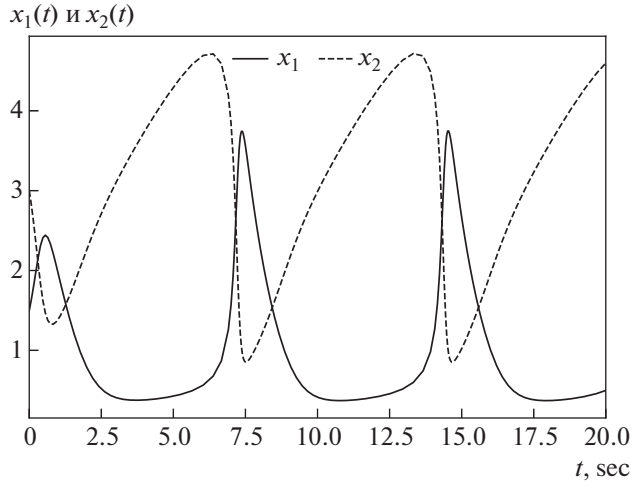

Рис. 2. Решение системы (2).

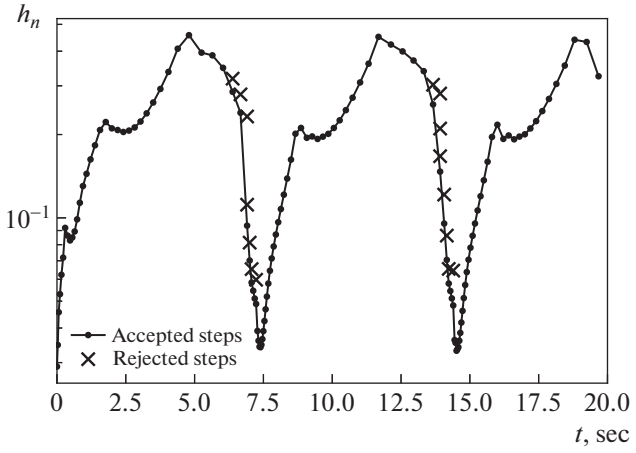

Рис. 3. Отброшенные и принятые шаги (2).

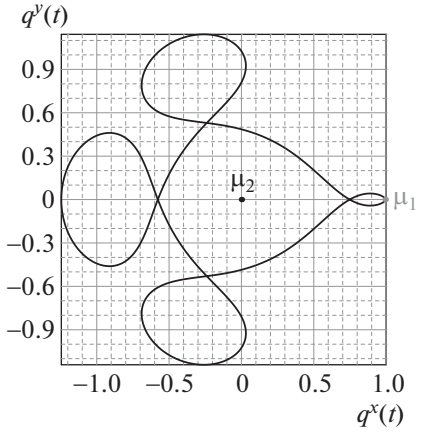

Рис. 4. Орбита для первой группы начальных значений (3).

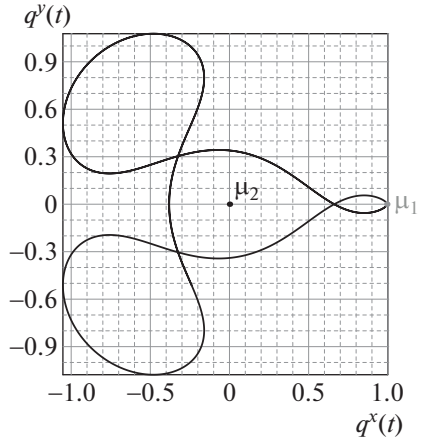

Рис. 5. Орбита для второй группы начальных значений (4).

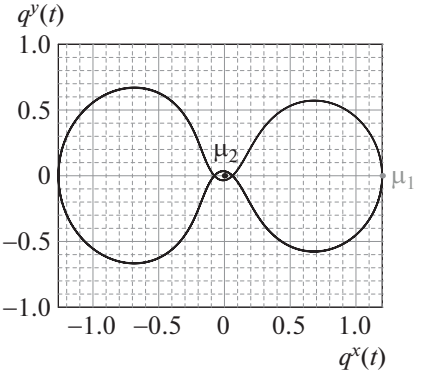

Рис. 6. Орбита для третьей группы начальных значений (5).

Таблица 1. Глобальные погрешности численных методов за один оборот по орбите для первой группы начальных значений

Метод	Погрешность
DPRK546S	1.84566×10^{-12}
DPRK547S	2.93634×10^{-12}
DPRK658M	4.22771×10^{-12}
Fehlberg45	7.42775×10^{-12}
DOPRI5	1.95463×10^{-13}
DVERK65	1.67304×10^{-12}
Fehlberg78B	5.45348×10^{-12}
DOPRI8	1.06343×10^{-11}

где коэффициент μ характеризует нелинейность и силу затухания колебаний.

Вторая система – уравнения твердого тела без внешних сил, или иначе, уравнения Эйлера твердого тела:

$$\begin{cases} \dot{x}_1(t) = I_1 x_2(t) x_3(t), \\ \dot{x}_2(t) = I_2 x_1(t) x_3(t), \\ \dot{x}_3(t) = I_3 x_1(t) x_2(t), \\ I_1 = -2, \quad I_2 = 1.25, \quad I_3 = -0.5, \\ \mathbf{x} = (0,1,1)^T, \quad t = [0,12]. \end{cases}$$

Для проверки работы алгоритма управления шагом применялась численная схема вложенного метода Рунге–Кутты с $p = 4$ и $\hat{p} = 3$ к системе уравнений брюсселятора:

$$\begin{cases} \dot{x}_1(t) = 1 + x_1^2 x_2 - 4x_1, \\ \dot{x}_2(t) = 3x_1 - x_1^2 x_2. \end{cases}$$

Для начальных условий $x_1(0) = 1.5$, $x_2(0) = 3$ уравнение брюсселятора было численно проинтегрировано методом ERK43b на отрезке $0 \le t \le 20$ с абсолютной и относительной погрешностями $A_{tol} = R_{tol} = 10^{-4}$. Были построены графики, идентичные графикам из книги [1, с. 170, рис. 4.1]. При сравнении результатов из книги [1] (см. рис. 1) с полученными нами результатами (см. рис. 2 и 3) видно, что реализованный нами метод работает практически также, как и использованный в [1], однако выбирает размер шага аккуратнее.

Еще одна часто применяющаяся для тестирования численных схем система уравнений – частный случай ограниченной задачи трех тел, называемый задачей по вычислению орбит Аренсторфа. Свое название данная задача получила в честь Ричарда Ф. Аренсторфа – американского физика, рассчитавшего стабильную орбиту малого тела между Луной и Землей.

В ограниченной задаче трех тел рассматривается движение малого тела в поле тяготения среднего и большого тел (Луна и Земля). Масса малого тела считается равной нулю, массы среднего и большого тел – μ_1 и μ_2 соответственно. Предполагается, что орбиты тел лежат в одной плоскости.

В задаче Аренсторфа в зависимости от начальных значений можно получить разные стабильные орбиты [1]. В безразмерных синодических координатах три группы начальных значений, дающих три разные орбиты, имеют следующий вид:

$$p_y = -1.00758510637908238, \quad p_x = 0.0, \quad q^x = 0.994, \quad q^y = 0.0; \tag{2}$$

$$p_y = -1.03773262955733680, \quad p_x = 0.0, \quad q^x = 0.994, \quad q^y = 0.0; \tag{3}$$

$$p_y = 0.15064248999999985, \quad p_x = 0.0, \quad q^x = 1.2, \quad q^y = 0.0. \tag{4}$$

Здесь p_x, p_y – обобщенные импульсы системы, q^x, q^y – обобщенные координаты системы.

Важно отметить, что начальные значения специально указаны с большой точностью, так как малое тело крайне близко подходит к среднему телу и даже небольшая погрешность вычислений может привести к ошибке, что физически интерпретируется как падение малого тела на среднее. Как раз из-за такой особенности данная задача хорошо подходит для проверки реализации численных схем.

Функция Гамильтона в синодических координатах записывается следующим образом:

$$H(p_x, p_y, q^x, q^y) = \frac{1}{2}(p_x^2 + p_y^2) + p_x q^y - p_y q^x - F(q^x, q^y),$$

где

$$F(q^x, q^y) = \frac{\mu_1}{r_1} + \frac{\mu_2}{r_2}, \quad \mu_1 + \mu_2 = 1,$$
$$r_1 = \sqrt{(q^x - \mu_2)^2 + (q^y)^2},$$
$$r_2 = \sqrt{(q^x + \mu_1)^2 + (q^y)^2};$$

$$\frac{\partial F}{\partial q^x} = -\frac{\mu_1(q^x - \mu_2)}{r_1^3} - \frac{\mu_2(q^x + \mu_1)}{r_2^3},$$
$$\frac{\partial F}{\partial q^y} = -\frac{\mu_1 q^y}{r_1^3} - \frac{\mu_2 q^y}{r_2^3}.$$

Канонические уравнения, к которым и будет применяться численная схема, имеют следующий вид:

$$\begin{cases} \dot{p}_x = +p_y + \dfrac{\partial F}{\partial q^x}, \quad \dot{q}^x = p_x + q^y, \\ \dot{p}_y = -p_x + \dfrac{\partial F}{\partial q^y}, \quad \dot{q}^y = p_y - q^x. \end{cases}$$

Численное решение проводится для $0 \le t \le 17.065216560157962558$ с абсолютной и относительной погрешностями $A_{tol} = 10^{-17}$ и $R_{tol} = 0$ и средней массой $\mu_1 = 0.012277471$. В результате получаем орбиты Аренсторфа (рис. 4, 5 и 6) для начальных значений (2), (3) и (4) соответственно.

Для оценки погрешности система решается численно для временного промежутка, равного одному периоду. В конечной точке промежутка малое тело должно вернуться в начальную точку, т.е. $(q^x(0), q^y(0)) = (q^x(T), q^y(T))$, поэтому погрешность $\|\mathbf{q}_0 - \mathbf{q}_n\|$ даст глобальную погрешность метода. Значения погрешности для первой группы начальных значений приведено в табл. 1.

5. ЗАКЛЮЧЕНИЕ

Таким образом, можно сформулировать основные результата нашей работы:

1. В качестве специализированной системы символьных вычислений применен язык шаблонизатора Jinja2. Относительно использования универсальной системы символьных вычислений данное решение позволило сделать программный комплекс более простым, компактным и увеличить его переносимость.

2. Создан набор сценариев на языке Python с использованием языка шаблонизатора Jinja2, которые генерируют код на языке Julia, реализующий численные схемы методов Рунге–Кутты без управления шагом, вложенных методов с управлением шагом и методов Розенброка также с управлением шагом (все программы находятся в открытом доступе и расположены по адресу https://bitbucket.org/mngev/rungekutta_generator).

3. Сгенерированные функции протестированы с помощью решения нескольких типовых задач (непосредственно в тексте статьи подробно разобрана задача Аренсторфа, как наиболее требовательная к точности численной схемы); модуль для языка Julia также доступен по адресу https://bitbucket.org/mngev/rungekutta-autogen.

6. БЛАГОДАРНОСТИ

Публикация подготовлена при поддержке Программы РУДН “5-100” и при финансовой поддержке РФФИ в рамках научного проекта 19-01-00645.

СПИСОК ЛИТЕРАТУРЫ

1. *Hairer E., Nørsett S.P., G. Wanner*. Solving Ordinary Differential Equations I. 2 edition. Berlin: Springer, 2008.
2. *Hairer E., Wanner G.* Solving Ordinary Diffrential Equations II. Stiff and Differential-Algebraic Problems. 2 edition. 1996.
3. Fortran and Matlab Codes. URL: https://www.unige.ch/~hairer/software.html.
4. *Dormand J.R., Prince P.J.* A family of embedded Runge–Kutta formulae // Journal of computational and applied mathematics. 1980. Vol. 6, no. 1. P. 19–26.
5. *Prince P.J., Dormand J.R.* High order embedded Runge–Kutta formulae // Journal of Computational and Applied Mathematics. 1981. V. 7. № 1. P. 67–75.
6. Matlab. 2020. URL: https://www.mathworks.com/products/matlab.html.
7. GNU Octave. 2020. URL: https://www.gnu.org/software/octave/.
8. Jones E., Oliphant T., Peterson P., Others. SciPy: Open source scientific tools for Python. URL: http://www.scipy.org/.
9. Scilab. 2020. URL: http://www.scilab.org/.
10. Boost. Boost C++ Libraries. 2020. URL: http://www.boost.org/.
11. Gevorkyan M.N., Velieva T.R., Korolkova A.V. et al. Stochastic Runge–Kutta Software Package for Stochastic Differential Equations // Dependability Engineering and Complex Systems. Springer International Publishing, 2016. V. 470. P. 169–179. arXiv: 1606.06604.
12. *Kulyabov D.S., Gevorkyan M.N., Demidova A.V. et al.* Implementation Diffi«culties Analysis of Stochastic Numerical Runge–Kutta Methods // 2nd International Scientific Conference “Convergent Cognitive Information Technologies”, Convergent 2017 / Ed. by M. Shneps-Shneppe, V. Sukhomlin, E. Zubareva. Vol. 2064 of CEUR Workshop Proceedings. Moscow: CEUR-WS, 2017. 11. P. 28–40.
13. *Gevorkyan M.N., Demidova A.V., Korolkova A.V., Kulyabov D.S.* Issues in the Software Implementation of Stochastic Numerical Runge–Kutta // Distributed Computer and Communication Networks / Ed. by Vladimir M. Vishnevskiy, Dmitry V. Kozyrev. Cham: Springer International Publishing, 2018. Vol. 919 of Communications in Computer and Information Science. P. 532–546. arXiv: 1811.01719.
14. Bezanson J., Edelman A., Karpinski S., Shah V.B. Julia: A Fresh Approach to Numerical Computing. 2014.
15. *Bezanson J., Karpinski S., Shah V.B., Edelman A. Julia*: A Fast Dynamic Language for Technical Computing. 2012.

16. *Kwong T.* Hands-On Design Patterns and Best Practices with Julia. Birmingham B3 2PB, UK: Packt Publishing, 2020.

17. *Геворкян М.Н., Королькова А.В., Кулябов Д.С., Севастьянов Л.А.* Пример модульного расширения системы компьютерной алгебры // Программирование. 2020. № 2. С. 30–37.

18. Python Reference Manual: Rep.; Executor: Guido Rossum. Amsterdam, The Netherlands, The Netherlands: 1995.

19. Jinja2 offial site. URL: http://http//jinja.pocoo.org.

20. *Butcher J.C.* Numerical Methods for Ordinary Differential Equations. 2 edition. New Zealand: Wiley, 2003.

21. *Dormand J.R., Prince P.J.* A reconsideration of some embedded Runge–Kutta formulae // Journal of Computational and Applied Mathematics. 1986. V. 15. № 2. P. 203–211.

22. *Fehlberg E.* Klassische Runge–Kutta Formeln fünfter und siebenter Ordnung mit Schrittweiten-Kontrolle // Computing. 1969. 6. Bd. 4, H. 2. S. 93–106.

23. *Fehlberg E.* Klassische Runge–Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme // Computing. 1970. 3. Bd. 6, H. 1–2. S. 61–71.

24. *Cash J., Karp A.* A variable order Runge–Kutta method for initial value problems with rapidly varying right-hand sides // ACM Trans. Math. Softw. 1990. V. 16. № 3. P. 201–222.