ПРОГРАММИРОВАНИЕ, 2023, № 5, с. 47–58

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 004.421.6

О РЕАЛИЗАЦИИ ЧИСЛЕННЫХ МЕТОДОВ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ В СИСТЕМАХ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

© 2023 г. А. Баддур[a], М. М. Гамбарян[a], Л. Гонсалес[a], М. Д. Малых[a,b,*]

[a] *Российский университет дружбы народов*
117198 Москва, улица Миклухо-Маклая, 6, Россия
[b] *Объединенный институт ядерных исследований*
141980 Дубна Московской области, Россия
**E-mail: malykh_md@pfur.ru*

Поступила в редакцию 21.06.2022 г.
После доработки 28.07.2022 г.
Принята к публикации 30.10.2022 г.

В статье представлен оригинальный пакет для исследования численных решений обыкновенных дифференциальных уравнений, встраиваемый в систему компьютерной алгебры Sage. Этот проект направлен на более тесную интеграцию численных и символьных методов и прежде всего преследует цель создания удобного инструмента для работы с численными решениями в Sage. В этом пакете определено два новых класса – начальные задачи и приближенные решения. Внутри первого класса определены инструменты для символьных вычислений, связанных с начальными задачами, внутри второго – инструменты для интерполяции значений символьных выражений на приближенном решении и оценивания ошибки по методу Ричардсона. Затем кратко описана реализация метода Рунге–Кутты, главная особенность которой – возможность работы с произвольными таблицы Бутчера и произвольными числовыми полями.

DOI: 10.31857/S013234742302005X, **EDN:** GZBPBR

1. ВВЕДЕНИЕ

Ввиду востребованности интегрирования дифференциальных уравнений при решении прикладных задач первые интеграторы были созданы на заре появления компьютерной техники [1]. Численные интеграторы обыкновенных дифференциальных уравнений (ОДУ), находящиеся сейчас во всеобщем употреблении, были разработаны в 1980-х годах. Ряд удачных разработок был относительно недавно переписан на python и доступен в библиотеке SciLab [2]. Среди многочисленных альтернатив этому собранию, следует выделить проект nodepy (`https://github.com/ketch/nodepy`), позволяющий проводить компьютерные эксперименты со схемами высокого порядка [3].

При сравнении разностных схем внимание исследователей всегда было сосредоточено на количественной близости точных и приближенных решений, а вопрос о сохранении качественных свойств точного решения до сих пор остается недостаточно изученным. Заманчивая возможность “определить характер динамического процесса с использованием только грубых вычислений с большим шагом сетки” по симплектической схеме была отмечена в [4]. В теории дифференциальных уравнений в частных производных дискретизации, наследующие некоторые свойства дифференциальных уравнений, называются миметическими (mimetic), то есть подражающими [5, 6]. Поэтому представляется целесообразным рассматривать разработку новых разностных схем в контексте концепции наследования алгебраических и качественных свойств динамических систем, разумеется, уточняя саму концепцию подражания.

Исторически первым был выделен класс схем Рунге–Кутты, сохраняющих симплектическую структуру гамильтоновых динамических систем. Из общих соображений можно было бы ожидать, что сохранение симплектической структуры должно иметь следствием сохранение всех алгебраических интегралов движения, однако оказалось, что сохраняются точно только линейные и квадратичные интегралы [7]. Можно конструировать разностные схемы, сохраняющие все алгебраические интегралы движения динамической системы, напр., задачи многих тел, однако при этом приходится жертвовать гамильтоновой структурой [8, 9], а можно – схемы, сохраняющие свойство обратимости исходной динамической системы [10].

Инструменты, необходимые для проектирования и исследования алгебраических свойств таких схем, уже имеются в системах компьютерной алгебры. Однако сами эти системы предоставляют весьма бедный инструментарий для работы с приближенными решениями, поскольку этот вопрос традиционно относится к численным методам. Напр., в Sage [11] по умолчанию доступен один единственный численный метод решения ОДУ – метод Рунге–Кутты 4-го порядка с постоянным шагом. При этом сама реализация, очевидно, не писалась специально для Sage, поскольку она не позволяет менять поле, над которым ведутся расчеты.

В настоящей статье представлен оригинальный пакет `fdm`, встраиваемый в систему компьютерной алгебры Sage. Наша цель – создать удобный инструмент для численно-аналитического исследования разностных схем, хорошо интегрированный с алгебраическим инструментарием.

2. ОПИСАНИЕ НАЧАЛЬНОЙ ЗАДАЧИ

В известных нам системах задание начальной задачи и отыскание ее приближенного решения выполняется в одно действие. Такой подход препятствует четкому разделению корректно поставленной математической задачи и метода ее решения. Поэтому в нашем пакете начальная задача описывается отдельно от метода ее решения как элемент специально созданного для этого класса `Initial_problem`. Начальная задача

$$\begin{cases} \dfrac{dx_i}{dt} = f_i(t, x_1, \ldots, x_n), \quad i = 1, \ldots, n \\ x_i(0) = x_i^{(0)}, \quad i = 1, \ldots, n \end{cases} \tag{2.1}$$

на отрезке $0 < t < T$ описывается списком $[x, f, x_0, T]$, где $x = [x_1, \ldots, x_n]$ – список используемых переменных, $f = [f_1, \ldots, f_n]$ – список правых частей, элементы которого являются символьными выражениями, $x_0 = [x_1^{(0)}, \ldots, x_n^{(0)}]$ – список начальных данных, и T – конечное время. Начальные задачи рассматриваются как элементы класса `Initial_problem`.

Напр., начальная задача

$$\begin{cases} \dfrac{dx_1}{dt} = x_2, \quad \dfrac{dx_2}{dt} = -x_1, \\ x_1(0) = 0, \quad x_2(0) = 1 \end{cases} \tag{2.2}$$

на отрезке $0 < t < 10$ описывается так:

```
var("t,x1,x2")
pr1=Initial_problem([x1, x2], [x2, -x1],
[0,1], 10)
```

В нашей системе независимая переменная всегда обозначается как t и интерпретируется как время, за начальный момент всегда принимается $t = 0$. Это соглашение позволяет использовать в аргументах `Initial_problem` списки одной длины. В стандартной реализации метода Рунге–Кутты в Sage эти списки отличаются, что является источником многочисленных опечаток при ее использовании.

Начальные условия $x_1^{(0)}, \ldots, x_n^{(0)}$ и конечный момент времени T являются “числами”, однако с точки зрения компьютерной алгебры обычно они задаются символьными выражениями, которые можно преобразовать в элемент поля вещественных чисел $\mathbb{R}$. Напр., в качестве начального значения может быть использован и $\sqrt{2}$, и sin1. Преобразование в десятичные дроби такого рода выражений вносит ошибку округления и поэтому в нашей системе делается на стадии отыскания численного решения.

Функция `latex`, определенная в этом классе, возвращает начальную задачу в нотации LaTeX, что очень удобно для проверки корректности задания задачи.

Выделение начальных задач в особый класс позволило реализовать внутри этого класса вычисление в символьном виде ряда полезных для пользователя конструкций. Дело в том, что не зная решения начальной задачи, мы можем вычислить в символьном виде полную производную по t любого символьного выражения u, зависящего явно от x, t, многочлен Тейлора для этой функции, кривизну интегральной кривой в заданной точке и т.п. Приведем несколько примеров.

Для начальной задачи (2.2) полная производная функции $u = x_1^2 + x_2^2$ вычисляется так:

```
pr1.diff(x1^2+x2^2)
```

В данном случае возвращается нуль. Разложение в ряд Тейлора функции $v = x_1^2$ до членов второго порядка вычисляется так:

```
pr1.taylor(x1^2,2)
```

$$-(x_1^2 - x_2^2)(t-\tau)^2 - 2(t-\tau)x_1x_2 + x_1^2$$

Здесь τ – центр разложения. Кривизна интегральной кривой в точке (x_1, x_2) вычисляется так:

```
pr1.curvature()
```

$$\frac{(x_1^2 + x_2^2)^2 + x_1^2 + x_2^2}{(x_1^2 + x_2^2 + 1)^{\frac{3}{2}}}$$

В будущем мы планируем интегрировать в этот класс инструменты для отыскания рациональных интегралов, используя программное обеспечение для Sage, созданное в рамках диссертационного исследования Юй Ин [12].

Для генерации начальной задачи тоже можно использовать символьные вычисления, что осо-

бенно удобно в случае, если тел действительно много.

3. ИНСТРУМЕНТЫ ДЛЯ РАБОТЫ С ЧИСЛЕННЫМИ РЕШЕНИЯМИ НАЧАЛЬНОЙ ЗАДАЧИ

Численное решение задачи (2.1) выполняется над некоторым полем. Условимся обозначать поле, над которым мы работаем, как $\mathbb{K}$. По умолчанию в качестве такого используется стандартная реализация поля вещественных чисел $\mathbb{R}$ как множества десятичных дробей с плавающей запятой и фиксированным количеством бит, используемых для представления мантиссы. Как уже отмечалось выше, пользователь не обязан задавать начальные условия как элементы этого класса, но система должна иметь возможность преобразовать начальные условия в элементы поля $\mathbb{K}$.

Численное решение задачи (2.1) представляет собой список точек, каждый элемент которого – список координат точек решения: первым идет значение t, а затем $x_1,\ldots,x_n$ в том порядке, в котором они идут в x. Здесь используется тот же формат списка, что и в стандартной реализации метода Рунге-Кутты в Sage. Однако сами числа, входящие в списки, не обязаны принадлежать именно полю $\mathbb{R}$, но должны принадлежать полю $\mathbb{K}$.

Целый ряд манипуляций с этим списком можно делать независимо от метода, которым он был получен, поэтому мы создали для него особый класс – `Numsol`. Помимо списка точек приближенного решения элемент этого класса содержит указание на начальную задачу, порядок аппроксимации этой задачи разностной схемой и шаг равномерной сетки Δt или аналог h этой величины для квазиодномерной сетки.

В нашем пакете все решатели возвращают элемент класса `Numsol`. Напр., явный метод Рунге–Кутты реализован в виде функции `erk`, аргументами которой служат начальная задача и, опционально, число N отрезков, на которые делится рассматриваемый интервал (см. ниже). Она возвращает не список точек приближенного решения, но элемент класса `Numsol`. Напр., для задачи (2.2)

```
Q=erk(pr1,N_)
```

Конечно, предусмотрена возможность вывести список точек приближенного решения:

```
Q.list()
```

Однако пользователь может и не знать об этой возможности, поскольку в нашем пакете предусмотрена возможность вывести значение любого символьного выражения u от t,x в любой точке отрезка $[0,T]$. Напр., значение выражения x_1+t при $t=\pi$ можно найти так:

```
sage: Q.value(x1+t, pi)
```

3.15802440240400.

Для вычисления значения выражения u в точках, не попавших в узлы разбиения отрезка $[0,T]$, используется аппроксимация многочленом Тейлора, порядок которого согласован с порядком аппроксимации, указанным решателем при создании решения как элемента класса `Numsol`.

Замечание. Мы пытались использовать вместо затратной операции разложения по формуле Тейлора сплайны. Однако встроенная в Sage функция для аппроксимации сплайнами не позволяет менять поле $\mathbb{K}$ и ее не удалось согласовать с порядком аппроксимации разностной схемы при больших порядках аппроксимации.

Метод `zeros` позволяет найти нули выражения u на отрезке $0 < t \le T$. Напр., найдем нули x_1:

```
Q.zeros(x1)
```

[3.1430180411731743,…]

Определение нулей происходит по перемене знака, поэтому нули четной кратности найдены не будут.

Метод `plot` позволяет построить двумерный график зависимости одного символьного выражения от любого другого. Напр., график зависимости $x_1^2 - x_2^2$ от $x_1^2 + x_2^2$ можно вывести так:

```
sage: Q.plot(x1^2+x2^2,x1^2-x2^2)
```

При этом оси подписываются автоматически, при $N < 51$ указываются точки приближенного решения, при большем числе точек они соединяются сплошной линией. Этот метод поддерживает стандартные опции: `axes_labels`, `linestyle`, `color`. Остальные опции, традиционно используемые в графике для Sage, можно определить через метод `show`. Таким образом, вычисление символьного выражения на решении и построение его графика не требуют от пользователя понимания того, как устроено численное решение.

Для того, чтобы охарактеризовать численный метод, полезно использовать метод `plot_dt`, который строит двумерный график зависимости шага от t:

```
Q.plot_dt()
```

В данном случае, шаг постоянный и зависимость тривиальная.

4. РЕАЛИЗАЦИЯ МЕТОДА РИЧАРДСОНА

В работах Ричардсона для оценок ошибок, возникающих при вычислении определенных интегралов по методу конечных разностей, было предложено сгущать сетку, а в работах Рунге схожий прием был приложен к исследованию обыкновенных дифференциальных уравнений. Этот подход был систематически разработан в работах

Н.Н. Калиткина и его учеников [13–16] и будет далее называться метод Ричардсона.

На наш взгляд, этот метод особенно просто описывается как метод оценки ошибки вычисления значения символьного выражения u в заданной точке $t = a$ [17]. Итак, пусть задана начальная задача, момент времени $t = a$, попадающий на интервал $0 < a \le T$, и символьное выражение u, зависящее от $x_1,\ldots,x_n$ и t. Найдем одним и тем же методом, но с разным шагом $h = h_1, h_2,\ldots$, приближенные решения этой начальной задачи (1). Затем вычислим значения $u_1, u_2,\ldots$ выражения u в точке $t = a$ для каждого из этих решений, используя при необходимости описанную выше интерполяцию. Если порядок аппроксимации равен r и интерполяция согласована с этим порядком, то

$$u_i - u = ch_i^r + c' h_i^{r+1} + \ldots, \quad i = 1,2,\ldots \qquad (4.1)$$

Здесь $c, c',\ldots$ – неизвестные нам константы, которые не зависят от шага и характеризуют начальную задачу и метод ее решения, но, конечно, зависят от рассматриваемого момента времени $t = a$. Если $c \ne 0$, то при достаточно малых h можно оставить только главный член, отсюда

$$u_i - u = ch_i^r.$$

В таком случае

$$u_1 - u_2 = c(h_1^r - h_2^r).$$

Поэтому ошибку аппроксимации

$$u \simeq u_2$$

можно оценить как

$$ch_2^r = \frac{u_1 - u_2}{\left(\frac{h_1}{h_2}\right)^r - 1}.$$

В нашем пакете функция `richardson(P,Q,u,a)` возвращает по двум численным решениям P, Q значение символьного выражения u при $t = a$ и оценку совершаемой при этом ошибки E. Формат ответа согласован с функций `numerical_integral`, используемой в Sage для численного вычисления интегралов.

Пример 1. Рассмотрим решение задачи

$$\frac{dx}{dt} = t^2 + x, \quad x(0) = 0 \qquad (4.2)$$

на отрезке $0 < t < 1$ по методу `rk4` (см. п. 5.2). Найдем, напр., значение выражения $u = x^2$ при $t = 0.8$ и оценим ошибку по методу Ричардсона:

```
var("x,t")
pr2=Initial_problem(x,t^2+x,0,1)
P=erk(t^2+x,x,0, T=1, N=10)
Q=erk(t^2+x,x,0, T=1, N=20)
richardson(P,Q,x^2,0.8)
```

$$[0.0445555409432967, -7.89293834874139\times10^{-9}]$$

Замечание. Традиционно, интересуются суммой $u_2 + E$, которую интерпретируют как уточнение значения u по методу Ричардсона. Однако, на наш взгляд два числа – u_2, E – важны порознь: второе характеризует порядок ошибки численного метода. Разумеется, пользователь может сложить два элемента списка, который возвращает функция `richardson` и получить уточнение по Ричардсону.

Замечание Следует подчеркнуть, что наша оценка носит локальный характер, она зависит от выбора $t = a$. Из общих соображений можно ожидать, что с ростом a ошибка становится все больше и больше. Однако мы сталкивались со случаями, когда зависимость ошибки от a не была монотонной, напр., при исследовании системы Калоджеро [18].

Метод Ричардсона дает неверные оценки, если следующие члены ряда (3) все еще велики. Так будет, напр., если в рассматриваемой точке $t = a$ имеется суперсходимость, то есть $c = 0$ и порядок аппроксимации выше чем порядок аппроксимации разностной схемы. Поэтому перед применением метода Ричардсона рекомендуется убедиться в том, что в рассматриваемом диапазоне изменения шага h зависимость ошибки E от h является степенной и оценить ее показатель r. Для этого следует вычислить десяток-другой решений

$$L = [P_1, P_2,\ldots]$$

с разным шагом и построить зависимость E от h в двойном логарифмическом масштабе (диаграмма Ричардсона). Если эта зависимость – степенная ($E = ch^r$), то должна получиться прямая, наклон которой равен r. Мы, конечно, не знаем ошибку точно, поэтому вместо них используются оценки по Ричардсону.

В нашем пакете функция

`richardson_plot(L,u,a)`

отмечает на плоскости h, E в двойной логарифмической шкале точки

$$(h_i, E_i),$$

где h_i – значение шага, а E_i – оценки ошибки в вычислении значения выражения u при $t = a$ по Ричардсону. К этой диаграмме добавляется прямая, которая проходит к этим точкам ближе всего в смысле метода наименьших квадратов. Метод Ричардсона применим в рассматриваемом диапазоне изменения шагов, если точки с графической точностью ложатся на прямую, а наклон прямой

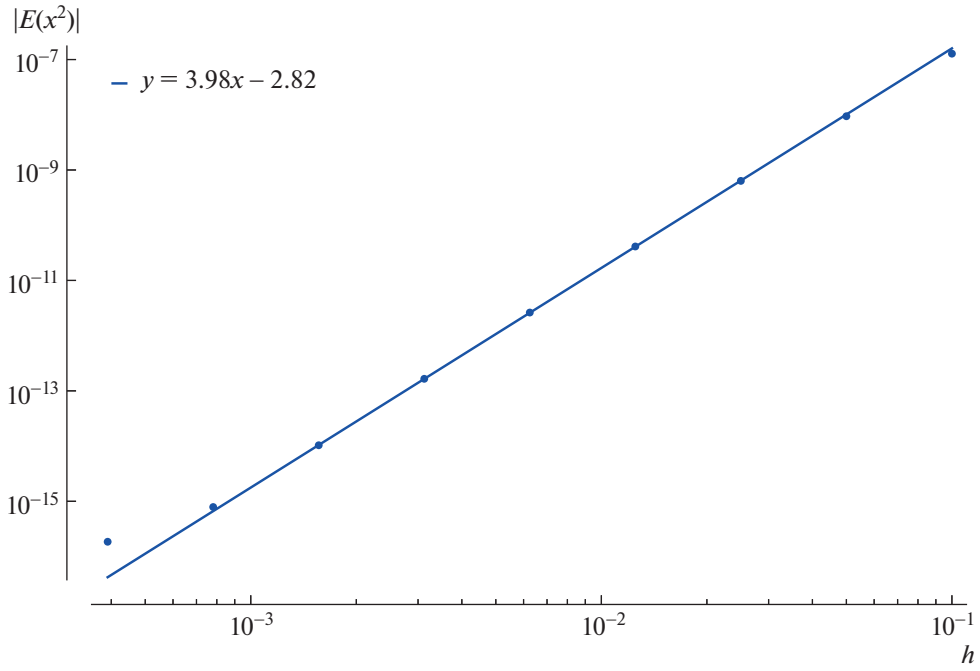

Рис. 1. Диаграмма Ричардсона для значения x_1^2 при $t = 0.8$ для примера 1.

указывает на фактический порядок аппроксимации.

Пример 2. Возвращаясь к примеру 1, мы можем построить диаграмму Ричардсона следующим образом:

```
@parallel
def foo(n):
    return erk(pr2, N=2^n*10)
L = list(foo(list(range(10))))
richardson_plot([L[i] [1] for i in
range(len(L))], x^2, 0.8, nmin=2, nmax=7)
```

Функция `richardson_plot(L,u,a)` имеет две опции (`nmin` и `nmax`), позволяющие исключить из аппроксимации прямой первые и последние точки. Результат представлен на рис. 1. Хорошо видно, что наклон равен 3.98 против ожидаемых 4. При больших шагах оказывают приметное влияние следующие члены в разложении (3), а при слишком малых h сказывается ошибка округления.

Поскольку расчеты численных решений при различных шагах никак друг с другом не связаны, построение диаграммы Ричардсона допускает естественное распараллеливание, использованное в примере 2. Благодаря декоратору `@parallel`, встроенному в Sage, на многоядерных процессорах построение диаграммы занимает столько же времени, сколько вычисление одного решения с самым мелким шагом.

Если этот порядок заметно отличается от порядка аппроксимации, то при дальнейшем применении функции `richardson` следует воспользоваться опцией `delta`, значение которой указывает на сколько следует увеличить порядок.

Метод Ричардсона может быть применен и к вычислению нулей. Напр., пусть требуется найти по методу Рунге–Кутты нули решения $x_1 = \sin t$ задачи (2.2), обозначенной выше как задача `pr1`. Вычислим список L численных решений, постепенно сгущая сетку:

```
L=[erk(pr1, N=2^n*10) for n in range(10)]
```

Вычислим нули по наиболее точному из решений:

```
L[-1].zeros(x1)
```

[3.141592653590175,…]

Функция `richardson_plot_zeros` позволяет построить диаграмму Ричардсона для каждого из нулей (нули считаются слева направо, начиная с нуля; указываются путем задания опции num):

```
richardson_plot_zeros(L, x1, num=2)
```

Результат представлен на рис. 1. Хорошо видно, что наклон графика ожидаемо равен 4, а ошибка на последних элементах списка L имеет порядок 10^{-12}. Более точно:

```
richardson_zeros(L[-1],L[-2], x1, num=2)
```

$[9.424777960770523, 1.1424342953129477 \times 10^{-12}]$

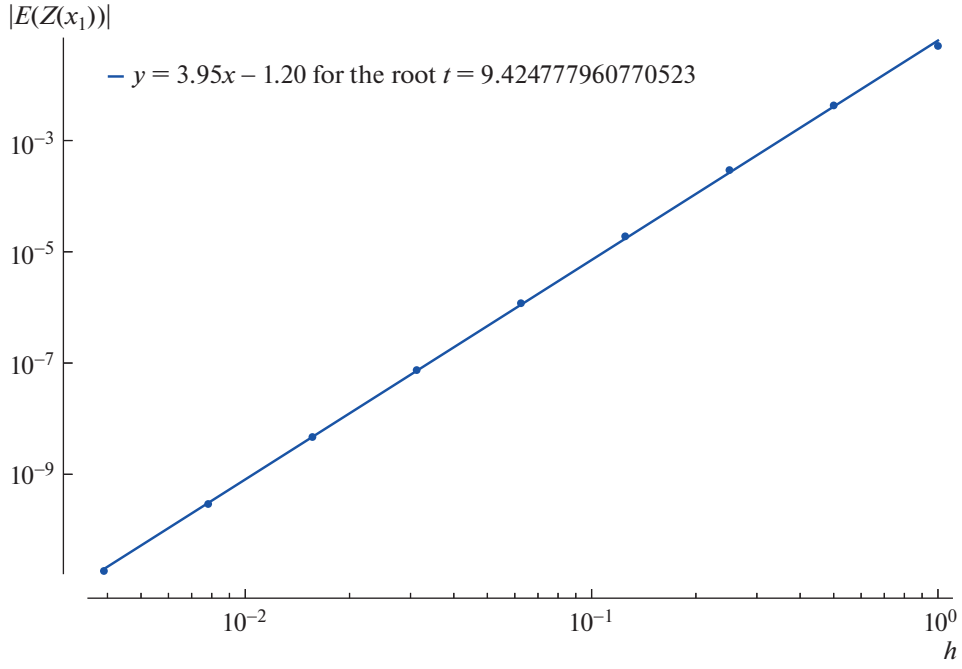

Рис. 2. Диаграмма Ричардсона для последнего нуля x_1.

Таким образом, 3π вычисляется с точностью до 11 знака после запятой, в чем в данном случае можно убедиться непосредственной проверкой.

5. РЕАЛИЗАЦИЯ МЕТОДА РУНГЕ–КУТТЫ

Для однозначной идентификации метода Рунге–Кутты достаточно указать таблицу Бутчера [19]. При реализации метода Рунге–Кутты в пакете fdm мы стремились к тому, чтобы эта реализация могла работать с любыми полями и любыми матрицами Бутчера. В настоящей момент доступны 2 реализации: отдельно для явного, отдельно для неявного методов. Первая написана Л. Гонсалесом, вторая – Али Баддуром.

5.1. Таблицы Бутчера

В нашем пакете для работы с таблицами Бутчера мы завели отдельный класс `Butcher_tableau`. Для задания таблицы нужно указать порядок аппроксимации и саму таблицу в виде списка, первым элементом которой будет матрица a, а вторым – столбец b. Столбец c вычисляется по этим данным. Опционально далее указываются краткое название метода, основанного на этой таблице, и описание. Напр., для задания таблицы Бутчера стандартного метода Рунге–Кутты

$$\begin{array}{c|cccc} \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & & \frac{1}{2} & & \\ 1 & & & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

мы пишем:

```
rk4=Butcher_tableau(4,[[[0,0,0,0],
[1/2,0,0,0], [0,1/2,0,0],[0,0,1,0]],
 [1/6,1/3,1/3,1/6]], 'rk4',
 'Standard rk method')
```

Функции, определенные в классе `Butcher_tableau`, носят вспомогательный характер и используются в нашей реализации метода Рунге–Кутты. Для пользователя может быть полезна лишь одна из них – `latex`, которая возвращает таблицу Бутчера в нотации LaTeX.

В наш пакет интегрирована функция `butcher_eqs(p,s)`, написанная Юй Ин [20], эта функция возвращает систему уравнений на элементы таблицы Бутчера по заданным порядку p и числу стадий s и опции implicit, которая может принимать два значения True/False.

Пример 3. Двустадийная явная таблица Бутчера имеет вид

$$\begin{array}{c|cc} c_0 & & \\ c_1 & a_{10} & \\ \hline & b_0 & b_1 \end{array}$$

Эта таблица задает разностную схему порядка $p = 2$, если ее коэффициенты удовлетворяют двум уравнениям, которые возвращает функция `butcher_eqs`:

```
butcher_eqs(2,2, implicit=False)
```

$$-b_0 - b_1 + 1 = 0, \quad -2a_{10}b_1 + 1 = 0. \qquad (5.1)$$

Поэтому имеется однопараметрическое семейство явных методов Рунге–Кутты второго порядка с двумя стадиями.

В 2000-х годах появились первые программы для численного, а позже и символьного решения систем уравнений для определения элементов таблицы Бутчера [21–25]. Большая коллекция таблиц Бутчера до 12 порядка аппроксимации была собрана П. Стоуном [26], который производил вычисления в системе Maple. Вычислять с нуля каждый раз таблицы Бутчера большого порядка весьма неразумно, поэтому в fdm имеется коллекция таблиц Бутчера, реализованная пока в виде списка `butchers_list`, который задан в файле `butchers_list.sage`. Эта коллекция в существенном основана на коллекции Стоуна.

Замечание. При перенесении таблиц Бутчера высокого порядка возникают неизбежные опечатки, таблицы, используемые в нашей системе, проверены по методу Ричардсона на тестовых задачах. На данный момент, одна из таблиц, взятых на сайте Стоуна, отмечена в `butchers_list.sage` как сомнительная.

5.2. Явный метод Рунге–Кутты

Функция `erk` реализует явный метод Рунге–Кутты, при этом пользователь может выбрать любую матрицу Бутчера из базы или указать свою. По умолчанию используется матрица rk4. Аргументами этой функции служит начальная задача. Предусмотрено еще две опции: N – число отрезков, на которое делится производный отрезок $[0, T]$ и `tableau` – таблица Бутчера.

По методу Ричардсона мы можем проверить правильность определения порядка аппроксимации, что весьма существенно для корректного заполнения базы таблиц Бутчера.

Пример 4. В качестве первого примера рассмотрим начальную задачу (4) на отрезке $0 < t < 1$, точное решение которой известно:

```
x_exact=-t^2 - 2*t + 2*e^t - 2
```

Сравним его с двумя решениями, найденными при помощи двух различных матриц Бутчера. В качестве первой таблицы возьмем стандартную таблицу метода rk4.

```
P=erk(pr2, N=10)
Q=erk(pr2, N=20)
richardson(P,Q,x,0.8)
```

$$[0.211081834707056, -1.86964065451711\times10^{-8}]$$

Хорошо видно, что оценка ошибки вычисления $x(0.8)$ по Ричардсону совпадает по порядку величины с фактической ошибкой:

```
0.211081834707056-x_exact.subs(t=0.8)
```

$$-2.22778795411216\times10^{-8}$$

В качестве второй таблицы возьмем таблицу 8-го порядка, приведенную у Стоуна за no. 8b-1.

```
P=erk(pr2, N=10, tableau=B)
Q=erk(pr2, N=20, tableau=B)
richardson(P,Q,x,0.8)
```

$$\left[0.211081856984935, 2.39459868056406\times10^{-17}\right]$$

Оценка ошибки по Ричардсону несколько занижена по сравнению с фактической ошибкой:

```
0.211081856984935-x_exact.subs(t=0.8)
```

$$-5.27355936696949\times10^{-16}$$

Диаграмма Ричардсона (рис. 3) вполне проясняет это затруднение: мы практически сразу выходим на ошибку округления, которая имеет порядок 10^{-16}.

Наша реализация метода Рунге–Кутты не зависит от поля $\mathbb{K}$, это позволяет без труда менять число бит, отведенных на одно число и сдвинуть ошибку округления. Разумеется, и элементы таблицы Бутчера, и начальные условия переводятся в это поле. Таким образом, ошибки округления согласованы друг с другом.

Пример 5. Если в методе Рунге–Кутты из примера 4 отвести 300 бит на число, то получится диаграмма Ричардсона (рис. 4), имеющая правильный наклон 8. Для ее построения в функции `erk` была добавлена опция `field=RealField(300)`.

Отмеченный выше вопрос о том, как распорядиться оставшимся произволом в выборе элементов таблицы Бутчера, остается до сих пор нерешенным [19]. В нашем пакете можно попробовать экзотические варианты, напр., взять комплексные значения элементов таблицы, добавив в verb|erk| опцию `field=CC`.

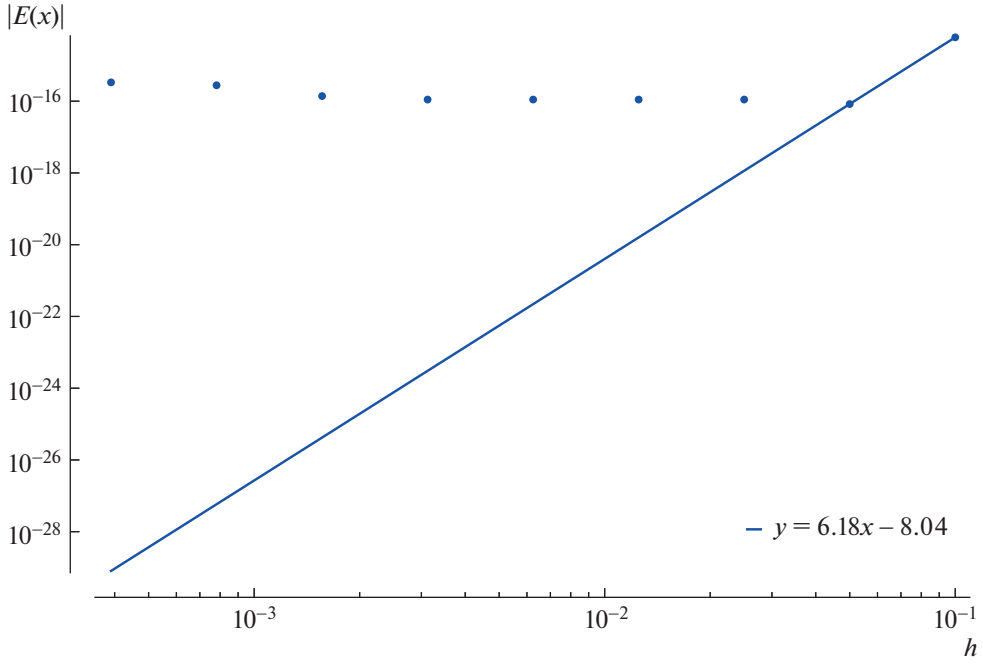

Рис. 3. Диаграмма Ричардсона для значения x_1 при $t = 0.8$ для примера 4.

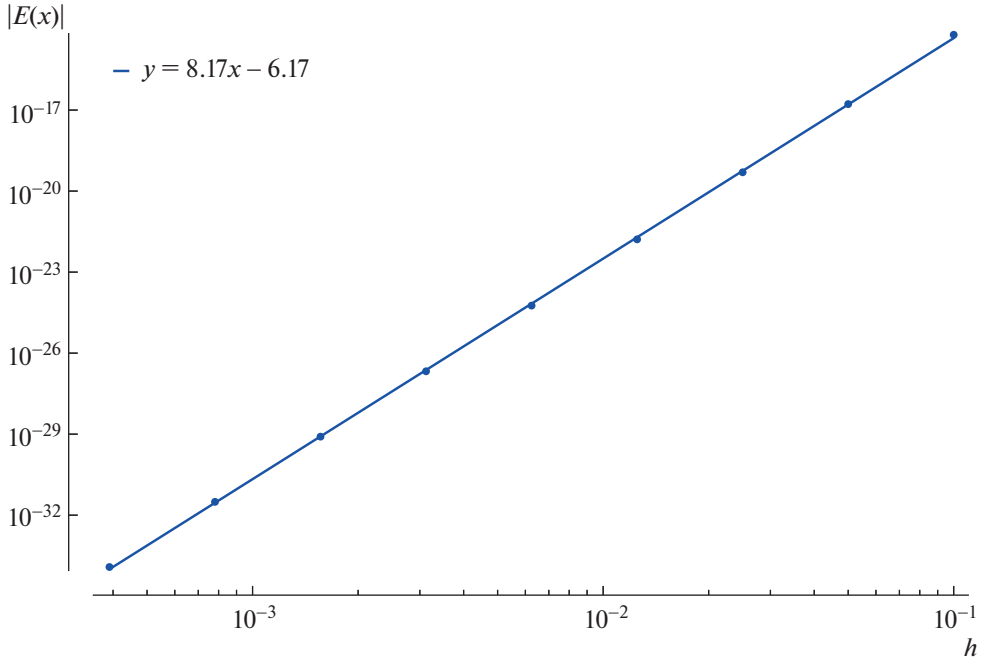

Рис. 4. Диаграмма Ричардсона для значения x_1 при $t = 0.8$ для примера 5.

Пример 6. Таблица

$$\begin{array}{c|cc} i & i & \\ \hline & \frac{1}{2}i+1 & -\frac{1}{2}i \end{array}$$

удовлетворяет системе (5.1) и поэтому задает разностную схему 2-го порядка и в этом смысле ничем не лучше классической схемы средней точки с таблицей

$$\begin{array}{c|cc} 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Для линейных задач результаты расчетов с вещественной и мнимой таблицами совпадают. Разли-

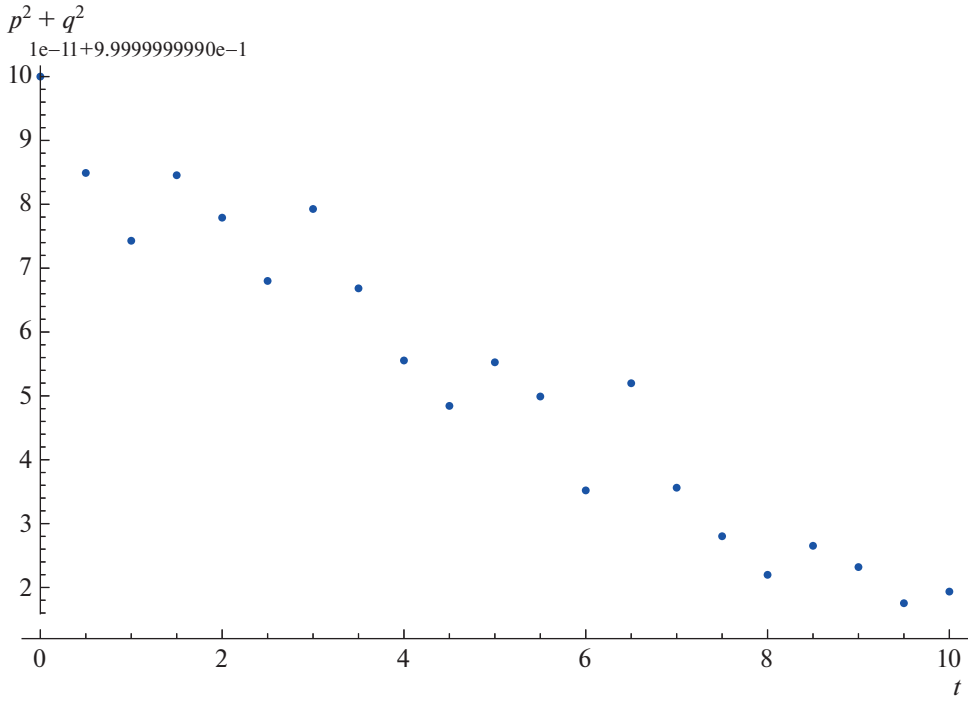

Рис. 5. График зависимости интеграла движения $p^2 + q^2$ осциллятора Якоби от t, вычисленный по схеме средней точки, $N = 20$.

чие проявляется только в нелинейных задачах. Возьмем для примера осциллятор Якоби [27] на отрезке $0 < t < 10$ и сравним значение p при $t = 10$:

```
var("p,q,r,t")
k=1/2
pr3=Initial_problem([p,q,r], [q*r, -p*r, -k^2*p*q], [0,1,1], 10)
  erk(pr3, N = 200, tableau=B, field=CC) .list()[-1] [1]
```

$$0.110646443410733$$

```
erk(pr3, N = 200, tableau=Bi, field=CC) .list()[-1] [1]
```

$$0.111457845209401 - 0.000811190390574542 \cdot i$$

5.3. Неявный метод Рунге–Кутты

Функция `irk` реализует неявный метод Рунге–Кутты, синтаксис тот же, что и у `erk`. Здесь на каждом шаге решается система нелинейных уравнений методом простых итераций. Итерации происходят до тех пор, пока норма разности двух последовательных итераций не оказывается меньше числа $\epsilon > 0$. По умолчанию используется таблица Бутчера

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$

задающая схему средней точки, и $\epsilon = 10^{-10}$.

Пример 7. Рассмотрим вновь осциллятор Якоби (задача `pr3` из примера 6). Вычислим приближенное решение и построим график зависимости интеграла движения $p^2 + q^2$ от t:

```
P=irk(pr3, N = 20)
P.plot(t,p^2+q^2)
```

В результате получится график, приведенный на рис. 5. Хорошо видно, что рассматриваемый квадратичный интеграл сохраняется с точностью до 10^{-10}. В силу теоремы Купера [7] этот интеграл сохраняется на решениях, полученных по симплектическим схемам Рунге–Кутты, точно. Ошибка округления и параметр ϵ, входящий в критерий остановки итерационного процесса, делают изменение интеграла движения малым, но все же отличным от нуля.

Опция `field`, как и раньше, позволяет менять число бит, отводимых на десятичную дробь, однако теперь ее нужно согласовывать со значением ϵ, которое определяется опцией `eps`, и максимальным числом итераций, которое определяется опцией M.

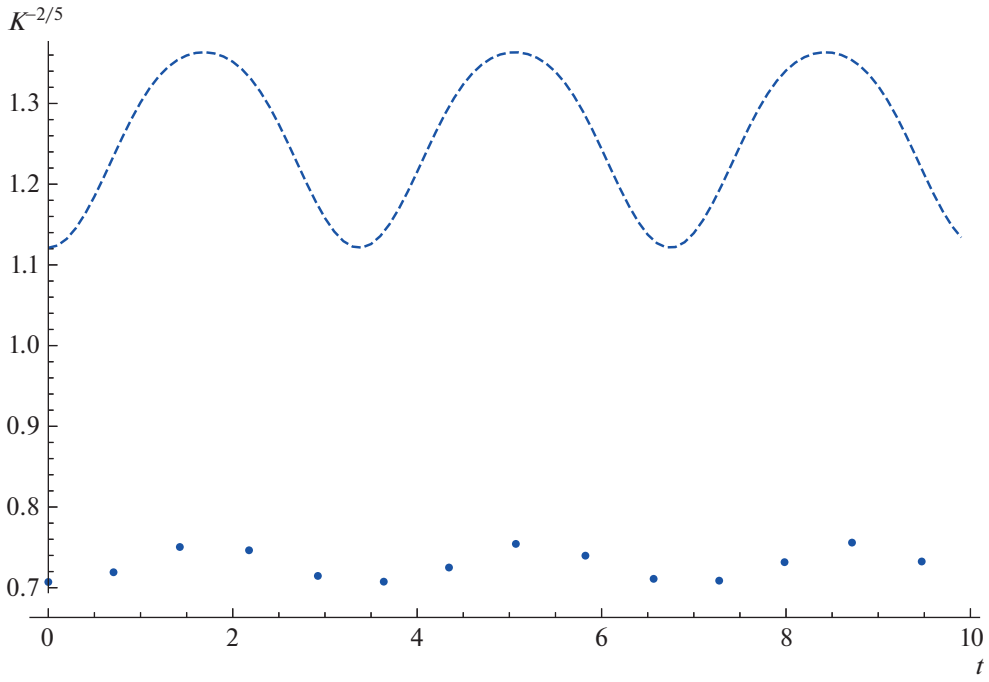

Рис. 6. График изменения шага, использованного для колебаний осциллятора Якоби от t, вычисленный по схеме средней точки, $h = 1$, (сплошная), и кривизна K интегральной кривой (пунктир).

Сходимость метода последовательных итераций накладывает некоторые условия на величину шага. Поэтому в нашей системе реализована версия неявного метода Рунге–Кутты, которая подстраивает шаг под эти условия. Соответствующая функция названа `irk_adaptive`. Подстройка шага мешает заранее узнать число шагов, потребных для достижения конченого значения $t = T$, поэтому вместо опции N, указывающей число отрезков, на которые делится сегмент $[0,T]$, используется опция h, характеризующая длину переменного шага.

Пример 8. Построим график зависимости $p^2 + q^2$ от t по адаптивной схеме:

```
Q=irk_adaptive(pr3, h=1)
Q.plot(t,p^2+q^2)
```

При этом можно посмотреть и на изменение шага

```
Q.plot_dt()
```

График этой зависимости представлен на рис. 6. В [14] рекомендовалось брать шаг адаптивных методов, исходя из формулы

$$\Delta t = \frac{L}{K^{\frac{2}{5}}},$$

где K – кривизна интегральной кривой. Нетрудно видеть, что в данном случае зависимость шага от t повторяет в существенном ход зависимости $K^{-\frac{2}{5}}$ от t.

6. ЗАКЛЮЧЕНИЕ

В настоящей статье представлен оригинальный пакет `fdm`, встраиваемый в систему компьютерной алгебры Sage, при создании которого мы исходили из следующих общих принципов.

• Реализации методов не зависят от поля $(\mathbb{R},\mathbb{C})$ и тем более от числа бит, отведенных на одно число.

• Действия, которые могут быть выполнены аналитически, выполняются аналитически. Начальные задачи рассматриваются как элементы нового класса, в определении которого предусмотрены инструменты для проведения символьных вычислений.

• Численные решения рассматриваются как элементы нового класса, в определении которого предусмотрены инструменты для интерполяции и визуализации.

В тексте рассмотрены основные инструменты для работы с начальными задачами и численными решениями, которые оторваны от конкретных численных методов, и реализация метода Рунге–Кутты. Помимо рассмотренного выше метода Рунге–Кутты, пакет `fdm` содержит реализацию метода Адамса [28, п. 82], примечательного возможностью проведения части вычислений в символьном виде, и метода, основанного на обратимых схемах [10].

Мы надеемся, что созданный инструмент будет удобен для численно-аналитического иссле-

дования разностных схем и со временем станет глубоко интегрирован с алгебраическим инструментарием Sage.

Благодарности. Работа выполнена при финансовой поддержке РНФ (проект № 20-11-20257).

7. ИСТОЧНИК ФИНАНСИРОВАНИЯ

Работа выполнена при финансовой поддержке РНФ (проект № 20-11-20257).

СПИСОК ЛИТЕРАТУРЫ

1. *Runge C., König H.* Vorlesungen über numerisches Rechnen. Springer-Verlag, 2013.
2. SciPy documentation, 2022. Access mode: https://docs.scipy.org.
3. *Ketcheson D.I., bin Waheed U.* A comparison of high order explicit Runge-Kutta, extrapolation, and deferred correction methods in serial and parallel // CAMCoS. 2014. V. 9. № 2. P. 175–200.
4. *Геворкян М.Н.* Конкретные реализации симплектических численных методов // Вестник РУДН. Серия: Математика. Информатика. Физика. 2013. № 3. P. 77–89.
5. *Castillo J.E., Miranda G.F.* Mimetic discretization methods. Chapman and Hall/CRC, 2013.
6. *Da Veiga L.B., Lipnikov K., Manzini G.* The mimetic finite difference method for elliptic problems. Springer, 2014. V. 11.
7. *Hairer E., Wanner G., Lubich Ch.* Geometric Numerical Integration. Structure-Preserving Algorithms for Ordinary Differential Equations. Berlin Heidelberg New York : Springer, 2000.
8. On the Quadratization of the Integrals for the Many-Body Problem / Yu Ying, Ali Baddour, Vladimir P. Gerdt et al. // Mathematics. 2021. V. 9. № 24.
9. *Baddour A., Malykh M.* On Difference Schemes for the Many-Body Problem Preserving All Algebraic Integrals // Phys. Part. Nuclei Lett. 2022. V. 19. P. 77–80.
10. *Baddour A., Malykh M., Sevastianov L.* On Periodic Approximate Solutions of Dynamical Systems with Quadratic Right-Hand Side // J. Math. Sci. 2022. V. 261. P. 698–708.
11. *Stein W.A.* Sage Mathematics Software (Version 6.7). The Sage Development Team, 2015. Access mode: http:// www.sagemath.org.
12. *Малых М.Д., Юй Ин* Методика отыскания алгебраических интегралов дифференциальных уравнений первого порядка // Вестник Российского университета дружбы народов. Серия: Математика, информатика, физика. 2018. Т. 26. № 3. С. 285–291.
13. Вычисления на квазиравномерных сетках / Н.Н. Калиткин, А.Б. Альшин, Е.А. Альшина, Б.В. Рогов. Москва : ФИЗМАТЛИТ, 2005. ISBN: 5-9221-0565-5.
14. *Belov A.A., Kalitkin N.N., Poshivaylo I.P.* Geometrically adaptive grids for stiff Cauchy problems // Doklady Mathematics. 2016. V. 93. № 1. P. 112–116.
15. *Belov A.A., Kalitkin N.N.* Nonlinearity Problem in the Numerical Solution of Superstiff Cauchy Problems // Mathematical Models and Computer Simulations. 2016. V. 8. № 6. P. 638–650.
16. Explicit methods for integrating stiff Cauchy problems / A.A. Belov, N.N. Kalitkin, P.E. Bulatov, E.K. Zholkovskii // Doklady Mathematics. 2019. V. 99. № 2. P. 230–234.
17. *Баддур Али, Малых М.Д.* Richardson–Kalitkin method in abstract description // Discrete and Continuous Models and Applied Computational Science. 2021. V. 29. № 3. P. 271–284.
18. Numerical determination of the singularity orderof a system of differential equations / Али Баддур, М.Д. Малых, А.А. Панин, Л.А. Севастьянов // Discrete and Continuous Models and Applied Computational Science. 2020. V. 28. № 1. P. 17–34.
19. *Hairer E., Wanner G., Norsett S.P.* Solving Ordinary Differential Equations I. 3 edition. Springer, 2008.
20. *Yu Ying.* The symbolic problems associated with Runge-Kutta methods and their solving in Sage // Discrete and Continuous Models and Applied Computational Science. 2019. V. 27. № 1. P. 33–41.
21. *Хашин С.И.* Численное решение уравнений Бутчера // Вестник ИвГУ. 2000. № 3. P. 155–164.
22. *Хаммуд Г.М., Хашин С.И.* Шестимерное семейство 6-шаговых методов Рунге–Кутта порядка 5 // Науч. тр. ИвГУ. Математика. 2001. № 4. P. 114–122.
23. *Хашин С.И.* Альтернативная форма уравнений Бутчера // Вестник ИвГУ. 2007. № 3. P. 94–103.
24. *Хашин С.И.* A Symbolic-Numeric Approach to the Solution of the Butcher Equations // Canadian Applied Mathematics Quarterly. 2009. V. 17. № 3. P. 555–569.
25. *Хашин С.И.* Три упрощающих предположения для методов Рунге–Кутта // Вестник ИвГУ. 2012. № 2. С. 142–150.
26. *Stone P.* Maple worksheets on the derivation of Runge-Kutta schemes, 2021. Access mode: http://www.peterstone.name/Maplepgs/ RKcoeff.html.
27. *Сикорский Ю.С.* Элементы теории эллиптических функций с приложениями к механике. М.-Л. : ОНТИ, 1936.
28. *Скарборо Дж.* Численные методы математического анализа. М.-Л.: ГТТИ, 1934.

On Implementation of Numerical Methods for Solving Ordinary Differential Equations in Computer Algebra Systems

© 2023 г. A. Baddour[a], M. M. Gambaryan[a], L. Gonzalez[a], and M. D. Malykh[a,b,#]

[a]*Peoples' Friendship University of Russia, ul. Miklukho-Maklaya 6, Moscow, Russia*

[b]*Joint Institute for Nuclear Research, Dubna, Moscow oblast, 141980 Russia*

[#]*e-mail: malykh_md@pfur.ru*

This paper presents an original package for investigating numerical solutions of ordinary differential equations, which is built in the Sage computer algebra system. This project is focused on a closer integration of numerical and symbolic methods while primarily aiming to create a convenient tool for working with numerical solutions in Sage. The package defines two new classes: initial problems and approximate solutions. The first class defines tools for symbolic computations related to initial problems, while the second class defines tools for interpolating values of symbolic expressions on an approximate solution and estimating the error with the use of the Richardson method. An implementation of the Runge–Kutta method is briefly described, with its main feature being the possibility of working with arbitrary Butcher tableaux and arbitrary numeric fields.