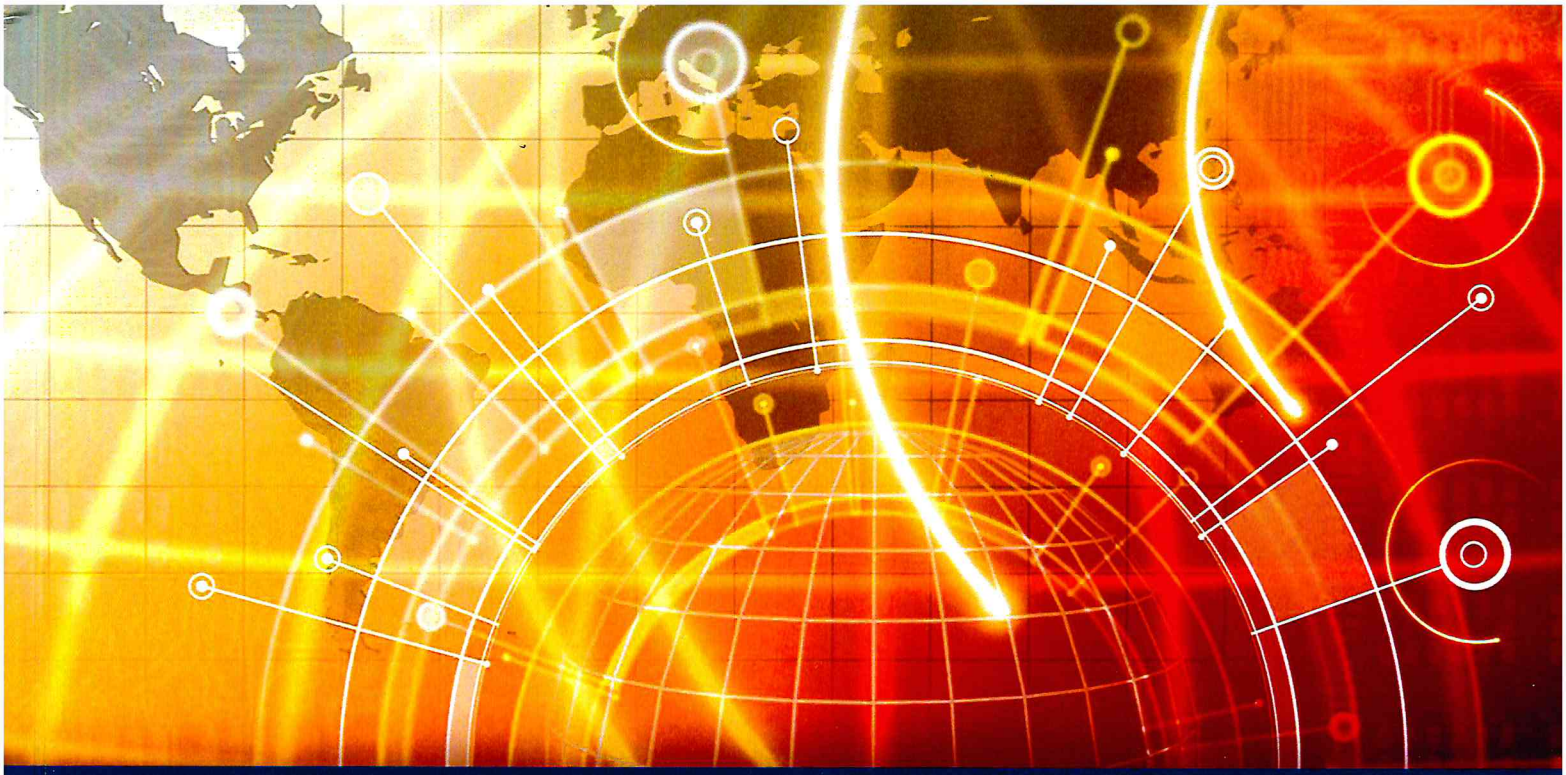

ИНФОРМАТИЗАЦИЯ ОБРАЗОВАНИЯ И НАУКИ

Государственная политика в области информатизации образования и науки

Информационные технологии

Телекоммуникации

Системы защиты информации

Автоматизация и управление технологическими процессами и производствами

Системный анализ, управление и обработка информации

Управление в социальных и экономических системах

Министерство образования и науки Российской Федерации

Informika

Федеральное государственное автономное учреждение «Государственный научно-исследовательский институт информационных технологий и телекоммуникаций»

№3[39]

ИЮЛЬ 2018

Научно-методический журнал
«Информатизация образования и науки»
№ 3(39) / 2018

Учредитель:
Федеральное государственное автономное учреждение «Государственный научно-исследовательский институт информационных технологий и телекоммуникаций»
(ФГАУ ГНИИ ИТТ «Информика»)
Министерства образования и науки Российской Федерации

Редакция:

Куракин Д.В.
Абдрахманова Э.Р.
Королева И.В.
Федорчук Е.В.

Тел. 8(495) 969-26-17 доб.1112

Журнал включен в Перечень ведущих рецензируемых научных журналов и изданий ВАК

Тираж журнала
500 экз.

Зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Свидетельство о регистрации средства массовой информации ПИ №ФС77-48849 от 7 марта 2012 г.)

Подписной индекс 32788
в каталоге «Газеты. Журналы»
ОАО Агентства «РОСПЕЧАТЬ»

Отпечатано в типографии
ФГАУ ГНИИ ИТТ «Информика»
Адрес: 125009, Москва,
Брюсов пер., д. 21

По вопросам редакционной подписки обращаться по адресу:
125315, Москва,
ул. Часовая, д. 21/Б, ком. 31

СОДЕРЖАНИЕ

ФУНКЦИИ MATLAB: ОБЯЗАТЕЛЬНЫЕ И НЕОБЯЗАТЕЛЬНЫЕ ПАРАМЕТРЫ

MATLAB FUNCTIONS: REQUIRED AND OPTIONAL PARAMETERS

Ревинская Ольга Геннадьевна / Olga G. Revinskaya,
кандидат педагогических наук, доцент, профессор Российской Академии Естествознания; доцент кафедры физики плазмы (Национальный исследовательский Томский государственный университет); зав. лабораторией (Национальный исследовательский Томский политехнический университет) / Candidate of Pedagogical Sciences, docent, Professor of the Russian Academy of Natural History; Associate Professor of the Department of Plasma Physics (National Research Tomsk State University);chief of laboratory (National Research Tomsk Polytechnic University),
ogr@tpu.ru

Осеева Анастасия Михайловна / Anastasia M. Oseeva,
студентка физического факультета (Национальный исследовательский Томский государственный университет) / student, National Research Tomsk State University)

Аннотация

MatLab, позиционируемый разработчиками как пакет прикладных программ, активно используется в различных научных и инженерно-технических разработках. Программные решения предлагаются потребителям в виде функций, многие из которых имеют большое количество параметров. Эффективность использования этих (стандартных) функций обеспечивается, в том числе и гибкостью использования их параметров, которые могут быть как обязательными, так и необязательными. В статье рассмотрены возможности, предоставляемые MatLab (в том числе и его последними версиями) программистам, для разработки нестандартных функций, которые поддерживали бы такой же гибкий механизм использования параметров, как и стандартные функции, поставляемые разработчиками.

Abstract

MatLab, positioned by developers as a package of applied programs, is actively used in various scientific and engineering developments. Software solutions are offered to consumers in the form of functions, many of which have a large number of parameters. The efficiency of using these (standard) functions is ensured, among other things, by the flexibility of using their parameters, which can be either required or optional. The article considers the possibilities provided by MatLab (including its latest versions) to programmers, for developing non-standard functions that would support the same flexible mechanism for using parameters, as well as the standard functions supplied by developers.

Ключевые слова: MatLab, стандартная функция, нестандартная функция, входные и выходные параметры, обязательные и необязательные параметры.

Keywords: MatLab, standard function, non-standard function, input and output parameters, required and optional parameters.

Введение

Подпрограммы в программировании являются одним из ведущих средств не только структурирования кода, но и распространения наиболее удачных его реализаций. Именно благодаря наличию большого количества библиотек, объединявших различные подпрограммы, в свое время Фортран был очень популярен в физических исследованиях. В настоящее время тенденция использования готовых программных решений в прикладных исследованиях становится еще более эффек-

тивной благодаря появлению и активному развитию математических пакетов: Mathematica, Mathcad, MatLab и т.д. Эти пакеты, как правило, сочетают в себе не только самостоятельный язык программирования, но и написанные на этом языке подпрограммы (процедуры, функции), воспроизводящие те или иные алгоритмы. Чем больше алгоритмов реализовано в виде хорошо отлаженных подпрограмм, тем реже при использовании математического пакета у пользователя будет возникать потребность в программировании как таковом. В этом случае решение большого количества прикладных задач, по сути, сводится к правильному использованию поставляемых разработчиком подпрограмм. Ярким примером развития этой тенденции в настоящее время можно считать MatLab, библиотеки которого активно используются при решении задач радиолокации [1], проектирования освещения [2], безопасности космического аппарата в полете [3], управления водным транспортом [4, 5], анализа инвестиций [6], движения грунтовых вод [7], теплообмена [8], моделирования плотности вещества экзопланет [9] и ряда других задач. Авторы этих и многих других публикаций широко используют поставляемые разработчиками MatLab подпрограммы. Некоторые авторы [7–14], используя MatLab, прибегают не только к использовании готовых подпрограмм, но и к созданию собственных. Большое количество публикаций, посвященных применению MatLab в исследованиях различной направленности, свидетельствует о высоком качестве поставляемых разработчиком подпрограмм. Но эффективность использования этих подпрограмм может снижаться за счет некорректности их вызова.

Анализ публикаций [3–6, 15–25] показывает, что многие поставляемые разработчиком подпрограммы MatLab в разных ситуациях можно вызывать поразному (с разными и по содержанию и по количеству параметрами). Это существенно расширяет область применения таких подпрограмм.

При этом следует отметить, что, несмотря на большое количество публикаций по применению MatLab, принципы написания и вызова подпрограмм (функций) излагаются в них очень кратко. В результате возникает диссонанс между изложенными в справочниках [26] и учебниках [27–29] правилами, которые рекомендуется программисту при разработке и вызове собственных (нестандартных) подпрограмм, и правилами вызова стандартных (поставляемых производителем) подпрограмм (функций), получивших широкое распространение. В частности, при анализе правила вызова различных стандартных функций легко заметить, что не все параметры (как входные, так и выходные), используемые при вызове функций, являются обязательными. Однако информацию, как сделать необязательными параметры нестандартной функции, можно найти только в справочной системе MatLab [30]. Учитывая, что понимание принципов разработки функций существенным образом сказывается на корректности их использования, детально рассмотрим несколько способов конструирования в MatLab таких функций с необязательными параметрами, выполнение которых не приводит к прерыванию программы и появлению сообщения об ошибке.

1. Общие правила написания функций в MatLab

Обычно в MatLab под функцией понимают любую подпрограмму без выделения среди них процедуры. Каждую подпрограмму, как правило, записывают в отдельном файле, а вызывают из другого файла (управляющей программы или функции) или из командного окна, если файл с функцией находится в папке, считающейся в MatLab текущей. Далее для определенности будем вызывать все рассматриваемые функции в командном окне (рис. 1). Такой подход является предпочтительным при обучении, т.к. позволяет в одном и том же окне видеть и команду вызова функции, и результат ее выполнения.

В процессе практического программирования, как правило, функции вызывают из другого m-файла (функции или управляющей программы). И старые, и новые версии MatLab накладывают немного ограничений на структуру файла, в котором может храниться функция:

function [<выходные параметры>] =

<идентификатор функции> (<входные параметры>)

% комментарии

<операторы – тело функции>

end

В этой структуре обязательными являются только служебные слова function, end и идентификатор функции. Название файла (с расширением m) должно совпадать с идентификатором функции.

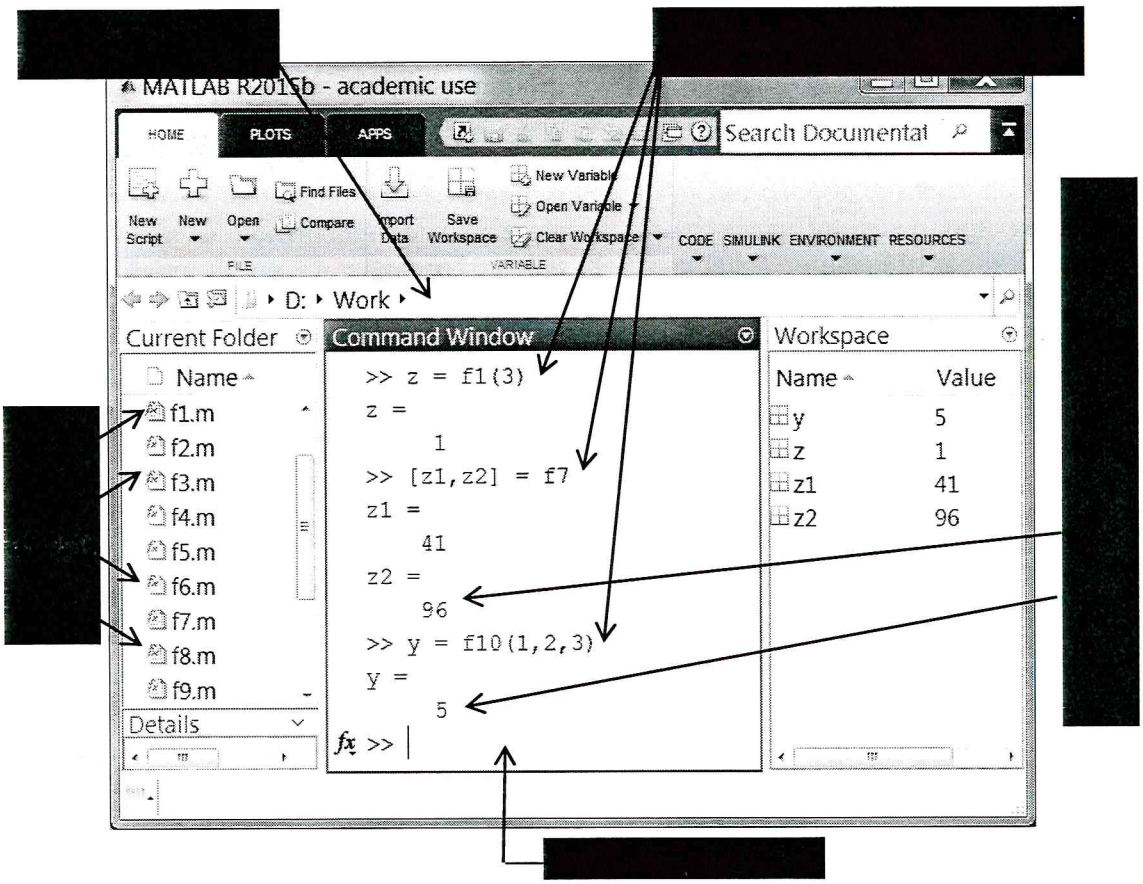

Рис. 1. Среда MatLab: вызов в командном окне нестандартных функций, хранящихся в текущей папке

Входные и выходные параметры перечисляются через запятую без указания типов (в виде списков идентификаторов). Их количество и последовательность определяется программистом, т.е. в MatLab можно написать функцию как с входными и (или) выходными параметрами, так и без них. Если при описании (написании) функции указаны несколько входных и несколько выходных параметров, обычно в справочниках и учебниках [26–29] говорится, что при вызове функции следует указать такое же количество параметров соответственно. Например,

если описание функции в m-файле имеет вид:

```
function y = f1( x )
y = 2*x-5;
end
```

то ее вызов в командном окне: >> z = f1(3)

Но возникают вопросы:

можно ли благополучно вызвать функцию, указав меньше (или больше) параметров, чем при ее описании (ведь именно так часто и поступают при вызове стандартных функций);

потребуется ли при этом каким-то специальным образом организовывать тело функции.

2. Функции со списком параметров фиксированной длины

Выходные параметры. Если выходные параметры – это список идентификаторов конечной длины, то в теле функции каждому параметру должно быть присвоено значение (хотя бы один раз), а количество выходных параметров, указанных при вызове функции, не должно превышать количество выходных параметров при ее описании. Если при вызове такой функции указать меньше выходных параметров, чем при описании, функция возвращает только часть сгенерированной ею информации (прерывание не происходит, сообщение об ошибке не появляется).

Например, описание функции в m-файле:

```
function [ y1,y2,y3,y4 ] = f2( x )
y1 = x; y2 = x^2; y3 = x^3; y4 = x^4;
end
```

Варианты ее вызов в командном окне:

>> [z1,z2,z3,z4] = f2(2) % возвращает все 4 значения,

>> [q1,q2] = f2(2) % возвращает только первые 2 значения

Возвращаемая информация распределяется по переменным в порядке их следования в списке выходных параметров, то есть, если при вызове указано меньше выходных параметров, чем при описании функции, то невостребованной остается информация, которая в теле функции помещена в один или несколько последних параметров. Если при вызове функции нет необходимости возвращать информацию, помещенную в теле функции в один из первых параметров, то при вызове функции на соответствующее место в списке результирующих параметров вместо переменной можно поставить символ ~.

Например:

>> [q1,~,q3] = f2(2) % возвращает только первое и третье значения

>> [~,~,q3] = f2(2) % возвращает только третье значение

Следует отметить, что функция, вызванная без выходных параметров, возвращает только значение первого параметра из списка конечной длины, указанного при описании.

Например, в результате вызова функции >> f2(2) получится только первое значение, которое автоматически присваивается служебной переменной ans.

Если при описании функции выходные параметры отсутствуют, такую функцию всегда следует вызывать без присвоения результатов каким-либо переменным (без выходных параметров).

Например, если описание функции в m-файле имеет вид:

```
function f3( x,y,z )
disp([x y z])
end
```

то ее вызов в командном окне может быть таким: >> f3(3,4,5)

В этом случае функция f3 не возвращает никаких значений.

Входные параметры в теле функции могут использоваться как исходные данные, изменяться или не использоваться вовсе. Это не приводит к появлению сообщения об ошибке. Однако если входные параметры – это список идентификаторов конечной длины, то превышение количества входных параметров, указанных при вызове функции, по сравнению с количеством этих параметров, перечисленных при ее описании, приводит к появлению сообщения об ошибке (Too many input arguments).

Технически MatLab позволяет при вызове функции, входные параметры которой описаны в виде списка идентификаторов конечной длины, указывать меньше входных параметров, чем при описании (символом ~ входные параметры заменять нельзя). Но тогда тело функции должно быть организовано так, чтобы при любом варианте вызова функции в ней в качестве исходных данных не использовались незаданные переменные, т.е. для корректной работы функция должна анализировать либо фактическое количество входных параметров, либо анализировать все ли входные параметры являются заданными. И в том, и в другом случае необходимо привлечение дополнительных возможностей MatLab, которые обычно в справочной и учебной литературе [26–29] не описываются, но будут рассмотрены далее. Без этих дополнительных возможностей примеров, когда функцию, объявленную со списком входных параметров конечной длины, вызывают с меньшим количеством входных параметров, очень мало, и они имеют смысл только для понимания языка программирования, но не имеют практической значимости.

Например, описание функции в m-файле:

```
function f4( x,y,z )
disp('test')
end
```

Варианты ее вызов в командном окне:

```
>> f4(3,4,5)
>> f4(3,4)
>> f4()
```

Каждый из этих вариантов приведет к благополучному вызову и завершению функции f4. Однако ни количество, ни значения входных параметров такой функции не влияют на ее работоспособность и результат, поэтому не имеют практического смысла. Подобную функцию следовало бы написать совсем без параметров. Однако рассмотренный пример полезен для понимания того, что MatLab не отслеживает, все ли входные параметры используются в теле функции. В то же время MatLab контролирует, чтобы всем параметрам, объявленным как выходные, в теле функции было присвоено значение хотя бы один раз.

3. Анализ параметров, указанных при вызове функции

Как было показано выше, даже если при объявлении функции указаны списки входных и выходных параметров конечной длины, технически возможно вызвать такую функцию с меньшим количеством параметров. Если предположить, что всем выходным параметрам присвоены значения, все входные параметры используются как исходные данные, то для обеспечения работоспособности такой функции с меньшим количеством параметров в ее теле необходимо анализировать количество (или существование) параметров, фактически указанных при вызове функции. Начиная с версии 2006, в MatLab включены стандартные функции nargin и nargout, которые возвращают фактическое количество входных и выходных параметров соответственно [30], а также функция exist, которая позволяет проверить, существует ли указанная переменная в памяти MatLab. Если функции nargin и (или) nargout вызываются в теле другой функции f без параметров, то каждая из них возвращает фактическое количество соответствующих параметров функции f, указанных при ее вызове.

Например, если в m-файле написана функция:

```
function [ z1,z2,z3,z4 ] = f5( x1,x2,x3,x4 )
z1 = randi(10); z2 = randi(10);
z3 = randi(10); z4 = randi(10);
disp(['Количество ВХОДНЫХ параметров:' int2str(nargin)])
disp(['Количество ВЫХОДНЫХ параметров:' int2str(nargout)])
end
```

то, вызывая эту функцию с разным количеством входных и выходных параметров, можно получить следующие результаты:

```
>> [z1,z2,z3,z4] = f5(5,4,3,2);
Количество  ВХОДНЫХ параметров: 4
Количество ВЫХОДНЫХ параметров:4
>> [z1] = f5(5,4,3,2);
Количество  ВХОДНЫХ параметров:4
Количество ВЫХОДНЫХ параметров:1
>> f5(5,4,3,2);
Количество  ВХОДНЫХ параметров:4
Количество ВЫХОДНЫХ параметров:0
>> [z1,z2,z3,z4] = f5(5,4,3);
Количество  ВХОДНЫХ параметров:3
Количество ВЫХОДНЫХ параметров:4
>> [z1,z2,z3,z4] = f5(5,4);
Количество  ВХОДНЫХ параметров:2
Количество ВЫХОДНЫХ параметров:4
>> [z1,z2,z3,z4] = f5();
Количество  ВХОДНЫХ параметров:0
Количество ВЫХОДНЫХ параметров:4
>> [~,~,z3,z4] = f5();
Количество  ВХОДНЫХ параметров:0
Количество ВЫХОДНЫХ параметров:4
```

Из приведенных результатов видно, что выходные параметры, задаваемые при вызове функции в виде ~, также считаются фактически запрашиваемыми.

При анализе значений, возвращаемых функциями nargin и (или) nargout, можно избежать обращения к фактически незаданным входным параметрам и не вычислять незапрашиваемые результирующие параметры. Далее приведен пример использования функции nargin для написания функции f6, которую можно вызывать с разным количеством входных параметров:

```
function [ y ] = f6( x1, x2, x3, x4 )
y = 0;
if nargin>0, y = y+x1; end
if nargin>1, y = y+x2; end
if nargin>2, y = y+x3; end
if nargin>3, y = y+x4; end
if nargin>0, y = y/nargin; end
end
```

При этом считается, что если функция nargin возвращает 3 для функции, в описании которой список входных параметров имеет длину 4, то незаданным может быть только последний из параметров и т.д.

Тогда в командном окне можно получить следующие результаты:

```
>> z = f6(6,4,2,16)
z =
7
>> z = f6(6,4,2)
z =
4
>> z = f6(6,4)
z =
5
>> z = f6(6)
z =
6
>> z = f6()
z =
0
```

Функцию f6 с той же функциональностью можно написать, используя exist вместо nargin. Функция exist возвращает код объекта, идентификатор которого указан в качестве ее первого входного

параметра. Идентификатор необходимо указывать в строковом виде. С помощью второго входного параметра необходимо указать, какого типа объект функция exist будет искать в памяти MatLab. Если необходимо проверить существует ли переменная, в качестве второго параметра этой функции используют 'var'. Если указанная переменная существует, то функция exist возвращает 1 (иначе – 0). Тогда функцию f6 можно записать в виде:

```
function [ y ] = f6( x1, x2, x3, x4 )
y = 0; n = 0;
if exist('x1', 'var') == 1, y = y+x1; n = n +1; end
if exist('x2', 'var') == 1, y = y+x2; n = n +1; end
if exist('x3', 'var') == 1, y = y+x3; n = n +1; end
if exist('x4', 'var') == 1, y = y+x4; n = n +1; end
if n>0, y = y/n; end
end
```

Следует отметить, что даже такую простую функцию как f6 без проверки значения nargin и exist реализовать не удастся, так как не всегда (не для всякого количества аргументов) значения всех входных параметров существуют. MatLab инициализирует переменную только тогда, когда ей сопоставлено какое-то значение, и только такие переменные могут использоваться в функции в качестве начальных данных для получения результирующих значений. Если при вызове функции входной параметр не задан, то его идентификатор не может участвовать в вычислениях и других операторах в смысле исходных данных (появится сообщение об ошибке). Поэтому, чтобы избежать использования фактически не инициализированных переменных в функциях, необходимо проверять либо значение nargin, либо существование в памяти MatLab (с помощью функции exist) каждой из переменных, которыми обозначены входные параметры функции.

Используя проверку значения nargout, можно написать функцию f7, которую можно вызывать с разным количеством выходных параметров.

Например:

```
function [ z1,z2,z3,z4 ] = f7
if nargout>0, z1 = randi(50); end
if nargout>1, z2 = 50 + randi(50); end
if nargout>2, z3 = 100 + randi(50); end
if nargout>3, z4 = 150 + randi(50); end
end
```

Как и для входных параметров, незапрашиваемыми считаются выходные параметры, расположенные в конце списка.

Следует отметить, что функцию f7 можно написать и не проверяя значение nargout, и она будет иметь ту же функциональность:

```
function [ z1,z2,z3,z4 ] = f7
z1 = randi(50);
z2 = 50 + randi(50);
z3 = 100 + randi(50);
z4 = 150 + randi(50);
end
```

Но здесь каждый выходной параметр вычисляется независимо от того, есть ли в этом фактическая необходимость. Если получение каждого из выходных параметров занимает длительное время, то использование проверки значения nargout поможет избежать невостребованных расчетов и сократить время работы программы, в которой соответствующая функция вызывается с разным количеством результирующих (выходных) параметров.

Функцией exist для проверки, является ли данный выходной параметр реально запрашиваемым, воспользоваться невозможно.

Таким образом, в MatLab даже при описании входных и выходных параметров нестандартной функции в виде списка идентификаторов конечной длины такую функцию можно вызывать с меньшим количеством параметров, чем указано при описании: в теле функции уделить внимание проверке на соответствие входных параметров фактически заданным, а каждый из выходных параметров – реально запрашиваемым.

4. Функции со списком параметров переменной длины

MatLab позволяет при описании параметров нестандартных функций использовать не только списки фиксированной длины, но и списки переменной длины. Для объявления входных и выходных параметров в виде списков переменной длины используют стандартные переменные varargin и varargout, соответственно [30]. Каждая из этих переменных является одномерным массивом ячеек (массивом, в котором одновременно могут храниться данные разного типа: числа, строки, числовые массивы и т.д.). Массивы ячеек varargin и varargout заполняются только после вызова функции, в объявлении которой они использованы. Для обращения к элементу массива ячеек varargin или varargout используют фигурные скобки: varargin{k} или varargout{k}. Фактическое количество входных и выходных параметров в теле функции можно узнать либо с помощью функций nargin и nargout (как при рассмотренном ранее использовании списков фиксированной длины), либо с помощью функции length(varargin) и length(varargout), возвращающей длину соответствующего массива.

Так, используя переменную varargin, функцию f6 можно записать следующим образом:

```
function [ y ] = f6(varargin)
y = 0;
for k = 1:nargin, y = y+ varargin{k};
end
if nargin>0, y = y/nargin; end
end
```

А используя переменную varargout, функцию f7 можно записать в виде:

```
function [varargout] = f7
for k = 1:nargout
varargout{k} = (k-1)*50 + randi(50);
end
end
```

Таким образом, использование переменных varargin и (или) varargout для обозначения списков входных и выходных параметров позволяет в теле функции организовывать циклы с этими параметрами (если с параметрами выполняются однотипные операции), а также вызывать эти функции не только с меньшим, но и с произвольным количеством параметров. Например, если функции f6 и f7 описаны последним из приведенных выше способов, то каждый из следующих вызовов этих функций будет успешным:

```
>> z = f6(6,4,2)
>> z = f6()
>> z = f6(6,4,2,16,7)
>> [z1,z2,z3,z4] = f7
>> [z1,z2,z3,z4,z5] = f7
>> [~,z2,~,z4] = f7
>> [z1] = f7
>> f7
```

Следует отметить, что при таком варианте написания функция f6, вызванная без входных параметров z = f6(), возвращает 0, а функция f7, вызванная без выходных параметров f7, ничего не возвращает. Ни то, ни другое не приводит к прерыванию и появлению сообщения об ошибке.

В приведенных примерах входные параметры функции f6 и выходные параметры функции f7 обрабатываются и генерируются по одному и тому же принципу, сколько бы их ни было. В этих случаях вместо массивов ячеек varargin и (или) varargout иногда можно использовать одномерные массивы однотипных элементов. Тогда при описании функции f6 достаточно указать один входной параметр X, подразумевая, что он массив. Количество элементов в этом массиве можно определить с помощью length(X). То есть функцию f6 можно объявить следующим образом:

```
function [ y ] = f6(X)
y = 0;
for k = 1:length(X), y = y+ X(k); end
if length(X)>0, y = y/length(X); end
end
```

При вызове такой функции входные параметры необходимо записывать в квадратных скобках через запятую:

```
>> z = f6([6,4,2,16])
>> z = f6([6])
>> z = f6([])
```

В качестве выходного параметра функции f7 также можно указать одно-

мерный массив и генерировать его элементы, но в этом случае потребуется использовать входной параметр для управления длиной генерируемого массива (т.к. определить длину несуществующего массива с помощью length(Z) невозможно). Тогда функцию f7 можно объявить следующим образом:

```
function [Z] = f7(n)
Z = [];
for k = 1:n, Z(k) = (k-1)*50 + randi(50); end
end
```

Вызывать такую функцию придется иначе, чем предыдущие ее варианты:

```
>> Z = f7(5)
>> Z = f7(2)
```

Кроме того, такая функция всегда возвращает все сгенерированные элементы: невозможно запросить, например, только последние сгенерированные элементы массива Z и игнорировать первые.

Таким образом, даже для списка однотипных параметров переменной длины (входных и выходных) при программировании тела функции использование массивов ячеек varargin и varargout дает более широкие возможности, чем использование массивов однотипных элементов.

5. Ограничения на количество и тип параметров функции

Как было показано ранее, если параметры функции описаны в виде списка фиксированной длины, MatLab анализирует количество фактически заданных параметров и не допускает только превышения указанного в списке количества параметров. Если параметры функции описаны с помощью переменных varargin и (или) varargout, то по умолчанию MatLab допускает использование любого количества фактических параметров без ограничений. Однако алгоритм обработки данных, оформленный в виде функции, сам может накладывать ограничения на количество входных и (или) выходных параметров. Например, некоторые алгоритмы не могут быть реализованы, если не задано некоторое минимально необходимое количество входных параметров. Алгоритм может предусматривать ограничения по максимальному количеству входных или выходных параметров. Для многих алгоритмов входные параметры должны содержать данные определенного типа. Следовательно, в теле функций, реализующих такие алгоритмы, необходимо проверять тип входных параметров, а также соответствие количества входных и выходных параметров, указанных при фактическом вызове таких функций, определенному диапазону значений.

В 2011 г. в MatLab появилась функция narginchk [30], прерывающая работу функции, если фактическое количество входных параметров находится вне указанного диапазона (меньше минимально допустимого или больше максимально допустимого). Если количество входных параметров соответствует указанному диапазону, функция narginchk ничего не делает. Функция nargoutchk, выполняющая аналогичную проверку для выходных параметров, существует в MatLab еще с 2006 г. И для той и для другой функции диапазон, которому должно соответствовать количество входных или выходных параметров, задается двумя целыми числами (сначала нижняя граница, затем верхняя). Если верхняя и нижняя границы – одинаковые числа, то функцию, в теле которой использована narginchk и (или) nargoutchk, можно успешно вызывать только с фиксированным количеством соответствующих параметров; если нижняя граница равна 0, то функцию можно вызывать как с параметрами, так и без параметров; если верхняя граница равна inf (бесконечность), то максимальное количество соответствующих параметров такой функции не регламентировано.

Например:

```
function [varargout] = f8
nargoutchk(4,4)
for k = 1:4
varargout{k} = (k-1)*50 + randi(50);
end
end
```

При вызове этой функции f8 всегда следует указывать 4 выходных параметра (все 4 параметра являются обязательны-

ми), но при необходимости некоторые из них можно заменять символом ~:

```
>> [a,b,c,d] = f8
>> [~,b,~,d] = f8
```

При вызове функции f8 с другим количеством параметров появится сообщение об ошибке: либо «Not enough output arguments», либо «Too many output arguments», соответственно.

Следующую функцию f9:

```
function f9( varargin )
narginchk(2,4)
disp('Обязательный параметр:'), disp(varargin{1})
disp('Обязательный параметр:'), disp(varargin{2})
 if nargin>2
disp('Необязательный параметр:'), disp(varargin{3})
end
if nargin>3
disp('Необязательный параметр:'), disp(varargin{4})
 end
end
```

можно успешно вызывать либо с двумя, либо с тремя, либо с четырьмя входными параметрами (в остальных случаях будет генерироваться сообщение об ошибке):

```
>> f9(9,23,17,31)
>> f9(9,23,17)
>> f9(9,23)
```

Результаты, полученные функцией f9, легко предсказать, поэтому не имеет смысла их приводить.

Здесь первые два входных параметра функции являются обязательными (их нужно указывать при любом варианте вызова функции), а вторые два – необязательными. Для необязательных входных параметров можно задать значения по умолчанию. Например:

```
function y = f10( varargin )
 narginchk(1,3)
 x = varargin{1};
 a = 5; b = 10; % значения по умолчанию
 if nargin>1, a = varargin{2}; end
 if nargin>2, b = varargin{3}; end
 y = a*x + b;
end
```

Если обязательный для функции f10 первый входной параметр задать равным 1, а остальные необязательные параметры оставить не заданными, то в теле функции f10 будут использованы значения переменных a и b, заданные по умолчанию (5 и 10, соответственно):

```
>> q = f10(1)
q =
15
```

Если функцию f10 вызвать с тремя параметрами f10(1,2,3), то в теле функции переменным a и b будут присвоены значения 2 и 3, соответственно:

```
>> z = f10(1,2,3)
z =
5
```

Аналогичную функцию можно написать со списком входных параметров конечной длины:

```
function y = f10( x,a,b )
narginchk(1,3)
% значения по умолчанию
if nargin<2, a = 5; end
if nargin<3, b = 10; end
y = a*x + b;
end
```

Здесь значения по умолчанию присваиваются переменным a и b, если соответствующие входные параметры функции f10 не заданы. Оба приведенные варианта функции f10 вызываются одинаково и возвращают одинаковые результаты. И в том, и в другом варианте функция narginchk(1,3) обеспечивает контроль за фактическим количеством входных параметров функции f10 и разделяет эти параметры на обязательные и необязательные.

Большинство нестандартных функций, создаваемых программистами, предназначено для работы с входными параметрами, количество которых может варьироваться только в узком диапазоне. Следует признать нужным (а порой и обязательным) при разработке таких функций использование narginchk в их теле.

Кроме контроля за необходимым и достаточным количеством входных параметров нестандартной функции для большинства алгоритмов, реализуемых в них,

критичным является передача каждым входным параметром информации о функции определенного типа. По умолчанию MatLab не контролирует ни тип выходных, ни тип входных параметров. Поэтому некоторые функции можно одинаково успешно вызывать с фактическими параметрами самых разных типов. Например, в качестве входных параметров функции f9 можно использовать числовые скаляры >> f9(9,23,17), числовые массивы >> f9([9,1], [1,23;7,0], [17;99;35]), символьные массивы и строки >> f9('9', '23', '17') и т.д., а также их комбинации >> f9(9, '23', [17,3,5]). Учитывая свойства операторов, использованных в теле функции f9, можно утверждать, что все приведенные здесь вызовы функции f9 будут успешными.

Например:

```
>> f9(9, '23', [17,3,5])
Обязательный параметр:
9
Обязательный параметр:
23
Необязательный параметр:
17   3   5
```

Однако на практике такие функции, выполняющие универсальные действия с входными параметрами любого типа, встречаются крайне редко. Для большинства реализуемых в нестандартных функциях алгоритмов исходные данные, которые, как правило, передаются в функцию через входные параметры, должны относиться к определенным типам. Так как MatLab по умолчанию не проверяет типы входных параметров, для того чтобы гарантировать работоспособность реализуемого алгоритма, эту проверку необходимо организовать в теле функции, написав соответствующий программный код. Для этого MatLab предлагает большой набор функций [31], каждая из которых проверяет, относится ли информация, хранящаяся в указанной переменной, к определенному типу. Каждая из этих функций возвращает логическую 1, если проверка прошла успешно, или 0, если анализируемая переменная имеет любой другой тип кроме проверяемого. Например, функции isscalar(x), isvector(x), ismatrix(x), iscell(x) проверяют, является ли переменная x скаляром, одномерным массивом, двумерным массивом, массивом ячеек, соответственно; функции isnumeric(x), ischar(x), isstring(x) – является ли переменная x числом (или числовым массивом), символом, строкой соответственно; функции isinteger(x), isfloat(x), isreal(x), islogical(x) – хранится ли в переменной x целое, вещественное или логическое число.

Используя эти и подобные им функции проверки [31], можно организовать прерывание работы нестандартной функции, если фактическое значение какого-либо из входных параметров нельзя отнести к определенному типу. Это позволит избежать выполнения реализованного в нестандартной функции алгоритма с некорректными данными. Например, рассмотрим функцию f11, входные параметры которой являются списком идентификаторов фиксированной длины:

```
function r = f11( x,y )
narginchk(2,2)
if isscalar(x) && isnumeric(x)
if isscalar(y) && isnumeric(y)
r = sqrt(x^2+y^2);
else
error('2-й параметр функции f11 '...
'должен быть числовым скаляром')
end
else
error('1-й параметр функции f11 '...
'должен быть числовым скаляром')
end
end
```

Тогда, указав при вызове функции f11 два числовых значения (скаляра), можно получить числовой результат. Если вместо хотя бы одного из входных параметров поставить не число или не скаляр, выполнение функции прервется и появится сообщение об ошибке:

```
>> z = f11(3,4)
z =
5
>> z = f11(3,[4,2])
Error using f11 (line 7)
```

2-й параметр функции f11 должен быть числовым скаляром

```
>> z = f11('3',4)
Error using f11 (line 10)
```

1-й параметр функции f11 должен быть числовым скаляром

Аналогичным образом можно написать функцию f12, входные параметры которой являются массивом ячеек переменной длины:

```
function r = f12( varargin )
narginchk(1,5)
r = 0;
for k = 1:nargin
x = varargin{k};
if isscalar(x) && isnumeric(x)
r = r+x^2;
else
error([int2str(k) '-й параметр функции f11 '...
'должен быть числовым скаляром'])
end
end
r = sqrt(r);
end
```

Вызывая функцию f12 с числовыми и нечисловыми входными параметрами, можно получить следующие результаты:

```
>> z = f12(3,4,5,5,5)
z =
10
>> z = f12(3,4,5,'5',5)
Error using f12 (line 9)
```

4-й параметр функции f11 должен быть числовым скаляром.

Для генерации сообщений об ошибках в телах функций f11 и f12 использована стандартная функция error, которая прерывает работу соответствующей функции и выводит сообщение, текст которого является обязательным параметром функции error.

Таким образом, ограничение количества обязательных и возможных входных и выходных параметров, а также проверка корректности типа данных, передаваемых в функцию с помощью входных параметров, существенно облегчает программирование нестандартных функций с переменным количеством параметров, повышает надежность их использования.

6. Функции с опциями

Во всех рассмотренных выше видах нестандартных функций с переменным и постоянным количеством параметров фактические значения входных параметров интерпретировались каждой из функций в порядке их следования. Чаще всего информация, поступившая в функцию через первый параметр, преобразуется одним способом, а информация, поступившая через второй параметр, – другим и т.д. Благодаря такому механизму интерпретации данных, выполняемой слева направо, нет необходимости помимо значения передавать в функцию какой-либо идентификатор, указывающий на способ интерпретации поступившей информации. При работе с функциями это позволяет экономить память, но требует от программиста тщательного соблюдения порядка следования параметров при вызове функций (и стандартных, и нестандартных) с учетом способа их использования в телах этих функций. Такие параметры называются позиционированными. Именно поэтому незаданными могут быть только входные позиционированные параметры, расположенные в конце списка.

Опциями называются такие входные параметры функции (подпрограммы), для которых всегда существуют значения, заданные по умолчанию. Поэтому при вызове функции некоторые (или все) из этих параметров можно опустить. Если опций несколько и для какой-то из них нужно задать значение, отличное от значения по умолчанию, то принцип последовательной интерпретации входных параметров требует, чтобы заданными явным образом (не по умолчанию) были все параметры, расположенные в списке перед этой опцией, то есть для опций, расположенных в списке входных параметров раньше, теряется смысл существования значения по умолчанию, если они позиционированы. Чтобы при вызове функции была возможность явным образом указывать только те значения опций, которые отличаются от заданных по умолчанию, необходимо, чтобы значения опций при вызове функции можно было указывать в любом порядке

(непозиционированные параметры). Но тогда для правильной идентификации передаваемых в функцию данных необходимо каждое значение сопровождать информацией, к какой именно опции оно относится. Поэтому в MatLab при вызове стандартных и нестандартных функций опции задаются двумя параметрами: 1) уникальный для данной функции идентификатор опции в строковом формате; 2) значение опции. Например, при вызове широко используемой стандартной функции plot(X,Y, 'Color', [1 0 0]) параметры X и Y интерпретируются по их положению в списке параметров, а для опции с названием 'Color' указано значение [1 0 0], отличное от заданного по умолчанию. Остальные опции функции plot в этом примере остались заданными по умолчанию.

Чтобы реализовать непозиционированный подход к вызову функций с опциями, необходимо в теле функции не только задать значения всех опций по умолчанию, но и организовать в списке фактически переданных функции параметров поиск названия каждой опции с последующим присвоением сопутствующего ему переданного значения.

При объявлении функций с опциями чаще всего для опций используют переменную varargin, а для остальных входных параметров – список идентификаторов фиксированной длины, значения в котором интерпретируются по их расположению в списке – позиционированные параметры. Чтобы интерпретация параметров была корректной, переменную varargin всегда располагают после списка идентификаторов фиксированной длины, все (или несколько первых) элементы которого считаются обязательными параметрами при вызове такой функции.

Например, рассмотрим функцию f13:

```
function y = f13(x, varargin)
narginchk(1,7)
% значения опций по умолчанию
a = 0; b = 1; c = 0;
% присвоение фактически заданных значений опций
if nargin>1
for k = 1:2:(length(varargin)-1)
if varargin{k}=='a', a = varargin{k+1}; end
if varargin{k}=='b', b = varargin{k+1}; end
if varargin{k}=='c', c = varargin{k+1}; end
end
end
% получение возвращаемого значения
y = a*x^2 + b*x + c;
end
```

Эта функция имеет один обязательный входной параметр (x) и три опции ('a', 'b' и 'c'). Функцию f13 можно вызывать как с опциями (одной, двумя или тремя), так и без них:

```
>> q = f13(5)
q =
5
>> q = f13(5,'a',1)
q =
30
>> q = f13(5,'b',2)
q =
10
>> q = f13(5,'c',3)
q =
8
```

Даже при вызове такой простой функции, как f13, опции можно задавать в любом порядке:

```
>> q = f13(5,'a',1,'c',3)
q =
33
>> q = f13(5,'c',3,'a',1)
q =
33
>> q = f13(5,'c',3,'a',1,'b',2)
q =
38
```

Объявление функций с опциями требует не только корректной идентификации этих параметров, но и также как для позиционированных параметров контроля правильности типа данных, передаваемых с помощью соответствующей опции. Это значительно усложняет предварительный анализ и разбор данных, поступивших че-

рез входные параметры функции. Для облегчения этого процесса в MatLab [30], начиная с версии 2007, появились функции parse, inputParser, addRequired, addOptional, addParamValue (начиная с версии 2013b, функция addParamValue была заменена функцией addParameter с тем же синтаксисом). Функция parse выполняет разбор и анализ некоторых данных в соответствии с заданной структурой. Остальные функции позволяют создать и детализировать эту структуру:

inputParser – создает пустую структуру параметров;

addRequired добавляет один обязательный позиционированный, addOptional – необязательный позиционированный параметр, addParameter – опцию (необязательный непозиционированный параметр).

Для каждого из элементов создаваемой структуры можно с помощью соответствующей функции (addRequired, addOptional или addParameter) указать идентификатор, значение по умолчанию (для необязательных параметров) и логическую функцию, которую необходимо использовать для проверки корректности типов фактических значений параметров. Например, если функция имеет один обязательный позиционированный параметр (числовой скаляр) и три опции, необходимо сначала создать пустую структуру параметров:

```
ParametrStruct = inputParser;
```

После этого добавить к этой структуре ParametrStruct обязательный параметр x, не имеющий значения по умолчанию:

```
addRequired (ParametrStruct, 'x')
```

Если этот параметр должен быть числовым скаляром, то логическую функцию проверки типа данных, сопоставляемых данному параметру, можно записать в виде безымянной @-функции, объединяющей isnumeric и isscalar, в качестве третьего параметра функции addRequired:

```
addRequired (ParametrStruct, 'x', ...
@(x) isnumeric(x) && isscalar(x))
```

Если логическая @-функция не указана, то проверка типа данных далее при разборе фактических параметров осуществляться не будет.

Чтобы к структуре ParametrStruct добавить опцию, используют (для версии MatLab 2013b и далее) функцию addParameter с указанием строкового идентификатора опции ('a'), логической @-функции (при необходимости) и значения по умолчанию (0):

```
addParameter (ParametrStruct, 'a', 0)
```

или

```
addParameter (ParametrStruct, 'a', 0, ...
@(x) isnumeric(x) && isscalar(x))
```

Остальные опции добавляют в структуру ParametrStruct аналогичным образом. После того как структура полностью сформирована ее можно использовать для разбора и анализа фактических параметров объявляемой нестандартной функции с помощью стандартной функции parse, подставив идентификатор структуры ParametrStruct в качестве ее первого параметра. В качестве второго и последующих параметров функции parse необходимо указать все входные параметры объявляемой нестандартной функции.

Например:

```
function y = f14(x, varargin)
narginchk(1,7)
% создание структуры параметров
ParametrStruct = inputParser;
addRequired(ParametrStruct, 'x')
% для версии MatLab 2013b и позднее
addParameter(ParametrStruct, 'a', 0)
addParameter(ParametrStruct, 'b', 1)
addParameter(ParametrStruct, 'c', 0)
% анализ и разбор параметров
parse(ParametrStruct, x, varargin{:})
% получение возвращаемого значения
y = ParametrStruct.Results.a*ParametrStruct.Results.x^2 + ...
ParametrStruct.Results.b* ParametrStruct.Results.x + ...
ParametrStruct.Results.c;
end
```

Следует отметить, что в большинстве случаев для корректного разбора при

вызове функции parse после идентификатора структуры ParametrStruct следует через запятую перечислить все формальные параметры функции f14 в том же порядке, что и в заголовке (x, varargin), но не в виде массива ячеек, а виде списка, т.е.: x, varargin{:}. Значения всех фактически переданных в функцию f14 параметров, успешно проанализированные функцией parse, помещаются в поле Results созданной структуры ParametrStruct. Для каждого параметра в поле Results создается отдельное поле, название которого совпадает с идентификатором параметра, указанным при формировании структуры.

Например, фактическое значение опции 'b' хранится в поле ParametrStruct.Results.b. Кроме поля Results структура ParametrStruct содержит еще несколько полей, в которых сохраняется остальная информация о входных параметрах (в частности, значения по умолчанию и т.д.).

Приведенный выше вариант функции f14 работает и вызывается также как и функция f13. При анализе и разборе фактических параметров аналогично функции f13 проверка корректности данных не выполняется. Но, если это необходимо сделать, то для этого в функции f14 при формировании структуры достаточно только указать соответствующие логические функции:

```
function y = f14(x, varargin)
narginchk(1,7)
% создание структуры параметров
ParametrStruct = inputParser;
addRequired(ParametrStruct, 'x',
@(x) isnumeric(x) && isscalar(x))
% для версии MatLab 2013b и позднее
addParameter(ParametrStruct, 'a', 0, ...
@(x) isnumeric(x) && isscalar(x))
addParameter(ParametrStruct, 'b', 1, ...
@(x) isnumeric(x) && isscalar(x))
addParameter(ParametrStruct, 'c', 0, ...
@(x) isnumeric(x) && isscalar(x))
% анализ и разбор параметров
parse(ParametrStruct, x, varargin{:})
% получение возвращаемого значения
y = ParametrStruct.Results.a*ParametrStruct.Results.x^2 + ...
ParametrStruct.Results.b*ParametrStruct.Results.x + ...
ParametrStruct.Results.c;
end
```

Тогда при попытке вызвать функцию f14 со строковым или нескалярным значением опции 'b' появится следующее сообщение:

```
>> q = f14(5,'b','2')
Error using f14 (line 33)
Argument 'b' failed validation @(x)isnumeric(x)&&isscalar(x).
>> q = f14(5,'b',[2 4])
Error using f14 (line 33)
Argument 'b' failed validation @(x)isnumeric(x)&&isscalar(x).
```

Прерывание функции f14 произойдет прежде, чем она попытается получить возвращаемое значение.

Аналогичный вызов функции f13 (или первого варианта функции f14) не приведет к появлению ошибки:

```
>> q = f13(5,'b',[2 4])
q =
10   20
>> q = f13(5,'b','2')
q =
250
>> q = f14(5,'b',[2 4])
q =
10   20
>> q = f14(5,'b','2')
q =
250
```

За счет встроенных возможностей MatLab, если один входной параметр (a, b или c) является массивом (например, [2 4]), получение возвращаемого значения осуществляется поэлементно, и в результате получится не числовой скаляр, а числовой массив. В приведенном примере строковое значение параметра b автоматически было преобразовано в числовое. Эти результаты обусловлены способом получения возвращаемого функцией зна-

чения и принятыми в MatLab умолчаниями по работе с массивами и преобразованием типов, которые приходится учитывать, если не контролировать типы фактически заданных входных параметров в теле функции. При другом способе получения возвращаемого значения неверный тип фактически заданных входных параметров может (если тип не контролируется) привести к появлению сообщения об ошибке в процессе получения возвращаемого значения (например, если функцию f13 вызвать следующим образом: >> q = f13([5 1]), где обязательный параметр x задан в виде одномерного массива).

Таким образом, использование функции parse для анализа и разбора фактически переданных в объявляемую функцию параметров позволяет создавать нестандартные функции, которые можно вызывать с опциями, сохраняя читаемость и структурную простоту программного кода соответствующей части объявляемой функции.

7. Заключение

Проведенный анализ показал, что уже на уровне ядра (без подключения дополнительных библиотек) MatLab предоставляет программисту широкие возможности по разработке нестандартных функций, которые можно вызывать как с полным, так и с неполным списком входных и выходных параметров. Входные параметры функции могут быть позиционированными (т.е. интерпретироваться функцией по их расположению в списке) или непозиционированными (интерпретироваться с помощью специальных строковых констант), выходные параметры могут быть только позиционированными.

Позиционированные входные параметры могут являться обязательными (их значения необходимо указывать при вызове функции всегда) и необязательными (такие параметры при вызове функции можно опустить). Все опции являются непозиционированными параметрами и всегда считаются необязательными.

Учитывая принятые способы интерпретации, всегда подразумевается, что входные параметры в заголовке функции располагаются в следующем порядке: обязательные позиционированные, необязательные позиционированные, опции. Но синтаксически ни одна из групп входных параметров никак не выделяется.

Возможность вызывать функции с полным или неполным списком входных и выходных параметров, а также с опциями, практически не сказывается на синтаксисе заголовка функции в m-файле. Но в теле функции возникает необходимость контролировать фактическое существование и соответствие определенному типу входных параметров. Начиная с версии 2007, в MatLab введены стандартные функции, помогающие программисту выполнять эти операции в теле объявляемой им нестандартной функции.

Стандартные функции, предоставляемые разработчиками MatLab, используют как обязательные, так и необязательные параметры и опции во всех возможных сочетаниях. Этим и объясняется многообразие вариантов вызова стандартных функций в зависимости от задачи, для решения которой они используются.

Анализ возможностей MatLab и приобретение личных навыков создания и использования нестандартных функций с переменным количеством параметров позволяет программисту увеличить гибкость и широту их применения. Приобретенный в результате этого опыт, в свою очередь, положительно отражается на корректности использования не только нестандартных, но и стандартных функций, предоставляемых разработчиками MatLab. Поэтому в преподавании MatLab студентам физико-математических и технических специальностей необходимо уделять внимание созданию и использованию функций, которые можно вызывать с переменным количеством параметров, например, на факультативных занятиях.

Литература

1. Калинин Т.В., Барцевич А.В., Петров С.А., Хрестинин Д.В. Программный комплекс моделирования системы радиолокационного распознавания // Программные продукты и системы. – 2017. – № 4. – С. 733-738.
2. Cai, H. Luminance gradient for evaluating lighting // Lighting Research and Technology. – 2016. – Vol. 48, No. 2. – P. 155–175.
3. Палюх Б.В., Катулев А.Н., Ягольников С.В., Храмичев А.А., Зыков И.И. Показатели безопасности космического аппарата в полете и генерация информации для предупреждения о высокоскоростном взаимодействии // Программные продукты и системы. – 2017. – № 4. – С. 726-732.
4. Чертков А.А. Автоматизация выбора кратчайших маршрутов судов на основе модифицированного алгоритма Беллмана-Форда // Вестник государственного университета морского и речного флота им. адмирала С.О. Макарова. – 2017. – Т. 9. № 5. – С. 1113-1122.
5. Дмитриенко Д.В. Исследование операций – инструмент для повышения эффективности управления водным транспортом // Вестник государственного университета морского и речного флота им. адмирала С.О. Макарова. – 2017. – Т. 9. № 5. – С. 1131-1141.
6. Воронов С.С., Жалнин В.П., Забнев В.С., Тюрин И.Ю. Автоматизация анализа долгосрочных инвестиций в среде MATLAB // Международный научно-исследовательский журнал. – 2017. – № 3-4 (57). – С. 22-28.
7. Жамбаа С., Касаткина Т.В., Бубенчиков А.М. Применение метода П.П. Куфарева к решению задачи о движении грунтовых вод под гидротехническими сооружениями // Вестник Томского государственного университета. Математика и механика. – 2017. – № 47. – С. 15-21.
8. Марголис Б.И. Программа идентификации условий теплообмена для изделий плоской формы // Программные продукты и системы. – 2017. – № 1. – С. 148-151.
9. Джалмухамбетов А.У., Кравченко И.В., Джалмухамбетова Е.А. Моделирование в среде MATLAB распределения плотности вещества экзопланет // Символ науки. – 2016. – № 10-1 (22). – С. 22-25.
10. Попов А.М. Алгоритм выявления монотонного тренда // Перспективы науки. – 2017. – № 6 (93). – С. 22-25.
11. Гарынина С.В., Попов А.М. Реализация алгоритма Гуда-Тьюринга в пакете MATLAB // Системный анализ и аналитика. – 2017. – № 4 (5). – С. 55-63.
12. Кузнецова К.С. Применение квадратур Гаусса-Эрмита для оценки ожидаемой полезности инвестиционного портфеля с использованием пакета MATLAB // Научный журнал. – 2016. – № 6 (7). – С. 8-10.
13. Жидкова Н.В., Волков В.Л. Моделирование бесплатформенной системы ориентации // Современные проблемы науки и образования. – 2015. – № 1-1. [Электронный ресурс] – Режим доступа: http://www.science-education.ru/pdf/2015/1/12.pdf.
14. Афонин В.В., Федосин С.А. О структурировании лабораторно-практических занятий при изучении дисциплин программирования // Образовательные технологии и общество. – 2014. – Т. 17. № 4. – С. 497-506.
15. Дмитриенко Д.В., Сахаров В.В., Чертков А.А. Алгоритм оценки матрицы прямых затрат в модели анализа межотраслевых связей В. Леонтьева // Транспортное дело России. – 2017. – № 3. – С. 97-99.
16. Попов А.М. Расчет основных характеристик текстового массива в пакете MATLAB // Системный анализ и аналитика. – 2017. – № 3 (4). – С. 71-91.

17. Попов А.М. Анализ структуры статистической языковой модели в пакете MATLAB // Системный анализ и аналитика. – 2017. – № 3 (4). – С. 92-110.
18. Пономарев С.Ю., Попов А.М. Комбинированный метод экспертного оценивания // Системный анализ и аналитика. – 2017. – № 4 (5). – С. 24-34.
19. Афонин В.В. Программа определения положительно определенной матрицы на основе решения линейно-квадратичной задачи управления // APRIORI. Серия: Естественные и технические науки. – 2016. – № 5. / [Электронный ресурс] – Режим доступа: http://www.apriori-journal.ru/seria2/5-2016/Afonin.pdf.
20. Гришенцев А.Ю., Коробейников А.Г., Бондаренко И.Б. К вопросу о сверхразрешении чувствительных матриц // Журнал радиоэлектроники. – 2016. – № 10. / [Электронный ресурс] – Режим доступа: http://jre.cplire.ru/jre/oct16/3/text.pdf.
21. Герасимов Н.А. Распределение финансовых рядов // Перспективы науки. – 2016. – № 10 (85). – С. 24-26.
22. Бородин Г.А., Титов В.А., Маслякова И.Н. Использование среды MATLAB при решении задач линейного программирования // Фундаментальные исследования. – 2016. – № 11-1. – С. 23-26.
23. Власова Е.А., Попов В.С., Пугачев О.В. Создание фонда оценочных средств и новых образовательных технологий с использованием MATLAB при изучении линейной алгебры // Вестник Московского государственного областного университета. Серия: Физика-математика. – 2016. – № 4. – С. 77-85.
24. Ким А.В., Новиков М.Ю. Стабилизация линейных систем с запаздыванием // Современные тенденции развития науки и технологий. – 2015. – № 9-1. – С. 26-29.
25. Сирота А.А. Методы и алгоритмы анализа данных и их моделирование в MATLAB: учеб. пособие. СПб.: БХВ-Петербург, 2016. – 384 с.
26. Дьяконов В. П. MATLAB. Полный самоучитель. М.: ДМК Пресс, 2014. – 768 с.
27. Ревинская О.Г. Основы программирования в MatLab: учеб. пособие. СПб.: БХВ-Петербург, 2016. – 208 с.
28. Васильев А.Н. MATLAB. Самоучитель. Практический подход. 2-е издание. СПб.: Наука и техника, 2015. – 448 с.
29. Солонина А.И., Клионский Д.М., Меркучева Т.В., Петров С.Н. Цифровая обработка сигналов и MATLAB: учеб. пособие. СПб.: БХВ-Петербург, 2013. – 512 с.
30. Input and Output Arguments - MATLAB & Simulink / [Электронный ресурс] – Режим доступа: http://www.mathworks.com/help/matlab/input-and-output-arguments.html (дата доступа 27.02.2018).
31. Detect state - MATLAB is* / [Электронный ресурс] – Режим доступа: http://www.mathworks.com/help/matlab/ref/is.html?s_tid=doc_ta (дата доступа 27.02.2018).