### *Тема 3.4.М-файлы и программирование средствами MatLab*

**3.4.1. Основные понятия и средства программированиям в MatLab**

**3.4.2. m-файлы, программ и функций**

**3.4.3. Работа с m-файлами**

**3.4.4. Редактор m-файлов**

**3.4.5. Основные операторы m-языка**

#### 3.4.1. Основные понятия и средства программированиям в MatLab

До сих пор мы в основном использовали систему **MatLab** в режиме непосредственного счета –*в* **командном режиме (командном окне)***.* Однако при решении серьезных задач возникает необходимость сохранения используемых последовательностей вычислений, а также их дальнейшей модификации. Иными словами, существует необходимость **программирования** решения задач.

Это может показаться отходом от важной цели, которая преследуется разработчиками большинства математических систем, – выполнения математических вычислений без использования традиционного программирования. Однако это не так. Выше было показано, что множество математических задач решается в системе MatLab без программирования. С использованием языков высокого уровня для их решения потребовалось бы написать и оттестировать сотни программ.

Практически невозможно предусмотреть в одной, даже самой большой и мощной, математической системе возможность решения всех задач, которые могут интересовать пользователя. Программирование в системе MatLab является эффективным средством ее расширения и адаптации к решению специфических проблем. Оно реализуется с помощью **языка программирования** системыMatLab.

Большинство объектов этого языка, в частности все команды, операторы и функции, одновременно являются объектами **входного языка общения с системой в командном режиме работы**. Так что фактически мы приступили к описанию языка программирования системы MatLa**b** с первых строк данной книги.

Так в чем же отличие входного языка от языка программирования? В основном – в способе фиксации создаваемых ими кодов. Сессии в командном режиме работы не сохраняются в памяти компьютера. А вот программы на языке программирования MatLab сохраняются в виде текстовых файлов (**m**-**файлов)**. При этом могут сохраняться как целые программы в виде **файлов-программ**, так и отдельные **программные модули — функции**. Кроме того, важно, что программа может менять структуру алгоритмов вычислений в зависимости от входных данных и данных, создаваемых в ходе вычислений.

С позиций программиста язык программирования системы является типичным **проблемно-ориентированным языком программирования высокого уровня**. Точнее говоря, это даже язык сверхвысокого уровня, содержащий сложные операторы и функции, реализация которых на обычных языках (например, Бейсике, Паскале или С++) потребовала бы много усилий и времени. К таким функциям относятся матричные функции, функции быстрого преобразования Фурье и др., а к операторам – операторы построения разнообразных графиков, генерации матриц определенного вида.

Итак, программами в системе MatLab являются программы, оформленные как **m**-**файлы** текстового формата, содержащие запись программ в виде программных кодов. Язык программирования системы MatLab имеет следующие средства:

* данные различного типа;
* константы и переменные;
* операторы, включая операторы математических выражений;
* встроенные команды и функции;
* функции пользователя;
* управляющие структуры;
* системные операторы и функции;
* средства расширения языка.

Тексты программ в системе MatLab пишутся на языке высокого уровня, достаточно понятном для пользователей умеренной квалификации в области программирования. Язык программирования MatLab является типичным **интерпретатором***.*

Интерпретация означает, что MatLab не создает исполняемых конечных программ. Они существуют лишь в виде **m**-**файлов**, для выполнения которых необходима среда MatLab. Однако для программ на языке MatLab созданы компиляторы, транслирующие программы MatLa**b** в коды языков программирования С и C++. Это решает задачу создания исполняемых программ, первоначально разрабатываемых в среде MatLab. Компиляторы для системы MatLab являются вполне самостоятельными программными средствами.

Начальное представление о **переменных, встроенных константах и функциях** уже было дано в предшествующих главах.

В MatLab определены следующие основные типы данных, в общем случае представляющих собой многомерные массивы:

* **single**— числовые массивы с числами одинарной точности;
* **doubl**e — числовые массивы с числами удвоенной точности;
* **char** — строчные массивы с элементами-символами;
* **sparse** — наследует свойства double, разреженные матрицы с элементами-числами удвоенной точности;
* **сеll**— массивы ячеек; ячейки, в свою очередь, тоже могут быть массивами;
* **struct** — массивы структур с полями, которые также могут содержать массивы;
* **function\_handle** — дескрипторы функций:
* **int32, uint32** — массивы 32-разрядных чисел со знаком и без знаков;
* **int16,uint16** — массивы 16-разрядных целых чисел со знаком и без знаков;
* **int8, uint8** — массивы 8-разрядных целых чисел со знаками и без знаков.

Каждому типу данных можно соотнести некоторые характерные для него операции, называемые **методами***.* Поскольку в иерархии типов данных сверху находятся данные типа **array**, это значит, что все виды данных в MatLab являются массивами.

  Язык программирования системы MatLab вобрал в себя почти все средства, необходимые для реализации различных технологий программирования:

* процедурного;
* операторного;
* функционального;
* логического;
* структурного (модульного);
* объектно-ориентированного;
* визуально-ориентированного.

В основе **процедурной***,* **операторной** и **функциональной** технологии программирования лежат **процедуры**, **операторы** и **функции**, используемые как основные объекты языка. Эти типы объектов присутствуют в MatLab. **Логическое** программирование реализуется в MatLab с помощью **логических операторов** и **функций**.

Наиболее ярко в MatLab представлены идеи **структурного программирования**. Подавляющее большинство функций и команд языка представляют собой вполне законченные модули, обмен данными между которыми происходит через их входные параметры, хотя возможен обмен информацией и через глобальные переменные. Программные модули оформлены в виде текстовых **m**-**файлов**, которые хранятся на диске и подключаются к программам по мере необходимости. Важно отметить, что в отличие от многих языков программирования, применение тех или иных модулей не требует предварительного объявления, а для создания и отладки самостоятельных модулей **MatLab** имеет все необходимые средства. Подавляющее большинство команд и функций системы **MatLab** поставляется в виде таких модулей.

**Объектно-ориентированное** программирование также широко представлено в системе MatLab. Оно особенно актуально при программировании задач графики. Что качается **визуально-ориентированного** программирования, то в MatLab оно представлено в основном в пакете моделирования заданных блоками устройств и систем **Simulink**.

  Здесь необходимо отметить, что для языка системы MatLab различие между **командами** (выполняемыми при вводе с клавиатуры) и **программными операторами** (выполняемыми из программы) является условным. И команды, и программные операторы могут выполняться как из программы, так и в режиме прямых вычислений.

В общем виде **функция** преобразует одни данные в другие. Для многих функций характерен **возврат значений,**  в ответ на обращение к ним с указанием списка **входных параметров**–**аргументов**. Например, говорят, что функция **sin(x)**в ответ на обращение к ней возвращает значение синуса аргумента **х***.* Поэтому функцию можно использовать в арифметических выражениях, например **2\*sin(x+1)***.* Для операторов (и команд), не возвращающих значения, такое применение обычно абсурдно.

Важным фактором является **двойственность операторов** и **функций**. Многие операторы имеют свои аналоги в виде функций. Так, например, оператор «+» имеет аналог в виде функции **sum().**Команды, записанные в виде ***Commandargument*** нередко имеют форму записи и в виде функции***Command(' argument')*.**

Указанная **двойственность** лежит в основе выбора между процедурным и функциональным подходами к программированию, каждый из которых имеет своих поклонников и противников, и может (в той или иной мере) подходить для решения всех классов задач. При этом переход от одного подхода программирования к другому возможен в пределах одной программы и происходит настолько естественно, что большинство пользователей даже не задумывается над тем, каким же подходом (или стилем) программирования они преимущественно пользуются.

Имеющиеся в языке MatLab управляющие структуры: условных операторы **if... else...elseif...end, case,**циклы**for...end** и **while...end**, похожи на те, которые используются в языках Бейсик, Паскаль и С+**+.**

Программирование простых задач в среде MatLab очень напоминает программирование на языке **Bаsic**. Во многих случаях программы на языке Bаsic можно почти дословно перевести на язык системы, учтя небольшие отличия в синтаксисе этих языков.

#### 3.4.2. m-файлы, программ и функций

Итак, мы установили, что работа в командном режиме (сессия) не является программированием. Внешним атрибутом последнего в MatLab служит задание последовательности действий по программе, записанной в виде **m**-**файла**. Для создания **m**-**файлов** может использоваться как редактор, встроенный в MatLab, так и любой текстовый редактор. Подготовленный и записанный на диск **m**-**файл**с соответствующим именем становится частью системы, и его можно вызывать как из командной строки, так и из другого **m**-файла. Имеется два типа **m**-файлов: **файлы-программы** и **файлы-функции**. Важно, что в процессе своего создания они проходят синтаксический контроль с помощью встроенного в систему MatLab**редактора/отладчикаm**-**файлов**.

**Файл*-*программа***,* именуемый также **Script-файлом**, является просто записью последовательности команд без входных и выходных параметров. Он имеет следующую структуру:

|  |
| --- |
| **-Имя\_файла** |
| **% Основной комментарий**  **% Дополнительный комментарий**  ***Тело файла с любыми выражениями*** |

Файлы-программы имеют следующие особенности:

* они не имеют входных и выходных аргументов;
* работают с данными из рабочей области;
* в процессе выполнения не компилируются;
* представляют собой зафиксированную в виде файла последовательность операций, полностью аналогичную той, что используется во время сессии.

Основным комментарием является первая строка текстовых комментариев, а дополнительным – последующие строки комментариев. Это связано с тем, что основной комментарий выводится при выполнении команд **lookfor** и **helpимя\_каталога**. Полный комментарий выводится при выполнении команды **help Имя\_файла**. В качестве примера рассмотрим файл-программу **PlotSin**и пример вызова ее.

|  |
| --- |
| **Пример 3.4.1 m-файла** |
| **% Построение графика синусоиды линией красного цвета**  ***% с* масштабной сеткой в интервале [xmin.xmax]**  **x=xmin:0.1:xmax;**  **plot(x,sin(x),е6'r')**  **gridon** |

|  |
| --- |
| **Пример 3.4-1 сессии** |
| **>>xmin=-1;**  **>>xmax=1;**  **>>Пример5.5-1;**  **>>** |

Первые две строки **m**-файла **Пример 3.4-1 –** это комментарий, остальные – тело файла. Знак ***%***в комментариях должен начинаться с первой позиции строки. В противном случае команда **help*name\_файла*** не будет воспринимать комментарий и возвратит сообщение вида

**Nohelpcommentsfoundin-name.m.**

Обратите внимание на то, что такой файл нельзя запустить без предварительной подготовки, сводящейся к заданию значений переменным **xmin** и **хmах**, которые используются в теле файла. Это следствие первого свойства файлов-программ– они работают с данными из рабочей области. Переменные, используемые в файлах-программах, являются глобальными, т. е. они действуют одинаково в командах сессии и внутри программного блока, которым является файл-сценарий. Поэтому заданные в сессии значения переменных используются и в теле файла. Имена файлов-программ нельзя использовать в качестве параметров функций, поскольку файлы-программы не возвращают значений. Можно сказать, что файл-программа– это простейшая программа на языке программирования MatLab.

Результаты работы сессии Примера 3.4-1. приведены на рис. 3.4-1.

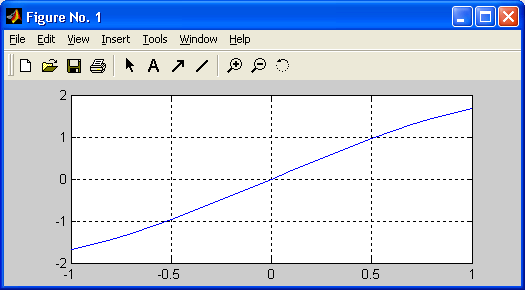


Рис. 3.4-1. Результат работы файла-программы Пример 3.4-1

**m-функция** является типичным объектом языка программирования системы MatLab. Одновременно он является полноценным модулем с точки зрения структурного программирования, поскольку содержит входные и выходные параметры и использует аппарат локальных переменных. Структура такого модуля с одним выходным параметром выглядит следующим образом:

|  |
| --- |
| **имяm-файла** |
| **function var=f\_name(Cnиcoк\_napaмeтpов)**  **% Основной комментарий**  **% Дополнительный комментарий**  **Тело m-файла**  **vаr=выражение** |

**m-функции**имеют следующие особенности:

* он начинается с объявления **function**, после которого указывается имя переменной **var**- выходного параметра, имя самой функции и список ее входных параметров;
* функция возвращает свое значение и может использоваться в виде **f**\_**name (список\_параметров)** в математических выражениях;
* все переменные, имеющиеся в теле файла-функции, являются **локальными***,*т. е. действуют только в пределах тела функции;
* файл-функция является самостоятельным программным модулем, который общается с другими модулями через свои входные и выходные параметры;
* правила вывода комментариев те же, что у файлов-программ;
* файл-функция служит средством расширения системы MatLab;
* при обнаружении файла-функции он компилируется и затем исполняется, а созданные машинные коды хранятся в рабочей области системы MatLab.

Последняя конструкция **vаг=выражение** вводится, если требуется, чтобы функция возвращала результат вычислений.

Приведенная форма файла-функции характерна для функции с одним выходным параметром. Если выходных параметров больше, то они указываются в квадратных скобках после слова **function**. При этом структура модуля имеет следующий вид:

|  |
| --- |
| **имя-файла** |
| **function [varl,var2....]=f\_name(Список\_параметров)**  **% Основной комментарий**  **% Дополнительный комментарий**  **Тело файла с любыми выражениями**  **vаг1=выражение**  **vаг2=выражение** |

Такая функция во многом напоминает процедуру. Ее нельзя слепо использовать непосредственно в математических выражениях, поскольку она возвращает не единственный результат, а множество результатов — по числу выходных параметров.

Если функция используется как имеющая единственный выходной параметр, но имеет ряд выходных параметров, то для возврата значения будет использоваться первый из них. Это зачастую ведет к ошибкам в математических вычислениях. Поэтому, как отмечалось, данная функция используется как отдельный элемент программ вида:

**[var1,var2,... ]=f\_nаmе(Список\_параметров)**

После его применения переменные выхода **varl, var2,...** становятся определенными и их можно использовать в последующих математических выражениях и иных сегментах программы. Если функция используется в виде **f\_nаmе(Список\_параметров**), то возвращается значение только первого выходного параметра – переменной **varl**.

  Переменные, указанные в списке параметров функции, являются **локальными**и служат для переноса значений, которые подставляются на их место при вызовах функций.

Эта особенность переменных-параметров хорошо видна при разборе примера 3.2-3, результат.

|  |
| --- |
| **Пример 3.4-2** |
| **function z=fun(x,y)**  **% Пример5.3.2-2**  **z=x^2+y^2;** |

|  |
| --- |
| **Пример** 3.4**-2** |
| **>> z=fun(2,3);**  **>> x=**  **0**  **>> y=**  **0**  **>>z=**  **13**  **>>** |

В этом примере в окне редактора создана функция **fun(x,y)** двух переменных **х**и **у**, вычисляющая **z = х 2 + у 2***.* Поскольку переменные **х**и **у**указаны как параметры функции **fun(x, у),**то они являются локальными. В примере вне тела функции им заданы нулевые значения. Очевидно, что при вычислении значения **fun(2, 3)** в теле функции задается **х=2** и **у=3**. Поэтому результат -**z=13**. Однако после выхода из тела функции переменные **х** и **у** принимают свои исходные значения, равные нулю. Так что эти переменные меняют свои значения на значения параметров функции только локально — в пределах тела функции.   
А каков статус переменной **z** в нашем примере? Она, как и любая переменная, определенная в теле функции, также будет локальной. Изначально ее значение не определено. В теле функции переменная принимает значение **z=13**. А после возврата из функции, как нетрудно увидеть из примера 3.4-1, переменная **z,** несмотря на ее применение в теле функции, остается неопределенной. На это указывает сообщение, отображаемое после попытки вывода значения переменной **z**.

Возврат из функции производится после обработки всего тела функции, т. е. при достижении конца файла функции. При использовании в теле функции условных операторов, циклов или переключателей иногда возникает необходимость осуществить возврат функции раньше, чем будет достигнут конец файла. Для этого служит команда **return**. В любом случае, результатом, возвращаемым функцией, являются значения выходных параметров (в нашем случае выходным параметром является переменная **z**), присвоенные им на момент возврата.

У нашей функции имеется один недостаток — вывод на индикацию значения **z=13** из тела функции, хотя после этого **z**остается равным **0**. Чтобы убрать побочный эффект вывода значения **z**, достаточно установить знак**;** после математического выражения, определяющего **z**.

Этот пример наглядно показывает, что пропуск любого слова или даже простого оператора (вроде знака ;) может привести к не сразу понятным побочным эффектам и даже неверной работе функции. Программирование требует особой точности и педантичности.

Из сказанного ясно, что переменные в файлах-программах являются **глобальными***,* а в файлах-функциях - **локальными***.* Нередко применение глобальных переменных в программных модулях может приводить к побочным эффектам. Применение локальных переменных устраняет эту возможность и отвечает требованиям структурного программирования.

Однако передача данных из модуля в модуль в этом случае происходит только через входные и выходные параметры, что требует тщательного планирования такой передачи. Однако и при создании файлов-функций порой желательно применение глобальных переменных. Ответственность за это должен брать на себя программист, создающий программные модули.

Команда

**globalvarl var2...**

позволяет объявить переменные модуля-функции глобальными. Таким образом, внутри функции могут использоваться и такие переменные, если это нужно по условиям решения вашей задачи. Причем, чтобы несколько программных модулей могли совместно использовать глобальную переменную, она должна быть объявлена как **global**во всех модулях.

Часто в ходе вычислений возникают ошибки. Например, при вычислении функции **sin(x)/x***-*при ***х* = 0** имеет место ошибка вида «деление на ноль». При появлении ошибки вычисления могут завершиться досрочно с выводом сообщения об ошибке. Следует, однако, отметить, что не все ошибки вызывают остановку вычислений. Некоторые сопровождаются только выдачей предупреждающей надписи. Такие ситуации должны учитываться программистом, отмечаться как ошибочные и по возможности устраняться. Для вывода сообщения об ошибке служит команда

**error('Сообщение об ошибке');**

при выполнении, которой вычисления прерываются, и выдается сообщение об ошибке, заданное в апострофах. Ниже дан пример вычисления функции **sd(x)=sin(x)/x***,* в котором задано сообщение об ошибке:

|  |
| --- |
| **Пример\_**3\_4**\_3** |
| **function f=sd(x)**  **ifx==0 error('Ошибка - делениена 0'). end**  **f=sin(x)/x ;** |

Для выявления ситуации об ошибке использован оператор условного перехода if, который будет описан детально несколько позднее. Результат выполнения данной функции приводится ниже:

|  |
| --- |
| **Пример** 3.4**-3** |
| **>>x=0;**  **>>f=sd(x);**  **???Ошибка - деление на 0**  **>>x=1;**  **>>f=sd(x);**  **>> f=**  **>>** |

Если остановка программы при появлении ошибки нежелательна, то может использоваться команда вывода предупреждающего сообщения

**warning('Предупреждающее сообщение').**

Эта команда выводит стоящее в апострофах сообщение, но не препятствует дальнейшей работе программы. Признаком того, что является ошибкой, а что -предупреждением, являются символы **???** и слово **Warning** в соответствующих сообщениях.

  При создании функций со специальными свойствами весьма полезны две приведенные ниже функции:

* **nargin** — возвращает число входных параметров данной функции;
* **nargout**— возвращает число выходных параметров данной функции.

Пусть, к примеру, мы хотим создать функцию, вычисляющую сумму квадратов пяти аргументов **xl, х2, хЗ, х4 и х5**.

Обычный путь состоит в следующем — создаем функцию с именем **sum2\_5**:

|  |
| --- |
| **sum2\_5\_Пример\_**3.4-4 |
| **function f=sum2\_5(x1,x2,x3,x4,x5);**  **f=x1^2+x2^2+x3^2+x4\*2+x5^\*2;** |

|  |
| --- |
| **Пример**3.4-4 |
| **» sum2\_5\_Пример\_5\_3\_2\_4 (l,2,3,4,5)**  **ans =**  **55**  **» sum2\_5(l,2)**  **??? Input argument 'хЗ' is undefined.**  **Error in ==> C:\MATI\_AB\bin\sum2\_5.m**  **On line 2 ==> f=x1^2+x2^2+x3^2+x4^2+x5^2;**  **>>** |

Итак, при наличии всех пяти аргументов функция работает корректно. Но если аргументов менее пяти, она выдает сообщение об ошибке. С помощью функции **nargin** можно создать функцию **sum2\_5m**, которая работает корректно при любом числе заданных входных аргументов в пределах от **1** до **5**:

|  |
| --- |
| **sum2\_5\_Пример\_3\_4\_5** |
| **f unction f=sum2\_5m(x1 ,x2 , хЗ ,x4 , x5) ;**  **n=nargin;**  **if n==1 f=x1^2; end**  **if n==2 f=x1^2+x2^2;end**  **if n==3 f=x1^2+x2^2+x3^2; end**  **if n==4 f=x1^2+x2 ^ 2+x3^2+x4 ^ 2: end**  **if n==5 f=x1^2+x2^2+x3^2+x 4^2+x5^2** |

В данной функции используется условный оператор **if...end**, который будет детально описан далее. Но и без этого ясно, что благодаря применению функции **nargin** и условного оператора, вычисления всякий раз идут по формуле с числом слагаемых, равным числу входных аргументов - от одного до пяти. Это видно из приведенных ниже примеров:

|  |
| --- |
| **Пример**3.4**-5** |
| **» sum2\_5m(1)**  **ans =**  **1**  **» sum2\_5m(1,2)**  **ans =**  **5**  **» sum2\_5m( 1,2,3)**  **ans =**  **14**  **» sum2\_5m(1,2,3,4)**  **ans =**  **30**  **» sum2\_5m(1,2,3,4,5)**  **ans=**  **55**  **» sum2\_5m(1,2,3,4,5,6)**  **??? Error using ==> sum2\_5m**  **Too many input arguments.**  **>>** |

Итак, при изменении числа входных параметров от **1** до **5** вычисления проходят корректно. При большем числе параметров выводится сообщение об ошибке. Это уже действует встроенная в интерпретатор **MatLab** система диагностики ошибок

    Как отмечалось, команда **helpname**, где **name** — имя **m**-**файла,** обеспечивает чтение первой строки с текстовым комментарием и тех строк с комментариями, которые следуют непосредственно за первой строкой. Комментарий, расположенный за пределами этой области, не выводится. Это позволяет создавать невыводимый программный комментарий.

Пустая строка прерывает вывод комментария при исполнении команды **helpname**. Команда **typename** выводит текст программы со всеми комментариями, в том числе и следующими после пустых строк.

Команда **helpcatalog**, где **catalog** — имя каталога с **m**-**файлами**, позволяет вывести комментарий, общий для всего каталога. Такой комментарий содержится в файле **contents.m**, который пользователь может создать самостоятельно с помощью редактора **m**-**файлов**. Если такого файла нет, то будет выведен список первых строк комментариев для всех m-файлов каталога.

**m**-**файлы-функции** могут использоваться как в командном режиме, так и вызываться из других **m**-**файлов**. При этом необходимо указывать все входные и выходные параметры. Исключением является случай, когда выходной параметр единственный — в этом варианте функция возвращает единственный результат и может использоваться в математических выражениях. При использовании глобальных переменных они должны быть объявлены во всех **m**-**файлах**, используемых в решении заданной задачи, и во всех входящих в них встроенных подфункциях.

Имена функций должны быть уникальными. Это связано с тем, что при обнаружении каждого нового имени **MatLab** проверяет, относится ли это имя к переменной, подфункции в данном **m**-**файл**е, частной функции в каталогах **PRIVATE** или функции в одном из каталогов пути доступа. Если последняя встречается, то будет исполнена именно эта функция.

Если аргумент функции используется только для вычислений и его значения не меняются, то аргумент передается ссылкой, что уменьшает затраты памяти. В других случаях аргумент передается значением. Для каждой функции выделяется своя (рабочая) область памяти, не входящая в область, предоставляемую системе **MatLab**. Глобальные переменные принадлежат ряду областей памяти. При их изменении меняется содержимое всех этих областей.

При решении задач с большим объемом данных может ощущаться нехватка оперативной памяти. Признаком этого становится появление сообщения об ошибке **«Outofmemory».**

В этом случае может быть полезным применение следующих мер:

* стирание ставших ненужными данных, прежде всего больших массивов;
* увеличение размеров файла подкачки **Windows**;
* уменьшение размера используемых данных;
* снятие ограничений на размеры используемой памяти;
* увеличение объема физической памяти компьютера.

Чем больше емкость **ОП** компьютера, на котором используется система **MatLab** тем меньше вероятность возникновения указанной ошибки. Опыт показывает, что даже при решении задач умеренной сложности емкость **ОЗУ** не должна быть менее **256 Мбайт**.

#### 3.4.3. Работа с m-файлами

Во время работы в **MatLab** часто необходимо создавать или редактировать **m-файлы**, а после этого возвращаться в командное окно **MatLab**для отладки или вычислений. Для этого имеется специальный редактор/отладчик, в котором можно исправлять текст и выполнять пошаговую отладку программы. После исправления необходимо сохранить сделанные изменения. Причем **m-файлы**, с которыми работают, должны быть доступны. Для этого либо текущая директория должна быть директорией с вашими файлами, либо необходимо проложить туда путь. Это можно сделать с помощью пункта меню **File/SetPath**, который позволяет сделать директории доступными.

Для поиска **m-файлов** система **MatLab** использует механизм путей доступа, поскольку **m-файлы** записываются в каталоги или папки файловой системы. Например, при поиске файла с именем **foo,МatLab**выполняет следующие действия:

* просматривает, не является ли **foo**именем переменной;
* просматривает, не является ли **foo**встроенной функцией;
* ищет в текущем каталоге **m**-**файл** с именем **foo.m**;
* ищет **m**-**файл** с именем **foo.m**во всех каталогах **списка путей доступа.**

Реально применяемые правила поиска являются более сложными. Однако приведенный выше упрощенный порядок поиска точно отражает механизм поиска **m**-**файлов**, с которыми обычно работает пользователь.

В процессе сеанса работы можно вывести на дисплей или внести изменения в список путей доступа, используя следующие функции:

* **path**- выводит на экран список путей доступа;
* **path(s)**- заменяет существующий список списком **s**;
* **addpath/home/lib**и **path(path, ’/home/lib’)**-добавляют новый каталог текущего подкаталога в список путей доступа;
* **rmpath /home/lib**-удаляетпуть**/home/lib**изсписка.

Список путей доступа, используемый по умолчанию, определен в файле **pathdef.m**, который размещен в каталоге **lосаl**; этот файл выполняется при каждом запуске системы **MatLab**.

Кроме работы из командной строки, существует средство просмотра путей доступа **PathBrowser**, которое поддерживает удобный графический интерфейс для просмотра и изменения списка путей.

Система **MatLab** использует понятие **текущего каталога** при работе с **m**-файлами во время сеанса работы. **Начальный текущий каталог** определен в файле запуска.

Для вывода текущего каталога на экран предназначена команда **cd**. Для изменения текущего каталога следует использовать команду

**сd<новый путь доступа>**.

Как было указано выше, при работе в системе **Windows** имеется специальное средство для просмотра и изменения путей доступа **SetPath** (рис. 3.3-1). Показанное далее окно открывается либо из меню **File/SetPath** основного окна, либо с помощью кнопки на инструментальной панели.

После дополнения списка путей доступа необходимо сохранить новый путь с помощью пункта меню **File/SavePath**, в противном случае установленный путь будет известен системе только на время одного сеанса работы.

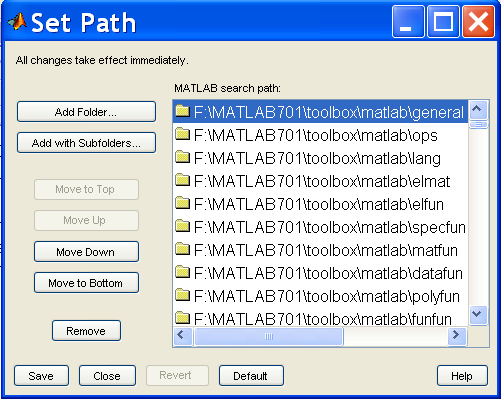


Рис. 3.4-1. Окно **SetPath**

**Редактор/отладчик** предоставляет из себя как средства редактирования текста **m**-**файла**, так и средства пошаговой его отладки. Один из способов вызова редактора – вызов из командной строки **MatLab**с помощью команды **edit**. Например, команда **editpoof**откроет встроенный редактор для редактирования файла **poof.m**, если в меню **File** в диалоговом окне **Preferences**не установлен вами другой редактор.

Можно открыть редактор и другим способом – с помощью меню **File/New** или кнопки **NewFile** на панели инструментов. Для открытия существующего **m**-**файла** выберите пункт **File/Open** или щелкните на кнопке **OpenFile**.

После вызова редактор/отладчик будет иметь вид, показанный на рис. 3.4-2.



Рис. 3.4-2. Общий вид редактора/отладчика

Редактор, используемый в системе, имеет синтаксическую раскраску, т.е. слово или символ по мере ввода приобретают тот цвет, который соответствует их типу. Редактор различает такие типы вводимых слов:

* комментарии
* ключевые слова
* незаконченные строки
* законченные строки
* другой текст.

С помощью пункта меню **Tools/Fonts** можно настроить такие важные параметры, как используемый шрифт. В основном данный редактор не отличается от обычного многооконного текстового редактора - в нем работают все редактирующие клавиши (**Del, Bspace, Home**и т.д.). При редактировании файлов вы можете непосредственно перейти к требуемой строке при помощи пункта меню **Edit/GoToLine**и указать номер требуемой строки в появившемся окне. После редактирования файла и повторного его запуска из командного окна желательно предварительно **сохранить новый вариант файла**. Но можно запускать редактируемый файл на выполнение, не выходя из редактора (т.е. не переходя в командное окно) с помощью пункта меню **Debug/Run**. При этом предварительное сохранение текста исправлений не требуется.

Одной из важных особенностей данного редактора является то, что после проведения вычислений можно в редакторе просмотреть значения переменных, которые они имеют в текущий момент в рабочей области. Для этого достаточно установить курсор мыши на этой переменной, и появится прямоугольник с желтым фоном, на котором выводится текущее значение переменной. Если переменная является большой матрицей, то таким образом увидеть целиком ее не удастся. Для просмотра (и возможного исправления при отладке) всех значений матрицы есть специальный пункт меню **View/WorkspaceBrowser**. Имеется еще два интересных пункта в меню **View**. Это пункт **EvaluateSelection**, который позволяет вычислять значение выделенного выражения и помещать результат в командное окно, и пункт **AutoIndentSelection**, который выполняет автоматическое форматирование текста программы с отступами в соответствии с правилами **MatLab**. Выбор пункта меню **View/Options**позволяет получить доступ к диалоговому окну, в котором можно настроить параметры редактора.

Отладка программного кода - это процесс, в ходе которого могут быть выявлены ошибки двух видов:

* **синтаксические,** которые связаны с неточностью записи имен m-функций или арифметических выражений. **MatLab** обнаруживает большинствосинтаксических ошибок, сопровождая их сообщением об ошибке с указанием номера строки соответствующего m-файла;
* **ошибки времени выполнения,** которые, как правило, имеют алгоритмическую природу и проявляются в том, что приводят к непредвиденным результатам.

Достаточно легко можно исправить синтаксические ошибки, которые сопровождаются сообщениями о причинах их возникновения. Ошибки времени выполнения выявить более сложно, потому что локальная рабочая область m-функции оказывается потерянной, если ошибка приводит к возврату в рабочую область системы **MatLab**. Чтобы определить причину такой ошибки, можно использовать один из следующих приемов:

* реализовать вывод результатов промежуточных вычислений на дисплей, удалив в соответствующих операторах точки с запятой, которые подавляют вывод на экран промежуточных результатов;
* добавить в **m**-**файл** команды **keyboard**, которые останавливают выполнение **m**-**файла** и разрешают проверить и изменить переменные рабочей области вызываемой m-функции(в этом режиме появляется специальное приглашение ), а возврат к выполнению функции реализуется командой **return**);
* закомментировать заголовок функции и выполнить m-файл как сценарий (это позволяет проследить результаты промежуточных вычислений в рабочей области системы);
* использовать отладчик системы **МatLab**.

Отладчик полезен для исправления ошибок во время выполнения программы именно потому, что он дает возможность отслеживать рабочие области функции и проверять или изменять значения соответствующих переменных. Отладчик позволяет устанавливать и удалять контрольные точки, то есть специальным образом помеченные строки **m**-**файла**, в которых выполнение останавливается. Это дает возможность изменять содержимое рабочей области, просматривать стек вызова m-функций и выполнять **m**-**файл** построчно. Отладчик может функционировать как в режиме командной строки, так и в режиме графического интерфейса пользователя.

Далее мы рассмотрим отладку только в режиме графического интерфейса пользователя, поскольку он наиболее прост и нагляден. Рассмотрим возможности отладки, которые нам предоставляет **Editor/Debugger**.

Для его запуска используется команда **edit<имя\_файла>**или пункт меню **File/Open.** Можно открыть окно редактора/отладчика и с помощью пункта меню **File/New/M-file.** При таком варианте имя отлаживаемого файла открывается уже из меню самого редактора/отладчика. Общий вид окна редактора показан на рис. 3.2-2.

Способ использования кнопок понятен из их названия. Часть кнопок представляют собой обычные редактирующие кнопки, используемые при сохранении, копировании, печати и поиске файлов. Другая часть связана непосредственно с отладкой. Это кнопки установки и очистки точек остановки, кнопки пошагового перемещения по программе с заходом в подпрограммы (**Stepin**) и без захода в под-

программы (**SingleStep**), кнопка продолжения вычислений (**Continue**) и кнопка остановки отладки (**QuitDebugging**). Как было отмечено выше, с помощью пункта меню **View/WorkspaceBrowser**можно не только просмотреть, но и изменить значение любой переменной. При этом в одном из окон редактора открывается таблица, подобная электронной таблице, и в ней можно не только просматривать, но и исправлять значения переменных.

#### 3.4.4. Редактор m-файлов

В **MatLab** имеется редактор **m-файлов**, для запуска которого следует нажать кнопку **Newm-file** на панели инструментов рабочей среды, либо выбрать в меню **File** в команду **New/m-file**. На экране появляется окно редактора. Наберите в нем какие-либо команды, например для построения графика:

|  |
| --- |
| **Пример3.4-6** |
| **%**  **%**  **x=[-1:0.01:1];**  **y=exp(x);**  **plot(x,y)**  **gridon**  **title('Экспоненциальная функция')** |

Для запуска программы или ее части есть несколько способов. Первый, самый простой – выделить операторы при помощи мыши, удерживая левую кнопку, или при помощи клавиши **<Shift>** со стрелками, **<PageUp>,<PageDown>** и выбрать **Text** пункт **EvaluateSelection** (или нажать **<F9>**). Выделенные операторы выполняются последовательно, точно так же, как если бы они были набраны в командной строке. Очевидно, что работать в **m**-файле удобнее, чем из командной строки, поскольку можно сохранить программу, добавить операторы, выполнять отдельные команды, не пробегаясь по истории команд, как в случае командной строки.

После того, как программа сохранена в **m**-**файле**, к примеру, в **myprog.m,** для ее запуска можно использовать пункт **Run** меню **Debug**, либо набрать в командной строке имя **m-файла** (без расширения) и нажать **<Enter>,** то есть выполнить, как команду **MatLab**. При таких способах запуска программы следует учесть важное обстоятельство — путь к каталогу с **m**-**файлом** должен быть известен **MatLab**. Сделайте каталог с файлом **myprog** текущим.

В **MatLab** в меню **File** рабочей среды перейдите к пункту **SetPath…** Появляется диалоговое окно **PathBrowser** (навигатор путей). В строке ввода **CurrentDirectory**установите требуемый каталог. Воспользуйтесь кнопкой, расположенной справа от строки ввода, для выбора каталога.

В **MatLab** установка текущего каталога производится из окна **CurrentDirectory** рабочей среды. Если это окно отсутствует, то следует выбрать пункт **CurrentDirectory** меню **View** рабочей среды. Для выбора желаемого каталога на диске нажмите кнопку, расположенную справа от раскрывающегося списка.

Когда текущий каталог установлен, то все **m**-**файлы**, находящиеся в нем, могут быть запущены из командной строки, либо из редактора **m**-**файлов**. Все переменные файл-программ после ее запуска доступны в рабочей среде, т. е. являются глобальными. Убедитесь в этом, выполнив команду **whos.** Более того, файл-программа может использовать переменные рабочей среды. Например, если была введена команда:

|  |
| --- |
| **-Пример3.4-7** |
| **>> a=[0.1 0.4 0.3 1.9 3.3];**  **>>** |

то файл-программа,

|  |
| --- |
| **-Пример\_3\_4\_7** |
| **%**  **%**  **bar(а)** |

содержащая строку **bar(а),** построит столбцевую диаграмму вектора **a** (разумеется, если он не был переопределен в самой файл-программе).

Файл-функции отличаются от файл-программ тем, что они могут иметь входные и выходные аргументы, а все переменные, определенные внутри файл-функции, являются локальными и не видны в рабочей среде. **m**-**файл**, содержащий файл-функцию, должен начинаться с заголовка, после него записываются операторы **MatLab**. Заголовок состоит из слова function, списка выходных аргументов, имени файла-функций и списка входных аргументов. Аргументы в списках разделяются запятой. Пример3.4-3 содержит простейшей файл-функции с двумя входными и одним выходным аргументами.

|  |
| --- |
| **-Пример\_3\_4\_8** |
| **Function c=mysum(a,b)**  **c=a+b;** |

Наберите этот пример в новом файле в редакторе и сохраните его. Обратите внимание, что **MatLab** предлагает в качестве имени **m**-**файла** название файла-функций, т.е. **mysum.m**. Всегда сохраняйте файл-функцию в **m**-**файле**, имя которого совпадает с именем файл-функции! Убедитесь, что каталог с файлом **mysum.m** является текущим и вызовите файл-функцию **mysum** из командной строки:

|  |
| --- |
| **Пример3.4-8** |
| **>> s=mysum(2,3)**  **s =**  **5**  **>>** |

При вызове файл-функции **mysum** произошли следующие события:

* входной аргумент **a**получил значение 2;
* входной аргумент **b** стал равен 3;
* сумма **a** и **b** записалась в выходной аргумент **c**;
* значение выходного аргумента **c** получила переменная **s** рабочей среды и результат вывелся в командное окно.

Заметьте, что оператор **c=a+b** в файл-функции **mysum** завершен точкой с запятой для подавления вывода локальной переменной **c** в командное окно. Для просмотра значений локальных переменных при отладке файл-функций, очевидно, не следует подавлять вывод на экран значений требуемых переменных.

Практически все функции **MatLab** являются файл-функциями и хранятся в одноименных**m-**файлах. Функция **sin** допускает два варианта вызова: **sin(x) и y=sin(x)**, в первом случае результат записывается в **ans**, а во втором — в переменную **y**. Наша функция **mysum** ведет себя точно так же. Более того, входными аргументами **mysum** могут быть массивы одинаковых размеров или массив и число.

Разберем теперь, как создать файл-функцию с несколькими выходными аргументами. Список выходных аргументов в заголовке файл-функции заключается в квадратные скобки, сами аргументы отделяются запятой. В качестве Примера 3.4.4-4 приведена файл-функция **quadeq**, которая по заданным коэффициентам квадратного уравнения находит его корни.

|  |
| --- |
| **Пример\_3\_4\_9** |
| **function [x1,x2]=quadeq(a,b,c)**  **%**  **D=b^2-4\*a\*c;**  **x1=(-b+sqrt(D))/(2\*a);**  **x2=(-b-sqrt(D))/(2\*a);** |

|  |
| --- |
| **Пример3.4-9** |
| **%При вызове quadeq из командной строки используйте квадратные скобки для %указания переменных, в которые будут занесены значения корней:**  **>> [r1,r2]=quadeq(1,3,2)**  **r1 =**  **-1**  **r2 =**  **-2**  **>>** |

Заметьте, что файл-функцию **quadeq** можно вызвать без выходных аргументов, или только с одним выходным аргументом. В этом случае вернется только первый корень.

Файл-функция может и не иметь входных или выходных аргументов. Заголовки таких файл-функций приведены ниже:

**functionnoout(a,b),**

**function [v,u]=noin,**

**functionnoarg()**

Умение писать собственные файл-функции и файл-программы необходимо как при программировании в **MatLab**, так и при решении различных задач средствами **MatLab** (в частности, поиска корней уравнений, интегрирования, оптимизации). Разберем только один пример, связанный с построением графика функции

**y=exp(-x).\*(sin(x)+0.1\*sin(100\*pi\*x))**на отрезке **[0;1].**

Используйте поэлементные операции для того, чтобы **myfun( )** можно было вызывать от вектора значений аргумента и получать вектор соответствующих значений функции.

|  |
| --- |
| **Пример\_3.4-10** |
| **function y=myfun(x);**  **%**  **y=exp(-x).\*(sin(x)+0.1\*sin(100\*pi\*x));**  **%График y можно получить двумя способами. Первый способ состоит в %создании вектора %значений аргумента, скажем с шагом 0.01, заполнении %вектор значений функции и вызове plot:**  **>> x=[0:0.01:1];**  **>> y=myfun(x);**  **>> plot(x,y)**  **>>** |

|  |
| --- |
| **Пример3.4-10** |
| **%%График y можно получить двумя способами. Первый способ состоит в %создании вектора значений аргумента, скажем с шагом 0.01, заполнении %вектор значений функции и вызове plot:**  **>> x=[0:0.01:1];**  **>> y=myfun(x);**  **>> plot(x,y)**  **>>** |

#### 3.4.5. Основные операторы m-языка

Помимо программ с **линейной структурой***,* инструкции которых исполняются строго по порядку, существует множество программ, структура которых **нелинейна***.* При этом ветви программ могут выполняться в зависимости от определенных условий, иногда с конечным числом повторений - циклов, иногда в виде циклов, завершаемых при выполнении заданного условия. Практически любая серьезная программа имеет нелинейную структуру. Для создания таких программ необходимы специальные управляющие структуры. Они имеются в любом языке программирования, и в частности в **MatLab**.

***Операторы ввода/вывода*.**Приведем простой пример диалоговой программы.

|  |
| --- |
| **Пример\_3\_4\_11** |
| *%* Вычисление длины окружности с диалоговым вводом радиуса  r=0;  while r>=0,  r=input('Введите радиус окружности r=');  if r>=0 disp(' Длинаокружности l='); disp(2\*pi\*r), end  end |

Эта программа служит для многократного вычисления длины окружности по вводимому пользователем значению радиуса r. Обратите внимание на то, что здесь мы впервые показываем пример организации простейшего диалога. Он реализован с помощью команды **input:**

**input(`'Введите радиус окружности r=');**

При выполнении этой команды вначале выводится запрос в виде строки, затем происходит остановка работы программы и ожидается ввод значения радиуса **r** (в общем случае числа). Ввод, как обычно, подтверждается нажатием клавиши <**Enter>**, после чего введенное число присваивается переменной **r**. Следующая строка

**if r>=0 disp(' Длина окружности l = '); disp(2\*pi\*r);end**

с помощью команды **disp** при **r>=0** выводит надпись **«Длина окружности 1=»** и вычисленное значение длины окружности. Она представляет собой одну из наиболее простых управляющих структур типа **if...end**. В данном случае она нужна для остановки вычислений, если вводится отрицательное значение **г** (прием, который любят начинающие программисты).

Приведенные строки включены в управляющую структуру **while...end**. Это необходимо для циклического повторения вычислений с вводом значений **r**. Пока **r>=0**. цикл повторяется. Но стоит задать **r<0**, вычисление длины окружности перестает выполняться, а цикл завершается.

Если данная программа записана в виде **m**-**файлаПример\_3\_4\_1**, то работа с ней будет  выглядеть следующим образом:

|  |
| --- |
| **Пример** 3.4**-11** |
| **Введите радиус окружности R=1**  **Длина окружности l=**  **6.2832**  **Введите радиус окружности R=2**  **Длина окружности l=**  **12.5664**  **Введите радиус окружности R=-1**  **»** |

Функция **input**может использоваться и для ввода произвольных строковых выражений. При этом она задается в следующем виде:

**input('Комментарий', V)**

При выполнении этой функции она останавливает вычисления и ожидает ввода строкового комментария. После ввода возвращается набранная строка. Это иллюстрирует следующий пример:

|  |
| --- |
| **Пример**3.4**-12** |
| **» S=input('Введите выражение ','s')**  **Введите выражение (Вводим) 2\*sin(l)**  **S =**  **2\*sin(l)**  **» eval(S)**  **ans =**  **1.6829**  **>>** |

Обратите внимание на то, что функция **eval** позволяет вычислить выражение, заданное в символьном виде.

***Условный оператор*.** Условный оператор **if** в общем виде записывается следующим образом:

**if***Условие1*

*Инструкции\_1*

**еlself***Условие2*

*Инструкции\_2*

**else**

*Инструкции\_3*

**еnd**

Эта конструкция допускает несколько частных вариантов. В простейшем

**if***Условие*

*Инструкции*

**end**

Пока *Условие*возвращает логическое значение **1** (то есть «истина»), выполняются *Инструкции*, составляющие тело структуры **if...end**. При этом оператор **end** указывает на конец перечня инструкций. Инструкции в списке разделяются оператором**,** (запятая) или **;** (точка с запятой). Если *Условие*не выполняется (дает логическое значение **0**, «ложь»), то *Инструкции* также не выполняются.

Еще одна конструкция

**if***Условие*

*Инструкции\_1*

**else**

*Инструкции\_2*

**end**

выполняет *Инструкции\_1*, если выполняется *Условие*, или *Инструкции\_2* в противном случае.

*Условия* записываются в виде:

*Выражение\_1 Оператор\_отношения Выражение\_2,*

причем в качестве *Операторов\_отношения* используются следующие операторы: **==, <, >, <=, >= или ~=.**

***Оператор цикла –for...end*.**Оператор цикла типа **for...end** обычно используются для организации вычислений с заданным числом повторяющихся циклов. Конструкция такого цикла имеет следующий вид:

**forvаг=s:d:e**

*Инструкция*

*….*

*Инструкция*

**end**

где **s**— начальное значение переменной цикла **var**, **d**— приращение этой переменной и **е** - конечное значение управляющей переменной, при достижении которого цикл завершается. Возможна и запись в виде **s:е** (в этом случае **d=l**). Список выполняемых в цикле инструкций завершается оператором **end**.

Следующие примеры поясняют применение цикла для получения квадратов значений переменной цикла:

|  |
| --- |
| **Пример**3.4**.-13** |
| **» for 1=1:5 i^2, end;**  **ans =**  **1**  **ans =**  **4**  **ans =**  **9**  **ans =**  **16**  **ans =**  **25**  **» for x=0:.25:1 Х^ 2, end:**  **ans =**  **0**  **ans =**  **0.0625**  **ans =**  **0.2500**  **ans =**  **0.5625**  **ans =**  **1**  **>>** |

Оператор ***continue*** передает управление в следующую итерацию цикла, пропуская операторы, которые записаны за ним, причем во вложенном цикле он передает управление на следующую итерацию основного цикла. Оператор ***break*** может использоваться для досрочного прерывания выполнения цикла. Как только он встречается в программе, цикл прерывается. Возможны вложенные циклы, например:

|  |
| --- |
| **Пример\_3\_4\_14** |
| **%**  **for i=1:3**  **for j=l:3**  **A(i.j)=i+j;**  **end**  **end** |

В результате выполнения этого цикла (файл **Пример\_3\_5\_4**.**m**) формируется матрица**А**:

|  |
| --- |
| **Пример** 3.4**-14** |
| **>> Пример4-3-12**  **%%%**  **>>А**  **А =**  **2 3 4**  **3 4 5**  **4 5 6**  **>>** |

Следует отметить, что формирование матриц с помощью оператора**:** (двоеточие) обычно занимает намного меньше времени, чем с помощью цикла. Однако применение цикла нередко оказывается более наглядным и понятным. **MatLab** допускает использование в качестве переменной цикла массива**А** размера ***т****х****п****.* При этом цикл выполняется столько раз, сколько столбцов в массиве **А**, и на каждом шаге переменная **var**представляет собой вектор, соответствующий текущему столбцу массива **А**:

|  |
| --- |
| **Пример** 3.4**-15** |
| **» А=[1 2 3:4 5 6]**  **А =**  **1 2 3**  **4 5 6**  **» for var=A; var, end**  **var = 1 4**  **var = 2 5**  **var= 3 6**  **>>** |

***Оператор цикла - while...end.***Оператор цикл типа **while** выполняется до тех пор, пока выполняется **Условие**:

**while***Условие*

*Инструкции*

**end**

|  |
| --- |
| **Пример\_3\_4\_16** |
| **%**  **%**  **x=[-1:0.01:1];**  **y=exp(x);**  **plot(x,y)**  **gridon**  **title('Экспоненциальная функция')** |

|  |
| --- |
| **Пример3\_4\_16** |
| **%При вызове quadeq из командной строки используйте квадратные скобки для указания %переменных, в которые будут занесены значения корней:**  **>> [r1,r2]=quadeq(1,3,2)**  **r1 =**  **-1**  **r2 =**  **-2**  **>>** |

Досрочное завершение циклов реализуется с помощью операторов ***break*** или ***continue****.*

***Оператор множественного выбора - switch*.**Для осуществления множественного выбора (или ветвления) используется конструкция с переключателем типа **switch:**

**switch***switch\_Bыражение*

**case***саsе\_Выражение*

*Список\_инструкций*

**case {***саsе\_Выражение1,*

*саsе\_выражение2, саsе\_ВыражениеЗ....}*

*Список\_инструкций*

**otherwise.** *Список\_инструкций***end**

Если выражение после заголовка **switch** имеет значение одного из выражений *саsе\_Выражение...,* то выполняется блок операторов **case**, в противном случае — список инструкций после оператора **otherwise**. При выполнении блока **case** исполняются те списки инструкций, для которых *саsе\_Выражение* совпадает со *switch\_Bыpaжением*. Обратите внимание на то, что саsе\_Выражение может быть числом, константой, переменной, вектором ячеек или даже строчной переменной. В последнем случае оператор **case** истинен, если функция **strcmp(***значение, выражение***)** возвращает логическое значение «**истинa**».

Поясним применение оператора **switch**на примере **m**-**файлаПример\_3\_4\_5\_7.m**:

|  |
| --- |
| **Пример\_3\_4-17** |
| **%Пример 5-3.4-7**  **switchvan**  **case {1,2,3}**  **disp('Первыйквартал')**  **case {4,5,6}**  **disp('Второйквартал')**  **case {7,8,9}**  **disp('Третий квартал')**  **case {10.11,12}**  **disp('Четвертый квартал')**  **otherwise**  **disp('Ошибкавзадании')**  **end** |

Эта программа в ответ на значения переменной **van**– номера месяца – вычисляет, к какому кварталу относится заданный месяц, и выводит соответствующее сообщение:

|  |
| --- |
| **Пример** 3.4**-17** |
| **>>var=2; swl**  **Первый квартал**  **>>var=4;swl**  **Второй квартал**  **>>var=7:swl**  **Третий квартал**  **>>var=12;swl**  **Четвертый квартал**  **>var=-l;swl**  **Ошибка в задании**  **>>** |

Для остановки программы используется оператор ***pause***. Он используется в следующих формах:

* **pause**— останавливает вычисления до нажатия любой клавиши;
* **pause(N)** — останавливает вычисления на N секунд;
* **pauseon**— включает режим отработки пауз;
* **pauseoff** — выключает режим отработки пауз.

Следующий пример поясняет применение команды **pause:**

|  |
| --- |
| **Пример\_3\_4-18** |
| **%Пример 5.3.4-9**  **fori=1:20;**  **х =rand(1,40); у =rand(1,40); z = sin(x.\*y);**  **tri = delaunay(x.y);**  **trisurf(tri,x,y,z)**  **pause;**  **end** |

Команда **pause** обеспечивает показ 20 рисунков - построений трехмерных поверхностей из треугольных окрашенных областей со случайными параметрами.