Министерство науки и высшего образования Российской Федерации
Томский государственный университет
систем управления и радиоэлектроники

А.Е. Максимов
Д.В. Клюкин
С.П. Куксенко

ПРИКЛАДНАЯ ИНФОРМАТИКА

Методические указания по лабораторным работам и самостоятельной работе

Томск 2024

УДК 004.9
ББК 16
М17

Рецензент:
Суровцев Р.С., доцент кафедры ТУ, канд. техн. наук

Максимов, Александр Евгеньевич
Клюкин, Дмитрий Владимирович
Куксенко, Сергей Петрович

М17 Максимов, А.Е. Прикладная информатика: Методические указания по лабораторным работам и самостоятельной работе / А.Е. Максимов, Д.В. Клюкин, С.П. Куксенко. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2024. – 44 с.

Настоящие методические указания составлены с учетом требований федерального государственного образовательного стандарта высшего образования (ФГОС ВО). Методические указания охватывают семь лабораторных работ в пакете прикладных программ GNU Octave и системах компьютерной алгебры SMath Studio и Maxima. Методические указания предназначены для студентов бакалавриата и специалитета высших учебных заведений, обучающихся по техническим направлениям подготовки.

Одобрено на заседании кафедры ТУ, протокол №9 от 12.12.2024 г.

УДК 004.9
ББК 16

ОГЛАВЛЕНИЕ

ЛАБОРАТОРНЫЕ РАБОТЫ

1 РЕШЕНИЕ СЛАУ С ПОМОЩЬЮ LU-РАЗЛОЖЕНИЯ

1.1 Краткая теоретическая справка

Для решения системы линейных алгебраических уравнений (СЛАУ) (рисунок 1.1) в пакете прикладных программ GNU Octave необходимо перейти от координатного вида записи к матричному **Ax** = **b** (рисунок 1.2).

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{p1}x_1 + a_{p2}x_2 + \ldots + a_{pn}x_n = b_p \end{cases}$$

Рисунок 1.1 – СЛАУ

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{p1} & a_{p2} & \ldots & a_{pn} \end{pmatrix} – \text{основная матрица СЛАУ;}$$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} – \text{матрица-столбец неизвестных переменных;}$$

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} – \text{матрица свободных членов.}$$

Рисунок 1.2 – Матричный вид записи СЛАУ

Затем надо выполнить операцию матричного левого деления **x** = **A** \ **b**. Однако более эффективным методом решения СЛАУ является метод разложения на треугольные матрицы, или LU-разложение. LU-разложение матрицы **A** можно представить в виде ***A*** = ***LU***, где ***L*** – нижнетреугольная матрица с единичной диагональю, а **U** – верхнетрегуольная матрица. Тогда решение СЛАУ при известном LU-разложении сводится к решению системы

$$\begin{cases} \mathbf{Ly} = \mathbf{b} \\ \mathbf{Ux} = \mathbf{y} \end{cases},$$

причем сначала находится LU-разложение матрицы, далее решается первое уравнение, а затем второе.

Существует множество вариантов реализации разложения. Чаще всего используются так называемые *ijk*-формы со следующим обозначением индексов:

– *k* – номер исключаемой переменной;

– *i* – номер строки, т. е. модифицируемого уравнения;

– *j* – номер столбца, т. е. коэффициента в модифицируемом уравнении.

Далее приведены используемые в данной работе *ijk*-алгоритмы.

LU-разложение, *ikj*-алгоритм

Для $i = 2, \ldots, N$
  Для $k = 1, \ldots, i - 1$
    $a_{i,k} = a_{i,k} / a_{k,k}$
    Для $j = k + 1, \ldots, N$
      $a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
    Увеличить j
  Увеличить k
Увеличить i

Данный алгоритм позволяет пересчитать i-ю строку матрицы $\boldsymbol{A}$ в i-ю строку матриц $\boldsymbol{L}$ и $\boldsymbol{U}$. Каждые 1, 2,…, $j - 1$ строки участвуют в определении j-й строки матриц $\boldsymbol{L}$ и $\boldsymbol{U}$, но сами больше не модернизируются. Таким образом, доступ к элементам матрицы A производится по строкам. Исключение по строкам. Модификации отложенные.

LU-разложение, *ijk*-алгоритм

Для $i = 2, \ldots, N$
  Для $j = 2, \ldots, i$
    $a_{i,j-1} = a_{i,j-1} / a_{j-1,j-1}$
    Для $k = 1, \ldots, j-1$
      $a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
    Увеличить k
  Увеличить j
  Для $j = i+1, \ldots, N$
    Для $k = 1, \ldots, i-1$
      $a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
    Увеличить k
  Увеличить j
Увеличить i

Доступ к элементам матрицы $\boldsymbol{A}$ производится по строкам. Исключение по строкам. Модификации отложенные. Первый цикл по j элементы i-й строки матрицы $\boldsymbol{L}$. Второй цикл по j элементы i-й строки $\boldsymbol{U}$.

LU-разложение, *kij*-алгоритм

Для $k = 1, \ldots, N - 1$
  Для $i = k + 1, \ldots, N$
    $a_{i,k} = a_{i,k} / a_{k,k}$
    Для $j = k + 1, \ldots, N$
      $a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
    Увеличить j
  Увеличить i
Увеличить k

Доступ к элементам матрицы $\boldsymbol{A}$ производится по строкам. Исключение по столбцам. Модификации немедленные.

LU-разложение, *kji*-алгоритм

Для $k = 1, \ldots, N - 1$
  Для $s = k + 1, \ldots, N$
    $a_{s,k} = a_{s,k} / a_{k,k}$
  Увеличить s
  Для $j = k + 1, \ldots, N$
    Для $i = k + 1, \ldots, N$
      $a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
    Увеличить i
  Увеличить j
Увеличить k

Доступ к элементам матрицы $\boldsymbol{A}$ производится по столбцам. Исключение по столбцам. Модификации немедленные.

1.2 Задание на лабораторную работу

1. Изучить теоретическую справку.
2. Реализовать представленные алгоритмы в виде функций с именами вида LUikj(), …

3. Для реализованных функций сравнить время, затрачиваемое на LU-разложение, для произвольно заданных матриц порядка 100, 300, …, 1500. Также сравнить с результатами встроенной функции lu(). Построить полученные зависимости.

4. Реализовать 2 функции для решения СЛАУ. Данный процесс состоит из решения двух уравнений с треугольными матрицами, сначала с ***L***, а затем с ***U***. Далее приведен пример функции для решения уравнения вида ***LY*** = ***B***. Аналогичную функцию для решения уравнения вида ***UX*** = ***Y*** с именем *backsubU*() требуется реализовать самостоятельно.

```
function Y = backsubL(A, B)
n = length(B);
Y = zeros(n, 1);
Y(1) = B(1);
for  k = 2:n
  Y(k) = (B(k)-A(k,k-1:-1:1)*Y(k-1:-1:1,1));
end
end
```

5. С помощью самой быстрой из функций для LU-разложения, реализованных ранее, решить тестовую СЛАУ (рисунок 1.3).

$$\begin{cases} 1{,}83x_1+4{,}34x_2-7{,}49x_3+11{,}07x_4=1 \\ -2{,}15x_1+4{,}94x_2-3{,}89x_3+6{,}48x_4=2 \\ -0{,}50x_1+1{,}94x_2-3{,}32x_3+4{,}33x_4=3 \\ -4{,}27x_1+8{,}45x_2-7{,}71x_3+12{,}30x_4=4 \end{cases}$$

Рисунок 1.3 – Тестовая СЛАУ

6. Оформить отчет. Оформление должно соответствовать ОС ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления».

2 ЧИСЛЕННЫЕ МЕТОДЫ ИНТЕГРИРОВАНИЯ

2.1 Краткая теоретическая справка

Анонимные функции

Анонимные функции (или лямбда-функции) в GNU Octave представляют собой способ создания функций без явного объявления имени. Они часто используются для определения коротких и простых функций, которые могут быть переданы в качестве аргументов другим функциям или использованы для выполнения вычислений в одной строке кода.

Анонимные функции создаются с использованием ключевого слова «@(аргументы) выражение», где:

– @(аргументы) – это спецификация аргументов функции (один или несколько), заключенных в круглые скобки.

– выражение – это математическое выражение, которое определяет, что делает функция. Выражение может содержать переменные и операторы, например

```
f = @(x) x^2; % Анонимная функция f(x) = x^2
```

Присвоение анонимной функции переменной и вызов функции с аргументом:

```
f = @(x) x^2;
result = f(3); % Вызов функции
result = 9;
```

Присвоение анонимной функции переменной и построение графика этой функции (рисунок 2.1):

```
f = @(x) exp(-x.^2).*log(x).^2;
x = 1 : 0.1 : 5; % задание интервала построения
plot(x,f(x)) % построение графика функции
```

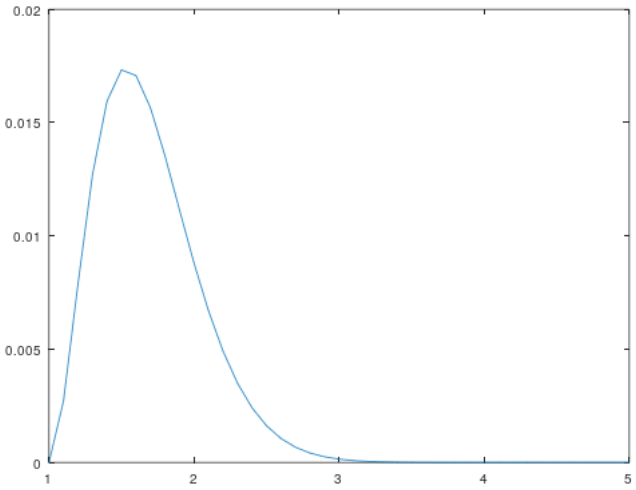

Рисунок 2.1 – Результат выполнения примера

Численные методы интегрирования

Интегрирование – это важная математическая операция, которая находит широкое применение во многих научных и инженерных областях. Существует ряд методов для вычисления приближенного значения определенного интеграла от функции на заданном интервале. Эти методы, называемые численными методами интегрирования, играют важную роль в различных областях науки, инженерии и финансах, где аналитическое вычисление интегралов может быть затруднительным или невозможным. Среди наиболее

распространенных методов численного интегрирования можно выделить метод прямоугольников, метод трапеций, метод Симпсона и метод Монте-Карло. Каждый из этих методов имеет свои особенности и применяется в зависимости от характеристик интегрируемой функции и требований к точности результата.

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 (рисунок 3), то метод будет заключаться в приближенном 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 (рисунок 2.2).

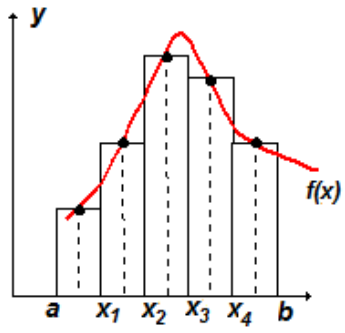

Рисунок 2.2 – К пояснению метода прямоугольников

Так, пусть требуется определить значение интеграла функции на отрезке $[a, b]$. Этот отрезок делится точками $x_0, x_1, \ldots, x_{n-1}, x_n$ на n равных отрезков длиной $h = (b - a) / n$, тогда искомый интеграл можно представить в виде

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

Ниже представлен алгоритм метода прямоугольников:

1. Определить функцию $f(x)$ и задать интервал $[a, b]$, на котором будет проводиться интегрирование;
2. Задать число разбиений n и вычислить шаг интегрирования h как $(b - a) / n$;
3. Инициализировать переменную `sum` с нулевым значением;
4. Задать цикл для каждого разбиения от $i = 1$ до n;
5. Для каждой итерации вычислить значение функции в точке $f(x_i - h / 2)$ и добавить это значение в переменную `sum`;
6. Умножить итоговое значение переменной `sum` на шаг интегрирования h;
7. Полученное значение `sum` является приближенным значением определенного интеграла функции $f(x)$ на интервале $[a, b]$.

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 (рисунок 2.3).

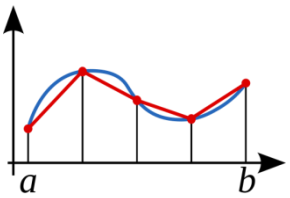

Рисунок 2.3 – К пояснению метода трапеций

Так, пусть требуется определить значение интеграла функции на отрезке $[a, b]$. Этот отрезок делится точками x_0, x_1, …, x_{n-1}, x_n на n равных отрезков длиной $h = (b - a) / n$, тогда искомый интеграл можно представить в виде

$$\int_a^b f(x)\,dx \approx h\left(\frac{f(x_0)+f(x_n)}{2}+\sum_{i=1}^{n-1} f(x_i)\right).$$

Ниже представлен алгоритм метода трапеций:

1. Определите функцию $f(x)$ и задайте интервал $[a, b]$, на котором будет проводиться интегрирование;
2. Задайте число разбиений n и вычислите шаг интегрирования h как $(b - a) / n$;
3. Инициализируйте переменную `sum` со значением $(f(x_0) + f(x_n)) / 2$;
4. Задайте цикл для каждого разбиения от $i = 1$ до $n - 1$;
5. Для каждой итерации вычислите значение функции в точке $f(x_i)$ и добавьте это значение в переменную `sum`;
6. Домножте итоговое значение переменной `sum` на шаг интегрирования h;
7. Полученное значение `sum` является приближенным значением определенного интеграла функции $f(x)$ на интервале $[a, b]$.

Метод Симпсона – метод численного интегрирования функции одной переменной, получивший свое название в честь британского математика Томаса Симпсона. Суть метода заключается в приближении подынтегральной функции на отрезке $[a, b]$ интерполяционным многочленом второй степени, то есть приближение графика функции на отрезке параболой (рисунок 2.4).

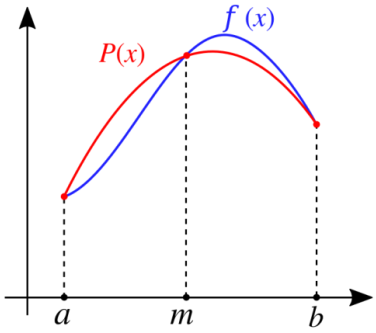

Рисунок 2.4 – К пояснению метода Симпсона

Так, пусть требуется определить значение интеграла функции на отрезке $[a, b]$. Использовав три точки отрезка интегрирования, можно заменить подынтегральную функцию параболой. Обычно в качестве таких точек используют концы отрезка и его среднюю точку. В этом случае формула имеет очень простой вид

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

Приближение функции одним полиномом на всем отрезке интегрирования, как правило, приводит к большой ошибке в оценке значения интеграла. Поэтому 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 элементарных отрезков. Значение исходного интеграла является суммой результатов интегрирования на составных отрезках

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

где $h = (b - a) / N$ – величина шага разбиения, а $i = 1,2$ означает, что индекс меняется от единицы с шагом равным двум.

Ниже представлен алгоритм метода Симпсона:

1. Определите функцию $f(x)$ и задайте интервал $[a, b]$, на котором будет проводиться интегрирование;

2. Задайте число разбиений N и вычислите шаг интегрирования h как $(b - a) / N$;

3. Инициализируйте переменную `sum` с нулевым значением;

4. Задайте цикл для каждого разбиения от $i = 1$ до $N - 1$ с шагом 2;

5. Для каждой итерации вычислите значения функции в точках $f(x_{i-1})$, $f(x_i)$, $f(x_{i+1})$ и согласно формуле (4) добавьте эти значения в переменную `sum`;

6. Домножьте итоговое значение переменной `sum` на $h / 3$;

7. Полученное значение `sum` является приближенным значением определенного интеграла функции $f(x)$ на интервале $[a, b]$.

Методы Монте-Карло – группа численных методов для изучения случайных процессов. Суть метода заключается в следующем: процесс описывается математической моделью с использованием генератора случайных величин, модель многократно обсчитывается, на основе полученных данных вычисляются вероятностные характеристики рассматриваемого процесса. Так, предположим, необходимо взять интеграл от некоторой функции на интервале $[a, b]$

$$\int_a^b f(x)\,dx.$$

Рассмотрим случайную величину u, равномерно распределенную на отрезке интегрирования $[a, b]$. Тогда $f(u)$ также будет случайной величиной, причем ее математическое ожидание выражается как

$$\mathrm{E}f(u) = \int_a^b f(x)\,\varphi(x)\,dx,$$

где $\varphi(x)$ – плотность распределения случайной величины u, равная $1 / (b - a)$ на интервале $[a, b]$. Таким образом, искомый интеграл выражается как

$$\int_a^b f(x)\mathrm{dx} = (b-a)\,\mathrm{E}f(u).$$

Однако математическое ожидание случайной величины $f(u)$ можно легко оценить, смоделировав эту случайную величину и посчитав выборочное среднее. Так, берем N точек, равномерно распределенных на $[a, b]$ для каждой точки u_i вычисляем $f(u_i)$. Затем вычисляем выборочное среднее $\frac{1}{N}\sum_{i=1}^{N} f(u_i)$. В итоге получаем оценку интеграла

$$\int_a^b f(x)\,dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(u_i).$$

Для малого числа измерений интегрируемой функции производительность Монте-Карло интегрирования гораздо ниже, чем производительность методов представленных выше. Тем не менее, в некоторых случаях, когда функция задана неявно, а необходимо определить область, заданную в виде сложных неравенств, такой метод может оказаться более предпочтительным.

Ниже представлен алгоритм метода Монте-Карло:

1. Определите функцию $f(x)$ и задайте интервал $[a, b]$, на котором будет проводиться интегрирование.

2. Сгенерируйте выборку из случайных величин x равномерно распределенных по интервалу $[a, b]$ (для этого можно воспользоваться функцией *rand*: $x = a + (b - a) \cdot rand(n, 1)$).

3. Вычислите значение функции в сгенерированных точках.

4. Вычислите выборочное среднее для полученных значений.

5. Домножте итоговое значение на ширину интервала ($b - a$).

6. Полученное значение является приближенным значением определенного интеграла функции $f(x)$ на интервале $[a, b]$.

2.2 Задание на лабораторную работу

1. Изучить теоретическую справку.

2. Выполнить реализацию методов численного интегрирования в виде функций, с аргументами: f – анонимная функция, по которой будет выполняться интегрирование, a и b – нижняя и верхняя границы интегрирования и n – число разбиений / случайный точек.

3. Построить график функции и найти значение интеграла в заданном интервале для функций:

а) $f(x) = x^5 - 6x^4 + 13x^3 - 12x^2 + 4x - 5$, интервал [–1, 4];

б) $f(x) = \sin(20x) + \frac{1}{2}e^{-10x^2}$, интервал [0,1];

в) $f(x) = \sin(x) \cdot (1 + 0{,}5 \cdot \cos(2x))$, интервал $[0, 2\pi]$.

Для проверки корректности решения использовать встроенную функцию численного интегрирования `integral(f,a,b)`.

4. На примере любой из представленных функций выполнить оценку точности вычисления интеграла с последовательным увеличением числа точек интегрирования N для всех реализованных методов интегрирования.

5. Выполнить оценку скорости вычисления всех реализованных методов интегрирования при последовательном увеличении числа точек интегрирования N.

6. Оформить отчет. Оформление должно соответствовать ОС ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления».

3 АППРОКСИМАЦИЯ МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ

3.1 Краткая теоретическая справка

Аппроксимация является важным инструментом в анализе данных, который позволяет описывать сложные явления или зависимости с помощью более простых моделей. Она широко используется в различных областях науки и инженерии для анализа экспериментальных данных, предсказания будущих значений и принятия решений на основе имеющихся данных. Аппроксимацией (приближением) функции $f(x)$ называется нахождение такой функции $g(x)$ (аппроксимирующей функции), которая была бы близка заданной (рисунок 3.1). При этом критерии близости функций могут быть различные. На практике наиболее широкое распространение получил принцип минимизации суммы квадратов разностей между фактическими значениями и значениями аппроксимирующей функции. Этот принцип лежит в основе метода наименьших квадратов (МНК).

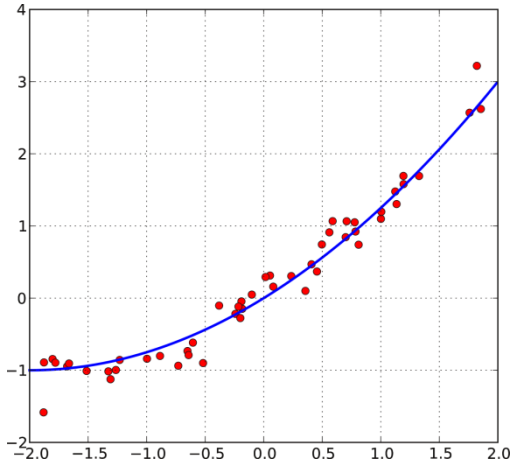

Рисунок 3.1 – Аппроксимации набора точек экспериментальных данных

Суть метода наименьших квадратов заключается в следующем. Пусть входная зависимость $y = f(x)$ представлена следующим набором данных $[x_1, x_2, \ldots, x_n]$ и $[y_1, y_2, \ldots, y_n]$. Для аппроксимации данной функции вводится некоторая скалярная функция $g(x, \boldsymbol{\alpha})$, которая определяется вектором неизвестных параметров $\boldsymbol{\alpha}$. Необходимо найти такой вектор $\boldsymbol{\alpha}$, при котором совокупность погрешностей $r_i = y_i - g(x_i, \boldsymbol{\alpha})$ была минимальной. Решением этой задачи является вектор $\boldsymbol{\alpha}$, который минимизирует функцию

$$S(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \left(y_i - g\left(x_i, \boldsymbol{\alpha}\right)\right)^2.$$

В простейшем случае $f(x) = \boldsymbol{\alpha}$, и тогда результатом МНК будет среднее арифметическое входных данных. Однако для более сложных случаев используется аппроксимация полиномами. Полином – функция, которая является суммой произведений различных степеней независимой переменной. Общий вид полинома степени n выглядит как

$$F(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_n x^n,$$

где $a_0, a_1, \ldots, a_n$ – неизвестные коэффициенты, которые определяются в процессе аппроксимации.

Степень полинома, которая будет использоваться для аппроксимации данных, определяется на основании физических или геометрических соображений. Например, экспериментальные точки могут быть нанесены на график, где можно приблизительно определить общий вид зависимости путем сравнения полученной кривой с графиками известных функций.

Рассмотрим общий случай аппроксимации полиномом степени m. Тогда формула для определения суммы квадратов отклонений примет вид

$$S(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \left(y_i - \left(a_0 + a_1 x_i + a_2 x_2 + \ldots + a_m x^m \right) \right)^2.$$

Известно, что в точке минимума все частные производные от S, по a_0, a_1, …, a_m равны нулю:

$$\frac{\partial S}{\partial a_0} = 0, \frac{\partial S}{\partial a_1} = 0, \ldots, \frac{\partial S}{\partial a_m} = 0.$$

Тогда приравнивая выражения к нулю и собирая коэффициенты при неизвестных, получим следующую систему линейных уравнений

$$\begin{cases} n a_0 + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + \ldots + a_m \sum_{i=1}^{n} x_i^m = \sum_{i=1}^{n} f(x_i), \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + \ldots + a_m \sum_{i=1}^{n} x_i^{m+1} = \sum_{i=1}^{n} x_i f(x_i), \\ \ldots \\ a_0 \sum_{i=1}^{n} x_i^m + a_1 \sum_{i=1}^{n} x_i^{m+1} + a_2 \sum_{i=1}^{n} x_i^{m+2} + \ldots + a_m \sum_{i=1}^{n} x_i^{2m} = \sum_{i=1}^{n} x_i^m f(x_i), \end{cases}$$

или в матричном виде $\mathbf{X}\boldsymbol{\alpha} = \mathbf{f}$:

$$\begin{pmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^m \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_i^m & \sum_{i=1}^{n} x_i^{m+1} & \cdots & \sum_{i=1}^{n} x_i^{2m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} f(x_i) \\ \sum_{i=1}^{n} x_i f(x_i) \\ \vdots \\ \sum_{i=1}^{n} x_i^m f(x_i) \end{pmatrix}.$$

Решением данной системы уравнений является вектор неизвестных параметров полинома – $\boldsymbol{\alpha} = [a_0, a_1, \ldots, a_n]$. Подставляя данные параметры в (2) определяется аппроксимирующая функция.

Весь процесс аппроксимации МНК можно представить в виде следующей последовательности действий:

1. Подготовка данных. Исходные данные представляются в виде двух векторов $x = [x_1, x_2, \ldots, x_n]$ и $y = [y_1, y_2, \ldots, y_n]$.

2. Определение степени аппроксимирующего полинома. Выполняется построение диаграммы рассеяния исходных данных (построение x и y на графике) и определение общий вид зависимости путем сравнения полученной диаграммы с графиками известных функций.

3. Формирование системы линейных уравнений. Вычисляется матрица X и вектор f из уравнения (6). Строки матрицы X содержат значения x_i в различных степенях для каждой точки данных, тогда как вектор f содержит произведения значений y_i и значений x_i в соответствующих степенях. (*Примечание: для вычисления* X *и* f *удобнее всего использовать матрично-векторные операции*).

4. Решение системы уравнений. Решением системы уравнений является вектор неизвестных параметров α аппроксимирующего полинома.

5. Построение аппроксимирующей функции. Задается интервал для построения графика полинома, для каждой точки которого вычисляется значение найденного полинома. После чего по полученным значениям выполняется построение графика.

6. Оценка качества аппроксимации. Оценивается качество аппроксимации путем сравнения предсказанных значений с фактическими данными, используя выражение $r_i = y_i - g(x_i)$.

3.2 Задание на лабораторную работу

1. Изучить теоретическую справку.

2. Выполнить аппроксимацию набора данных, представленного в таблице 3.1:

2.1 Исходя из диаграммы рассеяности входных данных, определить подходящую степень аппроксимирующего полинома.

2.2 Вычислить неизвестные коэффициенты аппроксимирующего полинома согласно МНК.

2.3 Построить график аппроксимирующего полинома и сравнить его с исходными данными.

2.4 Выполнить оценку качества аппроксимации.

3. Выполнить аппроксимацию набора данных, представленного в таблице 3.2:

3.1 Используя МНК, выполнить аппроксимацию набора данных для различной степени аппроксимирующего полинома $m = 1 \ldots 15$.

3.2 Построить графики аппроксимирующих полиномов для каждой степени m и сравнить их с исходной зависимостью

$$y(x) = \sin(20x) + \frac{1}{2}e^{-10x^2}.$$

3.3. Выполнить оценку качества аппроксимации для каждой степени m аппроксимирующего полинома.

4. Оформить отчет. Оформление должно соответствовать ОС ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления».

Таблица 3.1 – Набор данных 1

x	7	31	61	99	129	178	209
y	13	10	9	10	12	20	26

Таблица 3.2 – Набор данных 2

x	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
y	0,50	1,36	–0,42	–0,08	1,09	–0,50	–0,52	0,99	–0,29	–0,75	0,91

4 ОБРАБОТКА ИЗОБРАЖЕНИЙ

4.1 Краткая теоретическая справка

В информатике изображения рассматриваются, как прямоугольные матрицы, каждый элемент которых соответствует яркости пикселя изображения («pixel» от англ. «picture element»). Открытие изображений в средах научного программирования позволяют применять к ним средства, доступные для анализа матриц. Представление матрицы в виде изображения показано на рисунке (рисунок 4.1).

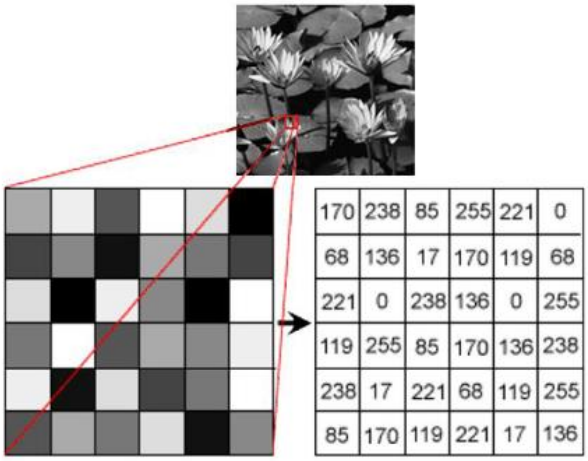

Рисунок 4.1 – Представление фрагмента серого изображения в виде матрицы

При этом следует помнить, что «серые», одноканальные изображения представляют собой одну матрицу, тогда как изображения «цветные» представляют собой три канала R, G, B – красный, зеленый, голубой, т. е. представляются тремя матрицами. Это нужно учитывать при открытии цветного изображения.

Для работы программы обработки изображений в GNU Octave, требуется скачать модуль *image* (доступен по ссылке https://sourceforge.net/projects/octave/files/Octave Forge Packages/Individual Package Releases/image-2.14.0.tar.gz/download?use_mirror=unlimited), а затем установить его с помощью консольной команды Octave

```
pkg install C:\my_folder\image-2.14.0.tar.gz
```

Здесь `C:\my_folder\` – папка, в которой находится скачанный архив с модулем `image`, `image-2.14.0.tar.gz` – имя файла архива.

После установки необходимо перезапустить программу. Загрузка модулей не автоматизирована и требует ввода консольной команды:

```
pkg load image
```

Для загрузки изображения в GNU Octave необходимо использовать функцию `imread()`, в качестве аргумента для которой передается путь к изображению. Если изображение и исполняемый файл находятся в одной папке, указывается только имя файла изображения. Для корректной работы модуля `image` имя файла и путь к нему не должны содержать кириллических символов. Так, например, откроем изображение «Lena.png»:

```
Img = imread('Lenna.png');
```

Изображение содержит три канала, а значит, открытие создаст трехмерную матрицу. Рассмотрим подробнее свойства переменной `Img` (рисунок 4.2).

```
  Attr   Name        Size                     Bytes  Class
  ====   ====        ====                     =====  =====
         Img       300x300x3                 270000  uint8

Total is 270000 elements using 270000 bytes
```

Рисунок 4.2 – Свойства переменной `Img`

Переменная `Img` представляет собой трехмерный массив размерностью 300x300 (300x300x3), занимающий в памяти 270000 байт, элементы массива представляют собой данные класса `uint8` (8-ми битное целое). В элемент класса `uint8` можно записать данные от 0 до $2^8 = 255$, т.е. всего 256 значений. Ясно, что этот тип данных наиболее приемлем для хранения в них матриц изображений. Для вывода изображения на экран используется следующая команда (результат выполнения команды представлен на рисунке 4.3).

```
imshow(Img);
```

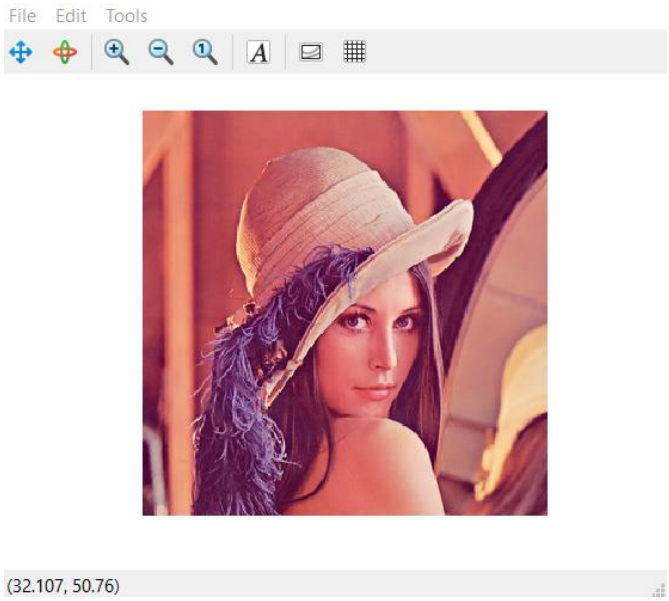

Рисунок 4.3 – Результат выполнения команды «imshow»

В GNU Octave доступны любые методы, применимые для преобразования изображений и, зачастую, используемые в графических редакторах, такие как обрезка, масштабирование, фильтры и т.д. В рамках данной лабораторной работы подробно рассмотрим фильтрацию изображений, т. е. процесс применения различных типов фильтров для уменьшения или устранения шума на изображении.

Шум изображения – это нежелательный побочный продукт захвата изображения, который скрывает желаемую информацию. Шум изображения может варьироваться от почти незаметных пятнышек на цифровой фотографии, сделанной при хорошем освещении, до оптических и радиоастрономических изображений, которые почти полностью представляют собой шум, из которых можно получить небольшой объем информации путем сложной обработки. Рассмотрим некоторые типы шумов.

Гауссов шум. Этот вид шума имеет вероятностное распределение интенсивности, подчиняющееся нормальному закону. Это означает, что большинство пикселей будет иметь шум около среднего значения, а экстремальные значения шума (очень высокие или очень низкие) встречаются реже. Гауссов шум часто возникает из-за естественных процессов, таких как тепловой шум в электронике или из-за случайных флуктуаций в процессе захвата изображения. Этот тип шума является аддитивным, то есть он складывается с исходным сигналом, и его воздействие не зависит от интенсивности самого сигнала. Пример изображения, подвергшегося воздействию шума Гаусса, представлен на рисунке 4.4.

Рисунок 4.4 – Пример изображения, подвергшегося воздействию шума Гаусса

Соль и перец – это вид шума на изображениях, при котором на фотографии появляются случайно распределенные белые и черные пиксели, напоминающие по внешнему виду соль и перец. Этот тип шума обычно возникает из-за быстрых скачков напряжения в процессе передачи изображений или из-за ошибок в сенсорах камеры, когда некоторые пиксели не корректно записывают освещенность. В отличие от шума Гаусса, который добавляет пикселям изображения равномерный шум, шум «соль и перец» проявляется в виде отдельных и резко выделяющихся точек, что делает его особенно заметным и иногда затрудняет обработку изображения. Пример изображения, подвергшегося воздействию шума «соль и перец», представлен на рисунке 4.5.

Рисунок 4.5 – Пример изображения, подвергшегося воздействию шума «соль и перец»

Дробовой шум, или шум Пуассона. Этот вид шума возникает из-за случайного количества фотонов или электронов, попадающих на датчик камеры. Это вызывает изменения в яркости и цвете пикселей, особенно при слабом освещении, когда фотонов мало. Дробовой шум является неотъемлемой частью любого процесса захвата изображения и считается непреодолимым, так как он прямо связан с фундаментальными свойствами света. Дробовой шум следует распределению Пуассона, которое, за исключением очень высоких уровней интенсивности, приближается к распределению Гаусса. Пример изображения, подвергшегося воздействию дробового шума, представлен на рисунке 4.6.

Рисунок 4.6 – Пример изображения, подвергшегося воздействию дробового шума

Мультипликативный шум (спекл-шум) образуется в результате когерентной суперпозиции от пространственно случайных источников рассеяния. Рассеянные волны накладываются друг на друга, вызывая тем самым появление спекл-шума на изображении. В отличие от аддитивного шума, который просто добавляется к сигналу, мультипликативный шум умножается на значение сигнала. Это означает, что яркие области изображения будут иметь больше шума по сравнению с темными областями. Пример изображения, подвергшегося воздействию мультипликативного шума, представлен на рисунке 4.7.

Рисунок 4.7 – Пример изображения, подвергшегося воздействию мультипликативного шума

Для снижения или полного удаления шума на цифровом изображении применяются различные виды фильтров, специализирующиеся для конкретных видов шумов. В рамках лабораторной работы рассмотрим базовые методы фильтрации.

Усредняющий фильтр – это базовый вид фильтра в обработке изображений, используемый для сглаживания и уменьшения шума. Он работает, заменяя значение каждого пикселя на среднее значение всех пикселей в некотором определенном окне вокруг данного пикселя. Например, если окно фильтра 3x3, значение центрального пикселя будет заменено на среднее значение этих 9 пикселей (включая сам центральный пиксель). Пример работы усредняющего фильтра представлен на рисунке 4.8.

Рисунок 4.8 – Пример фильтрации дробового шума усредняющим фильтром

Гауссов фильтр – это тип фильтра в обработке изображений, использующийся для сглаживания и уменьшения шума, особенно гауссова типа. В отличие от усредняющего фильтра, который применяет одинаковый вес ко всем пикселям в заданном окне, гауссов фильтр присваивает веса пикселям на основе гауссовой функции. Это означает, что центральные пиксели в окне имеют больший вес по сравнению с пикселями на краях, что обеспечивает более естественное и менее размытое сглаживание. Пример работы фильтра Гаусса представлен на рисунке 4.9.

Рисунок 4.9 – Пример фильтрации дробового шума фильтром Гаусса

Медианный фильтр – это тип нелинейного фильтра, используемый для уменьшения шума на изображениях, особенно эффективный против шума типа «соль и перец». В отличие от усредняющего или гауссова фильтра, медианный фильтр работает, заменяя значение каждого пикселя медианным значением (средним по величине) всех пикселей в определенной окрестности этого пикселя. Пример работы медианного фильтра представлен на рисунке 4.10.

Рисунок 4.10 – Пример фильтрации дробового шума медианный фильтром

Билатеральный фильтр – это продвинутый метод фильтрации изображений, который сочетает в себе пространственное и частотное сглаживание для уменьшения шума, сохраняя при этом резкие края. В отличие от усредняющего, гауссова или медианного фильтров, билатеральный фильтр учитывает разницу в интенсивности (яркости) пикселей, а также их пространственное расположение. Это позволяет ему сохранять края и детали изображения, в тоже время, уменьшая шум в однородных областях изображения. Пример работы билатерального фильтра представлен на рисунке 4.11.

Рисунок 4.11 – Пример фильтрации дробового шума билатеральным фильтром

Для оценки качества фильтрации изображений используются различные методы и метрики, которые позволяют как качественно, так и количественно анализировать результаты обработки. Эти подходы могут включать субъективную оценку, при которой человек оценивает улучшение визуального качества изображения, и объективную оценку, использующую числовые метрики для измерения изменений в изображении после фильтрации. Среди объективных метрик наиболее распространены:

Среднеквадратичная ошибка (MSE), измеряющая среднюю разницу между исходным и фильтрованным изображениями, что дает представление о степени искажения. Среднеквадратичная ошибка для двух монохромных изображений определяется как

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(I(i,j)-K(i,j))^2,$$

где m – число пикселей изображения по горизонтали;

n – число пикселей изображения по вертикали $I(i,j)$ – пиксель эталонного изображения;

$K(i,j)$ – пиксель исследуемого изображения.

Когда изображение содержит три цветовых канала (RGB), MSE рассчитывается отдельно для каждого канала, а затем результаты усредняются для получения общего значения MSE.

Пиковое отношение сигнала к шуму (PSNR), которое показывает отношение максимально возможной мощности сигнала к мощности искажающего шума, давая оценку качества изображения на основе его шума:

$$PSNR = 10\log_{10}(\frac{MAX_I^2}{MSE})$$

где MAX_I – это максимальное значение, принимаемое пикселем изображения эталонного изображения.

Зашумление изображения в GNU Octave выполнятся при помощи команды `imnoise`, из пакета `image`:

```
J = imnoise (I, name)
```

где `I` – оригинальное изображение;

`name` – вид шума (`'gaussian'`, `'salt & pepper'`, `'poisson'`, `'speckle'`);

`J` – зашумленное изображение.

Для фильтрации изображений используется команда `imsmooth`, в которой реализованы все рассмотренные виды фильтров:

```
J = imsmooth(I, name, options)
```

где `I` – зашумленное изображение;

`name` – название используемого фильтра (`'Average'`, `'Gaussian'`, `'Median'`, `'Bilateral'`);

`options` – настройки фильтра (для фильтров `'Average'`, `'Gaussian'` и `'Median'` задается третий параметр N, определяющий размер окна NxN в котором будет производиться фильтрация для каждого пикселя);

`J` – результат фильтрации.

Функции расчета MSE и PSNR предлагается реализовать самостоятельно.

4.2 Задание на лабораторную работу

1. Изучить теоретическую справку.
2. Загрузить исходное изображение в GNU Octave.
3. Исказить исходное изображение, используя шум Гаусса, шум «соль и перец», дробовой шум и мультипликативный шум.
4. Выполнить измерения значений MSE и PSNR для искаженных изображений. В качестве эталона должно быть использовано оригинальное изображение из пункта 2.
5. Выполнить фильтрацию искаженных изображений с использованием следующих фильтров: усредняющий фильтр, фильтр Гаусса, медианный фильтр и билатеральный фильтр.
6. Выполнить измерения значений MSE и PSNR для отфильтрованных изображений.
7. Повторить пункты 5 и 6, подбирая оптимальные параметры для усредняющего фильтра, фильтра Гаусса и медианного фильтра.
8. Оформить отчет. Оформление должно соответствовать ОС ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления».

5 РЕШЕНИЕ СЛАУ МЕТОДОМ ЯКОБИ

5.1 Краткая теоретическая справка

Метод Гаусса, как и метод, основанный на LU-разложении, относятся к прямым методам решения СЛАУ, т. е., если отбросить ошибку округления, решение, полученное посредством данных методов, является точным. Другую группу составляют итерационные методы, при помощи которых решение находится путем последовательных приближений. Если итерационный процесс сходится, то каждое последующее приближение уточняет предыдущее. Представим матрицу **A** системы **Ax** = **b** в виде

$$\mathbf{A} = \mathbf{L} + \mathbf{P} + \mathbf{D}, \tag{5.1}$$

где

$$\mathbf{D} = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 & 0 \\ 0 & a_{22} & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 0 & a_{nn} \end{pmatrix}, \mathbf{L} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ a_{21} & 0 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{n,n-1} & a_{nn} \end{pmatrix},$$

$$\mathbf{P} = \begin{pmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1,n-1} & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2,n-1} & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

Предполагаем, что $a_{ii} \neq 0$, $i = 1, 2, \ldots, n$. Тогда систему **Ax** = **B** с учетом (5.1) можно записать в виде:

$$\mathbf{Lx} + \mathbf{Px} + \mathbf{Dx} = \mathbf{b} \tag{5.2}$$

или

$$\mathbf{x} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{P})\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}\,. \tag{5.3}$$

Сравнивая систему **Ax** = **b** с (5.3), видно, что

$$\mathbf{B} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{P}),\ \mathbf{C} = \mathbf{D}^{-1}. \tag{5.4}$$

На основе формулы (5.3) записывается итерационный процесс метода Якоби:

$$\mathbf{x}_{k+1} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{P})\mathbf{x}_k + \mathbf{D}^{-1}\mathbf{b}\,. \tag{5.5}$$

Пусть $\mathbf{x}_k = \left(\mathbf{x}_1^{(k)}, \mathbf{x}_2^{(k)}, ..., \mathbf{x}_n^{(k)}\right)$. Тогда матричная формула (5.5) с учетом вида матриц **L**, **P** и $\mathbf{D}^{-1}$ записывается покомпонентно в виде

$$\begin{cases} x_1^{(k+1)} = \dfrac{b_1}{a_{11}} - \dfrac{a_{12}}{a_{11}} x_2^{(k)} - \dfrac{a_{13}}{a_{11}} x_3^{(k)} - \ldots - \dfrac{a_{1n}}{a_{11}} x_n^{(k)} \\ x_2^{(k+1)} = \dfrac{b_2}{a_{22}} - \dfrac{a_{21}}{a_{22}} x_2^{(k)} - \dfrac{a_{23}}{a_{22}} x_3^{(k)} - \ldots - \dfrac{a_{2n}}{a_{22}} x_n^{(k)} \\ \cdots \\ x_n^{(k+1)} = \dfrac{b_n}{a_{nn}} - \dfrac{a_{n1}}{a_{nn}} x_2^{(k)} - \dfrac{a_{n3}}{a_{nn}} x_3^{(k)} - \ldots - \dfrac{a_{n,n-1}}{a_{nn}} x_n^{(k)} \end{cases},$$

или

$$x_i^{k+1} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)},\ i = 1, 2, ..., n. \tag{5.6}$$

Тогда матрица **B** имеет вид:

$$\mathbf{B}=\begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} & \dots & -\frac{a_{1,n-1}}{a_{11}} & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & \dots & -\frac{a_{2,n-1}}{a_{22}} & -\frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & -\frac{a_{n3}}{a_{nn}} & \dots & -\frac{a_{n,n-1}}{a_{nn}} & 0 \end{pmatrix}.$$

Для сходимости метода Якоби достаточно, чтобы норма матрицы, обозначаемая как ||**B**||, была больше 1. Достаточным условием сходимости метода Якоби является

$$\sum_{\substack{j=1 \\ j\neq i}}^{n}\left|a_{ij}\right|<\left|a_{ii}\right|,\ i=1,2,\dots n.. \tag{5.7}$$

Неравенство (5.7) означает диагональное преобладание в исходной матрице **A**, которого следует добиться до вычислений согласно формуле (5.6).

Рассмотрим пример. Пусть дана система уравнений:

$$\begin{cases} 100x_1+30x_2-70x_3=60 \\ 15x_1-50x_2-5x_3=-40 \\ 6x_1+2x_2+20x_3=28 \end{cases},$$

где

$$\mathbf{A}=\begin{bmatrix} 100 & 30 & -70 \\ 15 & -50 & -5 \\ 6 & 2 & 20 \end{bmatrix},\ \mathbf{b}=\begin{bmatrix} 60 \\ -40 \\ 28 \end{bmatrix}.$$

В исходной матрице **A** диагонального преобладания нет. Поэтому до основных вычислений выполним преобразования. Сложим первые два уравнения:

$$\begin{cases} 115x_1-20x_2-75x_3=20 \\ 15x_1-50x_2-5x_3=-40 \\ 6x_1+2x_2+20x_3=28 \end{cases}.$$

В результате матрица СЛАУ преобразуется к матрице с диагональным преобладанием. Тогда итерационный процесс можно записать следующим образом:

$$\begin{cases} x_1^{(k+1)}=\frac{20}{155}+\frac{20}{155}x_2^{(k)}+\frac{75}{155}x_3^{(k)} \\ x_2^{(k+1)}=\frac{40}{50}+\frac{15}{50}x_1^{(k)}-\frac{5}{50}x_3^{(k)} \\ x_3^{(k+1)}=\frac{28}{20}-\frac{6}{20}x_1^{(k)}-\frac{2}{20}x_2^{(k)} \end{cases}$$

При этом матрица **B** примет вид:

$$\mathbf{B}=\begin{bmatrix} 0 & 0,17391 & 0,65217 \\ 0,3 & 0 & -0,1 \\ -0,3 & -0,1 & 0 \end{bmatrix}.$$

Нормы матрицы $||\mathbf{B}||_1 = 0,826 < 1$, $||\mathbf{B}||_2 = 0,75217 < 1$, что говорит о выполнении условий сходимости метода Якоби. В качестве начального приближения возьмем:

$$\mathbf{x}_0 = \mathbf{C} \cdot \mathbf{b} = \begin{bmatrix} 0{,}17391 \\ 0{,}8 \\ 1{,}4 \end{bmatrix}.$$

Результат вычисления 6 итераций приведен ниже:

$$\begin{cases} x_1^{(1)} = 1{,}2260 \\ x_2^{(1)} = 0{,}7122, \\ x_3^{(1)} = 1{,}2678 \end{cases} \begin{cases} x_1^{(2)} = 1{,}2246 \\ x_2^{(2)} = 1{,}0410, \\ x_3^{(2)} = 0{,}9610 \end{cases} \begin{cases} x_1^{(3)} = 1{,}1917 \\ x_2^{(3)} = 0{,}9662, \\ x_3^{(3)} = 1{,}0014 \end{cases}$$

$$\begin{cases} x_1^{(4)} = 0{,}9950 \\ x_2^{(4)} = 1{,}0574, \\ x_3^{(4)} = 0{,}9459 \end{cases} \begin{cases} x_1^{(5)} = 0{,}9817 \\ x_2^{(5)} = 1{,}0413, \\ x_3^{(5)} = 1{,}2829 \end{cases} \begin{cases} x_1^{(6)} = 0{,}9747 \\ x_2^{(6)} = 1{,}0039. \\ x_3^{(6)} = 0{,}9958 \end{cases}$$

Из полученных данных хорошо видна сходимость от итерации к итерации.

5.2 Задание на лабораторную работу

1. Изучить теоретическую справку.

2. Выполнить преобразования над матрицей **A** для удовлетворения условия сходимости метода Якоби следующих систем:

1) $\begin{cases} 10x_1 + x_2 - x_3 = 11 \\ x_1 + 10x_2 - x_3 = 10 \\ -x_1 + x_2 + 10x_3 = 10 \end{cases}$ 2) $\begin{cases} 20{,}9x_1 + 1{,}2x_2 + 2{,}1x_3 = 31 \\ 1{,}2x_1 + 21{,}2x_2 + 1{,}5x_3 = 10 \\ 2{,}1x_1 + 1{,}5x_2 + 19{,}8x_3 = 14 \end{cases}$ 3) $\begin{cases} 14x_1 + 3{,}5x_2 + 7{,}1x_3 = 7{,}14 \\ 5{,}1x_1 + 17x_2 + 1{,}5x_3 = 2 \\ 8{,}1x_1 + 0{,}1x_2 + 11x_3 = 3 \end{cases}$

3. Реализовать метод Якоби для решения СЛАУ с точностью ε = 0,001. Перед нахождением решения в программе выполнить проверку на соответствие исходных данных требованиям метода. При нахождении решения для *i*-ой итерации записать вектор $\mathbf{V}_i = [x_1, x_1, \ldots, x_k]$, где x_k – получаемое *k*-е решение системы уравнений. По достижении заданной точности построить график зависимости $\mathbf{V}_i$ от номера итерации *i*. Оценить по построенному графику скорость сходимости метода.

4. Оформить отчет. Оформление должно соответствовать ОС ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления».

6 ЗНАКОМСТВО С СИСТЕМОЙ SMATH STUDIO

6.1 Краткая теоретическая справка

Численное вычисление математических выражений

Для численного вычисления математических выражений в SMath Studio используется оператор численного вычисления (=). Для ввода оператора численного вычисления на клавиатуре используется символ = (равно). Оператор также можно ввести кнопкой = на панели Арифметика при помощи мыши.

Значение числового выражения получить в SMath Studio совсем просто. После ввода выражения нужно ввести оператор численного вычисления и все.

Пример 1. Найдем значение числового выражения:

$$–0{,}636 / 0{,}6 + 0{,}6 \cdot 0{,}1 \qquad (6.1)$$

Вводим выражение в SMath Studio. После ввода всего выражения при любом положении курсорной группы вводим оператор численного вычисления и сразу получаем ответ (рисунок 6.1).

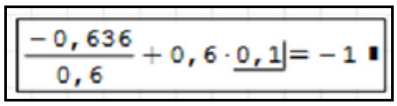

Рисунок 6.1 – Результат выполнения примера 1

Метка-заполнитель после результата предназначена для ввода наименования единицы измерения результата. Единицы измерения мы использовать не будем, поэтому завершаем ввод и получаем формулу.

В SMath Studio можно вычислить и численное значение математического выражения с переменными. Следует запомнить важное правило. Переменная, входящая в выражение, должна получить численное значение выше области, в которой вычисляется значение выражения с этой переменной. Если правило не выполнено, то вычисление значения выражения становится невозможным, поскольку вычисления на листе проводятся от области к области сверху вниз. Если выражение было обозначено, то для вычисления достаточно записать обозначение и оператор численного вычисления после него.

Пример 2. Найдем значение выражения:

$$x^2 + y^2 \text{ при } x = 3 \text{ и } y = 4. \qquad (6.2)$$

Сначала на рабочем листе должны быть заданы значения переменных. Ниже на листе вводим выражение, которое завершаем оператором численного вычисления, рисунок 6.2.

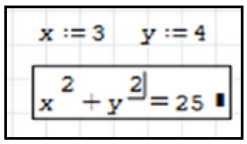

Рисунок 6.2 – Результат выполнения примера 2

Далее выделим на листе область с выражением и перетащим ее выше областей с присваиванием значений переменным. Вычисление значения выражения становится невозможным, поскольку вычисления на листе проводятся от области к области сверху вниз, рисунок 6.3.

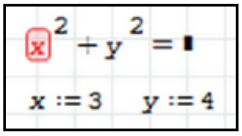

Рисунок 6.3 – Результат выполнения примера 2 с перенесенной областью с выражением

Численное решение уравнений и систем

В пакете SMath Studio численные решения уравнений и систем получают с помощью специальных функций. `solve(U(x);x)` – функция пакета SMath Studio, которая находит все действительные корни уравнения на интервале (–20; 20). Здесь `U(x)` – обозначает уравнение, `x` – имя переменной в уравнении. Если корней несколько, то функция показывает их в виде вектора. Если уравнение действительных корней не имеет, то формула обрамляется красной рамкой и под ней появляется надпись «Действительных корней нет».

Пример 3. Решим уравнение

$$x^2 + 5x - 4 = 0 \tag{6.3}$$

Вводим на рабочий лист пакета символы `solve(`, появляется метка-заполнитель для аргумента функции и вторая скобка. Вводим уравнение, в котором знак равенства вводится с клавиатуры комбинацией «Ctrl» + «=» или щелчком по кнопке = Булево «равно» на панели Булева. Далее вводим `;` и второй аргумент `x`. Затем вводим оператор численного вычисления (=), пакет выводит 2 корня, рисунок 6.4.

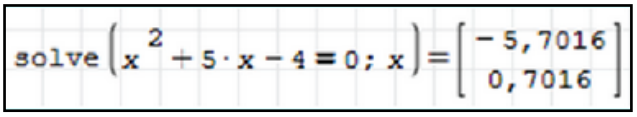

Рисунок 6.4 – Результат выполнения примера 3

`roots(S;Z;Zo)` – функция пакета SMath Studio, которая находит одно численное решение системы в форме вектора. Здесь `S` – запись системы в форме вектора, `Z` – вектор переменных системы, `Zo` – вектор начального приближения, который является примерной оценкой решения системы. В векторе `S` число компонент задается числом уравнений в системе. В векторах `Z` и `Zo` число компонент задается числом переменных. Если система имеет несколько решений, то функция ищет решение, ближайшее к `Zo`. Если решения системы сильно отличаются от `Zo`, то функция `roots()` может решение не найти и вывести сообщение «Действительных корней нет».

Пример 4. Решим систему уравнений:

$$\begin{cases} 2y = x + 2, \\ 2xy = 3. \end{cases} \tag{6.4}$$

Составим вектор `S` для заданной системы, в которой два уравнения. Вводим символы `S:=`, а затем вводим шаблон матрицы и меняем его размеры, чтобы получить один столбец с двумя элементами. Вводим уравнения (знак равенства – «Ctrl» + «=»), рисунок 6.5.

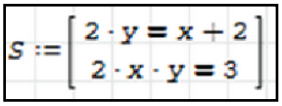

Рисунок 6.5 – Заполнение матрицы `S`

Вводим вектор переменных `Z` и вектор начального приближения `Zo` с двумя компонентами, так как в системе две переменных. Введем компонентами вектора `Zo` две единицы, рисунок 6.6.

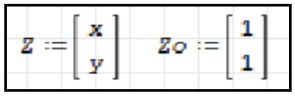

Рисунок 6.6 – Заполнение векторов Z и Zo

Далее вводим функцию roots() созданными ранее аргументами. Затем вводим оператор численного вычисления (=), рисунок 6.7.

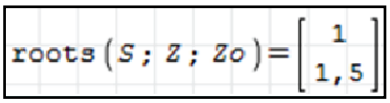

Рисунок 6.7 – Результат выполнения примера 4 с Zo = [1; 1]

Копируем Zo и roots() и выводим их ниже. Меняем компоненты Zo на 1 и –1, рисунок 6.8.

Рисунок 6.8 – Результат выполнения примера 4 с Zo = [1; -1]

Копируем Zo и roots() и выводим их ниже. Меняем компоненты Zo на –1 и –1, рисунок 6.9.

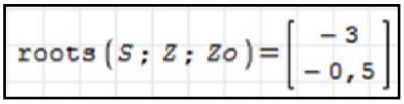

Рисунок 6.9 – Результат выполнения примера 4 с Zo = [-1; -1]

Снова копируем. Меняем компоненты Zo на –1 и 1 и получаем уже известное решение. В итоге получили два решения системы.

Для систем двух линейных уравнений с двумя переменными пакет всегда находит единственное решение, если оно есть, и в записи функции roots() можно не указывать аргумент Zo.

Символьное вычисление математических выражений

Символьное вычисление – это тождественное преобразование выражений с переменными. Для символьного вычисления математических выражений в SMath Studio используется оператор символьного вычисления →.

Оператор символьного вычисления на рабочем листе выглядит, как обычный знак равенства, а выводится на лист комбинацией «Ctrl» + «.» с клавиатуры или щелчком по кнопке → на панели Арифметика. Чтобы провести символьное преобразование, на лист вводится исходное математическое выражение, затем оператор символьного вычисления →, а пакет в ответ автоматически выводит математическое выражение-результат.

Пример 5. Преобразуем выражение

$$\frac{2ab-a}{4ab-a^2} \tag{6.5}$$

в тождественно равное. Вводим выражение на лист и добавляем оператор символьного вычисления →. После завершения ввода получаем, рисунок 6.10

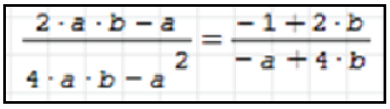

Рисунок 6.10 – Результат выполнения примера 5

Пакет провел сокращение рациональной дроби. Переставить слагаемые в результате пользователю не составит труда. Пакет настроен, прежде всего, на вынесение общего множителя, помнит формулы сокращенного умножения, сумму рациональных дробей приводит к общему знаменателю, но раскрывать скобки и приводить подобные члены не умеет, но вычислению это не мешает.

SMath Studio позволяет находить производные функций при помощи специальной конструкции, которая вводится на лист щелчком по кнопке  Дифференцирование на панели Функции.

Пример 6. Найдем производную функции

$$f(x) = 2x^3 - 4. \qquad (6.6)$$

Щелчком по кнопке Дифференцирование на панели Функции выводим на лист конструкцию с двумя метками-заполнителями. В правую метку-заполнитель вводим математическое выражение функции. Щелчком мыши выбираем нижнюю метку-заполнитель и в нее вводим имя переменной. Потом вводим оператор символьного вычисления → и завершаем ввод. Получаем выражение производной заданной функции, рисунок 6.11.

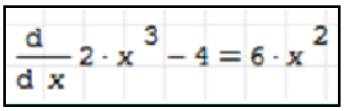

Рисунок 6.11 – Результат выполнения примера 6

Графическая область

Пакет SMath Studio позволяет создавать на рабочем листе графические области нескольких видов. Мы будем работать с графической областью «График двумерный», которая вставляется на рабочий лист командой меню Вставка → График → Двумерный (2D) или вводом с клавиатуры символа @.

Графическая область появляется на месте указателя места ввода в режиме ввода данных и отображает часть некоторой координатной плоскости (рисунок 6.12) Для ввода данных в графическую область служит метка-заполнитель под координатной сеткой.

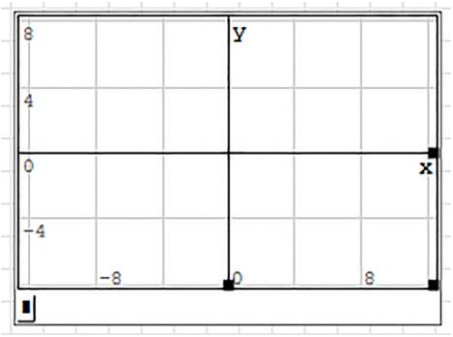

Рисунок 6.12 – Графическая область

Чтобы в SMath Studio построить график функции $y = f(x)$, нужно:

1) создать функцию $f(x)$
2) ниже вывести графическую область «График двумерный»;
3) в метку-заполнитель области ввести имя $f(x)$ с аргументом.

Имя функции может быть любым, но аргумент обязательно x. График будет построен автоматически.

Пример 7. Построим график функции

$$y = \sin(x). \qquad (6.7)$$

Определяем функцию на рабочем листе, рисунок 6.13.

$$f(x) := \sin(x)$$

Рисунок 6.13 – Ввод функции

Далее вставляем графическую область, а в метку-заполнитель области вводим `f(x)` и завершаем ввод, рисунок 6.14.

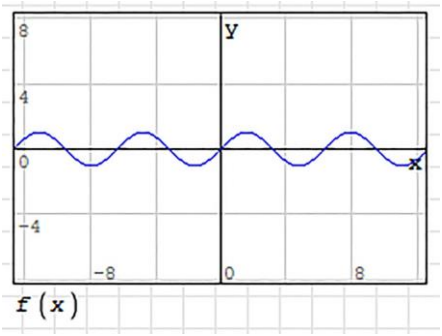

Рисунок 6.14 – Результат выполнения примера 7

Помимо аналитического задания графика, пользователь программы может задать его в виде матрицы, где каждый столбец будет содержать координаты точек для осей *X* и *Y* соответственно.

Пример 8. Построим в графической области ломаную из четырех отрезков, вершины которой имеют координаты (–8; –4), (–6; 4), (2; 2), (3; –3), (8; 6).

Строим матрицу `V` координат вершин ломаной. Вводим символы `V:=`, вызываем шаблон матрицы и изменяем его размеры на 2 столбца и 5 строк (по числу вершин). Последовательно заполняем матрицу (рисунок 6.15).

$$V := \begin{bmatrix} -8 & -4 \\ -6 & 4 \\ 2 & 2 \\ 3 & -3 \\ 8 & 6 \end{bmatrix}$$

Рисунок 6.15 – Заполнение матрицы **V**

Ниже вводим графическую область и в метку-заполнитель вводим имя V матрицы, рисунок 6.16.

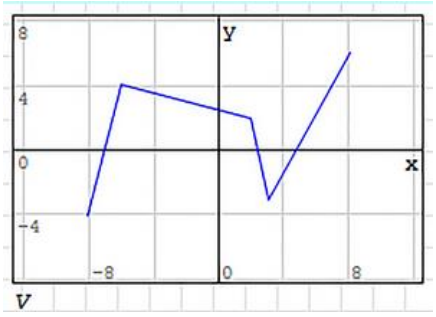

Рисунок 6.16 – Результата выполнения примера 8

Чтобы в заданных точках графической области «График двумерный» вывести текстовые надписи, нужно построить матрицу параметров текста, а затем добавить ее к выводимым в области графикам или фигурам. Матрица параметров текста имеет пять столбцов, а количество строк определяется числом точек вывода надписей.

Пример 9. Пусть построена матрица, рисунок 6.17.

$$P := \begin{bmatrix} -6 & 4 & "A" & 15 & "Green" \\ 2 & 6 & "B" & 15 & "" \\ 5 & -1 & "C" & 15 & "Red" \end{bmatrix}$$

Рисунок 6.17 – Матрица параметров текста

Три строки означают, что в графическую область будет выведено три надписи. Первый и второй элементы строки содержат координаты надписи. Третий элемент строки задает текст надписи (это большие буквы-обозначения). Четвертый элемент задает размер шрифта, а пятый – цвет. Пустые кавычки означают, что текст получит цвет по умолчанию – черный. Результат вывода показан на рисунке 6.18.

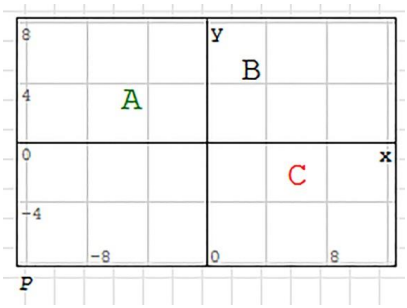

Рисунок 6.18 – Результат выполнения примера 9

Сложность в том, что обозначения часто нужно выводить в графическую область одновременно с выводом графика. В таких случаях в метку-заполнитель графической области вводят знак Алгебраическая система (последняя кнопка панели Функции). Под областью появляется конструкция, представленная на рисунке 6.19.

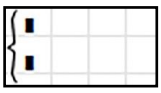

Рисунок 6.19 – Метка-заполнитель графической области

В метки-заполнители для одновременного отображения в области можно ввести обозначения функций и матрицы. Это позволяет вывести одновременно два графика функций или график и текстовые данные на одну область построения.

6.2 Задание на лабораторную работу

1. Изучить теоретическую справку.

2. Выполнить задачи, представленные далее. Все задания снабдить комментариями, используя команду Вставка → Текстовая область.

2.1 Вычислить числовые выражения:

$$1.\ -7{,}14{:}0{,}7+120\cdot 0{,}01;\quad 2.\ \left(\sqrt{27}+\sqrt{3}\right)^2;\quad 3.\ \left(3-\sqrt{2}\right)^2+\sqrt{\left(8-6\sqrt{2}\right)^2}.$$

2.2 Найти значения выражений при $x = 2$ и $y = –1$:

$$1.\ 3x^2y-7xy^2;\quad 2.\ \left(-\frac{3}{8}x^4y^5\right)^2;\quad 3.\ \sqrt{\left(x-3y\right)^2+\left(3x+y\right)^2};\quad 4.\ \frac{\left|4x^2-2y^2\right|}{3x^2+5y^2}.$$

2.3 Для каждого выражения создать функцию и вычислить ее значение при заданном значении переменной:

$$1.\ \frac{x^3-27x-54}{x^3},\ x=1{,}55;\quad 2.\ \frac{8t^2}{\left|t^2-6\right|},\ t=-0{,}75;\quad 3.\ \left|u-1\right|\cdot\ln u,\ u=1{,}3.$$

2.4 Решить уравнения:

1. $x^2-5x+4=0$; 2. $x^3+5x^2-4=0$; 3. $\sqrt{x-1}-\sqrt{2x+6}=6$; 4. $2^x3^{x+1}=108$.

2.5 Решить системы линейных уравнений с двумя неизвестными:

1. $\begin{cases} x+5y=7, \\ 3x-2y=4; \end{cases}$ 2. $\begin{cases} x^2-4y=-6, \\ 2x+y=4; \end{cases}$ 3. $\begin{cases} 2x+3y=5, \\ 3x-y=-9. \end{cases}$

2.6 Решить системы уравнений:

1. $\begin{cases} 2x^2+y=1, \\ 5x+3y=2; \end{cases}$ 2. $\begin{cases} 2x^2-2y^2-xy+2x-y+1=0, \\ 2x^2-y^2+xy+3y-5=0; \end{cases}$ 3. $\begin{cases} x^2-y=7, \\ 2x-3^y=-3. \end{cases}$

2.7 Разложить выражения на множители, используя символьное вычисление:

1. a^2-b^2; 2. a^3+b^3; 3. a^4-b^4; 4. a^4+b^4; 5. a^6-b^6.

2.8 Упростить выражения, используя символьное вычисление:

1. $\frac{15x^2y}{12xy^2}$; 2. $\frac{a^2+4a}{3}\frac{6}{a+4}-3$; 3. $5a+\frac{a^2-3a}{4a}\frac{8a}{a-3}$; 4. $\left(\left(\frac{a^2+b^2}{b}-a\right)\left(\frac{1}{b}+\frac{1}{a}\right)^{-1}\right)\left(a^2-b^2\right)$.

2.9 Вычислить значения математических выражений, используя численное и символьное вычисление:

1. $1-0,75$; 2. 1,1+1,232; 3. $\cos\left(\frac{\pi}{4}\right)$; 4. $\sin\left(\frac{\pi}{3}\right)$; 5. $\arcsin\left(\frac{1}{2}\right)$; 6. $\mathrm{tg}\left(\frac{\pi}{4}\right)$.

2.10 Найти производные функций:

1. $f(x)=x^5-4x^3-2x$; 2. $f(x)=x+2\cos(2x-1)$; 3. $f(x)=3^{tg(5x)}$.

2.11 Построить графики функций $y = f(x)$:

1. $f(x)=x^2$; 2. $f(x)=x^2+3x-5$; 3. $f(x)=\frac{1}{x}$; 4. $f(x)=tg\left(x+2\right)$; 5. $f(x)=\ln x$.

2.12 Вывести графическую область, а в ней треугольник с координатами вершин A(–8; 2), B(0; 12), C(7; –12) и обозначения его вершин буквами.

2.13 Вывести графическую область, а в ней прямоугольник и обозначения его вершин буквами.

3. Оформить отчет. Оформление должно соответствовать ОС ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления». В дополнении к отчету приложить файл с проектом SMath Studio.

7 ЗНАКОМСТВО С СИСТЕМОЙ MAXIMA

7.1 Краткая теоретическая справка

Основные арифметические операции и работа с переменными

Обозначения арифметических операций в Maxima ничем не отличаются от классического представления, используются математические знаки: `+ − * /` (рисунок 7.1). Возведение в степень можно обозначать тремя способами: `^` , `^^` , `**`. Извлечение корня степени *n* записывают, как степень `^^(1/n)`.

```
(%i2)  4**2;
(%o2)  16
(%i3)  4^2;
(%o3)  16
(%i4)  6!;
(%o4)  720
(%i5)  4^^1/2;
(%o5)  2
```

Рисунок 7.1 – Основные арифметические операции

Для хранения результатов промежуточных расчетов применяются переменные. Заметим, что при вводе названий переменных, функций и констант важен регистр букв, так переменные x и X – это две разные переменные.

Присваивание значения переменной осуществляется с использованием символа «`:`» (двоеточие), например `x: 5`;. Если необходимо удалить значение переменной (очистить ее), то применяется метод `kill` (рисунок 7.2):

`kill(x)` – удалить значение переменной `x`;

`kill(all)` – удалить значения всех используемых ранее переменных

```
(%i1)  x:3;y:5;
(%o1)  3
(%o2)  5
(%i3)  kill(all);
(%o0)  done
(%i1)  x/y;
(%o1)  x/y
```

Рисунок 7.2 – Пример использования функции `kill`

Кроме того, метод `kill` начинает новую нумерацию для исполняемых команд (обратите внимание, что ответом на команду `(%i3)`, приведенную выше, оказался ответ с номером ноль `(%o0) done`, и далее нумерация команд продолжилась с единицы).

В Maxima имеется достаточно большой набор встроенных математических функций. Вот некоторые из них (рисунок 7.3). Для записи функции необходимо указать ее название, а затем, в круглых скобках записать через запятую значения аргументов. Если значением аргумента является список, то он заключается в квадратные скобки, а элементы списка также разделяются запятыми.

Функции	Обозначение
Тригонометрические	sin (синус) cos (косинус) tan (тангенс) cot (котангенс)
Обратные тригонометрические	asin (арксинус) acos (арккосинус) atan (арктангенс) acot (арккотангенс)
Гиперболические	sinh (гиперболический синус) cosh (гиперболический косинус) tanh (гиперболический тангенс) coth (гиперболический котангенс)
Натурального логарифма	log
Квадратного корня	sqrt
Модуля	abs
Остатка от деления	mod
Минимума	mm(x1, …, xN)
Максимума	max(x1, …, xN)

Рисунок 7.3 – Встроенные математические функции Maxima

Вычисление выражений

Функция `rat ()` позволяет упростить рациональные выражения. Она приводит дроби к общему знаменателю и упрощает выражение, рисунок 7.4. Эта функция преобразовывает рациональное выражение к так называемой канонической форме (Canonical Rational Expression, CRE). То есть раскрывает все скобки, затем приводит все к общему знаменателю, суммирует и сокращает; кроме того, приводит все числа в конечной десятичной записи к рациональным.

Синтаксис: `rat (выражение);`

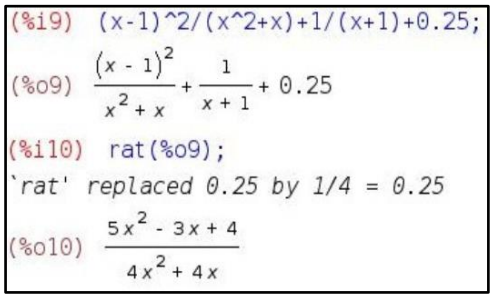

Рисунок 7.4 – Пример использования функции `rat`

Следующая функция раскрывает скобки в рациональном выражении и называется `ratexpand();`

Синтаксис: `ratexpand (выражение);`

Различая между функциями `rat()` и `ratexpand()` заключается в том, что после `ratexpand();` и в числителе, и в знаменателе дроби все скобки будут раскрыты, в случае же `rat();` слагаемые, где присутствуют, скажем, две переменных, будут сгруппирован, и одна из них будет вынесена за скобки (в документации такая форма записи называется «рекурсивной» (recursive). Также для фунции `ratexpand` существует опция `ratdenomdivide`; по умолчанию она установлена в `true`, это приводит к тому, что каждая дробь, в которой числитель является суммой, распадается на сумму дробей с одинаковым знаменателем. Если же сбросить эту опцию в `false`, тогда все дроби с одинаковым

знаменателем будут, напротив, объединены в одну дробь с числителем в виде суммы числителей изначальных дробей. То есть внешне результат будет в этом случае выглядеть почти так же, как и у функции `rat()`, рисунок 7.5.

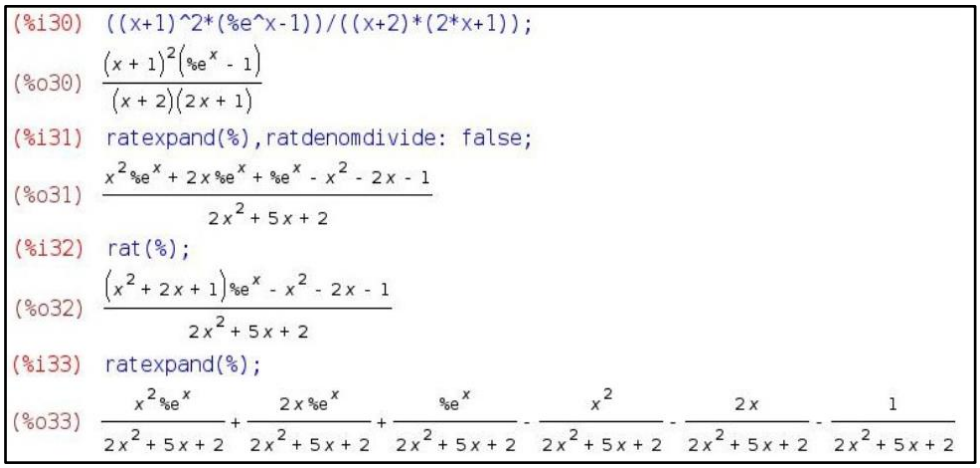

Рисунок 7.5 – Пример использования функций `rat` и `ratexpand`

Функция `factor()` позволяет разложить выражение на множители. Это полезно для поиска корней полинома и других задач. Пример использования функции `factor()` представлен на рисунке 7.6.

Синтаксис: `factor(выражение)`.

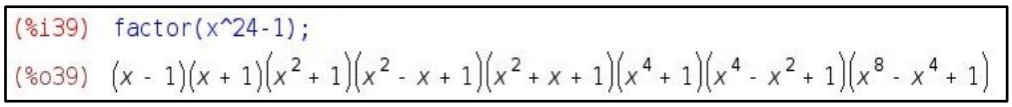

Рисунок 7.6 – Пример использования функций `factor`

Если функции `factor();` передать целое число, она разложит его на простые множители; если же передать рациональное число, на множители будут разложены его числитель и знаменатель, рисунок 7.7.

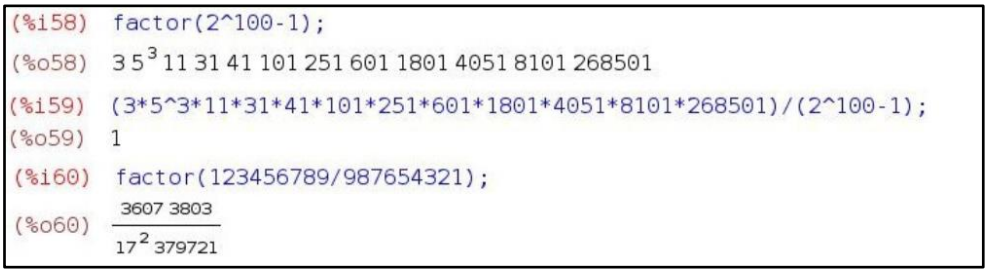

Рисунок 7.7 – Пример использования функций `factor`

У функций, используемых для преобразования тригонометрических формул, присутствует общая для всех приставка – `trig`. Функция `trigsimp()` используется для упрощения тригонометрических выражений. Она применяет различные тригонометрические тождества для упрощения выражения.

Производные и интегралы

Функция `diff();` позволяет найти производные как первого, так и более высоких порядков. При наличии у функции нескольких переменных можно найти частную производную по одной из них.

Синтаксис: `diff(функция, переменная, порядок производной);`

Пример: найти первую производную функции $y(x)=\frac{e^x}{x^2}$.

Сначала введем функцию: `f(x):=exp(x)/x^2;` (обратите внимание, что далее в строке `%i4`, в отличие от присвоения значения переменной, используется комбинация символов «`:=`» (двоеточие и равно)), а затем дается команда найти ее производную по переменной `x`.

Для вычисления производной вводится команда: `diff(f(x),x,1);` или `diff(y(x),x);`, рисунок 7.8. В случае первой производной ее порядок можно не указывать.

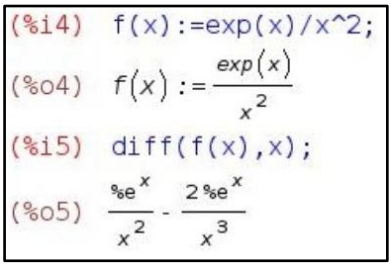

Рисунок 7.8 – Пример работы функции `diff`

Для нахождения неопределенного интеграла в качестве аргументов указывается функция и переменная интегрирования.

Синтаксис: `integrate(функция, переменная);`

Пример: вычислить интеграл от функции x^2+5x+3 по переменной x, рисунок 7.9.

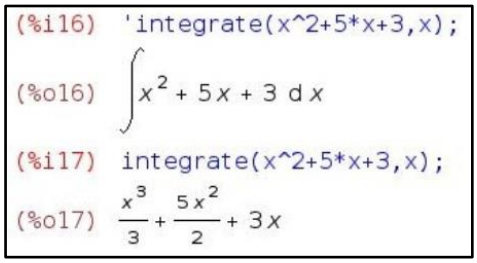

Рисунок 7.9 – Пример работы функции `integrate`

При нахождении значения определенного интеграла помимо рассмотренных параметров указываются пределы интегрирования. В качестве пределов интегрирования могут фигурировать бесконечность (`inf`) и минус бесконечность (`minf`).

Синтаксис: `integrate(функция, переменная, нижний предел, верхний предел);`

Пример: вычислить интеграл функции sin(x) по переменной x на отрезке от 0 до u.

Пример: вычислить интеграл от функции x^2+5x+3 по переменной x, рисунок 7.10.

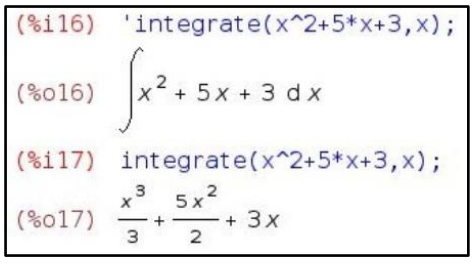

Рисунок 7.10 – Пример работы функции `integrate`

В случае, когда интеграл расходится, Maxima выдает сообщение «Integral is divergent». Например, `integrate(1/x, x, 0, inf);` выдаст именно такое сообщение.

Работа с матрицами

Для задания матрицы используется функция `matrix`. Рассмотрим основные действия с матрицами на примере. Для этого зададим две матрицы: **A** и **B**, рисунок 7.11.

```
(%i3) A: matrix([15,2],[-7,10]);
(%o3)  [ 15   2 ]
       [ -7  10 ]

(%i4) B: matrix([-6,4],[5,13]);
(%o4)  [ -6   4 ]
       [  5  13 ]
```

Рисунок 7.11 – Пример использования функций `matrix`

Пример поэлементного сложения, вычитания и умножения матриц на число представлен на рисунке 7.12.

```
(%i5) A+B;
(%o5)  [  9   6 ]
       [ -2  23 ]

(%i6) A-B;
(%o6)  [ 21  -2 ]
       [ -12 -3 ]

(%i7) k.A;
(%o7)  k . [ 15   2 ]
           [ -7  10 ]
```

Рисунок 7.12 – Пример поэлементного сложения, вычитания и умножения матриц на число

Вычисление матрицы, обратной данной можно выполнить, возведя искомую матрицу в –1 степень. Так же обратную матрицу можно получить с помощью функции `invert`, рисунок 7.13.

```
(%i9) A^^-1;
(%o9)  [  5/82   -1/82  ]
       [ 7/164   15/164 ]

(%i10) invert(matrix([15,2],[-7,10]));
(%o10) [  5/82   -1/82  ]
       [ 7/164   15/164 ]
```

Рисунок 7.13 – Нахождения обратной матрицы

Функции для работы с матрицами:
`determinant` – нахождение определителя матрицы, рисунок 7.14.

```
(%i11) determinant(matrix([15,2],[-7,10]));
(%o11) 164
```

Рисунок 7.14 – Нахождения обратной матрицы

`eigenvalues` – нахождение собственных значений матрицы, рисунок 7.15.

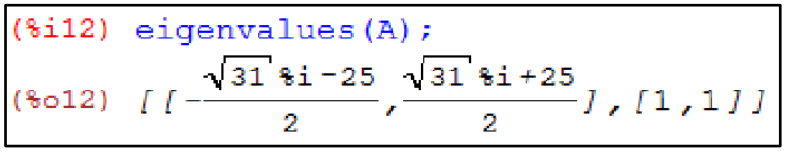

Рисунок 7.15 – Нахождения обратной матрицы

`transpose` – нахождение транспонированной матрицы.

Решение уравнений

Уравнения и системы уравнений решаются в Maxima одной и той же функцией `solve`. Функция `solve` в своем простейшем варианте, для решения одиночного уравнения, в качестве аргументов принимает либо уравнение и символ, относительно которого его надо решать, либо только уравнение, если символ в нем всего один. А возвращает список, состоящий из всех корней заданного уравнения, рисунок 7.16.

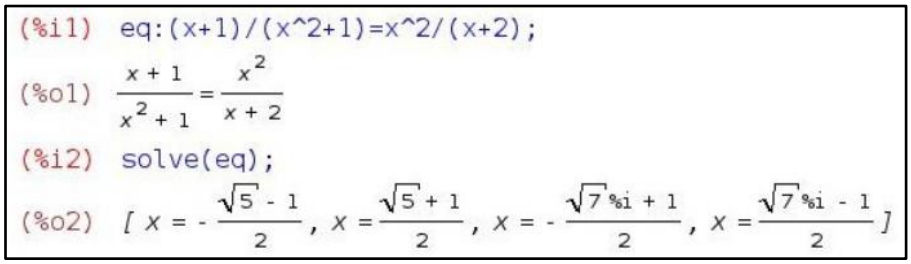

Рисунок 7.16 – Пример решения уравнения

Для решения систем уравнений синтаксис функции `solve` следующий:

`solve ([уравнение1, уравнение2, …], [переменная1, переменная2, …])`, рисунок 7.17.

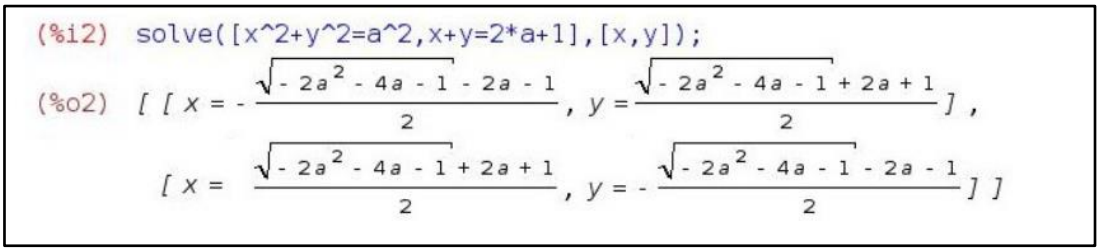

Рисунок 7.17 – Пример решения системы уравнений

Графики

Maxima предоставляет мощные инструменты для построения графиков функций и данных. В этом разделе мы рассмотрим, как использовать функции для построения различных типов графиков.

Функция `plot2d()` используется для построения двумерных графиков.

Синтаксис: `plot2d([функция, переменная, начальное значение, конечное значение], [дополнительные параметры]);`

Пример: Построить график функции $y = x^2$ на интервале от -10 до 10.

Для построения воспользуемся командой `plot2d(x^2 , [x, -10, 10]);` (рисунок 7.18).

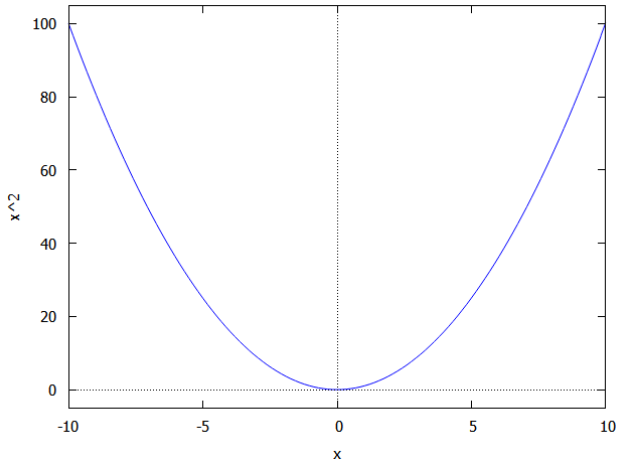

Рисунок 7.18 – Результат построения двумерного графика функции

Функция `plot3d()` используется для построения трехмерных графиков.

Синтаксис: `plot3d([функция, переменная1, начальное значение1, конечное значение1, переменная2, начальное значение2, конечное значение2], [дополнительные параметры]);`

Пример: Построить график функции $z = x^2+y^2$ на интервале от –5 до 5 по обеим осям.

Для построения воспользуемся командой `plot3d(x^2 + y^2, [x, -5, 5], [y, -5, 5])` (рисунок 7.19).

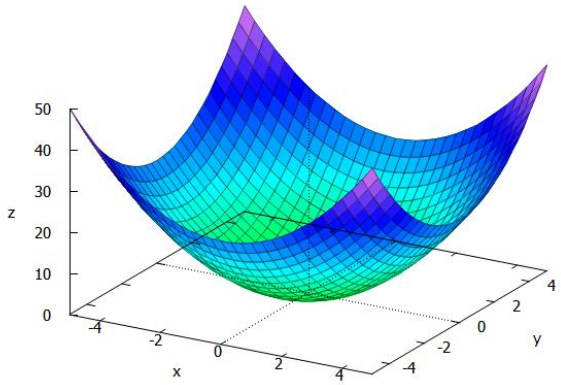

Рисунок 7.19 – Результат построения трехмерного графика функции

7.2 Задание на лабораторную работу

1. Изучить теоретическую справку.
2. Выполнить задачи, представленные далее.

Основные арифметические операции

а) Вычислить: $-7{,}14:0{,}7+120\cdot 0{,}01$.

б) Вычислить: $\left(\sqrt{27}+\sqrt{3}\right)^2$.

в) Вычислить: $\left(3-\sqrt{2}\right)^2+\sqrt{\left(8-6\sqrt{2}\right)^2}$.

г) Найти остаток от деления числа 17 на 3.

д) Найти наибольший общий делитель чисел 54 и 24.

е) Найти наименьшее общее кратное чисел 15 и 25.

ж) Найти действительную часть числа 2+3*i* (функция `realpart(z)`).

з) Найти модуль комплексного числа 3 +4*i* (функция `abs()`).

Работа с переменными

а) Создать переменные $x = 2$ и $y = 1$.

б) Найти значения выражения: $3x^2y - 7xy^2$.

в) Найти значения выражения: $\sqrt{(x-3y)^2 + (3x+y)^2}$.

г) Найти значения выражения: $\dfrac{|4x^2 - 2y^2|}{3x^2 + 5y^2}$.

Вычисление выражений

а) Упростить выражение $\dfrac{x^2 + 5x + 6}{x + 2}$.

б) Раскрыть скобки в выражении $(2x - y + 3)^3$.

в) Разложить на множители выражение $x^3 + 6x^2 + 11x + 6$.

г) Упростить выражение $(x^2 - 1)/(x - 1)$.

д) Преобразовать тригонометрическое выражение $\sin^2(x) + \cos^2(x)$.

Решение уравнений

Найти корни уравнений:

а) $x^2 + 2x + 1 = 0$.

б) $2x^2 + 3x - 2 = 0$.

в) $x^3 - 6x^2 + 11x - 6 = 0$.

г) $x^3 - 1 = 0$.

Работа с матрицами

а) Создать две матрицы $\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ и $\mathbf{B} = \begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}$.

б) Найти сумму и произведение матриц **A** и **B**.

в) Найти транспонированную матрицу **A**.

г) Найти обратную матрицу **A**.

д) Вычислить определитель матрицы **A**.

Производные и интегралы

а) Найти первую производную функции $f(x) = x^3 + 3x^2 + 3x + 1$.

б) Найти неопределенный интеграл функции $f(x) = 3x^2 + 6x + 3$.

в) Найти вторую производную функции $f(x) = \sin(x) \cdot \cos(x)$.

Решение систем уравнений

Решить системы уравнений:

а) $\begin{cases} 2x + 3y = 7, \\ 3x + 2y = 6. \end{cases}$

б) $\begin{cases} 115x_1 - 20x_2 - 75x_3 = 20, \\ 15x_1 - 50x_2 - 5x_3 = -40, \\ 6x_1 + 2x_2 + 20x_3 = 28. \end{cases}$

в) $\begin{cases} 1x_1 - 2x_2 - 3x_4 = -19, \\ -2x_1 + 2x_3 - 4x_4 = -22, \\ -3x_1 - 5x_2 + 4x_3 + x_4 = -23, \\ 4x_1 + 4x_2 - 1x_3 = 21. \end{cases}$

Графики

а) Построить график функции $y = x^2 - 4$ на интервале от –5 до 5.

б) Построить график функции $y = \sin(x)$ на интервале от -2π до 2π.

в) Построить 3D-график функции $z = x^2 + y^2$ на интервале от -5 до 5 по обеим осям.

г) Построить 3D-график функции $z = \sin(x) \cdot \cos(y)$ на интервале от $-\pi$ до π по обеим осям.

д) Построить кривую Лиссажу с параметрами $A = 1$, $B = 1$, $a = 3$, $b = 2$, $\delta = \pi / 2$, представленную уравнениями:

$$x(t) = A\sin(at + \delta),\ y(t) = B\sin(bt).$$

3. Оформить отчет. Оформление должно соответствовать ОС ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления». В дополнении к отчету приложить файл с проектом Maxima.

САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

ПЕРЕЧЕНЬ ВОПРОСОВ ДЛЯ САМОПОДГОТОВКИ (GNU OCTAVE)

1. Если **A** – квадратная матрица размерности *n*, **c** – вектор-строка из *n* элементов, то результат операции **A**.***c** – это
   а) квадратная матрица размерности *n*.
   б) число.
   в) вектор-строка размерности *n*.
   г) вектор-столбец размерности *n*.
2. Какие функции используются для построения трехмерных пространственных графиков?
   а) *surf*
   б) *plot3*
   в) *mesh*
   г) *plot*
3. Если **A** и **B** – квадратные матрицы размерности *n*, результат операции **A***B** – это
   а) квадратная матрица размерности *n*
   б) число
   в) вектор-строка размерности *n*
   г) вектор-столбец размерности *n*
4. Дана система линейных алгебраических уравнений в матричной форме **A***x=**b**. Выберите выражения для корректного вычисления столбца неизвестных **x**.
   а) **A\b**
   б) *inv*(**A**)***b**
   в) **A**/**b**
   г) **b** * *inv*(**A**)
5. *z*=4+*j*3. Результатом выполнения функции *abs*(*z*) будет
   а) 5
   б) 4
   в) 3
   г) 25
6. Зарезервированной переменной не является
   а) *pi*
   б) *i*
   в) *ans*
   г) *e*
7. Для переноса длинных формул на другую строку используется символ
   а) двоеточие
   б) точка с запятой
   в) многоточие
   г) запятая
8. Заданы векторы **a**=[2 4 5 6 8] и **b**=[4 6 9]. В результате выполнения некоторой операции получен вектор **c**=[2 5 8]. Какая операция была выполнена?
   а) c=unique(a, b)
   б) c=union(a, b)
   в) c=setdiff(a, b)
   г) c=intersect(a, b)

9. Как можно разбить графическое окно на шесть частей (три по горизонтали и две по вертикали) и в четвертое окно вывести график функции в полярных координатах?
   а) plot (x,y), subplot(2.3,4)
   б) subplot(2,3,4), polar (x,y)
   в) subplot(3,2,4), polar (x,y)
   г) polar (x,y), subplot(3,2,,4)
10. Функция *surf* строит
   а) закрашенную поверхность
   б) каркасную поверхность
   в) сглаженную закрашенную поверхность
   г) сглаженную каркасную поверхность

ПЕРЕЧЕНЬ ЭКЗАМЕНАЦИОННЫХ ВОПРОСОВ

1. Векторное и матричное исчисление в GNU Octave.
2. Операторы ветвления и циклы в GNU Octave.
3. Построение двумерных и трехмерных графиков в GNU Octave.
4. Встроенные функции в GNU Octave.
5. Пользовательские функции в GNU Octave.
6. Работа со строками в GNU Octave.
7. Структуры и ячейки в GNU Octave.
8. Работа с файлами в GNU Octave.
9. Принципы работы в системе компьютерной алгебры SMath Studio.
10. Принципы работы в системе компьютерной алгебры Maxima.

СПИСОК ЛИТЕРАТУРЫ

1. Программирование в системе MatLab: учебное пособие / составитель Е. Р. Урмакшинова. – Улан-Удэ: БГУ, 2017. – 46 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/154293 (дата обращения: 12.08.2024).

2. Шельмина, Е.А. Прикладная информатика: Методические указания по выполнению лабораторных и самостоятельных работ / Е.А. Шельмина. – Томск: ТУСУР, 2018. – 35 с. – URL: https://edu.tusur.ru/publications/7356 (дата обращения: 12.08.2024).

3. Коткин, Г.Л. Компьютерное моделирование физических процессов с использованием Matlab: учебное пособие для вузов / Г.Л. Коткин, Л.К. Попов, В.С. Черкасский. – 2-е изд., испр. и доп. – Москва: Издательство Юрайт, 2022. – 202 с. – Текст: электронный // Образовательная платформа Юрайт. – URL: https://urait.ru/bcode/494583 (дата обращения: 12.08.2024).

4. Электромагнитная совместимость: вычислительные методы: Учебно-методическое пособие / С.П. Куксенко. – Томск: ТУСУР, 2017. – 163 с. – URL: https://edu.tusur.ru/publications/7887 (дата обращения: 12.08.2024).