

## 4 Символьные вычисления в среде MATLAB

В последней лабораторной работе мы рассмотрим использование возможностей символьных вычислений математического пакета MATLAB для решения задач вариационного исчисления. Ниже излагаются основные сведения о работе с символьными выражениями. Более детальную и полную информацию можно получить, воспользовавшись встроенной справочной системой MATLAB. Для этого достаточно выполнить команды *help symbolic* или *doc symbolic*.

### 4.1 Символьные выражения

#### 4.1.1 Объявление символьных переменных и констант

В процессе символьных вычислений используются переменные и константы особого типа, так называемые символьные объекты. Хотя обычно в коде MATLAB тип переменных определяется динамически и нет нужды объявлять его явно, для символьных объектов дело обстоит иначе. Для объявления символьных переменных служит команда *syms*, которая в качестве аргументов принимает имена переменных, перечисленные через пробел. Например, так:

```
>> syms x y
>> syms a b real % объявляемые объекты обозначают вещественные переменные
```

Объявление символьных констант осуществляется при помощи функции *sym*. Она может принимать в качестве аргумента строку, содержащую специальные переменные, численное выражение или вызов функции, как в примерах ниже:

```
>> sym_pi = sym('pi')
>> sym_delta = sym('1/10')
>> sym_sqrt2 = sym('sqrt(2)')
```

Использование символьных констант полезно тем, что вычисления с ними производятся точно (т.е. без вычислительных погрешностей) до тех пор, пока не потребуется вычислить некоторое числовое значение. Заметим, что при выводе содержимого рабочего пространства командой *whos* символьные переменные и константы отображаются как представители класса *sym object*.

#### 4.1.2 Символьные выражения и манипуляции над ними

После объявления, с символьными переменными можно обращаться примерно так же, как и с обычными числовыми. В частности, для них определены операторы  $+$   $-$   $*$   $/$   $^$ , с помощью которых можно составлять символьные выражения:

```
>> syms s t A
>> f = s^2 + 4*s + 5
f =
s^2 + 4*s + 5
>> g = s + 2
g =
s + 2
>> h = f*g
h =
(s^2 + 4*s + 5)*(s + 2)
>> z = exp(-s*t)
```

```

z =
exp(-s*t)
>> y = A*exp(-s*t)
y =
A*exp(-s*t)

```

В приведенных командах символьные переменные  $s$ ,  $t$ ,  $A$  используются для составления символьных выражений, создавая новые символьные переменные  $f$ ,  $g$ ,  $h$ ,  $z$ ,  $y$ . При этом последние автоматически объявляются символьными и их значение не вычисляется и никак не преобразуется.

При манипуляциях с символьными объектами часто бывает полезно узнать, какие *независимые* переменные содержатся в выражении, заданном строкой  $S$ . Для этой цели можно использовать функцию *findsym*. Например, продолжая пример выше, можно выполнить команду

```

>> findsym(z)
ans =
A, s, t

```

Заметим, что MATLAB всегда остается прежде всего матричным процессором и потому к символьным переменным можно свободно применять матричную и векторную запись и соответствующие встроенные операторы и функции. Приведем пример:

```

>> n = 3;
>> syms x;
>> B = x.^((0:n)')*(0:n))
B =
[ 1, 1, 1, 1]
[ 1, x, x^2, x^3]
[ 1, x^2, x^4, x^6]
[ 1, x^3, x^6, x^9]

```

Продemonстрируем некоторые простые алгебраические манипуляции, которые можно осуществлять над символьными объектами.

<code>expand(S)</code>	Раскрывает скобки в выражении $S$
<code>factor(S)</code>	Разлагает на множители выражение $S$
<code>simplify(S)</code>	Упрощает каждый элемент символьной матрицы $S$
<code>subs(S, oldvar, newvar)</code>	Заменяет в выражении $S$ каждое вхождение символической переменной $oldvar$ новой переменной $newvar$

Примеры использования можно увидеть ниже:

#### • Раскрытие скобок

```

>> syms s;
>> A = s + 2;
>> B = s + 3;
>> C = A*B
C =
(s+2)*(s+3)
>> C = expand(C)

```

```
C =
s^2 + 5*s + 6
```

- **Разложение на множители**

```
>> syms s;
>> D = s^2 + 6*s + 9;
>> D = factor(D)
D =
(s+3)^2
>> p = s^3 - 2*s^2 - 3*s + 10;
>> P = factor(p)
P =
(s+2)*(s^2 - 4*s + 5)
```

- **Сокращение общего множителя**

```
>> syms s;
>> H = (s^3 + 2*s^2 + 5*s + 10)/(s^2 + 5);
>> H = simplify(H)
H =
s+2
>> factor(s^3 + 2*s^2 + 5*s + 10)
ans =
(s+2)*(s^2 + 5)
```

- **Подстановка переменной**

Пусть имеется выражение  $H(s) = \frac{s+3}{s^2+6s+8}$  и требуется вычислить  $G(s) = H(s)|_{s=s+2}$ .

```
>> syms s;
>> H = (s + 3)/(s^2 + 6*s + 8);
>> G = subs(H, s, s+2)
G =
(s+5)/((s+2)^2 + 6*s + 20)
>> G = collect(G)
G =
(s+5)/(s^2 + 10*s + 24)
```

Таким образом,  $G(s) = \frac{s+5}{s^2+10s+24}$ .

Отметим, что конечный результат символьных преобразований далеко не всегда выглядит так, как хотелось бы пользователю и бывает неудобно работать с ним дальше.

## 4.2 Вычисление символьных выражений и отрисовка

Рано или поздно наступает момент, когда нам необходимо вычислить числовое значение результата выкладок или нарисовать график функции, выраженной символьным объектом. Рассмотрим, какие возможности для этого предоставляет нам MATLAB.

### 4.2.1 Вычисления значения выражения

`double(S)` Преобразует символьную матрицу  $S$ , заменяя ее элементы их численными значениями. Матрица  $S$  не должна содержать символьных переменных

Как следует из описания функции, в символьическом выражении перед вычислением его численного значения необходимо сделать все возможные подстановки, чтобы избавиться от свободных символьных переменных. Рассмотрим пример:

```
>> syms s;
>> E = s^3 -14*s^2 + 65*s - 100;
>> F = subs(E, s, 7.1)
F =
13671/1000
>> G = double(F)
G =
13.6710
```

### 4.2.2 Рисование при помощи функции *ezplot*

Символьическое выражение может быть нарисовано непосредственно при помощи функции *ezplot*:

`ezplot(f)` Рисует график функции  $f(x)$ , где  $f$  является математической функцией переменной  $x$ . Интервал, на котором изображается график, по умолчанию равен  $[-2\pi, 2\pi]$ .

`ezplot(f, xmin, xmax)` Рисует график на заданном интервале. Например, график кубического многочлена

$$A(s) = s^3 + 4s^2 - 7s - 10$$

на интервале  $[-1, 3]$  можно нарисовать следующим образом:

```
>> syms s;
>> A = s^3 + 4*s^2 - 7*s - 10;
>> ezplot(A, -1, 3), ylabel('A(s)')
```

Отметим, что название оси абсцисс и всего графика определяется автоматически. Они могут быть изменены обычным способом (см. команды *xlabel*, *title*).

### 4.2.3 Рисование при помощи функции *plot*

График функции можно построить и используя функцию *plot*:

`plot(xh, y, frmt)` Рисует график функции по точкам, координаты которых заданы массивами  $x$  и  $y$ . Параметр *frmt* определяет внешний вид графика.

В качестве примера мы нарисуем графики двух функций. Одна будет задана явно, другая - символьным выражением. Одновременно продемонстрируем некоторые возможности для настройки внешнего вида графика.

```
clear all; % Очищаем рабочее пространство
syms x y; % Определяем символьные переменные
z = (x^2 - y^2)/(x^2 + y^2); % Определяем функцию  $z(x, y) = \frac{x^2 - y^2}{x^2 + y^2}$ 
```

```

x1 = linspace(0, pi);           % Задаем массив абсцисс точек графика
y1 = sin(x);                    % Функция задается явным выражением M-кода
zy1 = subs(z, y, 1);           % Вычисляем  $z(x, 1) = \frac{x^2 - 1}{x^2 + 1}$ , график которой и нарисуем
z1 = subs(zy1, x, x1);          % Вычисляем ординаты точек

plot(x1, y1, 'b', x1, z1, 'r') % Рисуем графики обеих функций разным цветом
set(get(gcf, 'CurrentAxes'), 'FontSize', 10) % Меняем шрифт для надписей
title('\bf2D-график');          % Задаем заголовок графика
xlabel('\itx');                  % Задаем метку оси абсцисс
ylabel('\ity');                  % Задаем метку оси ординат

da = daspect;                   % Получаем текущий масштаб графика
da(1:2) = min(da(1:2));         % Выравниваем масштаб графика
daspect(da);                    % Устанавливаем новый масштаб
xlim([0, pi]);                  % Устанавливаем пределы по оси абсцисс

```

Отметим, что изменение параметров объектов при помощи функций *get*, *set* является типичным для MATLAB, особенно при работе с графикой. Примерно таким же образом можно нарисовать трехмерные графики (детали можно узнать в документации по функциям *plot3*, *mesh*, *surf*).

## 4.3 Решение уравнений и систем уравнений

### 4.3.1 Решение уравнений при помощи функции *solve*

MATLAB можно использовать для решения алгебраических и трансцендентных уравнений и систем уравнений, заданных в виде массива символьных выражений. Основным инструментом для этого - функция *solve*. Она может вызываться в разной форме:

```

solve(E1, E2, ..., EN)
solve(E1, E2, ..., EN, var1, var2, ..., varN)

```

Здесь  $E1, E2, \dots, EN$  — символьные выражения или переменные, в которых они содержатся, а  $var1, \dots, varN$  — переменные, относительно которых следует разрешить систему уравнений  $E1=0, E2=0, \dots, EN=0$ . Разумеется, первая форма вызова этой функции допустима лишь в случае, когда нет неоднозначностей относительно того, что именно следует найти. Функция *solve* возвращает единственное символьное выражение, если уравнение (система уравнений) имеет единственное решение и вектор решений в противном случае. Если уравнение содержит периодические функции и может поэтому иметь бесконечное число решений, функция ограничивается тем, что возвращает корни за один период в окрестности нуля.

Рассмотрим примеры.

- **Единственное решение**

```

>> syms s;
>> E = s + 2;
>> s = solve(E);
s =
-2

```

- **Несколько решений**

```
>> syms s;
>> D = s^2 + 6*s + 9;
>> s = solve(D)
s =
[ -3]
[ -3]
```

- **Уравнение с параметрами**

```
>> syms theta x z;
>> E = z*cos(theta) - x;
>> theta = solve(E, theta)
theta =
acos(x/z)
```

- **Уравнение с комплексными корнями**

```
>> syms x;
>> E = exp(2*x) + 4*exp(x) - 32;
>> x = solve(E)
x =
[ log(-8)]
[ log(4)]
>> log(-8)
ans =
2.0794 + 3.1416i
>> log(4)
ans =
1.3863
```

- **Уравнение с бесконечным числом решений**

```
>> E = cos(2*theta) - sin(theta);
>> solve(E)
theta =
[ -1/2*pi]
[ 1/6*pi]
[ 5/6*pi]
```

Напомним, что численное (т.е. приближенное) значение решения можно получить, воспользовавшись функцией *double*.

#### 4.3.2 Численное решение уравнений, заданных символьным выражением

Разумеется, не каждое уравнение можно решить аналитически. Для численного решения системы нелинейных уравнений (алгебраических или трансцендентных) можно воспользоваться функцией *fsolve*. Эта функция в самом простом случае принимает два аргумента. Первый — анонимная функция или имя функции (не математической, а в смысле М-языка), которая описывает правую часть системы уравнений, записанной в нормальной форме. Вторым аргументом задается начальное приближение для итерационной процедуры поиска решения.

Процедура численного решения часто является лишь небольшим блоком в программе, поэтому заранее неизвестно, как именно выглядит система уравнений. Простейший,

хотя и не самый красивый способ заключается в том, чтобы «на лету» создавать файл с описанием необходимой задачи и затем передавать его имя функции *fsolve*. Для нас этот способ интересен тем, что знакомит с работой с файловой системой в MATLAB и в дальнейшем, при решении задач вариационного исчисления, будет использоваться именно он.

Приведем пример, снабдив его исчерпывающими, на наш взгляд, комментариями. Итак, пусть необходимо решить систему уравнений вида

$$\begin{cases} r(\varphi - \sin \varphi) = 3 \\ r(1 - \cos \varphi) = 2 \end{cases}$$

Опишем необходимые действия:

```
clear all; % Очищаем память

eq1 = 'r*(phi - sin(phi)) - 3'; % Описываем первое уравнение
eq2 = 'r*(1 - cos(phi)) - 2'; % Описываем второе уравнение

% Начинаем создавать содержимое будущей функции

s{1} = 'function y = MyFunction(x)'; % Заголовок функции
s{2} = 'r = x(1); phi = x(2);'; % Определяем переменные
s{3} = ['y(1) = ' vectorize(eq1) '']; % Определяем результат
s{4} = ['y(2) = ' vectorize(eq2) '']; % выполнения функции

filename = fullfile(pwd, 'MyFunction.m'); % Определяем полное имя файла
fid = fopen(filename, 'w'); % Открываем файл на запись
fprintf(fid, '%s\n', s{:}); % Записываем содержимое
fclose(fid); % Закрываем файл

% Решаем уравнение численно и печатаем результат

xinit = ones(2,1); % Задаем начальное приближение
[xzero, yzero, eflag, opt] = fsolve('MyFunction', xinit); % Решаем
fprintf('Количество вычислений правой части: %d\n', opt.iterations);
fprintf('Найденное решение: \nr = %12.7f\nphi = %12.7f\n', xzero);
delete(filename); % Удаляем более не нужный файл

Советуем сравнить результат вычислений с результатами следующей команды:
>> solve(eq1, eq2, 'r,phi')
```

## 4.4 Элементы математического анализа

### 4.4.1 Дифференцирование

Чтобы найти производную символьного выражения, следует воспользоваться одной из следующих форм функции *diff*:

<code>diff(S)</code>	Дифференцирует выражение $S$ по независимым переменным, определенным функцией <i>findsym</i>
<code>diff(S, t)</code>	Дифференцирует выражение $S$ по переменной $t$
<code>diff(S,n)</code>	$n$ раз дифференцирует выражение $S$
<code>diff(S, t, n)</code>	$n$ раз дифференцирует выражение $S$ по переменной $t$

Приведем несколько простых примеров.

```
>> syms s n;
>> p = s^3 + 4*s^2 - 7*s - 10;
>> d = diff(p)
d =
3*s^2+8*s-7
>> e = diff(p, 2)
e =
6*s+8
>> g = s^n;
>> h = diff(g)
h =
s^n*n/s
>> h = simplify(h)
h =
s^(n-1)*n
>>
>> f = exp(-(x^2)/2);
>> diff(f)
ans =
-x*exp(-1/2*x^2)
```

#### 4.4.2 Интегрирование

Как и дифференцирование, символьное вычисление интегралов (как определенных, так и неопределенных) выполняется одной функцией *int*:

<code>int(S)</code>	Интегрирует выражение $S$ по независимым переменным, определенным функцией <i>findsym</i> . Если $S$ — константа, то интегрирование выполняется по $x$
<code>int(S, t)</code>	Интегрирует выражение $S$ по переменной $t$
<code>int(S, a, b)</code>	Вычисляет определенный интеграл на промежутке $[a, b]$
<code>int(S, t, a, b)</code>	Вычисляет определенный интеграл по указанной переменной

Приведем несколько примеров:

```
>> syms x n a b t
>> int(x^n)
ans =
```



```

x^(n+1)/(n+1)
>> int(x^3 + 4*x^2 + 7*x + 10)
ans =
1/4*x^4+4/3*x^3+7/2*x^2+10*x
>> int(x, 1, t)
ans =
1/2*t^2-1/2
>> int(x^3, a, b)
ans =
1/4*b^4-1/4*a^4
>> int(cos(x))
ans =
sin(x)
>> int(exp(-x^2))
ans =
1/2*pi^(1/2)*erf(x)

```

## 4.5 Решение дифференциальных уравнений

С решением обыкновенных дифференциальных уравнений ситуация обстоит примерно так же, как с решением алгебраических уравнений: в некоторых случаях их можно решить аналитически, но чаще приходится решать численно. Ниже рассмотрим, как это можно сделать.

### 4.5.1 Решение ОДУ при помощи функции *dsolve*

Функция *dsolve* может находить как общее решение ОДУ, так и частное решение для заданных начальных или граничных условий. При этом следует соблюдать определенные ограничения на форму записи уравнения (или системы уравнений). Так, если неизвестная функция обозначена символьической переменной  $y$ , то ее производные следует обозначать как  $D[n]y$ , где в квадратных скобках указан порядок производной. Таким образом, производная должна обозначаться в символьном выражении  $Dy$ , вторая производная —  $D2y$  и т.д. Функция *dsolve* может вызываться с разным набором параметров, в зависимости от типа решаемой задачи и порядка системы уравнений. Детали можно найти в документации, мы же ограничимся небольшим примером.

Пусть необходимо решить задачу Коши вида

$$y'' + 2y' + y = x \sin 2x, \quad y(0) = 1, y'(0) = -1.$$

Опишем необходимые для этого действия.

```

clear all;                                     % Очищаем память

eq = 'D2y + 2*Dy + y - x*sin(2*x)';          % Описываем уравнение
x0 = 0;                                         % Начальные условия
y0 = 1;
y10 = -1;

ic1 = sprintf('y(%d) = %d', x0, y0);          % Преобразуем их
ic2 = sprintf('Dy(%d) = %d', x0, y10);        % в символьные выражения

```

```

fprintf('Дифференциальное уравнение: \n%s=0\n', eq);
fprintf('Начальные условия: \n%s\n%s\n', ic1, ic2);

Sol = dsolve(eq, ic1, ic2, 'x');           % Решаем начальную задачу
if isempty(Sol)                             % Если решений нет, то ...
    disp('Решения не найдены');           % печатаем ответ
else                                         % В противном случае ...
    n = length(Sol);
    fprintf('Найдено решений: %d\n', n)
    for i = 1:n                             % печатаем все решения
        fprintf('Решение %d: \ny=%s\n', i, char(Sol(i)));
    end
end
end

```

#### 4.5.2 Численное решение уравнений, заданных символьным выражением

Численное решение ОДУ осуществляется в MATLAB целым набором различных функций, которые реализуют самые разные методы решения. Вообще говоря, используются они примерно так же, как мы до этого использовали *fsolve*: уравнение реализуется в виде функции, вычисляющей правую часть и эта функция передается в качестве аргумента так называемому «решателю» ОДУ (*ODE solver*). Кроме того, указывается ряд параметров и начальное условие для задачи Коши.

Чтобы продемонстрировать всю эту процедуру, продолжим предыдущий пример и решим рассматриваемую задачу Коши численно, воспользовавшись «решателем» *ode45*, который реализует метод Рунге-Кутты 4-5 порядка точности. Заметим, что численные методы решения в большинстве случаев требуют представления уравнения (системы уравнений) в виде системы первого порядка. Это потребует предварительных преобразований.

Код, приведенный ниже, следует понимать как продолжение предыдущего примера.

```

D2y = solve(eq, 'D2y');                    % Разрешаем относительно D2y
% Производим замену переменных
f2 = subs(D2y, {sym('y'), sym('Dy')}, {sym('y(1)'), sym('y(2)')});

% Начинаем создавать содержимое будущей функции

s{1} = 'function dydx = MyEquation(x)';     % Заголовок функции
s{2} = 'dydx = zeros(2,1);';               % Заготовка результата
s{3} = 'dydx(1) = y(2);';                  % Первое уравнение системы
s{4} = ['dydx(2) = ' char(f2) '];          % Второе уравнение системы

filename = fullfile(pwd, 'MyEquation.m');   % Определяем полное имя файла
fid = fopen(filename, 'w');                 % Открываем файл на запись
fprintf(fid, '%s\n', s{:});                 % Записываем содержимое
fclose(fid);                               % Закрываем файл

% Решаем уравнение численно

xfinish = 5;                               % Задаем конечную точку
xr = linspace(x0, xfinish, 20);            % Создаем массив абсцисс

```

```

yinit = [y0; y10]; % Задаем начальные условия
[xx, YY] = ode45('MyEquation', xr, yinit); % Решаем ОДУ

% Рисуем вначале решение, полученное аналитически при помощи dsolve

xp1 = linspace(x0, xfinish); % Создаем массив координат
yp1 = subs(Sol(1), sym('x'), xp1); % для отрисовки решения
plot(xp1, yp1, 'b'); % Рисуем график
hold on; % Замораживаем полотно графика

% Теперь рисуем график численного решения

plot(xx, YY(:,1), 'r'); % Рисуем численное решение по точкам
hold off; % Размораживаем полотно графика
title('\bfРешение задачи Коши'); % Задаем заголовок графика
xlabel('\itx'); % Задаем названия осей
ylabel('\ity'); %
xlim([x0, xfinish]); % Задаем пределы по оси абсцисс

delete(filename); % Удаляем не нужный более файл

```

## 4.6 Дополнительная литература

Для выполнения заданий лабораторной работы №4 мы рекомендуем ознакомиться с электронным онлайн-курсом [3], посвященным решению вариационных задач, главным образом, используя символьные вычисления в MATLAB. Этот материал, дополненный и углубленный, можно найти также в книге С.П. Иглина [2]. Для углубленного знакомства с возможностями пакета MATLAB можно порекомендовать как учебные пособия и справочные материалы на русском языке (например, [1], [5]), так и оригинальную документацию, которую можно найти на сайте [4] компании-производителя.

## Список литературы

- [1] Кетков Ю.Л., Кетков А.Ю., Шульц М.М. *MATLAB 7: программирование, численные методы*. — СПб.: БХВ-Петербург, 2005. — 752 с.
- [2] Иглин С.П. *Математические расчеты на базе MATLAB*. — СПб.: БХВ-Петербург, 2005. — 640 с.
- [3] <http://www.exponenta.ru/educat/systemat/iglin/2/index.asp>
- [4] <http://www.mathworks.com/help/techdoc/>
- [5] <http://matlab.exponenta.ru/index.php>