И.Б.Бадриев, В.В. Бандеров, О.А.Задворнов

РАЗРАБОТКА ГРАФИЧЕСКОГО ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА В СРЕДЕ MATLAB

Казань
Казанский государственный университет
имени В.И.Ульянова-Ленина

2010

УДК 519.6

Печатается по решению редакционно-издательского совета факультета
вычислительной математики и кибернетики ГОУ ВПО
"Казанский государственный университет им. В.И. Ульянова-Ленина"
протокол № 1 от 8 февраля 2010 г.
заседания кафедры экономической кибернетики
протокол № 12 от 9 декабря 2009 г.

Научный редактор:
Доктор физико-математических наук, профессор КГУ В.Р. Фазылов

Рецензенты:
Доктор физ.-мат. наук, заведующий кафедрой
математической статистики КГУ В.С. Желтухин,
Кандидат физ.-мат. наук, доцент кафедры
автоматизации технологических процессов и производств
Нижнекамского химико-технологического института Н.Н. Саримов

Бадриев И.Б., Бандеров В.В., Задворнов О.А.

Разработка графического пользовательского интерфейса в среде MATLAB. Учебное пособие / И.Б.Бадриев, В.В. Бандеров, О.А.Задворнов. – Казань: Казанский государственный университет, 2010. – 113 с.

В данном учебном пособии излагаются основные принципы создания графического пользовательского интерфейса в среде MATLAB.

Мы предполагаем, что читатель знаком с тем, как программируются файл-функции с подфункциями, файл-функции с переменным числом входных и выходных аргументов, имеет навыки работы с числовыми массивами, строками, структурами, ячейками и массивами строк, структур и ячеек, пользовался функциями высокоуровневой графики, а также умеет использовать конструкции встроенного языка программирования. В пособии мы не ставили целью предоставить полное описания всех команд, функций и свойств графических объектов, тем более что они рассмотрены в многочисленной литературе и приведены в документации по системе. Мы обсудим лишь общие приемы создания пользовательского интерфейса и покажем основные моменты, возникающие при работе с графическими объектами и их свойствами.

Пособие предназначено для студентов старших курсов ВУЗов, обучающихся по специальностям, связанным с применением IT-технологий для решения прикладных задач.

© Бадриев И.Б.,
Бандеров В.В.,
Задворнов О.А., 2010
© Казанский государственный
университет, 2010

Оглавление

Введение

В данном учебном пособии излагаются основные принципы создания графического пользовательского интерфейса в среде MATLAB.

Графические возможности системы MATLAB являются мощными и разнообразными. В первую очередь целесообразно изучить наиболее простые в использовании возможности. Их часто называют высокоуровневой графикой. С понятием графики связано представление о графических объектах, имеющих определенные свойства. В большинстве случаев об объектах можно забыть, если только вы не занимаетесь объектно-ориентированным программированием задач графики. Большинство команд высокоуровневой графики, ориентированной на конечного пользователя, автоматически устанавливает свойства графических объектов и обеспечивает воспроизведение графики в нужной системе координат, палитре цветов, масштабе и т.д. Несмотря на обилие графических команд, их синтаксис достаточно прост и легко усваивается даже начинающими пользователями.

На более низком уровне используется ориентированная на программиста дескрипторная графика (Handle Graphics), когда каждому графическому объекту ставится в соответствие

особое описание – дескриптор (указатель), на который возможны ссылки при использовании графического объекта. Дескрипторная графика позволяет осуществлять визуальное программирование объектов пользовательского интерфейса – управляющих кнопок, текстовых панелей и т.д. Команды дескрипторной графики могут использоваться в высокоуровневой графике, например, для удаления осей, изменения цвета и т.п. в уже построенных графических объектах. Эти обширные возможности делают графику MATLAB одной из лучших среди графических подсистем систем компьютерной математики.

Для создания приложений с графическим интерфейсом в состав MatLab входит специализированная среда GUIDE. Работа в этой среде достаточно проста – элементы управления (кнопки, раскрывающиеся списки и др.) размещаются при помощи мыши, а затем программируются события, которые возникают при обращении пользователя к данным элементам управления.

Сначала в учебном пособии приводится иерархия графических объектов MATLAB и основные приемы работы с графическими объектами. Вторая часть учебного пособия посвящена созданию приложения с графическим интерфейсом в среде GUIDE. В ней рассматриваются принципы работы с различными элементами управления, обсуждаются вопросы про-

граммирования событий (функций), возникающих при обращении пользователя к элементам управления, а также различные возможности обмена данными между подфункциями.

Мы предполагаем, что читатель знаком с тем, как программируются файл-функции с подфункциями, файл-функции с переменным числом входных и выходных аргументов, имеет навыки работы с числовыми массивами, строками, структурами, массивами строк и массивами ячеек, умеет пользоваться функциями высокоуровневой графики, конструкциями встроенного языка программирования. В пособии мы не ставили целью представить полное описание всех команд, функций и свойств графических объектов, тем более что их рассмотрению посвящена многочисленная литература и они приведены в документации по системе. Мы обсудим лишь общие приемы создания пользовательского интерфейса и основные моменты, возникающие при работе с графическими объектами и их свойствами.

Пособие предназначено для студентов старших курсов ВУЗов, обучающихся по специальностям, связанным с применением IT-технологий для решения прикладных задач.

1 Введение в дескрипторную графику.

Для создания приложений с графическим интерфейсом пользователя в состав MATLAB входит специализированная среда GUIDE. Работа в этой среде достаточно проста – элементы управления (кнопки, раскрывающиеся списки и т.д.) размещаются при помощи мыши, а затем программируются события, которые возникают при обращении пользователя к данным элементам управления.

Приложение может состоять как из одного основного окна, так и из нескольких окон, и осуществлять вывод графической и текстовой информации в основное окно приложения и в отдельные окна. Ряд функций MATLAB предназначен для создания стандартных диалоговых окон открытия и сохранения файла, печати, выбора шрифта, окна для ввода данных и др., которыми можно пользоваться в собственных приложениях.

Вначале обсудим организацию иерархической структуры и свойства графических объектов.

Рассмотрим процесс создания графических объектов на примере построения графика функции $\sin(x)$ при изменении аргумента x с шагом 0.2 на отрезке $[0; 10]$(см. рис. 1):

```
x = 0:0.2:10;
y = sin(x);
plot(x, y);
```

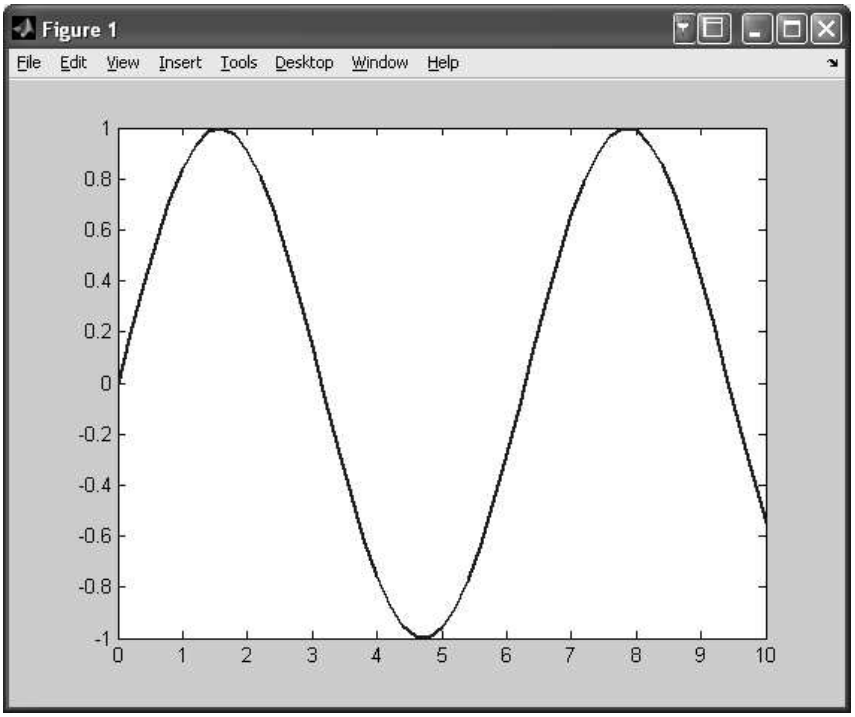

Рис. 1: Графическое окно.

Если не были открыты графические окна, то высокоуровневая графическая функция plot создает ряд *графических объектов*: сначала – графическое окно, затем – оси и, наконец – линию. Все графические объекты MATLAB выстроены в определенную иерархию, оси являются "потомком" графического окна и не могут существовать сами по себе. В свою очередь, графическое окно, – "предок" для осей. Аналогичным образом дело обстоит с линией. Она является "потомком" осей, а оси – ее "предком". Одновременно может существовать несколько графических окон, каждое из которых может содержать несколько потомков (осей), а каждая из этих

осей – по несколько потомков (линий, поверхностей и других графических объектов). В случае, если существует несколько графических окон, то функция plot создаст оси в текущем графическом окне (конечно, если до этого оси не были созданы) и построит линию в текущих (или только что созданных) осях. Если в одном графическом окне имеется несколько осей, то функция plot построит линию в текущих осях.

Замечание 1.1 *MATLAB выводит графические объекты в специальных графических окнах, имеющих в заголовке слово Figure (изображение, внешний вид, фигура).*

Иерархия графических объектов для MATLAB версии 7 представлена на рис. 2. Как видно из этого рисунка, графических объектов достаточно много, поэтому в данном пособии мы обсудим лишь общие приемы создания пользовательского интерфейса и основные моменты, возникающие при работе с графическими объектами.

2 Текущий графический объект, указатели на объекты

Рассмотрим более подробно работу с приложением, содержащим одновременно несколько графических объектов.

Предположим, что открыто несколько графических окон,

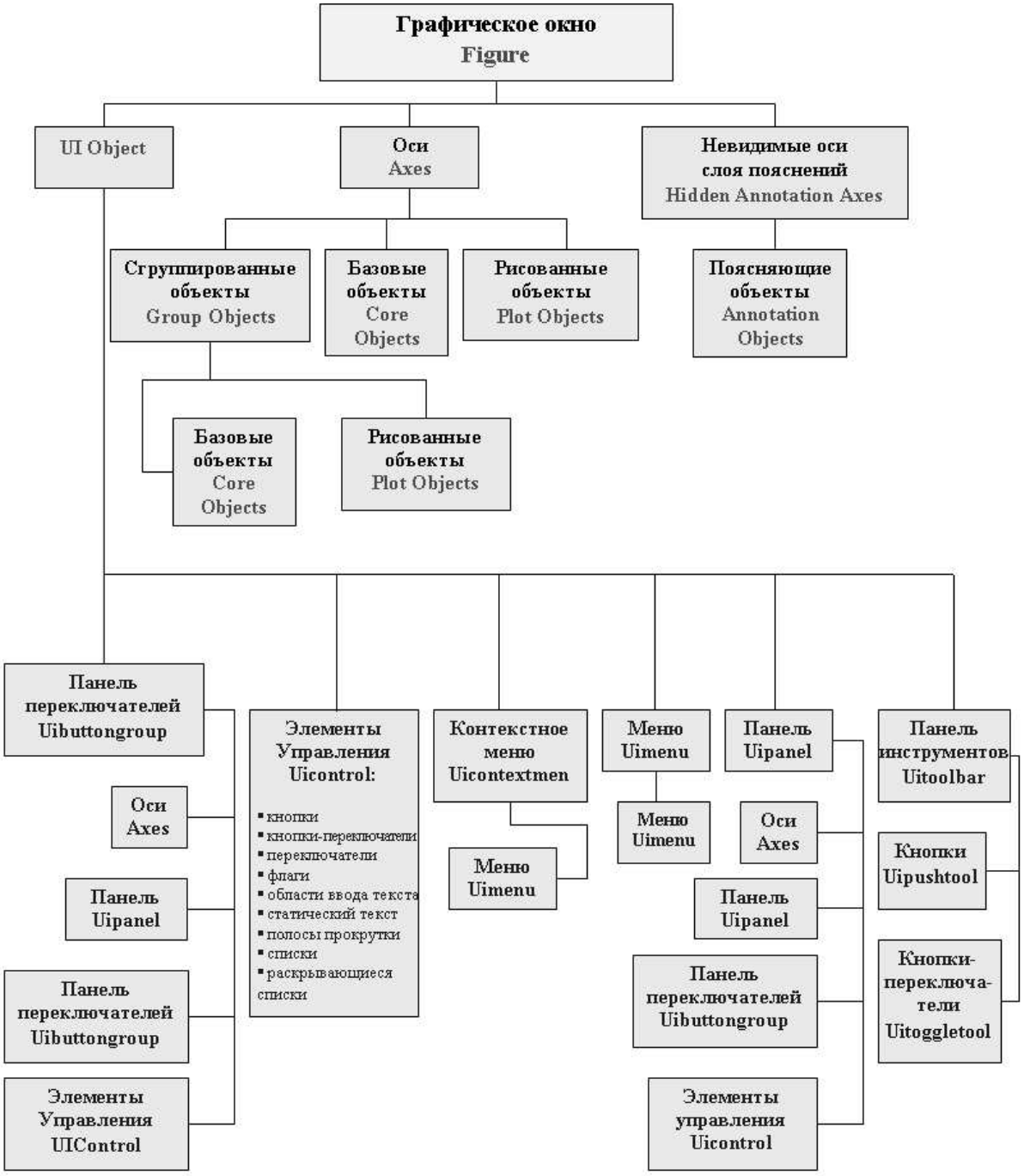

Рис. 2: Иерархическая структура графических объектов

содержащих одну или несколько пар осей. Как говорилось выше, функция plot выведет график на *текущие оси*, то есть последние созданные оси, или те, которые были сделаны текущими, например, при помощи щелчка по ним мышью. Если приложение осуществляет вывод на разные оси, то необходимо уметь делать текущими оси (да и любые объекты – графические объекты, линии и поверхности) в ходе работы приложения. Для этого в MATLAB предусмотрена возможность работы с указателями на объект. Использование указателей позволяет пользователю идентифицировать объект, а также дает возможность обращаться к объекту для получения и установки значений его свойств.

Работу с указателями на объект рассмотрим на примере создания графического окна.

Для создания графического окна служит функция figure. Вызов ее с выходным аргументом приводит не только к появлению графического окна, но и к записи в выходной аргумент *указателя* на созданный объект – графическое окно.

Замечание 2.1 *Каждому графическому объекту при его создании MATLAB присваивает уникальный числовой идентификатор (handle) – указатель на созданный объект. Идентификатор объекта root равен 0. Для остальных объектов идентификатор – это число вещественного типа. Вызов*

большинства функций высокоуровневой графики с выходным аргументом обеспечивает запись в выходной аргумент указателя на созданный графический объект.

Каждый объект имеет свойства Parent и Children. Значением свойства Parent является указатель на предка объекта в иерархии объектов. Значением свойства Children является указатель или вектор указателей на потомки этого объекта.

Создадим два графических окна, записав указатели на них в переменные hF1 и hF2:

```
hF1 = figure;

hF2 = figure;
```

Теперь для того, чтобы в любом месте приложения сделать текущим графическое окно с указателем hF1, достаточно обратиться к функции figure со входным аргументом – указателем на окно:

```
figure(hF1);
```

При этом графическое окно не только станет текущим, но и расположится поверх остальных окон.

Предположим, что в некотором месте программы требуется в графическое окно с указателем hF1 вывести графики функций $\sin(x)$ и $\cos(x)$. Создадим в первом графическом окне оси при помощи функции axes, запомнив указатель на оси в пере-

менной hA1:

```
hA1 = axes;
```

Построим теперь на этих осях графики функций. Для наглядности установим цвет графика функции $\sin(x)$ – красным, а графика $\cos(x)$ – зеленым. Сохраним в переменные hL1 и hL2 значение указателей на соответствующую линию, обратившись к функции plot с выходным аргументом:

```
x = 0:0.2:10;
f = sin(x);
g = cos(x);
hL1 = plot(x, f, 'Color','red');
% Если  второй график нужно вывести поверх первого графика", то
% перед исполнением  второй графической команды plot, нужно
% выполнить команду hold on, которая предназначена для
%  удержания текущего графического окна.
hold on;
hL2 = plot(x, g, 'Color','green');
```

Напомним, что если строка начинается с символа "%", то она является комментарием и не выполняется.

Теперь мы можем обращаться как ко второму графику функции, так и к первому. Например, если после выполнения приложением некоторых действий нам потребуется удалить график первой функции, то для этого достаточно воспользоваться функцией delete, которая удаляет графический объект

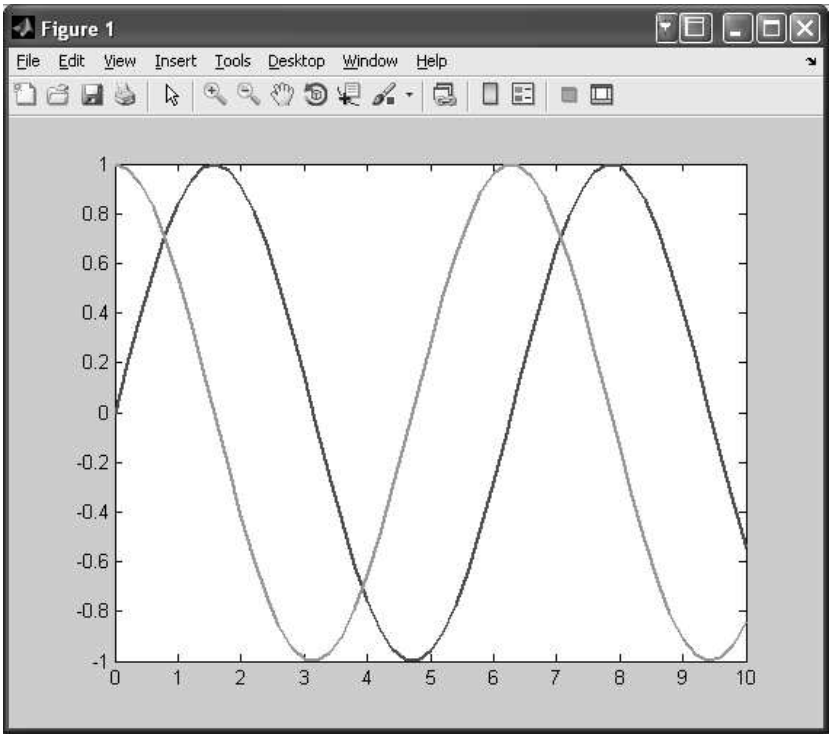

Рис. 3: Графическое окно.

с заданным указателем:

```
delete(hL1);
```

Для построения графиков на осях с указателем hA1 в любом месте программы достаточно сделать их текущими:

```
axes(hA1);
```

Отметим, что при создании графического объекта полезно сохранять указатели на него в переменных, поскольку появляется возможность манипулировать объектами. Если в нашем примере удалить оси, то удалится и оставшаяся на них линия, поскольку в иерархии объектов линия является потомком осей и не может существовать отдельно от них:

```
delete(hA1);
```

Аналогичным образом удаление графического окна вызовет исчезновение всех объектов, лежащих ниже в иерархии: осей, размещенных в этом окне, и всех принадлежащих им поверхностей и линий. Разумеется, объекты можно не только удалять при помощи функции delete, но и копировать или осуществлять поиск одного или нескольких объектов с нужными свойствами.

Мы разобрали, как делать объект текущим, зная указатель на него. Обратная задача – получение указателя на текущий объект – решается с привлечением функции gco (сокращение от get current object):

```
hCO=gco;
```

Эта функция позволяет определить, какой объект сделан текущим, скажем, щелчком мыши.

Перечислим некоторые функции для получения идентификатора графического объекта в MATLAB-программе:

функция findobj – выполняет поиск объекта по заданным свойствами (например, имени) и возвращает идентификатор объекта;

функция gcb – возвращает идентификатор окна, содержащего объект, для которого вызвана процедура обработки со-

бытия;

функция gcbo – возвращает идентификатор объекта, для которого вызвана процедура обработки события;

функция gcf – возвращает идентификатор текущего (активного) графического окна; обычно используется при работе с элементами управления;

функция gcbf – возвращает идентификатор повторно вызываемого графического окна;

функция gca – возвращает идентификатор текущего (активного) объекта axes; обычно используется при работе с графикой на дескрипторном ("низком") уровне.

Все эти функции можно использовать, например, для установки значений свойствам только что созданным графическим объектам в приложении.

3 Создание простого приложения в среде GUIDE

Приложение может состоять как из одного основного окна, так и из нескольких окон, и осуществлять вывод графической и текстовой информации в основное окно приложения и в отдельные окна. Ряд функций MATLAB предназначен для создания стандартных диалоговых окон открытия и сохранения файла, печати, выбора шрифта, окна для ввода данных и др., которыми можно пользоваться в собственных приложениях.

Замечание 3.1 *Приложение с графическим интерфейсом может быть написано и без применения среды GUIDE. В качестве примера можно привести приложение bspligui, входящее в состав Spline ToolBox. Желающим разобраться в создании приложений без среды GUIDE можно посоветовать запустить приложение bspligui в режиме отладки, проследить за созданием окна приложения, элементов управления и обработкой событий (достаточно открыть файл bspligui.m, установить точку остановки на первой исполняемой строке и запустить приложение).*

При создании приложений с графическим интерфейсом пользователя будут полезны следующие разделы справочной системы MATLAB:

– "MATLAB: Creating Graphical User Interfaces";

– "MATLAB: Functions – Categorical List: Creating Graphical User Interfaces";

– "MATLAB: Handle Graphics Property Browser"(справочник свойств графических объектов).

В справочной системе MATLAB 7 в разделе "Demo" есть десятиминутная демонстрация создания приложения с графическим интерфейсом в среде GUIDE.

В качестве простейшего примера рассмотрим создание в среде GUIDE приложения, позволяющего строить график функции. Пусть требуется создать приложение с графическим интерфейсом graphic, запуск которого, например, из командной строки

```
» graphic;
```

приводил бы к появлению окна приложения, представленного на рис 4. После нажатия пользователем на кнопку "построить" в правой части окна приложения строится график введенной функции (см. рис. 5); если функция не введена, то будет строиться график функции, выбранной в выпадающем списке.

Для создания приложения запустим среду GUIDE с помощью команды

```
» guide;
```

в окне команд, или через меню, как показано на рис. 6

В результате запустится мастер создания графического интерфейса и появится диалоговое окно GUIDE Quick Start (см. рис. 7).

Диалоговое окно GUIDE Quick Start имеет две вкладки:

- вкладка Create New GUI (создание нового приложения), которая позволяет создавать новое приложение. На ней

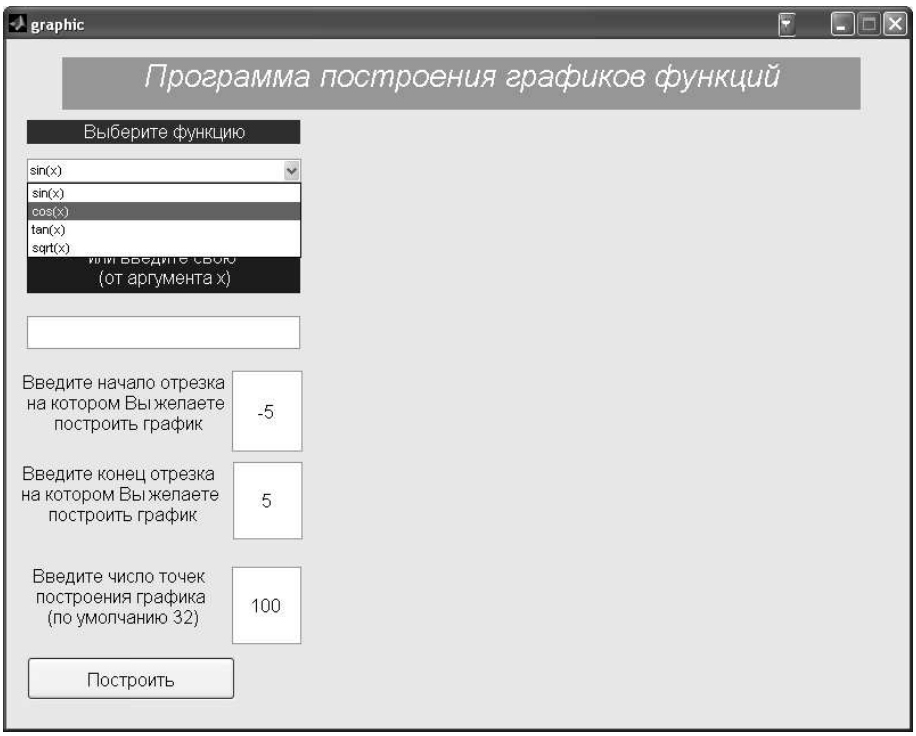
graphic
Программа построения графиков функций
Выберите функцию
sin(x)
sin(x)
cos(x)
tan(x)
sqrt(x)
(от аргумента x)
Введите начало отрезка на котором Вы желаете построить график
-5
Введите конец отрезка на котором Вы желаете построить график
5
Введите число точек построения графика (по умолчанию 32)
100
Построить

Рис. 4:

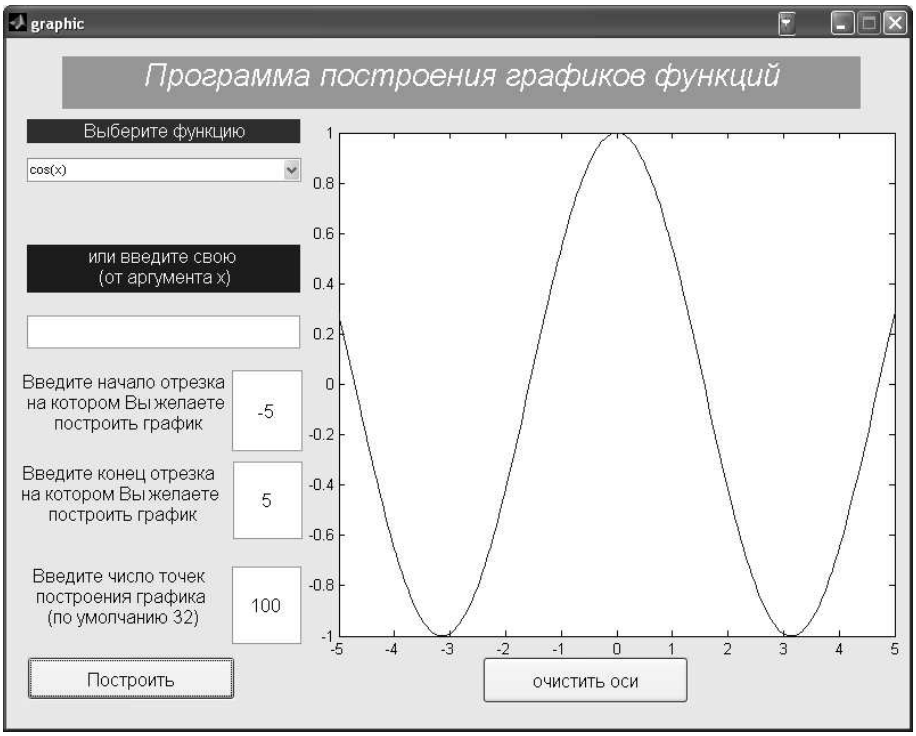
graphic
Программа построения графиков функций
Выберите функцию
cos(x)
или введите свою (от аргумента x)
Введите начало отрезка на котором Вы желаете построить график
-5
Введите конец отрезка на котором Вы желаете построить график
5
Введите число точек построения графика (по умолчанию 32)
100
Построить
очистить оси
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-5
-4
-3
-2
-1
0
1
2
3
4
5

Рис. 5:

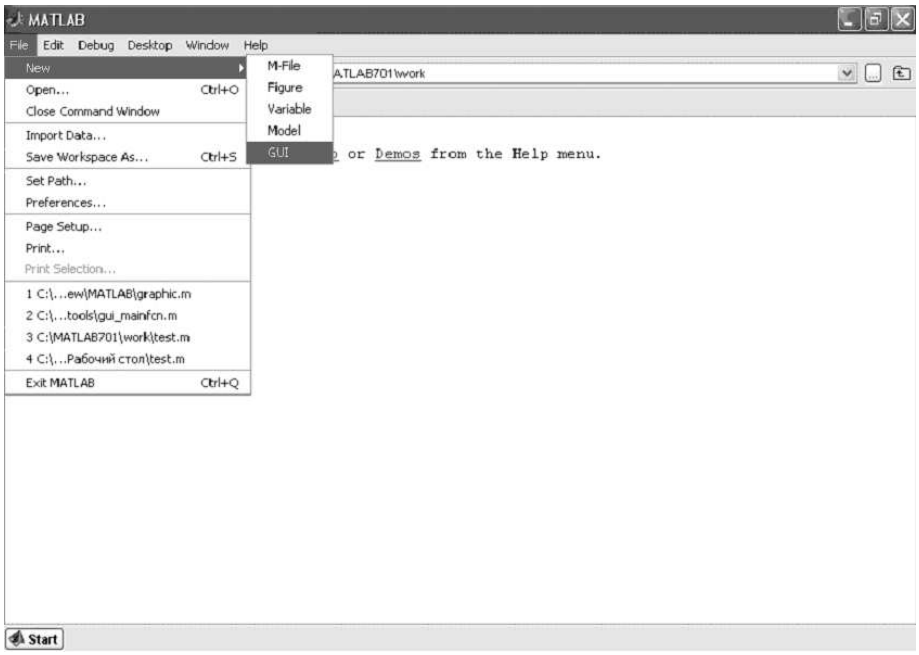

Рис. 6:

можно выбрать четыре заготовки: Blank GUI (пустое окно приложения), GUI with Uicontrols (заготовка с кнопками, переключателями и областями ввода), GUI with Axes and Menu (заготовка с осями, меню, кнопкой и раскрывающимся списком), Modal Question Dialog (заготовка для модального окна);

- вкладка Open Existing GUI (открытие существующего приложения).

Кроме того, во вкладке Create New GUI внизу есть флаг, установка которого позволяет сразу задать имя файла, в котором будет храниться наша программа. Однако, поскольку приложение всегда можно будет сохранить в процессе редак-

тирования, то этот флаг устанавливать необязательно.

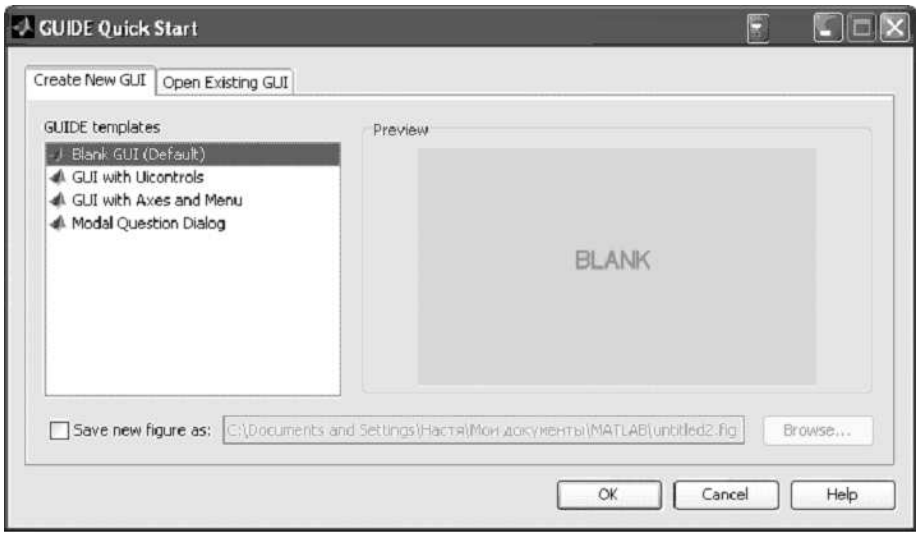

Рис. 7:

Выберем на вкладке Create New GUI строку Blank GUI и нажмем кнопку <OK>. При этом появляется основное окно среды GUIDE, содержащее заготовку для окна приложения, панель инструментов для добавления элементов интерфейса, управляющей панели и меню (рис. 8). Для добавления элементов управления в заготовку окна приложения необходимо на панели инструментов выбрать соответствующий элемент и при помощи мыши перетащить его в окно приложения.

Сначала добавим кнопку. Для этого при помощи мыши выберем инструмент Push Button (его пиктограмма изображена в виде кнопки <OK>, а имя появляется на всплывающей подсказке) и щелчком мыши поместим кнопку на заготовку окна приложения (рис.9).

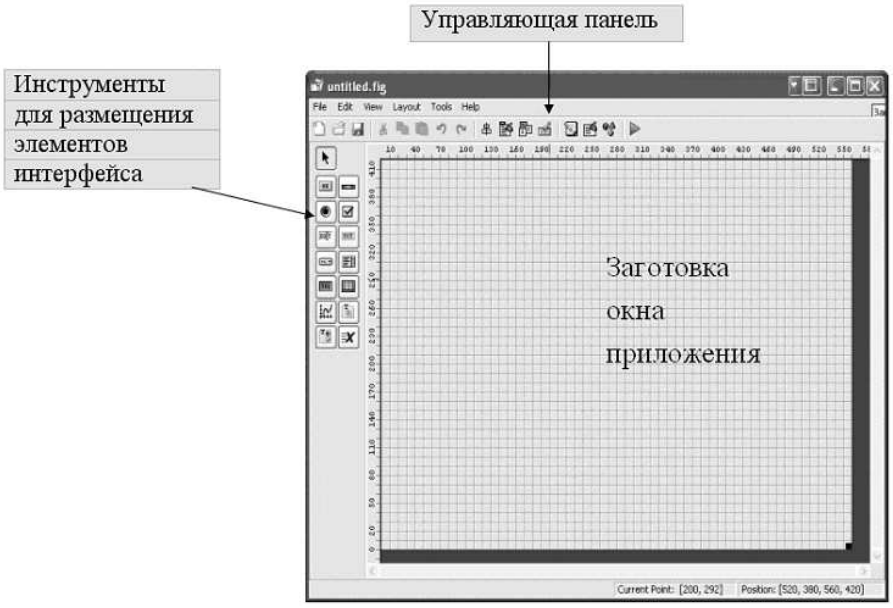
Управляющая панель
Инструменты
для размещения
элементов
интерфейса
untitled.fig
File Edit View Layout Tools Help
Заготовка
окна
приложения
Current Point: [200, 292]
Position: [520, 380, 560, 420]

Рис. 8:

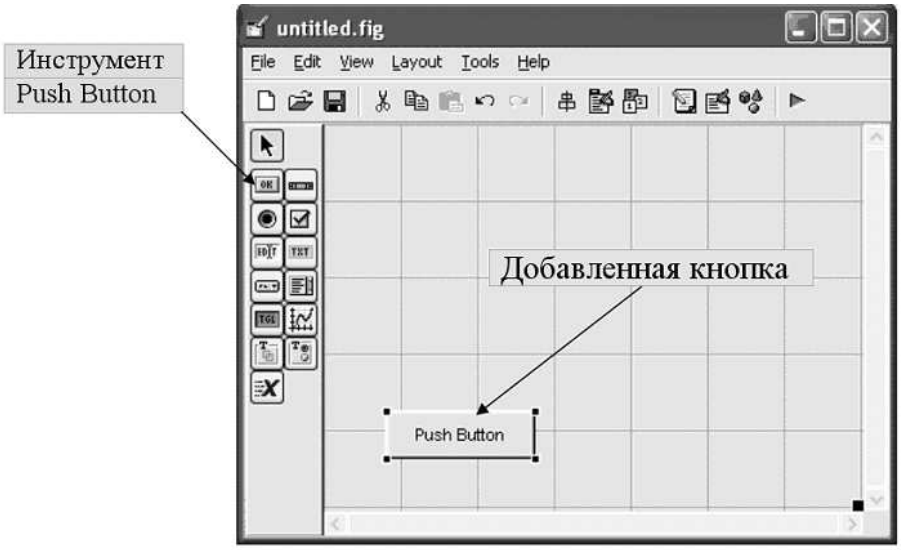
Инструмент
Push Button
untitled.fig
File Edit View Layout Tools Help
Добавленная кнопка
Push Button

Рис. 9:

Аналогично добавим в заготовку окна приложения остальные элементы графического интерфейса: поля для ввода (Edit Text), выпадающий список (pop-up menu), оси (axes), поля отображения текстовой информации (Text) и кнопку, предназначенную для очистки осей (Push Button)(рис. 10). Для этого воспользуемся соответствующими инструментами так же, как и при добавлении кнопки (рис. 9)

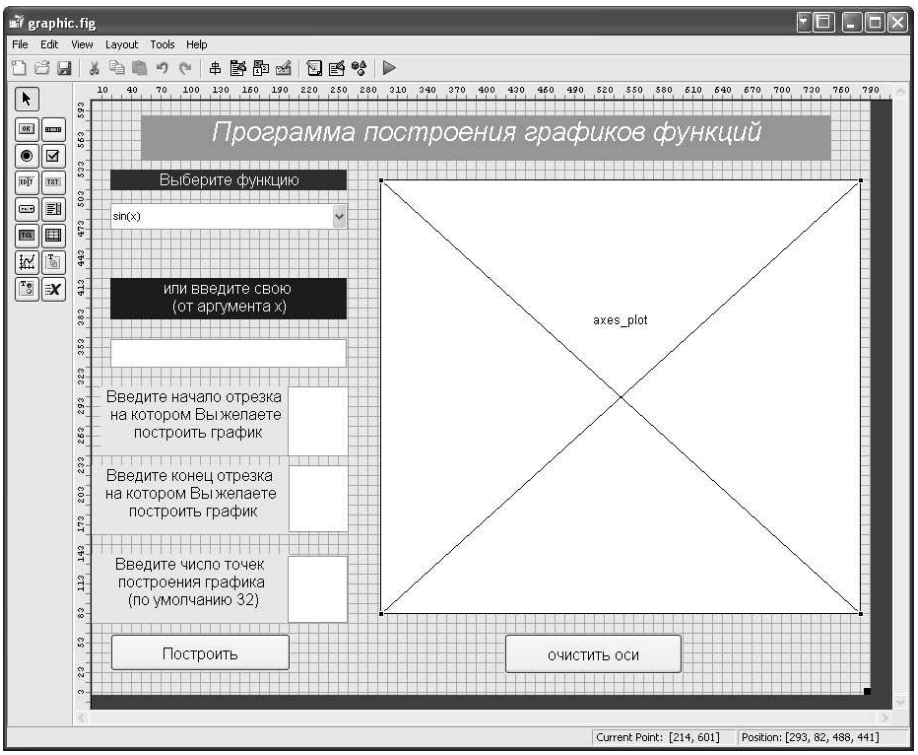

Рис. 10:

Замечание 3.2 *При размещении элементов управления следует выбрать соответствующий инструмент, а затем либо щелкнуть мышью по заготовке окна приложения (тогда элемент управления примет стандартный размер), либо*

нарисовать элемент управления на заготовке окна, удерживая нажатой левую кнопку мыши. Всегда можно изменить размер элемента управления, выделив его в заготовке окна при помощи мыши (должен быть включен режим выделения, то есть выбран соответствующий инструмент Select) и потянув мышью за квадратные маркеры. В режиме выделения возможно перемещать элементы управления по заготовке окна приложения.

При работе с объектами необходимо уметь обращаться к объектам для получения и установки значений их свойствам и программирования событий, возникающих при обращении пользователя к элементу управления, например, при нажатии на кнопку. При размещении пользователем элемента управления в заготовке окна системой GUIDE автоматически генерируется и присваивается созданному объекту приложения его уникальное имя, которое мы будем называть тегом. Значением свойства Tag является строка. Теги используются для доступа к свойствам объектов приложения (для получения и изменения значений свойств объектов приложения), чаще всего в подфункциях обработки различных событий других объектов. Конечно, можно оставить сгенерированный тег объекта и без изменений, но при написании объемного программного продукта, содержащего большое количество различных объ-

ектов, удобнее давать объектам мнемонические названия!

Для задания тега следует перейти к инспектору свойств. Это можно сделать двумя способами: либо двойным щелчком мыши по добавленному объекту, либо, щелкнув правой кнопкой мыши на объекте, выбрать из выпадающего меню строку Property Inspector. При этом появляется окно инспектора свойств (Property Inspector), в котором отображены свойства объекта (например, кнопки (объекта Uicontrol)). Найдем в левом столбце таблицы свойство Tag и в области ввода справа от него изменим автоматически сгенерированное значение pushbutton1 на btnPlot, затем нажмем клавишу <Enter>.

Присвоим тегам других объектов значения из таблицы 1.

За хранение надписи, содержащейся в объекте "текстовое поле", отвечает свойство String. При помощи Property Inspector в объектах Text (предназначенным для отображения текстовой информации) зададим свойству String значения, соответствующие тексту, изображенному на рис. 4. Отметим, что свойство String также отвечает за надписи на кнопках, текст, содержащийся в объекте "поле для ввода" и за текст в объекте "выпадающий список". Поэтому у объекта "поле для ввода" в свойстве String установим пустую строку, просто удалив в инспекторе свойств то значение, которое присваивается по умолчанию, а у объекта "выпадающий список (pop-up menu)"в

Таблица 1: Теги и их значения

Значение свойства Tag	**Описание**
txt_title	Текстовое поле: Программа построения графиков функций
txt_option_fcn	Текстовое поле: Выберите функцию
popupmenu_fcn	Выпадающий список
txt_enter_fcn	Текстовое поле: или введите свою (от аргумента x)
edt_fcn	Область ввода функции
txt_begin, txt_end, txt_step	Текстовые поля начало отрезка, конца отрезка и количества точек соответственно
edt_begin, edt_end, edt_step	Области ввода начало отрезка, конец отрезка и количества точек соответственно
btn_clear	кнопка очистить оси
axes_plot	оси
figure_graphic	Окно figure

свойстве String введем через <ENTER> следующие символьные строки, являющиеся названиями функций от аргумента x: sin(x), cos(x), tan(x), sqrt(x).

Замечание 3.3 *В нашем случае у объекта "поле для ввода" область ввода является однострочной – такой она добавляется по умолчанию. Область ввода может допускать ввод нескольких строк, разделяемых нажатием на клавишу <Enter>. Для того, чтобы сделать область ввода многострочной, следует в объекте "поле для ввода" установить свойствам Max и Min значения так, чтобы их разность была больше единицы.*

Замечание 3.4 *Для завершения ввода многострочного текста используется сочетание клавиш <Ctrl>+<Enter>. В случае однострочного текста значением свойства String является строка, а в случае многострочного – массив ячеек, каждый элемент которого содержит строку.*

При запуске программы нам не требуется отображение осей и кнопки с надписью "Очистить оси". Поэтому свойству Visible в объекте axes и кнопке "Очистить оси" установим значение "off".

Осталось задать цвет статического текста при помощи свойства ForegroundColor, цвет фона статического текста при помощи свойства BackgroundColor. Начертание шрифта статического текста и надписей на кнопках задается при помощи свойств, название которых начинается со слова Font. Размер

шрифта указывается в единицах измерения, заданных свойством FontUnits (по умолчанию – пункты: 1pt=1/72 дюйма).

Отметим, что все созданные объекты являются графическими объектами Uicontrol (это указано в верхней части окна инспектора свойств). Вид элемента управления, то есть объекта Uicontrol, определяется значением его свойства Style. Для объекта отображения статического текста свойство Style принимает значение 'text', для кнопки – 'pushbutton'.

Ниже приводится еще несколько возможных значений свойства Style

Значение свойства Style	**Описание**
pushbutton(по умолчанию)	кнопка
togglebutton	кнопка-переключатель
radiobutton	переключатель
checkbox	флаг
edit	область ввода текста
text	статический текст
slider	полоса скроллинга
listbox	список
popuptenu	выпадающий список

Замечание 3.5 *У всех объектов Uicontrol многие остальные свойства имеют схожие значения, например: Position, Units, FontName, FontSize. Некоторые свойства имеют раз-*

ный смысл - в зависимости от объекта, к которому это свойство относится. Если значение свойства String кнопки или флага – надпись на кнопке или флаге, то для списка – его содержимое.

Теперь, после того как мы полностью создали каркас приложения, его необходимо сохранить. Для этого в меню File среды GUIDE выберем пункт Save as; появится диалоговое окно сохранения файла, в котором выберем папку или создадим новую и зададим имя файла graphic (автоматически добавится расширение fig). Отметим, что при сохранении приложения автоматически создается файл graphic.m с таким же именем, но с расширением m. Этот файл открывается в редакторе М-файлов. Таким образом, наше приложение содержится в двух файлах: с расширением fig (графическое окно с размещенными на нем элементами управления) и с расширением m (файл-функция graphic с подфункциями, которые обрабатывают различные события, возникающие в ходе взаимодействия приложения с пользователем).

Рассмотрим теперь процесс обработки событий, возникающих в ходе работы пользователя с приложением.

Итак, при обращении пользователя к объекту (например, при нажатии кнопки "Построить") возникнет событие Callback. Поэтому свойству Callback выбранного объекта необ-

ходимо задать подфункцию обработки события Callback. Это можно сделать двумя способами. Первый способ – в программе (в файле с расширением m) задать в качестве значения свойства Callback указатель на подфункцию, обрабатывающую событие Callback, и описать указанную подфункцию. Второй способ – перейти к заготовке окна приложения и, вызвав контекстное меню (щелчок правой кнопкой мыши на кнопке), выбрать в пункте View Callbacks подпункт Callback. При этом происходит переход в редактор M-файлов к подфункции обработки события btnPlot_Callback, заголовок которой и комментарии генерируются автоматически:

```
% -- Executes on button press in btnGraphic.
function btnPlot_Callback(hObject, eventdata, handles)
% hObject handle to btnPlot (see GCBO)
% eventdata reserved - to be defined in a future version of
% MATLAB
% handles structure with handles and user data (see GUIDATA)
```

Затем требуется описать тело функции, то есть записать те операторы, которые будут выполняться при нажатии на кнопку "Построить".

Имя файл-функции состоит из тега объекта (btnPlot) и названия события Callback, которое будет обрабатываться (отметим, что есть и другие события). Эта функция имеет сле-

дующие входные аргументы.

Аргумент hObject содержит указатель на кнопку "Построить", то есть объект Uicontrol с тегом btnPlot (он нам сейчас не понадобится). Таким образом, hObject содержит идентификатор (handle) элемента управления, для которого вызвана процедура обработки.

Аргумент eventdata зарезервирован для использования в следующих версиях MATLAB.

Аргумент handles является структурой с указателями на все объекты приложения. Название полей структуры handles совпадают с названиями тегов созданных объектов. Например, handles.btnPlot содержит указатель на объект ”кнопка Построить”, handles.figure1 – указатель на окно приложения, handles.edt_fcn – указатель на объект ’поле для ввода текста’, handles.axes_plot – указатель на объект ’оси’, handles.btn_clear – указатель на кнопку "Очистить оси" и т.д.

Замечание 3.6 *В соответствии с парадигмой событийно-ориентированного программирования каждой компоненте управления сопоставлена одна или несколько процедур (callback) обработки событий. Вызов каждой такой процедуры связан с выполнением пользователем определенных действий (нажатие кнопки, выбор пункта меню, перемещение курсора мыши по элементам управления и т.п.). Содержа-*

ние callback-процедур определяет пользователь, разрабатывающий пользовательский интерфейс.

Параметрами вызова callback-процедуры, как правило, являются: hObject – идентификатор (handle) элемента управления, для которого вызвана процедура обработки; eventdata – параметр зарезервирован; handles – структура, содержащая идентификаторы всех графических элементов, присутствующих на заготовке окна.

Наиболее распространенными типами callback-процедур в GUIDE являются:

– ButtonDownFcn – вызывается путем нажатия кнопки мыши по компоненте управления, для которой установлено свойство Enable. В случае, если эта процедура используется для объекта Figure, процедура выполняется, когда пользователь производит щелчок мыши по области окна, не занятой другими объектами;

– WindowsButtonDownFcn, WindowsButtonMotionFcn – вызываются при нажатии пользователем кнопки мыши или при перемещении мыши (в пределах окна);

– Callback – вызывается при выполнении функционального действия компоненты (нажатия кнопки или выбор пункта меню);

– CloseRequestFcn – вызывается непосредственно перед закрытием формы;

– CreateFcn – вызывается при создании компоненты управления, но до отображения формы на экране. Шаблон, генерируемый GUIDE для данной процедуры, обычно реализует установку цвета отображения компоненты на экране;

– DeleteFcn – вызывается при удалении компоненты из памяти;

– KeyPressFcn – вызывается в случае, когда пользователь нажимает клавишу на клавиатуре, а компонента управления является текущей.

Приступим теперь к программированию события Callback кнопки "Построить".

Для начала нам необходимо получить значение того или иного свойства графического объекта. Для этого предназначена функция get, которая вызывается с входными параметрами – указатель на интересующий объект и название свойства. Ее выходным аргументом является значение данного свойства.

Итак, получим данные введенные пользователем в объекты "поле для ввода" и "выпадающий список". Как отмечалось ранее, доступ к тексту, введенному в объект "поле для ввода", можно получить из свойства String. Поэтому добавим в тело функции btnPlot_Callback следующую строку:

```
FcnName=get(handles.edt_fcn,'String');
```

```
% где FcnName - имя переменной, которая будет содержать
% введенную пользователем функцию.
```

Поскольку свойство 'String' принимает строковые значения, то для получения границ изменения аргумента придется не только считать строковые значения свойства String области ввода edt_begin, edt_end и edt_step, но и преобразовать их в числовые, обратившись к функции str2num.

```
  FcnBegin=str2num(get(handles.edt_begin,'String'));
% где FcnBegin - имя переменной, которая будет содержать начало
% отрезка, на  котором строится график.

  FcnEnd=str2num(get(handles.edt_end,'String'));
% где FcnEnd - имя переменной, которая будет содержать
% конец отрезка, на  котором строится график.

  FcnStep=str2num(get(handles.edt_step,'String'));
% где FcnStep - имя переменной, которая будет содержать
% количество точек, на  которых строится график.
```

Теперь рассмотрим процесс получения данных из объекта "выпадающий список". Как отмечалось в замечании 3.5, за содержимое в выпадающем списке отвечает свойство String. Получить это содержимое можно командой

```
  FcnPop=get(handles.popupmenu_fcn,'String');
```

В выпадающем списке может быть введено несколько последовательностей символов, при этом каждая последователь-

ность может иметь разное число символов, поэтому переменная FcnPop будет массивом, причем массивом ячеек.

Таким образом, команда

```
FcnPop=get(handles.popupmenu_fcn,'String');
```

даст следующий результат:

```
FcnPop =
'sin(x)'
'cos(x)'
'tan(x)'
'sqrt(x)'
```

Узнать, какая из строк в выпадающем списке является в данный момент текущей, можно с помощью свойства Value, то есть если указатель стоит на функции 'cos(x)', то команда

```
PopId=get(handles.popupmenu_fcn,'Value');
```

присвоит переменной PopId значение, равное 2.

Теперь нам осталось построить график введенной или выбранной функции и отобразить кнопку "Очистить оси".

Для построения графика функции одной переменной воспользуемся встроенной функцией fplot. Отметим, что при построении графика с помощью данной функции нет необходимости в вычислении значений функции и значений аргумента. Обращение к функции имеет следующий вид:

```
fplot('Имя функции', [интервал], n);
```

здесь имя функции – имя M-файла с текстом процедуры вычисления значения желаемой функции по заданному значению ее аргумента; интервал – вектор-строка из двух чисел, задающих соответственно нижнюю и верхнюю границы изменения аргумента; n – количество частей, на которые будет разбит указанный интервал. По умолчанию n=25.

Чтобы отобразить кнопку "Очистить оси" нам необходимо задать свойству Visible значение 'on'. Свойства графических объектов задаются при помощи функции set. Указанная команда имеет следующий вид:

```
set(handles.btn_clear,'Visible','on');
```

Таким образом, подфункция btnPlot_Callback должна выглядеть следующим образом:

```
function btnPlot_Callback(hObject, eventdata, handles);
% hObject handle to pushbutton1 (see GCBO) eventdata
% reserved -- to  be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
FcnName=get(handles.edt_fcn,'String');
% Проверим, введено ли в "поле для ввода" название функции
  if isempty(FcnName)
    FcnPop=get(handles.popupmenu_fcn,'String');
    PopId=get(handles.popupmenu_fcn,'Value');
```

```
% Обратите внимание, что FcnPop -- массив ячеек, а доступ к
% элементам массива ячеек осуществляется через фигурные скобки
    FcnName=FcnPop{PopId};
  end;
  FcnBegin=str2num(get(handles.edt_begin,'String'));
  if isempty(FcnBegin) FcnBegin=0; end;
  FcnEnd=str2num(get(handles.edt_end,'String'));
  if isempty(FcnEnd) FcnEnd=1; end;
  FcnStep=str2num(get(handles.edt_step,'String'));
  fplot(FcnName,[FcnBegin FcnEnd],FcnStep);
  set(handles.btn_clear,'Visible','on');
```

Задание 3.1 *Самостоятельно запрограммируйте событие Callback кнопки "Очистить оси". Данная подфункция должна не только очищать оси, но и очищать поля для ввода. Для очистки осей можно воспользоваться функцией cla.*

Теперь приложение graphic можно запустить, воспользовавшись зеленой кнопкой Run на панели управления среды GUIDE и пунктом Run в меню Debug. Перед запуском может появиться окно, приведенное на рис. 11, с сообщением о том, что папка с файлами приложения не является текущей. В этом окне можно либо сделать эту папку текущей (переключатель Change MATLAB current directory), либо добавить папку в начало пути поиска MATLAB (переключатель Add directory to

the top of the MATLAB path), либо в конец пути поиска (переключатель Add directory to the bottom of the MATLAB path). Установим верхний переключатель (как по умолчанию); вряд ли стоит добавлять папку с таким простым приложением в путь поиска. Скорее всего оно будет нужно не часто.

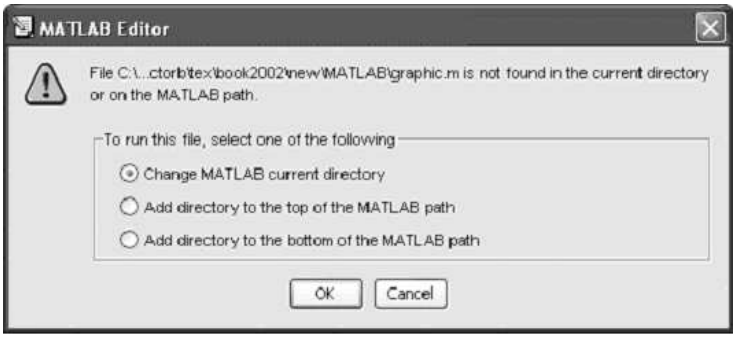

Рис. 11: Диалоговое окно с сообщением о том, что папка с файлами приложения не является текущей.

Нажатие на кнопку "Построить" в работающем приложении приводит к появлению графика в правой части окна приложения. Закроем работающее приложение, нажав на кнопку с крестиком на заголовке окна. Мы снова находимся в режиме редактирования. Можно вставлять элементы управления в заготовку окна приложения, задавать их теги и другие свойства, программировать события, запускать приложение и смотреть на результат. В качестве упражнения уменьшим окно приложения в среде GUIDE и запустим приложение снова.

Обратим внимание, что если приложение работает, то его повторный запуск из среды GUIDE не приводит к появлению второго окна приложения.

В этом разделе мы обсудили создание простейшего приложения, принимая ряд опций приложения с графическим интерфейсом такими, какими их по умолчанию предлагает среда GUIDE.

Среда GUIDE обладает рядом средств, которые облегчают проектирование приложения:

– сетка с возможностью привязки объектов к ней, линейки и линии выравнивания (меню Tools, пункт Grid and Rules);

– инструменты выравнивания объектов (меню Tools, пункт Align Objects или кнопка Align Objects на панели управления среды GUIDE).

В среду GUIDE входят также :

– редактор меню, который позволяет создавать меню приложения и контекстные меню (меню Tools, пункт Menu Editor или кнопка Menu Editor на панели управления среды GUIDE);

– браузер объектов для быстрого перехода к их свойствам (кнопка Object Browser на панели управления среды GUIDE);

– редактор порядка обхода элементов управления клавишей Tab (меню Tools, пункт Tab Order Editor или кнопка Tab Order

Editor на панели управления среды GUIDE).

Вызов инспектора свойств происходит не только после двойного щелчка мышью по объекту, но также из контекстного меню объекта, или при помощи кнопки Property Inspector на панели управления среды GUIDE.

3.1. Запуск приложения без использования среды GUIDE и редактирование

Разумеется, созданное в предыдущем разделе приложение graphic не требует для запуска среду GUIDE. Закроем окно среды GUIDE (если оно открыто) и перейдем в командное окно MATLAB. Убедимся в том, что папка с приложением является текущей (она должна быть выбрана в раскрывающемся списке Current Directory рабочей среды MATLAB). Если это не так, то сделаем ее текущей, воспользовавшись кнопкой справа от раскрывающегося списка Current Directory или окном Current Directory.

Для запуска приложения достаточно набрать его имя в командной строке и нажать <Enter>:

» graphic

Приложение можно вызывать не только из командной строки, но и из любого другого приложения, при этом MATLAB должен быть способен найти путь к нему, то есть папка с при-

ложением должна быть текущей, или содержаться в пути поиска MATLAB

Для редактирования существующего приложения с графическим интерфейсом пользователя есть два способа:

1. Запуск среды GUIDE командой guide и переход в появившемся диалоговом окне GUIDE Quick Start (см. рис. 7) к вкладке Open Existing GUI с последующим выбором требуемого приложения.

2. Вызов guide с указанием в качестве входного аргумента имени файла с приложением:

```
» guide('graphic.fig')
```

При таком способе папка с приложением должна быть текущей или содержаться в пути поиска MATLAB.

3.2. Как работает приложение, созданное в среде GUIDE?

Это важный вопрос для тех, кто хочет создавать сложные приложения. Если Ваша цель состоит в написании простых приложений, то достаточно научиться размещать элементы интерфейса и программировать их события в подфункциях так, как описано выше.Простое приложение состоит из одного основного окна, которое содержит различные элементы управления, области вывода текстовой информации и оси. Использование стандартных диалоговых окон облегчает работу с

файлами, ввод данных, выбор шрифта, цвета и печать результатов. Однако, если Вы планируете программировать многооконные приложения, то прочтите этот раздел сейчас или вернитесь к нему по мере надобности.

Обсудим работу приложения с графическим интерфейсом пользователя graphic, созданию которого посвящен раздел "Создание приложения graphic в среде GUIDE". Перейдем в режим редактирования приложения одним из способов, приведенных в предыдущем разделе, например:

```
» guide('graphic.fig')
```

Откроем в редакторе М-файлов файл-функцию graphic. Это функция с переменным числом входных и выходных аргументов, ниже она приведена без автоматически созданных комментариев и подфункций.

```
function varargout = graphic(varargin)
  gui_Singleton = 1;
   gui_State = struct( 'gui_Name',mfilename,...
                       'gui_Singleton',gui_Singleton,...
                       'gui_OpeningFcn',@graphic_OpeningFcn,...
                       'gui_OutputFcn',@graphic_OutputFcn,...
                       'gui_LayoutFcn',[],...
                       'gui_Callback',[]);
  if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
  end
```

```
if nargout
  [varargout{1:nargout}] = gui_mainfcn(gui_State,...
  varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
```

Поставим точку остановки в редакторе M-файлов в строке с первым исполняемым оператором gui_Singleton = 1. Запустим приложение graphic, например, из командной строки:

```
» graphic
```

и выполним операторы graphic по шагам, используя клавишу <F10> (или кнопку <Step>).

Сначала переменной gui_Singleton присваивается значение 1, затем формируется структура gui_State с полями:

- gui_Name – имя M-файла с работающей в данный момент файл-функцией приложения, которое возвращается с помощью функции mfilename;
- gui_Singleton – количество копий приложения, которые могут быть запущены одновременно – в нашем случае это поле содержит 1, и это означает, что может быть запущена только одна копия приложения (0 означает, что могут быть одновременно запущены несколько копий);

- gui_OpeningFcn – указатель на подфункцию, выполняемую перед тем, как окно приложения появится на экране (в файле graphic.m – подфункция graphic_OpeningFcn);
- gui_OutputFcn – указатель на подфункцию (в файле с именем graphic.m – подфункция graphic_OutputFcn), которая определяет, что возвращает функция graphic, вызванная с выходным аргументом (по умолчанию – указатель на графическое окно приложения);
- gui_LayoutFcn – пустой массив (по умолчанию) – может быть указателем на функцию, которая определяет способ появления приложения;
- gui_Callback – пустой пока массив. При возникновении события, получаемого с помощью некоторого элемента управления, он будет содержать указатель на функцию graphic с необходимыми входными аргументами, которые и определяют исполняемую подфункцию в graphic.m.

После заполнения структуры gui_State проверяется, была ли функция graphic вызвана с входными аргументами (nargin равно числу входных аргументов) и является ли первый из них строкой. При запуске приложения входных аргументов не было. Они появляются при возникновении событий от элементов управления. Действительно, если в инспекторе свойств отоб-

разить свойства кнопки "Построить" и посмотреть, чему равно значение ее свойства Callback, то станет понятно, что при возникновении события Callback кнопки вызывается функция btnPlot: graphic('btnPlot_Callback',gcbo,[],guidata(gcbo)). Тогда в поле gui_Callback структуры gui_State заносится соответствующий указатель при помощи функции str2func.

Замечание 3.7 *Функция str2func конструирует указатель на функцию, заданную строкой, например:*

```
» f=str2func('exp')

f = @exp
```

Следующий оператор if проверяет, была ли функция graphic вызвана с выходными аргументами (значение nargout равно числу выходных аргументов), и вызывает специальную функцию gui_mainfcn от структуры gui_State и входных аргументов graphic.

При первом вызове входных аргументов не было, и функция gui_mainfcn создает окно приложения. Последующие вызовы graphic с входными аргументами, связанные с возникновением событий от элементов управления, приведут к обращению к соответствующим подфункциям обработки событий в graphic.m. Это можно проследить путем пошагового выполнения программы в редакторе M-файлов.

4 Создание меню и работа с диалоговыми окнами в среде Guide

После того, как мы создали простое приложение, состоящее из одного основного окна, содержащего различные элементы управления: области ввода и вывода текстовой информации, выпадающий список, кнопки и оси, – усовершенствуем функциональные возможности нашего приложения, добавив в него новые элементы управления интерфейсом. Создадим приложение, позволяющее пользователю с помощью меню, а также контекстного меню, устанавливать цвет выводимого графика, задавать цвет текстовых полей и сохранять полученную картинку в заданный файл. При этом в строке меню будет всего два поля File и Format.

1. Меню File с пунктами:

1.1. Save (при выборе этого пункта появится диалоговое окно для сохранения файла, и после ввода имени файла построенный график функции будет сохранен на жесткий диск);

1.2. Exit (для закрытия окна приложения).

2. Меню Format с пунктом:

2.1. Line, имеющим следующие подпункты:

2.1.1 Color – при выборе данного пункта появится диалоговое окно выбора цвета линии;

2.1.2 Width – при выборе данного пункта появится диалоговое окно задания толщины линии.

На рис. 12 изображены пункты меню, а также контекстное меню.

Рис. 12: Приложение с меню.

Прежде, чем создавать меню, изучим принципы работы со стандартными диалоговыми окнами.

Использование стандартных диалоговых окон облегчает процесс разработки интерфейса прикладной программы. Стандартные диалоговые окна в среде GUIDE позволяют пользователю работать с файлами, вводить данные, выбирать шрифт, цвет и выводить на печать результаты. Кроме того, существуют окна вывода различной информации пользователю (подсказки, сообщения об ошибках и т.д.).

В MATLAB имеется 15 функций для создания стандартных диалоговых окон. Вид диалоговых окон может быть определен путем задания входных аргументов этих функций. Полный список названий функций, создающих диалоговые окна,

можно найти в приложении 1 к настоящему пособию. В данном приложении мы ограничимся подробным рассмотрением лишь диалогового окна выбора цвета и окна сохранения файла.

Поскольку графика MATLAB обеспечивает получение цветных изображений, имеется ряд команд для управления цветом и различными цветовыми эффектами. Среди них важное место занимает команда установки палитры цветов. Палитра цветов RGB задается матрицей из трех столбцов, определяющих значения интенсивности красного (red), зеленого (green) и синего (blue) цветов. Их интенсивность задается в относительных единицах от 0.0 до 1.0. Например, [0 0 0] задает черный цвет, [1 1 1] – белый цвет, [0 0 1] – синий цвет. При изменении интенсивности цветов в указанных пределах возможно задание любого цвета. Таким образом, цвет соответствует общепринятому формату RGB.

Диалоговое окно выбора цвета отображается в результате выполнения функции uisetcolor.

Функция uisetcolor может использоваться в двух целях:

1) Для создания диалогового окна выбора цвета и записи выбранного цвета в вектор в формате RGB;

2) Для создания диалогового окна выбора цвета и измене-

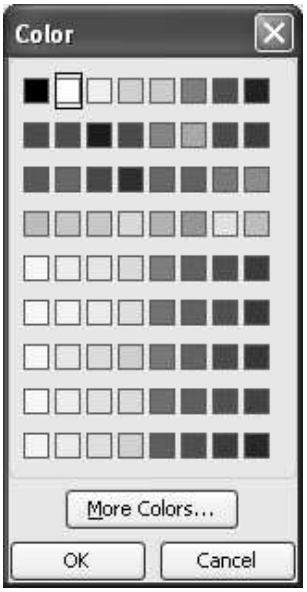

Рис. 13: **Диалоговое окно выбора цвета**

ния графических объектов, обладающих свойством Color, в соответствии со сделанными в этом окне установками.

Вызов функции uisetcolor может иметь следующий вид:

```
C=uisetcolor;
```

Эта команда приводит к появлению диалогового окна выбора цвета (по умолчанию текущим является белый цвет), и при выборе пользователем цвета происходит запись этого цвета в вектор C в формате RGB. Если пользователь не выбрал цвет и закрыл окно, то возвращаемое функцией значение равно нулю.

Функция uisetcolor допускает вызов с входными значениями. В качестве входного допускается передавать вектор, значения которого характеризует интенсивность цветов в относи-

тельных единицах от 0.0 до 1.0. Следующая команда приводит к появлению диалогового окна выбора цвета, в котором по умолчанию текущим является цвет, заданный в векторе C0:

```
C = uisetcolor(C0);
```

Кроме того, функция uisetcolor предоставляет возможность вывода диалогового окна выбора цвета с заданным заголовком. Для этого в качестве входного аргумента в функцию требуется передать строку, например:

```
C = uisetcolor('Заголовок окна');
```

или

```
C = uisetcolor(C0, 'Заголовок окна');
```

Для установки цвета у ранее созданных графических объектов, требуется в качестве входного аргумента передать указатель на объект (графический объект должен иметь свойство Color):

```
uisetcolor(h);
```

или

```
uisetcolor(h, 'Заголовок окна');
```

или

```
C = uisetcolor(h,'Заголовок окна');
```

Отметим, что в последнем примере, кроме изменения цвета объекта, информация о выборе пользователем цвета записы-

вается в вектор C в формате RGB.

Приведем пример изменения цвета графических объектов.

```
hA = axes;
x = 0:0.1:10;
y = sin(x);
hL = plot(x,y);
hT = title('Random Graph');
uisetcolor(hL, 'Select line color');
uisetcolor(hT, 'Select title color');
uisetcolor(hA, 'Select axes color');
```

Теперь рассмотрим процесс создания диалогового окна сохранения файла.

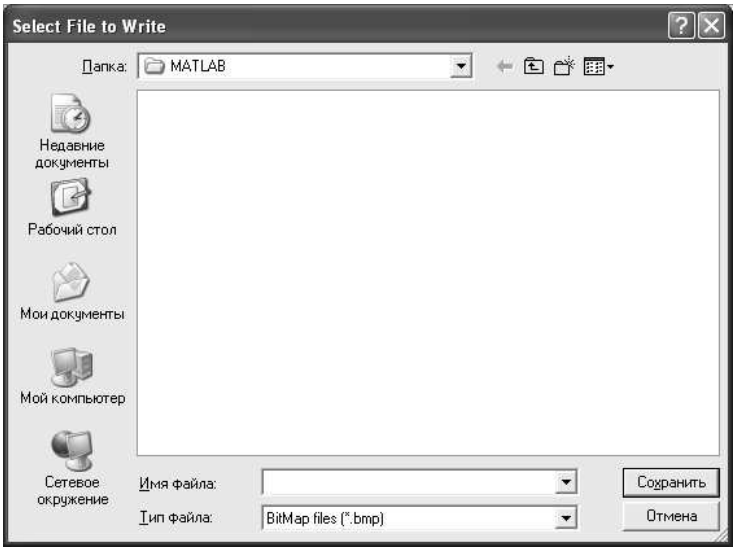

Рис. 14: **Диалоговое окно сохранения файла**

Функция uiputfile создает диалоговое окно сохранения фай-

ла с содержимым текущего каталога. Фильтр файлов установлен в ALL MATLAB files; отображаются только те файлы, расширения которых поддерживаются MATLAB. В раскрывающемся списке Files of type можно выбрать только M-файлы или только графические окна, или все файлы.

Обращение к функции имеет следующий вид:

```
[FileName, DirName] =uiputfile;
```

Если в диалоговом окне сохранения файла пользователь выберет имя существующего файла, то появится окно подтверждения. Нажатие на кнопку <Yes> приводит к завершению диалога сохранения файла, а нажатие на кнопку <No> – к возврату в окно сохранения файла. Если пользователь выбрал файл или набрал имя файла в строке "Имя файла" и нажал кнопку <Сохранить>, то FileName будет содержать строку с именем файла и соответствующим расширением, а DirName – путь к файлу. Если пользователь не выбрал файл и закрыл окно нажатием на кнопку <Отмена>, то FileName = 0 и DirName = 0. Поэтому после обращения к функции uiputfile следует проверить, был ли выбран файл. Если была нажата кнопка <Сохранить>, то необходимо объединить полученные строки в полный путь к файлу:

```
[FName, DirName] = uiputfile;
if ~ isequal(FName, 0)
```

```
FullName = strcat(DirName, FName);

end;
```

Замечание 4.1 *Функция uiputfile не записывает данные на жесткий диск. С ее помощью возможно лишь отобразить диалоговое окно сохранения данных и получить путь к файлу, в который пользователь хочет сохранить полученный результат. Для сохранения данных необходимо использовать, например, функции save, print, fwrite, save as.*

Обращение к функции uiputfile может быть осуществлено с входными параметрами. В качестве первого входного параметра указывается фильтр файлов – например, для размещения описания типов файлов в списке Files of Type следует указывать массив ячеек из двух столбцов. В каждой его строке задается расширение файла и его описание:

```
Filter={'*.txt', 'Text files (*.txt)';...
        '*.dat', 'Data files (*.dat)'; ...
        '*.res', 'Results (*.res)';...
        '*.*','AllFiles (*.*)'};
[FileName, DirName] = uiputfile(Filter);
```

Если необходимо узнать, какой по счету тип файлов пользователь выбрал в списке Files of Type, то следует обратиться к uiputfile с тремя выходными аргументами:

```
[FileName, DirName, FilterIndex] = uiputfile(Filter);
```

Переменной FilterIndex функция uiputfile присваивает но-

мер выбранной строки в массиве ячеек Filter.

Для задания собственного заголовка окна открытия файла (вместо Select File to Open) функция uiputfile вызывается с вторым входным аргументом:

```
[FileName, DirName] = uiputfile(Filter, 'Сохранить File');
```

Для помещения заданного имени файла в строку File Name при создании окна сохранения файлов необходимо указать его в третьем входном аргументе (имя файла может быть полным):

```
[FileName, DirName] = ...
uiputfile(Filter, 'Сохранить File', 'output.txt');
```

Таким образом можно отобразить содержимое любой папки, например, корневого каталога диска D:

```
[FileName, DirName] =...
uiputfile(Filter, 'Сохранить File', 'd:\');
```

Замечание 4.2 *Если мы хотим сохранять файлы MATLAB так, как предлагает uiputfile по умолчанию, но задать собственный заголовок окна (и папку или файл, предлагаемые по умолчанию в диалоговом окне открытия файлов), то первый входной аргумент – фильтр – пропускать нельзя. Он должен быть пустой строкой:*

```
[FileName, DirName] = uiputfile('', 'Сохранить File', 'd:\');
```

Теперь рассмотрим процесс создания приложения с меню и контекстными меню при помощи среды визуального программирования GUIDE.

Продолжим работу с приложением, начатую в предыдущем параграфе. В нем мы рассмотрели вопросы запуска среды GUIDE, создания заготовки окна приложения с управляющими объектами, сохранения приложения и обработки возникающих событий.

Добавим в приложение меню со следующими пунктами:

1. Меню File с пунктом:

1.1. Save (При выборе этого пункта меню появится диалоговое окно сохранения файла).

2. Меню Format с пунктом:

2.1. Line, имеющим подпункты:

2.1.1 Color (при выборе этого пункта в меню появится диалоговое окно задания цвета линии);

2.1.2 Width (при таком выборе в меню появится диалоговое окно задания толщины линии).

Откроем редактор меню, выбрав в меню Tools среды GUIDE пункт Menu Editor (или нажав на кнопку Menu Editor) на горизонтальной панели инструментов среды GUIDE. При этом появляется окно редактора меню, изображенное на рис. 15.

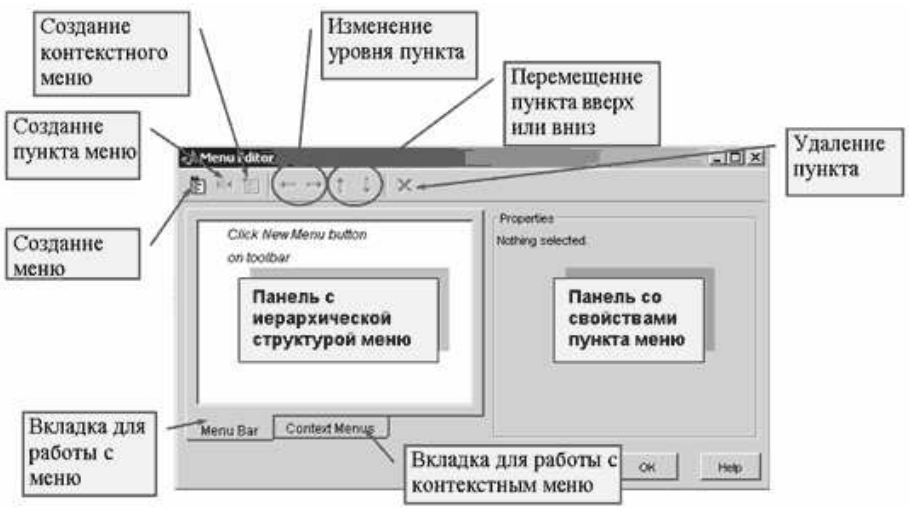

Рис. 15: Окно редактора меню

Создадим меню File. Для этого следует нажать на кнопку <New Menu>, находясь на вкладке Menu Bar. Сначала на левой панели с иерархической структурой меню появляется меню Untitled1.

Если на левой панели выделить при помощи мыши заголовок Untitled1, то на правой панели со свойствами отобразятся свойства, значения которых и следует задать.

Зададим свойству Label, отвечающему за надпись в строке меню, значение File, а свойству Tag (уникальному идентификатору) – значение Menu_File. На рис. 16 приведены основные элементы управления окна Menu Editor.

Для быстрого выбора пункта меню в списке Accelerator можно задать сочетание клавиш <Ctrl> + <буквенная клавиша> (нежелательно переопределять стандарт-

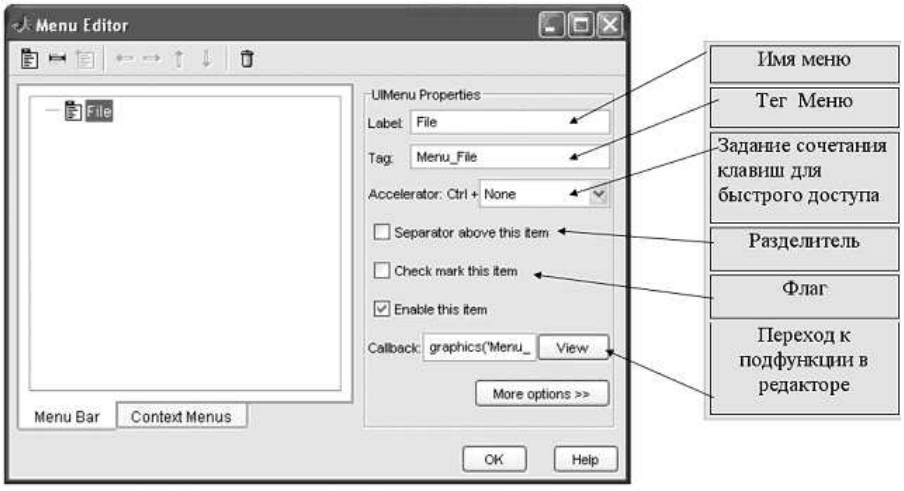

Рис. 16: Элементы управления окна Menu Editor

ные для Windows сочетания <Ctrl>+<c>, <Ctrl>+<v>, <Ctrl>+<x>) . Следует иметь в виду, что сочетание клавиш задается только для тех пунктов меню, которые не имеют потомков (то есть последних в иерархии). В нашем примере в меню File можно задавать сочетание клавиш для быстрого доступа к пункту Save.

Над пунктом меню можно поставить горизонтальную линию для формирования групп пунктов в меню. Для этого необходимо установить флаг Separator above this item.

Если требуется создать пункт меню с включенным флагом (который сбрасывается или включается при каждом последующем выборе пользователем этого пункта), то следует установить флаг Check mark this item. При этом в самом начале работы приложения этот пункт будет с включенным флагом.

Для того, чтобы в работающем приложении меню или пункты были доступны, необходимо убедиться, что установлен флаг Enable this item.

В строке ввода Callback после нажатия кнопки View будет записана функция, которая вызывается при выборе пользователем соответствующего пункта меню (пока ее нажимать не надо). После нажатия кнопки View в редакторе меню запускается редактор m-файлов, в котором создается и выделяется заголовок соответствующей подфункции. В нашем примере – это заголовок:

```
function Menu_File(hObject, eventdata, handles);
```

Программировать эту подфункцию не нужно, поскольку действия должны выполняться после выбора не меню, а одного из его пунктов.

Добавим теперь в меню File пункт Save. Для этого на панели с иерархией меню в редакторе меню надо сделать меню File текущим при помощи щелчка мыши по нему и добавить пункт, нажав на кнопку New Menu Item. Затем зададим свойству Label значение Save, а свойству Tag – значение Menu_Save. Для быстрого доступа зададим сочетание клавиш <Ctrl>+<S>.

Сгенерируем для пункта Save подфункцию обработки его

события Callback. Для этого нажмем кнопку View. В результате, в m-файле с расширением .m должен появиться заголовок подфункции, Menu_Save_Callback, которая будет выполняться при выборе пользователем пункта Save в меню File:

```
function Menu_Save_Callback(hObject, eventdata, handles);

% hObject    handle to Menu\_Save (see GCBO)  eventdata
% reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Замечание 4.3 *Иерархию пунктов и порядок можно изменять при помощи кнопок с горизонтальными и вертикальными стрелками на панели инструментов редактора меню. Для передвижения выделенного пункта меню или меню по иерархии вниз или вверх применяются кнопки Move Selected Item Backward и Move Selected Item Forward, а для перемещения вверх или вниз по списку – кнопки Move Selected Item Up и Move Selected Item Down. Для удаления выделенного пункта меню или меню целиком служит кнопка Delete Selected Item.*

Осталось добавить меню Format с пунктом Line имеющим подпункты Color, Width. Для добавления нового меню следует нажать на кнопку New Menu (не важно, что при этом выделено: меню или пункт) и действовать аналогично вышеописанному.

Зададим следующие теги:

– для меню Format – тег Menu_Format;

– для пункта Line – тег Menu_Line,

– для подпунктов Color и Width теги Menu_Color и Menu_Width соответственно.

Нажмем у подпунктов Color и Width в строке ввода Callback кнопку View.

В результате в m-файле должны появиться заголовки двух подфункций: Menu_Color_Callback, которая выполняется при выборе пользователем подпункта Color в пункте Line меню Format; Menu_Width_Callback, которая будет выполняться при выборе пользователем подпункта Width в пункте Line меню Format.

Окно редактора с иерархической структурой меню должно выглядеть так, как показано на рис. 17.

Если теперь на панели инструментов среды GUIDE нажать кнопку <Run>, то запустится наше приложение graphic, в нем можно раскрывать меню и выбирать их пункты. Разумеется, пока происходить ничего не будет. Осталось запрограммировать действия, которые должны выполняться при выборе пунктов меню пользователем.

Сначала запрограммируем действие при вызове пользовате-

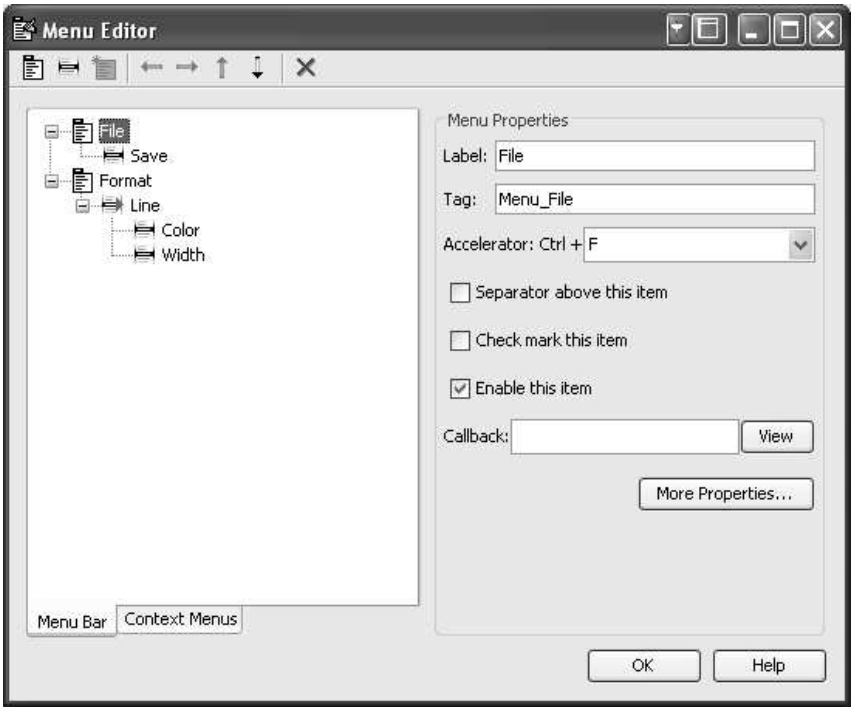

Рис. 17: Иерархическая структура меню

лем пункта Save в меню File. Обработка этого действия будет происходить в подфункции Menu_Save_Callback. В начале работы этой функции должно появиться диалоговое окно записи файла, которое вернет путь и название данного файла. После этого необходимо сохранить нашу фигуру на жестком диске.

Приведем полный текст функции Menu_Save_Callback.

```
function Menu_Save_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Save (see GCBO)
% eventdata reserved - to be defined in a future
% version of MATLAB
% handles structure with handles and user data (see GUIDATA)
Filter={'*.bmp','BitMap files (*.bmp)';'*.eps','Eps files'};
```

```
[FName, DirName, FilterIndex] = uiputfile(Filter);
if ~ isequal(FName, 0)
  FullName = strcat(DirName, FName);
  saveas(gcf, FullName, Filter{FilterIndex}(3:end));
end;
```

Замечание 4.4 *При сохранении окна figure мы использовали функцию saveas. На самом деле в системе MatLab сохранение на диск окна figure реализовано при помощи стандартной функции print, которая считывает текущее графическое окно на экране (по аналогии с сочетанием клавиш <Alt> + <Print Screen>) и записывает его на диск. Для корректного сохранения необходимо установить значение auto свойству PaperPositionMode окна figure. Кроме того, необходимо уметь работать с позицией объекта "окно figure" на экране.*

За расположение объекта отвечает свойство Position. Его значением является вектор из четырех чисел [x y width height], где x – абсцисса левого нижнего угла осей; y – ордината левого нижнего угла осей; width – ширина осей;height – высота осей.

Эти величины задаются в системе координат графического окна с началом в его левом нижнем углу. Единицы измерений по умолчанию являются нормализованными, т. е. как высота, так и ширина графического окна, полагаются равны-

ми единице (можно выбрать и другие единицы измерений).

Опишем тело функции Menu_Line_Callback. При выборе пользователем подпункта Line пункта Color меню Format должно появиться диалоговое окно выбора цвета линии. После того, как пользователь выберет требуемый цвет, этот цвет запишется в переменную My_Color. Значение этой переменной необходимо будет передать в функцию btnPlot_Callback, в которой и происходит построение графика.

Итак, наша функция будет иметь вид:

```
function Menu_Line_Callback(hObject,eventdata, handles);
% hObject handle to Menu_Color (see GCBO)
% eventdata reserved - to be defined in a future
% version of MATLAB
% handles structure with handles and user data (see GUIDATA)

My_Color=uisetcolor;
```

В результате значением переменной My_Color будет матрица, задающая интенсивность палитры цветов в формате RGB. Возникает вопрос: каким образом передать значение переменной My_Color в функцию btnPlot_Callback? Ответ на этот вопрос будет дан в следующем параграфе.

5 Обмен данными между подфункциями обработки событий

Необходимость обмена данными между подфункциями обработки событий возникает очень часто при программировании приложений с графическим интерфейсом. Этот обмен можно осуществлять двумя способами. Первый способ состоит в следующем: данные, которые требуется сохранить при выполнении некоторой подфункции, записываются в поля структуры handles; при этом желательно создать новое поле и уже в него сохранить данные. Затем в этой же подфункции структура handles сохраняется при помощи функции guidata. Функция guidata работает следующим образом:

функция guidata(object_handle, handles) сохраняет в структуру handles данные приложения, где первый входной аргумент object_handle должен быть указателем на окно приложения или одного из его потомков;

Команда handles = guidata(object_handle) записывает в handles структуру с данными приложения, входной аргумент object_handle также должен быть указателем на окно приложения или одного из его потомков;

Поэтому в подфункциях обработки событий элементов управления мы будем писать

```
guidata(gcbo, handles);
```

В качестве первого входного параметра в функцию передается указатель на текущий базовый объект (при помощи функции gcbo или handles.figure_graphic), второй входной параметр – структура – handles, которую необходимо сохранить.

Для получения данных в другой подфункции следует обратиться к соответствующему полю структуры handles.

Приведем полный текст наших подфункций:

```
function Menu_Line_Callback(hObject,eventdata, handles)
% hObject handle to Menu_Color (see GCBO)
% eventdata reserved - to be defined in a future
% version of MATLAB
% handles structure with handles and user data (see GUIDATA)
My_Color=uisetcolor;
% Создаем новое поле в структуре handles и сохраняем в него
% значение переменной My_Color
handles.new_My_color=My_Color;
% Сохраняем структуру handles при помощи функции guidata.
guidata(handles.figure_graphic, handles)

function btnPlot_Callback(hObject, eventdata, handles)
% hObject handle to btnPlot (see GCBO)
% eventdata reserved - to be defined in a future
% version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
FcnName=get(handles.edt_fcn,'String');
if isempty(FcnName)
  FcnPop=get(handles.popupmenu_fcn,'String');
  PopId=get(handles.popupmenu_fcn,'Value');
  FcnName=FcnPop{PopId}; end;
FcnBegin=str2num(get(handles.edt_begin,'String'));
if isempty(FcnBegin) FcnBegin=0; end;
FcnEnd=str2num(get(handles.edt_end,'String'));
if isempty(FcnEnd) FcnEnd=1; end;
FcnStep=str2num(get(handles.edt_step,'String'));
if isempty(FcnStep) FcnStep=10; end;
My_Color=[0 0 0];
% Проверяем, существует ли поле в структуре handles
if isfield(handles,'new_My_Color')
  My_Color=handles.new_My_Color;
end
[x,y]=fplot(FcnName,[FcnBegin FcnEnd],FcnStep);
plot(x,y,'Color',My_Color)
set(handles.btn_clear,'Visible','on');
```

Запустите наше приложение и посмотрите как оно работает.

Второй способ основан на использовании глобальных переменных. Для этого предварительно требуется задать значения цвета по умолчанию. Задание начальных значений лучше производить в функции, выполняющейся при открытии окна

приложения. Поэтому в функцию graphic_OpeningFcn добавим следующие строки:

```
global My_Color;

My_Color=[0 0 0];
```

Затем, добавим строку "global My_Color;" в функции обработки событий btnPlot_Callback и Menu_Line_Callback.

После этого, закоментируем в функции Menu_Line_Callback следующие строки:

```
My_Color=[0 0 0];
if isfield(handles,'new_My_Color')
My_Color=handles.new_My_Color;
end;
```

Запустим наше приложение и проверим его работоспособность.

6 Создание контекстного меню

В этой части мы опишем процесс создания контекстного меню, имеющего два пункта Save и Line с подпунктами Color и Width, выбор которых приводит к сохранению полученного графика или заданию цвета линии и шрифта. Кроме того, пункты контекстного меню и пункты меню должны вести себя согласованно.

Для того чтобы в среде GUIDE добавить приложению кон-

Рис. 18: Приложение с меню.

текстное меню, следует выполнить три действия:

1) создать контекстное меню вместе с его пунктами в редакторе меню среды GUIDE, дать им имена и теги и при необходимости, установить флаги и горизонтальные разделители между группами пунктов меню;

2) связать контекстное меню с объектом окна приложения, установив тег контекстного меню в качестве значения свойства UIContextMenu объекта;

3) запрограммировать события Callback пунктов контекстного меню в автоматически созданных подфункциях приложения.

Событие Callback пункта контекстного меню возникает при его выборе пользователем. Для создания контекстного меню необходимо в меню Tools среды GUIDE выбрать пункт Menu Editor, что приведет к появлению редактора меню. В редакторе меню надо перейти на вкладку Context Menus и при помощи кнопки New Context Menu создать контекстное меню. Для

добавления в него пунктов служит кнопка New Menu Item.

Создадим пункты контекстного меню согласно рис. 19.

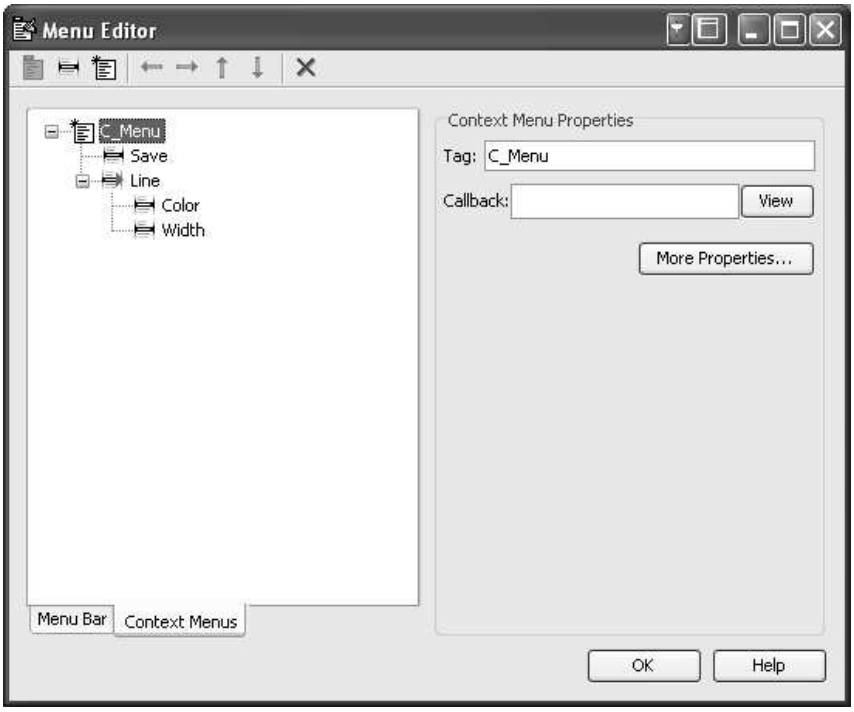

Рис. 19: Приложение с меню.

Замечание 6.1 *В целом, процесс создания контекстного меню в редакторе меню не отличается от способа создания обычного меню. Для изменения последовательности пунктов в контекстном меню служат две кнопки – повысить уровень – <Move Selected Item Backward> и понизить уровень – <Move Selected Item Forward>. Перемещение вверх или вниз пункта меню производится при помощи кнопок Move Selected Item Up и Move Selected Item Down. Удаление ненужного пункта меню или меню в целом, осуществляется на-*

жатием на кнопку Delete Selected Item.

На правой панели в окне редактора меню вводятся имя контекстного меню, имена и теги его пунктов. Введем следующие имена и теги:

Значение свойства Tag	**Пункт контекстного меню**
C_Menu	Тег контекстного меню (понадобится при указании для какого объекта будет вызываться данное меню).
CSave	пункт Save
CColor	пункт Color
CLine	пункт Line
CWidth	пункт Width

Итак, контекстное меню создано. Теперь требуется указать, на каком объекте при щелчке правой кнопкой мыши, будет вызываться данное меню.

Для связывания контекстного меню с объектом следует перейти к свойствам объекта в окне инспектора свойств и установить значение свойства UIContextMenu, равным значению тега контекстного меню, то есть CMenu, выбрав его из раскрывающегося списка справа от названия свойства.

Поскольку контекстное меню вызывается как при щелчке

мыши на окне приложения, так и при щелчке на осях, то зададим свойству UIContextMenu объектов figure и axes значение CMenu.

Теперь осталось только запрограммировать событие Callback, которое программируется точно так же, как и события Callback пунктов строки меню.

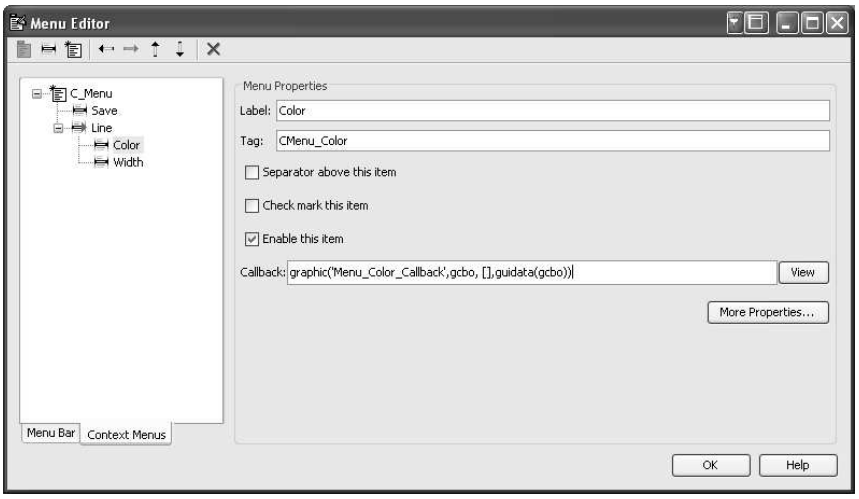

Рис. 20: Окно создания пунктов меню.

Замечание 6.2 *Часто возникает ситуация, когда при выборе пункта контекстного меню и пункта из строки меню должны выполняться одни и те же действия. В этом случае для обработки события Callback пункта меню и пункта контекстного меню должна выполняться одна и та же функция.*

Для этого необходимо в редакторе меню для пунктов и контекстного меню в свойстве Callback установить в каче-

стве значения одну и ту же функцию. Например, для пункта Save в контекстном меню в строке ввода Callback введем следующую строку:

```
graphic('Menu_Save_Callback',gcbo, [],guidata(gcbo));
```

Задание 6.1 *Самостоятельно добавьте и запрограммируйте пункт меню exit в меню file, отвечающий за окончание работы приложения. При программировании воспользуйтесь диалоговым окном подтверждения (функция questdlg) выхода из приложения и функцией delete.*

Задание 6.2 *Самостоятельно запрограммируйте пункт меню Width, отвечающий за задание толщины линии. При программировании воспользуйтесь диалоговым окном inputdlg, предназначенным для ввода информации пользователем.*

Замечание 6.3 *Полный текст нашего приложения приведен в последнем пункте пособия.*

7 Построение нескольких графиков в одном графическом окне.

Созданное нами приложение позволяет строить график функции, однако, на практике часто возникает необходимость

в построении нескольких графиков функций в одном графическом окне. Это можно сделать следующим способом. Разместить в одном графическом окне несколько объектов оси (при помощи инструмента Axes), задать каждому объекту значение свойства Tag и делать их текущими при помощи вызова команды axes, подавая в качестве входного параметра указатель на объект "оси". Кроме того, существует возможность выводить на каждых осях по несколько графиков. Для этого свойству осей NextPlot требуется задать значение add.

Иллюстрируя сказанное, добавим в наше приложение возможность построения нескольких графиков на одних осях, возможность построения графика на разных осях, а также возможность выбора осей, на которых будет строиться график. Кроме того, усовершенствуем наше приложение, предоставив пользователю возможность задавать цвет и толщину линии не только до построения графика, но и после того как график был построен.

Итак, добавим в наше приложение вторую пару осей, а также radio-кнопки выбора осей, на которых будет строиться график.

Изменим нашу заготовку окна согласно рис. 21.

Вначале изменим нашу заготовку окна: уменьшим размер осей, добавим объект axes и присвоим свойству tag этого объ-

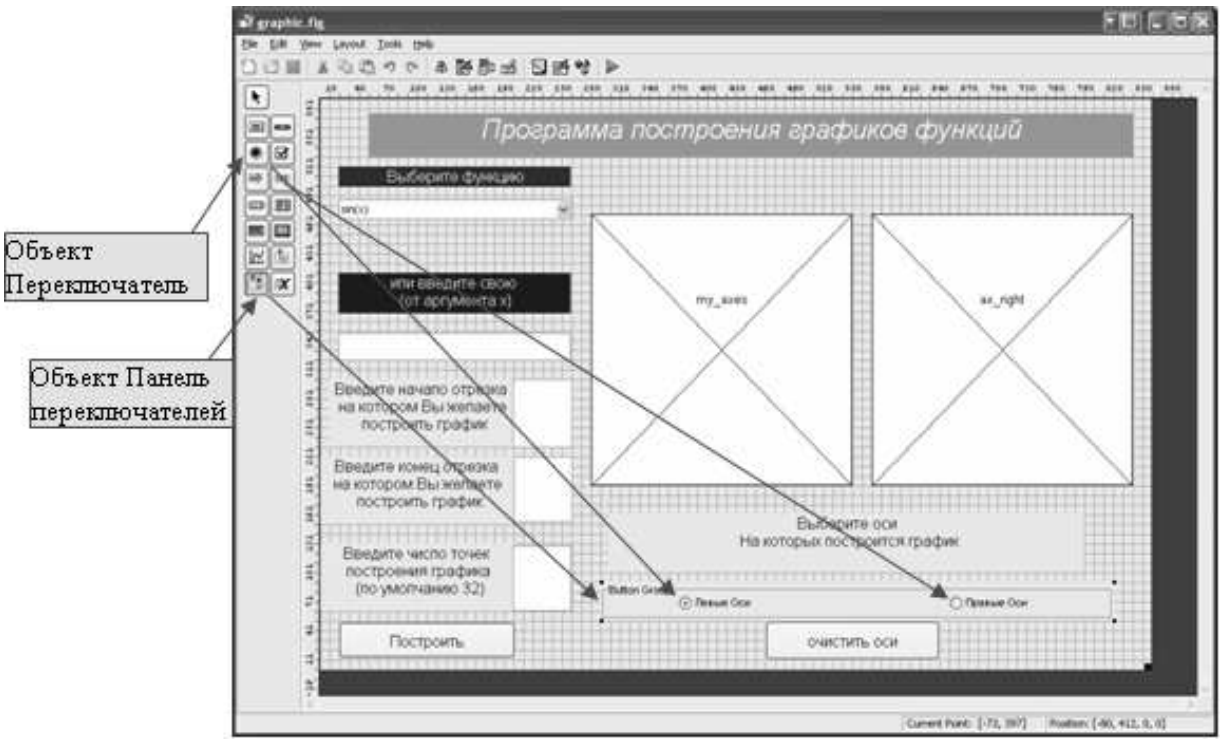

Рис. 21: Приложение myplot

екта значение ax_right. Зададим у объектов axes значение 'on' в свойстве 'Visible' и значение 'add' в свойстве NextPlot.

Добавим в приложение переключатели. Для того, чтобы переключатели вели себя согласованно в работающем приложении, необходимо их разместить на панели переключателей. Поэтому добавим в заготовку окна панель переключателей, как показано на рис. 21, и присвоим тегу значение pnl_axes. Обратите внимание на свойство SelectedObject, имеющее в качестве значения указатель на выбранный пользователем объект на панели переключателей. После этого расположим на добавленной панели два переключателя с тегами rb_left и

rb_right. Обратим Ваше внимание на свойство 'UserData', предназначенное для сохранения данных пользователя.

Опишем идею алгоритма. Сначала в функции открытия окна при помощи функции set сохраним в свойство 'UserData' правого и левого переключателя (rb_left и rb_right) указатель на правую и левую оси. При возникновении события Callback у объекта "кнопка Построить", в теле функции btnPlot_Callback, мы из свойства SelectedObject, получим указатель на выбранный объект:

```
(get(handles.pnl_axes,'SelectedObject'));
```

Затем, используя полученный указатель, у выбранного объекта получим значение свойства 'UserData', содержащего указатель на ось, и сделаем ось текущей при помощи команды axes. Таким образом, делать выбранные оси текущими можно при помощи команды:

```
axes(get(get(handles.pnl_axes,'SelectedObject'),'UserData'));
```

Для того, чтобы задавать цвет и толщину линии после того, как график был построен, необходимо сохранять указатели на построенные линии. С этой целью, при построении графика функции в структуру handles добавим новое поле, которое будет содержать указатели на графики, а также название функции:

```
if isfield(handles,'myplot')
```

```
  handles.myplot{length(handles.myplot)+1}=...
  plot(x,y,'LineWidth',2,'Color',My_Color);
  handles.myfun{length(handles.myfun)+1}=FcnName;
  else
  handles.myplot{1}=plot(x,y,'LineWidth',2,'Color',My_Color);
  handles.myfun{1}=FcnName;
  end;
```

После этого сохраним структуру handles.

Теперь в функциях выбора цвета и задания ширины линии мы можем при помощи указателей на линию задавать ширину и цвет построенной линии.

Приведем тексты основных функций:

```
function btnPlot_Callback(hObject, eventdata, handles)
% hObject handle to btnPlot (see GCBO)
% eventdata reserved - to be defined in a future
% version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global My_Color;
FcnName=get(handles.edt_fcn,'String');
if isempty(FcnName)
  FcnPop=get(handles.popupmenu_fcn,'String');
  PopId=get(handles.popupmenu_fcn,'Value');
  FcnName=FcnPop{PopId};
end;
```

```
  FcnBegin=str2num(get(handles.edt_begin,'String'));
if isempty(FcnBegin) FcnBegin=0; end;
FcnEnd=str2num(get(handles.edt_end,'String'));
if isempty(FcnEnd) FcnEnd=1; end;
FcnStep=str2num(get(handles.edt_step,'String'));
if isempty(FcnStep) FcnStep=32; end;
[x,y]=fplot(FcnName,[FcnBegin FcnEnd],FcnStep);
axes(get(get(handles.pnl_axes,'SelectedObject'),'UserData'));
if isfield(handles,'myplot')
  handles.myplot{length(handles.myplot)+1}=...
  plot(x,y,'LineWidth',2,'Color',My_Color);
  handles.myfun{length(handles.myfun)+1}=FcnName;
else
  handles.myplot{1}=plot(x,y,'LineWidth',2,'Color',My_Color);
  handles.myfun{1}=FcnName;
end;
set(handles.btn_clear,'Visible','on');
guidata(hObject, handles);

function btn_clear_Callback(hObject, eventdata, handles)
% hObject handle to btn_clear (see GCBO)
% eventdata reserved - to be defined in a future
% version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.edt_end,'String','');
```

```
axes(get(get(handles.pnl_axes,'SelectedObject'),'UserData'));
cla;


function Menu_Color_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Color (see GCBO)
% eventdata reserved - to be defined in a future
% version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global My_Color;
ListString={'NewGraphic'};
if isfield(handles,'myfun')
  ListString=[ListString,handles.myfun];
end;
[Selection,ok] = listdlg('ListString', ListString);
if Selection==1
  My_Color=uisetcolor;
else
  uisetcolor(handles.myplot{Selection-1});
end;
```

8 Функции для создания диалоговых окон

В MATLAB определены 15 функций для создания стандартных диалоговых окон. Вид диалогового окна может быть настроен путем задания входных аргументов этих функций. Ни-

же приведены функции, создающие диалоговые окна.

1. Функция errordlg предназначена для создания диалогового окна с сообщением об ошибках. Самый распространенный вариант вызова имеет следующий вид:

```
h = errordlg('Текст сообщения', 'Заголовок окна');
```

Параметр h является указателем на созданное графическое окно с сообщением об ошибке.

Если требуется вывести многострочное сообщение, разбитое на строки заданным образом, то в качестве входного параметра функции errordlg следует задать массив ячеек строк, например:

```
h = errordlg({'Wrong data format';'or'; 'unmatched data'}, ...
      'dlgname');
```

Многократный вызов функции errordlg приводит к созданию разных окон с сообщениями об ошибках. Например, последовательность следующих команд создает два окна об ошибках:

```
h = errordlg('Division by zero', 'Calculator');
h =errordlg('Incorrect operation', 'Calculator');
```

Для активизации окна (расположения его поверх остальных окон) и вывода в него нового сообщения предусмотрен вызов функции errordlg с третьим аргументом 'on', например, если создано диалоговое окно при помощи команды

```
h = errordlg('Division by zero', 'Calculator');
```

то после выполнения следующей команды:

```
h = errordlg(' Incorrect operation ', 'Calculator', 'on');
```

новое окно не создастся.

Окно с сообщением об ошибках не является модальным (оно позволяет переходить к другим окнам приложений MATLAB). Для того, чтобы сделать его модальным, следует после создания окна функцией errordlg и присвоения переменной h указателя на графическое окно, установить свойство WindowStyle графического окна в 'modal' при помощи следующей функции:

```
set(h,'WindowStyle', 'modal').
```

2. Функция warndlg предназначена для создания диалогового окна предупреждения.

Функция warndlg работает практически так же, как errordlg, но не предусмотрен вариант вызова с третьим входным аргументом 'on'. Например,

```
h = warndlg('File is out of date', 'Preprocessor');
```

приводит к созданию диалогового окна предупреждения с указателем h.

Окно с предупреждением отличается от окна с сообщением об ошибках только пиктограммой.

Повторный вызов warndlg с теми же самыми входными ар-

гументами приводит к созданию нового окна, даже если старое еще не было закрыто.

Многострочность текстового сообщения, модальность окна и приостановление выполнения приложения до закрытия окна выполняются так же, как и при вызове errordlg.

3. Функция helpdlg предназначена для создания диалогового окна вывода пояснительной информации. Эта функция работает в основном так же, как и warndlg, и отличается от окна с предупреждением лишь пиктограммой. Приведем пример обращения к данной функции:

```
h = helpdlg('File is out of date', 'Preprocessor');
```

4. Функция msgbox предназначена для создания диалогового окна вывода сообщения. Эта функция позволяет создавать окна с сообщением об ошибках с предупреждением и со справочной информацией, рассмотренные выше. Для получения многострочного текста следует в качестве первого аргумента задать массив ячеек из строк. Кроме того, возможно создание окна с произвольной пиктограммой. Управление модальностью окна и созданием нового с тем же заголовком производится при помощи входных аргументов msgbox. Функция msgbox возвращает указатель на создаваемое ею окно. Рассмотрим примеры использования функции msgbox.

Самый распространенный вариант вызова имеет следующий вид:

```
h = msgbox('Текст сообщения', 'Заголовок окна','Вид окна');
```

Параметр 'Вид окна' может иметь одно из следующих значений:

а) error – создается окно с сообщением об ошибках и стандартной пиктограммой (аналогично errordlg);

б) warn – создает окно с предупреждением и стандартной пиктограммой (аналогично warndlg);

в) help – создается окно со справочной информацией и стандартной пиктограммой (аналогично helpdlg);

```
h = msgbox({'Very serious error!'; 'Don''t tell anyone'},...
    'Program', 'error');
```

г) none – создает окно с текстовой информацией без пиктограммы;

д) custom – создает окно с пиктограммой пользователя.

Приведем пример создания окна с пиктограммой пользователя. Предварительно считываем рисунок и карту цветов из файла с помощью функции imread:

```
[pic, map] = imread('phone.gif');
h = msgbox('Someone is calling you', 'Phone', 'custom',...
    pic, map);
```

Если требуется вывести многострочное сообщение, разбитое на строки заданным образом, то в качестве первого входного параметра функции msgbox следует задать массив ячеек строк, например:

```
h = msgbox({'Very serious error!'; 'Don''t tell anyone'},...
    'Program', 'error');
```

Кроме того, имеется возможность управлять видом и способом вывода окна с текстовой информацией. Для этого следует указать в качестве последнего входного аргумента функции одно из следующих значений:

а) 'modal' – создается модальное окно.

б) 'non-modal' – создается не модальное окно.

в) 'replace' – заменяется предыдущее окно.

Приведем пример использования параметра 'replace':

```
h = msgbox('Very serious error!', 'Program', 'error',...
    'replace');
```

Если было создано окно заголовком Program, то текст в нем заменится, если такого окна нет, то создается новое окно.

5. Функция questdlg создает модальное диалоговое окно подтверждения. Входные параметры позволяют указать, какая из кнопок будет находиться в фокусе. Выходной аргумент функции questdlg возвращает либо строку с названием нажа-

той пользователем кнопки, либо пустую строку, если окно было закрыто без нажатия на какую-либо кнопку. Для отображения многострочного текста следует использовать массив ячеек из строк. Если задана длинная строка, то она переносится автоматически.

Рассмотрим несколько примеров.

Для создания стандартного окна подтверждения с кнопками <Yes>, <No>, <Cancel> следует использовать следующую команду:

```
button = questdlg('Are you sure?');
```

Отметим, что по умолчанию кнопка <Yes> находится в фокусе.

Если требуется создать стандартное окно подтверждения с заголовком и с кнопками <Yes>, <No>, <Cancel>, то используется команда с двумя параметрами:

```
button = questdlg('Are you sure?', 'Program');
```

Если в качестве последнего параметра указать имя кнопки, то данная кнопка будет находиться в фокусе. Например:

```
button = questdlg('Are you sure?', 'Program', 'No');
```

Кроме того, имеется возможность изменять надписи на кнопках. Следующая команда создает стандартное окно подтверждения с тремя кнопками <Yes>, <No> и <I don't

know>, при этом в фокусе находится кнопка <No>:

```
button = questdlg('Are you sure?', 'Program', ...
    'Yes', 'No', 'I don"t know', 'No');
```

6. Функция inputdlg создает диалоговое окно с заданным числом строк ввода. Число строк ввода и заголовки к ним определяются при помощи входных аргументов. Окно содержит кнопки <OK> и <Cancel>. Если пользователь нажал кнопку <OK>, то выходным аргументом функции inputdlg является массив ячеек с информацией, введенной пользователем в строки ввода. Если окно было закрыто нажатием на кнопку <Cancel> или любым другим способом (кроме нажатия на кнопку <OK>), то выходной аргумент – пустой массив ячеек.

Приведем пример использования функции inputdlg:

```
M = inputdlg({'TIME' 'VALUES'});
```

Для того, чтобы можно было вводить многострочный текст, следует указать третий входной аргумент – число строк в каждой области ввода, при этом если третий аргумент является числом, большем единицы, то все области ввода имеют одинаковую высоту и снабжены полосами прокрутки:

```
M = inputdlg('matrix A' 'Matrix B', 'Input Data', 3);
```

Если ввод завершен нажатием на кнопку <OK>, то каж-

дая ячейка массива M содержит массив символов. В массиве символов столько строк, сколько было введено в соответствующую область ввода.

Для задания высоты областей ввода по отдельности следует указывать число строк каждой области ввода в вектор-столбце в качестве третьего входного аргумента, например:

```
M = inputdlg('vector A' 'Matrix B','Input Data', [1; 4]);
```

Для получения окна с заполненными полями (для подсказки формата ввода пользователю) указывается четвертый входной аргумент. Он должен быть массивом ячеек того же размера, что и первый входной аргумент, например:

```
M = inputdlg('Name' 'Year', 'Info', [2; 1],...
    char('John','Smith') '1990');
```

Чтобы пользователь смог изменять размеры диалогового окна, необходимо задать пятый аргумент со значением 'on', например:

```
M = inputdlg('Name' 'Year', 'Info', [2; 1],...
char('John','Smith') '1990', 'on');
```

7. Функция listdlg создает диалоговое окно со списком из заданных строк и кнопками <OK> и <Cancel>. Строки списка задаются массивом ячеек. Поддерживается выбор нескольких элементов списка. В выходных аргументах возвращаются:

Selection – номер или вектор с номерами выбранных строк и ok – информация о том, был ли выбор завершен нажатием кнопки <OK> (ok = 1) или окно было закрыто другим способом (ok =0). Если ok = 0, то Selection – пустой массив. По умолчанию допускается выбор нескольких элементов списка (щелчком мыши с удержанием клавиши <Ctrl>) и присутствует кнопка <Select All> для выбора всех элементов.

В качестве примера рассмотрим диалоговое окно выбора со списком из трех строк: Linear, Quadratic, Cubic:

```
[Selection,ok] = listdlg('ListString', ...
    {'Linear'; 'Quadratic'; 'Cubic'});
```

Если пользователь выбрал, к примеру, Quadratic и нажал кнопку <OK>, то Selection = 2, ok = 1. Если пользователь выбрал Linear и Cubic и нажал кнопку <OK>, то аргумент Selection = [1 3], аргумент ok = 1.

В общем случае функция listdlg имеет следующий вид:

[Selection, ok] = listdlg('ListString', массив ячеек строк, параметр1, значение1, параметр2, значение2, ...)

Укажем названия параметров и их возможные значения:

– 'SelectionMode' – режим выбора строк в списке, значения: 'single' (для выбора только одной строки) или 'multiple' (по умолчанию – для выбора нескольких строк);

– 'ListSize' – размер области списка в пикселях, значение задается вектором [ширина высота] (по умолчанию – [160 300]);

–'InitialValue' – начальное выделение строк, значением является вектор с номерами строк, которые будут выделены при отображении окна со списком (по умолчанию – 1);

– 'Name' – заголовок окна, значение задается строкой (по умолчанию – пустая строка);

– 'PromptString' – текст над списком, значения: строка или массив строк или ячеек из строк (для многострочного текста). По умолчанию – пустой массив ячеек;

– 'OKString' – надпись на кнопке для выбора и закрытия окна, значением является строка (по умолчанию – строка 'OK');

– 'CancelString' – надпись на кнопке для закрытия окна без выбора, значением является строка (по умолчанию – 'Cancel');

– 'uh' – высота кнопок окна, значение задается в пикселях (по умолчанию – 18);

– 'fus' – расстояние между кнопками и списком, значение задается в пикселях (по умолчанию – 18);

– 'ffs' – расстояние от границы окна до списка, значение задается в пикселях (по умолчанию – 8).

8. Функция pagesetupdlg предназначена для создания диалогового окна установки параметров листа и способа печати

содержимого графического окна.

Функция pagesetupdlg вызывает диалоговое окно установки параметров страницы (Page Setup) для текущего графического окна (если нет ни одного окна, то создается новое графическое окно). Если существует несколько графических окон, то функция pagesetupdlg позволяет в качестве входного параметра использовать указатель на графическое окно. Например, следующая команда вызывает диалоговое окно Page Setup для графического окна с указателем hF :

```
pagesetupdlg(hF);
```

Настройки, сделанные в диалоговом окне Page Setup, влияют на соответствующие свойства графического окна: PaperSize, PaperType, PaperPozition и т. д.

Функция pagesetupdlg позволяет настроить параметры только одного графического окна, то есть hF не может быть вектором указателей. Для одновременной настройки основных параметров нескольких графических окон следует использовать функцию pagedlg, входным аргументом которой может быть вектор указателей на графические окна.

Окно Page Setup является модальным, выполнение приложения приостанавливается до тех пор, пока окно не будет закрыто.

9. Функция printdlg вызывает стандартное диалоговое окно Windows для печати текущего графического окна (если нет ни одного окна, то создается новое). Если существует несколько графических окон, то функция printdlg позволяет в качестве входного параметра использовать указатель на графическое окно. Например, следующая команда вызывает стандартное диалоговое окно Windows для печати графического окна с указателем hF:

```
printdlg(hF);
```

Функция printdlg позволяет вызывать диалоговое окно Print системы MATLAB, например:

```
printdlg('-crossplatform', hF);
```

Диалоговое окно печати является модальным, выполнение приложения приостанавливается до тех пор, пока окно не будет закрыто.

10. Функция uigetfile предназначена для создания диалогового окна открытия файла. Обращение к данной функции имеет следующий вид:

```
[FName, PName] = uigetfile;
```

Функция uigetfile выводит диалоговое окно открытия файла с содержимым текущего каталога. В окне установлен фильтр файлов в положение ALL MATLAB files; это приводит к отоб-

ражению только тех файлов, расширения которых поддерживаются MATLAB. В раскрывающемся списке Files of type можно выбрать только M-файлы или только графические окна, или все файлы.

Если пользователь выбрал файл (щелчком мыши по названию с файла и нажатием кнопки <Open>, либо двойным щелчком мыши по значку с файлом, либо набором имени файла в строке File Name и нажатием кнопки <Open>), то переменная FName будет содержать строку с именем файла и расширением, а переменная PName будет содержать строку – путь к файлу. Если пользователь не выбрал файл и закрыл окно, то переменные FName и PName будут равны нулю. После обращения к функции uigetfile следует проверить, был ли выбран файл. Если пользователь выбрал файл, то для получения полного имени файла следует сцепить эти строки:

```
[FName, PName] = uigetfile;
if ~ isequal(FName, 0)
FullName = strcat(PName, FName);
% дальше считываем данные из файла
end;
```

Кроме того, функция uigetfile предоставляет возможность задавать расширения отображаемых файлов. Для этого следует вызвать функцию uigetfile с входным аргументом. Например, следующая команда создает диалоговое окно открытия файлов, список Files of Type содержит две строки: "*.txt" и

"All Files (*.*)":

```
[FName, PName] = uigetfile('*.txt');
```

Можно указывать больше одного расширения, но тогда в качестве параметра следует использовать массив ячеек из строк:

```
[FName, PName] = uigetfile({'*.txt'; '*.dat'});
```

Для размещения описания типов файлов в списке Files of Type следует указывать массив ячеек из двух столбцов. В каждой его строке задается расширение файла и его описание (при этом строка "All Files (*.*)" не добавляется в список Files of Type диалогового окна открытия файла), например:

```
Filter={'*.txt', 'Text files (*.txt)'; ...
'*.dat', 'Data files (*.dat)'; '*.res', 'Results (*.res)'};
[FName, PName] =uigetfile(Filter);
```

В качестве фильтра может быть и строка с именем файла, тогда имя файла находится в строке File Name при появлении диалогового окна открытия файла, а его расширение принимается за расширение по умолчанию и появляется в списке Files of Type вместе с "All Files (*.*)".

Если необходимо узнать, какой по счету тип файлов пользователь выбрал в списке Files of Type, то следует обратиться к uigetfile с тремя выходными аргументами:

```
[FName, PName, FilterIndex] = uigetfile(Filter);
```

Переменной FilterIndex присваивается номер типа файлов.

Для задания собственного заголовка окна открытия файла (вместо Select File to Open) требуется вызвать функцию uigetfile со вторым входным аргументом, например:

```
[FName, PName] = uigetfile(Filter, 'Open File');
```

Кроме того, имеется возможность помещения заданного имени файла в строку File Name при создании окна открытия файлов, для этого необходимо указать в третьем входном аргументе имя файла (имя файла может быть полным):

```
[FName, PName] = uigetfile(Filter, 'Open File',
'input.txt');
```

Приведем пример отображения содержимого любой папки, например корневого каталога диска D:

```
[FName, PName] = uigetfile(Filter, 'Open File', 'd:\');
```

Замечание 8.1 *Если мы хотим открывать файлы MATLAB так, как предлагает uigetfile по умолчанию, но задать собственный заголовок окна (и папку или файл, предлагаемые по умолчанию в диалоговом окне открытия файлов), то первый входной аргумент – фильтр – пропускать нельзя. Он должен быть пустой строкой:*

```
[FName, PName] = uigetfile('', 'Open File', 'd:\');
```

Для того, чтобы пользователь мог выбрать несколько фай-

лов, следует вызвать uigetfile одним из следующих способов:

```
[FName, PName] = uigetfile('MultiSelect', 'on');
```

или

```
[FName, PName] = uigetfile('*.dat', 'MultiSelect', 'on');
```

или

```
[FName, PName, FilterIndex] = ...
uigetfile('*.dat', 'Open File','d:\', 'MultiSelect', 'on');
```

Если пользователь выбрал несколько файлов в диалоговом окне, то выходной аргумент FName является массивом ячеек из имен выбранных файлов (PName, разумеется, остается строкой, так как файлы выбирались в одной папке). В качестве примера приведем программу организующую получение массива ячеек с полными именами файлов (функция strcat умеет сцеплять строку с массивом ячеек из строк):

```
[FName, PName] = uigetfile('MultiSelect', 'on');
if ~ isequal(FName, 0)
FullName = strcat(PName, FName);

end;
```

11. Функция uiputfile предназначена для создания диалогового окна сохранения файла.

Использование функции uiputfile схоже с использованием функции uigetfile, рассмотренной выше. Точно так же задаются фильтр расширений, заголовок окна, файл или путь по

умолчанию. Отличие состоит в том, что в uiputfile нет опции 'MultiSelect' (она и не нужна при сохранении). Кроме того, если в диалоговом окне сохранения файла пользователь выберет имя существующего файла, то появится окно подтверждения. Нажатие на кнопку <Yes> приводит к завершению диалога сохранения файла, а нажатие на кнопку <No> – к возврату в окно сохранения файла. Проверка действий пользователя и получение полного имени файла выполняется точно так же, как и в случае функции uigetfile.

12. Функция uigetdir предназначена для создания диалогового окна выбора каталога. Функция uigetdir открывает диалоговое окно выбора каталога Browse for folder и возвращает путь к выбранному каталогу, либо нуль, если пользователь не сделал выбора. Проверка, был ли выбран каталог, делается точно так же, как и в случае функции uigetfile. Укажем некоторые способы вызова uigetdir.

В первом варианте использования функции uigetdir происходит открытие диалогового окна выбора каталога, в котором выделен текущий каталог MATLAB. Переменной dname присваивается строка содержащая выбранный каталог:

```
dname = uigetdir;
```

Кроме того, возможен вызов функции uigetdir с входным аргументом. В качестве входного аргумента возможно исполь-

зовать путь к определенной папке. Следующий пример иллюстрирует диалоговое окно выбора каталога, в котором выделен заданный каталог, например, "Диск С":

```
dname = uigetdir('c:\');
```

Кроме, того имеется возможность открыть диалоговое окно выбора каталога, в котором выделен заданный каталог и поместить свой текст над окном с иерархической структурой каталогов:

```
dname = uigetdir('c:', 'text');
```

Если нужно открыть диалоговое окно выбора каталога, в котором выделен текущий каталог MATLAB, то вместо первого входного аргумента следует указать пустую строку.

13. Функция uisetfont предназначена для создания диалогового окна выбора шрифта.

Функция uisetfont может использоваться в двух целях:

1) для создания диалогового окна выбора шрифта и записи информации о выбранном шрифте (название, размер, начертание) в структуру;

2) для создания диалогового окна выбора шрифта и изменения созданных ранее текстовых объектов, подписей осей или элементов управления в соответствии со сделанными в этом окне установками.

Функция uisetfont приводит к появлению диалогового окна выбора шрифта и, если выбор в нем был сделан, то возвращается структура, поля которой имеют следующие назначения:

– поле FontName содержит строку с названием шрифта;

– поле FontUnits содержит единицы измерения ('points');

– поле FontSize содержит размер шрифта;

– поле FontWeight содержит жирность шрифта и может принимать два значения: 'normal'(обычный) или 'bold'(жирный);

– поле FontAngle содержит начертание шрифта и может принимать следующие значения: 'normal' (обычный) или 'italic' (курсив).

Если пользователь не выбрал шрифт и закрыл окно, то возвращаемое функцией значение равно нулю.

Функция uisetfont допускает вызов с входными значениями. В качестве входного значения допускается передавать структуру, поля которой задают характеристика шрифта. Следующая программа приводит к появлению диалогового окна выбора шрифта, в котором по умолчанию выбран шрифт в соответствии со значениями полей структуры S0:

```
F0.FontName = 'Arial';
F0.FontUnits = 'points';
F0.FontSize = 20;
F0.FontWeight = 'bold';
```

```
F0.FontAngle = 'normal';

F = uisetfont(F0);
```

Кроме того, функция uisetfont предоставляет возможность вывода диалогового окна выбора шрифта с заданным заголовком. Для этого в качестве входного аргумента в функцию требуется передать строку, например:

```
S = uisetfont('Заголовок окна');
```

или

```
S = uisetfont(S0, 'Заголовок окна');
```

Для установки шрифта у ранее созданных объектов требуется в качестве входного аргумента передать указатель на объект:

```
uisetfont(h);
```

или

```
uisetfont(h, 'Заголовок окна');
```

или

```
S = uisetfont(h);
```

или

```
S = uisetfont(h, 'Заголовок окна');
```

В рассмотренных примерах переменная h является указателем на текстовый объект, оси или элемента управления. Отметим, что в последнем примере информация о выборе поль-

зователя дополнительно возвращается в структуре S.

Приведем пример изменения шрифта графических у созданных объектов.

```
hA = axes;
plot(rand(10));
hT = title('Random Graph');
uisetfont(hA, 'Set font for axes');
uisetfont(hT, 'Set font for title');
```

14. Функция uisetcolor предназначена для создания диалогового окна выбора цвета.

Функция uisetcolor так же, как и функция uisetfont, может использоваться в двух целях:

1) для создания диалогового окна выбора цвета и записи выбранного цвета в вектор в формате RGB;

2) для создания диалогового окна выбора цвета и изменения графических объектов, у которых есть свойство Color, в соответствии со сделанными в этом окне установками.

Использование функции uisetcolor аналогично использованию функции uisetfont.

15. Функция waitbar создает окно с полосой прогресса и позволяет обновлять ее. Обращение к функции имеет следующий вид:

```
h =waitbar(x, 'текст');
```

Данная команда отображает окно с полосой прогресса и заданным текстом над полосой. Длина полосы прогресса пропорциональна значению x, причем $x \in [0, 1]$. Выходной аргумент h – указатель на графическое окно с полосой прогресса.

Следующая команда увеличивает длину полосы прогресса с указателем h в соответствии со значением x:

```
waitbar(x, h);
```

Кроме перечисленных функций имеется функция dialog, предназначенная для создания пустого графического окна.

9 Листинг программы

```
function varargout = graphic(varargin)

% GRAPHIC M-file for graphic.fig

% GRAPHIC, by itself, creates a new GRAPHIC or raises the
% existing singleton*.

% H = GRAPHIC returns the handle to a new
% GRAPHIC or the handle to the existing singleton*.

% GRAPHIC('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in GRAPHIC.M with the given input arguments.

% GRAPHIC('Property','Value',...) creates a new GRAPHIC or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before graphic_OpeningFcn gets called. An unrecog-
% nized property name or invalid value makes property application stop.
% All inputs are passed to graphic_OpeningFcn via varargin.
```

```
   % *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
   % only one instance to run (singleton)".

   % See also: GUIDE, GUIDATA, GUIHANDLES

   % Edit the above text to modify the response to help graphic

   % Begin initialization code - DO NOT EDIT

   % Количество копий приложения
   gui_Singleton = 1;
   % Формируем структуру содержащую информацию о приложении

gui\_State = struct('gui\_Name',       mfilename, ...
                    'gui\_Singleton',  gui\_Singleton, ...
                    'gui\_OpeningFcn', @graphic\_OpeningFcn, ...
                    'gui\_OutputFcn',  @graphic\_OutputFcn, ...
                    'gui\_LayoutFcn',  [] , ...
                    'gui\_Callback',   []);

   % Проверяем была ли функция graphic вызвана с входными аргументами и
   % является ли первый аргумент строкой
   if nargin && ischar(varargin{1})
     % В структуру gui_State присваиваем полю gui_Callback указатель на
     % функцию вызвавшую приложение graphic
     gui_State.gui_Callback = str2func(varargin{1});
   end
   % Проверяем была ли функция graphic вызвана с выходными аргументами
   % и создаем окно приложения при помощи функции gui_mainfcn
   if nargout
     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
   else
     gui_mainfcn(gui_State, varargin{:});
   end
```

```
% End initialization code - DO NOT EDIT
% -- Executes just before graphic is made visible.
function graphic_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to graphic (see VARARGIN)
% Choose default command line output for graphic
handles.output = hObject;
% Определяем глобальную переменную отвечающую за цвет линий
global My_Color Pos
My_Color=[0 0 0];
% Update handles structure
% Сохраняем в свойство 'UserData' правого и левого переключателя
% (rb_left и rb_right) указатель на правую и левую оси.
set(handles.rb_left,'UserData',handles.axes_plot)
set(handles.rb_right,'UserData',handles.ax_right)
% Добавляем в структуру handles новое поле, отвечающую за ширину линии
handles.ValWidth=2;
% Update handles structure
% Сохраняем структуру handles
guidata(hObject, handles);
% UIWAIT makes graphic wait for user response (see UIRESUME)
% uiwait(handles.figure_graphic);

% -- Outputs from this function are returned to the command line.
function varargout = graphic_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
```

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;

% -- Executes on selection change in popupmenu_fcn.
function popupmenu_fcn_Callback(hObject, eventdata, handles)
% hObject handle to popupmenu_fcn (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns popupmenu_fcn
% contents as cell array
% contents{get(hObject,'Value')} returns
% selected item from popupmenu_fcn

% -- Executes during object creation, after setting all properties.
function popupmenu_fcn_CreateFcn(hObject, eventdata, handles)
% hObject handle to popupmenu_fcn (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edt_fcn_Callback(hObject, eventdata, handles)
% hObject handle to edt_fcn (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edt_fcn as text
% str2double(get(hObject,'String')) returns
% contents of edt_fcn as a double

% -- Executes during object creation, after setting all properties.
function edt_fcn_CreateFcn(hObject, eventdata, handles)
% hObject handle to edt_fcn (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edt_begin_Callback(hObject, eventdata, handles)
% hObject handle to edt_begin (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edt_begin as text
% str2double(get(hObject,'String')) returns
% contents of edt_begin as a double
% -- Executes during object creation, after setting all properties.
function edt_begin_CreateFcn(hObject, eventdata, handles)
% hObject handle to edt_begin (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
```

```
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edt_end_Callback(hObject, eventdata, handles)
% hObject handle to edt_end (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edt_end as text
% str2double(get(hObject,'String')) returns contents of
% edt_end as a double
% -- Executes during object creation, after setting all properties.
function edt_end_CreateFcn(hObject, eventdata, handles)
% hObject handle to edt_end (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

function edt_step_Callback(hObject, eventdata, handles)
% hObject handle to edt_step (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edt_step as text
```

```
% str2double(get(hObject,'String')) returns contents of
% edt_step as a double

% -- Executes during object creation, after setting all properties.
function edt_step_CreateFcn(hObject, eventdata, handles)
% hObject handle to edt_step (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
  get(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
end

% -- Executes on button press in btnPlot.
function btnPlot_Callback(hObject, eventdata, handles)
% hObject handle to btnPlot (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Указываем, что в этой функции используется
% глобальная переменная My_Color
global My_Color;
% Получаем данные введенные пользователем в объекты "поле для ввода"
FcnName=get(handles.edt_fcn,'String');
% Проверка ввел ли пользователь в данные "поле для ввода"
if isempty(FcnName)
% Если значение переменной FcnName пустое, тогда получаем
% название функции из объекта "выпадающий список"
FcnPop=get(handles.popupmenu_fcn,'String');
```

```
PopId=get(handles.popupmenu_fcn,'Value');
FcnName=FcnPop{PopId};
end;
% Считываем начало отрезка, если пользователь не ввел данные
% в объект "поле для ввода", то устанавливаем значение
% равное нулю
FcnBegin=str2num(get(handles.edt_begin,'String'));
if isempty(FcnBegin)
FcnBegin=0;
end;
% Считываем конец отрезка, если пользователь не ввел данные
% в объект "поле для ввода", то устанавливаем значение
% равным единицы
FcnEnd=str2num(get(handles.edt_end,'String'));
if isempty(FcnEnd)
FcnEnd=1;
end;
% Аналогичные действия проделываем для переменной FcnStep
FcnStep=str2num(get(handles.edt_step,'String'));
if isempty(FcnStep)
FcnStep=10;
end;
% При помощи стандартной функции fplot вычислим значения
% введенной пользователем функции, и значения ее аргумента
[x,y]=fplot(FcnName,[FcnBegin FcnEnd],FcnStep);
axes(get(get(handles.pnl_axes,'SelectedObject'),'UserData'));
if isfield(handles,'myplot')
handles.myplot{length(handles.myplot)+1}=...
plot(x,y,'LineWidth',handles.ValWidth,'Color',My_Color);
handles.myfun{length(handles.myfun)+1}=FcnName;
```

```
else
handles.myplot{1}=...
plot(x,y,'LineWidth',handles.ValWidth,'Color',My_Color);
handles.myfun{1}=FcnName;
end;
set(handles.btn_clear,'Visible','on');
guidata(hObject, handles);

% -- Executes on button press in btn_clear.
function btn_clear_Callback(hObject, eventdata, handles)
% hObject handle to btn_clear (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
axes(get(get(handles.pnl_axes,'SelectedObject'),'UserData'));
set(handles.edt_fcn,'String','');
set(handles.edt_begin,'String','');
set(handles.edt_end,'String','');
set(handles.edt_step,'String','');
cla;
  % --------------------------------
function Menu_File_Callback(hObject, eventdata, handles)
% hObject handle to Menu_File (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --------------------------------
function Menu_Format_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Format (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% --------------------------------
function CSave_Callback(hObject, eventdata, handles)
% hObject handle to CSave (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --------------------------------
function CLine_Callback(hObject, eventdata, handles)
% hObject handle to CLine (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --------------------------------
function Menu_Line_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Line (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --------------------------------
function Menu_Save_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Save (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global Pos
Filter={'*.bmp', 'BitMap files (*.bmp)';'*.eps', 'Eps files'};
[FName, DirName, FilterIndex] = uiputfile(Filter);
if ĩsequal(FName, 0) FullName = strcat(DirName, FName);
NewPos=get(handles.figure_graphic,'Position');
set(handles.figure_graphic,'Position',Pos);
```

```
saveas(handles.figure_graphic, FullName, Filter{FilterIndex}(3:end));
set(handles.figure_graphic,'Position',NewPos);
end;
 % --------------------------------
function Menu_Exit_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Exit (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
button = questdlg('Are you sure?', 'Program', 'No');
if strcmp(button,'Yes')
delete(handles.figure_graphic);
end;

% --------------------------------
function Menu_Color_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Color (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global My_Color;
ListString={'NewGraphic'};
if isfield(handles,'myfun')
ListString=[ListString,handles.myfun];
end
[Selection,ok] = listdlg('ListString', ListString);
if length(Selection)==1
if Selection==1
My_Color=uisetcolor;
else
uisetcolor(handles.myplot{Selection-1});
end;
```

```
else
for k=1:length(Selection)
if Selection(k)==1
My_Color=uisetcolor;
else
uisetcolor(handles.myplot{Selection(k)-1});
end;
end
end

% --------------------------------
function Menu_Width_Callback(hObject, eventdata, handles)
% hObject handle to Menu_Width (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
ListString={'NewGraphic'};
if isfield(handles,'myfun')
ListString=[ListString,handles.myfun];
end
dlg_title = 'Input Line Width ';
CValWidth=inputdlg( ListString,dlg_title);
for k=1:length(CValWidth)
ValWidth=str2num(CValWidth{k});
if ValWidth>0 & k>1
set(handles.myplot{k-1},'LineWidth',ValWidth);
elseif ValWidth>0 & k==1
handles.ValWidth= ValWidth;
guidata(hObject, handles);
end
end
```

```
   % ---------------------------------
   function CColor_Callback(hObject, eventdata, handles)
   % hObject handle to CColor (see GCBO)
   % eventdata reserved - to be defined in a future version of MATLAB
   % handles structure with handles and user data (see GUIDATA)
   % ---------------------------------
   function CWidth_Callback(hObject, eventdata, handles)
   % hObject handle to CWidth (see GCBO)
   % eventdata reserved - to be defined in a future version of MATLAB
   % handles structure with handles and user data (see GUIDATA)
   % --------------------------------- function C_Menu_Callback(hObject,
eventdata, handles)
   % hObject handle to C_Menu (see GCBO)
   % eventdata reserved - to be defined in a future version of MATLAB

   % handles structure with handles and user data (see GUIDATA)
```

Литература

[1] *И. Ануфриев, А. Смирнов, Е. Смирнова.* MATLAB 7.0. В подлиннике. СПб.: БХВ-Петербург, 2005.

[2] *В. П. Дьяконов* Matlab 6.5 SP1/7 + Simulink 5/6 в математике и моделировании . М.: Солон-Пресс, 2005.-576с

[3] *В. Кондрашов, С. Королев.* Matlab как система программирования научно-технических расчетов. М.: Мир, Институт стратегической стабильности Минатома РФ, 2002.

[4] *В. Потемкин. .* Среда проектирования инженерных приложений. М.: Диалог-МИФИ, 2003. 5. В.Потемкин. Система MATLAB. Справочное пособие. М: Диалог-МИФИ, 1997.