Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Факультет компьютерные науки и прикладная математика

Курсовой проект

по курсу «Математический практикум»

3 семестр

Вариант 18

Руководитель

Беляков Д.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (дата)

Студент

Группа М80-213Б-21

Соломатина С.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (дата)

(оценка)

**Оглавление**

[Введение 2](#_heading=h.30j0zll)

[Метод половинного деления 3](#_heading=h.1fob9te)

[Метод простой итерации и Ньютона для решения нелинейных уравнений 4](#_heading=h.2et92p0)

[Метод простых итераций и Зейделя для решения СЛАУ 5](#_heading=h.4d34og8)

[Метод простых итераций и Ньютона для решения СНАУ 7](#_heading=h.26in1rg)

[Вычисление параметров линейной и квадратичной регрессии. Построение полиномов Лагранжа и Ньютона для табличной функции 10](#_heading=h.1ksv4uv)

[Вычисление приближённого занчения интеграла 11](#_heading=h.44sinio)

[Вычисление дифференциальных уравнений методами Эйлера-Коши и Рунге-Кутта 4-го порядка 15](#_heading=h.z337ya)

[Заключение 16](#_heading=h.1pxezwc)

## Введение

В данном курсовом проекте представлены программы, исполненные в пакете прикладных программ MATLAB для решения задач варианта 18. Также представлены методы решения задач вычислительной математики, предоставлены теоретические выкладки, а в задачах, где это возможно, графики решений и аналитические решения. Протоколы программ прикреплены в приложениях.

## Метод половинного деления

**Задача 1:**

Отделить корни уравнения и найти его методом половинного деления с точностью .

Метод дихотомии, или метод половинного деления, заключается в делении отрезка пополам и его сужении в два раза на каждом шаге итерационного процесса в зависимости от знака выражения . Это связано с тем, что, если на данном отрезке существует хотя бы один корень уравнения, то значения функции на концах этого отрезка имеют разные знаки: <0.

Условием остановы итерационного процесса будет выполнение неравенства .

Новый интервал вычисляется по формулам

, =, если ;

или по формулам

=, , если .

Приближенное значение корня к моменту окончания итерационного процесса вычисляется по формуле

## Метод простой итерации и Ньютона для решения нелинейных уравнений

**Задача 2:**

Отделить корни уравнения, найти отрезок, на котором лежит один из корней. Найти отличный от нуля корень уравнения с четырьмя верными знаками после запятой методами простой итерации и Ньютона:

**Метод простых итераций**

Идея метода итераций заключается в замене исходного уравнения равносильным уравнением вида с выделенным линейным членом.

Достаточное условие сходимости метода:. Это условие необходимо проверить перед началом решения задачи, так как функция может быть выбрана неоднозначно, причем в случае неверного выбора указанной функции метод расходится.

Начальное приближение корня вычисляется по формуле: ; итерационный процесс задаёт формула , условие его окончания:

**Метод Ньютона**

Метод Ньютона, или метод касательных, заключается в том, что если

− некоторое приближение к корню уравнения , то следующее приближение определяется как корень касательной к функции , проведённой в точке . Найдём формулу для нахождения следующего приближения методом Ньютона из исходного уравнения :

- Уравнение касательной имеет вид

- Положим .

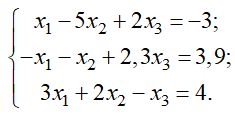
- Тогда

Кроме того, для начала вычислений требуется задание начального приближения удовлетворяющего условию сходимости метода к корню . Вычисления производятся, пока

## Метод простых итераций и Зейделя для решения СЛАУ

**Задача 3:**

Решить систему линейных уравнений методами простых итераций и Зейделя с точностью



Пусть задана система уравнений

Выразим через остальные члены *i*-го уравнения:

Полученная запись СЛАУ приводит к двум итерационным процессам:

1. Метод простой итерации:
2. Метод Зейделя:

При этом задаётся, где ; *k −* номер итерации.

Процесс ведётся до выполнения условия

Достаточный признак сходимости обоих методов состоит в выполнении условия диагонального преобладания:

.Проверим выполнение сходимости для данной системы уравнений (диагональное преобладание):

1. е уравнение: －не выполняется
2. е уравнение: －не выполняется
3. е уравнение: －не выполняется

Таким образом, условие диагонального преобладания для исходной системы уравнений не выполнено, поэтому не может быть гарантирована сходимость итерационных методов к решению. Для данной системы можно добиться выполнения этого условия следующей перестановкой:

## Метод простых итераций и Ньютона для решения СНАУ

Задача 4:

Найдите численное решение системы уравнений методом Ньютона с точностью . Найдите решение системы с той же точностью простыми итерациями. Сравните способы решения (точность, количество итераций).

**Метод простых итераций**

Метод простых итераций применяется в тех случаях, когда каждая из функций  может быть аналитически разрешена относительно , т.е. система уравнений приводима к виду

Алгоритм решения заключается в последовательном вычислении

Начиная с некоторого начального приближения, пока:

* будет выполнено условие , где - заданная точность расчета,
* зарегистрирован факт расхождения итерационного процесса;
* превышено заданное предельное число итераций.

Для сходимости итерационного процесса достаточно, чтобы вблизи истинных значений корней выполнялись неравенства

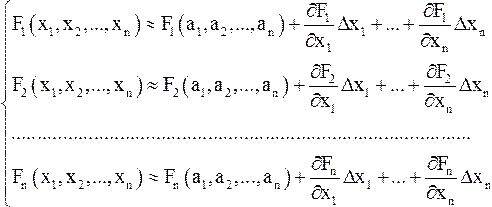
.

Метод простых итераций рекомендуется применять в тех случаях, когда исходная система легко преобразуется к виду = и известно хорошее начальное приближение.

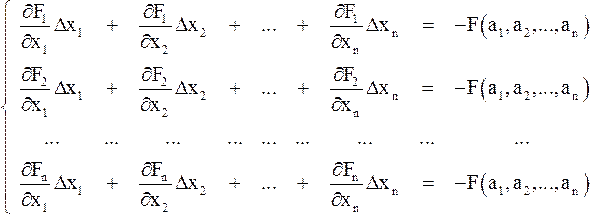
Необходим глубокий анализ вида системы в каждом отдельном случае с точки зрения сходимости метода, возможности получения желаемого решения с требуемой точностью. Как правило, метод последовательных приближений является, в лучшем случае, линейно сходящимся.

**Метод Ньютона**

В основе метода Ньютона для системы уравнений лежит использование разложения функций в ряд Тейлора, причем члены, содержащие вторые производные (и производные более высоких порядков), отбрасываются. Пусть приближенные значения неизвестных системы (например, полученные на предыдущей итерации) равны соответственно . Задача состоит в нахождении приращений (поправок) к этим значениям благодаря которым решение исходной системы запишется в виде: . Проведем разложение левых частей уравнений исходной системы в ряд Тэйлора, ограничиваясь лишь линейными членами относительно приращений:



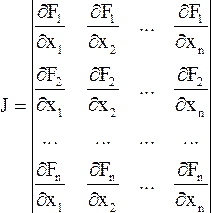
Поскольку левые части этих выражений должны обращаться в нуль, то можно приравнять к нулю и правые части:



в матричном виде: IMG_262

Значения и их производные вычисляются при .

Определителем последней системы является якобиан:

.

Для существования единственного решения системы якобиан должен быть отличным от нуля на каждой итерации.

Таким образом, итерационный процесс решения системы нелинейных уравнений методом Ньютона состоит в определении приращений IMG_267IMG_268к значениям неизвестных на каждой итерации. Счет прекращается, если все приращения становятся малыми по абсолютной величине:

IMG_269.

В методе Ньютона также важен удачный выбор начального приближения для обеспечения хорошей сходимости. Сходимость ухудшается с увеличением числа уравнений системы. Итак, за расчетную формулу примем

IMG_270или IMG_271.

Критерий окончания. Будем считать, что заданная точность достигнута, если IMG_299или IMG_300.

## Вычисление параметров линейной и квадратичной регрессии. Построение полиномов Лагранжа и Ньютона для табличной функции

**Задача 5:**

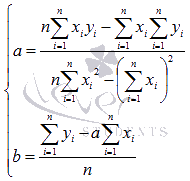
Определить параметры линейной регрессии *y = kx + b* и квадратичной регрессии методом наименьших квадратов. Построить для заданной табличной функции полиномы Лагранжа и Ньютона и упростить их.

|  | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10.2 | 10.37 | 10.5 | 10.6 | 10.76 | 10.8 | 10.9 | 11 | 11.1 | 11.2 |

**Метод наименьших квадратов**

Суть метода заключается в нахождении коэффициентов линейной зависимости, при которых функция двух переменных а и b

принимает наименьшее значение. То есть, при данныха и b сумма квадратов отклонений экспериментальных данных от найденной прямой будет наименьшей. Таким образом, решение примера сводится к нахождению экстремума функции двух переменных.

формулы для нахождения коэффициентов по методу наименьших квадратов (МНК). 

# 

# 

# 

# 

# 

# 

# 

# Интерполяционные полиномы Лагранжа и Ньютона

**Полином Лагранжа**

Полином , где сомножители задаются как называется интерполяционным полиномом Лагранжа степени N.

Для вычисления значения интерполяционного полинома в некоторой точке х, не прибегая к явному вычислению коэффициентов полинома, применяется схема Невилла. На первом этапе зададим значения . Далее значения вспомогательных полиномов в точке х определяются согласно рекуррентному соотношению, называющимся интерполяционным полиномом Лагранжа степени N.

и окончательно мы получаем

Для вычисления значения интерполяционного полинома в некоторой точке х, не прибегая к явному вычислению коэффициентов полинома, применяется схема Невилла. На первом этапе зададим значения

. Далее значения вспомогательных полиномов в точке х определяются согласно рекуррентному соотношению

и окончательно мы получаем .

**Полином Ньютона**

Пусть заданы различные точки и значения в этих точках Разделенные разности нулевого порядка есть просто значения :

Разделенные разности порядка k в точке вычисляются по следующим рекуррентным формулам:

Тогда, используя введенные обозначения, получим интерполяционный полином Ньютона, который записывается в виде:

Используя это представление, значение интерполяционного полинома Ньютона в некоторой точке х может быть вычислено простым и эффективным способом. Этот алгоритм называется схемой Горнера и задается следующими рекуррентными формулами:

;

Тогда требуемое значение полинома равно последнему элементу этой последовательности, т.е. .

## Вычисление приближенного значения интеграла

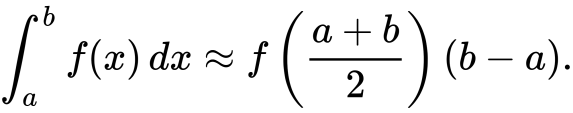
**Задача 6:**

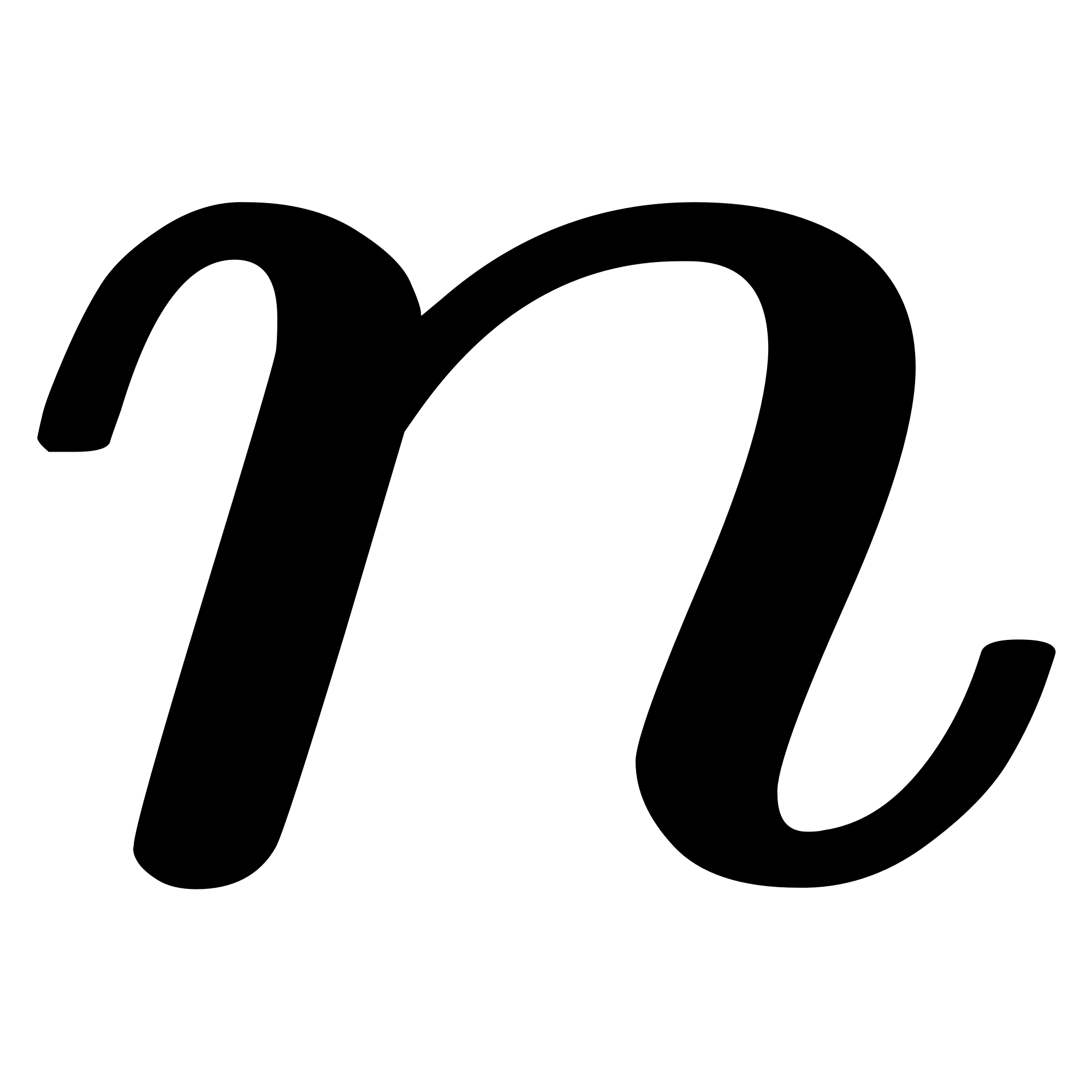
Вычислить интегралы по формулам прямоугольников, трапеций и Симпсона с точностью . Вычислить точное значение интеграла (если возможно) и найти отклонение точного значения от приближенного. В несобственном интеграле выполнить преобразование и особенность убрать.

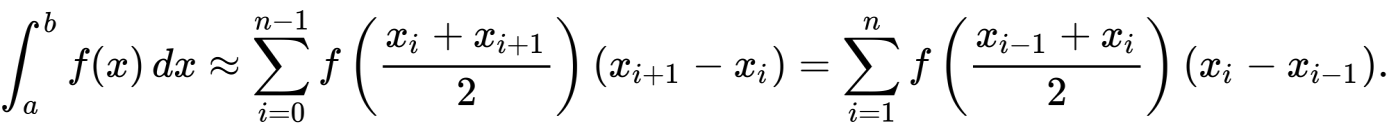
**Метод прямоугольников:**

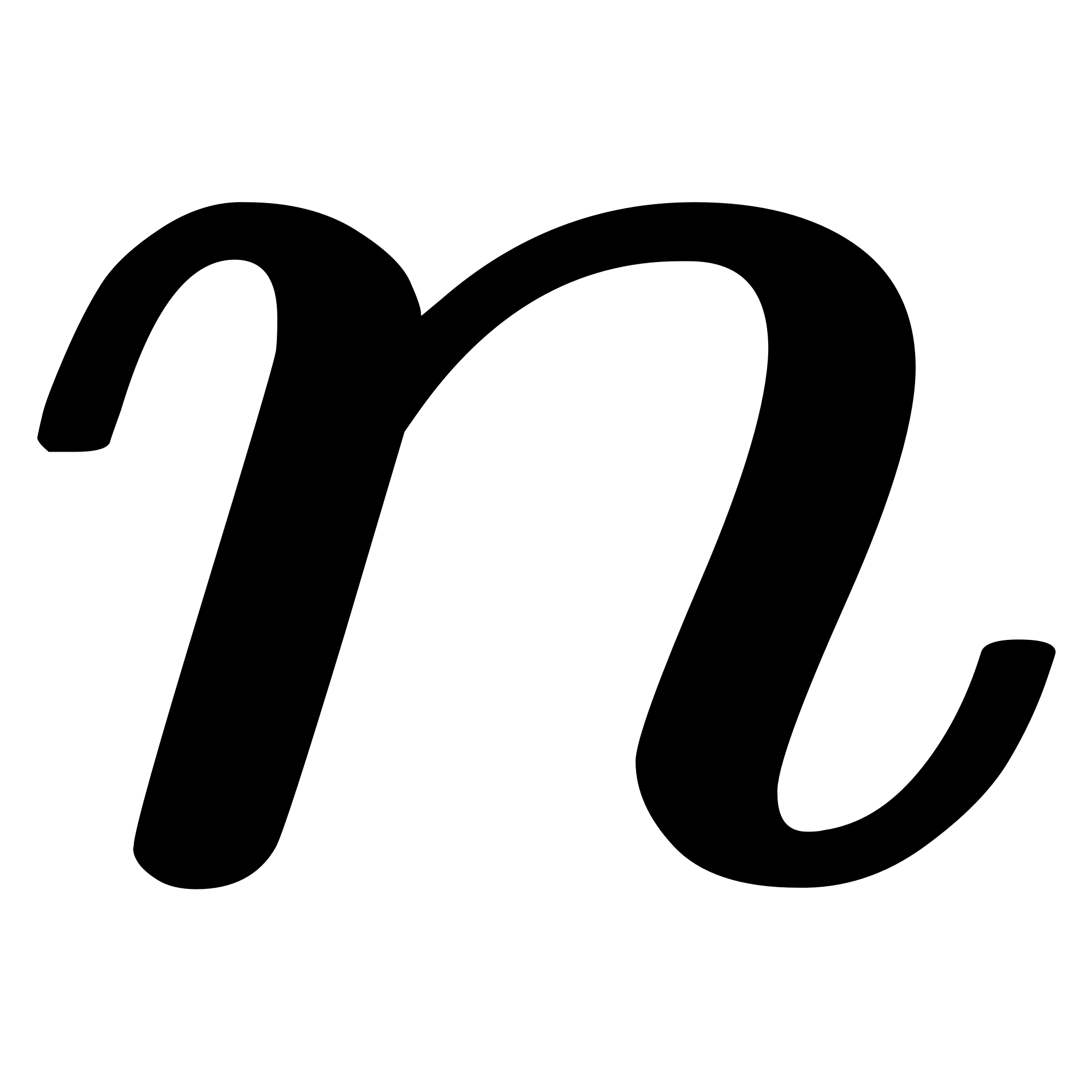
Метод прямоугольников — метод [численного интегрирования](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) функции одной переменной, заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть константу, на каждом элементарном отрезке. Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота — значением подынтегральной функции в этих узлах. [Алгебраический порядок точности](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0) равен 0, но для формулы средних прямоугольников равен 1, поэтому далее под методом прямоугольника будем иметь формулу средних прямоугольников.

Если отрезок  является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле:



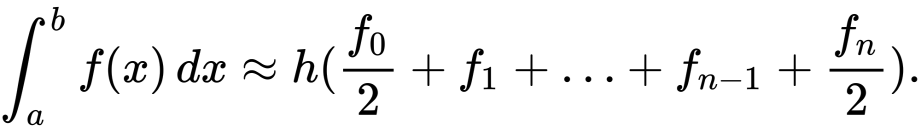
В случае разбиения отрезка интегрирования на  элементарных отрезков приведённая выше формула применяется на каждом из этих элементарных отрезков между двумя соседними узлами. В результате, получается составная квадратурная формула:



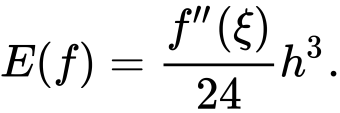
С увеличением  точность получаемого по приближённым формулам результата возрастает.

Равномерную сетку можно описать следующим набором формул:

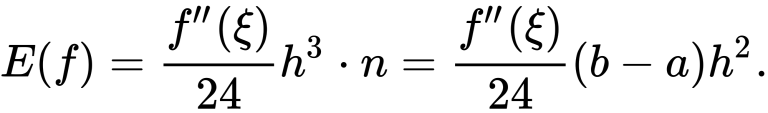
Для равномерных сеток формулу прямоугольников можно записать в виде следующей формулы Котеса, соответствующей формуле трапеций:



Погрешность для формулы прямоугольников составляет



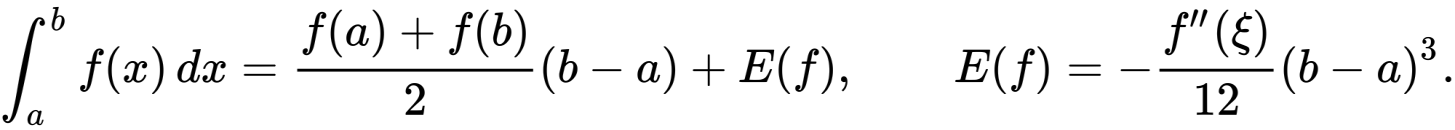
Погрешность для составной формулы прямоугольников:



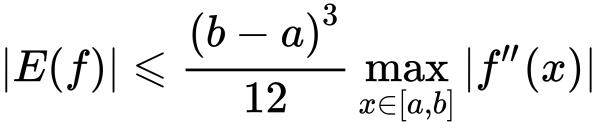
**Метод трапеций:**

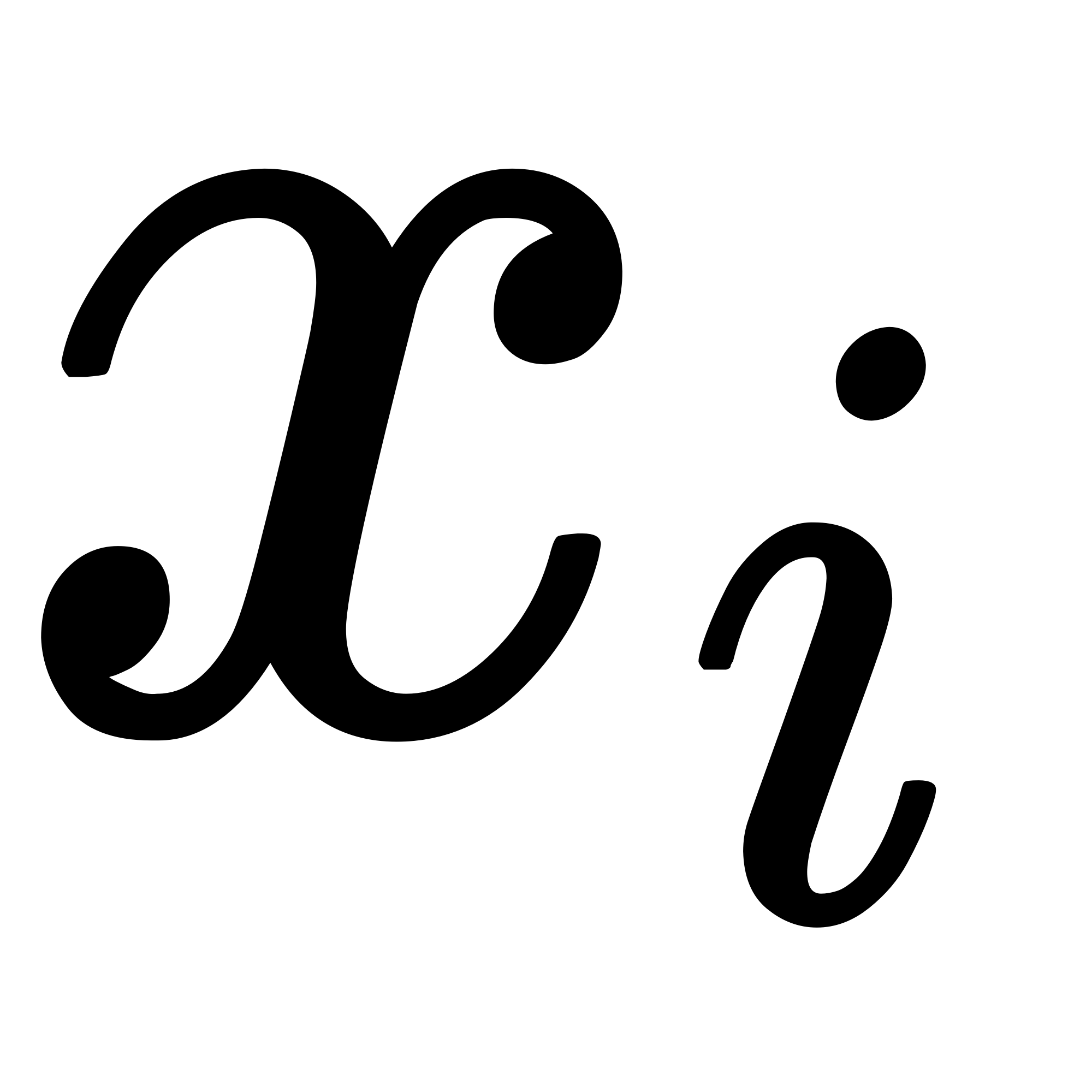
Метод трапеций — метод [численного интегрирования](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) функции одной переменной, заключающийся в замене на каждом элементарном отрезке подынтегральной функции на многочлен первой степени, то есть линейную функцию. Площадь под графиком функции аппроксимируется прямоугольными [трапециями](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BF%D0%B5%D1%86%D0%B8%D1%8F). [Алгебраический порядок точности](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0) равен 1.

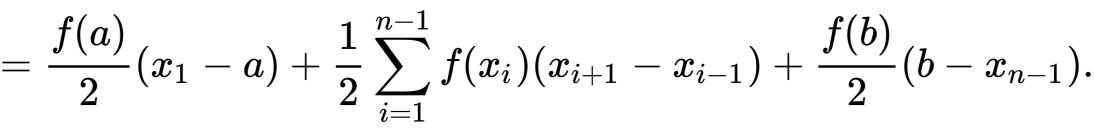
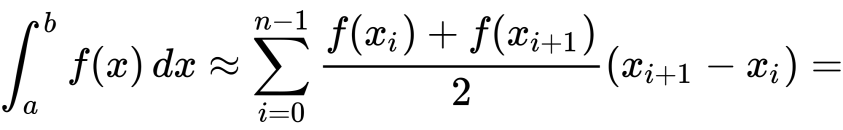
Если отрезок  является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле



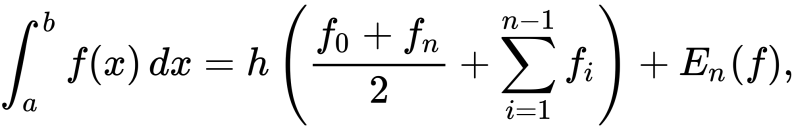
Это простое применение формулы для площади трапеции — произведение полусуммы оснований, которыми в данном случае являются значения функции в крайних точках отрезка, на высоту (длину отрезка интегрирования). Погрешность аппроксимации для элементарного отрезка можно оценить через максимум второй производной



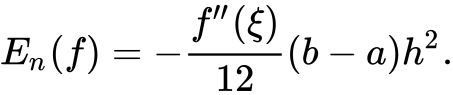
Если отрезок  разбивается узлами интегрирования ,  IMG_258, так что  и , и на каждом из элементарных отрезков  применяется формула трапеций, то суммирование даст составную формулу трапеций



В случае равномерной сетки , где  — шаг сетки, составная формула трапеций упрощается:



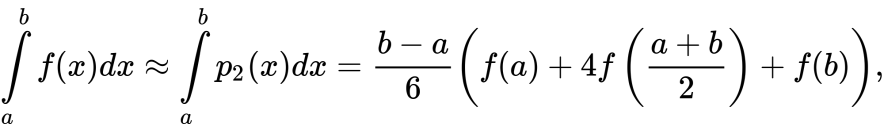
причём для погрешности справедлива оценка



**Метод Симпсона:**

Формула Симпсона заключается в приближении подынтегральной функции на отрезке  [интерполяционным многочленом](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D1%8B) второй степени , то есть приближение графика функции на отрезке параболой. Метод Симпсона имеет [порядок погрешности](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%BF%D0%BE%D0%B3%D1%80%D0%B5%D1%88%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1) 4 и [алгебраический порядок точности](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0) 3.

Формулой Симпсона называется интеграл от интерполяционного многочлена второй степени на отрезке :



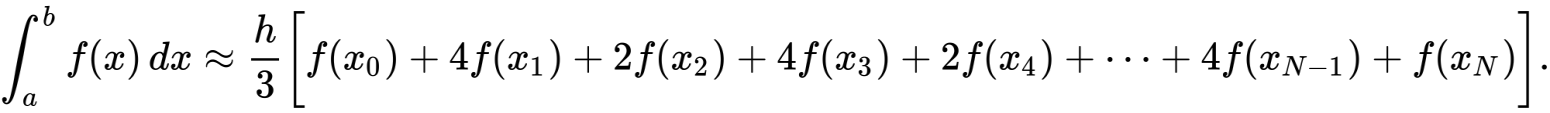
где ,    и  — значения функции в соответствующих точках (на концах отрезка и в его середине).

При условии, что у функции  на отрезке  существует четвёртая [производная](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8), погрешность :

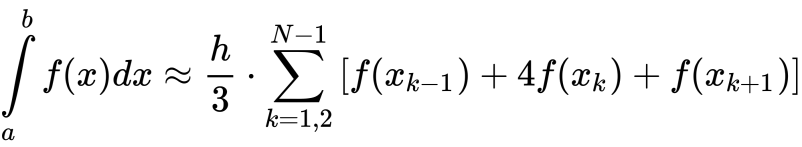
Для более точного вычисления интеграла интервал разбивают на  элементарных отрезков одинаковой длины и применяют формулу Симпсона на составных отрезках. Каждый составной отрезок состоит из соседней пары эементарных отрезков. Значение исходного интеграла является суммой результатов интегрирования на составных отрезках:

где  — величина шага, а  — чередующиеся границы и середины составных отрезков, на которых применяется формула Симпсона. Один подобный составной отрезок состоит из двух элементарных отрезков . Таким образом, если проводить параллели с простой формулой Симпсона, то в данном случае серединой отрезка, на котором применяется формула Симпсона, становится .

Обычно для равномерной сетки данную формулу записывают в других обозначениях (отрезок  разбит на N отрезков) в виде

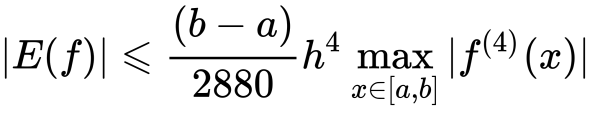


Также формулу можно записать используя только известные значения функции, то есть значения в узлах:



где  означает что индекс меняется от единицы с шагом, равным двум.

Общая погрешность  при интегрировании по отрезку  с шагом  (при этом, в частности, , ) определяется по формулe:

.

## Вычисление дифференциальных уравнений методами Эйлера-Коши и Рунге-Кутта 4-го порядка

**Задача 7:**

Применяя методы Эйлера-Коши и Рунге-Кутта 4-го порядка, найти решение дифференциального уравнения с данным начальным условием в указанной точке. По возможности найти аналитическое решение. Нарисовать графики найденных решений. Сравнить результаты численных и точного решений.

**Метод Эйлера-Коши**

Метод Эйлера является явным, одношаговым методом первого порядка точности. Он основан на аппроксимации [интегральной кривой](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%B2%D0%B0%D1%8F) кусочно-линейной функцией, так называемой ломаной Эйлера.

Пусть дана [задача Коши](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%9A%D0%BE%D1%88%D0%B8) для уравнения первого порядка:

,

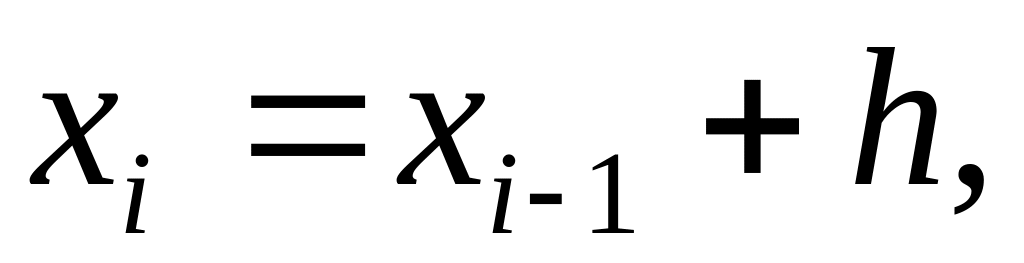
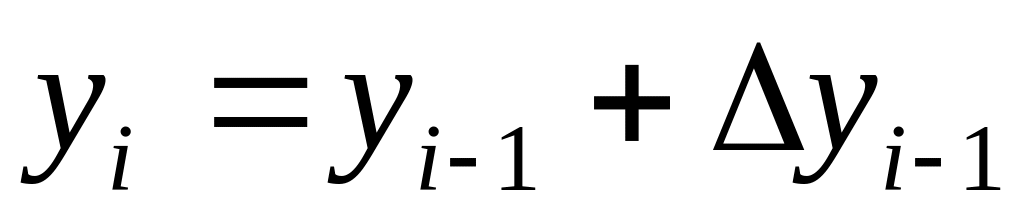
где функция IMG_258 определена на некоторой области  . Решение ищется на полуинтервале . На этом промежутке введём узлы: . Приближенное решение в узлах , которое обозначим через , определяется по формуле:

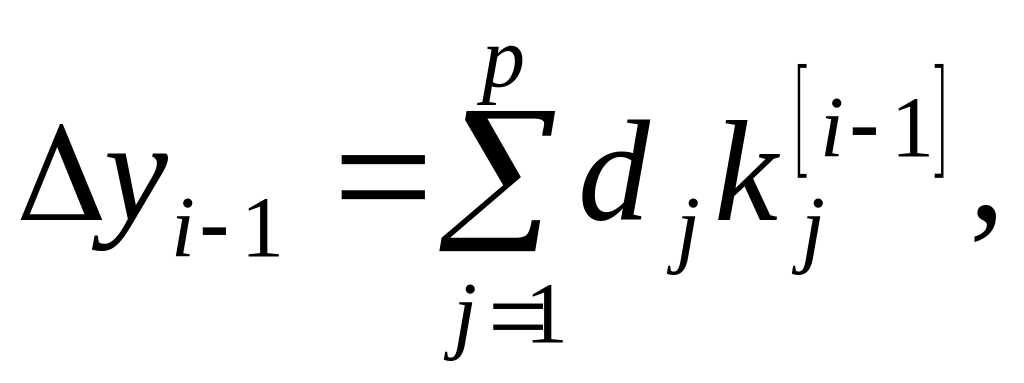
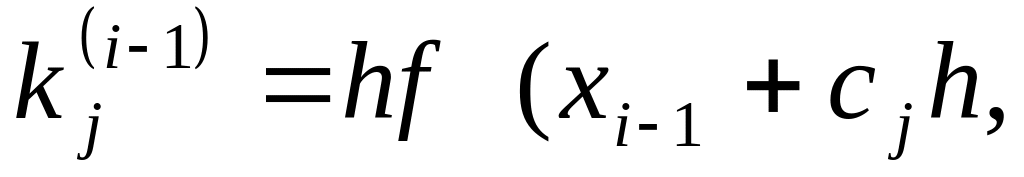
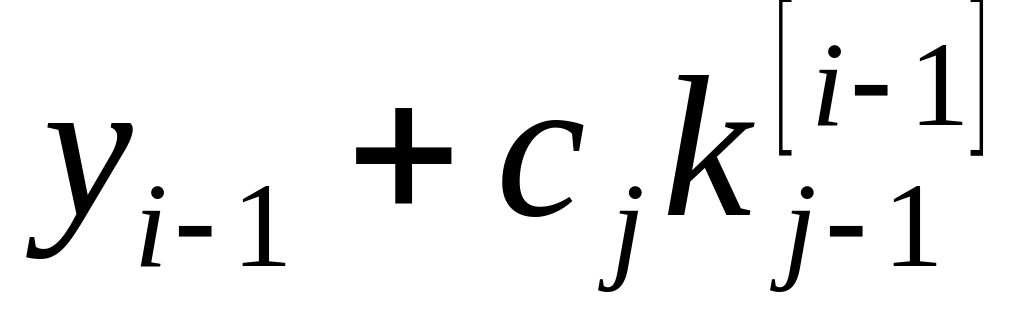
.

Эти формулы непосредственно обобщаются на случай систем обыкновенных дифференциальных уравнений. Существуют модифицированные методы Эйлера, повышающие точность и устойчивость вычисления решения.

**Метод Рунге-Кутта 4-го порядка**

Численные методы решения задачи Коши  на равномерной сетке отрезка [a, b] с шагом   называются методами Рунге - Кутта, если, начиная с данных , решение ведётся по следующим рекуррентным формулам:

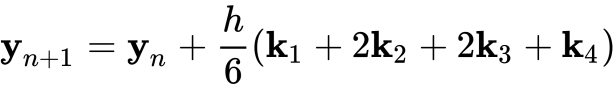
   )

Метод Рунге — Кутты четвёртого порядка при вычислениях с постоянным шагом интегрирования столь широко распространён, что его часто называют просто методом Рунге — Кутты.

Рассмотрим [задачу Коши](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%9A%D0%BE%D1%88%D0%B8) для системы обыкновенных дифференциальных уравнений первого порядка. IMG_256, а IMG_257).

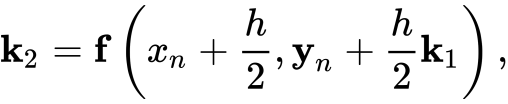
IMG_258

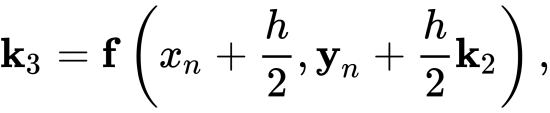
Тогда приближенное значение в последующих точках вычисляется по итерационной формуле:



Вычисление нового значения проходит в четыре стадии:

IMG_260





IMG_263

где IMG_264 — величина шага сетки по IMG_265.

Этот метод имеет четвёртый порядок точности. Это значит, что ошибка на одном шаге имеет порядок IMG_266, а суммарная ошибка на конечном интервале интегрирования имеет порядок IMG_267 .

## Заключение

Численные методы компенсируют недостаток аналитических методов – использование целого ряда допущений и предположений в процессе построения математических моделей и невозможность, в некоторых случаях, получить решение в явном виде из-за неразрешимости уравнений в аналитической форме, отсутствия первообразных для подынтегральных функций и т.п. Тем не менее, каждый метод имеет свои преимущества и недостатки, и, в отличие от аналитических методов, погрешность, которую надо учитывать при применении вычислений.

Использованный в качестве среды разработки для составления программ вычислений пакет MATLAB даёт точные результаты, успешно работает с векторами и матрицами, итерационными процессами. Его можно использовать, как для проверки результатов, так и для написания своих программ по алгоритмам методов, отображения графиков, вычисленных приближённых значений.

В ходе этой курсовой работы я изучила и применила на практике при составлении программ в пакете MATLAB следующие вычислительные методы:

* Метод половинного деления для решения нелинейных уравнений
* Метод простой итерации и Ньютона для решения нелинейных уравнений
* Метод простых итераций и Зейделя для решения СЛАУ
* Метод простых итераций и Ньютона для решения СНАУ
* Построение полиномов Лагранжа и Ньютона для табличной функции
* Методы прямоугольников, трапеций, Симпсона для приближенного вычисления определенного интеграла
* Эйлера-Коши и Рунге-Кутта 4-го порядка для вычисления приближенного значения дифференциальных уравнений

Литература, использованная при составлении курсовой работы приведена в приложении 1. Протоколы программ, составленных для выполнения задач курсовой работы в пакете MATLAB прикреплены в приложениях 2-8.

## Приложение 1

1. Курс высшей математики: Учебное пособие. Часть I./ В.Г. Зубков, В.А. Ляховский, А.И. Мартыненко, В.Б. Миносцев. Под ред. В.Б. Миносцева. –М: МГИУ, 2005. – 480 с.

2. Курс высшей математики: Учебное пособие. Часть II./ В.Г. Зубков, В.А. Ляховский, А.И. Мартыненко, В.Б. Миносцев. Под ред. В.Б. Миносцева. –М: МГИУ, 2005. –517с.

3. Документация MATLAB https://ch.mathworks.com/help/matlab/index.html

## Приложение 2

% Half Division Method 1 task

clc, clear;

x = 0.0 : 0.001 : 2.0;

f= @(x) x.^3+x-3;

plot(x, f(x), 'g'), grid on;

hold on

a = 0.0;

b = 2.0;

e = 0.001;

iteration = 0;

while abs(a - b) > e

c = (a + b) / 2;

if sign(f(c)) == sign(f(a))

a = c;

else

b = c;

end

iteration = iteration + 1;

end

disp(['Answer gotten with Half Division Method: ' num2str(c,8)]);

disp(['Answer found at iteration number: ' num2str(iteration)]);

plot(c, 0, "r\*");

## 

## 

## Приложение 3

clear, clc;

% Simple Iterations and Newton's method

eps = 0.00001;

x = -pi/2: pi/20: pi/2;

y1 = atan(x.^2+1./x);

y2 = x;

plot(x', [y1' y2']);

% roots are in intervals (-0.8; -0.7) (1.1;1.2)

% let phi'(x) = (2\*x.^3-1)/(x.^6+2\*x.^3+x.^2+1) for phi = atan(x.^2+1./x)

% for the interval (1.1;1.2) the convergence condition of the simple

% iterations method is met, but for the interval it's not (-0.8; -0.7)

equation = @(x) atan(x.^2+1./x);

x0 = 1.15;

x1 = equation(x0);

while(abs(x1-x0) > eps)

x0 = x1;

x1 = equation(x0);

end

fprintf('Answer found using Simple Iterations method:\n\t%.6f\n\n', x1);

clear;

equation = @(x) atan(x.^2 + 1/x) - x;

derivative = @(x) (2 \* x.^3 - 1) / (x.^6 + 2 \* x.^3 + x.^2 + 1) - 1;

x0 = -0.75;

x1 = x0 - equation(x0) / derivative(x0);

while (abs(x1 - x0) > eps)

x0 = x1;

x1 = x0 - equation(x0) / derivative(x0);

end

fprintf("First root found using Newton's method:\n\t%.6f\n\n", x1);

x0 = 1.5;

x1 = x0 - equation(x0) / derivative(x0);

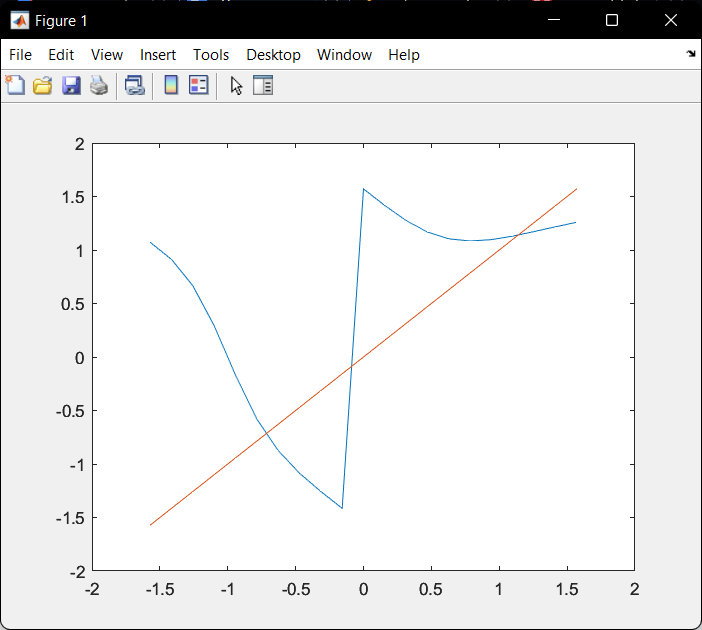
while (abs(x1 - x0) > eps)

x0 = x1;

x1 = x0 - equation(x0) / derivative(x0);

end

fprintf("Second root found using Newton's method:\n\t%.6f\n\n", x1);



## Приложение 4

clear, clc;

A = [3 2 -1; 1 -5 2; -1 -1 2.3];

b = [4; -3; 3.9];

eps = 0.0001;

kmax = 100;

n = length(b);

fprintf('\n Matrix A of coefficients\n');

for i = 1:n

fprintf('%6.2f',A(i,:));

fprintf('\n');

end

fprintf('\n Vector b of free odds\n');

fprintf('%6.2f \n', b);

%% Seidel method

x = zeros(n, 1);

for k = 1:kmax

z = zeros(n, 1);

for i = 1:n

s(i) = b(i);

for j = 1:n

if(i ~= j)

s(i) = s(i) - A(i,j) \* x(j);

end

end

s(i) = s(i) / A(i,i);

z(i) = z(i) + abs(x(i) - s(i));

x(i) = s(i);

end

if(max(z) < eps)

break;

end

end

fprintf('\n Answer computed with Seidel method:\n');

fprintf('%6.2f \n', x);

fprintf('\n Count of iterations k = %3d \n', k);

%% Iterations method

x = zeros(n, 1);

for k = 1:kmax

z = zeros(n, 1);

for i = 1:n

s(i) = b(i);

for j = 1:n

if(i ~= j)

s(i) = s(i) - A(i,j) \* x(j);

end

end

s(i) = s(i) / A(i,i);

z(i) = z(i) + abs(x(i) - s(i));

x1(i) = s(i);

end

if (max(z) < eps)

x = x1;

break;

else

x = x1;

end

end

fprintf('\n Answer computed with Iteration Method:\n');

fprintf('%6.2f \n', x);

fprintf('\n Count of iterations k =%3d \n', k);

%% Check

fprintf("\n Check-up with MATLAB built-in function\n");

check = linsolve(A,b);

fprintf('%6.2f \n', check);

## Приложение 5

***Файл main.m***

clear, clc;

%% build a graphic of system of nonlinear equations

first = ezplot('x^2+y^2-4');

set(first,'Color','g','LineWidth',2);

hold on;

second = ezplot('y-exp(x\*y)');

set(second,'Color','b','LineWidth',2);

grid on;

ptr1 = [0.5 1];

%% Find root with simple iterations method

ans\_iter1 = iter\_non\_linear(ptr1, 0.0001, @equat\_iter);

fprintf(' Found answer with simple iterations: %.6f\t%.6f\n Count of iterations: %d\n\n', ans\_iter1(1), ans\_iter1(2), ans\_iter1(3));

%% Find root with Newton method

disp('Finding a root with Newton method:')

newton(ptr1(1), ptr1(2))

%% Checking if right

[xr, fr, ex] = fsolve(@for\_fsolve, ptr1, optimset('TolX',1.0e-2));

plot(xr(1), xr(2), '\*r');

fprintf('Found answer with MATLAB built-in functions: %15.9f %15.9f\n', xr(1), xr(2));

***Файл iter\_non\_linear.m***

function answer = iter\_non\_linear(approx, eps, funct)

prev = approx;

curr = funct(prev);

count = 0;

while ((abs(prev(1) - curr(1)) > eps) || (abs(prev(2) - curr(2)) > eps))

prev(1) = curr(1);

prev(2) = curr(2);

curr = funct(prev);

count = count + 1;

end

answer(1) = curr(1);

answer(2) = curr(2);

answer(3) = count;

end

***Файл equat\_iter.m***

function res = equat\_iter(in)

res(1) = log(abs(in(2))) / in(2);

res(2) = sqrt(abs(4 - (in(1))^2));

end

***Файл newton.m***

function newton(x, y)

tol = 1e-6;

diff = tol + 1;

n = 0;

nmax = 1000;

disp(' n x(n) y(n) |f(x)|');

X = [x; y];

Y = F(X);

while diff >= tol && n <= nmax

changeX = -dF(X)\Y;

X = X + changeX;

Y = F(X);

diff = norm(changeX);

n = n + 1;

fprintf('%3d %15.9f %15.9f %10.5g \n', n, X(1), X(2), norm(Y));

end

if n > nmax

disp('Failed to converge');

else

fprintf('Root found with Newton method: %15.9f %15.9f\n', X(1), X(2));

end

end

***Файл F.m***

function Y = F(X)

x = X(1); y = X(2);

Y = [x.^2 + y.^2 - 4 ; y - exp(x \* y)];

end

***Файл dF.m***

function J = dF(X)

x = X(1); y = X(2);

J = [2 \* x, 2 \* y; -y \* exp(x \* y), 1 - x \* exp(x \* y)];

end

***Файл for\_fsolve.m***

function f = for\_fsolve( x )

f(1) = x(1).^2 + x(2).^2 - 4;

f(2) = x(2) - exp(x(1) \* x(2));

end

## Приложение 6

clear, clc;

%% least square method

% node coordinates

x = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1];

y = [10.2, 10.37, 10.5, 10.6, 10.76, 10.8, 10.9, 11, 11.1, 11.2];

N = length(x); % nodes count

ex1 = sum(x);

ex2 = sum(x.^2);

ex3 = sum(x.^3);

ex4 = sum(x.^4);

ey = sum(y);

exy = sum(x.\*y);

exy2 = sum(y.\*x.^2);

A = [ex4 ex3 ex2; ex3 ex2 ex1; ex2 ex1 N];

B = [exy2;exy;ey];

aa = A \ B;

% coefficients

ax = aa(1);

bx = aa(2);

cx = aa(3);

disp(ax); disp(bx); disp(cx); % display coeffs

% calculate the resulting line

x2 = linspace(min(x),max(x),200);

y2 = ax.\*x2.^2 + bx.\*x2 + cx;

figure

hold on

plot(x, y, '\*B')

plot(x2, y2, 'R'), grid on;

legend('Original function','Least square method');

%% Lagrange polynomial

x0 = 0.2;

y0 = lagrange(x,y,x0);

syms X

for i = 1:length(x)

u = x;

u(i)=[]; % puncture the i-th element

P(i,1) = prod(X-u)/prod(x(i)-u); % calculate i-th Lagrange polynomial

end

Y = y\*P; % sum of products of y and i-th elements

pretty(collect(Y,X))

% calculate Lagrange points for plotting

xx = linspace(min(x), max(x), 200);

yy = lagrange(x, y, xx);

figure

plot(x,y,'or',xx,yy,':r',x0,y0,'\*b'), grid on

legend('Data','Lagrange Interpolation','Dot', 'location', 'northwest')

%% Newton polynomial

% calculate Newton points to build a graph

xx = linspace(min(x),max(x),200);

yy = newton(x,y, xx);

figure

plot(x,y,'or',xx,yy,':r'), grid on

legend('Data','Newton interpolation', 'location', 'northwest')

**Файл newton.m**

**function yy = newton(x, y, xx)**

**% Вычисление интерполяционного полинома в форме Ньютона**

**% x – массив с абсциссами точек, через которые должен проходить интерполяционный полином**

**% y – массив ординат точек, через которые должен проходить интерполяционный полином**

**% xx – массив значений независимой переменной,**

**% для которых надо вычислить интерполяционный полином**

**% yy – вычисленные значения интерполяционного полинома**

**% определяем число точек**

**N = length(x);**

**% вычисляем разделенные разности**

**DIFF = y;**

**for k = 1 : N-1**

**for i = 1: N - k**

**DIFF(i) = (DIFF(i+1) - DIFF(i)) / (x(i+k) - x(i));**

**end**

**end**

**% вычисляем значения интерполяционного полинома в точках xx**

**% с использованием операции поэлементного умножения .\***

**% для получения сразу всех значений полинома yy**

**yy = DIFF(1) \* ones(size(xx));**

**for k = 2 : N**

**yy = DIFF(k) + (xx - x(k)) .\* yy;**

**end**

**Файл lagrange.m**

**function [L\_n P] = lagrange(x,y,xx)**

**% x - массив координат узлов**

**% y - массив значений интерполируемой функции**

**% xx - массив значений точек интерполяции**

**L\_n = zeros(size(xx));**

**for k=1:length(x)**

**P = ones(size(xx));**

**for i=1:length(x)**

**if k~=i**

**P = P.\*(xx-x(i))./(x(k)-x(i));**

**end**

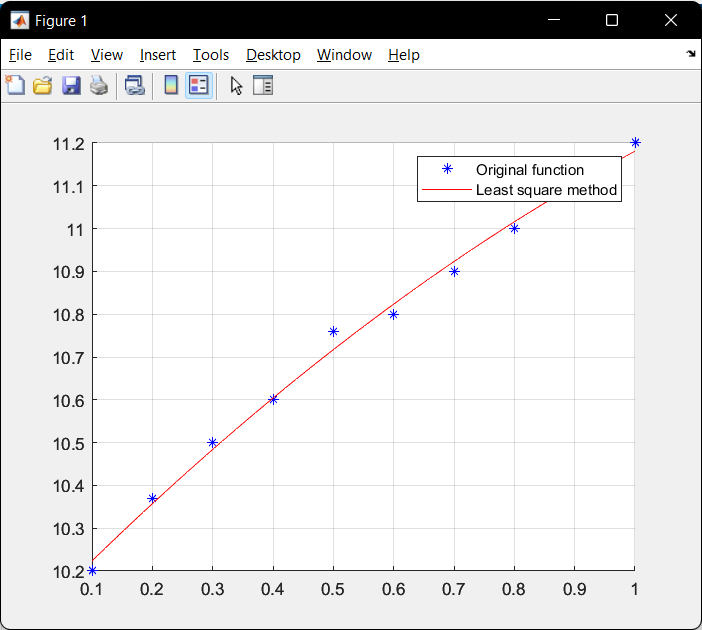
**end**

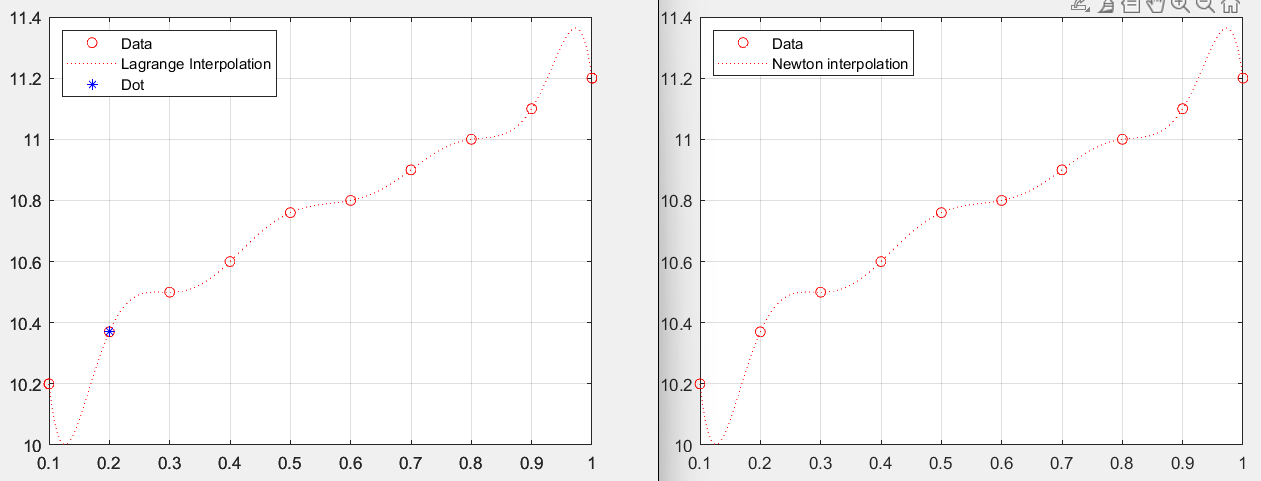
**P0(k,:) = P;**

**L\_n = L\_n + y(k)\*P;**

**end**

**end**





## Приложение 7

clear, clc;

funct = @(x) (sqrt(x)./ (1 + x.^2));

upp\_limit = 1;

low\_limit = 0;

step = 0.00001;

%% Precise answer

integ\_answ = integral(funct, 0, 1);

fprintf('Answer found with MATLAB integral: %.6f\n\n', integ\_answ);

%% Rectangular method

count\_of\_iters = ((upp\_limit - low\_limit) / step) + 1;

i = 1:count\_of\_iters;

x = low\_limit:step:upp\_limit;

y = feval(funct, x);

m = 2:count\_of\_iters;

y1(m - 1) = y(m);

rectan\_answ = sum(step \* y1);

fprintf('Answer found with Rectangular method: %.6f\n', rectan\_answ);

fprintf('Deviation from MATLAB one: %.6f\n\n', abs(integ\_answ - rectan\_answ));

%% Trapezoids method

trapz\_input = funct(x);

trapez\_answ = trapz(x, trapz\_input);

fprintf('Answer found with Trapezoids method: %.6f\n', trapez\_answ);

fprintf('Deviation from MATLAB one: %.6f\n\n', abs(integ\_answ - trapez\_answ));

%% Simpson method

step = 10000;

count\_of\_iters = (upp\_limit - low\_limit) / step;

simps\_answ = funct(low\_limit);

for i = 1:1:((step / 2) - 1)

x = low\_limit + 2 \* count\_of\_iters \* i;

simps\_answ = simps\_answ + 2 \* funct(x) + 4 \* funct(x + count\_of\_iters);

end

simps\_answ = count\_of\_iters \* simps\_answ / 3;

fprintf('Answer found with Simpson method: %.6f\n', simps\_answ);

fprintf('Deviation from MATLAB one: %.6f\n\n', abs(integ\_answ - simps\_answ));

syms f(x)

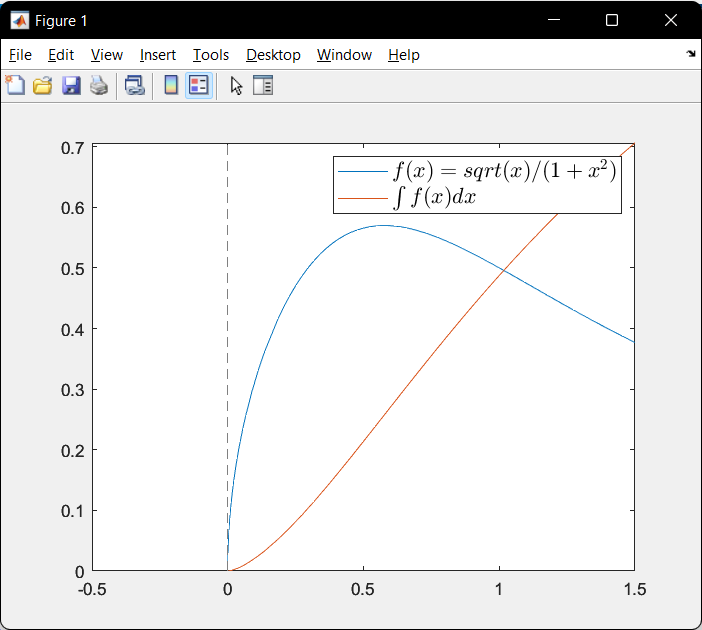
f(x) = sqrt(x)./ (1 + x.^2);

f\_int = int(f(x),x);

fplot([f,f\_int], [-0.5, 1.5])

legend({'$f(x) = sqrt(x)/ (1 + x^2)$','$\int f(x)dx$'},'Interpreter','latex','FontSize',13)

hold on;



## Приложение 8

clear, clc;

syms y(x) ;

enq = diff(y,x) == (y.^2\*log(x) - y) ./ x;

S = dsolve(enq);

cond = y(1) == 1;

Sc = dsolve(enq, cond);

fprintf('Answer found with MATLAB built-in function:\n%s\n\n', Sc);

v = symvar(Sc);

dya = @ (X) double (subs (Sc, v, X));

f = @(x1,y1) (y1.^2\*log(x1) - y1) ./ x1;

x = 1:0.1:3;

dya\_x = dya(x);

plot(x, dya\_x);

y0 = 1; % y(1) = 1

%% Euler method

dye\_x = [];

dye\_x(1) = y0;

n = length(x);

h = x(2) - x(1);

for i=1:n-1

dye\_x(i+1) = dye\_x(i) + h\*f(x(i), dye\_x(i));

end

plot(x, dya\_x, 'r\*', x, dye\_x, 'k\*', x, dya\_x, 'b');

legend({'Answer with MATLAB built-in function', 'Answer with Euler method'},'Interpreter','latex','FontSize',10)

%% Runge-Kutt

dyr\_x = [];

dyr\_x(1) = y0;

n = length(x) ;

h = x(2) - x(1);

for i=1:n-1

k1 = h\*f(x(i),dyr\_x(i));

k2 = h\*f(x(i)+h/2,dyr\_x(i)+k1/2);

k3 = h\*f(x(i)+h/2,dyr\_x(i) +k2/2);

k4 = h\*f(x(i)+h/2,dyr\_x(i) +k3);

dyr\_x(i+1) = dyr\_x(i) + (1/6)\*(k1 + 2\*k2 + 2\*k3 + k4);

end

figure;

plot(x, dya\_x, 'r\*',x, dyr\_x, 'k\*', x, dya\_x, 'b');

legend({'Answer with MATLAB built-in function', 'Answer with Runge-Kutta method'},'Interpreter','latex','FontSize',10)

