Для управления работой приложения в MATLAB предусмотрены такие элементы управления, как кнопки, переключатели, флажки, списки и т.п., которые являются графическими объектами и располагаются в графических окнах. Они могут быть созданы с помощью специальных функций. Наряду с этим в MATLAB предусмотрена визуальная среда GUIDE[1] с набором заготовок, позволяющая создавать на их основе приложения с графическим интерфейсом, включающим элементы управления, и программировать события.

Создание интерфейса в среде GUIDE включает в себя расположение элементов управления в пределах графического окна и программирование реакций на события, генерируемые при работе с элементами управления. При создании интерфейса пользователя рассмотрим три задачи:

1. Конструирование интерфейса (программирование событий).
2. Создание диалоговых окон.
3. Создание меню приложений.

Варианты вызова визуальной среды

1. Для выбора инструмента используется команда *File → New → GUI.*
2. Ввести в командной строке **`guide`**

Настройка визуальной среды осуществляется в появившемся окне *GUIDE Quick Start (рис.1)*.

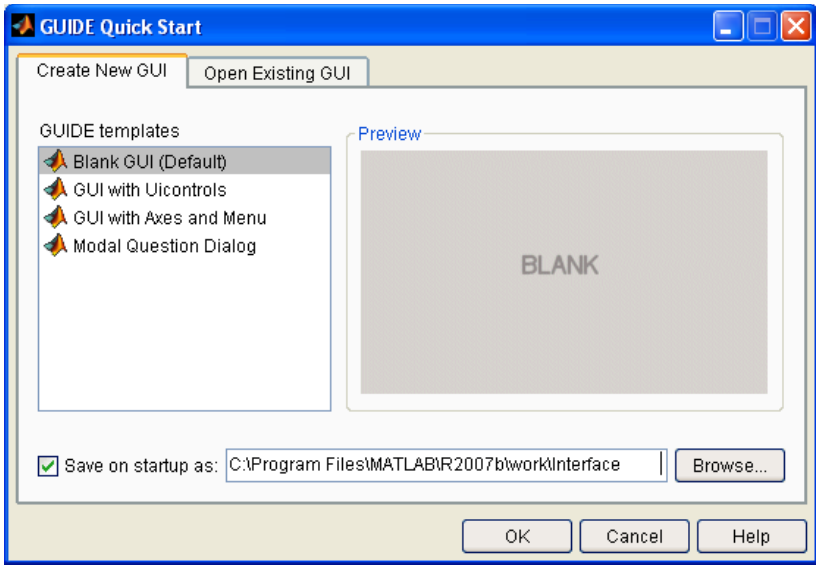


Рис. 1.

На вкладке *Open Existing GUI* выбирается существующее приложение. Для создания нового приложения надо выбрать вкладку *Create New GUI*. Возможны четыре варианта:

1. *Blank GUI (Default)* – пустое окно приложения без элементов управления и графических объектов.
2. *GUI with Uicontrols* – окно приложения со строками ввода, переключателями и кнопками.
3. *GUI with Axes and Menu* – окно приложения, содержащее оси, раскрывающийся список, кнопки и меню.
4. *Modal Question Dialog* – модальное диалоговое окно с кнопками *Yes* и *No*.

Имя файла, в котором будет сохранено окно приложения, можно задать в поле строки *Save on startup as* при установленном в этой же строке флажке.

[1] GUIDE (Graphic User Interface Development Environment – среда разработки GUI).

Информация о разработанном интерфейсе по умолчанию сохраняется в двух файлах с одним и тем же именем, но с разными расширениями. Один – с расширением `.fig`, в котором содержится описание графического окна, а второй – с расширением `.m`, в котором запрограммированы реакции на события. Для запуска приложения необходимо вызвать `m`-файл из командной строки или из другого приложения.

После выбора *Blank GUI (Default)* и нажатия кнопки *OK* будет вызван редактор среды GUIDE (рис. 2). Этот редактор содержит строку меню, панель инструментов управления приложением, заготовку окна приложения, панель инструментов для добавления элементов интерфейса на окно приложения. В нижней части редактора расположены два поля: *Current Point* и *Position*. В первом отображаются координаты курсора на заготовке окна приложения, во втором – координаты элемента интерфейса и поперечные размеры окна заготовки приложения. Изменить вид редактора можно в диалоговом окне *Preferences*, которое вызывается командой *Preferences* меню *File*.

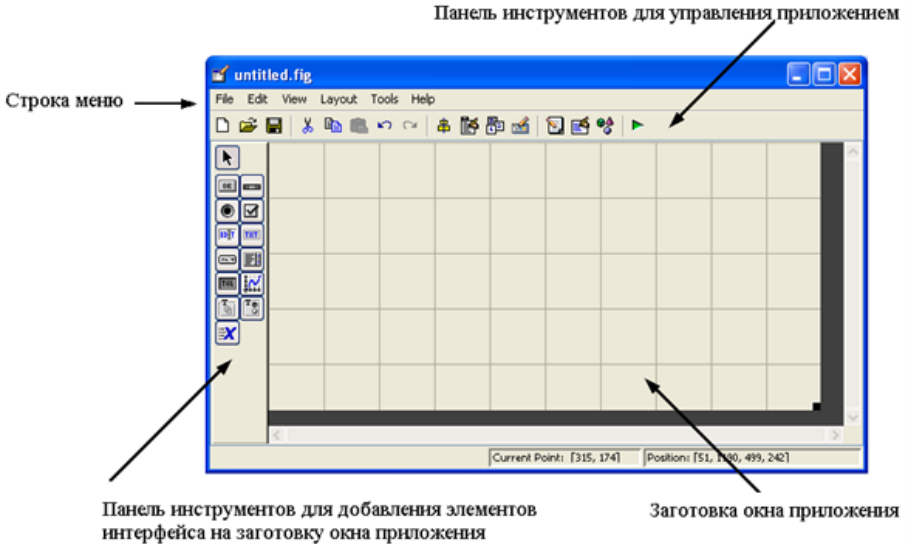


Рис. 2.

При работе с GUIDE компоненты GUI выбираются из палитры компонентов инструмента GUIDE мышью и переносятся в рабочую область GUI.

С помощью панели инструментов для добавления элементов интерфейса на заготовке окна приложения располагаются элементы интерфейса: кнопка, переключатель, окно ввода текста, раскрывающийся список, кнопка-переключатель, полоса прокрутки, флаг, текстовая область, список, оси, панель для группы переключателей, обычная панель, ActiveX-компоненты (рис. 3). Добавление элемента интерфейса на заготовку окна приложения приводит к автоматическому созданию соответствующей функции в файле с расширением .m, которую пользователю необходимо наполнить операторами, обрабатывающими события, возникающие при обращении к этому элементу интерфейса.

Для создания **GUI** в **MATLAB** используются следующие компоненты:

К о м п о н е н т	Н а з н а ч е н и е
Push Button	Кнопка. При нажатии отображается нажатой, при отпускании генерирует действие и отображается отжатой.
Toggle Button	Переключаемая кнопка. При нажатии отображается нажатой, при отпускании генерирует действие, но остается нажатой.
Check Box	Независимый переключатель. Генерирует действие, когда нажат, и отображает включенное состояние птичкой в прямоугольнике.
Radio Button	Зависимый переключатель (Селективная кнопка). Генерирует действие,

	когда нажат, отображает включенное состояние точкой в кружке. Похож на Check Box, но при использовании в группе при включении выключает остальные.
Edit Text	Редактор текста. Действие генерируется при нажатии клавиши Enter.
Static Text	Статический текст (станет прописью в интерфейсе).
Slider	Ползунок. Отображает численные значения в выбранном диапазоне с выбор значения перемещением ползунка.
ListBox	Список тем для выбора. Позволяет выбрать одну или несколько тем.
Pop-Up Menu	Выпадающее меню (иначе ComboBox). Подобен ListBox, но список открывается только при выборе.
Axes	Координатные оси. Используются для рисования графиков.
Panel	Панель. Это контейнер для группы компонент GUI. Панель используется для облегчения понимания назначения элементов управления путем их визуального объединения.
Button Group	Группа кнопок. Подобна панели, но используется только для селективных кнопок и переключаемых кнопок.
ActiveX Control	Это элементы управления, разработанные Microsoft для операционной системы *Windows*. При переносе этого компонента в рабочую область вызывается список ActiveX элементов для выбора.

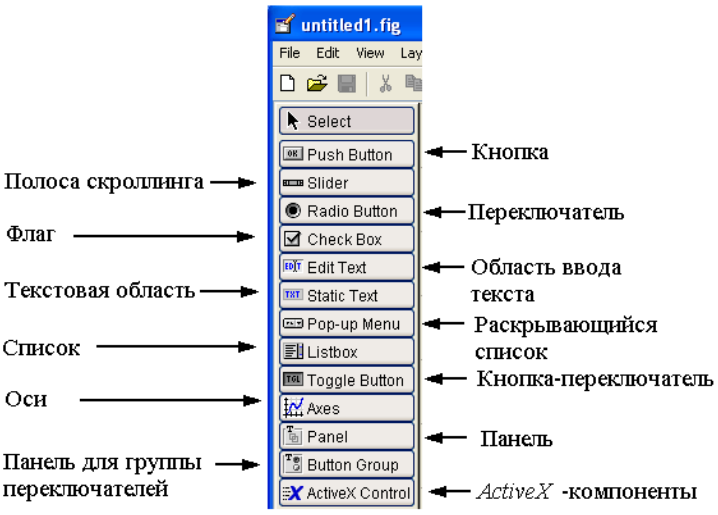


Рис. 3.

Для определения всех свойств объекта на стадии проектирования используется *Property Inspector* (рис. 4), который вызывается командой *Property Inspector* из меню, появляющемся после нажатия правой кнопки мыши на редактируемом объекте, или командой *View → Property Inspector* из среды GUIDE. Здесь задаются свойства элементов управления: поперечные размеры, координаты расположения в окне пользователя, первоначальные значения и т.д. Имена элементов управления, которые станут прописями в приложении пользователя, задаются в свойстве *String*, а соответствующие им названия функций, обрабатывающих события в файле с расширением .m, – в свойстве *Tag*. Первоначальный доступ ('on' или 'off') задается в свойстве *Enable*, числовые значения – в *Value*, присутствие (видимое – 'on', невидимое – 'off') в интерфейсе при запуске – в *Visible*. Назначение основных интерфейсных компонент, содержащихся в *Property Inspector*, описано в табл. 4.3.

Файл, в котором пользователь программирует события как реакцию на изменения состояния элементов управления, создается автоматически и вызывается командой *View → Component Callbacks →Callbacks*

из меню среды GUIDE или командой *View Callbacks →Callbacks* из меню, появляющегося после нажатия правой кнопкой мыши на необходимый графический элемент.

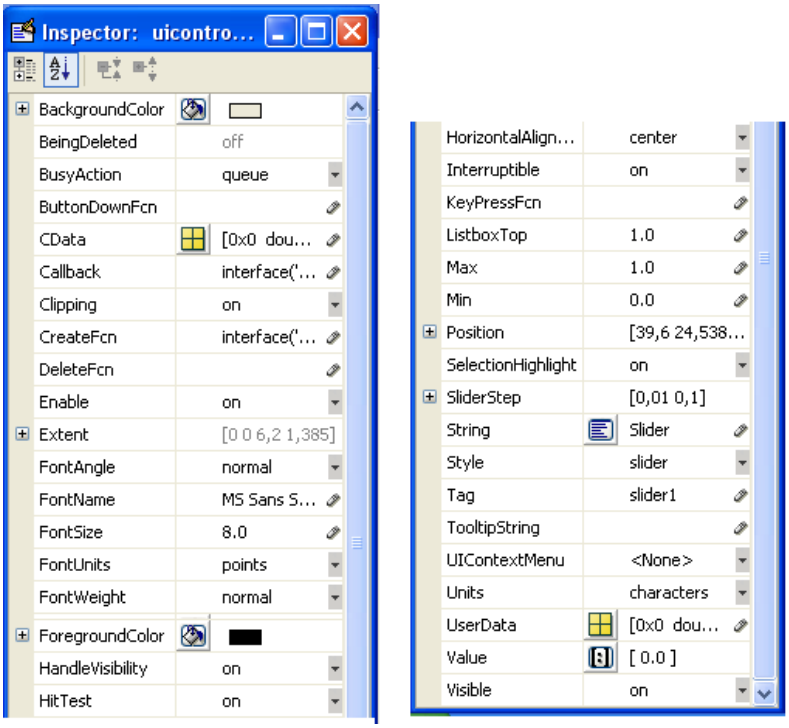


Рис. 4.

Табл. 4.3.

Свойство	Назначение	Допустимые значения
BackgroundColor	Цвет фона	Вектор, задающий RGB компоненты
Callback	Указатель на функцию, обрабатывающую событие *Callback*	
Enable	Признак доступа к объекту	*on* – разрешен *off* – запрещен
FontAngle	Признак наклона букв	*normal*, *italic*, *oblique*
FontName	Имя шрифта	
FontSize	Размер шрифта	
FontUnits	Единицы измерения	
FontWeight		
ForegroundColor	Цвет рисования	
HorizontalAlignment	Способ размещения надписи в поле объекта	*left*, *right*, *center*
Max	Максимальное значение свойства *Value*	
Min	Минимальное значение свойства *Value*	
Position	Вектор, определяющий положение и размеры объекта	[*x y width height*]
SliderStep	Вектор, определяющий перемещение ползунка	

String	Символьная строка, задающая надпись или значение, приписанное объекту	
Tag	Символьная строка, в которой хранится имя объекта	
TooltipString	Текст всплывающей подсказки	
UIContextMenu	Всплывающее меню, привязанное к данному объекту	
Value	Значение, характерное для данного объекта	
Visible	Признак видимости объекта в интерфейсе	*on* – видимый *off* – невидимый

Полный список команд и функций для проектирования пользовательского интерфейса можно получить по команде **help uitools**.

Для обработки (программирования) события используется подфункция обработки событий **Callback**.

Функции, обрабатывающие события элементов управления, имеют три входных аргумента – **hObject**, **eventData** и **handles**. Синтаксис функции `Name_Callback`, обрабатывающей событие:

```
function Name_Callback(hObject, eventData, handles).
```

В аргументе `hObject` хранится указатель на объект, событие которого обрабатывается. `Handles` является структурой с указателем на все объекты данного интерфейса. Имена полей этой структуры совпадают со значениями свойств `Tag` элементов интерфейса и, соответственно, `handles.Name` будет являться указателем на элемент управления (функцию) `Name`. Аргумент `eventData` зарезервирован для будущих версий MATLAB.

Обмен данными между подфункциями можно осуществить при помощи структуры handles.

Пример: *в некоторой подфункции сохранить массив в поле* `dat` *структуры* `handles`:

```
handles.dat=[ 1 2 3 4 5]
guidata(gcbo, handles)
```

а затем использовать его в другой подфункции

```
max(handles.dat)
```

Для изменения свойств элемента управления с именем `Name`, содержащихся в *Property Inspector*, используется функция **set**, а для получения – функция **get**:

```
set(handles.Name, 'Property', Value),
Value = get(handles.Name, 'Property').
```

В первом случае свойство `Property` элемента `Name` примет значение `Value`, а во втором случае `Value` будет возвращено значение свойства `Property` элемента `Name`.

Для изменения свойств элемента управления с именем Name внутри функции Name можно использовать команду

`set(hObject, 'Property', Value),`

а для получения –

`Value = get(hObject, 'Property').`

Нажатие на кнопки *Push Button* и *Toggle Button* (для этой кнопки – независимо от текущего положения) приводит к вызову соответствующих им функций **`Callback`** и выполнению написанных в них команд. Свойство `Value` кнопки *Toggle Button* в "утопленном" положении принимает значение 1, а в "не утопленном" – 0.

Свойство `Value` переключателя *Radio Button* и флага *Check Box* может принимать два значения:

0 – «флага» нет,

1 – «флаг» есть.

Изменение этого свойства происходит после щелчка мышью по этим элементам управления, или с помощью функции **`set`** в любой из функций **`Callback`** интерфейса пользователя.

Нажатие на полосу прокрутки приводит к вызову функции, обрабатывающей это событие. При изменении положения ползунка свойство `Value` принимает значение из диапазона [0,1], соответствующее значению относительной координаты ползунка в окне перемещения. Размеры последнего задаются в строках `width` и `height` свойства *Position* (рис. 4.17). Шаг смещения ползунка в долях от значения `width` при нажатии на кнопку "прокрутки" задается в строке X свойства *Slider Step*, а при нажатии на область справа или слева от ползунка – в строке Y. Этим же значением определяется и ширина ползунка в долях от значения `width`.

4.17.

Поле ввода текста *Edit Text* предназначено для ввода текстовой информации. Текст, который выводится в окно "по умолчанию", прописывается в свойстве *String*. Допускается как прописать в этом свойстве, так и ввести в окно ввода только одну строку. Введенный в окно ввода текст после нажатия *Enter* становится значением свойства *String*. Расположение текста в окне ввода задается в свойстве *HorizontalAlignment*. Значение свойства *String* возвращает функция **`get`**. ***Например***, для функции обработки текстовой области с заголовком

`function InputText_Callback(Txt, eventData, handles)`

после выполнения команды

`StringText = get(Txt, 'String')`

или

```
StringText = get(handles.InputText, 'String')
```

в `StringText` будет возвращено значение свойства *String*. Второй способ определения свойства *String* можно использовать и вне функции `InputText`.

Для работы со списками предназначены элементы управления **`Listbox`** (открытый список) и **`Popup menu`** (раскрывающийся список). Свойство *String* этих элементов определяется как массив ячеек. При работе с приложением значения свойства *String* прописываются в окнах для вывода в элементах `Listbox` и `Popup menu`. После обработки команды нажатия на какую-нибудь строку значение `Value` этих элементов принимает значение номера выбранной в списке строки. Для обработки выбранного пользователем элемента списка удобнее использовать функцию **`switch`**.

Свойство *String* можно изменять в приложениях пользователя. Т.е., в список можно добавить строки или удалить ненужные в процессе работы приложения, а также присваивать им новые значения, используя функцию **`num2cell`** или символы **`{}`**.

Пример:

```
StringText = get(handles.Listbox, 'String')
StringText(1) = num2cell('New');
set(handles.Listbox, 'String',StringText).
```

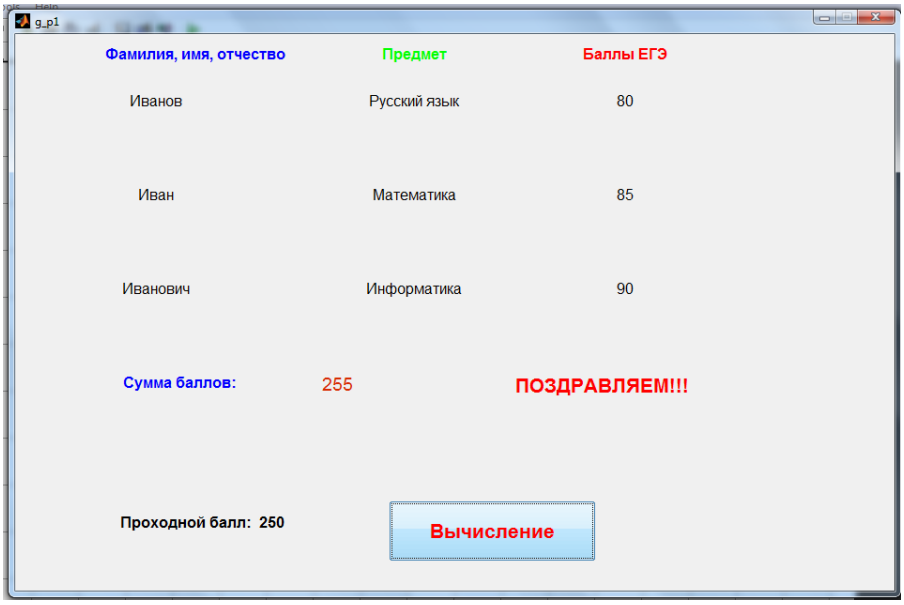
В этом примере сначала `StringText` возвращается значение свойства Stri*n*g элемента управления `Listbox`, затем первый элемент `StringText` изменяется на `New` и старое значение свойства *String* в этом элементе управления заменяется на новое. После выполнения этих команд в интерфейсе пользователя в первой строке списка `Listbox` старая пропись будет заменена на `New`.

Для полного изменения старого списка `Listbox` на новый с *n* строками можно использовать следующую последовательность команд

```
set(handles.Listbox, 'Value', 1)
StringText = num2cell('New''Name_1'; 'Name_2';… 'Name_n');
set(handles.Listbox, 'String',StringText).
```

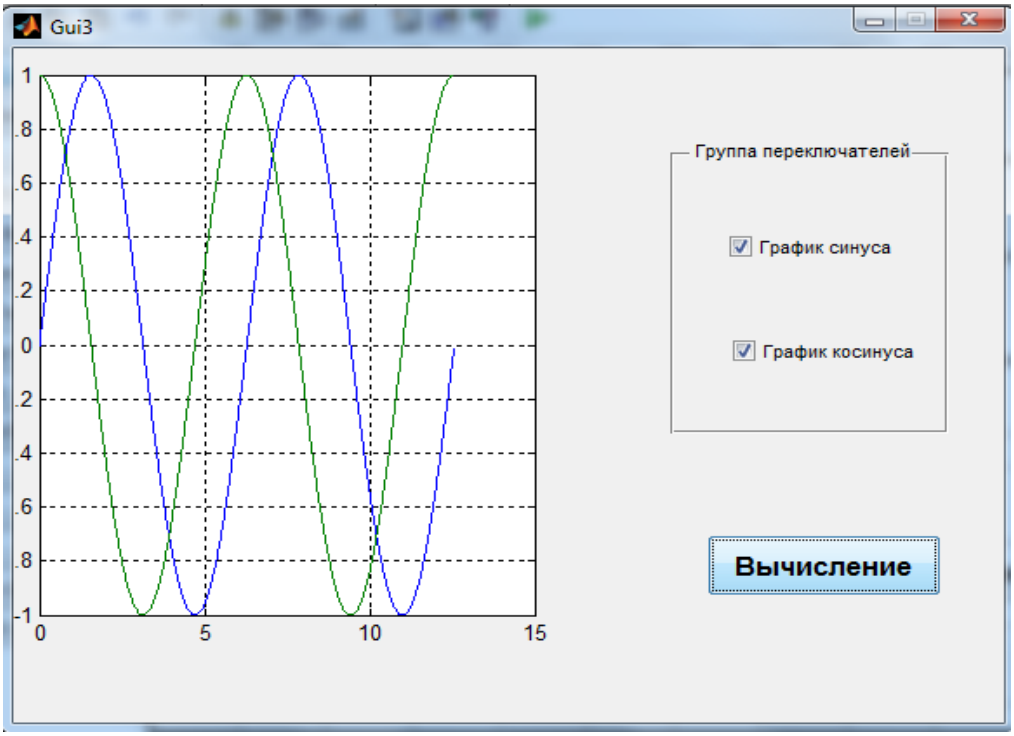
После выполнения этих команд в списке `Listbox` интерфейса пользователя появятся *n* строк с прописями `Name_1`, `Name_2`,…, `Name_n`.
Элементы **`Panel`** и **`Button Group`** создают выделенные области, в которых можно располагать элементы управления.

Задание 1: *Ввести в три поля фамилию имя отчество, сдаваемые ЕГЭ и баллы, проходной балл, просуммировать баллы, выдать результат – поступил или не поступил.*




```
s1=get(handles.text7,'string');
s2=get(handles.text8,'string');
s3=get(handles.text9,'string');
s4=str2num(s1)+str2num(s2)+str2num(s3);
s5=num2str(s4);
set(handles.text16,'string',s5);
s6='ПОЗДРАВЛЯЕМ!!!';
s7='Увы, Вы не прошли!';
if s4>250
     set(handles.text17,'string',s6);
else
   set(handles.text17,'string',s7);
end
```

Задание 2: *Создать GUI с двумя независимыми переключателями (в группе), которые вызывают рисование в графическом поле axes графиков функций sin(x) и cos(x) с двумя периодами при нажатии на кнопку. (x=[0:0.01:4*pi];y1=sin(x);y2=cos(x);)*



Вариант решения:

```
function pushbutton1_Callback(hObject, eventdata, handles)
axes(handles.axes1);
cla;
x=[0:0.01:4*pi];y1=sin(x);y2=cos(x);
a1=get(handles.checkbox1,'value');
a2=get(handles.checkbox2,'value');
if a1==1 plot(x,y1);grid on;
end
if a2==1 plot(x,y2);grid on;
end
if a1*a2==1 plot(x,y1,x,y2);grid on;
end
```

Задание 3: *Создать GUI с двумя графиками, x=-2:0.2:2;y=exp(-x.^2); x1=0:0.1:4*pi; y1=sin(x1); Использовать 4 кнопки (2 «Построить» и 2 «Очистить»). Сделать поясняющие подписи.*

```
function pushbutton1_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)
x=-2:0.2:2;
y=exp(-x.^2);
axes(handles.axes1);
plot(x,y)

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
x1=0:0.1:4*pi;
```

```
y1=sin(x1);
axes(handles.axes2);
plot(x1,y1)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
cla

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes2);
cla

function PlotFplot
% основная функция приложения

% создание графического окна
figure('MenuBar', 'none',...
    'NumberTitle', 'off',...
    'Name', 'PlotFplot',...
    'Units', 'pixels',...
    'Position', [100 100 420 400],...
    'Resize', 'off');
% создание кнопки Plot и связывание с ее событием Callback функции
btnPlot
uicontrol('Style','pushbutton',...
    'String','Plot',...
    'Units','pixels',...
    'Position',[320 10 40 20],...
    'Callback',@btnPlot_Callback);
% создание кнопки Clear с тегом  btnClear
% и связывание с ее событием Callback функции btnClear
uicontrol('Style','pushbutton',...
    'String','Clear',...
    'Units','pixels',...
    'Position',[370 10 40 20],...
    'Enable','off',...
    'Tag','btnClear',...
    'Callback',@btnClear_Callback);
% создание области ввода с тегом edtFun
uicontrol('Style','edit','Units','pixels',...
    'Position',[10 10 300 20],...
    'Tag','edtFun');
% создание осей с тегом axMain
axes('Units','pixels',...
    'Position',[40 50 350 330],...
    'Box','on',...
    'NextPlot', 'add',...
    'Tag','axMain');

function btnPlot_Callback (hObject, eventdata)
```

```
% функция обработки события Callback кнопки Plot

% получение структуры указателей на объекты приложения
handles = guihandles(hObject);
% получение содержимого строки ввода
fun = get(handles.edtFun, 'String');
% построение графика с обработкой исключительных ситуаций
try
    % делаем оси текущими
    axes(handles.axMain)
    % получаем таблицу значений функции
    [x,y] = fplot(fun, [0 10]);
    % строим график линией случайного цвета и толщины 3пт.
    plot(x,y, 'Color', rand(1,3), 'LineWidth', 3)
    % делаем доступной кнопку Clear
    set(handles.btnClear, 'Enable' ,'on')
catch
    % при построении графика произошла ошибка, выводим сообщение об
    % ошибке в функции
    errordlg('Ошибка в функции')
end

function btnClear_Callback (hObject, eventdata)
% функция обработки события Callback кнопки Clear

% получение структуры указателей на объекты приложения
handles = guihandles(hObject);
% делаем оси текущими
axes(handles.axMain)
% очищаем оси
cla
% делаем кнопку Clear недоступной
set(hObject, 'Enable' ,'off')
```