**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ………………………………………………………………………. 4

1.ВВЕДЕНИЕ В MATLAB……………………………………………………… 5

1.1. Рабочие панели MATLAB …………………………..……………... 5

1.2. Ввод и редактирование информации………….……………………. 7

1.3. Функции, формирующие векторы и матрицы……………..……….. 8

1.4. Извлечение и вставка частей матриц….…………………………... 11

1.5. Специфика выполнения арифметических операций….………….. 13

1.6. Действия над векторами и матрицами……………......…………… 14

1.7. Комплексные числа…………………………………......…………... 17

1.8. Ключевые слова……………………………………………….…….. 18

1.9. Логические операторы……………………………………………… 18

1.10. Элементарные математические функции………………………... 19

1.11. Оформление графиков…………………………………………….. 20

1.12. Основы программирования в среде MATLAB…………………... 26

1.12.1. Операторы управления вычислительным процессом….. 26

1.12.2. Создание файл — функций………………………………. 28

2. МЕТОДЫ ОПТИМИЗАЦИИ……………………………………………….. 30

2.1 Необходимое условие экстремума…………………………………. 30

2.2. Методы внутренней точки………………………………………….. 31

2.3. Метод Ньютона……………………………………………………… 32

2.4. Метод Ньютона с ограничениями типа равенств…………………. 33

2.5. Прямой метод внутренней точки…………………………………... 34

2.6. Прямо-двойственный метод внутренней точки…………………... 35

2.7. Линейное программирование………………………………………. 35

2.8. Квадратичное программирование………………………………….. 38

2.9. Решение задач нелинейной оптимизации в системе Matlab……... 44

2.10. Безусловная оптимизация…………………………………………. 44

2.11. Условная оптимизация…………………………………………….. 46

2.12. Упражнения ……………………………………………………… 47

3. ЛАБОРАТОРНЫЕ РАБОТЫ………………………………………………... 50

ЛАБОРАТОРНАЯ РАБОТА № 1……………………………………….. 50

ЛАБОРАТОРНАЯ РАБОТА № 2……………………………………….. 57

ЛАБОРАТОРНАЯ РАБОТА № 3……………………………………….. 61

ЛАБОРАТОРНАЯ РАБОТА № 4……………………………………….. 65

**Введение**

Лабораторные работы по курсу "Математические основы теории систем" позволяют на реальных практических задачах приобрести навыки моделирования и цифровой обработки сигналов.

Все лабораторные работы проводятся с использованием интерактивной системы инженерных и научных расчетов MATLAB. Матлаб в настоящее время по праву считается основным инструментом разработчика в самых различных областях человеческой деятельности, от космических аппаратов до финансово- экономических исследований. В настоящее время MATLAB принят официальным средством проектирования и оформления инженерной документации даже в таких областях, как автоматика, авиация, военная техника. Математический аппарат MATLAB опирается на матричные и векторные вычисления, что при определенном навыке предельно сокращает трудоемкость расчетов. Богатый арсенал графических средств максимально приспособлен к требованиям на инженерную документацию.

Освоение методов работы в среде MATLAB является второй задачей лабораторного практикума. Ограниченный объем лабораторных работ не позволяет рассчитывать на полное освоение системы, включающей несколько десятков пакетов, техническое описание каждого из которых содержит до 800 страниц.

Поэтому в настоящих Методических указаниях приведено краткое описание тех разделов MATLAB, которые будут использоваться на лабораторных занятиях: ввод и редактирование информации, графическое оформление работы, основы программирования, а также описание основных команд пакетов Signal Processing Toolbox, Simulink и Control System.

Освоение специфики работы с векторами, матрицами и их визуализация проводится на первых двух лабораторных занятиях.

Следующие три лабораторных работы посвящены изучению классических методов обработки сигналов: цифровой фильтрации, корреляционному и спектральному анализу сигналов.

**1.ВВЕДЕНИЕ В MATLAB**

**1.1. Рабочие панели MATLAB**

По умолчанию после запуска пакета *Matlab* на экране появляется комбинированное окно, включающее четыре наиболее важные панели — *Command Window (Окно команд), Command History (История команд), Workspace (Рабочее пространство) и Current Directory (Текущий каталог).* Две последние панели закрывают друг друга, и для выдвижения нужной панели на передний план следует щелкнуть по соответствующей вкладке. Три окна, вписанные в главное окно системы (рис. 1.1), "поставлены на якоря". Они передвигаются вместе с главным окном системы, вместе с ним изменяют свои размеры, границы между окнами можно передвигать. Каждое из них можно снять с якоря (кнопки c наклонной стрелкойUndock (Отстыковать), размещенные в правых верхних углах окон) и тогда оно может занимать автономную позицию на экране.

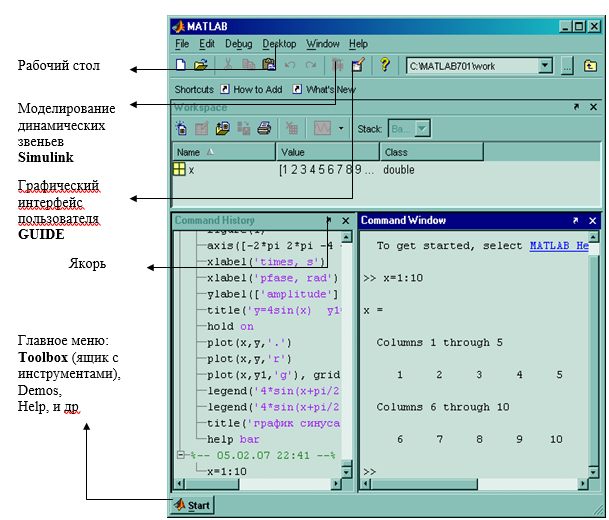


Рис. 1.1 – Главное окно системы

В командном окне можно обратиться за помощью по поводу того или иного термина с помощью одной из команд — doc, help или lookfor. Присутствие двух других окон во время сеанса работы загромождает экран, и их целесообразно уменьшить в размере или временно закрыть (такого рода операция выполняется не только кнопками, но и командами меню *Desktop* (Рабочий стол)). Для начинающего пользователя самым болезненным свойством этой панели является вывод на экран результатов всех выполняемых команд, если *после команды не поставлена точка c запятой.*

Самой используемой панелью является *Command Window* (Окно команд). В ней набираются команды пользователя, подлежащие немедленному исполнению. Результат операции будет выведен на экран, *если после команды не поставлена точка с запятой, предотвращающая вывод результата.*

Еще одна особенность *Matlab* является запрет на исправление предыдущих команд. Если у вас появилось такое желание, мышкой выделите нужную группу команд и перетащите всю группу вниз к галочкам приглашения к диалогу (»). Пока не нажата клавиша Enter, вы можете вносить любые исправления. Исправить предыдущую команду (или повторить одну из предыдущих) можно курсором, щелкнув по кнопке со стрелкой вверх. Еще более удобный способ — *писать всю программу в блокноте Editor Untitled,* создать который можно щелчком мышки на белом листе в левом углу командной строки меню. Блокнот избавит вас и от загромождающих экран промежуточных результатов, и от многочисленных ошибочных версий программы. Для исполнения программы блокнота достаточно нажать клавишу F5.

Окно *Command History (История команд)* хранит все команды, набираемые пользователем, однако в отличие от содержимого Command Window сюда не попадают сообщения системы и результаты вычислений. Эта информация может оказаться полезной для формирования программы, исполняемой в автоматическом режиме. Это окно — нечто среднее между блокнотом и Окном команд. С одной стороны, разработчики предусмотрели удобный способ перетаскивания мышкой команд, выделенных вCommand Historyв рабочее окно (не надо даже тащить: достаточно просто выделить и щелкнуть мышкой). С другой стороны, Command Historyхранит все ошибочные версии программы и по завершению работы вы не имеете возможности сохранить отлаженную версию программы.

Окно *Workspace*(Рабочее пространство) отображает текущий набор переменных, заведенных пользователем в командном окне. Здесь можно увидеть их имена (колонка Name (Имя)), значения скалярных переменных (колонка Value (Значение)) и тип представляемых данных (колонка Class (Тип данных)). Точно такую же информацию можно увидеть в командном окне после исполнения команды whos. В него удобно заглядывать в тех случаях, когда вам понадобится откорректировать значения элементов какого-либо массива с помощью Array Editor (Редактор массивов). Это новый инструмент, появившийся в 7-й версии. Для его вызова достаточно щелкнуть по имени переменной в поле Workspace „MATLAB"

Поскольку самой популярной ошибкой является неправильное задание размера массива (на третьем курсе многие еще верят, что вектор t=0:1:10; имеет размерность 1\*10), окно Workspace -хорошая подсказка.

**1.2. Ввод и редактирование информации**

MATLAB является системой, которая предназначена для осуществления сложных операций с векторами, матрицами и полиномами. Под вектором в MATLAB понимается одномерный массив чисел, а под матрицей — двумерный массив. При этом по умолчанию принято, что любая заданная переменная представляет собой вектор или матрицу. Например, отдельное заданное число система воспринимает как матрицу размером 1x1, а вектор-строку, состоящую из n элементов, — как матрицу размером1хn.

*Ввод значений векторов и матриц.* Начальные значения векторов можно вводить с клавиатуры поэлементно. Для этого в строке следует сначала указать имя вектора, потом поставить знак присваивания "**=**", далее — *открывающую квадратную скобку,* а за ней ввести заданные значения элементов вектора, разделенные пробелами или запятыми. Заканчивается строка закрывающей квадратной скобкой. Например, строка v=[1.2 -0.3 1.2е-5] задает вектор v, который состоит из трех элементов со значениями 1,2, -0,3 и 1,2е-5:

» V=[1.2 -0.3 -1.2е-5] (! точки с запятой нет, и МАТЛАБ выводит результат)

V = 1.2000 -0.3000 -0.0000

После ввода вектора система выводит его элементы на экран. В приведенном примере последний элемент представлен значением 0, поскольку в установленном по умолчанию формате Short отображается не более четырех цифр после десятичной точки.

Длинный вектор можно вводить частями, которые потом следует объединять с помощью операции объединения (**конкатенации**) векторов в строку: v=[vl\_v2] (между векторами либо пробел, либо запятая):

» Vl=[l 2 3];

» V2=[4 5 6];

» V=[V1,V2]

V =1 2 3 4 5 6

Язык MATLAB предоставляет возможность *сокращенного ввода вектора*, значения элементов которого составляют *арифметическую прогрессию*. Это могут быть, например, порядковые номера дискретных отсчетов или значения временных интервалов. Если обозначить начальное значение этой прогрессии (значение первого элемента вектора) как nz, конечное значение прогрессии (значение последнего элемента вектора) — как kz, а разность прогрессии (шаг) — как h, то вектор можно будет ввести с помощью короткой записи: v=nz: h: kz. Например, ввод строки v=1:5:31 даст результат:

v=1 6 11 16 21 26 31.

Если средний параметр (шаг) не указан, то он по умолчанию равен 1. Например, ввод выражения

» t = -5 : 2

приводит к формированию такого вектора:

ans

-5.000 -4.0000 -3.0000 -2.0000 -1.0000 0.0000 1.0000 2.0000.

Мы показали, как вводятся векторы-строки. Вектор-столбец вводится аналогично, но значения элементов отделяются точкой с запятой. Можно также использовать операцию транспонирования вектора: **'**.

В MATLAB значения *элементов матрицы* вводятся *в квадратных скобках*, по строкам. При этом элементы строки матрицы разделяются пробелом или запятой, а строки отделяются одна от другой точкой с запятой.

» А=[2 4 б 8 10; 5.5 6.3 -6.8 8 8.6]

А =

2.0000 4.0000 б.0000 8.0000 10.0000

5.5000 б.3000 -6.8000 8.0000 8.6000

**1.3. Функции, формирующие векторы и матрицы**

В MATLAB имеется несколько встроенных функций, которые позволяют формировать векторы и матрицы определенного вида. Описание этих функций и примеры их применения приведены ниже.

Функция **zeros (M,N)** создает матрицу размером M\*N с нулевыми элементами.

» zeros(3,5)

ans =

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

Функция **ones(M,N)** создает матрицу из единиц:

» ones(3,5)

ans =

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

Функция **eye(M.N)** создает единичную матрицу размером M\*N, то есть матрицу с единицами по главной диагонали и остальными нулевыми элементами.

» eye(3.5)

ans =

1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

Функция **rand(M.N)** создает матрицу размером M\*N из случайных чисел, равномерно распределенных в диапазоне от 0 до 1.

Функция **randn(M.N)** создает матрицу размером M\*N из случайных чисел, распределенных по **нормальному** (гауссову) закону с нулевым математическим ожиданием и стандартным (среднеквадратичным) отклонением, равным 1.

Функция **hadamard(N)** создает матрицу Адамара размером N\*N

» hadamard(4)

ans =

1 1 1 1

1 -1 1 -1

1 1 -1 -1

1 -1 -1 1

В языке MATLAB предусмотрено несколько функций, которые позволяют формировать одну матрицу на основе другой (заданной) или на основе некоторого заданного вектора. К таким функциям принадлежат следующие.

Функция **triu(А)** образует верхнюю треугольную матрицу на основе матрицы А путем обнуления ее элементов ниже главной диагонали.

»triu(A)

ans =

1 2 3 4 5 6

0 8 9 10 11 12

0 0 15 16 17 18

Функция **diag(x)** формирует или извлекает диагональ матрицы. Если х является вектором, то данная функция создает квадратную матрицу, у которой элементы вектора х размещены на главной диагонали.

» diag(v)

ans =

-5 0 0 0

0 6 0 0

0 0 7 0

0 0 0 4

Чтобы поместить элементы вектора на другую диагональ, при обращении к функции необходимо указать еще один параметр — номер диагонали, выраженный целым числом (при этом диагонали отсчитываются, начиная от главной по направлению вверх).

» diag(v,-1)

ans =

0 0 0 0 0

-5 0 0 0 0

0 6 0 0 0

0 0 7 0 0

0 0 0 4 0

Когда х является матрицей, функция diag(x) создает вектор-столбец, который состоит из элементов главной диагонали заданной матрицы х, например, для матрицы А, указанной в примере применения функции triu.

» diag(A)

ans = 1 8 15

Если при этом указать дополнительно номер диагонали, то можно получить вектор-столбец из элементов любой диагонали матрицы х, например:

» diag(A,3)

ans = 4 11 18

**1.4. Извлечение и вставка частей матриц**

Как и в других алгоритмических языках, к элементам матриц и векторов обращаются, используя индексы. В отличие от языка С, в системе MATLAB индексы отсчитываются от 1 и заключаются в круглые скобки — х(1), а(2,3). Довольно интересной особенностью языка MATLAB является возможность выполнения однотипных операций над подмножеством компонентов векторов или матриц. Для этого вместо индекса указывается диапазон индексов, разделяемых двоеточием.

Прежде всего отметим, что обращение к любому элементу заданной матрицы в MATLAB осуществляется путем указания (в круглых скобках, через запятую) после имени матрицы двух целых положительных чисел, определяющих соответственно номер строки и столбца матрицы, на пересечении которых расположен этот элемент. Допустим, мы имеем такую матрицу А:

» А — [1 2 3 4; 5 б 7 8; 9 10 11 12]

А =

1 2 3 4

5 6 7 8

9 10 11 12

Получить значение элемента этой матрицы, расположенного на пересечении второй строки с третьим столбцом, можно таким образом:

» А(2,3)

ans = 7

Если же нужно *поместить на указанное место* некоторое число, например, pi, выполните следующее:

» А(2,3) = pi;

А= 1.0000 2.0000 3.0000 4.0000

5.0000 6.0000 **3.1416** 8.0000

9.0000 10.0000 11.0000 12.0000

Такая операция называется **"вырезкой."** Вырезать(вставить) можно как один элемент, так и целую матрицу меньших размеров. Допустим, вам надо вырезать **всю** вторую строку матрицы

» А(:)

» B=A(2,**:**)

B= 5.0000 6.0000 3.1416 8.0000

Аналогично можно вставить матрицу В вместо любой строки А

» А(1,**:**) = В

А = 5.0000 6.0000 **3.1416** 8.0000

5.0000 6.0000 **3.1416** 8.0000

9.0000 10.0000 11.0000 12.0000

Если верхней границей изменения номеров элементов матрицы является ее размер в этом измерении, вместо него можно использовать служебное слово end. Например:

» A(2:end,2:end)

Эти операции удобно использовать для формирования матриц, большинство элементов которых одинаковы, в частности так называемых разреженных матриц, состоящих в основном из нулей. Для примера рассмотрим формирование разреженной матрицы размером 5x7 с единичными элементами в центре:

» А - zeros(5,7);

» В - ones(3,3);

» А(2:4,3:5) = В

А =

0 0 0 0 0 0 0

0 0 1 1 1 0 0

0 0 1 1 1 0 0

0 0 1 1 1 0 0

0 0 0 0 0 0 0

«Растянуть» матрицу А в единый вектор v можно с помощью обычной записи v=A(:). При этом создается вектор-столбец с количеством элементов m\*n.

Богатые возможности для формирования матриц большой размерности с повторяющимися блоками дает операция **кронекеровского произведения kron(A,B) .** При этой операции каждый элемент левой матрацы умножается на всю правую матрицу:

»А=[2 3;4 5]; B=[1 1;1 1];

»C= kron(A, B)

C= 2 2 3 3

2 2 3 3

4 4 5 5

4 4 5 5

**1.5. Специфика выполнения арифметических операций**

При составлении алгебраических выражений MATLAB разрешает использование традиционных знаков арифметических операций и символов специальных операций, список которых приведен в табл. 1.1.

*Таблица 1.1*

|  |  |
| --- | --- |
| **Символы** | **Выполняемое действие** |
| + | Покомпонентное сложение числовых массивов одинаковой размерности; добавление скалярной величины к каждому элементу массива |
| - | Покомпонентное вычитание числовых массивов одинаковой размерности; вычитание скалярной величины из каждого элемента массива |
| \* | Умножение матриц в соответствии с правилами линейной алгебры (чис­ло столбцов первого сомножителя должно быть равно числу строк вто­рого сомножителя); умножение всех компонентов массива на скаляр |
| .\* | Покомпонентное умножение элементов массивов одинаковой размерности |
| / | Деление скаляра на скаляр; покомпонентное деление всех элементов массива на скаляр: А/В = А\*В-[[1]](#footnote-1)= А \* inv(B)  (а, в — квадратные матрицы одного порядка) |
| ./ | Покомпонентное деление элементов массивов одинаковой размерности |
| \ | А\B= А-1\*B —левое матричное деление  (A— квадратная матрица) |
| .\ | А. \B— покомпонентное деление элементов B на A (левое поэлементное деление) |
| ^ | Возведение скаляра в любую степень; вычисление целой степени квадратной матрицы |
| = | Знак присваивания |
| ' | Транспонирование матрицы |
| [ ] | Задание числовых значений элементов матрицы; задание выходных параметров функции. |
| ( ) | Указание индексов (порядковых номеров) элемента вектора или матрицы;  задание порядка выполнения операций |
| . | Десятичная точка;  знак поэлементного выполнения операции |
| .. | Переход по дереву каталогов на один уровень вверх |
| … | Признак продолжения строки |
| , | Разделение элементов вектора;  отделение операторов |
| ; | Подавление вывода на экран;  отделение строк матрицы |
| : | Формирование векторов;  выделение строк, столбцов; |
| % | Указатель начала комментария |

**1.6. Действия над векторами и матрицами**

*Сложение векторов*. Как известно, суммироваться могут только векторы одного типа (те, которые являются либо векторами-строками, либо векторами-столбцами), имеющие одинаковую длину (одинаковое количество элементов). Если х и у -именно такие векторы, то их сумму, z, можно получить, введя команду v=x+y, например:

»х= [1 2 3] ; у= [4 5 6];

» v =x + y

v -= 5 7 9

*Вычитание векторов.* Это действие осуществляется аналогично, с помощью арифметического знака минус (-). Оно выполняется над векторами, имеющими одинаковую структуру (v=x-y).

» v = х — у

v = -3 -3 -3

*Транспонирование вектора*. Осуществляется с применением знака апострофа ('), который записывается сразу после имени транспонируемого вектора.

»х'

ans = 1 2 3

*Умножение вектора на число*. Осуществляется в MATLAB с помощью арифметического знака умножения (\*) таким образом: v=x\*r или v=r\*x, где г — некоторое действительное число.

» v -=2 \* х

v = 2 4 6

*Умножение двух векторов*. Определено в математике только для векторов одинакового размера (одинаковой длины) и лишь тогда, когда один из векторов — множителей — строка, а второй — столбец. Иначе говоря, если векторы х и у являются строками, то математический смысл имеют лишь две формы умножения данных векторов: m=х' \*у и v=x\*y'. При этом в первом случае результатом будет квадратная матрица, а во втором — число. Первая форма носит *название "внешнее произведение*", вторая- "*скалярное произведение*" В MATLAB умножение векторов задается посредством символа «\*», который записывается между множителями-векторами:

>>х= [1 2 3]; у= [4 5 6];

» v = х'\*у

V =

4 5 6

8 10 12

12 15 18

» v = х\*у'

v = 32

*Векторное произведение двух векторов* (для трехкомпонентных векторов). Для выполнения этой операции в MATLAB предусмотрена функция cross, которая позволяет найти векторное произведение двух векторов. Если заданы два трехкомпонентных вектора vl и v2, достаточно ввести команду cross(vl,v2).

» vl = [1 2 3]; v2 = [4 5 б]; » cross(vl,v2)

ans = -3 6 -3

*Поэлементное преобразование векторов.* В языке MATLAB предусмотрено выполнение ряда операций, позволяющих преобразовать заданный вектор в другой вектор, имеющий такой же размер и тип. Подобные операции, строго говоря, не является математическими. К таким операциям относятся, в частности, все операции, осуществляемые с помощью элементарных математических функций, имеющих один аргумент В языке MATLAB запись вида y=sin(x), где х — некоторый известный вектор, приводит к формированию нового вектора у (имеющего тот же тип и размер, что и вектор-аргумент), элементы которого равны синусам соответствующих элементов вектора-аргумента х.

»х= [-2,-1,0,1.2];

»у = sin (x)

y = -0.9093 -0.8415 0 0 .8415 0 .9093

Кроме описанных операций в MATLAB предусмотрено несколько операций поэлементного преобразования. Они задаются с помощью обычных знаков арифметических операций и применяются к векторам одинакового типа и размера. Результатом их является вектор аналогичного типа и размера.

* Добавление числа к каждому элементу (вычитание числа из каждого элемента) вектора. Осуществляется с помощью символа **«+» («-»)**;
* Поэлементное умножение векторов. Производится с помощью комбинации символов **«.\*»,** которые записываются между именами перемножаемых векторов. В результате получается вектор, каждый элемент которого является произведением соответствующих элементов векторов-«сомножителей»;
* Поэлементное деление векторов. Осуществляется с помощью комбинации символов **«./»**. В результате получается вектор, каждый элемент которого является частным от деления соответствующего элемента первого вектора на соответствующий элемент второго;
* Поэлементное деление векторов в обратном направлении. Осуществляется с помощью комбинации символов **«.\.»** В результате получают вектор, каждый элемент которого является частным от деления соответствующего элемента второго вектора на соответствующий элемент первого;
* Поэлементное возведение в степень. Осуществляется с помощью комбинации символов **«.^»**. В результате получается вектор, каждый элемент которого является соответствующим элементом первого вектора, возведенным в степень.

**1.7. Комплексные числа**

Константам **i** и **j** первоначально присваивается значение, равное sqrt(-l). Они используются для ввода комплексных чисел.

При вводе комплексных чисел допустимы следующие формы записи:

3+2i, 3+2\*i, 3+2j, 3+2\*j, 3+2\*sqrt(-1), complex(3,2).

Все формы представления соответствуют одному и тому же комплексному числу 3 + 2i.

Следует отметить, что с символами i и j могут быть ассоциированы и другие величины — например, переменная цикла или индекс элемента массива, и в этом случае они *временно* теряют значение мнимой единицы. Поэтому в системе MATLAB 6.5, где используется ускоритель JIT-accelerator, следует понимать, какой способ формирования комплексного числа является самым быстрым.

Выполним оценку времени вычисления 1 млн. одинаковых комплексных чисел, формируемых на основе четырех различных форм записи. Оценка получена усреднением по 101 реализации на компьютере с тактовой частотой 1.7 ГГц:

|  |  |
| --- | --- |
| Способ вычисления | Длительность, с |
| **complex(3,2)** | 0.05 ± 0.006 |
| 3 + 2j | 0.10 ± 0.01 |
| 3 + 2\*sqrt(-l) | 1.50 ± 0.08 |
| 3 + 2\*j | 8.40 ± 0.06 |

Из анализа этой таблицы следует, что в пересчете на одну операцию время вычисления с помощью встроенной функции **complex**:

* в 2 раза быстрее, чем с использованием константы j;
* в 30 раз быстрее, чем с использованием М-функции sqrt;
* в 168 раз быстрее, чем при умножении на переменную j, которую необходимо постоянно сверять со списком идентификаторов.

При использовании комплексного числа в математических выражениях для устранения неоднозначности его заключают в круглые скобки.

Функции **real** и **imag** позволяют получить вещественную и мнимую часть комплексного числа, а функции **abs** и **angle** — соответственно модуль и фазу (в радианах)

Для получения комплексно-сопряженного числа используется функция **conj.**

**1.8. Ключевые слова**

В языке MATLAB зарезервированы следующие 17 ключевых слов, которые используются при формировании операторов: *'break','case','catch','continue','else','elseif','end','for','function','global', 'if','otherwise','persistent','return','switch','try','while'.*

Эти слова можно использовать только по прямому назначению при построении конструкций языка. Их нельзя применять для иных целей, это будет порождать сообщения об ошибке.

**1.9. Логические операторы**

Для сравнения элементов массивов предусмотрены следующие операторы:

< меньше

<= меньше или равно

> больше

>= больше или равно

= = тождественно равно

~ = не равно

При сравнении массивов, допускающих операцию сравнение их элементов, образуется массив логических значений 1 на позициях, где проверяемое отношение выполняется, и 0 — где оно ложно. Если элементы массивов имеют значения 1 и 0 (логические массивы), для них предусмотрены логические операции (табл. 1.2):

*Таблица 1.2*

|  |  |  |
| --- | --- | --- |
| **Логический оператор** | **Математическая функция** | **Функция MATLAB** |
| Отрицание | not(A) | ~A |
| Логическое И (конъюнкция) | and(A,B) | A&B |
| Логическое ИЛИ (дизъюнкция) | or(A,B) | A|B |
| Исключающее ИЛИ (сложение по модулю 2) | xor(A,B) | xor(A,B) |

*Продолжение табл. 1.2*

|  |  |  |
| --- | --- | --- |
| Проверка истинности всех элементов |  | all(A<=>B) |
| Проверка истинности хотя бы одного элемента |  | any(A) |

**1.10. Элементарные математические функции**

Полный перечень всех математических операций можно найти в справочной системе Help.

Если вас интересует процедура выполнения конкретной операции, например, вычисление синуса, наберите команду help, задав в качестве параметра ключевое слово:

>> help sin

Если названия команды у вас нет, воспользуйтесь Help-навигатором. В его левой панели имеются 4 вкладки:

* Contents (Содержание) — оглавление доступных разделов;
* Index (Указатель) -набор ключевых слов, упорядоченных по алфавиту;
* Search (Поиск) — окно поиска ключевого слова;
* Demos (Примеры) — оглавление тестовых примеров.

В приведенной ниже таблице 1.3 даны только часто встречающиеся элементарные функции

*Таблица 1.3*

|  |  |
| --- | --- |
| **Категория функций** | **Наименование функций** |
| Тригонометрические(аргумент в радианах) | cos, cot, csc, sec, sin, tan cosd, cotd |
| Тригонометрические (аргумент в градусах) | cscd, secd, sind, tand acos, acot, acsc |
| Обратные тригонометрические (результат в радианах) | asec, asin, atan, atan2 acosd, acotd |
| Обратные тригонометрические (результат в градусах) | acscd, asecd, asind, atand |
| Гиперболические | cosh, coth, csch, sech, sinh, tanh acosh |
| Обратные гиперболические | acoth, acsch, asech, asinh, atanh |
| Степени логарифмы, корни | exp, expml, log, log1p, log2, loglO, nextpow2, pow2, reallog, realsqrt, sqrt |

*Продолжение табл. 1.3*

|  |  |
| --- | --- |
| Округления | ceil, fix, floor, round |
| Наибольший общий делитель | gcd |
| Наименьшее общее кратное | lem |
| Модуль числа | ab |
| Знак числа | sign |
| Остаток от деления с учетом знака делимого | mod |
| Остаток от деления | rem |
| Разложение числа на простые множители | factor |
| Вычисление факториала | factorial |

**1.11. Оформление графиков**

График- наглядное отображение *результатов* вашей работы. За 2 часа лабораторных занятий вы создадите не менее 20 графиков. Если они не будут снабжены заголовками, названиями координатных осей, сеткой координатных линий, легендой с наименованием каждой из совмещенных на одном графике кривых, считайте, что работали вы зря. Поэтому сразу привыкайте к полному и тщательному оформлению графиков.

Основная функция построения графика одной переменной — **plot(x,y).**

Пример:

>> x = -4\*pi : pi/100 : 4\*pi;

>> y= 3\* sin(x+ pi/2);

>> **plot (x,y)**

Можно и проще — plot(y). Но если аргумент не указан в явном виде, по умолчанию в качестве аргумента принимается номер отсчета вектора y.

Масштаб по оси x потерян, теперь это просто картинка. Сравните 2 графика (рис. 1.2):

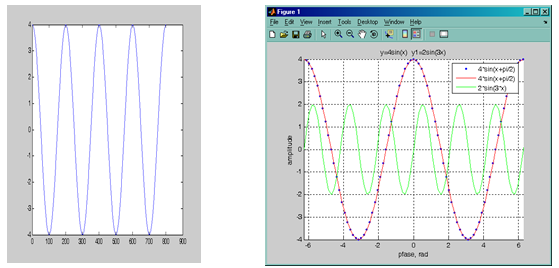


Рис. 1.2 - Графики

Это далеко не все возможности МАТЛАБа. Разберем этот пример (комментарии в МАТЛАБ выделяются знаком %)

x=-4\*pi:pi/20:4\*pi; % задание аргумента (отсчетов фазы)

y=4\*sin(x+pi/2); % первая функция

y1=2\*sin(3\*x); % вторая функция

figure(1) % если не писать эту команду, все графики будут

% выводиться в одно окно; при этом предыдущий график будет потерян

axis([-2\*pi 2\*pi -4 4]) % задание масштаба осей по форме [xmin xmax ymin ymax]

xlabel('pfase, rad') % наименование горизонтальной оси

ylabel('amplitude') % наименование вертикальной оси

title('y=4sin(x) y1=2sin(3x)') % наименование графика

hold on % команда включения наложения (совмещения) графиков

plot(x,y,'.') % первый график строится в виде точек (вид линии указан в апострофах ' .')

plot(x,y,'r') % точки графика соединены прямыми красного цвета ('r')

plot(x,y1,'g'), grid % теперь цвет линии зеленый и на график наложена сетка(grid)

legend('4\*sin(x+pi/2)','4\*sin(x+pi/2)','2\*sin(3\*x)') % формирует окно в правом углу графика с названиями и видом линий всех переменных.

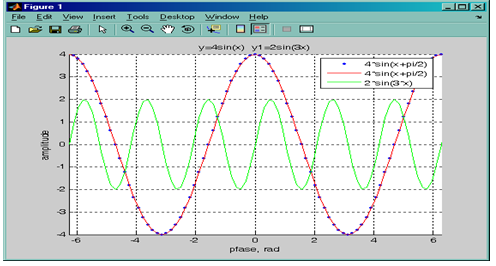


Рис. 1.3 – Результат работы программы

В Word первый график скопирован из меню Figure(1) командой Edit►Copy Figure, а второй график скопирован в буфер клавишами <Alt> +< Print Screen> и вставлен клавишами <Ctrl>+<V>.

Еще одна маленькая хитрость: все надписи в заголовках и на осях графика при попытке ввести русский шрифт превращаются в абракадабру. Но выход есть. Выполните команду меню View►Property Editor (Вид►Редактор свойств). К графическому окну пристыкуется редактор свойств Property Editor. Этот редактор может редактировать либо оси (Property Editor -Axes), либо линии, либо текст (Property Editor-text): его переключение производится щелчком левой кнопки мыши на соответствующем объекте графика. В Property Editor-text произведите замену шрифта в линейке Font c Helvetica на Courier или Arial Cyr, и можете писать по-русски.

Ниже приведены наиболее часто употребляемые команды графики:

**fplot(y,[xmin,xmax])** Отличается от команды plot неравномерным расположением отсчетов на графике: МАТЛАБ сам выбирает их исходя из заданной точности. Поэтому вторым аргументом этой функции пишется диапазон 'x'.

**plotyy(x1,y1,x1,y2)** На график выводится **две** вертикальных оси (слева и справа) с масштабами, нормированными по y1 и y2

**bar(x,y,h)** Построение столбцовой диаграммы. Третий аргумент определяет ширину столбцов. По умолчанию h =0.8

**contour(x,y,k)** Cтроит линии равного уровня. Количество уровней задается параметром k.

**subplot(n,m, k)** Делит графическое окно на n строк и m столбцов. Текущий график размещается в окне с номером k (если считать слева-направо)

**semilogx(x,y)** График в логарифмическом масштабе по одной из осей (x). По второй оси масштаб линейный

**loglog(x,y)** График с двойным логарифмическим масштабом

**pie(x)**  Круговая диаграмма

**plot3(M)** Трехмерный график. Аргументом может быть матрица или три вектора: (Z,X,Y) где Z=f(X,Y)

**pie3(X, n)** Трехмерная круговая диаграмма с отделенным сектором, стоящим на позиции n (например, n=001000 — третий сектор будет показан отдельно).

**mesh(M)** Основная команда трехмерной графики. Тоже, что и plot3.

**meshgrid (x,y)** Генерирование сетки на прямоугольной области задания функции: (например: [X,Y]=meshgrid(-10:.1:10, 0:0.05:1);)

**surf(X,Y,Z)** Трехмерная каркасная поверхность, заливаемая выбранным цветом .В остальном — аналог команды mesh

**contour3**  Трехмерная поверхность из линий равного уровня

**meshc, surfc** Трехмерная каркасная поверхность, совмещенная с линиями уровня на плоскости X,Y

Проиллюстрируем последнюю команду (рис. 1.4):

>>[X,Y]=meshgrid(-1:0.05:1,0:0.05:1);

>>Z=4\*sin(2\*pi\*X).\*cos(1.5\*pi\*Y).\*(1-X.^2).\*Y.\*(1-Y);

>>surfc(X,Y,Z)

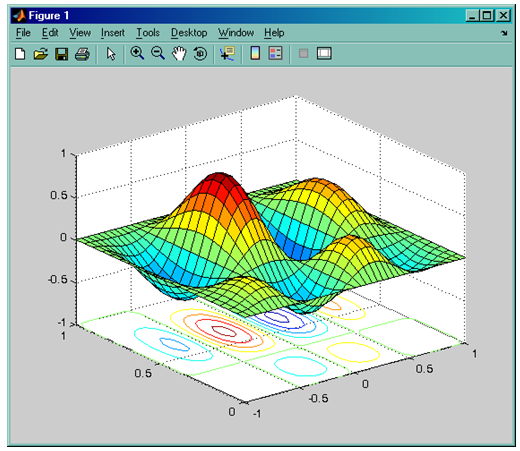


Рис. 1.4 - Результат работы программы

Как правило, графические объекты имеют не один, а несколько аргументов. Это существенно расширяет возможности пользователя. Возьмем, например, графический объект cylinder. Можно просто задать радиус цилиндра:

>> R=2;

>> [X,Y,Z]=cylinder(R);

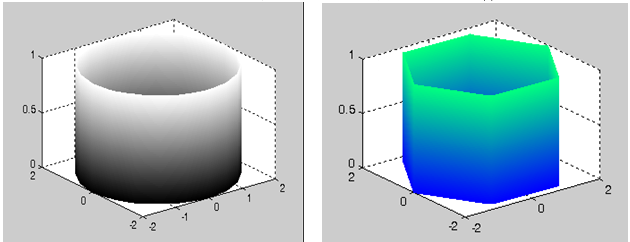
>> surf(X,Y,Z)

>> shading interp % вместо проволочного каркаса — поверхность, плавно залитая цветом

>> colormap(gray) % палитра оттенков серого (рисунок 1.5,а)

Но можно и по-другому: определить второй аргумент команды — количество точек на образующей цилиндра:

>> [X,Y,Z]=cylinder(R, 6); (рисунок 1.5,б- палитра изменена на "winter")



|  |  |
| --- | --- |
| а) | б) |

Рис. 1.5 - Палитры

Однако оценить подлинные возможности Матлаба можно, если в качестве аргумента использовать не скалярную величину, а функцию. Пусть R меняется от нуля до2 с шагом 0.1 (рис. 1.6):

>> R=0:0.1:2;

>> [X,Y,Z]=cylinder(R);

>> surf(X,Y,Z)

>> shading faceted % тип поверхности — " проволочный каркас"

>> colormap(gray)

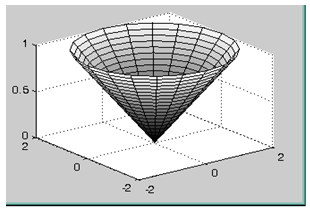


Рис. 1.6. – Результат работы программы

Перевернем вектор R задом-наперед, переставив его отсчеты: R1=R(end:-1:1) и сформируем новый вектор сцеплением строк (рис. 1.7):

>>R3=[R1,R];

>>[X,Y,Z]=cylinder(R3);

>>surf(X,Y,Z)

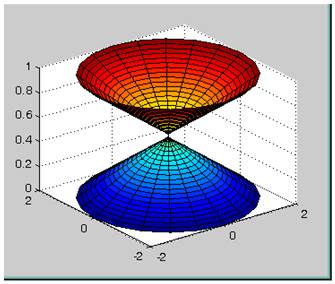


Рис. 1.7 – Результат работы программы

А от этой фигуры до рюмки (рис. 1.8) — всего один шаг (Минздрав предупреждает: если сидеть за компьютером больше 4 часов, может и не такое получиться!).

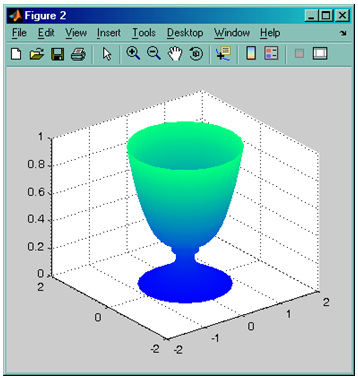


Рис. 1.8 – Результат работы программы

**1.12.** **Основы программирования в среде MATLAB**

**1.12.1. Операторы управления вычислительным процессом**

Все операторы циклов и условных переходов начинаются служебным словом *if, for, while, switch* изаканчиваются служебным словом *end.* Операторы, расположенные между ними, воспринимаются системой как составные части одного сложного оператора. Поэтому нажатие клавиши Enter при переходе к следующей строке не приводит к немедленному выполнению цикла.

**Операторы цикла**. Таких операторов два: *условный — while и арифметический — for***.**

Оператор с предусловием имеет вид:

*while <условие>*

*<операторы>*

*end*

Операторы внутри цикла выполняются до тех пор, пока выполняется условие после слова while. При этом среди операторов обязательно должны быть такие, которые изменяют переменную, записанную в условии цикла.

Пример:

>> i=1; % начальное значение переменной; после цикла оно изменится и вновь цикл не запустится.

>>while i<=4

x=i/5;

si=sin(x);

disp([x,si]) % вывод на экран нескольких переменных в одну строку: *очень удобная форма* при наличии в цикле нескольких операторов — получается *таблица.*

i=i+1;

end

0.20000000000000 0.19866933079506

0.40000000000000 0.38941834230865

0.60000000000000 0.56464247339504

0.80000000000000 0.71735609089952

Арифметический оператор цикла имеет вид:

*for <имя>= <нач. значение> : <шаг> : <конечное значение>*

*<операторы>*

*end*

Сравните два оператора цикла. В цикле for нет ни команд, изменяющих переменную цикла, ни условий проверки на его окончание. Все это автоматически делает счетчик цикла:

>> for i= 1:4 % предварительно (до цикла) начальное значение счетчика i задавать не надо.

x=i/5;

si=sin(x);

disp([i,x,si])

end

Чтобы досрочно выйти из цикла (например, при выполнении какого-нибудь условия) применяют оператор *break.*

Оператор условного перехода:

*if <условие>*% если условие выполняется, выполняются *операторы 1,* если нет -*операторы 2*

*<операторы 1>*

*else*

*<операторы 2>*

*end*

Условие может быть составным, т.е. состоять из нескольких простых условий, объединенных знаками логических операций: **&** (и), **|** (или), **~** (не). Можно усложнить конструкцию оператора, введя в него после <операторы1> еще одно условие командой elseif :

*if <условие>* % если условие выполняется, выполняются *операторы 1*

**<***операторы 1>*

*elseif <условие2>* % если условие 1 не выполняется, проверяется *условие 2*

**<***операторы 2>*% выполняются при выполнении условия 2

*else* % иначе — операторы 2

*<операторы 2>*

*end*

**1.12.2. Создание файл — функций**

При написании программ удобным способом упрощения повторяющихся процедур является создание собственных файл — функций.

Выберите в командном окне коману **Файл ► Новый ►М- файл.** На экране появится окно текстового редактора (блокнот). В нем наберите заголовок функции по форме:

function [y1,y2,…yn]=<имя процедуры>(<входные переменные>)

Теперь осталось только сохранить созданную функцию. В меню блокнота выберите **Файл** ►**Сохранить как** , и подтвердите свое согласие в открывшемся каталоге. Теперь вы можете пользоваться своей функцией точно также, как раньше пользовались, например, функцией sin.

Для примера создадим файл- функцию, вычисляющий сразу три функции:

y1=400 sin(x)/x;

y2=x2;

y3=400-x2;

Назовем эту функцию "трио", а все "y" объединим в одну матрицу размером (size(x)\*3):

function y= trio(x)

y(:,1)=400\*sin(x)./x;

y(:,2)=x.^2;

y(:,3)=400-x.^2;

Построим графики функций:

fplot('trio',[-20 20]), grid

title('График функции "trio"') (рисунок 1.9)

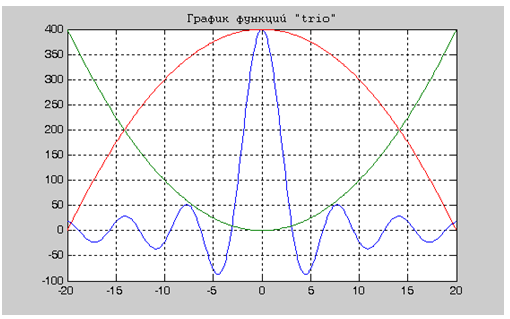


Рис. 1.9 – Функция «Трио»

**2. МЕТОДЫ ОПТИМИЗАЦИИ**

Рассмотрим следующую задачу оптимизации, называемую также задачей математического программирования:

 (2.1)

Здесь  – переменная (параметр) оптимизации,  и  – функциональные ограничения,  – целевая функция.

Множество  называют допустимым множеством. Точка  называется решением, если для всех точек  из некоторой окрестности  выполняется . Если , то говорят о глобальном минимуме, в противном случае о локальном. Напомним также, что задачу максимизации можно свести к приведенному выше виду заменой знака целевой функции.

В общем случае поиск глобального, и даже локального минимума – чрезвычайно сложная задача. Однако для некоторых классов задач существуют эффективные методы решения. Так задачи, в которых целевая функция и ограничения являются выпуклыми, могут быть эффективно решены даже при достаточно больших размерностях.

В этой главе мы сделаем акцент на задачах условной оптимизации: рассмотрим необходимое условие экстремума, опишем идею методов внутренней точки для выпуклых задач, подробнее остановимся на задачах линейного и квадратичного программирования. Задачи этого класса возникают в различных областях и для них существуют эффективные методы решения. Также они обладают огромной теоретической значимостью. Так, многие общие методы решения задач нелинейного программирования используют квадратичное программирование в качестве подзадачи (методы внутренней точки, последовательное квадратичное программирование и другие).

**2.1 Необходимое условие экстремума**

Предположим, что функции  непрерывно-дифференцируемые. Тогда *необходимое* условие экстремума задается теоремой Каруша-Куна-Такера:

Введем функцию Лагранжа  где , . Если  – точка экстремума, тогда существуют такие векторы  и , что



Для справедливости теоремы в точке  должны также выполняться условия регулярности, однако мы не будем на них останавливаться, считая здесь и далее, что они выполнены.

Ограничение-неравенство  называется активным в точке , если . Если в точке минимума неравенство неактивно, то его можно не учитывать при минимизации. Это обеспечивает условие дополняющей нежесткости .

Частными случаями теоремы Каруша-Куна-Такера являются метод множителей Лагранжа для задач с ограничениями типа равенство и теорема Ферма для безусловной оптимизации.

Для задач выпуклой оптимизации условия ККТ являются также и достаточными условиями глобального минимума.

Теорема Каруша-Куна-Такера задает условия экстремума. Возникает вопрос – почему бы нам не искать минимум, непосредственно решая полученную систему? Оказывается, что в общем случае это чрезвычайно сложная задача. Даже если присутствуют только ограничения-равенства, то получим систему нелинейных алгебраических уравнений. Если же присутствуют ограничения-неравенства, то возможны  наборов активных неравенств, для каждого из которых приходится решать систему уравнений. Несмотря на это, условия ККТ являются очень полезными для проверки решения и, как мы увидим далее, используются при разработке алгоритмов оптимизации.

## 2.2. Методы внутренней точки

Возможные способы решения задач условной оптимизации базируются на предположении, что мы умеем эффективно решать безусловные задачи. Тогда можно попытаться приблизить исходную задачу последовательностью безусловных. Эту идею используют метод штрафных функции и метод барьеров. В методе штрафных функций штраф осуществляется при выходе за границы допустимой области, в барьерном методе при подходе к границе изнутри, при этом начальная точка должна принадлежать допустимой области. По этим причинам класс таких методов называют методами внутренней точки.

Рассмотрим следующую задачу выпуклой оптимизации:



где функции  и  – выпуклые и дважды дифференцируемые, а допустимая область непуста. Выпуклые задачи обладают множеством хороших свойств, в частности любое решение является глобальным.

Для описания метода решения этой задачи последовательно изложим алгоритмы решения частных подзадач, при этом делая акцент на основных идеях, не останавливаясь на технических деталях. Подробнее с этими методами можно ознакомиться, например, в книге [1], по которой в значительной мере будет строиться изложение.

### 2.3. Метод Ньютона

Рассмотрим задачу безусловной оптимизации .

Одним из способов решения таких задач является метод Ньютона, к которому можно прийти разными способами.

Пусть мы находимся в точке , достаточно близкой от решения и хотим сделать шаг  такой, чтобы прийти в минимум. Тогда должно выполняться условие экстремума . Линеаризуя его, получим

,

откуда

.

Рассмотрим также другой способ вывода шага Ньютона. Для этого аппроксимируем функцию  первыми тремя членами разложения в ряд Тейлора:

.

Мы хотим найти такое направление , при котором функция наиболее уменьшится. Получили задачу безусловной квадратичной оптимизации

.

Разрешая относительно  необходимое условие экстремума, придем к шагу Ньютона .

Таким образом, следующую точку  можно найти по текущей  как , где  – длина шага, в оригинальном методе равная единице.

Метод Ньютона сходится квадратично, т.е. значительно быстрее градиентных методов, однако в общем случае обладает малой областью сходимости. Также недостатком является необходимость вычисления и обращения Гессиана на каждой итерации.

### 2.4. Метод Ньютона с ограничениями типа равенств

Рассмотрим теперь задачу с линейными ограничениями типа равенство:

.

Воспользуемся тем же приемом – аппроксимируем целевую функцию членами вплоть до квадратичного в разложении в ряд Тейлора:

.

Предположим, что текущая точка  принадлежит допустимой области, т.е. , тогда уравнение  эквивалентно . Таким образом, получили задачу квадратичной оптимизации с линейными ограничениями типа равенство

.

Уравнения ККТ для таких задач представляют собой линейную систему уравнений:

.

Получили, что на каждой итерации метода необходимо решать систему линейных уравнений, что может быть сделано вычислительно эффективно.

Заметим, что тот же результат можно получить, линеаризуя необходимые условия экстремума.

Еще раз отметим, что начальная точка в методе должна принадлежать допустимой области. В качестве упражнения предлагается модифицировать систему, чтобы устранить это условие.

### 2.5. Прямой метод внутренней точки

В предыдущих параграфах мы научились решать задачи с ограничениями в виде линейных уравнений. Поэтому попытаемся свести общую задачу к этому случаю, например, представив в следующем виде

,

где

.

К сожалению, индикаторная функция  недифференцируема, поэтому мы не можем воспользоваться ранее рассмотренными методами. Возможный выход состоит в аппроксимации: . Тогда, введя обозначение  и домножая целевую функцию на , получим следующую аппроксимацию исходной задачи:

 (2.2)

Полученная задача является выпуклой, и точность аппроксимации увеличивается при увеличении . Очевидным способом решения представляется решение полученной аппроксимации при достаточно большом . Однако при больших  решение полученной задачи методом Ньютона сопряжено с некоторыми трудностями, поскольку Гессиан возле границы изменяется чрезвычайно быстро. Поэтому в прямом методе внутренней точки решают последовательность полученных при аппроксимации задач (2.2) при увеличивающихся . При этом начальная точка  для каждой итерации по  берется как оптимальное значение  для предыдущего .

### 2.6. Прямо-двойственный метод внутренней точки

Можно показать, что для фиксированного  точка  является решением задачи , если существуют такие  и , что

 (2.3)

Эта система совпадает с системой ККТ для задачи , за исключением условия дополняющей нежесткости, которое в исходной задаче имеет вид . Т.е. введение логарифмического барьера эквивалентно возмущению условий ККТ.

Прямо-двойственный метод стартует из внутренней точки, а далее на каждой итерации производится решение линеаризованной возмущенной системы ККТ (без учета неравенств), которое задает шаг метода Ньютона по прямой переменной  и двойственным , .

Прямо-двойственый метод для многих задач эффективнее прямого.

## 2.7. Линейное программирование

Линейное программирование – частный случай математического программирования, когда целевая функция и ограничения линейные. Это наиболее простой класс выпуклых задач оптимизации. Для задач линейного программирования существуют эффективные алгоритмы и программы, позволяющие решать задачи с большим числом переменных.

Рассмотрим задачу линейного программирования в форме, принятой в команде linprog системы Matlab:



Допустимое множество , задаваемое ограничениями, является многограником, не обязательно ограниченным, а линии уровня целевой функции – гиперплоскостями. На рисунке 2.1 приведен пример для задачи





Рис. 2.1 – Допустимое множество и линии уровня

Исходя из геометрии допустимой области, возможны следующие случаи:

* Нет решения – когда допустимое множество пусто;
* Целевая функция неограничена;
* Единственное решение – вершина многогранника;
* Континуум решений – ребро многогранника (однако в этом случае оптимальное значение функции единственно).

Основными алгоритмами решения задач линейного программирования являются методы внутренней точки и симплекс-метод. Симплекс-метод использует тот факт, что решение, если оно существует, обязательно находится в вершине (или на ребре) многоугольника. По сути симплекс-метод является умным перебором вершин многогранника.

Для задач больших размерностей использование прямо-двойственного метода внутренней точки предпочтительней. Вывести для него шаг Ньютона предлагается в качестве упражнения.

В пакете Optimization Toolbox системы Matlab для решения задач линейного программирования используется функция linprog. Она имеет следующий синтаксис:

[xval,fval,exitflag] = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)

Здесь x0 и options являются необязательными параметрами, задающими начальную точку и настройки решателя соответственно. Если ограничения какого-нибудь типа отсутствуют, то соответствующие матрица и вектор заменяются на пустой массив. Так, если отсутствуют ограничения-равенства, то это можно указать как Aeq = []; beq = [].

Выходные переменные xval и fval возвращают оптимальное значение аргумента и функции соответственно. Переменная exitflag отображает статус решения задачи. Различные ее значения можно посмотреть в документации, здесь же отметим наиболее важные случаи:

* 1 – решение найдено;
* -2 – допустимая область пуста;
* -3 – целевая функция неограничена.

В качестве примера покажем, как можно решить задачу из примера к рисунку 2.1:

f = -[1 1];

lb = [0 0];

ub = [];

beq = [];

Aeq = [];

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

b = [5 11 1];

[xval, fval, exitflag] = linprog(f, A, b, Aeq, beq, lb, ub);

Подробнее про использование параметра options будет рассказано ниже, здесь же проиллюстрируем установку в качестве метода решения симплекс-метода (по умолчанию используется метод внутренней точки):

options = optimset('Algorithm', 'simplex');

[x,fval,exitflag] = linprog(f, A, b, Aeq, beq, lb, ub, [], options);

Рассмотрим пример задачи, приводящей к линейному программированию.

**Пример 1**. Транспортная задача. Есть два завода, которые производят товар для трех магазинов. Стоимость перевозки из заводов в магазины представлена в таблице.

|  |  |  |
| --- | --- | --- |
| 20 | 25 | 55 |
| 35 | 15 | 40 |

Известно, что спрос товара в первом магазине составляет 100 единиц, во втором 150, в третьем 200. В то же время первый завод производит 200 единиц товара, второй 250. Требуется минимизировать затраты на перевозку.

**Решение:** обозначим как  количество товара, перевозимое из -го завода в -ый магазин. Тогда требуется минимизировать затраты  при ограничениях, обеспечивающих выполнение спроса , ,  и ограничениях на производство , . Также очевидно следует наложить требование неотрицательности всех .

Полученную задачу можно решить следующим скриптом:

c = [20 25 55 35 15 40];

A = [-1 0 0 -1 0 0;

0 -1 0 0 -1 0;

0 0 -1 0 0 -1;

1 1 1 0 0 0;

0 0 0 1 1 1];

b = [-100; -150; -200; 200; 250];

lb = zeros(6, 1);

[x,fval,exitflag] = linprog(c, A, b, [], [], lb, [])

Откуда , , , .

## 2.8. Квадратичное программирование

Квадратичным программированием называют задачи с линейными ограничениями и квадратичной целевой функцией:



Предполагается, что матрица  симметричная, т.е. . Если же это не так, то ее можно привести к симметричному виду заменой , при этом решение не изменится. В системе Matlab такая замена производится автоматически.

Если матрица  является положительно определенной, то задача выпуклая, решение является глобальным и может быть эффективно найдено. В противном случае могут быть локальные решения, а поиск глобального –NP-трудная задача. Далее будем рассматривать только случай положительно-определенной матрицы.

Для решения задач квадратичного программирования может использоваться большое число алгоритмов. Так, например, метод Франка-Вульфа основан на линеаризации целевой функции и последовательном решении задач линейного программирования.

Для задач большой размерности, как правило, наиболее эффективным является прямо-двойственный метод Ньютона. Вывести шаг метода Ньютона предлагается в качестве упражнения.

В пакете Optimization Toolbox системы Matlab для решения задач квадратичного программирования используется функция quadprog. Она имеет следующий синтаксис:

[x, fval, exitflag] = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)

Здесь матрицы и векторы соответсвуют следующей форме записи задачи квадратичного программирования:



а необязательные параметры x0 и options задают начальную точку и параметры соответсвенно. Выходные переменные такие же, как в случае линейного программирования.

Рассмотрим в качестве примера следующую задачу:



Ее решение можно найти следующей последовательностью команд:

H = 2\*[4 1; 1 1];

f = [2; 3];

A = [-1 1; 1 1; 1 0];

b = [0 4 3];

[x, fval, exitflag] = quadprog(H, f, A, b);

Если мы будем решать эту же задачу, но пытаться максимизировать функцию:

H = -2\*[4 1; 1 1];

f = -[2; 3];

A = [-1 1; 1 1; 1 0];

b = [0 4 3];

options = optimset('Algorithm', 'active-set');

[x, fval, exitflag] = quadprog(H,f,A,b,[],[],[],[],[],options)

то Matlab выдаст решение и exitflag=1. Однако теперь задача невыпуклая и целевая функция неограничена. Если использовать алгоритм внутренней точки:

H = -2\*[4 1; 1 1];

f = -[2; 3];

A = [-1 1; 1 1; 1 0];

b = [0 4 3];

options = optimset('Algorithm','interior-point-convex');

[x, fval, exitflag] = quadprog(H,f,A,b,[],[],[],[],[],options)

то Matlab выдаст об предупреждение и exitflag=-6.

Таким образом, рекомендуется заранее проверять задачу на выпуклость или пользоваться алгоритмом interior-point-convex. Также рекомендуется для любых задач оптимизации всегда проверять решение, которое выдается решателем.

Рассмотрим пример задачи оптимального управления, приводящей к квадратичному и линейному программированию.

**Пример 2.** Задана линейная система  с начальным условием , где , , . Требуется найти управление, переводящее систему в точку  при ограничениях  и минимизирующее функционал

а) ,

б) .

**Решение**: одним из подходов к решению таких задач является последовательный: зная  и  для всех , можем вычислить :



В матричном виде это выражение можно записать как , где .

Таким образом, в случае квадратичного функционала получили задачу квадратичного программирования:



Решим эту задачу в системе Matlab при  ,  ,, , .

n = 3;

N = 30;

A = [-1 0.4 0.8; 1 0 0; 0 1 0];

b = [1 0 0.3]';

xdes = [7 2 -6]';

u\_max = 5;

x0 = ones(n, 1);

for k = 1:N

H(:, k) = A^(N-k)\*b;

end

Aeq = H;

beq = xdes — A^N\*x0;

Q = 2\*eye(N);

f = zeros(N, 1);

lb = -u\_max\*ones(N, 1);

ub = u\_max\*ones(N, 1);

[u, fval, exitflag] = quadprog(Q, f, [], [], Aeq, beq, lb, ub)

stairs(0:N-1, u)

axis tight

xlabel('t')

ylabel('u')

Рассмотрим теперь случай с функционалом . Для того, чтобы прейти к задаче линейного программирования, заметим, что задача минимизации модуля скалярной величины  эквивалентна задаче



Таким образом, исходная задача оптимального управления сводится к задаче линейного программирования с переменными оптимизации  и :



Для тех же данных решение в системе Matlab:

n = 3;

N = 30;

A = [-1 0.4 0.8; 1 0 0; 0 1 0];

b = [1 0 0.3]';

xdes = [7 2 -6]';

u\_max = 5;

x0 = ones(n,1);

for k = 1:N

H(:, k) = A^(N-k)\*b;

end

Aeq = [H zeros(n, N)];

beq = xdes — A^N\*x0;

c = [zeros(N, 1); ones(N, 1)];

lb = -u\_max\*ones(2\*N, 1);

ub = u\_max\*ones(2\*N, 1);

A = [-eye(N) -eye(N); eye(N) -eye(N)];

b = [zeros(N, 1); zeros(N, 1)];

[xval, fval, exitflag] = linprog(c, A, b, Aeq, beq, lb, ub)

u = xval(1:N);

stairs(0:N-1, u)

axis tight

xlabel('t')

ylabel('u')

График найденных управлений представлен на рисунке 2.2.



Рис. 2.2 – Оптимальные управления

## 2.9. Решение задач нелинейной оптимизации в системе Matlab

Перейдем к решению нелинейных задач математического программирования в системе Matlab. Для этих целей существует большое число решателей и пакетов, среди которых особо отметим CVX и YALMIP за простоту использования и большие возможности. Однако здесь остановимся на использовании пакета Optimization toolbox. Подробное описание можно посмотреть в документации, мы же отметим лишь основные моменты.

Optimization toolbox позволяет решать различные задачи, в том числе условной, безусловной и многокритериальной оптимизации. При решении задачи необходимо сначала определить, к какому классу она относится, затем выбрать подходящую функцию и алгоритм.

### 2.10. Безусловная оптимизация

Для решения задач безусловной оптимизации рекомендуется использовать функциию fminunc. Можно применять также функцию fminsearch, однако она использует методы нулевого порядка (бесградиентные) и в общем случае медленнее. В то же время она позволяет решать негладкие задачи.

Команда fminunc имеет следующий синтаксис:

[x, f, exitflag] = fminunc(fun, x0, options)

В отличие от функций linprog и quadprog, для этой команды обязательно указывать начальную точку x0. Так как функция ищет локальный минимум, то различные начальные точки могут приводить к различным решениям.

Входной аргумент fun является ссылкой на минимизируемую функцию. Ее можно записать, например, как функцию в отдельном файле или анонимную функцию. Рассмотрим оба варианта на стандартном примере – функции Розенброка .

В первом случае создается функция

function f = rosenbrock(x)

f = 100\*(x(2) — x(1)^2)^2 + (1 — x(1))^2;

Далее в командной строке или в другом файле осуществляется минимизация:

x0 = [1; 0.5] % пример начальных условий

[x, f, exitflag] = fminunc(@rosenbrock, x0, options)

Во втором случае все команды могут быть записаны в одном файле:

rosenbrock = @(x)(100\*(x(2) — x(1)^2)^2 + (1-x(1))^2);

x0 = [1; 0.5] % пример начальных условий

[x fval] = fminunc(rosenbrock, x0, options)

Многие алгоритмы для этой и других функций используют градиент функции. По умолчанию используется конечно-разностная аппроксимация. Для увеличения скорости и точности рекомендуется, если это возможно, указывать явную формулу для градиента. Так, функция Розенброка с градиентом может быть записана как

function [f g] = rosen\_grad(x)

f = 100\*(x(2) — x(1)^2)^2 + (1-x(1))^2;

if nargout > 1

g = [-400\*(x(2)-x(1)^2)\*x(1)-2\*(1-x(1));

200\*(x(2)-x(1)^2)];

end

Теперь укажем, что градиент будет вычисляться функцией и запустим процесс оптимизации:

options = optimoptions(@fminunc,'Algorithm','quasi-newton','GradObj','on');

x0 = [-1; 2];

[x fval] = fminunc(@rosen\_grad, x0, options)

Рекомендуется всегда проверять правильность вычисления градиента. Для этого присутсвует опция DerivativeCheck. Так, тот же пример с проверкой вычисления градиента:

options = optimoptions(@fminunc, 'Algorithm', 'quasi-newton', ...

'GradObj', 'on', 'DerivativeCheck', 'on');

x0 = [-1; 2];

[x fval] = fminunc(@rosen\_grad, x0, options)

Перед началом итераций метода выведено сообщение DerivativeCheck successfully passed, подтверждающее правильность вычисления градиента.

Рассмотрим теперь подробнее настройки options. Основные используемые опции, помимо вышеперечисленных, это отображение информации Display и критерии останова.

Параметр Display устанавливает настройки отображения работы алгоритма. По умолчанию установлено значение 'final', при котором информация выводится только после окончания работы. Другим часто используемым значением является 'iter', при котором выводится информация после каждой итерации.

Для настроек точности алгоритмов используются следующие параметры:

* TolFun – нижняя оценка на изменение . Алгоритм заканчивает работу, если текущее изменение меньше;
* TolX – нижняя оценка на шаг по . Алгоритм заканчивает работу, если текущий шаг меньше;
* MaxFunEvals – максимальное число вычислений функции;
* MaxIter – максимальное число итераций.

Параметры TolX и TolFun не рекомедуется выставлять меньше .

### 2.11. Условная оптимизация

Для задач условной оптимизации, не принадлежащих к задачам линейного и квадратичного программирования, могут использоваться функции fminbnd, fmincon и fseminf.

Функция fminbnd находит минимум функции  скалярного аргумента  на некотором интервале .

Функция fmincon является основной для решения задач условной оптимизации. Она имеет следующий синтаксис:

[x,fval,exitflag] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)

где A, b, Aeq, beq, lb, ub задают линейные ограничения, nonlcon нелинейные. Задание начальной точки x0 обязательно.

Нелинейные ограничения nonlcon задаются в виде отдельного файла или анонимной функции. Аналогично целевой функции, возможно использование градиента нелинейных ограничений.

При использовании функция fmincon рекомендется сперва использовать алгоритм внутренной точки 'interior-point', во вторую очередь последовательное квадратичное программирование 'sqp' и метод активных множеств 'active-set'.

Рассмотрим пример из документации: минимизация функции Розенброка при ограничении .

Целевая функция:

function f = rosenbrock(x)

f = 100\*(x(2) — x(1)^2)^2 + (1 — x(1))^2;

Ограничения:

function [c, ceq] = unitdisk(x)

c = x(1)^2 + x(2)^2 — 1;

ceq = [ ];

Функция, задающая нелинейные ограничения должна возвращать значения как неравенств , так и равенств  даже в том случае, когда какое-нибудь из этих ограничений отсутвует. В этом случае возвращается пустой массив.

После задания целевой функции и ограничений запустим алгоритм минимизации:

options = optimoptions(@fmincon, 'Display', 'iter', ...

'Algorithm','interior-point');

[x, fval, exitflag] = fmincon(@rosenbrock,[0 0],...

[],[],[],[],[],[],@unitdisk,options)

Функция fseminf решает задачи полубесконечного программирования. Они называются так, потому что в них конечное число переменных и бесконечное число ограничений. В общем виде такие задачи можно записать как



где  – интервал.

Такие задачи возникают во множестве приложений. Рассмотрим, например случай робастного линейного программирования



где ,  – заданные числа,  – скалярная переменная оптимизации, . То есть неравенство  задает бесконечное число ограничений. Например в рассмотренном примере из оптимального управления такая задача возникнет, если какой-нибудь элемент матрицы  известен с точностью до принадлежности интервалу.

Подробнее с функцией и ее использованием можно ознакомиться в документации.

## 2.12. Упражнения

1. Найти точку экстремума используя ККТ:  при .
2. Реализовать в системе Matlab метод Ньютона для системы . Запустить из начальных точек  и . Когда метод сходится, а когда нет?
3. За сколько итераций метод Ньютона сходится из произвольной точки для функции ?
4. Реализовать метод Ньютона для задачи  при ограничении  с начальными точками  и . Как вы можете объяснить увиденное?
5. Решить в системе Matlab задачу линейного программирования



1. Решите в системе Matlab задачу квадратичного программирования



1. Имеется завод, выпускающий шкафы и тумбочки. На сборку шкафа уходит 2.5 часа, тумбочки 1 час. Покраска и шкафа, и тумбочки занимает 3 часа. Упаковка шкафа занимает 1 час, тумбочки 2 часа. Требуется максимизировать прибыль, если на сборку можно потратить не более 20 часов, сушку 30 часов, упаковку 16 часов, а прибыль от продажи шкафа и тумбочки составляет 3 и 4 у.е. Свести к задаче оптимизации и найти ответ.
2. Сравнить быстродействие реализации симплекс-метода и метода внутренней точки команды linprog. Данные можно сегенировать, например, следующим образом:

rng(1234)

n = 400;

m = 800;

A = [];

b = [];

lb = zeros(n, 1);

ub = [];

Aeq = randn(m, n);

x = 5\*rand(n, 1) + 0.01;

beq = A\*x;

c = ones(n, 1);

Время выполнения команды можно замерить, например, поместив ее между командами tic и toc. Попробуйте решить эту же задачу командой fmincon с разными настройками алгоритмов.

***Дополнительные упражнения:***

1. Модифицируйте систему уравнений для шага метода Ньютона с ограничениями, чтобы он работал из начальных точек не из допустимой области. Протестируйте на задаче 4.
2. Выведите шаг прямо-двойственного метода для задачи линейного программирования.
3. Выведите шаг прямо-двойственного метода для задачи квадратичного программирования.
4. Другим способом решения задачи из примера 2 является оптимизация по  и , не выражая в явном виде . Таким образом решите задачу из примера 2 с функционалом .

**3. ЛАБОРАТОРНЫЕ РАБОТЫ**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**Генерирование тестовых сигналов и их преобразование.**

**Цель работы:**

* ‑ освоение методов создания в Matlab сигналов различной формы;
* ‑ знакомство с правилами работы в пакете Simulink;
* ‑ изучение методов преобразования сигналов: модуляция (умножение), суммирование, нелинейное преобразование.

**Теоретические положения.**

**Создание сигнала в Matlab.** Практически любая задача по обработке сигналов (процессов) требует создания тестовых сигналов, с помощью которых проверяется корректность используемых преобразований сигналов, измеряются параметры разработанной аппаратуры, моделируются различные режимы ее работы. Если сигнал зависит только от одной координаты (параметра), его называют одномерным. Процесс – это сигнал, аргументом которого является время. Если сигнал –функция двух параметров, он представляется матрицей и визуализируется трехмерным графиком (поверхностью).

Matlab предоставляет несколько возможностей для создания одномерных и многомерных сигналов.

*Задание сигнала дискретными отсчетами (по точкам)*: x=[1,2,5,2,1]; % квадратные скобки служат для задания цифровых значений вектора; круглые используются в математических выражениях для задания аргумента и для обозначения порядкового номера отсчета в векторе. Для создания этим способом матрицы надо в конце каждой строки вместо запятой ставить точку с запятой.

Этим простым способом можно создать достаточно сложные сигналы, если использовать его в сочетании со следующими операциями:

- задание вектора в форме арифметической прогрессии: y=1:5:26 (здесь шаг равен 5)

- вырезка (вставка) части вектора:

z=zeros(1,100); % вектор из ста нулей;

z(11:15)=x;

- конкатенация (объединение) векторов(матриц):

w=(x,y);.

*Задание вектора командой или функцией Matlab*. Для формирования часто используемых сигналов в Matlab существует большая группа функций:

* матрица (вектор) из n\*m нулей: zeros(n,m);
* матрица (вектор) из n\*m единиц: ones(n,m);
* тригонометрические функции: sin(x), sind(x) (аргумент в градусах), asin(x), cos(x),
* импульсы прямоугольной формы: rectpuls(t,w) –импульс ширины w на интервале t; square(t, w)- последовательность импульсов с периодом  и длительностью w(в процентах от периода);
* импульсы пилообразной и треугольной формы: sawtooth(t,w)- последовательность пилообразных импульсов с периодом  и длительностью нарастающего фронта w; w меняется от 0 до1, при w=0.5 синтезируются треугольные импульсы с равной шириной нарастающего и падающего фронта;
* гармонический сигнал с линейно меняющейся частотой: chirp(t,f0,f1,t1) c заданием частоты в двух точках: при t=0 (f0) и при t=t1 (f1);
* импульсы произвольной формы, задаваемой опцией func: pulstran(t,d,'func');
* и много других, описанных в help.

*Формирование по заданным точкам аналитической зависимости* в виде полинома, степень  которого выбирается разработчиком, а коэффициенты  рассчитывает Matlab: 

Единственное требование к векторам  и  – они должны быть одинаковой длины Задача, решаемая этим методом – построить такую кривую заданного порядка, отклонение которой от заданных точек было бы минимальным. Поскольку в качестве меры отклонения принята *сумма квадратов* разностей значений исходных точек от создаваемой кривой, метод называется *МНК* (метод наименьших квадратов). Эта достаточно сложная задача в пакете Matlab решается всего одной командой: p=polyfit (x,y,n), где n-степень полинома,  — вектор коэффициентов . Команда polyval(p,x1) рассчитывает искомую полиномиальную кривую (рис. 3.1):

x=[0,2,4,6,8,9,10,10,];

y=[1,3,2,6,4,6,12,16]; % задание координат исходных точек

plot(x,y,'o')

p1=polyfit(x,y,1) % расчет коэффициентов полинома первого порядка;

x1=0:0.5:9; % выбран новый диапазон x внутри исходного;

s1=polyval(p1,x1); % вычисление полинома;

hold on

plot(x1,s1)

p2=polyfit(x,y,2)

s2=polyval(p2,x);

plot(x,s2,'r')

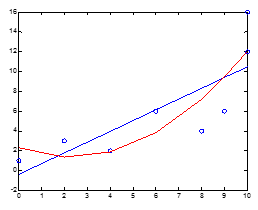


Рис. 3.1 - Полиномы первого и второго порядка, построенные по методу наименьших квадратов

*Использование генераторов сигналов из библиотеки пакета Simulink,.* раздел Sources (источники). Эта библиотека во многом повторяет уже рассмотренные нами в разделе 1.2 команды Matlab, однако есть и принципиальное отличие: пакет Simulink позволяет не только генерировать сигналы, но и использовать их в разнообразных моделях, легко синтезируемых с помощью той же библиотеки.

*Преобразование сигналов* на рис. 3.2 представлен пример создания модели Simulink, в которой тестовый сигнал пилообразной формы подвергается нелинейным преобразованиям звеном с регулируемой зоной нечувствительности и звеном с насыщением



Рис. 3.2 - Пример моделирования нелинейных преобразований

Результат преобразования можно наблюдать осциллографом Scope или Scope1 –разницу оцените и прокомментируйте в работе. Блоки simout позволяют экспортировать результат в окно Workspace Matlab (предварительно в окне параметров блока надо заменить формат со Structure на Array).

Построить процесс на выходе нелинейного звена можно и вручную, по точкам, как показано на рис. 3.3. Если входной сигнал x(t) разместить в четвертом квадранте, а характеристику "вход-выход" y=f(x) (в автоматике ее называют *статическая характеристика*) – в первом, выход нелинейного звена можно получить во втором квадранте.

По этой же методике можно построить выходной сигнал двух и более звеньев, располагая их характеристики друг за другом – в первом, втором и т.д. квадрантах. Если нелинейное звено охвачено обратной связью (рис. 3.4), нужно сперва построить эквивалентную статическую характеристику y=F(x) замкнутой системы. Чтобы построить одну точку этой характеристики, зададим воздействие на *входе нелинейного звена* (а не на входе всей системы !) в точке  рис. 3.5, и по статической характеристике  определим выходную величину .

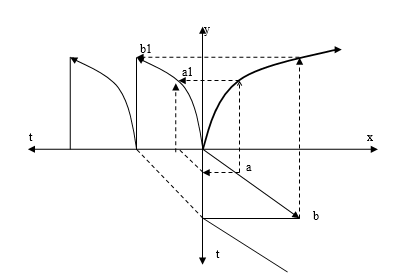


Рис. 3.3 - Построение выходного процесса нелинейного звена

Во втором квадранте на статической характеристике звена обратной связи найдем точку  и соответствующее ей значение . Вычитая отрезок  из отрезка , найдем точку е, соответствующую значению входной величины  при . Найденная точка с координатами  изображена крестом на рис. 3.5.

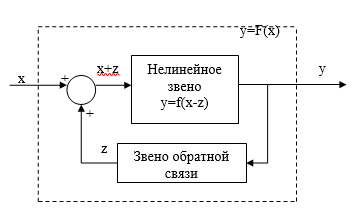


Рис. 3.4 - Замкнутая нелинейная система

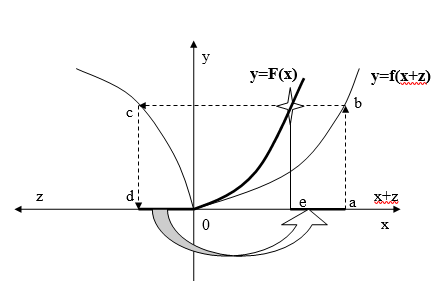


Рис. 3.5 - Построение эквивалентной статической характеристики замкнутой системы

**Порядок выполнения работы.**

Используемые команды 

* plot, bar, stem, stairs, mesh, surf, plot3 - команды построения графиков;
* polyfit, polyval – полиномиальная аппроксимация;
* square(t, w) chirp(t,f0,f1,t1).

1. Создание функции одного аргумента y=f(x).

Для такого простейшего преобразования в MATLAB предусмотрено более десятка форм графического отображения. Вам предстоит оценить особенности и достоинства некоторых из них.

Используя команды таблицы 1,операции врезки и конкатенации, создайте вектор , заданный преподавателем в графической форме. Задав номер фигуры (figure(1)), "нарисуйте" этот вектор , используя команды графики plot, stem, stairs. Определите, в чем отличие этих команд. Дайте пояснения к каждому графику. Воспользуйтесь операцией наложения графиков, используя различный тип линий.

Если шаг приращения аргумента выбран чрезмерно большим, на графиках наблюдаются большие искажения исходной зависимости. Уменьшить погрешность построения графика можно двумя способами: либо специально созданной для этой цели командой *fplot* , либо полиномиальной интерполяцией. Чтобы использовать команду *fplot*, по методике, изложенной в разделе 1.10.2., создайте файл-функцию для выбранной арифметической операции, задав ей имя, например, 'my'. Теперь Вы можете воспользоваться еще одним способом визуализации — *fplot('my',[xmin,xmax],'.').* Тип линии ('.') здесь выбран не случайно: обратите внимание, как Matlab выбирает расстояние между отсчетами, чтобы обеспечить заданную точность построения графика (по умолчанию 0.2 %).

Если между  нет функциональной связи (например, значения y при заданных x получены в эксперименте), используется полиномиальная интерполяция по методу наименьших квадратов. Вычисление коэффициентов полинома степени 



осуществляет функция *polyfit(x,y,n).*

Задайте два вектора  одинаковой длины (10-15 отсчетов), постройте график plot (x,y).Желательно, чтобы получившаяся кривая содержала 2 — 3 экстремума. Отсчеты  могут располагаться неравномерно. По этим векторам создайте несколько полиномов разных степеней n (начиная с первой). Постройте на том же графике эти полиномы и поясните полученный результат.

Создайте командой subplot несколько окон в одном графическом окне и разместите в них полученные графики.

2. Преобразование функций нелинейными звеньями.

В пакете Simulink создайте модель по рис. 3.2. Используя блоки simout, постройте графики преобразованных сигналов непосредственно из окна команд Matlab. Наложите на одном графике сигналы, полученные при разных уровнях ограничения. Включите нелинейные звенья последовательно. Прокомментируйте результат.

По методике, изложенной в разделе 1.4, постройте вручную выходной сигнал замкнутой системы, звенья в которой задаются преподавателем. Результат сравните с моделью в Simulink.

Важнейший принцип управления в автоматике – *управление по отклонению, или принцип обратной связи.* Замыкание системы отрицательной обратной связью линеаризирует характеристики системы, делает ее более быстрой, увеличивает ее точность. Определите, как влияет коэффициент усиления замкнутой системы на ее статическую ошибку. Для этого создайте систему, в прямую цепь которой поставьте два звена: усилитель и нелинейное звено, заданное преподавателем. В цепи обратной связи коэффициент передачи сделайте равным -1. На вход системы подайте треугольный сигнал. Меняя коэффициент усиления усилителя, регистрируйте величину ошибки , которую можно наблюдать на выходе сумматора. По структурной схеме определите, как связаны величины  и .

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**Матричные преобразования и трехмерная графика.**

**Цель работы:** освоение специфики матричных преобразований Matlab и сравнительный анализ различных форм графического отображения результатов.

**Теоретические положения.**

*Функция двух аргументов z=f(x,y) Трехмерная графика.* Можно привести несколько способов задания аргументов трехмерной поверхности. Самый простой из них описывался в предыдущих разделах. На области определения,  командой [x,y]=meshgrid ( ) создайте координатную сетку с заданным шагом. Теперь можно создавать саму функцию, например:

[x,y]=meshgrid(-10:0.3:10,-10:0.3:10);

>> z=(sin(x)./x).\*(sin(y)./y);

>> surf(x,y,z)

Полученная поверхность приведена на рисунке 3.6. Можно произвольно выбрать шкалу цветовых оттенков (см. help colormap).

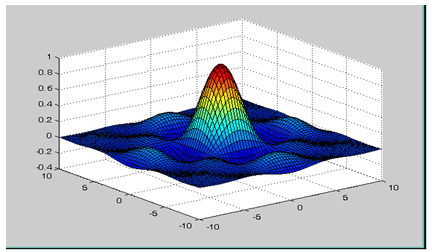


Рис. 3.6 - Трехмерная поверхность

Второй способ состоит в формировании двух взаимно перпендикулярных плоскостей X и Y. –аналогов двумерных осей ординат.

>> x=-10:10;

y=ones(1,21);

X=x'\*y; %постройте !

Y=y'\*x; % постройте!

Теперь используем координатные плоскости для построения конуса:

R=sqrt(X.^2+Y.^2); mesh(15-R), или пирамиды: R1=max(abs(X),abs(Y)) (рис. 3.7).

*Матричные операции., вычисление функций от матриц.* Кратко напомним основные матричные операции.

*Определитель*. Сумма произведений всех элементов какой – либо  — ой строки (столбца) на соответствующие им алгебраические дополнения  равна значению определителя: 

*Обратная матрица.* Элемент обратной матрицы  равен частному от деления алгебраического дополнения элемента на определитель матрицы  (правило Крамера):

*Умножение матриц.* Элемент матрицы — произведения на пересечении строки и столбца равен *сумме попарных произведений элементов строки* матрицы  и столбца  матрицы: 

*Возведение в степень* осуществляется ‑ кратным умножением матрицы на себя.

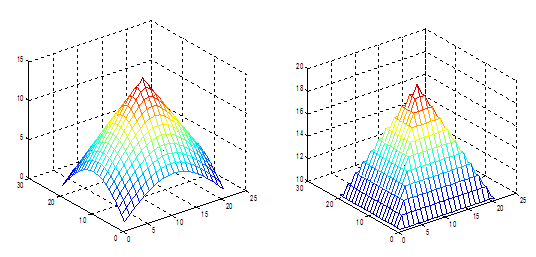


Рис. 3.7 - Трехмерные фигуры

**Порядок проведения работы.**

Используемые команды *Matlab:*

det(A) – определитель матрицы А;

eig(A) – собственные числа матрицы А;

eye(n) – единичная матрица порядка n с единицами на главной диагонали;

inv(A) – обратная матрица А-1;

sqrtm(A) – матричный квадратный корень А1/2;

logm(A) – матричный логарифм;

expm(A) – матричная экспонента eA.

1. В качестве исходной фигуры, на которой будем изучать матричные преобразования, выберем пирамиду (рис. 3.7). Присвоим ей имя R. Симметрия матрицы R относительно главной диагонали и антидиагонали делает такую матрицу вырожденной – ее определитель равен нулю (проверьте). Соответственно, большинство матричных операций для нее невыполнимо. Поэтому добавим к элементам на главной диагонали по единице, сложив ее с единичной матрицей (eye) того же размера. Теперь над матрицей можно производить как поэлементные, так и матричные операции.

Сравните (по графикам) результаты двух операций – обращения матрицы командой inv и поэлементного деления матрицы ones(n,n) на R.

Сравните матричные операции sqrtm(A), logm(A), expm(A) с аналогичными операциями, выполняемыми поэлементно.

Преобразуйте пирамиду R операциями врезки. "Отрежьте" какой- нибудь из углов, приравняв нулю выбранные элементы. Можно, например, так:

R1(:,1:5)=0; figure(2)

mesh(R1)

R1(10:15,:)=4; figure(3)

mesh(R1)

surfl(R1) % освещенная поверхность (без каркасной сетки)

shading interp % линейное изменение цвета

colormap('gray') % палитра серого

Очень полезной операцией размножения массивов (мультиплицирования) является операция кронекеровского умножения матриц, в которой *вся правая матрица умножается на каждый элемент левой:*



Создайте 9 конусов c помощью матрицы 3\*3(рис. 3.8).

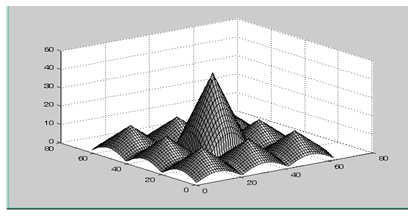


Рис. 3.8 - Мультипликация операцией kron

Поменяйте порядок матриц С и R в последней операции.

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**Корреляционный метод измерения задержки сигнала.**

**Цель работы:** анализ возможностей корреляционных методов выделения сигнала из шума и измерения его параметров.

**Теоретические положения.**

Корреляция (correlation), и ее частный случай для центрированных сигналов -ковариация — это мера похожести двух сигналов. Приведем один из вариантов использования корреляционной функции. Допустим, нам надо найти в тексте некоторый символ. Скользя по тексту, прикладываем символ к каждой букве, проверяя их сходство. Аналогично решается задача обнаружения некоторой последовательности x(t) известной формы и конечной длины Т в принимаемом сигнале y(t). Для поиска этой последовательности в скользящем по сигналу y(t) временном окне длиной Т вычисляется интеграл от произведения процессов y(t) и x(t):  В дискретной форме Z есть скалярное произведение векторов X и Y: .

Скалярное произведение используется как мера взаимосвязи двух векторов (процессов). Равенство нулю скалярного произведения означает крайнюю меру их независимости — их ортогональность. Чтобы проверить взаимосвязь векторов при различных сдвигах одного из них на время τ, необходимо  вычислять многократно, для каждого . Если еще ввести нормирующий множитель 1/N для исключения зависимости Z от длины реализации N, получим выражение, называемое *взаимной корреляционной функцией* (ВКФ): , и аналогично для случая непрерывного сигнала: 

Теперь скалярное произведение превратилось в *свертку* двух векторов.

R(τ) широко используется в различных областях человеческой деятельности. Например, в медицине корреляционная функция позволяет обнаружить самые минимальные изменения состояния организма от воздействия какого-либо неблагоприятного фактора или оценить длительность воздействия лекарства на пациента. Социологи используют ВКФ для оценки влияния на население новых законодательных актов. Астрономам ВКФ позволила обнаружить периодичность в изменении свечения звезд (пульсары). В приведенном примере поиска в тексте заданного символа корреляционная функция использовалась для распознавания или обнаружения сигнала. Еще более часто корреляционная функция используется для определения времени прихода (задержки) сигнала. Поскольку максимальное сходство (а., следовательно, и максимальное скалярное произведение) будет в момент полного совпадения x(t) и его копии в y(t), по положению максимума корреляционной функции можно судить о времени прихода сигнала. Этот метод лежит в основе измерения расстояния до цели (ракеты, самолета), определения высоты полета самолета или глубины погружения подводной лодки. Чтобы определить направление прихода радиосигнала (задача пеленгации), определяют разность времени прихода сигнала пеленгуемого объекта на две разнесенные антенны. Во всех этих примерах производится определение скалярного произведения сигнала *x(t) с собственной копией*, скользящей по аргументу, а функцию называют *автокорреляционной функцией (АКФ)*: .

Автокорреляция позволяет оценить среднестатистическую зависимость текущих отсчетов сигнала от своих предыдущих и последующих значений (так называемый радиус корреляции значений сигнала), а также выявить в сигнале наличие периодически повторяющихся элементов.

В корреляционном анализе часто вид корреляционной функции не имеет значения: интерес представляет один из ее параметров. В приведенных выше примерах хотя и вычисляется вся корреляционная функция, однако в первой задаче анализируется только величина максимального значения, а во второй — положение ее максимума на временной оси. Для анализа длительности лечебного воздействия прививок в медицине анализируют ширину корреляционной функции, и т.д.

КФ, вычисленная по центрированному значению сигнала x(t), представляет собой *ковариационную* функцию сигнала: .

Рассмотрим подробнее операцию вычисления ВКФ. На рисунке 3.9 приведена поэтапная процедура вычисления  сигнала x=sin(t) для одного значения сдвига (τ=6 отсчетов). На временном интервале от 33 до 37 отсчета перемножаемые функции имеют разный знак, в результате чего их произведение отрицательно. Следствием этого является падение накопленного значения R на этом интервале (точечная кривая).

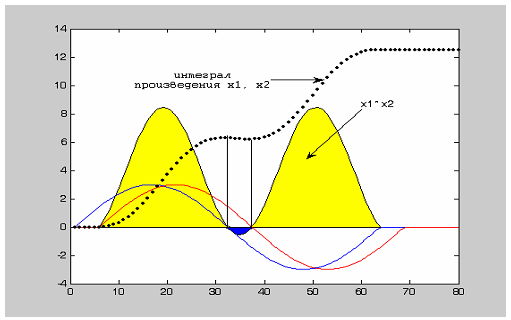


Рис. 3.9 - Вычисление автокорреляции

**Порядок проведения работы.**

1. В качестве решаемой задачи рассмотрим задачу измерения высоты полета самолета. Проще всего послать вертикально вниз короткий радиоимпульс и измерить при помощи корреляционной функции задержку им пульса, отраженного от земли. Создайте такие импульсы из длинной нулевой последовательности (zeros(n,m)) присвоением группе выбранных отсчетов некоторого значения (амплитуды импульса). Постройте , например, так:

X=zeros (1,1000);

Y= zeros (1,1000);

Х(1:100)=5; % излучаемый импульс с амплитудой 5 и длительностью 100 отсчетов

Y(301:400)=1; % принятый импульс запоздал на 300 отсчетов

Перемножив скалярно векторы X и Y, получим один отсчет  при фиксированном. Чтобы построить всю функцию, воспользуемся циклом for:

for i=1: 900

R(i)=X(1:100)\*(Y(i:i+99))';

end

plot(R)

hold on

plot(Y\*100,'r')

Можно, конечно, воспользоваться командой conv(X,Y), которая для выполнения свертки (скольжения по τ) предварительно увеличит вдвое длину вектора Y.

У такого способа измерения высоты полета есть два серьезных недостатка: малая помехозащищенность (можно поймать чужой импульс) и пологая форма , не позволяющая гарантировать точность определения задержки при наличии шума. Проверьте, как изменится вид , если посылать более сложный сигнал из двух или трех импульсов. Объясните, чем вызвано появление боковых лепестков и их временное положение.

2. Создайте еще более сложный сигнал, в котором нет периодических повторов импульсов. Его можно создать из случайной последовательности rand(1,m) .логической операцией сравнения с порогом (x>P). Сымитируйте задержку распространения сигнала и получите графики и . Объясните, чем вызвано их отличие.

3. Наложите на принятый (задержанный) сигнал шум с соотношением  Определите минимальное соотношение с/ш, при котором еще возможно измерение задержки.

4. В качестве посылаемого сигнала возьмите отрезок синусоиды (5-10 периодов). Наложите шум и постройте .Как теперь определить величину задержки?

**ЛАБОРАТОРНАЯ РАБОТА № 4**

**Спектр. Ряд Фурье.**

**Цель работы:**

* знакомство со спектральным представлением периодических и случайных процессов;
* изучение взаимосвязи преобразований сигналов во временной и частотной областях;
* оценка дефектов дискретного преобразования Фурье и методы их подавления.

**Теоретические положения.**

Поставим задачу: представить некоторый сложный периодический процесс x(t), заданный на интервале [0,T], в виде суммы простых периодических функций вида



Периодичность x(t) гарантирует, что в таком разложении будут присутствовать только гармоники кратных частот , -целое, . При n=0 получим постоянную составляющую, обычно записываемую в форме , а все разложение будет иметь вид:



называемое *рядом Фурье.*

Коэффициенты этого ряда рассчитываются по формулам:



Если воспользоваться формулой Эйлера , получим более простую запись преобразования Фурье в комплексной форме: - прямое преобразование Фурье или спектр процесса x(t),

-ряд Фурье или обратное преобразование Фурье.

Комплексные коэффициенты  обычно представляют в форме , где  -амплитудный спектр;  -фазовый спектр;  — реальная составляющая спектра;  -мнимая составляющая спектра.

Если реальный непрерывный процесс x(t) не периодический, спектр содержит частоты, не кратные ω0 и носит непрерывный характер.

Широкое распространение спектрального представления сигналов объясняется следующими причинами:

* Гармонические функции – единственные, не меняющие своей формы при прохождении через линейную систему: может измениться только их амплитуда и фаза, но не форма, а, значит, не частота;
* Простота синтеза гармонического колебания – для этого достаточно иметь колебательный контур или любую другую резонансную систему. Разложить в спектр Фурье оптический сигнал может любая двояковыпуклая линза, радиосигналы в эфире тоже представлены электромагнитными волнами – гармониками ряда Фурье;
* Графическое представление спектральных коэффициентов на частотной оси – *спектра сигнала* – позволяет получить наглядную картину распределения в сигнале низких и высоких частот;
* Частотные характеристики используются не только для анализа сигналов, но и для анализа свойств динамических систем.

Чтобы построить спектр с помощью ДПФ (БПФ), надо определить следующие параметры:

* количество спектральных составляющих N;
* шаг между соседними частотами –" разрешение по частоте" ƒ;
* частоту дискретизации ƒд;
* минимальную (нижнюю) частоту спектра;
* верхнюю частоту ƒверх;
* временной интервал анализа Т.

На самом деле, эти параметры жестко связаны друг с другом, и для однозначного построения спектра достаточно задать всего две величины.

Как правило, анализ начинается с выбора временной базы анализа Т и частоты дискретизации ƒд.. При этом оказываются определенными и количество отсчетов сигнала N=T\*ƒд, и минимальная частота спектра: =1/T. А поскольку количество спектральных коэффициентов равно количеству отсчетов сигнала N, оказываются определенными и верхняя частота преобразования ƒверх=\*N=2 ƒд , и шаг между соседними частотами ƒ=.

Совокупность функций для  называется *базисом Фурье.*

На рисунке 3.10 приведены первые восемь косинусных составляющих базиса Фурье, построенных для временного интервала N= 32. При N=32 базис Фурье должен содержать 32 функции. Однако уже 16-я функция на каждом периоде буде содержать всего два отсчета. При таком количестве отсчетов нарисовать более высокочастотные составляющие не удастся: 17-я гармоника будет содержать только 15 периодов, 18-я – 14, и т. д. Фактически, вся вторая половина базиса Фурье будет повторять зеркально отраженную первую половину. Это проявляется эффект нарушения *теоремы Котельникова*:  .



Рис. 3.10 - Восемь косинусных составляющих базиса Фурье

Поэтому и спектр сигнала (сейчас мы рассматриваем только вещественные функции) будет состоять из двух зеркально отраженных картин. Для комплексного спектра этого не происходит, однако в спектре появляются отрицательные частоты.

Преобразование Фурье в дискретной форме (ДПФ) имеет и другие недостатки. Главный из них – далеко не идеальная форма частотной характеристики. Функции базиса Фурье не "выхватывают" из непрерывного (по частоте) спектра реального сигнала отсчеты с целочисленным значением частоты, выкидывая остальное, как ненужное. При преобразовании Фурье происходит как бы "округление по частоте до целого", когда все соседние частоты, отличающиеся дробной частью, должны восприниматься как одна частота. Поэтому ДПФ можно рассматривать как набор полосовых фильтров, разбивающих весь частотный диапазон на N интервалов равной ширины, в каждом из которых производится вычисление усредненных значений амплитуды и фазы. В идеальном случае это разбиение должно быть без перекрытия соседних интервалов (рис. 3.11, пунктир). Реально частотная характеристика значительно шире (рис 3.11, сплошная линия). Это приводит к эффекту "растекания" спектра, когда одна частота воспринимается сразу несколькими фильтрами. Кроме того, непостоянство коэффициента передачи главного лепестка служит причиной \_искажения амплитудных значений спектральных коэффициентов ("эффект частокола").

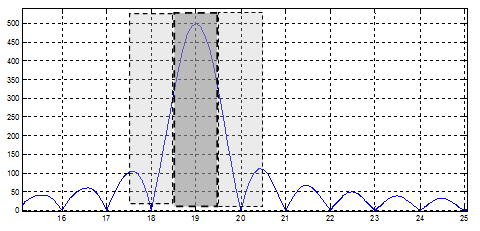


Рис. 3.11 - Преобразование Фурье

Причина описанных дефектов кроется в ограничении верхнего и нижнего предела интеграла Фурье. Представим интеграл Фурье в виде двух операций:





Операция есть ни что иное, как вычисление среднего на ограниченном временном интервале . Частотная характеристика такого фильтра нижних частот (ФНЧ) есть Фурье- преобразование его импульсной характеристики:



Модуль частотной характеристики вида и показана на рис. 4.11 сплошной линией. Однако каким образом один и тот же фильтр ФНЧ может смещаться по частотной оси, формируя всю совокупность частотных интервалов? Это делает первая часть интеграла Фурье — преобразование (). Смысл этого преобразования кроется в тригонометрическом тождестве:  Если вместо углов  и  взять , , легко понять, что *операция перемножения двух гармонических функций приводит к появлению двух новых частот – суммарной и разностной.* В радиотехнике такая операция транспонирования спектра в область низких или высоких частот (это зависит от того, что оставит фильтр – сумму или разность) называют *гетеродинированием*, а само перемножение – модуляцией.

Тогда преобразование Фурье можно рассматривать как N-кратное транспонирование спектра к нулевой частоте базисными функциями Фурье и подавление фильтром ФНЧ тех частот, которые не попали в его зону пропускания

При спектральном преобразовании сигнала могут использоваться и другие – не гармонические – функции. В этом случае говорят об *обобщенном преобразовании Фурье:***.**

Главные требования к функциям  — их *ортогональность и полнота*.

Два вектора Xi, Xj называют *взаимноортогональными,* если их скалярное произведение Y равно нулю: Xi=[xi(1),xi(2),….xi(N)]; Xj=[xj(1),xj(2),….xj(N)]; .

Если при этом норма каждого вектора равна 1: , векторы называют *ортонормальными.*

Для любого вектора размерности (длины) N можно найти N и только N ортогональных векторов. Такой набор векторов называют *полной ортонормальной системой,* или *базисом* N-мерного линейного пространства.

Рассмотрим для примера два базиса в пространстве N=23

Первый базис задается матрицей:



или в графической форме (каждая функция- одна из строк матрицы W3):



Рис. 3.12 - Первый базис

Второй базис зададим матрицей:

A =

0.1250 0.5026 0.4275 0.8550 0.2138 0.6917 0.4275 0.8550

0.2138 -0.2939 0.7310 -0.5000 0.3655 -0.4045 0.7310 -0.5000

0.2513 0.4755 -0.2939 -0.8090 0.4297 0.6545 -0.2939 -0.8090

0.3458 -0.3455 -0.4045 0.5878 0.5914 -0.4755 -0.4045 0.5878

0.5026 0.8090 0.8090 1.0000 -0.2939 -0.5878 -0.8090 -1.0000

0.5026 -0.8090 0.8090 -1.0000 -0.2939 0.5878 -0.8090 1.0000

0.6917 0.8090 -0.5878 -1.0000 -0.4045 -0.5878 0.5878 1.0000

0.6917 -0.8090 -0.5878 1.0000 -0.4045 0.5878 0.5878 -1.0000

или графически:



Рис. 3.13 - Второй базис

Сравним эти базисы. Прежде всего бросается в глаза равенство всех элементов матрицы W3. Это очень удобное свойство базисных функций: при умножении на такую функцию временные затраты минимальны. Во втором базисе это свойство потеряно. Первая строка базиса обычно служит для выделения постоянной составляющей из исследуемого сигнала и в классических базисах все ее элементы равны единице. При этом все остальные функции не реагируют на постоянную составляющую вследствие симметричности их относительно нуля. Этого во втором базисе тоже нет. Поэтому первый базис используется очень широко и носит название **базиса Уолша**, а второй базис кроме вас никто не видел.

Приведем еще один весьма широко используемый базис – базис Хаара (рис. 3.14):





Рис. 3.14 - базис Хаара

Его главное достоинство- большое количество нулевых элементов. Это не только ускоряет вычисления спектра, но и обеспечивает сжатое представление сигналов с резкими перепадами. Это объясняется тем, что каждая функция (за исключением первых двух) выделяет только некоторую локальную область существования сигнала.

Для упрощения здесь базис Хаара приведен в ненормированной форме: чем уже импульс, тем меньше его энергия.

**Порядок проведения работы.**

Используемые операторы Матлаба:

fft (x) –fast Fourier transform. — возвращает комплексные отсчеты спектра x;

ifft(у) – обратное преобразование Фурье

abs(fft(x)) – модуль спектра;

angle(y) , phase(y) – фаза спектра;

real(y), imag(y) – реальная и мнимая составляющая (y)

kron(A,B)- кронекеровское произведение матрицы В на А, при котором каждый элемент матрицы А умножается на всю матрицу В.

1. Создайте два сигнала

x1=cos(2\*pi\*f1\*t) ;

x2=4\*cos(2\*pi\*f2\*t).

Частоты f1 и f2 задаются преподавателем. Временной интервал – базу анализа — выберите так, чтобы более низкочастотный сигнал имел на нем 3-5 периодов. Задайте частоту дискретизации так, чтобы на периоде сигнала высокой частоты укладывалось 4-10 отсчетов.

Получите модуль спектра сигнала, постройте его график. Объясните, чем определяется номер гармоники в спектре.

2. Создайте еще два сигнала: x3=x1+x2; x4=x1.\*x2 и постройте их спектры. Объясните полученный результат.

На временном интервале  отсчетов создайте δ-импульс и получите его спектр (модуль и фазу). Как изменится спектр, если сдвинуть δ-импульс?

В цикле for последовательно увеличивайте ширину импульса, наблюдая соответствующие изменения его спектра. Для произвольной ширины импульса рассчитайте спектр вручную. Сделайте выводы.

3. На том же временном интервале создайте периодический прямоугольный сигнал со скважностью 2 (меандр) и количеством периодов, кратным двум. Постройте его спектр. Рассчитайте спектр вручную.

Операцией kron(A,А),где , создайте базис Уолша для . Постройте спектр того же сигнала. Сравните спектр Фурье и Уолша.

4. Определите форму и ширину частотной характеристики двух соседних каналов анализатора Фурье. Это можно сделать в цикле for, изменяя частоту анализируемого сигнала с достаточно малым шагом (0.1 – 0.2) и выделяя из спектра только отчет, принадлежащий выбранному каналу. Оцените, как меняется спектр моногармонического сигнала при его смещении по частотной оси. Для улучшения качества анализа используйте вместо прямоугольного временного окна, обрезающего сигнал, окно Ханнинга (hanning) Сравните результаты.

1. [↑](#footnote-ref-1)