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)

Москва, 2022

Оглавление

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 $x^3 + x - 3 = 0$ и найти его методом половинного деления с точностью $\varepsilon = 0,001$.

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

Условием остановы итерационного процесса будет выполнение неравенства $a - b < \varepsilon$.

Новый интервал вычисляется по формулам

$a_k = (a_{k-1} + b_{k-1}) / 2 > 0$, $b_k=b_{k-1}$, если

$f(a_k)f((a_{k-1} + b_{k-1}) / 2) > 0$;

или по формулам

$a_k=a_{k-1}$, $b_k = (a_{k-1} + b_{k-1}) / 2$, если $f(b_k)f((a_{k-1} + b_{k-1}) / 2) > 0$.

Приближенное значение корня к моменту окончания итерационного процесса вычисляется по формуле $x_* = (a_{\text{конечное}} + b_{\text{конечное}}) / 2$

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:

$$arctg(x^2 + \frac{1}{x}) = x$$

Метод простых итераций

Идея метода итераций заключается в замене исходного уравнения $F(x) = 0$ равносильным уравнением вида x $= f(x)$ с выделенным линейным членом.

Достаточное условие сходимости метода: $|\dot{f}(x)| < 1,\ x \in [a, b]$. Это условие необходимо проверить перед началом решения задачи, так как функция $f(x)$ может быть выбрана неоднозначно, причем в случае неверного выбора указанной функции метод расходится.

Начальное приближение корня вычисляется по формуле: $x_0 = (a + b)/2$; итерационный процесс задаёт формула $x_{n+1} = f(x_n)$, условие его окончания: $|x_n - x_{n+1}| < \varepsilon$.

Метод Ньютона

Метод Ньютона, или метод касательных, заключается в том, что если x_n – некоторое приближение к корню уравнения $f(x) = 0,\ f \in R$, то следующее приближение определяется как корень касательной к функции $f(x)$, проведённой в точке x_n. Найдём формулу для нахождения следующего приближения методом Ньютона из исходного уравнения $f(x_n) + f'(x_n)(x - x_n) = 0$:

- Уравнение касательной имеет вид $f'(x_n) = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$
- Положим y $= 0$.
- Тогда $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$

Кроме того, для начала вычислений требуется задание начального приближения x_0 удовлетворяющего условию сходимости метода к корню x_*: $f(x_0)\ \ddot{f}(x_0) > 0$. Вычисления производятся, пока $|x_0 - x_n| > \varepsilon$

Метод простых итераций и Зейделя для решения СЛАУ

Задача 3:
Решить систему линейных уравнений методами простых итераций и Зейделя с точностью $\varepsilon = 0,0001$

$$\begin{cases} x_1 - 5x_2 + 2x_3 = -3; \\ -x_1 - x_2 + 2,3x_3 = 3,9; \\ 3x_1 + 2x_2 - x_3 = 4. \end{cases}$$

Пусть задана система уравнений

$$\sum_{j=1}^{N} a_{ij} x_j = b_i, \qquad i = \overline{1,\ N}$$

Выразим x_i через остальные члены i-го уравнения:

$$x_i = \frac{1}{a_{ii}} (b_i - \sum_{j=1}^{i=1} a_{ij} x_j - \sum_{j=i+1}^{N} a_{ij} x_j), \qquad i = \overline{1,\ N}$$

Полученная запись СЛАУ приводит к двум итерационным процессам:

1) Метод простой итерации:

$$x_i^{k+1} = \frac{1}{a_{ii}} (b_i - \sum_{\substack{j=1 \\ j \neq i}}^{N} a_{ij} x_j^k),\ i = \overline{1,\ N}$$

2) Метод Зейделя:

$$x_i^{k+1} = \frac{1}{a_{ii}} (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{N} a_{ij} x_j^k),\ i = \overline{1,\ N}$$

При этом x_i^0 задаётся, где $i = \overline{1,\ N}$; k – номер итерации.

Процесс ведётся до выполнения условия $z_k = \|\bar{x}^{k+1} - \bar{x}^k\| < \varepsilon$

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

$$\sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{ij}| \leq q|a_{ii}, |, \qquad i = \overline{1,\ N}, \qquad 0 < q < 1$$

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

1- е уравнение: $1 < |-5| + 2 = 7$ — не выполняется

2- е уравнение: $|-1| < |-1| + 2,3 = 3,3$ — не выполняется

3- е уравнение: $|-1| < 2 + 3 = 5$ — 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:

$$\{3x_1 + 2x_2 - x_3 = 4\ x_1 - 5x_2 + 2x_3 = -3\ -x_1 - x_2 + 2,3x_3 = 3,9$$

Метод простых итераций и Ньютона для решения СНАУ

Задача 4:

Найдите численное решение системы уравнений методом Ньютона с точностью $\varepsilon = 0,0001$. Найдите решение системы с той же точностью простыми итерациями. Сравните способы решения (точность, количество итераций).

$$\begin{cases} x^2 + y^2 = 4 \\ y = e^{xy} \end{cases}$$

Метод простых итераций

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

$$\begin{cases} x_1 = \phi(x_1, x_2, \ldots, x_n) \\ x_2 = \phi(x_1, x_2, \ldots, x_n) \\ \quad \vdots \\ x_n = \phi(x_1, x_2, \ldots, x_n) \end{cases}$$

Алгоритм решения заключается в последовательном вычислении $x_i^{(k+1)} = \phi_i(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \ldots, x_n^{(k)}) \quad i = \overline{1, n}$ и $k = 1, 2, 3, \ldots$,

Начиная с некоторого начального приближения, пока:

- будет выполнено условие $|f_i(x_k)| < \varepsilon, \ i = \overline{1, n}$, где ε - 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

$$\sum_{k=1}^{n} \left|\frac{\partial \phi_k}{\partial x_i}\right| < 1, \qquad i = \overline{1, n}.$$

Метод простых итераций рекомендуется применять в тех случаях, когда исходная система легко преобразуется к виду $x_i = \phi_i(x_1, x_2, ..., x_n)$, $i = \overline{1, n}$ 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 $F_i(x_1, x_2, \ldots, x_n)$ 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 $a_1, a_2, ..., a_n$. Задача состоит в нахождении приращений (поправок) к этим значениям $\Delta x_1, \Delta x_2, \ldots, \Delta x_n$ благодаря которым решение исходной системы запишется в виде:
$x_1 = a_1 + \Delta x_1, x_2 = a_2 + \Delta x_2, \ldots, x_n = a_n + \Delta x_n$. Проведем разложение левых частей уравнений исходной системы в ряд Тэйлора, ограничиваясь лишь линейными членами относительно приращений:

$$\begin{cases} F_1(x_1, x_2, ..., x_n) \approx F_1(a_1, a_2, ..., a_n) + \dfrac{\partial F_1}{\partial x_1}\Delta x_1 + ... + \dfrac{\partial F_1}{\partial x_n}\Delta x_n \\ F_2(x_1, x_2, ..., x_n) \approx F_2(a_1, a_2, ..., a_n) + \dfrac{\partial F_2}{\partial x_1}\Delta x_1 + ... + \dfrac{\partial F_2}{\partial x_n}\Delta x_n \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ F_n(x_1, x_2, ..., x_n) \approx F_n(a_1, a_2, ..., a_n) + \dfrac{\partial F_n}{\partial x_1}\Delta x_1 + ... + \dfrac{\partial F_n}{\partial x_n}\Delta x_n \end{cases}$$

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

$$\begin{cases} \dfrac{\partial F_1}{\partial x_1}\Delta x_1 + \dfrac{\partial F_1}{\partial x_2}\Delta x_2 + ... + \dfrac{\partial F_1}{\partial x_n}\Delta x_n = -F(a_1, a_2, ..., a_n) \\ \dfrac{\partial F_2}{\partial x_1}\Delta x_1 + \dfrac{\partial F_2}{\partial x_2}\Delta x_2 + ... + \dfrac{\partial F_2}{\partial x_n}\Delta x_n = -F(a_1, a_2, ..., a_n) \\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \\ \dfrac{\partial F_n}{\partial x_1}\Delta x_1 + \dfrac{\partial F_n}{\partial x_2}\Delta x_2 + ... + \dfrac{\partial F_n}{\partial x_n}\Delta x_n = -F(a_1, a_2, ..., a_n) \end{cases}$$

в матричном виде: $\dfrac{\partial F}{\partial X}\cdot \Delta x = -F; \quad \Delta x = -\left(\dfrac{\partial F}{\partial X}\right)^{-1} F$

Значения $F_1, F_2, \ldots, F_n$ и их производные вычисляются при $x_1 = a_1, x_2 = a_2, \ldots, x_n = a_n$.

Определителем последней системы является якобиан:

$$J = \begin{vmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \dots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \dots & \frac{\partial F_2}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \dots & \frac{\partial F_n}{\partial x_n} \end{vmatrix}.$$

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 $\Delta x_1, \Delta x_2, \dots, \Delta x_n$ к значениям неизвестных на каждой итерации. Счет прекращается, если все приращения становятся малыми по абсолютной величине:

$$\max_i |\Delta x_i| < \varepsilon.$$

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

$$x = a + \Delta x = a - \left(\frac{\partial F}{\partial X}\right)^{-1} F \quad \text{или} \quad x^{k+1} = x^k - \frac{\partial F(x^k)}{\partial x} F(x^k).$$

Критерий окончания. Будем считать, что заданная точность достигнута, если $\max\left(\left|x^k - x^{k+1}\right|, \left|y^k - y^{k+1}\right|\right) < \varepsilon$ или $\left\|F(x^k)\right\| \to 0$.

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 $y = a_2x^2 + a_2x + a_0$ методом наименьших квадратов. Построить для заданной табличной функции полиномы Лагранжа и Ньютона и упростить их.

x_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y_i	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 a и b $F(a,\ b) = \sum_{i=1}^{n} (y_i - (ax_i + b))^2$ 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).

$$\begin{cases} a = \dfrac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \\ b = \dfrac{\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i}{n} \end{cases}$$

Интерполяционные полиномы Лагранжа и Ньютона
Полином Лагранжа

Полином $P_N(x_n) = \sum_{n=0}^{N} y_n l_n(x)$, где сомножители $l_n(x)$ задаются как $l_n(x) = \prod_{m=0,\, m\neq n}^{N} \frac{x-x_m}{x_n-x_m}$, $n = 0, \ldots, N$ называется интерполяционным полиномом Лагранжа степени N.

Для вычисления значения интерполяционного полинома в некоторой точке x, не прибегая к явному вычислению коэффициентов полинома, применяется схема Невилла. На первом этапе зададим значения $P_n^{(0)}(x_n) = y_n$, $n = 0, \ldots, N$. Далее значения вспомогательных полиномов в точке x определяются согласно рекуррентному соотношению, называющимся интерполяционным полиномом Лагранжа степени N.

$$P_n^{(k)} = (x) = \frac{(x-x_n)P_{n+1}^{(k-1)}(x)-(x-x_{n+k})P_n^{(k-1)}(x)}{x_{n+k}-x_n}, \quad k = 1, \ldots, N, \; n = 0, \ldots, N-1$$

и окончательно мы получаем

$$P_0^{(N)}(x) = P_N(x);\; P_N(x_n) = \sum_{n=0}^{N} y_n l_n(x), \text{ где } l_n(x) = \prod_{m=0,\, m\neq n}^{N} \frac{x-x_m}{x_n-x_m},\; n = 0, \ldots, N$$

Для вычисления значения интерполяционного полинома в некоторой точке x, не прибегая к явному вычислению коэффициентов полинома, применяется схема Невилла. На первом этапе зададим значения $P_n^{(0)}(x_n) = y_n$, $n = 0, \ldots, N$. Далее значения вспомогательных полиномов в точке x определяются согласно рекуррентному соотношению

$$P_n^{(k)}(x) = \frac{(x-x_n)P_{n+1}^{(k-1)}(x)-(x-x_{n+k})P_n^{(k-1)}(x)}{x_{n+k}-x_n},\; k = 1, \ldots, N;\; n = 0, \ldots, N-1 \text{ и}$$

окончательно мы получаем $P_0^{(N)} = P_N(x)$.

Полином Ньютона

Пусть заданы различные точки $x_0, \ldots, x_N$ и значения в этих точках $y_0, \ldots, y_{jV}$ Разделенные разности нулевого порядка есть просто значения y_n: $D_n^{(0)}(x_n) = y_n,\; n = 0, \ldots, N$

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

$$D_n^{(k)} = \frac{D_{n+1}^{(k-1)}-D_n^{(k-1)}}{x_{n+k}-x_n},\; k = 1, \ldots, N,\; n = 0, \ldots, N-k;$$

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

$$P_N(x) = D_0^{(0)} + \sum_{m=1}^{N} D_0^{(m)} \prod_{n=0}^{m-1} (x - x_n) \quad \text{или}$$

$$P_N(x) = \left\{ \ldots \left\{ D_0^{(N)}(x - x_{N-1}) + D_0^{(N-1)} \right\} (x - x_{N-2}) + \ldots + D_0^{(1)}(x - x_0) + D_0^{(0)} \right\}$$

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

$a_N = D_0^{(N)}$, $a_k = a_{k+1}(x - x_k) + D_0^{(k)}$, $k = N - 1, \ldots, 0$;

Тогда требуемое значение полинома равно последнему элементу этой последовательности, т.е. $P(x) = a_0$.

Вычисление приближенного значения интеграла

Задача 6:
Вычислить интегралы по формулам прямоугольников, трапеций и Симпсона с точностью $\varepsilon = 0,0001$. Вычислить точное значение интеграла (если возможно) и найти отклонение точного значения от приближенного. В несобственном интеграле выполнить преобразование и особенность убрать.

$$\int_0^1 \frac{\sqrt{x}}{1+x^2}\,dx$$

Метод прямоугольников:

Метод прямоугольников — метод численного интегрирования 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 равен 0, но для формулы средних прямоугольников равен 1, поэтому далее под методом прямоугольника будем иметь формулу средних прямоугольников.

Если отрезок $[a,\ b]$ является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле:

$$\int_a^b f(x)\,dx \approx f\left(\frac{a+b}{2}\right)(b-a).$$

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

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)(x_{i+1} - x_i) = \sum_{i=1}^{n} f\left(\frac{x_{i-1} + x_i}{2}\right)(x_i - x_{i-1}).$$

С увеличением n точность получаемого по приближённым формулам результата возрастает.

Равномерную сетку можно описать следующим набором формул:

$$x_i = a + ih, \qquad h = \frac{b-a}{n}, \text{ где } h - \text{шаг сетки}$$

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

$$\int_a^b f(x)\,dx \approx h(\frac{f_0}{2} + f_1 + \ldots + f_{n-1} + \frac{f_n}{2}).$$

Погрешность для формулы прямоугольников составляет

$$E(f) = \frac{f''(\xi)}{24}h^3.$$

Погрешность для составной формулы прямоугольников:

$$E(f) = \frac{f''(\xi)}{24}h^3 \cdot n = \frac{f''(\xi)}{24}(b-a)h^2.$$

Метод трапеций:

Метод трапеций — метод численного интегрирования 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. Алгебраический порядок точности равен 1. Если отрезок $[a,\ b]$ является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле

$$\int_a^b f(x)\,dx = \frac{f(a)+f(b)}{2}(b-a) + E(f), \qquad E(f) = -\frac{f''(\xi)}{12}(b-a)^3.$$

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

$$|E(f)| \leqslant \frac{(b-a)^3}{12}\max_{x\in[a,b]}|f''(x)|$$

Если отрезок $[a,\ b]$ разбивается узлами интегрирования x_i, $i = 0, 1, \ldots, n$, так что $x_0 = a$ и $x_n = b$, и на каждом из элементарных отрезков $[x_i,\ x_{i+1}]$ применяется формула трапеций, то суммирование даст составную формулу трапеций

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1}\frac{f(x_i)+f(x_{i+1})}{2}(x_{i+1}-x_i) =$$

$$= \frac{f(a)}{2}(x_1-a) + \frac{1}{2}\sum_{i=1}^{n-1}f(x_i)(x_{i+1}-x_{i-1}) + \frac{f(b)}{2}(b-x_{n-1}).$$

В случае равномерной сетки $x_j = a + jh$, где $h = (b - a)/n$ — шаг сетки, составная формула трапеций упрощается:

$$\int_a^b f(x)\,dx = h\left(\frac{f_0 + f_n}{2} + \sum_{i=1}^{n-1} f_i\right) + E_n(f),$$

причём для погрешности справедлива оценка

$$E_n(f) = -\frac{f''(\xi)}{12}(b-a)h^2.$$

Метод Симпсона:

Формула Симпсона заключается в приближении подынтегральной функции на отрезке $[a,\ b]$ интерполяционным многочленом второй степени $p_2(x)$, то есть приближение графика функции на отрезке параболой. Метод Симпсона имеет порядок погрешности 4 и алгебраический порядок точности 3.

Формулой Симпсона называется интеграл от интерполяционного многочлена второй степени на отрезке $[a,\ b]$:

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right),$$

где $f(a),\ f((a + b)/2)$ и $f(b)$ — значения функции в соответствующих точках (на концах отрезка и в его середине).

При условии, что у функции $f(x)$ на отрезке $[a,\ b]$ существует четвёртая производная, погрешность $E(f)$:

$$|E(f)| \leq \frac{(b-a)^5}{2880} max|f^{(4)}(x)|,\ x \in [a\,,\ b];$$

Для более точного вычисления интеграла интервал$[a,\ b]$ разбивают на $N = 2n$ 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:

$$\int_a^b f(x)dx \approx \frac{h}{3}[f(x_0) + 2\sum_{j=1}^{N/2-1} f(x_{2j}) + 4\sum_{j=1}^{N/2} f(x_{2j} - 1) + f(x_N)]$$

где $h = \frac{b-a}{N}$ — величина шага, а $x_j = a + jh$ — чередующиеся границы и середины составных отрезков, на которых применяется формула Симпсона. Один подобный составной отрезок $[x_{j-1}, x_{j+1}]$состоит из двух элементарных отрезков $[x_{j-1}, x_j],\ [x_j,\ x_{j+1}]$. Таким образом, если проводить параллели с простой формулой Симпсона, то в данном случае серединой отрезка, на котором применяется формула Симпсона, становится x_j.

Обычно для равномерной сетки данную формулу записывают в других обозначениях (отрезок $[a,\ b]$ разбит на N отрезков) в виде

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{N-1}) + f(x_N)\right].$$

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

$$\int_a^b f(x)dx \approx \frac{h}{3} \cdot \sum_{k=1,2}^{N-1} [f(x_{k-1}) + 4f(x_k) + f(x_{k+1})]$$

где $k = 1,\ 2$ означает что индекс меняется от единицы с шагом, равным двум.

Общая погрешность $E(f)$ при интегрировании по отрезку $[a,\ b]$ с шагом $x_i - x_{i-1} = h$ (при этом, в частности, $x_0 = a$, $x_N = b$) определяется по формуле:

$$|E(f)| \leqslant \frac{(b-a)}{2880} h^4 \max_{x\in[a,b]} |f^{(4)}(x)|.$$

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.

$$x\dot{y} + y = y^2 lnx, \qquad y(1) = 1$$

Метод Эйлера-Коши

Метод Эйлера является явным, одношаговым методом первого порядка точности. Он основан на аппроксимации интегральной кривой кусочно-линейной функцией, так называемой ломаной Эйлера. Пусть дана задача Коши для уравнения первого порядка:

$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0$

где функция f определена на некоторой области $D \subset R^2$. Решение ищется на полуинтервале $(x_0, b]$. На этом промежутке введём узлы:

$x_0 < x_1 < \ldots < x_n \leq b$. Приближенное решение в узлах x_i, которое обозначим через y_i, определяется по формуле:

$$y_i = y_{i-1} + (x_i - x_{i-1}) f(x_{i-1}, y_{i-1}),\ i = 1, 2, 3, \ldots, n.$$

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 $y' = f(x, y)$ на равномерной сетке $x_0 = a, x_1, x_2, \ldots x_m = b$ отрезка [a, b] с шагом $h = \frac{b-a}{m}$ называются методами Рунге - Кутта, если, начиная с данных (x_0, y_0), решение ведётся по следующим рекуррентным формулам:

$$x_i = x_{i-1} + h, \quad y_i = y_{i-1} + \Delta y_{i-1} \quad (i = \overline{1, m})$$

$$\Delta y_{i-1} = \sum_{j=1}^{p} d_j k_j^{[i-1]}, \quad k_j^{(i-1)} = hf\ (x_{i-1} + c_j h, \quad y_{i-1} + c_j k_{j-1}^{[i-1]})$$

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 для системы обыкновенных дифференциальных уравнений первого порядка. $\mathbf{y}, \mathbf{f}, \mathbf{k}_i \in \mathbb{R}^n$, а $x, h \in \mathbb{R}^1$).
$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{y}(x_0) = \mathbf{y}_0.$

Тогда приближенное значение в последующих точках вычисляется по итерационной формуле:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

Вычисление нового значения проходит в четыре стадии:
$\mathbf{k}_1 = \mathbf{f}\left(x_n, \mathbf{y}_n\right),$

$\mathbf{k}_2 = \mathbf{f}\left(x_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right),$

$\mathbf{k}_3 = \mathbf{f}\left(x_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_2\right),$

$\mathbf{k}_4 = \mathbf{f}\left(x_n + h, \mathbf{y}_n + h\,\mathbf{k}_3\right).$

где h — величина шага сетки по x.
Этот метод имеет четвёртый порядок точности. Это значит, что ошибка на одном шаге имеет порядок $O(h^5)$, а суммарная ошибка на конечном интервале интегрирования имеет порядок $O(h^4)$.

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

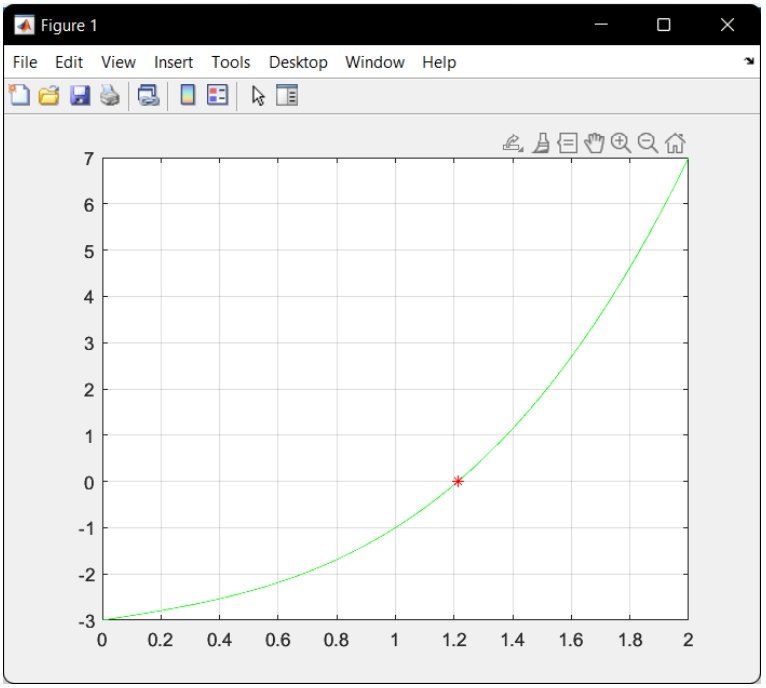
Литература, использованная при составлении курсовой работы приведена в приложении 1. Протоколы программ, составленных для выполнения задач курсовой работы в пакете MATLAB прикреплены в приложениях 2-8.

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

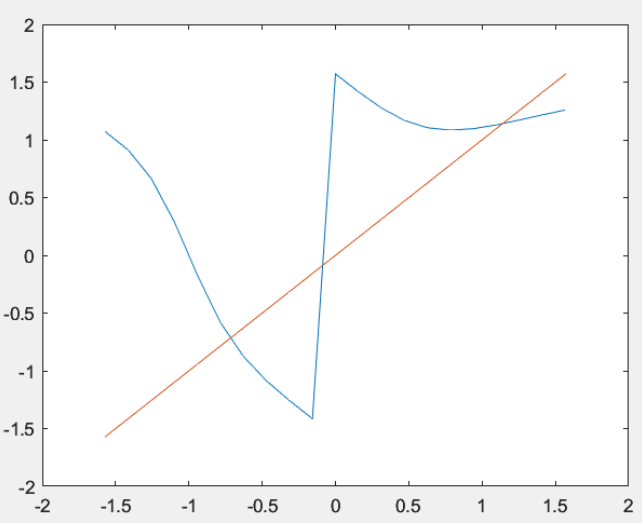
Приложение 2

```
% 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')
```

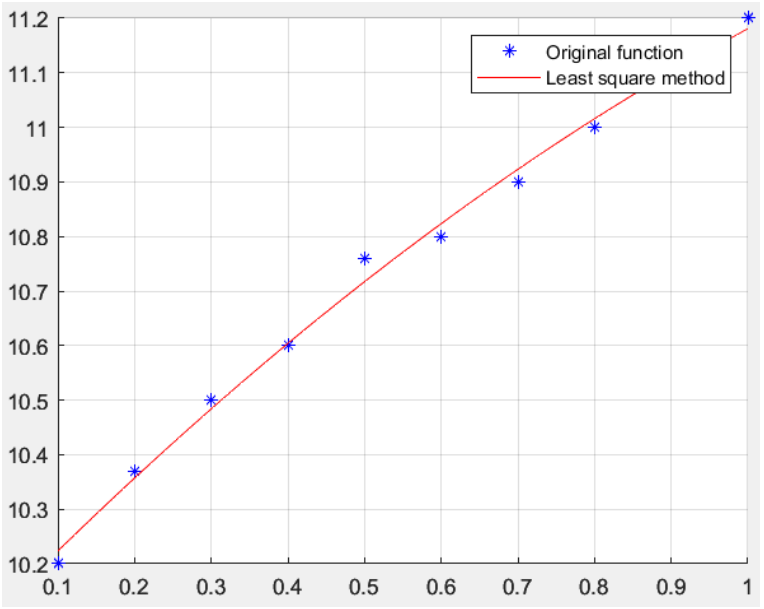
Файл newton.m

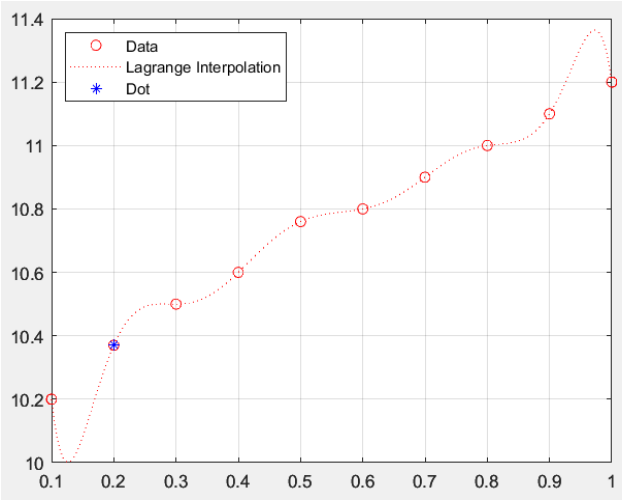
```
function yy = newton(x, y, xx)
% Вычисление интерполяционного полинома в форме Ньютона
% x - массив с абсциссами точек, через которые должен проходить
интерполяционный полином
% y - массив ординат точек, через которые должен проходить интерполяционный
полином
% xx - массив значений независимой переменной,
% 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
```

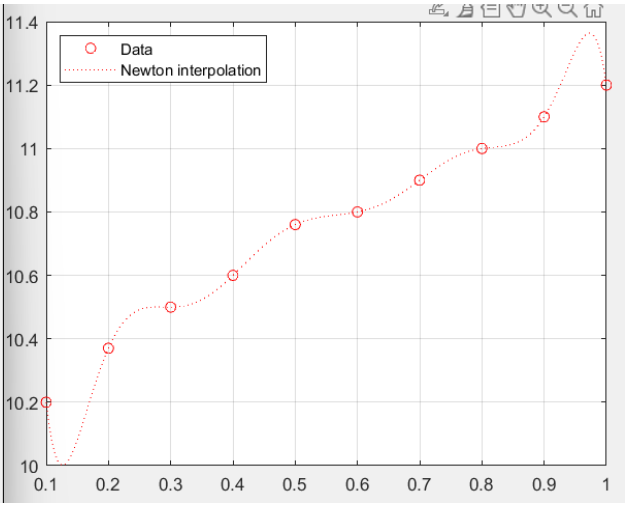
Файл lagrange.m

```
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
```


Original function
Least square method
11.2
11.1
11
10.9
10.8
10.7
10.6
10.5
10.4
10.3
10.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

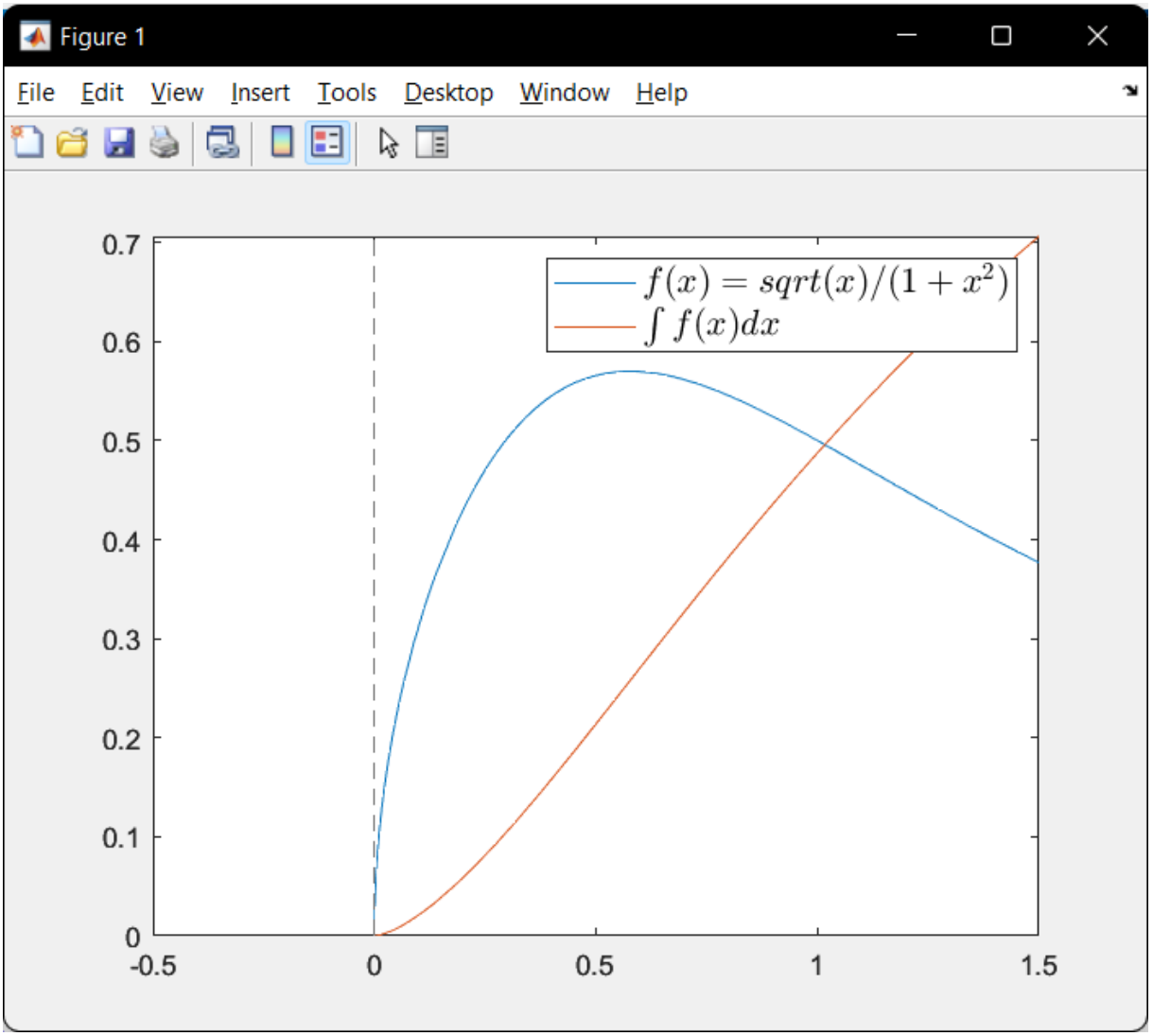

Data
Lagrange Interpolation
Dot
11.4
11.2
11
10.8
10.6
10.4
10.2
10
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Data
Newton interpolation
11.4
11.2
11
10.8
10.6
10.4
10.2
10
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

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)
```

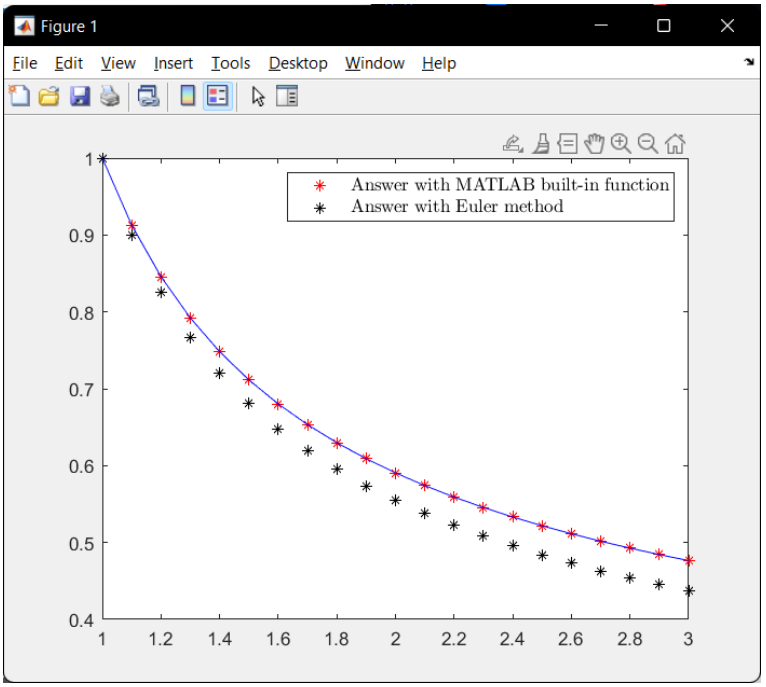


Figure 1
File Edit View Insert Tools Desktop Window Help
Answer with MATLAB built-in function
Answer with Euler method
1
0.9
0.8
0.7
0.6
0.5
0.4
1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 3

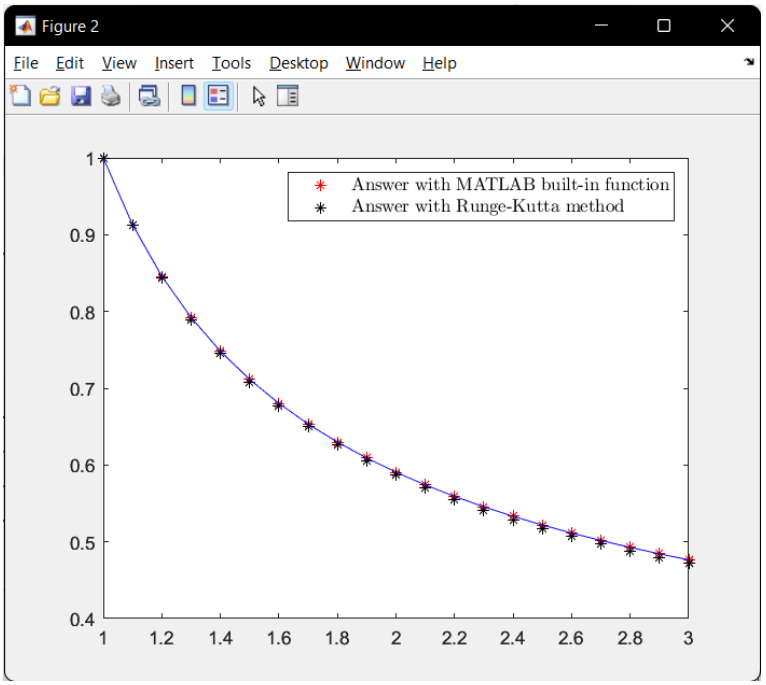


Figure 2
File Edit View Insert Tools Desktop Window Help
Answer with MATLAB built-in function
Answer with Runge-Kutta method
1
0.9
0.8
0.7
0.6
0.5
0.4
1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 3