ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ОБЩЕЙ ФИЗИКИ

БАКАЛАВРСКАЯ РАБОТА

«КОНТРОЛЬ ТОЧНОСТИ ПРИ ЧИСЛЕННОМ ИНТЕГРИРОВАНИИ ЖЕСТКИХ СИСТЕМ»

Выполнил студент

402 группы

Вергазов Артём Сергеевич

подпись студента

Научный руководитель:

к.ф.-м.н. Белов А.А.

подпись научного руководителя

Допущена к защите

Зав. кафедрой д.ф.-м.н., проф. Нефедов Н.Н.

подпись зав. кафедрой

Москва

2021

# 1. Методы численного интегрирования жестких систем

1.1. Длина дуги. Численное интегрирование задачи Коши для жестких систем обыкновенных дифференциальных уравнений (ОДУ) является одной из очень трудных задач вычислительной математики. Формально эта задача имеет следующий вид:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Здесь *t*– скаляр,  и  –векторные функции, а *M*– размерность системы.

Задачу традиционно считают жесткой, если  хотя бы в части отрезка . Решению жестких задач посвящена обширная литература, наиболее подробный обзор которой дан в монографии [1-2].

Различают два типа прикладных задач. Первый тип – это так называемые большие задачи, или задачи со многими процессами. Они описываются системами уравнений в частных производных, к которым подключена одна или несколько систем ОДУ. Примером могут служить задачи горения и взрыва. В них процесс горения, то есть реакции химических веществ, описывается системой уравнений химической кинетики; это система ОДУ. Выделяющееся тепло приводит к движению вещества. Это движение описывается уравнениями газодинамики, то есть системой уравнений в частных производных.

Расчет уравнений в частных производных гораздо более трудоемок, чем решение ОДУ. Поэтому в больших задачах шаг по времени *τ* определяется требованиями методов решения уравнений в частных производных. Этот же шаг *τ* вычислитель вынужден использовать для решения сопутствующей системы ОДУ. В этом случае естественным аргументом для решения задачи (1) является время *t*.

Второй класс задач содержит только систему ОДУ. Для этого класса задач оказывается выгоднее выбрать другой аргумент – длину дуги интегральной кривой в многомерном пространстве. Выполним переход к этому аргументу в два этапа.

Во‑первых, добавим новую неизвестную функцию , для нее имеем . Тогда система (1) преобразуется к виду

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

Общее число неизвестных теперь равно . Однако теперь правые части формально не содержат аргумента *t* в правых, то есть система становится автономной. Такую автономизацию называют тривиальной.

Во‑вторых, определим элемент длины дуги интегральной кривой соотношением

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

Тогда система (2) преобразуется к виду

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

При аргументе *l* сумма квадратов правых частей (4) равна 1, то есть норма правой части никогда не бывает большой. Это облегчает численное интегрирование системы. Такой прием полезен даже для нежестких ОДУ, а для жестких он кардинально облегчает решение задачи. Переход к длине дуги и различные преимущества этого метода подробно описаны в монографии [3]. Заметим, что правые части (4) не содержат аргумента *l*, то есть эта система является автономной.

Переход к длине дуги практически не увеличивает трудоемкость одного шага численных расчетов. Обычно основное время расчетов уходит на вычисление правых частей , которые могут быть достаточно сложными функциями своих аргументов. Переход от (1) к (4) включает лишь несколько дополнительных арифметических операций, трудоемкость которых невелика. Размерность системы увеличивается на 1, что так же мало существенно для прикладных задач, где *M* обычно довольно велико.

1.2. Выбор шага**.**Расчет с постоянным шагом по времени *τ* или по длине дуги *h* обычно невыгоден. Шаг целесообразно уменьшать там, где решение быстро меняется, то есть правые части ОДУ велики. На участках слабого изменения решения шаг можно увеличивать. В [2] подробно описаны алгоритмы автоматического выбора шага, принятые в мировой литературе. Традиционно используют два основных метода выбора шага. В первом методе каждый шаг выполняют по некоторой схеме ‑го порядка точности, в которую вложена схема *p*‑го порядка точности. Результат вложенной схемы берут в качестве ответа. По разности результатов двух схем выбирают величину следующего шага. Во втором методе шаг *τ* рассчитывают повторно, разбив его на два шага величиной . По разности этих расчетов вычисляют новый шаг.

На основе этих методов написано много пакетов программ. Большинство из них хорошо работают на нежестких задачах. Однако проверка этих пакетов на тестах с известными точными решениями показывает, что реальная точность расчетов лишь по порядку величины близка к запросу пользователя. Она обычно оказывается в несколько раз хуже или лучше. Поэтому в расчетах прикладных задач, где ответ неизвестен, пользователь не может быть вполне уверен в достижении заданной им точности.

Намного хуже ситуация для случая жестких задач. Для них расчеты на тестах показали [4‑6], что реальная точность может быть до  раз хуже заявленной. Это относится даже к таким тщательно выверенным программам, как пакеты Гира или программы Дормана-Принса dopri5. Кроме того, на жестких задачах возможны “срывы” шага [2]: иногда на участках слабо меняющегося решения программа без видимых причин уменьшает шаг в 100‑1000 раз. Затем шаг постепенно увеличивается, но снова срывается; это может повторяться неоднократно.

В [4] был предложен принципиально другой алгоритм автоматического выбора шага. Он основан на использовании длины дуги и кривизны *κ* интегральной кривой в многомерном пространстве. Интуитивно понятно, что чем больше кривизна *κ*, тем меньше должен быть шаг *h*. Но каким должен быть алгоритм, связывающий эти две величины?

1.3. Оптимальный шаг**.** Напомним определение кривизны кривой***u****(t)* в ‑мерном пространстве с координатами . Касательная к этой кривой определяется через производную . Деля на длину этого вектора, получим единичный вектор направления касательной:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Вектор кривизны и кривизна определяются через производную вектора направления касательной по длине дуги:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

Таким образом, кривизна является вектором даже в случае плоской кривой, то есть одного ОДУ; это радиус-вектор окружности, имеющей касание второго порядка с кривой ***u***(*t*). Наряду с этим говорят о скалярной кривизне *κ*, которая равна величине радиуса этой окружности.

В ранних работах [4,5] с помощью многих численных экспериментов была подобрана неплохая эвристическая закономерность: . Однако в пакетах прикладных задач [7] был использован постоянный шаг по длине дуги и несколько семейств, в которых конкретные схемы имели порядок точности от 1 до 4. Эти пакеты позволили провести расчеты ряда прикладных задач, например – образование окислов фосфора и серы при горении различных топлив.

Задача теоретического нахождения зависимости  крайне сложна. Из общих соображений понятно, что результат должен зависеть от того, какая именно разностная схема используется для интегрирования системы ОДУ. Однако удалось найти такой случай [8], для которого теоретически обосновывается формула выбора оптимального шага. Пусть схема интегрирования имеет точность ; это может быть явная или неявная схема Эйлера или явно-неявная L1-устойчивая схема Розенброка. Тогда

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

Качественный вид этой формулы совпадает с ранее найденным эвристическим видом, а оптимальный показатель степени лишь слабо отличается от эвристического.

Как часто оказывается, чистo теоретическая формула (7) для своего практического применения требует “кухонных” поправок.

Во‑первых, ясно, что при  она дает . Формально это правильно: если на некотором участке кривой , этот участок есть прямая, то есть ***u***(*t*) является линейной функцией. А для линейной функции любая численная схема дает точный ответ при сколь угодно большом шаге.

На практике  лишь в отдельных точках кривой (например, в точках перегиба), а попадание узла расчетной сетки именно в эту точку имеет нулевую вероятность. Однако возможно попадание счетного узла в малую окрестность точки перегиба и получение неприемлемо большого шага *h*. Надо ввести такую поправку, чтобы в любом случае число интервалов было не меньше некоторого разумного .

Во‑вторых, надо дать разумное определение константы в (7). В нее должна быть включена некоторая интегральная нормировка по длине дуги, обеспечивающая желательное количество интервалов сетки .

Исходя из этого, в [8] был предложен следующий алгоритм. Пусть требуется решить задачу (4) на отрезке  с априорно заданными , . Несколько видоизменим формулу (7) с учетом сделанных замечаний:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

В точках с очень малой кривизной  она дает . В точках с очень большой кривизной формула (8) переходит в (7); при этом константа такова, что . При любых  не может получиться  или . Это свидетельствует о разумности формулы (8).

1.4. Двухэтапная стратегия**.** Численный расчет может называться хорошим, если он гарантирует пользователю требуемую точность. Для нежестких задач, когда в качестве аргумента выбирают время *t*, этого нетрудно добиться. Расчет до заданного момента *T* проводят на равномерной сетке сначала с некоторым умеренным числом шагов *N*. Затем увеличивают *N* в 2 раза, одновременно уменьшая  вдвое, и проводят новый расчет. При этом узлы  первой сетки точно совпадают с четными узлами удвоенной сетки. В совпадающих узлах находят разность двух сеточных решений на соседних сетках и вычисляют норму разности (обычно это нормы L2 или C). В этом случае для оценки погрешности применим метод Ричардсона [1, 9‑12]. Если заданная точность не будет достигнута, то процедуру удвоения сетки повторяют. Этот метод дает асимптотически точную оценку погрешности.

Однако для жестких задач равномерная сетка непригодна. Чтобы шаг равномерной сетки стал достаточно малым на участках быстрого изменения решения, нужны огромные числа узлов *N*. Даже переход к аргументу *l* не позволяет использовать равномерные сетки.

Существуют так называемые квазиравномерные сетки [11,12]. Для них разработана процедура удвоения *N*, позволяющая пользоваться методом Ричардсона. Но для построения такой сетки нужно заранее знать поведение всей функции и строить густую сетку там, где функция быстро меняется. Однако до начала решения задачи мы не знаем, где расположены такие участки. Поэтому в [4,5] была предложена двухэтапная процедура сгущения сеток. С учетом новейших работ она формулируется следующим образом.

***Первый этап.*** Для начала расчета выбираем некоторые разумные значения  и , входящие в формулу шага (8). В эту формулу входят также *L* и интеграл. До начала расчетов они неизвестны, поэтому задаются их правдоподобные оценки. После этого проводят расчет с выбором шага согласно (8) и продолжают его до тех пор, пока  не превысит значение *T*. Полученный номер узла есть *N* для построенной сетки . При этом автоматически вычисляется значение полной длины дуги для данной сетки . Попутно в ходе расчета вычисляют интеграл в (8) по какой-нибудь квадратурной формуле; допустима даже формула левых прямоугольников.

Найденное в этом расчете значение *N* может не совпадать с исходным значением . Разумеется, вычисленные значения *L* и интеграла в (8) не будут совпадать с теми величинами, которые использовались для начала расчета.

Затем удвоим значения ,  и построим сгущенную примерно вдвое сетку. В этом расчете будем использовать значения интеграла и *L*, найденные на предыдущей сетке. Практика показывает, что при этом *N* окажется примерно вдвое большим, чем на предыдущей сетке. Однако новые значения интеграла и *L* могут сильно отличаться от предыдущих значений, поскольку на первой сетке они были выбраны произвольно. Очевидно, что новая сетка будет примерно вдвое подробнее предыдущей. Однако ее четные узлы не будут совпадать с узлами предыдущей сетки. Поэтому оценка точности по правилу Ричардсона невозможна.

Снова удвоим ,  и построим третью сетку. На третьей сетке значение *N* уже почти удваивается по сравнению со второй сеткой, а значения интеграла и *L* будут близки к соответствующим величинам второй сетки. Значения четных узлов третьей сетки так же будут существенно ближе к узлам второй сетки. Однако правило Ричардсона применять по-прежнему нельзя.

Эту процедуру удвоения сетки повторяем до тех пор, пока четные узлы новой сетки не станут достаточно близкими к узлам предыдущей сетки. Различные критерии близости сеток сравнивались в [13]; наилучшим был признан следующий критерий. Обозначим шаги исходной сетки через . Шаги удвоенной сетки обозначим через ; на практике , хотя точного равенства может не быть. Шагу  в удвоенной сетке соответствует сумма шагов . Будем считать сетки близкими, если

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

разумную величину *η* надо подбирать на основе практических расчетов. При этом значения интеграла и *L* на этих сетках будут совпадать с приемлемой точностью. При выполнении критерия (9) первый этап заканчивается.

***Второй этап.*** Будем считать, что последняя сетка первого этапа уже достаточно хорошо адаптирована к участкам быстрого изменения решения. Поэтому дальнейшие сгущения сеток будем производить так, чтобы можно было применять метод Ричардсона для оценки погрешности. Для этого надо проводить сгущения так, чтобы последовательность удвоенных сеток была квазиравномерной. Простейший способ такого сгущения имеет следующий вид.

Пусть исходная сетка имеет узлы  и шаги . Пусть интервал сетки  является внутренним: . Тогда узлы этого интервала являются четными узлами новой сетки  и , а шаг  делится на два шага новой сетки  и  по следующим формулам:

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

Для левого граничного интервала шаг  делится по формулам

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

Правый граничный шаг  делится следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

При таких формулах сгущения сетки число интервалов точно удваивается, а полная длина дуги *L* не меняется.

Можно доказать, что описанный способ удвоения сетки порождает последовательность квазиравномерных сеток. На такой последовательности правомерно однократное применение метода Ричардсона: по каждой паре соседних сеток можно находить разности функций в совпадающих узлах , вычислять нормы этих разностей и определять асимптотически точное значение погрешности по известному порядку точности формулы интегрирования.

Однако этот способ имеет одно ограничение: нельзя пользоваться рекуррентным методом Ричардсона, то есть повышать порядок точности, используя 3 и более соседние сетки.

# 2. Апробация на тестах

2.1. Требования к тестам**.** Общепринятым способом апробации различных численных методов является расчет тестовых задач. Хороший тест должен удовлетворять нескольким требованиям. Во-первых, он должен содержать типичные трудности, на преодоление которых ориентирован исследуемый численный метод. Во-вторых, у него должно существовать легко реализуемое точное решение. Обычно под этим подразумевают, что точное решение при любых значениях аргументов легко вычисляется с любой требуемой точностью. Реально для этого нужно, чтобы решение достаточно просто выражалось через элементарные функции; запись через специальные функции нежелательна, так как их реализация с любым требуемым числом значащих цифр далеко не всегда доступна. В‑третьих, желательно, чтобы тест содержал один или несколько параметров, которыми можно регулировать его жесткость и другие качественные свойства.

При наличии такого теста апробация метода несложна. Проводят расчет исследуемым методом, причем этот метод выбирает сетку по своим естественным правилам. В узлах этой сетки вычисляют точное решение и непосредственно находят точное значение погрешности, то есть разности между сеточным и точным решением. При этом можно вычислить любую норму погрешности. Это будет достоверная оценка.

Задачи, в которых решение известно только в отдельных реперных точках, непригодны в качестве теста. Во-первых, они не позволяют найти нормы ошибки. Во-вторых, для сравнения с ними приходится проводить решение на таких сетках, некоторые узлы которых попадают в реперные точки. Для большинства алгоритмов такое построение сеток неестественно.

Для обыкновенных дифференциальных уравнений с аргументом *t* сравнительно легко конструируются различные тесты, в том числе для случаев высокой и сверхвысокой жесткости. Однако при переходе к аргументу *l* проблема существенно усложняется. Недостаточно, чтобы в элементарных функциях выражалось . Нужно наличие решения в элементарных функциях так же для . При этом все формулы не только прямого вычисления решения, но и вычисления обратных функций так же должны выражаться в элементарных функциях, причем достаточно просто. В противном случае, вычисление погрешностей при переходе от одного аргумента к другому будет практически невозможным.

До сих пор такие тесты не были построены. Нам удалось сконструировать два требуемых теста.

2.2. Гиперболический тест**.** Рассмотрим тестовое уравнение, содержащее один свободный параметр

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

Точное решение этой задачи имеет следующий вид:

|  |  |  |
| --- | --- | --- |
|  |  | (14) |

Подставляя выражение (14) в формулу (2) и интегрируя, получим выражение для длины дуги:

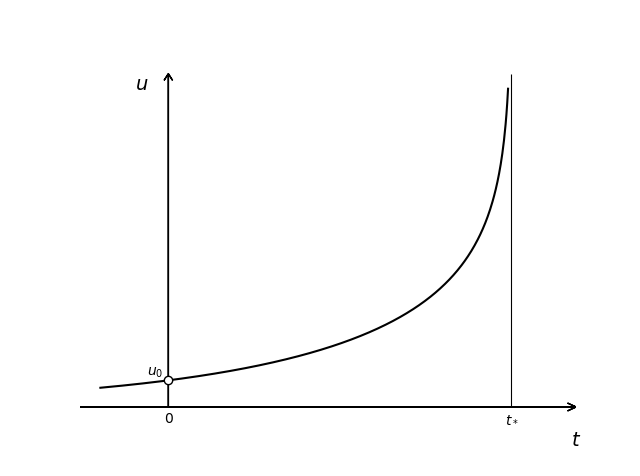
|  |  |  |
| --- | --- | --- |
|  |  | (15) |

Далее увидим, что жесткость задачи (13) быстро увеличивается с ростом *λ*. В задачу (13) нетрудно ввести еще один параметр — множитель *α* в правой части. Однако это не представляет интереса, так как заменой  и  задача сводится к форме (13).

Исследуем качественный вид решения (14). Функция *u*(*t*)положительна и монотонно возрастает. Она определена на конечном отрезке, поскольку

|  |  |  |
| --- | --- | --- |
|  | при | (16) |

Длина дуги при этом так же неограниченно возрастает:  при . Качественный вид решения *u*(*t*)изображен на рис. 1. Решение имеет продолжение влево на полуось .



*Рис. 1.* Точное решение (14); ○ – начальная точка.

Перейдем в уравнении (13) к переменной *l*. Учтем, что . Тогда форма записи (4) примет следующий вид:

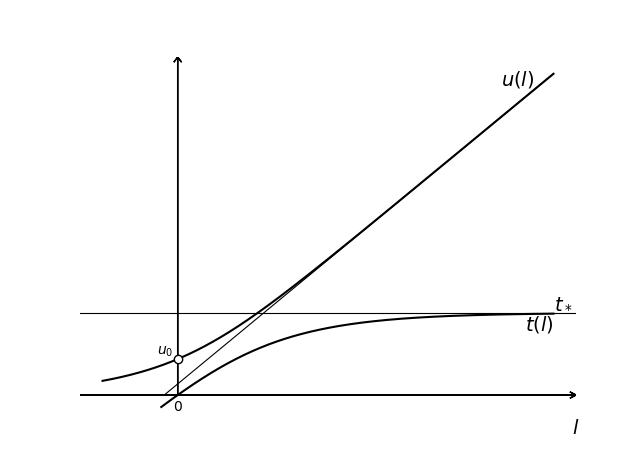
|  |  |  |
| --- | --- | --- |
|  |  | (17) |

Для первого уравнения системы (17) нетрудно написать точное решение:

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

Функция  так же явно выражается через длину дуги:

|  |  |  |
| --- | --- | --- |
|  |  | (19) |



*Рис. 2.* Точное решение (18‑19); ○ – начальная точка для *u*.

Качественный вид функции *u*(*l*)изображен на рис. 2. Интересно сопоставить эту зависимость с рис. 1. На рис. 1 решение резко возрастает при . Тем самым, численный расчет на этом участке возможен только очень малым шагом *τ*. Однако на рис. 2 этому участку соответствует правая часть графика, где наклон . Тем самым, при аргументе *l* эту область решения можно проходить большим шагом *h*.

Приведем еще некоторые явные формулы, из которых особенно важно выражение кривизны:

|  |  |  |
| --- | --- | --- |
|  |  | (20) |
|  |  | (21) |
|  |  | (22) |

Кривизна немонотонно меняется вдоль интегральной кривой на рис. 1 или рис. 2. В левой части обоих графиков она возрастает, затем достигает максимума при условиях

|  |  |  |
| --- | --- | --- |
|  |  | (23) |

и затем монотонно убывает до 0 в правой части графиков. Хорошо видна связь *λ* с максимальной кривизной интегральной кривой и жесткостью задачи.

Данный тест является уникальным. Во-первых, в нем все значимые величины выражаются друг через друга с помощью несложных комбинаций элементарных функций. Во-вторых, выбирая большие значения *λ*, можно сделать задачу сколь угодно жесткой.

***Трудность задачи.*** Трудность определяется тем числом шагов *N*, которое необходимо сделать при оптимальном выборе шага. Из формулы оптимального шага (8) можно предположить, что это число шагов пропорционально входящему в нее интегралу

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

Величина этого интеграла зависит не только от кривизны *κ*(*l*)*,* но и от отрезка интегрирования . Такой критерий определения трудности можно применять для любых тестовых или прикладных задач.

Для гиперболического теста зависимость *κ*(*l*) можно получить, подставляя формулу (18) в формулу (22):

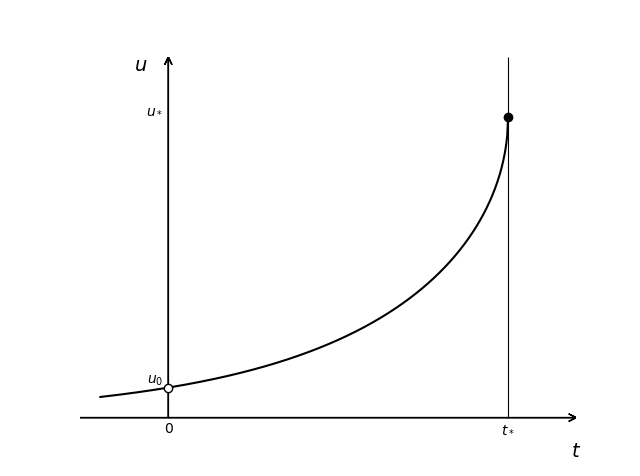
|  |  |  |
| --- | --- | --- |
|  |  | (25) |

Подстановка величины (25) в интеграл (24) и замена переменных  дают

|  |  |  |
| --- | --- | --- |
|  |  | (26) |

Пределы интегрирования , поэтому последний интеграл в (26) не превышает 2. С учетом множителя  перед интегралом неясно, можно ли назвать данный тест по-настоящему трудным. Однако в приведенных далее численных расчетах многие схемы давали срывы при больших *λ*. Кроме того, более трудного теста с такими уникальными выражениями через элементарные функции найти не удалось.

2.3. Тригонометрический тест**.** Упомянем еще один тест, в котором явно вычисляется длина дуги, и все имеющиеся величины выражаются через элементарные функции как аргумента *t*, так и аргумента *l*.

*Рис. 3.* Точное решение (28); ○ – начальное значение *u*, ● – конечное значение *u*.

Приведем набор формул этого теста (текстовые пояснения опускаем, поскольку они аналогичны гиперболическому тесту):

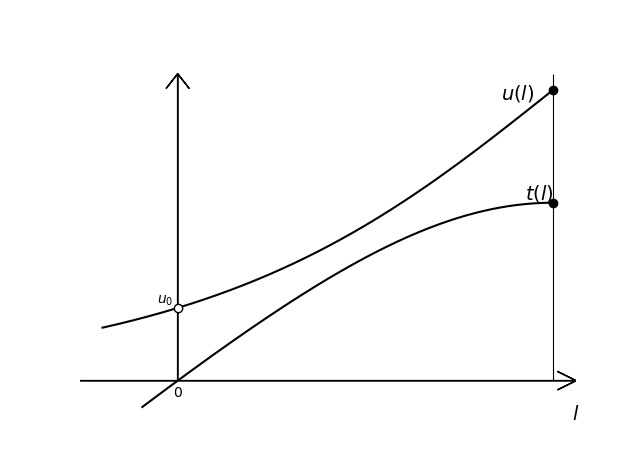
|  |  |  |
| --- | --- | --- |
|  |  | (27) |
|  |  | (28) |
|  |  | (29) |
|  |  | (30) |
|  |  | (31) |
|  |  | (32) |
|  |  | (33) |
|  |  | (34) |

Качественный вид решения при аргументе *t* изображен на рис. 3, а при аргументе *l* – на рис. 4. Поскольку мы положили , то решение *u*(*t*) монотонно возрастает на конечном промежутке времени от 0 до . При этом функция достигает значения , а производная в этой точке . Решение не имеет продолжения за .

Если , то  и полная длина дуги , хотя  остается конечным. Последнее объясняется тем, что при  правая часть , и движение вдоль интегральной кривой начинается очень медленно, так что полное время движения стремится к бесконечности; соответственно стремится к бесконечности длина дуги.

Отметим, что можно рассмотреть случай . В этом случае решение *u*(*t*) будет монотонно убывать, асимптотически стремясь к 0 при . Это решение несколько напоминает известный тест Далквиста.

Тригонометрический тест не позволяет получить задачи произвольно высокой жесткости. Поэтому он менее интересен для тестирования методов, чем гиперболический тест.



*Рис. 4.* Точное решение (29‑30); ○ – начальное значение *u*, ● – конечные значения *u* и *t*.

# 3. Численные расчеты

3.1. Условия тестирования**.** Для численных иллюстраций был выбран гиперболический тест со следующим набором параметров: ; для каждой используемой схемы *ν* увеличивали до тех пор, пока расчет не срывался. Для каждого *λ* начальный и конечный моменты расчета определялись так, чтобы в них выполнялось условие . Для , согласно формуле (22), полагалось

|  |  |  |
| --- | --- | --- |
|  |  | (35) |

Далее выполнялся расчет по выбранной схеме интегрирования; в нем вычислялись *u*, *L* и *κ*. Кривизна *κ* сначала возрастала от 1 до , а затем начинала убывать. Когда выполнялось условие , то расчет прекращался.

Опишем расчет . Для всех использованных схем он выполнялся одинаково по формулам точности . В начальный момент времени , согласно выбору начального условия, , , а вектор правых частей системы (4) для задачи (17) вычисляется по начальным данным. Мы можем вычислить величину первого шага  по кривизне  в формуле (8), провести вычисление по выбранной схеме и найти узел  и величины ,  в этом узле.

По найденным ,  вычисляем правые части системы (17) в узле . Находим длину вектора изменения правых частей и делим ее на величину шага по длине дуги; это дает нам кривизну в первом узле

|  |  |  |
| --- | --- | --- |
|  |  | (36) |

с точностью . Аналогичная процедура повторяется на всех последующих шагах.

Для расчета необходимо выбрать настроечные параметры формулы (8) для первой сетки. Мы не стали проводить специальные настройки и взяли следующие значения: , , , . Для перехода с первого этапа сгущения сеток на второй выбиралось значение критерия .

***Надежность программы.*** В жестких задачах сравнительно легко происходит срыв расчета, то есть программа оказывается недостаточно надежной. Опишем здесь две типичные ситуации.

**1°** В жестких задачах при аргументе *t* правые части  нередко оказываются сопоставимыми с максимально допустимыми на машине числами. При аргументе *l* правые части невелики: сумма их квадратов есть 1. Однако при вычислении знаменателей этих правых частей суммируются квадраты величин . При этом может произойти переполнение в промежуточных вычислениях. Это приводит к срыву расчетов.

Поэтому надо тщательно продумывать написание правых частей ОДУ, и при появлении слишком больших чисел автоматически производить явное сокращение максимальных множителей в числителях и знаменателях формул. По существу, это означает введение автоматического масштабирования. Этот прием полезен и для не особенно жестких задач.

**2°** Задачи малой жесткости все схемы считают успешно.Однако при достаточном увеличении жесткостикаждая схема может сорваться. В наших расчетах схема ERK4 успешно работала при , но срывалась при . Схемы ERK2 и ERK1 работали при , но срывались при .

Анализ показал, что на первом этапе шаг  оказывается огромным, и дает сетку только с одним интервалом: . При этом неправильно работают формулы дробления сетки на втором этапе. Удалось ликвидировать срывы, следующим образом исправив формулы сгущения сетки второго этапа (10‑12).

Если , то шаг  делится точно пополам. Если , то производится расчет только для граничных интервалов (11‑12). Если , то включается вычисление и для внутренних интервалов (10).

***Выбор схемы.*** Формула выбора шага (7) геометрически согласовывалась лишь со схемами первого порядка точности. Однако мы провели расчет с несколькими схемами разного порядка точности. Наиболее просты и мало трудоемки явные схемы. Поэтому мы взяли три явные схемы Рунге-Кутты ERK1, ERK2, ERK4 точностей ,  и  соответственно. Обычно надежность схем уменьшается с повышением их порядка точности, причем на жестких задачах это сказывается особенно сильно. Поэтому такое тестирование различных схем интересно.

3.2. Результаты расчетов**.** Все расчеты проводились с аргументом *l*. На каждой сетке локальная погрешность в каждой точке сетки  находилась путем вычитания численного решения  и  из точного *u*(*l*) и *t*(*l*). Затем вводилась интегральная погрешность

|  |  |  |
| --- | --- | --- |
|  |  | (37) |

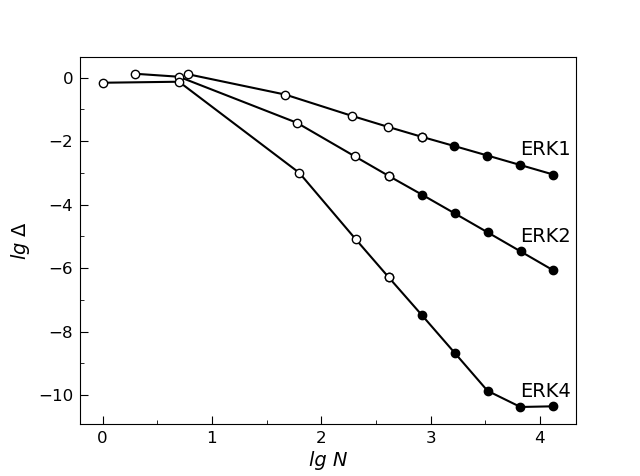
Такое определение погрешности удобно следующим: а) оно учитывает не абсолютные погрешности, а относительные, то есть адаптируется к масштабам конкретного решения; б) одновременно учитывает ошибки обеих функций системы (17); в) автоматически настраивается на длину дуги; г) является интегральной, что облегчает визуализацию погрешности и проведение сравнений. На дальнейших рисунках будет показана зависимость нормы погрешности от числа шагов сетки *N*. Рисунки выполнены в двойном логарифмическом масштабе. Тогда прямая линия означает степенной характер зависимости, а тангенс угла ее наклона есть эффективный порядок точности. Это легко позволяет проверить, соответствует ли скорость сходимости теоретическому порядку точности схемы.

***Точность.*** На рис. 5 показаны погрешности для схем разного порядка точности ERK1, ERK2, ERK4 при одинаковой жесткости . Эта жесткость не слишком велика, и все схемы ведут расчет в обычном режиме. Видно, что начала всех линий близки и соответствуют очень большой погрешности . Начала всех линий искривлены, но уже в пределах первого этапа линии переходят в прямые, а их наклоны соответствуют теоретическим порядкам точности.

Поэтому при увеличении числа узлов погрешность убывает тем быстрее, чем выше порядок точности схемы. Данные расчеты проводились до  узлов. При таком числе узлов погрешность для схемы ERK1 убывает до , а для схемы ERK2 – до . Для схемы ERK4 погрешность успевает выйти на ошибки округления, которые для данного *λ* составляют . Такие большие ошибки округления (при вычислениях теряется 6 знаков) связаны с достаточно большой жесткостью задачи.

Линии вторых этапов являются прямыми и практически точно продолжают прямые первых этапов; не наблюдается никакого скачка или излома между первым и вторым этапами. Последнее обстоятельство является неожиданным. Во всех ранее проводимых работах между первым и вторым этапом наблюдались изломы или скачки. По-видимому, это связано с тем, что данная работа является первым расчетом, в котором использован тест с явными выражениями всех величин через длину дуги.

Заметим, что для всех схем на первом этапе число узлов при каждом сгущении увеличивается не в 2 раза, особенно на первом сгущении; но на последующих сгущениях первого этапа отношение чисел интервалов *N* приближается к 2. Это свидетельствует о том, что заключительная сетка первого этапа близка к квазиравномерной.



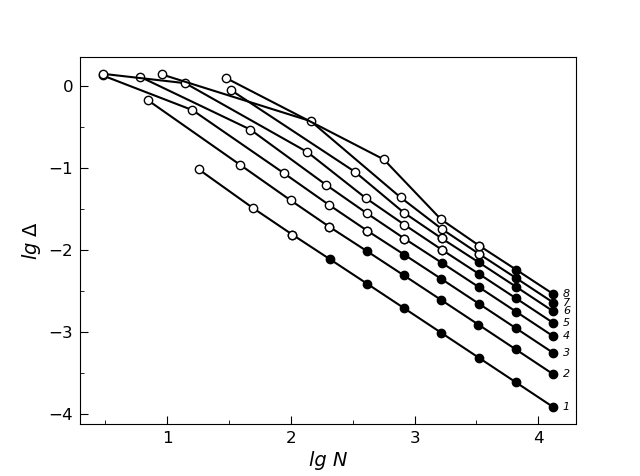
*Рис. 5.* Погрешности схем ERK для . Около каждой кривой указан порядок точности схемы. ○ – первый этап, ● – второй этап.

Разумеется, на втором этапе числа шагов *N* увеличиваются вдвое при каждом сгущении. Точное удвоение начинается с последней сетки первого этапа.

Отметим важное обстоятельство. Формула (7) выбора оптимального шага согласована лишь со схемами первого порядка точности, и само *κ* мы вычисляем по формуле точности . Однако это не препятствует тому, чтобы схемы ERK2 и ERK4 реализовали свой, более высокий порядок точности. Это свидетельствует об удачном выборе формулы оптимального шага.

***Надежность.*** На каждом из рис. 6‑8 даны графики погрешности только одной из схем ERK, зато для разных жесткостей. Значения жесткости указаны около каждой кривой. Сравним эти рисунки.

**1°** На рис. 6 приведены расчеты для схемы ERK1. Схема оказалась исключительно надежной, и даже при  не было обнаружено никаких срывов расчета; лишь при  расчет сорвался. При небольших  линии погрешности с самого начала являются прямыми с наклоном ‑1, соответствующим теоретическому порядку точности. При большей жесткости начала кривых искривлены, и могут стать даже немонотонными. Однако уже в пределах первого этапа линии становятся прямыми с наклоном ‑1, а затем гладко переходят в прямые второго этапа.



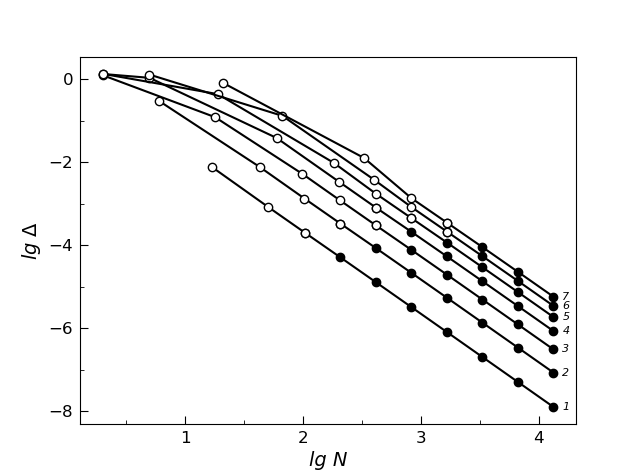
*Рис. 6.* Погрешности схемы ERK1 для различных жесткостей ; около линий указаны значения ν. ○ – первый этап, ● – второй этап.

При повышении жесткости от  до  происходит ухудшение точности расчета при одинаковом числе узлов. Однако наблюдаемую закономерность такого ухудшения мы не будем анализировать, так как она может относиться только к выбранному тесту.

Заметим, что при всех жесткостях переход с первого на второй этап происходит при почти одинаковом уровне погрешности  (но разумеется, при разных числах узлов). Остается открытым вопрос – является это обстоятельство случайным или закономерным? Ведь переход на второй этап производится не по уровню погрешности, а по критерию установления квазиравномерности сетки.

**2°** Расчеты по схеме ERK2 дали срыв при , то есть несколько ранее, чем для схемы ERK1. Поэтому по надежности схема ERK2 уступает схеме ERK1, как и следовало ожидать. Кривые погрешностей для жесткостей  приведены на рис. 7.

Поведение кривых качественно соответствует тому, что наблюдалось для схемы ERK1. Линии вторых этапов являются прямыми с наклоном ‑2, что соответствует теоретическому порядку точности. Конечные участки первых этапов так же являются прямыми с наклоном ‑2, и они гладко сопрягаются со вторыми этапами. Разумеется, поведение начальных участков первых этапов нерегулярно.



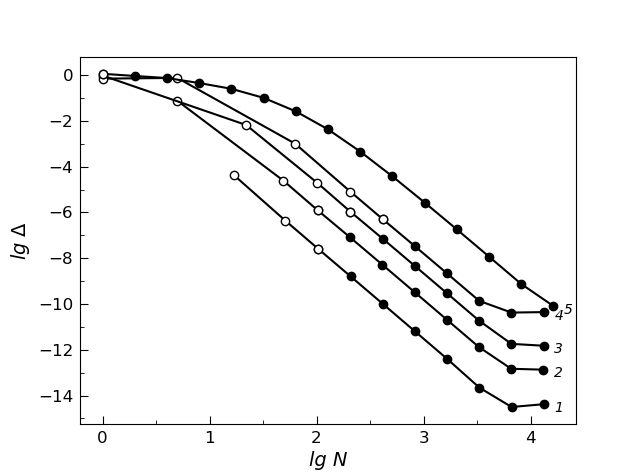
*Рис. 7.* Погрешности схемы ERK2; обозначения идентичны рис. 6.

Здесь так же переход с первого на второй этапы происходит при почти одинаковом уровне погрешности . Этот уровень погрешности существенно меньше, чем для схемы ERK1.

**3°** Расчеты по схеме ERK4 срывались при . Таким образом, схема оказалась менее надежной, чем ERK2, не говоря уже о ERK1. Кривые погрешности для  приведены на рис. 8.

На рис. 8 для  все линии вторых этапов являются прямыми с наклоном ‑4, что соответствует теории. Концы линий первых этапов так же являются прямыми с наклоном ‑4, и гладко сопрягаются с линиями вторых этапов. Видно, что при этих жесткостях схема ERK4 работает очень надежно.

Однако при  начало линии является сильно искривленным, и лишь при  линия становится прямой с теоретическим наклоном 4. При этом первый этап расчета фактически отсутствует (), поэтому сетка второго этапа оказывается не адаптированной, а просто равномерной. В результате при  не происходит построения оптимальной сетки. Можно сказать, что при такой жесткости схема ERK4 работает на грани срыва.



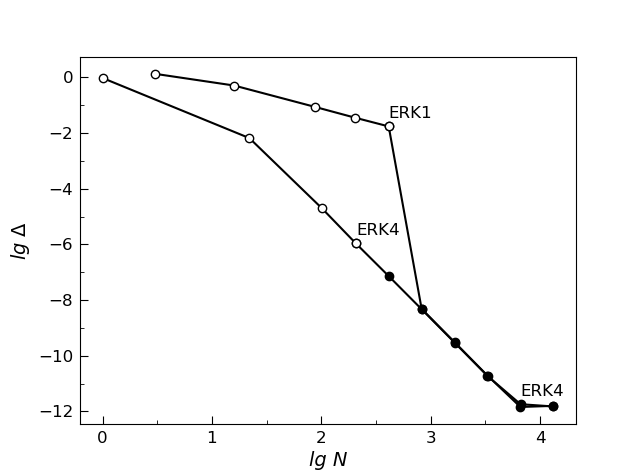
*Рис. 8.* Погрешности схемы ERK4; обозначения идентичны рис. 6.

Пока схема ведет себя надежно, расстояния между последовательными кривыми 1‑2, 2‑3, 3‑4 уменьшаются. Однако расстояние между прямолинейными участками 4‑5 сильно увеличивается. Это показывает, что если предел надежности схемы нарушен, то точность расчета сразу ухудшается.

В следствие высокого порядка точности схемы ERK4, погрешности быстро убывают при увеличении числа узлов. Поэтому каждая линия успевает выйти на ошибки округления. При всех жесткостях это происходит при примерно одинаковом числе узлов . Но уровень ошибок округления существенно возрастает при увеличении *λ*: это  при ,  при ,  при и  при  и .

***Замечание.*** Срывы всех схем зависят не только от величины *λ*, но и от параметров для выбора начальной сетки (8): , , *L* и интеграла. Мы намеренно выбрали небольшие значения этих параметров, чтобы тестировать схемы в трудных условиях. Если увеличивать  и , а также найти более удачные формулы для выбора *L* и интеграла, можно построить алгоритмы более высокой надежности с использованием тех же схем.

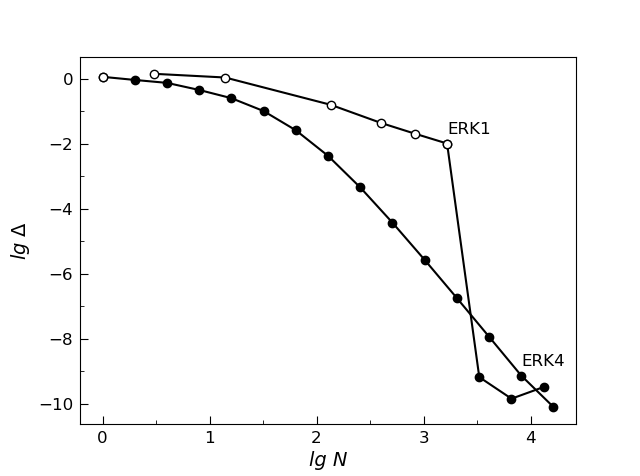
3.3. Смешанная стратегия**.** Рассмотрим несложный способ повышения надежности алгоритма. Схема ERK1 наиболее надежна среди явных схем. Проведем расчеты первого этапа по этой схеме. Результатом расчета даже в случае очень высокой жесткости будет построение хорошей начальной сетки для второго этапа.



*Рис. 9.* Смешанная стратегия. Погрешности для . Около линий подписаны схемы. ○ – первый этап, ● – второй этап.

Поскольку на первом этапе построена хорошая сетка, второй этап можно выполнять по менее надежной, но гораздо более точной схеме ERK4. При этом следует ожидать гораздо более высокой точности, чем при использовании схемы ERK1 на обоих этапах. Такая стратегия может дать хорошие результаты в тех случаях, когда расчет первого этапа по схеме ERK4 срывается из-за большой жесткости.

Были проведены расчеты по сравнению смешанной и «чистой» стратегий. На рис. 9 показаны результаты при умеренной жесткости . На нем видны две прямые линии. Верхняя прямая – расчет обоих этапов по схеме ERK1; ее наклон есть ‑1. Нижняя линия – расчет обоих этапов по схеме ERK4; ее наклон равен ‑4. Если выполнить первый этап по схеме ERK1, и затем перейти на схему ERK4, то второй этап расчета точно ложится на нижнюю линию.

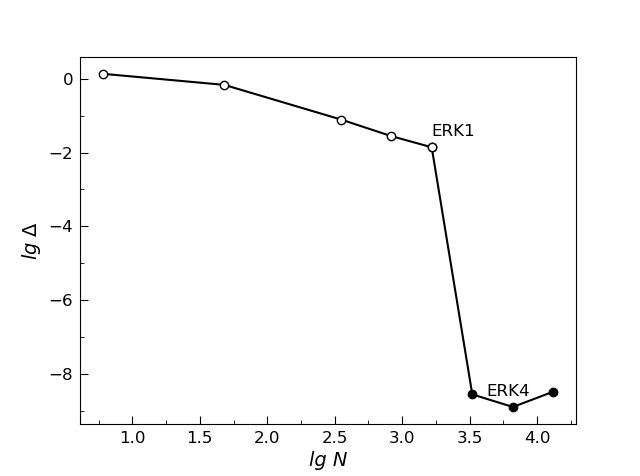


*Рис. 10.* Смешанная стратегия. Погрешности для ; обозначения идентичны рис. 9.

Поэтому при умеренной жесткости смешанная стратегия дает ту же самую точность, что «чистая» схема высокого порядка точности. Получается только два преимущества. Первое – небольшое уменьшение объема расчетов, так как первый этап выполняется по менее трудоемкой схеме ERK1. Второе – увеличение надежности, так как начало расчетов выполняется по более надежной схеме ERK1.

На рис. 10 показаны аналогичные расчеты для значительной жесткости . Здесь картина иная. «Чистый» расчет по схеме ERK4 близок к срыву и прямолинейный участок его кривой лежит довольно высоко. Расчет по смешанной стратегии надежно проходит первый этап, и затем резким скачком переходит ко второму этапу. Его линия второго этапа лежит существенно ниже, уже почти сразу приближаясь к ошибкам округления. Поэтому в данном случае смешанная стратегия дает существенное превосходство в точности. Она дает также заметный выигрыш в трудоемкости, так как расчет можно закончить на одну-две сетки раньше.

На рис. 11 показан расчет для большей жесткости . Расчет по «чистой» схеме ERK4 здесь срывается на первой же сетке. Однако расчет по смешанной стратегии благополучно проходит. При этом после первого этапа происходит переход на схему ERK4, и сразу же достигаются ошибки округления. Для такого уровня при «чистом» использовании схем ERK1 или даже ERK2 пришлось бы провести еще много сгущений сеток. Эти примеры наглядно иллюстрируют преимущество смешанной стратегии.



*Рис. 11.* Смешанная стратегия. Погрешности для ; обозначения идентичны рис. 9.

# 4. Пакет программ GEAM

Построенный численный метод реализован на языке Pyhton. Его выбор обусловлен относительной простотой использования и наличием множества готовых библиотек. Для визуализации данных использовался пакет matplotlib, а для эффективной работы с массивами – пакет numpy, в котором также реализованы вычислительные методы линейной алгебры.

## 4.1. Описание программы.

Программа написана таким образом, что все настройки задаются по умолчанию, и пользователю необходимо лишь задать правую часть уравнения и вызвать функцию run. Это называется штатным режимом.

***Структура пакета.*** Пакет GEAM (GEometrically Adaptive Mesh) представляет собой реализацию алгоритма, описанного в п. 1.2‑1.4. В него входят три файла.

1. solver\_data.py. Файл с описанием вспомогательного класса SolverData, целью, которого является хранение входных данных, промежуточных расчетов и результатов работы программы, а также упрощение изменения настроечных параметров для пользователя. Помимо этого, в классе реализованы вспомогательные методы для логарифмирования массивов чисел шагов и ошибок для построения графика ошибки. Этот класс необходим для упрощения кода основной программы, так как он содержит в себе все расчетные данные, позволяя избежать большого числа локальных переменных. В штатном режиме пользователю необязательно прибегать к его использованию, однако его придется редактировать для изменения стандартных значений настроечных параметров.
2. rhs\_func.py. В этом файле пользователь задает функцию правой части. Это единственный файл пакета, который пользователю необходимо редактировать в штатном режиме.
3. GEAM.py. Файл, в котором реализован непосредственно сам алгоритм. Также в нем написаны все вспомогательные процедуры: схемы Рунге-Кутты до 4 порядка включительно, переход в правой части к аргументу длина дуги, вычисление ошибки методом Ричардсона, сгущение сеток с адаптацией под кривизну решения на первом этапе, а также квазиравномерное сгущение сеток на втором этапе. Его изменение пользователем не предусмотрено, если только не понадобится по какой-то причине корректировать сам алгоритм.

Трехфайловая структура позволяет легко менять настроечные параметры, описание которых будет дано ниже. При этом пользователю не нужно разбираться в деталях реализации алгоритма. Все настройки вынесены в файлы rhs\_func.py и solver\_data.py. Алгоритм и вспомогательные процедуры оставлены в файле GEAM.py. Таким образом, пользователь может легко настраивать работу программы.

***Входные параметры.*** Для запуска нужно вызвать функцию run из файла GEAM.py. У нее три входных аргумента: t0 и T – начало и конец расчетного отрезка, а также u0 – вектор начальных условий, задаваемый в виде последовательности python, например в виде списка list или объекта numpy.ndarray.

Правые части исходной системы ОДУ задаются в файле rhs\_func.py в функции rhs.

***Результаты расчетов.*** Функция run возвращает последнюю сетку второй стадии в виде массива значений t и решение на этой сетке в виде матрицы u. Матрица имеет размерность , где число строк *N* соответствует числу узлов на последней сетке, а число столбцов *M* – числу компонент решения, то есть размерности исходной задачи до перехода к длине дуги. Иными словами, *n*-я строка матрицы – это вектор численного решения , а *m*-й столбец – значения *m*‑й компоненты решения во все моменты времени. Заметим, что размерность задачи *M* равна размерности вектора начальных условий u0, заданного пользователем.

После прекращения расчетов выводится график погрешности решения по Ричардсону в зависимости от числа шагов в двойном логарифмическом масштабе. Он должен состоять из двух участков. Второй участок представляет собой прямую с наклоном, равным порядку точности схемы на второй стадии. В конце кривой может возникнуть горизонтальный участок, связанный с достижением ошибок округления. На кривой, соответствующей первой стадии сгущения, допустимы небольшие пилообразные осцилляции. Причина в том, что сетки первого этапа сгущаются не квазиравномерно, и метод Ричардсона неприменим. Для наглядности точки, относящиеся к первому этапу, закрашены белым, а точки второго этапа – черным, как на графиках в предыдущем разделе.

График решения не выводится, так как решение может быть вектором и иметь большое число компонент.

***Сгущения сеток.*** Для начала расчета на первой сетке по умолчанию задаются следующие параметры: Nmin = 6, Nmax = 20, L\_old = 1. Они используются в формуле (8). Конкретные значения выбраны по результатам расчетов тестовых задач. Значение интеграла int\_old так же берется равным 1. После расчета на первой сетке становятся известны значения L\_old и int\_old, и эти новые значения подставляются в формулу (8) при следующем расчете. Значения Nmin и Nmax удваиваются. Расчеты продолжаются согласно п. 1.4, пока сетки не удовлетворят критерию близости crit1, который по умолчанию задается равным 0.03. На этом завершается первая стадия.

На втором этапе сетки сгущаются квазиравномерно согласно п. 1.4 до тех пор, пока погрешность по Ричардсону не станет меньше заданной точности crit2 (по умолчанию ). При этом должно быть выполнено не менее 3 сгущений, чтобы вычислить оценку погрешности по Ричардсону и определить скорость ее убывания. В коде это выражается соответствующим условием на продолжение цикла сгущений.

При сгущениях программа запоминает все промежуточные решения на каждой сетке в поле case.solutions переменной case класса SolverData. Также запоминаются все промежуточные размеры сеток в массивах case.grid\_sizes, расчетные длины дуг для каждой сетки в case.lengths, массивы шагов в case.steps, интегралы из формулы (8) в case.integrals и значения параметров  и  в case.Nmin и case.Nmax. Помимо этого, в поле case.switch сохраняется номер последней сетки первого этапа, после которой наступает переход на вторую стадию, при нумерации с 0. Это делается для удобства отладки, если понадобится посмотреть более детально на то, как именно происходил процесс сгущения.

Переменная case не возвращается функцией run, так как в штатном режиме она не нужна, но при необходимости пользователь может добавить ее в список возвращаемых аргументов и получить полную информацию о прохождении алгоритма.

***Настройка.*** Множество значений по умолчанию настроечных параметров программы были выбраны экспериментально при расчетах тестовой задачи. При решении других задач может ухудшиться сходимость из-за плохо подобранных начальных значений длины дуги или интеграла в формуле (8). Как отмечено выше, эти параметры вынесены в отдельный класс SolverData в файл solver\_data.py. В функции-конструкторе \_\_init\_\_ класса SolverData пользователь может изменить:

* length\_0 – значение длины дуги, используемое в формуле (8) и неизвестное до начала расчетов (по умолчанию 1);
* integral\_0 – значение интеграла в формуле (8), неизвестное до начала расчетов (по умолчанию 1);
* crit1 – критерий остановки первой стадии (по умолчанию 0,03);
* crit2 – критерий остановки второй стадии (по умолчанию );
* approx\_order\_1 – порядок точности схемы РК, используемой на первой стадии (по умолчанию 1);
* approx\_order\_2 – порядок точности схемы РК, используемой на второй стадии (по умолчанию 4);
* Nmin – параметр формулы (8), минимальное желательное число шагов первой сетки (по умолчанию 6);
* Nmax – параметр формулы (8), максимальное желательное число шагов первой сетки (по умолчанию 20).

Важно при изменении Nmin и Nmax оставить их списками из одного значения (то есть не 6 и 20, а [6] и [20]), так как далее при работе программы эти списки будут дополняться новыми значениями Nmin и Nmax со следующих сеток.

***Вспомогательные процедуры.*** В алгоритме используются следующие вспомогательные функции.

В функциях erk1, erk2, erk3 и erk4 реализован один шаг схемы РК порядка, соответствующего названию. На вход принимаются вектор u на этом шаге, величина шага h и заранее вычисленный вектор правой части F\_n. Каждая функция возвращает вектор численного решения на следующем шаге, вычисленный в соответствии со своим порядком точности.

Функция erk принимает на вход те же аргументы, что предыдущие функции, а также порядок точности p и возвращает значение одной из указанных выше функций в зависимости от указанного порядка.

Функция F принимает на вход вектор решения u, в котором нулевой компонентой является время, и вычисляет по нему вектор правых частей в длине дуги, используя функцию rhs, заданную пользователем.

Функция richardson вычисляет ошибку по Ричардсону между двумя решениями в текущий и предыдущий моменты времени. На вход подаются векторы u1 и u2, соответствующие текущей и предыдущей сетке, массив шагов предыдущей сетки H и порядок точности выбранной схемы p, так как он участвует в формуле погрешности по Ричадсону. Возвращается число R – значение ошибки.

Процедура run\_iterations отвечает за расчет на одной сетке первой стадии. На вход она принимает экземпляр case класса SolverData и, используя сохраненные в нем значения из массивов case.lengths и case.integrals, проводит один этап сгущения. Решение на новой сетке добавляется в массив case.solutions, новые рассчитанные длина дуги и интеграл из формулы (8) добавляются в массивы case.lengths и case.integrals, массив шагов добавляется в case.steps. Также сохраняется новое значение числа шагов в case.grid\_sizes.

Функция thicken принимает на вход массив шагов steps и вычисляет по нему новый массив шагов steps\_new по формулам (10‑12) квазиравномерного сгущения. Она используется на второй стадии.

Процедура stage1 принимает на вход экземпляр case класса SolverData и проводит серию сгущений первой стадии до выполнения условия завершения первой стадии, в цикле вызывая процедуру run\_iterations и сравнивая сетки по критерию близости case.crit1.

Процедура stage2 так же принимает экземпляр case класса SolverData и проводит серию сгущений уже второй стадии, аналогично процедуре stage1.

За построение графика ошибки отвечает метод plot\_error класса SolverData. На вход он ничего не принимает, так как вся необходимая информация уже содержится в классе.

Также в классе SolverData имеется метод log\_data, возвращающий прологарифмированные массивы шагов и ошибок. Этот метод используется методом plot\_error для построения графика.

## 4.2. Контрольный тест.

Для расчета, погрешности которого приведены на рис. 11, необходимо модифицировать файл с функцией правой части rhs\_func.py следующим образом:

import numpy as np

def rhs(u):

'''

Right-hand side function of differential equation.

Input

-------

u - vector, argument to the function

Returns

-------

Vector rhs(u)

'''

lambda\_ = 1e6

return np.array([1., np.sinh(lambda\_ \* u[1])])

Далее в командном окне или в отдельном файле необходимо импортировать из модуля GEAM функцию run и вызвать ее со следующими входными аргументами:

from GEAM import run

t0 = 0

T = 1

u0 = [1]

t, u = run(0, 1, [1])

## 4.3. Листинги.

Не обязательно. Объем и так большой

# 5. Заключение

Проведены численные расчеты жестких задач для ОДУ с оптимальным выбором шага по кривизне интегральной кривой. Построен тест, в котором решение выражается в элементарных функциях при использовании в качестве аргумента как времени *t*, так и длины дуги *l*. Этот тест позволил провести точное и количественное сравнение различных методов решения.

Показано, что при оптимальном выборе шага расчеты задач даже высокой жесткости удается проводить по явным схемам Рунге-Кутты. При этом схема первого порядка точности обладает наибольшей надежностью (расчет срывается только при очень высоких жесткостях), но точность ее мала. Схема второго порядка имеет заметно меньшую надежность, но более высокую точность. Схема четвертого порядка наиболее точна, но наименее надежна.

Предложена смешанная стратегия, при которой первый этап – построение удовлетворительно адаптированной к решению сетки – происходит по схеме первого порядка; второй этап – сгущение адаптированной сетки – выполняется по схеме четвертого порядка. Это обеспечивает хорошую надежность и высокую точность численного метода.

# Библиографический список

1. Хайрер Э., Нерсет С., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи. – М.: Мир, 1990.
2. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи. М.: Мир, 1999.
3. Шалашилин В.И., Кузнецов Е.Б. Метод продолжения решения по параметру и наилучшая параметризация. – М.: Эдиториал УРСС, 1999.
4. Белов А.А., Калиткин Н.Н., Пошивайло И.П. Геометрически-адаптивные сетки для жестких задач Коши // Доклады Академии наук. **466**:3 (2016), 276‑281.
5. Белов А.А., Калиткин Н.Н. Выбор шага по кривизне для жестких задач Коши // Математическое моделирование. **28**:11 (2016), 97‑112.
6. Белов А.А., Булатов П.Е., Калиткин Н.Н. Сравнительный анализ алгоритмов автоматического выбора шага для жестких задач Коши // Препринты ИПМ им. М.В. Келдыша. 2019. № 146, 34 с. https://keldysh.ru/papers/2019/prep2019\_146.pdf
7. Пошивайло И.П. Жесткие и плохо обусловленные нелинейные модели и методы их расчета. Диссертация на соискание ученой степени кандидата физико-математических наук: 05.13.18. – Москва, 2015. – 89 с.
8. Белов А.А., Калиткин Н.Н. Экономичные методы численного интегрирования задачи Коши для жестких систем ОДУ // Дифференциальные уравнения. **55**:7 (2019), 907‑918.
9. Richardson L.F., Gaunt J.A. The deferred approach to the limit Phil. Trans. A. 1927. Vol. 226. P. 299‑349.
10. Марчук Г.И., Шайдуров В.В. Повышение точности решений разностных уравнений. М.: Наука, 1979.
11. Калиткин Н.Н. Численные методы. М.: Наука, 1978.
12. Калиткин Н.Н., Альшин А.Б., Альшина Е.А., Рогов Б.В. Вычисления на квазиравномерных сетках. М.: Физматлит, 2005.
13. Жолковский Е.К., Белов А.А., Калиткин Н.Н. Решение жестких задач Коши явными схемами с геометрически-адаптивным выбором шага // Препринты ИПМ им. М.В. Келдыша. 2018. №227, 20 с. http://keldysh.ru/papers/2018/prep2018\_227.pdf

Оглавление

[1. Методы численного интегрирования жестких систем 2](#_Toc72523417)

[1.1. Длина дуги. 2](#_Toc72523418)

[1.2. Выбор шага 3](#_Toc72523419)

[1.3. Оптимальный шаг 4](#_Toc72523420)

[1.4. Двухэтапная стратегия 5](#_Toc72523421)

[2. Апробация на тестах 8](#_Toc72523422)

[2.1. Требования к тестам 8](#_Toc72523423)

[2.2. Гиперболический тест 9](#_Toc72523424)

[2.3. Тригонометрический тест 13](#_Toc72523425)

[3. Численные расчеты 15](#_Toc72523426)

[3.1. Условия тестирования 15](#_Toc72523427)

[3.2. Результаты расчетов 17](#_Toc72523428)

[3.3. Смешанная стратегия 22](#_Toc72523429)

[4. Пакет программ GEAM 25](#_Toc72523430)

[4.1. Описание программы. 25](#_Toc72523431)

[4.2. Контрольный тест. 29](#_Toc72523432)

[4.3. Листинги. 29](#_Toc72523433)

[5. Заключение 29](#_Toc72523434)

[Библиографический список 30](#_Toc72523435)