# Реферат

Решение многих технических, химических, а также биологических задач требует решения задачи Коши. Эту задачу можно решать разными способами, как аналитическими, так и численными, применяя ЭВМ. Очень часто бывает важно получить результат в сжатые сроки. В этом случае предпочтение отдается численным методам. Кроме того, встречаются такие сложные дифференциальные уравнения, найти аналитическое решение которых либо вообще не представляется возможным, либо для этого требуются очень большие затраты времени и сил.

В работе детально рассматривается метод Рунге-Кутты четвертого порядка с автоматическим выбором длины шага интегрирования (это обеспечивает гораздо более высокую точность вычислений по сравнению с методом, использующим шаг постоянной длины), приводится необходимая теоретическая сводка, описание метода, а также программа для ЭВМ, результаты ее выполнения и иллюстрации.

Ключевые слова: дифференциальное уравнение, метод Рунге-Кутты, метод Эйлера, порядок метода Рунге-Кутты, задача Коши, ряд Тейлора, отрезок, коэффициенты, шаг интегрирования, интегральная кривая.

Работа содержит 36 листов, включая 8 графиков, 4 иллюстрации и 12 таблиц.

#### Содержание

Введение

1. Теоретическая часть

1.1 Постановка задачи

1.2 Метод Эйлера

1.3 Общая формулировка методов Рунге-Кутты

1.4 Обсуждение методов порядка 4

1.5 «Оптимальные» формулы

1.6 Условия порядков для методов Рунге-Кутты

1.7 Оценка погрешности и сходимость методов Рунге-Кутты

1.7.1 Строгие оценки погрешности

1.7.2 Главный член погрешности

1.7.3 Оценка глобальной погрешности

1.8 Оптимальный выбор шага

2. Практическая часть

2.1 Описание программы «Ilya RK-4 версия 1.43»

# Заключение

# Список использованных источников

### Приложение А. Графики функций

### Приложение Б. Пример таблицы значений функции y(x)

# Приложение В. Листинг программы «Ilya RK-4 версия 1.43»

# Введение

Ввиду того, что для методов Рунге-Кутты не нужно вычислять дополнительные начальные значения, эти методы занимают особое место среди методов классического типа. Ниже будут рассмотрены их свойства, а также некоторые ограничения, присущие этим методам.

С увеличением числа этапов для больших задач, решаемых этими методами, возникли бы трудности с памятью ЭВМ, кроме того (и это важнее), для больших задач, как правило, всегда велики константы Липшица. В общем случае это делает методы Рунге-Кутты высокого порядка не пригодными для таких задач. Во всяком случае, другие методы обычно эффективнее и им следует отдавать предпочтение. Однако методы Рунге-Кутты четвертого порядка являются достаточно легко реализуемыми на ЭВМ, а наличие автоматического выбора шага дает возможность производить вычисления с хорошей точностью. Поэтому их целесообразно применять для довольно широкого множества задач.

Методы Рунге-Кутты имеют несколько весомых достоинств, определивших их популярность среди значительного числа исследователей. Эти методы легко программируются, обладают достаточными для широкого круга задач свойствами точности и устойчивости. Эти методы, как и все одношаговые методы, являются самостартующими и позволяют на любом этапе вычислений легко изменять шаг интегрирования.

В работе основное внимание сконцентрировано на вопросах точности и эффективности решения задач того типа, для которых методы Рунге-Кутты приемлемы.

Программная реализация методов Рунге-Кутты четвертого порядка с автоматическим выбором шага представлена в виде программы, написанной на языке высокого уровня *Borland C++ 3.1*. Программу можно запускать в среде *MS-DOS* или *Windows® 95/98/Me/2k/XP*. В качестве выхода программа пишет таблицу значений в файл на диск и рисует график на экране ЭВМ.

Для проверки результатов работы созданной программы одни и те же дифференциальные уравнения решались в математическом пакете *Waterloo Maple 9.01* и при помощи созданного приложения (версия 1.43), проводился анализ таблиц значений и графиков решений.

# 1. Теоретическая часть

# 1.1 Постановка задачи

Дано дифференциальное уравнение и начальное условие, то есть поставлена задача Коши:

(2.1.1)



Требуется отыскать интегральную кривую, удовлетворяющую поставленной задаче Коши с помощью метода Рунге-Кутты четвертого порядка с автоматическим выбором шага на отрезке . Задачу можно решить аналитически, найдя решение дифференциального уравнения и подставив в него начальное условие, тем самым, отыскав требуемую интегральную кривую. Но для нас интерес представляет решение данной задачи с применением численного метода, а конкретнее – метода Рунге-Кутты 4-го порядка с автоматическим выбором шага, то есть численное решение. Автоматический выбор шага – необходимое условие адекватного поведения программы при резко изменяющихся функциях, задающих интегральную кривую, позволяющее отразить все моменты в поведении интегральной кривой и добиться высокой точности.

**1.2 Метод Эйлера**

Метод Эйлера для решения начальной задачи (2.1.1) был описан Эйлером в 1768 году. Этот метод весьма прост. Его глобальная погрешность имеет вид , где  – постоянная, зависящая от задачи, и  – максимальная длина шага. Если желательно, скажем, получить 6 точных десятичных знаков, то требуется, следовательно, порядка миллиона шагов, что не слишком удовлетворительно. С другой стороны, еще со времен Ньютона известно, что можно найти гораздо более точные методы, если  не зависит от , то есть если мы имеем задачу (2.1.1), решаемую квадратурой

. (2.2.1)

В качестве примера можно рассмотреть первую квадратурную формулу Гаусса, также называемую «правилом средней точки»:

 (2.2.2)

где  и  – граничные точки подинтервалов, на которые разбит интервал интегрирования. Известно, что оценка глобальной погрешности этой формулы  имеет вид . Таким образом, если желаемая точность составляет 6 десятичных знаков, ее обычно можно получить приблизительно за 1000 шагов, то есть этот метод в тысячу раз быстрее. Поэтому Рунге поставил следующий вопрос: нельзя ли распространить этот метод на исходную задачу Коши? Первый шаг длины  должен иметь вид

. (2.2.3)

Но какое значение взять для ? За неимение лучшего естественно использовать один малый шаг метода Эйлера длины . Тогда из предыдущей формулы получим:

 (2.2.4)

Решающим обстоятельством здесь является умножение  в третьем выражении на , в результате чего влияние погрешности становится менее существенным. Точнее, вычислим для  разложение Тейлора по степеням :

 (2.2.5)

Его можно сравнить с рядом Тейлора для точного решения, который получается из того, что  путем повторного дифференцирования с заменой  на  каждый раз, когда оно появляется:

 (2.2.6)

Вычитая из последнего равенства предыдущее, получим для погрешности первого шага выражение

 (2.2.7)

Таким образом, если все частные производные  второго порядка ограничены, то

.

Чтобы получить приближенное значение решения исходной задачи в конечной точке , будем применять формулы (2.2.4) последовательно к интервалам . Приведенные выше формулы являются усовершенствованным методом Эйлера. Для вычислений с высокой точностью, однако, следует пользоваться другими методами, одним из которых как раз является метод Рунге-Кутты.

**1.3 Общая формулировка методов Рунге-Кутты**

Рунге и Хойн построили новые методы, включив в указанные формулы один или два добавочных шага по Эйлеру. Но именно Кутта сформулировал общую схему того, что теперь называется методом Рунге-Кутты.

Пусть  – целое положительное число (число стадий, этапов) и  – вещественные коэффициенты. Тогда метод

 (2.3.1)

называется -стадийным явным методом Рунге-Кутты для исходной задачи Коши (2.1.1)

Обычно коэффициенты удовлетворяют условиям



. (2.3.2)

Эти условия были приняты Куттом без каких-либо комментариев. Смысл их заключается в том, что все точки, в которых вычисляется , являются приближениями первого порядка к решению. Эти условия сильно упрощают вывод условий, определяющих порядок аппроксимации для методов высокого порядка. Однако для методов низких порядков эти предположения необходимыми не являются.

Метод Рунге-Кутты имеет порядок , если для достаточно гладких задач (2.1.1) справедливо неравенство

, (2.3.3)

то есть ряды Тейлора для точного решения  и для  совпадают до члена  включительно.

После статьи Бутчера вошло в обычай символически представлять метод (2.3.1) по средствам следующей таблицы:

|  |  |
| --- | --- |
|  |  |
|  |  |

**1.4 Обсуждение методов порядка 4**

Подойдем теперь вплотную к определению 4-стадийных методов Рунге-Кутты (2.3.1) с таким расчетом, чтобы они имели порядок 4. Для этого необходимо вычислить производные порядков 1, 2, 3 и 4 от  при  и сравнить их с производными точного решения. Теоретически при известных правилах дифференциального исчисления это совершенно тривиальная задача. Однако с использованием (2.3.2) получаются следующие условия:



Эти вычисления очень утомительны и емки. Их громоздкость очень быстро растет для более высоких порядков.

***Лемма 1.***

Если

 (2.4.2)

то уравнения d), g) и h) являются следствием остальных.

*Доказательство.*

Покажем это для g). C помощью уравнений c) и e) получим:



Для уравнений d) и h) процедура аналогична.

Покажем, что в нашем случае условие



является и необходимым.

***Лемма 2.***

При  (2.4.2) следует из уравнений (2.4.1) и уравнений (2.3.2).

Для доказательства потребуется следующая лемма 3.

***Лемма 3.***

Пусть  и  суть 3x3-матрицы, такие что

, (2.4.3)

тогда либо , либо , где .

*Доказательство.*

Если , то из  следует . Если же , то существует вектор , такой, что , и поэтому . Но тогда из (2.4.3) следует, что  должен быть пропорционален вектору .

Докажем теперь предыдущую лемму. Введем величины  для . Итак, надо доказать, что . Введем теперь матрицы

 (2.4.4)

Перемножение этих матриц с использованием условий (2.4.1) дает

, (2.4.5)

причем



Далее последний столбец  не может быть нулевым, так как из того, что , следует



в силу условия h). Таким образом, из последней леммы следует, что . Последнее тождество  вытекает из равенства , которое является следствием условий a) и b).

***Теорема****.*

Если выполнены предположения , то уравнения (2.4.1) эквивалентны следующим:

 (2.4.6)

*Доказательство.*

Из j) и h) следует, что

. (2.4.7)

Отсюда, в частности, вытекает, что в силу k) .

*Решение уравнений (2.4.6).* Уравнения a)-e) и k) выражают тот факт, что коэффициенты  и  являются весами и узлами квадратурной формулы четвертого порядка при  и . В силу (2.4.7) возможны следующие четыре случая:

1) . (2.4.8)

Тогда уравнения a)-e) образуют невырожденную линейную систему для определения . Эта система имеет решение:



Остальные три случая с двойными узлами основаны на правиле Симпсона:

2) ;

3) ;

4) .

После того, как выбраны  и , получаем  из уравнения j), и тогда два уравнения f) и i) образуют линейную систему для определения  и .

Определитель этой системы

,

согласно (2.4.7) не равен нулю. Наконец, из того, что  находим ,  и .

Особенно популярными стали два варианта, которые выбрал Кутта в 1901 году. Это случай 3) при  и случай 1) при . Оба метода обобщают классические квадратурные формулы, сохраняя их порядок. Первый из них более популярен, однако второй более точен.

*Правило 3/8*

|  |  |
| --- | --- |
|  |  |
|  |  |

*Классический метод Рунге-Кутты*

|  |  |
| --- | --- |
|  |  |
|  |  |

**1.5 «Оптимальные» формулы**

Предпринималось много исследования, чтобы выбрать возможно «лучшие» из множества различных формул Рунге-Кутты 4-го порядка.

Первой попыткой в этом направлении был очень популярный метод, который в 1951 году предложил Гилл. Он преследовал цель уменьшить на сколько возможно количество требуемой машинной памяти («регистров»). Этот метод широко использовался на первых компьютерах в пятидесятых годах и представляет поэтому исторический интерес. Гилл заметил, что больше всего машинной памяти нужно при вычислении , когда «требуются регистры для хранения в какой-либо форме» величин

.

Ясно, что для третьей стадии будет достаточно трех регистров, если подлежащие хранению величины линейно зависимы, то есть если

.

Гилл заметил, что это условие удовлетворяется для методов типа 3), если . Получающийся метод можно в таком случае переформулировать следующим образом:



В настоящее время производительность компьютеров очень сильно возросла по сравнению с машинами 50-х годов, что привело к прекращению использования данного метода на практике при расчетах. Справедливости ради стоит отметить, что этот метод все-таки рационально употреблять в случаях, когда требуется решать системы дифференциальных уравнений очень высокой размерности со сложными функциями.

**1.6 Условия порядков для методов Рунге-Кутты**

Рассмотрим структуру условий, определяющих порядок метода, или условий порядка, как их называют для краткости. Способ вывода условий порядка прошел большую эволюцию. Он совершенствовался главным образом под влиянием работ Бутчера.

Так как явные методы Рунге-Кутты являются частным случаем неявных, то можем выписать условия, при которых метод имеет заданный порядок.

Метод

|  |  |
| --- | --- |
|  |  |
|  |  |

(где на свободных местах должны стоять нули) имеет порядок , если удовлетворяется уравнение



(2.6.1)



для каждого дерева с корнем и не более чем с разветвлениями[[1]](#footnote-1).



При эти условия, обеспечивающие порядок 4, и соответствующие деревья имеют следующий вид:



|  |  |
| --- | --- |
|  | (2.6.2) |
|  | (2.6.3) |
|  | (2.6.4) |
|  | (2.6.5) |
|  | (2.6.6) |
|  | (2.6.7) |
|  | (2.6.8) |
|  | (2.6.9) |

Заметим, что для меньших значений мы берем соответствующее подмножество этих условий, а для меньших оставляем лишь некоторые из указанных членов.



Из (2.9) видим, что действительно необходимо 4 этапа, так как если бы их было меньше, то был бы опущен единственный член в левой части этого уравнения. Для явных методов в общем случае выполняется неравенство . Фактически (для тех значений, для которых это известно) минимальное значение для данного указано в следующей таблице:



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  | 1 | 2 | 3 | 4 | 6 | 7 | 9 |

Общие классы методов с этими значениями и легко найти в случае .



Для :



|  |  |
| --- | --- |
| 0 |  |
|  | 1 |

Это известный метод Эйлера.

Для :



|  |  |
| --- | --- |
|  |  |
|  |  |

Это однопараметрическое семейство имеет требуемый порядок для любого ненулевого значения .



Для имеется три семейства, из которых первые два таковы:



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

Каждое из них имеет один параметр . Третье семейство имеет в качестве параметров и , причем



.



Вывод методов с более сложен, но его можно упростить, положив



(2.6.10)



(что влечет равенство ), так как это позволяет опустить уравнения (2.6.3), (2.6.5), (2.6.8) и (2.6.9). Интересно также, что (2.6.10) является следствием (2.6.2) – (2.6.9).



План вывода конкретного метода этого порядка можно выполнить при условии, что не возникает несовместных систем.

*Шаг 1.* Выбираем значения , и полагаем .



*Шаг 2.* Из (2.6.2), (2.6.3), (2.6.4) и (2.6.6) находим .



*Шаг 3.* Из уравнения (это уравнение есть разность уравнений (2.6.5) и (2.6.7)) находим .



*Шаг 4.* Из (2.6.10) находим .



*Шаг 5.* Вычисляем .



В случае шаг 2 приводит к выбору и при условии, что , . В частности, имеем известный метод:



|  |  |
| --- | --- |
|  |  |
|  |  |

**1.7 Оценка погрешности и сходимость методов Рунге-Кутты**

Со времен работы Лагранжа и особенно Коши всякий установленный численно результат принято сопровождать надежной оценкой погрешности. Лагранж дал известные оценки погрешности многочленов Тейлора, а Коши вывел оценки для погрешности метода ломаных Эйлера. Через несколько лет после первых успехов методов Рунге-Кутты также пришел к заключению, что для этих методов нужны оценки погрешностей[[2]](#footnote-2).

**1.7.1 Строгие оценки погрешности**

Способ, которым Рунге получил оценку погрешности, делаемой на одном шаге («локальной погрешности»), может быть описан следующим образом. Для метода порядка  рассмотрим локальную погрешность

 (2.7.1)

и воспользуемся ее тейлоровским разложением:

, (2.7.2)

где  и . Явное вычисление  дает выражение вида

, (2.7.3)

где  и  содержат частные производные  до порядков  и  соответственно. Далее поскольку , имеем . Таким образом, если ограничены все частные производные  до порядка  включительно, имеем  и . Следовательно, существует постоянная  такая, что  и

. (2.7.4)

Бибербах использовал несколько иной подход. Запишем

 (2.7.5)

и воспользуемся тейлоровскими разложениями

 (2.7.6)

Для векторных функций эти формулы справедливы покомпонентно (возможно, с различным ). В силу условий порядка первые члены разложения (2.6.5) по степеням  обращаются в нуль. Таким образом, справедлива следующая теорема.

***Теорема.***

Если метод Рунге-Кутты (2.3.1) имеет порядок  и если все частные производные  до порядка  включительно существуют и непрерывны, то локальная погрешность метода (2.3.1) допускает следующую строгую оценку:

, (2.7.7)

или

. (2.7.8)

Продемонстрируем этот результат, применяя к скалярному дифференциальному уравнению первый метод Рунге-Кутты (2.2.4), который имеет порядок . Дифференцируя (2.1.1), получим

. (2.7.9)

Вторая производная величины  имеет вид



Если условия теоремы выполнены, то легко видеть, что выражения (2.7.9) и (2.7.10) ограничены постоянной, которая не зависит от , что и дает оценку (2.7.8).

**1.7.2 Главный член погрешности**

Для методов высших порядков строгие оценки погрешностей, подобные (2.7.7), становятся очень непрактичными. Поэтому гораздо более реалистично рассматривать первый ненулевой член в тейлоровским разложении погрешности.

***Теорема.***

Если метод Рунге-Кутты имеет порядок  и если  непрерывно дифференцируема  раз, то для главного члена погрешности имеем:

. (2.7.11)

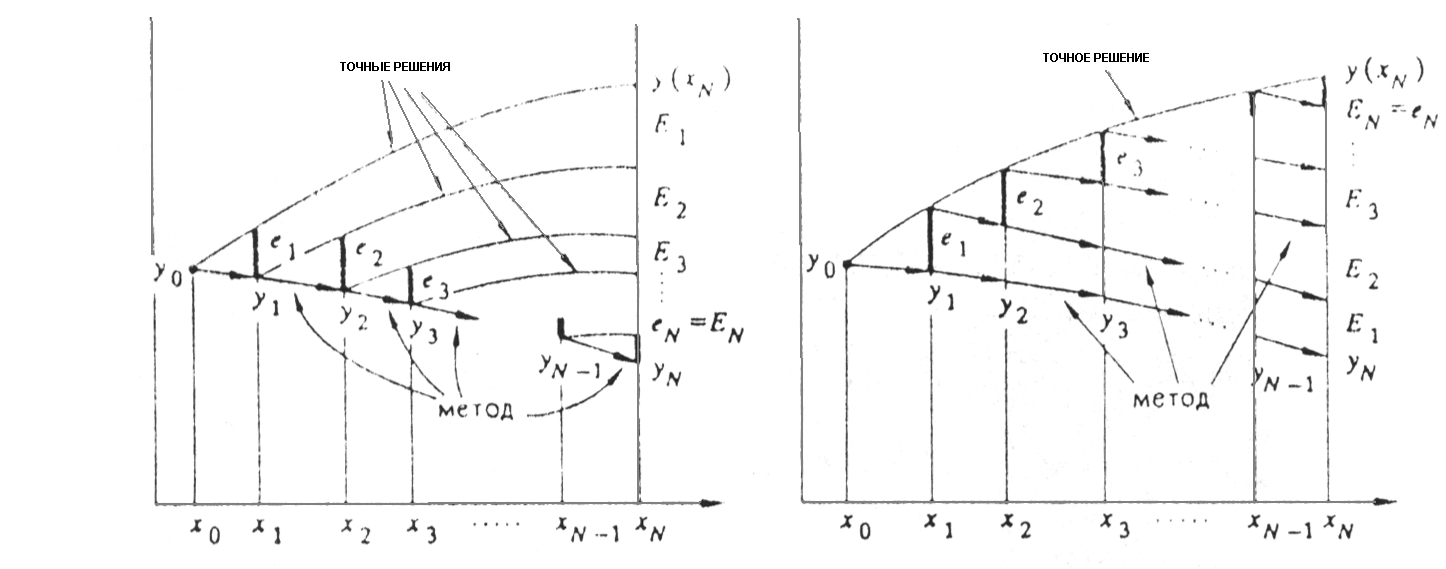
 (2.7.12)

**1.7.3 Оценка глобальной погрешности**

Глобальной (накопленной) погрешностью[[3]](#footnote-3) называется погрешность численного решения после выполнения нескольких шагов. Пусть мы имеем некоторый одношаговый метод, с помощью которого при заданных начальных данных  и длине шага  мы определяем численное решение , аппроксимирующее . Воспользуемся обозначениями Хенричи для этого процесса:

, (2.7.13)

и назовем  функцией приращения для данного метода.



*Оценивание глобальной погрешности методами a) и b)*

Тогда численное решение в точке  получается с помощью пошаговой процедуры

, (2.7.14)

и наша задача состоит в оценке глобальной погрешности

 (2.7.15)

Эта оценка находится простым способом: локальные погрешности переносятся в конечную точку и затем складываются. Этот «перенос погрешностей» можно выполнить двумя разными способами:

a) перенося погрешность вдоль кривых точных решений; этот способ может дать хорошие результаты, если известны хорошие оценки распространения погрешности для точных решений.

b) перенося погрешность -го шага посредством выполнения  шагов численного метода; этот способ использовали в своих доказательствах Коши (1824) и Рунге (1905), он легко обобщается на многошаговые методы.

В обоих случаях оценим сначала локальные погрешности:

. (2.7.16)

Займемся теперь оценкой перенесенных погрешностей .

a) ***Теорема.***

Обозначим  окрестность точки , где  – точное решение уравнения

.

Пусть в  справедливы оценки локальных погрешностей (2.7.16) и выполнено одно из условий:

 или . (2.7.17)

Тогда имеет место следующая оценка глобальной погрешности (2.7.15):

, (2.7.18)

где ,



и  достаточно мало для того, чтобы численное решение оставалось в .

*Доказательство.*

При  оценка (2.7.18) переходит в .

. (2.7.19)

Подставляя в неравенство



выражение (2.7.18) с учетом (2.7.16) и принимая во внимание, что , приходим к такому неравенству:

.

Выражение в квадратных скобках мажорируется следующими интегралами:

, (2.7.20)

. (2.7.21)

Отсюда вытекает справедливость оценки (2.7.18).

b) При втором способе переноса погрешностей рассмотрим кроме (2.7.14) еще одно численное решение, значения которого в соседних узлах связаны равенством

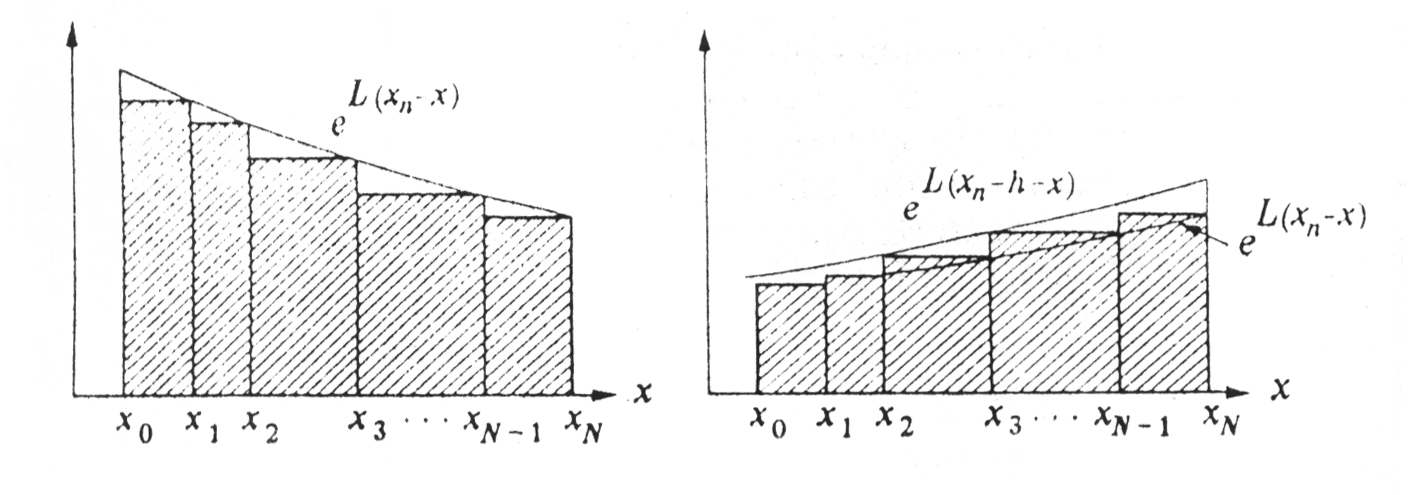
.

Оценим норму разности  через . Для  формулы метода Рунге-Кутты запишем в следующих обозначениях:



Вычитая из этих формул соответствующие формулы (2.3.1), получим для норм разностей такие оценки:





*Оценивание римановых сумм методом a) и b)*

Пусть  – постоянная Липшица для функции  и пусть . Тогда функция приращения  для метода (2.3.1) удовлетворяет неравенству

, (2.7.22)

где

. (2.7.23)

Из (2.7.22) получаем искомую оценку:

, (2.7.24)

и с её помощью оценку перенесенных погрешностей вместо оценки (2.7.19).

Предположим, что для начальных значений, лежащих на точном решении, локальная погрешность удовлетворяет оценке

 (2.7.25)

и что в окрестности решения функция приращения  удовлетворяет неравенству

. (2.7.26)

Тогда для глобальной погрешности (2.7.15) справедлива следующая оценка:

, (2.7.27)

где .

**1.8 Оптимальный выбор шага**

Предположим, что при интегрировании от точки до точки с шагом погрешность приближенно равна . Так как это соответствует росту погрешности со скоростью, приблизительно равной , то , где – функция, определяющая шаг. Положим и получим оценку интеграла, который приближенно равен полной погрешности:



С другой стороны, затраты будут пропорциональны числу шагов, которое приближенно равно



Методами вариационного исчисления можно показать, что если мы хотим минимизировать затраты при некотором фиксированном значении погрешности , то следует сохранять постоянной величину . Это означает, что окончательная погрешность должна быть одинаковой на каждом шаге.



В современных программах[[4]](#footnote-4), реализующих методы Рунге-Кутты, обязательно используется некоторый алгоритм автоматического изменения шага интегрирования. Интуитивно ясно, что на участках плавного изменения решения счет можно вести с достаточно крупным шагом. В то же время, на тех участках, где происходят резкие изменения поведения решения, необходимо выбирать мелкий шаг интегрирования. Обычно начальное значение шага задет пользователь или оно определено в программе. Далее шаг интегрирования изменяется в соответствии с величиной, получаемой в ходе вычисления оценки локальной погрешности.

Существует достаточно много способов оценки локальной погрешности, среди которых так называемое правило Рунге. Однако в моей программе я использовал самый простой и в то же время эффективный способ оценки локальной погрешности, который описан в разделе 3.1. «Описание программы Ilya RK-4 версия 1.43». Этот метод базируется на удвоении или делении пополам длины шага в зависимости от отношения локальной погрешности и максимально локальной допустимой погрешности .

# 2. Практическая часть

**2.1 Описание программы «Ilya RK-4 версия 1.43»**

## Программа для нахождения интегральной кривой, удовлетворяющей поставленной задаче Коши написана на языке высокого уровня *Borland C++ 3.1.* Программа состоит из четырех функций. При помощи директив препроцессора #define определены максимальный шаг и величина локальной максимальной погрешности, а также номер версии программы. Рассмотрим подробнее работу программы в комплексе.

Функция **title()** предназначена для печати на экране названия программы.

Функция **do\_step()** совершает один шаг Рунге-Кутты и возвращает полученное значение. В качестве входных параметров в нее передается текущее положение, значение искомой функции, вычисленное на предыдущем шаге и величина шага, с которым требуется произвести шаг.

Функция **f()** задает правую часть дифференциального уравнения, левая часть дифференциального уравнения равна . В качестве аргументов функции передается  и .

Функция **main()** – основная функция программы. У пользователя запрашивается точка, начиная с которой необходимо отобразить решение задачи Коши, точка, правая граница интегрирования и значение в левой точке, через которое обязана проходить искомая интегральная кривая. После этого программа начинает вычислительный процесс, выводя полученные значения на экран в виде списка и в текстовый файл *“rk4.txt”* на диск. После того, как будет достигнута правая граница интегрирования, процесс остановится и пользователю будет предложено нажать на любую клавишу для того, чтобы построить график. Для построения графика программа переключается в графический режим. График масштабируется с учетом того, чтобы он всегда был виден на экране вне зависимости от того, как высоко или низко от оси абсцисс он лежит. Кроме того, программа постарается сделать его максимально наглядным. Для этого будут проведены пунктирные линии, соответствующие минимальному и максимальному значению интегральной кривой, левому и правому концам интегрирования, а также значению интегральной кривой в указанной точке . Для того, чтобы пользователь мог легко ориентироваться на графике, рядом с пунктирными линиями пишутся координатные значения с точностью до двух десятичных знаков. Как показали многочисленные тесты, проведенные на компьютере на базе процессора Intel Pentium 4B с тактовой частотой 2.4 ГГц, построение графика происходит значительно быстрее, чем первичный расчет с выводом на экран и записью в файл. В этом легко убедиться[[5]](#footnote-5), если задать довольно большой отрезок интегрирования, например [-100,100].

Программа применяет следующий метод Рунге-Кутты четвертого порядка.



## Для получения более точных результатов и возможно даже увеличения скорости работы в определенных ситуациях программа использует автоматическое управление длиной шага. Производятся одна итерация с шагом , а затем две итерации подряд с шагом , начиная с позиции x\_cur. Разность между первым и последним упомянутым здесь результатами, взятая по модулю, считается локальной погрешностью. Если она больше, чем наперед заданная величина , то шаг делится пополам и тело цикла повторяется.

Для того, чтобы программа могла «разгоняться» после уменьшения шага, предусмотрено условие увеличения длины шага. Оно состоит в том, что если погрешность между вторым из двух значений, вычисленных с шагом  и значением, вычисленным с шагом , не превосходит , то шаг увеличивается вдвое и тело цикла повторяется. Отметим, что величина шага не может превосходить значения MAXSTEP, которое определяется директивой препроцессора #define. Если ни одно из двух описанных выше условий не выполняется, это означает, что шаг оптимален и программа производит вычисление значения функции с записью его в файл и отображением на экране.

Программа снабжена механизмом защиты от сбоев – в случае, если интегрируемая функция терпит разрыв (ее нельзя интегрировать на данном участке), программа останавливается и выдается сообщение о невозможности продолжать. Работоспособность этого механизма проверена на некоторых разрывных функциях, таких как тангенс и др.

# Заключение

В работе детально рассмотрен метод Рунге-Кутты четвертого порядка с автоматическим выбором длины шага, приведены необходимые теоретические сведения, освещены альтернативные методы и их эффективность.

Был разработан алгоритм программного модуля, позволяющий автоматически менять величину шага интегрирования при решении задачи Коши в зависимости от требуемой точности, что является непременным требованием, предъявляемым ко всем хорошим современным программам данного класса, написано приложение, решены примеры.

# Список использованных источников

[1]. Амоносов А.А., Дубинский Ю.А., Копченова Н.В. «Вычислительные методы для инженеров», М., Высшая школа, 1994, 544с.

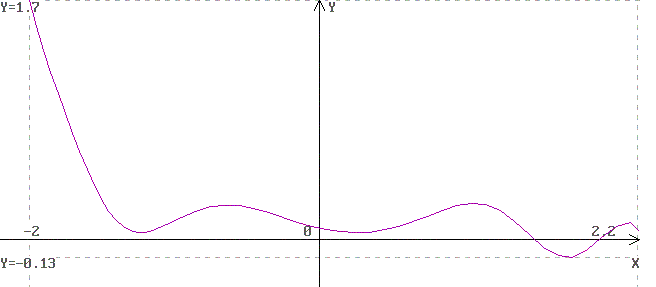
[2]. Хайрер Э., Нёрсетт С., Ваннер Г. «Решение обыкновенных дифференциальных уравнений. Нежесткие задачи», М., Мир, 1990, 512с.

[3]. Холл Д., Уатт Д. «Современные численные методы решения обыкновенных дифференциальных уравнений», М., Мир, 1979, 312с.

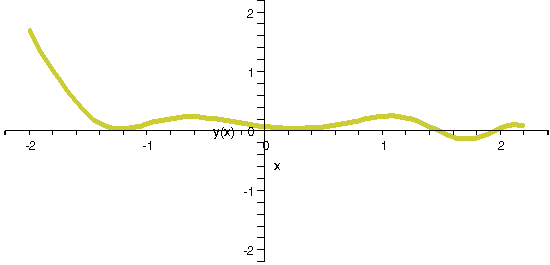
# Приложение А. Графики функций

В данном приложении рассмотрены три дифференциальных уравнения первого порядка. К каждому уравнению прилагается по два графика – первый из них построен созданным приложением, а второй создан в пакете *Maple 9.01.*



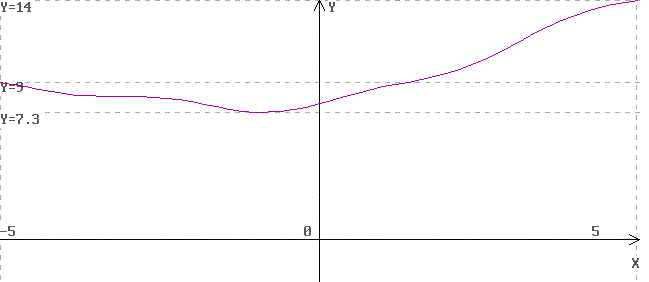


*Интегральная кривая, построенная приложением «Ilya RK-4 версия 1.43»*

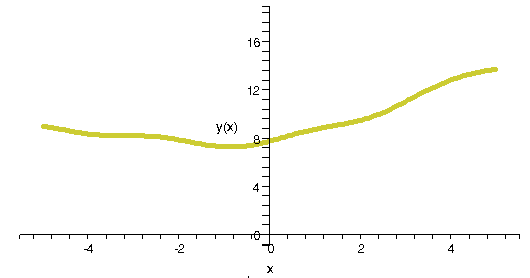


*Интегральная кривая, построенная математическим пакетом Waterloo Maple 9.01*

# 

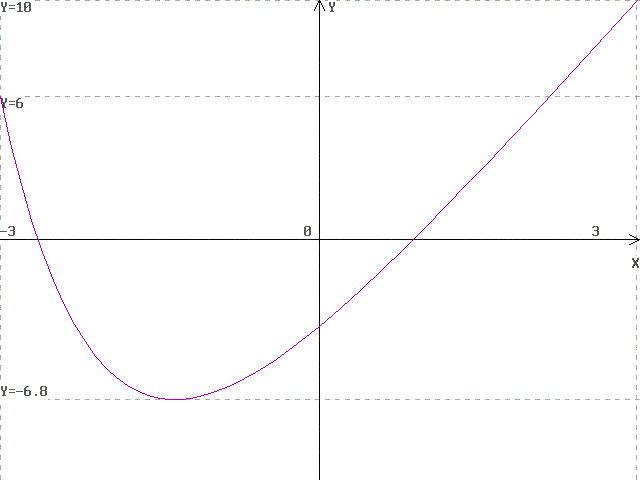


*Интегральная кривая, построенная приложением «Ilya RK-4 версия 1.43»*

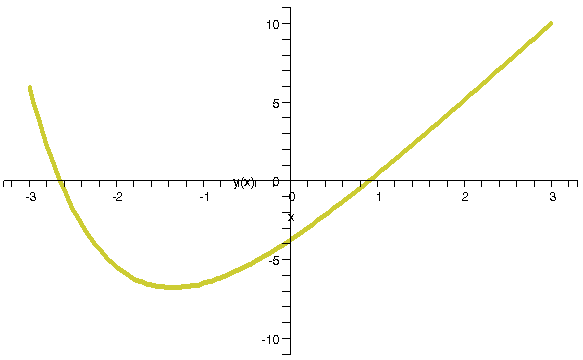


*Интегральная кривая, построенная математическим пакетом Waterloo Maple 9.01*

**

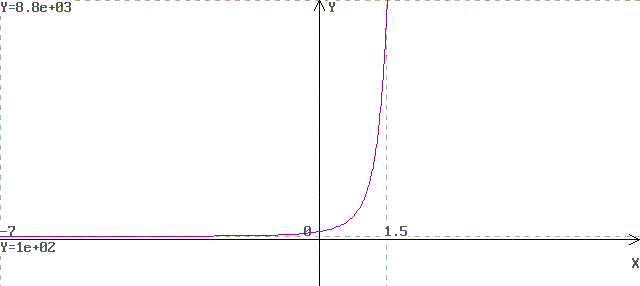


*Интегральная кривая, построенная приложением «Ilya RK-4 версия 1.43»*

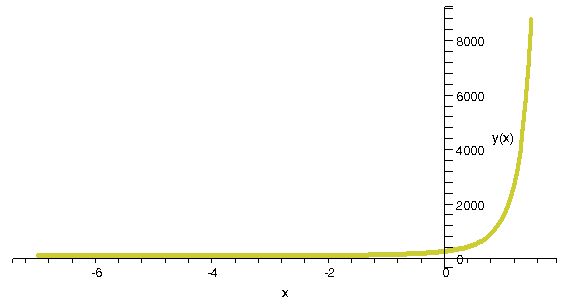


*Интегральная кривая, построенная математическим пакетом Waterloo Maple 9.01*





*Интегральная кривая, построенная приложением «Ilya RK-4 версия 1.43»*



*Интегральная кривая, построенная математическим пакетом Waterloo Maple 9.01*

**Приложение Б.**

# Пример таблицы значений функции y(x)

y(-7)=100, h=0.4

y(-6.6)=100.045, h=0.4

y(-6.2)=100.112, h=0.4

y(-5.8)=100.212, h=0.4

y(-5.4)=100.361, h=0.4

y(-5)=100.585, h=0.4

y(-4.6)=100.919, h=0.4

y(-4.2)=101.419, h=0.4

y(-3.8)=102.17, h=0.4

y(-3.4)=103.301, h=0.2

y(-3.2)=104.067, h=0.2

y(-3)=105.011, h=0.2

y(-2.8)=106.175, h=0.2

y(-2.6)=107.615, h=0.2

y(-2.4)=109.4, h=0.2

y(-2.2)=111.62, h=0.2

y(-2)=114.392, h=0.2

y(-1.8)=117.873, h=0.2

y(-1.6)=122.267, h=0.2

y(-1.4)=127.857, h=0.2

y(-1.2)=135.033, h=0.2

y(-1)=144.346, h=0.2

y(-0.8)=156.596, h=0.2

y(-0.6)=172.977, h=0.1

y(-0.5)=183.256, h=0.1

y(-0.4)=195.327, h=0.1

y(-0.3)=209.595, h=0.1

y(-0.2)=226.578, h=0.1

y(-0.1)=246.953, h=0.05

y(-0.05)=258.68, h=0.05

y(2.96985e-15)=271.608, h=0.05

y(0.05)=285.897, h=0.05

y(0.1)=301.73, h=0.05

y(0.15)=319.32, h=0.05

y(0.2)=338.919, h=0.05

y(0.25)=360.821, h=0.05

y(0.3)=385.374, h=0.05

y(0.35)=412.989, h=0.05

y(0.4)=444.156, h=0.05

y(0.45)=479.459, h=0.025

y(0.475)=498.877, h=0.025

y(0.5)=519.603, h=0.025

y(0.525)=541.747, h=0.025

y(0.55)=565.433, h=0.025

y(0.575)=590.794, h=0.025

y(0.6)=617.978, h=0.025

y(0.625)=647.149, h=0.025

y(0.65)=678.489, h=0.025

y(0.675)=712.199, h=0.025

y(0.7)=748.501, h=0.025

y(0.725)=787.645, h=0.025

y(0.75)=829.906, h=0.025

y(0.775)=875.592, h=0.025

y(0.8)=925.047, h=0.025

y(0.825)=978.656, h=0.025

y(0.85)=1036.85, h=0.025

y(0.875)=1100.11, h=0.025

y(0.9)=1168.98, h=0.025

y(0.925)=1244.07, h=0.0125

y(0.9375)=1284.16, h=0.0125

y(0.95)=1326.08, h=0.0125

y(0.9625)=1369.91, h=0.0125

y(0.975)=1415.77, h=0.0125

y(0.9875)=1463.78, h=0.0125

y(1)=1514.04, h=0.0125

y(1.0125)=1566.7, h=0.0125

y(1.025)=1621.89, h=0.0125

y(1.0375)=1679.75, h=0.0125

y(1.05)=1740.44, h=0.0125

y(1.0625)=1804.13, h=0.0125

y(1.075)=1871, h=0.0125

y(1.0875)=1941.23, h=0.0125

y(1.1)=2015.04, h=0.0125

y(1.1125)=2092.63, h=0.0125

y(1.125)=2174.24, h=0.0125

y(1.1375)=2260.12, h=0.0125

y(1.15)=2350.54, h=0.0125

y(1.1625)=2445.79, h=0.0125

y(1.175)=2546.16, h=0.0125

y(1.1875)=2652, h=0.0125

y(1.2)=2763.65, h=0.0125

y(1.2125)=2881.49, h=0.0125

y(1.225)=3005.94, h=0.0125

y(1.2375)=3137.43, h=0.0125

y(1.25)=3276.43, h=0.0125

y(1.2625)=3423.46, h=0.0125

y(1.275)=3579.07, h=0.0125

y(1.2875)=3743.83, h=0.0125

y(1.3)=3918.41, h=0.0125

y(1.3125)=4103.47, h=0.0125

y(1.325)=4299.77, h=0.0125

y(1.3375)=4508.11, h=0.0125

y(1.35)=4729.35, h=0.00625

y(1.35625)=4845.11, h=0.00625

y(1.3625)=4964.45, h=0.00625

y(1.36875)=5087.5, h=0.00625

y(1.375)=5214.41, h=0.00625

y(1.38125)=5345.3, h=0.00625

y(1.3875)=5480.34, h=0.00625

y(1.39375)=5619.66, h=0.00625

y(1.4)=5763.44, h=0.00625

y(1.40625)=5911.83, h=0.00625

y(1.4125)=6065, h=0.00625

y(1.41875)=6223.14, h=0.00625

y(1.425)=6386.44, h=0.00625

y(1.43125)=6555.09, h=0.00625

y(1.4375)=6729.28, h=0.00625

y(1.44375)=6909.25, h=0.00625

y(1.45)=7095.2, h=0.00625

y(1.45625)=7287.36, h=0.00625

y(1.4625)=7485.99, h=0.00625

y(1.46875)=7691.33, h=0.00625

y(1.475)=7903.64, h=0.00625

y(1.48125)=8123.19, h=0.00625

y(1.4875)=8350.28, h=0.00625

y(1.49375)=8585.2, h=0.00625

y(1.5)=8828.27, h=0.00625

# Приложение В.

# Листинг программы «Ilya RK-4 версия 1.43»

// ----------------------------------------------------------------------- //

#include <dos.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <graphics.h>

#include <stdlib.h>

#define EPSILON 0.00001

#define MAXSTEP 1

#define VERSION 1.43

// ----------------------------------------------------------------------- //

double f(double x, double y);

double do\_step(double h, double x\_cur, double y\_cur);

void title(void);

void main(void);

// ----------------------------------------------------------------------- //

double f(double x, double y)

{

// Правая часть ДУ f(x,y)

return (pow(2.718,x)\*y);

}

// ----------------------------------------------------------------------- //

void main(void)

{

int i; // Вспомогательный счетчик

int metka; // Метка на осях

int flag = 0; // Флаг правого конца интегрирования

int metka1, metka2; // Переменные меток на осях координат

double err = 0; // Погрешность

double x0, y0; // Координаты точки начального условия

double big2\_step\_res, super\_step\_res; // Результаты длинных шагов

double k = 1; // Коэффициент коррекции

double zoom = 1; // Масштаб на графике

double big\_step\_res, small\_step\_res; // Результаты шагов интегрирования

double a, b; // Границы

double temp; // Переменная для служебных нужд

double x\_cur, y\_cur; // Переменные метода РК

double h; // Шаг интегрирования

double f\_max = 0, f\_min = 0; // Максимальное и минимальное значение кривой

double norma = 0; // Норма (для корректного масштабирования графика)int c = 8; // Переменная цвета разделительных линий

FILE \*myfile; // Указатель на текстовый файл с таблицей значений

// Инициализируем графический адаптер

int gdriver = DETECT, gmode, errorcode;

initgraph(&gdriver, &gmode, "");

errorcode = graphresult();

if (errorcode != grOk)

{

printf("Ошибка инициализации графики: %s\n", grapherrormsg(errorcode));

getch();

}

textcolor(0);

setbkcolor(0);

title();

printf("y'=f(x,y), y(x0)=y(a)=y0, [a,b] - отрезок интегрирования\n");

label1: printf("\na=");

scanf("%lg", &a);

printf("b=");

scanf("%lg", &b);

// Авто смена границ при необходимости

if (a > b)

{

temp = a;

a = b;

b = temp;

}

if (a == b)

{

printf("Начало отрезка интегрирования совпадает с его концом, повторите ввод!\n");

goto label1;

}

printf("y(%lg)=", a);

scanf("%lg", &y0);

title();

printf("[%lg,%lg] - границы интегрирования, y(%lg)=%lg - начальное условие.\n", a, b, a, y0);

// Инициализация

h = fabs(b - a) / 10;

if (h > 0.1) h = 0.1;

x\_cur = a;

y\_cur = y0;

f\_max = y\_cur;

f\_min = y\_cur;

myfile = fopen("rk4.txt", "w");

fprintf(myfile, "Program: Ilya RK4 Version %g\n", VERSION);

fprintf(myfile, "Method: Runge-Kutta\n");

fprintf(myfile, "The order of method: 4\n");

fprintf(myfile, "Automatic integration step select: Enabled\n");

fprintf(myfile, "[a,b]=[%lg,%lg], y(%lg)=%lg\n", a, b, a, y0);

while (x\_cur <= b)

{

if (flag > 1) break;

big\_step\_res = do\_step(h, x\_cur, y\_cur);

temp = do\_step(h / 2, x\_cur, y\_cur);

small\_step\_res = do\_step(h / 2, x\_cur + h / 2, temp);

err = fabs(big\_step\_res - small\_step\_res);

// Уменьшение длины шага

if (err > EPSILON)

{

h = h / 2;

continue;

}

// Увеличение длины шага

big2\_step\_res = do\_step(h, x\_cur + h, big\_step\_res);

super\_step\_res = do\_step(2 \* h, x\_cur, y\_cur);

if (fabs(big2\_step\_res - super\_step\_res) < EPSILON / 2)

{

h \*= 2;

continue;

}

if (h > MAXSTEP) h = MAXSTEP;

// Защита от сбоев

if (h < pow(EPSILON, 2))

{

printf("Ошибка! Возможно, функция разрывна.\nПроинтегрировать на данном интервале невозможно. Скорее всего, g(%lg)=", x\_cur);

fprintf(myfile, "Ошибка! Возможно, функция разрывна.\nПроинтегрировать на данном интервале невозможно. Скорее всего, g(%lg)=", x\_cur);

if (y\_cur < 0)

{

printf("-oo.\n");

fprintf(myfile, "-oo.\n");

}

else

{

printf("+oo.\n");

fprintf(myfile, "+oo.\n");

}

getch();

fclose(myfile);

exit(1);

}

printf("y(%lg)=%lg, err=%lg, h=%lg\n", x\_cur, y\_cur, err, h);

if (y\_cur < f\_min) f\_min = y\_cur;

if (y\_cur > f\_max) f\_max = y\_cur;

fprintf(myfile, "y(%lg)=%lg, h=%lg\n", x\_cur, y\_cur, h);

if (x\_cur + h > b) h = fabs(b - x\_cur);

x\_cur += h;

y\_cur = big\_step\_res;

if (x\_cur >= b) flag++;

}

fclose(myfile);

printf("\nТаблица значений записана в файл rk4.txt.\n");

printf("\nНажмите любую клавишу для построения графика...");

flag = 0;

getch();

// Построение графика

cleardevice(); clrscr();

if (fabs(a) > fabs(b)) zoom = fabs(getmaxx() / 2 / a);

else zoom = fabs(getmaxx() / 2 / b);

// Рисуем границы

for (i = 0 ; i < getmaxy() ; i += 5)

{

if (c == 8) c = 0;

else c = 8;

setcolor(c);

line(a \* zoom + getmaxx() / 2, i, a \* zoom + getmaxx() / 2, i + 5);

line(b \* zoom + getmaxx() / 2 - 1, i, b \* zoom + getmaxx() / 2 - 1, i + 5);

}

if (fabs(f\_min) > fabs(f\_max)) norma = fabs(f\_min) \* zoom;

else norma = fabs(f\_max) \* zoom;

// Определение коэффициента коррекции

k = (getmaxy() / 2) / norma;

// Предотвращение чрезмерного масштабирования

if (k < 0.0001) k = 0.0001;

if (k > 10000) k = 10000;

for (i = 0 ; i < getmaxx() ; i += 5)

{

if (c == 8) c = 0;

else c = 8;

setcolor(c);

line(i, -y0 \* zoom \* k + getmaxy() / 2, i + 5, -y0 \* zoom \* k + getmaxy() / 2);

line(i, -f\_min \* zoom \* k + getmaxy() / 2, i + 5, -f\_min \* zoom \* k + getmaxy() / 2);

line(i, -f\_max \* zoom \* k + getmaxy() / 2, i + 5, -f\_max \* zoom \* k + getmaxy() / 2);

}

metka = ceil((-y0 \* zoom \* k + getmaxy() / 2) / 16);

if (metka <= 0) metka = 1;

if (metka == 15) metka = 16;

if (metka > 25) metka = 25;

gotoxy(1, metka);

printf("Y=%.2g", y0, metka);

metka = ceil((-f\_max \* zoom \* k + getmaxy() / 2) / 16);

if (metka <= 0) metka = 1;

if (metka == 15) metka = 16;

if (metka > 25) metka = 25;

gotoxy(1, metka);

printf("Y=%.2lg", f\_max, metka);

metka = ceil((-f\_min \* zoom \* k + getmaxy() / 2) / 16);

if (metka <= 0) metka = 1;

if (metka == 15) metka = 16;

if (metka > 25) metka = 25;

gotoxy(1, metka);

printf("Y=%.2lg", f\_min, metka);

// Пишем границы, делаем отметки на осях координат

metka1 = ceil((a \* zoom + getmaxx() / 2) / 8);

if (metka1 < 1) metka1 = 1;

if (metka1 > 75) metka1 = 75;

if (metka == 17) metka = 18;

gotoxy(metka1, 15);

if (a != 0) printf("%.2lg", a);

metka2 = ceil((b \* zoom + getmaxx() / 2 - 1) / 8);

if (metka2 - metka1 < 7) metka2 = metka1 + 7;

if (metka2 < 1) metka2 = 1;

if (metka2 > 75) metka2 = 75;

gotoxy(metka2, 15);

printf("%.2lg", b);

gotoxy(80, 17);

printf("X");

gotoxy(42,1);

printf("Y");

gotoxy(39, 15);

printf("0");

// Рисуем систему координат

setcolor(15);

line(0, getmaxy() / 2, getmaxx(), getmaxy() / 2);

line(getmaxx() / 2, 0, getmaxx() / 2, getmaxy());

line(getmaxx() / 2, 0, getmaxx() / 2 - 5, 10);

line(getmaxx() / 2, 0, getmaxx() / 2 + 5, 10);

line(getmaxx(), getmaxy() / 2, getmaxx() - 10, getmaxy() / 2 + 5);

line(getmaxx(), getmaxy() / 2, getmaxx() - 10, getmaxy() / 2 - 5);

setcolor(10);

h = fabs(b - a) / 10;

if (h > 0.1) h = 0.1;

y\_cur = y0;

x\_cur = a;

f\_max = y\_cur;

f\_min = y\_cur;

x0 = zoom \* a + getmaxx() / 2;

y0 = (zoom \* (-y\_cur)) \* k + getmaxy() / 2;

while (x\_cur <= b)

{

if (flag > 1) break;

big\_step\_res = do\_step(h, x\_cur, y\_cur);

temp = do\_step(h / 2, x\_cur, y\_cur);

small\_step\_res = do\_step(h / 2, x\_cur + h / 2, temp);

err = fabs(big\_step\_res - small\_step\_res);

if (err > EPSILON)

{

h = h / 2;

continue;

}

big2\_step\_res = do\_step(h, x\_cur + h, big\_step\_res);

super\_step\_res = do\_step(2 \* h, x\_cur, y\_cur);

if (fabs(big2\_step\_res - super\_step\_res) < EPSILON / 2)

{

h \*= 2;

continue;

}

if (h > MAXSTEP) h = MAXSTEP;

line (x0, y0, zoom \* x\_cur + getmaxx() / 2, zoom \* (-y\_cur) \* k + getmaxy() / 2);

x0 = zoom \* (x\_cur) + getmaxx() / 2;

y0 = (zoom \* (-y\_cur)) \* k + getmaxy() / 2;

if (x\_cur + h > b) h = fabs(b - x\_cur);

x\_cur += h;

y\_cur = big\_step\_res;

if (x\_cur >= b) flag++;

}

while (getch() != 0);

}

// ----------------------------------------------------------------------- //

void title(void)

{

// Печать заголовка программы

cleardevice(); clrscr();

printf(" Решение дифференциальных уравнений методом Рунге-Кутты 4-го порядка\n");

printf(" с автоматическим выбором длины шага\n");

printf(" Разработал Щербаков Илья, гр. 520212, версия %g\n", VERSION);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

}

// ----------------------------------------------------------------------- //

double do\_step(double h, double x\_cur, double y\_cur)

{

double k1, k2, k3, k4, delta\_y\_cur;

k1 = f(x\_cur, y\_cur);

k2 = f(x\_cur + (h / 2), y\_cur + (h / 2) \* k1);

k3 = f(x\_cur + (h / 2), y\_cur + (h / 2) \* k2);

k4 = f(x\_cur + h, y\_cur + h \* k3);

delta\_y\_cur = (h / 6) \* (k1 + 2 \* k2 + 2 \* k3 + k4);

return(y\_cur + delta\_y\_cur);

}

// ----------------------------------------------------------------------- //

1. Дж. Холл, Дж. Уатт «Современные численные методы решения обыкновенных дифференциальных уравнений», М., Мир, 1979, стр. 77. [↑](#footnote-ref-1)
2. «Между тем еще нет доказательства, что эти приближенные методы сходятся, или, что практически важнее, нет критерия, определяющего, сколь малым надо сделать шаги, чтобы достичь предписанной точности» – так писал Рунге в 1905 году. [↑](#footnote-ref-2)
3. Хайрер Э., Нёрсетт С., Ваннер Г. «Решение обыкновенных дифференциальных уравнений. Нежесткие задачи», М., Мир, 1990, стр. 169. [↑](#footnote-ref-3)
4. Амоносов А.А., Дубинский Ю.А., Копченова Н.В. «Вычислительные методы для инженеров», М., Высшая школа, 1994, стр. 445. [↑](#footnote-ref-4)
5. Тестирование проводилось на компьютере на базе процессора Intel Pentium 4B. На компьютерах, оснащенных другими процессорами, время выполнения первого и второго этапов может быть другим. [↑](#footnote-ref-5)