Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра сетей и устройств телекоммуникаций

А. А. Борискевич, А. Ю. Лагойко, В. Ю. Цветков

ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА МУЛЬТИМЕДИЙНОЙ ИНФОРМАЦИИ В ИНТЕГРИРОВАННОЙ СРЕДЕ MATLAB/C

МЕТОДИЧЕСКОЕ ПОСОБИЕ
по курсу
«Цифровая обработка и защита мультимедийной информации»
для студентов специальности
«Системы распределения мультимедийной информации»
всех форм обучения

Минск БГУИР 2011

УДК 004.032.6(076)
ББК 32.973.202-018.2я73
Б82

Р е ц е н з е н т:
доцент кафедры телекоммуникаций учреждения образования
«Белорусский государственный университет
информатики и радиоэлектроники»,
кандидат технических наук В. Н. Урядов

Борискевич, А. А.

Б82 Предварительная обработка мультимедийной информации в интегрированной среде MatLab/C : метод. пособие по курсу «Цифровая обработка и защита мультимедийной информации» для студ. спец. «Системы распределения мультимедийной информации» всех форм обуч. / А. А. Борискевич, А. Ю. Лагойко, В. Ю. Цветков. – Минск : БГУИР, 2011. – 53 с. : ил.

ISBN 978-985-488-529-2.

Рассмотрены вопросы разработки программных средств в средах MatLab и MS Visual Studio, создания графических пользовательских интерфейсов в среде MatLab, особенности доступа к медиаданным и их предварительной обработки с использованием встроенных функций MatLab. Приведены примеры программ доступа к файлам мультимедийных данных и дискретных информационных преобразований.

Предназначено для студентов специальности «Системы распределения мультимедийной информации», может быть использовано при курсовом и дипломном проектировании.

УДК 004.032.6(076)
ББК 32.973.202-018.2я73

ISBN 978-985-488-529-2

Содержание

Введение

Предварительная обработка является важной процедурой, выполняемой при различных операциях с мультимедийной информацией (фильтрация, сжатие, шифрование, стеганографическая защита, распознавание). Основной целью предварительной обработки медиаданных является согласование формата их представления с форматом обработки.

Одним из наиболее мощных программных пакетов обработки мультимедийных данных является MatLab. С его помощью может осуществляться программное моделирование процессов и систем передачи информации. Основные достоинства применения MatLab для программного моделирования состоят в простоте программирования (синтаксис команд интерпретатора MatLab аналогичен синтаксису С-программ), удобстве отладки программ (отсутствует необходимость в компиляции программы перед ее выполнением, после выполнения программы доступна информация о состояниях всех ее переменных), простоте и эффективности визуализации результатов выполнения программы (формирование одномерных, двухмерных и трехмерных графических объектов), поддержке большинства стандартных форматов представления и хранения файлов данных, аудио- и видеоинформации (неподвижных и подвижных изображений), наличии развитой библиотеки встроенных функций, реализующих основные операции обработки (информационные преобразования, фильтрация и т. д.) и хорошей документированности встроенных функций (поиск по функциям, описание функций, теоретические сведения по цифровой обработке сигналов, примеры программ).

Важной особенностью программного пакета MatLab является возможность его интеграции со средой разработки и компиляции программ, например пакетом MS Visual C++. В результате возможна разработка интегрированных MaLab/C-программ. Преимущества такого подхода при моделировании заключаются в следующем. Программы на С обеспечивают в ряде случаев более высокую скорость вычислений по сравнению с MatLab-программами. Однако программными средствами MatLab существенно проще, чем С-средствами, реализуются визуализация результатов, выделение памяти и доступ к медиафайлам. Это позволяет не только повысить эффективность моделирования, но и упростить процесс отладки С-программ.

Целью данного пособия является привитие навыков написания программ в средах MatLab, MS Visual C++, а также в интегрированной среде MatLab/C. В пособии отражен опыт многолетней научно-исследовательской работы авторов в области обработки изображений и звука [1 – 9].

1. Программное моделирование в интегрированной среде MatLab/C

1.1. Интегрирование программных сред MatLab и C

Среда моделирования MatLab и среда разработки программного обеспечения MS Visual C++ допускают интеграцию на уровне приложений, совместное выполнение и обмен данными между программными модулями MatLab и C, скомпилированными в формат dll, в составе MatLab-оболочки. В результате интеграции обеспечивается возможность создания быстродействующих приложений за счет реализации основных вычислительных процедур на C и сокращения сроков построения программных моделей за счет использования стандартных библиотек функций обработки сигналов, реализации процедур визуализации результатов моделирования в среде MatLab и поддержки средой MatLab наиболее распространенных форматов изображений, звука и видео.

Интеграция MatLab и C выполняется в среде MatLab после установки на рабочей станции соответствующих программных пакетов. Для пакета MatLab 6.5 процесс интеграции запускается с консоли рабочего окна MatLab в результате вызова и определения аргументов следующих функций:

mex –setup
mbuild –setup
cd(prefdir)
mcc save path

При вызове функций *mex* и *mbuild* в процессе диалога выбирается c-компилятор и среда разработки c-программ (например MS Visual C++ 6.0). В результате интеграции в среде MS Visual C++ появляется опция создания MatLab-проекта. При ее активации формируется c-файл, содержащий пустую mex-функцию – стандартную оболочку для встраивания c-кода в MatLab-приложение следующего вида.

```
#include "mex.h"
void mexFunction(
    int nlhs,           // Число возвращаемых в MatLab значений
    mxArray *plhs[],    // Массив указателей на возвращаемые значения
       int nrhs,        // Число аргументов, получаемых из MatLab
    const mxArray *prhs[]  // Массив указателей на аргументы
    )
{/* C-код. Может содержать вызовы других C-функций, размещенных как в этом, так и в различных C-файлах, связанных с данной функцией */}
```

C-код должен содержать описание аргументов и выходных значений, а также обеспечивать выделение для них памяти. Компиляция данной c-функции приводит к созданию dll-файла, который может быть вызван из m-файла по имени с указанием необходимых аргументов и возвращаемых значений.

1.2. Обмен данными между m- и c- программными модулями

Обмен данными между m- и c- программными модулями осуществляется через передачу из m-функции в c-функцию значений аргументов и передачу в обратном направлении возвращаемых значений. Для некоторой c-функции *some_func* обмен данными (аргументами *arg_1 .. arg_4* и возвращаемыми значениями *func_val_1 .. func_val_4*) обеспечивается следующей записью в m-файле или консоли MatLab

[func_val_1 ,func_val_2, func_val_3 func_val_4]=some_func(arg_1, arg_2, arg_3, arg_4)

Аргументы и формируемые значения могут быть простыми переменными, одномерными массивами типа «строка» или «столбец», а также двухмерными массивами. В целях исключения конфликтов программных приложений для аргументов и формируемых значений необходимо использовать формат *double*, обеспечивающий в MatLab и C одинаковое представление данных. Фрагмент c-кода, реализующий описание аргументов и формируемых значений, имеет следующий вид:

```
// Описание аргументов, принимаемых из MatLab
double* arg_1; // Простая переменная типа double
double* arg_2; // Одномерный массив типа «строка»
double* arg_3; // Одномерный массив типа «столбец»
double* arg_4; // Двухмерный массив

// Описание значений, передаваемых в MatLab
double* func_val_1; // Простая переменная типа double
double* func_val_2; // Одномерный массив типа «строка»
double* func_val_3; // Одномерный массив типа «столбец»
double* func_val_4; // Двухмерный массив

// Выделение памяти для аргументов, принимаемых из MatLab
arg_1 = mxGetPr(prhs[0]);
arg_2 = mxGetPr(prhs[1]);
arg_3 = mxGetPr(prhs[2]);
arg_4 = mxGetPr(prhs[3]);

// Выделение памяти для формируемых значений, передаваемых в MatLab
plhs[0]=mxCreateDoubleMatrix(1, 1, mxREAL); func_val_1= mxGetPr(plhs[0]);
plhs[1]=mxCreateDoubleMatrix(1, X, mxREAL); func_val_2= mxGetPr(plhs[1]);
plhs[2]=mxCreateDoubleMatrix(Y, 1, mxREAL); func_val_3= mxGetPr(plhs[2]);
plhs[3]=mxCreateDoubleMatrix(Y, X, mxREAL); func_val_4= mxGetPr(plhs[3]);
```

Локальные переменные c-кода, в том числе передаваемые в качестве аргументов и значений c-функций, могут иметь любой тип, требуемый в интересах обработки данных. В приведенном фрагменте *X* и *Y* – это переменные или константы формата *unsigned long int*, определяющие размер массива. Для создания локальных массивов необходимо выделение и освобождение памяти, посредством функций *calloc( )* и *free( )*.

1.3. Структура m-проекта

М-проект может включать один или несколько m-файлов.

Если m-проект состоит из одного файла, то в нем помещается текст программы, содержащий вызовы только стандартных функций MatLab.

Во втором случае один из m-файлов является главным (*main.m* на рис. 1.1) и содержит текст программы. Текст программы содержит вызовы стандартных функций (на рис. 1.1 не показаны), а также функций пользователя (*some_func_1* и *some_func_2* на рис. 1.1). Еще несколько файлов (*some_func_1.m*, *some_func_2.m*, *some_func_3.m* на рис. 1.1) содержат исходные тексты функций пользователя (*some_func_1*, *some_func_2*, *some_func_3.m*,). На рис. 1.1 из тела функции *some_func_1* осуществляется вызов функции *some_func_3*.

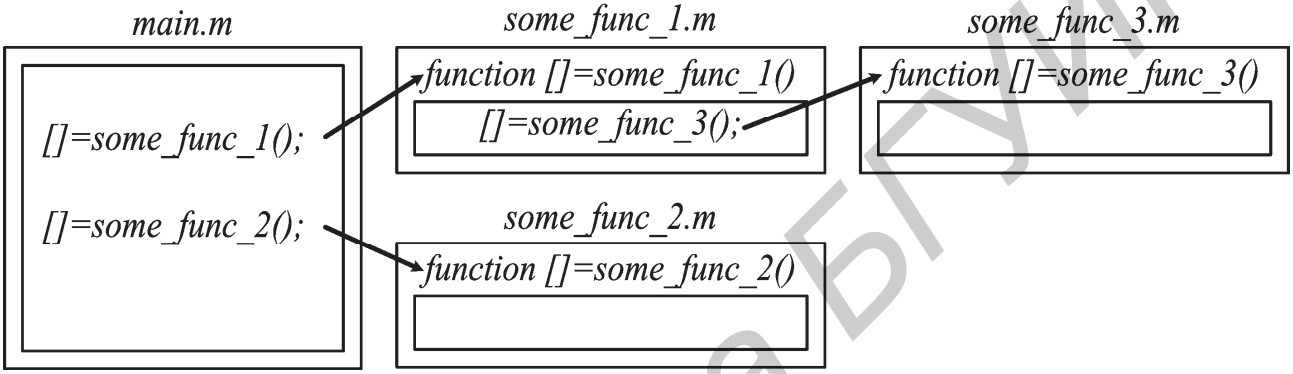

Рис. 1.1. Пример структуры m-проекта

Для лучшей читаемости m-проекта имена файлов и имена вложенных в них функций пользователя целесообразно выбирать одинаковыми.

1.4. Программирование в среде MatLab

Создание m-проекта из одного или нескольких m-файлов выполняется по следующему алгоритму.

1. Создать каталог проекта, в котором будут размещаться m-файлы проекта.
2. Выбрать кнопку «…» верхней панели главного окна MatLab, расположенную справа от поля «Current Directory».
3. В открывшемся диалоговом окне указать путь к каталогу проекта и выбрать кнопку «Ok».
4. В меню «File» главного окна MatLab выбрать последовательно опции «New» и «M-file».
5. В открывшемся окне редактора набрать текст программы или текст функции пользователя.
6. Сохранить набранный текст в файле каталога проекта (например, под именем *main.m* для главного файла проекта с текстом основной программы или под именем *some_func_x.m* для файла с кодом функции пользователя), воспользовавшись стандартными диалоговыми элементами окна редактора.
7. Создать при необходимости остальные файлы m-проекта, повторив пп. 4 – 6.

Запуск главного (или единственного) файла m-проекта на выполнение может производиться из консоли главного (командного) окна MatLab или из окна редактора. В первом случае необходимо набрать имя главного (или единственного) файла проекта в консоли командного окна MatLab и нажать клавишу «Enter». Во втором случае главный (или единственный) файл проекта необходимо открыть в редакторе и выбрать опцию «Run» в меню «Debug».

Результаты выполнения программы, а также сообщения об ошибках и предупреждения отображаются в командном окне MatLab. Состояние переменных программы, содержащиеся в главном (или единственном) файле, после окончания выполнения программы отображаются в поле «Workspace» главного окна MatLab. Данное поле может быть очищено вводом команды *clear all* в консоли командного окна MatLab.

Справка об операторах и функциях среды MatLab, а также о возможностях самой среды может быть вызвана выбором опции «Full Product Family Help» в меню «Help» главного окна MatLab или вводом команды *help* в консоли командного окна.

Далее приведено несколько примеров, реализующих в среде MatLab формирование матрицы $E_{2\times N_X} = \left\| e_{i,j} \right\|_{(i=\overline{1,2},\, j=\overline{1,N_X})}$ случайных значений по выражениям

$$\begin{cases} E_{1\times N_X}(1) = C_{1\times N_Y} \cdot \left(A_{N_Y\times N_X} + B_{N_Y\times N_X} \right) - D_{1\times 1}, \\ E_{1\times N_X}(2) = C_{1\times N_Y} \cdot \left(A_{N_Y\times N_X} - B_{N_Y\times N_X} \right) - D_{1\times 1}, \end{cases} \quad (1.1)$$

где $A_{N_Y\times N_X} = \left\| a_{i,j} \right\|_{(i=\overline{1,N_Y},\, j=\overline{1,N_X})}$ и $B_{N_Y\times N_X} = \left\| b_{i,j} \right\|_{(i=\overline{1,N_Y},\, j=\overline{1,N_X})}$ – двухмерные матрицы случайных значений; $C_{1\times N_Y} = \left\| c_i \right\|_{(i=\overline{1,N_Y})}$ – одномерной матрица случайных значений; $D_{1\times 1} = \left\| d \right\|$ – случайное значение (матрица 1×1); N_Y, N_X – переменные, определяющие размеры матриц.

Пример 1.1. М-проект из одного файла (*single_.m*) с текстом программы вычисления выражения (1.1).

```
% - это символ комментария, символы справа от него игнорируются
% интерпретатором MatLab
Ny=8; % число строк матриц A и B
Nx=16; % число столбцов матриц A и B
A=randn([Ny Nx]); % создание массива A случайных значений
B=randn([Ny Nx]);
C=randn([1 Ny]); % создание строки C случайных значений
D=randn([1 1]); % создание случайного значения D
E=zeros(2,Nx); % создание массива E результатов
E(1,:)=C*(A+B)-D;
E(2,:)=C*(A-B)-D % Отсутствие символа «;» в конце строки обеспечивает
% отображение элементов матрицы E в командном окне
```

Пример 1.2. М-проект, реализующий вычисление выражения (1) и состоящий из четырех файлов (*main.m, some_func_1.m*, *some_func_2.m*, *some_func_3.m*, интегрированных по схеме рис. 1.1) с текстами основной программы и функций пользователя. Файл *some_func_1.m* содержит реализацию функции *some_func_1*, вычисляющей значения выражений $ABC_{1\times N_X}(1) = C_{1\times N_Y} \cdot \left(A_{N_Y\times N_X} + B_{N_Y\times N_X}\right)$ и $ABC_{1\times N_X}(2) = C_{1\times N_Y} \cdot \left(A_{N_Y\times N_X} - B_{N_Y\times N_X}\right)$. Файл *some_func_2.m* содержит реализацию функции *some_func_2*, вычисляющей выражение $ABCD_{1\times N_X} = ABC_{1\times N_X} - D_{1\times 1}$.

Текст основной программы в файле *main.m*.

```
Ny=8; Nx=16;
A=randn([Ny Nx]); B=randn([Ny Nx]);
C=randn([1 Ny]); D=randn([1 1]); E=zeros(2,Nx);
[F, G]=some_func_1(A, B, C);
E(1,:)=some_func_2(F, D);
E(2,:)=some_func_2(G, D)
T(1,:)=C*(A+B)-D; T(2,:)=C*(A-B)-D
R=abs(E-T) % Матрица различий в результатах
```

Реализация функции пользователя *some_func_1* в файле *some_func_1.m*.

```
function [ABC1, ABC2]=some_func_1(A, B, C)
AB=A+B; ABC1=some_func_3(AB, C);
AB=A-B; ABC2=some_func_3(AB, C);
```

Реализация функции пользователя *some_func_2* в файле *some_func_2.m*.

```
function [ABCD]=some_func_2(ABC, D)
ABCD=ABC-D;
```

Реализация функции пользователя *some_func_3* в файле *some_func_3.m*.

```
function [ABC]=some_func_3(AB, C)
ABC=C*AB;
```

1.5. Структура интегрированного m/c-проекта

Интегрированный m/c-проект включает по крайней мере два файла: главный m-файл, из которого вызывается mex-функция пользователя (*mexFunction()*), и файл динамической библиотеки, который содержит реализацию mex-функции пользователя. Мех-функция пользователя вызывается по имени файла динамической библиотеки (*.dll). В общем случае главный m-файл интегрированного m/c-проекта может содержать обращение как к m-функциям пользователя, так и к mex-функциям пользователя.

В качестве примера на рис. 1.2 представлена структура интегрированного m/c-проекта, состоящего из 9 файлов: файла, содержащего m-код (*main.m*) с вызовом двух mex-функций пользователя (*some_func_1mc и some_func_2mc*), представленных dll-файлами (*some_func_1mc.dll и some_func_2mc.dll*). Файл ди-

намической библиотеки *some_func_1mc.dll* сформирован в результате компиляции исходных с-кодов, содержащихся в файлах *some_func_1.h, some_func_1.c, some_func_3.h, some_func_3.c*. Файл динамической библиотеки *some_func_2mc.dll* сформирован в результате компиляции исходного с-кода, содержащегося в файле *some_func_2.c*.

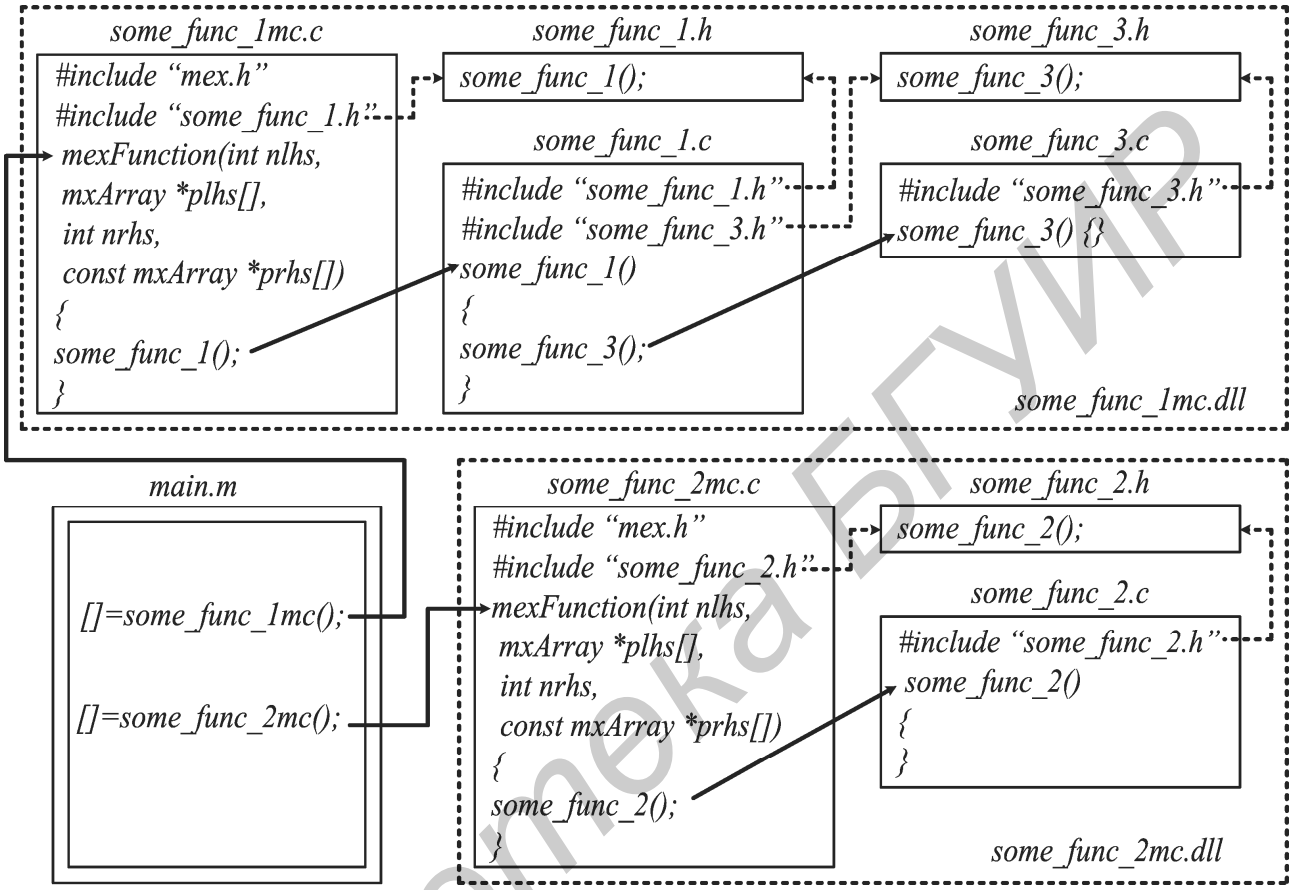

Рис. 1.2. Пример структуры интегрированного m/c-проекта

Для лучшей читаемости m/c-проекта имена файлов и имена вложенных в них функций пользователя целесообразно выбирать одинаковыми.

1.6. Программирование в интегрированной среде MatLab/C

Создание интегрированного m/c-проекта из одного или нескольких m-файлов и одного или нескольких файлов динамических библиотек выполняется отдельно для m- и dll-файлов.

Создание m-файлов выполняется по алгоритму, представленному в подразд. 1.4.

Создание файла динамической библиотеки для интегрированного m/c-проекта на языке программирования C выполняется по следующему алгоритму.

1. Загрузить среду MS Visual C++ (например MS Visual C++ 6.0).

2. В меню «File» выбрать опцию «New».

3. В открывшемся диалоговом окне «New» выбрать пункт «MATLAB Project Wizard», задать место расположения проекта на жестком диске, указав путь в поле «Location», задать имя проекта в поле «Project name» (например

some_func_xmc) и выбрать кнопку «Ok». При выборе кнопки «Ok» автоматически создается каталог с именем проекта (*some_func_xmc*), содержащий основной файл проекта с расширением dsw (*some_func_xmc.dsw*). Имя проекта совпадает с именем формируемого при компиляции dll-файла (*some_func_xmc.dll*). Пункт «MATLAB Project Wizard» в окне «New» появляется после интегрирования сред MatLab и MS Visual C как описано в подразд. 1.1.

4. В открывшемся диалоговом окне «MATLAB Project Wizard» выбрать значение «C-MEX DLL» в поле «Visual MATLAB Application Type» и выбрать кнопку «Finish».

5. Ознакомиться с информацией о создаваемом проекте, выводимой в окне «New Project Information» и выбрать кнопку «Ok» для подтверждения операции создания проекта или кнопку «Cancel» для отказа и возврата в окно «MATLAB Project Wizard». При выборе кнопки «Ok» в каталоге проекта (*some_func_xmc*) автоматически создается каталог «Debug» и с-файл (*some_func_xmc.c*), содержащий пустую реализацию mex-функции пользователя.

6. В открывшемся диалоговом окне «Please choose C-file to add to project» выбрать каталог проекта (*some_func_xmc*) и открыть с-файл (*some_func_xmc.c*), содержащий пустую реализацию mex-функции пользователя. При этом на экран выводится окно с содержимым текстового файла mcc-log.txt, которое может быть закрыто. В поле «FileView» основного окна MS Visual C++ отображается рабочее пространство проекта «Workspace (‘*some_func_xmc*’): 1 project(s)», содержащее папки «Source Files», «Header Files», «Resource Files» и «MATLAB C/C++ files», по которым распределяются с- и h-файлы проекта «(*some_func_xmc*) files».

7. Если создаваемый файл динамической библиотеки интегрированного m/c-проекта должен включать только один с-файл с mex-функцией пользователя, то необходимо выбрать папку «MATLAB C/C++ files» рабочего пространства проекта «Workspace (‘*some_func_xmc*’): 1 project(s)» и открыть с-файл (*some_func_xmc.c*) с mex-функцией пользователя. При этом на экран выводится окно (*some_func_xmc.c*), содержащее пустую реализацию mex-функции пользователя. Это исходный с-код, приведенный в подразд. 1.1 (текст после символа // и между символами /* и */ является комментарием). В фигурных скобках *{ }* необходимо заключить реализацию mex-функции пользователя и выполнить компиляцию программы, выбрав опцию «Build (*some_func_xmc.dll*)». При отсутствии ошибок в каталоге проекта сформируется файл динамической библиотеки (*some_func_xmc.dll*).

8. Если создаваемый файл динамической библиотеки интегрированного m/c-проекта должен включать кроме с-файла с mex-функцией пользователя также файлы с другими функциями пользователя, то необходимо создать эти файлы. Для этого следует выбрать опцию «New» в меню «File». В открывшемся диалоговом окне «New» выбрать в поле Files указать тип файла: «C++ Source File» для исходного файла (*.c) или «C/C++ Header File» для заголовочного (интерфейсного) файла (*.h). Установить флаг «Add to project» и задать имя файла в поле «File name:» в формате *.c, явно указав его расширение. Создаваемые та-

ким образом файлы автоматически распределяются по папкам «Source Files» и «Header Files» рабочего пространства проекта «Workspace (‘*some_func_xmc*’): 1 project(s)». Данные файлы должны содержать реализации и интерфейсы функций пользователя, вызываемых из основного файла проекта непосредственно или косвенно. Выполнить п. 7 алгоритма.

Созданные в соответствие с приведенным алгоритмом файлы динамических библиотек вызываются из главного m-файла интегрированного m/c-проекта по именам.

Запуск главного m-файла (*main.m*) интегрированного m/c-проекта на выполнение производится также как и главного (единственного) файла m-проекта (см. подразд. 1.4). Перед этим необходимо скопировать файлы динамических библиотек (*some_func_xmc.dll*) в каталог интегрированного m/c-проекта, где расположен главный m-файл (*main.m*).

Пример 1.3. M/c-проект, реализующий вычисление выражения (1) и состоящий из 9 файлов, интегрированных по схеме рис. 1.2 (*main.m, some_func_1mc.c*, *some_func_1.h*, *some_func_1.c*, *some_func_3.h*, *some_func_3.c*, *some_func_2mc.c*, *some_func_2.h*, *some_func_2.c*) с текстами основной m-программы, а также mex- и c- функций пользователя. Кроме того, m/c-проект включает m-файл *some_func_4.m* с функцией пользователя, реализующей сравнение результатов вычисления выражения (1) с помощью mex- и m-функций пользователя. Файлы *some_func_1mc.c*, *some_func_1.h*, *some_func_1.c*, *some_func_3.h*, *some_func_3.c* интегрируются в проект *some_func_1mc* и компилируются в файл динамической библиотеки *some_func_1mc.dll*. Файлы *some_func_2mc.c*, *some_func_2.h*, *some_func_2.c* интегрируются в проект *some_func_2mc* и компилируются в файл динамической библиотеки *some_func_2mc.dll*. Файл *some_func_1mc.dll* содержит реализацию mex-функции пользователя *some_func_1mc*, вычисляющей значения выражений $ABC_{1\times N_X}(1) = C_{1\times N_Y} \cdot \left(A_{N_Y\times N_X} + B_{N_Y\times N_X}\right)$ и $ABC_{1\times N_X}(2) = C_{1\times N_Y} \cdot \left(A_{N_Y\times N_X} - B_{N_Y\times N_X}\right)$. Файл *some_func_2mc.dll* содержит реализацию mex-функции пользователя *some_func_2*, вычисляющей выражение $ABCD_{1\times N_X} = ABC_{1\times N_X} - D_{1\times 1}$. Обмен данными между функциями пользователя осуществляется через структурные переменные, описанные в заголовочных файлах.

Текст основной программы в файле *main.m*.

```
Ny=8; Nx=16;
A=randn([Ny Nx]); B=randn([Ny Nx]);
C=randn([1 Ny]); D=randn([1 1]); E=zeros(2,Nx);
[F, G]=some_func_1mc(Ny, Nx, A', B', C); % символ ' означает транспонирование
E(1,:)=some_func_2mc(Nx, F, D); % символ : указывает на все элементы матрицы
E(2,:)=some_func_2mc(Nx, G, D)
R=some_func_4(A, B, C, D, E)
```

Реализация mex-функции пользователя *some_func_1mc* в файле *some_func_1mc.c*.

```
// - это символ комментария, символы справа от него игнорируются
// компилятором MS Visual C++
#include "mex.h" // подключение стандартной mex-библиотеки
#include "some_func_1.h" // подключение заголовочного (интерфейсного) файла
void mexFunction(
        int nlhs,           // Number of left hand side (output) arguments
        mxArray *plhs[],    // Array of left hand side arguments
        int nrhs,           // Number of right hand side (input) arguments
        const mxArray *prhs[]  // Array of right hand side arguments
)
{
// Объявление входных переменных
double* Ny_; // Объявление указателя на память типа double
double* Nx_;
double* A_;
double* B_;
double* C_;
//
// Объявление выходных переменных
double* F_;
double* G_;
//
// Объявление внутренних переменных
int r;
unsigned long int Nyx, Ny, Nx;
some_func_1_str sf1s; // объявление структурной переменной типа some_func_1_str
//
// Выделение памяти для входных переменных
Ny_=mxGetPr(prhs[0]);
Nx_=mxGetPr(prhs[1]);
A_=mxGetPr(prhs[2]);
B_=mxGetPr(prhs[3]);
C_=mxGetPr(prhs[4]);
//
// Инициализация внутренних переменных
Ny=(unsigned long int)*Ny_;
Nx=(unsigned long int)*Nx_;
Nyx=Ny*Nx;
//
// Выделение памяти для выходных переменных
plhs[0]=mxCreateDoubleMatrix(1, Nx, mxREAL);
F_=mxGetPr(plhs[0]);
plhs[1]=mxCreateDoubleMatrix(1, Nx, mxREAL);
```

```
G_=mxGetPr(plhs[1]);
//
sf1s.Ny=Ny; sf1s.Nx=Nx;
sf1s.A=A_; sf1s.B=B_; sf1s.C=C_; sf1s.ABC1=F_; sf1s.ABC2=G_;
r=some_func_1(&sf1s);
}
```

Интерфейс функции *some_func_1* в заголовочном файле *some_func_1.h*.

```
typedef struct
        {
        unsigned long int Ny;
        unsigned long int Nx;
        double* A;
        double* B;
        double* C;
        double* ABC1;
        double* ABC2;
        } some_func_1_str; // объявление структурированного типа данных
int some_func_1(some_func_1_str*); // объявление функции
```

Реализация функции пользователя *some_func_1* в файле *some_func_1.c*.

```
#include "some_func_1.h"
#include "some_func_3.h"
//
int some_func_1(some_func_1_str* sf1s)
{
int r;
unsigned long int i, Nyx;
double* AB1;
double* AB2;
some_func_3_str sf3s;
//
Nyx=sf1s->Ny*sf1s->Nx;
AB1=(double*)calloc(Nyx, sizeof(double)); // выделение памяти для переменной
AB2=(double*)calloc(Nyx, sizeof(double));
//
i=0;
while (i<Nyx)
        {
        *(AB1+i)=*(sf1s->A+i)+*(sf1s->B+i);
        *(AB2+i)=*(sf1s->A+i)-*(sf1s->B+i);
        i++;
        }
//
```

```
sf3s.Ny=sf1s->Ny; sf3s.Nx=sf1s->Nx; sf3s.C=sf1s->C;
//
sf3s.ABC=sf1s->ABC1; sf3s.AB=AB1;
r=some_func_3(&sf3s);
//
sf3s.ABC=sf1s->ABC2; sf3s.AB=AB2;
r=some_func_3(&sf3s);
//
free(AB1); // освобождение памяти
free(AB2);
return(0); // возврат значения функции
}
```

Интерфейс функции *some_func_3* в заголовочном файле *some_func_3.h*.

```
typedef struct
        {
        unsigned long int Ny;
        unsigned long int Nx;
        double* AB;
        double* C;
        double* ABC;
        } some_func_3_str;
int some_func_3(some_func_3_str*);
```

Реализация функции пользователя *some_func_3* в файле *some_func_3.c*.

```
#include "some_func_3.h"
//
int some_func_3(some_func_3_str* sf3s)
{
unsigned long int y, x, adr;
//
x=0;
while (x<sf3s->Nx)
        {
        *(sf3s->ABC+x)=0.;
        y=0;
        while (y<sf3s->Ny)
                {
                adr=y*sf3s->Nx+x;
                *(sf3s->ABC+x)=*(sf3s->ABC+x)+*(sf3s->AB+adr)*(*(sf3s->C+y));
                y++;
                }
        x++;
        }
```

```
//
return(0);
}
```

Реализация mex-функции пользователя *some_func_2mc* в файле *some_func_2mc.c*.

```
#include "mex.h"
#include "some_func_2.h"
//
void mexFunction(
        int nlhs,           // Number of left hand side (output) arguments
        mxArray *plhs[],    // Array of left hand side arguments
        int nrhs,           // Number of right hand side (input) arguments
        const mxArray *prhs[]  // Array of right hand side arguments
)
{
// Объявление входных переменных
double* Nx_;
double* ABC_;
double* D_;
//
// Объявление выходных переменных
double* ABCD_;
//
// Объявление внутренних переменных
int r;
unsigned long int Nx;
some_func_2_str sf2s;
//
// Выделение памяти для входных переменных
Nx_=mxGetPr(prhs[0]);
ABC_=mxGetPr(prhs[1]);
D_=mxGetPr(prhs[2]);
//
// Инициализация внутренних переменных
Nx=(unsigned long int)*Nx_;
//
// Выделение памяти для выходных переменных
plhs[0]=mxCreateDoubleMatrix(1, Nx, mxREAL);
ABCD_=mxGetPr(plhs[0]);
//
sf2s.Nx=Nx; sf2s.ABC=ABC_; sf2s.ABCD=ABCD_; sf2s.D=*D_;
r=some_func_2(&sf2s);
}
```

Интерфейс функции *some_func_2* в заголовочном файле *some_func_2.h*.

```
typedef struct
        {
        unsigned long int Nx;
        double* ABC;
        double D;
        double* ABCD;
        } some_func_2_str;
int some_func_2(some_func_2_str*);
```

Реализация функции пользователя *some_func_2* в файле *some_func_2.c*.

```
#include "some_func_2.h"
//
int some_func_2(some_func_2_str* sf2s)
{
unsigned long int x;
//
x=0;
while (x<sf2s->Nx)
        {
        *(sf2s->ABCD+x)=*(sf2s->ABC+x)-sf2s->D;
        x++;
        }
//
return(0);
}
```

2. Разработка пользовательских интерфейсов в среде MatLab

2.1. Объекты управляемой графики

Для решения сложных задач часто требуется разработка множества m- и mex-функций, которые вызываются в некоторой заданной последовательности наряду со стандартными функциями. Помимо этого требуется ввод исходных числовых и текстовых данных, а также визуализация промежуточных или конечных результатов в той или иной форме: часть результатов необходимо представить в текстовом виде, а часть – в графическом виде, при этом может потребоваться несколько графических окон для наглядной графической визуализации результатов.

MatLab является объектно-ориентированной системой, что позволяет просто и эффективно реализовывать графический интерфейс приложений. Для реализации графического интерфейса приложений в среде моделирования MatLab применяются графические объекты – графические окна, содержащие базисные элементы системы управления (кнопки, списки, переключатели, флаги, полосы скроллинга, области ввода, меню), а также оси и текстовые области для вывода результатов работы. Выделяют следующие основные виды объектов управляемой графики (рис. 2.1).

1. Объекты *Root* являются вершиной иерархии. Они соответствуют экрану компьютера и создаются автоматически в начале сеанса работы среды MatLab.

2. Объекты *Figure* – графические окна, за исключением командного окна.

3. Объекты *Uicontrol* – пользовательское управление интерфейсом. Они включают в себя *pushbutton* (кнопка), *radiobutton* (переключатель) и *slider* (полоса скроллинга).

4. Объекты *Axes* (оси) определяют область в графическом окне *Figure* и ориентацию дочерних объектов в этой области. Данные объекты существуют только с объектами *Figure*.

5. Объекты *Uimenu* представляют собой меню пользовательского интерфейса, которое расположено в верхней части графического окна *Figure*.

6. Объекты *Image* – это двухмерные объекты среды MatLab.

7. Объекты *Line* – основные графические базисные элементы для построения большинства двухмерных графиков.

8. Объекты *Surface* – трехмерное графическое представление данных матриц.

9. Объекты *Text* – строки символов, текстовые области для вывода результатов.

10. Объекты *Light* определяют источник света, действующий на все объекты в пределах области *Axes*.

11. Объекты *Uicontextmenu* – представляют собой контекстное (всплывающее) меню пользовательского интерфейса.

В результате графические окна, в которых реализованы все необходимые элементы управления, становятся удобным интегрирующим механизмом, упрощающим использование всего набора стандартных и разработанных функций, позволяющих решать сложные задачи и визуализировать результаты.

Создание приложений с графическим интерфейсом включает следующие этапы:

1. Расположение нужных элементов интерфейса в пределах графического окна;

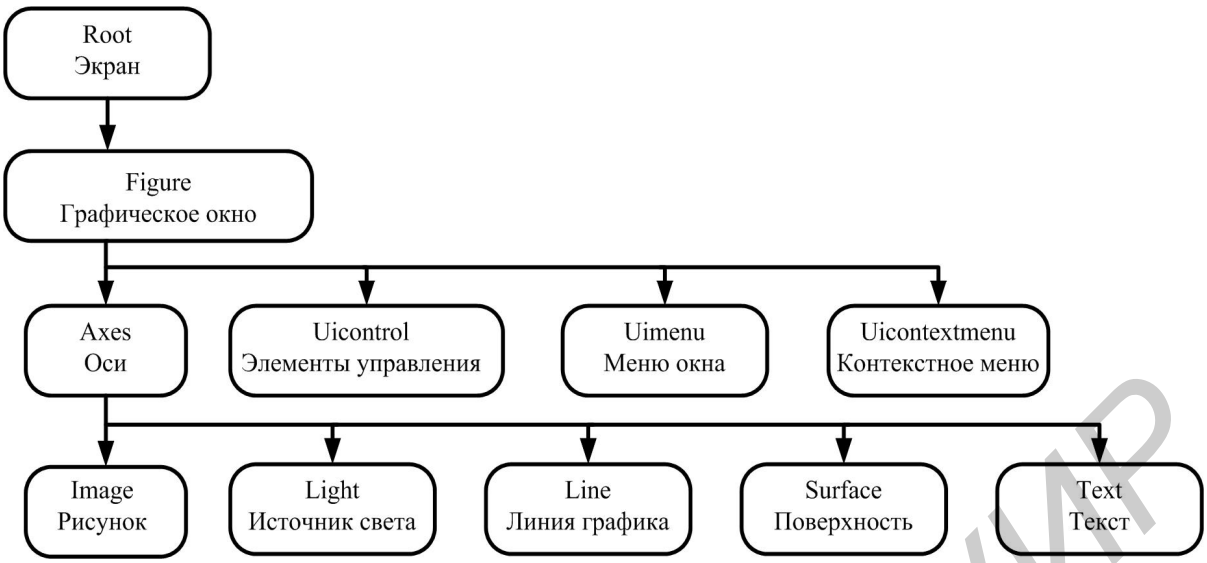

Рис. 2.1. Иерархия графических объектов

2. Определение действий (событий, команд), которые выполняются при обращении пользователя к данным объектам, например, при нажатии кнопки.

2.2. Среда GUIDE

Для создания приложений с графическим интерфейсом пользователя в состав среды моделирования MatLab входит среда GUIDE. Переход в среду GUIDE осуществляется выполнением *guide* в командной строке или через командное меню: *File->New->GUI*. При этом появляется диалоговое окно GUIDE *Quick Start* (рис. 2.2) с двумя вкладками.

1. Вкладка *Create New GUI* (создание нового приложения) с четырьмя заготовками: *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). Внизу вкладки *Create New GUI* есть флаг, установка которого позволяет сразу задать имя файла, в котором будет храниться графический интерфейс.

2. Вкладка *Open Existing GUI* (открытие существующего приложения).

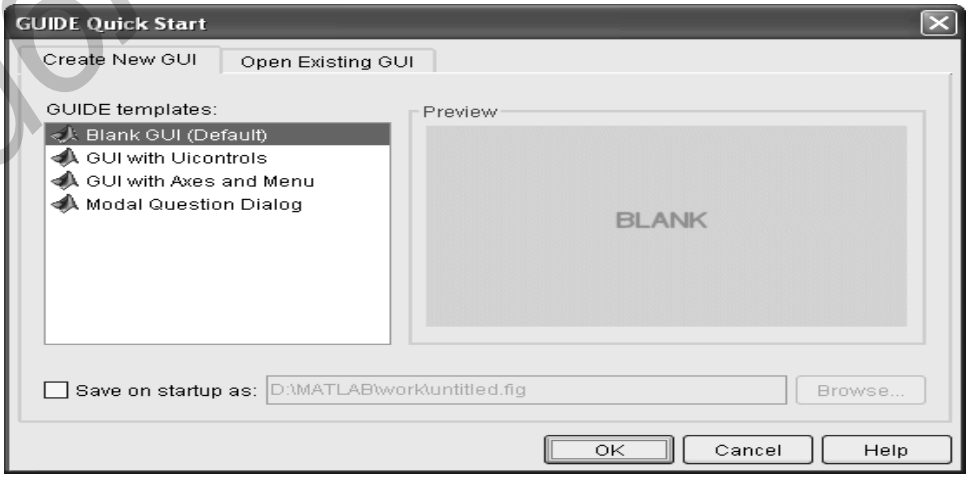

Рис. 2.2. Диалоговое окно GUIDE Quick Start

При создании нового приложения или открытии уже созданного появляется редактор окна приложения, содержащий строку меню, панель инструментов управления приложением, заготовку окна приложения, вертикальную и горизонтальную линейки, панель инструментов для добавления элементов интерфейса (рис. 2.3). Редактор приложения позволяет разместить различные элементы интерфейса (рис. 2.4). Для этого требуется нажать соответствующую кнопку на панели инструментов и поместить выбранный объект щелчком мыши в требуемое место заготовки окна приложения. Другой способ состоит в задании прямоугольной области объекта перемещением мыши по области заготовки окна с удержанием левой кнопки мыши. Размер и положение добавленных объектов изменяются при помощи мыши. Перед изменением размера следует выбрать режим выделения объектов и сделать объект текущим, щелкнув по нему клавишей мыши.

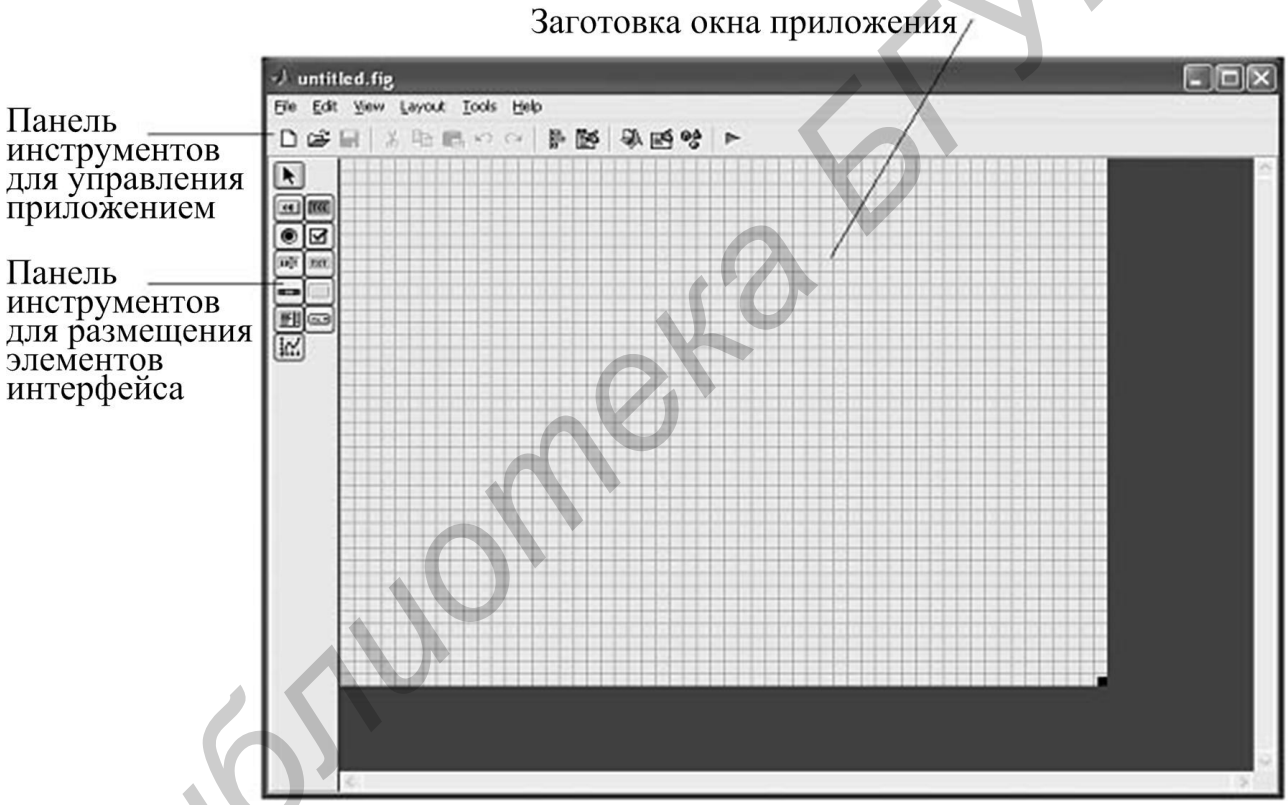

Рис. 2.3. Редактор приложения

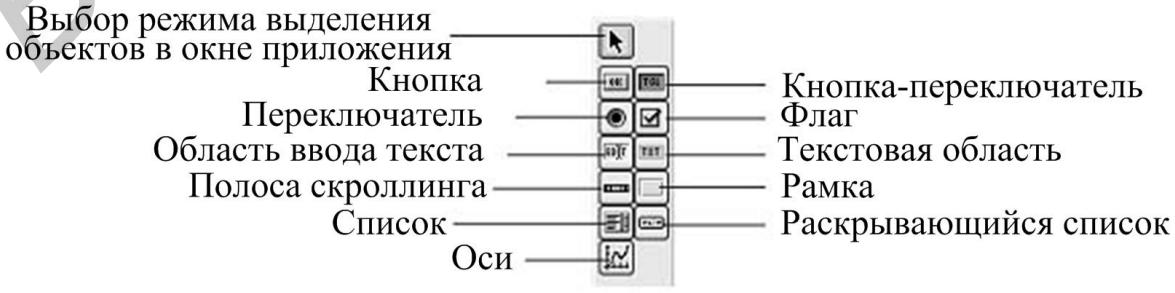

Рис. 2.4. Панель инструментов для добавления элементов интерфейса

Любой объект можно удалить с заготовки окна приложения при помощи <Delete>, предварительно выделив его. Среда GUIDE предлагает ряд средств, которые облегчают проектирование приложения:

1. Сетка с возможностью привязки объектов к ней, линейки и линии выравнивания (меню *Tools*, пункт *Grid and Rules*);

2. Инструменты выравнивания объектов (меню *Tools*, пункт *Align Objects* или кнопка *Align Objects* на панели управления среды GUIDE (рис. 2.5));

3. Редактор меню, который позволяет создавать меню приложения и контекстные меню (меню *Tools*, пункт *Menu Editor* или кнопка *Menu Editor* на панели управления среды GUIDE (рис. 2.5));

4. Браузер объектов для быстрого перехода к их свойствам (кнопка *Object Browser* на панели управления среды GUIDE (рис. 2.5));

5. Редактор порядка обхода элементов управления клавишей *Tab* (меню *Tools*, пункт *Tab Order Editor*);

6. Инспектор свойств элементов управления (кнопка *Property Inspector* на панели управления среды GUIDE (рис. 2.5)).

Рис. 2.5. Панель инструментов управления приложением

Важным средством проектирования приложения является инспектор свойств элементов управления (рис. 2.6). С помощью данного редактора для графического окна приложения задается имя заголовка приложения (пункт *Name*), задаются шрифты для вывода текстовой информации, позиция графического окна по умолчанию относительно экрана монитора, для элементов управления задается тег (пункт *Tag*), с помощью которого в дальнейшем осуществляется обращение к элементу управления, надписи на элементе управления (пункт *String*) и другие свойства. Необходимо отметить, что тег элемента управления необходимо задавать таким, чтобы он нес смысловую нагрузку о функциональном назначении элемента управления.

Редактор приложений содержит заготовку окна приложения с нанесенной сеткой и вертикальную и горизонтальную линейки (см. рис. 2.3). Линейка позволяет размещать элементы интерфейса в позиции с любыми координатами в пикселей, отсчитываемыми от левого нижнего угла заготовки окна приложения. Часто требуется, чтобы небольшое перемещение мыши вызывало изменение положения объекта на некоторый фиксированный шаг. Сетка редактора приложений позволяет осуществить такое дискретное движение. Выбор пункта

Grid and Rules меню *Layout* приводит к появлению диалогового окна, представленного на рис. 2.7.

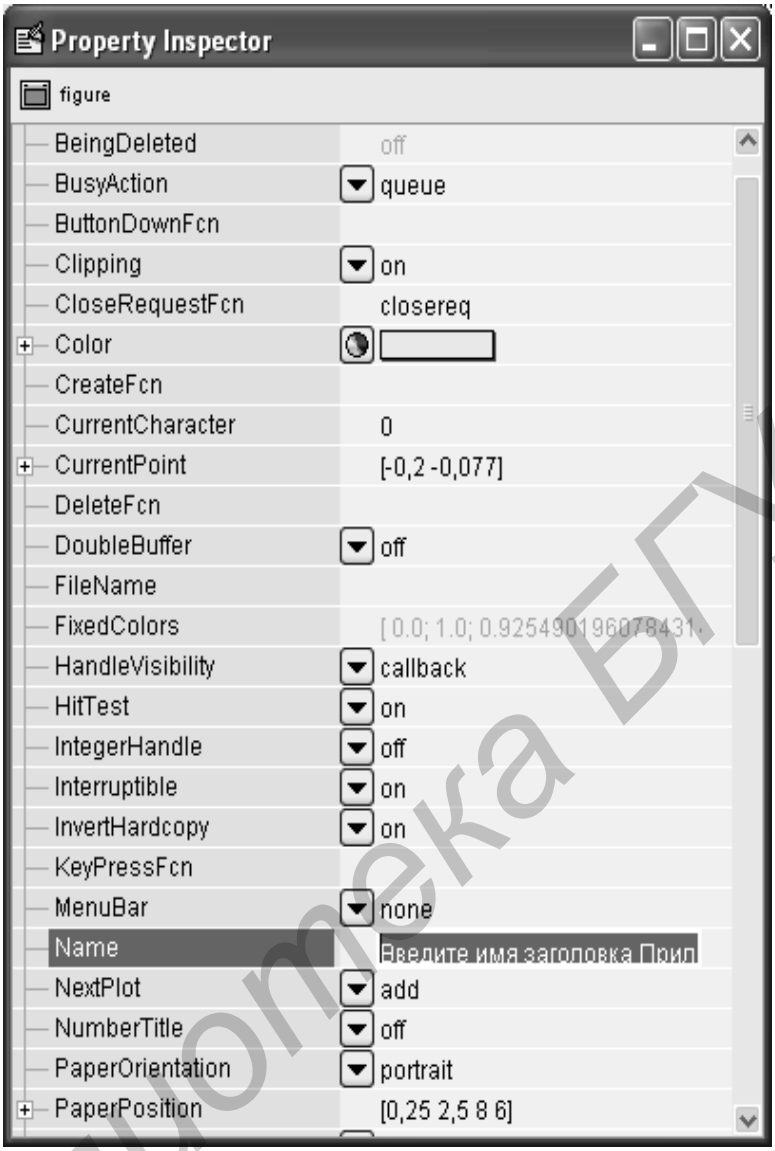

Рис. 2.6. Редактор свойств элементов управления

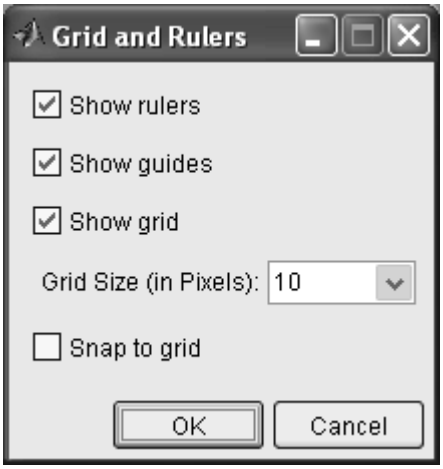

Рис. 2.7. Диалоговое окно *Grid and Rules*

Флаги *Show rules* и *Show grid* соответствуют отображению линеек и сетки в редакторе приложений, а раскрывающийся список *Grid Size* позволяет выбрать размер ячеек сетки. Минимально допустимый размер десять пикселей позволяет достаточно точно располагать элементы управления в окне приложения. Привязка перемещения к линиям сетки происходит при установленном флаге *Snap to grid*. Привязка разрешает разместить объект и изменить его размеры только при условии прохождения границы объекта по линиям сетки. Выбор мелкого шага сетки в сочетании с привязкой предоставляет разработчику возможность быстро оформить интерфейс приложения.

Размещение объектов в ряд по вертикали или горизонтали требует предварительного определения некоторой вспомогательной линии. Для этого используются горизонтальные и вертикальные линии выравнивания. Следует навести курсор мыши на соответствующую линейку и, удерживая левую кнопку мыши, вытащить на форму синюю линию (рис. 2.8). Перемещение объектов к линии выравнивания вызывает автоматическое расположение их границ на данной линии. Флаг *Show guides* окна *Grid and Rules* (см. рис. 2.7) отвечает за отображение линий выравнивания в области окна приложения.

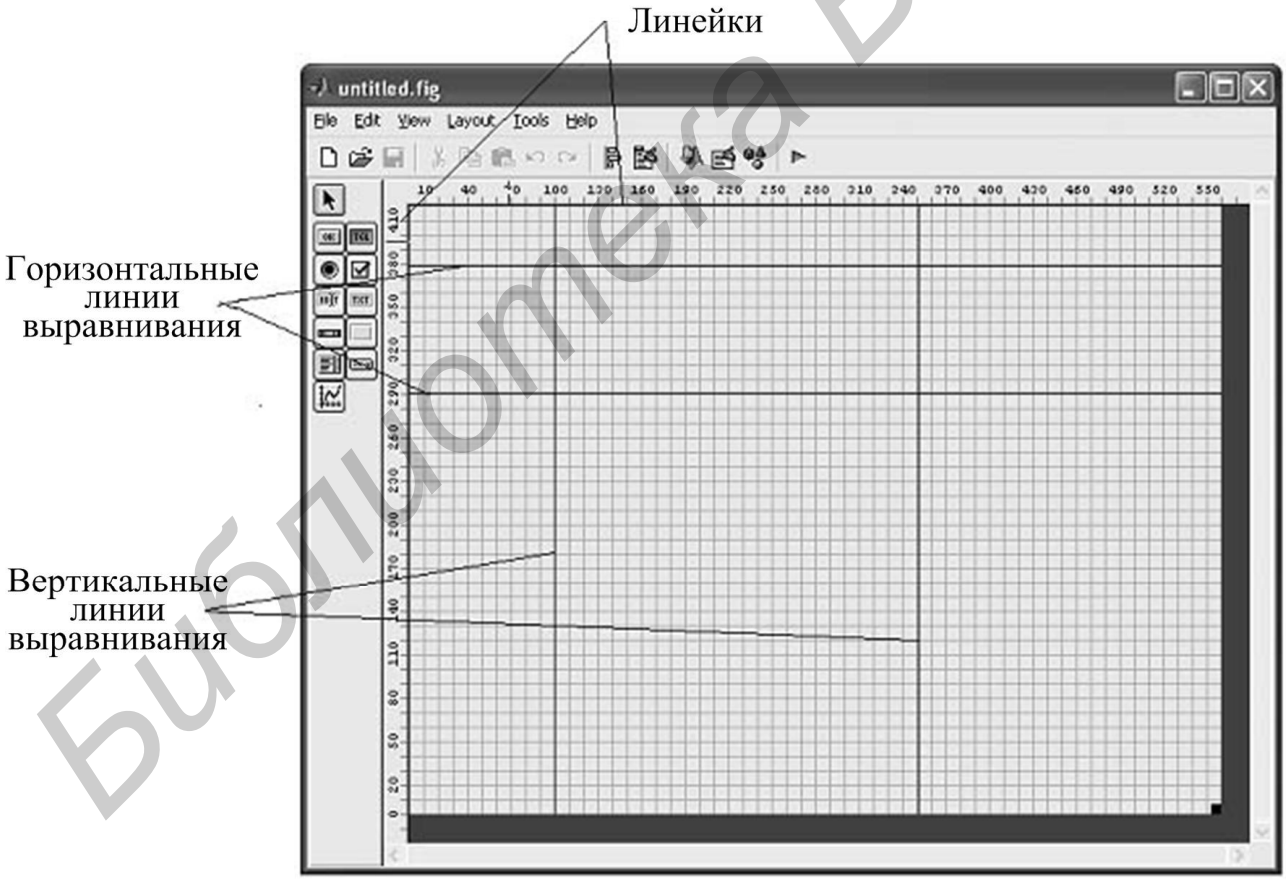

Рис. 2.8. Редактор приложения

Приложение с пользовательским интерфейсом хранится в двух файлах с расширениями .fig и .m, который формируется автоматически при сохранении приложения. Первый файл содержит информацию о размещении в графическом

окне приложения объектов управления, второй является m-файлом с основной функцией и подфункциями. Добавление элемента интерфейса из редактора приложения приводит к автоматическому созданию соответствующей подфункции обработки события. Данную подфункцию необходимо наполнить содержимым – операторами, которые выполняют обработку события, возникающего при обращении пользователя к элементу интерфейса.

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

1. «MATLAB: Creating Graphical User Interfaces»;
2. «MATLAB: Functions Categorical List: Creating Graphical User Interfaces»;
3. «MATLAB: Handle Graphics Property Browser».

Далее приведено несколько примеров, показывающих принцип создания приложения с пользовательским графическим интерфейсов в GUIDE.

Пример 2.1. Для m-проекта из одного файла (*single_.m*) с текстом программы вычисления выражения (1.1), приведенного в примере 1.1, создадим интерфейс, в котором будут присутствовать элементы ввода текстовой информации, кнопки и оси, для визуализации результата работы программы.

1. Задаем и выполняем *guide* в командной строке.
2. Создаем новый проект *Blank GUI* (пустое окно приложения, см. рис. 2.3).
3. Сохраняем проект и задаем имя проекта, например *mygui.fig*. Автоматически откроется редактор m-файлов, содержащий файл *mygui.m*
4. Задаем имя заголовка графического окна с помощью редактора свойств элементов управления (пункт *Name*), например, зададим имя «Пример 1.4». Для сохранения изменений в редакторе свойств необходимо нажать кнопку <Enter>.
5. Размещаем текстовую область и задаем надпись «Число строк матрицы».
6. Размещаем элемент ввода текста, с помощью которого будет вводится число строк в матрице. Задаем тег элемента (пункт *Tag*) *num_str* и очищаем значение пункта *String*.
7. Размещаем текстовую область и задаем надпись «Число столбцов матрицы».
8. Размещаем элемент ввода текста, с помощью которого будет вводится число столбцов в матрице. Задаем тег элемента – *num_stl* и очищаем значение пункта *String*.
9. Размещаем на графическом окне элемент управления кнопку. Задаем тег *calculation* и надпись на кнопке (пункт *String)* «Расчет». При нажатии на кнопку будут генерироваться значения матрицы, согласно выражению (1.1) примера 1.1.
10. Размещаем на графическом окне элемент управления кнопку. Задаем тег *plot* и надпись на кнопке «Построить». При нажатии на кнопку будут отображаться значения полученной матрицы в виде двухмерного графика для каждой строки матрицы.
11. Размещаем элемент управления оси и сохраняем проект.
12. Запускаем проект. На экране отображается диалоговое окно, приведенное на рис. 2.9.

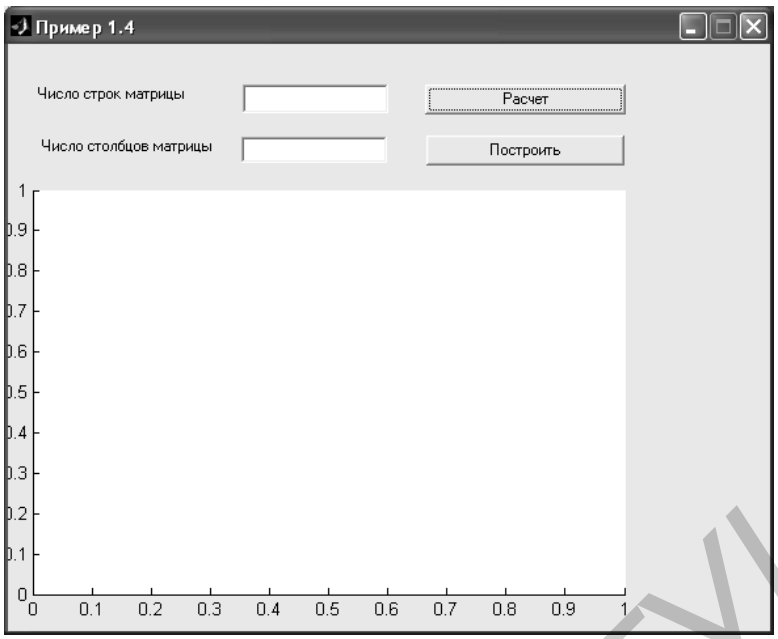

Рис. 2.9. Окно приложения

Автоматически все изменения и добавленные элементы управления будут описываться в файле *mygui.m*. Далее требуется запрограммировать события, которые будут выполняться при нажатии на кнопки, описать ввод данных для элементов ввода, для этого необходимо обратиться к файлу *mygui.m*. Данный файл имеет структуру без автоматически созданных комментариев, представленную ниже.

```
function varargout = mygui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton', gui_Singleton, ...
                   'gui_OpeningFcn', @mygui_OpeningFcn, ...
                   'gui_OutputFcn',  @mygui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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
```

gui_Name – имя М-файла с работающей в данный момент файл-функцией приложения, которое возвращается функцией mfilename.

gui_Singleton – сколько копий приложения может быть запущено одновременно: 1 – может быть запущена только одна копия приложения, 0 – может быть одновременно запущено несколько копий.

gui_OpeningFcn – указатель на подфункцию *mygui_OpeningFcn* (в файле *mygui.m*), выполняющуюся перед тем, как окно приложения появится на экране.

gui_OutputFcn – указатель на подфункцию *mygui_OutputFcn* (в файле *mygui.m*), которая определяет, что возвращает функция *mygui*, вызванная с выходным аргументом (по умолчанию, указатель на графическое окно приложения);

gui_LayoutFcn – по умолчанию пустой массив может быть указателем на функцию, которая определяет способ появления приложения.

gui_Callback – пока пустой массив, при возникновении события от некоторого элемента управления будет содержать указатель на функцию *mygui* с необходимыми входными аргументами, которые и определят исполняемую подфункцию в *mygui.m*.

Далее следуют подфункции обработки события *Callback* для элементов ввода данных *function num_str_Callback(hObject, eventdata, handles)* и *function num_stl_Callback(hObject, eventdata, handles)*, и для кнопок *function calculation_Callback(hObject,eventdata, handles)* и *function plot_Callback(hObject, eventdata, handles)*.

Имя подфункций образовано названием (тегом) элемента управления и события. Важно задавать имена объектам в свойстве *Tag* сразу после их добавления на окно приложения в редакторе приложений, иначе генерируемая подфункция получит имя, которое сохранится при последующем изменении значения *Tag* и повлечет ошибки при выполнении приложения.

Завершающий этап состоит в программировании событий, которые будут выполняться при обращении пользователем к элементам управления.

При обращение к области ввода текста *num_str и num_stl*, необходимо осуществить перевод текстовой информации в числовую. Код обработки события обращения пользователя к элементам ввода представлен ниже.

```
function num_str_Callback(hObject, eventdata, handles)
global Ny;
Ny = str2num(get(handles.num_str,'String'));
function num_stl_Callback(hObject, eventdata, handles)
global Nx;
Nx = str2num(get(handles.num_stl,'String'));
```

Переменные *global Ny* и *global Nx* объявлены как глобальные, поэтому будут доступны в любой подфункции программы, и хранят информацию о количестве строк и столбцов в генерируемой матрице. Функция *str2num* – является стандартной функцией MatLab и осуществляет перевод строкового значения в числовое. Функция *get(handles.num_str,'String')* выполняет обращение к элементу управления с тегом *num_str* и считывание текстовой информации, введенной пользователем. Ниже приведена подфункция обработки события, возникающего при нажатии пользователем на кнопку «Расчет».

```
function calculation_Callback(hObject, eventdata, handles)
global Ny;global Nx; global E;

A=randn([Ny Nx]); % создание массива A случайных значений
B=randn([Ny Nx]);
C=randn([1 Ny]); % создание строки C случайных значений
D=randn([1 1]); % создание случайного значения D
E=zeros(2,Nx); % создание массива E результатов
E(1,:)=C*(A+B)-D;
E(2,:)=C*(A-B)-D % Элементы матрицы отображаются в командном окне
```

При нажатии пользователем кнопки «Построить» возникает событие, связанное с построением двумерного графика для каждой строки сгенерированной матрицы. На рис. 2.10 представлен результат работы программы.

```
function plot_Callback(hObject, eventdata, handles)
global Nx;global E;
t=[1:Nx];
axes(handles.axes1);
plot(t,E); grid on;
```

Стандартная функция *plot* предназначена для визуализации функций одной переменной, векторных и матричных данных, *plot(t,E)* – строит график зависимости элементов вектора *E* от элементов вектора *t*.

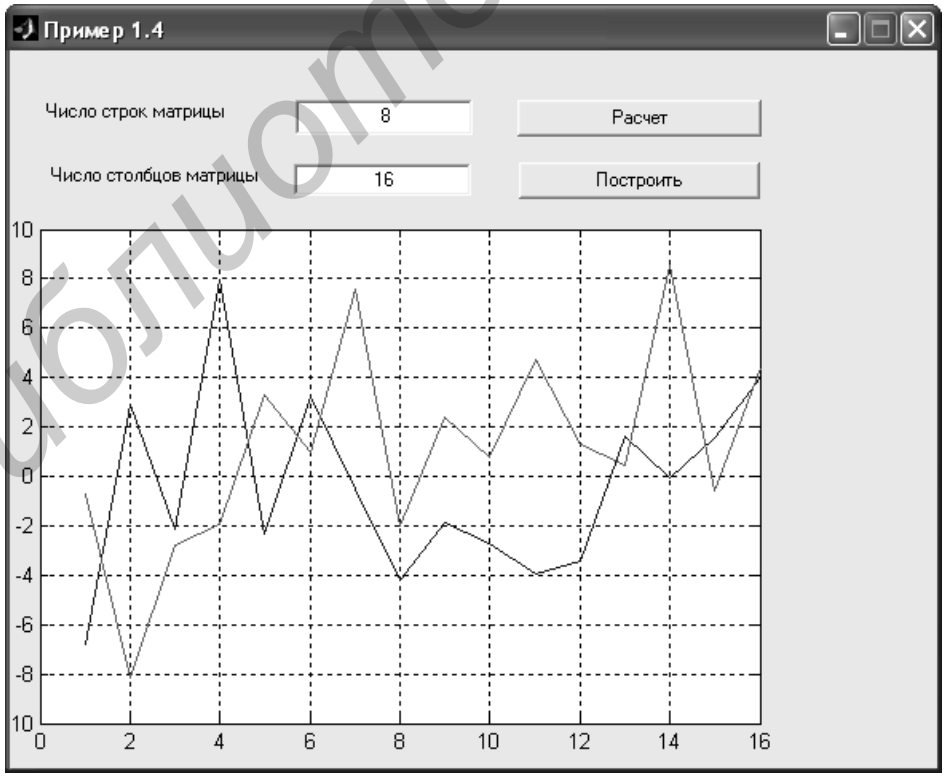

Рис. 2.10. Результат программирования событий

Пример 2.2. Пример основан на использовании уже созданного интерфейса в примере 2.1. В данном примере показано, как устанавливать для элементов ввода текстовой информации значения по умолчанию, программно изменять свойства элементов управления, создавать разные виды меню и управлять графикой.

Установка значений переменных по умолчанию, а также свойств элементов управления осуществляется в подфункции *function mygui_OpeningFcn(hObject, eventdata, handles, varargin)*. Установим по умолчанию количество строк матрицы равным 8, а число столбцов равным 16.

```
function mygui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
%устанавливаем по умолчанию количество строк
set(handles.num_str,'String',num2str(8));
%устанавливаем по умолчанию количество столбцов
set(handles.num_stl,'String',num2str(16));
```

Часто для корректной работы приложения, ограничения действий пользователя требуется программно изменять свойства элементов управления, например, по какому-то условию делать их активными или неактивными, делать видимыми или нет и т. д. Большинство свойств объектов можно устанавливать программно прямо в ходе работы приложения для обеспечения согласованного поведения элементов управления. Пусть при запуске программы доступными являются только элементы ввода текстовой информации и кнопка «Расчет». При нажатии кнопки «Расчет» становится доступной кнопка «Построить», а кнопка «Расчет» становится неактивной.

Решение поставленной задачи требует привлечения свойства *Enable* объекта управления. Свойство *Enable* объекта отвечает за возможность доступа к нему пользователем, значение *on* разрешает доступ, а *off* соответственно запрещает. Установка значений свойствам объектов в программе производится при помощи функции *set*. Функция *set* вызывается с тремя входными аргументами – указателем на объект, названием свойства и его значением, последние два аргумента заключаются в апострофы. В данном примере свойства одного объекта должны изменяться в блоке операторов обработки события *Callback* другого объекта. Следовательно, должна иметься возможность доступа к указателю на любой существующий объект. Аргументы *h* и *handles* подфункций, которые обрабатывают события элементов управления, содержат требуемые указатели. В аргументе *h* хранится указатель на тот объект, событие которого обрабатывается в данный момент, а *handles* является структурой указателей. Поля структуры совпадают со значениями свойств *Tag* существующих элементов интерфейса. Например, *handles.num_str* является указателем на элемент ввода количества строк матрицы.

Доступ к кнопке «Построить» должен быть запрещен в начале работы приложения, пока пользователь не нажмет кнопку «Расчет». Установить свойство *Enable* в состояние *off* для кнопки «Построить» можно в редакторе свойств или же с помощью функции *set* в подфункции *function mygui_OpeningFcn(hObject, eventdata, handles, varargin)*. Для разрешения и запрещения доступа к кнопкам нужно ввести дополнения в обработку их событий *Callback*. В подфункцию обработки события *Callback* кнопки «Расчет» необходимо добавить следующие изменения:

1. Установить свойства *Enable* кнопки «Построить» в значение *on*;
2. Установить свойства *Enable* кнопки «Расчет» в значение *off*.

Аналогичные изменения произвести в обработке события *Callback* кнопки «Построить», а именно:

1. Установить свойства *Enable* кнопки «Расчет» в значение *on*;
2. Установить свойства *Enable* кнопки «Построить» в значение *off*.

Ниже приведены изменения подфункции обработки соответствующих событий.

```
function mygui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
% устанавливаем по умолчанию количество строк
set(handles.num_str,'String',num2str(8));
% устанавливаем по умолчанию количество столбцов
set(handles.num_stl,'String',num2str(16));
% устанавливаем свойства Enable кнопки «Построить» в значение off.
set(handles.plot,'Enable','off');
```

```
function calculation_Callback(hObject, eventdata, handles)
global Ny;global Nx; global E;
A=randn([Ny Nx]);
B=randn([Ny Nx]);
C=randn([1 Ny]);
D=randn([1 1]);
E=zeros(2,Nx);
E(1,:)=C*(A+B)-D;
E(2,:)=C*(A-B)-D
% устанавливаем свойства Enable кнопки «Расчет» в значение off.
set(handles.calculation,'Enable','off');
% устанавливаем свойства Enable кнопки «Построить» в значение on.
set(handles.plot,'Enable','on');
```

```
function plot_Callback(hObject, eventdata, handles)
global Nx;global E;
t=[1:Nx];
axes(handles.axes1);
plot(t,E); grid on;
```

```
% устанавливаем свойства Enable кнопки «Построить» в значение off.
set(handles.plot,'Enable','off');
% устанавливаем свойства Enable кнопки «Расчет» в значение on
set(handles.calculation,'Enable','on');
```

Аналогичным образом можно сделать, чтобы при запуске программы, например, графические оси в интерфейсе приложения были невидны, и только при нажатии кнопки «Построить» отображались в интерфейсе. Решение поставленной задачи требует привлечения свойства *Visible* объекта управления. Свойство *Visible* объекта отвечает за отображение (видимость) в интерфейсе, значение *on* разрешает видимость, а *off* соответственно запрещает.

Ниже приведены изменения подфункций обработки соответствующих событий *function mygui_OpeningFcn(hObject, eventdata, handles, varargin)* и *function plot_Callback(hObject, eventdata, handles)*

```
function mygui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
% устанавливаем по умолчанию количество строк
set(handles.num_str,'String',num2str(8));
% устанавливаем по умолчанию количество столбцов
set(handles.num_stl,'String',num2str(16));
% устанавливаем свойства Enable кнопки «Построить» в значение off.
set(handles.plot,'Enable','off');
% устанавливаем свойства Visible осей в значение off. Оси невидимы
set(handles.plot,'Enable','off');

function plot_Callback(hObject, eventdata, handles)
global Nx;global E;
t=[1:Nx];
% устанавливаем свойства Visible осей в значение on. Оси видимы
set(handles.axes1,'Visible','on');
axes(handles.axes1);
plot(t,E); grid on;
set(handles.plot,'Enable','off');
set(handles.calculation,'Enable','on');
```

Рассмотрим процесс создания меню графического окна приложения. Приложение MatLab может использовать стандартное меню графического окна. Среда GUIDE позволяет дополнять стандартное меню или создавать собственное меню. Свойство *MenuBar* окна приложения (объекта *figure*) отвечает за наличие стандартных меню File, Edit, Tools, Window и Help в работающем приложении. Значение *figure* данного свойства соответствует отображению стандартных меню, а *none* приводит к приложению без строки со стандартным меню. В независимости от значения свойства *MenuBar* имеется возможность создавать и

размещать собственное меню. Размещение и программирование меню производится при помощи редактора меню.

Запуск редактора меню осуществляется из панели инструментов управления приложением (см. рис. 2.5). При запуске редактора меню из панели управления появляется окно *Menu Editor*, представленное на рис. 2.11.

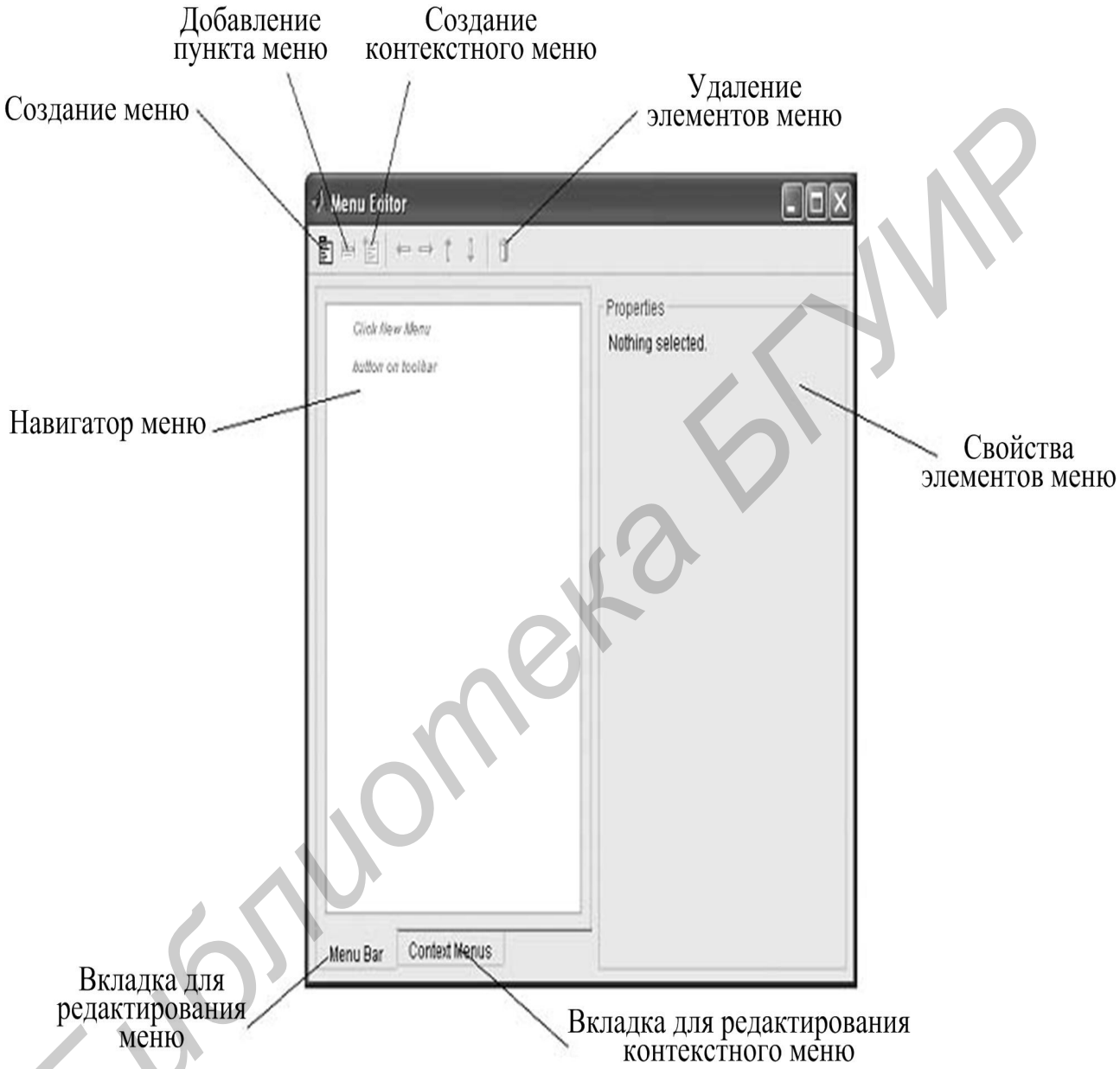

Рис. 2.11. Редактор меню

Окно редактора меню содержит две вкладки: *Menu Bar*, предназначенную для создания строки меню приложения, и *Context Menus* для создания контекстного меню. Для создания меню необходимо нажать соответствующую кнопку на панели инструментов редактора меню (см. рис. 2.11), в навигаторе появится строка *Untitled 1* (рис. 2.12). Строка *Label* служит для задания надписи меню или пункта меню, а *Tag* – для определения названия созданного объекта.

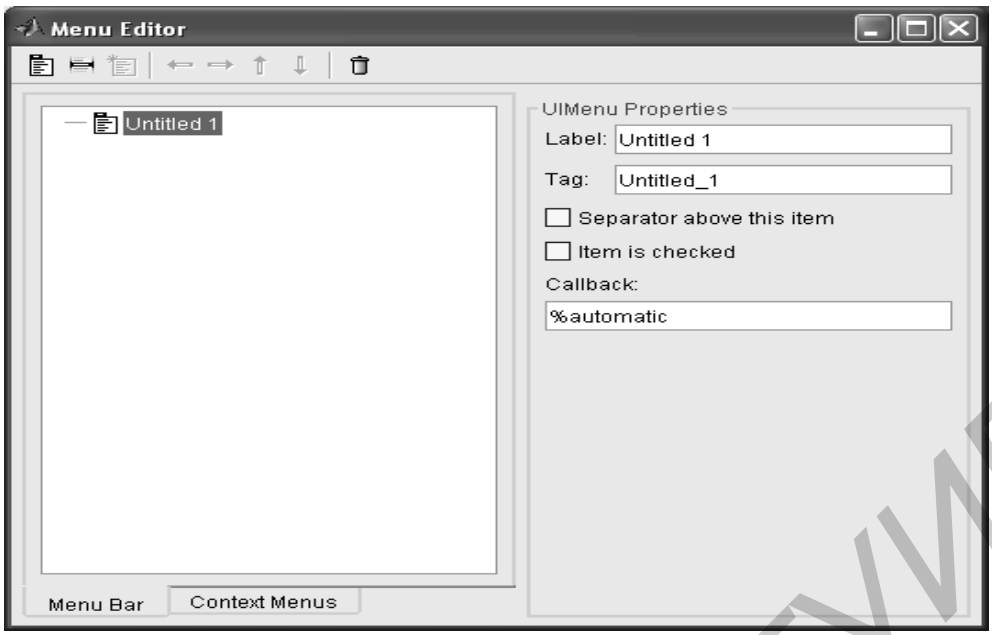

Рис. 2.12. Задание свойств меню в редакторе

Создадим для разработанного приложения меню «График», состоящее из двух пунктов: «Построить» и «Очистить». Навигатор меню должен содержать структуру, изображенную на рис. 2.13. Меню «График» имеет первый уровень, а пункты «Построить», «Очистить» – второй. После создания меню в редакторе необходимо запрограммировать события *Callback* пунктов меню. Событие *Callback* самого меню «График» не требует обработки, так как происходит автоматическое раскрытие меню. Выбор пунктов «Построить» и «Очистить» должен приводить, соответственно, к отображению результатов расчета и очистке осей.

Рис. 2.13. Иерархия элементов меню

Ниже приведены изменения подфункций обработки соответствующих событий, программирование событий для созданных пунктов меню и результат работы программы (рис. 2.14).

```
function varargout = mygui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @mygui_OpeningFcn, ...
                   'gui_OutputFcn',  @mygui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
```

```
                'gui_Callback',   []);
if nargin & isstr(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

function mygui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
set(handles.num_str,'String',num2str(8));
set(handles.num_stl,'String',num2str(16));
set(handles.plot,'Enable','off');
set(handles.axes1,'Visible','off');
% устанавливаем свойства Enable пункта меню «Очистить» в значение off.
set(handles.mnGraphClear,'Enable','off');
% устанавливаем свойства Enable пункта меню «Построить» в значение off.
set(handles.mnGraphPlot,'Enable','off');
% --- Executes on button press in calculation.

function calculation_Callback(hObject, eventdata, handles)
global Ny;global Nx; global E;
A=randn([Ny Nx]);
B=randn([Ny Nx]);
C=randn([1 Ny]);
D=randn([1 1]);
E=zeros(2,Nx);
E(1,:)=C*(A+B)-D;
E(2,:)=C*(A-B)-D
set(handles.calculation,'Enable','off');
set(handles.plot,'Enable','on');
set(handles.mnGraphPlot,'Enable','on');

% --- Executes on button press in plot.
function plot_Callback(hObject, eventdata, handles)
global Nx;global E;
t=[1:Nx];
set(handles.axes1,'Visible','on');
axes(handles.axes1);
plot(t,E); grid on;
set(handles.plot,'Enable','off');
set(handles.calculation,'Enable','on');
```

```
set(handles.mnGraphClear,'Enable','on');
set(handles.mnGraphPlot,'Enable','off');
% --------------------------------------------------------------------
function mnGraph_Callback(hObject, eventdata, handles)
% --------------------------------------------------------------------
function mnGraphClear_Callback(hObject, eventdata, handles)
cla;
set(handles.mnGraphClear,'Enable','off');
% --------------------------------------------------------------------
function mnGraphPlot_Callback(hObject, eventdata, handles)
global Nx;global E;
t=[1:Nx];
set(handles.axes1,'Visible','on');
axes(handles.axes1);
plot(t,E); grid on;
set(handles.plot,'Enable','off');
set(handles.calculation,'Enable','on');
set(handles.mnGraphClear,'Enable','on');
set(handles.mnGraphPlot,'Enable','off');
```

Рис. 2.14. Результат программирования событий

Пример 2.3. Для m-проекта, реализующего вычисление выражения (1.1) и состоящего из четырех файлов (*main.m, some_func_1.m*, *some_func_2.m*, *some_func_3.m*, интегрированных по схеме рис. 1.1) с текстами основной программы и функций пользователя, создадим интерфейс, представленный на рис. 2.15.

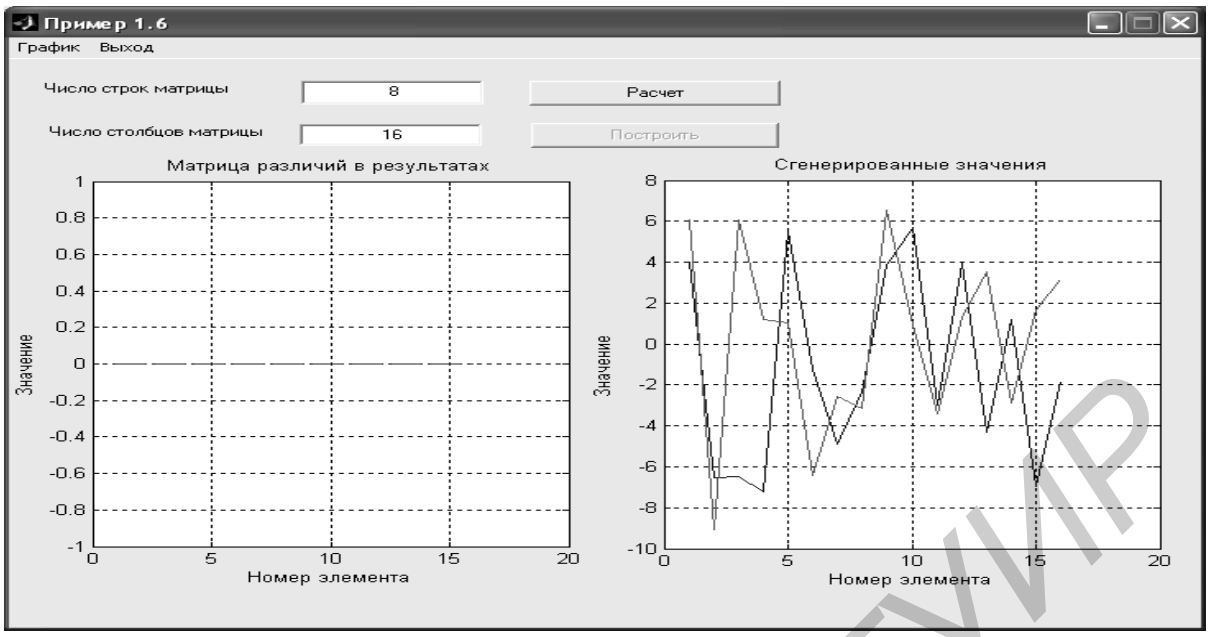

Рис. 2.15. Интерфейс приложения

Ниже приведены подфункции обработки событий, а также программирование событий для созданных пунктов меню. В основу положен интерфейс примера 2.1. Дополнительно введено меню выхода из приложения с использованием диалогового окна подтверждения выхода из приложения построения графика матрицы различий.

```
function varargout = mygui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @mygui_OpeningFcn, ...
                   'gui_OutputFcn',  @mygui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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

function mygui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
% устанавливаем по умолчанию количество строк
set(handles.num_str,'String',num2str(8));
%устанавливаем по умолчанию количество столбцов
```

```
set(handles.num_stl,'String',num2str(16));
% устанавливаем свойства Enable кнопки «Построить» в значение off.
set(handles.plot,'Enable','off');
% устанавливаем свойства Visible осей в значение off. Оси невидимы
set(handles.plot,'Enable','off');
% устанавливаем свойства Visible осей в значение off. Оси невидимы
set(handles.axes2,'Visible','off');
% устанавливаем свойства Enable пункта меню «Очистить» в значение off.
set(handles.mnGraphClear,'Enable','off');
% устанавливаем свойства Enable пункта меню «Построить» в значение off.
set(handles.mnGraphPlot,'Enable','off');

function num_str_Callback(hObject, eventdata, handles)
global Ny;
Ny = str2num(get(handles.num_str,'String'));

function num_stl_Callback(hObject, eventdata, handles)
global Nx;
Nx = str2num(get(handles.num_stl,'String'));

function calculation_Callback(hObject, eventdata, handles)
global Ny;global Nx; global R;global E;
A=randn([Ny Nx]);
B=randn([Ny Nx]);
C=randn([1 Ny]);
D=randn([1 1]);
E=zeros(2,Nx);
 [F, G]=some_func_1(A, B, C);
E(1,:)=some_func_2(F, D);
E(2,:)=some_func_2(G, D)
T(1,:)=C*(A+B)-D;
T(2,:)=C*(A-B)-D;
R=abs(E-T) % Матрица различий в результатах
set(handles.calculation,'Enable','off');
set(handles.plot,'Enable','on');
% устанавливаем свойства Enable пункта меню «Построить» в значение on
set(handles.mnGraphPlot,'Enable','on');

function plot_Callback(hObject, eventdata, handles)
global Nx;global R;global E;
t=[1:Nx];
set(handles.axes1,'Visible','on');
axes(handles.axes1);
plot(t,R,'r--'); grid on;
xlabel('Номер элемента');
ylabel('Значение');
```

```
    title('Матрица различий в результатах');
    set(handles.axes2,'Visible','on');
    axes(handles.axes2);
    plot(t,E); grid on;
    xlabel('Номер элемента');
    ylabel('Значение');
    title('Сгенерированные значения');
    set(handles.plot,'Enable','off');
    set(handles.calculation,'Enable','on');
    set(handles.mnGraphClear,'Enable','on');
    set(handles.mnGraphPlot,'Enable','off');

    function mnGraphClear_Callback(hObject, eventdata, handles)
    axes(handles.axes1);
    cla;
    axes(handles.axes2);
    cla;
    set(handles.mnGraphClear,'Enable','off');

    function mnGraphPlot_Callback(hObject, eventdata, handles)
    global Nx;global R;global E;
    t=[1:Nx];
    set(handles.axes1,'Visible','on');
    axes(handles.axes1);
    plot(t,R,'r--'); grid on;
    xlabel('Номер элемента');
    ylabel('Значение');
    title('Матрица различий в результатах');
    set(handles.axes2,'Visible','on');
    axes(handles.axes2);
    plot(t,E); grid on;
    xlabel('Номер элемента');
    ylabel('Значение');
    title('Сгенерированные значения');
    set(handles.plot,'Enable','off');
    set(handles.calculation,'Enable','on');
    set(handles.mnGraphClear,'Enable','on');
    set(handles.mnGraphPlot,'Enable','off');

    function mnExit_Callback(hObject, eventdata, handles)
    button = questdlg('Закрыть приложение?','Выход');
    if strcmp(button, 'Yes')
     delete(gcf)
    else return
end
```

3. Доступ к данным в среде MatLab

3.1. Доступ к аудиоданным в среде MatLab

Доступ к файлам аудиоданных в среде MatLab может осуществляться с помощью функций *wavread()* (считывание дискретов аудиоданных из файла) и *wavwrite()* (запись дискретов аудиоданных в файл). Для работы с файлами аудиоданных может также использоваться ряд других встроенных в MatLab функций (*auread()*, *auwrite()*, *wavplay()*, *wavrecord()*). Подробную информацию об этих и других функциях MatLab и особенностях их использования можно получить через встроенную в MatLab справочную систему «Help Navigator», загружаемую при выборе опции «Full Product Family Help» в меню «Help» главного окна MatLab.

Пример 3.1. М-программа (для пакета MatLab 6.5) считывания дискретов аудиоданных из wav-аудиофайла *fn_in* в массив *sound* и сохранения части из них (в количестве *Samples*, начиная с дискрета с номером *Start_Sample*) в wav-аудиофайл *fn_out* текущего каталога. Программа осуществляет также вывод на экран временной реализации записываемого аудиофрагмента с помощью функции *plot()*.

```
fn_in='jazz.wav'; % задание имени исходного аудиофайла
fn_out='jazz_.wav'; % задание имени результирующего аудиофайла
Start_Sample=1000;
Samples=3000;
Fs_e=44100; % задание частоты дискретизации (Гц) для результирующего
% аудиофайла
bits_e=16; % задание числа бит, представляющих дискреты результирующего
% аудиофайла
%
[sound,Fs,bits]=wavread(fn_in); % считывание из файла fn_in значений дискретов
% аудиоданных в массив sound, значение частоты дискретизации в переменную
% Fs, число бит на дискрет в переменную bits
%
wavwrite(sound(Start_Sample:Start_Sample+Samples,1),Fs_e,bits_e,fn_out);%запись
% в файл fn значений дискретов аудиоданных с разрядностью bits_e из массива
% sound с частотой дискретизации Fs_e
%
plot(sound(Start_Sample:Start_Sample+Samples,1));
```

3.2. Доступ к неподвижным изображениям в среде MatLab

Доступ к файлам неподвижных изображений в среде MatLab может осуществляться с помощью функций *imread()* (считывание пикселей изображения из файла) и *imwrite()* (запись пикселей изображения в файл). Эти функции поддерживают большинство наиболее распространенных форматов хранения и представления неподвижных изображений (bmp, hdf, jpg, jpeg, pbm, pcx, pgm,

png, pnm, ppm, tif, tiff, ras, xwd). Для работы с файлами изображений может также использоваться ряд других встроенных в MatLab функций (*imformats(), imfinfo()*). Подробную информацию об этих и других функциях MatLab и особенностях их использования можно получить через встроенную в MatLab справочную систему «Help Navigator», загружаемую при выборе опции «Full Product Family Help» в меню «Help» главного окна MatLab.

Пример 3.2. М-программа (для пакета MatLab 6.5) считывания пикселей изображения из bmp-файла *'lena512.bmp'* в двухмерный массив *Image_2D* и сохранения прямоугольной области пикселей двухмерного массива *Image_2D* в bmp-файл *'Bloc.bmp'* текущего каталога. Размер прямоугольной области пикселей по оси Y и оси X определяют значения переменных *Bloc_Size_Y* и *Bloc_Size_X* соответственно. Смещение прямоугольной области пикселей по оси Y и оси X определяют значения переменных *Start_Y* и *Start_X* соответственно. Обработка координат пикселей в двухмерном массиве осуществляется с использованием оператора цикла *while () / end* и оператора условия *if () / else / end*. Программа осуществляет также вывод на экран считанных из файла пикселей изображения и выделенной прямоугольной области пикселей с помощью функции *imshow()*, построение гистограмм исходного и полученного изображений с помощью функции *imhist()*, а также запись гистограмм в jpg-файлы с помощью функции *saveas()*. Функции *imshow()* и *imhist()* используются в сочетании с оператором *figure*, который обеспечивает вывод на экран монитора каждого изображения в отдельном окне.

```
Image_2D=imread('lena512.bmp');
Bloc_Size_Y=270;
Bloc_Size_X=200;
Start_Y=150;
Start_X=190;
%
close all; % оператор закрывает все открытые ранее окна при вызове
% функций imshow() и imhist()
Image_2D_=double(Image_2D); % копирование информации с преобразованием
%  типа данных
[siz_y, siz_x]=size(Image_2D_); % определение размера массива
Bloc_=zeros(Bloc_Size_Y, Bloc_Size_X); % выделение памяти
% под результирующий массив, предназначенный для хранения выделяемой
% из исходного изображения прямоугольной области
y=1;
while (y<=siz_y)&&(y<=Bloc_Size_Y+Start_Y)
   x=1;
   while (x<=siz_x)&&(x<=Bloc_Size_X+Start_X)
      if y>=Start_Y && x>=Start_X
         Bloc_(y-Start_Y+1, x-Start_X+1)=Image_2D_(y, x);
      end
```

```
      x=x+1;
    end
    y=y+1;
end
%
figure, imshow(Image_2D,256);
figure, imhist(Image_2D,256); saveas(gcf,'lena512_Histogram.jpg','jpg')
Bloc=uint8(Bloc_); imwrite(Bloc,'Bloc.bmp','bmp');
figure, imshow(Bloc,256);
figure, imhist(Bloc,256); saveas(gcf,'Bloc_Histogram.jpg','jpg')
```

3.3. Доступ к видеоданным в среде MatLab

Доступ к файлам видеоданных в среде MatLab может осуществляться с помощью функций *avifile()* (создание AVI-файла) и *addframe()* (запись видеофрейма в AVI-файл). Для работы с видеофайлами может также использоваться ряд других встроенных в MatLab функций (*close(),movie2avi()*). Подробную информацию об этих и других функциях MatLab и особенностях их использования можно получить через встроенную в MatLab справочную систему «Help Navigator», загружаемую при выборе опции «Full Product Family Help» в меню «Help» главного окна MatLab.

3.4. Доступ к бинарным данным в среде MatLab

Доступ к файлам данных в среде MatLab может осуществляться с помощью функций *fopen()* (открытие файла) и *fwrite()* (запись данных в файл), *fread()* (считывание данных из файла), *fclose()* (закрытие файла). Функции *fwrite()* и *fread()* поддерживают типы данных, представленные в табл. 3.1.

Таблица 3.1

Типы данных

MatLab	Язык программирования C	Интерпретация
schar	signed char	Signed character; 8 bits
uchar	unsigned char	Unsigned character; 8 bits
int8	integer*1	Integer; 8 bits
int16	integer*2	Integer; 16 bits
int32	integer*4	Integer; 32 bits
int64	integer*8	Integer; 64 bits
uint8	integer*1	Unsigned integer; 8 bits
uint16	integer*2	Unsigned integer; 16 bits
uint32	integer*4	Unsigned integer; 32 bits
uint64	integer*8	Unsigned integer; 64 bits
float32	real*4	Floating-point; 32 bits
float64	real*8	Floating-point; 64 bits
double	real*8	Floating-point; 64 bits

Для работы с файлами данных могут также использоваться функции count = *fprintf(), fscanf(), ftell()* и *ferror()*, встроенные в MatLab. Подробную информацию об этих и других функциях MatLab и особенностях их использования можно получить через встроенную в MatLab справочную систему «Help Navigator», загружаемую при выборе опции «Full Product Family Help» в меню «Help» главного окна MatLab.

Пример 3.3. М-программа (для пакета MatLab 6.5) записи массива M случайных данных (N случайных значений в формате double) в бинарный файл *'test.dat'* текущего каталога. Записанные в файл данные затем считываются со значения *Start_Sample* в количестве *Samples* и помещаются в массив M_. Затем считанные данные подвергаются статистической обработке с помощью функции *hist()*, которая формирует и выводит на экран монитора гистограмму значений в массиве M_.

```
fn='test.dat';
N=10000;
Start_Sample=1000;
Samples=3000;
%
close all;
M=randn([1 N]);
fid_out = fopen(fn,'wb'); % функция возвращает указатель на открытый файл,
% значение которого записывается в переменную fid_out
fwrite(fid_out, M,'double');
fclose(fid_out);
%
fid_in = fopen(fn,'rb');
fseek(fid_in, Start_Sample, 'bof'); % позиционирование указателя данных в файле
[M_, count] = fread(fid_in, Samples, 'double');
fclose(fid_in);
%
x = -3.9:0.05:3.9; % переменная ч устанавливает диапазон изменения значений
% в массиве M_, которые оцениваются функцией hist()
figure, hist(M_, x);
```

4. Дискретные информационные преобразования медиаданных в среде MatLab

4.1. Дискретное преобразование Фурье и спектры

Одномерное прямое и обратное преобразования Фурье вычисляются в соответствие с выражениями

$$X(k)=\sum_{j=1}^{N} x(j)\cdot \omega_N^{(j-1)\cdot(k-1)},\ k=\overline{1,N}, \tag{4.1}$$

$$x(j)=\frac{1}{N}\cdot\sum_{k=1}^{N} X(k)\cdot \omega_N^{-(j-1)\cdot(k-1)},\ j=\overline{1,N}, \tag{4.2}$$

где $\omega_N = e^{\frac{-2\cdot\pi\cdot i}{N}}$, N – число дискретов сигнала.

Для реализации одномерного преобразования Фурье в пакете MatLab предусмотрены функции *fft()* (прямое преобразование) и *ifft()* (обратное преобразование).

Пример 4.1. М-программа (из встроенной справочной системы «Help Navigator» пакета MatLab 6.5) построения спектра одномерного сигнала *y*, представляющего собой линейную комбинацию двух синусоид и шума в масштабе времени, определяемом вектором *t*. Программа открывает два окна, в которых представляет временную реализацию сигнала и его спектр мощности.

```
close all
t = 0:0.001:0.6;
x = sin(2*pi*50*t)+sin(2*pi*120*t);
y = x + 2*randn(size(t));
figure, plot(1000*t(1:50),y(1:50))
title('Signal Corrupted with Zero-Mean Random Noise') % формирование
% названия графика
xlabel('time (milliseconds)') % формирование подписи к оси X
%
Y = fft(y,512);
Pyy = Y.* conj(Y) / 512;
%
f = 1000*(0:256)/512;
figure, plot(f,Pyy(1:257))
title('Frequency content of y')
xlabel('frequency (Hz)')
```

Двухмерное прямое и обратное преобразования Фурье вычисляются в соответствие с выражениями

$$X(p,q)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x(m,n)\cdot e^{-i\cdot(2\cdot\pi/M)\cdot p\cdot m}\cdot e^{-i\cdot(2\cdot\pi/N)\cdot q\cdot n},\ p=\overline{0,M-1},\ q=\overline{0,N-1}, \tag{4.3}$$

$$x(m,n)=\frac{1}{M\cdot N}\cdot\sum_{p=0}^{M-1}\sum_{q=0}^{N-1} X(p,q)\cdot e^{i\cdot(2\cdot\pi/M)\cdot p\cdot m}\cdot e^{i\cdot(2\cdot\pi/N)\cdot q\cdot n},\ m=\overline{0,M-1},\ n=\overline{0,N-1}. \tag{4.4}$$

Для реализации двухмерного преобразования Фурье в пакете MatLab предусмотрены функции *fft2()* (прямое преобразование) и *ifft2()* (обратное преобразование). Для центрирования спектров, вычисляемых на основе преобразования Фурье, используется функция *fftshift()*. Подробную информацию об этих и других функциях MatLab и особенностях их использования можно получить через встроенную в MatLab справочную систему «Help Navigator», загружаемую при выборе опции «Full Product Family Help» в меню «Help» главного окна MatLab.

Пример 4.2. М-программа (из встроенной справочной системы «Help Navigator» пакета MatLab 6.5) построения спектра изображения *x* размером $N \times N$ пикселей, представляющего собой белый прямоугольник на черном фоне. Программа открывает четыре окна. В первом окне выводится исходное изображение, во втором окне – спектр в таком же разрешении, как исходное изображение ($N \times N$), в третьем окне – спектр с разрешением 256×256, в четвертом – центрированный спектр с разрешением 256×256. Спектры выводятся в логарифмическом масштабе.

```
close all;
N=30;
x=zeros(N,N);
x(5:24,13:17) = 1;
figure, imshow(x,'notruesize')
%
F = fft2(x);
F2 = log(abs(F));
figure, imshow(F2,[-1 5],'notruesize'); colormap(jet); colorbar
%
F = fft2(x,256,256);
figure, imshow(log(abs(F)),[-1 5]); colormap(jet); colorbar
%
F = fft2(x,256,256);
F2 = fftshift(F);
figure, imshow(log(abs(F2)),[-1 5]); colormap(jet); colorbar
```

Возможность отображения спектра двухмерного сигнала в трехмерном пространстве координат обеспечивает функция *surfc()*. Полученное в окне с ее помощью изображение спектра может быть повернуто и наклонено с помощью указателя мышки.

Пример 4.3. М-программа (для пакета MatLab 6.5) построения спектра изображения *'frame_64.bmp'* размером 64×64 пикселя и его пороговой обработки в спектральной области (обнуление действительной части частотных компонент, со значениями, меньше порога). Программа открывает 6 окон. В первом окне выводится исходное изображение, во втором окне – центрированный спектр в логарифмическом масштабе с разрешением $N \times N$, в третьем окне – трехмерное представление центрированного спектра в логарифмическом масштабе с разрешением $N \times N$, в четвертом окне – центрированный спектр с разрешением $N \times N$, в пятом

окне – трехмерное представление центрированного спектра с разрешением $N \times N$, в шестом – восстановленное после пороговой обработки изображения. Для оценки качества восстановления изображения программа рассчитывает значения среднеквадратической ошибки *MSE* и пикового отношения сигнал-шум *PSNR* в соответствие с выражениями

$$MSE = \frac{1}{N \times N} \sum_{i=0}^{n-1} \left(X_i - \widetilde{X}_i \right)^2 , \tag{4.5}$$

$$PSNR = 10 \log_{10} \left(\frac{\left(2^{BD} - 1\right)^2}{MSE} \right), \tag{4.6}$$

где $X_i, \widetilde{X}_i$ – исходное и восстановленное значения *i*-го пикселя изображения;

$N \times N$ – общее число пикселей изображения; *BD* – битовая глубина (bit depth – число бит на пиксель (bpp) для исходного изображения).

```
TH=1000; % переменная, определяющая значение порога
N=64; % переменная, определяющая разрешение спектрального представления
%
close all
[f, map] = imread('frame_64.bmp'); aaa(:,:)=f(:,:);
figure, imshow(f,map);
%
r=double(f);
f2=fft2(r,N,N); J_(:,:)=f2(:,:);
f3=fftshift(f2);
f4=abs(f3);
f5 = log(1+f3.*conj(f3)/N);
%
umin=(min(min(f5))); umax=(max(max(f5)));
as=(f5-umin)/(umax-umin);
f1_=uint8(255*as);
figure, imshow(f1_,256);
cs=as;
% 3D
fx=1:1:N; fy=1:1:N;
figure, surfc(fx,fy,as); colormap([1  1  0; 0  1  1]); axis([0 N 0 N 0 1.]);
%
umin=(min(min(f4))); umax=(max(max(f4)));
as=(f4-umin)/(umax-umin);
f1_=uint8(255*as);
figure, imshow(f1_);
% 3D
fx=1:1:N; fy=1:1:N;
figure, surfc(fx,fy,as); colormap([1  1  0; 0  1  1]); axis([0 N 0 N 0 1.]);
```

```
%
% Поиск максимума и вычисление значения порога
J=abs(J_);
Max_=max(max(J));
Min_=min(min(J));
if Max_>abs(Min_)
   Maxxx=Max_;
else
   Maxxx=abs(Min_);
end
threshold=Maxxx/TH;
% Пороговая обработка
y=1;
while y<=64
   x=1;
   while x<=64
      if abs(J(y,x))<=threshold
         J_(y,x)=0;
      end
      x=x+1;
   end
   y=y+1;
end
%
Y = abs(ifft2(J_));
Y_=uint8(Y);
imwrite(Y_,'Re.bmp','bmp');
figure, imshow(Y_,256);
%
aaa_=double(aaa);
MSE=double(sum(sum((Y-aaa_).^2))/(64*64)) % MSE
PSNR=double(10*log10(255^2/double(MSE))) % PSNR
```

4.2. Дискретное косинусное преобразование

Одномерное прямое и обратное дискретные косинусные преобразования вычисляются в соответствие с выражениями

$$y(k)=\omega(k)\cdot\sum_{n=1}^{N}x(n)\cdot\cos\frac{\pi\cdot(2\cdot n-1)\cdot(k-1)}{2\cdot N},\ k=\overline{1,N}, \tag{4.7}$$

$$x(n)=\sum_{k=1}^{N}\omega(k)\cdot y(k)\cdot\cos\frac{\pi\cdot(2\cdot n-1)\cdot(k-1)}{2\cdot N},\ n=\overline{1,N}, \tag{4.8}$$

где $\omega(k)=\begin{cases}\dfrac{1}{\sqrt{N}}, k=1,\\ \sqrt{\dfrac{2}{N}}, 2\le k\le N.\end{cases}$

Для реализации одномерного дискретного косинусного преобразования в пакете MatLab предусмотрены функции *dct()* (прямое преобразование) и *idct()* (обратное преобразование).

Двухмерное прямое и обратное дискретные косинусные преобразования вычисляются в соответствие с выражениями

$$y_{p,q}=\alpha_p\cdot\alpha_q\cdot\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}x_{m,n}\cdot\cos\frac{\pi\cdot(2\cdot m+1)\cdot p}{2\cdot M}\cdot\cos\frac{\pi\cdot(2\cdot n+1)\cdot q}{2\cdot N},\ \begin{cases}p=\overline{0,M-1},\\ q=\overline{0,N-1}.\end{cases} \quad (4.9)$$

$$x_{m,n}=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\alpha_p\cdot\alpha_q\cdot y_{p,q}\cdot\cos\frac{\pi\cdot(2\cdot m+1)\cdot p}{2\cdot M}\cdot\cos\frac{\pi\cdot(2\cdot n+1)\cdot q}{2\cdot N},\ \begin{cases}m=\overline{0,M-1},\\ n=\overline{0,N-1},\end{cases} \quad (4.10)$$

где $\alpha_p=\begin{cases}\dfrac{1}{\sqrt{M}}, p=0,\\ \sqrt{\dfrac{2}{M}}, 1\le p<M,\end{cases}$ $\alpha_q=\begin{cases}\dfrac{1}{\sqrt{N}}, p=0,\\ \sqrt{\dfrac{2}{N}}, 1\le q<N.\end{cases}$

Для реализации двухмерного дискретного косинусного преобразования в пакете MatLab предусмотрены функции *dct2()* (прямое преобразование) и *idct2()* (обратное преобразование). Подробную информацию об этих и других функциях MatLab и особенностях их использования можно получить через встроенную в MatLab справочную систему «Help Navigator», загружаемую при выборе опции «Full Product Family Help» в меню «Help» главного окна MatLab.

Пример 4.4. M-программа (из встроенной справочной системы «Help Navigator» пакета MatLab 6.5) построения и обработки dct-спектра одномерного сигнала *x*.

```
x = (1:100) + 50*cos((1:100)*2*pi/40);
X = dct(x);
[XX,ind] = sort(abs(X)); ind = fliplr(ind);
i = 1;
while (norm([X(ind(1:i)) zeros(1,100-i)])/norm(X)<.99)
  i = i + 1;
end
```

Пример 4.5. M-программа (для пакета MatLab 6.5) построения dct-спектра изображения *'frame_64.bmp'* размером 64×64 пикселя и его пороговой обработки в спектральной области (обнуление частотных компонент со значениями меньше порога). Программа открывает 4 окна. В первом окне выводится dct-спектр, во втором окне – трехмерное представление dct-спектра, в третьем окне – трехмерное

представление dct-спектра после пороговой обработки, в четвертом окне – восстановленное после пороговой обработки изображение. Для оценки качества восстановления изображения программа рассчитывает значения среднеквадратической ошибки *MSE* и пикового отношения сигнал-шум *PSNR*.

```
close all;
image_fn='frame_64.bmp';
[Image_in,map]=imread(image_fn);
Image=double(Image_in);
J = dct2(Image);
imshow(log(abs(J)),[]), colormap(jet(64)), colorbar
% 3D
NN=64; fx=1:1:NN; fy=1:1:NN;
figure, surfc(fx,fy,J); colormap([1  1  0; 0  1  1]); axis([0 NN 0 NN -500 500]);
Max_=max(max(J)); Min_=min(min(J));
if Max_>abs(Min_)
   Maxxx=Max_;
else
   Maxxx=abs(Min_);
end
percent = 0.8; threshold=Maxxx/100*percent;
y=1;
while y<=64
   x=1;
   while x<=64
      if abs(J(y,x))<=threshold
         J(y,x)=0;
      end
      x=x+1;
   end
   y=y+1;
end
%%%%%%%%%%%%%%%%%%%%
% 3D
NN=64; fx=1:1:NN; fy=1:1:NN;
figure, surfc(fx,fy,J); colormap([1  1  0; 0  1  1]); axis([0 NN 0 NN -300 250]);
%%%%%%%%%%%%%%%%%%%%%%%%
Reconstruction_Image = idct2(J);
B=uint8(Reconstruction_Image);
figure, imshow(B,256);
%imshow(log(abs(B)),[])

num = int2str(percent*10);
namefile = strcat('prmd_test1_', num);
```

```
namefile = strcat(namefile, '.bmp');
imwrite(B,namefile,'bmp');
MSE=double(sum(sum((double(B)-Image).^2))/(64*64)) % MSE
PSNR=double(10*log10(255^2/double(MSE))) % PSNR
```

4.3. Дискретное вейвлет-преобразование

Для одномерного исходного сигнала $\vec{x} = \left(x(0), ..., x(n), ..., x(N-1)\right)$ длиной $N = 2^J$ отсчетов, где J – положительное целое или разрешение сигнала $\vec{x}$, одномерное ДВП вычисляется с помощью иерархического алгоритма Mallat [10], который использует взаимосвязь между коэффициентами ДВП соседних масштабов и операции линейной фильтрации. Таким образом, при представлении ДВП в понятиях банка фильтров, определяемого как множества субполосных фильтров, коэффициенты преобразования (аппроксимационные $s_j(k)$ и детализирующие $d_j(k)$ коэффициенты при разрешении 2^j) на текущем масштабе j $(1 \le j \le J)$ вычисляются на основе аппроксимационных $s_{j-1}(k)$ коэффициентов предыдущего масштаба $j-1$ с использованием соотношений

$$s_j(k) = \varphi\left(h_\varphi, s_{j-1}(k)\right) = \sum_{l=-\lfloor L/2 \rfloor}^{\lfloor L/2 \rfloor} h_\varphi(l) s_{j-1}\left((2k+l) \bmod N_{j-1}\right), \tag{4.11}$$

$$d_j(k) = \psi\left(h_\psi, s_{j-1}(k)\right) = \sum_{l=-\lfloor L/2 \rfloor}^{\lfloor L/2 \rfloor} h_\psi(l) s_{j-1}\left((2k+l) \bmod N_{j-1}\right), \tag{4.12}$$

где $\varphi(\cdot)$ и $\psi(\cdot)$ – функции линейной свертки сигнала аппроксимационных коэффициентов и конечных импульсных характеристик низкочастотного скейлинг-фильтра h_φ и высокочастотного вейвлет-фильтра h_ψ длиной $(L+1)$; $J = \log_2 N$ – максимально возможное количество уровней разложения исходного сигнала $\vec{x}$; $s_0(k) = x(k)$; $k = \overline{0, 2^{J-j} - 1} = 0, ..., N_j - 1$; $N_j = N/2^j$ – число аппроксимационных или детализирующих коэффициентов на масштабе j; $\bmod N_{j-1}$ – операция по модулю N_j для учета границ сигнала конечной длины, $\lfloor\ \rfloor$ – операция округления с недостатком.

На рис. 4.1 представлены особенности структуры стандартного одномерного быстрого вейвлет-преобразования.

Для двухканального биортогонального банка фильтра, обеспечивающего восстановление исходного сигнала и состоящего из $\left\{h_\varphi, \widetilde{h}_\varphi, h_\psi, \widetilde{h}_\psi\right\}$, взаимосвязь между двумя низкочастотными биортогональными фильтрами h_φ и $\widetilde{h}_\varphi$ и двумя высокочастотными биортогональными фильтрами h_ψ и $\widetilde{h}_\psi$ имеет вид

$$h_\psi(l) = (-1)^l \widetilde{h}_\varphi(l), \tag{4.13}$$

$$\widetilde{h}_\psi(l) = -(-1)^l h_\varphi(l), \tag{4.14}$$

где $\{h_\varphi, h_\psi\}$ и $\{\tilde{h}_\varphi, \tilde{h}_\psi\}$ – пары биортогональных фильтров для декомпозиции (анализа) и восстановления (синтеза) сигналов соответственно.

Кроме того, в этом случае справедливы следующие биортогональные соотношения:

$$\sum_l h_\varphi(l) = \sum_l \tilde{h}_\varphi(l) = \sqrt{2}, \qquad \sum_l h_\psi(l) = \sum_l \tilde{h}_\psi(l) = 0, \qquad \sum_l h_\varphi(l)\tilde{h}_\varphi(l+2k) = \delta_{k0},$$

$$\sum_l h_\psi(l)\tilde{h}_\psi(l+2k) = \delta_{k0}, \quad \sum_l \tilde{h}_\varphi(l)h_\psi(l+2k) = 0, \quad \sum_l h_\varphi(l)\tilde{h}_\psi(l+2k) = 0,$$ где $2k$ – сдвиг на четное число точек; $\delta_{k0} = \begin{cases} 1 & \text{при } k = 0, \\ 0 & \text{при } k \neq 0 \end{cases}$ – символ Кронекера или функция двух целых переменных.

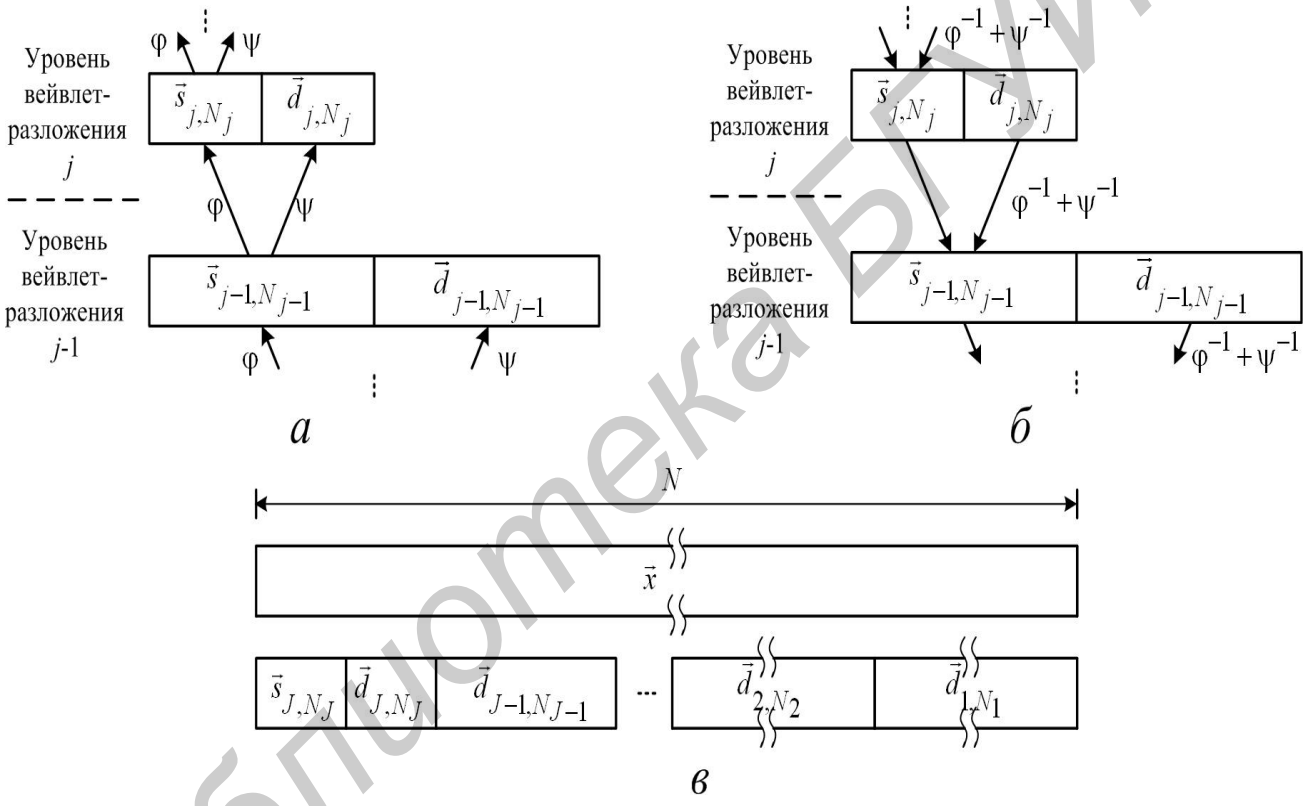

Рис. 4.1. Структура одномерного быстрого вейвлет-преобразования

Обратное рациональное вейвлет-преобразование (рис. 4.1, *б*) представляется с помощью следующего рекуррентного выражения:

$$\vec{s}_{j-1,N_{j-1}} = \varphi^{-1}\left(\tilde{h}_\varphi, \vec{s}_{j,N_j}\right) + \psi^{-1}\left(\tilde{h}_\psi, \vec{d}_{j,N_j}\right), \tag{4.15}$$

где $\varphi^{-1}(\cdot)$ и $\psi^{-1}(\cdot)$ – функции линейной свертки сигналов аппроксимационных и детализирующих коэффициентов и конечных импульсных характеристик низкочастотного $\tilde{h}_\varphi$ и высокочастотного $\tilde{h}_\psi$ фильтров соответственно.

Прямое и обратное целочисленные лифтинг-вейвлет-преобразования [11, 12] для одного уровня разложения имеют следующий вид:

$$\begin{cases} a_0(k) = s_{j-1}(2k), \quad b_0(k) = s_{j-1}(2k+1), \\ b_i(k) = b_{i-1}(k) - \left\lfloor \sum_l p_i(l) a_{i-1}(k-l) + \frac{1}{2} \right\rfloor, \\ a_i(k) = a_{i-1}(k) + \left\lfloor \sum_l u_i(l) b_i(k-l) + \frac{1}{2} \right\rfloor, \\ s_j(k) = a_I(k)/K, \; d_j(k) = K b_I(k), \end{cases} \tag{4.16}$$

$$\begin{cases} a_I(k) = K s_j(k), \; b_I(k) = d_j(k)/K, \\ a_{i-1}(k) = a_i(k) - \left\lfloor \sum_l u_i(l) b_i(k-l) + \frac{1}{2} \right\rfloor, \\ b_{i-1}(k) = b_i(k) + \left\lfloor \sum_l p_i(l) a_{i-1}(k-l) + \frac{1}{2} \right\rfloor, \\ s_{j-1}(2k) = a_0(k), \quad s_{j-1}(2k+1) = b_0(k), \end{cases} \tag{4.17}$$

где $p_i(l)$ и $u_i(l)$ – коэффициенты предсказания $b_{i-1}(k)$ и уточнения $a_{i-1}(k)$ по ближайшим соседним коэффициентам на i-м лифтинг-шаге соответственно; $a_i(k)$ и $b_i(k)$ – аппроксимирующие и детализирующие коэффициенты вейвлет-преобразования на i-м лифтинг-шаге соответственно; $\lfloor \cdot \rfloor$ – операция округления с недостатком; $i = \overline{1, I}$; I – количество лифтинг-шагов для используемой вейвлет-функции; K – масштабирующий коэффициент.

Вейвлет-преобразование двухмерных сигналов (изображений) осуществляется как последовательность одномерных преобразований над строками и столбцами изображения.

Для реализации дискретного вейвлет-преобразования в пакете MatLab предусмотрены функции MatLab, описанные в разделе Wavelet Toolbox справочной системы «Help Navigator».

5. Лабораторная работа «Предварительная обработка мультимедийной информации на основе дискретных информационных преобразований»

5.1. Цель работы

Ознакомиться с принципами программного моделирования в интегрированной среде MatLab/C, стандартными функциями доступа к данным пакета MatLab, приобрести навыки по программной реализации предварительной обработки медиаданных на основе дискретных информационных преобразований в среде MatLab, оценить эффективность дискретных информационных преобразований для компактного представления медиаданных.

5.2. Описание лабораторной работы

Лабораторная работа выполняется на базе ПЭВМ с установленными пакетами MatLab и MS Visual Studio, а также приложениями архивирования файлов (Zip, Rar и т. п.). В качестве исходных данных используются тестовые полутоновые изображения в формате bmp.

5.3. Предварительное задание для лабораторной работы

1. Ознакомиться с теоретической частью пособия.
2. Ознакомиться с установленными на ПЭВМ программными пакетами MatLab и MS Visual Studio.
3. Разработать программные m-модули, обеспечивающие доступ к полутоновым изображениям в формате bmp.
4. Разработать программные m-модули дискретных информационных преобразований над двухмерными массивами данных. Реализовать часть функций на языке программирования C и подключить к программным m-модулям в виде dll-библиотеки.
5. Разработать программный m-модуль оболочки предварительной обработки изображений на основе дискретных информационных преобразований.
6. Осуществить интеграцию программного m-модуля оболочки и программных m-модулей дискретных информационных преобразований.
7. Разработать пользовательский интерфейс для программного проекта предварительной обработки изображений на основе дискретных информационных преобразований.

5.4. Порядок выполнения и методические указания

1. Выбрать тестовые полутоновые изображения в формате bmp.
2. С помощью разработанных программных средств выполнить над тестовыми изображениями дискретное информационное преобразование.
3. Сохранить значения коэффициентов преобразования в бинарный файл.
4. С помощью архиватора осуществить сжатие файла коэффициентов преобразования и вычислить коэффициент сжатия.
5. Выполнить пп. 2 – 4 для различных дискретных информационных преобразований.

6. Оценить компактность представления информации с помощью различных базисных функций.

7. Удалить из полученных матриц коэффициентов преобразования одинаковое количество информации, начиная с менее значимых бит.

8. Сохранить значения обработанных коэффициентов преобразования в бинарные файлы.

9. С помощью архиватора осуществить сжатие файлов коэффициентов преобразования и вычислить коэффициенты сжатия.

10. С помощью разработанных программных средств выполнить над матрицами обработанных коэффициентов обратные дискретные информационные преобразования.

11. С помощью выражений (4.5) и (4.6) оценить качество восстановления тестовых изображений.

12. Выполнить пп. 7 – 11 для различного количества удаляемой информации.

13. Построить графики зависимости значений MSE и PSNR от коэффициента сжатия для всех используемых тестовых изображений и дискретных информационных преобразований.

14. Оценить эффективность предварительной обработки полутоновых изображений с помощью различных дискретных информационных преобразований для их последующего сжатия с потерями.

5.5. Контрольные вопросы

1. Принципы интегрирования программных сред MatLab и C.
2. Обеспечение обмена данными между m- и c-программными модулями.
3. Создание m-проекта.
4. Создание интегрированного m/c-проекта.
5. Графические объекты пользовательских интерфейсов в среде MatLab.
6. Редактор приложения среды MatLab.
7. Программирование событий в среде MatLab.
8. Редактор меню среды MatLab.
9. Функции MatLab для доступа к аудиоданным.
10. Функции MatLab для доступа к неподвижным изображениям.
11. Функции MatLab для доступа к видеоданным.
12. Функции MatLab для доступа к бинарным данным.
13. Функции MatLab для реализации дискретного преобразования Фурье.
14. Функции MatLab для построения спектров.
15. Функции MatLab для вычисления дискретного косинусного преобразования.
16. Функции MatLab для вычисления дискретного вейвлет-преобразования.

Литература

1. Борискевич, А. А. Поиск и селективное шифрование объектов изображений // А. А. Борискевич, В. Ю. Цветков // Обеспечение безопасности информации в информационных системах : материалы конф. / Академия управления при Президенте Республики Беларусь. – 2004. – С. 145–147.

2. Борискевич, А. А. Метод шифрования речи и данных на основе рекурсивных разверток и муаровых ключей / А. А. Борискевич, В. Ю. Цветков // Доклады БГУИР. – 2005. – №5. – С. 66–67.

3. Борискевич, А. А. Исследование характеристик хаос-генератора двухмерных образов на основе модели клеточного автомата // А. А. Борискевич, И. Л. Горнович, В. Ю. Цветков / Совр. средства связи : Материалы конф. / Изв. Бел. инж. акад. – 2005. – №1 (19)/4. – С. 42–44.

4. Цветков, В. Ю. Метод двумерного представления аудио и речевой информации на основе рекурсивных разверток / В. Ю. Цветков // Доклады БГУИР. – 2005. – №6. – С. 34–40.

5. Борискевич, А. А. Ретинальная обработка изображений на основе модели клеточного автомата // А. А. Борискевич, В. Ю. Цветков / Инженерный вестник. – 2005. – №1 (20). – С. 9–13.

6. Борискевич, А. А. Модель многослойного двухсвязного клеточного автомата для генерации последовательности хаотических образов // А. А. Борискевич, И. Л. Горнович, В. Ю. Цветков / Докл. БГУИР. – 2006. – №6 (12). – С. 30–39.

7. Борискевич, А. А. Оценка влияния модификации вейвлет-коэффициентов на сжатие медиаданных / А. А. Борискевич, В. Ю. Цветков // Докл. БГУИР. – 2006. – №4 (16). – С. 17–24.

8. Борискевич, А. А. Компактное описание вейвлет-коэффициентов для сжатия медиаданных // А. А. Борискевич, В. Ю. Цветков // Информатика. – 2007. – №1 (13). – С. 46–56.

9. Борискевич, А. А. Компактное описание и формирование N-мерных рекурсивных разверток // А. А. Борискевич, В. Ю. Цветков // Информатика. – 2007. – №2 (14). – С. 5–15.

10. Mallat S. // IEEE Trans. Pattern Analysis and Machine Intelligence. – 1989. – N. 7. – P. 674–693.

11. Sweldens W. // Applied and Computational Harmonic Analysis. – 1996. – N. 2. – P. 186–200.

12. Adams M., Antoniou A. // Proc. of IEEE Pacific Rim Conference. – 1997. – Vol. 1. – P. 489–492.

Св. план 2009, поз. 107, 108

Учебное издание

Борискевич Анатолий Антонович
Лагойко Алексей Юрьевич
Цветков Виктор Юрьевич

ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА МУЛЬТИМЕДИЙНОЙ ИНФОРМАЦИИ В ИНТЕГРИРОВАННОЙ СРЕДЕ MATLAB/C

МЕТОДИЧЕСКОЕ ПОСОБИЕ
по курсу
«Цифровая обработка и защита мультимедийной информации»
для студентов специальности
«Системы распределения мультимедийной информации»
всех форм обучения

Редактор Е. Н. Батурчик
Корректор А. В. Тюхай
Компьютерная верстка М. В. Гуртатовская

Подписано в печать	Формат 60x84 1/16	Бумага офсетная.
Гарнитура «Таймс».	Отпечатано на ризографе.	Усл. печ. л.
Уч.-изд. л. 2,1.	Тираж 100 экз.	Заказ 734.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6