Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №2**

По дисциплине «Логические основы интеллектуальных систем»

Тема: «Программирование в среде системы MATLAB»

**Выполнил:**

Студент 2 курса

Группы ИИ-21

Кирилович А. А.

**Проверил:**

Скарубо А.О.

Брест 2023

**Цель работы:** изучить принципы создания М-функций и использования их в среде MATLAB, а также возможностей создания законченных приложений на базе графического интерфейса пользователя системы MATLAB.

***Вычисление определенных интегралов***

Реализовать М-функцию, вычисляющую интеграл вида

двумя методами интегрирования:

- методом *трапеций*, изменяя шаг интегрирования. Последовательно уменьшая шаги интегрирования в 10 раз (до dx = 0.00001), добиться тем самым стабилизации результата. Контроль достигнутой точности осуществлять путем сравнения последовательных результатов;

- методом Симпсона.

Программа должна запрашивать ввод метода с клавиатуры. Оформить выполнение метода Симпсона подфункцией (вспомогательной функцией). Определить и сравнить производительность М-функции для каждого метода.

Функция в файле integrate.m:

function I = integrate(method)

% Функция для вычисления интеграла

% Определяем переменные

a = 0; b = 5; I = 0; dx = 0.1;

% Вычисляем интеграл

if method == 1

% Метод трапеций

while dx >= 0.00001

% Определяем вектор значений аргумента и функции

x = a:dx:b;

y = sin(x).\*exp(-x);

% Суммируем площади трапеций

I\_prev = I;

I = trapz(x,y);

% Проверяем достижение точности

if abs(I - I\_prev) < 1e-8

break

end

% Уменьшаем шаг интегрирования в 10 раз

dx = dx/10;

end

I = trapz(x,y);

elseif method == 2

% Определяем вектор значений аргумента и функции

y = @(x) sin(x).\*exp(-x);

% Метод Симпсона

I = simpson(y, a, b);

else

error('Некорректный метод интегрирования')

end

% Выводим результат

disp(['Значение интеграла: ', num2str(I)])

end

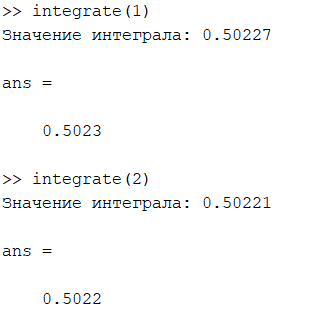
Функция в файле simpson.m:

function I = simpson(y, a, b)

% Функция для вычисления интеграла методом Симпсона

I = quad(y, a, b,'simpson');

end



***Решение систем дифференциальных уравнений***

Реализовать сценарий для решения системы дифференциальных уравнений методом Рунге-Кутта 4-5 порядка.



Начальные условия у1(0) = 0; у2(0) = 1. К – коэффициент нелинейности задачи.

Найти решение системы дифференциальных уравнений на отрезке [0, 20] с заданными начальными условиями при К = 0 (задача становится чисто линейной и описывает гармонические колебания). Повторить решение для нелинейного случая, т.е при К= 0.01.

Визуализировать полученные результаты. Визуализацию осуществить с помощью функции plot (X, y1) и plot (X, y2), где у1 – первый столбец матрицы Y, а y2 – второй столбец матрицы Y, совместив оба графика в одном графическом окне. Линейность либо нелинейность задачи реализовать с помощью выбора кнопки меню.

Функция в файле runge\_kutta.m:

function runge\_kutta(K)

% Функция для вычисления решения системы дифференциальных уравнений

% y1' = K \* x^2 + y2

% y2' = -y1

% с начальными условиями y1(0) = 0, y2(0) = 1

% на отрезке [0, 20]

% K - коэффициент нелинейности

% Задаем функцию правых частей системы

f = @(x, y) [K \* x \* x + y(2); -y(1)];

% Задаем начальные условия

y0 = [0; 1];

% Задаем отрезок интегрирования

tspan = [0, 20];

% Вызываем функцию ode45 для решения системы

[t, y] = ode45(f, tspan, y0)

% Выводим результаты

plot(t, y(:, 1), t, y(:, 2));

legend('y1', 'y2');

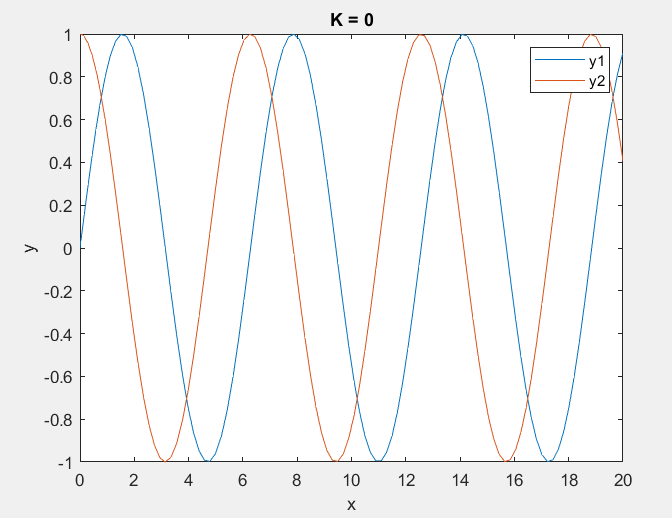
xlabel('x');

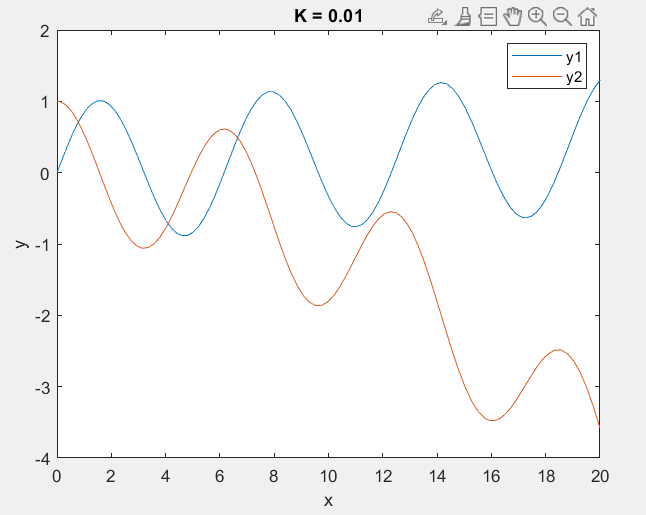
ylabel('y');

title(sprintf('K = %g', K));

end







***Решение системы жестких дифференциальных уравнений***

Реализовать сценарий для решения уравнение Ван-дер-Поля, которое описывает нелинейные релаксационные колебания в различных электронных устройствах имеет вид:



Начальные условия у1(0) = 2; у2(0) = 0; К = 1000.

Найти решение данного уравнения на отрезке [0, 3000] с заданными начальными условиями. Визуализировать решение, а именно первый столбец матрицы Y. Оценить производительность решения.

Функция в файле vanderpoldyn.m:

function dydt = vanderpoldyn(t, y, K)

dydt = [y(2); -y(1) + K\*(1-y(1)^2)\*y(2)];

end

Функция в файле vdp.m:

function vdp()

% Определяем начальные условия

y0 = [2; 0];

K = 1000;

% Решаем уравнение

[t, Y] = ode15s(@(t,y) vanderpoldyn(t,y,K), [0, 3000], y0);

% Визуализируем решение

plot(t, Y(:,1));

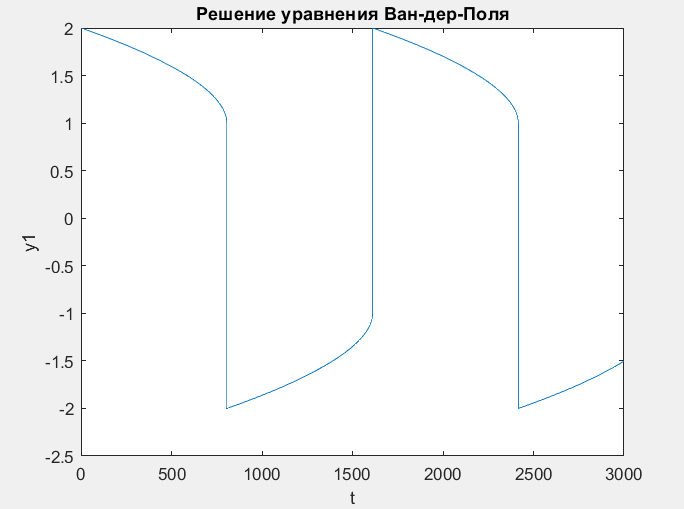
xlabel('t');

ylabel('y1');

title('Решение уравнения Ван-дер-Поля');

end





***Использование графических элементов управления***

Реализовать М-файл, создающий в графическом окне один объект axes, три редактируемых поля для ввода числовой информации, три текстовых поля, содержащих подписи редактируемых полей, одну командную кнопку. Использовать полученное графическое окно для построения графиков функций вида Y = f(X). Массив Х задается левой границей диапазона х0, приращением dx и верхней границей диапазона xn, которые вводятся с клавиатуры в редактируемые поля. По нажатию кнопки «Построить» вывести графическое меню выбора функции. В качестве функций использовать стандартные функции (встроенные в MATLAB, такие как sin, cos, tan, exp) и собственные М-функции. По выбору функции должен строиться ее график.

Функция в файле createAxes.m:

function createAxes()

% Создаем графическое окно с объектом axes

fig = figure();

ax = axes('Parent', fig, 'Position', [0.1 0.3 0.8 0.6]);

% Создаем редактируемые поля и текстовые поля

uicontrol('Style', 'text', 'String', 'x0:', 'Position', [50 240 30 20]);

x0\_edit = uicontrol('Style', 'edit', 'Position', [80 240 100 20], 'String', '0');

uicontrol('Style', 'text', 'String', 'dx:', 'Position', [50 210 30 20]);

dx\_edit = uicontrol('Style', 'edit', 'Position', [80 210 100 20], 'String', '0.1');

uicontrol('Style', 'text', 'String', 'xn:', 'Position', [50 180 30 20]);

xn\_edit = uicontrol('Style', 'edit', 'Position', [80 180 100 20], 'String', '10');

% Создаем командную кнопку

uicontrol('Style', 'pushbutton', 'String', 'Построить', ...

'Position', [50 130 130 30], 'Callback', @plotFunction);

% Функция для построения графика

function plotFunction(~, ~)

% Получаем значения из редактируемых полей

x0 = str2double(get(x0\_edit, 'String'));

dx = str2double(get(dx\_edit, 'String'));

xn = str2double(get(xn\_edit, 'String'));

% Создаем массив x

x = x0:dx:xn;

% Список доступных функций

functions = {'sin', 'cos', 'tan', 'exp', 'myFunction'};

% Диалог выбора функции

[indx,tf] = listdlg('ListString',functions,'Name','Выберите функцию');

if tf == 1

% Выбрана одна из стандартных функций

func\_name = functions{indx};

y = feval(func\_name, x);

plot(ax, x, y);

elseif indx == 5

% Выбрана пользовательская функция

y = myFunction(x);

plot(ax, x, y);

end

end

end

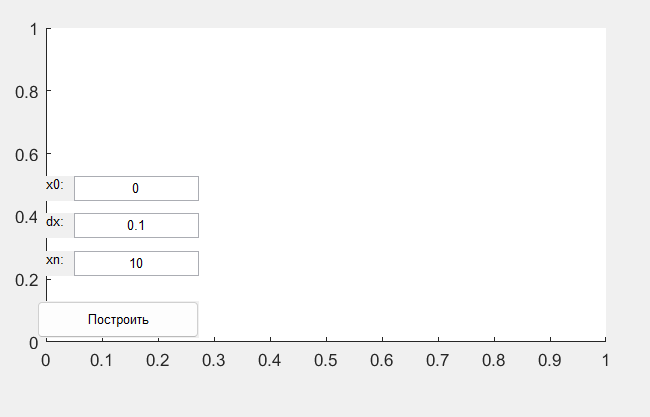
function y = myFunction(x)

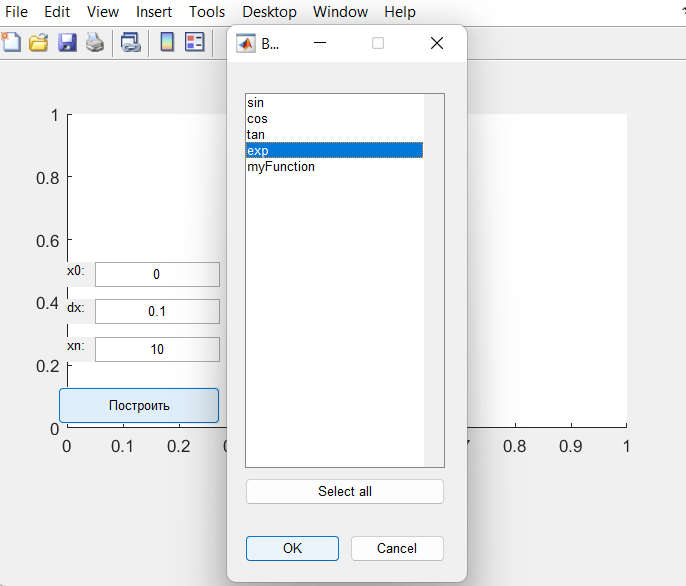
% Замените содержимое этой функции на свою собственную функцию

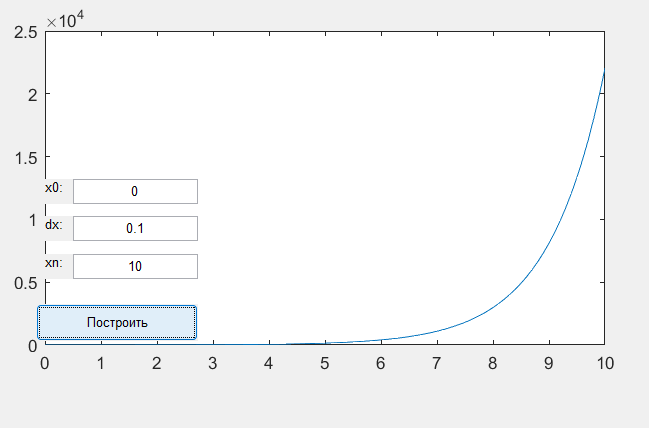
y = x.^2;

end









***Динамическая перестройка элементов управления***

Реализовать проект, в котором на этапе построения графика функции объект axes расширяется и перекрывает элементы управления для ввода начальных значений и запуска вычислений, но при этом появляется новая кнопка, нажатие на которую возвращает все в исходное положение. Реализовать проект для решения жестких дифференциальных уравнений, реализуемых с помощью функции ode15s (п. 2.3).

Функция из файла createAxes2.m:

function createAxes2()

% Создаем графическое окно с объектом axes

fig = figure();

ax = axes('Parent', fig, 'Position', [0.1 0.3 0.8 0.6]);

% Создаем редактируемые поля и текстовые поля

uicontrol('Style', 'text', 'String', 'b1:', 'Position', [50 270 30 20]);

b1\_edit = uicontrol('Style', 'edit', 'Position', [80 270 100 20], 'String', '0');

uicontrol('Style', 'text', 'String', 'b2:', 'Position', [50 240 30 20]);

b2\_edit = uicontrol('Style', 'edit', 'Position', [80 240 100 20], 'String', '10');

uicontrol('Style', 'text', 'String', 'y1:', 'Position', [50 210 30 20]);

y1\_edit = uicontrol('Style', 'edit', 'Position', [80 210 100 20], 'String', '0.1');

uicontrol('Style', 'text', 'String', 'y2:', 'Position', [50 180 30 20]);

y2\_edit = uicontrol('Style', 'edit', 'Position', [80 180 100 20], 'String', '0.1');

% Создаем командную кнопку для построения графика

uicontrol('Style', 'pushbutton', 'String', 'Построить', ...

'Position', [50 130 130 30], 'Callback', @plotFunction);

% Создаем командную кнопку для сброса графика и элементов управления

reset\_button = uicontrol('Style', 'pushbutton', 'String', 'Сброс', ...

'Position', [50 80 130 30], 'Callback', @resetFunction, 'Visible', 'off');

% Функция для построения графика

function plotFunction(~, ~)

% Получаем значения из редактируемых полей

b1 = str2double(get(b1\_edit, 'String'));

b2 = str2double(get(b2\_edit, 'String'));

y1 = str2double(get(y1\_edit, 'String'));

y2 = str2double(get(y2\_edit, 'String'));

% Список доступных функций

functions = {'1', '2', '3'};

% Диалог выбора функции

indx = listdlg('ListString', functions, 'Name', 'Выберите функцию');

switch indx

case 1

f = @(x, y) [y(2); -y(1) + 0.01\*(1-y(1)^2)\*y(2)];

case 2

f = @(x, y) [y(2); -y(1) + 0.1\*(1-y(1)^2)\*y(2)];

case 3

f = @(x, y) [y(2); -y(1) + 1\*(1-y(1)^2)\*y(2)];

end

[t, y] = ode15s(f, [b1, b2], [y1, y2]);

% Визуализируем решение

plot(t, y(:,1));

xlabel('t');

ylabel('y1');

% Расширяем объект axes, чтобы он перекрывал элементы управления

ax.Position = [0.1 0.1 0.8 0.8];

% Скрываем элементы управления

set(b1\_edit, 'Visible', 'off');

set(b2\_edit, 'Visible', 'off');

set(y1\_edit, 'Visible', 'off');

set(y2\_edit, 'Visible', 'off');

set(reset\_button, 'Visible', 'on');

end

% Функция для сброса графика и элементов управления

function resetFunction(~, ~)

% Очищаем график

cla(ax);

% Возвращаем объект axes в исходное положение

ax.Position = [0.1 0.3 0.8 0.6];

% Показываем элементы управления

set(b1\_edit, 'Visible', 'on');

set(b2\_edit, 'Visible', 'on');

set(y1\_edit, 'Visible', 'on');

set(y2\_edit, 'Visible', 'on');

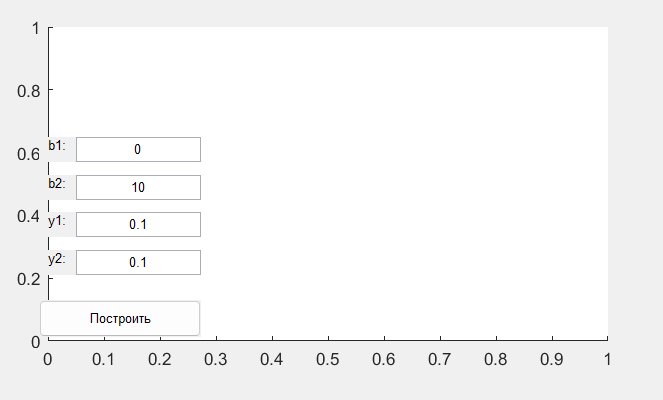
set(gcbo, 'Visible', 'on'); % gcbo - объект, инициировавший вызов функции

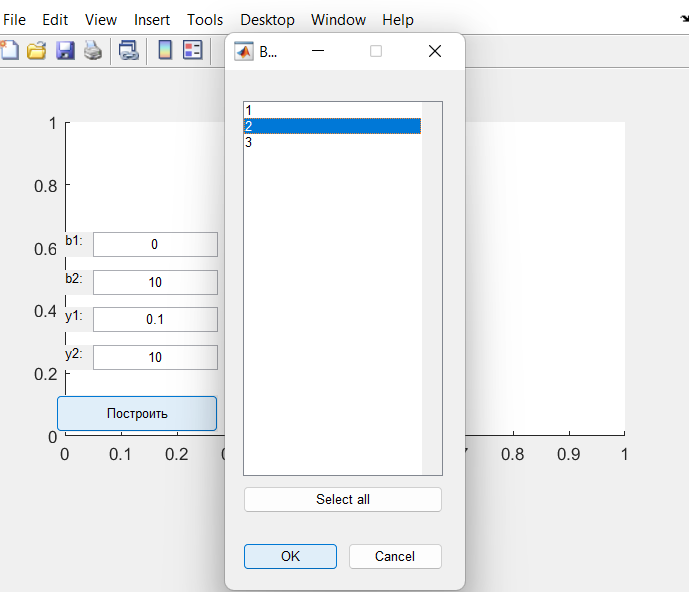
set(reset\_button, 'Visible', 'off');

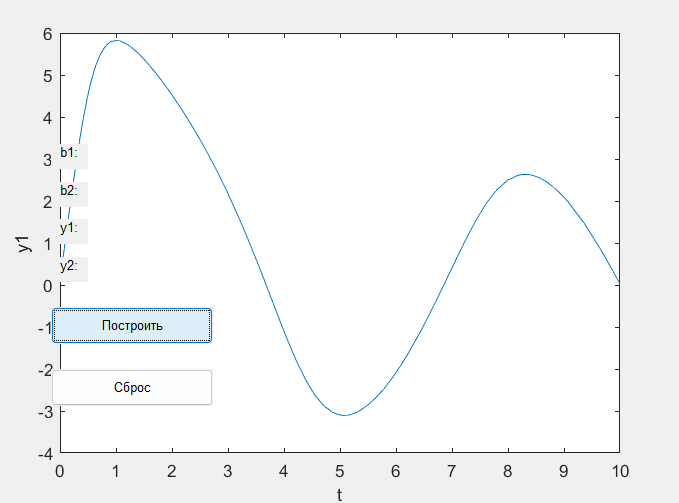
end

end









***Контрольные вопросы***

1. Сценарии и М-функции - это два разных типа файлов, которые используются в MATLAB для выполнения задач. Сценарий - это последовательность команд MATLAB, сохраненных в файле с расширением .m. Когда сценарий запускается, MATLAB выполняет команды последовательно. Сценарии не имеют входных и выходных аргументов и не возвращают значений.

М-функция - это файл MATLAB, который содержит определение одной или нескольких функций. М-функции могут иметь входные и выходные аргументы, могут возвращать значения и могут быть вызваны из других функций, сценариев или командного окна MATLAB.

2. Синтаксис определения и вызова М-функций:

Определение функции:

function [возвращаемое\_значение] = имя\_функции(входные\_аргументы) % Описание функции % тело функции

end

Вызов функции:

[результат] = имя\_функции(аргумент1, аргумент2, ...);

3. Функции MATLAB интерактивного взаимодействия М-функций с пользователем: В MATLAB есть несколько функций, которые позволяют М-функциям взаимодействовать с пользователем, например: input, menu, msgbox, uicontrol и др.

Например, функция input запрашивает ввод данных от пользователя, функция menu выводит меню для выбора опции, функция msgbox отображает сообщение в модальном окне.

4. Механизмы передачи информации между М-функциями и командным окном: М-функции могут получать информацию от командного окна, используя входные аргументы. Результат выполнения М-функции может быть передан обратно в командное окно с помощью выходных аргументов.

5. Как оценить производительность М-функций? Для оценки производительности М-функций можно использовать функцию tic-toc. Она измеряет время выполнения кода между вызовами функций tic и toc.

Например:

tic

% код, время выполнения которого нужно измерить

toc

6. Графические элементы управления: В MATLAB есть несколько графических элементов управления, таких как кнопки, текстовые поля, списки, флажки и т.д. Они могут быть добавлены на графический интерфейс пользователя (GUI) с помощью функций uicontrol, uipanel и др.

7. Функции-конструкторы и функции обработки событий в MATLAB используются для создания графических элементов управления и определения их поведения.

Функции-конструкторы, такие как uicontrol, uimenu, uipanel и другие, используются для создания графических элементов управления, таких как кнопки, меню, панели и т.д. Эти функции принимают параметры, такие как тип элемента управления, его положение и размер, а также обработчики событий. Функции-конструкторы возвращают объекты, которые представляют графические элементы управления.

Функции обработки событий используются для определения поведения графических элементов управления при возникновении определенных событий, таких как нажатие кнопки мыши или изменение значения текстового поля. Функции обработки событий связываются с графическими элементами управления с помощью свойств объектов, таких как Callback, и выполняются при возникновении определенного события.

Например, для создания кнопки с помощью функции-конструктора uicontrol и связывания ее с функцией обработки события можно использовать следующий код:

hButton = uicontrol('Style', 'pushbutton', 'String', 'Нажми меня', 'Position', [10 10 100 50], 'Callback', @buttonCallback);

function buttonCallback(src, event)

disp('Кнопка была нажата');

end

Этот код создаст кнопку с текстом "Нажми меня" и свяжет ее с функцией buttonCallback, которая будет вызвана при нажатии кнопки и выведет сообщение в командное окно.

Функции-конструкторы и функции обработки событий позволяют создавать интерактивные пользовательские интерфейсы в MATLAB, которые позволяют пользователям взаимодействовать с данными и выполнять различные операции.

**Вывод:** изучил принципы создания М-функций и использования их в среде MATLAB, а также возможностей создания законченных приложений на базе графического интерфейса пользователя системы MATLAB.