МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ им. М.В.ЛОМОНОСОВА

Механико-математический факультет

П.А.Кручинин

ОСНОВЫ ПРОГРАММИРОВАНИЯ В СРЕДЕ MATLAB.

М о с к в а 2001 год

В СРЕДЕ MATLAB Учебное пособие. — Издательство механико-математического факультета МГУ

Настоящее пособие представляет собой описание основных возможностей пакета MATLAB.
Для студентов и аспирантов математических факультетов университетов, специализирующихся в области прикладной математики и механики.

Рецензенты

??????????????????????????????????????
??????????????????????????????????????

Доктор физико-математических наук, профессор *Л.А.Мироновский*,
Доктор физико-математических наук *??????????*

Содержание

Введение

В настоящее время широкое распространение получили интегрированные среды математических вычислений, позволяющие решать сложные математические задачи. Одной из наиболее распространенных систем этого типа является MATLAB.

Разработка пакета была начата в 1972 году по инициативе известных американских специалистов в области вычислительной математики Дж. Форсайта и Дж. Уилкинсона. Первоначально MATLAB создавался как пакет программ, реализующих наиболее эффективные численные алгоритмы линейной алгебры. Наполнение пакета проходило также и в направлении расширения возможностей графического представления результатов вычислений, облегчения вывода результатов на печать и т.д. С появлением ПЭВМ и ОС типа Windows, разработчики MATLAB'а (фирма Mathworks) создала достаточно удобную среду.

К настоящему моменту MATLAB представляет собой интегрированную вычислительную среду включающую язык программирования высокого уровня, средства редактирования, отладки и выполнения программ. Отметим также, что язык программирования MATLAB'а с одной стороны обладает упрощенным синтаксисом, что облегчает его освоение неопытным пользователем, а с другой стороны позволяет опытному пользователю создавать законченные приложения, использующие разработанные функции на некоторых других языках программирования.

Различным аспектам работы в системе MATLAB посвящена обширная литература [1]-[11]. В большинстве своем представленная литература носит справочный характер и мало пригодна для первого знакомства с пакетом. Исключение составляют брошюры [6] - [9], опирающиеся на соответствующие учебные курсы.

Появление настоящего учебного пособия вызвано с одной стороны потребностью в компактном изложении первоначальных сведений о языке программирования MATLAB'а. Это пособие ориентировано на использование его студентами механиками механико-математического факультета МГУ, на начальных стадиях своей научной деятельности. В нем автор постарался изложить минимальный объем сведений о функциях пакета используемых при написании курсовых работ студентами кафедры прикладной механики и управления. В связи с этим в пособии более подробно, чем в традиционном изложении, рассматриваются функции MATLAB'а, реализующие методы, знакомству с которыми в лекционных курсах механико-математического факультета МГУ уделяется недостаточное внимание. Изложение снабжено иллюстрирующими примерами.

Автор старался подать материал таким образом, чтобы данное пособие можно было использовать в качестве самоучителя. При этом старался отметить особенности языка, функций библиотеки стандартных программ и использованных в ней вычислительных методов, не очевидные для неискушенного читателя.

От читателя пособия, пытающегося реализовать полученные сведения в своей практической деятельности, не требуется предварительного знания каких-либо языков программирования. Тем не менее автор предполагал, что читатель хорошо знаком с традиционным дизайном окон Windows и умеет ими пользоваться.

1 Команды и функции пакета MATLAB

Запуск системы MATLAB в системе Windows не представляет собой сколько-нибудь значительных проблем. После запуска программы MATLAB на экране компьютера появляется главное окно, содержащее меню, инструментальную линейку с кнопками и клиентскую часть окна с приглашением ≫. Эту часть принято называть командным окном.

В командном окне с клавиатуры вводятся команды, состоящие из букв, цифр и знаков операций. Нажатие клавиши "Enter"служит сигналом для выполнения набранной команды.

1.1 Полезные команды пакета.

Знакомство с пакетом MATLAB полезно начать с команд вызова операторов общего назначения призванных облегчить работу пользователя. К этим операторам отнесем следующие

- *help* – выводит список подключенных TOOLBOX'ов.
- *help help* – выводит информацию о работе справочника help.
- *help « имя функции »* – выводит информацию о функции. Информацией о функции считается текст помещенный в начальные строки комментариев М-файла с текстом функции, помеченных символом '%'. Для встроенных функций пакета в поддиректориях директории TOOLBOX заводится М-файл, содержащий только строки комментариев. Имя этого файла совпадает с именем функции.
- *help « имя файла»* – выводит информацию о содержимом М-файла аналогично информации о функции.
- *lookfor « набор символов»* – выводит список М-файлов, доступных для обращения в текущий момент времени, у которых в строках комментариев встречается приведенный набор символов. Эту команду удобно использовать для поиска по ключевому слову или словосочетанию необходимых пользователю функций.
- *demo* – выводит список демонстрационных примеров.
- *tour* – предлагает меню обзор демонстрационных примеров (MATLAB 5).
- *!« команда»* – выполняет команду DOS.
- *casesen « on/off»* – переключает режим различения регистров при отсутствии опций, опция *on* устанавливает режим различения регистров, а опция *off* - отменяет его.
- *diary « имя файла/on/off »* – управляет режимом параллельной записи в файл. Если указано имя файла, то устанавливается режим, при котором все, что выводится на экран одновременно записывается в файл с этим именем. Опция *off* прерывает этот режим, а опция *on* возвращает снова. Существующий файл открывается на дозапись.
- *format « flag »* – управляет форматом вывода результатов в командное окно. Все вычисления в MATLAB'е производятся с точностью 32-разрядной арифметики. Однако вывод значений в командное окно производится с меньшей точностью. При следующих значениях параметра *« flag »* устанавливаются следующие виды формата вывода
  - *short* – формат с фиксированной точкой с 5 значащими цифрами (используется по умолчанию). Этот формат устанавливается при обращении *format* без указания опции *flag*.
  - *long* – формат с фиксированной точкой с 15 значащими цифрами.
  - *short e* – формат с плавающей точкой с 5 значащими цифрами.
  - *long e* – формат с плавающей точкой с 15 значащими цифрами.
  - *short g* – выбирается лучший из двух предшествующих форматов с 5 значащими цифрами.
  - *long g* – выбирается лучший из двух предшествующих форматов с 15 значащими цифрами.
  - *hex* – шестнадцатеричный формат.
  - *+* – отображается только знак.
  - *rat* – приближение результата в виде отношения целых чисел.

Эти же форматы вывода можно установить в окне *General* в результате вызова пункта подменю *Preferences* в пункте *File* главного меню.

- *home* – устанавливает курсор с текущей командной строкой в верхний левый угол окна.
- *clc* – очистка командного окна.
- *computer* – дает справку о типе компьютера (Для операционных систем после Windows 95 и Windows NT и их наследников команда практически бесполезна, так как сообщает только тип операционной системы).
- *exit, quit* – эти команды прекращают работу MATLAB.
- ; – очень полезный символ. Вывод на экран результата выполнения команды в рабочее окно подавляется, если команда заканчивается этим символом.

1.2 Переменные. Вектора и матрицы.

Имена переменных в MATLAB'е могут обозначаться произвольным набором букв, цифр и знаков подчеркивания ('_'). Они должны начинаться с буквы и содержать не более 31 символа. Последующие символы будут проигнорированы. При этом не рекомендуется использовать имена операторов и функций MATLAB'а и имена стандартных переменных используемых MATLAB'ом:

- i, j – мнимая единица ($\sqrt{-1}$);
- inf – неопределенность типа $1/0 \quad (\infty)$;
- NaN – неопределенность типа $0/0$ (Not a number);
- ans – результат последней выполняемой операции или функции (если в командной строке отсутствует оператор присваивания, результат операции присваивается переменной ans автоматически);
- pi – $\pi = 4 * atan(1) = 3.1415926\ldots$;
- $rand$ – псевдослучайное число равномерно распределенное на интервале $[0, 1]$;
- eps – относительная точность вычислений. За эту величину принимается расстояние от 1 до следующего действительного числа доступного компьютеру;
- $realmin, realmax$ – минимальное и максимальное действительные числа.

Основным объектом MATLAB'а является матрица. Вектором называется матрица, один из размеров которой равен единице, скаляром - матрица 1×1. Все основные функции и операторы в MATLAB'е - матричные, кроме случаев, оговариваемых особо.

Ввести скаляр можно простым оператором присваивания $a = 2.34$.

Для формирования матриц и векторов используются символы «[...] ». Например матрицу 2×3 можно записать в виде $a = [1 \quad 2.3 \quad 4; 3.4 \quad 2 \quad 3]$. Различные элементы в такой записи разделяются пробелом (либо запятой), строки отделяются друг от друга символом ';'. Различное число чисел в строках или столбцах порождает ошибку, о которой MATLAB сообщает пользователю. В соответствии с вышеописанным оператор $b = [1 \quad 2 \quad 3]$ вводит вектор строку, а $c = [1; 2; 3]$ - вектор столбец.

Подобными правилами можно также пользоваться при формировании блочных матриц. Например, если выполнены присваивания предыдущего абзаца, оператор $d = [a; b]$ порождает матрицу

$$d = \begin{bmatrix} 1 & 2.3 & 4 \\ 3.4 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

Обращение к элементу матрицы a, расположенному на пересечении строки с номером i и столбца с номером j имеет вид $a(i, j)$. Вектор столбец совпадающий с j-м столбцом матрицы a получим в результате обращения $a(:, j)$, а i-й вектор-строку – $a(i, :)$.

Подматрицу матрицы a можно получить, указав вместо индексов вектора u и v, содержащие номера строк и столбцов матрицы a, на пересечении которых расположены элементы подматрицы. Поясним это на примере: для матрицы d приведенной ранее и векторов $u = [1 \quad 3]$ и $v = [1 \quad 2]$

$$a = d(u, v) = \begin{bmatrix} 1 & 2.3 \\ 1 & 2 \end{bmatrix} \tag{1}$$

В общем случае значения компонент векторов u и v в приведенном операторе при вычислении индексов будут округлены до ближайшего целого числа.

Блок матрицы можно получить также использовав символ перечисления ':'. Дело в том, что в MATLAB'е оператор

$$u = start : step : fin \tag{2}$$

задает вектор, состоящий из конечного числа членов арифметической прогрессии первый член которой равен $start$, шаг прогрессии $step$, а последний есть ближайший к значению fin член прогрессии, принадлежащий сегменту, граничными значениями которого являются $start$ и fin. При $step > 0$ последним компонентом сформированного вектора будет максимальный член, принадлежащий сегменту $[start, fin]$, а при $step < 0$ - минимальный член, принадлежащий сегменту $[fin, start]$ В записи $v = start : fin$ величина $step$ принимается равной 1. Использовав такой способ записи матрицу a из (1) можно также получить, например, в результате записи $a = d(1 : 1.6 : 3, 1 : 2)$.

В MATLAB'е можно также задать символьные переменные, которые представляют собой вектора символов. Проще всего задать такую переменную командой типа

c='Hallo!'

Она эквивалентна команде *c=['H','e','l','l','o','!']*.

Необходимые операции над строками в MATLAB'е можно выполнять по правилам векторных операций, описанных в этом параграфе.

1.3 Простейшие арифметические операции.

Здесь уместно привести символьные обозначения основных операций, проводимых над переменными:

- = – присваивание;
- + – сложение;
- * – умножение;
- \ – деление слева (для матричных величин результат выполнения операции $X = A\backslash B$ примерно то же, что и $A^{-1} * B$, и эквивалентен решению матричного уравнения $A * X = B$, вычисленное методом исключения Гаусса. Если плохо обусловлена или вырождена – выводится предупреждение. Если матрица размера $\times N$, а B – столбец длиной M, решение матричного уравнения $A*X = B$ производится методом наименьших квадратов (см. также функцию *pinv*).;
- / – деление справа (для матричных величин результат выполнения операции $X = A/B$ примерно то же, что и $A * B^{-1}$, и эквивалентен решению матричного уравнения $X * B = A$ подобно тому, как это реализовано для оператора \);
- ^ – возведение в степень;
- ' – –.. * – – ;
- ./ – поэлементное деление справа;
- .^ – поэлементное возведение в степень;

Эти операции выполняются с учетом требований традиционной матричной алгебры. Существенным достоинством пакета MATLAB является организованная проверка размерностей, осуществляемая при проведении вычислений. При несоответствии размерностей операндов выдается сообщение об ошибке.

Последовательность операций определяется в соответствии с обычными алгебраическими правилами, причем для выделения первоочередных операций традиционно используются круглые скобки (. . .).

1.4 Рабочая область и операции над ней.

Для переменных, заданных при работе в MATLAB'е, выделяется область памяти компьютера. Область памяти, в которой храняться доступные для вычислений переменные называется рабочей областью (*workspace*). Просмотреть содержимое рабочей области позволяют команды *who* и *whos*.

- *who* - выводит на экран список переменных, хранящихся в рабочей области.

- *whos* - выводит на экран список этих переменных, дополненный информацией об их размере и типе. В качестве примера приведем результат работы этой команды для случая, когда при работе в MATLAB 5.2 определены следующие действительные переменные - скаляр a, вектор строка b из 3-х элементов, матрица c размера 2×3 и комплексный вектор-столбец d из 2-х элементов.

Name	Size	Bytes	Class
a	1x1	8	double array
b	1x3	24	double array
c	2x3	48	double array
d	2x1	32	double array (complex)

Grand total is 12 elements using 112 bytes

Проверить наличие переменной a в рабочей области памяти позволяет функция $exist('a')$. Она возвращает 1, если переменная a существует в рабочем пространстве; 2 - если 'a' -это имя файла на диске, 0 - если a не существует.

Для сохранения и восстановления переменных, размещенных в рабочей области используются команды *save* и *load*.

- *save* – сохраняет все переменные рабочей области в бинарном файле *matlab.mat*.
- *save « имя файла»* – сохраняет все переменные рабочей области в бинарном файле с расширением *.mat*. Это же действие можно совершить также в результате вызова пункта подменю *Save Workspace as* в пункте *File* главного меню.
- *save « имя файла» « переменные»* – сохраняет переменные в бинарном mat-файле с заданным именем. В качестве переменных задается список их имен разделенных пробелом. Например
- *save « имя файла» « переменные» -ascii* – сохраняет переменные в файле, в стандартных кодах ASCII. Имя файла может иметь произвольное расширение.
- *load* – восстанавливает переменные из файла *matlab.mat* .
- *load « имя файла»* – восстанавливает переменные в рабочую область из *mat*-файла с заданным именем. Это же действие можно совершить также в результате вызова пункта подменю *Load Workspace* в пункте *File* главного меню.
- *load « имя файла» -ascii* – читает в рабочую область из файла, матрицу записанную в кодах ASCII в виде таблицы, столбцы которой разделены пробелом. Умные версии MATLAB'а (4 и выше) при отсутствии ключа $-ascii$ сами благополучно разбираются в чем дело и работают нормально. Результат выполнения этой операции помещается в массив, имя которого совпадает с именем файла (естественно с отброшенным расширением). Понятно, что имя такого файла должно начинаться с буквы и не должно содержать знаков типа '–', '+' и т.д. [1)]

Наконец удалить переменные из рабочей области помогает команда *clear*.

- *clear* – удаляет все переменные из рабочей области.
- *clear « переменные»* – удаляет перечисленные списком через пробел переменные.

1.5 Рабочая папка.

При отсутствии указания полного пути к читаемым или записываемым файлам пакет MATLAB осуществляет поиск файла для чтения и производит запись в папку, называемую рабочей. Все версии MATLAB'а позволяют установить путь в рабочую папку, выбранную пользователем. Универсальным является способ, использующий команду *cd*

- *cd « путь»* – устанавливает папку, указанную переменной *« путь»* в качестве рабочей.

[1)] При желании последнее требование можно нарушить, но тогда имя файла должно везде присутствовать только в виде символьной константы. Работа с таким именем сильно затруднена и ее описание отсутствует.

Просмотреть содержимое этой папки позволяют команды

- *dir* – выводит список файлов, содержащихся в рабочей папке;
- *what* – выводит список файлов MATLAB'а ($*.m$, $*.mat$ $*.mex$), содержащихся в рабочей папке. [2)]

В современных версиях MATLAB'а существуют способы установить рабочую папку, используя инструментальную панель. Начиная с версии 5.0 в ниспадающем меню пункта *"File"*
главного меню командного окна введен пункт установки путей *"Set Path..."*
, при обращении к которому предоставляется возможность сменить параметр *"Current directory"*
, указывающий путь в рабочую папку. Значек этого действия выведен также и на панель инструментов. Во всех вариантах 4-й версии специальный пункт меню отсутствует, однако для смены рабочей папки достаточно попытаться открыть файл, использовав подпункт *"open"*
в ниспадающем меню пункта *"File"*
главного меню командного окна. При такой попытке MATLAB предлагает выбрать путь в папку, в которой находится файл предназначенный пользователем для чтения и редактирования. После открытия такого файла редактором, папка в которой он находился устанавливается в качестве рабочей. Если в выбранной папке пользователь не желает открывать никаких файлов, то можно отказаться в последний момент от операции открытия файла, выбрав в окне открытия файла кнопку *"cancel"*
. Последняя из папок, просмотренных пользователем будет установлена в качестве рабочей.

1.6 Программирование в MATLAB'е. Сценарии и функции..

До сих пор читателю предлагался простой способ работы в системе MATLAB – последовательное введение команд в командном окне, немедленная их обработка пакетом и выдача вычисленного результата. Более эффективным представляется способ предварительной подготовки последовательности команд, запись ее в виде файла и последующее неоднократное выполнение этих команд. В MATLAB'е существует два типа файлов, позволяющих реализовать такую возможность: *M-сценарий* и *M-функция*.

- *M-сценарием (Script-файлом)* называется файл, содержащий последовательность команд, операций и функций MATLAB'а, выполняющих действия над переменными рабочей области. M-сценарий не имеет входных и выходных аргументов.
- *M-функцией* (в дальнейшем просто функцией) называется последовательность команд и операций, помещенная в файл снабженный специальным заголовком, и осуществляющая операции только над входными переменными.

Дав определение разберемся в существе дела.

1.6.1 Сценарии.

Script-сценарий помещается в текстовый файл с расширением $*.m$ и содержит последовательность операторов MATLAB'а. В современных версиях MATLAB'а *script*-сценарий не требует дополнительных заголовков или комментариев. В качестве примера приведем следующий простейший *script*-сценарий

```
c=3.5;
d=a/c*cos(b);
e=a*(c*sin(b)+cos(pi-b));
f=e/d
```

Создайте в текущей рабочей папке любыми удобными Вам средствами файл с именем *my1prog.m* (в версиях MATLAB 5.0 и выше для этого удобно воспользоваться программой *MATLAB Editor/Debugger*), наберите в нем вышеуказанный текст и запишите его на диск. Теперь введем в командном окне значения $a = 7; b = pi/4;$ и наберем команду *my1prog*, которая вызывает выполнение указанной последовательности операций над переменными, хранящимися в рабочей области памяти. Все использованные переменные c, d, e и f также сохраняются в рабочей области памяти и доступны для дальнейших операций.

В программах обрабатывающих большие массивы даннных сохранение воспомогательных переменных и массивов крайне неудобно, так как ведет к излишнему засорению оперативной памяти. Из-за этого при программировании на языке MATLAB'а введено понятие функции.

[2)] Заметим, что среди *mex* файлов перечисляются и файлы $*.dll$.

1.6.2 Функции.

Текст функции также помещается в текстовый файл с расширением $*.m$. Однако этому файлу предъявляются некоторые дополнительные требования. Первый выполняемый оператор файла содержит описание функции вида

function [outpars]=progname(inpars)

Это описание сообщает о том, что в файле записана функция с именем *progname*, которая имеет входные параметры, перечисленные в списке *inpars*, и выходные параметры в списке *outpars*. Для примера переделаем предыдущий сценарий в функцию, которая получает входные параметры a и b, и вычисляет величины e и f.

Хорошим тоном считается несколько строк функции, следующих за начальным описанием, посвятить комментариям к функции. Эти строки начинаются символом % и как правило содержат описание обращения к функции, информацию о входных и выходных параметрах и комментируют нетрадиционные методы вычислений. Указанные комментарии будут выдаваться в командное окно MATLAB'а при вызове оператора вида *help «progname»*. Комментарии в файлах могут быть написаны и на русском языке, однако при их выводе программой по команде *help* в различных версиях могут возникать трудности.

Выполнение функции прекращается в следующих случаях:

- все операторы выполняются последовательно до конца файла;
- очередной выполняемый оператор *return* прекращает выполнение текущей функции и передает управление функции, его вызвавшей;
- признаком завершения тела функции может служить также команда объявления следующей функции. Пример такого типа будет приведен позже в параграфе 5.4.

```
function [e,f]=my1prog(a,b)
% function [e,f]=my1prog(a,b)
% It is my first function in MATLAB
% Inputs: scalar constants a and b
% Outputs: scalar constants e and f
% Calculation formulas is in text of file.
c=3.5;
d=a/c*cos(b);
e=a*(c*sin(b)+cos(pi-b));
f=e/d;
```

Теперь обращение к функции из командного окна может иметь вид

$$[e1, f1] = my1prog(a, b);$$

В качестве формальных параметров можно было также напрямую указать их значения 7 и $pi/4$.

При обращении *help my1prog* в командном окне появляется надпись

```
function [e,f]=my1prog(a,b)
It is my first function in MATLAB
Inputs: scalar constants a and b
Outputs: scalar constants e and f
Calculation formulas is in text of file.
```

Сделаем ряд полезных замечаний, поясняющих особенности работы функций в пакете MATLAB.

1. При работе в MATLAB'е на первых порах рекомендуется соблюдать правило: один файл – одна функция.
2. Главным именем функции служит имя файла, в котором функция записана. В случае если имя записанное в описании функции не совпадает с именем файла, при обращении к функции MATLAB будет искать файл с указанным именем. Поэтому соблюдайте правило – имя функции и имя файла совпадают.
3. Функция в MATLAB'е может вызывать другую функцию и число вложений строго говоря не ограничено. Это дает обширные возможности к созданию рекурсии (автовызова функции) и зацикливанию программ. Опасайтесь этой возможности. Основной недостаток MATLAB'а состоит в том, что прервать выполнение такой функции возможно только в результате аварийного прерывания работы при одновременном нажатии клавиш $CTRL - BREAK$.

4. Входные параметры a и b передаются функции по значению, и произвольные их изменения при работе функции не отражается на значениях в командном окне или вызывающей функции.

5. В пакете MATLAB каждая функция обязательно имеет две переменные *nargin* и *nargout*, которые позволяют получить информацию о количестве входных и выходных параметров функции.

1.7 Наиболее употребительные стандартные функции пакета

В пакете MATLAB реализован большой набор стандартных функций. Эти функции объединены в пакеты - TOOLBOX'ы. Все TOOLBOX'ы как правило размещаются в поддиректории TOOLBOX. Основным TOOLBOX'ом следует считать TOOLBOX с именем MATLAB. Файлы этого TOOLBOX'а размещены в поддиректориях этой директории, однако значительная их часть содержит только описание соответствующих встроенных функций, чье выполнение осуществляется непосредственно ядром программы. Набор таких функций достаточно велик и достаточно полное описание их содержится в документации пакета MATLAB [13] и в многочисленной литературе [1]- [3] и [5]. Поэтому здесь ограничимся только кратким описанием наиболее употребительных функций.

Первой отметим функцию $disp(x)$. Эта функция не имеет выходных параметров и выводит в рабочее окно MATLAB'а значение переменной x.

1.7.1 Элементарные математические функции

Аргументами всех указанных в этом пункте функций – матрицы, как и результаты вычислений. Функции вычисляются для заданных матриц поэлементно.

- $y = exp(x)$ - поэлементная экспонента ($y_{ij} = e^{x_{ij}}$),
- $y = sin(x)$ - синус,
- $y = cos(x)$ - косинус,
- $y = tan(x)$ - тангенс,
- $y = asin(x)$ - арксинус,
- $y = acos(x)$ - арккосинус,
- $y = atan(x)$ - арктангенс,
- $y = sqrt(x)$ - квадратный корень,
- $y = abs(x)$ - абсолютная величина,
- $y = log(x)$ - натуральный логарифм,
- $y = log10(x)$ - десятичный логарифм,
- $y = sign(x)$ - сигнум-функция,

$$y_{ij} = \begin{cases} 1 & ,\text{если} x_{ij} > 0 \\ 0 & ,\text{если} x_{ij} == 0 \\ -1 & ,\text{если} x_{ij} < 0 \end{cases}$$

1.7.2 Функции округления и им сопутствующие.

- $y = round(x)$ - округление элементов x до ближайшего целого (округление в традиционном смысле),
- $y = ceil(x)$ - округление элементов x до ближайшего целого в сторону увеличения,
- $y = fix(x)$ - округление элементов x до ближайшего целого в сторону нуля (служит для выделения целой части чисел),
- $y = floor(x)$ - округление элементов x до ближайшего целого в сторону уменьшения,
- $y = rem(x, y)$ - остаток. $rem(x, y) = x - y * N$, где $N = fix(x/y)$.
- $y = gcd(x, y)$ - наибольший общий делитель для элементов матриц x и y.

1.7.3 Функции над комплексными числами

Комплексные числа в MATLAB'е задаются простыми операциями вида $a = 6.7 + 7.897i$, либо $d = a + i * b$, если a и b заданные ранее действительные переменные. Комплексные числа могут являться элементами матрицы и соответственно, если аргументами нижеследующих функций являются комплекснозначные матрицы, то функции вычисляются поэлементно.

- $y = angle(x)$ - аргумент комплексного числа.
- $y = conj(x)$ - комплексное сопряжение.
- $y = imag(x)$ - мнимая часть.
- $y = real(x)$ - действительная часть.
- $y = sign(x)$ - сигнум-функция, для комплексного x, $sign(x) = x./abs(x)$.

Здесь же упомянем функции преобразования типов переменных

- *num2str* – преобразует числовую переменную в строку символов;
- *str2num* – преобразует строку символов в числовую переменную. Строка может содержать цифры, десятичные точки, знаки '+' и '-', обозначение мнимой единицы i и букву 'e'при представлении числа с плавающей запятой.

1.7.4 Формирование векторов и матриц

- $y = linspace(xmin, xmax)$ - формирует вектор y из 100 элементов равномерно расположенных между точками $xmin$ и $xmax$. Если вектор должен содержать другое число компонент N, обращение к этой функции должно иметь вид $y = linspace(xmin, xmax, N)$.
- $y = logspace(xmin, xmax)$ - формирует вектор y из 100 элементов логарифмически расположенных между точками $xmin$ и $xmax$. Если вектор должен содержать другое число компонент N, обращение к этой функции должно иметь вид $y = linspace(xmin, xmax, N)$.
- $y = zeros(N, M)$ - формирует нулевую матрицу размерности $N \times M$. При наличии единственного операнда N формируется квадратная матрица размерности $N \times N$.
- $y = ones(N, M)$ - формирует матрицу размерности $N \times M$, все элементы которой равны 1. При наличии единственного операнда N формируется квадратная матрица размерности $N \times N$.
- $y = eye(N)$ - формирует единичную матрицу I размерности $N \times N$.
- $y = rand(N, M)$ - формирует матрицу размерности $N \times M$, элементами которой являются случайные числа, равномерно распределенные на интервале $(0.0, 1.0)$.
- $y = diag(v, n)$ - формирует матрицу, на n-ой наддиаганали которой расположен вектор v. При $n = 0$ результатом выполнения этой функции является диагональная матрица, на диагонали которой расположены элементы вектора v. При значении $n < 0$ компоненты вектора v расположены на поддиагонали с номером $|n|$. Например $diag([1, 2, 3], 2)$ порождает матрицу

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

  а $diag([5, 2], -1)$ - матрицу

$$\begin{bmatrix} 0 & 0 & 0 \\ 5 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}.$$

- $y = diag(A, n)$ - вырезает n-ную наддиагональ из матрицы A. Матрица может не являться квадратной. При $n < 0$ выводится поддиаганаль с соответствующим номером.
- Для поворота векторов и матриц используются следующие функции:

- $fliplr(A)$ - переставляет столбцы матрицы A симметрично относительно вертикальной оси (в версиях MATLAB'а с номерами меньше 5-го существовало также обращение $flipy(A)$);
- $flipud(A)$ - переставляет строки матрицы A симметрично относительно горизонтальной оси (в версиях MATLAB'а с номерами меньше 5-го существовало также обращение $flipx(A)$);
- $rot90(A, n)$ - n раз поворачивает матрицу A на 90^o против часовой стрелки. При отсутствии параметра n производится поворот матрицы на 90^o. При нецелом значении n функция не производит никаких действий, не выдавая ни предупреждений ни сообщений об ошибке.

Кроме того имеются функции, генерирующие "именные"матрицы Ганкеля, Теплица, Гильберта, Адамара, Уилкинсона и т.д. Эти функции носят соответствующие имена и о их вызове проще всего узнать использовав оператор *help*. При этом следует учитывать, что эти имена записываются в общепринятой транскрипции языка оригинала: *hankel*, *toeplitz*, *hilb*, *hadamard*, *wilkinson* и т.д.

1.7.5 Операции над векторами и матрицами

- $y = cross(u, v)$ - векторное произведение двух векторов u и v длины 3;

Если для следующих функций аргументами являются вектора, то результатом их применения является скаляр. Если же аргументами являются матрицы, то результат вычислений вектор, j-й компонент которого - результат применения соответствующей функции к j-ому столбцу матрицы.

- $y = sum(x)$ - сумма компонент вектора ;
- $y = prod(x)$ - произведение компонент вектора ;
- $y = min(x)$ - минимальная компонента вектора;
- $y = max(x)$ - максимальная компонента вектора;
- $y = mean(x)$ - среднее арифметическое компонент вектора;
- $y = std(x)$ - среднее квадратическое отклонение компонент вектора;
- $y = median(x)$ - медианная компонента вектора. Под медианой в этом случае понимается величина, полученная в результате следующих операций: все n компонент вектора располагаются в порядке возрастания и в новом векторе выбирается компонента с номером $(n + 1)/2$ для нечетных n и среднее арифметическое компонент с номерами $n/2$ и $n/2 + 1$ для четных n;
- $y = finite(x)$ - возвращает 1, если все компоненты вектора x конечны и 0 в противном случае;
- $y = isNaN(x)$ - возвращает 1, если хотя бы один из компонентов вектора x принимает значение NaN и 0 в противном случае;
- $y = all(x)$ - возвращает 1, если все компоненты вектора x не равны и 0 в противном случае;
- $y = any(x)$ - возвращает 1, если хотя бы одна компонента вектора x не равны 0 и 0 в противном случае;;
- $y = find(x)$ - возвращает номера ненулевых компонент вектора. В случае, если аргумент x является матрицей, используется сквозная построчная нумерация;

Если для следующих функций аргументами являются вектора, то результатом их применения является вектор. Если же аргументами являются матрицы, то результат вычислений матрица, j-й столбец которой - результат применения соответствующей функции к j-ому столбцу исходной матрицы.

- $y = cumsum(x)$ - кумулятивная сумма компонент (i-й компонент вектора y равен сумме i первых компонент вектора x);
- $y = cumprod(x)$ - кумулятивное произведение компонент вектора (i-й компонент вектора y равен произведению i первых компонент вектора x);
- $y = sort(x)$ - сортирует компоненты вектора по возрастанию (столбцы в матричном x сортируются независимо);

- $y = diff(x)$ - конечноразностное дифференцирование вектора. Для вектора x длиной N вектор y имеет длину $N-1$ и $y_k = x_{k+1} - x_k$. Для матрицы x производится дифференцирование столбцов $y = x(2:N,:) - x(1:N-1,:)$. При обращении $y = diff(x,n)$ - аналогичным образом вычисляется n-ая конечноразностная производная;
- $[Px, Py] = gradient(x)$ - вычисление конечноразностных матричных градиентов. При обращении $[Px, Py] = gradient(x, Dx, Dy)$ аргументы Dx и Dy задают величины шага по осям x и y.
- $z = trapz(x, y)$ - вычисляет методом трапеций интеграл по значениям ординат y, которые интерпретируются как значения заданной функции в точках с абсциссами, записанными в x.

1.7.6 Нормы векторов и матриц

$y = norm(v, N)$ - норма вектора или матрицы. Функция вычисляет различные нормы в зависимости от того v - вектор или матрица, и от значения, которое принимает аргумент N:

v - вектор (матрица один из размеров которой равен 1)

- $N = 2$ или этот аргумент отсутствует - евклидова норма вектора v

$$||v||_2 = \left(\sum_{i=1}^{n} v_i^2 \right)^{1/2}$$

- $N = 1$ - первая норма вектора v

$$||v||_1 = \sum_{i=1}^{n} |v_i|$$

- $N = inf$ - бесконечная (чебышевская) норма вектора v

$$||v||_\infty = \max_i |v_i|$$

- $N = -inf$ - норма вектора v вида

$$||v||_{-\infty} = \min_i |v_i|$$

- $N = p$ - для любого p величина вида

$$||v||_p = \left(\sum_{i=1}^{n} v_i^p \right)^{1/p}$$

$y = norm(A, N)$, где A - матрица

- $N = 2$ или этот аргумент отсутствует - спектральная норма матрицы A равна наибольшему сингулярному числу этой матрицы

$$||A||_2 = \max_i \lambda_i^{1/2} \left(A^T A \right) = max_i \sigma_i(A)$$

- $N = 1$ - первая (столбцовая) норма матрицы A

$$||A||_1 = \max_j \sum_{i=1}^{n} |A_{ij}|$$

- $N = inf$ - бесконечная (строчная) норма матрицы A

$$||A||_\infty = \max_i \sum_{j=1}^{m} |A_{ij}|$$

- $N =' fro'$ - фробениусова норма матрицы A

$$||A||_F = \mathrm{trace}^{1/2}(A^T A) = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij}^2 \right)^{1/2}$$

1.7.7 Элементарные операции над матрицами

В начале параграфа напомним, что транспонирование матрицы производится оператором $'$ ($A^T = A'$).

- $[N, M] = size(A)$ - размеры матрицы (A - матрица размера $N \times M$;
- $L = length(A)$ - максимальный размер матрицы ($L = \max(size(A))$);
- $y = det(A)$ - определитель матрицы;
- $y = trace(A)$ - след матрицы;
- $y = inv(A)$ - обращение матрицы;
- $y = pinv(A)$ - псевдообращение матрицы ($y = (A^T A)^{-1} A^T$);
- $y = sqrtm(A)$ - корень квадратный из матрицы;
- $y = expm(A)$ - экспонента от матрицы;
- $y = logm(A)$ - логарифм матрицы.
- $v = poly(A)$ - вычисляет вектор коэффициентов характеристического полинома (коэффициенты расположены в векторе в порядке убывания степеней).
- $[V, D] = eig(A)$ - вычисление собственных чисел и собственных векторов матрицы A. Диагональные элементы матрицы D (жордановой канонической формы) являются собственными числами матрицы A, а столбцы матрицы V являются собственными векторами. При наличии единственного выходного параметра $L = eig(A)$ - выход L - вектор-столбец собственных чисел матрицы;
- $[U, S, V] = svd(A)$ - вычисление сингулярного разложения матрицы A. Матрица A представляется в виде произведения трех матриц $A = USV^T$. Здесь матрицы U и V - ортогональные матрицы, а S - диагональная матрица сингулярных чисел

  $$\sigma_i(A) = \lambda_i^{1/2} \left(A^T A\right).$$

  При обращении $S = svd(A)$ выдается только матрица сингулярных чисел S. [3)]
- $y = rank(A)$ - ранг матрицы;

Здесь следует уточнить некоторые подробности о том, что в MATLAB'е понимается под рангом. MATLAB оперирует с численными оценками и вычислительными операциями. В соответствии с этим при достаточно малых, хотя и ненулевых значениях детерминанта численное обращение матрицы оказывается невозможным. Таким образом при вычислениях вводится обобщенное понятие ранга следующим образом: рангом считается количество сингулярных чисел матрицы, превышающих заданный порог $tol = \max(size(A)) * ||A||_2 * eps$.

С особенностями вычислительных процедур при обращении матриц, вычислении собственных чисел и других операциях связан также ряд важных функций MATLAB'а. С теоретическими основами соответствующих вычислительных алгоритмов можно познакомиться, например, в книгах [14] и [15]. Здесь же мы приведем только функции пакета MATLAB, выполняющие соответствующие операции.

- $c = cond(A)$ - вычисляет число обусловленности матрицы $\mu = ||A^{-1}||_2 * ||A||_2 = \sigma_{\max}(A)/\sigma_{\min}(A)$.
- $[T, B] = balance(A)$ - балансировка матрицы используется для улучшения обусловленности матрицы. Процедура балансировки состоит в том, что ищется диагональная матрица T такая, что матрица $B = T^{-1} * A * T$ имеет примерно равные столбцовую и строчную нормы $||B||_1 \approx ||B||_\infty$. Такая процедура эквивалентна масштабированию переменных решаемой задачи.

  ПРИМЕЧАНИЕ Стандартная программа *eig* вычисления собственных чисел автоматически проводит балансировку матриц. Для отказа от этой процедуры необходимо добавить в обращение второй аргумент – текстовую константу 'nobalance' ($[V, D] = eig(A, 'nobalance')$).

[3)] Присутствуют также функции вычисляющие широко распространенные QR и LU разложения.

- $Z = null(A)$ - вычисление ортонормального базиса нулевого подпространства матрицы. Базис нулевого подпространства образуют столбцы матрицы Z. Обращение $Z = null(A,'r')$ приводит к вычислению "рационального"базиса, когда компонентами векторов являются рациональные числа.

- $Q = orth(A)$ - вычисление ортонормального базиса матрицы. Столбцы матрицы Q образуют то же подпространство, что и столбцы матрицы A, причем $Q^T * Q = I$.

Для ознакомления приведем примеры вычислительно плохо обусловленных матриц. Плохо обусловленной с точки зрения обращения является квадратная матрица Гильберта высокого порядка. Ее элементы вычисляются по формуле $\Gamma_{ij} = 1/(i+j-1)$, $i, j = 1 \dots n$ - номера строк и столбцов, а n - размерность матрицы. В MATLAB'е для формирования этой матрицы можно использовать функцию $a = hilb(n)$, где n - размерность квадратной матрицы a. Рассмотрим простой пример матрицы a Гильберта при $n = 6$ и матрицы b, все компоненты которой равны компонентам матрицы a, кроме одной. Эта компонента b_{16} отличается от компоненты a_{16} на 1%. Рассмотрим следующую последовательность вычислений

```
> a=hilb(6)
a =
```

1.0000	0.5000	0.3333	0.2500	0.2000	0.1667
0.5000	0.3333	0.2500	0.2000	0.1667	0.1429
0.3333	0.2500	0.2000	0.1667	0.1429	0.1250
0.2500	0.2000	0.1667	0.1429	0.1250	0.1111
0.2000	0.1667	0.1429	0.1250	0.1111	0.1000
0.1667	0.1429	0.1250	0.1111	0.1000	0.0909

```
> b=a;
> b(1,6) = b(1,6)*1.01;
> (b ^ (-1)-a^(-1))./a^(-1)
ans =
```

−1.2762	−2.1878	−2.8715	−3.4033	−3.8287	−4.1768
−1.2762	−1.6409	−1.9144	−2.1271	−2.2972	−2.4365
−1.2762	−1.4586	−1.5953	−1.7017	−1.7867	−1.8564
−1.2762	−1.3674	−1.4358	−1.4890	−1.5315	−1.5663
−1.2762	−1.3127	−1.3401	−1.3613	−1.3783	−1.3923
−1.2762	−1.2762	−1.2762	−1.2762	−1.2762	−1.2762

```
> eig(a)
ans =
```

0.0000
0.0000
0.0006
0.0163
0.2424
1.6189

```
> eig(b)
ans =
```

1.6191
0.2420
0.0165
0.0006
0.0000
−0.0000

```
> d=cond(a)
d = 1.4951e+007
> det(a)
ans = 5.3673e-018
```

В приведенном примере относительная ошибка вычисления компонент обратной матрицы составляет более 100 % при ошибке задания одной из компонент матрицы в 1 %. При этом собственные числа обеих матриц на первый взгляд вычисляются вполне приемлемо. Основным признаком возможного неблагополучия в данном примере следует считать большое значение числа обусловленности и малое значение определителя матрицы.

Пример плохой обусловленности при вычислении собственных чисел дает функция *gallery*.[4)]. Рассмотрим пример

> a=gallery(5)

a =

$$\begin{array}{rrrrr} -9 & 11 & -21 & 63 & -252 \\ 70 & -69 & 141 & -421 & 1684 \\ -575 & 575 & -1149 & 3451 & -13801 \\ 3891 & -3891 & 7782 & -23345 & 93365 \\ 1024 & -1024 & 2048 & -6144 & 24572 \end{array}$$

> b=a;
*> b(5,1)=b(5,1)*1.01;*
> [eig(a),eig(b)]
> ans =

$$\begin{array}{rrrr} -0.0408 & & -0.9201 & +50.8135i \\ -0.0119 & +0.0386i & -0.9201 & -50.8135i \\ -0.0119 & -0.0386i & -0.0000 & \\ 0.0323 & +0.0230i & 0.9201 & +0.4053i \\ 0.0323 & -0.0230i & 0.9201 & -0.4053i \end{array}$$

> cond(a)
ans =
2.0253e+018

Собственные числа матриц a и b в этом примере отличаются на два порядка. Признаком такого поведения собственных чисел служит число обусловленности $\sim 10^{18}$.

В заключение этого параграфа заметим, что в пакете MATLAB реализован богатый набор функций осуществляющих приведение матриц к различным каноническим формам, таким как формы Жордана, Фробениуса, Шура, Хессенберга, а также часто используемые функции lu и qr разложений.

2 Программирование в пакете MATLAB. Условные переходы, циклы, переключатели.

В этом параграфе обсудим организацию основных элементов программирования в пакете MATLAB - условные переходы и организацию циклов.

2.1 "Логические"переменные.

В пакете MATLAB заданы 6 логических операций отношений <, >, <=, >=, ==, ~=. Эти операции выполняют поэлементное сравнение двух матриц. Результатом является матрица того же размера, элементы которого равны 1, если соответствующее условие выполняется, и равны 0 в противном случае.

Аналогично определены логические операции

- & – логическое И (AND). Результатом операции $A\&B$ является матрица того же размера, каждый компонент которой равен 1, если оба соответствующих компонента матриц A и B не равны 0, и равен 0 в противном случае (т.е. если хотя бы один из компонент равен 0).
- | – логическое ИЛИ (OR). Результатом операции $A|B$ является матрица того же размера, каждый компонент которой равен 1, если хотя бы один из соответствующих компонент матриц A и B не равны 0, и равен 0 в противном случае (т.е. если оба компонента равны 0).

[4)] Эта функция позволяет вычислять значения различных "стандартных"матриц. Список этих матриц и параметры можно посмотреть с помощью команды *help gallery*

- $\sim$ – логическое НЕ (NOT). Результатом операции $\sim A$ является матрица того же размера, каждый компонент которой равен 1, если соответствующий компонент матрицы A равны 0, и равен 0 в противном случае (т.е. если компонент не равен 0).

Например,

$$
\begin{array}{rcl}
> a & = & [1,2;3,4] \\
a & = & \\
 & & \begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix} \\
> b & = & [4,2;3,1] \\
b & = & \\
 & & \begin{matrix} 4 & 2 \\ 3 & 1 \end{matrix} \\
> c & = & a >= b \\
c & = & \\
 & & \begin{matrix} 0 & 1 \\ 1 & 1 \end{matrix} \\
> d & = & \sim (b <= c)\&a \\
d & = & \\
 & & \begin{matrix} 1 & 1 \\ 1 & 0 \end{matrix}
\end{array}
$$

2.2 Условный оператор.

В пакете MATLAB для программирования условных переходов используется условный оператор, который в общем виде записывается следующим образом:

if Логическая переменная 1,
    Инструкции 1
elseif Логическая переменная 2,
    Инструкции 2
else
    Инструкции 3
end

В качестве логической переменной может быть использовано также логическое выражение, то есть правая часть присваивания описанного в предыдущем пункте.

Под инструкциями здесь понимаются набор команд, которые будут выполняться если все элементы матричного "логического"переменного ненулевые.

В приведенном условном операторе конструкции, начинающиеся с *elseif* и *else*, могут быть опущены.

Для ясности приведем два следующих примера

$$
\begin{array}{lll}
> a & = & [1,0;0,1] \\
a & = & \\
 & & \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \\
> b & = & [1,1;1,1] \\
b & = & \\
 & & \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \\
> d & = & [1,1;1,0] \\
d & = & \\
 & & \begin{matrix} 1 & 1 \\ 1 & 0 \end{matrix} \\
> if & d, & c = a; else \quad c = b; end; c \\
c & = & \\
 & & \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \\
> if & a <= b & , c = a; else \quad c = b; end; c \\
c & = & \\
 & & \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}
\end{array}
$$

Само собой все распространяется обычным образом на случай скалярных величин, и именно такое традиционное обращение оператору сравнения можно встретить в программах

if $a < b$,
    c=a;
elseif $a==b$,
    c=1;
else
    c=b;
end

2.3 Организация циклов.

При программировании в пакете MATLAB возможно использование двух видов циклов

- цикл типа *for-end*
- цикл типа *while*

Первая из этих конструкций - цикл типа *for-end* - используется для организации вычислений с заданным числом повторяющихся циклов. Конструкция такого цикла имеет вид:

for var = Выражение,
    Инструкции
end

В качестве "Выражения"в предлагаемой конструкции можно использовать произвольную матрицу размера $m \times n$. В этом случае "Инструкции"тела цикла будут повторяться n раз, причем параметр *var* принимает последовательно векторные значения совпадающие со столбцами матрицы. Это хорошо видно из следующего примера

$$
\begin{array}{rcl}
> a & = & [1,2,3;4,5,6] \\
a & = & \\
 & & \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}
\end{array}
$$

$$
\begin{array}{rcl}
> for & & i = a; i, end \\
i & = & \\
 & & 1 \\
 & & 4 \\
 & & \\
i & = & \\
 & & 2 \\
 & & 5 \\
 & & \\
i & = & \\
 & & 3 \\
 & & 6
\end{array}
$$

Однако чаще всего "выражение"записывается в более традиционном виде и обращение к циклу *for-end* в программах имеет вид

for $i = 1 : 3$,
  for $j = 1 : 3$,
    a(i,j)=i+j;
  end
end

В результате выполнения *script*-файла с указанным набором операторов будет получена матрица

$$
\begin{array}{ccc} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{array} \tag{3}
$$

Второй тип циклов позволяет выполнять инструкции в теле цикла до тех пор пока выполняется заданное условие. Конструкция такого цикла имеет вид:

while Логическая переменная,
    Инструкции
end

В этом выражении "Логическая переменная"имеет тот же смысл, что и для оператора if в параграфе 2.2 и вычисления продолжаются до тех пор, пока в матрице использованной в качестве "Логической переменной"нет нулевых компонентов. Так же как и в 2.2 в конструкции *while* общепринято использовать условный оператор из параграфа 2.1. В связи с этим приведем традиционный и нетрадиционный примеры использования *while*.

Традиционный пример:

$p = 1; k = 0;$ *while* $p < 10$, $p = p * 2; k = k + 1;$ *end; k*

вычисляет минимальное значение показателя степени k такое, что $2^k >= 10$. Очевидно, что по окончании работы цикла $k = 4$ и $p = 16$.

Менее традиционным следует считать пример вида

a=[2.5,3;4,5]; while a, a=a-1; end

В этом примере заданная матрица a преобразуется следующим способом: все элементы ее уменьшаются на каждом шаге на 1, до тех пор, пока одно из них не станет равным 0. Таким образом процедура завершается если хотя бы одна из компонент матрицы a - положительное целое число - обратится в 0. Если это не так, то произайдет зацикливание программы.

Для защиты от такой ситуации можно использовать оператор *break*, который прекращает действие последнего по вложению цикла *for-end* или *while* внутри которого расположен оператор *break*. Предыдущий пример прерванный таким оператором переписанный в файл примет вид

a =[2.5,3;4,5];

while a,
$a = a - 1$;
if $a < 0$, *break*
end
end

2.4 Переключатель *switch-case-otherwise-end*.

Конструкция переключателя *switch* используется для множественного выбора (или ветвления). Она определена в версиях MATLAB'а старше 5-ой и в общем случае имеет вид:

switch switch-выражение,
case case-Выражение ,
Инструкции 1
case { case-Выражение 1, case-Выражение 2, case-Выражение 3, ... } ,
Инструкции 2
otherwise
Инструкции 3
end

В качестве switch-выражения используется любой из описанных ранее объектов MATLAB'а. Если текущее значение этого выражения совпадает с одним из case-Выражений выполняются операторы из соответствующих "Инструкций". Если ни одно из case-Выражений не подошло, выполняются "Инструкции"следующие за оператором *otherwise*.

В качестве примера рассмотрим сценарий asw.m

```
switch var
case {1,2,12,'January','February','December'}
    disp('Winter')
case {3,4,5,'March','April','May'}
    disp('Spring')
case {6,7,8,'June','July','August'}
    disp('Summer')
case {9,10,11,'September','October','November'}
    disp('Autumn')
otherwise
    disp('It is not a month.')
end
```

Обращение к этому сценарию приводит к следующим результатам

```
> var = 3;
> asw
Spring
> var = 'May';
> asw
Spring
> var = 10;
> asw
Autumn
> var = 'Autumn';
> asw
It is not a month.
```

3 Специальные возможности при обращении к функциям

.

Подробное рассмотрение стандартных функций пакета MATLAB позволяет сделать вывод о реализации дополнительных нетривиальных возможностей при создании функций пакета, таких как выполнение различных вычислений в зависимости от числа и типа аргументов и выходных параметров функции, передача функции имени вызываемой функции, передача произвольного числа аргументов этой воспомогательной функции и т.д.

Понятно, что выполнение различных вычислений в зависимости от числа аргументов и выходных параметров функции, несложно организовать, используя параметры *nargin* и *nargout*, уже упоминавшиеся раньше, и операторы *if* или *switch*. Также можно проверить размерность и тип аргументов.

Некоторые особенности MATLAB'а, которые будут использованы в функциях, описанных в дальнейшем рассмотрим в этой главе. Для того, чтобы пользоваться функциями, описанными в последующих главах, читать эту главу необязательно, но её содержание помогает понять, как устроены те или иные функции и позволит более аккуратно, писать собственные функции с аналогичными свойствами.

3.1 Как обращаться к функции по имени.

В ряде случаев (например при решении нелинейных уравнений или численном интегрировании обыкновенных дифференциальных уравнений) необходимо передать функции в качестве аргумента имя некоторой другой функции (например, имя функции, вычисления правых либо левых частей уравнений), которая будет вызвана по этому имени. Имя функции или переменной в таком случае хранится в виде символьных одномерных массивов (векторов). Такой массив можно передать функции в качестве формального параметра.

Выполнение соответствующей команды или вызов функции позволяют сделать функции *eval* и *feval*.

Функция *eval* позволяет выполнить команду системы MATLAB. Обращение к ней имеет вид

eval(command)

Текстовая переменная *command* здесь содержит текст команды. В качестве примера приведем отрывок программы, в котором создаются 5 матриц Уилкинсона размерности от 5 до 10, имена которых устанавливаются *M5, M6, M7, M8, M9, M10* соответственно.

```
for k=5:10,

eval(['M',num2str(k),'=wilkinson(',num2str(k),');'])
end
```

Функция *feval* позволяет по имени вызвать функцию MATLAB'а. Обращение к ней имеет вид

[outpars]=feval(FunName,inpar1,inpar2,...)

Текстовая переменная *FunName* здесь содержит имя вызываемой функции, а аргументы *inpar1,inpar2,...* – содержат аргументы вызываемой функции. Через *[outpars]* обозначен список результатов выполнения вызываемой функции.

Например, при обращении

v=feval(fname,0,10,50);

при *fname = 'linespace'* формируется вектор, 50 компонент которого равномерно распределены на сегменте [0,10], а при *fname = 'logspace'* эти компоненты будут иметь логарифмическое распределение.

3.2 Несколько замечаний о многомерных массивах.

Наряду с традиционными двумерными массивами - матрицами, в пакете MATLAB, начиная с 5-й версии введены многомерные таблицы. С их помощью в MATLAB'е отображаются в тензорные величины. Здесь мы уделяем внимание таким массивам в связи с тем, что они используются при представлении результатов некоторыми функциями моделирования и исследования систем управления.

Обращение к элементам многомерного массива имеет традиционный вид $a(i1, i2, i3, \ldots, in)$. Ввести многомерный массив можно увеличивая число размерностей матрицы. Рассмотрим, например последовательность команд

```
» a=[1 2; 3 4]
a =
```

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

```
» a(:,:,2)=3
a(:,:,1) =
```

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

a(:,:,2) =

$$\begin{matrix} 3 & 3 \\ 3 & 3 \end{matrix}$$

Создают многомерный массив также и функции *ones*, *zeros*, *rand*, *randn*. При обращении вида

$$a = ones(n1, n2, \ldots, nn)$$

эти функции возвращают массив, размерность которого равна числу аргументов функции,причем $n1, n2, \ldots, nn$ "длины"соответствующих размерностей. Указанные функции создают многомерные массивы все компоненты которых принимают соответственно значения: для *ones* – единица; для *zeros* – ноль; для *rand* – случайное число равномерно распределенное на интервале $(0, 1)$; для *randn* – нормально распределенное случайное число с нулевым средним и единичной дисперсией.

Перечислим здесь основные функции, позволяющие работать с многомерными массивами

- функция
$$n = ndims(a)$$
позволяет определить число размерностей многомерного массива a.

- функция
$$n = size(a)$$
позволяет определить размер многомерного массива a по каждой размерности.

- Функция $B = cat(dim, A1, A2, A3, A4, ...)$ позволяет объединить массивы $A1, A2, A3, A4, ...$ в один вдоль размерности dim.Рассмотрим простой пример

  » A1=ones(2);

  » A2=eye(2);

  » B1=cat(1,A1,A2)

  B1 =

$$\begin{matrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{matrix}$$

  » B2=cat(2,A1,A2)

  B2 =

$$\begin{matrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{matrix}$$

  » B3=cat(3,A1,A2)

  B3(:,:,1) =

$$\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$$

  B3(:,:,2) =

$$\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

  Здесь видно, что в первом случае действие оказалось эквивалентно команде $B1 = [A1; A2]$, во втором случае $B2 = [A1, A2]$, а в третьем функция создала трехмерный массив $B3$.

- Функция $B = permute(A, order)$ позволяет переставлять местами размерности массива в порядке, устанавливаемом целочисленным вектором перестановок $order$. Например обращение $C3 = permute(B3, [213])$ для массива $B3$ из предыдущего примера приведет к транспонированию матриц $B3(:,:,1)$ и $B3(:,:,2)$.

- Функция $C = ipermute(B, order)$ обратная по отношению к функции $permute$.

- Функция $B = shiftdim(B, n)$ производит сдвиг размерностей влево, на величину n. При $n \leq 1$ такое действие эквивалентно циклической перестановке размерностей и увеличения их числа не происходит. При $n < 0$ происходит сдвиг размерностей вправо. При этом число размерностей увеличивается на $|n|$ и младшие размерности заполняются удиницами.

- Функция $C = squeese(B)$ позволяет удалить размерности длиной 1. Дело в том, что для MATLAB'а обращение $B3(:,:,1)$ в использованном выше примере описывает трехмерный массив, в то время, как на самом деле это матрица и размерность ее на 1 меньше. Такой объект не удается использовать при обращении к стандартным матричным операторам и функциям. Преобразование этого объекта, понижающее число размерностей и производит функция $squeese$.

3.3 Ячейки и операции над ними.

Массивы ячеек используются в MATLAB'е для хранения и передачи данных разных типов. Создать массив ячеек можно двумя способами: с использованием функции *cell* или заключением объектов в фигурные скобки. Например, если в рабочую область памяти введена матрица a размера 2×2, следующая команда вводит массив ячеек размера 2×3

»b={min(a) max(a) 3; 5 1 'text' }

b =

[1 × 2*double*]	[1 × 2*double*]	[3]
[5]	[1]	*'text'*

Обращение $b = cell(N, M)$ создает пустой массив ячеек размера $N \times M$, а при отсутствии аргумента M массив размера $N \times N$. Для обращения к элементам массива используются фигурные скобки, причем на такие обращения распространяются все основные синтаксические правила для работы с традиционными матрицами

» b{2,3}

ans =

text

» b{:,1}

ans =

1 2

ans =

5

Многомерный массив ячеек заполняется по правилам, аналогичным правилам заполнения многомерных массивов. При этом для массивов ячеек определены функции $squeeze$ и $shiftdim$.

Отметим, что в массиве ячеек хранятся копии объектов и изменение самих объектов не изменяет содержимого массива ячеек и наоборот. Другими словами массив ячеек не следует путать с массивом адресов.

В заключение приведем несколько функций и команд для работы с массивом ячеек.

- $celldisp(C)$ – выводит на экран содержимое массива ячеек C.

- $cellplot(C)$ – выводит на экран содержимое массива ячеек C в виде графического окна.

- $C = cellstr(s)$ – преобразует массив строк s в массив символьных ячеек C.

- $k = iscell(C)$ – возвращает логическое $TRUE(1)$, если C – массив ячеек, и $FALSE(0)$ в противном случае.

- $C = num2cell(A)$ – преобразует массив A в массив ячеек, размещая каждый элемент массива в отдельной ячейке.

С понятием массивов ячеек связана возможность описания списков входных и выходных аргументов переменной длины. Для этого используются переменные *varargin* и *varargout*.

Переменная *varargin* позволяет объединить произвольное количество входных переменных. Она представляет собой одномерный массив ячеек, который содержит аргументы вызываемой функции. Эта переменная должна завершать список входных аргументов функции.

Аналогично переменная *varargout* позволяет объединить произвольное количество выходных переменных. Она представляет собой одномерный массив ячеек, который содержит аргументы выхода вызываемой функции. Эта переменная должна завершать список выходных аргументов функции.

Приведем пример описания такой функции

```
function [z,varargout]=my1fun(a,b,varargin)
  ...
  varargout=my2fun(b,varargin);
  ...
```

Примером использования этих переменных служит обращение к функциям типа *ode*, которые будут обсуждаться в параграфе 5.

3.4 Глобальные переменные

В предыдущем параграфе описан способ передачи произвольного числа переменных воспомогательной функции. Этот способ во многих случаях не является наиболее эффективным [12]. Более эффективным и быстрым является способ передачи переменных с использованием конструкции *global*.

Команда

global имя1 имя2 ...

определяет переменные с именами *имя1, имя2, ...* как "глобальные". Если несколько функций и сценарий объявляют некоторую переменную глобальной, то все они используют одну и ту же копию этой переменной. Изменение ее значения в одной из функций приведет к тому, что при последующем обращении ко всем остальным функциям указанная переменная принимает измененное значение. Это же измененное значение принимает переменная с этим именем в рабочей области (та которую можно посмотреть в командном окне).

Приведем пример использования глобальных переменных

```
function z=my1fun(a,b)
global c
c=b;
d=my2fun(a);
disp ([c,d])
  ...
  function d=my2fun(b);
d=my3fun(b);
  ...
  function d=my3fun(b);
global c
d=b+c;
c=5;
```

Обращение к этой функции из командного окна приводит к результату

```
» my1fun(3,4)
     5     7
```

При этом в командном окне и в функции $my2fun$ переменная c либо не определена, либо не меняет своего значения (если оно было задано).

4 Графические функции пакета MATLAB.

4.1 Двумерная графика.

4.1.1 Рисование графиков в декартовых координатах.

Основной функцией рисования графиков в пакете MATLAB является функция *plot*. Обращения к этой функции имеют вид

- *plot*(x, y) строит график, откладывая по оси абсцисс значения компонент вектора x и по оси ординат значения компонент вектора y, имеющего ту же длину, что и вектор x. В дальнейшем во всех подобных случаях будем говорить о построении графика y от x. Построение графика заключается в нанесении на координатную плоскость последовательных точек и соединении их прямыми. Соответственно такой способ построения не связан ограничениями при отображении неоднозначных функций.

- *plot*(y) строит аналогичный график, откладывая значения компонент вектора y по оси ординат, а по оси абсцисс откладывает номер соответствующей компоненты в векторе.

  Для обоих вариантов обращения к функции *plot* для заданной матрицы y размера $n \times m$ на одной координатной плоскости будет построено m графиков (по одному для каждого столбца матрицы y). Из первых семи графиков каждый новый будет иметь свой цвет. Далее цвета циклически повторяются.

  Например, последовательное выполнение команд

  t=0:0.05:pi;

  plot(t,[sin(t);cos(t)]);

  Эти действия вызовут открытие графического окна MATLAB'а с изображением, приведенным на рис. 1

Рис. 1: Результат действия функции *plot*

Двумерным может быть также и массив x. Если при этом массив y представляет собой вектор, то строятся графики, на которых по оси абсцисс откладываются значения компонент векторов - столбцов матрицы x, а по оси ординат значения компонент вектора y. Если x и y двумерные массивы одной размерности, строятся зависимости столбцов матрицы y от столбцов матрицы x.

- *plot*(x, y, s).

  Это обращение позволяет построить график функции, указав в строковой константе s цвет и способ отображения линии, а также вид узловых точек. Как и прежде вектора (или матрицы) x и y задают значения абсцисс и ординат точек на графике, а строковая константа s может содержать по одному символу из трех наборов приведенных в таблице

	Цвет		Узловая точка		Вид линии
y	желтый (yellow)	.	точка (•)	-	сплошная
m	фиолетовый (magenta)	o	окружность (◯)	:	пунктир
c	голубой (cyan)	x	крест (×)	-.	штрих-пунктир
r	красный (red)	+	плюс (+)	–	штриховая
g	зеленый (green)	*	звездочка (∗)		
b	синий (blue)	s	квадрат		
w	белый (white)	d	ромб (⋄)		
k	черный (black)	v	набла (▽)		
		∧	треугольник (△)		
		<	треугольник (левый) (◁)		
		>	треугольник (правый) (▷)		
		p	пятиконечная звезда (⋆)		
		h	шестиконечная звезда ()		

Приведенные выше способы обращения к функции *plot* имеют существенный недостаток, так как не позволяют строить в одних координатных осях несколько графиков у которых вектора абсцисс и ординат имеют различную длину. Эта трудность преодолевается следующими способами.

Первый из них – обращение

plot$(x1, y1, s1, x2, y2, s2, x3, y3, s3, \dots)$.

В этом случае в одних и тех же координатных осях будут построены графики $y1$ от $x1$ в соответствии со строковой константой $s1$, графики $y2$ от $x2$ в соответствии со строковой константой $s2$ и т.д. При этом конечно же должно соблюдаться соответствие размеров внутри пар массивов $x1$ и $y1$, $x2$ и $y2$ и т.д., но число точек в разных парах могут и не совпадать. В качестве примера приведем следующую последовательность команд

```
t=0:0.05:pi;
t1=0:0.05:2*pi;
plot(t,sin(t),'b-',t1,cos(t1),'r:');
```

В результате в открытом графическом окне MATLAB'а будет построено изображение, приведенное на рис. 2

Другой способ построения нескольких различных графиков связан с поэтапным построением графиков в режиме сохранения содержимого графического окна. Как читатель уже заметил, при построении очередного графика MATLAB по умолчанию полностью перерисовывает изображение в графическом окне. Это значит, что режим сохранения координатных осей. Для его включения служит оператор *hold*.

- Обращение *hold* $\{on/off\}$ включает или отключает режим сохранения координатных осей.
- Обращение *hold* изменяет текущее состояние режима сохранения координатных осей на противоположное.
- Наконец функция $k = ishold$ выдает информацию о текущем состоянии режима: $k = 1$ при включенном режиме, и $k = 0$ в противном случае.

Изображение на рис. 2 можно, например, получить с использованием следующей последовательности команд

```
t=0:0.05:pi;
hold on
plot(t,sin(t),'b-')
t=0:0.05:2*pi;
plot(t,cos(t),'r:');
hold off
```

Заключительный оператор *hold* отключает режим сохранения содержимого графического окна, и использован для восстановления первоначального состояния принятого по умолчанию.

В заключение отметим, что при обращении

```
comet(x,y,p)
```

Рис. 2: Еще один пример действия функции *plot*

функция строит график зависимости y от x поточечно, показывая во времени последовательность рисования точек графика. Пояснить, как работает эта функция достаточно тяжело, так как для этого надо описывать процесс, разворачивающийся во времени. Проще предложить читателю выполнить команды

t=0:0.0001:pi;

*comet(sin(2*t),cos(t))*

и увидеть самому, как выглядит результат. Необязательный параметр p управляет длиной "хвоста".

4.1.2 Графики в других системах координат.

В теории управления, акустике и ряде других областей часто используются логарифмические оси, когда вдоль оси откладываются значения не самой величины, а ее логарифма. Графики в таких осях позволяют строить следующие функции:

- *semilogx*(x, y) - строит график с логарифмическим масштабом оси абсцисс;
- *semilogy*(x, y) - строит график с логарифмическим масштабом оси ординат;
- *loglog*(x, y) - строит график с логарифмическим масштабом по обеим осям.

Все эти функции допускают такие же формы обращения, как и описанная ранее функция *plot*.

Аналогично построение графиков в полярных координатах, задаваемых углом *phi* и радиусом *rho*, осуществляет функция

polar(phi,rho,s)

В качестве примера приведем команды, позволяющие построить кривую, называемую улиткой Паскаля. Ее уравнение в полярных координатах имеет вид $\rho = a\cos\varphi + \ell$. При $a = 0.5$ и $\ell = 0.15$

*phi=0:0.05:2*pi; polar(phi,0.5*cos(phi)+0.15)*

Результат выполнения этих команд приведен на рис. 3.

4.1.3 Сетка, надписи и пояснения на графиках.

Все предшествующие графики, построенные здесь не имели сетки и не содержали никаких надписей, между тем они необходимы при оформлении результатов вычислений.

Показать сетку на графике позволяет команда *grid*, которая последовательно производит включение и отключение сетки. Команда *grid on* добавляет сетку к текущему графику, а команда *grid off* соответственно отключает сетку.

Рис. 3: Результат действия функции *polar*

Сделать надписи на рисунках помогают следующие функции

- $title(stext)$ - выводит в графическое окно символьную константу $stext$ в качестве надрисуночной надписи;
- $xlabel(stext)$ - выводит в графическое окно символьную константу $stext$ в качестве надписи к оси x;
- $ylabel(stext)$ - выводит в графическое окно символьную константу $stext$ в качестве надписи к оси y;
- $text(x, y, stext)$ - выводит в графическое окно символьную константу $stext$ в качестве надписи, левый верхний угол которой помещается в точке с координатами x и y. Если x и y представляют собой вектора, то текст помещается во все позиции задаваемые этими векторами. Координаты текстов задаются в физических координатах;
- $gtext(stext)$ - выводит в графическое окно символьную константу $stext$ в качестве надписи, левый верхний угол которой помещается в точке с координатами указываемыми мышью;
- $legend(stext1, stext2, \dots)$ - создает в графическом окне пояснение к графикам. Соответствие текстовых констант $stext1$, $stext2$, $\dots$ линиям на графике отвечает порядку выведения графиков. Возможны также следующие формы обращения к функции
  - $legend(M)$, где M - массив строк одинаковой длины.
  - $legend\ off$ удаляет пояснения.
  - $legend(stext1, stext2, \dots, n)$ - целый аргумент n устанавливает предельное значение позиций для размещения пояснения. $n = -1$ – пояснение размещается вне области графика. $n = 0$ – пояснение размещается в области графика, если места для этого достаточно.

Продемонстрируем использование этих функций на следующем примере
```
t=0:0.05:2*pi;
plot(t,[sin(t);cos(t);sin(2*t)])
title('Тригонометрические функции')
xlabel('Абсцисса')
ylabel('Ордината')
text(5.5,0.2,'2*pi')
```

legend('sin(t)','cos(t)','sin(2t)',0)
Результаты выполнения этих команд приведены на рис. 4

Рис. 4: Образец надписей к графикам

4.1.4 Несколько графиков в одном окне.

Просмотрев набор различных демонстрационных программ MATLAB'а Вы наверно заметили, что для удобства анализа результатов вычислений часто строятся несколько графиков в одном графическом окне. Проводить такие построения позволяет функция *subplot*. Возможны следующие формы обращения к этой функции

- $hp = subplot(n, m, p)$ и $hp = subplot(nmp)$ - создает разбиение текущего графического окна на $n \times m$ "подокон". При этом n число "подокон"по горизонтали, а m - число "подокон"по вертикали. В "подокне"с номером p эта команда строит оси и делает их активными для последующего обращения к функции *plot*. Номер p указывает на j-е "подокно"в i-ом ряду, где $i = floor(p/n)$, а j остаток от деления ($j = rem(p, n)$). Иначе говоря, для "подокон"принята сквозная нумерация по рядам слева направо и сверху вниз. Возвращает функция *subplot* указатель hp - адрес структуры для осей рвсположенных в p-ом "подокне".

  При вызове этой функции для существующей системы "подокон"в случае, когда величины n и m не изменились, функция устанавливает активным "подокно"с номером p. Если хотя бы одно из чисел m и n изменилось, функция стирает предшествующее содержимое графического окна, и производит разбиение заново.

- $subplot(hp)$ - устанавливает активным "подокно"с указателем (адресом) hp.

- *subplot('position',[left bottom width height])* - позволяет создать "подокно"в области графического окна, указываемой программно. В этом обращении *'position'*- ключевое слово, сообщающее функции о том, что далее будет задана прямоугольная область для размещения осей "подокна", а $left\ bottom\ width\ height$ аргументы задающие положение левого нижнего угла "подокна", а также его ширину и высоту в долях от размеров графического окна.

Продемонстрируем соответствующие возможности на следующем примере
*t=0:0.05:4*pi;*
h1=subplot(221);
*plot(cos(t).*exp(-0.1*t),sin(t).*exp(-0.1*t))*
h2=subplot(222);

*polar(t,exp(-0.1*t))*
h3=subplot('position',[0.1,0.1,0.8,0.35])
*plot(t,sin(t).*exp(-0.1*t),'-',t,cos(t).*exp(-0.1*t),'–')*
xlabel('Время, с. Угол, рад.')
ylabel('Координаты, м.') >
subplot(h2)
title('Траектория в полярных координатах')
subplot(2,2,1)
xlabel('Дальность, м.')
ylabel('Высота, м.')
title('Траектория в декартовых координатах. ')
В итоге в графическом окне появится изображение, приведенное на рис. 5

Рис. 5: Разбиение графического окна на "подокна"

Конечно же в этом примере надрисуночные заголовки и подписи под координатными осями можно было бы сделать сразу после построения графиков функцией *plot*. Возвращение к соответствующим осям производится специально для демонстрации возвращения с помощью функции *subplot*.

Примечание. В "подокне", вообще говоря, можно строить любое изображение, в том числе и трехмерное. Просто речь о построении трехмерных изображений впереди.

4.1.5 Графические окна и управление ими.

Конечно же MATLAB допускает одновременное создание нескольких графических окон и рисование графиков в них.

- Для создания нового графического окна используем функцию $figure$. При отсутствии формальных параметров обращение $h = figure$ создает новое графическое окно и делает его активным. Теперь все вызванные функции рисования будут строить изображение в этом графическом окне. Обращение $h = figure(N)$ для любого целого числа N создает новое графическое окно с номером N, если такого окна не существовало, и делает его активным. В обоих случаях функция возвращает указатель на указанное графическое окно. Если h указатель на графическое окно обращение $figure(h)$ делает это графическое окно активным.

- Узнать указатель на активное графическое окно позволяет обращение $h = gcf$.
- Очистить активное графическое окно позволяет оператор clf.
- Закрыть активное графическое окно позволяет оператор *close*. Обращение $close(h)$ закрывает графическое окно с указателем h, а $close(N)$ графическое окно с номером N. Обращение *closeall* закрывает все открытые графические окна.

Важную возможность рассмотреть детали на графиках в графическом окне дает оператор *zoom*. Обращение к нему может принимать следующие формы

- *zoom* - переключает состояние режима изменения масштаба графика.
- *zoom on* - включает режим изменения масштаба графика.
- *zoom off* - выключает режим изменения масштаба графика.
- *zoom xon* и *zoom yon*- включает режим изменения масштаба графика только по одной из осей x или y соответственно.
- *zoom out* - восстанавливает исходный масштаб графиков в активном окне.
- *zoom reset* - устанавливает текущий масштаб графиков в активном окне в качестве исходного (неуменьшаемого начала отсчета).
- $zoom(F)$ - устанавливает масштаб графиков в активном окне в соответствии с параметром F, другими словами увеличивает изображение в F раз ($F > 1$).

Масштабирование графика при основных формах обращения к оператору *zoom* производится с помощью мыши. Для этого курсор мыши надо подвести к центру интересующей пользователя области рисунка. Если режим *zoom* включен, то нажатие левой клавиши увеличивает масштаб вдвое, а нажатие правой клавиши - уменьшает вдвое. При нажатой левой клавише мыши можно выделить пунктирным прямоугольником нужный участок графика - при отпускании клавиши этот участок появится в увеличенном виде и в том масштабе, который соответствует размерам выделяющего прямоугольника.

4.1.6 Диаграммы, гистограммы, вектора.

Специальные возможности предоставляет MATLAB для построения различного вида диаграмм и гистограмм и прочих стандартных видов представления информации.

- **Столбцовая диаграмма.** Такая диаграмма строится с использованием функции *bar*.
  - При обращении $bar(Y)$ функция строит n групп столбчатых диаграмм по m столбцов в группе (здесь $n \times m$ размер матрицы Y). По оси абсцисс откладывается номер столбца, в котором содержится соответствующее значение.
  - При обращении $bar(x, Y)$ функция по матрице Y строит также строит группы диаграмм, но позиции столбцов определяются вектором x.
  - $bar(x, Y, 'stack')$ - строит единственную группу диаграмм, в которой для каждой строки из Y строится один столбец диаграммы.
  - $bar(\ldots, s)$ - при таком обращении в строковой константе s задается цвет диаграммы в соответствии с таблицей приведенной при описании функции *plot*.
- **Горизонтальная столбцовая диаграмма.** Эта диаграмма строится с использованием функции *barh* состоит из горизонтальных столбцов и основные формы обращения к этой функции такие же, как и у функции *bar*.
- **Круговые диаграммы.** Обращение $pie(x)$ позволяет построить круговую диаграмму. При обращении $pie(x, explode)$ на круговой диаграмме отделяются отдельные сектора. Номера этих секторов задаются вектором *explode*, длина которого совпадает с длиной вектора x. Ненулевые компоненты вектора *explode* отмечают сектора отделяемые от диаграммы.
- **Гистограммы.** Функция построения гистограмм *hist* имеет несколько форм обращения.

- $N = hist(Y, M)$ - возвращает вектор числа попаданий значений из вектора Y в M интервалов, равномерно разделяющих интервал $[min(Y), max(Y)]$. При отсутствии аргумента M число интервалов равно 10. Если Y матрица размера $n \times m$, то N также является матрицей размера $M \times m$, i-ый столбец которой представляет собой гистограмму i-ой строки матрицы Y.
- $N = hist(Y, x)$ - возвращает вектор числа попаданий значений из вектора Y в интервалы, середины которых заданы компонентами вектора x.
- $[N, X] = hist(Y, M)$ - возвращает кроме вектора числа попаданий N, вектор X содержащий середины интервалов.

- **Круговые гистограммы.** Функция построения круговой гистограммы *rose* имеет формы сходные с функцией *hist*.
- **Ступенчатый график.** $stairs(x, Y, s)$ - представляет зависимость Y от x в виде ступенчатой функции. Формы обращения к ней те же, что и для функции *plot*.
- **График дискретных отсчетов.** $stem(x, Y, s)$ - представляет дискретные значения зависимости Y от x, соединенные вертикальной линией с осью абсцисс. Формы обращения сходны с функцией *plot*.
- **График с информацией о погрешностях.** $errorbar(x, Y, L, U)$ - представляет дискретные значения зависимости Y от x с указанием доверительных интервалов значений функции. Верхние и нижние границы этих интервалов задаются аргументами L и U. При обращении $errorbar(x, Y, V)$ границы доверительных интервалов устанавливаются $L = Y - V$ и $U = Y + V$.
- **График проекций векторов на плоскость.** Для построения векторов на плоскости используется функция $feather$. При обращении $feather(U, V)$ на координатной плоскости будут построены вектора, начала которых находятся в точках с нулевыми ординатами и абсциссами $X0 = 1 : length(U)$, а окончания в точках с координатами $(X0 + U, V)$. Обращение $feather(Z)$ для комплекснозначного вектора Z эквивалентно обращению $feather(real(Z), imag(Z))$.
- **График векторов в полярных координатах.** Для изображения векторов используется также функция *compass*. $compass(U, V)$ строит на плоскости вектора, начала которых находятся в точке с нулевыми координатами, а окончания в точках с координатами (U, V). Обращение $compass(Z)$ для комплекснозначного вектора Z эквивалентно обращению

  $compass(real(Z), imag(Z))$.

Конечно описание данное здесь описанным функциям неполно, и приведенный список служит только "маршрутным листом"для любознательного читателя при знакомстве с соответствующими функциями.

4.2 Трехмерная графика.

4.2.1 Построение кривых в пространстве. Первое знакомство с функцией *plot3*.

Простейшее пространственное изображение, которое мы будем здесь рассматривать, можно получить в результате рисования графика, параметрически заданной трехмерной кривой. Для построения такой кривой используется функция *plot*3, обращение к которой аналогично функции *plot*, рассмотренной ранее в разделе 4.1.1

*plot*3($x1, y1, z1, s$).

Это обращение в основном совпадает с обращением к функции построения графика на плоскости и отличается только добавлением вектора $z1$. Аргумент – строка s может принимать значения, указанные в таблице раздела 4.1.1. Результат действия этой функции рассмотрим на примере, результат действия которого приведен на рис. 6.

```
x=(1:1000)/1000*3*pi;
plot3(x, sin(10*x),x. ^ 2);
```

Заметим здесь, что справедливо обращение и ко всем остальным формам функции *plot*3, повторяющим формы функции *plot*.

Менее удачный пример построения трехмерной кривой приведен на рис. 7. Эту кривую строит сценарий

```
x=(1:1000)/1000*3*pi;
```

Рис. 6: Результат действия оператора *plot*3

*plot*3($sin(25 * x), cos(25 * x), cos(2 * x)$);

Для приведенного примера понять, где начало и конец построенной кривой не представляется возможным. Непонятно также, как строятся нарисованные кривые, так как функция *plot* "строит"изображение в памяти компьютера, и выдает на экран полученный результат. Как построение графика разворачивается во "времени"можно посмотреть, обратившись к функции *comet*3

*comet*3($sin(25 * x), cos(25 * x), cos(2 * x)$);

При этом следует позаботиться о том, чтобы на экране одновременно были видны и командное и текущее графическое окна MATLAB'а. Итоговый результат построения будет таким же, но внимательно отследив происходящее на экране удается понять, что построение приведенной кривой аналогично наматыванию нитки на катушку и далее по виду аргументов прикинуть число витков, число "движений руки"с ниткой вниз-вверх и т.д.

Надписи к изображению и конкретным осям для трехмерных изображений выполняются уже известным способом, описанным в разделе 4.1.3. Для трехмерной графики дополнительно введены функции *zgrid* и *zlabel* аналогичные описанным ранее *xgrid*, *ygrid* и *xlabel*, *ylabel*.

4.2.2 Поверхности в пространстве.

Самый простой метод построения поверхности в пространстве представляет уже знакомая функция *plot*3. Если в качестве аргументов этой функции использовать матрицы одинакового размера, то функция *plot*3 строит набор кривых, совокупность которых образует поверхность. Рассмотрим пример вида

```
X=0.1:0.1:3;
for i=2:30, X(i,:)=X(1,:); end
Y=X';
Z=sin(0.3*X.*Y); Z=1./(sin(0.3*X.*Y)+1);
plot3(X,Y,Z)
title('Plot3 surface illustration')
xlabel('Absciss axis')
ylabel('Ordinate axis')
zlabel('Z-axis')
```

Для приведенного примера несложно построить сетчатую поверхность. Для этого можно воспользоваться симметрией массивов X и Y и использовать обращение *plot*3($X, Y, Z, ' k', Y, X, Z, ' k'$).

В итоге в графическом окне появится изображение представленное на рис. 8. В приведенном примере автор сознательно приведен пример не самого наглядного графика такого вида, хотя в популярных изданиях приводятся более удачные примеры.

Рис. 7: Непонятный результат действия оператора $plot3$.

Для удобного графического представления трехмерных поверхностей, в MATLAB'е предусмотрен обширный набор специальных функций.

- $mesh(X, Y, Z)$ - строит цветную сетчатую поверхность $Z(X, Y)$, причем цвет узлов задается высотой поверхности.
- $mesh(X, Y, Z)$ - строит цветную сетчатую поверхность $Z(X, Y)$ того же типа, что и $mesh$, а на плоскости X, Y проводятся линии уровня поверхности.
- $meshz(X, Y, Z)$ - строит цветную сетчатую поверхность $Z(X, Y)$, того же типа, что и $mesh$, а края поверхности отмечены столбчато. Для иллюстрации работы этой функции лучше использовать поверхность вида $Z1 = sin(0.3 * X. * Y) + 1;$.
- $surf(X, Y, Z)$ - строит цветную ячеистую поверхность $Z(X, Y)$, причем цвет ячеек задается высотой поверхности.
- $surf(X, Y, Z)$ - строит цветную ячеистую поверхность $Z(X, Y)$, того же типа, что и $surf$, а на плоскости X, Y проводятся линии уровня поверхности.
- $surfl(X, Y, Z)$ - строит цветную ячеистую поверхность $Z(X, Y)$, того же типа, что и $surf$, но цвет поверхности имитирует подсветку от дополнительного источника света. При обращении $surfl(X, Y, Z, S)$ - трехмерный вектор $S = [Sx, Sy, Sz]$ задает положение источника света в декартовой системе координат, а $S = [Az, El]$ - в системе сферических координат. Уточним, что для декартовых координат на самом деле здесь речь идет о параллельном световом потоке, направление которого задается вектором, направленным из точки S в начало координат.
- $waterfall(X, Y, Z)$ строит цветную слоеную поверхность $Z(X, Y)$ похожую на $meshz$, однако стиль изображения более напоминает порезанный на вертикальные куски пирог.
- $contour3(X, Y, Z)$ строит цветную слоеную поверхность $Z(X, Y)$ в которой слои отмечены линиями равного уровня, проведенными в оксанометрической проекции. Такое изображение напоминает классический слоеный пирог. В обращении $contour3(X, Y, Z, n)$ аргумент n задает количество уровней.

Для всех вышеперечисленных функций допустимы виды обращений типа

- $surf(x, y, Z)$, где x и y вектора одинаковой длины n, а Z - квадратная матрица размера $n \times n$. В этом случае функция сама строит матрицы X и Y.

Рис. 8: Результат рисования поверхности функцией *plot*3.

- $surf(Z)$ строит поверхность откладывая по осям абсцисс и ординат номера столбцов и строк матрицы Z.

В заключение настоящего подраздела заметим, что построить линии уровня на плоскости xy для данной поверхности позволяет функция *contour*, обращение к которой целиком совпадает с обращением к функции *contour*3. Обе эти функции возвращают матрицу Cs описания контурных линий. Маркировать эти линии для результирующих графиков обеих функций позволяет функция *clabel* в результате обращения $clabel(Cs)$, где матрицу Cs вычисляют и возвращают функции *contour* и *contour*3. В результате обращения $clabel(Cs, v)$ маркируются только высоты, указанные в векторе v. Функции построения линий уровня могут также при обращении $[Cs, H] = contour(X, Y, Z)$ возвращать вектор , содержащий дополнительную информацию о высотах. При обращениях $clabel(Cs, H)$ и $clabel(Cs, H, v)$ надписи линий уровня без дополнительных маркеров будут вставлены в разрывы контурных линий.

Представить характер поверхности помогает также функция *quiver*. В результате обращения $quiver(X, Y, U, V)$ в графическом окне строится изображение градиентного поля. В точках заданных координатами из матриц X и Y изображаются вектора, величины и направления которых задаются матрицами U и V. Часто бывает удобно использовать эту функцию для построения градиентного поля поверхности, заданной матрицей Z следующим образом

$[px, py] = gradient(Z, hx, hy)$;

$quiver(X, Y, px, py)$

В приведенном ранее примере значения шагов hx и hy по осям абсцисс и ординат следует указать равными 0.1.

Наконец заметим, что для построения матриц X и Y по векторам x и y можно использовать специальную функцию, обращение к которой имеет вид

$[X, Y] = meshgrid(x, y)$.

4.2.3 Хочешь посмотреть на изображение с разных сторон?

После построения трехмерной поверхности у пользователя возникает естественное желание, рассмотреть ее с другой стороны. Выбрать точку обзора, отличную от той, которую предоставляет стандарт оксанометрической проекции, позволяет функция *view*. Обращения к ней могут иметь вид

- $view(Cv)$, где Cv - вектор, задающий направление просмотра.

  Для трехмерного вектора $Cv = [Cx, Cy, Cz]$ - Cv - декартовы координаты "точки просмотра". Линия визирования представляет собой луч, проведенный из начала координат и проходящий через точку Cv.

Для двумерного вектора $Cv = [Az, El]$ - величины Az и El задают линию визирования используя углы азимута и возвышения. Так же будет воспринято и обращение $view(Az, El)$.

Изображение, выведенное на экран представляет собой проекцию на плоскость ортогональную линии визирования, проведенную вне границ объекта.

- $view(2)$ устанавливает штатное положение "точки просмотра"для двумерной графики ($Az = 0^o,\ El = 90^o$).
- $view(3)$ устанавливает штатное положение "точки просмотра"для трехмерной графики (оксанометрическая проекция - $Az = -37.5^o,\ El = 30^o$).
- $[Az, El] = view$ - возвращает текущее значение углов азимута и возвышения.
- $T = view$ возвращает текущее значение обобщенной матрицы T преобразований, используемой для получения изображения. Эту матрицу можно получить также обратившись к функции $T = viewmtx(Az, El)$. Обращение $view(T)$ устанавливает положение "точки просмотра", задаваемое матрицей T.

В качестве примера рассмотрим поверхность
*x=(1:100)/100*3*pi-pi;*
y=(1:100)/100-0.5;
[X,Y]=meshgrid(x,y);
*Z=sin(X.*Y);*
mesh(X,Y,Z)
Обращение $view(2)$ дает представление о виде сверху на изображенную поверхность, $view(30, 30)$ позволяет взглянуть на нее "сбоку", а $view(45, -60)$ – снизу. Забавно, что в последнем случае MATLAB строит координатные оси для $Az = 60$???.

Разобраться в ситуации с картинкой в этой ситуации кроме Вашего пространственного воображения позволяет функция *colorbar*, которая добавляет к текущему графику шкалу палитры. При обращении к этой функции без аргументов или обращении $colorbar('vert')$ эта шкала вертикальная. При обращении $colorbar('horiz')$ - она горизонтальная. Шкала палитры изображенная в графическом окне представляет собой отдельный график, нарисованный в "подокне"описанном в подразделе 5. Это "подокно"остается активным по окончании выполнения функции *colorbar*, и это обстоятельство необходимо учитывать при дальнейшем дорисовывании изображений в графическом окне.

4.2.4 Печать, хранение и экспорт изображений.

Напечатать изображение из графического окна на принтере, подсоединенном к компьютеру, на котором работает пользователь, несложно. Для этого достаточно в меню графического окна выбрать позицию "File", а в открывшемся подменю выбрать позицию "Print". Дальнейшие действия также соответствуют стандартным действиям в системе Windows.

Перенести изображение на компьютер с **такой же** версией MATLAB'а удается сохранив его. Для этого необходимо в меню графического окна выбрать позицию "File", а в открывшемся подменю позиции "Save"или "Save as". Различные версии MATLAB'а предлагают различные способы сохранения изображений: версии более ранние чем MATLAB 5.2 используют для сохранения $*.m$ и $*.mat$ файлы, версия MATLAB 5.3 - $*.m$ и $*.fig$ файлы. При открытии этих $*.fig$ файлов с использованием позиции подменю "Open"изображение появляется в текущем графическом окне. Для восстановления изображения из $*.m$ и $*.mat$ следует выполнить соответствующий сценарий из $*.m$ файла. Однако не все версии MATLAB'а нормально читают чужие файлы.

Проблемы возникают в случае, если Вам необходимо включить изображение в отчет. При наборе отчета в редакторе MS Word можно воспользоваться стандартным шаблоном M-book, поставляемым совместно с системой MATLAB. Для использования других редакторов следует экспортировать изображение используя Clipboard системы Windows. Для этого необходимо в меню графического окна выбрать позицию "Edit", а в открывшемся подменю позицию "Copy Figure". В зависимости от установок производится копирование изображения в одном из форматов "Windows Metafile"или "Windows Bitmap". Эти изображения можно восстановить в любом графическом редакторе с использованием пункта подменю "Paste"либо комбинации клавиш Ctrl-v. Для формата "Windows Metafile"при восстановлении изображения в редакторах Paint, Paintbrush, Adobe Photoshop и им подобных наблюдается искажение шрифтов в надписях. При использовании формата "Windows Bitmap"шрифты передаются без искажений. Для установки соответствующего формата необходимо в меню графического окна выбрать позицию "File", а в открывшемся подменю позицию "Preferences...". Во вновь открывшемся окне следует

выбрать страницу с закладкой "Copying Options"и в разделе "Clipboard Format"установить выбранный Вами формат.

5 Численное решение задачи Коши для обыкновенных дифференциальны уравнений

В пакете MATLAB существует несколько возможностей решения задачи Коши для обыкновенных дифференциальных уравнений: стандартные ODE-решатели, специализированные программы исследования моделей управляемых систем, средства имитации аналогового моделирования среды SIMULINK. Изложение средств среды SIMULINK может является предметом отдельного обсуждения (см., например, [10], [5]). Две первые возможности рассмотрим здесь подробнее.

5.1 Стандартные ODE-решатели.

Стандартные ODE-решатели представляют собой пакет программ, состоящий из M-функций: *ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb*, которые в дальнейшем будем обозначать через *solver*
.

Указанные функции реализуют следующие методы решения систем обыкновенных дифференциальных уравнений

- *ode45* – одношаговые явные методы Рунге-Кутта 4 и 5 порядка. Этот классический метод, рекомендуется для начальной пробы решения, обычно дает удовлетворительные результаты.
- *ode23* – одношаговые явные методы Рунге-Кутта 2 и 3 порядка. При умеренных требованиях к точности решения для нежестких систем метод может дать выигрыш в скорости решения.
- *ode113*– многошаговый метод Адамса-Башворта-Мултона переменного порядка. Этот адаптивный метод призван обеспечить высокую точность решения.
- *ode15s* – многошаговый метод переменного порядка (от 1 до 5). Этот адаптивный метод призван обеспечить высокую точность решения для жестких систем.
- *ode23s* – одношаговый модифицированный метод Розенброка 2-го порядка. Призван обеспечить высокую скорость вычислений для жестких систем при низкой точности.
- *ode23t* – метод трапеций с интерполяцией. Метод дает хорошую точность при решении жестких задач, описывающих осцилляторы с почти периодическим выходным сигналом.
- *ode23tb* – неявный метод Рунге-Кутта в начальной стадии решения и метод, использующий формулы обратного дифференцирования 2-го порядка в последующем. При низкой точности для жестких систем этот метод может оказаться эффективней, чем *ode23s*.

Все указанные функции предназначены для численного решения задачи Коши для систем обыкновенных дифференциальных уравнений вида

$$\begin{cases} \dfrac{dy}{dt} &= f(y,t) \\ y(0) &= y_0 \end{cases} \tag{4}$$

где y - вектор длины n.

Решатели *ode15s, ode23s, ode23t, ode23tb* предназначены также для решения задачи Коши для систем обыкновенных дифференциальных уравнений неявного вида

$$\begin{cases} M(t)\dfrac{dy}{dt} &= f(y,t) \\ y(0) &= y_0 \end{cases} \tag{5}$$

где $M(t)$ – матрица массовых коэффициентов. Для функции *ode23s* матрица M не должна зависеть от аргумента t.

Обращение к функциям численного интегрирования может принимать следующие формы:

$$\begin{aligned} [T,Y] &= solver('Fun', tspan, y0) \\ [T,Y] &= solver('Fun', tspan, y0, options) \\ [T,Y] &= solver('Fun', tspan, y0, options, p1, p2, ...) \\ [T,Y,TE,YE,IE] &= solver('Fun', tspan, y0, options, p1, p2, ...) \end{aligned}$$

- Fun - имя файла, в котором помещена М-функция вычисления правых частей системы дифференциальных уравнений. Описание этой функции принимает одну из следующих форм

$$function \quad f = \quad Fun(t, y) \tag{6}$$
$$function \quad f = \quad Fun(t, y, flag) \tag{7}$$
$$function \quad f = \quad Fun(t, y, flag, p1, p2, ...) \tag{8}$$

- $tspan$ – вектор значений моментов времени. Если вектор $tspan$ состоит из двух компонент $[t0, tfin]$, то эти значения воспринимаются как начальное и финальное значение аргумента. Для вектора большей длины $[t0, t1, \ldots, tfin]$, компоненты которого расположены в порядке возрастания или убывания, результаты численного интегрирования выдаются для указанных значений аргумента.

- $y0$ – вектор начальных значений.

- $options$ – аргумент задающий параметры для численного интегрирования. Этот аргумент задается функцией $odeset$, использование которой будет обсуждаться ниже.

- $p1, p2, \ldots$ – произвольные параметры, передаваемые в функцию Fun.

- T – вектор значений аргумента, для которого выдаются значения функции y.

- Y – матрица решений, каждая строка которой соответствует значению аргумента, возвращенному в Y.

- TE, YE, IE – выходные матрицы, в которых выводятся значения аргумента t, функции y и номера события ('event') для работы опции фиксирования событий.

5.2 Простейший пример численного интегрирования.

Прежде чем разбирать дальше различные возможности решения задач численного интегрирования систем обыкновенных дифференциальных уравнений, приведем простой пример, который может послужить образцом для решения 90% задач такого типа.

Будем искать решение уравнения Ван дер Поля

$$\ddot{x} = -x + 0.3 * \dot{x} * (1 - x^2)$$

при начальных условиях

$$x(0) = 1; \qquad \dot{x}(0) = 0$$

Сделаем замену переменных $y_1 = x$ и $y_2 = \dot{x}$. Преобразуем задачу к виду

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = -y_1 + 0.3 * y_2 * (1 - y_1^2) \\ y_1(0) = 1 \\ y_2(0) = 0 \end{cases} \tag{9}$$

Для этой системы создадим функцию вычисления правых частей с описанием типа 6, поместив эту функцию в файл *vanderp.m* в рабочей папке

```
function f=vanderp(t,x)
% function f=vanderp(t,x)
% right part for VanDerPol equation calculation
% dx1/dt=x2
% dx2/dt=-x1+e*x2*(1-x1 ^ 2)
f(1,1)=x(2);
f(2,1)=-x(1)+0.3*x(2)*(1-x(1) ^ 2);
```

Результат получим набрав в командном окне MATLAB'а команду

```
ode45('vanderp',[0,30],[1,0])
```

В результате такого обращения MATLAB откроет графическое окно и будет выводить в него зависимость $x(t)$ по мере численного интегрирования. Результат приведен на рис. 9. Графики различных компонент вектора y выводятся различными цветами. Маркерами отмечены точки, в которых вычислены значения.

Рис. 9: Результат действия функции *ode*45

Чтобы получить численные значения решений используем обращение
[T,Y]=ode45('vanderp',[0,30],[1,0]);
В этом случае изображение на экране не отображается, но в выходные массивы T, Y заполнены. С их использованием можно получить, например, фазовый портрет для найденного решения. Для этого построим зависимость y_2 от y_1 с помощью команды
plot(Y(:,1),Y(:,2))
Результат построения приведен на рис. 10.

Рис. 10: Фазовый портрет решений уравнения Ван дер Поля.

Любопытствующему читателю советую также посмотреть зависимость длины шага интегрирования от номера шага. Это несложно сделать, выполнив команду $plot(diff(T))$. Полученный Вами график наглядно продемонстрирует переменность шага в ходе численного интегрирования. Подобный автоматизм в выборе шага может оказать плохую услугу при интегрировании систем с разрывной правой частью. Автоматический выбор шага при частом "переключении"приводит "точные"методы к непомерному увеличению времени вычислений.

5.3 Опции решателя

В этом параграфе мы обсудим некоторые возможности использования аргумента *options*. С помощью этого массива устанавливаются различные специальные параметры *ode*-решателей. Перечислим те из них, которые представляются наиболее важными.

1. $RelTol$ – относительная погрешность. Скалярная величина по умолчанию принимается равной 10^{-3}. Ошибка на каждом шаге интегрирования оценивается величиной $e_i \leq \max(RelTol * abs(y_i), AbsTol_i)$.

2. $AbsTol$ – вектор абсолютных погрешностей (по умолчанию все его компоненты равны 10^{-6}).

3. *Refine* – положительное целое число – фактор качества, повышающий количество выводимых значений аргумента. Используется для сглаживания вывода. По умолчанию решатели кроме *ode45* используют значение 1, а *ode45* – 4. Опция *Refine* не используется, если длина вектора *tspan* больше 2.

4. *OutputFcn* - текстовая константа '*AFun*', содержащая имя дополнительной функции вывода. Решатель будет вызывать указанную функцию по окончании каждого шага интегрирования. Обычно по умолчанию дополнительная функция вывода отсутствует, но при отсутствии выходных параметров она принимает значение '*odeplot*'. Ее работу мы и видели в первом из приводимых простых примеров. Для того, чтобы на экран компьютера выводился график при наличии выходных параметров необходимо установить значение опции *OutputFcn* равное '*odeplot*'. При необходимости можно использовать также функции

   - *odephas*2 – изображает двумерный фазовый портрет;
   - *odephas*3 – изображает трехмерный фазовый портрет;
   - *odeprint* – выводит в командное окно промежуточные результаты численного интегрирования.

   Пользователь может самостоятельно написать функцию такого типа. Единственным требованием к такой функции является то, что ее описание должно иметь вид

   $$function \quad AFun(t, z)$$

   где z – вектор y переменных состояния решаемой системы уравнений или вектор меньшей длины, который состоит из компонент вектора y порядок которых задается опцией *OutputSel*.

5. *OutputSel* - вектор номеров компонент вектора состояния y, используемых решателем при обращении к функции дополнительного вывода *OutputFcn*. По умолчанию используются все компоненты вектора состояния.

6. *Stats* - отображает вычислительную статистику по окончании работы решателя (принимает значения $\{on\}/off$.

7. *Jacobian*- признак возможности вычисления Якобиана $\{on\}/off$. Для использования этой опции необходимо, чтобы обращение к функции Fun вычисления правых частей системы уравнений имело вид (7). При этом при значении переменной $flag$ равном $'Jacobian'$ указанная программа должна вычислять матрицу Якоби, компоненты которой имеют вид $\partial f_i/\partial x_j$. Матрица Якоби является возвращаемым парамтром функции Fun. Использование этой опции может значительно снизить число шагов и время численного интегрирования.

8. *JConstant* – эта опция должна принимать значение $'on'$ в случае, если матрица Якоби для системы дифференциальных уравнений является постоянной.

9. *Events* - признак использования обработчика событий $\{on\}/off$. Для использования этой опции необходимо, чтобы обращение к функции Fun вычисления правых частей системы уравнений имело вид (7). При значении переменной $flag$ равном $'Events'$ указанная программа должна возвращать следующие величины

   - *value* - вектор величин, для которых должно быть зафиксирован момент изменения знака,
   - *isterminal* – логический вектор, компоненты которого принимают значения 0 или 1 в зависимости от того, следует ли останавливать численное интегрирование при изменении знака соответствующей компоненты вектора *value*.
   - *direction* – вектор той же длины, указывающий направление смены знака, при котором соответствующее событие будет зафиксировано. Компоненты этого вектора принимают значения 1 и -1 в зависимости от направления смены знака. Значение соответствующей компоненты равное 0 означает, что направление изменения знака несущественно.

10. *Mass* - признак использования вычисляемой матрицы "масс"$M(t)$ (инерционных коэффициентов при производных в левой части уравнений) $[on|\{off\}]$. Для использования этой опции необходимо, чтобы обращение к функции Fun вычисления правых частей системы уравнений имело вид (7). При значении переменной $flag$ равном $'Mass'$ указанная программа должна возвращать матрицу инерционных коэффициентов - "масс". Этот параметр обрабатывают функции *ode*15*s*, *ode*23*s*, *ode*23*t*, *ode*23*tb*. (Заметим, что переменную матрицу масс обрабатывает только функция *ode*15*s*).

11. *MassConstant* -признак использования постоянной матрицы "масс"(инерционных коэффициентов при производных в левой части уравнений) $[on|\{off\}]$. Этот параметр должен принимать значение *'on'*, если при обращении к функции Fun с переменной $flag$ равной $'Mass'$. Это значение информирует о том, что функция Fun возвращает постоянную матрицу инерционных коэффициентов ("масс").

12. *MaxStep* - положительный скаляр - максимальное значение шага интегрирования (по умолчанию принимает значение десятой части интервала времени tspan).

13. *InitialStep* - положительный скаляр - начальное значение шага интегрирования. По умолчанию программы определяют начальный размер шага автоматически.

14. *MaxOrder* - максимальный порядок для интегрирующей функции $ode15s$ и принимает одно из значений [1 | 2 | 3 | 4 | 5].

5.4 Пример использования опций решателя.

Рассмотрим в качестве примера систему, изображенную на рис. 11. Эта система состоит из груза массой M, перемещающихся по вертикальным направляющим. В нижней части груза укреплены упругий и вязкий элементы. Эти элементы вторым концом могут касаться опорной поверхности в точке A, однако возможно движение с отрывом.

Рис. 11: Прыгающая механическая система.

Уравнения движения такой системы запишем в виде

$$\begin{cases} \frac{dx_1}{dt} = x_2 \\ \frac{dx_2}{dt} = \begin{cases} -10 - 10x_1 - 0.1x_2 & \text{при} \quad x_1 \le 0 \\ -10 & \text{при} \quad x_1 > 0 \end{cases} \\ x_1(0) = 30 \\ x_2(0) = 0 \end{cases}$$

Ниже приведено содержимое М-файла *jump.m*, осуществляющего численное интегрирование представленной системы уравнений

```
function varargout=jump(t,x,flag)
switch flag
case ''      % Return dy/dt = f(t,y).
  varargout {1} = ft(t,x);
```

```
case 'init'       % Return default [tspan,y0,options].
  [varargout{1:3}] = init;
case 'events'       % Return [value,isterminal,direction].
     [varargout{1:3}] = events(t,x);
case 'demo'
     ode45('jump');
otherwise
     error(['Unknown flag ''' flag '''.']); end
%-----------------------------
function [tspan,y0,options] = init
tspan = [0; 50];
y0 = [30; 0];
options = odeset('OutputFcn', 'odephas2', 'OutputSel',[2,1],'Events','on')
%-----------------------------
function [value,isterminal,direction] = events(t,y)
value = y(1);
isterminal = 0;
direction = -1;
%-----------------------------
function F=ft(t,x)
F(1,1)=x(2);
if x(1) <= 0,
     F(2,1)=-10-10*x(1)-0.1*x(2);
else
     F(2,1)=-10+0.1*x(1);
end
```

В этом файле функция $jump$ является заглавной и ей передается управление при обращении $jump(t, x, flag)$;. В зависимости от значения переменной $flag$, в результате обработки оператором $switch$ управление передается одной из функций $init$, $events$, ft. Две первые обрабатывают соответствующие значения переменной $flag$, а последняя производит вычисления правых частей.

При обращении к функции $jump$ при значении переменной $flag =' demo'$ запускается процесс численного интегрирования задачи, с описанными выше начальными условиями, определяемыми функцией $init$. При этом в процессе численного интегрирования в графическое окно выводится фазовый портрет, в матрицах T и X по окончании вычислений выводятся результаты численного интегрирования, а в матрицы Te и Xe моменты времени и значения вектора состояния при касании точки A опорной поверхности. Результат получаемый при обращении к функции $jump$ командой

```
[T,X,Te,Xe,Ie]= jump([],[],'demo');
```

приведен на рис. 12.

5.5 Программы численного интегрирования линейных моделей управляемых систем.

Одним из основных достоинств пакета MATLAB является наличие широкого набора библиотек - TOOLBOX'ов - предназначенных для решения специальных типов математических и инженерных задач. В настоящем параграфе рассмотрим использование нескольких функций библиотеки "Control system toolbox"для решения систем линейных дифференциальных уравнений. Более подробно с функциями из "Control system toolbox"можно познакомиться в [11].

Модель линейной динамической управляемой системы запишем в виде системы дифференциальных уравнений 1-го порядка

$$\begin{array}{rcl} \dfrac{dx}{dt} & = & Ax + Bu \\ y & = & Cx + Du \end{array} . \qquad (10)$$

где x - вектор состояния системы, u - вектор управлений, y - вектор выходов, A, B, C, D - матрицы с постоянными коэффициентами.

В библиотеке "Control"представлены следующие функции, позволяющие решать задачи интегрирования:

- $initial$ - функция решения задачи Коши для при заданных начальных условиях. Обращение к этой функции может иметь вид:

Рис. 12: Результат обращения к функции *jump*.

$$[Y, T, X] = initial(A, B, C, D, x0, Tf)$$

Здесь A, B, C, D - матрицы, описывающие систему вида (10); $x0$ - вектор начальных условий, Tf - значение момента времени до которого осуществляется интегрирование. Выходными параметрами являются матрицы Y и X, в которых помещаются построчно значения векторов y и x для моментов времени из соответствующего вектора T.

- *impulse*- вычисление весовой функции системы (10). Функция решения задачи о поведении системы при импульсном возмущении входа с номером i сигналом типа дельта-функции

$$u_i(t) = \delta(t) = \begin{cases} 1 & \text{при } t = 0 \\ 0 & \text{при } t \neq 0 \end{cases}$$

при нулевых начальных условиях.

Обращение к этой функции может иметь вид:

$$[Y, T, X] = impulse(A, B, C, D, i, Tf)$$

Здесь A, B, C, D - матрицы, описывающие систему вида (10); i - номер возмущаемого входа, Tf - значение момента времени до которого осуществляется интегрирование. Выходными параметрами являются матрицы Y и X, в которых помещаются построчно значения векторов y и x для моментов времени из соответствующего вектора T.

- *step*- вычисление переходной функции системы (10). Функция решения задачи о поведении системы при возмущении входа с номером i сигналом в виде единичной ступеньки

$$u_i(t) = \begin{cases} 0 & \text{при } t < 0 \\ 1 & \text{при } t \geq 0 \end{cases}$$

Обращение к этой функции может иметь вид:

$$[Y, T, X] = step(A, B, C, D, i, Tf)$$

Аргументы и выходные параметры функции *step* имеют тот же смысл, что и для функции *impulse*.

- *lsim* - вычисляет решение системы для случая произвольного входного сигнала, заданного массивом своих отсчетов - матрицей U. Обращение к этой функции может иметь вид:

$$[Y, T, X] = lsim(A, B, C, D, U, T, x0)$$

Здесь A, B, C, D - матрицы, описывающие систему вида (10); U и T - матрица значений векторной функции u и соответствующих им моментов времени. Выходными параметрами являются матрицы Y и X, в которых помещаются построчно значения векторов y и x для моментов времени из соответствующего вектора T. Матрица X и вектор T из числа выходных параметров может быть опущена. Если аргументом $x0$ отсутствует, программа производит вычисления с нулевыми начальными условиями.

Во всех случаях матрица X из числа выходных параметров может быть опущена. Если аргументом Tf является вектор, программа воспринимает его как вектор моментов времени, в которые выводится решение.

Эффективность использования указанных процедур связана с тем, что они осуществляются не посредством численного интегрирования, а в результате решения задачи о собственных числах и собственных векторах матрицы A и последующего использования аналитических зависимостей для решений указанной системы.

Список литературы

[1] Потемкин В.Г. Система MATLAB.// М.: Изд. "Диалог-МИФИ", 1997.

[2] Потемкин В.Г. MATLAB 5 для студентов.// М.: Изд. "Диалог-МИФИ", 1998.

[3] Потемкин В.Г. Система инженерных и научных расчетов MATLAB 5.x.// в 2-х томах. М.: Изд. "Диалог-МИФИ", 1999.

[4] Дьяконов В.П. Справочник по применению системы PC MATLAB.// М.: Физматлит, 1993.

[5] Дьяконов В.П., Абраменкова И.В. MATLAB 5.0/5.3. Система символьной математики. // М.: Нолидж, 1999.

[6] Мартынов Н.Н., Иванов А.П. MATLAB 5.X. Вычисления, визуализация, программирование.- М.: КУДИЦ-ОБРАЗ, 2000.

[7] Конев В.Ю., Мироновский Л.А. Основные функции пакета MATLAB.// СПб, Изд. СПГААП, 1992.

[8] Коробова Н.Л., Загашвили Ю.В. Комплекс автоматизированного проектирования MATLAB-CTRL.// СПб, Изд. СПГААП, 1993.

[9] Андрушевский Б.Р., Фрадков А.Л. Элементы математического моделирования в программных средах MATLAB и Scilab.// СПб, изд. Наука, 2001.

[10] Гультяев А. MATLAB 5.2. Имитационное моделирование в среде Windows. // СПб.: КОРОНАпринт, 1999.

[11] Медведев В.С. Потемкин В.Г. Control System Toolbox. MATLAB 5 для студентов.// М.: Изд. "Диалог-МИФИ", 1999.

[12] Селезнев А.В. Эффективное программирование в среде MATLAB.// в кн. Тезисы докладов всероссийской научной конференции "Проектирование научных и инженерных приложений в среде MATLAB"// М.: ИПУ РАН. 2002. с.199-200.

[13] PC MATLAB: User's Guide. MathWorks Inc., 1998.

[14] Голуб Дж., Ван Лоун Ч. Матричные вычисления.// М.: Мир. 1999.

[15] Воеводин В.В., Кузнецов Ю.А. Матрицы и вычисления.// М.: Наука. 1984.

КРУЧИНИН Павел Анатольевич [2mm]
ОСНОВЫ ПРОГРАММИРОВАНИЯ
В СРЕДЕ MATLAB.

Учебное пособие

Оригинал-макет подготовлен издательской группой механико-математического факультета МГУ
с использованием издательской системы LaTeX2ε
на механико-математическом факультете МГУ.

Подписано в печать 07.03.2001 г.
Формат 60×90 1/16. Объем 7,5 п.л.
Заказ Тираж 100 экз.

Издательство ЦПИ при механико-математическом
факультете МГУ, г. Москва, Воробьевы горы
Лицензия на издательскую деятельность ЛР № 040746
от 12.03.1996 г.

Отпечатано на типографском оборудовании механико-
математического факультета МГУ им. М. В. Ломоносова
и Франко-русского центра им. А. М. Ляпунова

Предметный указатель