Федеральное агентство по образованию Российской Федерации
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Энергетический факультет

А.Н. Рыбалев

ИНФОРМАТИКА. СПЕЦИАЛЬНЫЕ ГЛАВЫ

ПОСОБИЕ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Благовещенск
2007

Содержание

Введение

Лабораторные работы по курсу «Информатика. Спец. главы» выполняются в компьютерном классе с использованием пакета Matlab. Выбор пакета в основном связан с тем обстоятельством, что Matlab включает в себя мощные средства для расчета и моделирования система автоматического управления, которые понадобятся студентам на старших курсов при изучении многих дисциплин.

Порядок выполнения работ следующий.

В начале занятия (кроме первого) студенты выполняют индивидуальные задания по теме предыдущего занятия. Зачет по теме ставится, если студент выполняет задание в течение контрольного времени (5 – 10 мин. в зависимости от сложности).

Далее под контролем и в случае необходимости – руководством преподавателя разбирается материал по новой теме (30 – 60 мин). План изучения по темам изложен в соответствующих разделах данного пособия.

В конце занятия студенты выполняют самостоятельные задания по теме занятия.

1. Знакомство с основными пользовательскими интерфейсами системы Matlab

1.1. Основы работы с оболочками Matlab

Основными интерфейсами пользователя Matlab являются:

Окно команд.

Редактор-отладчик.

Средство просмотра рабочей области.

Средство просмотра и редактирования путей доступа.

Окно помощи.

Окно команд – основное окно, в котором работает пользователь. Его закрытие приводит к завершению работы с системой и закрытию остальных окон.

Окно команд обеспечивает

– организацию вычислений;

– доступ ко всем остальным интерфейсам.

Организация вычислений предполагает ввод пользователем выражений в командной строке и вывод результатов в этом же окне (ниже) (результат не выводится, если после выражения стоит ';') либо в графических окнах.

В качестве выражений могут быть любые допустимые языком Matlab операторы, вызов на исполнение функций и script-файлов (файлов-сценариев).

Script-файлы вызываются на исполнение путем ввода имени файла в командной строке. Например, вызов script-файла myscript.m:

```
>> myscript
```

Файлы-функции вызываются аналогично, но с учетом передачи параметров. Например, вызов функции `myfun.m` с параметрами 1.4 и 2.5:

```
>> myfun(1.4, 2.5)
```

Кроме того, вызов функции может участвовать в выражениях, например

```
>> a = 3.5/myfun(b+55,11)
```

Доступ ко всем остальным интерфейсам Matlab осуществляется как из командной строки (исполнением специальных команд типа edit, help), так и с помощью панели инструментов.

Редактор-отладчик – средства для набора, редактирования и отладки script-файлов и файлов- функций.

Запускается из Окна команд с помощью кнопок панели инструментов илипо команде `edit, edit <имя файла>` в командной строке.

Позволяет:

– редактировать файлы программ (с выделением ключевых слов и синтаксических ошибок);

– просматривать значения переменных, по их идентификаторам в текстах программ (с помощью мыши);

– запускать программы на исполнение (Пункт "Run" меню "Инструменты");

– производить отладку программ (останов в контрольных точках, по шаговый режим и т.д.).

К сожалению, редактор-отладчик не выводит сообщения об ошибках (они выводятся в Окне команд)

Средство просмотра рабочей области – позволяет просмотреть список переменных рабочей области с указанием:

– имен;

– размеров (все переменные в Matlab являются массивами);

– размеров занимаемой памяти в байтах;

– класса (типа).

Ту же информацию можно получит по команде whos.

Кроме того, средство позволяет удобно для пользователя представить и отредактировать численные переменные (двойной щелчок мыши по имени переменной - вызов редактора-отладчика).

Средство просмотра и редактирования путей доступа – предназначено для просмотра и редактирования путей доступа – списка каталогов (папок), из которых Matlab извлекает файлы на выполнение.

Средство позволяет:

– добавлять и удалять каталоги из списка;

– изменять последовательность поиска.

Доступно из панели инструментов Окна команд или по команде `editpath` в командной строке.

Окно помощи – представляет удобный доступ к заголовочным частям библиотечных функций (комментариям).

Доступно из панели инструментов Окна команд или по команде `helpwin` в командной строке.

1.2. Справочные и управляющие команды Matlab

Все они (либо файлы программ, либо файлы помощи для встроенных команд) располагаются в \Matlab\general.

Основными являются следующие.

Получение помощи:

`help <имя файла>` – получение помощи (комментария по файлу);

`help <имя каталога>` – получение помощи по функциям каталога (пакета);

`demo` - запуск демонстрационных программ.

Управление переменными:

`who, whos` – информация о переменных рабочей области;

`clear` – удаление переменных и функций из памяти;

`load, save` – загрузка переменных из файлов и запись их в файлы (используются как файлы спец. формата типа MAT, так и обыкновенные текстовые).

Информация по файлам:

`type` – вывод файла на экран;

`what` – список файлов каталога с сортировкой по типу;

`which` – определение пути доступа к файлу;

`edit` – запуск редактора-отладчика.

2. Простейшие векторно-матричные вычисления

2.1. Элементарные действия над матрицами в Matlab

Ввод матриц (\MATLAB\elmat):

матрица-строка

```
A = [1 2 3]
```

матрица-столбец

```
B = [1; 2; 3]
```

прямоугольная матрица

```
C = [1 2 3;4 5 6]
```

единичная диагональная матрица размером 3 на 3

```
E = eye(3)
```

диагональная матрица с заданной диагональю

```
D = diag([1 2 3])
```

то же самое, с учетом, что A = [1 2 3]

```
D = diag(A)
```

матрица с нулевыми элементами 3 на 4

```
Z = zero(3,4)
```

Определение размеров матриц (\MATLAB\elmat).

Вычисление размеров:

```
size(A)
[m,n] = size(C)
```

Вычисление "длины"

```
length(B)
n = length(C)
```

Поиск максимального, минимального элементов (\MATLAB\datafun).

Введем, например, следующую матрицу

```
M = [2 4 6 3; 6 8 3 0]
```

тогда

```
max(M)
[n,i] = max(M)
```

или, допустим, имеем еще

```
N = [1 4 3 5; 8 4 6 0]
```

тогда

```
max(M,N)
```

Аналогично работает функция `min`.

2.2. Математические операции

Введем матрицы

```
A = [1 2; 3 4]
B = [2 3; 0 1]
```

Умножение:

матричное

```
A*B
```

поэлементное

```
A.*B
```

сложение

```
A+B
A+3
```

Возведение в степень

```
A^2
A.^2
```

Транспонирование

```
A'
```

Логические операции и функции

```
A==B
A<B
A>3
```

```
all(A>3)
any(A>3)
```

2.3. Линейная алгебра (\MATLAB\matfun)

Решим систему линейных уравнений

$3x_1 + 2x_2 + 4x_3 + 6x_4 = 6$

$x_1 - 3x_2 + 2x_3 - 3x_4 = 4$

$-2x_1 + 9x_2 - 3x_3 + x_4 = -3$

$2x_1 + 2x_2 + 7x_3 + 5x_4 = 1$

В матричном виде система может быть представлена как

AX =B

Введем матрицы A и B

```
A = [3 2 4 6; 1 -3 2 -3; -2 9 -3 1; 2 2 7 5]
B = [6; 4; -3; 1]
```

Решение

```
X = A\B
```

Определитель матрицы A:

```
det(A)
```

Собственные числа матрицы A:

```
eig(A)
```

Характеристический полином матрицы A:

```
p = poly(A)
```

Его корни (\MATLAB\polyfun)

```
roots(p)
```

3. Типы данных языка Matlab

Основные типы языка Matlab:

double – массивы, в том числе многомерные, чисел удвоенной точности, в том числе комплексных;

char – массивы символов;

cell – массивы ячеек, могут содержать данные различных типов;

struct – структуры.

Tun double

Правила ввода массивов и работы с ними были рассмотрены выше. В основном они справедливы и для многомерных массивов.

Пример:

```
% Создадим нулевой массив 2 на 3 на 4
M = zeros(2,3,4)
% Заполним четвертую «страницу» массива
M(:,:,4) = [1 2 3; 4 5 6]
% Обратимся к столбцу №3 четвертой «страницы»:
M(:,3,4)
```

Специфические функции многомерных массивов:

cat – объединение массивов.

ndims – вычисление числа размерностей;.

ndgrid – формирование массивов в виде расчетной сетки.

permute – перестановка размерностей.

ipermute – обратная перестановка размерностей.

shiftdim – сдвиг размерности.

squeeze – удаление единичных размерностей.

В Matlab нет специального типа данных для представления комплексных чисел, – все числа по умолчанию считаются комплексными. Действительные числа рассматриваются как комплексные с нулевой мнимой частью.

При запуске программы автоматически инициализируются мнимыми единицами две переменные: `i` и `j`. В дальнейшем они могут использоваться в этом качестве, а могут быть и переопределены. Определение любой переменной значением мнимой единицы может быть выполнено следующим образом:

```
i = sqrt(-1)
```

С такой переменной можно задать любое комплексное число:

```
a =  2+i, b = 1+2*i, c =exp(4+2*i)
```

Альтернативный способ задания комплексных чисел не использует специальных переменных (и знаков умножения) и поэтому является более универсальным:

```
a =  2+1i, b = 1+2j, c =exp(4+2i)
```

В данном случае `i` и `j` – это не переменные, а просто символы.

Специфические операции для работы с комплексными числами:

abs – вычисление абсолютной величины (длины вектора) для векторного представления комплексного числа;

angle – вычисление фазового угла вектора для векторного представления комплексного числа;

conj – вычисление комплексного сопряжения;

imag – вычисление мнимой части комплексной переменной;

real – вычисление действительной части комплексной переменной.

Tun char

Для задания переменных такого типа используются одинарные кавычки:

```
S = 'String'.
```

В Matlab строковые переменные рассматриваются в виде векторов-строк, содержащих ASCII - коды символов. Поэтому над ними разрешаются все действия, разрешенные для соответствующих «численных» векторов. Однако способ вывода результатов на экран (в численном или строковом виде) зависит от произведенной операции. Приведем наиболее важные функции Matlab для работы со строками:

abs – преобразует строку в вектор ASCII - кодов символов;

setstr – преобразует вектор ASCII - кодов символов в строку;

sprintf – преобразует число в строку-представление заданного формата;

sscanf – читает строку-представление числа заданного формата.

Для ознакомления с остальными функциями введите

```
help strfun
```

Tun cell

Cell – массивы ячеек, которые могут содержать в себе данные любых типов. Для работы с ними определена операция {}.

Пример:

```
A =[1 2; 3 4]  %матрица
S = 'abc'      %строка
C = {A,S}      % массив ячеек
% Обратимся к матрице из массива ячеек
C{1}
% Обратимся к элементу (1,2) матрицы
C{1}(1,2)
```

О специфических функциях для работы с массивами ячеек можно узнать, введя

```
help datatypes
```

и обратившись к разделу

```
Cell array functions
```

Tun struct

Переменные типа struct представляют собой структуры – собрания именованных полей различными типами данных. Для структур определена операция «.».

Пример:

```
A =[1 2; 3 4]  %матрица
S = 'abc'      %строка
% Введем структуру
ST. Array = A
ST.String = S
% Обратимся к матрице из структуры
ST.Array
% Обратимся к элементу (1,2) матрицы
ST.Array(1,2)
```

О специфических функциях для работы с массивами ячеек можно узнать, введя

```
help datatypes
```

и обратившись к разделу

```
Structure functions
```

Массивы ячеек и структуры используются очень широко при обмене данными с функциями.

4. Графическая система Matlab

В Matlab используется так называемая дескрипторная графика: изображение формируется из объектов, каждый из которых однозначно адресуется дескриптором – уникальным числом. Основные типы графических объектов:

figures – фигуры (графические окна), имеют целочисленные дескрипторы, равные номерам фигур;

axes – оси, имеют дробные дескрипторы;

lines – линии, имеют дробные дескрипторы;

surfaces – поверхности, имеют дробные дескрипторы.

Каждый графический объект характеризуется определенным для его типа набором свойств, управляющим его отображением. Для работы со свойствами используются функции get и set.

Пример:

```
%Создание фигуры 3
figure(3)
% Сделаем белым фон
set(3, 'Color',[0 0 0])
a = axes;
% Узнаем позицию осей
get (a,'Position')
% Зададим новую позицию и размеры осей
set(a, 'Position', [.5 .5 .4 .4])
```

```
% Нанесем на оси линию
l = line([1 2],[3 4])
% Сделаем ее красной
set(l, 'Color','red')
```

Основные свойства графических объектов доступны непосредственно в графических окнах.

В основном пользователю системы нет необходимости напрямую работать с графическими объектами, поскольку система имеет достаточно большой набор графических функций высокого уровня. Ниже рассмотрены наиболее часто используемые функции.

Двумерный график (\matlab\graph2d)

Расчет графика функции $y = 2\sin(3x^2)$ на интервале от 0 до 3 с шагом 0.01:

```
x = 0:.01:3;
y = 2*sin(3*x.^2)
```

Построение графика и сетки

```
plot(x,y), grid
```

Закроем все графические окна

```
close all
```

Управление окнами (\matlab\graphics)

Допустим, мы хотим одновременно вывести в разные окна графики sin(x), cos(x) и y. Для каждого графика создаем новое окно

```
figure, plot(x,sin(x)),grid, title('sin(x)')
figure, plot(x,cos(x)),grid, title('cos(x)')
figure, plot(x,y),grid, title('y(x)')
```

Теперь закроем окна 2 и 3

```
close([2 3])
```

выберем окно 1

```
figure(1)
```

добавим туда график cos(x)

```
hold on
plot(x,cos(x),'r')
hold off
```

Трехмерный график (\matlab\graph2d)

Построим поверхность $Z = \sin^2(x)+\cos^2(y)$ над прямоугольником $-5 < x < 5$, $-4 < y < 4$:

```
x = -5:.1:5;
y = -4:.1:4;
[X,Y] = meshgrid(x,y);
Z = sin(X).^2 + cos(Y).^2;
mesh(x,y,Z)
```

Полный список графических функций можно получить, введя

`help graphics` (дескрипторная графика)

`help graph2d` (двумерные графики)

`help graph3d` (трехмерные графики)

Кроме того, полезно ознакомиться с демонстрационными примерами

`demo` (раздел Graphics)

5. Программирование в Matlab

5.1. Работа с функциями

Пользователь Matlab может создавать свои функции для решения определенных задач. В отличии от Script-файлов, функции имеют некоторые элементы оформления и по умолчанию оперируют локальными переменными.

Например, создадим функцию minsqr для аппроксимации опытных данных многочленом типа $Z=a_1+a_2XY$, использующую метод наименьших квадратов.

Входными аргументами такой функции естественно принять вектора x и y (представляющие собой, например, известные значения двух входных воздействий на некоторый объект) и матрицу Z экспериментальных данных (реакция объекта).

Выходным аргументом функции будет вектор коэффициентов a=[a_1;a_2].

```
function a = minsqr(x,y,Z)
% Аппроксимация опытных данных многочленом
% типа Z=a1+a2*X*Y
% с использованием метода наименьших квадратов.
[X,Y] = meshgrid(x,y);
[m,n] = size(X);
A = zeros(2,2); B = zeros(2,1);
A(1) = m*n;
A(2) = sum(sum(X.*Y));
A(3) = A(2);
A(4) = sum(sum(X.^2.*Y.^2));
B(1) = sum(sum(Z));
B(2) = sum(sum(Z.*X.*Y));
a = A\B;
```

Текст функции помещается в одноименный файл.

Для проверки работоспособности функции сформируем матрицы X и Y и рассчитаем матрицу Z, приняв, например, что Z=2+3XY.

```
x = 0:.1:1; y = 2:.1:6;
[X,Y] = meshgrid(x,y);
Z=  2+3*X.*Y;
```

Если функция работает правильно, результатом ее вызова, очевидно, будет вектор [2;3]:

```
a = minsqr(x,y,Z)
```

Функции inline (\matlab\funfun\@inline)

В отдельных случаях функции могут оформляться не в виде отдельных файлов, а путем формирования т.н. объектов `inline`.

С точки зрения иерархии типов данных Matlab объекты `inline` представляют собой пользовательский класс данных (user), несмотря на то, что их поддержка обеспечена самими производителями программы Matlab. Для данного класса определена структура данных и переопределены некоторые функции Matlab (например, `feval`). Подробнее см.

```
help inline
methods inline
```

При формировании объектов inline необходимо явно задать строку вычисления функции, например:

```
y = inline('sin(x)*cos(x)')
```

Если входных аргументов более одного, необходимо также указать их имена:

```
z = inline('sin(x)*cos(y)','x','y')
```

В большинстве случаев вызов функции inline не отличается от вызова обычных функций-файлов:

```
y(pi/3)
z(pi/3,pi/6)
```

Необходимо лишь помнить, что, например, в данном случае y и z – имена функций.

Управляющие конструкции

Как и во всех языках высокого уровня в Matlab имеется полный набор управляющих конструкций:

Условные переходы

```
if…elseif…else…end;
```

циклы

```
while…end,
for…end;
```

переключатель

```
switch…case…otherwise…end
```

О правилах их использования можно узнать из справочной системы, введя, например

```
help if
```

6. Аппроксимационные и оптимизационные задачи

6.1. Аппроксимация зависимости y(x) полиномом (метод наименьших квадратов). Используется функция

```
p = polyfit(x,y,n)
```

где x,y - вектора (экспериментальные данные), n - порядок полинома, p - вектор коэффициентов полинома, начиная с коэффициента при старшей степени (\matlab\polyfun).

Пример:

Рассчитаем некоторую функцию-полином

```
x = 0:.1:10;
y = 3*x.^2 + 2*x +1;
```

Аппроксимируем данные полиномами первого и второго порядков:

```
p1 = polyfit(x,y,1)
p2 = polyfit(x,y,2)
```

Вычислим значения полиномов в точках x и построим соответствующие графики:

```
y1 = polyval(p1,x);
y2 = polyval(p2,x);
plot(x,y1,x,y2), grid
```

6.2. Минимизация функции одной переменной

```
xmin=fminbnd(N_of_F,x1,x2)
xmin=fminbnd(N_of_F,x1,x2,options)
```

где N_of_F - имя минимизируемой функции, x1,x2 - границы по x, в пределах которых ищется минимум, options - строка опций процедуры.

Значения опций по умолчанию можно получить с помощью функции `foptions`, см.

```
help foptions
```

Изменить опции, можно, например, так (зададим "терминальную" точность вычислений 10^{-5}):

```
options = foptions
options(2) = 1e-5
```

(\matlab\funfun)

Пример

Определим минимум функции $y = \sin(x)/(5x^2-x)$ на интервале x = 3..6.

Зададим функцию и построим ее график на интервале

```
y = inline('sin(x)./(5.*x.^2-x)')
x = 3:.05:6;
plot(x,y(x)),grid
```

Определим координаты минимума

```
xmin = fminbnd(y,3,6)
```

6.3. Минимизация функции нескольких переменных (симплекс-метод)

```
xmin= fminsearch (N_of_F,x0)
xmin= fminsearch (N_of_F,x0,options),
xmin= fminsearch (N_of_F,x0,options,[],P1,P2,..),
```

где N_of_F - имя минимизируемой функции, x0 - вектор начальных приближений, P1, P2,... доп. параметры для минимизируемой функции.

Функция `fmins` требует описания минимизируемой функции в виде

```
y = N_of_F(x),
y = N_of_F(x,P1,P2,..),
```

где x - вектор аргументов, по которым ищется минимум (\matlab\funfun).

Пример:

Найти минимум функции $y = (x_1-1)^2+(x_2+2)^2$ (естественно, он находится в точке $x_1=1$, $x_2= -2$).

Опишем функцию как объект inline:

```
y = inline('(x(1)-1)^2+(x(2)+2)^2')
```

Найдем координаты минимума, начиная поиск с точки $x_1=0$, $x_2=0$.

```
xmin = fminsearch (y,[0 0])
```

Функция `fminsearch` удобна для аппроксимации данных различными аналитическими зависимостями. В этом случае минимизировать необходимо «разницу» между экспериментальными значениями неизвестной функции и ее аналитически вычисленными значениями. При этом аналитическая зависимость может иметь любой вид (в отличии, например, от метода наименьших квадратов).

Пример:

Аппроксимировать данные многочленом $a_1 + a_2xy$.

Минимизируемую функцию можно разместить в файле er:

```
function err = er(a,x,y,z)
err = sum(sum((z-a(1)-a(2)*x.*y).^2));
```

либо задать ее как объект inline:

```
er=inline('sum(sum((z-a(1)-…
                a(2)*x.*y).^2))','a','x','y','z')
```

Минимизируемая функция представляет собой сумму квадратов разности между z и a_1+a_2xy по всем точкам. Минимум находится по вектору a.

Для проверки метода сформируем соответствующие матрицы и найдем a_1 и a_2, соответствующие минимуму функции er:

```
x = 0:.1:1; y = 2:.1:6;
[X,Y] = meshgrid(x,y);
Z=  2+3*X.*Y;
```

Найдем a_1 и a_2:

```
a = fminsearch ('er',[0 0],[],[],X,Y,Z)
```

– если er – файл с функцией,

```
a = fmins(er,[0 0],[],[],X,Y,Z)
```

– если er – inline-объект

Более широкие возможности по решению оптимизационных задач предоставляет пакет Optimization Toolbox. Для обзора рекомендуется запустить демонстрационный файл

```
tutdemo
```

7. Задача на решение линейных дифференциальных уравнений с постоянными коэффициентами и их систем

Для решения линейных дифференциальных уравнений и их систем могут использоваться функции пакета Control Systems. Данный пакет в основном предназначен для исследования и синтеза линейных систем управления, однако его возможности позволяют решать линейные дифференциальные уравнения и их системы безотносительно проблемам теории управления.

Пакет Control Systems построен на основе объектно-ориентированной технологии и включает несколько классов, представляющих математическое описание линейных объектов. Мы будем использовать классы

`tf` (transfer function) – передаточные функции;

`ss` (state space) – описание в пространстве состояний;

Эти классы являются потомками виртуального класса `lti` (linear time-invariant – линейные системы с постоянными параметрами) и наследуют от него поля (свойства) для описания объектов и методы (функции) для работы с ними.

Объекты `tf`, `ss` описываются соответствующими структурами, поля которых заключают свойства объекта. Набор свойств для всех классов можно просмотреть введя:

```
help ltiprop
```

Доступ к полям объектов в Matlab осуществляется с помощью функций `set` (установить) и `get` (получить) и путем непосредственного обращения к полю. Однако в большинстве случаев необходимости в этом нет, так как основные поля заполняются и изменяются функциями пакета.

Для детального ознакомления с функциями воспользуйтесь справочной системой:

меню окна управления Matlab;

командами помощи: `help control, help 'function_name';`

командой демонстрации: `demo`.

Большинство функций пакета Control работает со всеми тремя классами объектов, меняя алгоритм своей работы в зависимости от типа объекта. Для создания объектов применяется функции-конструкторы, название которых совпадает с именем класса: `tf, ss`.

Передаточные функции

Передаточной функцией системы называют отношение изображения по Лапласу выходной переменной к изображению по Лапласу входной переменной при нулевых начальных условиях:

$$W(p) = \frac{x_{âûõ}(p)}{x_{âõ}(p)} \qquad (1)$$

при

$$x_{âûõ}(0) = \frac{dx_{âûõ}}{dt}(0) = \frac{d^2x_{âûõ}}{dt^2}(0) = \ldots = \frac{d^{n-1}x_{âûõ}}{dt^{n-1}}(0) = 0, \qquad (2)$$

где n – порядок дифференциального уравнения, описывающего звено или систему; p – оператор Лапласа.

Как известно, изображение по Лапласу функции f(t) находят по формуле:

$$f(p) = \int_0^\infty f(t)e^{-pt}dt. \qquad (3)$$

Обратное преобразование вычисляется следующим образом:

$$f(t) = \frac{1}{2\pi j}\int_{-j\infty}^{j\infty} f(p)e^{pt}dp. \qquad (4)$$

Зная передаточную функцию системы и входное воздействие $x_{вх}(t)$, можно определить его (ее) реакцию на это воздействие при нулевых начальных условиях. Для этого нужно, предварительно рассчитав по (3) изображение входной величины $x_{вх}(p)$, определить изображение выходной величины

$$x_{âûõ}(p) = x_{âõ}(p)W(p) \qquad (5)$$

и далее с помощью обратного преобразования (4) вычислить $x_{вых}(t)$.

Линейные звенья и системы описываются дифференциальными уравнениями вида

$$\grave{a}_n \frac{d^n x_{âûõ}}{dt^n} + \grave{a}_{n-1} \frac{d^{n-1} x_{âûõ}}{dt^{n-1}} + ... + \grave{a}_1 \frac{dx_{âûõ}}{dt} + a_0 x_{âûõ} =$$

$$= b_m \frac{d^m x_{âõ}}{dt^m} + b_{m-1} \frac{d^{m-1} x_{âõ}}{dt^{m-1}} + ... + b_1 \frac{dx_{âõ}}{dt} + b_0 x_{âõ}. \qquad (6)$$

В соответствии с одной из теорем операционного исчисления изображение k-й производной функции f(t) при нулевых начальных условиях:

$$\frac{d^k f}{dt^k} \Rightarrow p^k f(p) \qquad (7)$$

при

$$f(0) = \frac{df}{dt}(0) = \frac{d^2 f}{dt^2}(0) = ... = \frac{d^{k-1} f}{dt^{k-1}}(0) = 0. \qquad (8)$$

Кроме того, известно, что изображение линейной комбинации функций равно линейной комбинации изображений.

Поэтому, преобразовав по Лапласу левую и правую части уравнения (6), получим:

$$\grave{a}_n p^n x_{âûõ}(p) + \grave{a}_{n-1} p^{n-1} x_{âûõ}(p) + ... + \grave{a}_1 p x_{âûõ}(p) + a_0 x_{âûõ}(p) =$$
$$= b_m p^m x_{âõ}(p) + b_{m-1} p^{m-1} x_{âõ}(p) + ... + b_1 p x_{âõ}(p) + b_0 x_{âõ}(p), \qquad (9)$$

откуда

$$W(p) = \frac{x_{âûõ}(p)}{x_{âõ}(p)} = \frac{b_m p^m + b_{m-1} p^{m-1} + ... + b_1 p + b_0}{\grave{a}_n p^n + \grave{a}_{n-1} p^{n-1} + ... + \grave{a}_1 p + a_0}. \qquad (10)$$

Таким образом, зная дифференциальное уравнение линейного звена (системы), можно легко получить его передаточную функцию, и наоборот. С этой точки зрения дифференциальные уравнения и передаточные функции, – эквивалентные понятия.

Рассмотрим теперь дифференциальное уравнение в том виде, в каком оно рассматривается в математике:

$$\text{à}_n \frac{d^n y}{dx^n} + \text{à}_{n-1} \frac{d^{n-1} y}{dx^{n-1}} + ... + \text{à}_1 \frac{dy}{dx} + a_0 y = F(x). \qquad (11)$$

Положив x = t, y $= x_{вых}$, F(x) = F(t) = $x_{вх}$, получим

$$\text{à}_n \frac{d^n x_{âûõ}}{dt^n} + \text{à}_{n-1} \frac{d^{n-1} x_{âûõ}}{dt^{n-1}} + ... + \text{à}_1 \frac{dx_{âûõ}}{dt} + a_0 x_{âûõ} = x_{âõ}. \qquad (12)$$

Передаточная функция, составленная по (12) имеет вид

$$W(p) = \frac{x_{âûõ}(p)}{x_{âõ}(p)} = \frac{1}{\text{à}_n p^n + \text{à}_{n-1} p^{n-1} + ... + \text{à}_1 p + a_0}. \qquad (13)$$

Ввести объект – передаточную функцию в Matlab можно с помощью функции tf, например, для дифференциального уравнения

$$\frac{d^3 y}{dx^3} + 2\frac{d^2 y}{dx^2} + 2\frac{dy}{dx} + y = 0{,}5x + \sin^2(x) = F(x) \qquad (14)$$

```
sys = tf(1,[1 2 2 1])
```

Решим уравнение (14) на интервале x = 0…10 при нулевых начальных условиях:

1) зададим вектор x, например, от нуля до десяти с шагом 0,1:

```
x = 0:.1:10
```

2) рассчитаем вектор значений правой части (14):

```
f = .5*x + sin(x).^2
```

3) рассчитаем вектор y:

```
y = lsim(sys,f,x)
```

4) построим график y(x)

```
plot(x,y), grid
```

Смотрите также

```
help lsim
```

Для решения дифференциального уравнения при ненулевых начальных условиях, оно должно быть преобразовано в систему уравнений первого порядка. Например, для уравнения (14) это можно сделать, положив

$$y_1 = y, y_2 = \frac{dy}{dx}, y_3 = \frac{d^2y}{dx^2}, \tag{15}$$

тогда получим систему

$$\begin{cases} \frac{dy_1}{dt} = y_2; \\ \frac{dy_2}{dt} = y_3; \\ \frac{dy_3}{dt} = F(x) - y_1 - 2y_2 - 2y_3. \end{cases} \tag{16}$$

с начальными условиями

$$y_1(0) = y(0), y_2(0) = \frac{dy}{dx}(0), y_3(0) = \frac{d^2y}{dx^2}(0) \tag{17}$$

Ее решение производится с помощью ss-модели.

Описание систем в пространстве состояний

State Space-объект описывает с помощью матриц уравнений состояний и выхода систему с несколькими входами и выходами

Любой объект (система) описывается в пространстве состояний уравнениями вида:

$$\dot{X} = F_1(X, U), \tag{18}$$

$$Y = F_2(X, U), \tag{19}$$

где X – вектор независимых координат объекта (системы), однозначно описывающих его (ее) состояние (вектор переменных состояния); Y – вектор выходных (измеряемых) величин; U – вектор входных воздействий. В общем случае вектор U может включать как управляющие (задающие), так и возму-

щающие воздействия, однако часто возмущения «выносятся» в отдельный вектор.

Для линейных объектов и систем уравнения (18,19) принимают вид:

$$\dot{X} = AX + BU, \tag{20}$$

$$Y = CX + DU, \tag{21}$$

где A – квадратная матрица состояний; B – матрица управления; C – матрица выхода; D – так называемая матрица «прямого обхода» (управление U как бы «в обход» внутренних состояний непосредственно действует на выход Y). Для реальных объектов и систем практически всегда D = 0.

Размеры матриц A, B, C и D определяются размерностями векторов состояния, управления и выхода. Если $X \in R^n, U \in R^m, Y \in R^\ell$, то матрицы имеют следующие размеры: $A - [n \times n],\ B - [n \times m],\ C - [\ell \times n],\ D - [\ell \times m]$.

Пусть, например, имеется система уравнений

$$\begin{cases} \dfrac{dx_1}{dt} = -2x_1 + x_2 - x_3; \\ \dfrac{dx_2}{dt} = -x_1 - x_2 + 2t; \\ \dfrac{dx_3}{dt} = -3x_1 - x_3 + 2\sin(t); \end{cases} \tag{22}$$

Необходимо найти функции $x_1(t)$, $x_2(t)$, $x_3(t)$ на интервале t = 0…10 при начальных условиях $x_1(0)=1$, $x_2(0)=0$, $x_3(0)= -1$.

Для приведения к форме (20),(21) представим систему (22) в виде

$$\begin{cases} \dfrac{dx_1}{dt} = -2x_1 + x_2 - x_3; \\ \dfrac{dx_2}{dt} = -x_1 - x_2 + u_1; \\ \dfrac{dx_3}{dt} = -3x_1 - x_3 + u_2; \end{cases} \tag{23}$$

$$y_1 = x_1; y_2 = x_2; y_3 = x_3. \tag{24}$$

Уравнение (23) – уравнение состояний, уравнение (24) – уравнение выхода.

В матричном виде уравнения (23), (24):

$$\begin{bmatrix} \frac{dx_1}{dt} \\ \frac{dx_2}{dt} \\ \frac{dx_3}{dt} \end{bmatrix} = \begin{bmatrix} -2 & 1 & -1 \\ -1 & -1 & 0 \\ -3 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad (25)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \quad (26)$$

Зададим матрицы описания:

```
A = [−2 1 −1; −1 −1 0; −3 0 −1]
B =[0 0; 1 0; 0 1]
C= eye(3)
D = zeros(3,2)
```

Создадим ss – объект:

```
sys1 = ss(A,B,C,D)
```

Сформируем $u_1(t)$ и $u_2(t)$ и объединим их в вектор-столбец U

```
t = 0:.1:10
u1 = 2*t;
u2 = 2*sin(t)
U = [u1;u2]
```

Сформируем начальные условия

```
x0 = [1;0;−1]
```

Найдем $x_1(t)$, $x_2(t)$, $x_3(t)$:

```
X = lsim(sys1,U,t,x0)
```

Смотрите

Построим графики функций:

```
plot(t,X), grid
```

8. Задача на численное интегрирование

Для решения систем обыкновенных дифференциальных уравнений в Matlab имеется ряд функций-«решателей»: ode45, ode23, ode113, ode23t, ode15s, ode23s, ode23tb, которые различаются точностью вычислений и возможностями.

Часть из них предназначена для решения нежестких систем, часть - для решения жестких систем. (Жесткими системами называются системы, у которых незначительное изменение переменных вызывает значительное изменение их производных. В частности это системы с разрывными правыми частями).

Подробнее о решателях можно узнать по

```
help funfun
```

Все функции-решатели предназначены для решения проблемы

dx/dt = F(t,X) – т.е. системы обыкновенных дифференциальных. уравнений в форме Коши.

Некоторые из них (ode23t,ode15s,ode23s,ode23tb), могут также решать задачу

Mdx/dt = F(t,X),

где M – так называемая «массовая матрица» (квадратная).

Кроме того, функции ode15s,ode23t,ode23tb решают ту же задачу и в случае нестационарной матрицы M:

M(t)dx/dt = F(t,X)

Описание системы диф. уравнений должно быть помещено в отдельную функцию. Минимальные требования к оформлению такой функции состоят в том, что она, принимая в качестве входных аргументов значение независимой переменной (как, правило, такой переменной является время) и вектор переменных системы, должна вычислить вектор производных переменных системы по независимой переменной (времени). Т.е., коротко говоря, функция описания должна вычислить правые части системы.

Таким образом, минимальный формат описания является следующим:

function pX = Name_of_fun(t,X)

где t - независимая переменная, X - вектор переменных системы, pX - вектор производных переменных системы по времени.

Вызов решателя для системы, описанной в файле Name_of_fun, в простейшем случае производится так:

[t,x] = solver('Name_of_fun',tspan,x0)

где

solver – функция-«решатель», например, ode45;

tspan - вектор моментов времени, в которые должно быть определено решение. Первый и последний элементы tspan определяют начальное и конечное время интегрирования. Промежуточные значения времени могут быть опущены, тогда tspan = [t0 tfinal], а промежуточные значения выбираются решателем самостоятельно;

x0 – вектор начальных значений переменных системы;

t – вектор-столбец расчетных моментов времени;

x – матрица, состоящая из столбцов - решений для каждой переменной системы в расчетные моменты времени.

Пример:

Решить систему дифференциальных уравнений

$dx_1/dt = -3x_1+2x_2x_3$;

$dx_2/dt = x_1 - 2x_2 + x_3$;

$dx_3/dt = -x_3 + \sin(2t)$.

Создадим функцию описания системы sys_fun и поместим ее в одноименный m-файл:

```
function px = sys_fun(t,x)
px =    [-3*x(1)+2*x(2)*x(3);…
          x(1) - 2*x(2) + x(3);…
         -x(3) + sin(t)];
```

Решим систему на промежутке t = 0..10 с нулевыми начальными условиями. (Система, похоже, нежесткая, - применим ode45.)

```
[t,x] = ode45('sys_fun',[0 10],[0 0 0]);
```

Построим графики изменения переменных системы во времени

```
plot(t,x), grid
```

Конечно, в приведенном выше примере предпочтительнее описать функцию как объект inline (проделайте самостоятельно).

Дополнительные возможности при решении систем дифференциальных уравнений открываются, если функция-решатель вызывается в форматах

[t,x] = solver('Name_of_fun',tspan,x0,options)

[t,x] = solver('Name_of_fun',tspan,x0,options,p1,p2)

где options - структура, описывающая опции решателя; p1, p2 - доп. параметры, передаваемые функции описания системы

Структура options создается функцией odeset, см.

```
help odeset
```

Другой формат вызова некоторых решателей

[t,x,te,xe,ie] = solver('Name_of_fun',tspan,x0,options)

позволяет отслеживать события - переход каких-либо величин через ноль:

te - время возникновения события;

xe - значение вектора x во время события;

ie - "индекс" события.

Использование опций, не принятых по умолчанию, позволяет:

– задавать точность вычислений (AbsTol, RelTol);

– задавать функцию вывода результатов решателя на каждом шаге вычислений (OututFcn);

– выводить статистику по окончанию расчета (Stats);

– использовать матрицу Якоби (dF/dx) для ускорения расчета (Jcobian, JConstant, JPattern);

– отслеживать «события» – переход каких-либо величин через ноль, и даже останавливать расчет при возникновении событий (Events);

– решать уравнения с «массовой матрицей» при векторе производных (Mass, MassConstant);

– задавать максимальный и начальный шаг интегрирования, а также максимальный порядок метода (MaxStep, InitialStep и MaxOrder)

и др.

В ряде случаев файл описания системы имеет более сложный вид и использует систему «флагов», подфункции и case-конструкцию. См.

```
help odefile
```

odefile.m – шаблон для создания файлов универсального описания систем.

Пример:

Пусть необходимо решить систему уравнений

$dx_1/dt + e^{-0,02t}dx_2/dt = -2x_1 - 0,05x_1x_2;$

$3dx_1/dt - dx_2/dt = -x_2^2 + x_1 + 10.$

Расчет провести до момента времени, когда координата x_2, снижаясь, достигнет значения 3, кроме того, фиксировать событие $x_1 = 1$.

При расчете использовать матрицу Якоби.

Создадим файл-описание системы, используя odefile.m как шаблон.

```
function varargout = sys_fun(t,x,flag)
switch flag
case ''
  varargout{1} = f(t,x);
case 'init'
  [varargout{1:3}] = init;
case 'jacobian'
  varargout{1} = jacobian(t,x);
case 'mass'
  varargout{1} = mass(t,x);
```

```
case 'events'
  [varargout{1:3}] = events(t,x);
otherwise
  error(['Unknown flag ''' flag '''.']);
end

%---- Вычисление правых частей----
function rigth_parts = f(t,x)
rigth_parts = [-2*x(1)-.05*x(1)*x(2);…
               -x(2)*x(2)+x(1)+10];

%-----Инициализация-------

function [tspan,x0,options] = init
tspan = [0 10];
x0 = [0 0];
options =  odeset('Jacobian','on','Mass','on', ...
'MassConstant', 'off', 'Events', 'on', ...
'OutputFcn', 'odeplot', 'Stats', 'on');

% -----Расчет матрицы Якоби--------
function dfdx = jacobian(t,x)
dfdx = [-2-.05*x(2) -.05*x(1);1 -2*x(2)];

% --- Расчет "массовой" матрицы -----
function M = mass(t,x)
M = [1 exp(-.02*t)-2; 3 -1];

%-----Контроль событий------
function [value,isterminal,direction] = events(t,x)
```

```
value = [x(1)-1 x(2)-3]   % условие: x1 = 1, x2 = 3
isterminal = [0 1];  % окончить ли расчет? [Нет Да]
direction = [0 -1];  % направление изменения
                      % координаты ? [Любое Вниз]
```

Расчет и построение графика

```
[t,x,te,xe,ie] = ode15s('sys_fun');
Evevts_t_x1_x2_num =[ te, xe, ie]
```

9. Построение Simulink-модели системы

Система Simulink – программный продукт семейства Matlab предназначенный для моделирования динамических систем. Она работает в тесном взаимодействии со средой Matlab, обмениваясь с ней данными.

Создание модели системы в Simulink происходит в наиболее простой и естественной форме - путем сборки модели из блоков, представляющих динамические звенья, статические характеристики и другие алгоритмы преобразования сигналов (в том числе и нелинейные). Расчет моделей производится с помощью методов численного интегрирования.

При запуске системы Simulink формируется пустое окно для набора модели системы и окно базовой библиотеки блоков Simulink .

Базовая библиотека Simulink содержит 13 разделов, важнейшими из которых являются

Continues – блоки непрерывной обработки;

Discontinues – блоки, реализующие разрывные функции;

Discrete – блоки дискретной обработки;

Math Operations – математические функции;

Ports & Subsystems – средства создания подсистем;

Signal Routings – управление распространением сигналов;

Sources – источники сигналов;

Sinks – приемники сигналов.

Кроме того, пакеты прикладных программ Matlab добавляют свои разделы блоков.

Модель системы набирается перетаскиванием мышью блоков из разделов библиотеки в окно набора модели и внутри его с последующим соединением блоков стрелками, показывающими распространение сигналов от блока к блоку. Копирование блоков в пределах окна модели также осуществляется перетаскиванием с помощью правой кнопки мыши.

Модель обязательно должна содержать хотя бы один блок-источник сигналов из раздела Sources, например, блок Step (ступенчатое изменение входного сигнала). Для «измерения» и запоминания контролируемых переменных (сигналов) модели применяются блоки-«приемники сигналов» раздела Sinks, чаще всего блок Scope (вывод временного графика изменения сигнала).

Модель динамики исследуемых объектов, процессов и систем набирается с помощью блоков разделов Continues, Discontinues, Discrete, Math Operations, Ports & Subsystems, Signal Routings. В последнем разделе представлены блоки для выполнения вспомогательных операций по соединению блоков, мультиплексированию сигналов и созданию подсистем.

После помещения необходимых блоков в окно набора модели необходимо задать их параметры: значения коэффициентов, временные задержки и т.д. Доступ к параметрам возможен по двойному щелчку левой кнопкой мыши на интересующем блоке. При щелчке правой кнопкой мыши появляется меню, позволяющее работать с буфером обмена Windows, а также настроить внешний вид блока (например, изменить его ориентацию в окне модели).

Линии связи (стрелки), соединяющие блоки модели, могут показывать направление передачи как одного (скалярного) сигнала, так и нескольких (векторного сигнала). В последнем случае блоки, как правило, обрабатывают одинаковым образом все компоненты вектора.

Некоторые блоки, однако, специально предназначены для обработки векторов, например, операции матричного умножения или свертки векторов.

Просмотреть «ширину» линии можно, пометив пункт Signal Dimensions меню Format. Выделить векторные линии жирным, можно, пометив пункт Wide Vector Lines.

После набора модели можно запустить процедуру расчета, предварительно задав ее параметры (при необходимости). Под параметрами в данном случае понимаются название функции-решателя, начальное и конечное время интегрирования, а также параметры обмена данными между системой Simulink и средой Matlab.

В Simulink используется два типа функций-решателей (методов численного интегрирования): с постоянным и с переменным шагом. При выборе первого типа пользователь может задать величину шага, которая и будет определять точность расчета.

Для методов с переменным шагом можно задать максимальное и минимальное значение шага, а также абсолютную и относительную точность вычислений. Методы с переменным шагом условно разделяются на методы решения «нежестких» задач и методы решения «жестких» задач.

В названии последних присутствует буква «s».

Все методы, доступные в Simulink, отличаются друг от друга точностью и скоростью вычислений. В качестве практической рекомендации по их выбору можно привести следующее.

Если модель не содержит разрывных функций (нежесткая задача) следует пользоваться методами ode45 или ode23 (методы Рунге-Кутты 4-5 и 2-3 порядков соответственно), в противном случае (жесткая задача) следует пользоваться методами ode15s и ode23s, специально разработанными для жестких систем.

И, наконец, если в модели возможны переключения с бесконечной теоретически скоростью (скользящие режимы), единственным решением будет применение методов с постоянным шагом, так как другие методы останавливают вычисления, бесконечно уменьшая шаг.

Подробнее о методах см. справочную системы Matlab.

Обмен данными между системой Simulink и средой Matlab позволяет формировать «внешнее управление» моделью из среды и выводить расчетные данные в среду для их дальнейшей обработки. При этом необходимо задать имена соответствующих переменных (вкладка Workspace I/O) и, в случае необходимости, режим внешнего управления, пометив пункт External меню Simulation

Запуск, приостановка расчета и досрочная остановка осуществляются с помощью кнопок панели инструментов окна модели. Просмотреть графики изменения переменных в окнах блоков Scope можно как после окончания процедуры вычисления, так и во время ее выполнения.

Модели и библиотеки Simulink хранятся в текстовых файлах специального формата с расширением .mdl. Запуск системы и открытие окна модели производится путем ввода имени модели в командной строке Matlab.

В качестве упражнения постройте модель системы

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = -x_1 - 2x_2 + k(y_{çàä} - x_1) + f, \\ y = x_1. \end{cases} \qquad (27)$$

и осуществите расчет ее реакции на ступенчатое изменение сначала сигнала $y_{зад}$, а потом сигнала f. Обратите внимание, что для этого необходимо настроить параметры блоков Step.

Окно модели показано на рис. 1.

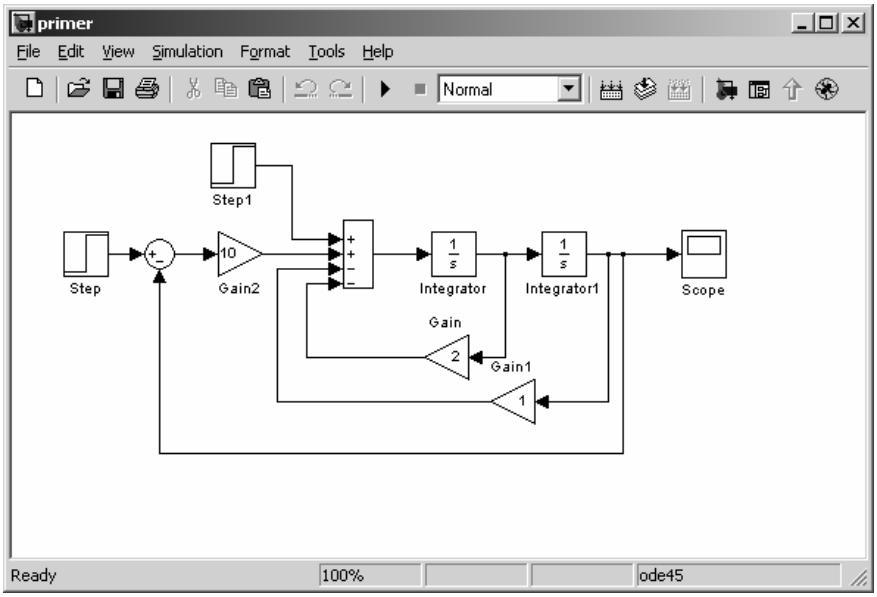

Рис. 1. Окно модели системы в Simulink.

Расчет производился при следующих параметрах:

Step: Step time = 0, Initial Value = 0, Final value = 1;

Step1: Step time = 10, Initial Value = 0, Final value = 1;

Simulation parameters: Start time =0, Stop time = 20, Solver Type – ode45,

Relative tolerance = 1e–6.

Результаты расчета приведены на рис. 2.

Рис. 2. Результаты моделирования в окне блока Scope.