

Практикум (лабораторный)

Практикум состоит из 8 лабораторных работ.

Для успешного выполнения лабораторных работ необходимо изучение соответствующих модулей теоретического блока (лекций).

Общие требования к содержанию, оформлению и порядку выполнения

Перед выполнением лабораторной работы необходимо создать папку «Ваша фамилия Lab №__» (Использовать только буквы латинского алфавита. Например: «Ivanov I.P. Lab №__»). В эту папку в ходе выполнения работы необходимо сохранять требуемые материалы.

Лабораторные работы необходимо выполнять согласно своему варианту. Перед выполнением лабораторной работы изучите теоретическую часть, далее необходимо изучить пример выполнения лабораторной работы, а затем приступать к выполнению своего варианта лабораторной работы.

Задания лабораторной работы необходимо выполнять последовательно, при необходимости результат выполнения сохранять в свою папку. Папку с результатами необходимо заархивировать, создав один файл архива в формате ZIP. Файлу архива необходимо дать имя в формате: «Ваша фамилия Lab№__» (Использовать только буквы латинского алфавита. Например: «Ivanov I.P. Lab №__.zip»). Полученный файл архива необходимо загрузить на страницу задания «Лабораторная работа №__».

Способ оценки результатов выполненных работ:

Лабораторная работа №1. Работа со строками и текстовыми файлами в MatLab

Цель работы:

Целью лабораторной работы является получение навыков самостоятельной алгоритмической и программной реализации на компьютерной технике методов обработки строк и текстовых файлов в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-5	3-1	Способен к восприятию и воспроизведению информации
	3-2	Знает алгоритмы целеполагания и выбора путей их достижения
	3-2	Знает методы формализации решения

		прикладных задач
--	--	------------------

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-5	У-1	Способен к восприятию и воспроизведению информации
	У-2	Знает алгоритмы целеполагания и выбора путей их достижения
ПК-25	У-1	Знает различные математические методы решения прикладных задач
	У-2	Знает методы формализации решения прикладных задач

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-5	B-1	Способен к восприятию и воспроизведению информации
	B-2	Знает алгоритмы целеполагания и выбора путей их достижения
ОК-7	B-1	Знает требования к построению речевого взаимодействия
	B-2	Находит аргументы и логически строить высказывание
ОК-10	B-1	Понимает необходимость саморазвития
	B-2	Выражает желание повышать свою квалификацию и мастерство
ПК-1	B-1	Знает основы современных технологий сбора, обработки и представления информации
	B-2	Способен ориентироваться в информационном потоке в глобальных компьютерных сетях
ПК-25	B-1	Знает различные математические методы решения прикладных задач
	B-2	Знает методы формализации решения прикладных задач

Теоретическая часть
Работа со строками

Переменные, содержащие строки, будем называть строковыми переменными. Для ограничения строки используются апострофы, например, **оператор присваивания**

```
>> str='Hello, World! '
```

приводит к образованию строковой переменной

```
str =
```

```
Hello, World!
```

Убедитесь, что строковая переменная `str` действительно является массивом, обратившись к ее элементам: `str(8)`, `str(1:5)`. Команда `whos` позволяет получить подробную информацию о `str`, в частности, `str` хранится в виде вектор-строки, ее длина равна 13. Поскольку строковые переменные являются массивами, то к ним применимы некоторые функции и операции, рассмотренные нами ранее.

Длина строковой переменной, т.е. число символов в ней, находится при помощи функции `length`.

Допустимо **сцепление строк** как вектор-строк с использованием квадратных скобок. Создайте еще одну строковую переменную `str1`, содержащую текст 'My name is Igor.', и осуществите **сцепление**:

```
>> strnew=[str str1]
```

```
strnew =
```

```
Hello, World! My name is Igor.
```

Сцепление строк может быть проведено как с использованием квадратных скобок, так и при помощи функции `strcat`. Входными аргументами `strcat` являются сцепляемые строки, их число неограничено, а результат возвращается в выходном аргументе.

Функция `strcat` игнорирует пробелы в конце каждой строки. Для строк: `s1='abc'` и `s2='def'` сцепления `s=[s1 s2]` и `s=strcat(s1,s2)` приведут к разным результатам.

Поиск позиций вхождения подстроки в строку производится при помощи функции `findstr`. Ее входными аргументами являются строка и подстрока, а выходным — вектор позиций, начиная с которых подстрока входит в строку, например:

```
>> s='abcabcddefbcc';
```

```
>> s1='bc';
```

```
>> p=findstr(s,s1)
```

```
p =
```

```
2 4 10
```

Порядок входных аргументов не важен, подстрокой всегда считается аргумент меньшего размера.

Функция `strcmp` предназначена для **сравнения двух строк**, которые указываются во входных аргументах: `strcmp(s1,s2)`. Результатом является логическая единица или ноль. Функция `strncmp` сравнивает только часть строк от первого символа до символа, номер которого указан в третьем входном аргументе: `strncmp(s1,s2,8)`. Данные функции имеют аналоги:

`strcmp` и `strncmp`, которые проводят сравнение без учета регистра, т.е., например, символы 'A' и 'a' считаются одинаковыми.

Для замены в строке одной подстроки на другую служит функция `strrep`. Пусть, например, требуется заменить в строке `str` подстроку `s1` на `s2` и записать обновленную строку в `strnew`. Тогда вызов функции `strrep` должен выглядеть так: `strnew=strrep(str,s1,s2)`. Здесь важен порядок аргументов.

Преобразование всех букв строки в строчные (прописные) производит функция `lower (upper)`, например: `lower('MatLab')` приводит к '`matlab`', а `upper('MatLab')` — к '`MATLAB`'.

То, что строки являются массивами символов, позволяет легко писать собственные файл-функции обработки строк. Предположим, что требуется переставить символы в строке в обратном порядке. Файл-функция, приведенная на листинге 1, решает поставленную задачу.

Листинг 1. Файл-функция для перестановки символов в строке

```
function sout=strinv(s)
L=length(s);
for k=1:L
sout(L-k+1)=s(k);
end
```

Перейдем теперь к изучению массивов строк. Можно считать, что массив строк является вектор-столбцом, каждый элемент которого есть строка, причем длины всех строк одинаковы. В результате получается прямоугольная матрица, состоящая из символов. Например, для переменных `s1='March'`, `s2='April'`, `s3='May'`, операция `S=[s1; s2]` является допустимой, и приводит к массиву строк:

```
S =
March
April
```

Аналогичное объединение трех строк `S=[s1; s2; s3]` вызовет вывод сообщения об ошибке. Можно, конечно, дополнить при помощи сцепления, каждую строку пробелами справа до длины наибольшей из строк и затем производить формирование массива.

Функция `char` как раз и решает эту задачу, создавая из строк разной длины массив строк:

```
>> S=char(s1,s2,s3)
S =
March
April
May
```

Проверьте при помощи команды `whos`, что `S` является массивом размера 3 на 5. Для определения размеров массива строк, так же, как и любых массивов, используется `size`.

Обращение к элементам массива строк производится аналогично обращению к элементам числового массива — при помощи индексирования числами или двоеточием, например: **S(3,2)**, **S(2,:)**, **S(2:3,1:4)**. При выделении строки из массива строк в конце могут остаться пробелы. Если они не нужны, то их следует удалить, воспользовавшись функцией **deblank**. Сравните, к чему приводят **S(3,:)** и **deblank(S(3,:))**.

Поиск подстроки в массиве строк выполняется функцией **strmatch**. Входными аргументами **strmatch** являются подстрока и массив строк, а выходным — вектор с номерами строк, содержащих подстроку:

```
>> p=strmatch('Ma',S)
p =
1
3
```

Если требуется искать номера строк, точно совпадающих с подстрокой, то следует задать третий дополнительный входной аргумент '**exact**'.

Некоторые функции обработки строк могут работать и с массивами строк. Например, при сцеплении массивов с одинаковым числом строк при помощи **strcat**, сцепление применяется к соответствующим строкам массивов, образуя новый массив строк. Среди входных аргументов **strcat** может быть и строка. В этом случае она добавляется ко всем строкам массива.

Среди символов строки могут быть цифры и точка, т.е. строка может содержать числа. Важно понимать, что оператор присваивания **b=10** приводит к образованию числовой переменной **b**, а оператор **s='10'** — строковой.

Преобразование целого числа в строку производится при помощи функции **int2str**, входным аргументом которой является число, а выходным — строка, например:

```
>> str=['May, ' int2str(10)]
str =
May, 10
```

Заметьте, что нецелые числа перед преобразованием округляются.

Для перевода нецелых чисел в строки служит функция **num2str**:

```
>> str=num2str(pi)
str =
3.1416
```

Дополнительный второй входной аргумент **num2str** предназначен для указания количества цифр в строке с результатом: **str=num2str(pi,11)**.

Возможно более гибкое управление преобразованием при помощи строки специального вида, определяющей формат (экспоненциальный или с плавающей точкой) и количество позиций, отводимых под число. Стока с форматом задается в качестве второго входного аргумента **num2str**, начинается со знака процента и имеет вид '**%A.ах**', где:

A — количество позиций, отводимое под все число;

a — количество цифр после десятичной точки;

x — формат вывода, который может принимать, например, одно из следующих значений: *f* (с плавающей точкой), *e* (экспоненциальный) или *g* (автоматический подбор наилучшего представления).

Сравните результаты следующих преобразований числа в строку:

```
>> z=198.23981  
>> sf=num2str(z, '%12.5f')  
>> se=num2str(z, '%12.5e')  
>> sg=num2str(z, '%12.5g')
```

Если входным аргументом функций *int2str* и *num2str* является матрица, то результатом будет массив строк. В этом случае в строке с форматом часто полезно указать разделитель, например запятую и пробел:

```
>> R=rand(5)  
>> S=num2str(R, '%7.2e, ')
```

Обратная задача, а именно, преобразование строковой переменной, содержащей число, в числовую переменную решается при помощи функции **str2num**. Входным аргументом *str2num* может быть строка с представлением целого, вещественного или комплексного числа в соответствии с правилами MatLab, например:

```
str2num('2.9e-3'), str2num('0.1'), str2num('4.6+4i').
```

Работа с текстовыми файлами

MatLab предлагает достаточно универсальные способы считывания данных из текстовых файлов и записи данных в требуемом виде в текстовый файл.

Работа с текстовыми файлами состоит из трех этапов:

1. открытие файла;
2. считывание или запись данных;
3. закрытие файла.

Для **открытия файла** служит функция **fopen**, которая вызывается с двумя входными аргументами: именем файла и строкой, задающей способ доступа к файлу. **Выходным аргументом** *fopen* является идентификатор файла, т.е. переменная, которая впоследствии используется при любом обращении к файлу. Функция *fopen* возвращает *-1*, если при открытии файла возникла ошибка.

Существует четыре основных способа открытия файла:

1. *f=fopen('myfile.dat','rt')* — открытие текстового файла *myfile.dat* только для чтения из него;
2. *f=fopen('myfile.dat','rt+')* — открытие текстового файла *myfile.dat* для чтения и записи данных;
3. *f=fopen('myfile.dat','wt')* — создание пустого текстового файла *myfile.dat* только для записи данных;
4. *f=fopen('myfile.dat','wt+')* — создание пустого текстового файла *myfile.dat* для записи и чтения данных.

При использовании двух последних вариантов следует соблюдать осторожность — если файл myfile.dat уже существует, то его содержимое будет уничтожено. После открытия файла появляется возможность считывать из него информацию, или заносить ее в файл.

По окончании работы с файлом необходимо закрыть его при помощи **fclose(f)**.

Построчное считывание информации из текстового файла производится при помощи функции **fgetl**. Входным аргументом fgetl является идентификатор файла, а выходным — текущая строка. Каждый вызов fgetl приводит к считыванию одной строки и переводу текущей позиции в файле на начало следующей строки. Команды, приведенные на листинге 2, последовательночитывают из файла myfile.dat первые три строки в строковые переменные str1, str2, str3.

Листинг 2. Считывание трех первых строк из текстового файла

```
f=fopen('myfile.dat','rt');
str1=fgetl(f);
str2=fgetl(f);
str3=fgetl(f);
fclose(f);
```

При **последовательном считывании** рано или поздно будет достигнут конец файла, при этом функция fgetl вернет минус 1. Лучше всего перед считыванием проверять, не является ли текущая позиция в файле последней. Для этого предназначена функция **feof**, которая вызывается от идентификатора файла и возвращает единицу, если текущая позиция последняя и ноль, в противном случае. Обычно последовательное считывание организуется при помощи цикла while. Листинг 3 содержит файл-функцию viewfile, которая считывает строки файла, и выводит их в командное окно.

Листинг 3. Файл-функция viewfile

```
function viewfile(fname)
f=fopen(fname,'rt');
while feof(f)==0
s=fgetl(f);
disp(s)
end
fclose(f);
```

Вызов viewfile('viewfile.m') приводит к отображению текста самой файл-функции в командном окне.

Строки записываются в текстовый файл при помощи функции **fprintf**, ее первым входным аргументом является идентификатор файла, а вторым — добавляемая строка. Символ \n служит для перевода строки. Если поместить его в конец добавляемой строки, то следующая команда fprintf будет осуществлять вывод в файл с новой строки, а если \n находится в начале, то текущая команда fprintf выведет текст с новой строки.

Например, последовательность команд (листинг 4) приведет к появлению в файле my.txt текста из двух строк (листинг 5).

Листинг 4. Вывод строк в текстовый файл

```
f=fopen('my.txt','wt');
fprintf(f,'текст ');
fprintf(f,'еще текст\n');
fprintf(f,'а этот текст с новой строки');
fclose(f);
```

Листинг 5. Результат работы операторов листинга 9.3

```
текст еще текст
а этот текст с новой строки
```

Аналогичного результата можно добиться, переместив `\n` из конца строки второй функции `fprintf` в начало строки третьей функции `fprintf`. Аргументом `fprintf` может быть не только строка, но и строковая переменная. В этом случае для обеспечения вывода с новой строки ее следует сцепить со строкой '`\n`'.

Строка, предназначенная для вывода в текстовый файл, может содержать как текст, так и числа. Часто требуется выделить определенное количество позиций под число и вывести его в экспоненциальном виде или с плавающей точкой и с заданным количеством цифр после десятичной точки. Здесь не обойтись без форматного вывода при помощи `fprintf`, обращение к которой имеет вид:

```
fprintf(идентификатор файла, 'форматы', список
переменных)
```

В списке переменных могут быть как числовые переменные (или числа), так и строковые переменные (или строки).

Второй аргумент является строкой специального вида с форматами, в которых будут выводиться все элементы из списка. Каждый **формат начинается со знака процента**. Для **вывода строк** используется формат `s`, а для **вывода чисел** — `f` (с плавающей точкой) или `e` (экспоненциальный).

Число перед `s` указывает количество позиций, отводимых под вывод строки.

При выводе значений числовых переменных перед `f` или `e` ставится два числа, разделенных точкой. Первое из них означает количество позиций, выделяемых под все значение переменной, а второе — количество знаков после десятичной точки. Таким образом, под каждый из элементов списка отводится поле определенной длины, **выравнивание** в котором по умолчанию производится **по правому краю**. Для **выравнивания по левому краю** следует после знака процента поставить **знак минус**.

Ниже приведены варианты вызова `fprintf` и результаты, незаполненные позиции после вывода (пробелы) обозначены символом `o`.

```
fprintf(f,'%6.2f', pi) oo3.14
```

```
fprintf(f,'%-6.2f', pi) 3.14oo
```

```
fprintf(f,'%14.4e', exp(-5)) ooo6.7379e-003
```

```
fprintf(f, '%-14.4e', exp(-5)) 6.7379e-003
fprintf(f, '%8s', 'текст')    текст
fprintf(f, '%-8s', 'текст')  текст
```

В случае, когда зарезервированного поля не хватает под выводимую строку или число, MatLab автоматически увеличивает длину поля, например:
fprintf(f, '%5.4e', exp(-5)) 6.7379e-003

При одновременном выводе чисел и текста пробелы могут появляться из-за неполнотой заполненных полей, выделенных как под числа, так и под строки. Приведенные ниже операторы и результат их выполнения демонстрируют расположение полей в строке текстового файла. Волнистой линией подчеркнуты поля, выделенные под числа, а прямой — под строки, символ \circ по-прежнему обозначает пробелы.

```
x=0.55;
fprintf(f, '%-5s%6.2f%6s%20.8e', 'x=', x, 'y=', exp(x))
x=0.55y=1.73325302e+000
```

Форматный вывод удобен при формировании файла с таблицей результатов. Предположим, что необходимо записать в файл f.dat таблицу значений функции $f(x) = x^2 \sin x$ для заданного числа n значений $x \in [a, b]$, отстоящих друг от друга на одинаковое расстояние. Файл с таблицей значений должен иметь такую структуру, как показано на листинге 6.

Листинг 6. Текстовый файл с таблицей значений функции

```
f(0.65)=2.55691256e-001
f(0.75)=3.83421803e-001
f(0.85)=5.42800093e-001
f(0.95)=7.34107493e-001
f(1.05)=9.56334106e-001
```

Очевидно, что следует организовать цикл от начального значения аргумента до конечного с шагом, соответствующим заданному числу точек, а внутри цикла вызывать fprintf с подходящим списком вывода и форматами. Символ \n, предназначенный для перевода строки, помещается в конец строки с форматами (см. листинг 7).

Листинг 7. Файл-функция tab, выводящая таблицу значений функции

```
function tab(a,b,n)
h=(b-a)/(n-1);
f=fopen('f.dat', 'wt');
for x=a:h:b
fprintf(f, '%2s%4.2f%2s%15.8e\n',
'f(',x,')=',x^2*sin(x))
end
fclose(f);
```

Обратимся теперь к **считыванию данных из текстового файла**. Функция fscanf осуществляет обратное действие по отношению к fprintf. Каждый вызов fscanf приводит к занесению данных, начинающихся с текущей позиции, в переменную. Тип переменной

определяется заданным форматом. В общем случае, обращение к fscanf имеет вид:

a=fscanf(идентификатор файла, 'формат', число считываемых элементов)

Для считывания строк используется формат %s, для целых чисел — %d, а для вещественных — %g.

Предположим, что файл exper.dat содержит текст, приведенный на листинге 7. Данные отделены друг от друга пробелами. Требуется считать дату проведения эксперимента (число, месяц и год) в подходящие переменные, а результаты занести в числовые массивы TIME и DAT.

Листинг 8. Файл с данными exper.dat

Результаты экспериментов 10 мая 2002

```
t= 0.1 0.2 0.3 0.4 0.5  
G= 3.02 3.05 2.99 2.84 3.11
```

Считывание разнородных данных (числа, строки, массивы) требует контроля, который достигается применением форматов. Первые два элемента файла exper.dat являются строками, следовательно можно сразу занести их в одну строковую переменную head. Очевидно, надо указать формат %s и число 2 считываемых элементов. Далее требуется считать целое число 10, строку 'мая' и снова целое число 2002. Обратите внимание, что перед заполнением массива TIME еще придется предусмотреть считывание строки 't=''. Теперь все готово для занесения пяти вещественных чисел в вектор-столбец TIME при помощи формата %g. Данные из последней строки файла exper.dat извлекаются аналогичным образом (листинг 9).

Листинг 9. Применение fscanf для считывания данных

```
f=fopen('exper.dat','rt');  
head=fscanf(f, '%s', 2)  
data=fscanf(f, '%d', 1)  
month=fscanf(f, '%s', 1)  
year=fscanf(f, '%d', 1)  
str1=fscanf(f, '%s', 1)  
TIME=fscanf(f, '%g', 5)  
str2=fscanf(f, '%s', 1)  
DAT=fscanf(f, '%g', 5)  
fclose(f);
```

Информация в текстовом файле может быть представлена в виде таблицы. Для считывания такой информации следует использовать массив структур с подходящим набором полей. Считывание всей информации реализуется в цикле while, в условии которого производится проверка на достижение конца файла при помощи функции feof.

Функция fscanf предоставляет возможность считывать числа из текстового файла не только в вектор-столбец, но и массив заданных размеров. Расположение чисел по строкам в файле не имеет значения. Размеры формируемого массива указываются в векторе в третьем входном аргументе

fscanf. Рассмотрим считывание числовых данных из файла matr.txt (листинг 10).

Листинг 10. Текстовый файл matr.txt с матрицей

```
1.1 2.2 3.3 4.4  
5.5 6.6 7.7 8.8  
0.2 0.4 0.6 0.8
```

Функция fscanf формирует столбец матрицы, последовательно считывая числа из файла (т. е. построчно). Следовательно, для заполнения матрицы с тремя строками и четырьмя столбцами, необходимо считать данные из файла в массив четыре на три, а затем транспонировать его (листинг 11).

Листинг 11. Считывание матрицы из текстового файла

```
f=fopen('matr.txt');  
M=fscanf(f, '%g', [4 3]);  
M=M'  
fclose(f);
```

Обратите внимание, что массив может иметь размеры не только 3 на 4. Поскольку данныечитываются подряд, то они могут быть занесены и в массив 2 на 6, и 4 на 3 и т. д.

Общая постановка задачи

Написать файл-функцию для решения следующих задач.

1. Определить количество символов в первой строке варианта без учета пробелов.
2. Первая строка является предложением, в котором слова разделены пробелами. Переставить первое и последнее слово.
3. Заменить в первой строке цифры числительными (вместо 1, 2,... — один, два, три,...).
4. Задана строка (первая строка варианта), содержащая текст и числа, разделенные пробелами, выделить числа в числовой массив.
5. Записать данные, указанные в соответствующем варианте, в файл inN.txt, где N – номер варианта.
6. Считать матрицы и вектора из файла в подходящие по размеру массивы. Обратите внимание, что в файлах содержится рядом две или три матрицы или вектора, их следует занести в разные массивы.

Список индивидуальных данных

Варианты:

1	Алексеев Сергей 1980 5 4 4 5 3 5 0.1 0.2 0.3 9.91 1.9 0.4 0.1 8.01 4.7 5.1 3.9 7.16	2	23 марта 2002 0.36 0.32 0.28 0.25 1.399 2.001 9.921 3.21 0.12 0.129 1.865 8.341 9.33 8.01 9.136 8.401 7.133 3.12 3.22
3	Иванов Константин 1981 3 4 3 4 3	4	Time= 0.0 0.1 0.2 0.3 0.4 0.5

	5 1 2 3 4 99 80 5 6 7 8 33 21 15 90		0.6 10 20 40 50 12 19 21 32 44 -1 -2 -3 -4 32 10
5	195251 СПб Политехническая 29 1 2 3 4 100 6 7 8 9 0.1 0.2 0.3 0.4 200 0.5 0.6 0.7 0.8 300	6	Петров Олег 1980 5 5 5 4 4 5 2.1 0.2 0.3 9.91 1.9 0.4 0.1 8.01 7.7 5.1 3.9 5.16
7	195256 СПб Науки 49 4.79 2.001 9.921 3.21 0.25 1.129 1.865 8.341 9.33 8.01 8.136 8.401 7.133 3.12 3.44	8	Результаты 2.1 2.3 2.3 1.9 1.8 2.4 1 2 3 4 99 80 5 6 7 8 33 21 15 90
9	Сидоров Николай 101 521 899 10 20 40 50 12 19 21 32 44 -1 -2 -3 -4 32 10	10	Тимофеев Сергей 570 100 203 1 2 3 4 100 6 7 8 9 0.1 0.2 0.3 0.4 200 0.5 0.6 0.7 0.8 57

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.

Лабораторная работа №2. Работа с матрицами и многомерные вычисления в Matlab

Цель работы:

Целью лабораторной работы является получение навыков самостоятельной алгоритмической и программной реализации на компьютерной технике методов преобразования матриц в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат

ОК-7	3-1	Знает требования к построению речевого взаимодействия
	3-2	Находит аргументы и логически строить высказывание
ПК-14	3-1	Знает методы анализа прикладной области на логическом уровне
	3-2	Знает методы анализа прикладной области на алгоритмическом уровне
ПК-25	3-1	Знает различные математические методы решения прикладных задач
	3-2	Знает методы формализации решения прикладных задач

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-7	У-1	Знает требования к построению речевого взаимодействия
	У-2	Находит аргументы и логически строить высказывание
ОК-10	У-1	Понимает необходимость саморазвития
	У-2	Выражает желание повышать свою квалификацию и мастерство
ПК-14	У-1	Знает методы анализа прикладной области на логическом уровне
	У-2	Знает методы анализа прикладной области на алгоритмическом уровне

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-2	B-1	Способен анализировать социально-экономические проблемы
	B-2	Знает методы расчетов математических моделей
ПК-14	B-1	Знает методы анализа прикладной области на логическом уровне
	B-2	Знает методы анализа прикладной области на алгоритмическом уровне

Теоретическая часть

Матрицы небольших размеров удобно вводить из командной строки. Существует три способа ввода матриц. Например, матрицу

$$A = \begin{bmatrix} 0.7 & -2.5 & 9.1 \\ 8.4 & 0.3 & 1.7 \\ -3.5 & 6.2 & 4.7 \end{bmatrix}$$

можно ввести следующим образом: набрать в командной строке (разделяя элементы строки матрицы пробелами): $A=[0.7 \ -2.5 \ 9.1$ и нажать **<Enter>**. Курсор перемещается в следующую строку (символ приглашения командной строки **>>** отсутствует). Элементы каждой следующей строки матрицы **набираются через пробел**, а **ввод** строки завершается **нажатием на <Enter>**. При вводе последней строки в конце ставится закрывающая квадратная скобка:

```
>> A=[0.7 -2.5 9.1  
8.4 0.3 1.7  
-3.5 6.2 4.7]
```

Если после закрывающей квадратной скобки не ставить точку с запятой для подавления вывода в командное окно, то матрица выведется в виде таблицы.

Другой способ ввода матрицы основан на том, что матрицу можно **рассматривать как вектор-столбец**, каждый элемент которого **является строкой матрицы**. Поскольку **точка с запятой** используется для разделения элементов вектор-столбца, то ввод, к примеру, матрицы

$$B = \begin{bmatrix} 6.1 & 0.3 \\ -7.9 & 4.4 \\ 2.5 & -8.1 \end{bmatrix}$$

осуществляется оператором присваивания:

```
>> B=[6.1 0.3; -7.9 4.4; 2.5 -8.1];
```

Ведите матрицу B и отобразите ее содержимое в командном окне, набрав в командной строке B и нажав **<Enter>**.

Очевидно, что допустима такая трактовка матрицы, при которой она считается вектор-строкой, каждый элемент которой является столбцом матрицы. Следовательно, для ввода матрицы

$$C = \begin{bmatrix} 0.4 & -7.2 & 5.3 \\ 0.1 & -2.1 & -9.5 \end{bmatrix}$$

достаточно воспользоваться командой:

```
>> C=[[0.4; 0.1] [-7.2; -2.1] [5.3; -9.5]]
```

Обратите внимание, что внутренние квадратные скобки действительно нужны. Оператор $C=[0.4; 0.1 \ -7.2; \ -2.1 \ 5.3; \ -9.5]$ является недопустимым и приводит к сообщению об ошибке, поскольку оказывается, что в первой строке матрицы содержится только один элемент, во второй и третьей — по два, а в четвертой — снова один.

Воспользуйтесь командой **whos** для получения информации о переменных *A*, *B* и *C* рабочей среды. В командное окно выводится таблица с информацией о размерах массивов, памяти, необходимой для хранения каждого из массивов, и типе — **double array**:

```
>> whos A B C
Name  Size Bytes Class
A  3x3  72 double array
B  3x2  48 double array
C  2x3  48 double array
```

Функция **size** позволяет установить размеры массивов, она возвращает результат в виде вектора, первый элемент которого равен числу строк, а второй — столбцов:

```
>> s=size(B)
s =
3 2
```

Сложение и вычитание матриц одинаковых размеров производится с использованием знаков +, -.

Звездочка * служит для вычисления **матричного произведения**, причем соответствующие размеры матриц должны совпадать, например:

```
>> P=A*B
P =
46.7700 -84.5000
53.1200 -9.9300
-58.5800 -11.8400
```

Допустимо **умножение матрицы на число** и числа на матрицу, при этом происходит умножение каждого элемента матрицы на число и результатом является матрица тех же размеров, что и исходная.

Апостроф ' предназначен для **транспонирования** вещественной матрицы или нахождения сопряженной к комплексной матрице. Для **возведения** квадратной матрицы в степень применяется знак ^.

Вычислите для тренировки матричное выражение $R = (A - BC)^3 + ABC$, в котором *A*, *B* и *C* — определенные выше матрицы. Ниже приведена запись в MatLab этого выражения:

```
>> R=(A-B*C)^3+A*B*C
R =
1.0e+006 *
-0.0454 0.1661 -0.6579
0.0812 -0.2770 1.2906
-0.0426 0.1274 -0.7871
```

MatLab обладает многообразием различных функций и способов для работы с матричными данными.

Для **обращение к элементу двумерного массива** следует указать его строчный и столбцевой индексы в круглых скобках после имени массива, например:

```
>> C(1,2)
```

```
ans =
-7.2000
```

Индексация двоеточием позволяет **получить часть матрицы — строку, столбец или блок**, например:

```
>> c1=A(2:3,2)
c1 =
0.3000
6.2000
>> r1=A(1,1:3)
r1 =
0.7000 -2.5000 9.1000
```

Для **обращения ко всей строке или всему столбцу** не обязательно указывать через двоеточие начальный (первый) и конечный индексы, то есть операторы `r1=A(1,1:3)` и `r1=A(1,:)` эквивалентны.

Для доступа к элементам строки или столбца **от заданного до последнего** можно использовать `end`, так же как и для векторов: `A(1,2:end)`. Выделение блока, состоящего из нескольких строк и столбцов, требует индексации двоеточием как по первому измерению, так и по второму. Пусть в массиве T хранится матрица:

$$T = \begin{bmatrix} 1 & 7 & -3 & 2 & 4 & 9 \\ 0 & -5 & -6 & 3 & 8 & 7 \\ 2 & 4 & 5 & -1 & 0 & 3 \\ -6 & -4 & 7 & 2 & 6 & 1 \end{bmatrix}$$

Для выделения ее элементов (обозначенных курсивом) со второй строки по третью и со второго столбца по четвертый, достаточно использовать оператор:

```
>> T1=T(2:3,2:4)
T1 =
-5 -6 3
4 5 -1
```

Индексация двоеточием так же очень полезна при различных перестановках в массивах. В частности, для **перестановки первой и последней строк** в произвольной матрице, хранящейся в массиве A , подойдет последовательность команд:

```
>> s=A(1,:);
>> A(1,:)=A(end,:);
>> A(end,:)=s;
```

MatLab поддерживает такую операцию, как **вычеркивание строк или столбцов** из матрицы. Достаточно удаляемому блоку присвоить пустой массив, задаваемый квадратными скобками. Например, вычеркивание второй и третьей строки из массива T , введенного выше, производится следующей командой:

```
>> T(2:3,:)=[]
T =
```

```

1 7 -3 2 4 9
-6 -4 7 2 6 1

```

Индексация двоеточием упрощает заполнение матриц, имеющих определенную структуру. Предположим, что требуется создать матрицу

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Первый шаг состоит в определении нулевой матрицы размера пять на пять, затем заполняются первая и последняя строки и первый и последний столбцы:

```

>> W(1:5,1:5)=0;
>> W(1,:)=1;
>> W(end,:)=1;
>> W(:,1)=1;
>> W(:,end)=1;

```

Ряд встроенных функций, приведенных в таблице 2.1, позволяет ввести стандартные матрицы заданных размеров. Обратите внимание, что во всех функциях, кроме **diag**, допустимо указывать размеры матрицы следующими способами:

- числами через запятую (в двух входных аргументах);
- одним числом, результат — квадратная матрица;
- вектором из двух элементов, равных числу строк и столбцов.

Таблица 2.1. Функции для создания стандартных матриц

Функция	Результат и примеры вызовов
<code>zeros</code>	Нулевая матрица <code>F=zeros(4,5)</code> <code>F=zeros(3)</code> <code>F=zeros([3 4])</code>
<code>eye</code>	Единичная прямоугольная матрица (единицы расположены на главной диагонали) <code>I=eye(5,8)</code> <code>I=eye(5)</code> <code>I=eye([5 8])</code>
<code>ones</code>	Матрица, целиком состоящая из единиц <code>E=ones(3,5)</code> <code>E=ones(6)</code> <code>E=ones([2 5])</code>
<code>rand</code>	Матрица, элементы которой — случайные числа, равномерно распределенные на интервале (0,1) <code>R=rand(5,7)</code> <code>R=rand(6)</code> <code>R=rand([3 5])</code>
<code>randn</code>	Матрица, элементы которой — случайные числа, распределенные по нормальному закону с нулевым средним и дисперсией равной единице <code>N=randn(5,3)</code> <code>N=randn(9)</code> <code>N=randn([2 4])</code>
<code>diag</code>	1) диагональная матрица, элементы которой задаются во входном аргументе — векторе <code>D=diag(v)</code>

- | | |
|--|---|
| | <p>2) диагональная матрица со смещенной на k позиций диагональю (положительные k — смещение вверх, отрицательные — вниз), результатом является квадратная матрица размера <code>length(v)+abs(k)</code>
 <code>D=diag(v,k)</code></p> <p>3) выделение главной диагонали из матрицы в вектор
 <code>d=diag(A)</code></p> <p>4) выделение k-ой диагонали из матрицы в вектор
 <code>d=diag(A,k)</code></p> |
|--|---|

Данные функции очень удобны, когда требуется создать стандартную матрицу тех же размеров, что и некоторая имеющаяся матрица. Если, к примеру, A была определена ранее, то команда `I=eye(size(A))` приводит к появлению единичной матрицы, размеры которой совпадают с размерами A , так как функция `size` возвращает размеры матрицы в векторе.

Разберем, как получить трехдиагональную матрицу размера семь на семь, приведенную ниже, с использованием функций MatLab.

$$T = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 5 & 3 & -3 & 0 & 0 & 0 \\ 0 & 0 & 5 & 4 & -4 & 0 & 0 \\ 0 & 0 & 0 & 5 & 5 & -5 & 0 \\ 0 & 0 & 0 & 0 & 5 & 6 & -6 \\ 0 & 0 & 0 & 0 & 0 & 5 & 7 \end{bmatrix}$$

Ведите вектор v с целыми числами от одного до семи $v=[1\ 2\ 3\ 4\ 5\ 6\ 7]$ и используйте его для создания диагональной матрицы и матрицы со смещенной на единицу вверх диагональю. Вектор длины шесть, содержащий пятерки, заполняется, например, так: `5*ones(1,6)`. Этот вектор укажите в первом аргументе функции `diag`, а минус единицу — во втором и получите третью вспомогательную матрицу. Теперь достаточно вычесть из первой матрицы вторую и сложить с третьей:

```
>> T=diag(v)-diag(v(1:6),1)+diag(5*ones(1,6),-1)
```

Поэлементные вычисления с матрицами производятся практически аналогично, разумеется, необходимо следить за совпадением размеров матриц:

$A.*B$, $A./B$ — поэлементные умножение и деление;

$A.^p$ — поэлементное возведение в степень, p — число;

$A.^B$ — возведение элементов матрицы A в степени, равные соответствующим элементам матрицы B ;

$A.'$ — транспонирование матрицы (для вещественных матриц A' и $A.^{'}$ приводят к одинаковым результатам);

Иногда требуется не просто транспонировать матрицу, но и "развернуть" ее. **Вращение матрицы** на 90° против часовой стрелки осуществляет функция **rot90**:

```
>> Q=[1 2;3 4]
Q =
1 2
3 4
>> R=rot90(Q)
R =
2 4
1 3
```

Допустимо записывать сумму и разность матрицы и числа, при этом сложение или вычитание применяется, соответственно, ко всем элементам матрицы. Вызов математической функции от матрицы приводит к матрице того же размера, на соответствующих позициях которой стоят значения функции от элементов исходной матрицы.

В MatLab определены и матричные функции, например, **sqrtm** предназначена для вычисления квадратного корня. Найдите квадратный корень из матрицы

$$K = \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}$$

и проверьте полученный результат, возведя его в квадрат (по правилу матричного умножения, а не поэлементно!):

```
>> K=[3 2; 1 4];
>> S=sqrtm(K)
S =
1.6882 0.5479
0.2740 1.9621
>> S*S
ans =
3.0000 2.0000
1.0000 4.0000
```

Матричная экспонента вычисляется с использованием **expm**. Специальная функция **funm** служит для вычисления произвольной матричной функции.

Все функции обработки данных могут быть применены и к двумерным массивам. Основное отличие от обработки векторных данных состоит в том, что эти функции работают с двумерными массивами по столбцам, например, функция **sum** суммирует элементы каждого из столбцов и возвращает вектор-строку, длина которой равна числу столбцов исходной матрицы:

```
>> M=[1 2 3; 4 5 6; 7 8 9]
M =
1 2 3
4 5 6
7 8 9
```

```
>> s=sum(M)
s =
12 15 18
```

Если в качестве второго входного аргумента **sum** указать 2, то суммирование произойдет по строкам.

Для вычисления суммы всех элементов матрицы требуется дважды применить **sum**:

```
>> s=sum(sum(M))
s =
45
```

Очень удобной возможностью MatLab является **конструирование матрицы из матриц** меньших размеров. Пусть заданы матрицы:

$$M_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad M_2 = \begin{bmatrix} -2 & -3 & -5 \\ -1 & -5 & -6 \end{bmatrix}, \quad M_3 = \begin{bmatrix} 9 & 8 \\ -7 & -5 \\ 1 & 2 \end{bmatrix}, \quad M_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Требуется составить из M_1 , M_2 , M_3 и M_4 блочную матрицу M

$$M = \begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix}$$

Можно считать, что M имеет размеры два на два, а каждый элемент является, соответственно, матрицей M_1 , M_2 , M_3 или M_4 . Следовательно, для получения в рабочей среде MatLab массива M с матрицей M требуется использовать оператор:

```
>> M=[M1 M2; M3 M4]
```

Многомерные вычисления в MatLab.

Построение графиков поверхностей

Цель работы:

Изучить правила организации вложенных циклов, правила получения многомерных результатов, вывод многомерных данных в табличной форме, использование объемной графики, контурной график.

В работе предусмотрены две задачи, в каждой из которых вычисляется двумерная функция, описывающая объемную фигуру, и строятся поверхности и контурные графики с использованием различных графических функций. В первой задаче каждый график выводится в свое окно, во второй в подокна общего окна.

Для вывода графиков $y=f(x)$ в отдельное окно используется функция $plot(x,y)$, где x и y – векторы, одинаковой размерности:

Например, построение дуги окружности:

```
x1=2; r1=5;
x=[x1-r1:0.01:x1+r1];
y01=y1+sqrt(r1*r1-(x-x1).^2);
plot(x,y01,'b');
```

Тип линии, цвет и маркеры определяются значением третьего дополнительного аргумента функции plot. Этот аргумент указывается в апострофах, например, вызов plot(x,f,'ro:') приводит к построению графика красной пунктирной линией, размеченной круглыми маркерами. Обратите внимание, что абсциссы маркеров определяются значениями элементов вектора x.

Всего в дополнительном аргументе может быть заполнено три позиции, соответствующие цвету, типу маркеров и стилю линии. Обозначения для них приведены в следующей таблице. Порядок позиций может быть произвольный, допустимо указывать только один или два параметра, например, цвет и тип маркеров. Посмотрите на результат выполнения следующих команд: plot(x,f,'g'), plot(x,f,'ko'), plot(x,f,:').

Обозначения для цвета, типа маркеров и стиля линий

Цвет		Тип маркера	
y	Желтый	.	Точка
m	Розовый	o	Кружок
c	Голубой	x	Крестик
r	Красный	+	Знак плюс
g	Зеленый	*	Звездочка
b	Синий	s	Квадрат
w	Белый	d	Ромб
k	Черный	v	Треугольник вершиной вниз
Тип линии		^	Треугольник вершиной вверх
-	Сплошная	<	Треугольник вершиной влево
:	Пунктирная	>	Треугольник вершиной вправо
-.	Штрих-пунктирная	p	Пятиконечная звезда

Для вывода графиков нескольких функций в одно окно используется функция задержки вывода изображения в окно:

```
% задержать вывод изображения
hold on
% нарисовать черный маркер-звездочка в точке
x0=0;
y0=0;
plot(x0,y0, 'k*' );
% нарисовать окружность синим цветом
x1=1;
y1=-1;
r1=3;
x=[x1-r1:0.01:x1+r1];
y01=y1+sqrt(r1*r1-(x-x1).^2);
y02=y1-sqrt(r1*r1-(x-x1).^2);
plot(x,y01, 'b' );
plot(x,y02, 'b' );
axis square
hold off
```

Для вывода графиков в различные окна (например, вывод в окно с номером 2) используются следующие команды:

```
figure(2);  
plot(x,y);
```

Вывод графиков в отдельные подокна общего окна выполняется следующим образом. Например, необходимо вывести график во второе подокно общего окна, имеющего 3x3 подокон:

```
subplot(3,3,2),plot(x,y);
```

Для формирования поверхностного или контурного графика необходимо:

- задать число N точек графика по координатам X и Y, (для всех вариантов N=40, шаг изменения равен $h=4 \pi/N$,
- создать вложенные циклы по X и Y, вычислить функцию $Z=f(X,Y)$,
- подготовить данные для прорисовки основания графика поверхности Z

```
[X,Y]=meshgrid([-N:1:N]);
```

- ввести номер графического окна, вывести туда график выбранного типа.

При вычислении точек поверхностей следует использовать графики:

- трехмерный с аксонометрией, функция plot3(X,Y,Z),
- трехмерный с функциональной окраской, функция mesh(X,Y,Z),
- трехмерный с функциональной окраской и проекцией, функция meshc(X,Y,Z),
- трехмерный с функциональной окраской и проекцией, функция surf(X,Y,Z),
контурный, функция contour(X,Y,Z),
- объемный контурный, функция contour3 (X,Y,Z),
- трехмерный с освещением, функция surfl (X,Y,Z).

В каждом окне можно рисовать несколько графиков с наложением друг на друга. В списке параметров для каждого графика параметры перечисляются группами последовательно (в работе график для окна один). В каждую группу входят:

- X - первая координата площадки основания,
- Y - вторая координата площадки основания,
- Z - значение функции.

Пример выполнения

Пусть задана функция:

$$z = \frac{\sin(x)}{x} \cdot \frac{\sin(y)}{y}$$

Пределы изменения аргументов x и y: $-2\pi \dots 2\pi$.

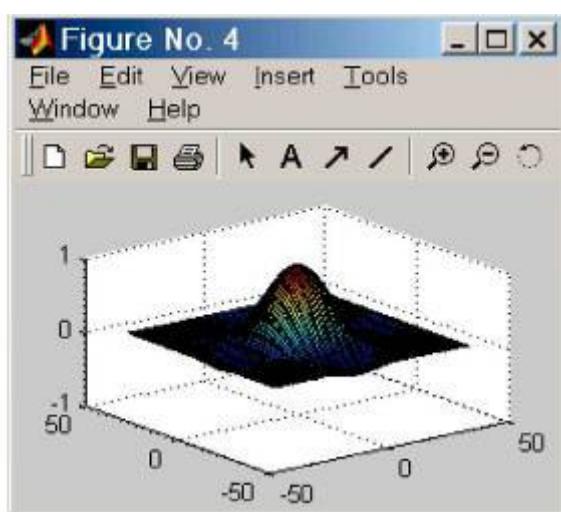
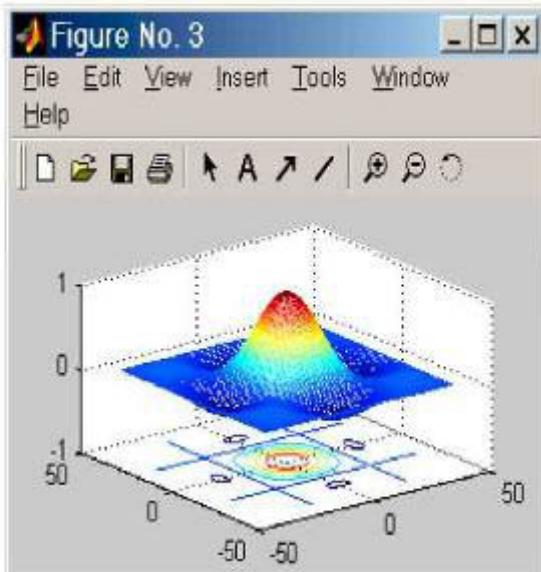
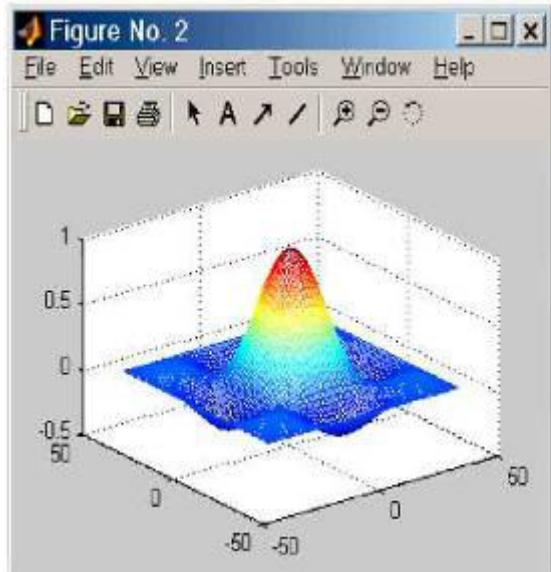
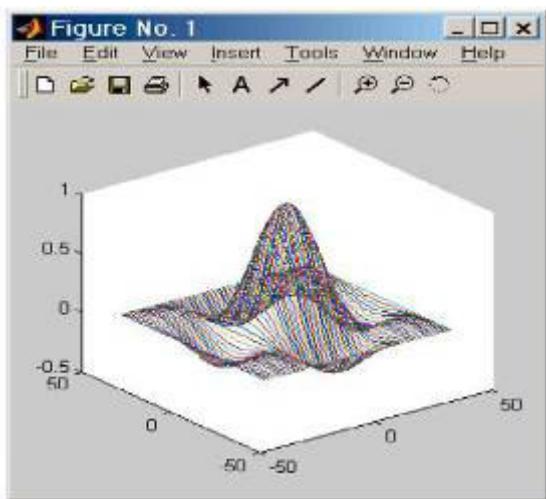
Задание 1

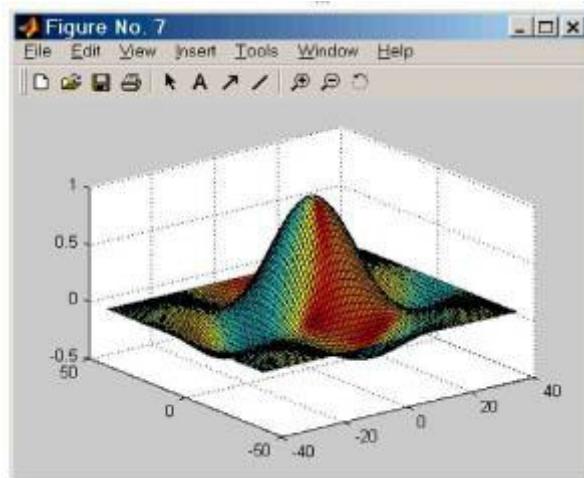
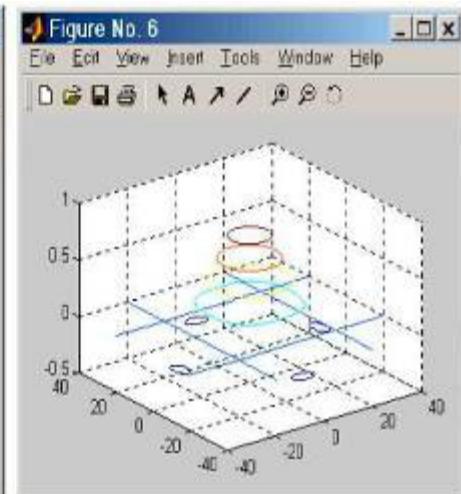
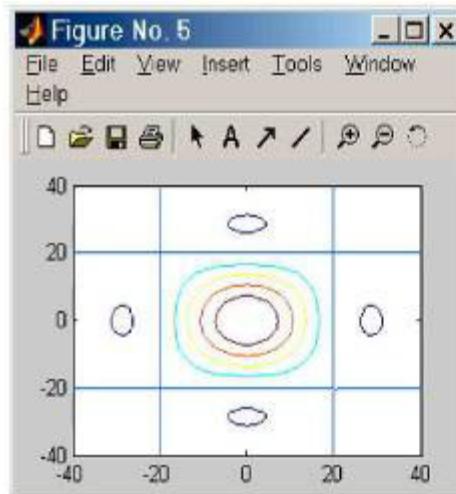
```

% Число точек и шаг
N=40;
h=pi/20;
% Расчет матрицы
for n=1:2*N+1
    if n==N+1 A(n)=1; else A(n)=sin(h*(n-N-1))/(h*(n-N-1));
    end;
end;
for n=1:2*N+1
    for m=1:2*N+1
        Z(n,m)=A(n)*A(m);
    end;
end;
% Задание площадки
[X,Y]=meshgrid([-N:1:N]);
% Вывод графика в аксонометрии в окно 1
figure(1);
plot3(X,Y,Z);
% вывод трехмерного графика с функциональной окраской в окно 2
figure(2);
mesh(X,Y,Z);
%вывод трехмерного графика с функциональной окраской и проекцией в окно 3
figure(3);
meshc(X,Y,Z);
% вывод трехмерного графика с проекцией в окно 4
figure(4);
surf(X,Y,Z);
% Вывод контурного графика в окно 5
figure(5);
contour(X,Y,Z)
% Вывод объемного контурного графика в окно 6
figure(6);
contour3(X,Y,Z);
% Вывод объемного графика с освещением в окно 7
figure(7);
surfl(X,Y,Z)

```

Результат выполнения задания 1 представлен в виде следующих окон:



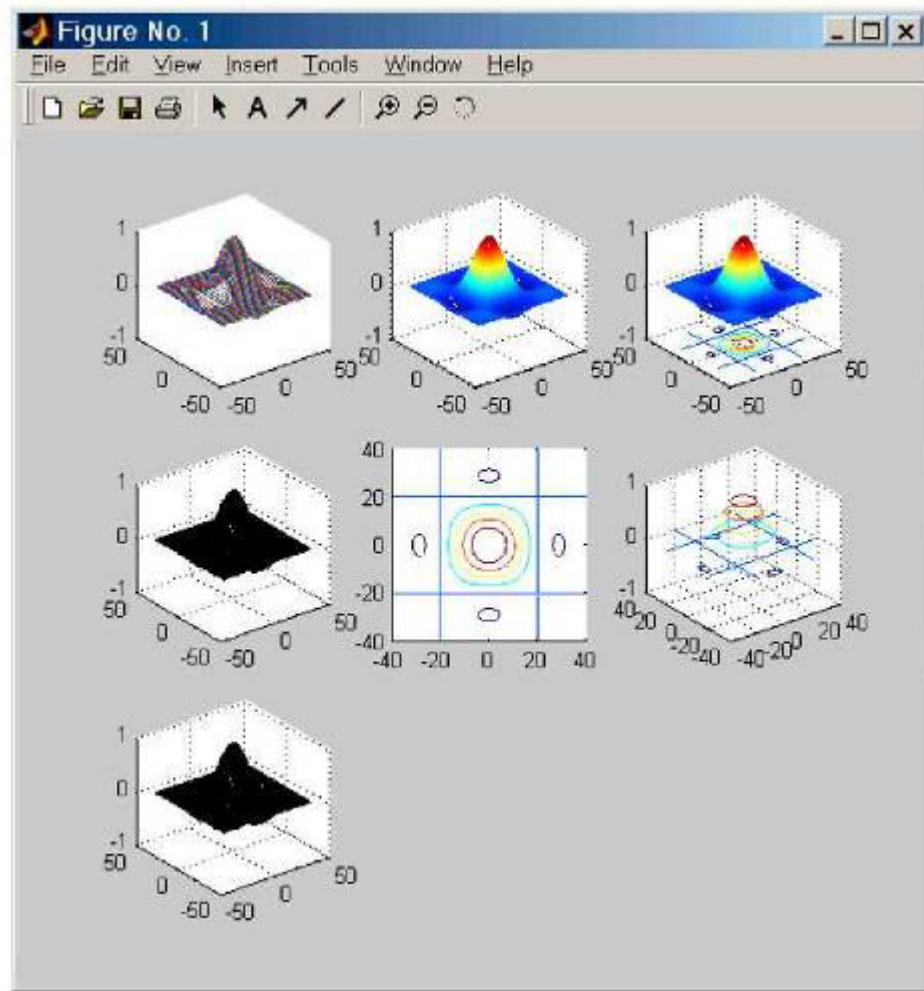


Задание 2

```
% Задача 2
% Число точек и шаг
N=40;
h=pi/20;
% Расчет матрицы
for n=1:2*N+1
    if n==N+1 A(n)=1; else A(n)=sin(h*(n-N-1))/(h*(n-N-1));
    end;
end;
for n=1:2*N+1
    for m=1:2*N+1
        Z(n,m)=A(n)*A(m);
    end;
end;
% Задание площадки
[X, Y]=meshgrid([-N:1:N]);
% Вывод графика в аксонометрии в подокно 1
subplot(3,3,1),plot3(X,Y,Z);
% вывод трехмерного графика с функциональной окраской в подокно 2
subplot(3,3,2),mesh(X,Y,Z);
% вывод трехмерного графика с функциональной окраской и проекцией в
подокно 3
subplot(3,3,3),meshc(X,Y,Z);
% вывод трехмерного графика с проекцией в подокно 4
subplot(3,3,4),surf(X,Y,Z);
```

```
% Вывод контурного графика в подокно 5
subplot(3,3,5), contour(X,Y,Z);
% Вывод объемного контурного графика в подокно 6
subplot(3,3,6), contour3(X,Y,Z);
% Вывод объемного графика с освещением в подокно 7
subplot(3,3,7), surfl(X,Y,Z);
```

Результат выполнения задания 2 представлен в виде следующих окон:



Общая постановка задачи

В результате выполнения заданий лабораторной работы студенты **должны уметь** создавать программно-алгоритмическую поддержку для компьютерной реализации требуемых методов преобразования матриц в MatLab.

Список индивидуальных данных

В лабораторную работу входит 4 задания.

Задание 1

При помощи встроенных функций для заполнения стандартных матриц, индексации двоеточием и, возможно, поворота, транспонирования или вычеркивания получите следующие матрицы:

Варианты:

$$1. \begin{bmatrix} 2 & 3 & 0 & 0 & 0 & 0 & 5 \\ -1 & 3 & 3 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 & 3 & 0 & 0 \\ 0 & 0 & 0 & -1 & 6 & 3 & 0 \\ 0 & 0 & 0 & 0 & -1 & 7 & 3 \\ 5 & 0 & 0 & 0 & 0 & -1 & 8 \end{bmatrix}$$

$$3. \begin{bmatrix} 4 & 3 & 3 & 3 & 3 & 3 & 3 & 9 \\ 3 & 4 & 3 & 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 4 & 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 4 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 4 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 4 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 4 & 3 \\ 9 & 3 & 3 & 3 & 3 & 3 & 3 & 4 \end{bmatrix}$$

$$5. \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$7. \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 1 \\ 2 & 0 & 0 & 0 & 7 & 0 & 1 \\ 3 & 0 & 0 & 7 & 0 & 9 & 1 \\ 4 & 0 & 7 & 0 & 9 & 0 & 1 \\ 5 & 7 & 0 & 9 & 0 & 0 & 1 \\ 6 & 0 & 9 & 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 1 \end{bmatrix}$$

$$9. \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 5 \\ -1 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 6 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 7 & 1 \\ 5 & 0 & 0 & 0 & 0 & -1 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 1 \\ 2 & 0 & 0 & 0 & 8 & 0 & 1 \\ 3 & 0 & 0 & 8 & 0 & 7 & 1 \\ 4 & 0 & 8 & 0 & 7 & 0 & 1 \\ 5 & 8 & 0 & 7 & 0 & 0 & 1 \\ 6 & 0 & 7 & 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 1 \end{bmatrix}$$

$$4. \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 1 \\ 2 & 0 & 0 & 0 & 7 & 0 & 1 \\ 3 & 0 & 0 & 7 & 0 & 7 & 1 \\ 4 & 0 & 7 & 0 & 7 & 0 & 1 \\ 5 & 7 & 0 & 7 & 0 & 0 & 1 \\ 6 & 0 & 7 & 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 1 \end{bmatrix}$$

$$6. \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & -1 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & -1 & 0 & 2 \\ 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$8. \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 0 & 0 & 0 & 7 & 1 \\ 1 & 0 & 2 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$10. \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Задание 2

Сконструировать блочные матрицы (используя функции для заполнения стандартных матриц) и применить функции обработки данных и поэлементные операции для нахождения заданных величин.

Варианты:

$$1. \quad A = \begin{bmatrix} 1 & 2 & 3 & -1 & 0 & 0 \\ 1 & 2 & 3 & 0 & -1 & 0 \\ 1 & 2 & 3 & 0 & 0 & -1 \\ 0 & 0 & 7 & 2 & 2 & 2 \\ 0 & 7 & 0 & 2 & 2 & 2 \\ 7 & 0 & 0 & 2 & 2 & 2 \end{bmatrix}, \quad m = \max_{i=1,2,\dots,6} \min_{j=1,2,\dots,6} a_{ij}$$

$$2. \quad A = \begin{bmatrix} 1 & 0 & 0 & 1 & -3 & -3 \\ 0 & 1 & 1 & 0 & -3 & -3 \\ 0 & 1 & 1 & 0 & -3 & -3 \\ 1 & 0 & 0 & 1 & -3 & -3 \\ -2 & -2 & -2 & -2 & 4 & 4 \\ -2 & -2 & -2 & -2 & 4 & 4 \end{bmatrix}, \quad m = \sum_{i=1}^6 \max_{j=1,2,\dots,6} (a_{ij} + a_{ji})$$

$$3. \quad A = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ -1 & -1 & -1 & 4 & 0 & 0 \\ -1 & -1 & -1 & 0 & 4 & 0 \\ -1 & -1 & -1 & 0 & 0 & 4 \end{bmatrix}, \quad m = \prod_{i=1}^6 \sum_{j=1}^6 (a_{ij})^2$$

$$4. \quad A = \begin{bmatrix} 1 & 1 & -3 & -3 & -3 & -3 \\ 1 & 1 & -3 & -3 & -3 & -3 \\ -3 & -3 & 2 & 0 & 0 & 0 \\ -3 & -3 & 0 & 2 & 0 & 0 \\ -3 & -3 & 0 & 0 & 2 & 0 \\ -3 & -3 & 0 & 0 & 0 & 2 \end{bmatrix}, \quad m = \sum_{i=1}^5 a_{ii+1}$$

$$5. \quad A = \begin{bmatrix} -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 0 & 4 & 4 \end{bmatrix}, \quad m = \sum_{j=1}^6 a_{jj} + \sum_{i=1}^5 a_{ii+1}$$

$$6. \quad A = \begin{bmatrix} 1 & 0 & 0 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 4 & 0 \\ 0 & 0 & 1 & 0 & 0 & 4 \\ 2 & 2 & 2 & 3 & 3 & 3 \\ 2 & 2 & 2 & 3 & 3 & 3 \\ 2 & 2 & 2 & 3 & 3 & 3 \end{bmatrix}, \quad m = \max_{j=1,2,\dots,6} \left\{ \sum_{i=1}^6 a_{ij}^2 \right\}$$

$$7. \quad A = \begin{bmatrix} 2 & 2 & -1 & -1 & 2 & 2 \\ 2 & 2 & -1 & -1 & 2 & 2 \\ 2 & 2 & -1 & -1 & 2 & 2 \\ 2 & 2 & -1 & -1 & 2 & 2 \\ 2 & 2 & -1 & -1 & 2 & 2 \\ 2 & 2 & -1 & -1 & 2 & 2 \end{bmatrix}, \quad m = \sum_{i=1}^6 \sum_{j=1}^6 |a_{ij}|$$

$$8. \quad A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ -1 & -2 & -3 & -4 & -5 & -6 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}, \quad m = \min_{i,j=1,2,\dots,6} a_{ij}^3$$

$$9. \quad A = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 3 & 0 & 0 \\ 2 & 2 & 2 & 0 & 3 & 0 \\ 2 & 2 & 2 & 0 & 0 & 3 \end{bmatrix}, \quad m = \sum_{k=1}^6 a_{kk}^3$$

$$10. \quad A = \begin{bmatrix} 0 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 3 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 & 0 & 4 \\ 3 & 0 & 0 & 0 & 0 & 4 \\ 1 & 1 & 1 & 1 & 1 & -2 \end{bmatrix}, \quad m = \sum_{i=1}^6 \sum_{j=1}^6 \sin\left(\frac{\pi}{6} a_{ij}^2\right)$$

Задание 3

Пусть модель поведения объекта с параметрами, заданными в матрице А, описывается функцией $f(x)$. Необходимо рассчитать параметры модели, а именно:

Вычислить значения функции для всех элементов матрицы и записать результат в матрицу того же размера, что и исходная.

Варианты:

$$1. \quad f(x) = e^{x^2+x+1}; \quad A = \begin{bmatrix} -4.53 & -2.12 & -6.54 & -3.21 \\ 3.43 & 7.43 & -0.25 & 1.64 \end{bmatrix}$$

$$2. f(x) = \frac{\sqrt[3]{x^2 + 1}}{|x| + 3}; \quad A = \begin{bmatrix} 0.23 & 3.89 & -4.23 & -7.25 \\ 5.84 & 5.13 & -0.89 & 3.55 \end{bmatrix}$$

$$3. f(x) = x^3 - 2x^2 + \sin x - 4; \quad A = \begin{bmatrix} 9.33 & -4.01 & 8.19 & 2.64 \\ 0.55 & 3.81 & 3.32 & 5.07 \end{bmatrix}$$

$$4. f(x) = \frac{e^x - x}{e^x + x}; \quad A = \begin{bmatrix} 9.32 & 0.21 & -9.89 & 3.11 \\ 0.54 & 4.99 & 5.01 & -0.03 \end{bmatrix}$$

$$5. f(x) = \frac{x^3 + \sin x}{x^3 - \cos x}; \quad A = \begin{bmatrix} 0.64 & 6.34 & 0.32 & -4.23 \\ 1.19 & 3.23 & 1.54 & 0.43 \end{bmatrix}$$

$$6. f(x) = \arcsin(\cos x^2); \quad A = \begin{bmatrix} \pi & 2.2\pi & -2\pi & 0.3\pi \\ 3\pi & -\pi & 0.1\pi & 5\pi \end{bmatrix}$$

$$7. f(x) = \sqrt{1 + \sqrt{|x|^3 + 1}}; \quad A = \begin{bmatrix} -1.54 & 0.49 & 3.11 & 2.99 \\ 4.05 & -5.85 & 3.72 & 0.11 \end{bmatrix}$$

$$8. f(x) = e^x \sin x - e^{-x} \cos x; \quad A = \begin{bmatrix} -9.04 & 3.36 & 3.09 & -2.49 \\ -4.33 & -5.09 & 9.74 & 1.65 \end{bmatrix}$$

$$9. f(x) = \ln(|x|) \sin \pi x; \quad A = \begin{bmatrix} 0.33 & 0.95 & 7.12 & -9.22 \\ -0.64 & 3.76 & 1.34 & -0.33 \end{bmatrix}$$

$$10. f(x) = 1 + \frac{1+x}{1-x^2}; \quad A = \begin{bmatrix} -5.84 & 9.84 & 0.23 & 1.59 \\ -9.25 & -0.25 & 1.54 & 0.43 \end{bmatrix}$$

Задание 4

1. Вычисление двумерной функции и построение ее объемных графиков в отдельных окнах:

- Ввести исходные данные.
- Вычислить двумерную функцию.
- Вывести функцию в виде 5 трехмерных графиков разного типа.
- Вывести функцию в виде 2 контурных графиков разного типа.

2. Вычисление двумерной функции и построение ее объемных графиков в подокнах общего окна.

Варианты:

№	Функция	Пределы изменения	
		x	y
1	$z = \sin(x)\cos(y)$	от -2π до 2π	от -2π до 2π
2	$z = \sin(x/2)\cos(y)$	от -2π до 2π	от -2π до 2π
3	$z = \sin(2x)\cos(y)$	от -2π до 2π	от -2π до 2π
4	$z = \sin(x)\cos(y/2)$	от -2π до 2π	от -2π до 2π
5	$z = \sin(x/2)\cos(2y)$	от -2π до 2π	от -2π до 2π
6	$z = \sin(2x)\cos(2y)$	от -2π до 2π	от -2π до 2π
7	$z = (1+\sin(x)/x)(\sin(y)/y)$	от -2π до 2π	от -2π до 2π

8	$z = (\sin(x)/x)\cos(y)$	от -2π до 2π	от -2π до 2π
9	$z = (\sin(x)/x) \cos(y) $	от -2π до 2π	от -2π до 2π
10	$z = (\sin(x)/x)y$	от -2π до 2π	от -2π до 2π
11	$z = (\sin(x)/x) y $	от -2π до 2π	от -2π до 2π
12	$z = (\sin(x)/x)\sin(y)$	от -2π до 2π	от -2π до 2π
13	$z = (\sin(x)/x) \sin(y) $	от -2π до 2π	от -2π до 2π
14	$z = (\sin(x)/x)(1-y)$	от -2π до 2π	от -2π до 2π
15	$z = (\sin(x)/x) y+0.5 $	от -2π до 2π	от -2π до 2π

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.

Лабораторная работа №3. Разработка графического пользовательского интерфейса GUI в среде Matlab.

Цель работы: Целью лабораторной работы является получение навыков самостоятельной программной реализации на компьютерной технике визуальных компонентов графического пользовательского интерфейса в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-10	3-1	Понимает необходимость саморазвития
	3-2	Выражает желание повышать свою квалификацию и мастерство
ПК-1	3-1	Знает основы современных технологий сбора, обработки и представления информации
	3-2	Способен ориентироваться в информационном потоке в глобальных

		компьютерных сетях
--	--	--------------------

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-1	У-1	Знает основы современных технологий сбора, обработки и представления информации
	У-2	Способен ориентироваться в информационном потоке в глобальных компьютерных сетях
ПК-5	У-1	Знает методы оценки проектов по информатизации
	У-2	Знает методы оценки проектов по автоматизации решения прикладных задач

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-5	B-1	Знает методы оценки проектов по информатизации
	B-2	Знает методы оценки проектов по автоматизации решения прикладных задач

Теоретическая часть

Matlab – это система инженерных и научных вычислений. Она обеспечивает математические вычисления, визуализацию научной графики программирование и моделирование процессов с использованием интуитивно понятной среды окружения, когда задачи и их решения могут быть представлены в нотации, близкой к математической.

Среда системы Matlab это совокупность интерфейсов, через которые пользователь поддерживает связь этой системой.

Пользовательский интерфейс носит дружественный характер и построен с учетом устоявшихся принципов программного обеспечения, разрабатываемого для операционной системы Windows.

В процессе создания больших программ возникает необходимость многократного запуска файла программы при других, изменённых параметрах решаемой задачи. Возникает неудобство: в постоянном редактировании исходного текста программы и повторном или очередном её запуске. При этом важен механизм управления переменными, который бы

обеспечивал удобный интерфейс между программой и пользователем. При решении других задач могут возникнуть трудности с визуализацией какого-либо процесса, то есть некоторая переменная изменяться динамически в процессе решения поставленной задачи.

Основные принципы построения графического интерфейса

Средства графического пользовательского интерфейса (GUI – Graphic User Interface) предназначены для создания в MATLAB приложений с пользовательским интерфейсом. В этих приложениях присутствуют управляющие элементы, при изменениях пользователем свойств которых вызываются подпрограммы, выполняющие некоторые действия. Использование графического интерфейса позволяет пользователю сделать программу более универсальной.

Как и любой процесс проектирования, процесс построения графического интерфейса пользователя можно разбить на следующие этапы:

1. Постановка задачи,
2. Создание формы интерфейса и создание на неё элементов управления.
3. Написание кода программы и кода обработки событий.

Этапы построения графического интерфейса пользователя:

1. На первом этапе проводиться анализ поставленной задачи и определяется количество и состав элементов управления необходимых для решения задачи.
2. На втором этапе создаётся форма графического интерфейса и на ней создаются и размещаются элементы управления. Здесь же описываются их свойства.

Для упрощения разработки GUI MATLAB включает инструмент GUIDE (Graphic User Interface development environment – среда разработки GUI).

Создание GUI

Для создания GUI используются два метода:

- Команда guide в командном окне.
- Выбор в меню MATLAB команды File=>New=>GUI.

В обоих случаях генерируется диалоговое окно создания GUI, которое содержит две вкладки:

- Creating New GUI – создание нового GUI.
- Open Existing GUI – открыть существующий GUI.

На вкладке создания нового GUI есть две панели: слева список шаблонов для выбора, справа предварительный просмотр. Можно выбрать один из предлагаемых шаблонов:

- Blank GUI (Default). Пустой GUI (по умолчанию)
- GUI with Uicontrols. GUI с элементами управления пользовательским интерфейсом.
- GUI with Axes and Menu. GUI с объектом Axes (координатные оси) и меню

- Modal Question Dialog. Модальное окно диалога с обязательным ответом на вопрос.

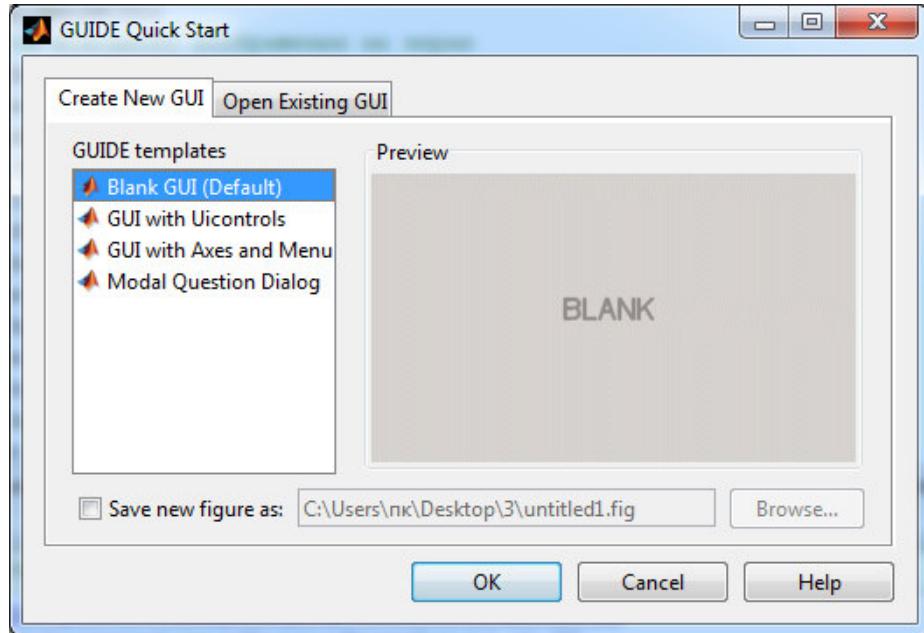


Рисунок 1 – Создание нового бланка графического интерфейса

Окно инструмента содержит поле, в котором размещаются компоненты. Доступные компоненты представлены в палитре, расположенной в правом поле.

Редактор GUI (Layout Editor)

Окно редактора GUI содержит:

- Меню.
- Панель инструментов.
- Набор доступных компонент слева.
- Рабочее поле (Layout Area) с координатной сеткой.

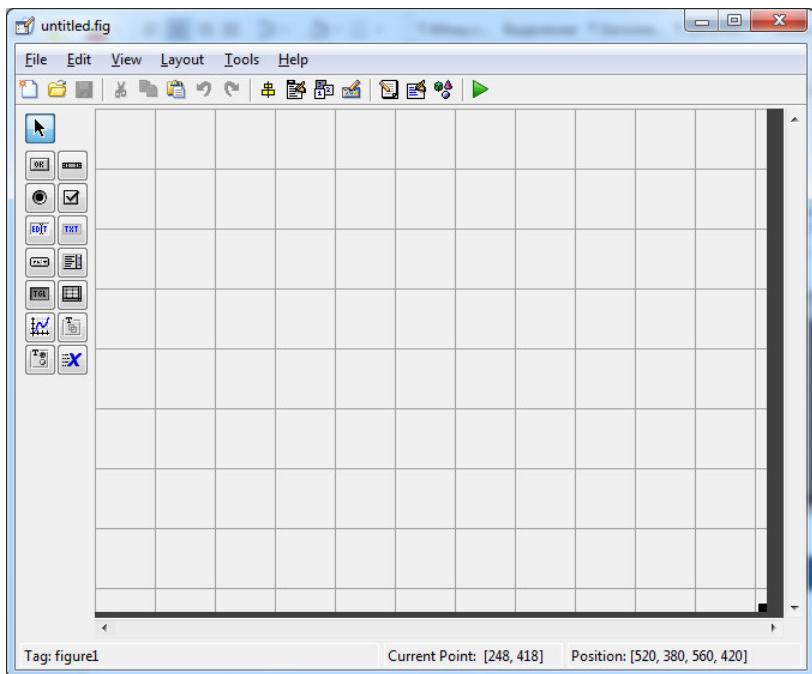


Рисунок 2 – Пустой бланк

При работе с GUI:

- Компоненты GUI выбираются из палитры компонент инструмента GUI мышью и переносятся в рабочую область GUI.
- GUI сохраняется в двух файлах (fig, m-файл).
- М-файл, управляющий работой GUI, генерируется автоматически при сохранении проекта.

Для создания GUI в MATLAB используются следующие компоненты:

Компонент	Назначение
Push Button	Кнопка. При нажатии отображается нажатой, при отпускании генерирует действие и отображается отжатой.
Toggle Button	Переключаемая кнопка. При нажатии отображается нажатой, при отпусканниии генерирует действие, но остается нажатой. Применяется для панелей инструментов.
Check Box	Независимый переключатель. Генерирует действие, когда нажат, и отображает включенное состояние птичкой в прямоугольнике. Используется в группах независимых переключателей для установки свойств объектов.
Radio Button	Зависимый переключатель. Генерирует действие, когда нажат, и отображает включенное состояние точкой в кружке. Похож на Check Box, но при использовании в группе при включении выключает остальные.
Edit Text	Редактор текста. Используется для ввода и редактирования строк текста. Действие генерируется при нажатии клавиши Enter для односторочного текста и Ctrl+Enter для многострочного текста.
Static Text	Статический текст. Используется для меток и для отображения значения Slider.
Slider	Ползунок. Отображает численные значения в выбранном диапазоне с выбор значения перемещением ползунка. Выбранное значение отображается в компоненте Static Text.

ListBox	Список тем для выбора. Позволяет выбрать одну или несколько тем.
Pop-Up Menu	Выпадающее меню (иначе ComboBox). Подобен ListBox, но список открывается только при выборе. Используется для экономии места.
Axes	Координатные оси. Используются для рисования в GUI графиков.
Panel	Панель. Это контейнер для группы компонент GUI. Панель используется для облегчения понимания назначения элементов управления путем их визуального объединения. Панель может иметь границы и заголовок. При перемещении панели ее дочерние компоненты тоже перемещаются.
Button Group	Группа кнопок. Подобна панели, но используется только для радиокнопок и переключаемых кнопок.
ActiveX Control	Это элементы управления, разработанные Microsoft для операционной системы Windows. Их можно использовать при работе в этой операционной системе. При переносе этого компонента в рабочую область вызывается список ActiveX элементов для выбора.

При сохранении проекта GUI сохраняются файл формы и m-файл. После каждого редактирования оба файла изменяются. Ниже листинг m-файла с пустой формой. В нем автоматически создаются строки исполняемого кода и комментарии.

```

function varargout = Lab_3(varargin)
% LAB_3 MATLAB code for Lab_3.fig
%     LAB_3, by itself, creates a new LAB_3 or raises the existing
%     singleton*.
%
%     H = LAB_3 returns the handle to a new LAB_3 or the handle to
%     the existing singleton*.
%
%     LAB_3('CALLBACK', hObject, eventData, handles,...) calls the local
%     function named CALLBACK in LAB_3.M with the given input
arguments.
%
%
% LAB_3('Property','Value',...) creates a new LAB_3 or raises the
% existing singleton*. Starting from the left, property value
pairs are
%
% applied to the GUI before Lab_3_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property
application
%
stop. All inputs are passed to Lab_3_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
one
%
instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Lab_3

% Last Modified by GUIDE v2.5 03-Oct-2015 13:08:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',         mfilename, ...
                   'gui_Singleton',    gui_Singleton, ...
                   'gui_OpeningFcn',   @Lab_3_OpeningFcn, ...
                   'gui_OutputFcn',   @Lab_3_OutputFcn, ...

```

```

        'gui_LayoutFcn', [], ...
        'gui_Callback', []));
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Lab_3 is made visible.
function Lab_3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Lab_3 (see VARARGIN)

% Choose default command line output for Lab_3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Lab_3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Lab_3_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

```

Для занесения компонента в рабочее поле достаточно выделить его мышью в палитре компонент и перенести в нужное место рабочей области.

Пример создания GUI

В примере с помощью GUI открывается изображение и выводится на форму. Далее производится преобразование изображения в оттенки серого и вывод его на экран.

На форму необходимо разместить:

- Объекты Axes, в которых строятся изображения.
- Панель с кнопками для выбора изображения и преобразования его в оттенки серого.

После того, как все элементы будут размещены на форме необходимо каждому элементу axes в Inspector:axes заменить значение Visible на off (рисунок 4). Сконструированное окно GUI имеет вид:

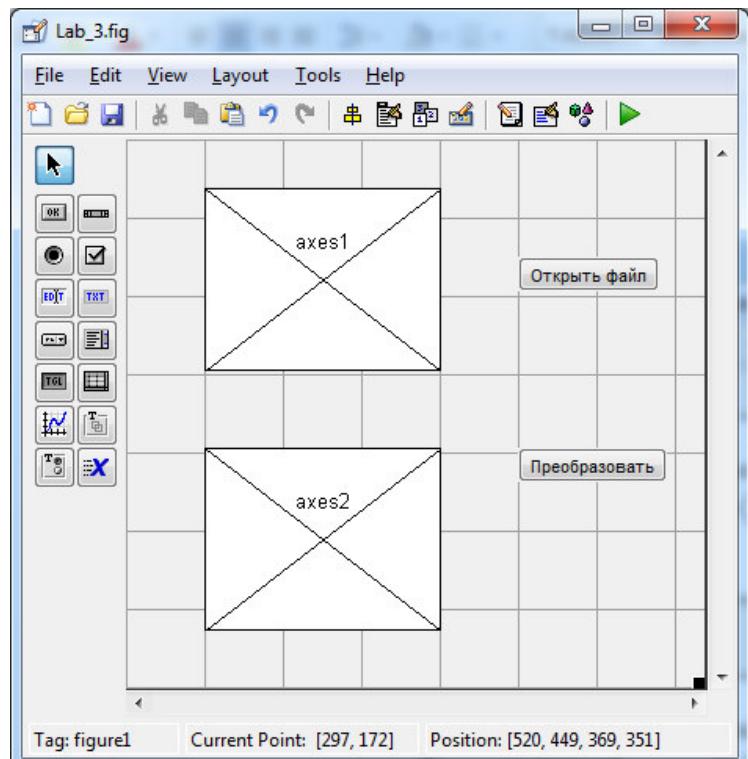


Рисунок 3 – Бланк с элементами

Для изменения свойств компонент можно использовать разные способы:

- Прямо в окне GUI, перемещая компонент или меняя его размеры манипулятором мыши можно изменить размеры и позицию.
- Использовать браузер объектов Object Browser, в котором выбирается нужный объект. При этом вызывается инспектор объектов.
- Property Inspector. Инспектор объектов, в котором отображаются свойства компонентов. Если выбирается конкретный компонент, то в инспекторе он выделяется цветом.
- Прямое редактирование кода в файле GUI.

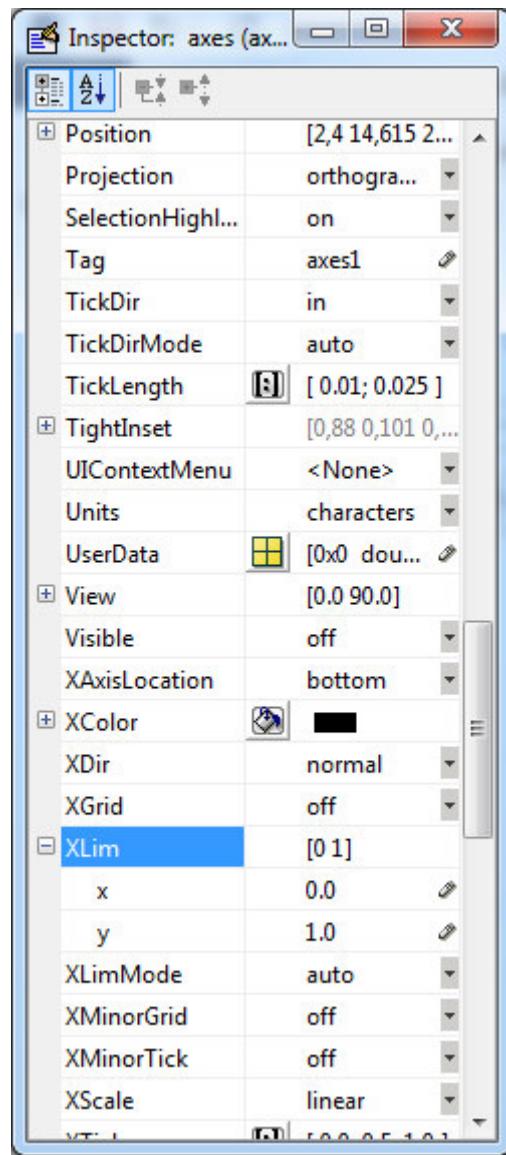


Рисунок 4 – Инспектор объектов

Компоненты GUI имеют множество свойств, задаваемых по умолчанию. Большинство менять не следует. В нашем примере для кнопок выбора изменяется пункт String (строка, содержащаяся в рамке кнопки).

После создания GUI он должен быть сохранен. Теперь его можно выполнить командой Tools=>Run (или кнопкой со стрелкой из панели инструментов).

Пример работы указан ниже.

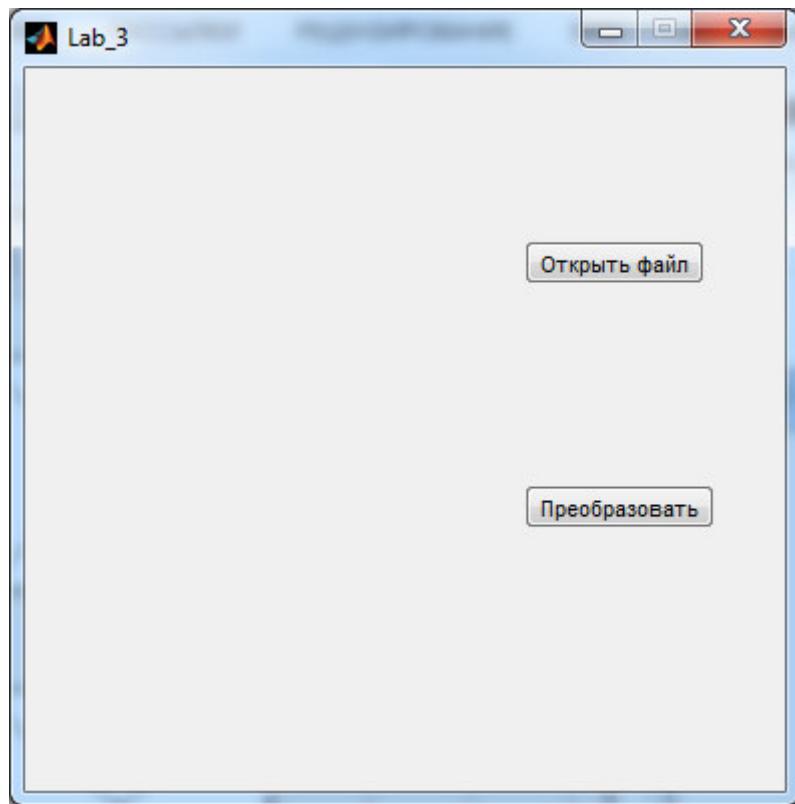


Рисунок 5 – Пример работы программы

После создания m-файла откроется окно M-File editor, где в комментариях описано назначение каждого элемента.

Для реализации открытия файла необходимо в окне редактора (M-editor) в `function pushbutton1_Callback(hObject, eventdata, handles)` добавить следующий код:

```
[fname,pname] = uigetfile('*.*', 'Enter data file'); % выбор изображения
global A; % создание глобальной переменной A
A=imread(fname); % считывание выбранного изображения
axes(handles.axes1); % указывается куда будет выведено изображение
set(handles.axes1);
imshow(A,[]);title('Исходное изображение'); % вывод выбранного
изображения на форму
```

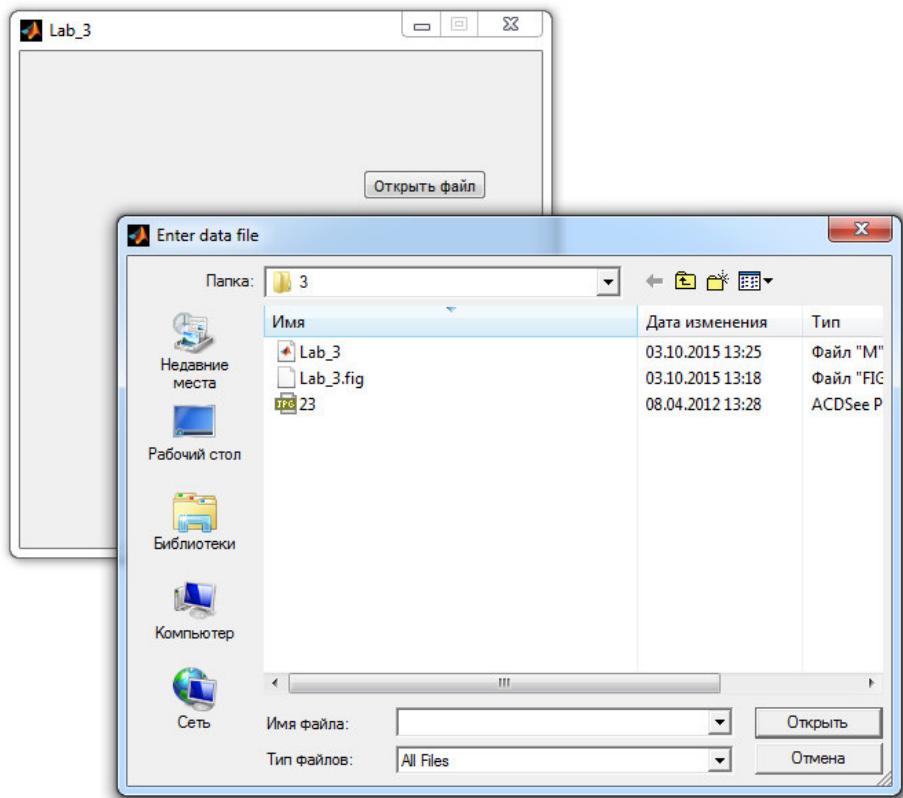


Рисунок 6 –Открытие файла

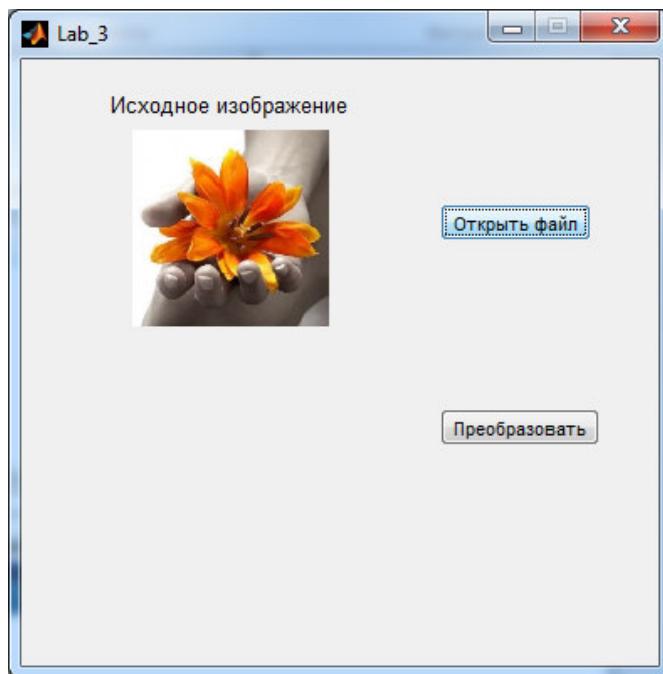


Рисунок 7 – Результат открытия файла

Далее необходимо добавить программный код для преобразования изображения в оттенки серого. Для этого в окне редактора (M-editor) в `function pushbutton2_Callback(hObject, eventdata, handles)` добавить следующий код:

```
global A; % создание глобальной переменной А  
global B; % создание глобальной переменной В
```

```

B=A(:,:,1); % выбрать первую цветовую компоненту изображения в формате
RGB
% Определить размеры изображения
[N1,N2]=size(B);
% Вывод исходного изображения на экран
axes(handles.axes2)
imshow(B, []);title('Результат преобразования');

```

Результат преобразования представлен на рисунке 8:

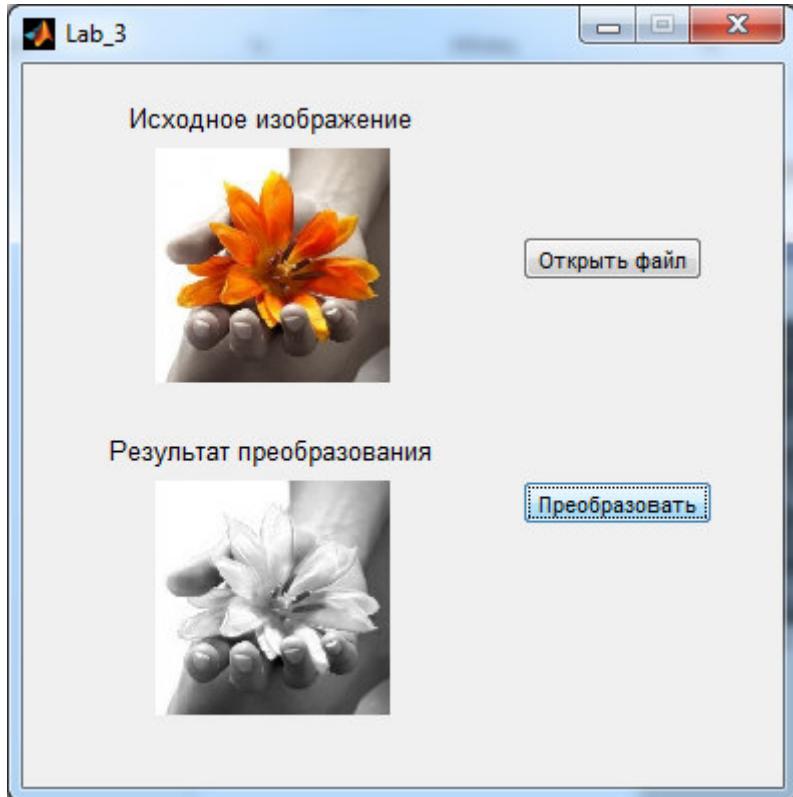


Рисунок 8 – Результат преобразования

Общая постановка задачи

В результате выполнения заданий лабораторной работы студенты **должны уметь** создавать программно-алгоритмическую поддержку для компьютерной реализации визуальных компонентов графического пользовательского интерфейса в MatLab.

Лабораторные занятия проводятся в компьютерных классах.

Список индивидуальных данных

Индивидуальными данными являются изображения, идентифицирующие личность студента.

Задание 1.

1. Реализовать пример, рассмотренный в теоретической части лабораторной работы.

2. Сохранить полученные результаты.

Задание 2.

1. На форме необходимо разместить следующие объекты:

- текстовое поле (edit) для значения цветовой компоненты (1, 2 или 3),
- поля для вывода изображений (исходного и результирующего),
- кнопки для открытия файла и преобразования изображения.

Реализовать пример, рассмотренный в теоретической части лабораторной работы.

2. Для того, чтобы получить значение из поля edit необходимо использовать следующий код:

```
G=str2double(get(handles.edit1, 'String'));
```

3. В результате преобразований получить 3 разных изображения и оформить отчет.

Примечание. В примере лабораторной работы рассмотрен результат преобразования изображения, где значение цветовой компоненты равно 1.

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.

Лабораторная работа №4. Преобразование изображений в пространственной области

Цель работы:

Целью лабораторной работы является получение навыков самостоятельной алгоритмической и программной реализации на компьютерной технике методов преобразования изображений в пространственной области в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-2	3-1	Способен анализировать социально-экономические проблемы
	3-2	Знает методы расчетов математических

		моделей
ПК-5	3-1	Знает методы оценки проектов по информатизации
	3-2	Знает методы оценки проектов по автоматизации решения прикладных задач

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-2	У-1	Способен анализировать социально-экономические проблемы
	У-2	Знает методы расчетов математических моделей

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-25	B-1	Знает различные математические методы решения прикладных задач
	B-2	Знает методы формализации решения прикладных задач

Теоретическая часть

Техника обработки пространственной области оперирует напрямую с пикселями изображения. Процессы в пространственной области можно обозначить уравнением

$$g(x,y)=T[f(x,y)],$$

где $f(x,y)$ – входное изображение, $g(x,y)$ – выходное (обработанное) изображение, а T – некоторый оператор (преобразование) над f , который определен в некоторой окрестности точки $f(x,y)$. Кроме того, оператор T может обрабатывать последовательность изображений, например, он может суммировать K входных изображений для подавления шума.

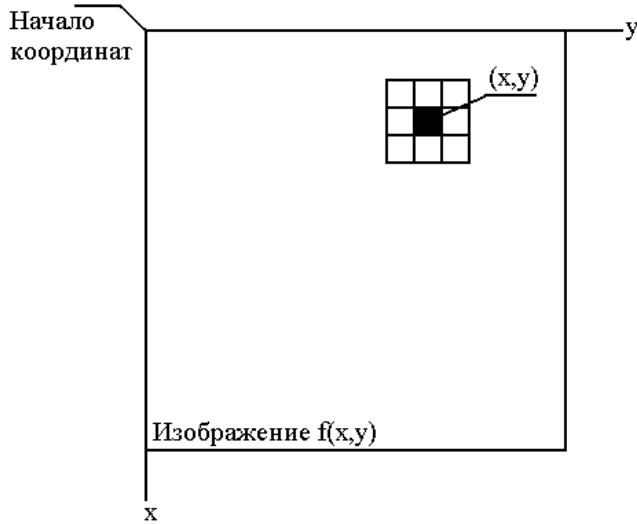


Рисунок 1 – Окрестность размера 3х3 вокруг точки (x,y) изображения

Главный подход к определению пространственной окрестности вокруг точки (x,y) состоит в использовании квадратной или прямоугольной области с центром в точке (x,y) , как показано на рисунке 1. Центр заданной шаблонной подобласти перемещается от пикселя к пикселу, начиная, скажем, из верхнего левого угла, и на своём пути он накрывает различные окрестности. Преобразование T применяется в каждой точке (x,y) , давая в результате выходное (обработанное) значение g для данной точки. В процессе вычислений используются только пиксели внутри заданной окрестности с центром в (x,y) .

Чтение изображения из файла

Для чтения изображения из файла используется следующий синтаксис:

```
D=imread(filename, fmt)
[X,map]=imread(filename, fmt)
[...]=imread(filename)
[...]=imread(..., idx)
[...]=imread(..., ref)
```

Функция **D=imread(filename, fmt)** читает из файла с именем `filename` бинарное, полутоновое или полноцветное изображение и помещает его в массив `D`. Функция **[X,map]=imread(filename, fmt)** читает из файла с именем `filename` палитровое изображение `X` с палитрой `map`.

Если MATLAB не может найти файл с именем `filename`, то ищется файл с именем `filename` и расширением `fmt`. Параметры `filename` и `fmt` являются строками. Параметр `fmt` в вызове функции может быть опущен, в этом случае формат файла автоматически определяется из его содержимого. В таблице 1 приведены типы изображений, которые могут быть прочитаны функцией `imread`.

Таблица 1. Типы изображений, которые могут быть прочитаны функцией `imread`.

Формат	Глубина цвета	Особенности
BMP	1, 4, 8, 24	Несжатые файлы
	4, 8	Файлы с RLE-сжатием
TIFF	1, 8, 24	Несжатые файлы
	1, 8, 24	Файлы, использующие Packbit-сжатие
	1	Файлы, использующие CCITT-сжатие
JPEG	8, 24	
PCX	1, 8, 24	
HDF	8, 24	
XWD	1, 8	

Функция `[...] = imread(..., idx)` читает одно изображение из TIFF-файла, содержащего несколько изображений. Номер изображения по порядку в списке IFD указывается в параметре `idx`. Если параметр `idx` при вызове функции не указан, то читается первое по порядку изображение в файле.

Функция `[...] = imread(..., ref)` читает одно изображение из HDF-файла, содержащего несколько изображений. Каждое изображение в HDF-файле имеет уникальный номер-описатель. Этот описатель указывается в параметре `ref`. Если параметр `ref` при вызове функции не указан, то читается первое по порядку изображение в файле.

Прочитанное из файла изображение имеет формат представления данных `uint8`.

Запись изображения в файл

Для записи изображения в файл используется следующий синтаксис:

```
imwrite(S, filename, fmt)
imwrite(X, map, filename, fmt)
imwrite(..., filename)
imwrite(..., Parameter, Value)
```

Функция `imwrite(S, filename, fmt)` записывает в файл с именем `filename` бинарное, полутоновое или полноцветное изображение `S`. Функция `imwrite(X, map, filename, fmt)` записывает в файл с именем `filename` палитровое изображение `X` с палитрой `map`. Формат файла определяется параметром `fmt`. Параметры `filename` и `fmt` являются строками. Возможные значения параметра `fmt` приведены в описании функции `iminfo`.

Функция `imwrite(..., filename)` аналогична описанным функциям, но формат файла определяется по расширению `filename`.

В таблице 2 приведены типы изображений, которые могут быть прочитаны функцией `imwrite`.

Таблица 2. Типы изображений, которые могут быть прочитаны функцией `imwrite`.

Формат	Тип изображений
BMP	8 бит/пиксел - палитровые, 24 бит/пиксел - палитровые
TIFF	Бинарные несжатые или с использованием Packbit- или CCITT-сжатия; 8 бит/пиксел - палитровые или полуточновые несжатые или с использованием Packbit-сжатия; несжатые или с использованием Packbit-сжатия
JPEG	8 бит/пиксел - полуточновые, 24 бит/пиксел - полноцветные; палитровые конвертируются в полноцветные
PCX	8 бит/пиксел - полуточновые
HDF	8 бит/пиксел - полуточновые и палитровые, 24 бит/пиксел - полноцветные
XWD	8 бит/пиксел-палитровые

При записи изображений в файлы форматов TIFF, JPEG, HDF можно указать ряд дополнительных параметров, влияющих на способ сохранения изображений. Для этого в функции `imwrite` после параметров `filename` и `fmt` передается одна или несколько пар параметров `Parameter, Value`.

Если запись осуществляется в JPEG-файлы, то можно указывать показатель качества сжатого изображения. Для этого `Parameter` должен быть строкой ‘Quality’, а `Value` - число, которое определяет степень сжатия изображения. Этот показатель может принимать значения в диапазоне [0, 100]. Чем меньше значение этого параметра, тем выше степень сжатия, но хуже качество изображения.

При записи изображений в TIFF-файл можно использовать следующие дополнительные параметры (табл. 3):

Таблица 3. Дополнительные параметры при записи изображений в TIFF-файл

Параметр	Возможные значения Value	Значение Value по умолчанию
‘Compression’	‘none’ - не использовать сжатие; ‘packbits’ - использовать метод сжатия Packbits; ‘ccitt’ - использовать метод сжатия CCITT	‘ccitt’ - для бинарных изображений; ‘packbits’ - для других типов изображений
‘Description’	Любая строка. Эта строка находится в поле	“”

	ImageDescription структуры, возвращаемой iminfo	
‘Resolution’	Разрешение в точках на дюйм	72

При записи изображений в HDF - файл можно использовать следующие дополнительные примеры (табл. 4):

Таблица 4. Дополнительные параметры при записи изображений в HDF - файл

Параметр	Возможные значения Value	Значение Value по умолчанию
‘Compression’	‘none’ – не использовать сжатие; ‘rle’ - использовать метод сжатия RLE; ‘jpeg’ – использовать метод сжатия JPEG	‘rle’
‘Quality’	Показатель качества при JPEG - сжатии, это число задается в диапазоне [0, 100]	75
‘WriteMode’	‘overwrite’ – переписать существующий файл; ‘append’ – добавить изображение в существующий файл	‘overwrite’

Если исходное изображение имеет формат представления данных double, то перед записью в файл данные изображения автоматически преобразуются в формат uint8.

Вывод изображения на экран

Для вывода изображения на экран используется следующий синтаксис imshow(I, n)

```
imshow(I, [low high])
imshow(BW)
imshow(X, map)
imshow(RGB)
imshow(..., display_option)
imshow(XData, YData, ...)
imshow filename
h=imshow(...)
```

Функция **imshow(I, n)** выводит на экран полутоновое изображение I, используя при выводе n уровней серого. Если при вызове функции опустить параметр n, то когда MATLAB запущен в графическом режиме TrueColor, для вывода полутонового изображения используется 256 градаций серого или

64 градации серого, когда MATLAB запущен в графическом режиме с меньшим количеством цветов.

Функция **imshow(I, [low high])** выводит на экран полутононое изображение I, дополнительно контрастируя выводимое изображение. Пиксели изображения I, яркость которых меньше либо равна low, отображаются черным цветом. Пиксели, яркость которых больше либо равна high, отображаются белым цветом. Пиксели, яркость которых имеет значение между low и high, отображаются серым цветом. Все уровни серого равномерно распределены от low до high. Если вызвать функцию **imshow(I, [])**, указав вторым аргументом пустой массив, то low будет присвоено минимальное значение в I($low = \min(I(:))$), а high будет присвоено максимальное значение в I($high = \max(I(:))$).

Функция **imshow(BW)** выводит на экран бинарное изображение BW. Пиксели, значение которых равно 0, отображаются черным цветом. Пиксели, значение которых равно 1, отображаются белым цветом.

Функция **imshow(X, map)** выводит на экран палитровое изображение X с палитрой map.

Функция **imshow(RGB)** выводит на экран полноцветное изображение RGB.

Дополнительно в перечисленные функции можно передать параметр **display_option** (**imshow(...,display_option)**), который может принимать значения 'truesize' и 'notruesize'. Если параметр **display_option** равен 'truesize', то **imshow** будет автоматически вызывать функцию **truesize**. Если параметр **display_option** равен 'notruesize', то вызова функции **truesize** не будет. Когда параметр **display_option** не определен, вызов функции **truesize** зависит от значения глобальной переменной IPT 'ImshowTruesize'.

Кроме того, в перечисленные функции можно передать два двухэлементных вектора XData и YData, определяющих диапазон изменения значений по осям пространственной системы координат: **imshow(XData, YData,...)**.

Функция **imshow filename** выводит на экран изображение из файла с именем **filename**. Для чтения файла **imshow** вызывает функцию **imread**.

Если для функций **imshow** определить выходной параметр **h=imshow(...)**, то в h будет возвращен описатель (handler) выведенного изображения как объекта графического интерфейса MATLAB.

Преобразования яркости изображений

Простейшая форма преобразования T получается, когда окрестность на рисунок 1 имеет размер 1×1 (т.е. состоит из одного пикселя). В этом случае значение g в точке (x, y) зависит только от значения f в этой точке, и T становится функцией преобразования яркости (также называемой функцией градационного преобразования). Эти два термина эквивалентны применительно к монохромным (полутоновым) изображениям. При обращении с цветными изображениями термин **яркость** используется для

обозначения цветовой компоненты изображения в конкретном цветовом пространстве.

Поскольку такие преобразования зависят только от значения яркости, но не от (x, y) , функцию преобразования яркости часто записывают в простой форме

$$s = T(r),$$

где r обозначает яркость f , а s - яркость g в любой сопутствующей точке (x, y) изображения.

Функция **imadjust** является базовым инструментом пакета IPT (Image Processing Toolbox) при преобразованиях яркости полутоночных изображений. **Imadjust** – увеличение контраста изображений путем изменения диапазона интенсивностей исходного изображения. Она имеет следующий синтаксис:

```
Id=imadjust(Is, [low high], [bottom top], gamma)
newmap=imadjust(map, [low high], [bottom top], gamma)
RGBd=imadjust(RGBs, [low high], [bottom top], gamma)
```

Функция **Id=imadjust(Is, [low high], [bottom top], gamma)** создает полуточное изображение Id путем контрастирования исходного полуточного изображения Is . Значения яркости в диапазоне $[low high]$ преобразуются в значения яркости в диапазоне $[bottom top]$. Значения яркости, меньшие low , принимают значение $bottom$, а значения яркости, большие $high$, принимают значение top . Значения top , $bottom$, low , $high$ должны принадлежать диапазону $[0,1]$. Если в качестве второго ($[low high]$) или третьего ($[bottom top]$) параметров передать пустой вектор $[]$, то по умолчанию будет использован вектор $[0,1]$. С помощью показателя $gamma$ можно дополнительно осуществлять преобразование, называемое *гамма - коррекцией*. Параметр $gamma$ определяет форму кривой характеристики передачи уровней яркости. Если $gamma$ меньше 1, то характеристика передачи уровней будет выпуклой и результирующее изображение будет светлее, чем исходное. Если $gamma$ больше 1, то характеристика передачи уровней будет вогнутой и результирующее изображение будет темнее, чем исходное. По умолчанию параметр $gamma$ равен 1, что соответствует линейной характеристике передачи уровней и отсутствию гамма - коррекции. При вызове функции показатель $gamma$ можно опустить.

Характеристики передачи уровней для различных значений $gamma$ приведены на рисунок 2.

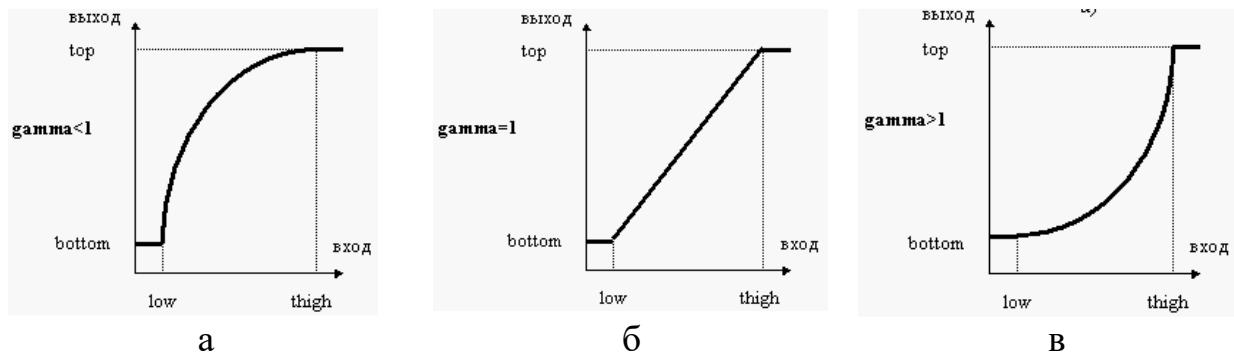


Рисунок 2 – Характеристика передачи уровней для различных значений gamma

Функция **newmap=imadjust(map, [low high], [boton top], gamma)** создает новую палитру newmap путем контрастирования исходной палитры map. Преобразование осуществляется аналогично преобразованию полутонового изображения отдельно для каждой R-, G-, B-составляющих. Диапазоны преобразования интенсивностей составляющих [low high] и [boton top], а также показатель gamma задаются одинаковыми для всех составляющих.

Функция **RGBd=imadjust(RGBs, [low high], [botton top], gamma)** создает полноцветное изображение RGBd путем контрастирования исходного полноцветного изображения RGBs. Преобразование осуществляется аналогично преобразованию полутонового изображения отдельно для каждой из R-, G-, B-составляющих. Диапазоны преобразования интенсивностей составляющих [low high] и [botton top], а также показатель gamma задаются одинаковыми для всех составляющих.

Однако существуют классы изображений, для которых линейное или нелинейное преобразование яркостных диапазонов не всегда эффективно. В первую очередь это касается изображений, градации яркостей которого занимают максимально возможный диапазон, а яркостной диапазон потенциально информативных участков изображения – небольшую часть диапазона. Для таких изображений рекомендуется применять метод кусочно-линейных или кусочно-нелинейных преобразований яркостных диапазонов. Этот метод позволяет максимально сберечь визуальную информацию на изображениях при их преобразованиях. Следует отметить, что известные методы трансформации яркостных диапазонов являются частным случаем метода кусочно-линейных (нелинейных) преобразований градационных характеристик изображения.

Форматы представления данных исходного и результирующего изображения совпадают.

Рассмотрим пример использования функции **imadjust**. Пример демонстрирует контрастирование изображения.

Полутоновое изображение I читается из файла и отображается на экране (рисунок 2а). Для анализа диапазона яркостей исходного изображения I строится гистограмма яркостей пикселов с помощью функции **imhist(I)** (рисунок 2б). По гистограмме видно, что пиксели изображения I имеют яркости в диапазоне [0, 75], изображение I недостаточно контрастное. Функция **imadjust** “растягивает” исходный диапазон яркостей на диапазон от минимально возможной яркости до максимально возможной. Результат преобразования выводится на экран в новое окно (рисунок 2в). Гистограмма яркостей пикселов результирующего изображения показана на рисунок 2г. Результирующее изображение контрастнее, чем исходное.

```
%Чтение исходного изображения и вывод его на экран  
I=imread('gory.tif');
```

```

imshow(I);
%Построение гистограммы исходного изображения
figure, imhist(I);
%Контрастирование исходного изображения.
I=imadjust(I, [0 75]/255, [ ], 1);
%Вывод преобразованного изображения на экран
figure, imshow(I);
%Вывод гистограммы преобразованного изображения
figure, imhist(I);

```

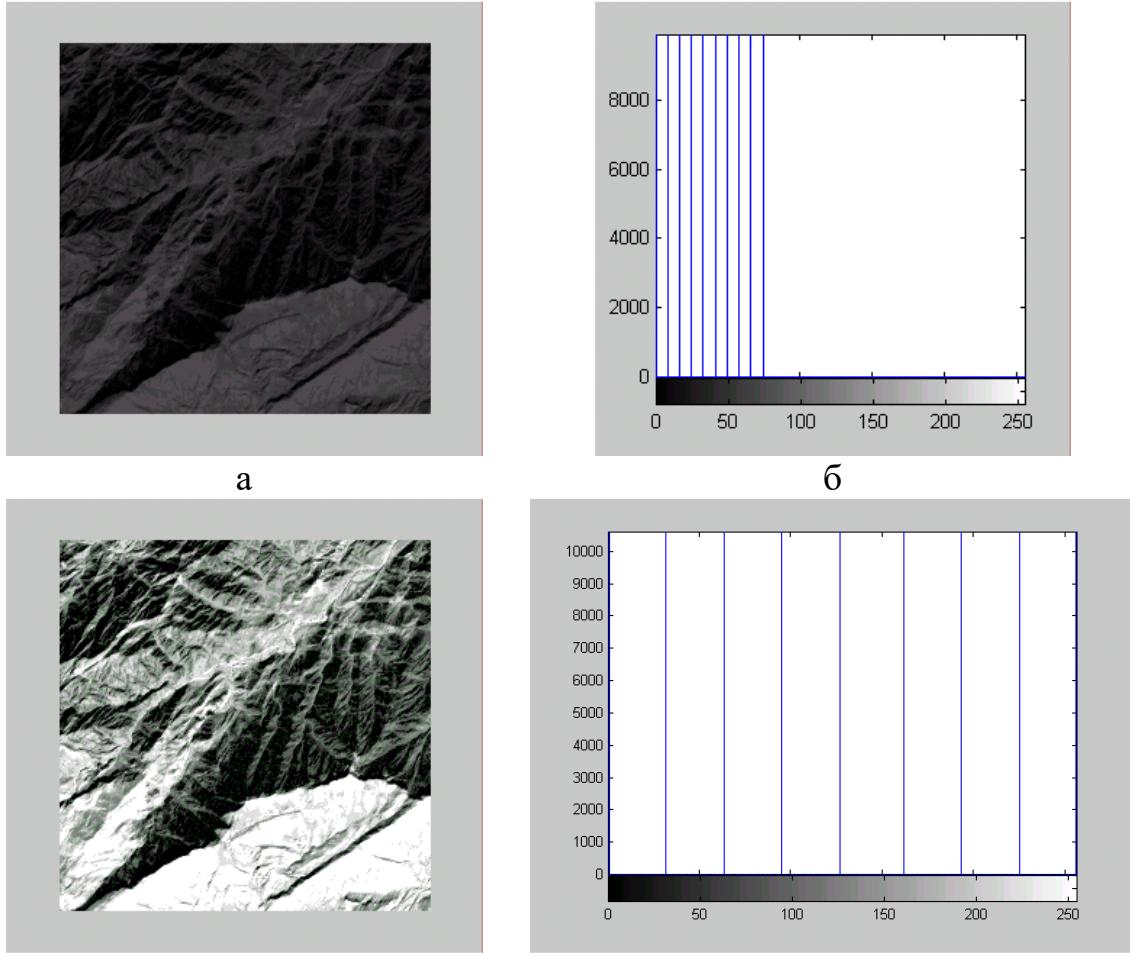


Рисунок 3 – Пример использования функции *imadjust*

Обработка гистограмм и построение графиков функций

Функции преобразования изображений, основанные на информации, которая извлекается из гистограмм яркости изображений, играют ключевую роль при обработке изображений, совершающейся при решении задач улучшения изображений, их сжатия, сегментации и описания.

Нахождение и построение гистограмм

Гистограммой цифрового изображения, число возможных уровней яркости которого равно L , лежащих в диапазоне $[0, G]$, называется дискретная функция

$$h(r_k) = n_k$$

где r_k — это k -ый уровень яркости из интервала $[0, G]$, а n_k — число пикселов изображения, уровень яркости которых равен r_k . Значение G равно 255 для изображений класса **uint8**, 65535 — для класса **uint16** и 1.0 — для класса **double**. Напомним, что индексы в MATLAB начинаются с 1, а не с 0, поэтому r_1 соответствует уровню яркости 0, r_2 соответствует уровню яркости 1 и так далее до r_L , что соответствует уровню G . Заметим, что $G = L - 1$ для изображений класса **uint8** и **uint16**.

Часто бывает удобно работать с *нормированными* гистограммами, которые получаются делением элементов $h=(r_k)$ на общее число пикселов изображения, которое мы обозначим n :

$$p(r_k) = \frac{h(r_k)}{n} = \frac{n_k}{n}$$

при $k = 1, 2, \dots, L$. С точки зрения теории вероятностей, число $p(r_k)$ — это вероятность (частота) появления (присутствия) уровня интенсивности r_k в данном изображении.

Стержневой функцией пакета для обращения с гистограммами служит функция **imhist** для построения гистограммы. Функция **imhist** имеет следующий синтаксис:

```
imhist(I, n)
imhist(BW, n)
[h, cx]=imhist(I, n)
[h, cx]=imhist(BW, n)
[h, cx]=imhist(BW, map)
```

Функции **imhist(I, n)** и **imhist(BW, n)** в текущем окне строят гистограммы яркостей пикселов соответственно полутонового и бинарного изображений. Гистограмма состоит из n столбцов. Значение n при вызове функции можно не указывать, в этом случае будут использованы значения по умолчанию: $n=256$ для полутонового изображения и $n=2$ для бинарного изображения. Под рисунком гистограммы выводится шкала яркостей.

Функция **[h, cx]=imhist(BW, map)** в текущем окне строит гистограмму индексов пикселов палитрового изображения X . Под рисунком гистограммы выводится палитра map .

Функция **[h, cx]=imhist(BW, map)** возвращает вектор гистограммы h и вектор положения центров столбцов гистограммы cx на оси яркостей (для полутоновых и бинарных изображений) или на оси индексов (для палитровых изображений), что позволяет производить дальнейшую обработку гистограммы h или, например, построить огибающую гистограммы с помощью функции **plot(cx, h)**.

На примере показано как для полутонового изображения I (рис. 5) строится гистограмма яркостей пикселов (рис. 4) и огибающая гистограммы в логарифмическом масштабе (рис. 6).

```
%Чтение изображения из файла
```

```
I=imread('lena.tif');
```

```
%Построение гистограммы яркостей пикселов и вывод ее на экран
```

```

imhist(I);
%Получение гистограммы в логарифмическом масштабе и вывод ее на
экран
[h, cx]=imhist(I);
h=log10(h);
figure, plot(cx, h);

```



Рисунок 4 – полутоновое изображение I

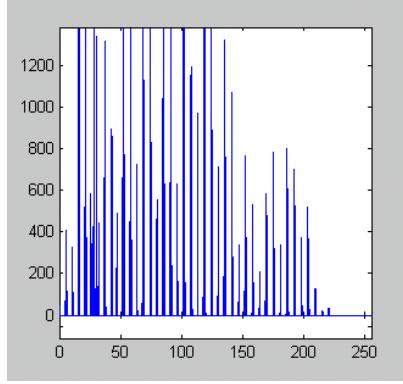


Рисунок 5 –
Гистограмма яркостей
пикселов

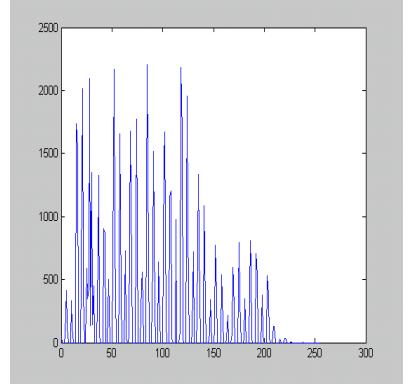


Рисунок 6 – Огибающая
гистограммы в
логарифмическом
масштабе

Гистограмму можно построить с помощью *столбчатых диаграмм*. Для этого служит функция **bar**, которая имеет следующий синтаксис:

```

bar(y) bar(x, y)
[xb, yb] = bar(...)
bar(y, '<тип линии>')
bar(x, y, '<тип линии>')

```

Команда **bar(y)** выводит график элементов одномерного массива **y** в виде столбцовой диаграммы.

Команда **bar(x, y)** выводит график элементов массива **y** в виде столбцов в позициях, определяемых массивом **x**, элементы которого должны быть упорядочены в порядке возрастания.

Если **X** и **Y** – двумерные массивы одинаковых размеров, то каждая диаграмма определяется соответствующей парой столбцов и они надстраиваются одна над другой.

Команды **bar(y, '<тип линии>')**, **bar(x, y, '<тип линии>')** позволяют задать тип линий, используемых для построения столбцовых диаграмм, по аналогии с командой **plot**.

Функция **[xb, yb] = bar(...)** не выводит графика, а формирует такие массивы **xb** и **yb**, которые позволяют построить столбцовую диаграмму с помощью команды **plot(xb, yb)**.

Рассмотрим примеры использования функции **bar**.

Пример 1:

```

x = -2.9:0.2:2.9;
bar(x,exp(-x.*x), 'r')

```

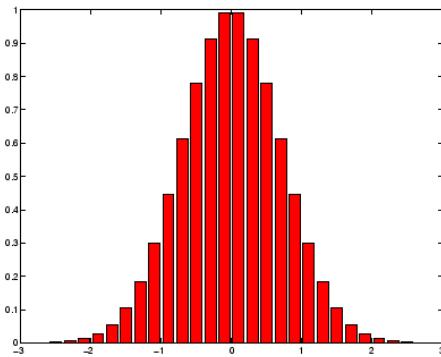


Рисунок 7 – Примеры использования функции bar (пример 1)

Пример 2:

```
Y = round(rand(5,3)*10);
subplot(2,2,1)
bar(Y,'group')
title 'Group'
subplot(2,2,2)
bar(Y,'stack')
title 'Stack'
subplot(2,2,3)
barh(Y,'stack')
title 'Stack'
subplot(2,2,4)
bar(Y,1.5)
title 'Width = 1.5'
```

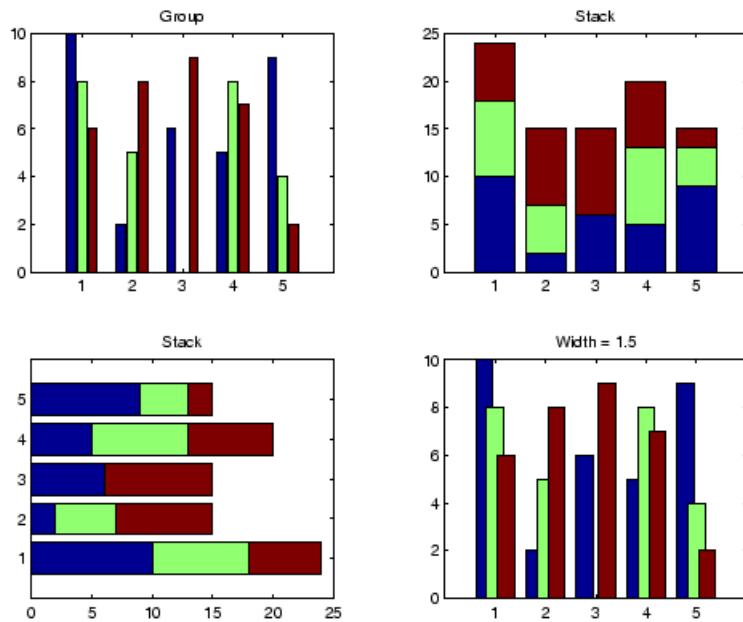


Рисунок 8 – Примеры использования функции bar (пример 2)

Для построения объемных диаграмм применяется функция **bar3**. Использование **barh** и **bar3** аналогично **bar**. **Bar3** является трехмерной столбцовой диаграммой. **Bar3** имеет следующий синтаксис:

```
bar3(Y)
bar3(x,Y)
bar3(...,width)
bar3(...,'style')
bar3(...,LineSpec)
bar3(axes_handle,...)
h = bar3(...)
bar3h(...)
h = bar3h(...)
```

Пример использования функции **bar3**:

```
Y = cool(7);
subplot(3,2,1)
bar3(Y,'detached')
title('Detached')
subplot(3,2,2)
bar3(Y,0.25,'detached')
title('Width = 0.25')
subplot(3,2,3)
bar3(Y,'grouped')
title('Grouped')
subplot(3,2,4)
bar3(Y,0.5,'grouped')
title('Width = 0.5')
subplot(3,2,5)
bar3(Y,'stacked')
title('Stacked')
subplot(3,2,6)
bar3(Y,0.3,'stacked')
title('Width = 0.3')
colormap([1 0 0;0 1 0;0 0 1])
```

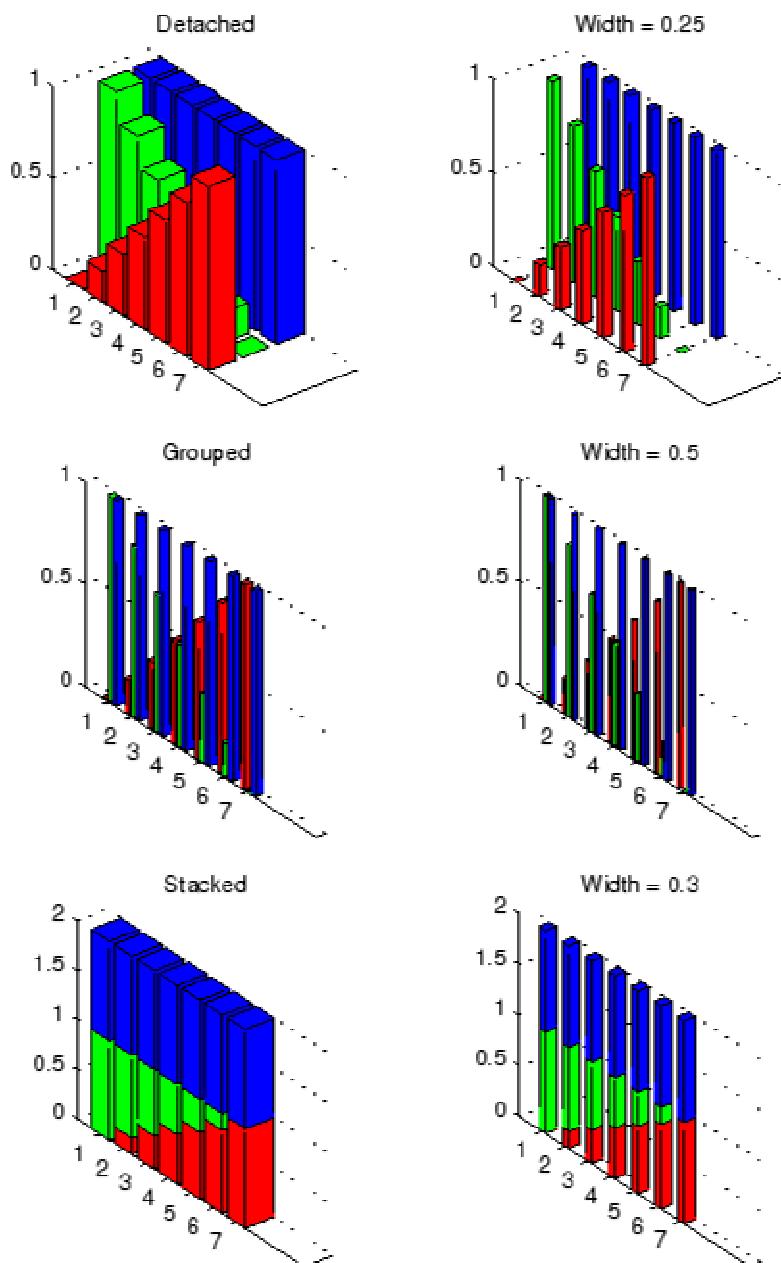


Рисунок 9 – Результаты использования функции `bar3`

Функция `axis` используется для масштабирования осей и вывод на экран имеет синтаксис:

```
axis([xmin xmax ymin ymax])
axis([xmin xmax ymin ymax zmin zmax])
axis('auto')
axis(axis)
v = axis
axis('ij')
axis('xy')
axis('square')
axis('equal')
axis('off')
```

```
axis('on')
[s1, s2, s3] = axis('state')
axis(s1, s2, s3)
```

Команда **axis** обеспечивает преемственность предшествующих версий системы MATLAB, ориентированных на символьную обработку, с версиями 4.x, ориентированными на графический интерфейс.

Команда **axis([xmin xmax ymin ymax])** устанавливает масштаб по осям x, y для активного графического окна.

Команда **axis([xmin xmax ymin ymax zmin zmax])** устанавливает масштаб по осям x, y, z для активного графического окна.

Команда **axis('auto')** возвращает масштаб по осям к штатным значениям (принятым по умолчанию).

Команда **axis(axis)** фиксирует текущие значения масштабов для последующих графиков, как если бы был включен режим hold.

Функция **v = axis** возвращает вектор-строку масштабов по осям для активного графика. Если график двумерный, то v имеет 4 компонента; если трехмерный - 6 компонентов.

Команда **axis('ij')** перемещает начало отсчета в левый верхний угол, сохраняет положение осей и реализует отсчет по вертикальной оси из верхнего левого угла (матричная система координат).

Команда **axis('xy')** возвращает декартову систему координат; начало отсчета находится в нижнем левом углу; ось x горизонтальна и размечается слева направо, ось y вертикальна и размечается снизу вверх.

Команда **axis('square')** устанавливает одинаковый диапазон изменения переменных по осям.

Команда **axis('equal')** устанавливает масштаб, который обеспечивает одинаковые расстояния между метками по осям x и y.

Команда **axis('image')** устанавливает масштаб, который обеспечивает квадратные размеры пикселей.

Команда **axis('normal')** восстанавливает полноразмерный масштаб, отменяя масштабы, установленные командами **axis('square')** и **axis('equal')**.

Команда **axis('off')** снимает с осей их обозначения и маркеры.

Команда **axis('on')** восстанавливает на осях их обозначения и маркеры.

Функция **[s1, s2, s3] = axis('state')** возвращает строку, определяющую вектор состояния объекта axes:

```
s1 = 'auto' | 'manual'.
s2 = 'on' | 'off'.
s3 = 'xy' | 'ij'.
```

Команда **axis(s1, s2, s3)** устанавливает параметры объекта axes в соответствии с вектором состояния [s1, s2, s3]. По умолчанию этот вектор принимает значения ['auto', 'on', 'xy'].

Функция **axes** создает графический объект и имеет синтаксис:

```

axes
axes('PropertyName',PropertyValue,...)
axes(h)
h = axes(...)

```

Стеблевая диаграмма строится аналогично столбчатой диаграмме. Ее синтаксис имеет вид:

```

stem(y)
stem(x, y)
stem(y, <тип линии>)
stem(x, y, <тип линии>)

```

Команда `stem(y)` выводит график элементов одномерного массива `y` в виде вертикальных линий, которые заканчиваются в точках графика, помечаемых кружочком.

Команда `stem(x, y)` выводит график элементов массива `y` в виде вертикальных линий в позициях, определяемых массивом `x`, элементы которого должны быть упорядочены в порядке возрастания.

Команды `stem(y, <тип линии>)`, `stem(x, y, <тип линии>)` позволяют задать тип линий, используемых для построения дискретного графика, по аналогии с командой `plot`.

На рисунке 10 пример использования функции `stem` для построения графика.

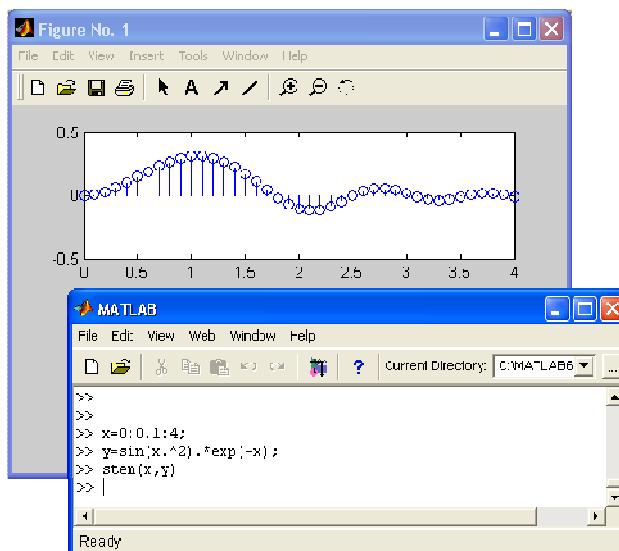


Рисунок 10 – Пример использования функции для построения стеблевой диаграммы

Рассмотрим функцию `plot`, которая строит график по точкам, соединяя их отрезками прямых линий. Она имеет синтаксис:

```

plot(y)
plot(x, y)
plot(x, y, s)

```

```
plot(x1, y1, s1, x2, y2, s2, ...)
```

Команда `plot(y)` строит график элементов одномерного массива `y` в зависимости от номера элемента; если элементы массива `y` комплексные, то строится график `plot(real(y), imag(y))`. Если `Y` - двумерный действительный массив, то строятся графики для столбцов; в случае комплексных элементов их мнимые части игнорируются.

Команда `plot(x, y)` соответствует построению обычной функции, когда одномерный массив `x` соответствует значениям аргумента, а одномерный массив `y` – значениям функции. Когда один из массивов `X` или `Y` либо оба двумерные, реализуются следующие построения:

- если массив `Y` двумерный, а массив `x` одномерный, то строятся графики для столбцов массива `Y` в зависимости от элементов вектора `x`;
- если двумерным является массив `X`, а массив `y` одномерный, то строятся графики столбцов массива `X` в зависимости от элементов вектора `y`;
- если оба массива `X` и `Y` двумерные, то строятся зависимости столбцов массива `Y` от столбцов массива `X`.

Команда `plot(x, y, s)` позволяет выделить график функции, указав способ отображения линии, способ отображения точек, цвет линий и точек с помощью строковой переменной `s`, которая может включать до трех символов

Разметку осей координат на графиках можно производить вручную, а также автоматически. Автоматически это делают с помощью функций `ylim` и `xlim`. Для наших целей достаточно использовать их синтаксис вида:

```
ylim ('auto') xlim('auto').
```

Другой возможный вариант синтаксиса этих функций допускает ручное задание опций:

```
ylim ([ymin ушах]) xlim([xmin xmax]).
```

Если ограничения установлены только для одной оси, то вторая ось оформляется по умолчанию.

Если в командной строке набрать команду `hold on`, то текущий график будет сохранен на экране, причем результаты выполнения последующих графических команд будут отображаться в этом же окне.

Эквализация гистограммы.

Предположим, что уровни яркости являются непрерывными величинами, распределенными в диапазоне $[0,1]$. Пусть $p_r(r)$ обозначает функцию плотности распределения вероятности (PDF, probability density function) уровней яркости данного изображения, где нижний индекс используется для различия PDF входного и выходного изображений. Рассмотрим следующее преобразование входных уровней для получения выходных (обработанных) уровней яркости s

$$s = T(r) = \int_0^r p_r(w) dw,$$

где w — переменная, по которой ведется интегрирование. Можно показать, что функция распределения плотности выходных уровней является равномерной, т. е.

$$p_s(s) = \begin{cases} 1 & \text{при } 0 \leq s \leq 1, \\ 0 & \text{иначе.} \end{cases}$$

Другими словами, предыдущее преобразование порождает изображение, уровни яркости которого являются равновероятными и покрывают весь интервал $[0,1]$. Результат этого процесса *эквализации* изображения состоит в увеличении динамического диапазона уровней яркости, что обычно означает большую контрастность выходного изображения. Заметим, что функция преобразования является не чем иным, как функцией кумулятивного (накопленного) распределения (CDF, cumulative distribution function).

Имея дело с дискретными величинами, нам приходится работать с гистограммами, поэтому в этом случае описанная выше техника называется *гистограммной эквализацией*, хотя в общем случае гистограмма обрабатываемого изображения не будет равномерной в силу самой природы дискретных величин. Используя обозначения, пусть $p_r(r_j)$, $j=1, \dots, L$, обозначает гистограмму уровней яркости некоторого исходного изображения. Напомним, что величины нормированной гистограммы являются приближениями вероятностей появления каждого уровня яркости на изображении. Для дискретных величин мы делаем суммирование (вместо интегрирования), и преобразование эквализации приобретает следующий вид:

$$S_k = T(r_k) = \sum_{j=1}^k p_r(r_j) = \sum_{j=1}^k \frac{n_j}{n}$$

при $k = 1, \dots, L$, где S_k — величина яркости выходного (обработанного) изображения, соответствующая значению яркости r_k входного изображения.

Эквализация гистограмм реализована в пакете IPT функцией **histeq**. **Histeq** — выполнение операции эквализации (выравнивания) гистограммы. В этом подходе увеличение контрастности изображения происходит путем преобразования гистограммы распределения значений интенсивностей элементов исходного изображения. Существуют также другие подходы к видоизменению гистограмм.

Функция **histeq** улучшает контраст изображения с помощью преобразования значений пикселов исходного изображения таким образом, чтобы гистограмма яркостей пикселов результирующего изображения приблизительно соответствовала некоторой предопределенной гистограмме [1,2]. Данная функция предназначена для преобразования полутонаовых изображений или палитровых изображений.

Функция **histeq** имеет синтаксис:

```
Id=histeq(Is, hgram)
Id, T]=histeq(Is, hgram)
Id=histed(Is, n)
```

```
[Id, T]=histeq(Is, n)
newmap=histeq(X, map, hgram)
[newmap, T]=histeq(X, map, hgram)
newmap=histeq(X, map)
[newmap, T]=histeq(X, map)
```

Функция **histeq(Is, hgram)** преобразует исходное полутоновое изображение Is таким образом, чтобы гистограмма яркостей пикселов результирующего полутонового изображения Id приблизительно соответствовала гистограмме, задаваемой вектором hgram. Количество элементов в hgram задает число столбцов гистограммы, а значение каждого элемента – относительную высоту каждого столбца. Значение элементов вектора hgram должно быть в диапазоне [0, 1]. Функция hgram автоматически масштабирует значения элементов hgram так, чтобы сумма значений элементов в гистограмме была равна количеству пикселов изображения, т.е. $\text{sum}(\text{hgram})=\text{prod}(\text{size}(Is))$. Гистограмма результирующего изображения Id будет лучше соответствовать заданной гистограмме hgram в том случае, когда количество столбцов hgram ($\text{length}(\text{hgram})$) много меньше количества градаций яркости исходного изображения Is.

Функция **Id=histeq(Is, n)** преобразует исходное полутоновое изображение Is таким образом, чтобы результирующее полутоновое изображение Id имело гистограмму яркостей пикселов, близкую к равномерной. Равномерная гистограмма hgram создается из n столбцов как $\text{hgram}=\text{ones}(1, n)*\text{prod}(\text{size}(Is))/n$. Чем меньше n по сравнению с количеством градаций яркости в изображении Is, тем более равномерной получается гистограмма яркостей пикселов результирующего изображения Id. По умолчанию значение n равно 64, и данный параметр можно не указывать при вызове функции. Формат результирующего изображения Id совпадает с форматом исходного Is.

Функция **[Id, T]=histeq(Is, n)** дополнительно возвращает вектор T, задающий характеристику передачи уровней яркости исходного изображения Is в уровнях яркости результирующего изображения Id.

Функция **newmap=histeq(X, map, hgram)** преобразует палитру map, связанную с палитровым изображением X, таким образом, чтобы гистограмма яркостей пикселов изображения X с палитрой newmap приблизительно соответствовала гистограмме hgram. Для получения значений яркости из палитры map используется функция **rgb2ntsc**. Количество элементов вектора hgram должно быть равно размеру палитры map.

Функция **newmap=histeq(X, map, hgram)** преобразует палитру map, связанную с палитровым изображением X, таким образом, чтобы гистограмма яркостей пикселов изображения X с палитрой newmap была близка к равномерной.

Функция **[newmap, T]=histeq(X, ...)** дополнительно возвращает вектор T, задающий характеристику передачи уровней яркости из исходной палитры map в уровнях яркости результирующей палитры newmap.

Существуют также другие, более сложные подходы к решению задачи улучшения изображений путем гистограммных преобразований. Они базируются не только на анализе гистограммы распределения яркостей элементов изображения в целом, но и ее отдельных компонент. Также существует ряд методов, основанных на анализе локальных гистограмм отдельных участков изображения. Существование большого количества методов гистограммных преобразований объясняется их относительной простотой и высокой эффективностью.

Характеристика передачи уровней яркости T выбирается путем минимизации функции $F(k) = |c_1(T(k)) - c_0(k)|$, где c_0 - сумма всех пикселов полутонаового изображения I с яркостью, меньше либо равной k ; c_1 - сумма значений заданной гистограммы h_{gram} для всех дискретных уровней яркости, меньших либо равных k . При минимизации на функцию T накладывают следующие ограничения: T должна быть монотонной и $c_1(T(k))$ не должно превышать $c_0(a)$ более, чем на половину количества точек с яркостью a .

Пространственная фильтрация

Окрестностная обработка изображений состоит из следующих действий:

- определение центральной точки (x, y) ;
- совершение операции, которая использует лишь значения пикселов в заранее оговоренной окрестности вокруг центральной точки;
- назначение результата этой операции «откликом» совершаемого процесса в *этой* точке;
- повторение всего процесса для каждой точки изображения.

В результате перемещения центральной точки образуются новые окрестности, отвечающие каждому пикселу изображения. Для описанной процедуры принято использовать термины *окрестностная обработка* и *пространственная фильтрация*, причем последний является более употребимым. Как будет объяснено в следующем параграфе, если операции, совершаемые над пикселями окрестности, являются линейными, то вся процедура называется *линейной пространственной фильтрацией* (также иногда используется термин *пространственная свертка*), в противном случае она называется *нелинейной пространственной фильтрацией*.

Линейная пространственная фильтрация

Понятие линейной фильтрации тесно связано с преобразованием Фурье при обработке сигналов в частотной области. Использование термина *линейная пространственная фильтрация* подчеркивает отличие этого процесса от *фильтрации в частотной области*.

Линейные операции состоят из умножения каждого пикселя окрестности на соответствующий коэффициент и суммирование этих произведений для получения результирующего *отклика* процесса в каждой точке (x, y) . Если окрестность имеет размер $m \times n$, то потребуется mn

коэффициентов. Эти коэффициенты сгруппированы в виде матрицы, которая называется *фильтром*, *маской*, *фильтрующей маской*, *ядром*, *шаблоном* или *окном*, причем первые три термина являются наиболее распространенными.

В пакете IPT линейная пространственная фильтрация реализована функцией **imfilter**, которая имеет следующий синтаксис:

```
B=imfilter(A, H)
B=imfilter(A, H, option1, option2, ...)
```

Функция **B=imfilter(A, H)** фильтрует многомерный массив A многомерным фильтром H. Массив A должен быть неразреженным числовым массивом любого формата и размерности. Результирующий массив B имеет ту же размерность и формат представления данных, что и массив A.

Каждый элемент результирующего массива B вычисляется с использованием чисел удвоенной точности с плавающей точкой. Если A представляет собой массив целых чисел, тогда элементы результирующего массива, превышающие допустимый диапазон, усекаются и округляются.

Функция **B=imfilter(A, H, option1, option2, ...)** выполняет многомерную фильтрацию в соответствии с заданными опциями. Аргументы опции могут принимать следующие значения.

Таблица 5. Границные опции.

<i>Опция</i>	<i>Описание</i>
X	Значения элементов внешних границ исходного массива принимают значения массива X. Когда граничные опции не определены, функция imfilter использует значение X=0.
'symmetric'	Значения элементов внешних границ исходного массива вычисляются как зеркальное отражение края этого массива.
'replicate'	Значения элементов внешних границ исходного массива допускаются равными по значениям ближайшим элементам края массива.
'circular'	Значения элементов внешних границ исходного массива вычисляются как периодическая структура исходного массива.

Таблица 6. Опции результирующих размеров.

<i>Опция</i>	<i>Описание</i>
'same'	Размеры результирующего массива совпадают с размерами исходного. Это свойство применяется по умолчанию, когда не указаны опции результирующих размеров.
'full'	Результирующий массив содержит полный результат фильтрации. Его размеры больше, чем у исходного массива.

Таблица 7. Опции корреляции и конволюции.

<i>Опция</i>	<i>Описание</i>
'corr'	Функция imfilter выполняет многомерную фильтрацию с

	использованием корреляции. Эта операция аналогична фильтрации, которая выполняется функцией <code>filter2</code> . Когда определены опции корреляции или конволюции, тогда функция <code>imfilter</code> использует корреляцию.
'conv'	Функция <code>imfilter</code> выполняет многомерную фильтрацию с использованием конволюции.

N-D конволюция относительно N-D корреляции является отражением матрицы фильтрации.

Рассмотрим пример функции `imfilter`:

```
rgb = imread('Lenna.png');
imshow(rgb), title('Original')
h = fspecial('motion', 10, 15);
rgb2 = imfilter(rgb, h);
figure, imshow(rgb2), title('Filtered')
rgb3 = imfilter(rgb, h, 'replicate');
figure, imshow(rgb3), title('Filtered with boundary replication')
```

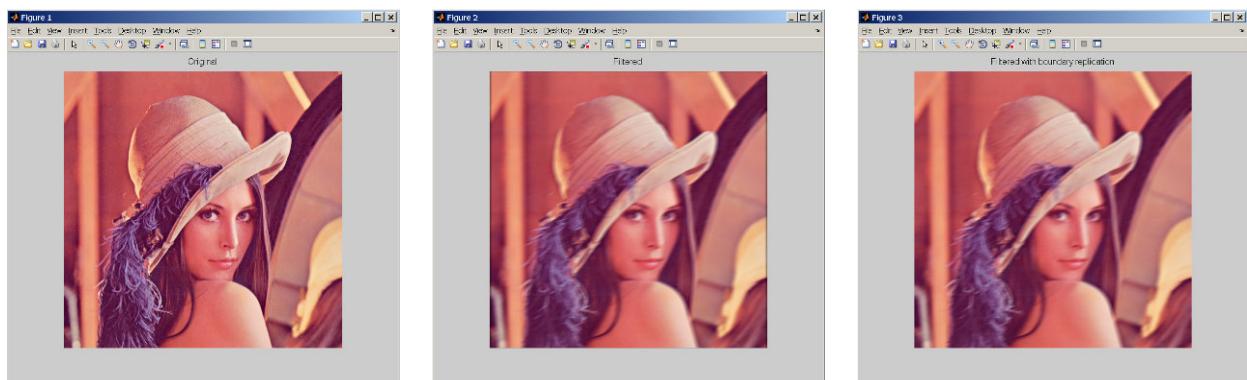


Рисунок 11 – Пример использования функции функции `imfilter`

Нелинейная пространственная фильтрация

Нелинейная пространственная фильтрация также основана на окрестностных операциях, причем механизм определения окрестности размера $m \times n$ и скольжения ее центра по изображению является таким же, как и в линейной фильтрации, описанной в предыдущем параграфе. Однако там, где линейная фильтрация использует сумму произведений (т. е. линейную операцию), нелинейная фильтрация основана (что следует из ее названия) на нелинейных операциях, совершаемых над пикселами текущей окрестности. Например, если положить, что отклик фильтра в каждой центральной точке равен максимальному значению в ее окрестности, то это определит нелинейную операцию фильтрации. Другое фундаментальное отличие состоит в том, что концепция маски не превалирует в нелинейных процессах. Идея фильтрации остается, но сам «фильтр» следует представлять себе в виде нелинейной функции, которая применяется к пикселам окрестности, и ее отклик состоит из отклика операции, примененной к центральному пикселу окрестности.

В пакете предусмотрены две функции для совершения общей нелинейной фильтрации: **nfilter** и **colfilt**. Первая из них совершаet операции непосредственно в матричной форме, а функция **colfilt** организует данные в форме столбцов. Хотя **colfilt** требует большего объема памяти, она выполняется существенно быстрее, чем **nfilter**. В большинстве приложений обработки изображений скорость вычисления является первостепенным фактором, поэтому функция **colfilt** является более предпочтительной при реализации общей нелинейной пространственной фильтрации.

Для заданного входного изображения f размера $M \times N$ и для заданной окрестности размера $m \times n$ функция **colfilt** строит матрицу, назовем ее A , максимального размера $m \times MN^2$, в которой каждый столбец соответствует пикселам, окруженным окрестностью с центром в некоторой точке изображения. Например, первый столбец соответствует пикселам, окруженным окрестностью, центр которой расположен в самом левом верхнем углу изображения f . Все необходимые расширения выполняются функцией **colfilt** (с помощью нулевых добавок).

Синтаксис функции **colfilt** имеет вид

```
D=colfilt(S, [m n], block_type, fun)
D=colfilt(S, [m n], block_type, fun, P1, P2, ...)
Xd=colfilt(Xs, 'indexed', ...)
```

Функция **D=colfilt(S, [m n], block_type, fun)** выполняет операции фильтрации, полностью аналогичные выполняемым функциям **blkproc** или **nfilter**, но значительно быстрее. Она предназначена для обработки полутоночных и бинарных изображений. Увеличение скорости обработки достигается за счет того, что обработке подвергается вспомогательное изображение, в котором каждый столбец представляет собой фрагмент исходного изображения S , передаваемый в функцию **fun**. Такой подход позволяет существенно уменьшить количество операций по чтению и записи отдельных пикселей изображения. Кроме того, каждый столбец может обрабатываться независимо от соседних. Для преобразования исходного изображения во вспомогательное и обратно в функции **colfilt** используются соответственно функции **im2col** и **col2im**.

Режим работы функции **colfilt** определяется значением параметра **block_type**:

- ‘*distinct*’ - функция **colfilt** аналогична функции **blkproc**, параметрами $[m n]$ задается размер неперекрывающихся блоков изображения;
- ‘*sliding*’ - функция **colfilt** аналогична функции **nfilter**, параметрами $[m n]$ задается размер маски фильтра.

Функция **D=colfilt(S, [m n], block_type, fun, P1, P2, ...)** позволяет передавать дополнительные параметры $P1, P2$ и так далее при вызове функции **fun**.

Формат представления данных изображений S и D определяется реализацией функции **fun**.

Аналогично функциям **blkproc** и **nfilter** с параметром ‘indexed’ функция **Xd=colfilt(Xs, ‘indexed’, ...)** предназначена для обработки палитровых изображений и при проведении вычислений дополняет изображение либо единицами при формате представления данных **Xs-double**, либо нулями при формате представления данных **Xs-uint8**.

Рассмотрим пример использования функции **colfilt**:

```
I = imread('tire.tif');
imshow(I)
I2 = uint8(colfilt(I,[5 5], 'sliding', @mean));
figure, imshow(I2)
```

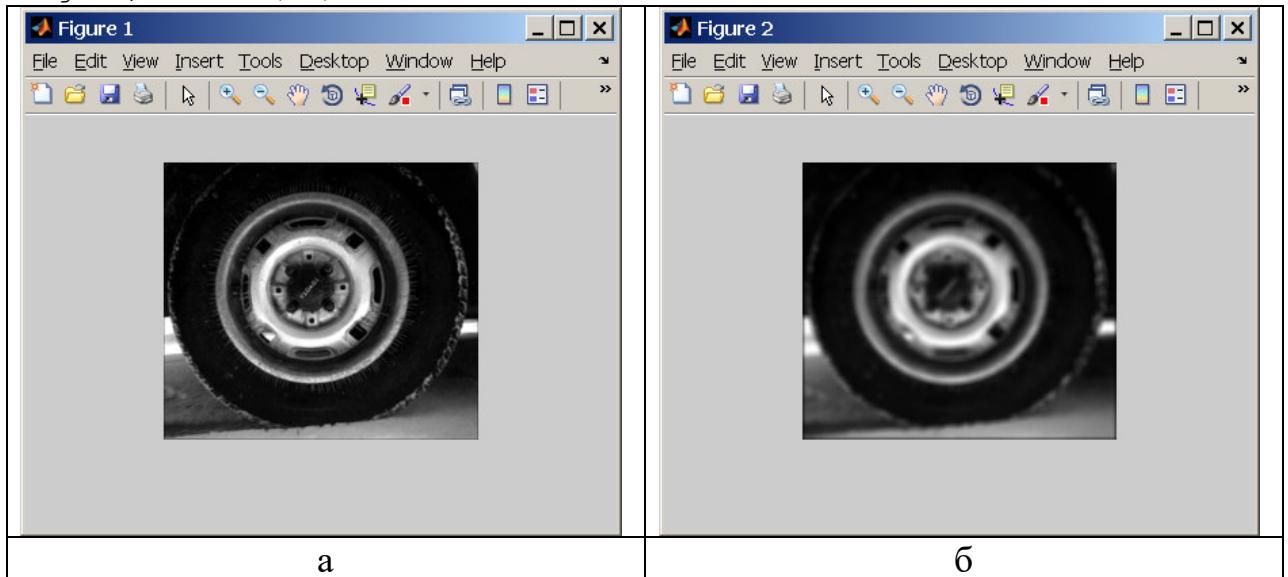


Рисунок 12 – Пример использования функции **colfilt**
а) исходное изображение; б) результат фильтрации

Линейная фильтрация имеет средства для расширения изображений, чтобы разрешать граничные проблемы, присущие пространственной фильтрации. Однако при использовании **colfilt** необходимо заранее расширить входное изображение до его фильтрации. Для этого служит функция **padarray**, которая в двумерном случае имеет синтаксис

```
B=padarray(A, padszie, method, direction)
```

Функция **B=padarray(A, padszie, method, direction)** наполняет массив **A** в соответствии с параметром **padszie** некоторым числом элементов вдоль каждого направления, используя описанный метод и направление наполнения. **N**-й элемент вектора **padszie** описывает размер наполнения для **N**-ого направления массива **A**.

Параметр **method** может быть числовым скаляром, в этом случае он описывает значения для всех элементов. Кроме числового скаляра, этот параметр может быть представлен одним значением, определяющим метод наполнения значений. По умолчанию, **method** устанавливает значение наполнения 0.

Таблица 8.

Значение	Описание
'circular'	Наполнение с круговым повторением элементов в данном направлении.
'replicate'	Наполнение с повторением граничных элементов массива.
'symmetric'	Наполнение массива с зеркальным отражением самого себя.

Параметр `direction` описывает вдоль какого направления наполняется массив. По умолчанию этот параметр задан как '`post`'.

Таблица 9.

Значение	Описание
'pre'	Наполнение впереди первого элемента по каждому направлению.
'post'	Наполнение позади последнего элемента вдоль каждого направления. Это значение устанавливается по умолчанию.
'both'	Наполнение впереди первого и позади последнего элементов вдоль каждого направления.

Когда выполняется наполнение константами, тогда массив `A` должен быть числовым или логическим. Когда производится наполнение с использованием методов '`circular`', '`replicate`' или '`symmetric`', тогда массив `A` может быть представлен в произвольном формате. Массив `B` имеет тот же формат, что и `A`.

Рассмотрим примеры использования функции **padarray**.

Пример 1.

```
a = [ 1 2 3 4 ];
b = padarray(a,[0 3], 'symmetric', 'pre')
```



Рисунок 13 – Пример использования функции **padarray** (пример 1)

Пример 2.

```
A = [1 2; 3 4];
B = padarray(A, [3 2], 'replicate', 'post')
```

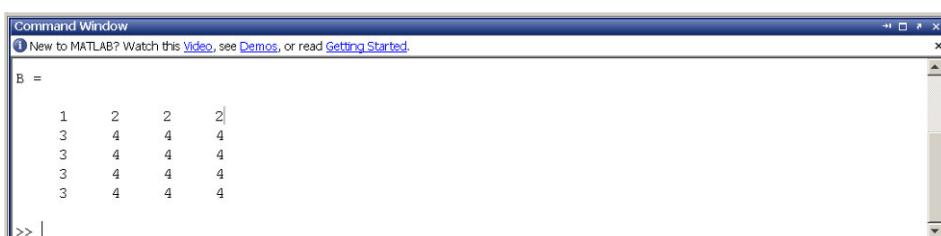


Рисунок 14 – Пример использования функции **padarray** (пример 2)

Линейные пространственные фильтры

В пакете IPT имеются некоторые стандартные двумерные линейные пространственные фильтры, которые можно получить из функции **fspecial**, которая генерирует маску фильтра h при выполнении команды

```
h=fspecial(type, P1, P2)
```

Функция **h=fspecial(type, P1, P2)** возвращает маску h предопределенного двумерного линейного фильтра, задаваемого строкой type. Маска h предназначена для передачи в функции **filter2** или **conv2**, выполняющих двумерную линейную фильтрацию. В зависимости от типа фильтра для него могут быть определены один или два дополнительных параметра $P1, P2$. Ниже будут рассмотрены возможные варианты функции **fspecial**.

Функция **h=fspecial('gausian', n, sigma)** возвращает маску h фильтра нижних частот Гаусса. Размер маски определяет параметр n . Если n - двухкомпонентный вектор, то размер маски $n(1) \times n(2)$. Если n скаляр, то размер маски $n \times n$. Параметр $sigma$ задает среднеквадратическое отклонение распределения Гаусса, которое используется при формировании маски h . Если при вызове функции параметры n и $sigma$ опущены, то размер маски устанавливается равным 3×3 , а среднеквадратическое отклонение 0.5 .

Функция **h=fspecial('sobel')** возвращает маску фильтра Собеля для выделения горизонтальных границ:

$$h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Для выделения вертикальных границ достаточно транспонировать данную маску h .

Функция **h=fspecial('prewitt')** возвращает маску фильтра Превита для выделения горизонтальных границ:

$$h = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}.$$

Для выделения вертикальных границ достаточно транспонировать данную маску h .

Функция **h=fspecial('laplacian', a)** возвращает маску h ВЧ фильтра Лапласа. Размер маски 3×3 . Параметр a управляет соотношением между центральным и граничными элементами маски. Данный параметр должен устанавливаться в диапазоне $[0, 1]$. По умолчанию $a=0.2$.

Функция **h=fspecial('log', n, sigma)** возвращает маску h фильтра, аналогичного последовательному применению фильтров Гаусса и Лапласа, так называемого лапласиана-гауссиана. Размер маски определяет параметр n . Если n - двухкомпонентный вектор, то размер маски $n(1) \times n(2)$. Если n - скаляр, то размер маски $n \times n$. Параметр $sigma$ задает среднеквадратическое

отклонение Гаусса, которое используется при формировании маски h . Если при вызове функции параметры n и σ опущены, то размер маски устанавливается равным 5×5 , а среднеквадратическое отклонение 0.5.

Функция $h=f\text{special}(\text{'average'}, n)$ возвращает маску h усредняющего НЧ фильтра. Размер маски определяет параметр n . Если n - двухкомпонентный вектор, то размер маски $n(1) \times n(2)$. Если n - скаляр, то размер маски $n \times n$. Если при вызове функции параметр n опущен, то размер маски устанавливается равным 3×3 .

Функция $h=f\text{special}(\text{'unsharp'}, a)$ возвращает маску h фильтра, повышающего резкость изображения. Размер маски 3×3 . Параметр a управляет соотношением между центральным и граничными элементами маски. Данный параметр должен устанавливаться в диапазоне $[0, 1]$. По умолчанию $a=0.2$.

Усредняющий фильтр относится к фильтрам низких частот. Он предназначен для фильтрации высокочастотного шума, и его работа сопровождается размытием изображения. Каждый элемент маски равен $1/MN$, где M и N - размеры маски (количество строк и столбцов).

Фильтр Гаусса также относится к НЧ фильтрам. В отличие от усредняющего фильтра он в меньшей мере размывает обрабатываемое изображение [1]. Маска фильтра такова, что центральный элемент маски имеет наибольшее значение, он соответствует пику распределения Гаусса. Значения остальных элементов уменьшаются по мере удаления от центрального элемента. Уменьшение происходит в соответствии с распределением Гаусса. Маска формируется с использованием следующих соотношений:

$$h_g(r,c) = e^{-\frac{(r^2+c^2)}{(2\sigma^2)}}; \quad h(r,c) = \frac{h_g(r,c)}{\sum_{r=1}^M \sum_{c=1}^N h_g(r,c)},$$

где M и N - размеры маски;

σ - среднеквадратическое отклонение распределения Гаусса.

Фильтр Лапласа относится к ВЧ фильтрам и предназначен для выделения границ (перепадов) во всех направлениях [1]. Маска фильтра конструируется следующим образом:

$$h = \frac{4}{(a+1)} \begin{bmatrix} \frac{a}{4} & \frac{1-a}{4} & \frac{a}{4} \\ \frac{1-a}{4} & -1 & \frac{1-a}{4} \\ \frac{a}{4} & \frac{1-a}{4} & \frac{a}{4} \end{bmatrix},$$

где a - параметр в диапазоне $[0,1]$, передаваемый в функцию `fspecial`.

Лапласиан-гауссиан также относится к ВЧ фильтрам, но в отличие от фильтра Лапласа выделяет более резкие перепады. Маска фильтра создается по формуле

$$h(r, c) = \frac{(r^2 + c^2 - 2\sigma^2)h_g(r, c)}{2\pi\sigma^6 \sum_{r=1}^M \sum_{c=1}^N h_g(r, c)},$$

где M и N - размеры маски;

σ - среднеквадратическое отклонение распределения Гаусса.

Формула для вычисления h_g выведена выше.

Маска фильтра, повышающего резкость изображения, создается следующим образом:

$$h = \frac{1}{(1+\alpha)} \begin{bmatrix} -\alpha & \alpha-1 & -\alpha \\ \alpha-1 & \alpha+5 & \alpha-1 \\ -\alpha & \alpha-1 & -\alpha \end{bmatrix},$$

где α - параметр в диапазоне [0,1], передаваемый в функцию fspecial.

Таблица 10. Пространственные фильтры функции fspecial

Тип	Синтаксис и параметры
'average'	fspecial ('average', [r c]). Прямоугольный усредняющий фильтр размера $r \times c$. По умолчанию 3×3 . Одно число на месте [r c] означает квадратный фильтр.
'disk'	fspecial ('disk', r). Круговой усредненный фильтр (внутри квадрата со стороной $2r + 1$) радиуса r. По умолчанию $r = 5$.
'gaussian'	fspecial ('gaussian', [r c], sig). Низкочастотный гауссов фильтр размера $r \times c$ со стандартным (положительным) отклонением sig. Значение по умолчанию 3×3 и 0.5. Одно число на месте [r c] означает квадратный фильтр.
'laplacian'	fspecial ('laplacian', alpha). Фильтр Лапласа 3×3 , форма которого задается параметром alpha из интервала [0, 1]. По умолчанию $alpha = 0.5$.
'log'	fspecial ('log', [r c], sig). Лаплас от гауссова фильтра (LoG) размера $r \times c$ со стандартным (положительным) отклонением sig. Значение по умолчанию 5×5 и 0.5. Одно число на месте [r c] означает квадратный фильтр.
'motion'	fspecial ('motion', len, theta). Выдает фильтр, который, будучи свернутым с изображением, приближает линейное перемещение (видеокамеры по отношению к изображению) на len пикселов. Направление перемещения задается углом theta, который измеряется в градусах от горизонтали против часовой стрелки. Значение по умолчанию 9 и 0, что соответствует перемещению на 9 пикселов в горизонтальном направлении.
'prewitt'	fspecial ('prewitt'). Выдает 3×3 маску Превитта wv, которая аппроксимирует вертикальный градиент. Маску горизонтального градиента можно получить, транспонировав результат wh = wv'.
'sobel'	fspecial ('sobel'). Выдает 3×3 маску Собела sv, которая

	аппроксимирует вертикальный градиент. Маску горизонтального градиента можно получить, транспонировав результат: $sh = sv'$.
'unsharp'	fspecial ('unsharp', alpha). Выдает 3×3 маску нечеткого фильтра. Параметр alpha контролирует форму, он должен быть не меньше 0 и не больше 1.0. По умолчанию alpha = 0.2.

Рассмотрим, каким образом влияет среднеквадратическое отклонение распределения Гаусса на элементы маски фильтра Гаусса, возвращаемые функцией fspecial. Получим значения маски фильтра Гаусса размера 5×5 для среднеквадратического отклонения 0.5 и 0.7. На рис. 1 а приведена частотная характеристика фильтра Гаусса с маской h05, а на рис. 1 б - частотная характеристика фильтра Гаусса с маской h07. Пример демонстрирует формирование маски фильтра Гаусса с помощью функции fspecial.

```
%Создание маски для фильтра Гаусса размера 5×5 и со
среднеквадратическим отклонением 0.5.
h05=fspecial('gaussian', 5, 0.5)
%Будет выведено:
%h05 =
% 0.0000 0.0000 0.0002 0.0000 0.0000
% 0.0000 0.0113 0.0837 0.0113 0.0000
% 0.0002 0.0837 0.6187 0.0837 0.0002
% 0.0000 0.0113 0.0837 0.0113 0.0000
% 0.0000 0.0000 0.0002 0.0000 0.0000
%Создание маски для фильтра Гаусса размера 5×5 и со
среднеквадратическим отклонением 0.7.
h07=fspecial('gaussian', 5, 0.7)
%Будет выведено:
%h07 =
% 0.0001 0.0020 0.0055 0.0020 0.0001
% 0.0020 0.0422 0.1171 0.0422 0.0020
% 0.0055 0.1171 0.3248 0.1171 0.0055
% 0.0020 0.0422 0.1171 0.0422 0.0020
% 0.0001 0.0020 0.0055 0.0020 0.0001
%Вывод на экран частотных характеристики линейных фильтров
%с масками h05 и h07.
freqz2(h05, [32 32]);
figure, freqz2(h07, [32 32]);
```

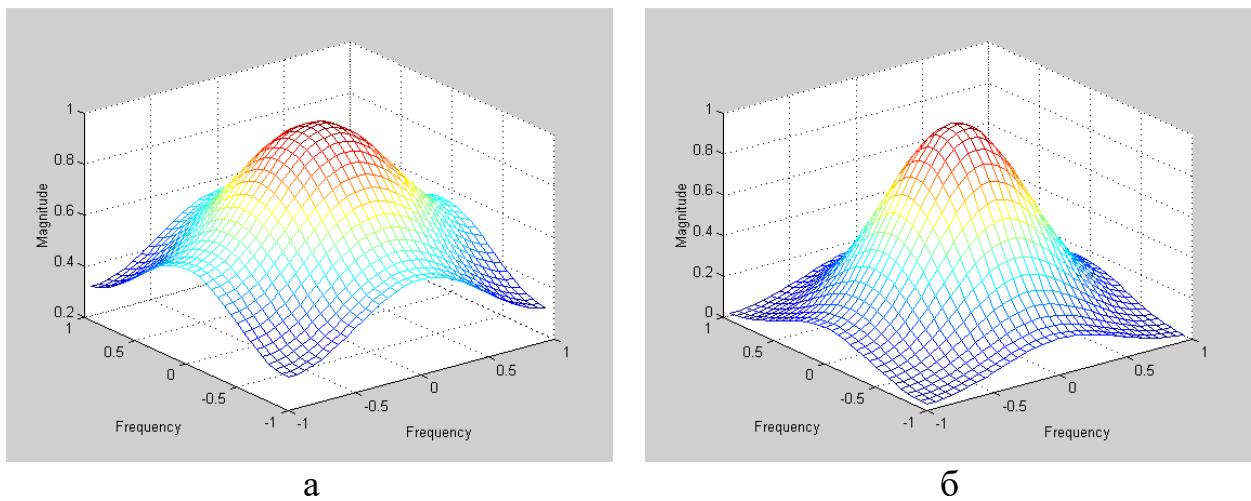


Рисунок 15 – Пример использования функции `fspecial`

Нелинейные пространственные фильтры

Наиболее употребительные нелинейные фильтры порождаются в пакете IPT функцией **`ordfilt2`**, которая строит *фильтры порядковых статистик* (их также называют *ранговыми фильтрами*). Отклики этих нелинейных пространственных фильтров основаны на предварительном упорядочении (ранжировании) пикселов изображения из текущей окрестности, после чего центральному пикселу присваивается значение, определенное в результате данного упорядочения. В этом параграфе все внимание будет сосредоточено на фильтрах, получающихся при помощи функции **`ordfilt2`**.

Синтаксис функции **`ordfilt2`** имеет следующий вид:

```
D=ordfilt2(S, order, domain);
D=ordfilt2(S, order, domain, S);
```

Функция **`D=ordfilt2(S, order, domain)`** создает полутонаовое изображение D, каждый пиксель которого формируется следующим образом. Пиксели исходного полутонаового изображения S, соответствующие ненулевым элементам маски фильтра `domain`, сортируются по возрастанию. Пикслю изображения D, соответствующему центральному элементу маски, присваивается значение с номером `order` в отсортированном множестве. Операция применяется нерекурсивно для всех положений маски. Фильтрацию такого вида называют *порядковой* или *ранговой*.

Для того чтобы размеры изображений S и D были одинаковыми, при проведении вычислений изображение S временно дополняется необходимым количеством строк и столбцов нулевых пикселей. Формат представления данных результирующего изображения D совпадает с форматом исходного изображения S.

Функция **`D=ordfilt2(S, order, domain, S)`** работает аналогично функции **`ordfilt2(S, order, domain, A)`**, за исключением того, что перед сортировкой к значениям пикселей, соответствующих ненулевым элементам маски фильтра `domain`, прибавляются значения из матрицы A. Матрицы `domain` и A должны

быть одинакового размера. Матрица A имеет формат представления данных double. Соответственно формат представления данных результирующего изображения D - также double.

Пример:

С помощью функции рангового фильтра можно осуществить морфологические операции эрозии и наращивания над полутоновым изображением. Эрозия соответствует замена значения центрального пикселя на минимальное из значений пикселей в пределах маски фильтра. Нарашению соответствует замена значения центрального пикселя на максимальное из значений пикселей в пределах маски фильтра.

На рис. 16,а приведено исходное изображение, на рис. 16,б и 16,в – соответственно результаты эрозии и наращивания фона с маской из единиц размера 3x3. Так как фон данного изображения серый, то эрозия приводит к уменьшению фона, а наращивание – к его увеличению. Пример демонстрирует операции эрозии и наращения над полутоновым изображением.

```
%Чтение исходного изображения и вывод его на экран.  
I=imread('image.tif');  
imshow(I);  
%Эрозия полутонового изображения.  
I_er=ordfilt2(I, 1, ones(3, 3));  
figure, imshow(I_er);  
%Нарашение полутонового изображения.  
I_dil=ordfilt2(I, 9, ones(3, 3));  
figure, imshow(I_dil);
```



Рисунок 16 – Пример использования функции **ordfilt2**

Медианная фильтрация весьма эффективна при удалении с изображений импульсных шумов, которые иногда называют шумами «соль и перец». Медианная фильтрация с функцией **medfilt2** имеет следующий синтаксис:

```
D=medfilt2(S, [m n])  
Xd=medfilt2(Xs, 'indexed', ...)
```

Медианная фильтрация является частным случаем ранговой фильтрации.

Функция **D=medfilt2(S, [m n])** создает полутоновое изображение D, каждый пиксель которого формируется следующим образом. Пиксели исходного полутонового изображения S, соответствующие всем элементам маски фильтра размера m×n, составляют упорядоченную последовательность A. Пикслю D(r, c), где r и с – координаты текущего положения центрального элемента маски, присваивается значение медианы последовательности A. Операция применяется нерекурсивно для всех положений маски.

Медианой упорядоченной последовательности A(i), где $i=1\dots N$, называется величина $A((N+1)/2)$, если N – нечетное, и $(A(N/2)+A((N+2)/2))/2$, если N – четное.

Для того чтобы размеры изображений S и D были одинаковыми, при проведении вычислений изображение S временно дополняется необходимым количеством строк и столбцов нулевых пикселей. Формат представления данных результирующего изображения D совпадает с форматом исходного изображения S.

Если вектор [m n] при вызове функции **D=medfilt2(S)** не задан, то в качестве маски фильтра используется маска размера 3×3.

Функция **Xd=medfilt2(Xs, 'indexed', ...)** аналогична рассмотренной выше, но предназначена для обработки палитровых изображений. При проведении вычислений исходное изображение временно дополняется либо единицами при формате представления данных Xs–double, либо нулями при формате представления данных Xs–uint8.

Пример:

Медианная фильтрация может эффективно применяться для устранения импульсного шума [1, 2]. На рис. 1 б приведен результат медианной фильтрации с маской 3×3 изображения на рис. 1, а. Пример демонстрирует работу медианного фильтра.

```
% Чтение исходного изображения и вывод его на экран.  
I=imread('bacteria.tif');  
imshow(I);  
% Медианная фильтрация.  
I=medfilt2(I);  
% Вывод на экран результата фильтрации.  
figure, imshow(I);
```

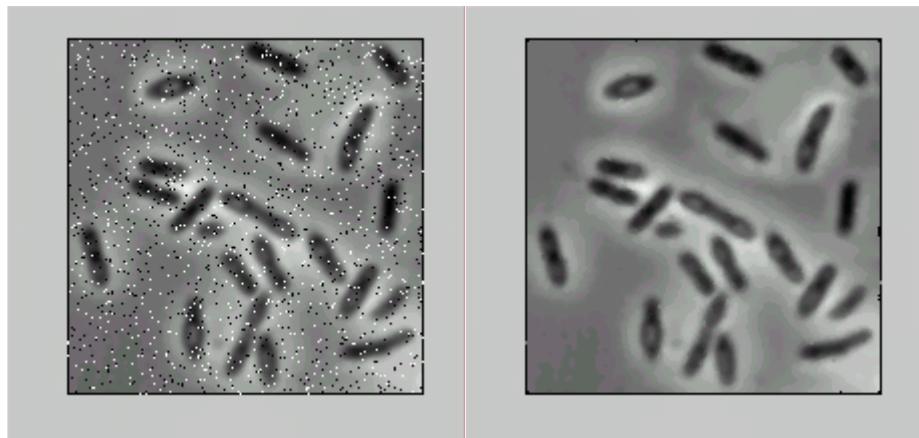


Рисунок 17 – Работа медианного фильтра

Общая постановка задачи

В результате выполнения заданий лабораторной работы студенты **должны уметь** создавать программно-алгоритмическую поддержку для компьютерной реализации требуемых методов преобразования изображений в пространственной области в MatLab.

Список индивидуальных данных

Написать файл-функцию и GUI интерфейс для решения следующих задач.

1. Выполните чтение изображения из файла, самостоятельно выбранного студентом. Используйте функцию **imshow** для вывода изображений на экран.
2. С помощью функции **Imadjust** увеличьте контраст изображения путем изменения диапазона интенсивностей исходного изображения.
3. С помощью функции **Imhist** постройте гистограмму исходного изображения.
4. помощью функции **Bar** постройте столбчатые диаграммы для исходного изображения.
5. С помощью функции **Stem** постройте стеблевые диаграммы для исходного изображения.
6. Выполните эквализацию гистограммы функцией **histeq**.
7. Выполните линейную пространственную фильтрацию с помощью функции **imfilter**.
8. Выполните нелинейную пространственную фильтрацию с помощью функции **colfilt**.
9. Используя линейные пространственные фильтры **fspecial** выполните фильтрацию пространственными фильтрами согласно варианту:

№ варианта	Пространственные фильтры
1.	average, disk, gaussian
2.	laplacian, log, motion
3.	prewitt, sobel, unsharp

4.	average, laplacian, prewitt
5.	disk, log, sobel
6.	gaussian, motion, unsharp
7.	average, log, unsharp
8.	disk, motion, prewitt
9.	gaussian, laplacian, sobel
10.	gaussian, log, prewitt

10. Используя нелинейные пространственные фильтры **ordfilt2** выполните операции:

- эрозии полутонового изображения;
- наращения полутонового изображения.

11. Выполните медианную фильтрацию с помощью функции **medfilt2**.

12. Сохраните полученные изображения и графики.

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.

Лабораторная работа №5. Преобразование изображений в области пространственных частот

Цель работы:

Целью лабораторной работы является получение навыков самостоятельной алгоритмической и программной реализации на компьютерной технике методов преобразования изображений в частотной области в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-5	3-1	Способен к восприятию и воспроизведению информации

	3-2	Знает алгоритмы целеполагания и выбора путей их достижения
ПК-25	3-1	Знает различные математические методы решения прикладных задач
	3-2	Знает методы формализации решения прикладных задач

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-5	У-1	Знает методы оценки проектов по информатизации
	У-2	Знает методы оценки проектов по автоматизации решения прикладных задач
ПК-14	У-1	Знает методы анализа прикладной области на логическом уровне
	У-2	Знает методы анализа прикладной области на алгоритмическом уровне
ПК-25	У-1	Знает различные математические методы решения прикладных задач
	У-2	Знает методы формализации решения прикладных задач

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-1	B-1	Знает основы современных технологий сбора, обработки и представления информации
	B-2	Способен ориентироваться в информационном потоке в глобальных компьютерных сетях
ПК-2	B-1	Способен анализировать социально-экономические проблемы
	B-2	Знает методы расчетов математических моделей

Теоретическая часть

Находясь в основе методов линейной фильтрации, преобразование Фурье обеспечивает значительную гибкость при разработке и реализации

алгоритмов фильтрации при решении задач улучшения, восстановления и сжатия изображений. Преобразование Фурье также лежит в фундаменте великого множества других важных практических приложений.

Двумерное дискретное преобразование Фурье

Пусть $f(x, y)$, при $x = 0, 1, 2, \dots, M - 1$ и $y = 0, 1, 2, \dots, N - 1$, обозначает изображение $M \times N$. Двумерное дискретное преобразование Фурье (DFT, Descrete Fourier Transform) изображения f , которое обозначается $F(u, v)$, задается уравнениями

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

при $u = 0, 1, 2, \dots, M - 1$ и $v = 0, 1, 2, \dots, N - 1$. Мы могли бы разложить экспоненту на синусы и косинусы от переменных u и v с соответствующими частотами (переменные x и y уйдут после суммирования). Частотной областью называется координатная система, задающая аргументы $F(u, v)$ частотными переменными u и v . Здесь можно обнаружить аналогию с заданием аргументов $f(x, y)$ пространственными переменными x и y . Прямоугольную область размера $M \times N$, задаваемую при $u = 0, 1, 2, \dots, M - 1$ и $v = 0, 1, 2, \dots, N - 1$, принято называть *частотным прямоугольником*. Видно, что частотный прямоугольник имеет те же размеры, что и исходное изображение.

Обратное дискретное преобразование Фурье задается уравнениями

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

при $x = 0, 1, 2, \dots, M - 1$ и $y = 0, 1, 2, \dots, N - 1$. Таким образом, зная $F(u, v)$, можно восстановить $f(x, y)$ с помощью обратного DFT. Величины $F(u, v)$ в этих уравнениях принято называть *коэффициентами разложения Фурье*.

В некоторых определениях коэффициент $1/MN$ помещается в прямое преобразование, а в других — в обратное. Для совместимости с реализацией преобразования Фурье в системе MATLAB мы будем считать, что коэффициент $1/MN$ расположен в формулах обратного преобразования, как это приведено выше. Поскольку индексы массивов в MATLAB начинаются с 1, а не с 0, в MATLAB формулы $F(1,1)$ и $f(1,1)$ соответствуют математическим величинам $F(0,0)$ и $f(0,0)$, которые стоят в прямом и обратном преобразованиях.

Значение преобразования Фурье в начале координат частотной области (т.е. величина $F(0,0)$) называется коэффициентом или компонентой *dc* преобразования Фурье. Эта терминология пришла из электротехники, где «dc» означает «direct current», постоянный электрический ток (ток с нулевой частотой). Легко показать, что величина $F(0,0)$ равна числу MN , умноженному на среднее значение функции $f(x, y)$.

Далее если изображение $f(x, y)$ вещественно, его преобразование Фурье является, как правило, комплексным. Основной метод визуального анализа этого преобразования заключается в вычислении его *спектра* (т. е.

абсолютной величины $|F(u, v)|$) и его отображения на дисплее. Пусть $R(u, v)$ и $I(u, v)$ обозначают вещественную и мнимую компоненты $F(u, v)$, тогда спектр Фурье задается выражением

$$|F(u, v)| = \sqrt{R^2(u, v) + I^2(u, v)}.$$

Фазовый угол (угол сдвига фазы) преобразования задается формулой

$$\phi(u, v) = \arctg \left[\frac{I(u, v)}{R(u, v)} \right].$$

Введенные выше функции можно использовать для представления $F(u, v)$ в стандартных полярных координатах разложения комплексных величин

$$F(u, v) = |F(u, v)| e^{j\phi(u, v)}.$$

Энергетический спектр (или *спектральная функция*) равен квадрату модуля

$$P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v).$$

С точки зрения визуализации не принципиально, какую функцию отображать на экране, $|F(u, v)|$ или $P(u, v)$.

Если $f(x, y)$ вещественно, то его преобразование Фурье является сопряженно-симметричным относительно начала координат, т. е.

$$F(u, v) = F^*(-u, -v),$$

что означает симметрию спектра Фурье относительно начала координат:

$$|F(u, v)| = |F(-u, -v)|.$$

Сделав прямую подстановку в уравнения для $F(u, v)$, можно показать, что

$$F(u, v) = F(u + M, v) = F(u, v + N) = F(u + M, v + N).$$

Другими словами, функция DFT является периодической по обеим переменным u и v , периоды которой равны M и N соответственно. Обратное преобразование Фурье также имеет свойство периодичности:

$$f(u, v) = f(u + M, v) = f(u, v + N) = f(u + M, v + N).$$

Это свойство часто приводит к затруднениям, так как интуитивно не очевидно, что взятие обратного преобразования Фурье приводит к периодической функции. Стоит помнить, что это обстоятельство просто является математическим свойством прямого и обратного DFT. Заметьте также, что в конкретных реализациях вычисляется только один период DFT, поэтому мы будем работать с массивами размера $M \times N$.

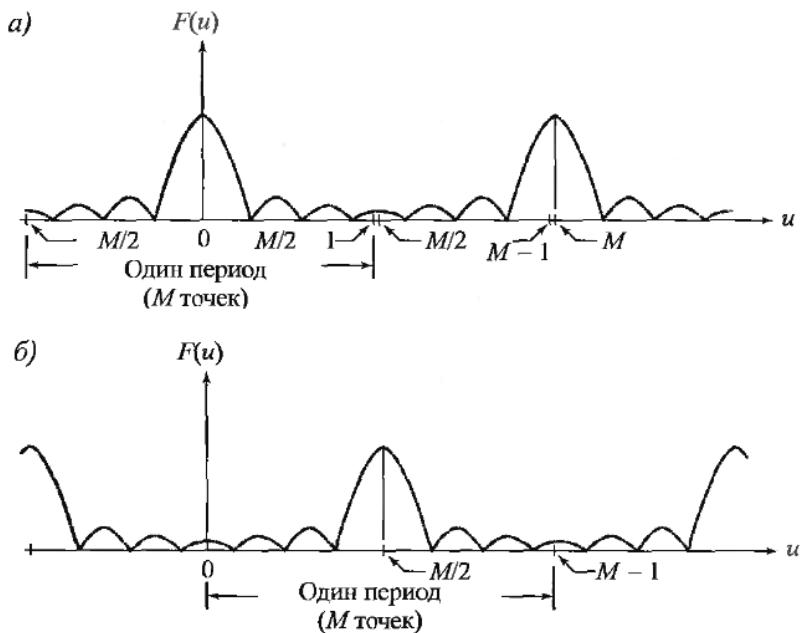


Рисунок 1 – а) Спектр Фурье показан составлением рядом двух полупериодов на интервале $[0, M - 1]$. б) Перемещение центра спектра в том же интервале, которое достигается умножением $f(x)$ на $(-1)^x$ до взятия преобразования Фурье

Свойство периодичности весьма важно при изучении того, как данные DFT связаны с периодами преобразования. Например, на рисунке 1а показан спектр $F(u)$ одномерного преобразования Фурье. В этом случае свойство периодичности выражается формулой $F(u) = F(u + M)$, откуда следует, что $|F(u)| = |F(u + M)|$, а из свойства симметрии заключаем, что $F(u) = F(-u)$. Свойство периодичности указывает на то, что $F(u)$ имеет период M , а свойство симметрии означает симметрию функции относительно начала координат, как показано на рисунке 1а. Этот график, а также предыдущие комментарии показывают, что модули величин преобразования от $M/2$ до $M - 1$ являются повтором величин из первой половины слева от начала координат. Поскольку в одномерном DFT необходимо вычислять M точек (т.е. для величин из интервала $[0, M - 1]$), то достаточно вычислить преобразование для первой половины точек. Нам нужен один, правильно упорядоченный период на интервале $[0, M - 1]$. Нетрудно, что нужный период получается умножением $f(x)$ на $(-1)^x$ до вычисления преобразования. На самом деле, все это означает перемещение начала координат преобразования в точку $u = M/2$, как показано на рисунке 1б. Теперь величина спектра при $u = 0$ на рисунке 1б соответствует $|F(-M/2)|$ на рисунке 1а. Аналогично, величины $|F(M/2)|$ и $|F(M - 1)|$ на рисунке 1б соответствуют величинам $|F(0)|$ и $|F(M/2 - 1)|$.

Похожая ситуация наблюдается для двумерных функций. Вычисление двумерного DFT дает точки преобразования в прямоугольной области, показанной на рисунке 2а, где затененная область показывает величины

$F(u,v)$, которые вычисляются по формулам, указанным в начале этого параграфа. Пунктирные прямоугольники получаются периодическим повторением, как на рисунке 1а. Затененная область показывает, что величины $F(u,v)$ распадаются на 4 периодических квадранта, которые соприкасаются в точке, указанной на рисунке 2а. Визуальный анализ спектра упрощается, если переместить начало координат преобразования в центр спектрального прямоугольника. Это можно сделать, умножив $f(x,y)$ на $(-1)^{x+y}$ до вычисления преобразования Фурье. Тогда периоды выстраиваются так, как показано на рисунке 2б. Как и при обсуждении одномерных функций, значение спектра в точке $(M/2, N/2)$ на рисунке 2б совпадает со значением в $(0,0)$ на рисунке 2а, а значение в $(0,0)$ на рисунке 2б совпадает со значением в $(-M/2, -N/2)$ на рисунке 2а. Аналогично, значение в точке $(M-1, N-1)$ на рисунке 2б равно значению в $(M/2-1, N/2-1)$ на рисунке 2а.



Рисунок 2 – а) Спектр Фурье размера $M \times N$ (затененная область); показан составлением 4 квадрантов. б) Спектр, полученный умножением $f(x,y)$ на $(-1)^{x+y}$ до взятия преобразования Фурье. Показан один период (затененный), поскольку только его необходимо вычислить по формуле

Приведенные выше рассуждения о центрировании преобразования с помощью умножения $f(x,y)$ на $(-1)^{x+y}$ представляют важную концепцию, которая здесь приводится для полноты изложения. Однако при работе с MATLAB подход заключается в вычислении преобразования без умножения на $(-1)^{x+y}$, после чего данные передвигаются с помощью функции *fftshift*.

Вычисление и визуализация двумерного DFT в MATLAB

Прямое и обратное DFT на практике вычисляются с помощью алгоритма быстрого преобразования Фурье (FFT, Fast Fourier Transform). Алгоритм FFT массива f изображения $M \times N$ реализован в пакете функцией *fft2*, которая имеет следующий синтаксис:

```

Y=fft2(X)
Y=fft2(X, m, n)

```

Функция ***Y=fft2(X)*** вычисляет двумерное ДПФ, возвращая результат в матрице комплексных чисел ***Y***, имеющей тот же размер, что и матрица ***X***; ***X*** может быть вектором, в этом случае возвращается вектор ***Y***, имеющий такую же ориентацию, что и вектор ***X***.

Функция ***Y=fft2(X, m, n)*** вычисляет двумерное преобразование Фурье матрицы ***mxn***, при необходимости дополняя нулями или усекая исходную матрицу ***X***. Возвращается матрица ***Y*** размера ***mxn***.

Массив ***Y*** имеет формат представления данных ***double***.

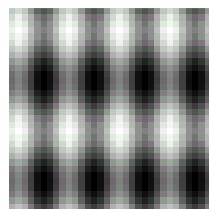
Двумерное преобразование Фурье вычисляется выполнением двух одномерных преобразований ***fft(fft(x).')***; сначала вычисляется ДПФ каждого столбца, а затем каждой строки результата [1, 2].

Рассмотрим пример. Сформируем двумерный сигнал, являющийся суммой двух пространственных волн. Выведем его на экран в виде полутонаового изображения (рисунок 3а) и трехмерной поверхности (рисунок 3в). Затем вычислим спектр сигнала и выведем его на экран (рисунок 3д). Так как спектр симметричен, 4 отчетливых пика на рисунке 3д соответствуют двум частотам волн, из которых сформировано исходное изображение. Удалим из спектра пики, соответствующие одной из частот (рисунок 3е). Восстановим с помощью обратного преобразования Фурье из отредактированного спектра двумерный сигнал. Результат восстановления выводится на экран в виде полутонаового изображения (рисунок 3б) и трехмерной поверхности (рисунок 3г).

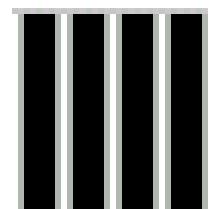
```

% Формирование исходного изображения
[x, y]=meshgrid(1:32);
I=sin(2*pi*x/8)+sin(2*pi*y/16);
% Вывод исходного изображения на экран.
imshow(mat2gray(I));
% Вывод исходного изображения в виде 3D-поверхности.
figure, mesh(x,y, I);
% Прямое преобразование Фурье.
h=fft2(I);
% Вывод на экран спектра исходного изображения.
figure, mesh(x,y, abs(h));
% Обнуление части спектра.
h(1:32,1:2)=0;
% Вывод на экран получившегося спектра.
figure, mesh(x,y, abs(h));
% Обратное преобразование Фурье.
I=ifft2(h);
% Вывод на экран результирующего изображения.
figure, imshow(mat2gray(I));
%return
% Вывод на экран результирующего изображения в виде 3D-поверхности.
figure, mesh(x, y, abs(I));

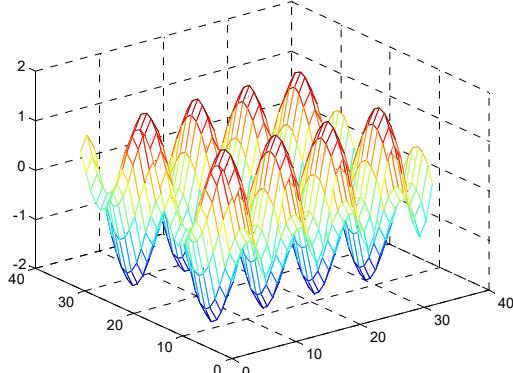
```



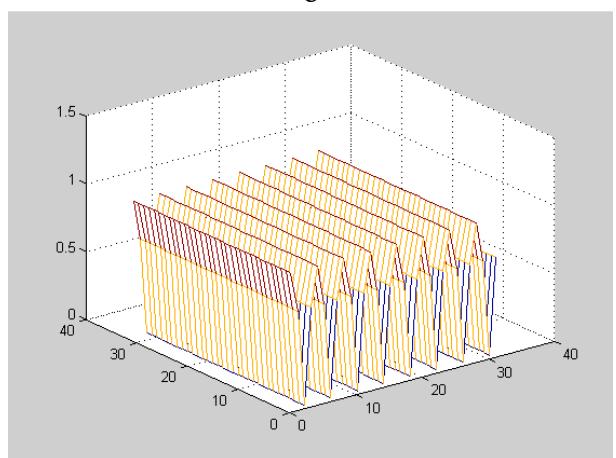
а



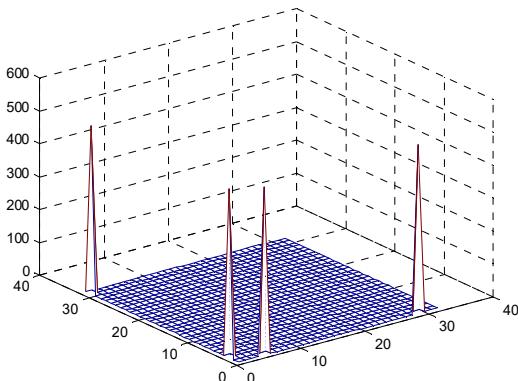
б



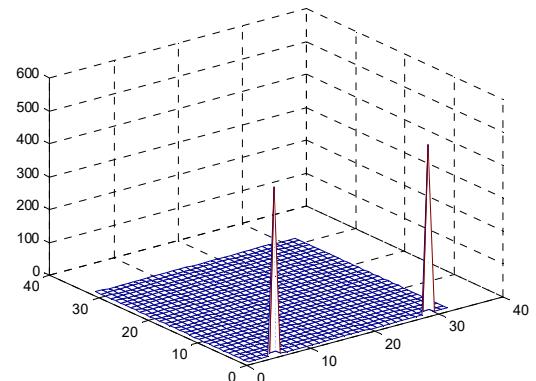
в



г



д



е

Рисунок 3 – Пример использования функции **fft2**

Функцию ***fftshift*** пакета можно использовать для смещения начала координат преобразования в центр частотной области. С помощью данной функции производится перегруппировка выходных массивов преобразований Фурье. Ее синтаксис имеет вид:

`Y = fftshift(X)`

Функция $Y = \text{fftshift}(X)$ перегруппировывает выходные массивы функций ***fft*** и ***fft2***, размещая нулевую частоту в центре спектра.

Если v – одномерный массив, то выполняется циклическая перестановка правой и левой его половины:

v	$\text{fftshift}(v)$
1 2 3 4 5	4 5 1 2 3

Если X – двумерный массив, то меняются местами квадранты: I \leftrightarrow IV и II \leftrightarrow III:

```
X = [ ' I '      ' II ' ; ...
      ' III '     ' IV ' ; ]
fftshift(X)
```

X		fftshift(X)	
I	II	IV	III
III	IV	II	I

Рассмотрим тот же пример, который рассматривался и для функции **fft**, но введем постоянную составляющую с уровнем 0.3:

```
t = 0:0.001:0.6;
x = sin(2 * pi * 50 * t) + sin(2 * pi * 120 * t);
y = x + 2 * randn(size(t)) + 0.3;
plot(y(1:50)), grid
```

Применим одномерное преобразование Фурье для сигнала y и построим график спектральной плотности. Здесь можно выделить 3 частоты, на которых амплитуда спектра максимальна. Это частоты 0, 58.4 и 140.1 Гц (рисунок 4а).

```
Y = fft(y);
Pyy = Y.*conj(Y)/512;
f = 1000 * (0:255)/512;
figure(1), plot(f, Pyy(1:256)), grid
ginput
```

58.4112	136.6337	140.1869	62.3762
58.4112	68.9109	140.1869	96.8317

Выполним операции

```
Y = fftshift(Y);
Pyy = Y.*conj(Y)/512;
figure(2), plot(Pyy), grid
```

Из анализа рисунка 4б следует, что нулевая частота сместились в середину спектра.

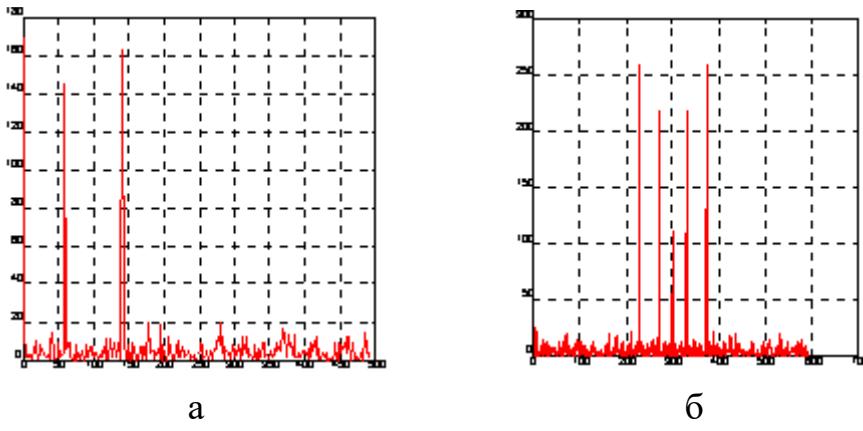


Рисунок 4 – Пример использования функции *fftshift*

Обратное преобразование Фурье вычисляется с помощью функции *ifft2*, синтаксис которой имеет вид:

```
Y=ifft2(X)
Y=ifft2(X, m, n)
```

Функция **Y=ifft2(X)** вычисляет обратное двумерное ДПФ, возвращая результат в матрице Y, имеющей тот же размер, что и матрица X; X может быть вектором, в этом случае возвращается вектор Y, имеющий такую же ориентацию, что и вектор X.

Функция **Y=ifft2(X, m, n)** вычисляет двумерное преобразование Фурье матрицы mxn, при необходимости дополняя нулями или усекая исходную матрицу X. Возвращается матрица Y размера mxn.

С точностью, определяющейся ошибками округления, **ifft2(ifft2(X))=X**, однако, если X содержит только действительные элементы, **ifft2(ifft2(X))** может содержать комплексные элементы с малыми по величине комплексными частями.

Массив Y имеет формат представления данных **double**.

Обратное двумерное преобразование Фурье вычисляется по тому же принципу, который реализован в функции *fft2*, т. е. двумерное преобразование выполняется с помощью двух последовательных одномерных обратных преобразований Фурье для строк и столбцов. Время расчетов минимально, если m и n являются степенями числа 2. Если m и n, простые числа – время расчетов максимально.

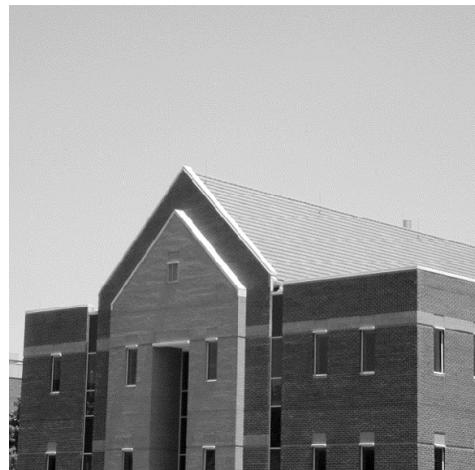
Рассмотрим пример вычисления и визуализации двумерного дискретного преобразования Фурье.

```
% чтение из файла
I = imread('picture.jpg');
% выбрать первую цветовую компоненту изображения в формате RGB
I1=I(:,:,1);
% Определить размеры изображения
[N1,N2]=size(I1)
% Вывод исходного изображения на экран
figure('Name','I'), imshow(I1, []);
```

```

% Прямое преобразование Фурье
h=fft2(I1);
% Вывод на экран спектра исходного изображения
figure('Name',' mesh abs(h)'), mesh(abs(h));
% Центрирование спектра
h1=fftshift(log(1+abs(h)));
% Вывод на экран центрированного спектра
figure('Name','imshow h1=fftshift(log(1+abs(h)))'), imshow(h1,[]);
% Обнуление части спектра
h0=fftshift(h);
h0(N1/2-10:N1/2+10,N2/2-10:N2/2+10)=0;
h2=ifftshift(h0);
% Вывод на экран получившегося спектра
h1=fftshift(log(1+abs(h2)));
figure('Name',' imshow h1=fftshift(log(1+abs(h))) with zeros'), imshow(abs(h1),[]);
% Обратное преобразование Фурье.
I2=real(ifft2(h2));
% Вывод на экран результирующего изображения.
figure('Name','I2')
imshow(I2,[]);

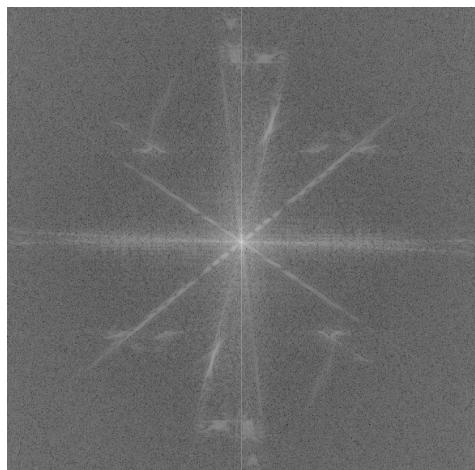
```



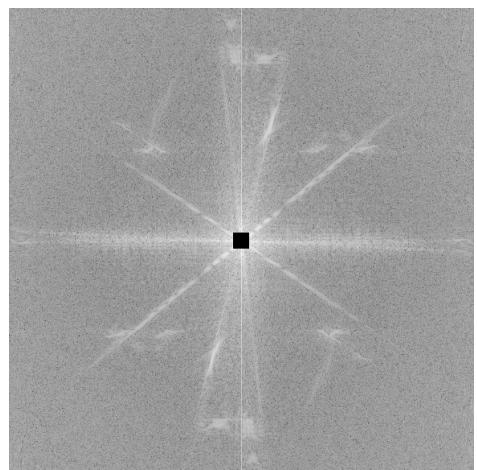
а



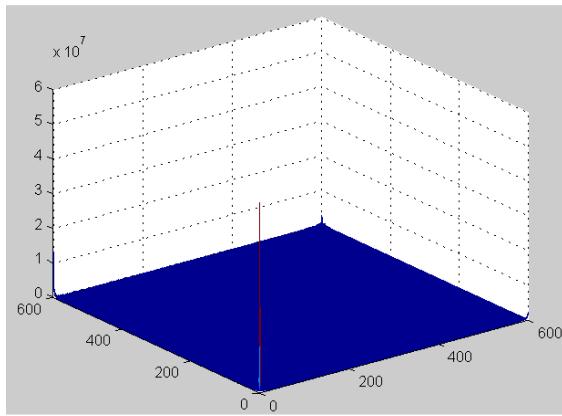
б



в



г



д

Рисунок 5 –Пример использования двумерного дискретного преобразования Фурье

- а) исходное изображение; б) результат преобразований; в) центрированный спектр исходного изображения; г) спектр с обнуленной частью; д) спектр исходного изображения

Фильтрация в области пространственных частот

Фильтрация в частотной области имеет весьма простую концепцию.

Основой линейной фильтрации в частотной и пространственной областях служит теорема о свертке, которую можно сформулировать так:

$$f(x, y) * h(x, y) \Leftrightarrow H(u, v)F(u, v),$$

и в обратную сторону

$$f(x, y)h(x, y) \Leftrightarrow H(u, v)* F(u, v).$$

Здесь символ «*» обозначает операцию свертки двух функций, а выражения по обе стороны от двойных стрелок определяют соответствующие пары при преобразовании Фурье. Например, первое выражение означает, что свертку двух пространственных функций можно получить, если вычислить обратное преобразование Фурье от произведения прямых преобразований Фурье этих двух функций. Наоборот, прямое преобразование Фурье свертки двух пространственных функций дает произведение их прямых преобразований Фурье.

Фильтрация в пространственной области состоит из свертки изображения $f(x, y)$ и маски $h(x, y)$. В соответствии с теоремой о свертке, тот же результат можно получить в частотной области, умножив $F(u, v)$ на $H(u, v)$ — преобразование Фурье пространственного фильтра. Принято называть $H(u, v)$ *передаточной функцией фильтра*.

Основная идея фильтрации в частотной области заключается в подборе передаточной функции фильтра, которая модифицирует $F(u, v)$ специфическим образом. Основываясь на теореме о свертке, мы знаем, что для получения соответствующего отфильтрованного изображения в пространственной области нам следует просто вычислить обратное преобразование Фурье от произведения $H(u, v)F(u, v)$. Результат этих действий будет идентичен выполнению операции свертки в пространственной области

с маской $h(x, y)$, которая есть обратное преобразование Фурье от $H(u, v)$. На практике пространственная фильтрация сильно упрощается при использовании малых масок, которые стремятся поймать выраженные характеристики их частотных двойников.

Изображения и их преобразования автоматически считаются периодическими, если фильтрация реализована на основе DFT. Нетрудно дать визуальные примеры свертки периодических функций, для которых возникают эффекты интерференции между смежными периодами, если эти периоды расположены близко относительно длины ненулевых частей функций. Эту интерференцию, которую принято называть *ошибкой возврата* или *ошибкой перекрытия*, можно подавить, дополняя функции нулями следующим образом.

Предположим, что функции $f(x, y)$ и $h(x, y)$ имеют размеры $A \times B$ и $C \times D$ соответственно. Мы формируем две *расширенные* функции, обе имеющие размеры $P \times Q$, путем добавления нулей к f и h . Можно показать, что ошибка перекрытия не происходит, если выбрать

$$P \geq A + C + 1 \text{ и } Q \geq B + D + 1.$$

В основном рассматриваются функции одинаковых размеров $M \times N$. В этом случае сформулированное правило имеет следующий вид:

$$P \geq 2M - 1 \text{ и } Q \geq 2N - 1.$$

Следующая функция, называемая *paddedsize*, вычисляет минимальные четные 4 значения для P и Q, которые удовлетворяют этим условиям. Имеется также опция, позволяющая дополнять входные данные до квадрата, длина стороны которого равна ближайшей степени числа 2. Время выполнения алгоритма FFT зависит, грубо говоря, от числа простых делителей P и Q. Этот алгоритм работает быстрее, когда P и Q являются степенями двойки, чем когда P и Q — простые числа. Кроме того, на практике можно посоветовать работать с квадратными изображениями и фильтрами, т.е. чтобы фильтрация имела одинаковые размеры по обоим измерениям. Функция *paddedsize* позволяет делать такой выбор достаточно гибко, исходя из входных параметров.

В функции *paddedsize* векторы AB, CD и PQ имеют элементы [A B], [C D] и [P Q] соответственно.

```
function PQ = paddedsize(AB, CD, PARAM)
if nargin == 1
    PQ = 2*AB;
elseif nargin == 2 & ~ischar(CD)
    PQ = AB + CD - 1;
    PQ = 2 * ceil(PQ / 2);
elseif nargin == 2
    m = max(AB);
    P = 2^nextpow2(2*m);
    PQ = [P, P];
elseif nargin == 3
    m = max([AB CD]);
    PQ = [m, m];
```

```

P = 2^nextpow2(2*m);
PQ = [P, P];
else
    error('Wrong number of inputs.')
end

```

Определив PQ с помощью функции *paddedsize*, вычислим быстрое преобразование Фурье по формуле

$$F = fft2(f, PQ(1), PQ(2)).$$

При этом к изображению f добавляется достаточное количество нулей, чтобы полученное изображение имело размеры $PQ(1) \times PQ(2)$, а затем вычисляется FFT. Отметим, что при использовании расширения функция фильтра в частотной области должна иметь размеры $PQ(1) \times PQ(2)$.

Пример 1. Эффекты фильтрации с использованием и без использования процедуры расширения.

Изображение f , приведенное на рисунке 5а, иллюстрирует фильтрацию с расширением и без расширения. Здесь также используется функция *lpfilter*, которая строит гауссов низкочастотный фильтр (подобный фильтру на рисунке 4б с заданным значением параметра сигма (sig)).

```

function H = lpfilter(type, M, N, D0, n)
disp('lpfilter')
[U, V] = dftuv(M, N);
D = sqrt(U.^2 + V.^2);
switch type
case 'ideal'
    H = double(D <= D0);
case 'btw'
    if nargin == 4
        n = 1;
    end
    H = 1./(1 + (D./D0).^(2*n));
case 'gaussian'
    disp('gaussian')
    H = exp(-(D.^2)./(2*(D0^2)));
otherwise
    error('Unknown filter type.')
end

```

Следующие команды совершают фильтрацию без расширения:

```

[M,N]=size(f);
F=fft2(f);
sig=10;
H=lpfilter('gaussian',M,N,sig);
G=H.*F;
g=real(ifft2(G));
figure (1)
imshow(f,[])

```

```
figure (2)
imshow(g, [])
```

Рассмотрим теперь фильтрацию с расширением. В этом примере параметр сигма равен $2*sig$, поскольку здесь размер фильтра в два раза больше, чем у фильтра без расширения.

```
PQ= paddedsize (size (f)) ;
Fp= fft2 (f, PQ(1),PQ(2));
Hp=lpfilter('gaussian',PQ(1),PQ(2),2*sig);
Gp=Hp.*Fp;
gp=real(ifft2(Gp));
gpc=gp(1:size(f,1),1:size(f,2));
figure (3)
imshow(gpc, [])
```

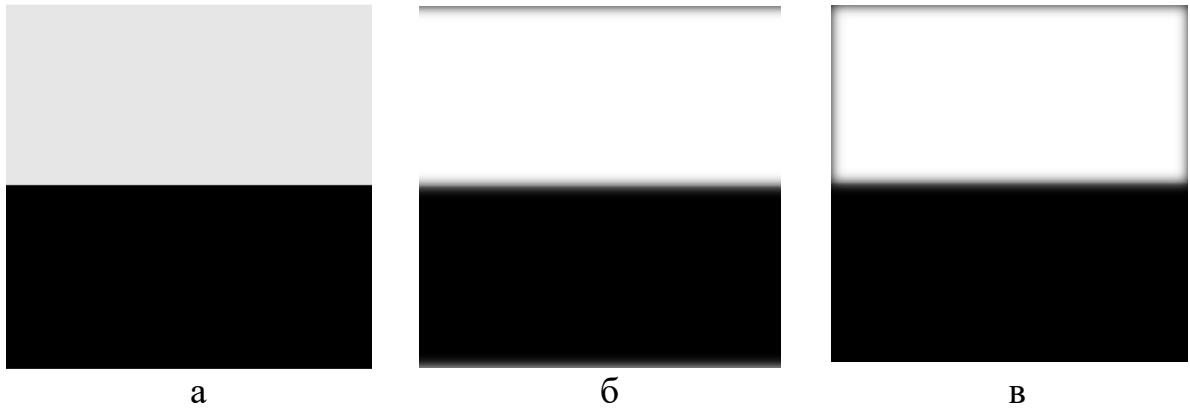


Рисунок 5 – Пример фильтрации с расширением

а) Простое изображение размера 256x256; б) Результат низкочастотной фильтрации без расширения; в) Результат низкочастотной фильтрации с расширением.

Основные шаги фильтрации в частотной области

Через f обозначим исходное изображение, а через g — результат фильтрации. Предполагается, что передаточная функция $H(u,v)$ имеет те же размеры, что и исходное изображение.

1. Получить параметры расширения с помощью *paddedsize*:

$$PQ = \text{paddedsize}(\text{size}(f));$$
2. Построить преобразование Фурье с расширением:

$$F = \text{fft2}(f, PQ(1), PQ(2));$$
3. Сгенерировать функцию фильтра H размера $PQ(1) \times PQ(2)$ одним из описываемых далее методов. Фильтр должен иметь формат, показанный на рис. 4.4, б). Если он был центрирован, до использования его в фильтрации следует выполнить команду $H = \text{fftshift}(H)$.
4. Умножить преобразование Фурье на передаточную функцию фильтра:

$$G = H.*F;$$

5. Найти вещественную часть обратного преобразования Фурье от G :

$$g = \text{real}(\text{ifft2}(G));$$

6. Вырезать верхний левый прямоугольник исходных размеров:

$$g = g(1 : \text{size}(f,1), 1 : \text{size}(f,2));$$

Эта процедура фильтрации схематически изображена на рисунке 6. Предварительная стадия обработки может состоять из определения размеров изображения, вычисления параметров расширения и генерации фильтра. Заключительная стадия обработки состоит в выделении вещественной части результата, обрезания изображения до исходного размера и его конвертации в класс *uint8* или *uint16* для сохранения на диске.

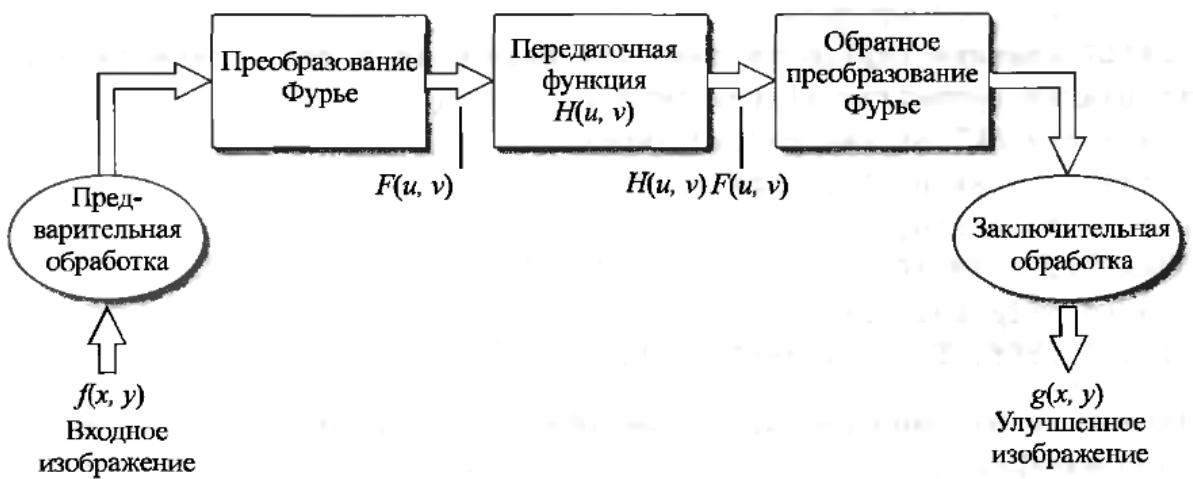


Рисунок 6 – Основные шаги фильтрации в частотной области

Передаточная функция фильтра $H(u, v)$ на рисунке 6 умножается на вещественную и мнимую части $F(u, v)$. Если функция $H(u, v)$ была вещественной, то фазовая часть произведения не меняется, что видно из фазового уравнения, так как при умножении вещественной и мнимой части комплексного числа на одно и то же вещественное число фазовый угол не меняется. Такие фильтры принято называть *фильтрами с нулевым сдвигом фазы*.

Из теории линейных систем хорошо известно, что при выполнении некоторых слабых условий подача одиночного импульса в линейную систему полностью описывает характеристики этой системы. При работе с конечными линейными системами, что имеет место в данной книге, отклик линейной системы, включая отклик на одиночный импульс, также является конечным. Если линейная система является пространственным фильтром, то можно полностью описать этот фильтр, подавая на вход одиночный импульс и наблюдая соответствующий отклик. Такие фильтры называются фильтрами с конечной импульсной характеристикой (FIR, Finite Impulse Response). Все пространственные фильтры в данной книге имеют тип FIR.

Рассмотрим функцию, аргументами которой являются изображение и передаточная функция фильтра, которая выполняет все необходимые

процедуры фильтрации и возвращает отфильтрованное и обрезанное изображение. Следующая функция как раз совершает все эти действия.

```
function g = dftfilt(f, H)
F = fft2(f, size(H, 1), size(H, 2));
g = real(ifft2(H.*F));
g = g(1:size(f, 1), 1:size(f, 2));
```

Построение фильтров в области пространственных частот по пространственным фильтрам

В общем случае фильтрация в пространственной области является более эффективной с вычислительной точки зрения, чем фильтрация в частотной области, когда используются «малые» фильтры. Точное определение этой *малости* является весьма трудным вопросом, ответ на который зависит от таких факторов, как тип компьютера и выбор алгоритма, а также от многих других параметров, например, от размеров буферов. Важную роль играет то обстоятельство, насколько удобно реализовано обращение с комплексными данными, а также множество других факторов, которые выходят за рамки нашего обсуждения. Фильтрация с использованием алгоритма быстрого преобразования Фурье FFT выполняется быстрее, чем пространственная реализация этого процесса, когда функция имеет примерно 32 точки, т.е. их число невелико. Поэтому полезно знать, как конвертировать пространственные фильтры в эквивалентную частотную форму, чтобы иметь возможность сравнивать эти два подхода.

Самый очевидный метод построения частотного фильтра H , отвечающего пространственному фильтру h , состоит в выполнении команды $H = fft2(h, PQ(1), PQ(2))$, где значения вектора PQ зависят от размеров изображения, которое будет фильтроваться. Тем не менее, в этом параграфе мы будем интересоваться следующими вопросами:

1. как преобразовать пространственный фильтр в частотный;
2. как сравнивать результаты, получаемые при пространственном фильтровании функцией *imfilter*, с результатами фильтрования в частотной области, которые обсуждались выше.

Раз функция *imfilter* использует корреляцию и начало координат фильтра расположено в его центре, то для достижения эквивалентных результатов при обоих подходах необходимо выполнить определенную предварительную обработку данных. В пакете имеется функция *freqz2*, которая в точности совершает эти действия, а выходом этой функции служит соответствующий фильтр в частотной области.

Функция *freqz2* вычисляет частотный отклик фильтра FIR, а в этой книге рассматриваются только такие фильтры. Тогда в результате получаем нужный фильтр в частотной области. Интересующая нас команда выглядит следующим образом:

```
[h, f1, f2] = freqz2(h, n1, n2)
```

```
[h, f1, f2]=freqz2(h, [n1 n2])
[H, f1, f2]=freqz2(h, f1, f2)
[...]=freq2(h, n1, n2, [dx dy])
[...]=freq2(h, n1, n2, dx)
[...]=freq2(h, f1, f2, [dx dy])
[...]=freq2(h, f1, f2, dx)
freq2(...)
```

Функция $[h, f1, f2]=\text{freqz2}(h, n1, n2)$ формирует матрицу H размера $n1 \times n2$, которая является амплитудно-частотной характеристикой (АЧХ) на частотах, содержащихся в векторах $f1$ и $f2$ линейного двумерного КИХ-фильтра с маской h . В векторы $f1$ и $f2$ функция помещает равномерно распределенные нормализованные значения частот в диапазоне $[-1, 1]$, где 1 соответствует половине частоты дискретизации (частоте Найквиста). Вектор $f1$ содержит $n1$ значений частоты, а вектор $f2$ содержит $n2$ значений частоты. Если при вызове функции параметры $n1$ и $n2$ опущены, то они берутся равными $n1=n2=64$.

Функция $[H, f1, f2]=\text{freqz2}(h, [n1 n2])$ полностью аналогична функции $[H, f1, f2]=\text{freqz2}(h, n1, n2)$.

Функция $[H, f1, f2]=\text{freqz2}(h, f1, f2)$ формирует матрицу H , которая является АЧХ на частотах $f1$ и $f2$ линейного двумерного КИХ-фильтра с маской h . В векторы $f1$ и $f2$ должны быть помещены нормализованные значения частот в диапазоне $[-1, 1]$. Матрица H имеет размер $\text{length}(f1) \times \text{length}(f2)$.

В функциях $[...]=\text{freq2}(h, n1, n2, [dx dy])$ и $[...]=\text{freq2}(h, f1, f2, [dx dy])$ параметры dx и dy задают диапазоны изменения значений по осям частот как $[-1/(2*dx), 1/(2*dx)]$ для оси абсцисс и $[-1/(2*dy), 1/(2*dy)]$ для оси ординат. Например, если dx равно 1, то нормализованные частоты по оси абсцисс принадлежат диапазону $[-0.5, 0.5]$, а если dx равно 5, то нормализованные частоты по оси абсцисс принадлежат диапазону $[-0.1, 0.1]$.

В функциях $[...]=\text{freq2}(h, n1, n2, dx)$ и $[...]=\text{freq2}(h, f1, f2, dx)$ параметр dx задает одинаковые диапазоны изменения значений по осям частот: $[-1/(2*dx), 1/(2*dx)]$.

Функция $\text{freq2}(...)$ без выходных параметров выводит в текущее окно на экране двумерную АЧХ.

Рассмотрим пример использования функции freq2 .

```
Hd = zeros(16,16);
Hd(5:12,5:12) = 1;
Hd(7:10,7:10) = 0;
h = fwind1(Hd,bartlett(16));
colormap(jet(64))
freqz2(h,[32 32]);
axis([-1 1 -1 1 0 1])
```

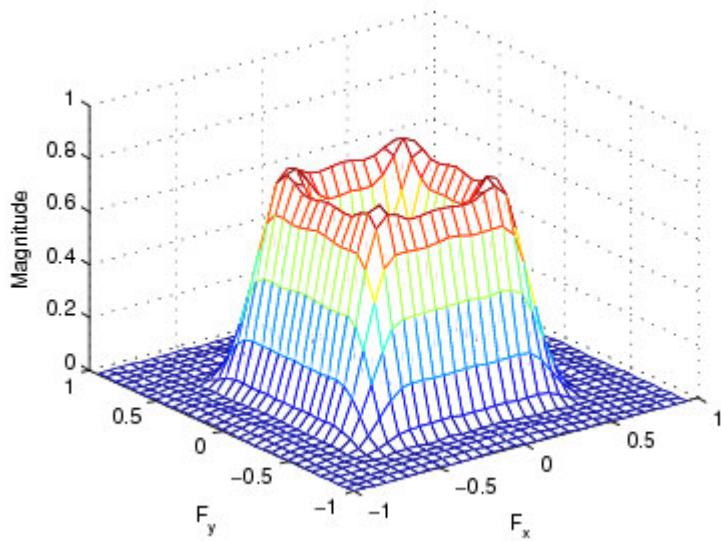


Рисунок 7 – Пример использования функции freq2.

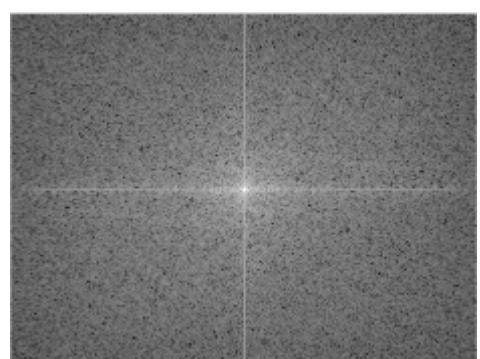
Пример 2. Сравнение фильтрации в пространственной и частотной областях.

Рассмотрим изображение f , приведенное на рисунке 8а. Сейчас мы сгенерируем фильтр H в частотной области, соответствующий пространственному фильтру Собела, который улучшает вертикальные края изображения. Затем мы сравним результаты фильтрации f в пространственной области с помощью маски Собела (используя *imfilter*) с результатами, полученными при совершении эквивалентного процесса в частотной области. На практике фильтрация с малыми фильтрами вроде маски Собела может быть реализована непосредственно в пространственной области, как говорилось раньше. Однако мы выбрали этот фильтр в демонстративных целях, так как он имеет простые коэффициенты, а результаты его фильтрации интуитивно ясны и легко поддаются сравнению. С большими фильтрами можно обращаться аналогично. На рисунке 8б дан спектр Фурье f , полученный следующими командами:

```
F=fft2(f);
S=fftshift(log(1+abs(F)));
S=gscale(S);
imshow(f), figure, imshow (S, []);
```



а



б

Рисунок 8 – а) Полутоновое изображение, б) его спектр Фурье

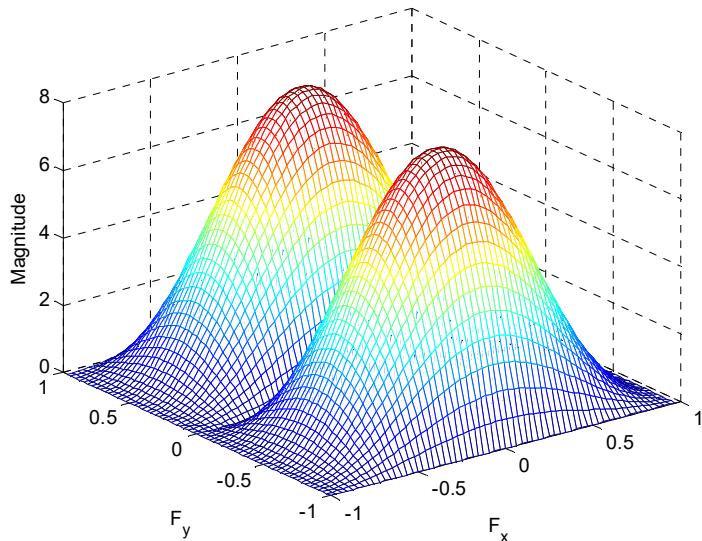
Теперь строим пространственный фильтр функцией *fspecial*:

```
h=fspecial('sobel')
```

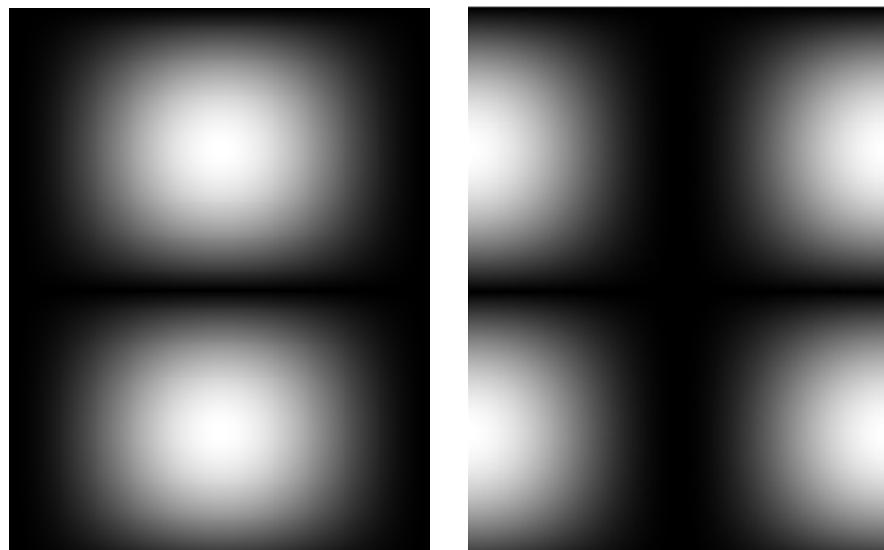


Чтобы увидеть график соответствующего фильтра в частотной области, надо набрать

```
>> freqz2(h)
```



a



б

в

Рисунок 9 – а) Абсолютное значение частотного фильтра, соответствующего вертикальной маске Собела; б) и в) эти же фильтры в виде изображений

На рисунке 9а приведен этот график с опущенными координатными осями. Сам фильтр строится командами

```
>> PQ=paddedsize(size(f))
>> H=freqz2(h, PQ(1), PQ(2))
>> H1=ifftshift(H)
```

где команда *ifftshift* выполняется для перестановки данных так, чтобы начало координат фильтра находилось в верхнем левом углу частотного прямоугольника. На рисунке 9б и 9в показаны абсолютные значения *H* и *H1* в виде полутонаовых изображений, построенных командами

```
>> figure, imshow(abs(H), [])
>> figure, imshow(abs(H1), [])
```

Теперь строим отфильтрованное изображение. В пространственной области используется команда

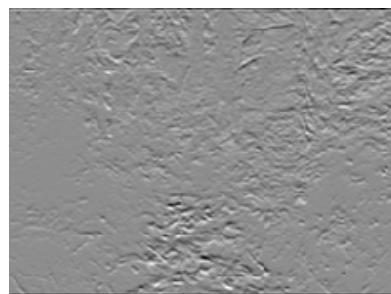
```
>> gs=imfilter(double(f), h)
```

 которая продолжает изображение за его границы по умолчанию нулями.
Изображение, отфильтрованное в частотной области, задается выражением

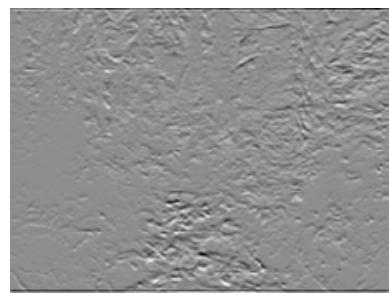
```
>> gf=dftfilt(f, H1)
```

На рисунке 10а приведен результат команд

```
>> figure, imshow(gs, []);
>> figure, imshow(gf, []);
```



а



б

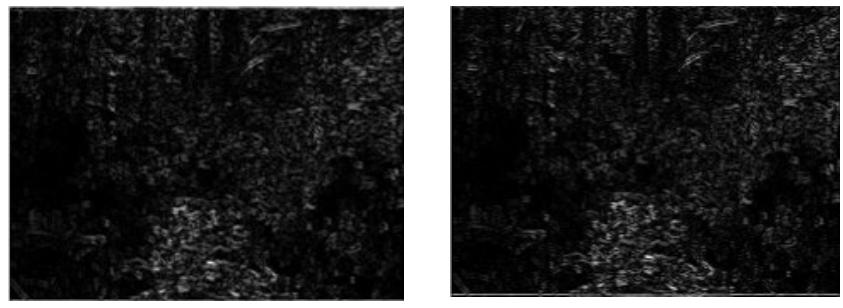


Рисунок 10 – Результаты фильтрации

На рисунке 10а показан результат фильтрации рисунка 8а в пространственной области вертикальной маской Собела. На рисунке 8б – результат фильтрации в частотной области с помощью фильтра, показанного на рисунке 9б. На рисунках 10в и 10г приведены абсолютные значения

Серый фон на этом рисунке объясняется тем, что оба изображения gs и gf имеют отрицательные значения, что приводит к изменению масштаба при выполнении команды *imshow*. Имеет смысл построить здесь абсолютные величины отфильтрованных выше изображений. Это сделано на рисунке 10в и 10г посредством команд:

```
>> figure, imshow(abs(gs), []);
>> figure, imshow(abs(gf), []);
```

Края предметов можно выделить более отчетливо, если создать соответствующие пороговые двоичные изображения:

```
>> figure, imshow(abs(gs)>0.3*abs(max(gs(:)))) )
>> figure, imshow(abs(gf)>0.3*abs(max(gf(:)))) )
```

где множитель 0.3 был выбран (достаточно произвольно) для того, чтобы выделить участки изображений, на которых уровни пикселей больше 30% от максимального значения изображений gs и gf . На рисунках 11а и 11б даны эти изображения.

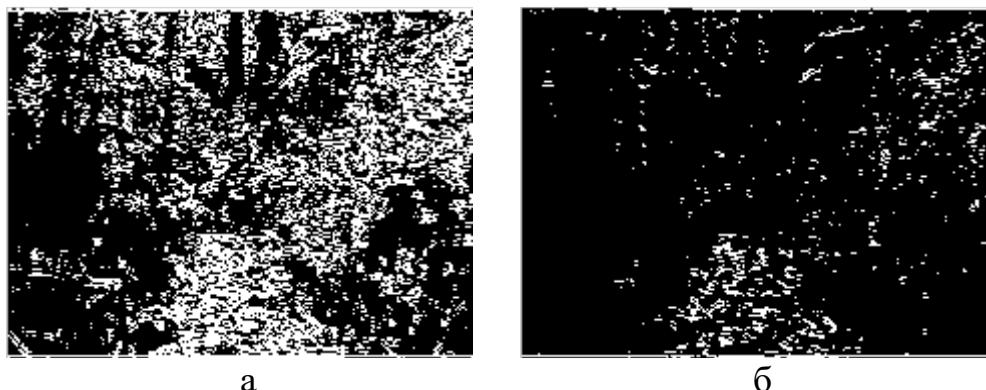
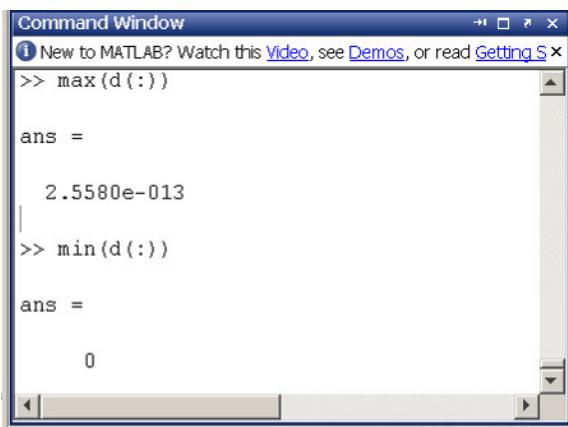


Рисунок 11 – Пороговая обработка рисунка 10в и 10г, соответственно, для более четкого выделения вертикальных краев предметов

Изображения, построенные методами фильтрации в пространственной и частотной областях, являются практически идентичными. Этот факт можно проверить, вычислив их разность

```
>> d=(abs(gs-gf))
```

и найдя максимальное и минимальное значения этой разности:



Продемонстрированные подходы можно использовать при реализации любых пространственных фильтров FIR произвольных размеров с помощью фильтров в частотной области.

Прямое построение фильтров в частотной области **Построение сеточных массивов для использования в фильтрах в частотной области**

В следующих М-функциях важную роль играет процедура вычисления расстояния между любыми точками частотного прямоугольника. Поскольку в MATLAB при выполнении FFT предполагается, что начало отсчета находится в верхнем левом углу частотного прямоугольника, вычисления расстояния ведутся от этой точки. Для лучшей визуализации центр данных можно смещать функцией *fftshift*.

Следующая функция *dftuv* создает сеточный массив, который используется при вычислении расстояний и при других подобных действиях. Сеточные массивы, которые строят функция *meshgrid*, уже приспособлены для применения в *fft2* и *ifft2* без дополнительного переупорядочения.

```
function [U , V]= dftuv( M, N )  
u=0:(M-1);  
v=0:(N-1);  
idx =find(u>M/2);  
u(idx) = u(idx)-M;
```

```

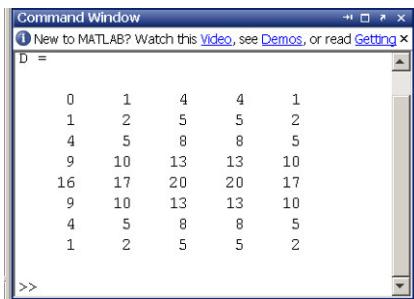
idy=find(v>N/2);
v(idy) = v(idy)-N;
[V, U]=meshgrid(v, u);
end

```

Пример 3. Употребление функции *dftuv*.

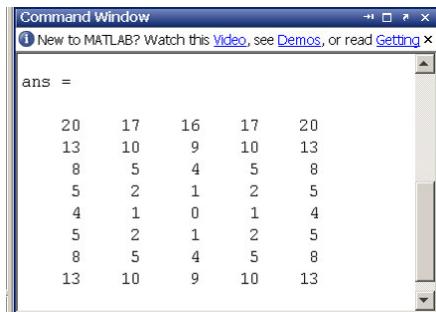
В качестве иллюстрации рассмотрим следующие команды для вычисления квадрата расстояния от начала координат до всех точек частотного прямоугольника размеров 8x5.

```
[V, U]=dftuv(8, 5);
D=U.^2+V.^2
```



Обратите внимание на то, что расстояние до левой верхней точки равно 0, а наибольшее расстояние — до центра прямоугольника, что соответствует формату, который объяснялся на рисунке 2а. Чтобы найти расстояния от центра прямоугольника до всех его точек, достаточно применить к массиву D функцию *fftshift*:

```
fftshift(D)
```



Низкочастотные фильтры

Идеальный низкочастотный фильтр (ILPF, Ideal Lowpass Filter) имеет передаточную функцию

$$H(u, v) = \begin{cases} 1, & D(u, v) \leq D_0 \\ 0, & D(u, v) > D_0 \end{cases}$$

где D_0 — это заданное неотрицательное число, а $D(u, v)$ — расстояние от центра фильтра до точки (u, v) . Геометрическое место точек (u, v) , для которых $D(u, v) = D_0$, является окружностью. Помня о том, что фильтр H умножается на преобразование Фурье изображения, можно заключить, что идеальный фильтр «срезает» (умножает на ноль) все компоненты F , лежащие вне этой окружности, и оставляет неизменными (умножает на 1) все

компоненты, находящиеся внутри или на границе окружности. Несмотря на то, что этот фильтр невозможно реализовать на практике в аналоговой форме с помощью электронных компонент, его, безусловно, можно смоделировать на компьютере с помощью заданной передаточной функции. Свойства идеального фильтра часто бывают полезными при объяснении таких явлений, как *ошибки перекрытия*.

Низкочастотный фильтр Баттерворта (BLPF, Butterworth LowPass Filter) порядка n с обрезанием частот на расстоянии D_0 от начала координат имеет передаточную функцию:

$$H(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_0} \right]^{2n}}$$

В отличие от идеального низкочастотного фильтра, функция фильтра BLPF не имеет разрыва в пороговой точке D_0 . Для фильтров с гладкой передаточной функцией принято задавать частоту срезания, которая определяется положением точек, для которых функция $H(u,v)$ меньше определенной доли ее максимального значения. В предыдущем уравнении значение $H(u,v)=0.5$ (т.е., 50% от максимального значения, которое равно 1), когда $D(u,v)=D_0$.

Передаточная функция *гауссова низкочастотного фильтра* (GLPF, Gaussian LowPass Filter) задается формулой

$$H(u,v) = e^{-D^2(u,v)/2\sigma^2},$$

где σ — это стандартное отклонение. Если положить $\sigma=D_0$, то получится следующее выражение в терминах срезающего параметра D_0 :

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}.$$

При $D(u,v)=D_0$ значение фильтра в этих точках меньше, чем 0.607 от максимального значения, которое равно 1.

Пример 4. Низкочастотная фильтрация.

Для пояснения введенных понятий мы применим гауссов низкочастотный фильтр к изображению f размера 500×500 пикселей, приведенному на рисунке 12а. Мы взяли D_0 , равное 5% от ширины расширения изображения. Совершая шаги фильтрации имеем:

```
PQ=paddedsize(size(f));
[U, V]=dftuv(PQ(1), PQ(2));
D0=0.05*PQ(2);
F=fft2(f, PQ(1), PQ(2));
H=exp(-(U.^2+V.^2)/(2*(D0.^2)));
g=dftfilt(f, H);
```

Построенный фильтр можно «увидеть» (рисунок 12б), выполнив команду:

```
figure, imshow (fftshift(H), [])
```

Аналогично, спектр Фурье можно отобразить на дисплее (рисунок 12в), набрав на клавиатуре

```
figure, imshow (log(1+abs(fftshift(F))), [ ])
```

Наконец, на рисунке 12г приведено изображение, полученное командой

```
figure, imshow (g, [ ])
```

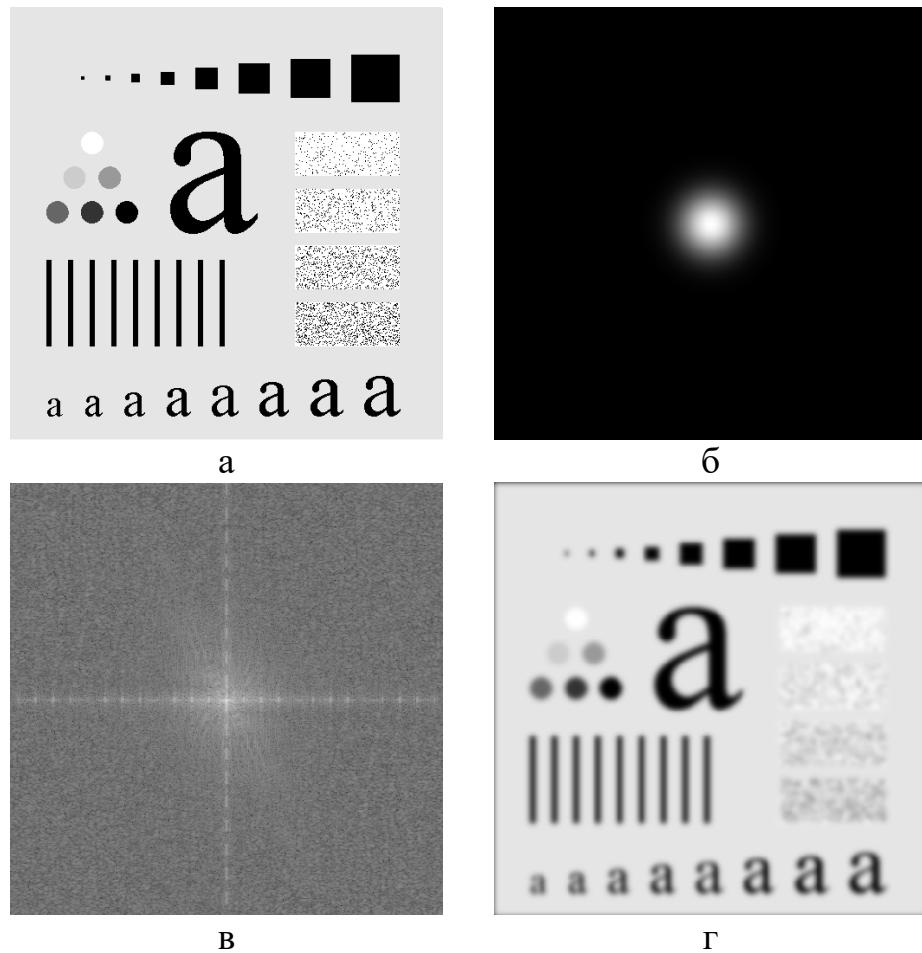


Рисунок 12 – Низкочастотная фильтрация

- а) Исходное изображение; б) Гауссов низкочастотный фильтр в виде изображения; в) Спектр а; г) Обработанное изображение

Как и ожидалось, отфильтрованное изображение представляет собой размытый образ исходного.

Следующая программа генерирует передаточную функцию всех перечисленных выше низкочастотных фильтров.

```
function H = lpfilter(type, M, N, D0, n)
disp('lpfilter')
[U, V] = dftuv(M, N);
D = sqrt(U.^2 + V.^2);
```

```

switch type
case 'ideal'
    H = double(D <= D0);
case 'btw'
    if nargin == 4
        n = 1;
    end
    H = 1./(1 + (D./D0).^(2*n));
case 'gaussian'
    disp('gaussian')
    H = exp(-(D.^2)./(2*(D0^2)));
otherwise
    error('Unknown filter type.')
end

```

Функция *lpfilter* будет использоваться как основа для построения высокочастотных фильтров.

Построение графиков каркасных контуров и поверхностей

Методы построения трехмерных каркасных (сеточных) контуров и поверхностей могут быть полезными при визуализации передаточных функций двумерных изображений. Самый простой способ нарисовать сеточный график данной двумерной функции H состоит в применении функции *mesh*, которая имеет базовый синтаксис:

$$\text{mesh}(H).$$

Эта функция рисует каркасный контур для точек $x = 1:M$ и $y = 1:N$, где $[M, N] = \text{size}(H)$. Вид каркасного контура становится очень плотным и неудобным для глаза, когда числа M и N — велики. В этом случае удобно строить только каждую k -ую точку по каждой оси с помощью команды

$$\text{mesh}(H(1:k:end, 1:k:end)).$$

Как правило, если использовать при построении каркасного контура от 40 до 60 точек по каждой оси, то результат будет вполне приемлемым компромиссом между разрешением и внешним видом изображения.

По умолчанию MATLAB строит сеточные графики в цвете. Команда

$$\text{colormap}([0 0 0])$$

устанавливает черную сетку. MATLAB также накладывает оси и линии координатной сетки на сеточный график. Эти атрибуты отключаются командой

$$\begin{aligned} &\text{grid off} \\ &\text{axis off}. \end{aligned}$$

Снова включить их можно соответствующей командой, в которой *off* заменено на *on*. Наконец, точка обзора (местоположение наблюдателя) контролируется функцией *view*, которая имеет синтаксис

$$\text{view}(az, el).$$

На рисунке 13 показано, что *az* и *el* представляют собой, соответственно, азимут и угол возвышения (измеренные в градусах).

Стрелками обозначены положительные направления. Значения по умолчанию: $az = -37.5$ и $el = 30$, которые помещают наблюдателя в квадрант, задаваемый полуосами $-x$ и $-y$, откуда он смотрит на квадрант, определяемый положительными полуосами x и y на рисунке 13.

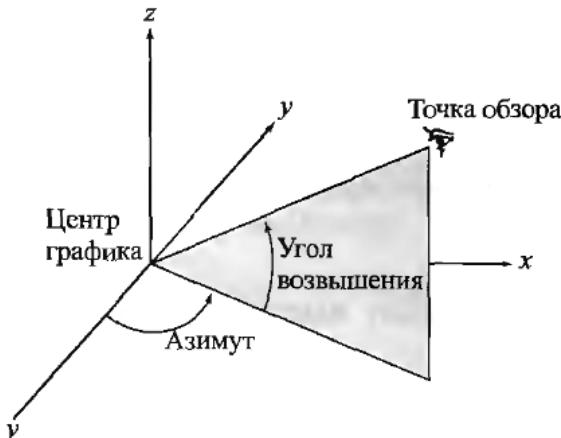


Рисунок 13 – Геометрия функции *view*

Чтобы узнать текущее значение углов обзора, следует набрать

```
>> [az,el]=view;
```

А чтобы присвоить этим параметрам значения по умолчанию, надо набрать

```
>> view(3)
```

Точку обзора можно менять интерактивно, если нажать мышью на кнопку *Rotate 3D* в строке инструментов окна графиков, а затем действовать мышью в окне графика.

При построении общих графиков удобно использовать описанный выше метод, т.к. в нем используется всего два параметра и он является более наглядным.

Пример 5. Построение сеточных графиков.

Рассмотрим гауссов низкочастотный фильтр из примера 4.

```
H =fftshift(lpfilt('gaussian',500,500,50));
```

На рисунке 14 показан сеточный (каркасный) график, построенный командами

```
figure('Name','H'), mesh(H(1:10:500,1:10:500))
```

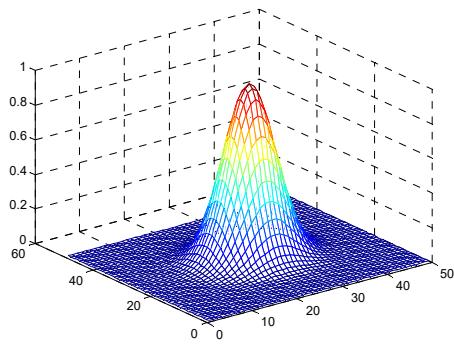


Рисунок 14 – График, построенный функцией mesh

Повышение резкости при частотной фильтрации

В противоположность низкочастотной фильтрации, которая приводит к размытию изображений, *высокочастотная фильтрация* повышает резкость изображения, ослабляя низкие частоты и оставляя высокие частоты преобразования Фурье относительно неизменными.

Основы высокочастотной фильтрации

Имея передаточную функцию $H_{lp}(u, v)$ низкочастотного фильтра, можно получить передаточную функцию соответствующего высокочастотного фильтра с помощью формулы

$$H_{hp}(u, v) = 1 - H_{lp}(u, v).$$

Значит, функцию *lpfilter* можно использовать для построения генератора высокочастотных фильтров:

```
function H = hpfilter(type, M, N, D0, n)
if nargin == 4
    n = 1;
end
Hlp = lpfilter(type, M, N, D0, n);
H = 1 - Hlp;
```

Пример 6. Высокочастотная фильтрация.

На рисунке 15а приведен тот же тестовый образец f , что и на рисунке 12а. Рисунок 15б получен с помощью следующих команд, результат которых демонстрирует применение высокочастотного гауссова фильтра к изображению f :

```
PQ=paddedsize(size(f));
D0=0.05*PQ(1);
H=hpfilter('gaussian',PQ(1), PQ(2),D0);
g=dftfilt(f, H);
figure, imshow ((f), [])
figure, imshow (g, [])
```

На рисунке 15б заметно повышение четкости границ и усиление других контуров. Вместе с тем, поскольку средняя яркость изображения задается

величиной $F(0,0)$, а высокочастотная фильтрация гасит низкие частоты в окрестности начала отсчета, то отфильтрованное изображение потеряло большую часть фоновой тональности, имевшуюся на исходном изображении.

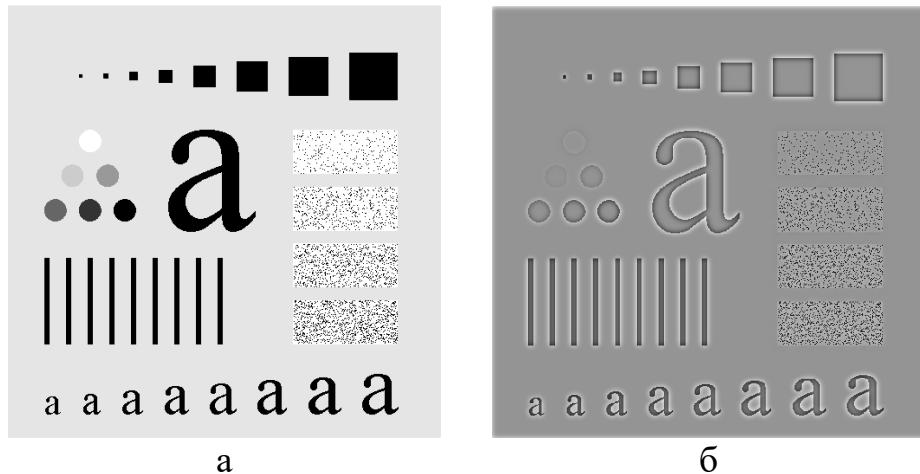


Рисунок 15 – а) Исходное изображение; б) Результат высокочастотной фильтрации Гаусса

Фильтрация с усилением высоких частот

Высокочастотные фильтры отбрасывают коэффициент dc , в итоге среднее становится равным нулю. Чтобы компенсировать этот недостаток, необходимо добавить некоторое смещение к высокочастотному фильтру. Если смещение комбинируется вместе с умножением фильтра на коэффициент усиления, больший 1, то такая процедура называется фильтрацией с усилением высоких частот. Поскольку этот коэффициент больше единицы, то в результате происходит увеличение высоких частот. Конечно, амплитуда низких частот при этом тоже увеличивается, однако этот эффект меньше влияет на низкие частоты, поскольку смещение мало по сравнению с коэффициентом усиления. Высокочастотное усиление имеет передаточную функцию

$$H_{hfe}(u, v) = a + bH_{hp}(u, v),$$

где a — это смещение, b — коэффициент усиления, а $H_{hp}(u, v)$ — передаточная функция фильтра высоких частот.

Выравнивание гистограммы осуществляется с помощью функции *histeq*, которая имеет следующий синтаксис:

```

Id=histeq(Is, hgram)
[Id, T]=histeq(Is, hgram)
Id=histed(Is, n)
[Id, T]=histeq(Is, n)
newmap=histeq(X, map, hgram)
[newmap, T]=histeq(X, map, hgram)
newmap=histeq(X, map)
[newmap, T]=histeq(X, map)

```

Функция **histeq** улучшает контраст изображения с помощью преобразования значений пикселов исходного изображения таким образом, чтобы гистограмма яркостей пикселов результирующего изображения приблизительно соответствовала некоторой предопределенной гистограмме [1,2]. Данная функция предназначена для преобразования полутонаовых изображений или палитровых изображений.

Функция **histeq(Is, hgram)** преобразует исходное полутонаовое изображение Is таким образом, чтобы гистограмма яркостей пикселов результирующего полутонаового изображения Id приблизительно соответствовала гистограмме, задаваемой вектором hgram. Количество элементов в hgram задает число столбцов гистограммы, а значение каждого элемента - относительную высоту каждого столбца. Значение элементов вектора hgram должно быть в диапазоне [0, 1]. Функция hgram автоматически масштабирует значения элементов hgram так, чтобы сумма значений элементов в гистограмме была равна количеству пикселов изображения, т.е. $\text{sum}(\text{hgram})=\text{prod}(\text{size}(Is))$. Гистограмма результирующего изображения Id будет лучше соответствовать заданной гистограмме hgram в том случае, когда количество столбцов hgram ($\text{length}(\text{hgram})$) много меньше количества градаций яркости исходного изображения Is.

Функция **Id=histeq(Is, n)** преобразует исходное полутонаовое изображение Is таким образом, чтобы результирующее полутонаовое изображение Id имело гистограмму яркостей пикселов, близкую к равномерной. Равномерная гистограмма hgram создается из n столбцов как $\text{hgram}=\text{ones}(1, n)*\text{prod}(\text{size}(Is))/n$. Чем меньше n по сравнению с количеством градаций яркости в изображении Is, тем более равномерной получается гистограмма яркостей пикселов результирующего изображения Id. По умолчанию значение n равно 64, и данный параметр можно не указывать при вызове функции. Формат результирующего изображения Id совпадает с форматом исходного Is.

Функция **[Id, T]=histeq(Is, n)** дополнительно возвращает вектор T, задающий характеристику передачи уровней яркости исходного изображения Is в уровнях яркости результирующего изображения Id.

Функция **newmap=histeq(X, map, hgram)** преобразует палитру map, связанную с палитровым изображением X, таким образом, чтобы гистограмма яркостей пикселов изображения X с палитрой newmap приблизительно соответствовала гистограмме hgram. Для получения значений яркости из палитры map используется функция **rgb2ntsc**. Количество элементов вектора hgram должно быть равно размеру палитры map.

Функция **newmap=histeq(X, map, hgram)** преобразует палитру map, связанную с палитровым изображением X, таким образом, чтобы гистограмма яркостей пикселов изображения X с палитрой newmap была близка к равномерной.

Функция **[newmap, T]=histeq(X, ...)** дополнительно возвращает вектор T, задающий характеристику передачи уровней яркости из исходной палитры map в уровнях яркости результирующей палитры newmap.

Существуют также другие, более сложные подходы к решению задачи улучшения изображений путем гистограммных преобразований. Они базируются не только на анализе гистограммы распределения яркостей элементов изображения в целом, но и ее отдельных компонент. Также существует ряд методов, основанных на анализе локальных гистограмм отдельных участков изображения. Существование большого количества методов гистограммных преобразований объясняется их относительной простотой и высокой эффективностью.

Характеристика передачи уровней яркости T выбирается путем минимизации функции $F(k) = |c_1(T(k)) - c_0(k)|$, где c_0 - сумма всех пикселов полутонаового изображения I с яркостью, меньше либо равной k ; c_1 - сумма значений заданной гистограммы h_{gram} для всех дискретных уровней яркости, меньших либо равных k . При минимизации на функцию T накладывают следующие ограничения: T должна быть монотонной и $c_1(T(k))$ не должно превышать $c_0(a)$ более, чем на половину количества точек с яркостью a .

Рассмотрим пример. Пример демонстрирует повышение контраста изображения с помощью выравнивания гистограммы.

Полутоновое изображение I читается из файла и выводится на экран (рисунок 16а). Строится гистограмма яркостей пикселов исходного изображения (рисунок 16б). Функция **histeq** выравнивает гистограмму изображения I . Результат преобразования и гистограмма яркостей его пикселов выводятся на экран (соответственно рисунок 16в и рисунок 16г).

```
%Чтение исходного изображения и вывод его на экран.
I=imread('image1.tif');
imshow(I);
%Построение гистограммы исходного изображения.
figure, imhist(I);
%Выравнивание гистограммы.
I=histeq(I, 80);
%Вывод преобразованного изображения на экран.
Figure, imshow(I);
%Вывод гистограммы преобразованного изображения.
figure, imhist(I);
```

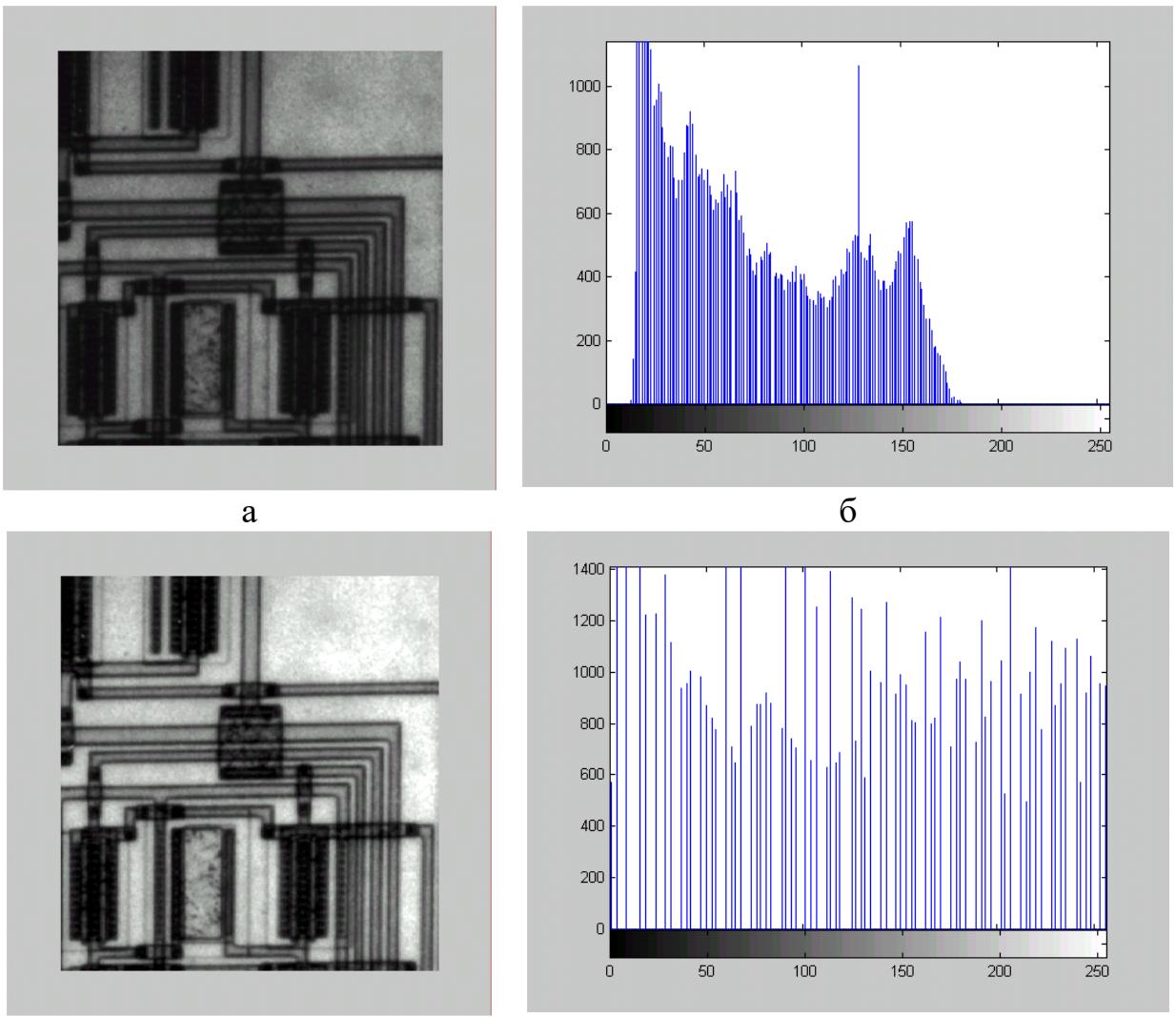


Рисунок 16 – Фильтрация с усилением высоких частот

Пример 7. Комбинированная фильтрация с усилением высоких частот и гистограммная эквализация.

На рисунке 17а представлен рентгеновский снимок грудной клетки *f*. Поскольку рентгеновские лучи хуже поддаются фокусировке, как это делается со световыми лучами при помощи линз, рентгеновские снимки, как правило, выглядят расплывчато. Цель этого примера заключается в повышении резкости данного изображения. В этом конкретном примере яркость изображения сдвинута в темную область градационной шкалы, поэтому мы воспользуемся этим обстоятельством для демонстрации пространственной обработки, которая удачно дополняет фильтрацию в частотной области.

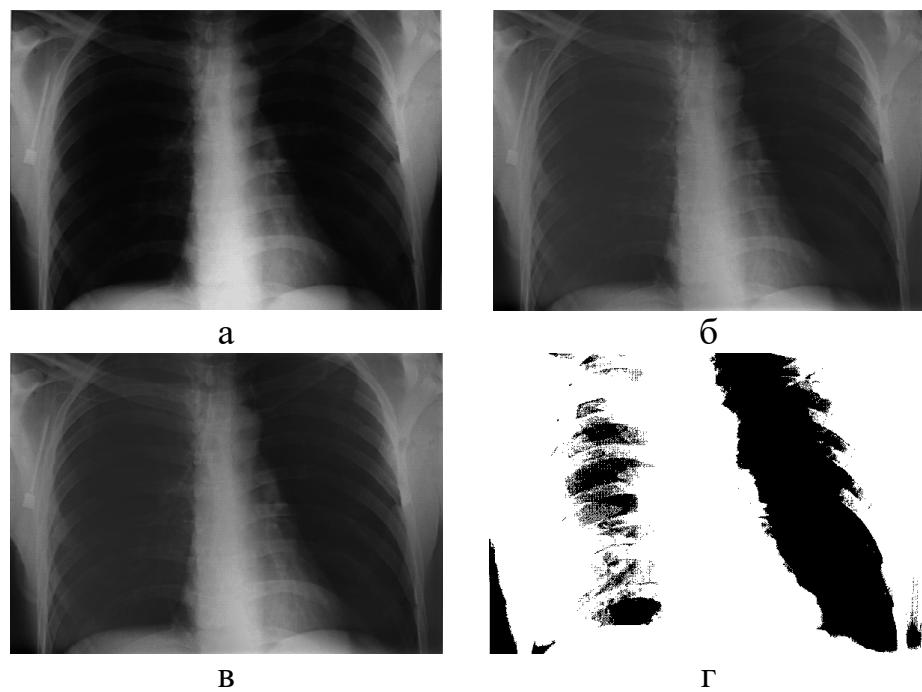


Рисунок 17 – Фильтрация с усилением высоких частот

На рисунке 17б изображен результат фильтрации рисунка 17а с использованием высокочастотного фильтра Баттервортса порядка 2 с частотой срезания D_0 , равной 5% от вертикального размера расширенного изображения. Высокочастотная фильтрация не очень сильно зависит от параметра D_0 , поскольку радиус фильтра не так мал, чтобы пропускать частоты, расположенные возле начала отсчета. Как и ожидалось, результат фильтрации не очень выразителен, однако на нем слабо различимы основные контуры изображения. Рисунок 17в демонстрирует преимущество фильтрации с усилением высоких частот (в данном случае, $a = 0.5$ и $b = 2.0$), при котором сохранен общий яркостной тон исходного изображения. Следующая последовательность команд была использована при построении изображений на рисунке 17, где f обозначает входное изображение.

```
PQ=paddedsize(size(f));
D0=0.05*PQ(1);
HBW=hpfilter('btw',PQ(1), PQ(2),D0,2);
H=0.5+2*HBW;
gbw=dftfilt(f, HBW);
ghf=dftfilt(f, H);
ghe=histeq(ghf,256)
figure, imshow ((f), [])
figure, imshow (gbw, [])
figure, imshow (ghf, [])
figure, imshow (ghe, [])
```

Изображение, характеризующееся узким диапазоном яркости, является идеальным кандидатом для гистограммной эквалайзации. Из рисунка 16г видно, что этот метод действительно продолжает улучшать изображение.

Обратите внимание на четкое выявление структур костей и других деталей, которых даже не видно на трех предыдущих изображениях. В последнем изображении можно обнаружить некоторое усиление шумов, но это явление весьма характерно для рентгеновских снимков при растяжении диапазона их яркости. Результат, полученный при совместном использовании фильтрации с усилением высоких частот и метода гистограммной эквализации, превосходит любой результат, который можно было бы получить, применяя каждый из этих методов по отдельности.

Общая постановка задачи

В результате выполнения заданий лабораторной работы студенты **должны уметь** создавать программно-алгоритмическую поддержку для компьютерной реализации требуемых технике методов преобразования изображений в частотной области в MatLab.

Список индивидуальных данных

№ варианта	Пороговая точка D_0
11.	2%
12.	3%
13.	4%
14.	5%
15.	6%
16.	7%
17.	8%
18.	9%
19.	10%
20.	1%

Написать файл-функцию и GUI интерфейс для решения следующих задач. Скопировать в папку для выполнения лабораторной работы все необходимые m-функции (lpfilter, hpfilter, dftuv, paddedsizе) и файлы изображений.

Задание 1. Вычисление и визуализация двумерного DFT

13. Выполните чтение изображения из файла, самостоятельно выбранного студентом. Выберите первую цветовую компоненту изображения в формате RGB и определите размер изображения. Используйте функцию **imshow** для вывода изображений на экран.
14. Выполните прямое преобразование Фурье, используя функцию **fft2**.
15. Выведите на экран спектр исходного изображения, используя функции **mesh**.
16. Выполните центрирование спектра, а затем выведите его на экран, используя функцию **imshow**.

17. Выполните обнуление части спектра, а затем выведите его на экран, используя функцию **imshow**.
18. Выполните повторно центрирование спектра, используя функцию **fftshift**.
19. Выполните обратное преобразование Фурье (**ifft2**) результата обнуления и повторного спектра.
20. Выведите на экран результирующее изображение.
21. Сохраните полученные изображения и графики.

Задание 2. Низкочастотная фильтрация с использованием и без использования процедур расширения

1. Выполните чтение изображения из файла, самостоятельно выбранного студентом. Выберите первую цветовую компоненту изображения в формате RGB и определите размер изображения. Используйте функцию **imshow** для вывода изображений на экран.
2. Постройте гауссов низкочастотный фильтр, применив функцию **lpfilter**. Выполните низкочастотную фильтрацию на основе БПФ и обратного БПФ (см. задание 1).
3. Расширить изображение, применив функцию **paddedsize**.
4. Постройте гауссов низкочастотный фильтр, применив функцию **lpfilter**. Выполните низкочастотную фильтрацию на основе БПФ и обратного БПФ (см. задание 1).
5. Сохраните полученные изображения и графики.

Задание 3. Высокочастотная фильтрация.

1. Выполните чтение изображения из файла, самостоятельно выбранного студентом. Используйте функцию **imshow** для вывода изображений на экран.
2. Построить гауссов высокочастотный фильтр, применив функцию **hpfilter** для изображения f согласно варианту.
3. Используйте функцию **dftfilt**, которая выполняет все необходимые процедуры фильтрации и возвращает отфильтрованное изображение.
4. Отобразите результаты фильтрации.
5. Отобразите спектр исходного изображения. Используйте функцию **fftshift** для смещения начала координат спектра в центр частотной области.
6. Сохраните полученные изображения и графики.

Задание 4. Комбинированная фильтрация с усилением высоких частот и гистограммная эквалайзация.

1. Выполните чтение изображения из файла, самостоятельно выбранного студентом. Используйте функцию **imshow** для вывода изображений на экран.
2. Построить высокочастотный фильтр Баттервортса , применив функцию **hpfilter** для изображения f согласно варианту.

3. Используйте функцию **dftfilt**, которая выполняет все необходимые процедуры фильтрации и возвращает отфильтрованное и обрезанное изображение.
4. Усилить высокие частоты. Повторить пункт 3.
5. Выровнять гистограмму отфильтрованного изображения, используя функцию **histeq**.
6. Отобразите результаты фильтрации.
7. Сохраните полученные изображения и графики.

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.

Лабораторная работа № 6. Восстановление изображений

Цель работы:

Целью лабораторной работы является получение навыков самостоятельной алгоритмической и программной реализации на компьютерной технике методов восстановления в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-2	3-1	Способен анализировать социально-экономические проблемы
	3-2	Знает методы расчетов математических моделей
ПК-5	3-1	Знает методы оценки проектов по информатизации
	3-2	Знает методы оценки проектов по автоматизации решения прикладных задач
ПК-14	3-1	Знает методы анализа прикладной

		области на логическом уровне
	3-2	Знает методы анализа прикладной области на алгоритмическом уровне
ПК-25	3-1	Знает различные математические методы решения прикладных задач
	3-2	Знает методы формализации решения прикладных задач

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-5	У-1	Способен к восприятию и воспроизведению информации
	У-2	Знает алгоритмы целеполагания и выбора путей их достижения
ПК-14	У-1	Знает методы анализа прикладной области на логическом уровне
	У-2	Знает методы анализа прикладной области на алгоритмическом уровне
ПК-25	У-1	Знает различные математические методы решения прикладных задач
	У-2	Знает методы формализации решения прикладных задач

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-1	B-1	Знает основы современных технологий сбора, обработки и представления информации
	B-2	Способен ориентироваться в информационном потоке в глобальных компьютерных сетях
ПК-2	B-1	Способен анализировать социально-экономические проблемы
	B-2	Знает методы расчетов математических моделей
ПК-25	B-1	Знает различные математические методы решения прикладных задач
	B-2	Знает методы формализации решения прикладных задач

Теоретическая часть

Задача восстановления изображения заключается в улучшении данного изображения в некотором заранее оговоренном смысле. Несмотря на значительное пересечение областей применения этих двух методов обработки, улучшение изображения является достаточно субъективным процессом, в то время, как процедуры восстановления изображения носят вполне объективный характер. Целью восстановления является реконструкция или восстановление изображения, которое ранее было искажено или испорчено в результате явление, про которые имеется достаточно определенная априорная информация. Поэтому методы восстановления изображения основаны на моделировании процессов искажения и применении обратных процессов для восстановления исходного изображения.

При таком подходе важно уметь правильно формулировать критерии качества, которые позволяют оценить результат восстановления. А при решении задач улучшения изображений используется иная методология, основанная на эвристических процедурах, визуальный результат которых зависит от особенностей человеческого зрения. Например, усиление контраста можно рассматривать, как процедуру улучшения изображения, поскольку после ее применения изображение становится более приятным для глаза, а обработку смазанных изображений с помощью соответствующих обратных процедур следует отнести к арсеналу средств по восстановлению изображений.

Моделирование процесса искажения/восстановления изображения

На рисунке 1 процесс ухудшения изображения смоделирован в виде функции искажения, которая вместе с аддитивным шумом действует на исходное изображение $g(x, y)$ и порождает искаженное изображение $g(x, y)$:

$$g(x, y) = H[f(x, y)] + \eta(x, y).$$

Имея функцию $g(x, y)$, обладая некоторой информацией об искажающем операторе H и зная основные характеристики аддитивного шума $\eta(x, y)$, можно построить некоторое приближение $\hat{f}(x, y)$ исходного изображения. Целью восстановления изображения является построение приближения $\hat{f}(x, y)$, которое было бы максимально близко к исходному изображению. При этом, чем больше мы знаем об операторе H и шуме $\eta(x, y)$, тем точнее мы сможем приблизить изображение $f(x, y)$ функцией $\hat{f}(x, y)$.

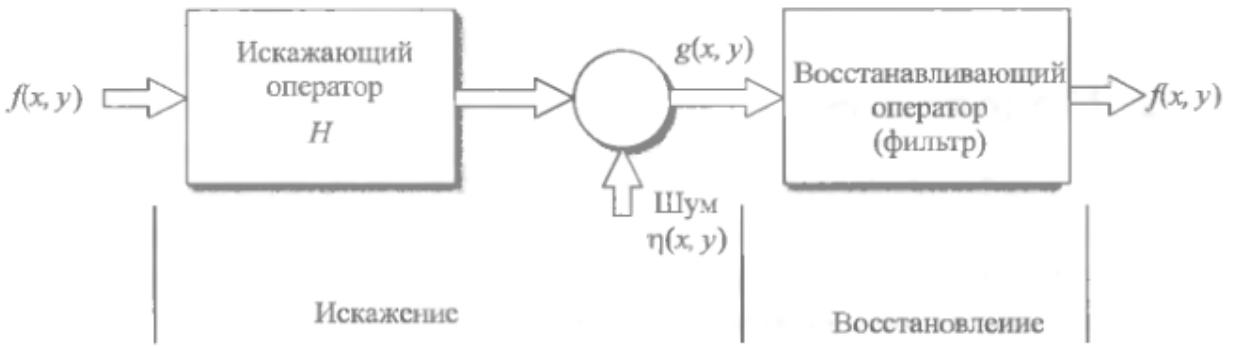


Рисунок 1 – Моделирование процесса искажения/восстановления изображения

Если известно, что H – линейный трансляционно-инвариантный оператор, то можно показать математически, что искаженное изображение представимо в пространственной области в следующем виде:

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y)$$

где, $h(x, y)$ – это пространственное представление искажающего оператора, а символ « $*$ » обозначает свертку (символ « $*$ » в строке уравнения обозначает свертку, а помещение его в верхний индекс формулы будет обозначать комплексное сопряжение. Кроме того, при написании программ звездочкой обозначается операция умножения. Поэтому следует быть внимательным, чтобы не перепутать различные использование одного и того же символа.). Свертка функций в пространственной области эквивалентна умножению в частотной области преобразований Фурье этих функций, поэтому приведенное выше уравнение модели искажения можно записать в эквивалентном представлении в частной области:

$$G(u, v) = H(u, v)F(u, v) + N(u, v),$$

где заглавными буквами обозначены соответствующие преобразования Фурье функций из уравнения свертки. Функцию $H(u, v)$ часто называют *оптической передаточной функцией* (OTF, Optical Transfer Function). Этот термин заимствован из анализа Фурье оптических систем. В пространственной области функция $h(x, y)$ называется *функцией разброса точек* (PSF, Point Spread Function). Этот термин возникает при действии функции $h(x, y)$ на точки света для получения характеристик искажения разных типов входных данных. Функции $h(x, y)$ и $H(u, v)$ переходят друг в друга под действием прямого и обратного преобразования Фурье, поэтому в пакете имеется две M-функции *otf2psf* и *psf2otf* для этих действий.

В силу того, что оператор линейного трансляционно-инвариантного искажения H можно смоделировать в виде свертки (конволюции), иногда этот процесс искажения называют «конволюцией изображения с PSF или OTF». По аналогии, обратный процесс восстановления называют *деконволюцией*.

Модели шума

Возможность смоделировать шум является ключевым моментом восстановления изображений. Рассмотрим два типа моделей шумов: шум в пространственной области (характеризующийся функцией плотности вероятности) и шум в частной области, который описывается разными характеристиками Фурье.

Добавление шума функцией `imnoise`

В пакете имеется функция `imnoise`, которая моделирует искажение изображения некоторым шумом. Функция `imnoise` создает новое изображение `g` путем добавления шума к исходному полутоновому изображению `f`. В основном данная функция предназначена для создания тестовых изображений, используемых при выборе методов фильтрации шума, однако может применяться и для придания фотoreалистичности искусственно синтезированным изображениям. Синтаксис этой функции имеет вид

```
g = imnoise (f, type, parameters),
```

где `f` – это исходное изображение, а смысл аргументов `type` и `parameters` будет объяснен позже. Функция `imnoise` сначала преобразует изображение в класс `double` в диапазоне [0, 1]. Это следует иметь в виду перед заданием параметров шума. Например, чтобы прибавить шум со средним 64 и дисперсией 400 к изображению класса `unit8`, следует сжать среднее до величины $64/256$, а дисперсию приравнять $400/(256)^2$, после чего эти параметры можно подставлять в функцию `imnoise`. Рассмотрим различные синтаксические формы этой функции.

`g = imnoise (f, 'gaussian', m, var)` добавляет к изображению `f` гауссов шум со средним `m` и дисперсией `var`. По умолчанию, `m = 0` и `var = 0.01`.

`g = imnoise (f, 'localvar', V)` добавляет к изображению `f` локальный гауссов шум с нулевым средним, дисперсия которого в каждой точке изображения `f` задается матрицей `V`, размера как у `f`.

`g = imnoise (f, 'localvar', image_intensity, var)` добавляет к изображению `f` гауссов шум с нулевым средним, в котором локальная дисперсия шума `var` является функцией значений яркости изображения `f`. Аргументы `image_intensity` и `var` являются векторами одинаковой размерности, а функция `plot(image_intensity, var)` строит график зависимости дисперсии `var` от яркости `image_intensity`. Вектор `image_intensity` должен содержать нормированные значения яркости в диапазоне [0,1].

`g = imnoise (f, 'salt & paper, d)` портит изображение шумом «соль и перец», где `d` – это плотность шума (т.е процент изображения, подтвержденного этому шуму). При этом приблизительно `d*numel(f)` пикселей будет испорчено. По умолчанию, `d = 0,05`

g = imnoise (f, ‘speckle’, var) добавляет к f мультипликативный шум по формуле $g = f + n*f$, где **n** – это равномерно распределенный шум с нулевым средним и дисперсией **var**. По умолчанию, **var** = 0,04.

g = imnoise (f, ‘poisson’) генерирует пуассоновский шум, зависящий от исходных данных, вместо добавления искусственного шума. Чтобы согласовываться со статистиками Пуассона, пиксели изображений **unit8** и **unit16** должны соответствовать числу фотонов (или другим квантам информации). Используется изображение с двойной точностью, когда число фотонов на один пиксель больше, чем 65535 (но меньше 10^{12}). Величины яркости пикселей меняются в пределах от 0 до 1 и соответствуют числу фотонов, деленному на 10^{12} .

Рассмотрим пример использования функции **imnoise**.

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02);
figure, imshow(I)
figure, imshow(J)
```

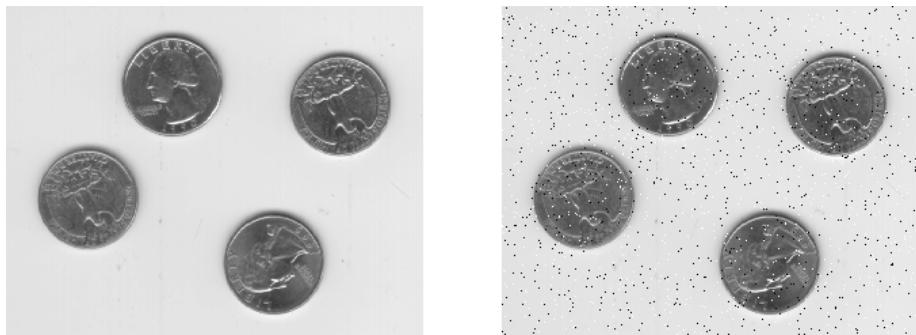


Рисунок 1– Пример использования функции **imnoise**

Генерация случайного пространственного шума с заданным распределением

Часто бывает необходимо сгенерировать шум, тип и параметры которого охватываются функцией **imnoise**. Значения пространственного шума являются случайными величинами, которые принято характеризовать функцией плотности вероятности (PDF, Probability Density Function) или соответствующей функцией (кумулятивного) распределения (CDF, Cumulative Distribution Function). Построение датчиков случайных чисел с интересующими нас распределениями делается с помощью некоторых простых правил теории вероятностей.

Большинство генераторов случайных чисел основано на переформулировании задачи в терминах случайных чисел с равномерной функцией распределения в интервале [0, 1]. В некоторых случаях таким базовым генератором может служить генератор гауссовых случайных чисел с нулевым средним и единичной дисперсией. Хотя оба этих генератора можно получить из функции **imnoise**, имеет смысл сделать это с помощью функции **rand** для равномерных случайных чисел и функции **randn** для нормальных

(гауссовых случайных чисел), которые реализованы в системе MATLAB. Эти функции будут описаны несколько позже в этом параграфе.

Основной описываемого здесь подхода является классический результат теории вероятностей, который утверждает, что если имеется случайная величина ω с равномерным распределением на отрезке $[0,1]$, то случайную величину z с заданной функцией распределения F_z , можно построить по формуле

$$z = F^{-1}(w).$$

Этот простой, но весьма полезный результат можно сформулировать в следующей эквивалентной форме: решить уравнение $F(z) = w$ относительно z .

Пример 1. Использование равномерно распределенных случайных чисел в генераторе случайных чисел с заданной функцией распределения.

Пусть имеется генератор случайных чисел ω с равномерным распределением в интервале $(0,1)$, и мы хотим построить случайную величину z с функцией распределения вероятностей Релея, которая имеет вид

$$F_z(z) = \begin{cases} 1 - e^{-(z-a)^2/b} & \text{при } z \geq a, \\ 0 & \text{при } z < a. \end{cases}$$

Чтобы получить z достаточно решить уравнение

$$1 - e^{-(z-a)^2/b} = w \text{ или } z = a + \sqrt{b \ln(1-w)}.$$

Поскольку квадратный корень – положительная функция, то генерируемые случайные величины будут всегда больше a , что требуется в определении функции Релея. Значит, равномерно распределенные случайные числа могут служить основой для генератора релеевских случайных чисел с заданными параметрами a и b .

В MATLAB этот результат легко обобщить, но массив случайных чисел R размером MxN с помощью выражения

```
>> R = a + sqrt(b*log(1 - rand(M, N));
```

где log – это натуральный логарифм, а функция **rand** генерирует равномерно распределенные случайные числа в интервале $(0,1)$. Если положить $M=N=1$, то получится скалярная случайная величина с релеевским распределением и с параметрами a и b .

Выражение вида $z = a + \sqrt{b \ln(1-w)}$. иногда называют уравнением генератора случайных чисел, поскольку в нем определяется, как вычислять требуемые случайные величины. В этом случае имеется простая формула для решения уравнений. Однако, это не всегда возможно, и проблему можно сформулировать следующим образом: как получить уравнение генератора случайных чисел, выход которого хорошо приближает случайную величину с заданной функцией плотности вероятности.

Таблица 1. Генераторы случайных чисел

Имя	Плотность	Среднее и дисперсия	Распределение	Генератор ¹
Равномерная	$p_z(z) = \begin{cases} \frac{1}{b-a}, & a \leq z \leq b \\ 0, & \text{иначе} \end{cases}$	$m = \frac{a+b}{2}, \quad \sigma^2 = \frac{(b-a)^2}{12}$	$F_z(z) = \begin{cases} 0, & z < a \\ \frac{z-a}{b-a}, & a \leq z \leq b \\ 1, & z > b \end{cases}$	rand (MATLAB)
Гауссова	$p_z(z) = \frac{1}{\sqrt{2\pi b}} e^{-(z-a)^2/2b^2}$ $-\infty < z < +\infty$	$m = a, \quad \sigma^2 = b^2$	$F_z(z) = \int_{-\infty}^z p_z(v) dv$	randn (MATLAB)
«Соль и перец»	$p_z(z) = \begin{cases} Pa, & z = a \\ Pb, & z = b, \quad (b > a) \\ 0, & \text{иначе} \end{cases}$	$m = aP_a + bP_b$ $\sigma^2 = (a-m)^2 P_a + (b-m)^2 P_b$	$F_z(z) = \begin{cases} 0, & z < a \\ Pa, & a \leq z < b \\ Pa + Pb, & z \geq b \end{cases}$	rand (MATLAB)
Логарифмическая нормальная	$p_z(z) = \frac{1}{\sqrt{2\pi bz}} e^{-(z-a)^2/2b^2}, \quad z > 0$	$m = e^{a+(b^2/2)},$ $\sigma^2 = [e^{b^2} - 1]e^{2a+b^2}$	$F_z(z) = \int_{-\infty}^z p_z(v) dv$	$z = ae^{bN(0,1)}$
Релея	$p_z(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/2b^2}, & z \geq a \\ 0, & z < a \end{cases}$	$m = a + \sqrt{2\pi b/4}, \quad \sigma^2 = \frac{b(4-\pi)}{4}$	$F_z(z) = \begin{cases} 1 - e^{-(z-a)^2/b}, & z \geq a \\ 0, & z < a \end{cases}$	$z = a + \sqrt{b \ln[1-U(0,1)]}$
Экспоненциальная	$p_z(z) = \begin{cases} ae^{-az}, & z \geq 0 \\ 0, & z < 0 \end{cases}$	$m = \frac{1}{a}, \quad \sigma^2 = \frac{1}{a^2}$	$F_z(z) = \begin{cases} 1 - e^{-az}, & z \geq 0 \\ 0, & z < 0 \end{cases}$	$z = -\frac{1}{a} \ln[1-U(0,1)]$
Эрланга	$p_z(z) = \frac{a^b z^{b-1}}{(b-1)!} e^{-az}, \quad z \geq 0$	$m = \frac{b}{a}, \quad \sigma^2 = \frac{b}{a^2}$	$F_z(z) = [1 - e^{-az} \sum_{n=0}^{b-1} \frac{(az)^n}{n!}], \quad z \geq 0$	$z = E1 + E2 + \dots + Eb,$ En — эксп. величины С параметром a .

В таблице 1 перечислены полезные случайные величины в свете нашего обсуждения. Для некоторых из них удается выписать формулу решения для обратной функции распределения, например, для экспоненциального распределения или для функции распределения Релея. В этих случаях имеется простая формула для выражения требуемых случайных чисел в терминах равномерных случайных чисел, как это проиллюстрировано в примере 1. В других случаях, как, например, в случае гауссовой или логарифмической нормальной плотности, такой простой формулы не существует. Тогда нужно искать альтернативный путь для

¹ Здесь $N(0,1)$ — нормальная (гауссова) случайная величина средним ноль и дисперсией 1. $U(0,1)$ — равномерная случайная величина из интервала $(0,1)$

реализации требуемой случайной величины. Например, для случайной величины z с логарифмической нормальной плотностью можно воспользоваться тем фактом, что величина $\ln(z)$ имеет гауссово распределение, и выписать распределение, приведенное в таблице 1, в терминах гауссовой случайной величины с нулевым средним и единичной дисперсией. И в других случаях бывает полезно переформулировать задачу для нахождения более легкого решения. Например. Можно показать, что случайные числа Эрланга с параметрами a и b можно построить, складывая b независимых случайных величин с экспоненциальным распределением, имеющих параметр a .

Генераторы случайных чисел, реализованные в ***imnoise*** и перечисленные в таблице 1, играют важную роль при моделировании поведения случайного шума и приложениях обработки изображений. Мы уже видели пользу от использования равномерного распределения при генерации случайных чисел с заданной функцией распределения. Гауссов шум используется в качестве естественного приближения в тех случаях, когда сенсорные детекторы изображения работают на пороге чувствительности. Шум типа «соль и перец» возникает в устройствах с ошибочной коммутацией. Размеры зерен на фотоэмulsionии являются случайными величинами с логарифмически нормальным распределением. Шум Релея образуется при фиксации удаленных изображений, а экспоненциальный шум и шум Эрланга используется при описании искажений на изображениях, полученных с помощью лазерного излучения.

М-функция ***imnoise2***, которая будет рассматриваться позже, генерирует случайные числа, которые имеют функцию распределения из списка в таблице 1. Это функция использует стандартную функцию **rand**, которая имеет следующий синтаксис:

$$A = \text{rand}(M, N).$$

Эта функция генерирует массив размера $M \times N$, элементами которого служат равномерно распределенные случайные величины в интервале $(0,1)$. Если аргумент N отсутствует, то по умолчанию он приравнивается к M . Если опущены оба аргумента, то **rand** генерирует скалярную случайную величину, которая меняется при каждом новом вызове функции **rand**. Аналогично функция

$$A = \text{randn}(M, N)$$

строит массив размера $M \times N$ с независимыми гауссовыми (нормальными) случайными величинами с нулевым средним и единичной дисперсией. Если N отсутствует, то по умолчанию $N = M$. Если аргументов нет, то генерируется скалярная случайная величина.

Функция ***imnoise2*** использует функцию **find** из MATLAB, которая имеет следующие синтаксические формы:

```
I = find (A)
[r, c] = find (A)
[r, c, v] = find (A) .
```

В I записываются индексы массива A, которые обозначают не нулевые элементы. Если таковые отсутствуют, то find возвращает пустую матрицу. Вторая форма возвращает строку и столбец, состоящие из индексов элементов матрицы A с нулевым содержимым, а в третьей форме так же возвращаются ненулевые элементы A в виде отдельного вектор-столбца v.

В первой форме команды матрица A трактуется в формате A(:), т.е. I – это вектор-столбец. Такая форма очень удобна при обработке изображений. Например, чтобы найти множество всех пикселей, меньше 128, и присвоить им значение 0, достаточно выполнить команды

```
>> I = find (A < 128);
>> A (I) = 0;
```

Напомним, что логическая формула A < 128 возвращает матрицу, в которой 1-цы стоят там, где логическое условие выполнено и 0 – там, где оно не выполнено. Чтобы присвоить значение 128 всем пикселям, принадлежащих замкнутому интервалу [64, 192] можно записать

```
>> I = find (A >= 64 & A <= 192);
>> A(I) = 128;
```

Первые две формы команды find будут часто использоваться в дальнейшем.

В отличие от imnoise, следующая функция imnoise2 порождает шумовую матрицу R MxN, которая не нормируется. Другое значительное отличие от функции imnoise состоит в том, что выходом imnoise служит зашумленное изображение, а imnoise2 порождает только шумовую матрицу. Пользователь сам должен задавать желаемые параметры шума. Заметьте, что шумовая матрица, генерируемая по методу «соль и перец», принимает три возможных значения: 0 – это «перец», 1 – это «соль», а 0,5 соответствует отсутствию шума. Эту матрицу необходимо еще обрабатывать для дальнейшего использования. Например, чтобы испортить изображение этим шумом, необходимо сначала найти (с помощью функции find) все координаты шума R, равные 0, и присвоить соответствующим пикселям изображения самое большое допустимое значение (обычно это 255 для 8-ми битовых изображений). Эта процедура моделирует практическое воздействие шума «соль и перец» на изображение.

```
function R = imnoise2(type, M, N, a, b)
if nargin == 1
    a = 0; b = 1;
    M = 1; N = 1;
elseif nargin == 3
```

```

a = 0; b = 1;
end
switch lower(type)
case 'uniform'
R = a + (b - a)*rand(M, N);
case 'gaussian'
R = a + b*randn(M, N);
case 'salt & pepper'
if nargin <= 3
a = 0.05; b = 0.05;
end
if (a + b) > 1
error('The sum Pa + Pb must not exceed 1.')
end
R(1:M, 1:N) = 0.5;
X = rand(M, N);
c = find(X <= a);
R(c) = 0;
u = a + b;
c = find(X > a & X <= u);
R(c) = 1;
case 'lognormal'
if nargin <= 3
a = 1; b = 0.25;
end
R = a*exp(b*randn(M, N));
case 'rayleigh'
R = a + (-b*log(1 - rand(M, N))).^0.5;
case 'exponential'
if nargin <= 3
a = 1;
end
if a <= 0
error('Parameter a must be positive for exponential type.')
end
k = -1/a;
R = k*log(1 - rand(M, N));
case 'erlang'
if nargin <= 3
a = 2; b = 5;
end
if (b ~= round(b) | b <= 0)
error('Param b must be a positive integer for Erlang.')
end
k = -1/a;
R = zeros(M, N);
for j = 1:b
R = R + k*log(1 - rand(M, N));
end
otherwise
error('Unknown distribution type.')
end

```

Пример 2. Построение гистограмм данных, сгенерированных функцией *imnoise2*.

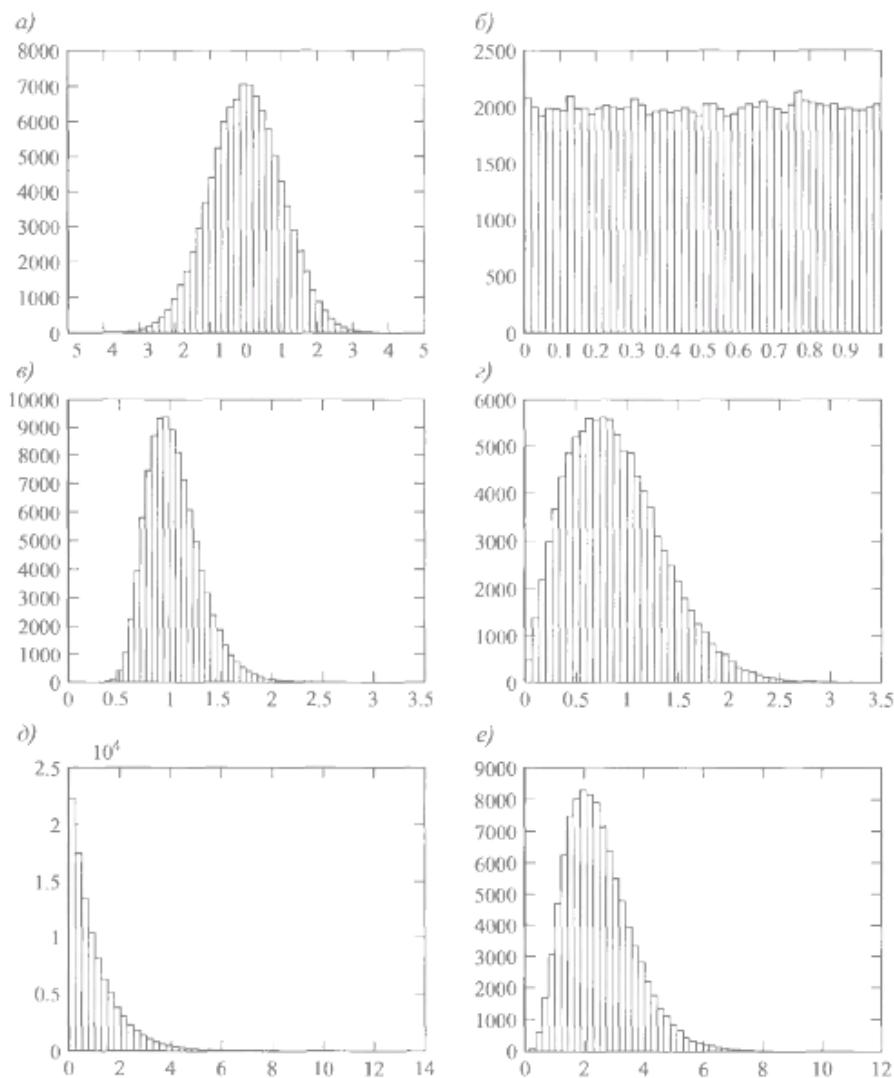


Рисунок 2 – Гистограммы случайных величин: *а)* гауссова, *б)* равномерная, *в)* логарифмически нормальная, *г)* релеевская, *д)* экспоненциальная, *е)* Эрланга.

В каждом случае использовались параметры, принятые по умолчанию в *imnoise2*

На рисунке 2 приведены гистограммы всех типов случайных чисел из таблице 1. Данные для каждого из этих графиков были сгенерированы с помощью функции *imnoise2*. Например, данные для рисунка 2*а* получаются следующей командой:

```
>> r = imnoise2('gaussian', 100000, 1, 0, 1);
```

Эта команда порождает вектор-столбец *r*, содержащий 100000 элементов, каждый из которых является случайной гауссовой величиной с нулевым средним и единичным стандартным отклонением. После этого гистограмма строится функцией *hist*, которая имеет синтаксис:

```
p = hist(r, bins),
```

где bins – это число корзин гистограммы. Чтобы построить гистограмму (рисунок 2а) мы взяли число bins = 50. Другие гистограммы строились аналогично. Во всех случаях всем параметрам шумов присваивались значения по умолчанию, перечисленных в соответствующих описаниях функции **imnoise2**.

Периодический шум

Периодический шум весьма типичен для оцифрованных изображений. Он возникает при интерференции различных электрических и электромеханических процессов. В этой главе будет рассматриваться только такой тип шума, который зависит от положения пикселя. С таким шумом можно справиться с помощью фильтрации в частотной области. Нашей моделью периодического шума будет двумерная синусоида, задаваемая формулой

$$r(x, y) = A \sin[2\pi u_0(x + B_x)/M + 2\pi v_0(y + B_y)/N],$$

где A – это амплитуда, u_0 и v_0 определяют синусоидальные частоты, соответственно по осям x и y , Bx и By – сдвиги фаз относительно начала отсчета. Дискретным преобразованием Фурье (DFT) размеров $M \times N$ от $r(x, y)$, является функция

$$R(u, v) = j \frac{A}{2} [(e^{j2\pi u_0 B_x / M}) \delta(u + u_0, v + v_0) - (e^{j2\pi v_0 B_y / N}) \delta(u - u_0, v - v_0)]$$

которая имеет вид пары комплексно сопряженных единичных импульсов, находящихся в точках (u_0, v_0) $(-u_0, -v_0)$ соответственно.

Следующая М-функция допускает произвольное число местоположений импульсов (координат частот), каждый со своей амплитудой, частотой и сдвигом фазы, и она вычисляет $r(x, y)$ в виде суммы синусоид в форме, представленной в форме суммы импульсов, а также спектр $S(u, v)$. Синусовые волны строятся по заданным положениям импульсов с помощью обратного DFT. Это позволяет лучше понять и визуально представить частотное содержание пространственных шумов. Для определения положения импульса достаточно знать одну пару координат. Программа строит сопряжено симметричные импульсы. Обратите внимание на использование здесь **ifftshift** для приведения R к виду, подходящему для применения операции **ifft2**.

```
function [r, R, S] = imnoise3(M, N, C, A, B)
K = size(C, 1);
if nargin < 4
    A = ones(1, K);
end
if nargin < 5
    B = zeros(K, 2);
end
R = zeros(M, N);
for j = 1:K
    u1 = floor(M/2) + 1 - C(j, 1);
    v1 = floor(N/2) + 1 - C(j, 2);
    u2 = M - u1;
    v2 = N - v1;
    R(u1, v1) = j * A(j) * exp(j * 2 * pi * C(j, 1) * u1 / M);
    R(u2, v1) = conj(R(u1, v1));
    R(u1, v2) = conj(R(u1, v1));
    R(u2, v2) = j * A(j) * exp(j * 2 * pi * C(j, 2) * v1 / N);
end
r = ifftshift(ifft2(R));
S = abs(r);

```

```

R(u1, v1) = i*M*N*(A(j)/2) * exp(-i*2*pi*(C(j, 1)*B(j, 1)/M ...
+ C(j, 2)*B(j, 2)/N));
u2 = floor(M/2) + 1 + C(j, 1);
v2 = floor(N/2) + 1 + C(j, 2);
R(u2, v2) = -i*M*N*(A(j)/2) * exp(i*2*pi*(C(j, 1)*B(j, 1)/M ...
+ C(j, 2)*B(j, 2)/N));
end
S = abs(R);
r = real(ifft2(fftshift(R)));

```

Пример 3. Использование, функции *innoise3*.

На рисунке 3а и 3б построен спектр и пространственный синусовый шум с помощью последовательности команд

```

>> C = [0 64; 0 128; 32 32; 64 0; 128 0; -32 32];
>> [r, R, S] = innoise3(512, 512, C);
>> imshow (S, [ ])
>> figure, imshow (r, [ ])

```

Напомним, что порядок частотных координат (u, v). Эти две величины обозначены по отношению к центру частотою прямоугольника. На рисунках 3в и 3г показан результат повторения предыдущих команд, но с параметрами

```
>> C = [0 32; 0 64; 16 16; 32 0; 64 0; -16 16];
```

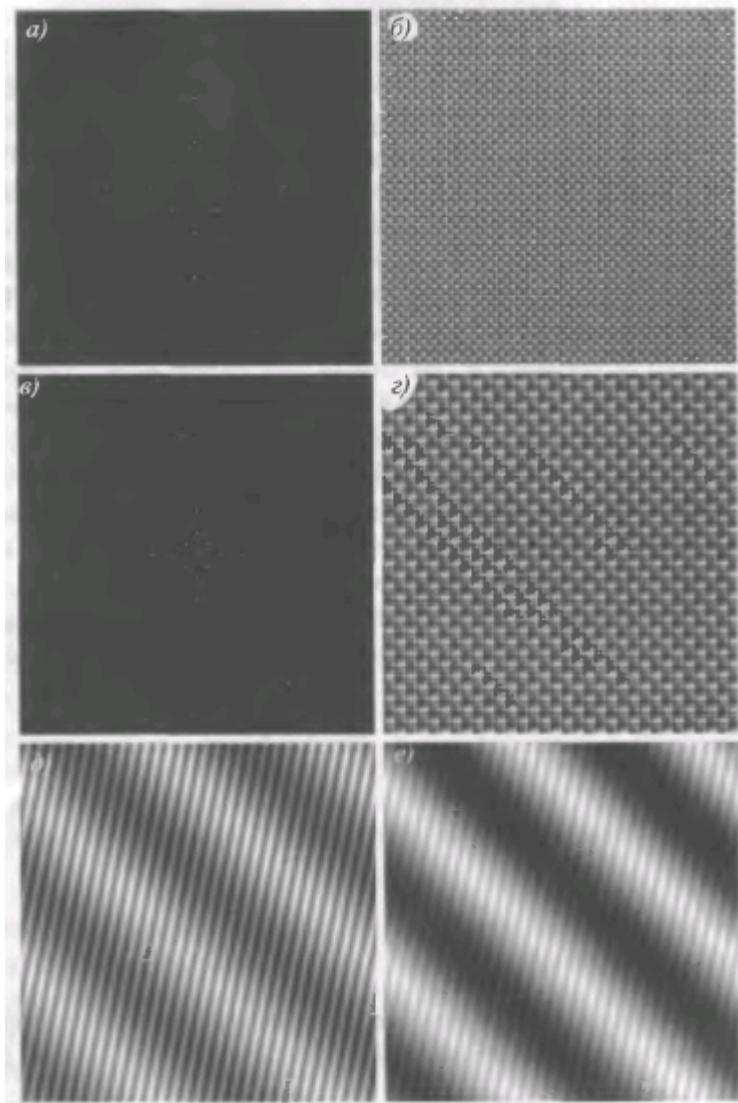


Рисунок 3 – а) Спектр заданных импульсов, б) Соответствующий синусовый шум. в) и г) Аналогичная пара изображений. д) и е) Два других шумовых образца. Белые точки на рис. а) и б) были увеличены для улучшения видимости

Аналогично, рисунок 3д построен при

```
>> C = [6 32;-2 2] ;
```

Рисунок 3е получен с тем же вектором С, но с модифицированным вектором амплитуд:

```
>> A = [15];
>> [Г, R, S] = imnoise3(512, 512, C, A);
```

Из рисунка 3е видно, что на изображении доминируют синусовые волны с низкой частотой. Это легко понять, так как их амплитуда в пять раз больше амплитуды высокочастотной компоненты.

Оценивание параметров шума

Параметры периодического шума можно оценить, анализируя спектр Фурье изображения. Периодический шум порождает частотные всплески, которые можно обнаружить даже визуально. Автоматический анализ возможен в случаях, когда эти всплески достаточно отчетливы или когда имеется дополнительная информация о частотах интерференции.

В случае пространственного шума параметры PDF можно частично узнать из спецификаций сенсоров, однако часто это делается с помощью анализа простых тестовых изображений. Соотношения между средней величиной m и дисперсией σ^2 шума и параметрами a и b , которые описывают шум, перечислены в таблице 1 для интересующих нас случайных величин. Поэтому задача заключается в оценивании среднего значения и дисперсии по тестовым образцам, которые можно подставить в уравнения для нахождения параметров a и b .

Пусть z_i — дискретная случайная величина, значениями которой являются уровни яркости изображения. Обозначим через $p(z_i)$, $i = 1, 2, \dots, L$ — 1 соответствующую нормированную гистограмму, где L — это число возможных значений яркости. Таким образом, число $p(z_i)$ приближает вероятность появления величины яркости z_i на изображении. Гистограмму можно рассматривать как приближение PDF яркости.

Используя стандартный метод, форму гистограммы можно описать с помощью статистических *центральных моментов*, которые вычисляются по формуле

$$\mu_n = \sum_{i=0}^{L-1} (z_i - m)^n p(z_i),$$

где n — это *порядок* момента, а m — среднее значение,

$$\mu_1 = \sum_{i=0}^{L-1} z_i p(z_i),$$

Поскольку гистограмма была нормирована, сумма всех ее компонентов равна 1, поэтому из приведенных формул следует, что $\mu_0 = 1$, $\mu_1 = 0$. Второй момент

$$\mu_2 = \sum_{i=0}^{L-1} (z_i - m)^2 p(z_i)$$

равен дисперсии случайной величины z . В этой главе нам понадобятся лишь среднее значение и дисперсия.

Функция **statmoments** вычисляет среднее и статистические моменты до порядка n включительно и возвращает эти величины в виде вектора-столбца v .

Поскольку моменты нулевого и первого порядков равны, соответственно, 1 и 0, statmoments их игнорирует и присваивает $v(1) = m$ и $v(k) = \mu_k$ при $k = 2, 3, \dots, n$. Синтаксис имеет вид (см. приложение B):

$$[v, unv] = statmoments(p, n),$$

где p — это вектор гистограммы, а n обозначает наибольший порядок

моментов. Требуется, чтобы число компонентов вектора p равнялось 2^8 для класса изображения `uint8`, 2^{16} для класса `uint16`, а также 2^8 или 2^{16} для класса `double`. Выходной вектор v содержит нормированные моменты, вычисленные на основе приведения случайных величин к диапазону $[0,1]$. Следовательно, все моменты будут также принадлежать этому интервалу. Вектор `unv` содержит те же моменты, что и v , но в диапазоне исходных данных. Например, если $\text{length}(p) = 256$ и $v(l) = 0.5$, то $uhv(1) = 127.5$, что равно половине от диапазона $[0, 255]$.

Часто бывает необходимо оценить параметры шума непосредственно по зашумленному изображению или по набору таких изображений. В этом случае следует выбрать на некотором изображении область, по возможности лишенную характерных черт, чтобы распределение яркости на ней было бы произведено главным образом шумом. Для выбора такой интересующей области (ROI, Region Of Interest) можно воспользоваться функцией `roipoly`, которая строит многоугольную область. Она имеет синтаксическую форму

$$B = \text{roipoly}(f, c, r),$$

где f – интересующее нас изображение, а c и r – векторы, соответствующие координатам (но столбцам и строкам) последовательных вершин многоугольника (заметьте, что в начале идут координаты по столбцам). Выходом функции служит двоичный массив того же размера, что и f , с единицами внутри интересующей области и с нулями вне ее. Изображение B используется в качестве маски для ограничения выполнения операций на выделенную область. Для интерактивного задания области ROI можно использовать синтаксис

$$B = \text{roipoly}(f).$$

В этом случае изображение f выводится на экран, и пользователю предлагается обозначить интересующую область с помощью мыши. Если параметр f опущен, функция оперирует с последним изображением на экране. Действуя левой кнопкой мыши, строим последовательные вершины многоугольника. Нажимая клавиши `Backspace` или `Delete`, можно удалить вершину, выбранную на предыдущем шаге. Нажимая левую кнопку мыши с одновременным удержанием клавиш `Shift`, правую кнопку мыши или сделав двойной щелчок левой кнопкой, добавляем последнюю вершину, после чего выбранный многоугольник заполняется единицами. Если нажать `Enter`, то построение многоугольника завершается без добавления последней вершины.

Чтобы получить бинарную маску и список вершин многоугольника, можно использовать конструкцию

$$[B, c, r] = \text{roipoly}(\dots),$$

где `roipoly` (\dots) обозначает любой допустимый пил этой функции, а c и r — векторы координат (по столбцам и строкам) вершин многоугольника, как

было описано раньше. Такой формат особенно удобен при интерактивном использовании roipoly, так как он выдает информацию об области ROI, которую можно копировать и использовать в последующих программах.

Следующая функция вычисляет гистограмму выбранной многоугольной области ROI на изображении, вершины которой заданы парой векторов *c* и *r*, как описано выше.

```
function [p, npix] = histroi(f, c, r).
B = roipoly(f, c, r);
p = imhist(f(B));
if nargout > 1
    npix = sum(B(:))
end
```

Пример 4. Оценивание параметров шума.

На рисунке 4а показано изучаемое изображение *f*. В этом примере мы воспользуемся представленными выше инструментами для определения типа шума и оценивания его параметров. На рисунке 4б изображена маска *B*, построенная интерактивно командой

```
>> [B, c, r] = roipoly(f);
```

Рисунок 4в получен при выполнении команд

```
>> [p, npix] = histroi(f, c, r);
>> figure, bar(p, 1);
```

Среднее значение и дисперсия области исходного изображения, накрываемой маской *B*, вычислены по формулам

```
>> [v, unv] = statmoments(h, 2);
>> v
v=
0.0063
unv
unv=
4109313
```

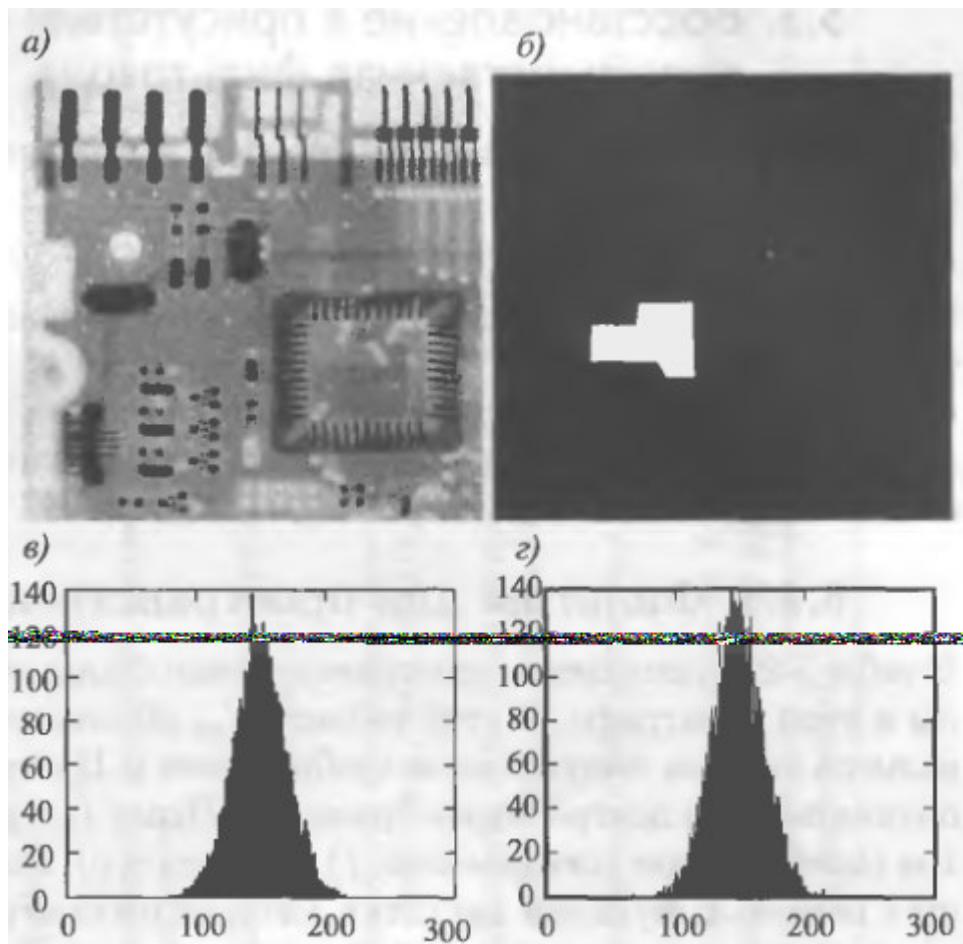


Рисунок 4 – а) Зашумленное изображение, б) Область ROI, построенная интерактивно, в) Гистограмма ROI г) Гистограмма Гауссова шума, сгенерированная функцией `imnoise2` (Исходное изображение представлено компанией Lixi, Inc.)

Из рисунка 4в видно, что шум похож на туесов. На самом деле, найти точное значение среднего и дисперсии шума в области В не представляется возможным, так как шум наложен на конкретное изображение. Тем не менее, выбирая однородную область без фоновых деталей (как это было сделано выше) и полагая шум гауссовым, можно считать, что средняя яркость изображения с шумом приблизительно равна средней яркости этого изображения без шума (мы считаем, что гауссов шум имеет нулевое среднее). Кроме того, поскольку выбранная область достаточно однородна по яркости, разумно утверждать, что дисперсия яркости в области В прежде всего определяется дисперсией шумового слагаемого. (Там, где это возможно, другой метод оценивания среднего и дисперсии шума состоит в использовании тестового изображения постоянной яркости.) На рисунке 4г приведена гистограмма множества из `pr1x` (это число возвращает функция `histroi`) случайных гауссовых величин со средним 147 и дисперсией 400, построенная командами

```
>> X = imnoise2('gaussian', npix, I, 147, 20);
>> figure, hist(X, 130)
```

```
>> axis ([0 300 0 140])
```

где число корзин в `hist` выбрано так, чтобы результат можно было сравнить с графиком на рисунке 4в. Эта гистограмма получена функцией `histroi` (см. предыдущий программный код), которая использует масштаб, отличный от масштаба, функции `hist`. Мы выбрали при σ случайных гауссовых величин и записали их в вектор X , чтобы иметь одинаковое число отсчетов па обеих гистограммах. Схожесть графиков на рисунке 4в и 4г ясно указывает па то, что шум, и в самом деле, хорошо аппроксимирован гауссовым распределением, параметры которого взяты из оценок $v(1)$ и $v(2)$.

Восстановление в присутствии одного шума - пространственная фильтрация

Если искажения вносятся в изображение исключительно шумом, то

$$g(x, y) = f(x, y) + \eta(x, y)$$

В этом случае методом удалении шума может служить пространственная фильтрация с использованием техники.

Фильтры для пространственного шума

В таблице 2 перечислены пространственные фильтры, которые будут рассмотрены в этом параграфе. В этой таблице S_{xy} обозначает под изображение (область) размера m^*n на зашумленном изображении d . Нижний индекс y S обозначает координаты (x, y) центра под изображения. Через $\hat{f}(x, y)$ обозначается отклик фильтра (приближение изображения f) в точке (x, y) . Линейные фильтры реализованы с помощью функции `medfilter`. Медианный, минимальный и максимальный фильтры являются нелинейными фильтрами порядковых статистик. Медианный фильтр можно непосредственно реализовать на базе функции `medfilt2` из пакета IPT. Максимальный и минимальный фильтры можно реализовать с, помощью более общей функции `ordfilt2` для нахождения порядковых статистик.

Мы построим функцию `spfilt`, которая совершає фильтрацию в пространственной области с помощью любого фильтра из таблицы 2. Обратите внимание па использование функции `imlincomb` (упомянутой в таблице 2) при вычислении линейной комбинации входных данных. Синтаксис этой функции имеет вид

$$B = imlincomb(c1, A1, c2, A2, \dots, ck, Ak).$$

Она реализует формулу

$$B = c1*A1 + c2*A2 + \dots + ck*Ak,$$

где c_i - это скалярные величины двойной точности, а A_i - числовые массивы

одного класса и одинаковых размеров. Заметьте также, как в подфункции gmean можно включить и выключить функцию warning. В этом случае имеется возможность подавить предупреждение, которое выдает MATLAB, когда аргумент функции log становится равен 0. В общем случае, функцию warning можно использовать в любой программе. Ее базовый синтаксис имеет вид

```
warning ('message').
```

Эта функция ведет себя в точности, как и функция disp, за тем исключением, что ее можно включать и выключать командой warning on или warning off.

```
function f = spfilt (g, type, m, n, parameter)
```

Таблица 2. Пространственные фильтры. Переменные m и n, соответственно обозначают число строк и столбцов окрестности фильтрации.

Имя фильтра	Уравнение	Комментарии
Арифметическое среднее	$\hat{f}(x, y) = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s, t)$	Реализуется с помощью функции IPT w = fspecial('average', [m, n]) и f = imfilter(g, w).
Геометрическое среднее	$\hat{f}(x, y) = \left[\prod_{(s,t) \in S_{xy}} g(s, t) \right]^{\frac{1}{mn}}$	Реализуется функцией gmean (см. функцию spfilt в этом параграфе)
Гармоническое среднее	$\hat{f}(x, y) = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s, t)}}$	Реализуется функцией harmean (см. функцию spfilt в этом параграфе)
Контрагармоническое среднее	$\hat{f}(x, y) = \frac{\sum_{(s,t) \in S_{xy}} g(s, t)^{Q+1}}{\sum_{(s,t) \in S_{xy}} g(s, t)^Q}$	Реализуется функцией charmean (см. функцию spfilt в этом параграфе)
Медиана	$\hat{f}(x, y) = median\{g(s, t)\}_{(s,t) \in S_{xy}}$	Реализуется с помощью функции IPT medfilt2: f = medfilt2(g, [m,n]).
Максимум	$\hat{f}(x, y) = \max\{g(s, t)\}_{(s,t) \in S_{xy}}$	Реализуется с помощью функции IPT ordfilt2: f = ordfilt2(g, m*n, ones(m,n)).
Минимум	$\hat{f}(x, y) = \min\{g(s, t)\}_{(s,t) \in S_{xy}}$	Реализуется с помощью функции IPT ordfilt2: f = ordfilt2(g, 1, ones(m,n)).
Срединная точка	$\hat{f}(x, y) = \frac{1}{2} \left[\max_{(s,t) \in S_{xy}} \{g(s, t)\} + \min_{(s,t) \in S_{xy}} \{g(s, t)\} \right]$	Реализуется в виде полусуммы фильтров максимума и минимума

<i>a</i> -усеченное среднее	$\hat{f}(x,y) = \frac{1}{mn-d} \sum_{(s,t) \in S_{xy}} g_r(s,t)$	Из области S_{xy} удаляются $d/2$ наибольших и $d/2$ наименьших значений пикселей. $gr(s,t)$ обозначает оставшиеся $mn-d$ значения окрестности. Реализуется с помощью функции alphatrim (см. Функцию spfilt)
-----------------------------	--	---

```

function f = spfilt(g, type, m, n, parameter)
if nargin == 2
    m = 3; n = 3; Q = 1.5; d = 2;
elseif nargin == 5
    Q = parameter; d = parameter;
elseif nargin == 4
    Q = 1.5; d = 2;
else
    error('Wrong number of inputs.');
end
switch type
case 'amean'
    w = fspecial('average', [m n]);
    f = imfilter(g, w, 'replicate');
case 'gmean'
    f = gmean(g, m, n);
case 'hmean'
    f = harmean(g, m, n);
case 'chmean'
    f = charmean(g, m, n, Q);
case 'median'
    f = medfilt2(g, [m n], 'symmetric');
case 'max'
    f = ordfilt2(g, m*n, ones(m, n), 'symmetric');
case 'min'
    f = ordfilt2(g, 1, ones(m, n), 'symmetric');
case 'midpoint'
    f1 = ordfilt2(g, 1, ones(m, n), 'symmetric');
    f2 = ordfilt2(g, m*n, ones(m, n), 'symmetric');
    f = imlincomb(0.5, f1, 0.5, f2);
case 'atrimmed'
    if (d <= 0) | (d/2 ~= round(d/2))
        error('d must be a positive, even integer.')
    end
    f = alphatrim(g, m, n, d);
otherwise
    error('Unknown filter type.')
end

function f = gmean(g, m, n)
inclass = class(g);
g = im2double(g);

```

```

warning off;
f = exp(imfilter(log(g), ones(m, n), 'replicate')).^(1 / m / n);
warning on;
f = changeclass(inclass, f);
function f = harmean(g, m, n)
inclass = class(g);
g = im2double(g);
f = m * n ./ imfilter(1./(g + eps), ones(m, n), 'replicate');
f = changeclass(inclass, f);
function f = charmean(g, m, n, q)
inclass = class(g);
g = im2double(g);
f = imfilter(g.^q+1, ones(m, n), 'replicate');
f = f ./ (imfilter(g.^q, ones(m, n), 'replicate') + eps);
f = changeclass(inclass, f);
function f = alphatrim(g, m, n, d)
inclass = class(g);
g = im2double(g);
f = imfilter(g, ones(m, n), 'symmetric');
for k = 1:d/2
    f = imsubtract(f, ordfilt2(g, k, ones(m, n), 'symmetric'));
end
for k = (m*n - (d/2) + 1):m*n
    f = imsubtract(f, ordfilt2(g, k, ones(m, n), 'symmetric'));
end
f = f / (m*n - d);
f = changeclass(inclass, f);

```

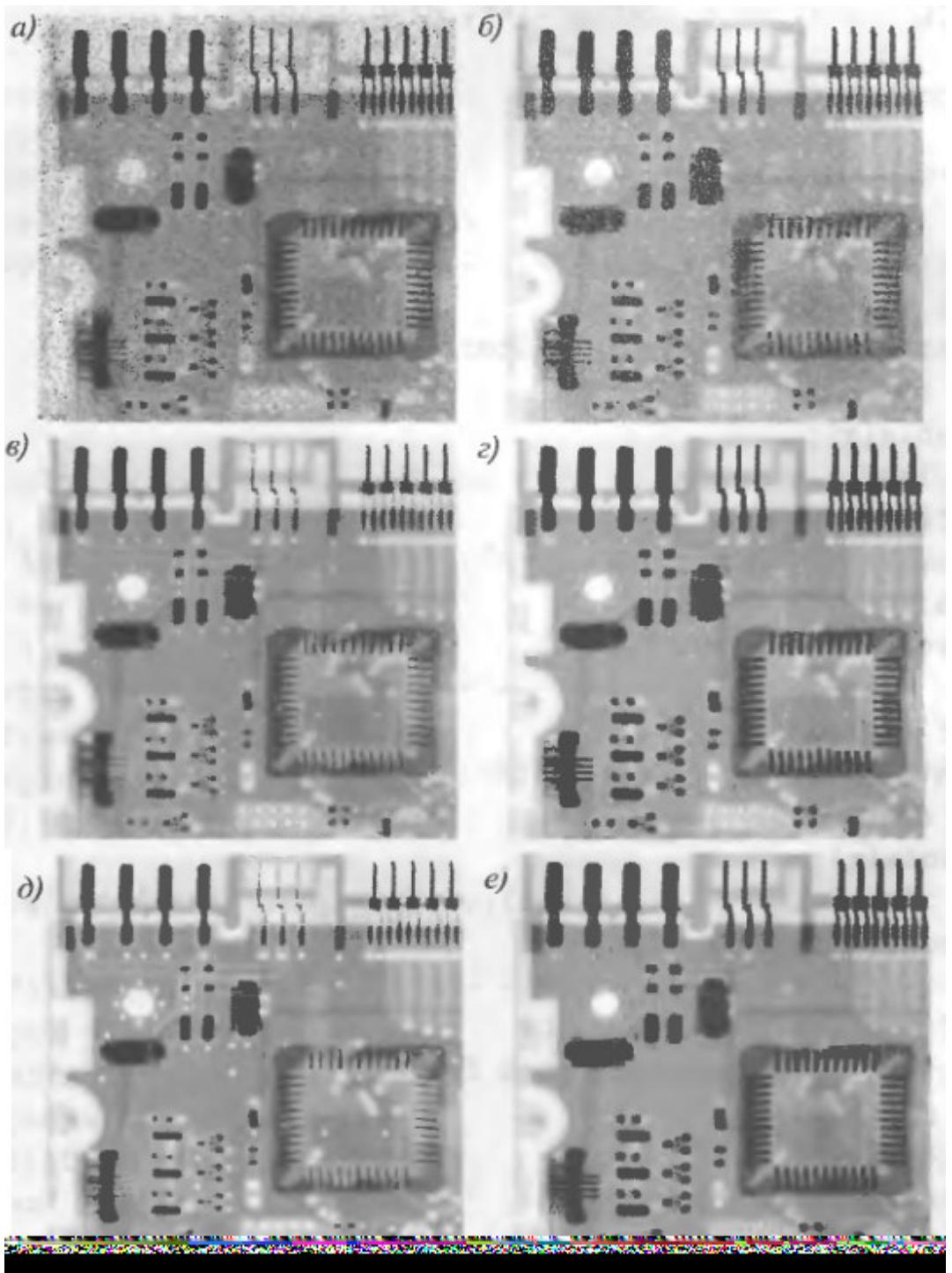


Рисунок 5 – а) Изображение, испорченное шумом «перец» с вероятностью 0.1. б) Изображение, испорченное шумом «соль» с вероятностью 0.1 в) Результат фильтрации а) контрагармоническим фильтром размера 3x3 порядка $Q = 1.5$. г) Результат фильтрации б) при $Q = -1.5$. д) Результат фильтрации а) максимальным фильтром размера 3x3. е) Результат фильтрации б) минимальным фильтром размера 3x3

Пример 5. Использование функции spfilt.

На рисунке 5а приведено изображение класса uint8, испорченное только шумом «перец» с вероятностью 0.1. Это изображение построено следующими командами

```

>> [M, N] = size(f);
>> R = imnoise2('salt & paper', M, N, 0.1, 0);
>> c = find(R == 0);
>> gp = f;
>> gp(c) = 0;

```

Рисунок 5б, подпорченный только шумом «соль», получен командами

```

>> R = imnoise2('salt & paper', M, N, 0, 0.1);
>> c = find(R == 1);
>> gs = f ;
>> gs(c) = 255;

```

Избавиться от шума «перец» можно с помощью контрагармонического фильтра с положительным значением Q . Рисунок 5в построен командой

```
>> fp = spfilt(gp, 'chmean', 3, 3, 1.5);
```

Аналогично для борьбы с шумом «соль» можно воспользоваться контрагармонического фильтра с отрицательным значением Q :

```
>> fs = spfilt(gs, 'chmean', 3, 3, -1.5);
```

На рисунке 5г показан результат. Можно также действовать минимальным и максимальным фильтрами. Например, рисунок 5д и 5е получены, соответственно, из 5а и 5б с помощью следующих команд

```

>> fpmax = spfilt(gp, 'max', 3, 3);
>> fpmin = spfilt(gs, 'min', 3, 3);

```

Другие решения с использованием функции `fpmi` можно получать аналогичным образом.

Адаптивные пространственные фильтры

Приведенные выше фильтры применяются одинаково ко всем пикселям изображения независимо от их местоположения. В некоторых случаях результат фильтрации можно улучшить, если фильтр имеет возможность менять свое действие в зависимости от характеристик изображения в области фильтрации. В качестве иллюстрации реализации адаптивной процедуры пространственной фильтрации на MATLAB рассмотрим адаптивный медианный фильтр. Как и раньше, S_{xy} обозначает окрестность фильтрации с центром в точке (x, y) на изображении, подвергаемом обработке. Алгоритм действует следующим образом. Обозначим

z_{\min} = минимум яркости в S_{xy} ,
 z_{\max} = максимум яркости в S_{xy} ,
 z_{med} = медиана яркости в S_{xy}
 z_{xy} = яркость в точке (x, y) .

Адаптивный алгоритм медианной фильтрации действует на двух уровнях, которые назовем уровень А и уровень В:

Уровень А Если $z_{\min} < z_{\text{med}} < z_{\max}$, то перейти на уровень В, иначе увеличить размер окна.

Если размер окна $<=$, то повторить уровень А, иначе подать на выход величину z_{med} .

Уровень В Если $z_{\min} < z_{xy} < z_{\max}$, то подать на выход z_{xy} , иначе подать на выход величину z_{med} .

Здесь S_{\max} обозначает максимально допустимый размер окна адаптивного фильтра. Другая особенность последнего шага на уровне А состоит в возвращении величины z_{xy} вместо медианы. Это приводит к немного меньшему размытию изображения, однако такая процедура может не заменить шум типа «соль» («перец») на постоянном заднем фоне, имеющем то же значение, что и шум «соль» («перец»).

М-функция, которая реализует этот алгоритм называется `admedian`. Она включена в приложении В и имеет синтаксис:

$$f = \text{admedian}(g, S_{\max}),$$

где g — это фильтруемое изображение. S_{\max} — определенный выше максимально допустимый размер окна адаптивной фильтрации.

Пример 6. Адаптивная медианная фильтрация.

В качестве примера рассмотрим снимок f монтажной платы на рисунке 6а, испорченный шумом «соль и перец», который наложен па изображение командой

```
>> g = imnoise(f, 'salt & paper', 0.25);
```

а на рисунке 6б приведен результат выполнения следующей команды

```
>> f1 = medfilt2(g, [7 7], 'symmetric');
```

Это изображение, очевидно, очищено от шума, но одновременно с этим оно стало слишком расплывчатым и искаженным (обратите внимание на то, как выглядят штырьки разъемов вверху изображения). Напротив, совершив действие

```
>> f2 = admedian(g, 7);
```

мы получим изображение 6в, на котором шум также отсутствует, но оно существенно четче, чем рисунок 6б.

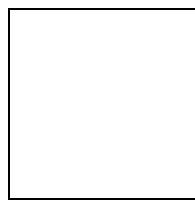


Рисунок 6 – а) Изображение, испорченно шумом «соль и перец» с плотностью ошибок 0.25. б) Результат применения медианного фильтра размеров 7x7. в) Результат адаптивной медианной фильтрации с $S_{\max}=7$

Подавление периодического шума с помощью фильтрации в частотной области

Периодический шум обнаруживается в виде импульсно-подобных

всплесков, которые можно наблюдать в его спектре Фурье. Основной метод борьбы с таким шумом основан на режекторной фильтрации. Передаточная функция режекторного фильтра Баттервортса порядка n имеет вид

$$H(u, v) = \frac{1}{1 + \left[\frac{D_0^2}{D_1(u, v)D_2(u, v)} \right]^n},$$

где

$$D_1(u, v) = [(u - M/2 - u_0)^2 + (v - N/2 - v_0)^2]^{1/2}$$

и

$$D_2(u, v) = [(u - M/2 + u_0)^2 + (v - N/2 + v_0)^2]^{1/2}$$

Здесь (u_0, v_0) и (в силу симметрии) $(-u_0, -v_0)$ — центры расположения «выемок», Радиус которых равен D_0 . Обратите внимание на то, что фильтр задан относительно центра частотного прямоугольника, т.е. его следует обработать функцией `fftshift` перед тем, как использовать в фильтрации средствами MATLAB.

Моделирование искажающих функций

Если имеется оборудование, генерирующее искаженные изображения, то можно установить природу искажения, экспериментируя с различными установками и параметрами этого оборудования. Однако доступность такого оборудования является скорее исключением, чем правилом. Более типичной является ситуация, когда производятся эксперименты с различными функциями PSF, которые тестируются с разными алгоритмами восстановления изображений. Другой подход состоит в математическом моделировании самих функций PSF. Такой подход выходит за рамки обсуждения в этой книге.

Одной из основных трудностей, с которой приходится сталкиваться при решении задач восстановления изображений, является проблема размытия и смазывания изображений. Размытие, возникающее, когда сцена и регистрирующее устройство находятся в покое по отношению друг к другу, можно смоделировать низкочастотными фильтрами в пространственной или частотной областях. Другая важная модель искажения изображений смазыванием соответствует равномерному Прямолинейному перемещению сцены относительно регистрирующих устройств и датчиков в процессе фиксирующей съемки. В этом случае размытие изображений можно смоделировать с помощью функции `fspecial` из пакета IPT:

`PSF = fspecial('motion', len, theta).`

Эта форма вызова `fspecial` возвращает PSF, которая аппроксимирует эффекты линейного перемещения камеры на `len` пикселов. Угловой параметр

theta измеряется в градусах, причем он отсчитывается от положительной горизонтальной полуоси против часовой стрелки. Значения по умолчанию: len = 9 и theta = 0, что соответствует смещению на 9 пикселов в горизонтальном направлении.

Мы используем функцию imfilter для построения изображения, искаженного фильтром PSF, который или известен, или построен приведенной выше командой:

```
>> g = imfilter(f, PSF, 'circular');
```

где параметр 'circular' используется для подавления граничных эффектов. После чего искажение вносится в изображение в виде аддитивного шума по формуле

```
>> g = g + noise;
```

где noise - это случайное шумовое изображение, которое имеет тот же размер, что и g, и строится одним из методов.

Изображение, которое строится функцией **checkerboard**, особенно удобно в этом смысле, поскольку у него очень просто изменять масштаб, не затрагивая его характерных черт. Функция имеет следующий синтаксис

```
C = checkerboard(NP, M, N),
```

где NP - это длина в пикселях каждой клетки, M — это число строк, а N — число столбцов шахматной доски. Если параметр N опущен, то он по умолчанию приравнивается к M. Если они оба опущены, то строится классическая шахматная доска 8x8. Наконец, если отсутствует NP, то по умолчанию NP = 10 пикселов. Светлые клетки в левой половине шахматной доски будут белыми, а в правой половине они будут светло-серыми. Чтобы построить шахматную доску, у которой все светлые клетки будут белыми, следует дать команду

```
>> K = im2double(checkerboard(NP, M, N) > 0.5);
```

Изображение, которое строит функция checkerboard, имеет класс double со значениями в интервале [0,1].

Многие алгоритмы восстановления изображений работают медленно с большими изображениями. Поэтому имеет смысл экспериментировать с малыми изображениями для сокращения времени вычислений, что позволяет скорее оптимизировать алгоритм. В этом случае для целей визуализации полезно иметь возможность увеличивать изображение посредством дублирования пикселей. Это делается с помощью следующей функции.

```
B = pixeldup(A, m, n).
```

Эта функция дублирует каждый пиксель m раз по вертикали и n раз по горизонтали. Если n пропущено, то по умолчанию $n = m$.

Пример 7. *Моделирование размытого зашумленного изображения.*
На рисунке 7а построено изображение с помощью команды

```
>> f = checkerboard(8);
```

Испорченное изображение 7б получено командами

```
>> PSF = fspecial('motion', 7, 45);
>> gb = imfilter(f, PSF, 'circular');
```

Заметьте, что PSF – это пространственный фильтр. Он имеет следующие значения:

```
>> PSF
PSF =
0000000000 0 0 0 0 0.0145 0
0 0 0 0 0.0376 0.1283 0.0145
0 0 0 0.0376 0.1283 0.0376 0
0 0 0.0376 0.1283 0.0376 0 0
0 0.0376 0.1283 0.0376 0 0 0
0.0145 0.1283 0.0376 0 0 0 0
0 0.0145 0 0 0 0 0
```

Шумовое изображение на рисунке 7в сгенерировано командой

```
>> noise = imnoise(zeros(size(f), 'gaussian', 0, 0.001);
```

Шум можно добавить к изображению прямо в команде `imnoise(gb, 'gaussian', 0, 0.001)`. Однако это шумовое изображения нам еще понадобится в этой главе, поэтому мы его здесь вычисляем отдельно.

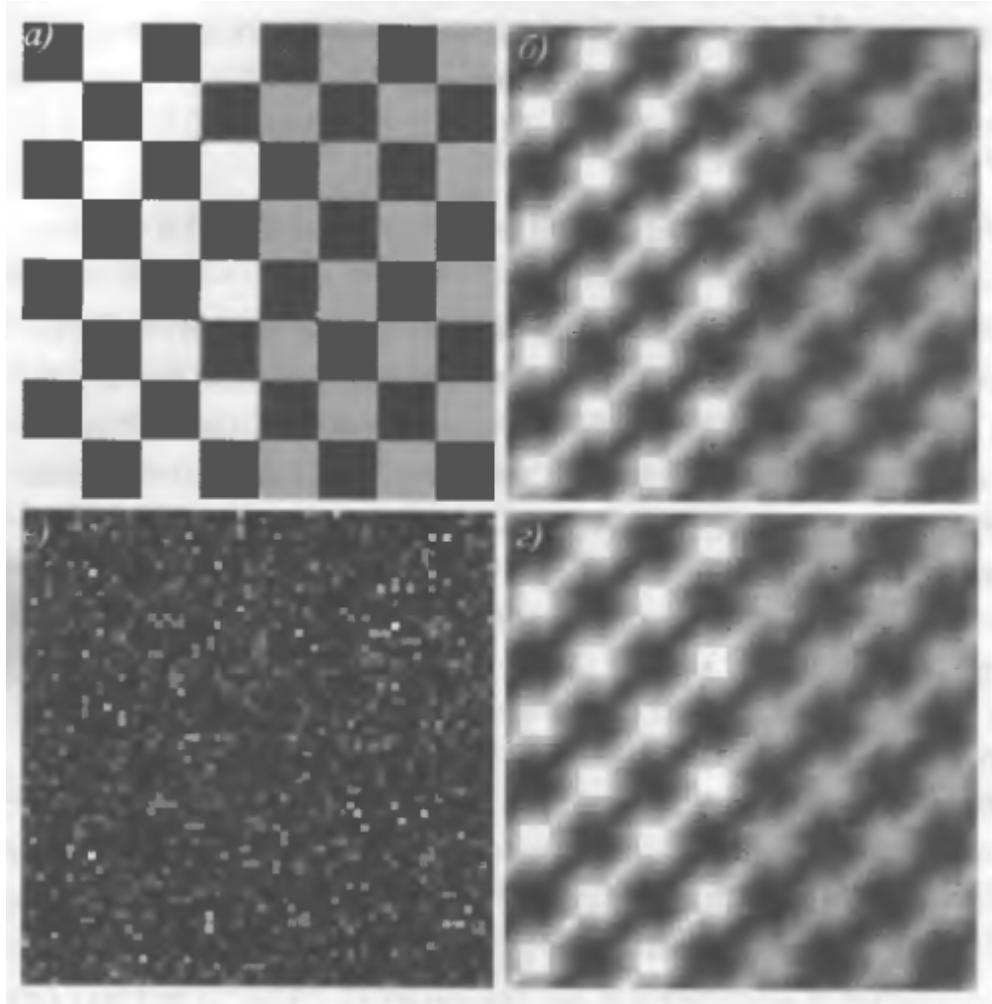


Рисунок 7 – а) Исходное изображение. б) изображение, размытое функцией fspecial с параметрами len = 7 theta = -45. в) Шумовое изображение. г) Сумма б) и в)

Смазанное и зашумленное изображение на рисунке 7г строится по команде

```
>> gb = g + noise;
```

Этот шум трудно визуально обнаружить на этом изображении, так как его максимальное значение имеет порядок 0.15, а максимальное значение пикселей изображения равно 1. Отметим, что все изображения на рисунке 7 были сжаты до размеров 512x512 и отображены командой типа

```
>> imshow(pixelup(f, 8), [ ] ).
```

Изображение на рисунке 7г будет восстановлено на рисунке 8 и 9.

Инверсная фильтрация

Самый простой подход к восстановлению испорченного изображения состоит в построении приближения вида

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)},$$

после чего следует применить обратное преобразование Фурье к функции $\hat{F}(u, v)$ (напомним, что $G(u, v)$ — это преобразование Фурье испорченного изображения). Этот подход удобно называть инверсной фильтрацией. Этот метод можно выразить в виде следующего приближения

$$\hat{F}(u, v) = F(u, v) + \frac{N(u, v)}{H(u, v)}.$$

Это обманчиво простое выражение говорит нам о том, что даже если мы знаем $H(u, v)$ точно, то мы не можем восстановить $F(u, v)$ (а значит, и оригинальное, неиспорченное изображение $f(n, v)$), поскольку шумовая компонента является случайной величиной, преобразование Фурье которой $N(u, v)$ остается неизвестной. Кроме того, на практике обычно имеются большие проблемы с функцией $H(u, v)$, которая может иметь нулевые значения. Даже если член $N(u, v)$ пренебрежимо мал, деление его на малые значения $H(u, v)$ может дать нежелательно большую прибавку к приближению $\hat{F}(u, v)$.

Типичный подход к совершению инверсной фильтрации состоит в построении частного $\hat{F}(u, v) = G(u, v)/H(u, v)$, у которого необходимо ограничить частотный диапазон «малой» окрестностью начала отсчета, а затем делать обратное преобразование. Идея здесь заключается в том, что нули функции $H(u, v)$ с меньшей степенью вероятности будут располагаться «близко» от начала частотных координат, так как амплитуда преобразования в этой области равна наибольшему значению этой величины. Имеется множество вариаций на эту тему, при которых по-разному разбираются значения (u, v) функции H в нуле или около нуля. Такой подход иногда называется *псевдоинверсной фильтрацией*. Но, вообще говоря, подходы, основанные на инверсной фильтрации такого типа, не очень точны, что видно из примера 5.8 в следующем параграфе.

Винеровская фильтрация

Винеровская фильтрация (названная в честь Н. Винера, предложившего этот метод в 1942 г.) является одним из самых старых и хорошо известных подходов в линейном восстановлении изображений. Винеровский фильтр ищет приближение \hat{f} , которое минимизирует среднеквадратическое отклонение

$$e^2 = E(f - \hat{f})^2,$$

где E — оператор математического ожидания, а f — неискаженное изображение. Решение этой экстремальной задачи в частотной области выражается формулой

$$\hat{F}(u, v) = \left[\frac{1}{H(u, v)} * \frac{|H(u, v)|^2}{H(u, v)|H(u, v)|^2 + S_\eta(u, v)/S_f(u, v)} \right] G(u, v),$$

где $H(u,v)$ – искажающая функция; $|H(u,v)|^2 = H^*(u,v)H(u,v)$; $H^*(u,v)$ – комплексно-сопряженная функция $H(u,v)$; $S\eta(u,v) = |N(u,v)|^2$ – энергетический спектр шума; $S_f(u,v) = |F(u,v)|^2$ – спектр неискаженного изображения; частное $S\eta(u,v)/S_f(u,v)$ называется *энергетическим соотношением шум/сигнал* (NSPR, Noise-to-Signal Power Ratio). Видно, что если спектр шума равен нулю для всех значений u и v , то это соотношение также равно нулю, и винеровский фильтр приводится к инверсному фильтру, который обсуждался в предыдущем параграфе.

Определим две полезные величины, которые называются *средняя энергия шума и средняя энергия изображения*, соответственно,

$$\eta A = \frac{1}{MN} \sum_u \sum_v S_\eta(u,v) \quad u \quad fA = \frac{1}{MN} \sum_u \sum_v S_f(u,v).$$

Здесь M и N обозначают вертикальный и горизонтальный размеры массивов изображения и шума. Эти величины являются скалярами, а их частное

$$R = \frac{\eta A}{fA},$$

которое также скалярно, иногда используется при построении постоянной матрицы, которая используется вместо $S\eta(u,v)/S_f(u,v)$. В этом случае, даже при неизвестном истинном соотношении шум/сигнал, получается простой способ интерактивного экспериментирования путем изменения этой константы и наблюдении результатов восстановления. Конечно, такой метод является весьма грубым, когда предполагается, что все функции являются константами. Замена $S\eta(u,v)/S_f(u,v)$ на константу R в приведенной выше формуле для $F^\wedge(u,v)$ называется *параметрическим винеровским фильтром*, В примере 5.8 показано, что даже такой простой прием может привести к существенно более лучшему результату, чем при инверсной фильтрации.

Винеровская фильтрация реализована в IPT с виде функции deconvwnr, которая имеет три возможные синтаксические формы. Во всех этих формах g обозначает искаженное, а fr – восстановленное изображение. Первая синтаксическая форма

```
fr = deconvwnr(g, PSF)
```

предполагает, что соотношение шум/сигнал равно нулю. Следовательно, такая форма винеровского фильтра совпадает с инверсным фильтром. Синтаксис

```
textfn = deconvwnr(g, PSF, NSPR)
```

предполагает, что соотношение шум/сигнал известно или в виде константы, или в виде массива; годится и то и другое. Этот синтаксис используется для реализации параметрической винеровской фильтрации. В этом случае аргумент NSPR может служить интерактивным входным параметром. Наконец, команда вида

```
fr = deconvwnr(g, PSF, NACORR, FACORR)
```

предполагает известными автокорреляционные функции NACORR и FACORR шума и неискаженного изображения. Обратите внимание на то, что в функции deconvnr используются автокорреляции η и f , а не энергетический спектр этих функций. По теореме о корреляции заключаем, что

$$|F(u, v)|^2 = \xi [f(x, y)^\circ f(x, y)],$$

где «о» обозначает операцию корреляции, а ξ — преобразование Фурье. Это уравнение дает возможность вычислить автокорреляционную функцию $f(x, y)^\circ f(x, y)$ для использования в deconvnr, находя обратное преобразование Фурье энергетического спектра. То же можно сказать про автокорреляционную функцию шума.

Если восстановленное изображение демонстрирует искажения типа «звоны», которые вносятся дискретным преобразованием Фурье, используемым в алгоритме, то иногда с этим явлением позволяет справиться функция edgetaper, которую следует применить до выполнения функции deconvnr. Ее синтаксис имеет вид

```
J = edgetaper(I, PSF).
```

Эта функции смазывает края входного изображения I, используя функцию разброса точек PSF. Выходное изображение J есть взвешенная сумма I и его размытой версии. Весовая матрица, которая определяется автокорреляционной функцией PSF, присваивает J значения I внутри изображения, а около границ J приравнивается размытой версии I.

Пример 8. Применение функции *deconvnr* при восстановлении размытого зашумленного изображения.

Рисунок 8а совпадает с рисунком 7г, а рисунок 8б получен командой

```
>> fr1 = deconvnr(g, PSF);
```

где g — это искаженное изображение, а PSF функции разброса точек вычислена в примере 7. Как уже отмечалось выше, fr1 — это результат прямого применения инверсной фильтрации, и, как ожидалось, на этом изображении преобладают шумовые эффекты. (Как и в примере 7, все изображения были обработаны функцией pixeldup для их увеличения до размера 512x512 пикселей.)

Число R, которое обсуждалось ранее: в этом параграфе, было получено с помощью исходного и шумового изображений из примера 7 по формулам

```
>> Sn = abs(fft2(noise)).^2; % noise power spectrum
```

```

>> nA = sim(Sn(:))/prod(size(noise)); % noise average power
>> Sf = abs(fft2(f)).^2; % image power spectrum
>> fA = sum(Sn(:))/prod(size(noise)); % image average power
>> R = nA/fA;

```

Для восстановления изображения с помощью этого соотношения R выполним команду

```
>> fr2 = deconvwnr(g, PSF, R);
```

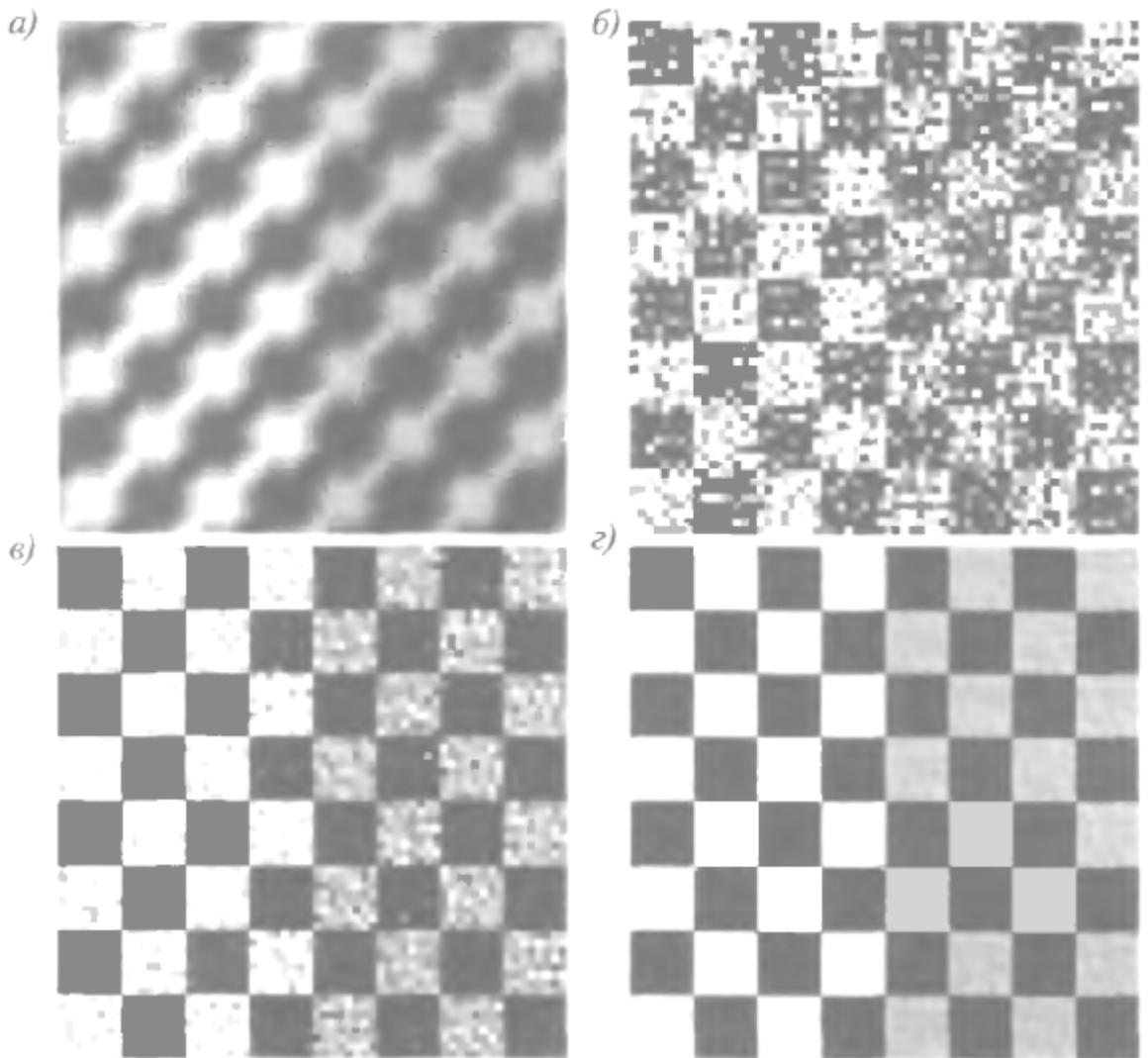


Рисунок 8 – а) Размытое и зашумленное изображение. б) Результат инверсной фильтрации. в) Результат винеровской фильтрации. г) Результат винеровской фильтрации с использованием автокорреляционных функций.

Из рисунка 8в видно, что такой подход дает существенное улучшение по сравнению с инверсной фильтрацией.

Наконец, используем автокорреляционные функции при восстановлении изображения (напомним о необходимости использования функции fftshift при центровании).

```

>> NCORR = fftshift(real(ifft2(Sn)));
>> ICDRR = fftshift(real(ifft2(Sf)));
>> fr3 = deconvwnr(g, PSF, NCORR, ICORR);

```

Из рисунка 8г видно, что результат этих операций весьма близок к оригиналу, хотя небольшой шум все еще проявляется. Поскольку нам известно само изображение и наложенный на него шум, мы можем оценить корректирующие параметры и сделать заключение, что рисунок 8г дает наилучший результат, который можно достигнуть винеровской деконволюцией. На практике, когда одна (или более) из этих величин неизвестна, приходится менять, функции для экспериментирования до тех пор, пока не будет достигнут удовлетворительный результат.

Сглаживающая фильтрация методом наименьших квадратов со связью

Другой хорошо зарекомендовавший себя метод линейного восстановления заключается в *фильтрации по методу наименьших квадратов с ограничениями*, который в документации IPT называется *сглаживающей фильтрацией*. Двумерная дискретная свертка задается выражением

$$h(x, y)^* f(x, y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x - m, y - n).$$

С помощью этой формулы можно выразить модель линейного искажения $g(x, y) = h(x, y)^* f(x, y) + \eta(x, y)$, которая обсуждалась в § 5.1, в векторно-матричной форме

$$\mathbf{g} = \mathbf{H}\mathbf{f} + \boldsymbol{\eta}.$$

Пусть, к примеру, $g(x, y)$ имеет размеры $M \times N$. Тогда можно сформировать первые N элементов вектора \mathbf{g} из элементов изображения, которые располагаются в первой строке $g(x, y)$, следующие N элементов взять из второй строки $g(x, y)$ и т.д. В результате получится вектор размера $MN \times 1$. Такую же размерность имеют векторы \mathbf{f} и $\boldsymbol{\eta}$, которые строятся по аналогичной схеме. В этом случае матрица \mathbf{H} имеет размеры $MN \times MN$. Ее элементы берутся из предыдущего уравнения свертки.

Логично заключить, что задача восстановления изображения может быть переформулирована в виде простых матричных действий. Однако в данном случае это не так. Предположим, что мы имеем дело с изображением средних размеров, например, у которого $M = N = 512$. Тогда векторы в описанных выше представлениях будут иметь размерность 262144×1 , а у матрицы \mathbf{H} будут размеры 262144×262144 . Обрабатывать векторы и матрицы таких размеров весьма непросто. Проблема еще более усложняется тем обстоятельством, что для \mathbf{H} может не существовать обратной матрицы в силу наличия нулей у передаточной функции. Таким образом, формулировка задачи восстановления в матричной форме не позволяет обличить технику восстановления изображений.

Несмотря на то, что мы здесь не собираемся обосновывать метод

наименьших квадратов со связями, стоит отметить, что центральным местом этого метода является чувствительность к обращению матрицы H , о чём говорилось в предыдущем абзаце. Одним из возможных путей преодоления этой трудности является оптимизация процедуры восстановления по сглаживающей мере, например, но вторым производным изображения (и, в частности, по лапласиану). Такая процедура будет иметь смысл, если ограничения задачи являются доступными параметрами. Таким образом, требуется найти минимум целевой функции C , которая определяется по формуле

$$C = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\nabla^2 f(x, y)]^2$$

при условии выполнения ограничения (связи)

$$\|g - H\hat{f}\|^2 = \|\eta\|^2,$$

где $\|w\|^2 \stackrel{\Delta}{=} w^T w$ — это евклидова норма вектора, \hat{f} — это приближение испорченного изображения.

Решение этой задачи по оптимизации, записанное в частотной области, задается выражением

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2 + \gamma |P(u, v)|^2} \right] G(u, v),$$

где γ — это некоторый параметр, который следует подобрать так, чтобы выполнялось уравнение связи (при $\gamma = 0$ получаем решение для инверсного фильтра), а $P(u, v)$ — преобразование Фурье функции

$$P(x, y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

В этой функции мы узнаем оператор Лапласа. В предыдущих формулах остались неизвестными две величины, γ и $\|\eta\|^2$. Однако γ можно вычислить интерактивно, если известна скалярная величина $\|\eta\|^2$, которая пропорциональна энергии шума.

Фильтрации методом наименьших квадратов со связями реализована в IPT в виде функции deconvreg, которая имеет синтаксис

$$\text{fr} = \text{deconvreg}(\text{g}, \text{PSF}, \text{NOISEPOWER}, \text{RANGE}),$$

где g — это искаженное изображение, fr — восстановленное изображение, NOISEPOWER пропорционально $\|\eta\|^2$, а RANGE — это диапазон для нахождения решения γ . По умолчанию это интервал $[10^{-9}, 10^9]$ (или в обозначениях MATLAB, $[1e-10, 1e10]$). Если два последних параметра в deconvreg отсутствуют, то эта функция совершает обычную инверсную

фильтрацию. Хорошим начальным приближением для NOISEPOWER может являться число $MN[\sigma_y^2 + \sigma_f^2]$, где M и N - размеры изображения, числа в квадратных скобках равны дисперсии и квадрату среднеквадратичного значения шума. Эта величина является лишь первым приближением, в то время как окончательное оптимальное значение может быть совсем иным, что видно в следующем примере.

Пример 9. Использование функции *deconvreg* при восстановлении смазанного зашумленного изображения.

Мы теперь восстановим изображение на рисунке 7 γ с помощью функции *deconvreg*. Это изображение имеет размеры 64x64, а из примера 7 нам известно, что его шум имеет нулевое среднее и дисперсию, равную 0.001. Следовательно, наше начальное приближенное значение для NOISEPOWER равно $(64)^2 \times [0.001 + 0] \sim 4$. На рис. 5.9, *a*) дан результат выполнения команды

```
>> fr = deconvreg(g, PSF, 4);
```

где g и PSF взяты из примера 7. Это изображение немного лучше исходного, однако ясно, что выбранное значение NOISEPOWER не годится с практической точки зрения. Если немного поэкспериментировать с этим параметром, а также с параметром RANGE, то можно найти оптимальный результат, показанный на рисунке 9 δ , который был построен командой

```
>> fr = deconvreg(g, PSF, 0.4, [1e-7 1e7]);
```

Итак, следует уменьшить величину NOISEPOWER на один порядок, а интервал диапазона выбрать покомпактнее. Результат винеровской фильтрации того же изображения приведен на рисунке 8 γ , однако он был получен при наличии полной информации о спектрах шума и исходного изображения. Без этих ключевых сведений результаты, достигаемые при экспериментировании с этими фильтрами, часто бывают вполне сравнимы друг с другом.

Если в восстановленном изображении наблюдаются «звоны», наведенные дискретным преобразованием Фурье, то справиться с ними позволяет функция *edgetaper*, которую следует применить до вызова *deconvreg*.

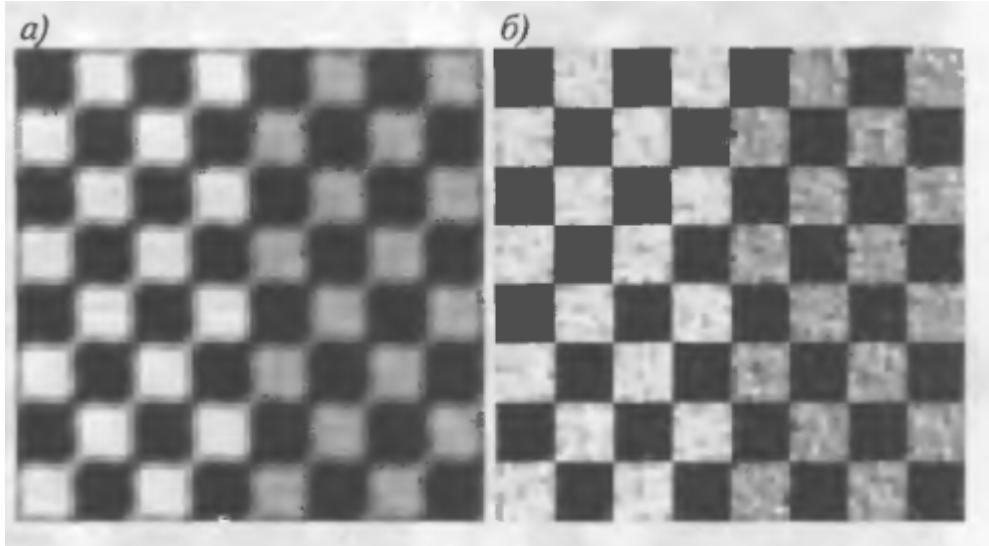


Рисунок 9 – а) Изображение из рисунка 7г восстановленное сглаживающим фильтром с pari-метром NOISEPOWER = 4 б) То же восстановленное изображение при NOISEPOWER = 4.0 и с диапазоном RANGE = [1e-7 1e7]

Алгоритм люси-ричардсона итерационного нелинейного восстановления

Методы восстановления изображений, которые обсуждались в трех предыдущих параграфах, являются линейными. Кроме того, они являются «прямыми» в том смысле, что, как только соответствующий фильтр построен, решение задачи восстановления находится однократным применением этого фильтра. Эта простота реализации вкупе со скромными требованиями к вычислительным ресурсам, а также хорошо развитая теоретическая база, сделали линейные методы основными инструментами восстановления изображений на многие годы.

В последние два десятилетия нелинейные итерационные технологии стали завоевывать популярность в области восстановления изображений, приводя часто к более лучшим результатам по сравнению с традиционными линейными методами. Основными недостатками нелинейных методов являются недостаточная предсказуемость их поведения, а также использование значительных вычислительных ресурсов. Первый недостаток часто теряет свою актуальность, поскольку показано, что нелинейные методы превосходят линейные по широкому спектру приложений. Второй недостаток также постепенно преодолевается ввиду впечатляющего роста производительности вычислительной техники за последние 10 лет. Алгоритм L-R формулируется в терминах метода максимального правдоподобия, в котором изображение моделируется в виде статистик Пуассона. Максимизация функции правдоподобия модели приводит к уравнению, которое выполняется при условии следующих итераций

$$\hat{f}_{k+1}(x, y) = \hat{f}_k(x, y) \left[h(-x, -y) * \frac{g(x, y)}{h(x, y) * \hat{f}_k(x, y)} \right],$$

где «*» обозначает свертку, \hat{f} это приближение неиспорченного

изображения. Алгоритм, очевидно, является итерационным, а его нелинейность происходит из-за деления на \hat{f} в правой части уравнения.

Как и в любом нелинейном методе, здесь трудно в общем виде ответить на вопрос: когда следует остановить алгоритм L-R? В конкретных приложениях этот подход часто сопровождается выводом на экран промежуточных результатов, и алгоритм останавливается при достижении приемлемых изображений.

Алгоритм L-R реализован в IPT функцией `deconvlucy`, которая имеет следующий синтаксис:

```
fr = deconvlucy(g, PSF, NUMIT, DAMPAR, WEIGHT),
```

где fr -это восстановленное изображение, g -искаженное изображение, PSF - функция разброса точек, $NUMIT$ - число итераций (по умолчанию 10), а величины $DAMPAR$ и $WEIGHT$ определяются следующим образом.

$DAMPAR$ — это скаляр, который определяет порог отклонения полученного изображения от g . Итерации останавливаются для пикселей, отклонение значений которых от исходных не превосходит этого порога. Это предотвращает образование шума в таких пикселях, сохраняя необходимые детали изображения. По умолчанию $DAMPAR = 0$ (нет порога остановки).

$WEIGHT$ — это массив размера, как у g , который каждому пикселью присваивает некоторый вес, отражающий его качество. Например, «плохие» пиксели, которые получились из дефектных областей на изображения, можно исключить из рассмотрения, присвоив им нулевой вес. Другое полезное применение этого массива состоит в подгонке весов пикселей для коррекции однородных областей при наличии дополнительной информации. При моделировании смазывания конкретным PSF (см. пример 7) параметр $WEIGHT$ можно использовать для исключения из вычислений пикселей, которые расположены на границах изображения и которые размываются функцией PSF . Если PSF имеет размеры $n \times n$, то в $WEIGHT$ используется обрамление из нулей ширины $\text{ceil}(n/2)$. По умолчанию $WEIGHT$ состоит из одних единиц и имеет размеры, как у изображения g .

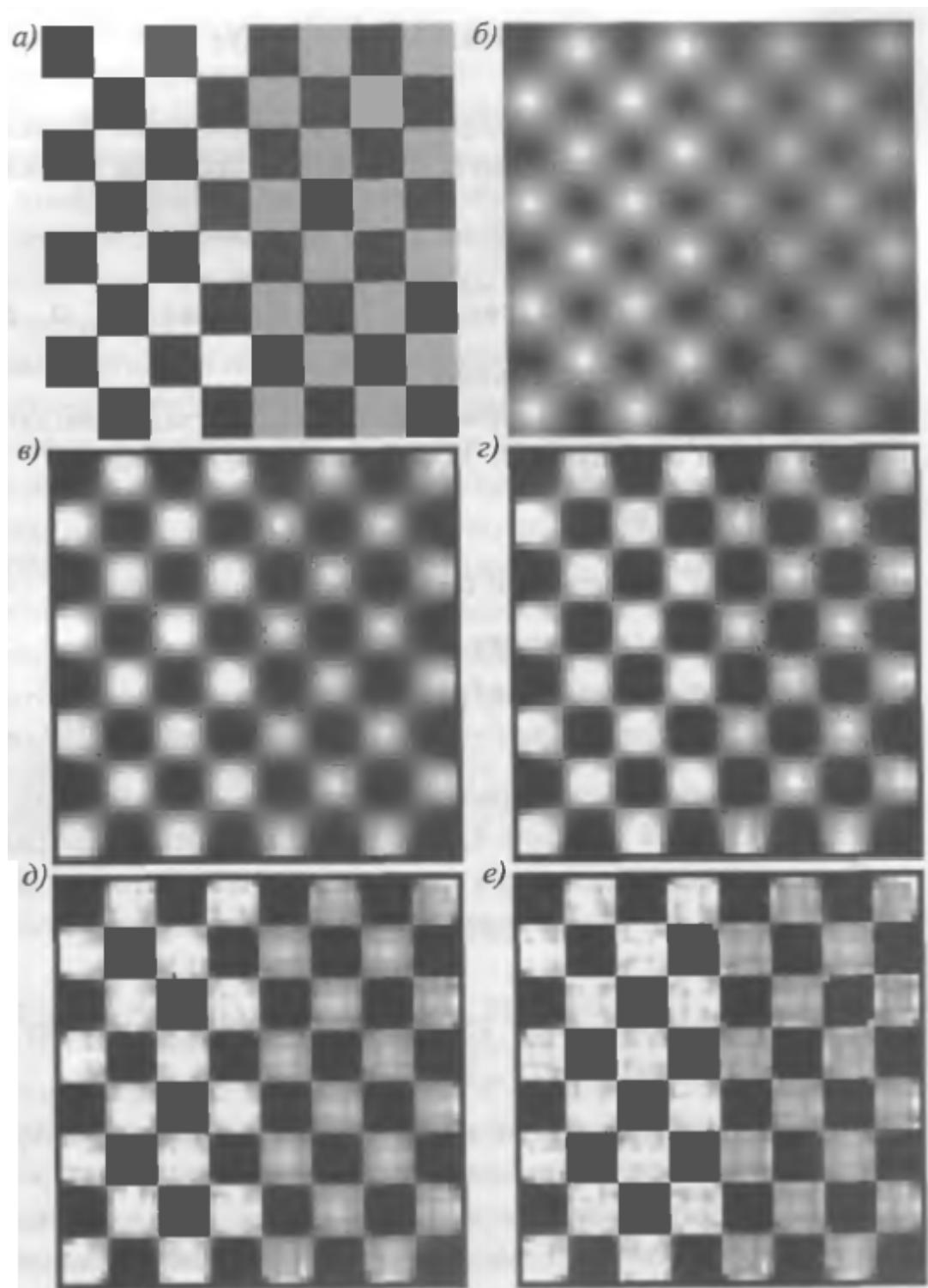


Рисунок 10 – а) Исходное изображение. б) Изображение порченое гауссовым шумом. в)-е) Изображения ритму L-R, соответственно, после 5, 10, 20 и 100 итераций

Если в восстановленном изображении наблюдаются «звоны», производимые дискретным преобразованием Фурье, то подавить их можно функцией `edgetaper`, которую надо применять до вызова `deconvlucy`.

Пример 10. Использование функции `deconvlucy` при восстановлении смазанного и зашумленного изображения.

На рисунке 10 a показано изображение, построенное командой

```
>> f = checkerboard(8);
```

и имеющее размеры 64x64 пикселей. Как и раньше, его размеры были увеличены до 512x512 функцией `pixeldup` для лучшего визуального восприятия:

```
>> imshow(pixeldup(f, 8));
```

Следующая команда создала гауссову PSF размера 7x7 со стандартным отклонением 10:

```
>> PSF = fspecial('gaussian', 7, 10);
```

Затем мы размыли изображение f с помощью функции PSF и прибавили к нему случайный гауссов шум с нулевым средним и стандартным отклонением 0.01:

```
>> SD = 0.01;
>> g = imnoise(imfilter(f, PSF), 'gaussian', 0, SD^2);
```

На рисунке 10 b дан результат.

Остальная часть примера относится к восстановлению изображения g с помощью функции `deconvlucy`. Зададим параметр DAMPAR:

```
>> DAMPAR = 10*SD;
```

Массив WEIGHT строится по описанной выше схеме:

```
>> LIM = ceil(size(PSF), 1)/2;
>> WEIGHT = zeros(size(g));
>> WEIGHT(LIM + 1:end - LIM, LIM + 1:end - LIM) = 1;
```

Размеры массива WEIGHT равны 64x64. На сто границах располагается бордюр нулевых пикселей ширины 4, а все остальные пиксели равны 1.

Остался неопределенным параметр NUMIT, который обозначает число итераций. На рисунке 10 c приведен результат выполнения команд

```
>> NUMIT = 5;
>> fr = deconvlucy(g, PSF, NUMIT, DAMPAR, WEIGHT);
>> imshow(pixeldup(fr, 8))
```

Изображение немного улучшилось, но осталось достаточно размытым. Рисунок 10 d и 10 d представляют результаты, полученные при $NUMIT = 10$ и 20. Последний из них можно смело считать восстановлением исходного смазанного и зашумленного изображения. На самом деле, дальнейшее

увеличение числа итераций не приводит к значительным улучшениям. Например, рисунок 10e получен после 100 итераций. Это изображение лишь чуть-чуть точнее и ярче результата после 20 итераций. Тонкий черный бордюр на всех восстановленных изображениях получился потому, что в массиве WEIGHT на соответствующих мостах стоят нули.

Слепая деконволюция

Одна из самых сложных проблем, возникающая при восстановлении изображений, состоит в получении подходящих приближений функции PSF для использования в алгоритмах восстановления, которые обсуждались в предыдущих параграфах. Как уже отмечалось ранее, методы восстановления изображений, в которых не используется информация, характеризующая функцию PSF, называются *алгоритмами слепой деконволюции*.

Метод слепой деконволюции, к которому было обращено внимание исследователей последние двадцать лет, основан на приближении по максимуму правдоподобия (MLE, Maximum-Likelihood Estimation) – стратегии оптимизации при построении приближений величин, искаженных случайнym шумом. Вкратце можно сказать, что в интерпретации MLE изображение считается случайно выбранным с некоторой определенной вероятностью из семейства других возможных случайных величин. Функция правдоподобия выражается через функции $g(x,y)$, $f(x,y)$ и $h(x,y)$, и задача заключается в нахождении максимума функции правдоподобия. При слепой деконволюции задача оптимизации решается итеративно при выполнении соответствующих ограничений и при условии сходимости всей процедуры. Максимизирующая пара функций $f(x, y)$ и $h(x, y)$ считается восстановленным изображением и соответствующей функцией PSF.

В пакете слепая деконволюция представлена функцией `deconvblind`, которая имеет синтаксис

```
[fr, PSFe] = deconvblind(g, INITPSF),
```

где g — искаженное изображение, INITPSF — первое приближение функции PSF, PSFe — конечное вычисленное приближение этой функции, а fr — восстановленное изображение на основе найденной функции PSF. Функция использует при вычислениях итеративный алгоритм L-R, описанный выше. Приближение PSF сильно зависит от размера начальной оценки и, в меньшей степени, от ее значений (массив из одних единиц является разумной начальной оценкой).

В этой форме функции число итераций равно 10 (принято по молчанию). Дополнительные параметры, которые можно включить в аргументы функции `deconvblind` контролируют число итераций и другие особенности процесса восстановления:

```
[fr, PSFe] = deconvblind(g, INITPSF, NUMIT, DAMPAR, WEIGHT).
```

Параметры NUMIT, DAMPAR и WEIGHT объяснялись при описании алгоритма L-R. в предыдущем параграфе.

Если в восстановленном изображении имеются «звоны» от дискретного преобразования Фурье, то подавить их поможет функция edgetaper, которую надо вызвать до применения deconvblind.

Пример 11. Использование функции *deconvblind* при построении приближения PSF.

На рисунке 11а изображена функция PSF, которая использовалась при построении искаженного изображения на рисунке 10б:

```
>> PSF = fspecial('gaussian', 7, 10);
>> imshow(pixelup(PSF, 73), [ ]);
```

Как и в примере 10, искаженное изображение, которое будет изучаться, получено командами

```
>> SD = 0.01;
>> g = imnoise(imfilter(f, PSF), 'gaussian', 0, SD^2);
```

В данном примере нас интересует использование функции deconvblind для построения приближения PSF, если известно только искаженное изображение g. На рисунке 11б приведен результат следующих команд:

```
>> INITPSF = ones(size(PSF));
>> NUMIT = 5;
>> [fr, PSFe] = deconvblind(g, INITPSF, NUMIT, DAMPAR,
WEIGHT);
>> imshow(pixelup(PSFe, 73), [ ]);
```

где значения DAMPAR и WEIGHT были взяты из примера 10.

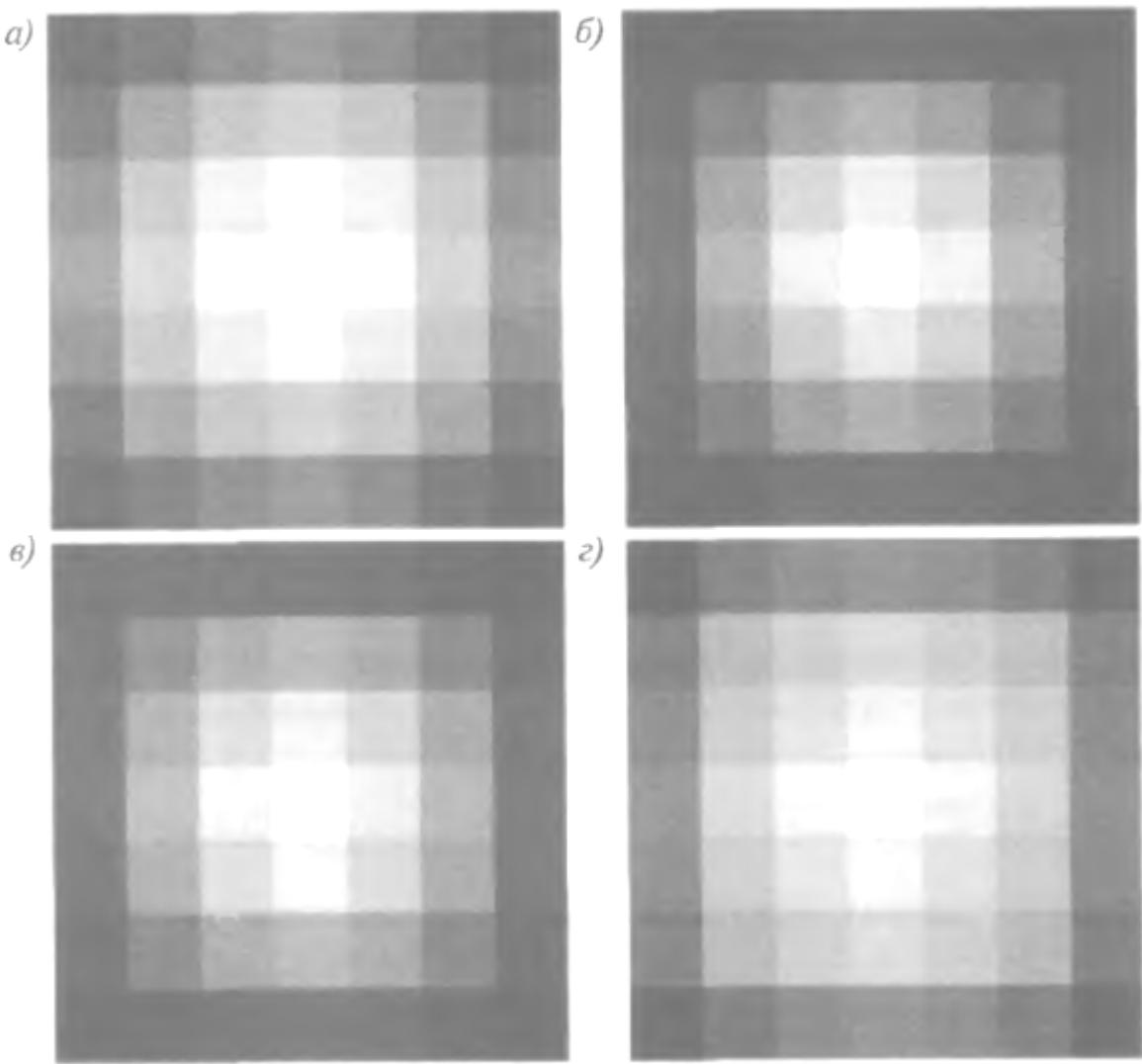


Рисунок 11 – а) Исходная функция PSF. б)-г) Приближения PSF, соответственно, после 5, 10 и 20 итераций функции deconvblind

Рисунки 11 ν и 11 ζ показывают функции PSF e , построенные теми же командами, после выполнения, соответственно, 10 и 20 итераций. Последний результат близок к исходной PSF из рисунка 11а.

Геометрические преобразования и регистрация изображений

В завершении этой главы мы рассмотрим геометрические преобразования, которые используются при восстановлении изображений. Геометрические преобразования изменяют пространственные соотношения между пикселями изображения. Их иногда называют *преобразованиями куска резины*, поскольку их можно себе представить, если нанести изображение на кусок резины, а затем растягивать его в соответствии с предписанными правилами.

Геометрические преобразования часто используются при *регистрации изображений*. В этом процессе участвуют два изображения на одной сцене, которые необходимо наложить друг на друга с целью визуализации или

количественного сравнения. В следующих подпараграфах обсуждаются:

(1) пространственные преобразования и способы их визуализации в MATLAB;

(2) применение пространственных преобразований к конкретным изображениям;

(3) задание пространственных преобразований для регистрации изображений.

Пространственные преобразования

Пусть изображение f заданное в координатной системе (w, z) , подвергается геометрической деформации, в результате которой получается изображение g в координатной системе (x, y) . Это (координатное) преобразование можно в виде формулы

$$(x, y) = T\{(w, z)\}.$$

Например, если $(x, y) = T\{(w, z)\} = (w/2, z/2)$, то «искажение», сводится к двойному сжатию по обеим пространственным измерениям, как показано на рисунке 12.

Чаще всего используется класс геометрических преобразований, представители которого называются *аффинными преобразованиями*. Аффинное преобразование можно записать в матричной форме

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} w & z & 1 \end{bmatrix} T = \begin{bmatrix} w & z & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}.$$

Такой формулой можно задать сжатие, поворот, перенос или сдвиг, соответствующим образом определяя элементы матрицы T . В табл. 5.3 показано, как выбирать эти величины для совершения различных преобразований.

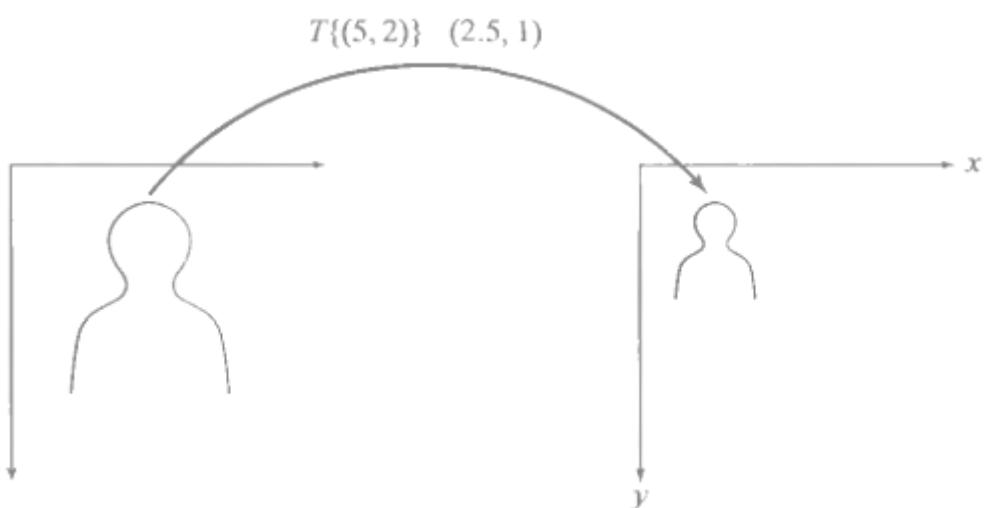


Рисунок 12 – Простое пространственное преобразование.

Таблица 3. Типы аффинных преобразований

Тип	Аффинная матрица Т	Координатное уравнение	Диаграмма
Тождество	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = z$	
Растяжение	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = s_x w$ $y = s_y z$	
Поворот	$\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w \cos \theta - z \sin \theta$ $y = w \sin \theta + z \cos \theta$	
Сдвиг (горизонтальный)	$\begin{bmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w + \alpha z$ $y = z$	
Сдвиг (вертикальный)	$\begin{bmatrix} 1 & \beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = \beta w + z$	
Перенос	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix}$	$x = w + \delta_x$ $y = z + \delta_y$	

В пакете ITP пространственное преобразование задается в виде т.н. tform – структуры. Один путь определения структуры состоит в использовании функции maketform, вызов которой имеет вид

```
tform = maketform(transform_type, transform_parameters).
```

Первый входной аргумент transform_type может принимать значения 'affine', 'projective', 'box', 'composite', или 'custom'. Эти типы преобразований объяснены в таблице 4. Остальные аргументы зависят от выбранного типа преобразования, и они разъясняются на справочной странице функции maketform. Рассмотрим некоторые аффинные преобразования. Например, можно определить аффинную tform-структуру, прямо описав ее матрицу Т

следующим способом:

```
>> T = [2 0 0; 0 3 0; 0 0 1] ;
>> tform = maketform('affine', T)

tform =
ndims_in : 2
ndims_out : 2
forward.fcn : @fwd_affine
inverse_fcn : @inv_affine
tdata : [1 x 1 struct]
```

Хотя пользователю не потребуется напрямую обращаться к полям структуры `tform`, сообщим, что информация о матрице T и о ее обратной T^{-1} хранится в поле `tdata`:

```
>> tform.tdata.
ans =
T:3 x 3 double
Tinv:3 x 3 double

>> tform.tdata.T
ans =
200
030
001
>> tform.tdata.Tinv
ans =
0.5000 0 0
00.3333 0
0 01.0000
```

В пакете IPT имеются две функции, с помощью которых пространственное преобразование применяется к точкам: `tformfwd` вычисляет прямое преобразование $T\{(w, z)\}$, а функция `tforminv` вычисляет обратное преобразование $T^{-1}\{(x,y)\}$. Форма вызова `tforminv` имеет вид $XY = tformfwd(WZ, tform)$. Здесь WZ - матрица $P \times 2$, в каждой строке которой стоят пары w и z координат одной точки из P . Аналогично, XY - это матрица, столбцы которой содержат x -и y -координаты P преобразованных точек. Например, следующие команды вычисляют прямое преобразование пары точек, после чего вычисляется обратное преобразование для проверки, получаются ли исходные данные:

```
>> WZ = [1 1; 3 2];
>> XY = tformfwd(WZ, tform)
XY
2 3
6 6

>> WZ2 = tforminv(XY, tform)
```

```
wz2
1 1
3 2
```

Чтобы лучше почувствовать то или иное пространственное преобразование, полезно посмотреть на то, как оно действует на изображение, содержащее координатную сетку. Следующая М-функция **vistformfwd** строит координатную сетку, преобразует ее с помощью **tformfwd**, а затем размещает рядом графики этой сетки и ее преобразования для удобства сравнения. Обратите внимание на совместное употребление функций **meshgrid** и **linspace** при построении сетки. Следующий программный код также иллюстрирует использование некоторых других функций, описанных ранее.

```
function vistformfwd(tform, wdata, zdata, N)
if nargin < 4
    N = 10;
end
[w, z] = meshgrid(linspace(wdata(1), zdata(2), N), ...
    linspace(wdata(1), zdata(2), N));
wz = [w(:) z(:)];
xy = tformfwd([w(:) z(:)], tform);
x = reshape(xy(:, 1), size(w));
y = reshape(xy(:, 2), size(z));
wx = [w(:); x(:)];
wxlimits = [min(wx) max(wx)];
zy = [z(:); y(:)];
zylimits = [min(zy) max(zy)];
subplot(1,2,1)
plot(w, z, 'b'), axis equal, axis ij
hold on
plot(w', z', 'b')
hold off
xlim(wxlimits)
ylim(zylimits)
set(gca, 'XAxisLocation', 'top')
xlabel('w'), ylabel('z')
subplot(1, 2, 2)
plot(x, y, 'b'), axis equal, axis ij
hold on
plot(x', y', 'b')
hold off
xlim(wxlimits)
ylim(zylimits)
set(gca, 'XAxisLocation', 'top')
xlabel('x'), ylabel('y')
```

Пример 12. Визуализация некоторых аффинных преобразований функцией **vistformfwd**.

В этом примере мы проиллюстрируем действие некоторых аффинных преобразований с помощью функции `vistformfwd`. Кроме того, для создания аффинной `tform` мы воспользуемся еще одним подходом использования функции `maketform`. Начнем с аффинного преобразования, которое растягивает изображение в 3 раза по горизонтали и в 2 раза по вертикали:

```
>> T1 = [3 0 0; 0 2 0; 0 0 1];
>> tform1 = maketform('affine', T1);
>> vistformfwd(tform1, [0 100], [0 100]);
```

На рисунках 13 a и 13 b даны результаты.

Эффекты сдвига наблюдаются, если элементы t_{21} и t_{12} в аффинной матрице Т ненулевыми:

```
>> T2= [1 0 0; .2 1 0; 0 0 1];
>> tform2 = maketform('affine', T2);
>> vistformfwd(tform2, [0 100], [0 100]);
```

На рисунках 13 c и 13 g показаны эффекты сдвига применительно к координатной сетке.

Важное свойство аффинных преобразований состоит в том, что любая их комбинация снова является аффинным преобразованием. Математически матрица Т этого аффинного преобразования получается перемножением соответствующих аффинных матриц. Следующий блок команд генерирует и отображает аффинное преобразование, которое получается комбинацией растяжения, поворота, и сдвига.

```
>> Tscale = [1.5 0 0; 0 2 0; 0 0 1];
>> Trotation - [cos(pi/4) sin(pi/4) 0;
- sin(pi/4) cos(pi/4) 0; 0 0 1];
>> Tshear = [1 0 0; .2 1 0; 0 0 1];
>> T3 = Tscale*Trotation*Tshear;
>> tform3 = maketform('affine', T3) ;
>> vistformfwd(tform3, [0 100], [0 100]);
```

На рисунках 13 d и 13 e приведен результат.

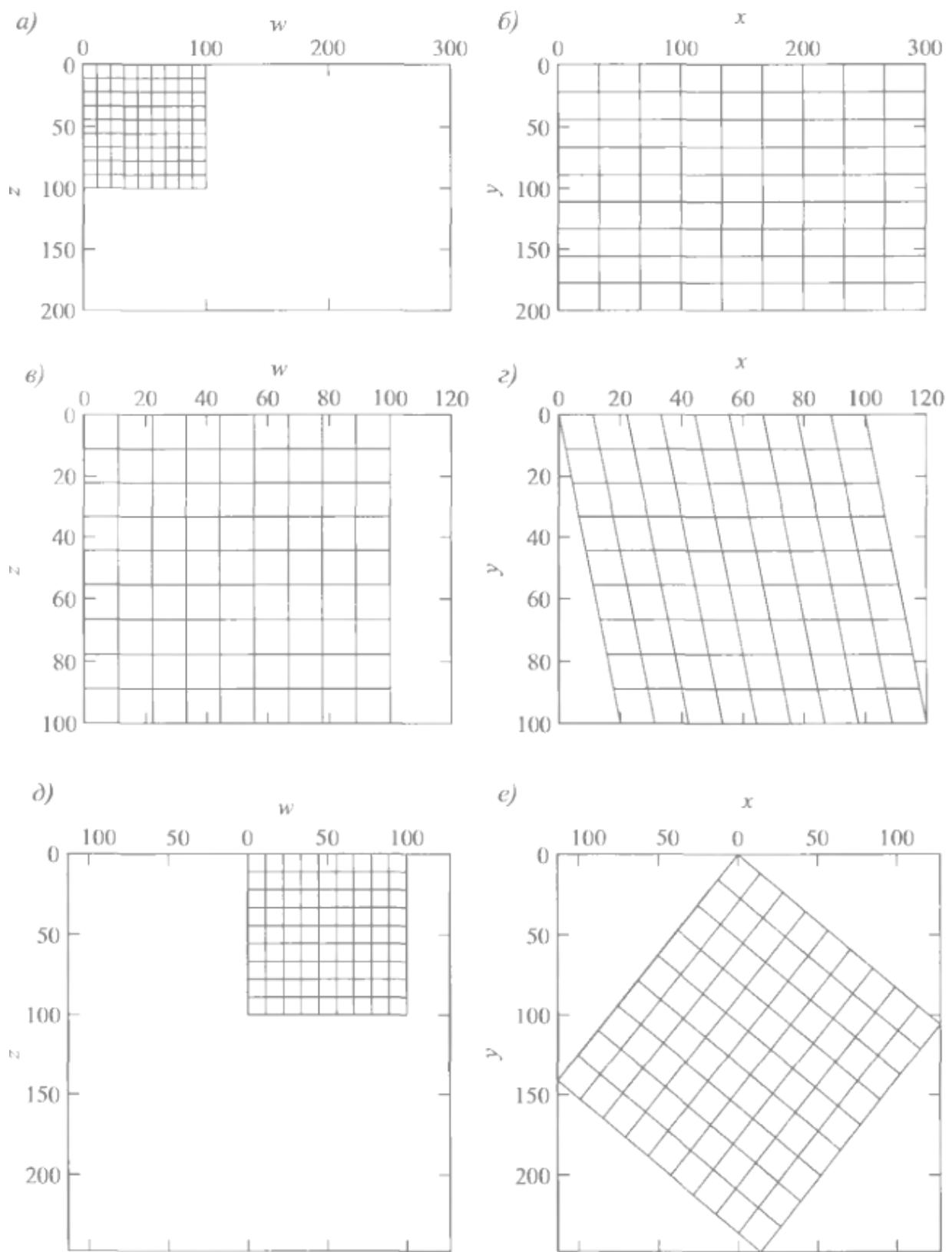


Рисунок 13 – Визуализация аффинных изображений с помощью сетки. *а)* Сетка
б) Сетка 1, преобразованная с помощью tform1. *в)* Сетка 2. *г)* Сетка 2,
 преобразованная с помощью tfonn2. *д)* Сетка 3. *е)* Сетка 3, преобразованная с
 помощью tform3

Применение пространственных преобразований к изображениям

Большинство вычислительных методов пространственного

преобразования изображений можно разделить на две категории: методы, использующие *прямое отображение*, и методы, основанные на *обратном отображении*. Методы прямого отображения берут каждый пиксель и копируют его в место на выходном изображении, координаты которого вычисляются по формуле $T\{(w, z)\}$. При этом возникает проблема: что делать с двумя или большим числом входных пикселей, которые преобразуются в один и тот же пиксель, т. е. как скомбинировать кратные величины входных пикселей для получения одного-единственного выходного пикселя. Другая потенциальная сложность состоит в том, что на выходном изображении могут оказаться точки, на которые не отобразился ни один входной пиксель. В более изощренной форме прямого отображении четыре вершины входного пикселя отображаются в вершины неправильного четырехугольника на выходном изображении. Входные пиксели распределяются среди выходных пикселей в соответствии с тем, сколько раз накрывается каждый пиксель относительно площади каждого выходного пикселя. Такая форма прямого отображения является более точной, однако она будет существенно сложнее и потребует больших вычислительных средств при реализации.

Функция IPT **imtransform**, напротив, использует обратное отображение. Процедура обратного отображения берет поочередно пиксели выходного изображения и вычисляет координаты соответствующих точек прообразов входного изображения по формуле $T^{-1}\{(x, y)\}$, а затем делает интерполяцию по ближайшим пикселям входного изображения. Обратное преобразование бывает проще реализовать, чем прямое:

Основная форма вызова функции **imtransform** имеет вид

```
g = imtransform(f, tform, interp),
```

где **interp** — эти строка символов, которая определяет метод интерполяции ближайших пикселей для вычисления значения выходного пикселя. Переменная **interp** имеет три возможных значения: '**nearest**', '**bilinear**' и '**bicubic**'. Если **interp** опущена, то по умолчанию используется '**bilinear**'. Как и в предыдущих примерах, в качестве тестового изображения при экспериментировании с пространственными преобразованиями будет использоваться шахматная доска доска.

Пример 13. Пространственное преобразование, изображений.

Мы воспользуемся функциями **checkerboard** и **imtransf** для исследования некоторых аспектов преобразования изображений. *Линейное конформное преобразование* является частным случаем аффинного преобразования, при котором сохраняются формы и углы. Линейное конформное преобразование задается коэффициентом растяжения/сжатия, углом поворота и вектором переноса. В этом случае аффинная матрица имеет вид

$$T = \begin{bmatrix} s \cos \theta & s \sin \theta & 0 \\ -s \sin \theta & s \cos \theta & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix},$$

Следующая последовательность команд строит линейное конформное преобразование и применяет его к тестовому изображению:

```
>> f = checkerboard(50);
>> s = 0.8;
>> theta = pi/6;
>> T = [s*cos(theta) s*sin(theta) 0
- s*sin(theta) s*cos(theta) 0
0 0 1];
>> tform = maketform('affine', T);
>> g = imtransform(f, tform);
```

На рисунках 14а и 14б показаны исходное и преобразованное изображения шахматной доски. Завершающий вызов функции **imtransform** использовал по умолчанию метод интерполяции '**bilinear**'. Как уже отмечалось, имеется возможность выбрать другой метод интерполяции, например, по ближайшему соседу, который задается явно при вызове функции **imtransform**:

```
>> g2 = imtransform(f, tform, 'nearest');
```

На рисунках 14в показан результат этой команды. Интерполяция по ближайшему соседу выполняется быстрее, чем билинейная интерполяция, поэтому она может оказаться более предпочтительной в определенных ситуациях, однако результат бывает хуже по качеству, чем при билинейной интерполяции.

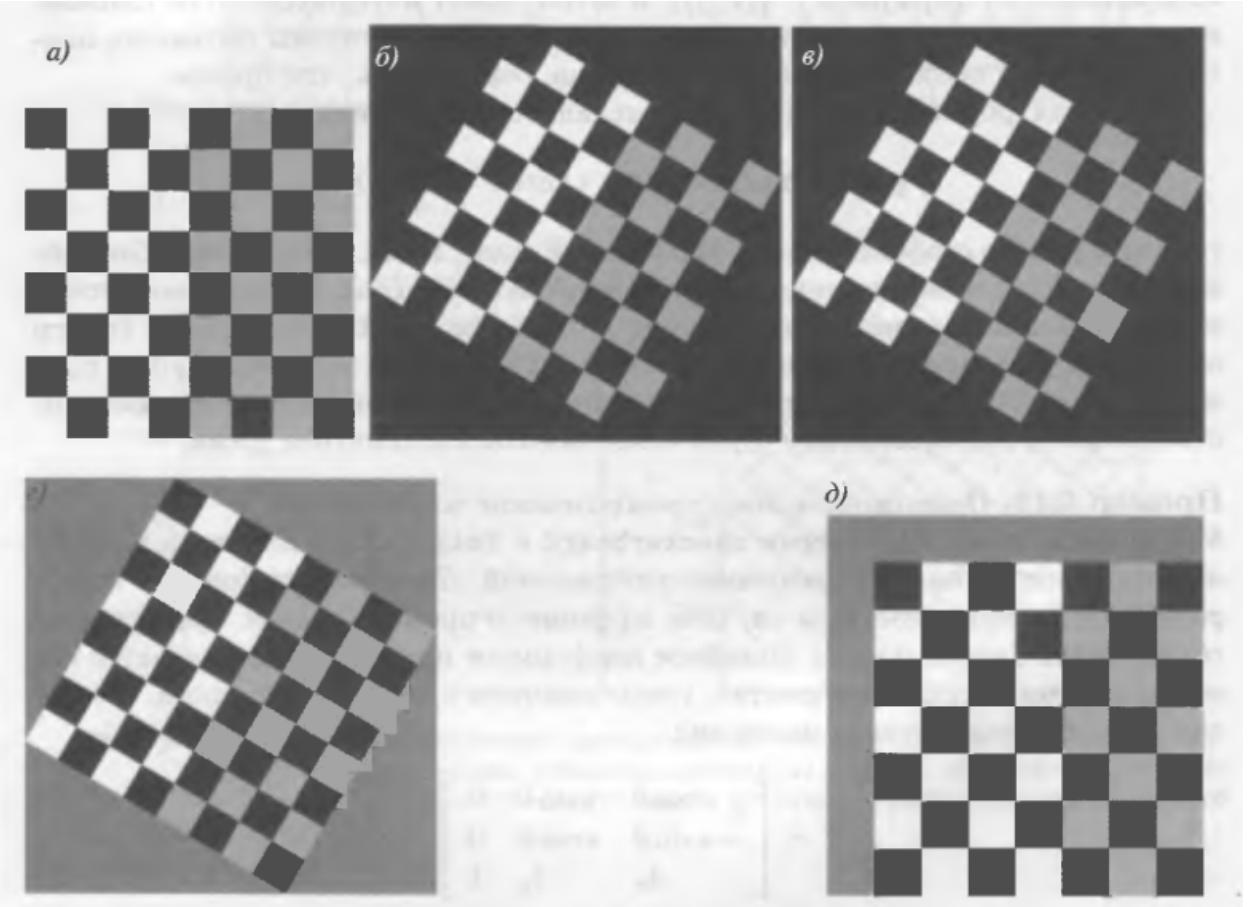


Рисунок 14 – Аффинные преобразования шахматной доски, а) Исходное изображение, б) Линейное конформное иреобразование с использованием (по умолчанию) билинейной интерполяции. в) Интерполяция по ближайшему соседу. г) Задание альтернативного цвета для внешнего заполнения, д) Контролирование выходного пространства для сохранения переносов

Функция **imtransform** имеет несколько опционных параметров, которые могут быть полезны. Например, параметр **FillValue** контролирует цвет, который функции использует для пикселей, которые находятся вне входного изображения:

```
>> g3 = imtransform(f, tform, 'FillValue', 0.5);
```

Из рисунка 14в видно, что внешняя часть изображения окрашена теперь в серый цвет, а не в черный.

Другие дополнительные параметры помогают разрешить иной источник недоразумений, связанных с переносом изображений с помощью **imtransform**. Например, следующие команды совершают обычный параллельный перенос:

```
>> T2 = [1 0 0; 0 1 0; 50 50 1] ;
>> tform2 = maketform('affine', T2);
>> g4 = imtransform(f, tform2) ;
```

Однако результат будет идентичен исходному изображению на рисунке 14а. В этом проявляется эффект функции **imtransform**, заложенный по умолчанию. А именно: функция **imtransform** определяет ограничивающий прямоугольник выходного изображения в выходной системе координат, и по умолчанию обратное преобразование совершаются только для пикселей этого прямоугольника. При этом отменяются переносы. Задав параметры XData и YData, можно точно указать функции **imtransform**, где именно в выходном пространстве следует вычислять результат. XData это вектор из двух компонент, который обозначает координаты верхнего левого угла выходного изображения, а YData содержит координаты нижнею правого угла. Следующая команда совершает вычисления в выходной области, расположенной между точками $(x,y) = (1,1)$ и $(x,y) = (400,400)$.

```
>> g5 = imtransform(f, tform2, 'XData', [1 400], 'YData', [1  
400], 'FillValue', 0.5);
```

На рисунке 14д приведен результат.

Другие настройки функции **imtransform** и связанные с ними функции IPT обеспечивают дополнительные возможности воздействия на результат пространственного преобразования, в частности, на совершение интерполяции. Относящуюся к этому вопросу информацию можно взять со справочной страницы функций **imtransform** и **makeresampler**.

Регистрация изображений

Методы регистрации изображений делают пригонку разных изображений одной и той же сцены. Например, эти изображения могли быть получены примерно в одно и то же время, но разными регистрирующими устройствами, например, сканированием магнитно-резонансным методом и одновременно позитронно-эмиссионной томографией. Или, возможно, изображения были сняты одним и тем же устройством, но в разные моменты времени. Например, фотоснимки со спутника, сфотографированные через несколько дней, месяцев или лет. В любом случае, комбинирование изображений, количественный анализ и сравнение требует компенсации геометрических aberrаций, происходящих от разных углов наклона фотокамеры, разницы расстояния и особенностей ориентации объектов съемки, а также из-за различия разрешения регистрирующих устройств, перемещения объектов и влияния других факторов.

В пакете поддерживается метод регистрации изображений на основе контрольных точек, иногда называемых *точками привязки*. Контрольные точки образуют подмножество пикселей, координаты которых на обоих изображениях известны или могут быть выбраны интерактивно. На рисунке 15 проиллюстрирована идея контрольных точек на тестовом изображении и но измененном тестовом изображении, подвергнутом проектному искажению. Если выбрано достаточное количество контрольных точек, функция **cp2tform** из IPT подбирает подходящее преобразование

предписанного типа по этим контрольным точкам (при этом используется метод наименьших квадратов). Типы пространственных преобразований, которые распознаются функцией `cp2tform`, перечислены в таблице 4.

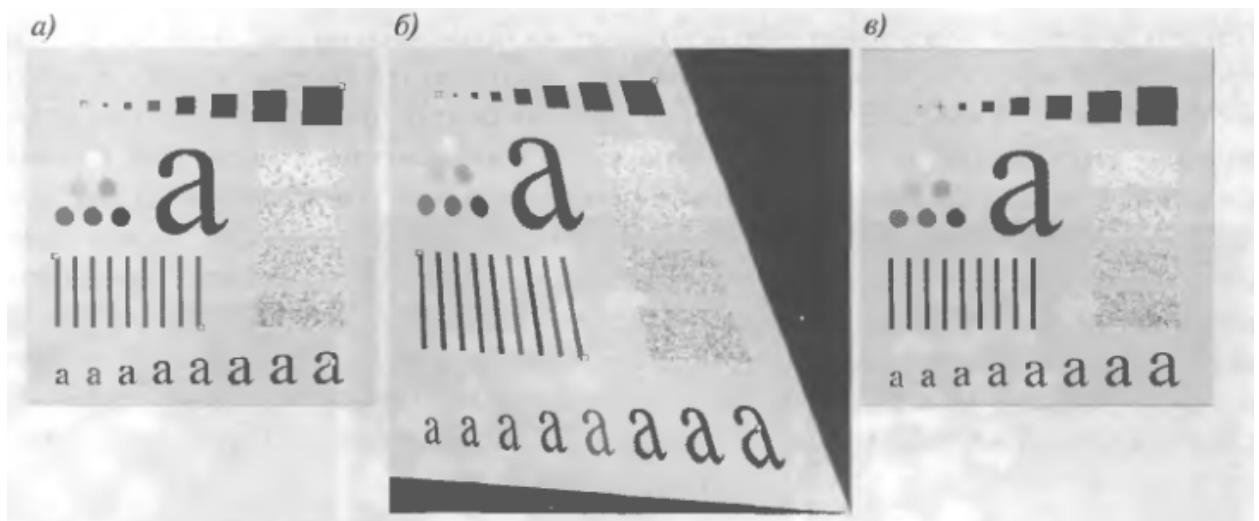


Рисунок 15 – Регистрация изображений с помощью контрольных точек
 а) Исходное изображение с контрольными точками (мелкие квадратики, наложенные на изображение) б) Геометрически искаженное изображение с контрольными точками, в) Скорректированное изображение с помощью проективного преобразования, построенного по контрольным точкам.

Пусть, к примеру, f обозначает изображение на рисунке 15а, а g - это изображение на рисунке 15б. Координаты контрольных точек на изображении f : (83,81), (450,56), (43,293), (249,392) и (436,442). Соответствующие контрольные точки на изображении g имеют координаты (68,66), (375,47), (42, 286), (275, 434), и (523,532). Тогда команда по пригонке изображений f и g имеет вид:

```
>> asepoints = [83 81; 450 56; 43 293; 249 392; 436 442];
>> inputpoints = [68 66; 375 47; 42 286; 275 434; 523 532];
>> tform = cp2tform(inputpoints, basepoints, 'protective');
>> gp = imtransform(g, tform, 'XData', [1 502], 'YData', [1
502]);
```

На рисунке 15в показано преобразованное изображение.

В пакете IPT также предусмотрен графический пользовательский интерфейс для ручного выбора контрольных точек на паре изображений. Рисунок 16 представляет собой снимок с дисплея компьютера окна этого инструмента, который вызывается командой `cpselect`.

Таблица 4. Типы преобразований, которые поддерживают функции `cp2tform` и `maketform`.

Тип	Описание	Функции
-----	----------	---------

преобразования		
Affine	Композиция растяжения/сжатия, поворота, сдвига и переноса. Прямые линии остаются прямыми, параллельные линии остаются параллельными	maketform cp2tform
Box	Независимое растяжение/сжатие и перенос по любой размерности; подмножество аффинных преобразований.	maketform
Composite	Семейство пространственных преобразований, которые применяются последовательно.	maketform
Custom	Пространственное преобразование, заданное пользователем, который определяет функции для вычисления T и T^{-1} .	maketform
Linear conformal	Растяжение/сжатие (одинаковое по обеим размерностям), попорот и перенос; подмножество аффинных преобразований.	cp2tform
LWM	Локально взвешенное среднее; локально варьируемое пространственное преобразование.	cp2tform
Piecewise linear	Локально варьируемое пространственное преобразование.	cp2tform
Polynomial	Входные пространственные координаты выражаются в виде полиномиальных функций входных координат.	cp2tform
Projective	Как и при аффинных преобразованиях, прямые линии остаются прямыми, однако параллельные переходят в непараллельные с удаленной точкой пересечения.	maketform cp2tform

Общая постановка задачи

В результате выполнения заданий лабораторной работы студенты **должны уметь** создавать программно-алгоритмическую поддержку для компьютерной реализации требуемых технике методов восстановления в MatLab.

Лабораторные занятия проводятся в компьютерных классах.

Список индивидуальных данных

Написать файл-функцию и GUI интерфейс для решения следующих задач.

Скопировать в папку для выполнения лабораторной работы все необходимые m-функции и файлы изображений.

Задание. Восстановление изображений.

1. Ввести изображение из файла, самостоятельно выбранного студентом. Используйте функцию **imshow** для вывода изображений на экран.
2. Добавить шум с помощью функции **imnoise** согласно варианту.

№ варианта	Параметры шума
1.	f, 'gaussian', m, var
2.	f, 'localvar', V
3.	f, 'localvar', image_intensity, var
4.	f, 'salt & paper', d
5.	f, 'speckle', var
6.	f, 'poisson'
7.	f, 'gaussian', m, var
8.	f, 'localvar', V
9.	f, 'localvar', image_intensity, var
10.	f, 'salt & paper', d

3. Используя функцию **imnoise2**, сгенерировать равномерно распределенные случайные числа в генераторе случайных чисел с заданной функцией распределения.
4. Построить гистограммы данных, сгенерированных функцией **imnoise2**.
5. Используя функцию **imnoise3**, добавить периодический шум.
6. Оценить параметры шума.
7. Удалить шум с помощью функции **spfilt**.
8. Выполнить адаптивную медианную фильтрацию.
9. Выполнить моделирование размытого зашумленного изображения.
10. Используя функцию **deconvwnr**, выполнить восстановление размытого зашумленного изображения.
11. Используя функцию **deconvreg**, выполнить восстановление смазанного зашумленного изображения.
12. Используя функцию **deconvlucy**, выполнить восстановление смазанного и зашумленного изображения.
13. Используя функцию **deconvblind**, построить приближение PSF.
14. Выполнить аффинные преобразования с помощью функции **vistformfwd**.
15. Выполнить пространственное преобразование изображений.
16. Сохранить полученные изображения и графики.

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора

в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.

Лабораторная работа №7. Морфологическая обработка изображений

Цель работы:

Целью лабораторной работы является получение навыков самостоятельной алгоритмической и программной реализации на компьютерной технике методов морфологической обработки изображений в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-1	3-1	Знает основы современных технологий сбора, обработки и представления информации
	3-2	Способен ориентироваться в информационном потоке в глобальных компьютерных сетях
ПК-2	3-1	Способен анализировать социально-экономические проблемы
	3-2	Знает методы расчетов математических моделей
ПК-5	3-1	Знает методы оценки проектов по информатизации
	3-2	Знает методы оценки проектов по автоматизации решения прикладных задач

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-14	У-1	Знает методы анализа прикладной области на логическом уровне
	У-2	Знает методы анализа прикладной области на алгоритмическом уровне

ПК-25	У-1	Знает различные математические методы решения прикладных задач
	У-2	Знает методы формализации решения прикладных задач

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-2	B-1	Способен анализировать социально-экономические проблемы
	B-2	Знает методы расчетов математических моделей
ПК-5	B-1	Знает методы оценки проектов по информатизации
	B-2	Знает методы оценки проектов по автоматизации решения прикладных задач

Теоретическая часть

Словом *морфология* принято называть область биологии, которая изучает форму и строение животных и растений. Мы будем использовать этот термин в контексте *математической морфологии*, которая является инструментом для извлечения определенных компонентов изображения, полезных для представления и описания форм объектов, например, их границ, оставов или выпуклых оболочек. Мы будем так же рассматривать морфологические методы, которые применяются на этапах предварительной и заключительной обработки изображений, например, морфологическая фильтрация, утончение и усечение.

Базовые понятия теории множеств

Пусть Z обозначает множество целых чисел. Процесс дискретизации при формировании цифровых изображений можно представить себе в виде разделения плоскости xy координатной сеткой на ячейки, а координаты центров этих ячеек являются парами декартова произведения Z^2 (Декартовым произведением множества Z на себя называется множество всех упорядоченных пар (z_i, z_j) , где z_i, z_j – любые целые числа. Его принято обозначить Z^2). Пользуясь терминологией теории множеств, мы скажем, что функция $f(x, y)$ называется *цифровым изображением*, если (x, y) – это целые числа из Z^2 , а f – отображение, которое сопоставляет значение яркости (которое принадлежит множеству вещественных чисел R) каждой паре координат (x, y) . Если значения яркости из R являются целыми числами (как это обычно предполагается в нашей книге), то цифровое изображение становится двумерной функцией, координатами и амплитудами (т.е. значениями яркости), которой служат целые числа.

Пусть A – некоторое множество из Z^2 , элементами которого являются координаты пикселов (x, y) . Если элемент $w = (x, y)$ принадлежит A , то это принято обозначать символической записью

$$w \in A.$$

В противном случае, если w не принадлежит A , то принято писать

$$w \notin A.$$

Множество пикселов B , удовлетворяющее некоторому условию, представляется в виде

$$B = \{ w \mid \text{условие} \}.$$

Например, множество всех пикселов, не принадлежащих множеству A , которое принято обозначать A^c , задается формулой

$$A^c = \{ w \mid w \notin A \}.$$

Это множество называется *дополнением* множества A .

Объединение двух множеств A и B , которое обозначается

$$C = A \cup B,$$

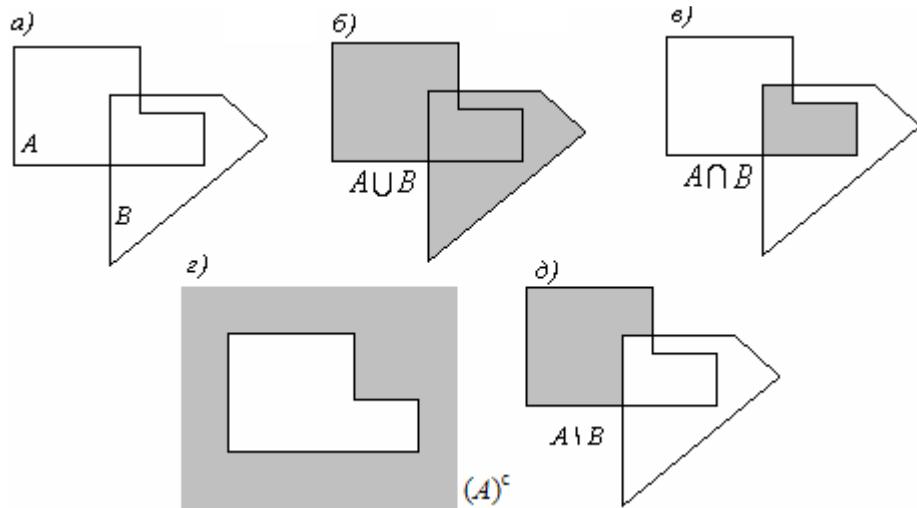
есть по определению множество всех пикселов, которые принадлежат или множеству A , или множеству B , или одновременно обоим множествам. Аналогично, *пересечение* двух множеств A и B состоит из всех элементов, которые одновременно принадлежат и множеству A , и множеству B , и обозначается

$$C = A \cap B.$$

Разность двух множеств A и B обозначается $A \setminus B$. Оно состоит из пикселов, которые принадлежат A , но не принадлежат B , т.е.

$$A \setminus B = \{ w \mid w \in A, w \notin B \}.$$

Рисунок 1 иллюстрирует введение выше операции над множествами, где результат каждой операции обозначен темным цветом.



Риунок 1 – а) Два множества A и B . б) Объединение множеств A и B . в) Пересечение множеств A и B . г) Дополнение множества A . д) Разность множеств A и B .

Вместе с введенными базовыми понятиями нам понадобятся еще две операции, широко используемые при морфологическом анализе изображений

и применяемые ко множествам, элементами которых служат координаты пикселов. Центральным отражением множества B называется множество B^\wedge , определяемое по формуле

$$B^\wedge = \{w \mid w = -b, b \in B\}.$$

Параллельный перенос (или сдвиг) множества A в точку $z = (z_1, z_2)$ обозначается $(A)_z$ и задается формулой

$$(A)_z = \{c \mid c = a + z, a \in A\}.$$

Рисунок 2 иллюстрирует эти две операции применительно к множествам из рисунка 1. Жирные точки обозначают начало координат для каждого множества.

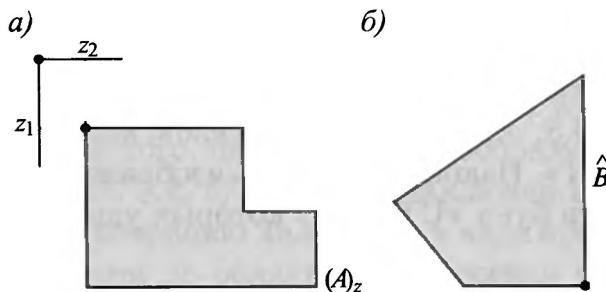


Рисунок 2 – а) Сдвиг множества A в точку z . б) Центральное отражение множества B . (Использованы множества A и B из рисунка 1)

Двоичные изображения, множества и логические операции

Язык и теория математической морфологии позволяют взглянуть на двоичные изображения как бы с двух сторон. С одной стороны, двоичное изображение (как часто в этой книге) можно себе представлять в виде функции с двумя возможными значениями координат x и y . Теория морфологии рассматривает двоичное изображение в виде множества его пикселов переднего плана (со значениями 1), которое лежит в пространстве Z^2 . Введенные выше операции над множествами типа объединение и пересечение можно напрямую применять к двоичным изображениям. Например, если A и B – двоичные изображения, то $C = A \cup B$ – также двоичное изображение, причем пиксели на изображении C являются пикселями переднего плана, если соответствующие пиксели на A , на B или на обоих этих изображениях принадлежат переднему плану. При первом, функциональном подходе, изображение C задается формулой

$$C(x, y) = \begin{cases} 1, & \text{если } A(x, y) = 1, \text{ или } B(x, y) = 1 \text{ или оба равны 1;} \\ 0 & \text{в противном случае.} \end{cases}$$

А, с точки зрения теории множеств, изображение C определяется выражением

$$C = \{(x, y) \mid (x, y) \in A, \text{ или } (x, y) \in B, \text{ или } (x, y) \in (A \text{ и } B)\}.$$

Операции над множествами, приведенные на рисунке 1, можно совершать над двоичными изображениями с помощью следующих логических операций MATLAB: OR() (логическое сложение), AND (&) (логическое умножение) и NOT (~) (отрицание), как показано в таблице 1.

Таблица 1. Использование логических операций MATLAB для совершения операций над двоичными изображениями

Операции	Выражение MATLAB для двоичных изображений	Имя
$A \cap B$	$A \& B$	AND
$A \cup B$	$A B$	OR
A^c	$\sim A$	NOT
$A \setminus B$	$A \& \sim B$	DIFFERENCE

В качестве простой иллюстрации рассмотрим рисунок 3, на котором показаны результаты применения ряда логических операций к двум двоичным изображениям элементов текста. (Мы следуем правилу IPT, по которому пиксели переднего плана (со значением 1) отображаются белым цветом.) Изображение на рисунке 3 g является объединением изображений «UTK» и «GT»; на нем присутствуют пиксели переднего плана обоих изображений. Наоборот, пересечение этих двух изображений, показанное на рисунке 3 d , состоит из пикселов, которые принадлежат перекрытию букв «UTK» и «GT». Наконец, разность изображений, приведенная на рисунке 3 e , показывает части букв «UTK», из которых удалены пиксели букв «GT».

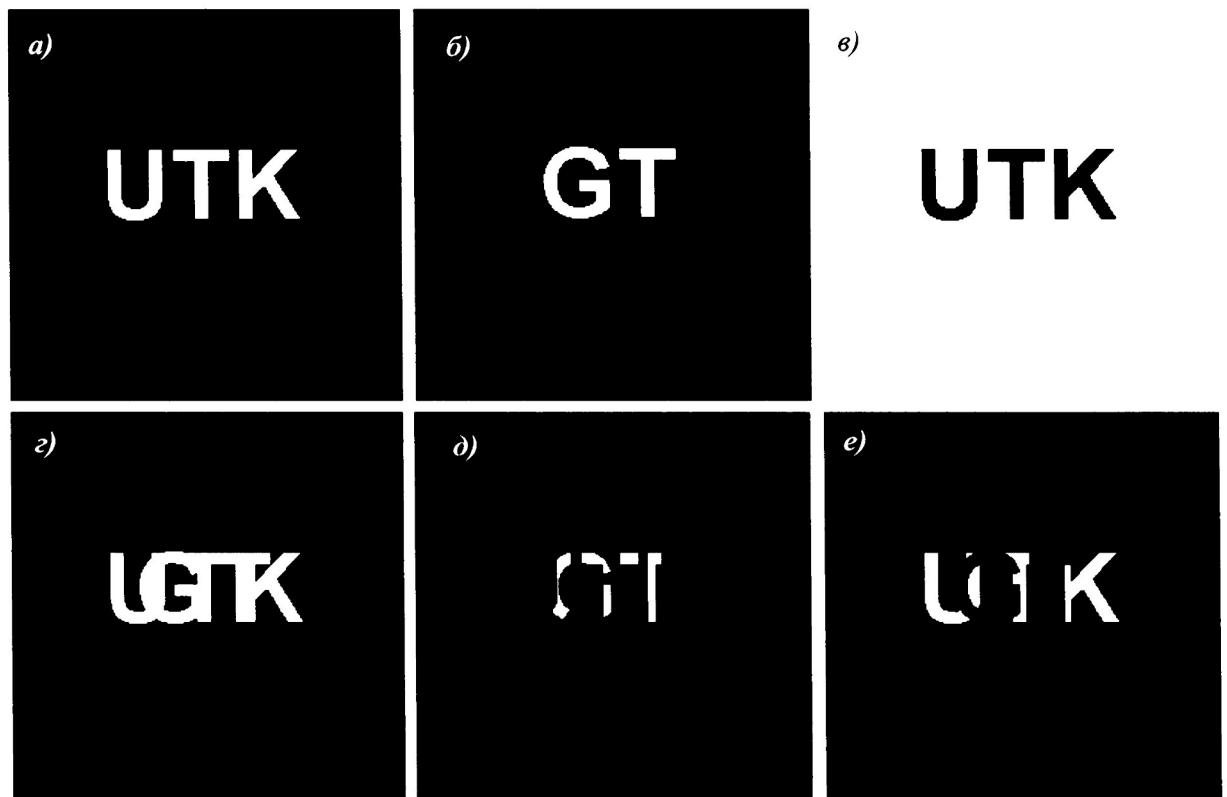


Рисунок 3 – а) Двоичное изображение А. б) Двоичное изображение В. в) Дополнение $\sim A$. г) Объединение $A | B$. д) Пересечение $A \& B$. е) Разность множеств $A \& \sim B$.

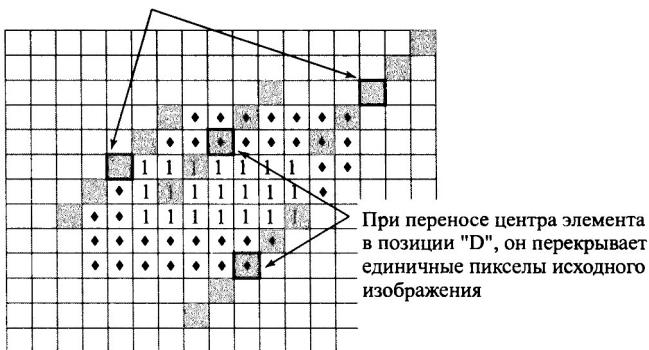
Дилатация и эрозия

Операции дилатации и эрозии имеют основополагающее значение при морфологической обработке изображений. Многие алгоритмы, рассматриваемые в этой главе, построены на этих операциях, которые далее определяются, иллюстрируются и обсуждаются.

Дилатация

Операция *дилатации* «наращивает» или «утолщает» объекты на двоичных изображениях. Способ и степень этого утолщения контролируется некоторой формой, которая называется *структурообразующим элементом*. Рисунок 4 иллюстрирует работу дилатации. На рисунке 4 a задано простое двоичное изображение прямоугольника. На рисунке 4 b показан структурообразующий элемент, состоящий из пяти пикселов, расположенных по диагонали. С точки зрения вычислений, структурообразующий элемент представляется в виде матрицы из нулей и единиц, но часто бывает удобно показывать только единицы, как изображено на рисунке. Кроме того, центр структурообразующего элемента должен быть ясно выделен. На рисунке 4 b его центр заключен в квадратик. Рисунок 4 c графически отображает операцию дилатации в виде процесса обноса центра структурообразующего элемента по области изображения и отметки тех положений центра, когда соответствующий элемент перекрывает некоторые пиксели переднего плана изображения (со значением 1). Полученное изображение показано на рисунке 4 d на котором значения 1 присвоены тем пикセルам, которые соответствуют положениям центра структурообразующих элементов, перекрывающих пиксели переднего плана исходного изображения.

- в) Структурообразующие элементы, перенесенные в эти положения, не перекрывают ни одного единичного пикселя исходного изображения.



г) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Рисунок 4 – Иллюстрация применения дилатации. а) Исходное изображение с прямоугольником. б) структурообразующий элемент из пяти пикселов по диагонали. Его центр обведен квадратиком. в) Структурообразующий элемент разнесен по нескольким положениям на изображении. г) Обработанное изображение.

Дилатацию можно строго определить математически в терминах операций над множествами. Дилатация множества A по множеству B (или относительно множества B) обозначается $A \oplus B$ и определяется по формуле

$$A \oplus B = \{z | (B^{\wedge})_z \cap A \neq \emptyset\},$$

где \emptyset обозначает пустое множество, а B – это структурообразующий элемент. Другими словами, дилатация A относительно B является множеством, которое состоит из всех положений центров структурообразующего элемента, который, будучи центрально отраженным и сдвинутым, имеет частичное перекрытие с множеством A . Заметим, что процесс сдвига структурообразующего элемента похож на механизм построения свертки. На рис. 4 не видно действие центрального отражения элемента B , так как сам он центрально-симметричен. На рис. 5 изображен нецентрально-симметричный структурообразующий элемент и его центральное отражение.

<i>a)</i>	<i>b)</i>
$\begin{matrix} & & 1 \\ & 1 & 1 \\ 1 & 1 & 1 & \boxed{1} & 1 & 1 \\ 1 & & \end{matrix}$	$\begin{matrix} & & & 1 \\ & 1 & 1 & \boxed{1} & 1 & 1 & 1 \\ 1 & 1 & & & & & \end{matrix}$

Рисунок 5 – Центральное отражение структурообразующего элемента. а) Несимметричный структурообразующий элемент. б) структурообразующий элемент, отраженный относительно своего центра.

Операция дилатации является *коммутативной*, т.е. $A \oplus B = B \oplus A$. Однако бывает удобно помещать на место первого операнда изображение, а на место

второго – структурообразующий элемент, который, обычно, существенно меньше обрабатываемого изображения. Мы будем так их располагать и в дальнейшем.

Пример 1. Простое применение дилатации.

В пакете IPT операция дилатации реализуется функцией `imdilate`. Ее основная форма вызова имеет вид

$$A_2 = \text{imdilate}(A, B),$$

где A и A_2 – это двоичные изображения, а B – матрица из нулей и единиц, которая обозначает структурообразующий элемент. На рисунке 6а дан пример двоичного изображения, которое содержит текст с разорванными буквами. Мы хотим применить дилатацию к этому изображению со структурообразующим элементом

$$\begin{matrix} 0 & 1 & 0 \\ 1 & \boxed{1} & 1 \\ 0 & 1 & 0 \end{matrix}$$

Следующая последовательность команд считывает изображение из файла, строит матрицу структурообразующего элемента, совершает дилатацию и показывает результат.

```
>> A=imread('broken_text.tif');
>> B=[ 0 1 0; 1 1 1; 0 1 0];
>> A2=imdilate(A,B);
>> imshow(A2)
```

На рисунке 6б показано обработанное изображение.

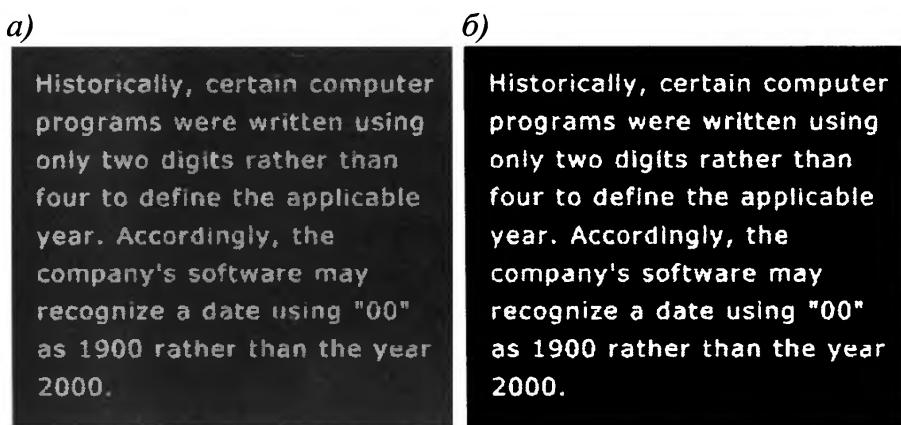


Рисунок 6 – Простой пример дилатации. *а)* Исходное изображение, содержащее разорванные символы. *б)* Изображение после применения дилатации.

Разложение структурообразующих элементов

Операция дилатации является ассоциативной, т.е. выполнено равенство

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C.$$

Предположим, что структурообразующий элемент B можно представить в виде дилатации двух других элементов B_1 и B_2 :

$$B = B_1 \oplus B_2.$$

Тогда $A \oplus B = A \oplus (B_1 \oplus B_2) = (A \oplus B_1) \oplus B_2$. Это означает, что дилатацию A по B можно построить, выполнив сначала дилатацию A по B_1 , а затем к результату применить дилатацию по B_2 . Говорят, что B *разложен* на два структурообразующих элемента B_1 и B_2 .

Свойство ассоциативности весьма важно, потому что время вычисления дилатации пропорционально числу ненулевых пикселов структурообразующего элемента. Рассмотрим, например, дилатацию с матрицей 5×5 , состоящую из одних единиц:

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & \boxed{1} & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

Этот структурообразующий элемент можно разложить на строку и столбец, состоящие из пяти единиц:

$$\begin{bmatrix} 1 & 1 & \boxed{1} & 1 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ \boxed{1} \\ 1 \\ 1 \end{bmatrix}$$

Число элементов исходного структурообразующего элемента равно 25, а общее число элементов разложения строка-столбец равно 10. Это означает, что если к некоторому изображению применить дилатацию по элементу-строке, а затем – по элементу-столбцу, то вся процедура будет работать в 2.5 раза быстрее, чем дилатация по целому массиву 5×5 . На практике ускорение вычислений будет несколько меньшим, поскольку имеются некоторые дополнительные операции, которые необходимо выполнять при каждой дилатации, а при использовании декомпозиции (разложения) требуется, по крайней мере, две дилатации. Однако выигрыш по скорости при использовании этого приема все равно остается значительным.

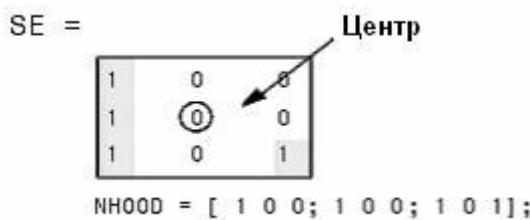
Функция `strel`

Функция `SE=strel(shape, parameters)` создает структурный элемент `SE`, тип которого описывается в параметре `shape`. В таблице перечислены все типы поддерживаемых форм структурных элементов. В зависимости от параметра формы `shape`, структурный элемент `strel` может иметь ряд дополнительных параметров. Синтаксис написания функции зависит от формы структурного элемента.

Плоские структурные элементы	
'arbitrary'	'pair'
'diamond'	'periodicline'
'disk'	'rectangle'

'line'	'square'
'octagon'	
Неплоские структурные элементы	
'arbitrary'	'ball'

Функция **SE=strel('arbitrary', NHOOD)** создает плоский структурный элемент, где параметр NHOOD описывает окрестность. NHOOD представляет собой матрицу, состоящую из нулей и единиц; расположение единиц в окрестности зависит от морфологической операции. Центром NHOOD является центральный элемент, который определяется из выражения $\text{floor}((\text{size}(NHOOD)+1)/2)$. Параметр 'arbitrary' можно опустить и вместо него использовать strel(NHOOD).

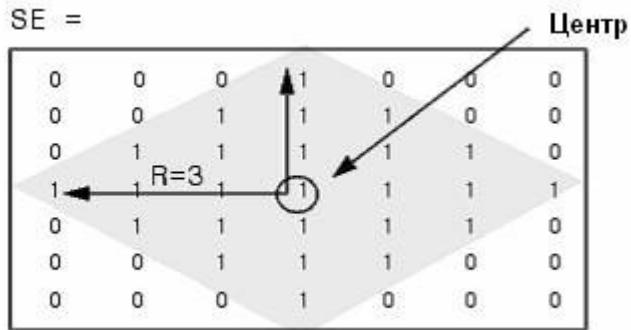


Функция **SE=strel('arbitrary', NHOOD, HEIGHT)** создает неплоский структурный элемент, где параметр NHOOD описывает окрестность. HEIGHT представляет собой матрицу той же размерности, что и NHOOD, содержащую значения, связанные с каждым ненулевым элементом в NHOOD. Матрица HEIGHT состоит из вещественных конечных значений. Параметр 'arbitrary' можно опустить и использовать выражение strel(NHOOD, HEIGHT).

Функция **SE=strel('ball', R, H, N)** создает неплоские, "шарообразные" (точнее эллипсообразные) структурные элементы с радиусом R в плоскости X-Y и высотой H. Отметим, что R должно выражаться положительным целым числом, H представляется вещественным скаляром, а N положительным целым числом. Когда N является больше 0, тогда шарообразные структурные элементы аппроксимируются последовательностью из N неплоских линейных структурных элементов. Когда N равно 0, аппроксимация не используется и структурные элементы формируются из всех пикселей, удаленных не более чем на от R от центра. Соответствующие значения определяются по формуле, которая выражает зависимость в эллипсе между R и H. Когда N не определено, тогда по умолчанию это значение равно 8.

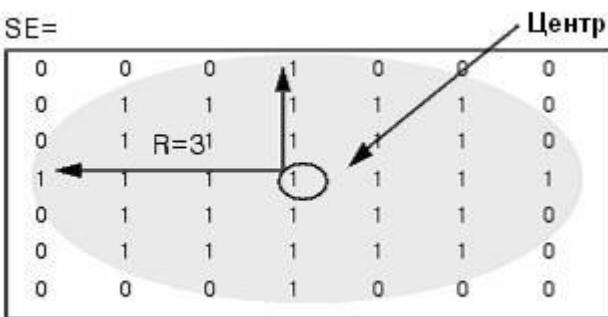
Примечание. Морфологические операции работают значительно быстрее, если структурные элементы аппроксимируются ($N>0$).

Функция **SE=strel('diamond', R)** создает плоский ромбообразный структурный элемент, где R описывает расстояние от центра структурного элемента до точки ромба. R должно представляться неотрицательными целыми числами.

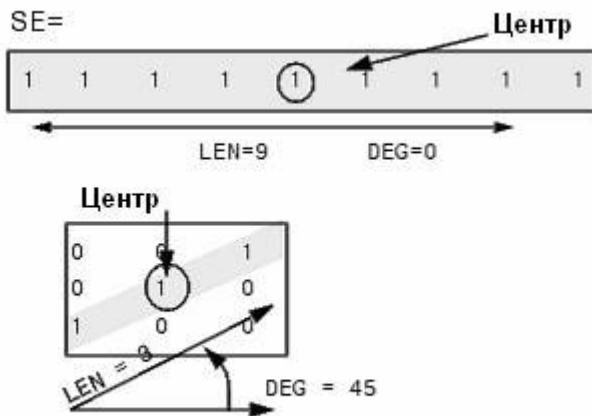


Функция **SE=strel('disk', R, N)** создает плоские, дискообразные структурные элементы, где параметр R описывает радиус. R должно быть неотрицательным вещественным числом. Параметр N должен принимать значения 0, 4, 6 или 8. Когда N больше 0, тогда дискообразный структурный элемент аппроксимируется последовательностью N периодично-линейных структурных элементов. Когда N равно 0, аппроксимация не используется и структурные элементы формируются из всех пикселей, удаленных не более чем на от R от центра. Код N не определено, тогда по умолчанию это значение равно 4.

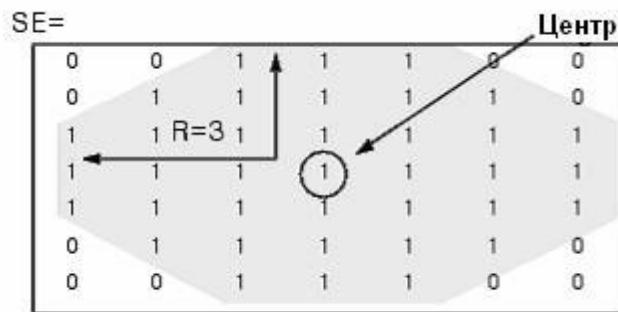
Примечание. Морфологические операции работают значительно быстрее, когда структурные элементы используют аппроксимацию ($N > 0$). Однако, структурные элементы, которые не используют аппроксимацию ($N = 0$) являются непригодными при решении задач гранулометрии. Иногда для формирования структурного элемента необходимо использовать аппроксимацию различных линейных структурных элементов. В этом случае число составных структурных элементов равно $N+2$.



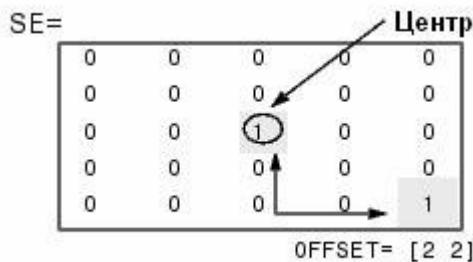
Функция **SE=strel('line', LEN, DEG)** создает плоский линейный структурный элемент, где параметр LEN описывает длину, а параметр DEG описывает угол (в градусах) линии, измеренный в направлении против часовой стрелки относительно горизонтальной оси. Параметр LEN характеризует дистанцию между центрами структурных элементов, расположенных на противоположных концах линии.



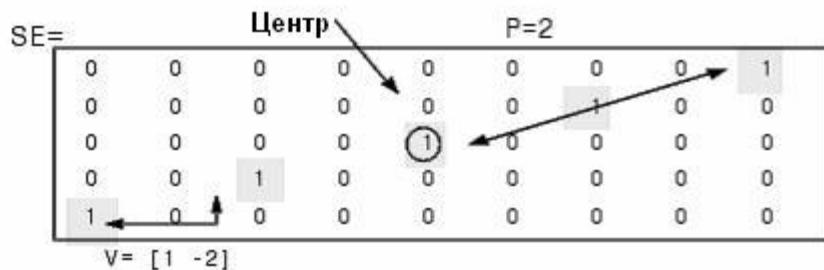
Функция **SE=strel('octagon', R)** создает плоские восьмиугольные структурные элементы, где R описывает расстояние между центром структурного элемента и стороной восьмиугольника, измеренное вдоль горизонтальной или вертикальной оси. Параметр R должен представляться неотрицательной трехэлементной составляющей.



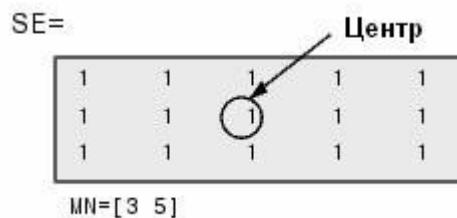
Функция **SE=strel('pair', OFFSET)** создает плоский структурный элемент, состоящий из двух частей. Одна часть расположена в центре. Расположение второй части зависит от описания, которое содержится в векторе OFFSET. OFFSET представляет собой двухэлементный вектор целых чисел.



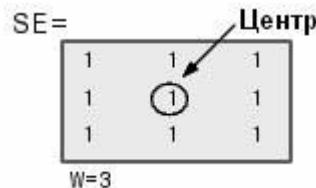
Функция **SE=strel('periodicline', P, V)** создает плоский структурный элемент, состоящий из $2*P+1$ частей. Параметр V является двухэлементным вектором, который содержит целочисленные значения строк и столбцов структурного элемента. Одна часть структурного элемента расположена в центре. Другие элементы расположены в $1*V, -1*V, 2*V, -2*V, \dots, P*V, -P*V$.



Функция **SE=strel('rectangle', MN)** создает плоский прямоугольный структурный элемент, где параметр MN описывает его размер. MN представляется двухэлементным вектором неотрицательных целых чисел. Первый элемент MN описывает число строк структурных элементов; второй элемент описывает число столбцов.



Функция **SE=strel('square', W)** создает структурные элементы в виде квадрата, размеры которого задаются параметром W. W должен представляться неотрицательными целыми числами.



Примечание. Для всех форм за исключением, представленной параметром 'arbitrary', структурные элементы создаются с помощью семейства технологий известных под общим названием *декомпозиция структурных элементов*. Принципиально то, что морфологические операции расширения для больших структурных элементов объектов выполняются быстрее, чем для последовательности небольших структурных элементов.

Метод.

В таблице перечислены методы, которые поддерживают STREL-объекты.

getheight	Получение высоты структурного элемента
getneighbors	Получение информации об окрестных элементах.
getnhood	Получение информации об окрестности.
getsequence	Получение последовательности составляющих структурного элемента.
isflat	Возвращение структурных элементов.
reflect	Отражение структурных элементов.
translate	Преобразование структурных элементов.

Функция **imdilate** автоматически использует форму разложения структурообразующего элемента, и при этом скорость вычисления возрастает примерно в три раза по сравнению с прямым использованием элемента в неразложенной форме (выигрыш по времени вычислений в $\approx 61/17$ раза).

Эрозия

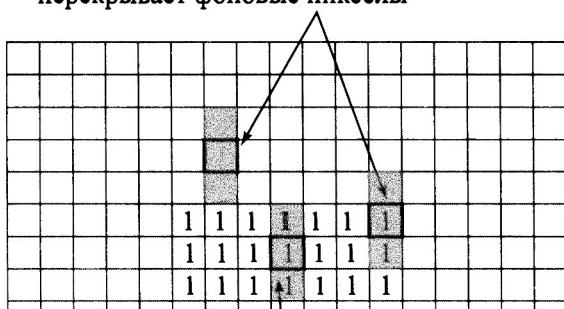
Операция эрозия «ужимает» или «утончает» объекты двоичных изображений. Как и при дилатации, вид и размер эрозии определяется структурообразующим элементом. Рисунок 7 иллюстрирует процесс эрозии. Рисунок 7 a совпадает с рисунком 4 a . На рисунке 4 b приведен структурообразующий элемент – короткий вертикальный отрезок. Рисунок 7 c графически описывает эрозию как процесс перемещения структурообразующего элемента по изображению и фиксации положений его центра, в которых этот элемент целиком состоит из пикселов переднего плана изображения. Выходом операции эрозии служит изображение со значениями 1 у тех пикселов, которые соответствуют положениям центра структурообразующих элементов, целиком состоящих из пикселов переднего плана исходного изображения (т.е. этим сдвигам элементов не принадлежат пиксели изображения со значением 0).

a)

б)

1

в) В этом положении выход равен нулю, так как структурообразующий элемент перекрывает фоновые пиксели



Здесь выход равен 1, так как структурообразующий элемент полностью состоит из фоновых пикселов

2)

Рисунок 7 – Иллюстрация применения эрозии. а) Исходное изображение с прямоугольником. б) Структурообразующий элемент из трех пикселов по вертикали. Его центр обведен квадратиком. в) Структурообразующий элемент разнесен по нескольким положениям на изображении. г) Обработанное изображение

Математическое определение операции эрозии похоже на определение дилатации. Эрозией множества A по B называется множество

$$A \ominus B = \{z | (B)_z \cap A^c = \emptyset\}.$$

Другими словами, эрозия A по B состоит из тех и только тех координат пикселов, для которых сдвиг множества B в эту точку не пересекается с фоном изображения A (напомним, что фон состоит из тех пикселов изображения, значения которых равны 0).

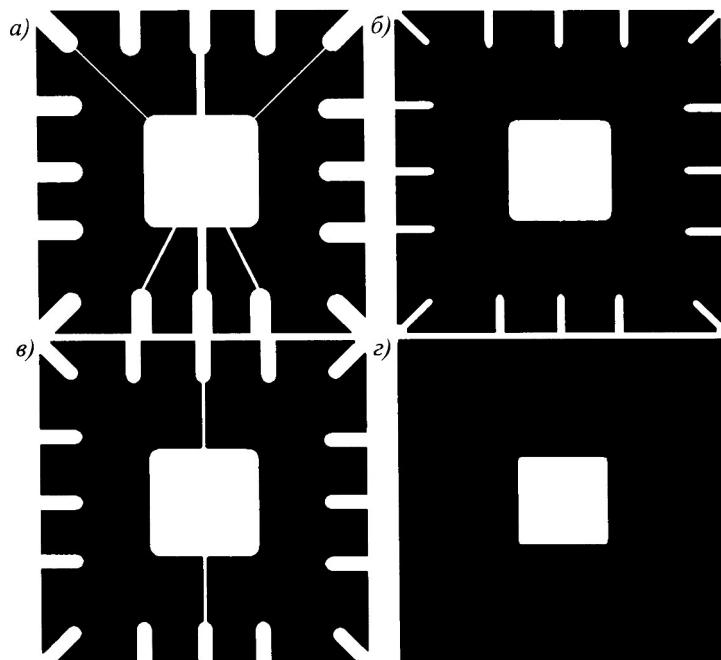


Рисунок 8 – Иллюстрация эрозии. а) Эрозия с кругом радиуса 10. б) Эрозия с кругом радиуса 5. в) Эрозия с кругом радиуса 20

Эрозия выполняется в ITP функцией `imerode`.

Функция `IM2=imerode(IM, SE)` проводит операцию утончения полутоновых, бинарных или упакованных бинарных изображений IM , возвращая утонченное изображение $IM2$. Аргумент SE является структурным элементом объекта или массивом структурных элементов, возвращаемым функцией `strel`.

Когда IM является логическим или структурным плоским элементом, функция `imerode` выполняет бинарное наращивание; в других случаях полутоновое утончение. Когда SE является массивом структурных элементов объекта, функция `imerode` выполняет многократное утончение исходного изображения, используя каждый структурный элемент из последовательности SE .

Функция **IM2=imerode(IM, NHOOD)** выполняет операцию утончения изображения IM, где NHOOD представляет собой массив нулей и единиц, описывающий структурные элементы окрестности. Синтаксически эта функция эквивалентна выражению **imerode(IM, strel(NHOOD))**. Функция **imerode** определяет центральный элемент окрестности за выражением $\text{floor}((\text{size}(NHOOD)+1)/2)$.

Функции **IM2=imerode(IM, SE, ПАСКОРТ, М)** или **imerode(IM, NHOOD, ПАСКОРТ, М)** определяют какой массив IM представляет собой бинарное изображение и проводят преобразование его размерностей M к виду исходного распакованного изображения. Параметр ПАСКОРТ может принимать следующие значения.

'ispacked'	Массив IM обработан как упакованное бинарное изображение с помощью функции bwpack . Изображение IM должно быть двумерным массивом с форматом представления данных <code>uint32</code> . Параметр SE также должен быть плоским двумерным структурным элементом.
'notpacked'	IM обработан как нормальный массив. Это значение принимается по умолчанию.

Когда параметр ПАСКОРТ принимает значение 'ispacked', необходимо точно определить значение M.

Функция **IM2=imerode(..., ПАДОПТ)** определяет размер результирующего изображения. Параметр ПАДОПТ может принимать одно из следующих значений.

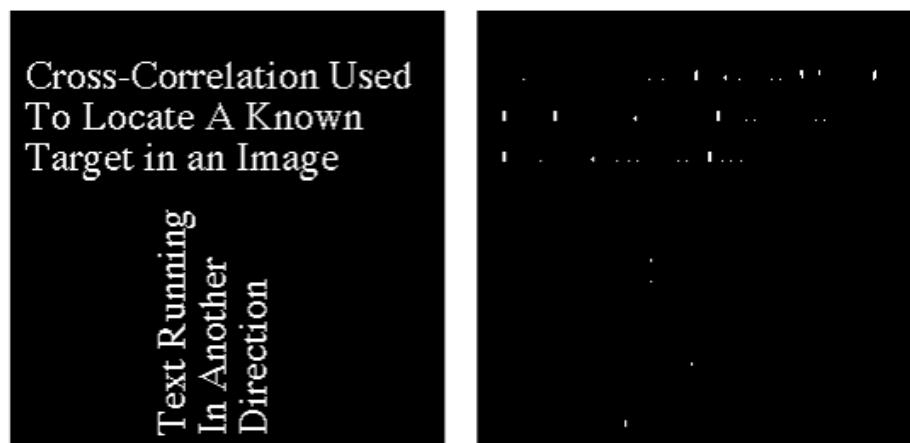
'same'	Размерность результирующего изображения совпадает с размерностью исходного изображения. Это значение принимается по умолчанию. Когда параметр ПАСКОРТ принимает значение 'ispacked', тогда ПАДОПТ должен принимать значение 'same'.
'full'	Вычисление полного утончения (эрозии).

Параметр ПАДОПТ является аналогом SHAPE, который является исходным параметром функций CONV2 и FILTER2.

Изображение IM должно быть представлено числовым или логическим массивом любой размерности. Когда массив IM логический, тогда структурные элементы SE плоские. Формат представления данных результирующего изображения совпадает с форматом исходного изображения. Когда исходное изображение упакованное, тогда результирующее изображение будет также упакованным.

Пример утончения бинарного изображения с вертикальными линиями.

```
bw=imread('text.tif');
se=strel('line', 11, 90);
bw2=imerode(bw, se);
imshow(bw), title('Original')
figure, imshow(bw2), title('Eroded')
```



Этот пример демонстрирует операцию утончения полутонового изображения с волновыми линиями.

```
I = imread('cameraman.tif');
se = strel('ball', 5, 5);
I2 = imerode(I, se);
imshow(I), title('Original')
figure, imshow(I2), title('Eroded')
```



Функция **imerode** автоматически принимает на себя преимущества декомпозиции структурных элементов объекта (когда декомпозиция возможна). Итак, когда выполняется наращивание с структурными элементами в виде их декомпозиции, функция **imerode** автоматически использует бинарное упакованное изображение для ускорения наращивания.

Комбинирование дилатации и эрозии

В конкретных приложениях обработки изображений операции дилатация и эрозия часто используется совместно. Обычно изображения подвергаются серии операций дилатации и/или эрозии с одним и тем же, а иногда и с разными структурообразующими элементами. В этом параграфе рассматриваются три часто используемые комбинации дилатации и эрозии: размыкание, замыкание и преобразование успех/неудача. Кроме того, вводятся операции с использованием поисковых таблиц, а также обсуждается функция **bwmorph** из пакета IPT, которая выполняет различные морфологические действия.

Размыкание и замыкание

Морфологическое размыкание A по B обозначается $A \circ B$ и определяется как эрозия A по B , после которой выполняется дилатация результата по B :

$$A \circ B = (A \Theta B) \oplus B.$$

Эту операцию можно также задать эквивалентной формулой

$$A \circ B = \bigcup \{(B)_z | (B)_z \subseteq A\},$$

где $\bigcup \{.\}$ обозначает объединение всех множеств внутри фигурных скобок, а формула $C \subseteq D$ означает, что множество C является подмножеством D . Такое представление имеет простую геометрическую интерпритацию. На рисунке 9а показано множество A и структурообразующий элемент B , имеющий форму круга. На рисунке 9б изображены некоторые сдвиги множества B , которые *полностью* содержатся в A . Объединение всех таких сдвигов выделено темным цветом на рисунке 9в. Эта область и является размыканием A по B . На этом рисунке белым цветом показаны области, где структурообразующий элемент целиком не поместился, поэтому они не принадлежат размыканию. Морфологическое размыкание удаляет те части объектов, в которых структурообразующий элемент полностью не помещается. Оно также сглаживает контуры объектов, разрушает тонкие соединения и удаляет острые выступы.

Морфологическое замыкание множества A по B обозначается $A \bullet B$. Эта операция представляет собой эрозию, примененную к результату дилатации:

$$A \bullet B = (A \oplus B) \Theta B.$$

Геометрически множество $A \bullet B$ представляет собой дополнение к объединению всех сдвигов множества B , которые не пересекаются с A . На рисунке 9г построено несколько сдвигов множества B , которые не пересекаются с A . Взяв дополнению к объединению всех таких сдвигов, мы получим область, затемненную на рисунке 9г, которое есть искомое замыкание. Подобно размыканию, морфологическое замыкание приводит к сглаживанию контуров объектов. Однако в отличие от размыкания, оно соединяет узкие разрывы, «заливает» длинные углубления малой ширины и заполняет малые отверстия, диаметр которых меньше размеров структурообразующего элемента.

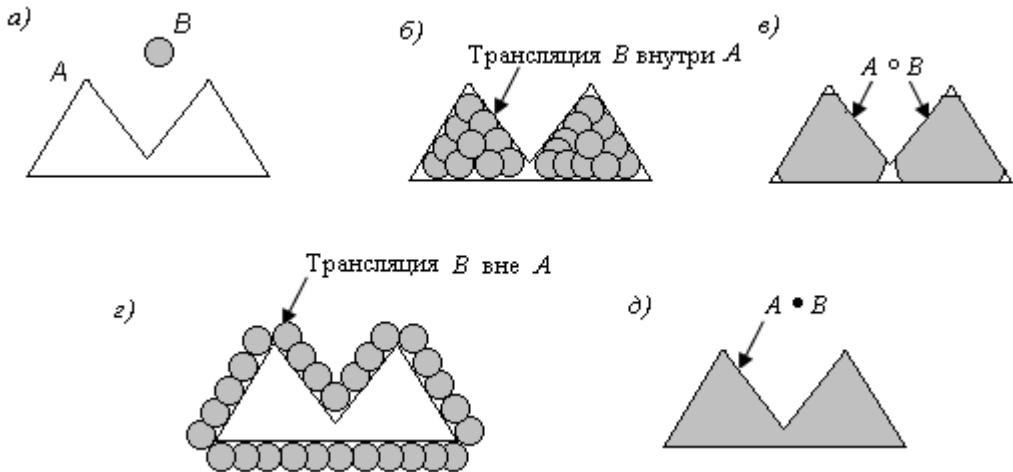


Рисунок 9 – Размыкание и замыкание как объединение сдвигов структурообразующих элементов. а) Множество A и структурообразующий элемент B . б) Сдвиги B , которые полностью помещаются внутри A . в) Полное размыкание A (затемненная область). г) Сдвиги B , которые лежат целиком вне A . д) Полное замыкание A (затемненная область).

Операции размыкания и замыкания реализованы в пакете IPT с помощью функций `imopen` и `imclose`. Обе эти функции имеют простые формы вызова

$C = \text{imopen}(A, B)$

и

$C = \text{imclose}(A, B)$

где A – это двоичное изображение, а B – матрица из 0 и 1, которая задает структурообразующий элемент. `Strel`-объект `SE` может использовать вместо аргумента B .

Пример 4. Использование функций `imopen` и `imclose`.

Этот пример иллюстрирует применение функций `imopen` и `imclose`. Изображение `shapes.tif`, построенное на рисунке 10а, имеет ряд характерных особенностей, которые помогут лучше понять действие размыкания и замыкания на такие объекты, как: острые выступы, тонкие перемычки, длинные углубления, изолированные отверстия, маленькие одиночные объекты и зубчатые элементы границ. Следующая последовательность команд размыкает изображение с помощью квадратного структурообразующего элемента 20×20 :

```
>> f=imread('shapes.tif');
>> se=strel('square', 20);
>> fo=imopen(f, se);
>> imshow(fo)
```

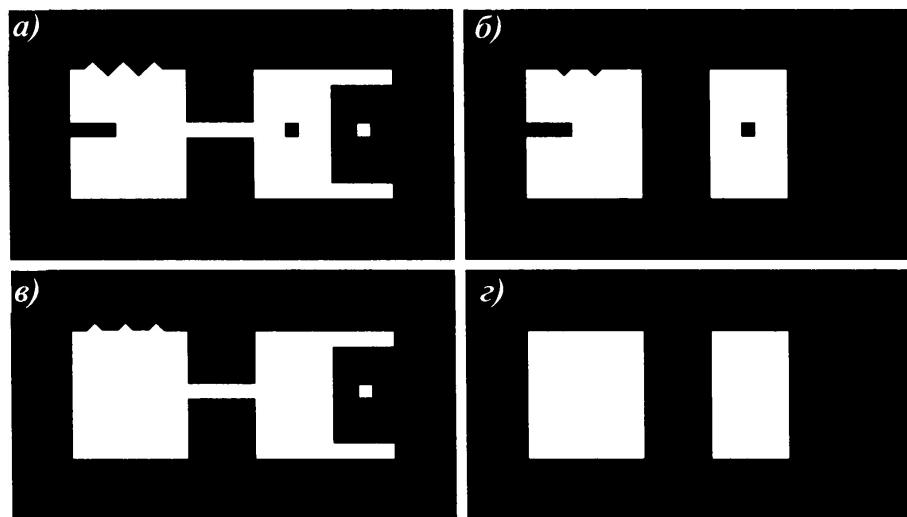


Рисунок 10 – Иллюстрация размыкания и замыкания. *а)* Исходное изображение. *б)* Размыкание. *в)* Замыкание. *г)* Замыкание

На рисунке 10*б* показан результат этих действий. Видно, что при этом исчезли тонкие выступы и острые зубцы, направленные во внешнюю сторону от границ объекта. Тонкое соединение и маленький изолированный объект также исчезли.

Команды

```
>> fc=imclose(f, se);
>> imshow(fc)
```

произвели результат, приведенный на рис. 9.10, *в*). Здесь были удалены узкое углубление, зубчатые неровности, направленные внутрь объекта, а также маленькое отверстие. Как будет показано в следующем абзаце, комбинация замыкания и размыкания может весьма эффективно удалять шумы. Имея в виду рисунок 10, применение замыкания к предыдущему результату размыкания выглядит как существенное сглаживание объектов изображения. Мы замыкаем разомкнутое выше изображение следующим образом:

```
>> foc=imclose(fe, se);
>> imshow(foc)
```

На рисунке 10*г* изображены полученные сглаженные объекты.

На рисунке 11 приведена другая иллюстрация полезности операций замыкания и размыкания при обработке нечетких и подпорченных отпечатков пальцев (рисунок 11*а*). Команды

```
>> f=imread('fingerprint.tif');
>> se=strel('square', 3);
>> fo=imopen(f, se);
>> imshow(fo)
```

совершают действия, результат которых приведен на рисунке 11*б*. Затем, что случайные мешающие точки были удалены при размыкании изображения, однако одновременно с этим были внесены лишние резервы и щели по краям отпечатков. Многие из этих недостатков можно исправить последующим замыканием результата размыкания:

```
>> foc=imclose (fo,se);  
>> imshow(foc)
```

Рисунок 11в показывает окончательный результат.



Рисунок 11 – Изображение испорченных отпечатков пальцев. а) Размыкание изображения. б) Замыкание после размыкания

Преобразование успех/неудача

В приложениях часто бывает необходимо уметь распознавать определенные конфигурации пикселов, например, изолированные пиксели переднего плана или пиксели, которые являются концами сегментов линий. *Преобразования успех/неудача* помогают при решении подобных задач. Преобразование успех/неудача множества A по множеству B обозначается через $A \otimes B$. Здесь B обозначает структурообразующую пару элементов $B = (B_1, B_2)$, а не один элемент, как это было раньше. По определению, преобразованием множества A по B называется множество

$$A \otimes B = (A \ominus B_1) \cap (A^c \ominus B_2).$$

На рисунке 12 показано, как преобразование успех/неудача можно использовать для определения местоположений следующей конфигурации пикселов, имеющей форму креста:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

На рисунке 12а такие конфигурации пикселов встречаются в двух местах. Эрозия по структурообразующему элементу B_1 устанавливает положение пикселов переднего плана, у которых одновременно имеются северный, восточный, южный и западный соседи переднего плана. Эрозия дополнения исходного изображения по элементу B_2 определяет положение всех пикселов, у которых одновременно имеются северо-восточный, юго-восточный, юго-западный и северо-западный соседи заднего плана (фона). Рисунок 12ж приводит результат пересечения (логическое AND) этих двух действий. Каждый пикセル переднего плана на этом изображении отвечает множеству пикселов, которое имеет искомую конфигурацию.

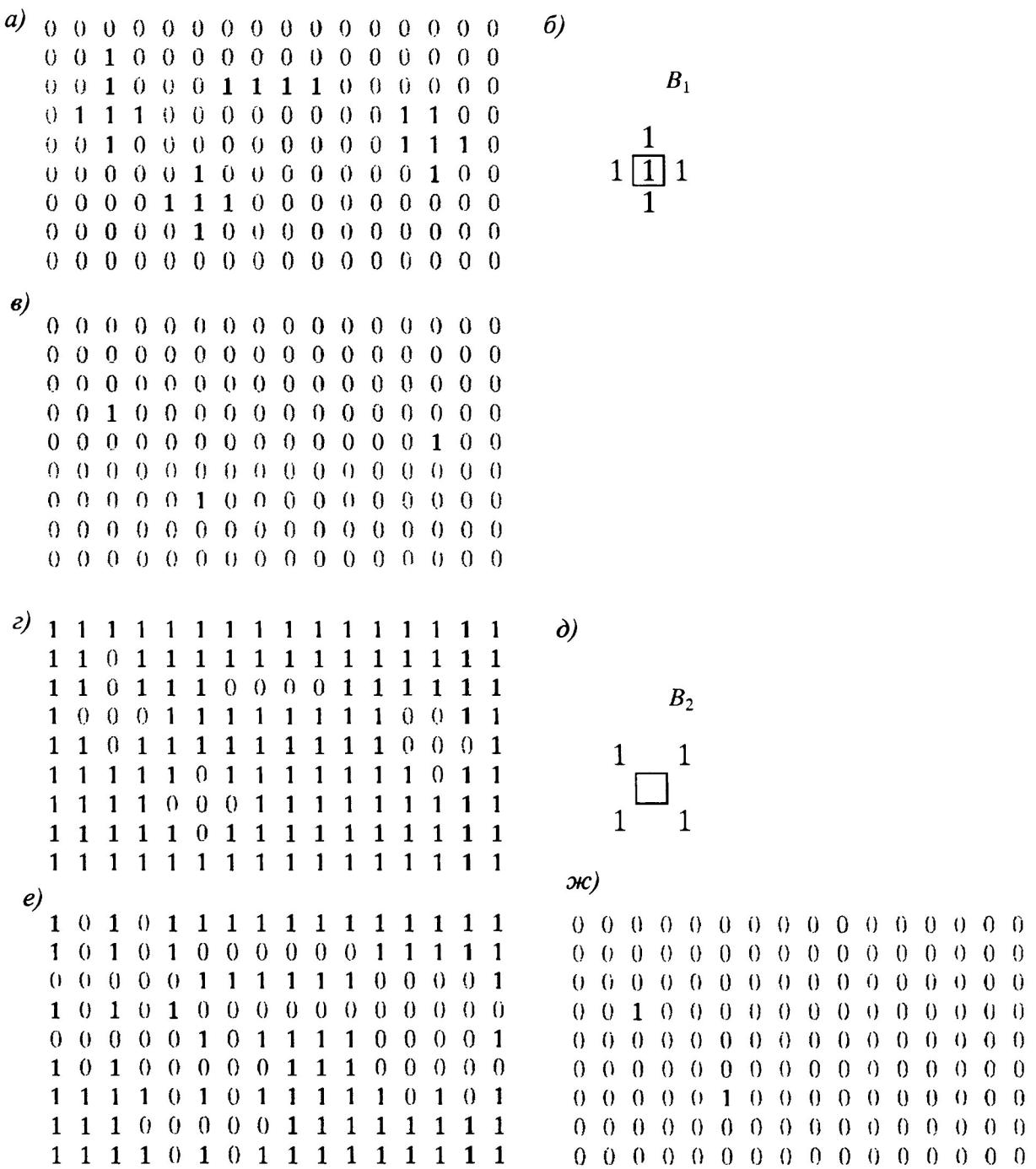


Рисунок 12 - а) Исходное изображение A. б) Структурообразующий элемент B1. в) Эрозия A по B1. г) Дополнение исходного изображения A^c. д) Структурообразующий элемент B2. е) Эрозия A^c по B2. ж) Результирующее изображение

Название операции «успех/ неудача» происходит от результата двух эрозий. Например, выходное изображение на рис. 9.12 состоит из всех позиций пикселов, которые подходят под B_1 («успех») и полностью не подходят под B_2 («неудача»).

Преобразование успех/ неудача реализовано в IPT функцией `bwhitmiss`, которая имеет синтаксис

$$C = \text{bwhitmiss}(A, B1, B2),$$

где С – это результат операции, А – исходное изображение, а В₁ и В₂ – структурообразующие элементы, задействованные в преобразовании, которые обсуждались выше.

Функция **BW2=bwhitmiss(BW1, SE1, SE2)** выполняет операции типа “hit-and-miss” с использованием структурных элементов SE1 и SE2. Операции типа “hit-and-miss” определяют те пиксели, окрестности которых совпадают по форме с структурным элементом SE1 и не совпадают по очертаниям с структурным элементом SE2. Структурные элементы SE1 и SE2 могут быть представлены в виде объекта из плоских структурных элементов, созданного на основе других структурных элементов или массива окрестности. Окрестности SE1 и SE2 не должны иметь перекрывающихся элементов. Синтаксис функции **bwhitmiss(BW1, SE1, SE2)** является эквивалентом выражения **imerode(BW1,SE1) & imerode(~BW1,SE2)**.

Функция **BW2=bwhitmiss(BW1,INTERVAL)** выполняет операции типа “hit-and-miss” с использованием одиночных массивов. Элементы массива могут быть равными 1, 0 или -1. 1 – значения элементов, которые составляют область SE1; -1 – значения элементов, которые составляют область SE2; и 0 – значения элементов, которые игнорируются. Синтаксически функция **bwhitmiss(INTERVAL)** является эквивалентом функции **bwhitmiss(BW1,INTERVAL==1, INTERVAL== -1)**.

BW1 должен быть логическим или цифровым массивом любой размерности, также он должен быть не разреженным. BW2 – это всегда логический массив того же размера, что и BW1. SE1 и SE2 должны быть плоскими структурными элементами (STREL objects), логическими или цифровыми массивами, содержащими элементы 1 или 0. INTERVAL представляет собой массив, содержащий элементы 1, 0 или -1.

Использование поисковых таблиц

Когда структурообразующие элементы, используемые в операции успех/ неудача, малы, то это преобразование можно вычислить быстрее, если воспользоваться поисковой таблицей LTU (LookUp Table). Метод заключается в предварительном вычислении значений выходных пикселов для дальнейшего использования. Например, всего имеется $2^9=512$ различных конфигураций значений пикселов двоичного изображения 3×3 .

Чтобы воспользоваться поисковой таблицей, необходимо присвоить уникальный индекс каждой возможной конфигурации пикселов. Простой способ сделать это, скажем, для случая 3×3 , заключается в умножении каждой конфигурации 3×3 поэлементно на матрицу

$$\begin{matrix} 1 & 8 & 64 \\ 2 & 16 & 128 \\ 4 & 32 & 256 \end{matrix}$$

и в сложности полученных произведений. Эта процедура назначает уникальное значение в интервале от 0 до 511 каждой двоичной окрестностной конфигурации 3×3 . Например, окрестности

$$\begin{matrix} 1 & 1 & 0 \end{matrix}$$

1	0	1
1	0	1

присваивается

число

$1(1)+2(1)+4(1)+8(1)+16(0)+32(0)+64(0)+128(1)+256(1)=399$, где первые множители являются коэффициентами из предыдущей матрицы, а в скобках стоят значения пикселов, взятые по столбцам.

В пакете IPT имеется две функции `makelut` и `applylut` (использование которых будет проиллюстрировано позже в этом параграфе), реализующие эти действия. Функция `makelut` строит поисковую таблицу на основе некоторой функции, заданной пользователем, а `applylut` обрабатывает входное изображение с помощью построенной таблицы. Если продолжить рассмотрение случая 3×3 , то использование `makelut` предполагает задание пользовательской функции, которая принимает на входе матрицы 3×3 , а возвращает одно значение, обычно 0 или 1. Функция `makelut` вызывает эту заданную пользователем функцию 512 раз, передавая ей все возможные окрестности 3×3 . Она возвращает результат вычислений в виде вектора из 512 элементов.

В качестве иллюстрации мы напишем функцию (назовем ее `endpoints.m`), которая использует `makelut` и `applylut` для обнаружения концевых точек на двоичном изображении. *Концевой точкой* мы будем называть пиксель переднего плана, у которого имеется в точности один сосед переднего плана. Функция `endpoints` вычисляет, а затем применяет поисковую таблицу для обнаружения концевых точек входного изображения. Стока программного кода

`persistent lut,`

которая используется в `endpoints`, определяет переменную с именем `lut` и объявляет, что она принадлежит типу `persistent` (устойчивый). MATLAB «помнит» значение переменных типа `persistent` в промежутках между вызовами данной функции. При первом вызове функции `endpoints` переменная `lut` автоматически инициализируется пустой матрицей (`[]`). Когда `lut` пуста, то функция вызывает `makelut`, передавая ее манипулятор подфункции `endpoint_fcn`. Затем функция `applylut` находит концевые точки с помощью поисковой таблицы. Поисковая таблица хранится в переменной `lut` типа `persistent`, поэтому при следующем вызове функции `endpoints` не придется заново строить эту таблицу.

```

function g = endpoints(f)
persistent lut
if isempty(lut)
    lut = makelut(@endpoint_fcn, 3);
end
g = applylut(f,lut);
function is_end_point = endpoint_fcn(nhood)
is_end_point = nhood(2,2) & (sum(nhood(:)) == 2);

```

Пример 6. Реализация игры Конвея «жизнь» на основе двоичного изображения и поисковой таблицы.

Интересным приложением поисковых таблиц может служить «игра жизни» Конвея, в которой участвуют «организмы», существующие на прямоугольной сетке и эволюционирующие из поколения в поколение. Мы воспользуемся ею для иллюстрации достоинства и простоты метода поисковых таблиц. Имеются простые правила, по которым организмы в игре Конвея рождаются, выживают и умирают при переходе от одного «поколения» к следующему.

Генетические правила Конвея определяют метод вычисления следующего поколения (или следующего двоичного изображения) по текущему поколению:

1. Каждый пиксель переднего плана, имеющий двух или трех соседей переднего плана, выживает в следующем поколении.
2. Каждый пиксель переднего плана, имеющий ноль, один или не менее четырех соседей, «умирает» в следующем поколении (т.е. становится фоновым пикселом) по причине его «изоляции» или «перенаселенности».
3. Каждый фоновый пиксель, у которого имеется ровно три соседних пикселя переднего плана, «рождает» пиксель, т.е. он становится пикселиом переднего плана.

Все «рождения» и все «смерти» происходят одновременно в процессе вычисления следующего двоичного изображения, определяя очередное поколение «организмов».

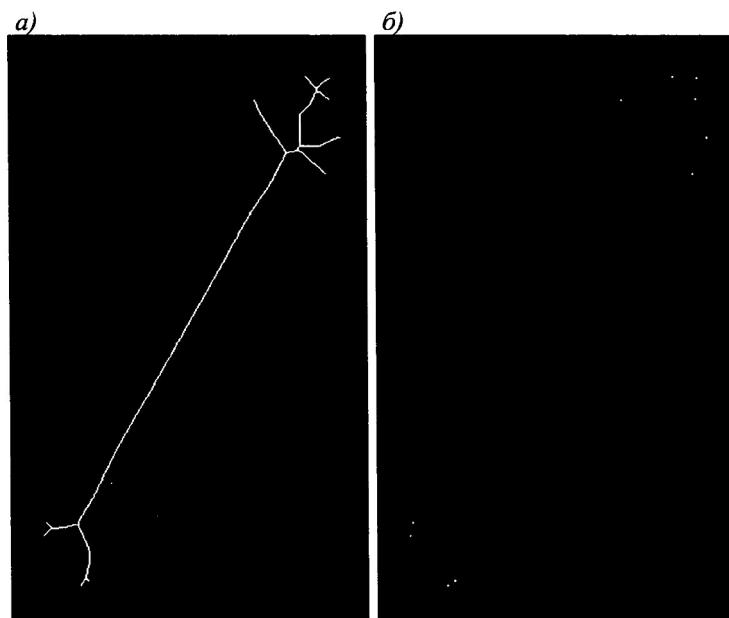


Рисунок 14 - а) Изображение морфологического остива. б) Выход функции endpoints. Пиксели этого изображения увеличены для лучшей наглядности.

Для реализации игры «жизнь» с помощью функций `makelut` и `applylut` мы сначала напишем функцию, которая применяет генетические законы Конвея к выделенному пикселу и его 3×3 соседям:

```
function out = conwaylaws(nhood)
num_neighbors = sum(nhood(:)) - nhood(2, 2);
if nhood(2, 2) == 1
    if num_neighbors <= 1
        out = 0;
    elseif num_neighbors >= 4
        out = 0;
    else
        out = 1;
    end
else
    if num_neighbors == 3
        out = 1;
    else
        out = 0;
    end
end
```

Затем строится поисковая таблица вызовом функции `makelut` с манипулятором функции `conwaylaws` в качестве аргумента:

```
>> lut = makelut ( conwaylaws , 3 );
```

Множество исходных изображений было придумано для демонстрации различных эффектов, возникающих в поколениях «организмов», подчиняющихся законам эволюции Конвея (см. [Gardner, 1970, 1971]). Рассмотрим, к примеру, начальное изображение, которое называется «конфигурацией чеширского кота»:

```
>> bw1 = [0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0
          0 0 0 1 0 0 1 0 0
          0 0 0 1 1 1 1 0 0
          0 0 1 0 0 0 0 1 0
          0 0 1 0 1 1 0 1 0
          0 0 1 0 0 0 0 1 0
          0 0 0 1 1 1 1 0 0
          0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0 0];
```

Следующие команды совершают вычисления и показывают три поколения этого изображения:

```
>> imshow( bw1,'n'), title ('Generation 1')
>> bw2=applylut(bw1,lut);
>> figure, imshow (bw2,'n'); title ('Generation 2')
>> bw3=applylut (bw2,lut);
>> figure, imshow(bw3,'n'); title ('Generation 3')
```

Функция `bwmorph`

Функция `bwmorph` из пакета ITR выполняет ряд полезных действий, построенных на основе операций дилатации, эрозии, а также с использованием поисковых таблиц. Она имеет следующую синтаксическую форму вызова:

$$g = \text{bwmorph}(f, \text{operation}, n),$$

где f – это входное двоичное изображение, `operation` – символьная строка, задающая конкретную операцию, а n обозначает, сколько раз необходимо выполнить эту операцию. Входной аргумент n не является обязательным. При его отсутствии операция выполняется только один раз. В табл. 3 перечислены все допустимые операции функции `bwmorph`.

Таблица 3. Операции, поддерживаемые функцией `bwmorph`

Операция	Описание
bothat	Совершает операцию «дно шляпы» с использованием структурообразующего элемента 3×3 . Для других элементов применять <code>imbothat</code>
bridge	Соединяет пиксели, разделенные промежутком в один пиксел.
clean	Удаляет изолированные пиксели переднего плана
close	Замыкает с использованием элемента 3×3 . Для других структурообразующих элементов применять операцию <code>imclose</code> .
diag	Делает заполнение вокруг диагонально связанных пикселов переднего плана.
dilate	Совершает дилатацию с использованием элемента 3×3 . Для других структурообразующих элементов применять операцию <code>imdilate</code> .
erode	Выполняет эрозию с использованием элемента 3×3 . Для других структурообразующих элементов применять операцию <code>imerode</code>
fill	Заполняет однопиксельные «дырки» (фоновые пиксели, окруженные со всех сторон пикселями переднего плана).
hbreak	Удаляет H-связанные пиксели переднего плана.
open	Делает пикセル p пикселом переднего плана, если не менее 5 пикселов $N8(p)$ являются пикселями переднего плана; в противном случае делает пикセル p фоновым.
majority	Размыкает с использованием элемента 3×3 . Для других структурообразующих элементов применять операцию <code>imopen</code> .
remove	Удаляет «внутренние» пиксели (пиксели переднего плана, у которых нет ни одного фонового соседа).
shrink	Сжимает объекты без внутренних дыр в точки. Сжимает объекты с дырами в кольца.
skel	Строит остов изображения.
spur	Удаляет отростки пикселов.
thicken	Утолщает объекты без соединения несвязанных частей.

thin	Утончает объекты без дыр до минимальной средней линии. Утончает объекты с дырами до колец.
tophat	Совершает операцию «верх шляпы» с использованием структурообразующего элемента 3×3 . Для других элементов применять imtophat

Операция *утончение* сокращает двоичные объекты или формы изображения до отдельных линий, которые имеют толщину всего в один пиксел. Например, попиллярные линии на отпечатках пальцев, показанные на рис. 22 ϵ), являются слишком толстыми. При анализе этих линий может потребоваться их утончение вплоть до толщины в один пиксел. Каждое применение соответствующей операции утончения из **bwmorph** удаляет один или два пикселя из толщины объектов двоичного изображения. Например, следующие команды демонстрируют результат применения операции утончения один и два раза.

```
>> f = imread ('fingerprint_cleaned.tif');
>> g1 = bwmorph (f, 'thin', 1);
>> g2 = bwmorph (f, 'thin', 2);
>> imshow (g1), figure, imshow (g2)
```

На рис. 15 a) и δ), приведены соответствующие утонченные изображения. Имеется важный вопрос: сколько раз следует применять эту операцию? Для ряда действий, включая утончение, функция **bwmorph** допускает для n значение **Inf** (т.е. бесконечность). Вызов **bwmorph** с этим значением предписывает исполнение операции до тех пор, пока изображение не перестанет изменяться. Иногда это называют повторением операции *до стабилизации*. Например,

```
>> ginf = bwmorph (f, 'thin', Inf);
>> imshow (ginf)
```

Как видно из рис. 15, ϵ), это действие приводит к существенному улучшению изображения по сравнению с рис. 9.11, ϵ).



Рисунок 15 - a) Отпечатки пальцев из рис. 11, ϵ) утончены один раз. δ) Утончение изображения произведено два раза. ϵ) Утончение изображения до стабилизации.

Построение остова представляет собой другой путь сокращения объектов двоичного изображения для получения множества тонких линий, которое несет важную информацию о формах исходных объектов. Функция `bwmorph` строит остов, когда параметр `operation` установлен в '`skel`'. Пусть `f` обозначает изображение объекта, похожего на кость, с рис. 16a). Для получения его остова мы вызываем `bwmorph` при `n = Inf`:

```
>> fs = bwmorph ( f , ' skel ' , Inf ) ;
>> imshow ( f ) , figure , imshow ( fs )
```

На рис. 16б) приведен полученный остов, который вполне приемлемо отражает основные формы исходного объекта.

Процедуры построения остова и утончения часто строят лишние короткие отростки, которые иногда называются *паразитическими компонентами*. Процесс подчищения (или удаления) этих компонент называется *усечением*. Для этих целей можно использовать функцию `endpoints` (см. § 9.3.3). Метод заключается в последовательном обнаружении концевых точек и их удаление. Следующие простые команды делают завершающую обработку построенного остова `fs` изображения, которые повторяются 5 раз, для удаления концевых точек:

```
>> for k = 1: 5
    fs = fs & ~ endpoints ( fs ) ;
end
```

Рис. 16в) показывает результат этих действий.

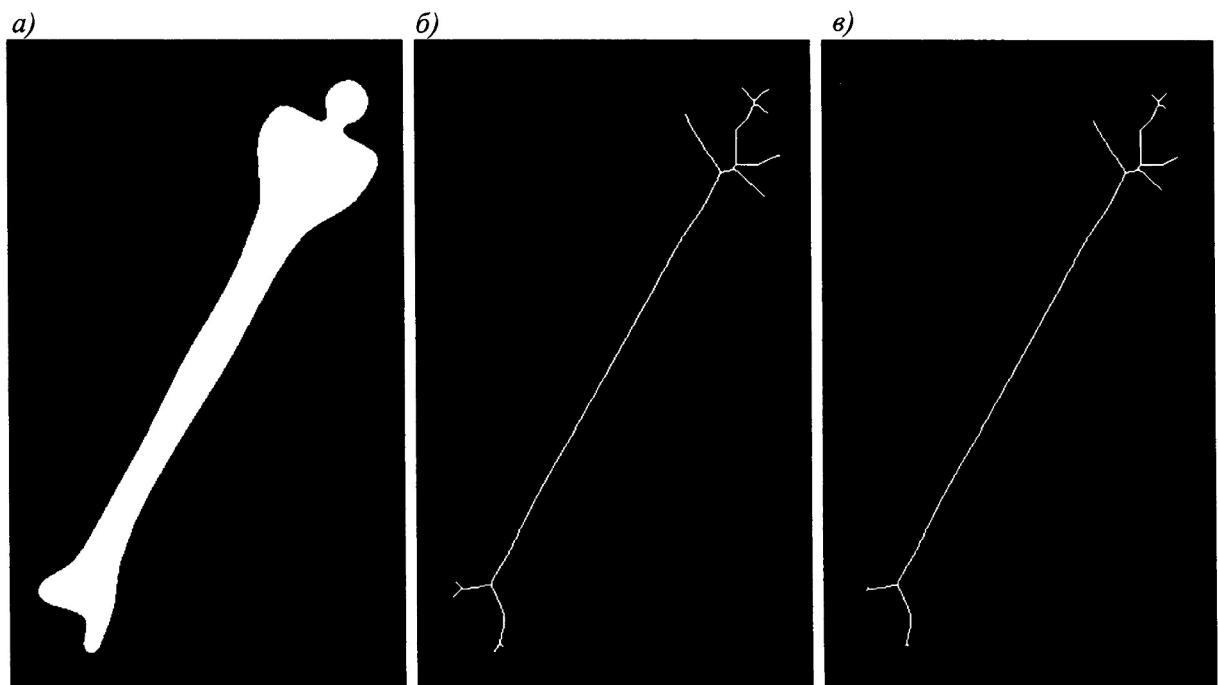


Рисунок 16 - а) Изображение в виде кости. б) Остов, полученный с помощью функции `bwmorph`. в) Окончательный остов после выполнения операции усечения функцией `endpoints`.

Выделение компонент связности

До этого момента все обсуждавшиеся операции применялись индивидуально к пикселям переднего (или заднего) плана или к их ближайшим соседям. В этом параграфе рассматриваются важные операции «среднего плана», которые занимают промежуточное положение между обработкой отдельных пикселов и действиями над всеми пикселями переднего плана. Это приводит к понятию *компонент связности*, которые мы будем также называть *объектами*.

Если предложить сосчитать объекты на рис. 17 a), то большинство людей скажет, что их там десять: шесть букв и четыре геометрические фигуры. На рис. 17 b) дан увеличенный прямоугольный фрагмент этого изображения. Какое отношение имеют 16 пикселов переднего плана на этом рисунке к десяти выделенным объектам? Несмотря на то, что они разделены на две группы, все эти 16 пикселов принадлежат букве “E” на рис. 17 a). Чтобы написать компьютерную программу, которая обнаруживает объекты и оперирует с ними, нам необходимо более точно определить ключевые понятия.

Пиксель p с координатами (x, y) имеет два горизонтальных и два вертикальных соседа с координатами $(x+1, y)$, $(x - 1, y)$, $(x, y+1)$ и $(x, y - 1)$. Эта четверка соседей p обозначается $N_4(p)$. Она выделена на рис. 18 a) серым цветом. Четыре диагональных соседа p имеют координаты $(x+1, y+1)$, $(x+1, y-1)$, $(x - 1, y+1)$ и $(x - 1, y-1)$. На рис. 18 b) изображены эти диагональные соседи; эта четверка обозначается $N_d(p)$. Объединение $N_4(p)$ и $N_d(p)$ на рис. 18 c) определяет *восьмерку соседей* пикселя p , которая обозначается $N_8(p)$.

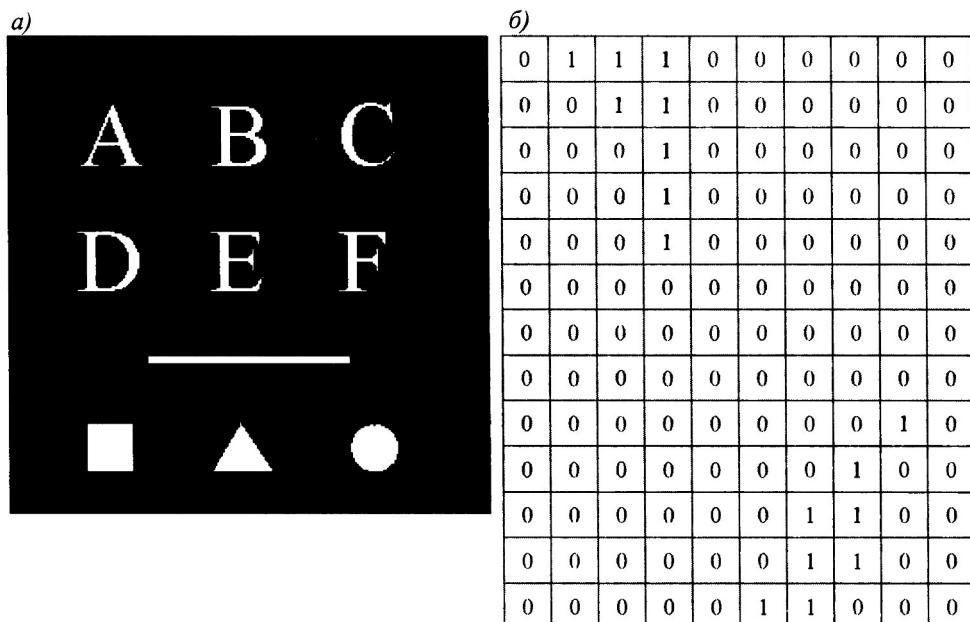


Рисунок 17 - а) Изображение, содержащее 10 объектов. б) Подмножество пикселов изображения.

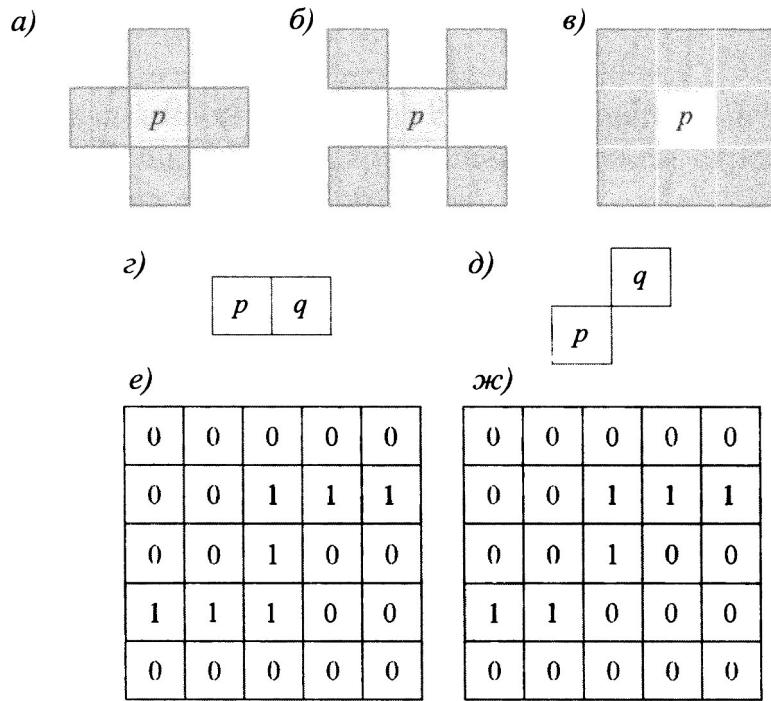


Рисунок 18 – а) Пиксель p и его четверка соседей $N_4(p)$. б) Пиксель p и его диагональные соседи $N_D(p)$. в) Пиксель p и его восьмерка соседей $N_8(p)$. г)

Пиксели p и q являются 4-смежными и 8-смежными. д) Пиксели p и q являются 8-смежными, но не 4-смежными. е) Затемненные пиксели являются одновременно 4-связными и 8-связными. ж) Затемненные пиксели являются 8-связными, но не 4-связными.

Два пикселя p и q называются *4-смежными*, если $q \in N_4(p)$. Аналогично, пиксели p и q называются *8-смежными*, если $q \in N_8(p)$. Рис. 18г) и д), иллюстрирует эти понятия. Путем между пикселями p_1 и p_n называется последовательность пикселов p_1, p_2, \dots, p_n , такая, что p_k является смежным для p_{k+1} при $k=1, 2, \dots, n-1$. Пусть может быть *4-связным* или *8-связным*, в зависимости от используемого определения смежности пикселов.

Два пикселя переднего плана p и q называются *4-связными*, если существует 4-связный путь между ними, который целиком состоит из пикселов переднего плана [рис. 18е)]. Пиксели p и q называются *8-связными*, если между ними имеется 8-связный путь [см. рис. 18ж)]. Для любого пикселя переднего плана p множество всех связанных с ним пикселов переднего плана называется *компонентой связности*, содержащий p .

Понятие *компоненты связности* было определено в терминах путей, а понятие пути, в свою очередь, зависит от определения смежности. Это означает, что природа связных компонент зависит от понятия смежности, из которых наиболее употребительным являются 4-связность и 8-связность. Рис. 19 иллюстрирует влияние, которое определение смежности может произвести при подсчете числа компонент связности изображения. На рис. 19а) приведено маленькое двоичное изображение с четырьмя

4-компонентами связности. Рис. 19б) показывает, что выбор 8-смежности сокращает число компонент 8-связности до 2.

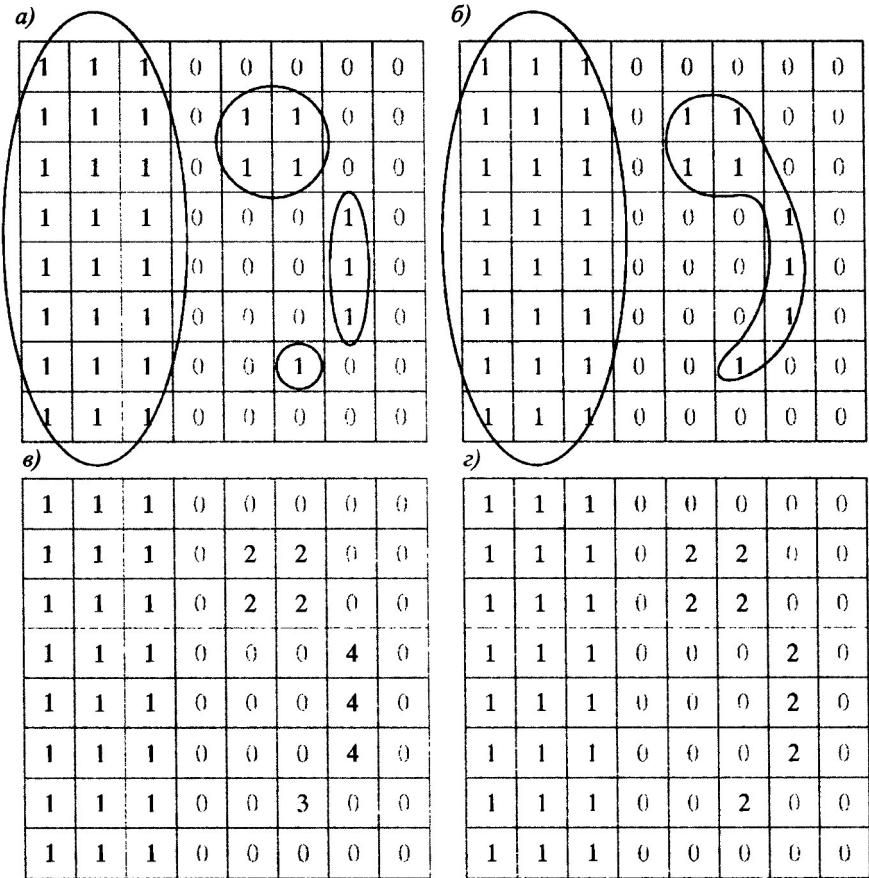


Рисунок 19 – Компоненты связности. а) Четыре 4-связные компоненты. б) Две 8-связные компоненты. в) Размечающая матрица, полученная для 4-связности. г) Размечающая матрица, полученная для 8-связности

Функция `bwlabel` из ITP находит все компоненты связности двоичного изображения. Форма ее вызова имеет вид

$$[L, num] = \text{bwlabel}(f, conn),$$

где f – это входное двоичное изображение, а параметр $conn$ обозначает желаемую связность (по 4- или 8-смежности). Выход L называется *размечающей матрицей*, а в (необязательный) выходной параметр num записывается общее число обнаруженных компонент связности. Если параметр $conn$ опущен, то он по умолчанию считается равным 8. На рис. 19, в) показана размечающая матрица, отвечающая рис. 19а) и вычисленная командой `bwlabel(f, 4)`. Пикселам каждой отдельной компоненты связности присваивается одинаковое число от 1 и до общего числа компонент связности. Другими словами, пиксели, помеченные 1, принадлежат первой компоненте связности, пиксели, помеченные 2, - второй компоненте, и т.д. Фоновые пиксели помечены 0. На рис. 19г) показана размечающая матрица, соответствующая рис. 19а), которая построена командой

$$\text{bwlabel}(f, 8).$$

Пример 7. Нахождение центров масс компонент связности.

В этом примере показано, как можно найти центр масс каждой компоненты связности рис. 17a). Прежде всего, мы воспользуемся функцией `bwlabel` для вычисления 8-связных компонент:

```
>> f = imread ('objects.tif');
>> [ L , n ] = bwlabel ( f );
```

Функция `find` может быть полезной при работе с размечивающими матрицами. Например, следующий вызов функции `find` возвращает индексы строк и столбцов всех пикселов, принадлежащих третьему объекту:

```
>> [ r , c ] = find ( L == 3 );
```

Функция `mean` с выходами `r` и `c` вычисляет центр масс этого объекта.

```
>> rbar = mean ( r );
>> cbar = mean ( c );
```

Для нахождения центров масс всех объектов изображения можно воспользоваться циклом. Чтобы сделать центры масс объектов видимыми на изображении, мы их обозначим символом «*» белого цвета, который разместим в центр черного кружка. Для этого выполним следующую серию команд:

```
>> imshow ( f )
>> hold on
>> for k = 1: n
    [ r , c ] = find ( L == k );
    rbar = mean ( r );
    cbar = mean ( c );
    plot ( cbar , rbar , 'Marker' , 'o' , 'MarkerEdgeColor' , 'k' , ...
            'MarkerFaceColor' , 'k' , 'MarkerSize' , 10 )
    plot ( cbar , rbar , 'Marker' , '*' , 'MarkerEdgeColor' , 'w' )
end
```

Рис. 20 показывает результат этого программного кода.

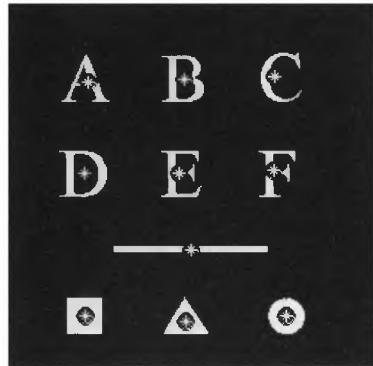


Рисунок 20 – Центры масс (белые звездочки), наложенные на соответствующие компоненты связности

Морфологическая реконструкция

Реконструкция является морфологическим преобразованием, в котором участвуют два изображения и один структурообразующий элемент (вместо одного изображения и одного структурообразующего элемента, как это было в предыдущих параграфах). Одно изображение, которое называется *маркером*, является исходной точкой преобразования. Другое изображение, *маска*, накладывает определенные ограничения (связи) на отображение. Используемый структурообразующий элемент определяет связность. В этом параграфе мы используем 8-связность (задаваемую по умолчанию), которая означает, что матрица B_1 используемая далее, есть 3×3 матрица из единиц, центр которой находится в точке с координатами (2, 2).

Если g – это маска, а f – маркер, то реконструкция g по f , которая обозначается $Rg(f)$, определяется следующей итеративной процедурой.

1. Инициализация: присвоить h_1 маркерное изображение f .
2. Построить структурообразующий элемент $B = \text{ones}(3)$.
3. Повторять:

$$h_{k+1} = (h_k \oplus B) \cap g$$

до тех пор, пока не станет $h_{k+1} = h_k$.

Маркер f должен быть подмножеством g т.е.

$$f \subseteq g.$$

Рис. 21 иллюстрирует эту итеративную процедуру. Заметим, что, несмотря на простоту и удобство итеративной формулировки процесса реконструкции, существуют другие, более быстрые вычислительные алгоритмы, реализующие это преобразование. Функция `imreconstruct` из пакета IPT выполняет «быструю гибридную реконструкцию». Эта функция имеет следующий синтаксис:

`out = imreconstruct(marker, mask),`

где `marker` и `mask` – это изображения, определенные в начале параграфа.

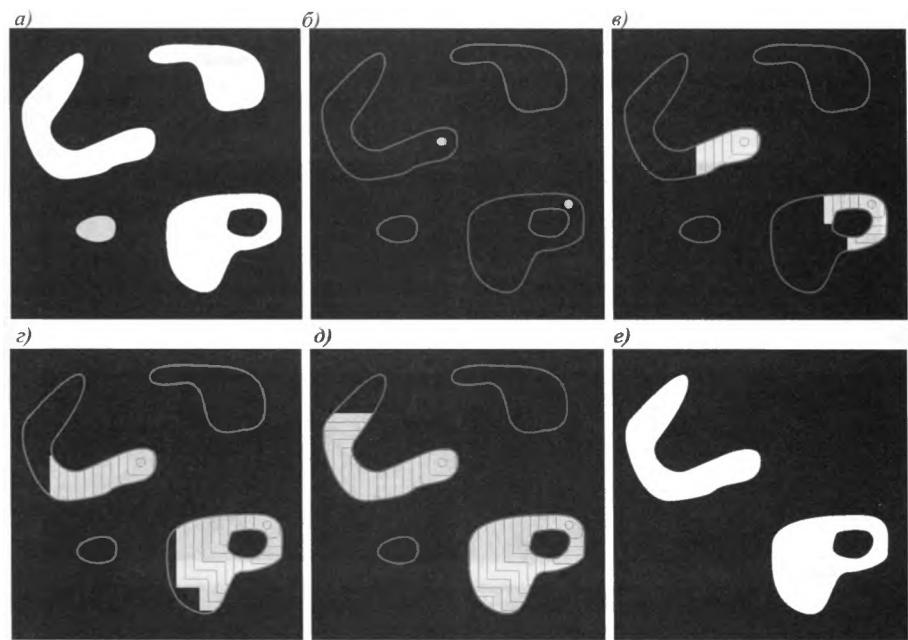


Рисунок 21 – Морфологическая реконструкция. *а)* Исходное изображение (маска). *б)* Маркерное изображение, *в) – д)* Промежуточные результаты после 100, 200 и 300 итераций, соответственно. *е)* Окончательный результат.
[Выделенные объекты маски наложены на рис. *б) – д)* для сравнения]

Размыкание реконструкцией

При морфологическом замыкании операции эрозия удаляет малые объекты, а последующая дилатация частично восстанавливает форму оставшихся объектов. Однако точность этого восстановления зависит от степени подобия форм объектов и структурообразующего элемента. Метод *размыкания реконструкцией*, обсуждающийся в этом параграфе, в частности восстанавливает форму объектов, которые остались после эрозии. Размыкание реконструкцией f с использованием структурообразующего элемента B определяется формулой $R_f(f \Theta B)$.

Пример 8. Размыкание реконструкцией.

Сравнение результатов морфологического размыкания и размыкание реконструкцией для изображения, на котором имеется фрагмент текста, показано на рис. 22. В этом примере мы хотим извлечь из рис. 22*a)* символы, у которых имеются длинные вертикальные черточки. Поскольку для размыкания реконструкцией требуется изображение, подвергнутое эрозии, мы выполним эту операцию на первом шаге, используя тонкий, вертикальный структурообразующий элемент, длина которого пропорциональна высоте букв:

```
>> f = imread( 'book_text_bw.tif' );
>> fe = imerode( f, ones( 51, 1 ) );
```

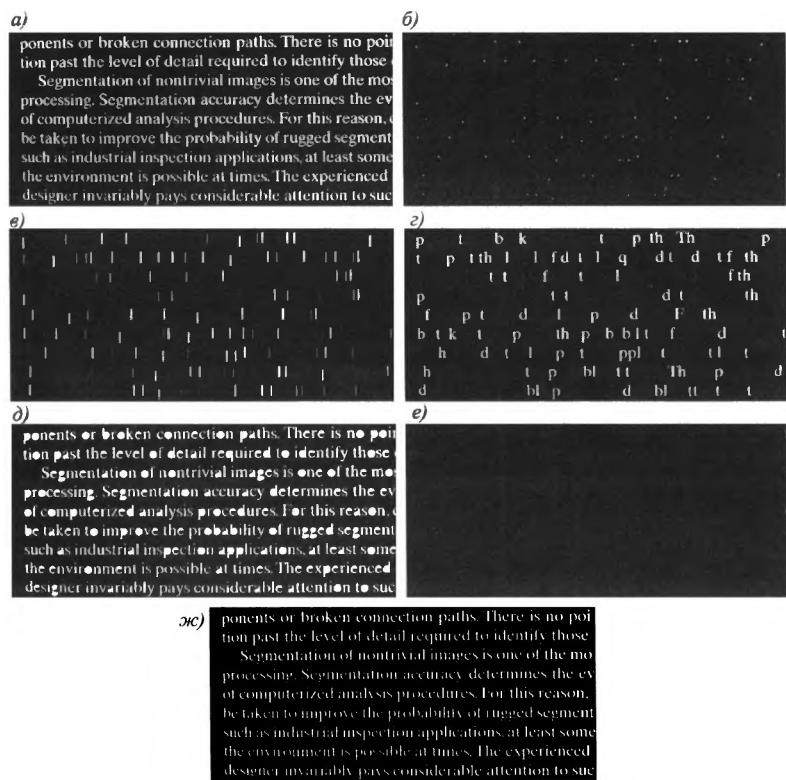


Рисунок 22 – Морфологическая реконструкция. а) Исходное изображение. б) Эрозия по вертикальной черточке. в) Размыкание по вертикальной черточке. г) Размыкание реконструкцией по вертикальной черточке. д) Заполнение дыр. е) Символы, соприкасающиеся с границей (см. правую границу). ж) Пограничные символы удалены.

На рис. 22б) приведен результат. Размыкание, показанное на рис. 22в), было выполнено функцией `imopen`:

```
>> fo = imopen(f, ones(51, 1));
```

Отметим, что вертикальные черточки были обнаружены, но содержащие их символы целиком восстановлены не были. Теперь выполняем размыкание реконструкцией по формуле

```
>> foBr = imreconstruct(fe, f);
```

Из результата, представленного на рис. 22г), видно, что символы, содержащие вертикальные черточки, были полностью восстановлены, а все остальные символы – удалены. Оставшиеся части рис. 22 иллюстрируют материалы следующих двух параграфов.

Заполнение отверстий

Морфологическая реконструкция имеет широкий спектр практических приложений, которые определяются выбором маркера и маски. Например, пусть в качестве маркерного изображения взято изображение f_m , которое равно 0 везде, кроме границ, а в точках границ оно равно $1 - f$:

$$f_m(x, y) = \begin{cases} 1 - f(x, y), & \text{если } (x, y) \text{ лежит на границе,} \\ 0 & \text{иначе.} \end{cases}$$

Тогда операция $g = [R_{fc}(f_m)]^c$ имеет эффект заполнения отверстий в f , как это проиллюстрировано на рис. 22д). Функция `imfill` выполняет эти действия автоматически при задании необязательного аргумента ‘holes’ в команде

$$g = \text{imfill}(f, \text{'holes'}).$$

Очистка от пограничных объектов

Другое полезное приложение реконструкции заключается в очистке изображений от объектов, которые соприкасаются с границей. Опять ключевой момент состоит в выборе подходящих маркера и маски для достижения желаемого результата. В данном случае в качестве маски используется исходное изображение, а маркерное изображение f_m задается формулой

$$fm(x, y) = \begin{cases} f(x, y), & \text{если } (x, y) \text{ лежит на границе,} \\ 0 & \text{иначе.} \end{cases}$$

Рис. 22е) показывает результат реконструкции $R_f(f_m)$, на котором остались лишь объекты, касающиеся границы. Разность множеств $f \setminus R_f(f_m)$, приведенная на рис. 22ж), состоит из объектов исходного изображения, которые не касаются границы. Функция IPT `imclearborder` автоматически совершает эту процедуру. Она имеет форму вызова

$$g = \text{imclearborder}(f, \text{conn}),$$

где f – это входное изображение, а g – результат. Параметр `conn` может быть равным 4 или 8 (по умолчанию). Эта функция «гасит» структуры, которые ярче своих окрестностей и которые примыкают к границам изображения. Изображение f может быть полутонаовым или двоичным. Выходом служит или полутонаовое, или двоичное изображение соответственно.

Полутоновая морфология

Все двоичные морфологические операции, обсуждающиеся в этой главе, за исключением преобразований «успех/неудача», имеют естественные расширения для обработки полутоновых изображений. В этом параграфе, как и в двоичном случае, мы начнем с рассмотрения операций дилатации и эрозии, которые для полутоновых изображений формулируются в терминах минимума и максимума, взятых по окрестностям пикселов.

Дилатация и эрозия

Полутоновая дилатация изображения f по структурообразующему элементу b обозначается $f^{\oplus} b$ и определяется формулой

$$(f^{\oplus} b)(x, y) = \max \{f(x - x', y - y') + b(x', y') \mid (x', y') \in D_b\}.$$

где D_b обозначает область определения b , и предполагается, что $f(x, y)$ равно $-\infty$ вне области определения f . Это уравнение реализует процесс, аналогичный процедуре нахождения пространственной свертки. Для

наглядности можно мысленно повернуть структурообразующий элемент на 180° вокруг его центра и разнести по всем точкам (x, y) изображения. В каждом положении нужно сложить соответствующие пиксели изображения с пикселями повернутого и перемещенного элемента, а затем вычислить максимум всех этих величин. Получится значение дилатации в точке (x, y) .

Важное отличие полутоновой дилатации от пространственной свертки состоит в том, что двоичная матрица D_b определяет, какие точки из окрестности использовать в операции взятия максимума. Другими словами, для любой пары координат (x_0, y_0) из области D_b сумма величин $f(x - x_0, y - y_0) + b(x_0, y_0)$ участвует в вычислении максимума только в случае, если значение матрицы D_b в ячейке (x_0, y_0) равно 1. В противном случае, т.е. когда этот элемент матрицы равен нулю, соответствующая сумма не участвует в операции взятия максимума. Это действие повторяется для всех координат $(x', y') \in D_b$, каждый раз, как только меняется координата (x, y) . Построение графика величины $b(x', y')$, как функции координат x' и y' , будет выглядеть как цифровая «поверхность», высота которой в каждой точке задается значением изображения b в этой точке.

На практике полутоновая дилатация обычно выполняется с использованием *плоских* структурообразующих элементов (см. табл. 2), для которых величина (высота) b равна 0 в каждой точке, принадлежащей D_b . То есть

$$b(x', y') = 0 \text{ при } (x', y') \in D_b.$$

В этом случае операция взятия максимума полностью определяется конфигурацией 0 и 1 двоичной матрицы D_b , а уравнения для полутоновой дилатации принимают более простой вид

$$(f \oplus b)(x, y) = \max \{f(x - x', y - y') \mid (x', y') \in D_b\}.$$

Таким образом, полутоновая дилатация является операцией взятия локального максимума, в которой максимум вычисляется по множеству пикселов окрестности, форма которой задается областью D_b .

Неплоские структурообразующие элементы можно строить с помощью функции `strel`, передавая ей две матрицы:

(1) одну матрицу из 0 и 1, которые обозначают область структурообразующего элемента D_b и

(2) другую матрицу, в которой записаны значения высот $b(x', y')$.

Например, команд

```
>> b = strel ([1 1 1], [1 2 1])
b =
Nonflat STREL object containing 3 neighbors.
Neighborhood:
 1 1 1
Neight:
 1 2 1
```

создает структурообразующий элемент 1×3 , значения высот которого равны $b(0, -1) = 1$, $b(0, 0) = b(0, 1) = 1$.

Плоские структурообразующие элементы для полутонаовых изображений строятся с использованием функции `strel` по тем же правилам, что и для двоичных изображений. Например, следующие команды показывают, как совершать дилатацию изображения f на рис. 9.23, *a*) с помощью плоского 3×3 элемента:

```
>> se = strel('square', 3);
>> gd = imdilate(f, se);
```

На рис. 23*b*) показан результат. Как и следовало ожидать, изображение получилось слегка размытым. Остальные части рис. 22 обсуждаются далее.

Полутонаовая эрозия f по структурообразующему элементу b обозначается

$f^\Theta b$ и определяется формулой

$$(f^\Theta b)(x, y) = \min \{f(x + x', y + y') - b(x', y') \mid (x', y') \in D_b\},$$

где D_b обозначает область определения b и предполагается, что $f(x, y)$ равно $+\infty$ вне области определения f . Опять эту процедуру можно мысленно представлять в виде перемещения центра структурообразующего элемента во все точки изображения. В каждой точке изображения значения структурообразующего элемента вычитаются из соответствующих значений пикселов изображения и берется минимум по всем таким разностям.

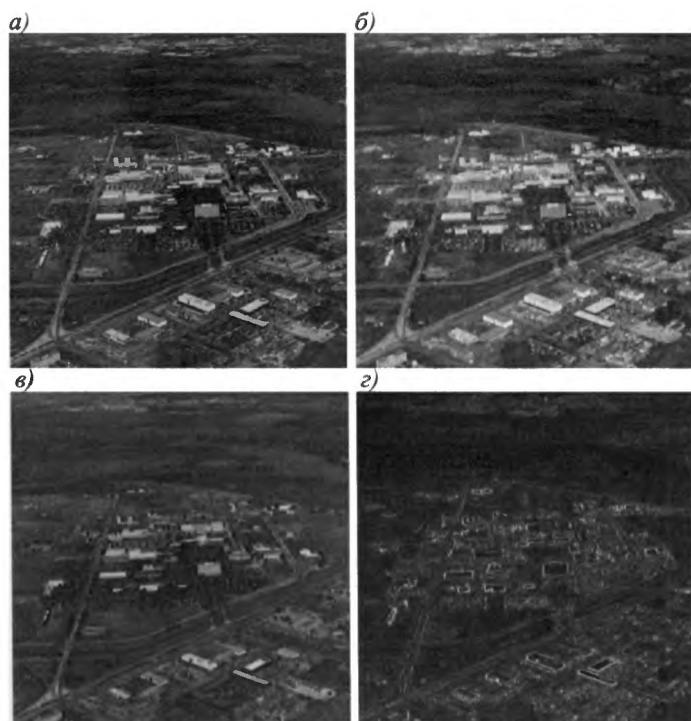


Рисунок 23 – Дилатация и эрозия. *а*) Исходное изображение. *б*) Изображение после дилатации. *в*) Изображение после эрозии. *г*) Морфологический градиент

Как и с дилатацией, полутоновая эрозия чаще всего применяется по плоским структурообразующим элементам. В этом случае уравнения для эрозии можно упростить и привести их к виду

$$(f \oplus b)(x, y) = \min \{f(x + x', y + y') \mid (x', y') \in Db\}.$$

Значит, полутоновая эрозия является операцией локального минимума, в которой минимум берется по множеству пикселов окрестности, форма которой задается по Db . Рис. 23в) изображает результат применения функции `imerode` с тем же структурообразующим элементом, что и на рис. 23б):

```
>> ge = imerode ( f, se );
```

Дилатацию и эрозию можно сочетать для достижения различных эффектов. К примеру, вычитания результатов эрозии изображения из результата дилатации задает «морфологический градиент», который представляет собой меру локальной полутоновой вариации изображения. Например, присвоив

```
>> morph_grad = imsubtract ( gd, ge );
```

получаем изображение на рис. 23а). На этом изображении подчеркнуто выделены граничные характеристики, подобно тому, как это делается операцией взятия градиента.

Размыкание и замыкание

Формулы размыкания и замыкания для полутоновых изображений имеют такой же вид, что и для двоичных изображений. Размыкание изображения f по элементу b обозначается $f^o b$ и определяется выражением

$$f^o b = (f^\Theta b) \oplus b.$$

Как и ранее, сначала выполняется эрозия f по b , за которой следует дилатация по b . Аналогично, замыкание f по b , обозначаемое $f \bullet b$, есть дилатация, после которой совершается эрозия:

$$f \bullet b = (f^\Theta b) \oplus b.$$

Обе эти операции имеют простую геометрическую интерпретацию. Представим изображение $f(x, y)$ в виде трехмерной поверхности, т.е. значения яркости представляются высотами над координатной плоскостью xy . Тогда процесс размыкания f по b представляет собой перемещение (перекатывание) структурообразующего элемента строго под трехмерной поверхностью изображения f , причем поверхность размыкания получается как огибающая высшие точки, которые достигают этот элемент.

Рис. 24 иллюстрирует эту процедуру в одномерном случае. Рассмотрим кривую на рис. 24а), которая представляет собой значения высот (яркости) вдоль какой-то строки изображения. На рис. 24б) показано несколько положений плоского структурообразующего элемента при перемещении под кривой. Результат замыкания дан на рис. 24в) жирной линией над серой областью. Поскольку структурообразующий элемент слишком велик, чтобы поместиться внутри узкого пика, расположенного посередине исходного изображения, это пик срезается при размыкании. В общем случае процедура размыкания используется для удаления узких всплесков яркости при

сохранении среднего полутонового фона и широких областей изменения яркости относительно неизменными.

Рис. 24 ε) иллюстрирует процесс замыкания. Теперь структурообразующий элемент перемещается сверху по поверхности кривой яркости. Результат замыкания, показанный на рис. 24 δ), получается нахождением всех самых нижних точек, которые достигаются элементом при движении по кривой. Видно, что замыкание подавляет более темные малые колебания яркости, размеры которых меньше размеров элемента.

Пример 9. Морфологическое сглаживание с помощью размыкания и замыкания.

Поскольку размыкание удаляет яркие детали, которых меньше, чем структурообразующих элементов, а замыкания, наоборот, подавляют аналогичные темные детали малых размеров, то эти процедуры широко используются в различных сочетаниях при сглаживании и удалении постороннего шума. В этом примере мы применяем функции `imopen` и `imclose` для сглаживания изображения деревянных шпонок на рис. 25 a):

```
>> f = imread ('plugs.jpg');
>> se = strel ('disk', 5);
>> fo = imopen (f, se);
>> foc = imclose (fo, se);
```

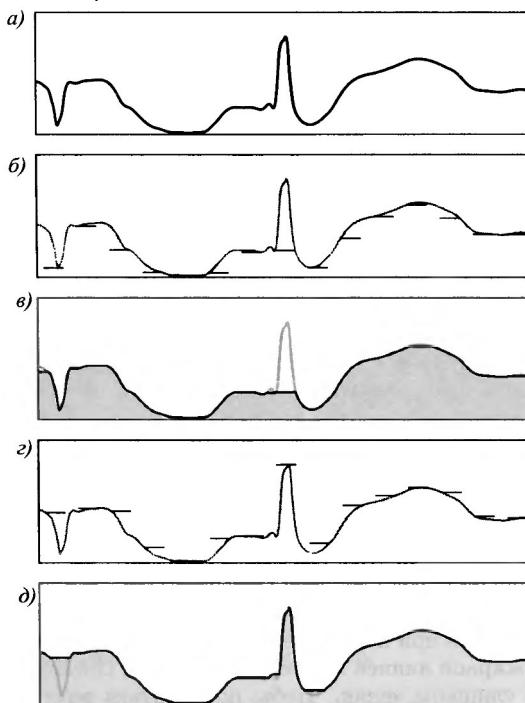


Рисунок 24 – Размыкание и замыкание в одном измерении. *а)* График исходного одномерного сигнала. *б)* Плоский структурообразующий элемент, пододвинутый снизу к графику сигнала в ряде точек. *в)* Результат размыкания. *г)* Плоский структурообразующий элемент, пододвинутый сверху к графику сигнала в ряде точек. *д)* Результат замыкания

На рис. 25б) показано разомкнутое изображение f_0 , а на рис. 25в) – замкнутое изображение f_{oc} . Обратите внимание на сглаживания фона и деталей объектов изображения. Эту процедуру часто называют *фильтрацией размыканием-замыканием*. *Фильтрация размыканием-замыканием* приводит к похожему результату.

Другой путь комбинирования размыкания и замыкания состоит в *альтернативной последовательной фильтрации*. Одна из форм альтернативной последовательной фильтрации заключается в серии фильтраций размыканием-замыканием рядом структурообразующих элементов с увеличивающимися размерами. Следующая последовательность команд иллюстрирует этот процесс, который начинается с малого структурообразующего элемента с последующим увеличением его размера до тех пор, пока он не сравняется с элементом, использованным при построении рис. 25б) и в):

```
>> fasf = f ;
>> for k = 2:5
    se = strel ('disk', k) ;
    fasf = imclose ( imopen ( fasf , se ) , se ) ;
end
```

Результат приведен на рис. 25г). Он является более гладким по сравнению с результатом простой фильтрации размыканием-замыканием, но платой служат дополнительные вычислительные затраты.

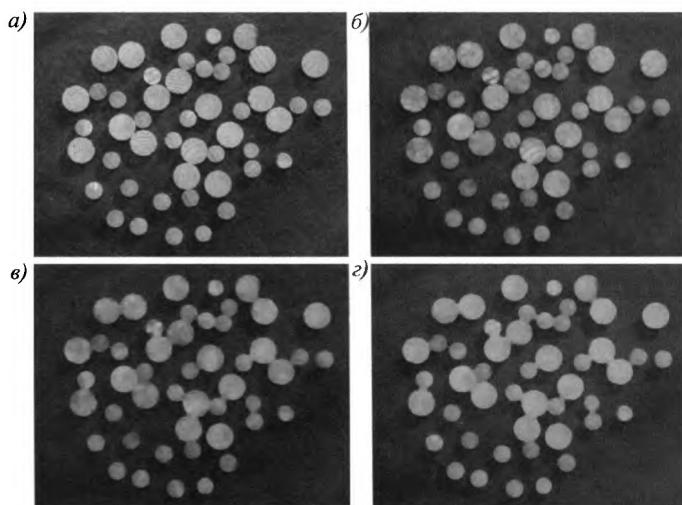


Рисунок 25 – Сглаживание при помощи размыкания и замыкания. а) Исходное изображение деревянных шпонок. б) Изображение, разомкнутое с помощью круга радиуса 5. в) Замыкание результата размыкания. г) Результат альтернативной последовательной фильтрации

Пример 10. Применение преобразования «верх шляпы».

Размыкание можно использовать для компенсации неравномерной яркости фона изображения. На рис. 26а) дано изображение f зерен риса, на котором фон в нижней части темнее, чем в верхней части изображения.

Неравномерный фон создает трудности при пороговой обработке изображений. Например, на рис. 25б) приведен результат пороговой обработки, на которой зерна из верхней части хорошо отделены от фона, а из нижней части – плохо. Размыкание изображения может дать правильное приближение фона, если структурообразующий элемент достаточно велик, чтобы целиком поместиться внутри каждого рисового зернышка. Например, командами

```
>> se = strel ('disk', 10);
>> fo = imopen (f, se);
```

совершается размыкание изображения на рис. 26в). Если вычесть это изображение из исходного, то получится изображение зерен на достаточно однородном фоне:

```
>> f2 = imsubtract (f, fo);
```

Рис. 26г) показывает результат этого действия, а на рис. 26д) приведено новое изображение после пороговой обработки. Налицо эффект улучшения.

Вычитание разомкнутого изображения из исходного называется преобразованием «верх шляпы». Функция IPT `imtophat` выполняет эти действия за один шаг:

```
>> f2 = imtophat (f, se);
```

Функцию `imtophat` можно вызывать командой `g = imtophat (f, NHOOD)`, где `NHOOD` – это массив из нулей и единиц, которые обозначают размер и форму используемого структурообразующего элемента. Этот синтаксис схож с вызовом этой функции в виде `imtophat (f, strel (NHOOD))`.

Сопряженная функция `imtophat` совершает преобразование «дно шляпы», которое по определению есть замыкание изображения минус само изображение. Ее форма вызова подобна синтаксису функции `imtophat`. Эти функции можно употреблять в сочетании друг с другом, например, для усиления контрастности командами вида

```
>> se = strel ('disk', 3);
>> g = imsubtract (imadd (f, imtophat (f, se)), imtophat (f, se));†
```

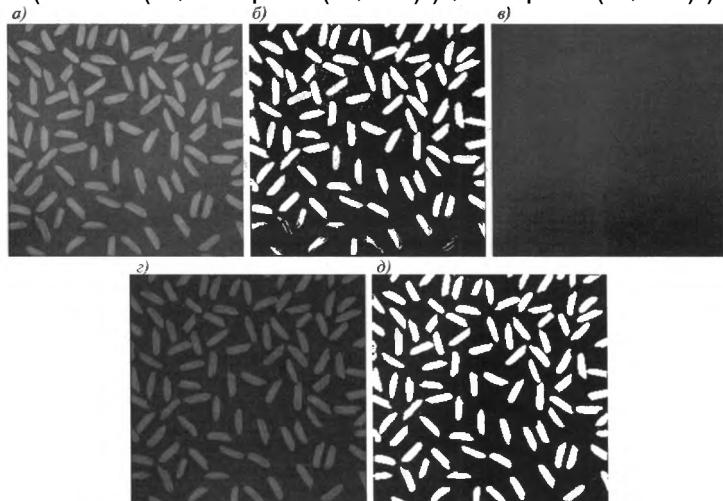


Рисунок 26 – Преобразование «верх шляпы». а) Исходное изображение. б) Изображение после пороговой обработки. в) Разомкнутое изображение. г) Преобразование «верх шляпы». д) «Верх шляпы» после пороговой обработки

Пример 11. Гранулометрия.

Методы определения размеров рассыпанных объектов на изображении составляют важную часть *гранулометрии*. Для нахождения размеров объектов можно воспользоваться неявным морфологическим подходом, т.е. без измерения размеров каждого отдельного объекта. Для объектов с регулярными формами, которые ярче, чем фон изображения, метод заключается в применении морфологического размыкания с увеличивающимися размерами. Для каждого акта размыкания вычисляется сумма всех значений пикселов. Эта сумма иногда называется *площадью поверхности* изображения. Следующие команды реализуют размыкание с помощью (структурообразующего элемента) круга, радиус которого меняется от 0 до 35, применительно к изображению 25a):

```
>> f = imread ( 'plugs.jpg' );
>> sumpixels = zeros ( 1 , 36 ) ;
>> for k = 0:35
    se = strel ( 'disk' , k ) ;
    fo = imopen ( f , se ) ;
    sumpixels ( k + 1 ) = sum ( fo (:)) ;
end
>> plot ( 0:35 , sumpixels ) , zlabel ( 'k' ) , ylabel ( 'Surface area' )
```

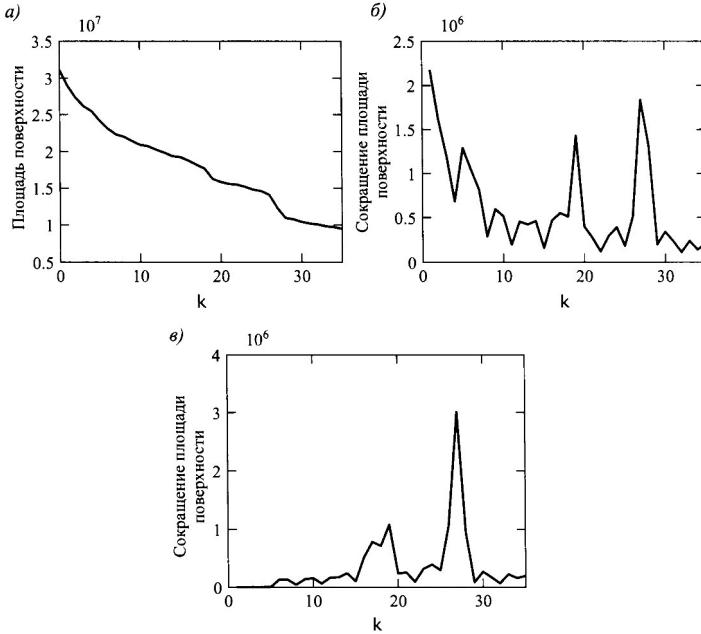


Рисунок 27 – Гранулометрия. а) Площадь поверхности как функция радиуса структурообразующего элемента. б) Уменьшение площади поверхности как функция радиуса структурообразующего элемента. в) Уменьшение площади поверхности как функция радиуса для сглаженного изображения

На рис. 27a) приведен график зависимости `sumpixels` от радиуса `k`. Более наглядным является график уменьшения площади поверхности между последовательными размыканиями:

```
>> plot ( -diff ( sumpixels ) )
>> xlabel ( 'k' )
```

```
>> ylabel('Surface area reduction')
```

Пики на рис. 27б) говорят о присутствии большого числа объектов, имеющих этот радиус. Поскольку исходное изображение является достаточно зашумленным, мы повторили эту процедуру применительно к его сглаженной версии на рис. 25г). Результат построен на рис. 27в). Теперь имеется достаточно оснований, чтобы сделать вывод о присутствии двух характерных размеров объектов на исходном изображении.†

Реконструкция

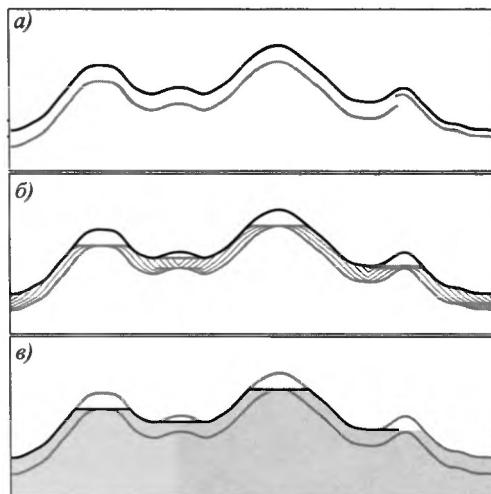


Рисунок 28 – Полутоновая морфологическая реконструкция в случае одного измерения. а) Кривые маски (сверху) и маркера. б) Итеративное вычисление при реконструкции. в) Результат реконструкции (черная кривая)

Морфологическая реконструкция полутоновых изображений определяется с помощью итеративной процедуры. Рис. 28 показывает, как это преобразование работает в одномерном случае. Верхняя кривая на рис. 28а) является маской, а нижняя (серая) кривая служит маркером. В этом примере маркер построен простым вычитанием некоторой константы из маски, однако в общем случае любой сигнал может служить маркером при условии, что ни одно из его значений не превосходит соответствующей величины маски. Каждая итерация процедуры реконструкции размазывает пики на кривой-маркерере, пока они усиливаются снизу кривой-маской [см. рис. 28б)].

Окончательный результат реконструкции показан на рис. 28в). Заметьте, что два малых пика исчезли на реконструкции, в то время как два высоких остались, но они стали ниже. Если маркерное изображение получается путем вычитания константы h из изображения-маски, то такая реконструкция называется *преобразованием h-минимума*. Преобразование h -минимума выполняется в IPT функцией *imhmin*. Она позволяет удалять малые локальные пики.

Другая полезная техника реконструкции выполняется процедурой *размыкание реконструкцией*, при которой исходное изображение сначала подвергается эрозии, как при обычном морфологическом размыкании. Затем,

однако, вместо того, чтобы применять замыкание, полученное изображение используется в качестве маркера при выполнении реконструкции. А в качестве маски используется само исходное изображение. На рис. 9.29, а) показан пример размыкания реконструкцией, полученный после выполнения команд

```
>> f = imread ('plugs.jpg');  
>> se = strel ('disk', 5);  
>> fe = imerode (f, se);  
>> fobr = imreconstruct (fe, f);
```



Рисунок 29 – а) Размыкание реконструкцией. б) Размыкание реконструкцией, после которого выполняется замыкание реконструкцией.

Реконструкцию можно применять для дальнейшей подчистки изображения `fobr`, применяя так называемую технику **замыкания реконструкцией**. Замыкание реконструкцией состоит из взятия дополнительного изображения, выполнения размыкания реконструкцией и вычисления вновь дополнения. Эти шаги совершаются командами:

```
>> fobrc = imcomplement (fobr);  
>> fobrce = imerode (fobrc, se);  
>> fobrcbr = imcomplement (imreconstruct (fobrce, fobrc));
```

На рис. 29б) показан результат размыкания реконструкцией, за которым применено замыкание реконструкцией. Сравните его с результатами применения фильтра размыкания-замыкания и альтернативного последовательного фильтра на рис. 25.

Пример 12. Использование реконструкции для удаления сложного фона изображения.

Задача полутоновой реконструкции заключается в выделении текстовых символов на изображении кнопок калькулятора, приведенного на рис. 30а). Первый шаг состоит в удалении линий горизонтальных отблесков наверху каждой кнопки. Для этого мы воспользуемся тем фактом, что эти отблески длиннее, чем любой текстовый символ изображения. Мы выполняем размыкание реконструкцией, используя структурообразующий элемент, представляющий длинный горизонтальный отрезок:

```

>> f = imread ('calculator.jpg');
>> f_обр = imreconstruct (imerode (f, ones (1, 71)), f);
>> f_о = imopen (f, ones (1, 71));

```

Результат размыкания реконструкцией показан на рис. 30б). Для сравнения на рис. 30в) приведен результат стандартного размыкания (f_o). Размыкание реконструкцией лучше извлекает фон между соседними кнопками по горизонтали. Процесс вычитания результата размыкания реконструкцией из исходного изображения называется преобразованием «верх шляпы» реконструкцией, и он показан на рис. 30г)

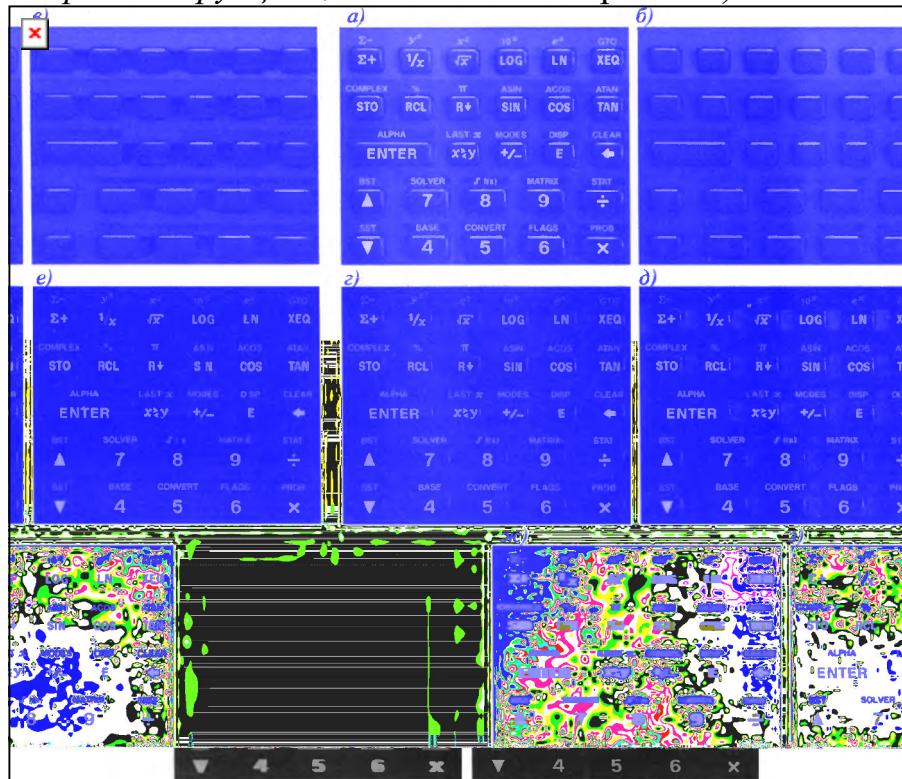


Рисунок 30 – Применение полутоновой реконструкции. а) Исходное изображение. б) Размыкание реконструкцией. в) Размыкание. г) «Верх шляпы» реконструкцией. д) «Верх шляпы». е) Размыкание реконструкцией г) с использованием горизонтального отрезка. ж) Дилатация е) с использованием горизонтального отрезка. з) Окончательный результат реконструкции

```

>> f_thr = imsubtract (f, f_обр );
>> f_th = imsubtract (f, f_o );

```

На рис. 30д) дан результат обычной операции «верх шляпы» (т.е. изображение f_{th}).

Теперь мы удалим линии вертикальных отблесков на правой стороне каждой кнопки на рис. 30г). Для этого мы воспользуемся размыканием реконструкцией с элементом, который есть короткий горизонтальный отрезок:

```
>> g_обр = imreconstruct (imerode (f_th, ones (1, 11)), f_th);
```

На полученном изображении [см. рис. 30e)] вертикальные отблески исчезли. Однако это же случилось с символами, у которых имеются тонкие вертикальные черточки, например, пропала косая черта у символа процента «%», и исчезла вся буква «I» в слове ASIN. Тогда мы воспользуемся тем обстоятельством, что буквы с удаленными по ошибке деталями будут близки к другим символам, если сначала выполнить дилатацию [рис. 30ж)],

```
>> g_obrd = imdilate ( g_обр , ones ( 1 , 21 ) );
```

а затем уже совершил окончательную реконструкцию с изображением f_thr в качестве маски и min (g_obrd) в качестве маркера:

```
>> f2 = imreconstruct ( min ( g_obrd , f_thr ) , f_thr );
```

На рис. 30a) приведен окончательный результат. Заметим, что на этом изображении успешно удалены все тени на фоне изображения и все отблески на кнопках.

Общая постановка задачи

В результате выполнения заданий лабораторной работы студенты **должны уметь** создавать программно-алгоритмическую поддержку для компьютерной реализации требуемых технике методов морфологической обработки изображений в MatLab.

Лабораторные занятия проводятся в компьютерных классах.

Список индивидуальных данных

Написать файл-функцию и GUI интерфейс для решения следующих задач.

Задание:

1. Выполнить чтение нескольких изображений из файла, самостоятельно выбранного студентом. Выбрать первую цветовую компоненту изображения в формате RGB и определить размер изображения. Использовать функцию **imshow** для вывода изображений на экран.
2. С использованием функции **imdilate** выполнить дилатацию изображения.
3. С использованием функции **imerode** выполнить эрозию изображения.
4. С использованием функций **imopen** и **imclose** выполнить размыкание и замыкание изображения.
5. С использованием функции **bwhitmiss** выполнить преобразование успех/ неудача для изображения.
6. С использованием функции **bwmorph** выполнить утончение и построение остова изображения.
7. Найти центры масс компонент связности.
8. Выполнить размыкание реконструкцией.
9. С использованием функции **imfill** выполнить заполнение отверстий изображения.
10. С использованием функции **imclearborder** выполнить очистку от пограничных объектов на изображении.

11. Выполнить морфологическое сглаживание с помощью размыкания и замыкания.
12. Применить преобразования «верх шляпы» для изображения.
13. Выполнить операцию Гранулометрия для изображения.
14. Использовать операцию реконструкции для удаления сложного фона изображения.
15. Вывести на экран результирующее изображение.
16. Сохранить полученные изображения.

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.

Лабораторная работа №8. Алгоритмы сжатия графической информации

Цель работы:

Целью лабораторной работы является получение навыков самостоятельной алгоритмической и программной реализации на компьютерной технике алгоритмов сжатия графической информации в MatLab.

В результате выполнения лабораторной работы обучающийся должен демонстрировать следующие результаты:

Знать:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-10	3-1	Понимает необходимость саморазвития
	3-2	Выражает желание повышать свою квалификацию и мастерство
ПК-1	3-1	Знает основы современных технологий сбора, обработки и представления информации
	3-2	Способен ориентироваться в информационном потоке в глобальных компьютерных сетях

Уметь:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ПК-1	У-1	Знает основы современных технологий сбора, обработки и представления информации
	У-2	Способен ориентироваться в информационном потоке в глобальных компьютерных сетях
ПК-2	У-1	Способен анализировать социально-экономические проблемы
	У-2	Знает методы расчетов математических моделей

Владеть:

Индекс компетенции	Индекс образовательного результата	Образовательный результат
ОК-5	B-1	Способен к восприятию и воспроизведению информации
	B-2	Знает алгоритмы целеполагания и выбора путей их достижения
ПК-2	B-1	Способен анализировать социально-экономические проблемы
	B-2	Знает методы расчетов математических моделей
ПК-25	B-1	Знает различные математические методы решения прикладных задач
	B-2	Знает методы формализации решения прикладных задач

Теоретическая часть

Задача сжатия изображения заключается в сокращении объема данных, необходимого для представления цифрового изображения. Эффект сжатия может быть достигнут путем удаления одного из трех типов избыточности:

- (1) кодовой избыточности, имеющей место, если используемые кодовые слова не являются оптимальными (например, не имеют минимальную длину);
- (2) межпиксельной избыточности, которая возникает при наличии определенной корреляции между близкими пикселями;

(3) визуальной избыточности, содержащейся в информации, которая не воспринимается органами зрения человека (т.е. детали изображения, несущественные для глаза).

Некоторые основы

Как видно из рис. 1, системы сжатия изображений состоят из двух различных структурных блоков: *кодера* и *декодера*. Изображение $f(x,y)$ подается на вход кодеров, который преобразует входные данные в определенный набор символов и использует его для представления исходного изображения. Пусть n_1 и n_2 обозначают число элементарных носителей информации (например, битов) исходного и закодированного изображения. Тогда меру сжатия можно выразить количественно с помощью *коэффициента сжатия* $C_R = n_1 / n_2$. Коэффициент сжатия 10 (или 10:1) указывает на то, что исходное изображение имеет 10 элементов хранения информации (т.е. битов) на каждый элемент хранения в сжатом наборе данных. В MATLAB частное числа битов, используемых для хранения и представления двух файлов изображений и/или переменных, можно вычислять с помощью следующей M-функции:

```
function cr = imratio(f1, f2)
error(nargchk(2, 2, nargin));
cr = bytes(f1) / bytes(f2);

function b = bytes(f)
if ischar(f)
    info = dir(f);           b = info.bytes;
elseif isstruct(f)
    b = 0;
    fields = fieldnames(f);
    for k = 1:length(fields)
        b = b + bytes(f.(fields{k}));
    end
else
    info = whos('f');       b = info.bytes;
end
```

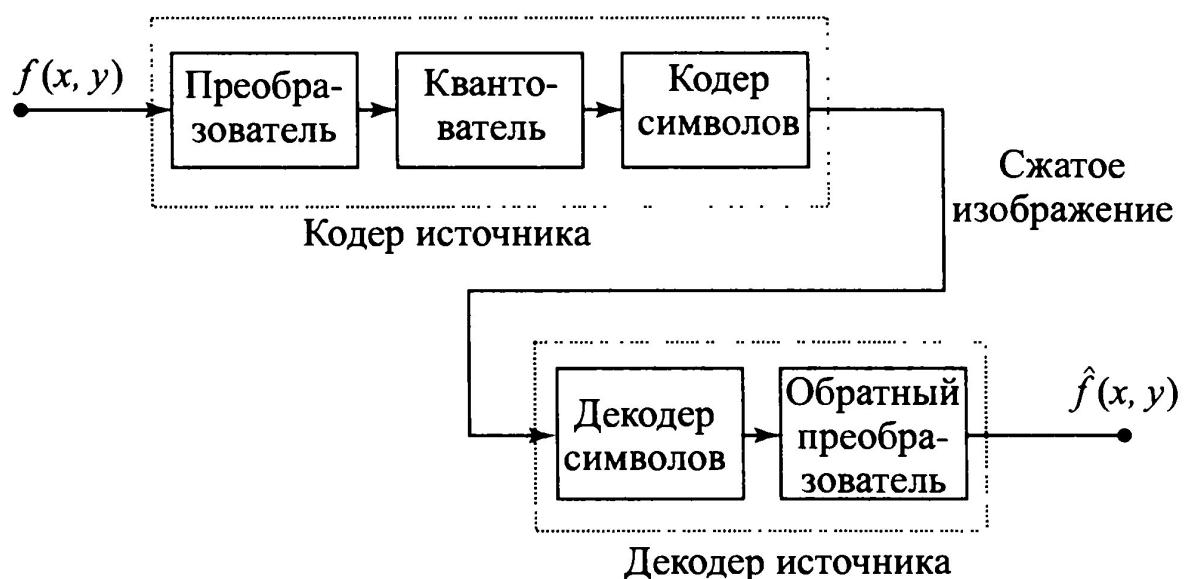


Рисунок 1 - Блок-схема модельной системы сжатия изображения

Например, коэффициент сжатия изображения на рис. 4в) при кодировании JPEG можно вычислить с помощью команды

```
>> r = imratio (imread ('bubbles25.jpg'), 'bubbles25.jpg')
r =
    35.1612
```

Заметим, что в `imratio` внешняя функция `b = bytes(f)` написана так, чтобы она возвращала число байтов

- (1) в файле,
- (2) в структурной переменной, и/или
- (3) в переменной, не являющейся структурой.

Если `f` не является структурной переменной, то функция `whos` возвращает размер этой переменной в байтах. Если `f`- это имя файла, то

аналогичное действие выполняет функция `dir`. В используемой форме синтаксиса функция `dir` возвращает структуру, которая имеет поля `name`, `data`, `bytes` и `isdir`. В этих полях записаны, соответственно, имя файла, дата его последнего изменения, размер в байтах и информация о том, является ли `f` папкой (директорией) (`isdir = 1`, если «да», иначе `isdir = 0`). Наконец, если `f` является структурой, то функция `bytes` вызывает рекурсивно сама себя для сложения числа байтов, используемых при хранении каждого отдельного поля структуры. Это позволяет исключить объем служебной информации, связанной с самой структурой переменной (124 байта хранения на одно поле), когда возвращается только число байтов, необходимых для хранения самых данных, записанных в полях. Функция `fieldname` используется для получения списка полей `f`, а операторы

```
for k = 1:length (fields)
    b = b+bytes (f. (fields {k})) ;
```

осуществляют рекурсию. Обратите внимание на применение *динамических имен полей структур* в рекурсивных вызовах функции `bytes`. Если `S`- это структура, а `F`- переменная символьной строки, содержащая имя поля, то операторы

```
S. (F) = f00;
field = S. (F);
```

употребляют синтаксис динамических имен полей структур для присвоения и/или прочтения содержимого поля `F` структуры `S`.

Чтобы увидеть и/или использовать сжатое (т.е. закодированное) изображение, его необходимо подать на декодер (см. рис. 1), который построит реконструированное изображение $f^*(x, y)$. В общем случае, изображение $f^*(x, y)$ может совпадать с исходным изображением $f(x, y)$ или может от него отличаться. В первом случае система называется *свободной от ошибок*, или системой кодирования или сжатия *без потери информации*. В противном случае в реконструированном изображении присутствуют определенные изображения и потери и система называется кодированием

или сжатием с потерей информации. Ошибку (невязку) $e(x, y)$ между $f(x, y)$ и $\hat{f}(x, y)$ можно вычислить для любых пикселов (x, y) по формуле

$$e(x, y) = \hat{f}(x, y) - f(x, y),$$

а величина общей ошибки между двумя изображениями равна

$$\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]$$

Среднеквадратическое отклонение (root - mean - square) e_{rms} между $f(x, y)$ и $\hat{f}(x, y)$ равно квадратному корню из средней квадратичной ошибки по всей матрице $M \times N$, т.е.

$$e_{rms} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\hat{f}(x, y) - f(x, y))^2 \right]^{1/2}.$$

Следующая M-функция вычисляет e_{rms} и строит (если $e_{rms} \neq 0$) изображение $e(x, y)$ и его гистограмму. Поскольку матрица $e(x, y)$ может содержать как положительные, так и отрицательные величины, вместо `imhist` (которая работает только с изображениями) используется функция `hist` для построения гистограммы.

```
function rmse = compare(f1, f2, scale)
error(nargchk(2, 3, nargin));
if nargin < 3
    scale = 1;
end
e = double(f1) - double(f2);
[m, n] = size(e);
rmse = sqrt(sum(e(:).^ 2) / (m * n));
if rmse
    emax = max(abs(e(:)));
    [h, x] = hist(e(:), emax);
    if length(h) >= 1
        figure; bar(x, h, 'k');

        emax = emax / scale;
        e = mat2gray(e, [-emax, emax]);
        figure; imshow(e);
    end
end
```

Заметим, что кодер на рис. 1 отвечает за сокращение всех трех типов избыточности (кодовой, межпиксельной и визуальной) исходного изображения. На первой стадии процесса кодирования преобразователь трансформирует входное изображение в некоторый (невизуальный) формат, приспособленный для понижения межпиксельной избыточности. На второй стадии блок квантователь понижает точность выхода преобразователя в соответствии с ранее оговоренным критерием точности. На этой стадии происходит сокращение визуальной избыточности, т.е. некоторое огрубление изображения, почти незаметное, однако, для глаза. Эта процедура является необратимой, поэтому не выполняется, если необходимо совершить сжатие без потери информации. На третьей и последней стадии процесса кодер символов строит оптимальный код (который сокращает кодовую

избыточность) для выхода квантователя и преобразует выходную последовательность в соответствии с построенным кодом.

Декодер на рис. 1 имеет только две компоненты: декодер символов и обратный преобразователь. Эти блоки совершают действия, обратные действиям блока кодера символов и блока преобразователя. Поскольку процедура квантования не является обратимой, блок обратного квантователя отсутствует в общей схеме декодера.

Кодовая избыточность

Пусть дискретная случайная величина r_k при $k = 1, 2, \dots, L$ с вероятностями $p_r(r_k)$ соответствует распределению уровней полутонового изображения, общее число которых равно L . Предполагается, что r_1 соответствует нулевому уровню (напомним, что индексы массивов в MATLAB начинаются с 1, а не с 0). Пусть также

$$p_r(r_k) = \frac{n_k}{n}, \quad k = 1, 2, \dots, L,$$

где n_k равно числу появлений на изображении пикселов k -го уровня серого цвета, а n – общее число пикселов изображения. Если число битов, используемых в цифровом представлении каждой величины r_k , равно $l(r_k)$, то среднее число битов, необходимое для представления каждого пикселя, равно

$$L_{\text{avg}} = \sum_{k=1}^L l(r_k) p_r(r_k).$$

Таким образом, средняя длина кодовых слов, назначаемых различным значениям уровней серого цвета, находится суммированием произведений числа битов представления каждого уровня на вероятность появления этого уровня. Значит, общее число битов для кодирования изображения размерами $M \times N$ равно $MN L_{\text{avg}}$.

Таблица 1. Иллюстрация кодовой избыточности: $L_{\text{avg}} = 2$ для кода 1 и $L_{\text{avg}} \approx 1.81$ для кода 2

r_k	$p_r(r_k)$	Код 1	$l_1(r_k)$	Код 2	$l_2(r_k)$
r_1	0.1875	00	2	011	3
r_2	0.5000	01	2	1	1
r_3	0.1250	10	2	010	3
r_4	0.1875	11	2	00	2

Если все уровни изображения представлены обычным m -битовым двоичным кодом, то правая часть последнего равенства, очевидно, сводится к m битам. В самом деле, $l(r_k)=m$ для любого r_k , и этот множитель выносится за знак суммирования, а сумма всех $p_r(r_k)$ равна 1, и в итоге $L_{\text{avg}} = m$. Из табл. 1 следует, что кодовая избыточность почти всегда присутствует, когда уровни градации серого цвета кодируются обычными фиксированной длины. В этой таблице даны обе схемы кодирования пикселов четырехуровневого изображения, распределение уровней которого дано во втором столбце. Кодирование каждого уровня двумя битами (Код 1 в третьем столбце

таблицы) дает среднюю длину кодового слова, равную 2 битам. А средняя длина кода пикселя при кодировании Кодом 2 (пятый столбец таблицы) равна

$$L_{\text{avg}} = \sum_{k=1}^4 l(r_k)p_r(r_k) = 3 \cdot 0.1875 + 1 \cdot 0.5 + 3 \cdot 0.125 + 2 \cdot 1.875 = 1.8125,$$

и достигнутый коэффициент сжатия равен $C_R = 2/1.8125 \approx 1.103$. Эффект сжатия при использовании Кода 2 достигается за счет того, что кодовые слова имеют переменную длину, и это позволяет назначать короткие коды значениям пикселов, чаще других появляющимся на изображении.

Тогда возникает естественный вопрос: сколько битов требуется для оптимального представления уровней пикселов изображения? Иными словами, какой минимальный объем данных является достаточным для полного описания изображения без потери информации? С математической точки зрения, ответы на подобные вопросы дает наука теория информации. Основная посылка этой теории заключается в том, что информационный поток можно моделировать в виде вероятностного процесса, измерение которого хорошо согласуется с нашей интуицией.

В соответствии с этим положением можно считать, что случайная величина E , наблюдаемая с вероятностью $P(E)$, несет

$$I(E) = \log \frac{1}{P(E)} = -\log P(E)$$

единиц информации. Если $P(E) = 1$ (событие всегда наступает), то $I(E) = 0$ и здесь нет никакой информации. В самом деле, в этом случае нет неопределенности в наступлении данного события, и поэтому нет необходимости сообщать или передавать информацию о том, что событие имело место. При таком подходе, чем реже наступает событие, тем оно «ценнее» и, следовательно, несет в себе большую информацию. Имея источник случайных событий из дискретного семейства возможных исходов $\{a_1, a_2, \dots, a_j\}$ и соответствующий набор вероятностей этих исходов $\{P(a_1), P(a_2), \dots, P(a_j)\}$, средняя информация, приходящаяся на один выход источника, называемая энтропией этого источника, вычисляется по формуле

$$H = - \sum_{j=1}^J P(a_j) \log P(a_j).$$

Если интерпретировать изображение как выборку, порождающую некоторым «полутоновым источником», то можно моделировать вероятности символов этого источника с помощью гистограммы наблюдаемого полутонового изображения. Тогда строится число, называемое оценкой первого порядка H для энтропии источника:

$$\tilde{H} = - \sum_{j=1}^L p_r(r_k) \log p_r(r_k).$$

Эту оценку можно вычислять с помощью следующей М-функции в предположении, что каждый уровень серого цвета кодируется независимо от других. В этом случае в теории информации доказано, что энтропия

задает нижнюю границу сжатия, которую можно достичнуть путем удаления кодовой избыточности.

```
function h = entropy(x, n)
error(nargchk(1, 2, nargin));
if nargin < 2
    n = 256;
end
x = double(x);
xh = hist(x(:), n);
xh = xh / sum(xh(:));
i = find(xh);
h = -sum(xh(i) .* log2(xh(i)));
```

Обратите внимание на употребление функции `find` из MATLAB для нахождения индексов ненулевых элементов гистограммы `xh`. Оператор `f ind(x)` эквивалентен команде `find(x ~= 0)`. Функция `entropy` использует `find` при построении индексного вектора `i` гистограммы `xh`, который будет использоваться в дальнейшем для удаления всех нулевых значений из формулы для вычисления энтропии в последней строке функции. Если не сделать этого, функция `log2` выдаст для `h` значение `NaN` (результатом команды `0*-inf` является *не число*), когда вероятность символа равна нулю.

Пример 1. Вычисление оценки первого порядка для энтропии.

Рассмотрим простое изображение 4x4, гистограмма которого (см. вектор `p` в следующем программном фрагменте) моделирует вероятности символов из табл. 1. Следующая последовательность команд строит одно такое изображение и вычисляет оценку первого порядка для его энтропии.

```
» f= [119 123 168 119; 123 119 168 168];
» f= [f; 119 119 107 119; 107 107 119 119]
f=
    119    123    168    119
    123    119    168    168
    119    119    107    119
    107    107    119    119
p = hist(f(:, 8);
p = p / sum(p)
P =
0.1875  0.5  0.125  0  0  0  0  0.1875
h = entropy(f)
h =
    1.7806
```

Код 2 из табл. 1 имеет среднюю длину $L_{avg} \sim 1.81$, которая приближает оценку первого порядка для энтропии, равной минимальной длине *двоичного* кода для изображения `f`. Заметим, что уровень 107 соответствует элементу r_1 и имеет двоичный код 011_2 в табл. 1, 109 — это r_2 с двоичным кодом 1_2 , а 123 и 168 имеют коды 010_2 и 00_2 соответственно.

Коды Хаффмана

При кодировании уровней полутонового изображения или выхода некоторой операции полутонового отображения (разности пикселов, длин

серий и т.д.) *коды Хаффмана* назначают символам источника наименьшее возможное число кодовых символов (например, битов) при условии, что символы источника (например, пиксели) кодируются *по отдельности*.

Первый шаг метода Хаффмана состоит в построении серии редуцированных источников путем упорядочивания вероятностей символов данного источника и объединения символов с наименьшими вероятностями в один символ, который будет их замещать в редуцированном источнике следующего уровня. Этот процесс проиллюстрирован на рис. 2a) для распределения уровней из табл. 1. В двух левых колонках исходный набор символов источника и их вероятности упорядочены сверху вниз по убыванию вероятностей. Для формирования первой редукции источника два символа с наименьшими вероятностями — в данном случае это a_1 и a_3 с вероятностями 0.1875 и 0.125 — объединяются в один «составной символ» с суммарной вероятностью 0.3125. Этот составной символ и связанная с ним вероятность помещаются в список первого редуцированного источника, который также упорядочивается от наибольшей вероятности к наименьшей [см. третью колонку рис. 2a)]. Этот процесс повторяется до тех пор, пока не образуется редуцированный источник всего с двумя символами [самая правая колонка на рис. 2a)].

a)

Исходный источник		Редуцированный источник	
Символ	Вероятность	1	2
a_2	0.5	0.5	0.5
a_4	0.1875	0.3125	0.5
a_1	0.1875	0.1875	
a_3	0.125		

б)

Исходный источник			Редуцированный источник			
Символ	Вероятность	Код	1	2	1	2
a_2	0.5	1	0.5	1	0.5	1
a_4	0.1875	00	0.3125	01	0.5	0
a_1	0.1875	011	0.1875	00		
a_3	0.125	010				

Второй шаг процедуры Хаффмана состоит в кодировании каждого редуцированного источника, начиная с источника с наименьшим числом символов и двигаясь к исходному источнику. Наименьший двоичный код для источника с двумя символами состоит, конечно, из символов 0 и 1. Как показано на рис. 2б), эти символы приписываются к двум символам источника справа (порядок присвоения не имеет значения — перестановка 0

и 1 даст абсолютно тот же результат). Поскольку второй символ редуцированного источника с вероятностью 0.5 был получен объединением двух символов предыдущего редуцированного источника (расположенного слева от него), кодовый символ 0 приписывается *каждому* из объединенных символов, после чего коды этих символов дополняются слева символами 0 и 1 (в произвольном порядке) для отличия их друг от друга. Затем эта операция повторяется для редуцированных источников всех уровней вплоть до исходного источника. Окончательные коды для символов исходного источника приведены в третьей колонке на рис. 2б).

Код Хаффмана на рис. 2б) (и из табл. 1) является мгновенно и однозначно декодируемым блоковым кодом. Код называется *блоковым*, поскольку каждый символ источника отображается в фиксированную последовательность кодовых символов. Он является *мгновенно декодируемым*, так как каждое кодовое слово в закодированной последовательности можно декодировать независимо от последующих кодовых символов. Это связано с тем, что в любом коде Хаффмана каждое кодовое слово не является префиксом (началом) никакого другого кодового слова. По этой же причине код Хаффмана является *однозначно декодируемым*, так как любая последовательность, состоящая из кодовых слов кода Хаффмана, может быть декодирована единственным способом. Таким образом, любую последовательность кодированных по Хаффману символов можно декодировать, анализируя ее слева направо. Изображение размером 4x4, рассмотренное в примере 1, кодируется по Хаффману в порядке строк сверху вниз и по столбцам слева направо с помощью кодов из рис. 2б), в результате чего получается последовательность из 29 кодовых символов 10101011010110110000011110011. Поскольку мы использовали мгновенно и однозначно декодируемый блоковый код, нет необходимости ставить разделительные символы между закодированными пикселями. Анализ этой кодовой последовательности слева направо обнаруживает, что первым допустимым кодовым словом является 1, которое является кодом символа a_2 , отвечающего уровню серого цвета 109. Следующее допустимое кодовое слово — это 010, которое соответствует уровню 123. Продолжая этот анализ, мы в итоге получим полностью декодированное изображение, которое эквивалентно изображению f из рассматриваемого примера.

Построение редуцированных источников и назначение кодовых слов по описанной выше схеме реализованы в следующей M-функции, которую мы назвали *huffman*:

```
function CODE = huffman(p)
error(nargchk(1, 1, nargin));
if (ndims(p) ~= 2) | (min(size(p)) > 1) | ~isreal(p) | ~isnumeric(p)
    error('P must be a real numeric vector.');
end
global CODE
CODE = cell(length(p), 1);

if length(p) > 1
    p = p / sum(p);
```

```

s = reduce(p);
makecode(s, []);
else
    CODE = {'1'};

function s = reduce(p);
s = cell(length(p), 1);

for i = 1:length(p)
    s{i} = i;
end
while numel(s) > 2
    [p, i] = sort(p);
    p(2) = p(1) + p(2);
    p(1) = [];

    s = s(i);
    s{2} = {s{1}, s{2}};
    s(1) = [];
end

function makecode(sc, codeword)
global CODE

if isa(sc, 'cell')
    makecode(sc{1}, [codeword 0]);
    makecode(sc{2}, [codeword 1]);
else
    CODE{sc} = char('0' + codeword);
end

```

Следующая последовательность команд использует кодирование Хаффмана для построения кодов рис. 2:

```

» p = [0.1875 0.5 0.125 0.1875];
>> c = huffman(p)
c =
    '011'
    '1'
    '010'
    '00'

```

Отметим, что выходом служит массив, состоящий из строк переменной длины, в котором каждый элемент представляет собой строку из нулей и единиц, что является двоичным кодом соответствующего индексированного символа из p. Например, символьная строка '010' (в массиве имеет индекс 3) является кодом уровня серого цвета с вероятностью 0.125.

В первых строках функции **huff**tal осуществляется проверка входного аргумента p (вектора вероятностей входных кодируемых символов) на предмет его корректности и производится инициализация *глобальной переменной* CODE в виде смешанного массива MATLAB, который имеет length(p) строк и один столбец. Все глобальные переменные MATLAB должны быть продекларированы в функциях, которые будут их использовать, с помощью команды

global X Y Z.

Эта команда делает переменные X, Y и Z доступными в тех функциях, в которых они будут продекларированы. Когда несколько функций декларируют одни и те же глобальные переменные, они используют одну и ту же копию этих переменных. В функции huffman основная программа, а также внешняя функция makecode используют общую глобальную переменную CODE. Обратим ваше внимание на то, что общепринято писать имена глобальных переменных заглавными буквами. Неглобальные переменные являются *локальными переменными*, и их можно использовать лишь в тех функциях, где они объявлены (но не в других функциях или в основном рабочем пространстве). Имена локальных переменных принято писать строчными буквами.

В **huffman** переменная CODE инициализируется с помощью функции cell, которая имеет следующий синтаксис:

$$X = \text{cell}(m, n).$$

При исполнении этой команды строится массив *m**n* матриц, к элементам которых можно обращаться как к смешанному массиву или по содержимому. Круглые скобки «()» используются для индексации смешанного массива; фигурные скобки «{ }» используются для *индексации содержимого*. Итак, команда $X(1) = []$ индексирует и удаляет элемент 1 из смешанного массива, в то время как $X\{1\} = []$ присваивает первому элементу массива пустую матрицу. То есть, $X\{1\}$ обращается к содержимому первого элемента (массива), а $X(1)$ обозначает сам этот элемент (а не его содержимое). Поскольку смешанные массивы могут находиться внутри других смешанных массивов, синтаксис типа $X\{1\}\{2\}$ обращается ко второму элементу смешанного массива, который является первым элементом смешанного массива X.

После инициализации переменной CODE и нормирования входного вектора вероятностей [по команде $p = p/\text{sum}(p)$] код Хаффмана для нормированного вектора вероятностей p строится за два шага. На первом шаге, который начинается оператором $s = \text{reduce}(p)$ основной программы, вызывается внешняя функция reduce, которая совершает редуцирование источника, проиллюстрированное на рис. 2a). В подпрограмме reduce элементы изначально пустого смешанного массива s редуцированного источника, размер которого согласован с CODE, инициализируются своими индексными значениями. Т.е. $s\{1\} = 1$, $s\{2\} = 2$ и т.д. После этого в цикле while $\text{numel}(s) > 2$ создается двоичное дерево редуцированных источников, эквивалентное смешанному массиву. На каждом шаге цикла вектор p упорядочивается в восходящем порядке вероятностей. Это совершается функцией sort, имеющей синтаксис

$$[y,i] = \text{sort}(x),$$

где в выход y записывается вектор упорядоченных элементов вектора x, а индексный вектор i удовлетворяет соотношению $y = x(i)$. После упорядочения вектора p две наименьшие вероятности сливаются путем помещения их суммы в p(2), а элемент p(1) выбрасывается. После этого смешанный массив редуцированного источника переупорядочивается, чтобы

соответствовать p на базе индексного вектора i с помощью операции $s = s(i)$. Наконец, $s\{2\}$ заменяется смешанным массивом из двух элементов, в котором записаны индексы слившихся вероятностей по формуле $s\{2\} = \{s\{1\}, s\{2\}\}$ (пример индексирования по содержимому), а индексирование по массиву используется для удаления первого из двух слившихся элементов $s\{1\}$ по правилу $s\{1\}=[]$. Процесс продолжается до тех пор, пока в s не останется только два элемента.

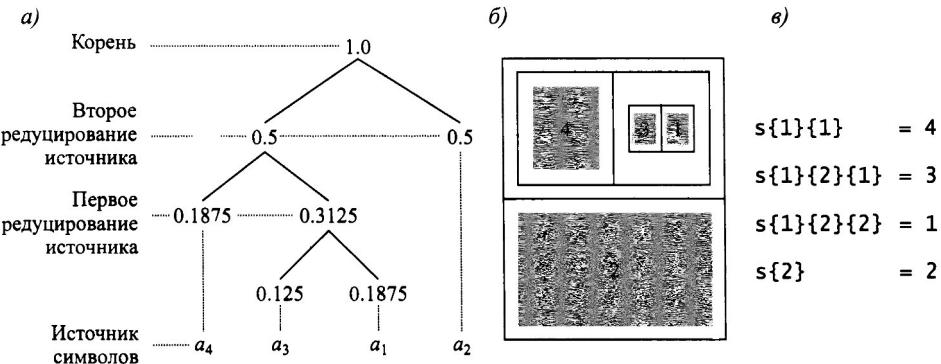


Рисунок 3 – Редуцирование источника на рис. 2а) с помощью функции Haffman а) Эквивалентное двоичное дерево; б) Изображение, построенное `cellplot(s)`; в) Выход команды `celldisp(s)`

На рис. 3 показан результат процесса обработки вероятностей символов из команд

`celldisp(s);`
`cellplot(s);`

между двумя последними исполняемыми строками основной программы `huffman`. Функция MATLAB `celldisp` рекурсивно печатает содержимое смешанного массива, а графическая функция `cellplot` представляет смешанный массив в виде семейства вложенных прямоугольных блоков. Обратите внимание на взаимно однозначное соответствие между элементами смешанного массива на рис. 3б) и узлами дерева редуцированного источника. На рис. 3а):

- (1) каждая пара разветвляющихся ветвей дерева (которая обозначает редуцирование источника) соответствует двухэлементному смешанному массиву в s ;
- (2) каждый двухэлементный смешанный массив содержит индексы символов, которые были слиты в один символ в процессе редуцирования.

Например, объединение символов a_3 и a_1 в основании дерева порождает двухэлементный смешанный массив $s\{1\}\{2\}$, где $s\{1\}\{2\}\{1\} = 3$ и $s\{1\}\{2\}\{2\} = 1$ (индексы a_3 и a_1 соответственно). Корень дерева, расположенный сверху, образует двухэлементный смешанный массив s самого верхнего уровня.

Последний шаг процесса построения кода (т. е. присвоение кодовых слов на основе редуцированного источника) осуществляется вызовом функции `makecode(s, [])`. Это обращение инициирует рекурсивную

процедуру присвоения кода, основанную на схеме рис. 2б). Хотя рекурсивное задание не обеспечивает сохранение в памяти (поскольку стек обработанных величин должен где-то храниться) или увеличения скорости, оно имеет то преимущество, что код получается более компактным и его легче понять, особенно при работе с такой рекурсивно заданной структурой данных, как дерево. Любую функцию MATLAB можно использовать рекурсивно, т.е. она может вызывать сама себя явно или неявно. При использовании рекурсии каждый вызов функции порождает новое множество локальных переменных, которое не зависит от всех предыдущих подобных множеств.

Внешняя функция makecode принимает два входные параметра: codeword, массив из нулей и единиц, и sc, элемент смешанного массива редуцированного источника. Когда sc сам является смешанным массивом, он состоит из двух символов источника (или составных символов), которые были объединены в процессе редуцирования. Поскольку их необходимо кодировать раздельно, делаются два рекурсивных обращения (к функции make с ode) наряду с двумя подходящим образом обновленными кодовыми словами (0 и 1 добавляются к входному codeword). Когда sc не содержит смешанный массив, он является индексом символа изначального источника и ему присваивается двоичная кодовая последовательность, построенная по codeword с помощью операции

$$\text{CODE}\{sc\} = \text{char}'0' + \text{codeword}.$$

Функция MATLAB char преобразует массив, содержащий положительные целые числа, представляющие коды символов в символьный массив MATLAB (первые 127 кодов представляют собой стандартные ASCII коды символов). Значит, к примеру, `char ('0' +[0 1 0])` породит символьную строку '010', поскольку добавление 0 к ASCII коду нуля даст ASCII '0', а прибавление 1 к ASCII '0' даст ASCII код 1, а именно '1'.

Таблица 2. Процесс назначения кодов для смешанного массива редуцированного источника на рис. 3

Вызов	Источник	sc	Кодовое слово
1	<code>main routine</code>	{1x2 cell}	[]
		[2]	
2	<code>makecode</code>	[4] {1x2 cell}	0
3	<code>makecode</code>	4	0 0
4	<code>makecode</code>	[3] [1]	0 1
5	<code>makecode</code>	3	0 1 0
6	<code>makecode</code>	1	0 1 1
7	<code>makecode</code>	2	1

Табл. 2 детализирует последовательность вызовов makecode, которые производятся в соответствии со смешанным массивом редуцированного

источника на рис. 3. Чтобы закодировать 4 символа изначального источника, потребовалось 7 вызовов этой функции. Первый вызов (строка 1 в табл. 2) совершается из основной программы huffman. Он запускает процесс кодирования с присвоением входам codeword и sc, соответственно, пустой матрицы и смешанного массива s. В соответствии со стандартным обозначением MATLAB, {1x2 cell} обозначает смешанный массив из одной строки и двух столбцов. Поскольку sc почти всегда является смешанным массивом при первом вызове (исключение составляет источник, состоящий из одного элемента), совершаются два рекурсивных вызова (см. строки 2 и 7). Первый из этих вызовов инициирует еще два вызова (строки 3 и 4), а второй инициирует два дополнительных вызова (строки 5 и 6). Во всех случаях, когда sc не является смешанным массивом, как это имеет место в строках 3, 5, 6 и 7 таблицы, дополнительные рекурсии не нужны; кодовая последовательность строится по codeword и назначается символу источника, чей индекс был передан как sc.

Кодирование Хаффмана

Построение кодов Хаффмана само по себе не является сжатием. Чтобы получить эффект сжатия, заложенный в эти коды, символы соответствующего источника, независимо от их природы (уровни серых тонов, длины серий или выходы иных операций отображения уровней), необходимо преобразовать или трансформировать (т. е. закодировать) в соответствии с построенным кодом.

Пример 2. Отображение кодов переменной длины в MATLAB.

Рассмотрим простое 16-ти байтовое изображение размерами 4x4:

```
>>f2 = uint8([2 34 2; 324 4; 2212; 112 2])
f2 =
 2   3   4   2
 3   2   4   4
          2   2   12
          112   2
vhos('f2')
Name      Size      Bytes      Class
f2          4x4         16  uint8 array
Grand total is 16 elements using 16 bytes
```

Каждый пиксель в f2 является байтом, состоящим из 8 битов. Для представления всего изображения используется 16 байтов. Поскольку уровни серых тонов в f2 не являются равновероятными, коды переменной длины (как отмечалось в предыдущем параграфе) позволяют сократить объем памяти, которая потребуется для хранения этого изображения. Функция huff man строит один из таких кодов:

```
>>c = huffman(hist(double(f2(:)), 4))
c =
'011'
'1'
'010'
```

'00'

Поскольку коды Хаффмана основаны на относительных частотах появления символов источника (но не на самих символах), которые требуется закодировать, код с идентичен коду, построенному для изображения из примера 1. На самом деле, изображение f2 можно получить из f (пример 1) путем отображения уровней 107,119,123 и 168 в числа 1, 2, 3 и 4. Оба изображения имеют одинаковый вектор частот $p = [0.1875 \ 0.5 \ 0.125 \ 0.1875]$.

Простейший путь кодирования f2 с помощью кода с заключается в совершении следующих операций:

```
» hlf2 = hlf2 =
    Columns 1 through 9
    '1'  '010'  '1'  '011'  '010'  '1'  '1'  '011'  '00'
    Columns 10 through 16
    '00'  '011'  '1'  '1'  '00'  '1'  '1'
    » whos('hlf2')
      Name      Size      Bytes      Class
      hlf2     1x16     1530     cell array
      Grand total is 45 elements using 1530 bytes
```

Здесь изображение f2 (двумерный массив класса `UINT8`) преобразован в смешанный массив `hlf2` размерами `1x16` (операция транспонирования применена для экономии бумаги). Элементами `hlf2` являются символьные строки переменной длины, а соответствующие пиксели `f2` расположены в порядке считывания сверху вниз и слева направо (т.е. по столбцам). Как видно из этой распечатки, закодированное изображение занимает 1530 байт, т. е. памяти требуется почти в 100 раз больше, чем при хранении самого изображения `f2`!

Использование смешанного массива `hlf2` вполне логично, поскольку это один из двух стандартных типов данных в MATLAB для представления разнородных данных. В случае с `hlf2` эта разнородность означает различие длин символьных строк, и платой за удобство обращения с ними посредством смешанного массива служит существенное увеличение требуемой памяти (присущее смешанным массивам), которое требуется для отслеживания положения элементов переменной длины. Этот излишний расход памяти можно устраниТЬ путем преобразования `hlf2` в обычный двумерный символьный массив:

```
» h2f2 = char (hlf2)'
h2f2 =
    1010011000011011  1      11
    1001  0 0 10 1 1
    » whos('h2f2')
      Name      Size      Bytes      Class
      h2f2     3x16     96     char array
      Grand total is 48 elements using 96 bytes.
```

Здесь смешанный массив h1f2 преобразован в символьный массив h2f2 размерами 3x16. Каждый столбец h2f2 соответствует пиксели в f2 в порядке сканирования сверху вниз и слева направо (т. е. по столбцам). Отметим, что в массиве h2f2 вставлены символы пробелов для правильного соответствия битов по столбцам. Поскольку два байта требуется для каждого символа '0' или '1' кодовых слов, для хранения h2f2 требуется 96 байт — все еще в 6 раз больше, чем исходные 16 байт, необходимые для хранения f2. Мы можем избавиться от вставки пустых символов с помощью операций

```
»h2f2 = h2f2(:);
» h2f2(h2f2 =='''')=[];
» whos('h2f2')
  Name      Size    Bytes  Class
  h2f2    29x1      58  char array
```

Grand total is 29 elements using 58 bytes

Но требуемый объем памяти по-прежнему превосходит 16 байт для хранения f2. Чтобы по-настоящему сжать f2, код с должен проявиться на битовом уровне, когда несколько закодированных пикселов пакуются в один байт:

```
»h3f2 =
mat2huff(f2)
h3f2 =
size:
[4 4]
min:
32769
hist:
[3 8 2 3]
code:
[43867 1944]
» whos(
'h3f2')
  Name
  Size    Bytes
  Class
  h3f2
  1x1      518
  struct array
```

Grand total is 13 elements using 518 bytes

Несмотря на то, что функция mat2huff возвращает структуру h3f2, которой требуется для хранения 518 байт памяти, большая ее часть связана со служебной информацией о структурной переменной, а также с тем, что mat2huff генерирует некоторую информацию, облегчающую будущее декодирование. Пренебрегая этими излишками памяти, которые пренебрежимо малы при работе с практически значимыми изображениями

(нормальных размеров), mat2huff сжимает f2 с коэффициентом 4:1, а именно: 16 пикселов по 8 бит в f2 сжимаются до двух 16-ти битовых слов — элементов поля code в h3f2:

```

»hcode      =      h3f2.code;
»whos('hcode')
  Name    Size    Bytes  Class
  hcode   1x2      4  uint16 array
  Grand total is 2 elements using 4 bytes
»dec2bin(double(hcode)) ans =
  1010101101011011
  000001110011000

```

Отметим, что функция dec2bin была использована для отображения индивидуальных битов в h3f2.code. Пренебрегая завершающими тремя битами, которые дополняют данные до объема, кратного 16, (т.е. нули в конце), можно сказать, что это кодирование 32-мя битами эквивалентно кодированию 29 битами мгновенно и однозначно декодируемым блоковым кодом 10101011010110110000011110011.

Как было отмечено в предыдущем примере, функция mat2huff помещает информацию, необходимую для декодирования закодированного входного массива (т. е. его исходные размеры и значения вероятностей символов), в одну структурную переменную MATLAB. Сведения об этой переменной задокументированы в справочной секции самой функции mat2huff.

```

function y = mat2huff(x)
if ndims(x) ~= 2 | ~isreal(x) | (~isnumeric(x) & ~islogical(x))
    error('X must be a 2-D real numeric or logical matrix.');
end
y.size = uint32(size(x));
x = round(double(x));
xmin = min(x(:));
xmax = max(x(:));
pmin = double(int16(xmin));
pmin = uint16(pmin + 32768);    y.min = pmin;
x = x(:)';
h = histc(x, xmin:xmax);
if max(h) > 65535
    h = 65535 * h / max(h);
end
h = uint16(h);    y.hist = h;

map = huffman(double(h));
hx = map(x(:) - xmin + 1);
hx = char(hx)';
hx = hx(:)';
hx(hx == ' ') = [];
ysize = ceil(length(hx) / 16);
hx16 = repmat('0', 1, ysize * 16);
hx16(1:length(hx)) = hx;
hx16 = reshape(hx16, 16, ysize);
hx16 = hx16' - '0';
twos = pow2(15:-1:0);
y.code = uint16(sum(hx16 .* twos(ones(ysize, 1), :, 2))';

```

Обратите внимание на то, что команда $y = mat2huff(x)$ кодирует по Хаффману матрицу x с помощью гистограммных корзин единичной длины, размещенных на интервале от минимального до максимального значения матрицы x . При дальнейшем декодировании данных, закодированных в $y.code$, нужный код Хаффмана должен быть воссоздан по $y.min$, минимальному значению x , и по $y.hist$, гистограмме x . Вместо того, чтобы хранить таблицу кода Хаффмана, функция $mat2huff$ сохраняет информацию о вероятностях символов, достаточную для восстановления этого кода. Имея эти данные, а также исходный размер матрицы x , который записан в поле $y.size$, функция $huff2mat$, рассматриваемая в следующем параграфе этой главы, сможет декодировать $y.code$ и восстановить x . Шаги при построении $y.code$ можно резюмировать следующим образом:

1. Вычислить гистограмму h входа x между минимальным и максимальным значениями x с помощью корзин единичной длины и нормировать ее так, чтобы ее значения помещались в вектор UINT16 .
2. Использовать функцию $huffman$ для построения кода Хаффмана, который называется тар, на основе гистограммы h .
3. Преобразовать x с помощью тар (при этом образуется смешанный массив) и конвертировать результат в символьный массив hx , удаляя пустые символы, которые были вставлены на манер примера 2 при обработке матрицы $h2f2$.
4. Сконструировать вариант вектора hx , в котором символы организованы в виде сегментов из 16-ти символов. Это делается разбиением hx на отрезки по 16 символов ($hx16$ в программе) и приданием им формы матрицы из 16 строк и $y.size$ столбцов, где $y.size = \text{ceil}(\text{length}(hx)/16)$. Напомним, что функция ceil округляет число в сторону положительной бесконечности. Обобщенная функция MATLAB

$$y = \text{reshape}(x, m, n)$$

возвращает матрицу размерами m на n , элементы которой взяты по столбцам матрицы x , и делает сообщение об ошибке, если в x нет $m*n$ элементов.

5. Конвертировать элементы по 16 символов из $hx16$ в 16-ти битовые числа (т. е. в формате uint16). Три завершающие операции выполняют действия, эквивалентные одной компактной команде $y = Tiint16(bin2dec(hx16'))$. Они являются ядром функции bin2dec , которая возвращает десятичный эквивалент двоичной строки (например, $\text{bin2dec}('101')$ дает результат 5), но выполняется существенно быстрее. Функция MATLAB $\text{pow2}(y)$ применяется для построения массива, равного числу 2, возведенному поэлементно в степени y ; т.е. $\text{twos} = \text{pow2}(15:-1:0)$ порождает массив $[32768163848192\dots8421]$.

Пример 3. Кодирование с помощью $mat2huff$.

Чтобы проиллюстрировать степень сжатия при кодировании Хаффмана, рассмотрим 8-ми битовое монохромное изображение 512×512 на рис. 4а). Сжатие этого изображения функцией $mat2huff$ осуществляется посредством следующих команд:

```

>> f = imread ('Tracy.tif');
>> c = mat2huff (f);
>> cr1 = imratio (f, c)
cr1 =
    1.2191

```

С помощью удаления кодовой избыточности, присутствующей в исходном изображении, представленном обычными 8-ми битовыми кодами, нам удалось сжать это изображение примерно до 80 % от его исходного размера (даже при включении в сжатый файл служебной информации).

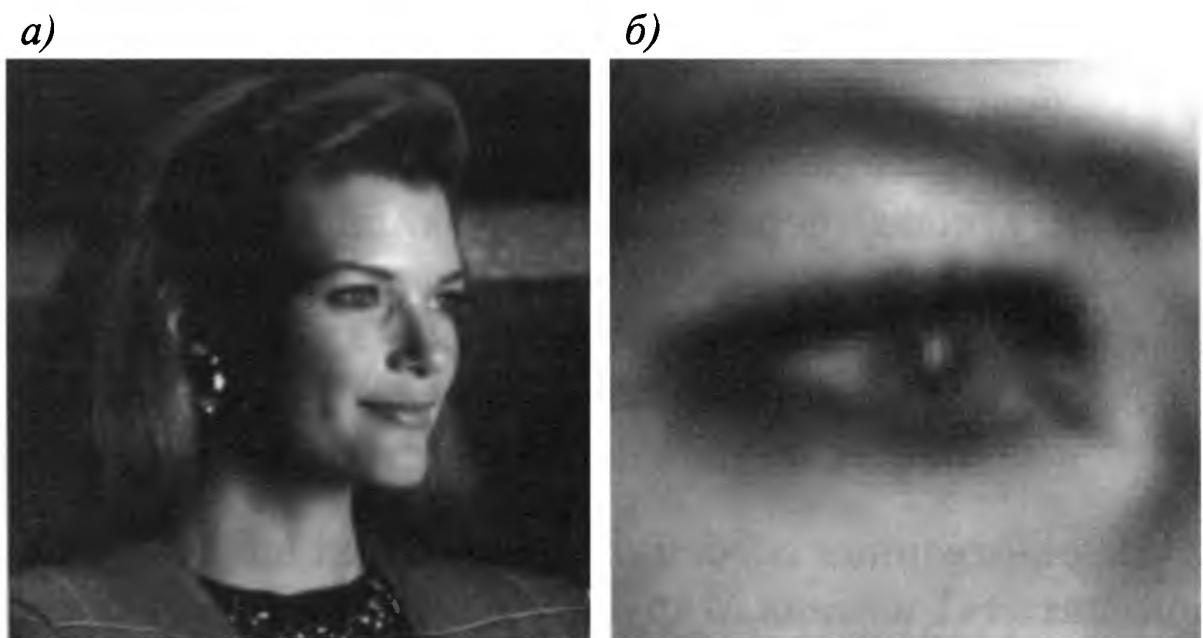


Рисунок 4 – Monoхромное 8-ми битовое изображение женщины размерами 512×512 и увеличенное изображение ее правого глаза

Поскольку выход mat2huff представляет собой структуру, мы запишем ее на диск с помощью функции save:

```

>> save SqueezeTracy c;
>> cr2 = imratio ('Tracy.tif', 'SqueezeTracy.mat')
cr2 =
    1.2365

```

Функция save, аналогично командам Save Workspace и Save Selection As из меню команд, добавляет расширение .mat к создаваемому файлу. Полученный файл – в нашем случае, SqueezeTracy.mat – называется МАТ-файлом. Он является файлом двоичных данных, в котором хранятся имена рабочих переменных и их значения. В нашем случае в файле хранится всего одна рабочая переменная c. Отметим, что небольшая разница в коэффициентах сжатия cr1 и cr2 связана с размерами служебных данных файлов MATLAB.

Декодирование Хаффмана

Изображения, закодированные по Хаффману, будут бесполезными, пока их нельзя продекодировать и получить исходные изображения, по которым они были построены. Для выхода $y = \text{mat2huff}(x)$ декодер должен сначала построить соответствующий код Хаффмана, по которому кодировалось изображение x (по его гистограмме и связанной информации в y), а затем сделать обратное отображение закодированных данных (которые так же извлекаются из y) для реконструкции x . Как видно из прилагаемого далее листинга функции mat2huff , эти действия можно разложить на 5 основных шагов:

1. Извлечь размеры m и n , а также минимальное значение x_{\min} (возможного выхода x) из входной структуры y .
2. Реконструировать код Хаффмана, которым было закодировано x , пропустив его гистограмму через функцию huffman . Полученный результат в программе носит имя map .
3. Построить структуру данных (таблицу переходов и выходов link) для упрощения декодирования данных в $y.\text{code}$ путем ряда эффективных процедур двоичного поиска.
4. Пропустить построенную структуру и закодированные данные (т.е. link и $y.\text{code}$) через С-функцию unravel . Эта функция минимизирует время выполнения операций двоичного поиска и строит выходной декодированный вектор x класса double .
5. Добавляет x_{\min} к каждому элементу x и придает ему форму и размеры исходного изображения x (т.е. форму матрицы с m строками и n столбцами).

Единственная особенность функции huff2mat заключается в вызове С-функции unravel из тела М-функции MATLAB (см. шаг. 4), что делает декодирование изображений нормального разрешения почти мгновенным.

```

function x = huff2mat(y)
if ~isstruct(y) | ~isfield(y, 'min') | ~isfield(y, 'size') | ...
    ~isfield(y, 'hist') | ~isfield(y, 'code')
    error('The input must be a structure as returned by MAT2HUFF.');
end
sz = double(y.size); m = sz(1); n = sz(2);
xmin = double(y.min) - 32768;
map = huffman(double(y.hist));
code = cellstr(char('', '0', '1'));
link = [2; 0; 0]; left = [2 3];
found = 0; tofind = length(map);
while length(left) & (found < tofind)
    look = find(strcmp(map, code{left(1)}));
    if look
        link(left(1)) = -look;
        left = left(2:end);
        found = found + 1;
    else
        len = length(code);
        link(left(1)) = len + 1;
        link = [link; 0; 0];
        code{end + 1} = strcat(code{left(1)}, '0');
        code{end + 1} = strcat(code{left(1)}, '1');
        left = left(2:end);
        left = [left len + 1 len + 2];
    end
end
x = unravel(link, m, n);
x = x + xmin;

```

```

    end
end
x = unravel(y.code', link, m * n);
x = x + xmin - 1;
x = reshape(x, m, n);

```

Как уже отмечалось ранее, декодирование huff2mat основано на сериях операций двоичного поиска с двумя исходящими решениями декодирования. Каждый элемент последовательно просматриваемой строке символов, закодированной по Хаффману (т.е. 0 или 1), переключает двоичное решение декодирования, основанное на таблице переходов и выходов link. Построение link начинается ее инициализацией командой link = [2; 0; 0]. Каждый элемент исходного массива link, состоящего из трех элементов, соответствует закодированной двоичной последовательности в смешанном массиве code. В начале code = cellstr (char(», '0', '1')). Пустая строка code(1) является исходной точкой (начальным состоянием) любого декодирования Хаффмана. Ассоциированное число 2 в link(1) обозначает два свободных состояния декодирования, которые следуют за добавлением к нулевой строке '0' или '1'. Если следующий закодированный бит равен '0', то следующее состояние декодирования есть link(2) (поскольку code (2) = '0' нулевая строка соединяется с '0'); если этот бит равен '1', то новое состояние будет link(3) [с индексом (2+1) или 3, со значением code (3) = '1']. Обратите внимание на то, что соответствующие значения в link равно 0. это означает, что они еще не были обработаны для получения подходящих решений для кода Хаффмана тар. При построении link, если в тар найдена строка (т.е. '0' или '1', что означает настоящее кодовое слово), то соответствующий 0 в link заменяется на соответствующий индекс тар со знаком минус (это декодированная величина). В противном случае новый (положительный) индекс тар вставляется по указателю на два новые состояния (возможные кодовые слова Хаффмана) в логической последовательности (или '00' и '01', или '10' и '11'). Эти новые и еще не обработанные элементы link увеличивают размер link (смешанный массив code также необходимо обновить), после чего процесс построения продолжается до тех пор, пока в link остаются необработанные элементы. Вместо того, чтобы непрерывно сканировать link на предмет необработанных элементов, функция huff2mat строит массив отслеживания с именем left, который в начале процесса равен [2, 3], и каждый раз обновляет его, чтобы в нем находились индексы неисследованных элементов link.

Таблица 3. Таблица декодирования смешанного массива редуцированного источника на рис. 3.

Индекс i	Значение link(i)
1	2
2	4
3	-2
4	-4
5	6
6	-3
7	-1

В табл. 3 показана таблица link, которая строится для кода Хаффмана из примера 2. Если каждый индекс link рассматривать в качестве состояния декодирования i , то каждое двоичное решение кодирования (при просмотре кодовой последовательности слева направо) и/или выход декодера Хаффмана определяется по $link(i)$ по следующим правилам:

- 1) Если $link(i) < 0$ (отрицательно), то кодовое слово Хаффмана уже было продекодировано. Выход декодера есть $|link(i)|$, где $|\cdot|$ обозначает абсолютную величину.
- 2) Если $link(i) > 0$ (положительно), и следующий обрабатываемый бит равен 0, то следующее состояние декодирования – это индекс $link(i)$, т.е. мы полагаем $i = link(i)$.
- 3) Если $link(i) > 0$ и следующий обрабатываемый бит равен 1, то следующее состояние декодирования – это индекс $link(i) + 1$, т.е. мы полагаем $i = link(i) + 1$.

Как ранее отмечалось, положительные компоненты link соответствуют двоичным переходам декодирования, а отрицательные компоненты определяют продекодированные выходные значения. После декодирования каждого кодового слова Хаффмана начинается новый двоичный поиск с индексом link равным $i = 1$. при декодировании строки 101010110101... из примера 2 последовательность состояний переходов: $i = 1, 3, 1, 2, 5, 6, 1, \dots$; а соответствующая последовательность выходов: $- , | - 2 | , - , - , | - 3 | , - \dots$, где знак '-' обозначает отсутствие выхода. Декодированные выходные значения 2 и 3 являются первыми двумя пикселями из первого столбца текстового изображения f2 в примере 2.

С – функция unravel принимает описанную выше структуру link и использует ее в процедуре двоичного поиска, нужного для декодирования входа hx. Блок – схема на рис. 5 отображает основные действия unravel, которые производятся в процессе принятия решений, которые объяснялись вместе с табл. 3. Отметим, что при написании программы на С потребовалось вносить некоторые поправки, связанные с тем, что индексы массивов в С начинаются с 0, а не с 1.

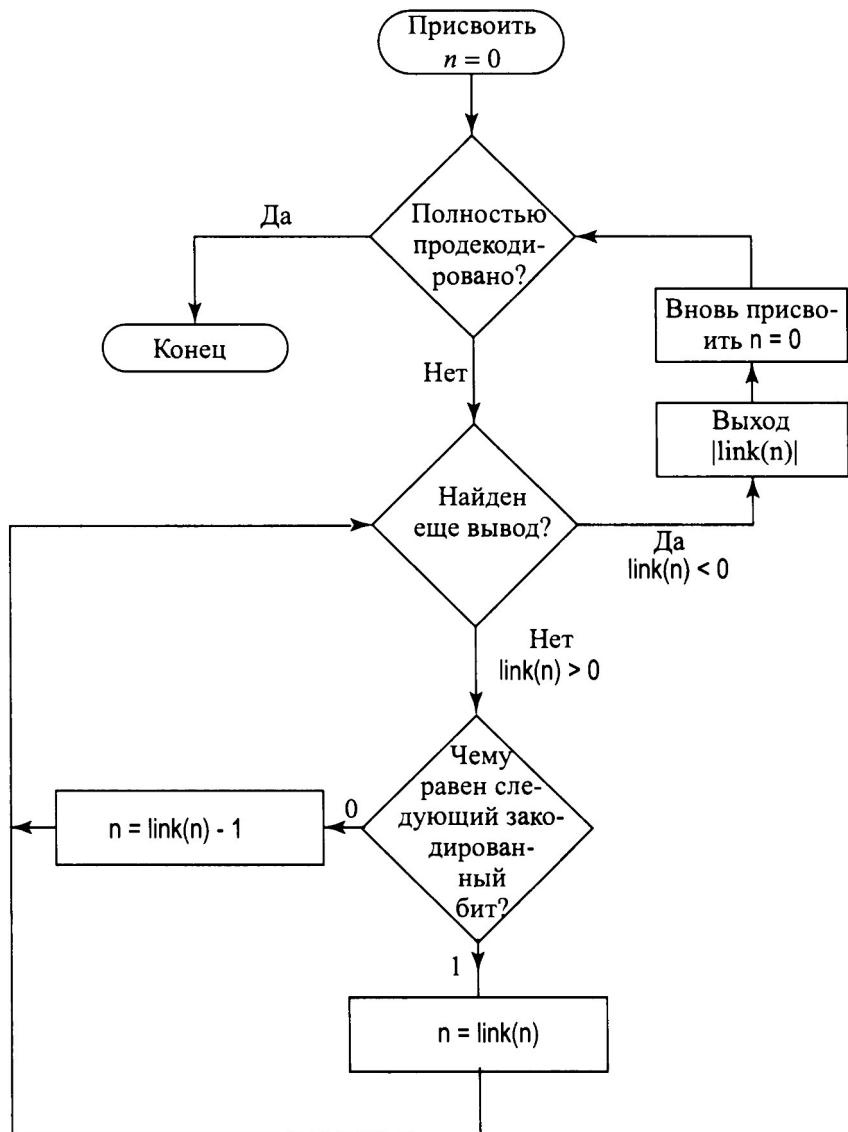


Рисунок 5 - Блок – схема С- функции `unravel`

В систему MATLAB можно внедрять функции, написанные на языках С и Fortran, что служит двум целям:

(1) появляется возможность вызывать в MATLAB огромное количество имеющихся программ на С и Fortran без необходимости их переписывания и отлаживания в виде М- файлов, и

(2) можно оптимизировать критически важные вычислительные процессы, т.е. алгоритмы, реализации которых на MATLAB работают недостаточно быстро, запрограммировать на С или на Fortran более эффективно, воспользовавшись близостью этих языков к машинному коду.

Пример 4. Декодирование с помощью `huff2mat`.

Изображение из примера 3 можно продекодировать следующей последовательностью команд:

```
>> load SqueezeTracy;
>> g = huff2mat (c);
>> f = imread ('Tracy.tif');
```

```

>> rmse = compare (f, g)
rmse =
    0

```

Обратите внимание на то, что процесс кодирования – декодирования полностью сохраняет информацию: среднеквадратическая ошибка между исходным и декодированным изображением равна нулю. В силу того, что значительная часть работы в функции huff2mat выполняется С – функцией unravel, время ее исполнения немного меньше времени кодирования функцией mat2huff. Отметим также использование функции load для повторной загрузки данных из МАТ – файла, в котором был сохранен результат кодирования, полученный в примере 3.

Межпиксельная избыточность

Рассмотрим два изображения на рис. 7a) и b), которые имеют практически одинаковые гистограммы. На этих гистограммах можно выделить три определяющие моды, которые означают присутствие на изображениях трех доминирующих интервалов серых тонов. Поскольку здесь уровни серого цвета не являются равномерно распределенными, можно использовать коды переменной длины для сокращения кодовой избыточности, которая будет присутствовать при кодировании пикселов по обычной схеме кодами фиксированной длины:

```

>> f1 = imread ('Random Matches.tif');
>> c1 = mat2huff (f1);
>> entropy (f1)
ans =
    7.4253
>> imratio (f1, c1)
ans =
    1.0704
>> f2 = imread ('Aligned Matches. tif');
>> c2 = mat2huff (f2);
>> entropy (f2)
ans =
    7.3505
>> imratio (f1, c1)
ans =
    1.0821

```

Заметим, что энтропийные оценки первого порядка обоих изображений близки друг к другу (7.4253 и 7.3505 бит/пикセル), и эти изображения одинаково сжимаются функцией mat2huff (с коэффициентами сжатия 1.0704 против 1.0821). Из этого сравнения видно, что коды переменной длины не используют при сжатии очевидное структурное преимущество рис. 7b), на котором спички лежат ровным рядом. На этом изображении имеется явная межпиксельная корреляция, однако эта же корреляция присутствует также на рис. 7a), т.к. значения пикселов можно в значительной степени предсказать

по значениям их соседей. Информация, которую несет каждый отдельный пиксель, достаточно мала. Большая часть визуального вклада индивидуальных пикселов является избыточной, и ее можно восстановить, зная вклад других близких пикселов. Эта зависимость и служит основой для межпиксельной избыточности.

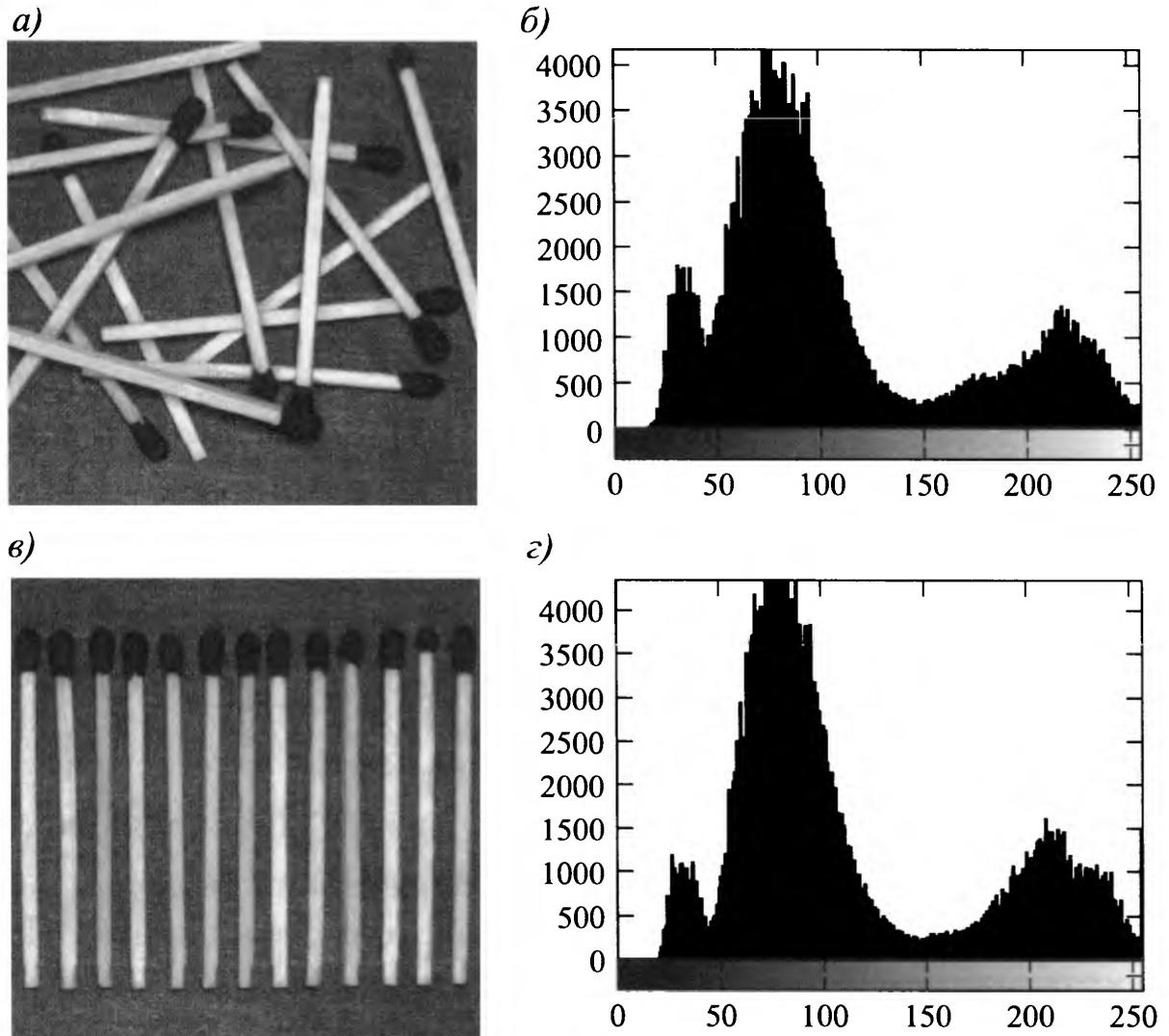


Рисунок 7 – Два изображения и их полутоноевые гистограммы

Для того, чтобы сократить межпиксельную избыточность, двумерные массивы, используемые для представления зрительных образов, необходимо преобразовать в более эффективный (но «визуальный») формат. Например, для представления изображений можно использовать разность примыкающих друг к другу пикселов. Преобразования такого типа (они удаляют межпиксельную избыточность) принято называть *отображениями*. Отображения называются обратимыми, если элементы исходного изображения можно однозначно восстановить (реконструировать) по множеству отображенных данных.

Простая процедура отображения показана на рис. 8. Этот подход, называемый кодированием с предсказанием без потерь, удаляет межпиксельную избыточность с помощью вычитания и кодирования лишь

новой (добавочной информации) пикселов. Новая информация пикселов определяется как разность между реальным значением пикселя и величиной его предсказания. Как видно, система состоит из кодера и декодера, и каждый из них имеет один и тот же блок предсказатель. Когда очередной пиксель входного изображения, обозначаемый f_n , поступает на вход кодера, предсказатель строит прогноз (оценку значения этого пикселя), основанный на некотором наборе предыдущих входных пикселов. Затем выход предсказателя округляется до ближайшего целого, обозначаемого \hat{f}_n , и используется для нахождения разности или ошибки предсказания

$$e_n = f_n - \hat{f}_n.$$

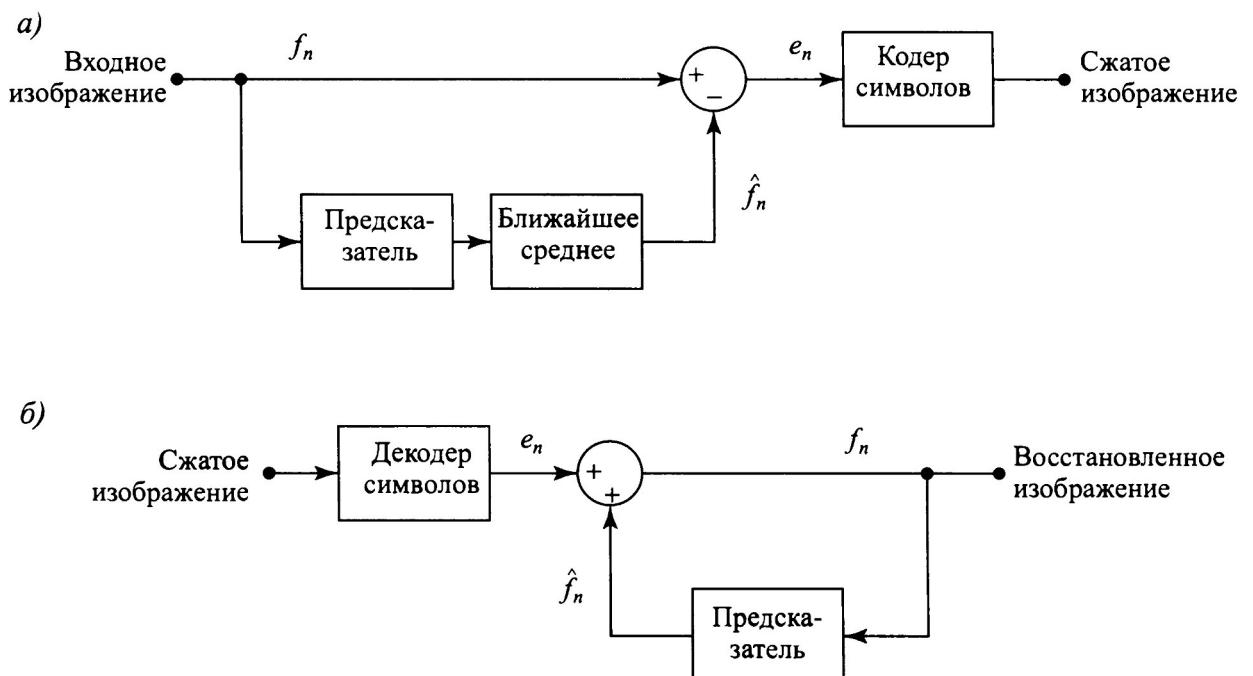


Рисунок 8 – Модель кодирования с предсказанием без потери информации: а) кодер; б) декодер

Эта ошибка кодируется кодом переменной длины (кодером символов), и тем самым генерируется очередной элемент сжатого потока данных. Декодер на рис. 8б) восстанавливает значение e_n по принятому кодовому слову переменной длины и совершает обратное преобразование

$$f_n = e_n + \hat{f}_n.$$

Для построения предсказания \hat{f}_n могут использоваться различные локальные, глобальные или адаптивные методы. Однако в большинстве случаев принято вычислять предсказание в виде линейной комбинации m предыдущих пикселов:

$$\hat{f}_n = \text{round} \left[\sum_{i=1}^m \alpha_i f_{n-i} \right],$$

$i = 1, 2, \dots, m$ — коэффициенты предсказания. Для одномерного кодирования с предсказанием это выражение можно переписать в виде

$$f(x, y) = \text{round} \left[\sum_{i=1}^m f_i \{x, y - i\} \right]$$

где каждая индексированная переменная теперь выражена явно в виде функции пространственных координат x и y . Отметим, что при таком подходе предсказание $f(x, y)$ зависит лишь от значений пикселов одной текущей обрабатываемой строки.

М-функции mat2lpc и lpc2mat реализуют процедуру кодирования/декодирования с предсказанием, описанную выше (за вычетом шагов символьного кодирования/декодирования). Функция кодирования mat2lpc использует цикл for для построения одновременного прогноза всех пикселов входа x . На каждом шаге итерации массив xs , который в начале равен x , сдвигается на одну позицию вправо (с нулевым заполнением слева), умножается на соответствующие коэффициенты предсказания и прибавляется к суммирующему массиву p . Поскольку обычно число коэффициентов линейного предсказания мало, процедура работает достаточно быстро. Отметим, что при исполнении следующей программы, если фильтр предсказания f не задан на входе, то используется простейший фильтр, состоящий из одного числа 1.

```
function y = mat2lpc(x, f)
error(nargchk(1, 2, nargin));
if nargin < 2
    f = 1;
end
x = double(x);
[m, n] = size(x);
p = zeros(m, n);
xs = x;      zc = zeros(m, 1);
for j = 1:length(f)
    xs = [zc xs(:, 1:end - 1)];
    p = p + f(j) * xs;
end
y = x - round(p);
```

Функция декодирования lpc2mat совершает действия, обратные преобразованиям функции кодирования mat2lpc. Как видно из следующего текста этой программы, она совершает n итераций цикла for, где n обозначает число столбцов закодированной входной матрицы y . На каждой итерации вычисляется только один столбец выхода декодера x , поскольку для этого требуется знать все предыдущие столбцы. Для ускорения вычислений цикла for матрица x расширяется до максимально необходимого размера перед началом цикла и заполняется нулями. Заметьте также, что предсказания вычисляются здесь точно в том же порядке, что и в функции lpc2mat. В этом случае удается избежать ошибок округления в вычислениях с плавающей точкой.

```
function x = lpc2mat(y, f)
error(nargchk(1, 2, nargin));
if nargin < 2
    f = 1;
```

```

end
f = f(end:-1:1);
[m, n] = size(y);
order = length(f);
f = repmat(f, m, 1);
x = zeros(m, n + order);
for j = 1:n
    jj = j + order;
    x(:, jj) = y(:, j) + round(sum(f(:, order:-1:1) .* ...
        x(:, (jj - 1):-1:(jj - order)), 2));
end
x = x(:, order + 1:end);

```

Пример 5. Кодирование с предсказанием без потери информации.
Рассмотрим кодирование изображения на рис. 7в) с помощью простого линейного предсказателя первого порядка

$$f(x, y) = \text{round}[af(x, y - 1)].$$

Такую форму предсказания принято называть *предсказанием по предыдущему пикселу*, а соответствующая схема кодирования называется *дифференциальным кодированием* или *кодированием по предыдущему пикселу*. На рис. 9а) приведено изображение ошибки предсказания при $a = 1$. Здесь уровень яркости 128 соответствует нулевой ошибке предсказания, а ненулевые (положительные и отрицательные) ошибки усилены функцией `mat2gray`, что делает их более яркими или более темными оттенками.

```

>> f = imread('Aligned Matches.tif');
>> e = mat2pc(f);
>> imshow(mat2gray(e));
>> entropy(e)
ans =
    5.9727

```

Заметьте, что энтропия ошибки предсказания e существенно меньше энтропии исходного изображения f . Энтропия уменьшилась с 7.3505 бит/пикセル до 5.9727 бит/пиксел, и это при том, что для m -битных изображений требуются $(t + 1)$ -битные числа для корректного представления полученной последовательности ошибок. Такое уменьшение энтропии означает, что изображение, составленное из ошибок предсказания, можно кодировать более эффективно по сравнению с исходным изображением, а как раз это и является целью отображения. В итоге мы имеем

```

>> c = mat2huff(e);
>> cr = imratio(f, c) cg =
    1.3311

```

откуда видно, что коэффициент сжатия вырос, как и ожидалось, с 1.0821 (при прямом кодировании уровней изображения по Хаффману) до 1.3311.

6 4 2 0 2 4 6

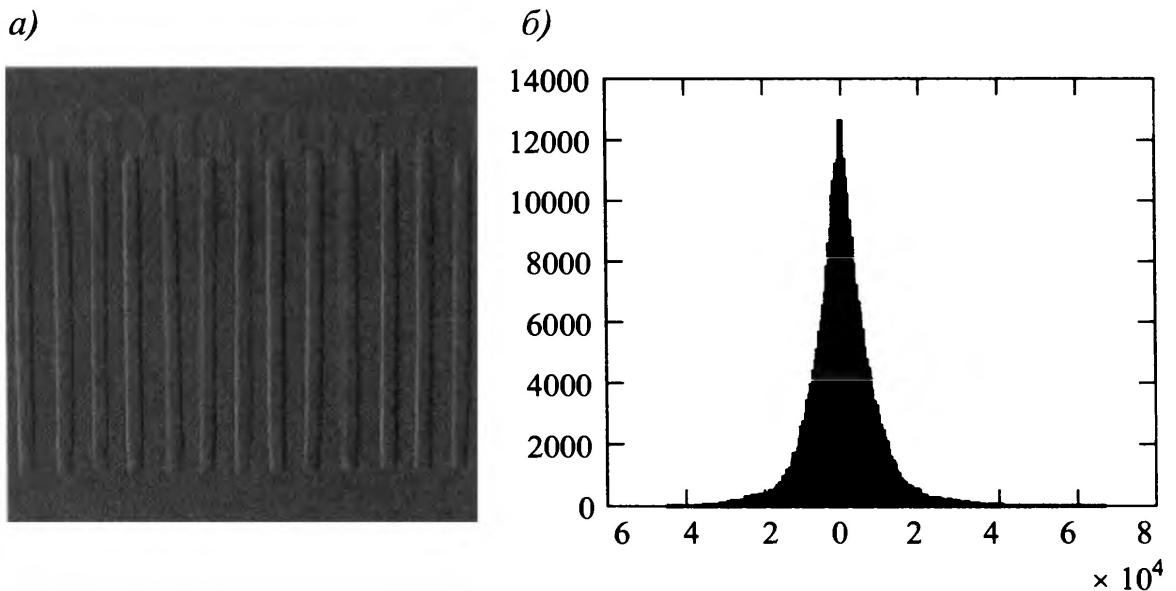


Рисунок 9 - а) Изображение ошибки предсказания для изображения на рис. 7в); б) Гистограмма ошибки предсказания
Гистограмма ошибок предсказания, приведенная на рис. 9б), вычислена командами

```

» [h, x] = hist(e(:) * 512, 512);
>> figure; bar(x, h, 'k');
изображением f:
Ipc2mat(huff2mat(c)); >> compare
(f, g) ans =
0

```

Обратите внимание на высокий пик возле 0 и на относительно небольшую дисперсию гистограммы по сравнению с гистограммой распределения уровней исходного изображения (см. рис. 7г)). Все это означает удаление значительной части межпиксельной избыточности в процессе дифференциального предсказания. Мы завершим этот пример демонстрацией полного сохранения информации при кодировании по этой схеме. Для этого мы декодируем s и сравниваем результат с начальным изображением f :

```

Ipc2mat(huff2mat(c)); >> compare
(f, g) ans =
0

```

Визуальная избыточность

В отличие от кодовой и межпиксельной избыточности, визуальная избыточность связана с настоящей визуальной информацией, поддающейся количественному измерению. Ее удаление желательно, поскольку эта информация сама по себе не существенна для обычного визуального восприятия. Поскольку уменьшение визуальных избыточных данных приводит к потери части количественной информации, этот процесс принято называть *квантованием*. Такая терминология хорошо согласуется с обычным смыслом этого слова, которое означает представление широкого набора

величин с помощью ограниченного множества допустимых значений. Эта процедура не является обратимой (после ее совершения происходит невосполнимая утрата части информации), т.е. квантование приводит к потере части сжимаемых данных.

Пример 6. Сжатие посредством квантования.

Рассмотрим изображения на рис.10. Рис. 10 a) показывает черно-белое изображение с 256 возможными градациями яркости. На рис. 10 b) представлено то же самое изображение после равномерного квантования на 16 уровней (4 бита). Полученный в результате коэффициент сжатия равен 2:1. Заметим, что на некоторых областях изображения, которые были гладкими, появились ложные контуры. В этом проявляется обычный видимый эффект слишком грубого представления уровней яркости изображения.

Рис. 10 c) иллюстрирует значительное улучшение изображения, возможное при использовании квантования, которое учитывает визуальные особенности зрительного аппарата человека. Несмотря на то, что коэффициент сжатия при этом квантовании также равен 2:1, ложные контуры значительно ослаблены за счет некоторой дополнительной, но мало заметной зернистости. Заметим, что в обоих случаях полное восстановление изображения после декодирования остается невозможным (квантование является необратимым преобразованием).

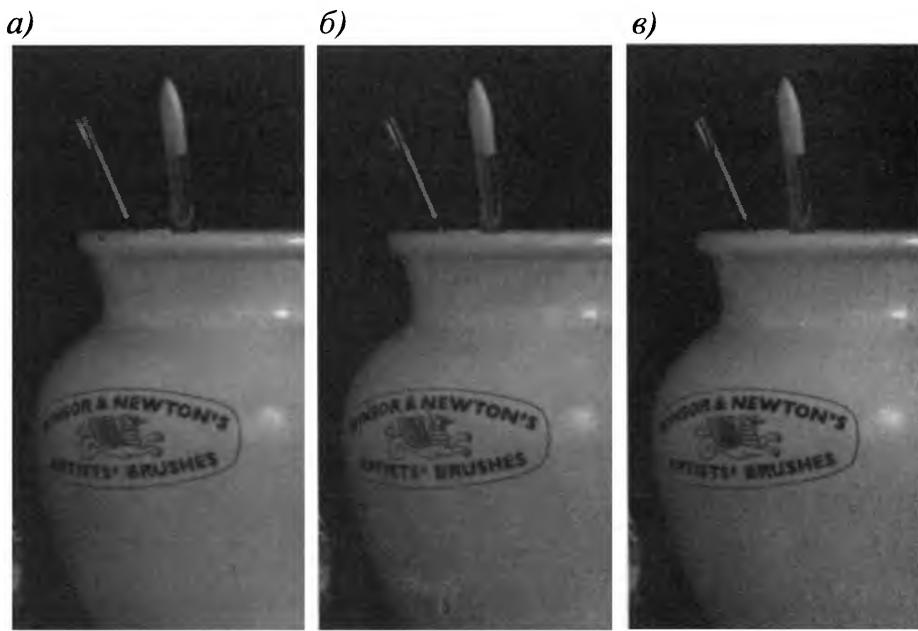


Рисунок 10 – а) Исходное изображение; б) Равномерное квантование на 16 уровней; в) Квантование IGS на 16 уровней

Метод квантования. Он учитывает свойственную глазу чувствительность к контурам и борется с ложными контурами с помощью добавления ко всем пикселам малых псевдослучайных чисел, которые строятся по младшим битам значений окрестных пикселов до из квантования. Поскольку младшие биты, как правило, достаточны случайны, то это действие эквивалентно добавлению некоторого уровня случайности,

зависящего от локальных свойств изображения, что приводит к разрушению четкости ровных перепадов, которые выглядят как ложные контуры. Ниже приводится текст функции **quantize**, которая совершает оба типа квантования: метод IGS и стандартное отсекание младших битов. Заметим, что реализация IGS векторизована так, что вход x обрабатывается по столбцам за один шаг. Чтобы построить столбец результата по 4 бита на рис. 10 ν), сумма по столбцам s (которая обнуляется в начале) вычисляется как сумма одного столбца x и четырех наименее значимых битов существующей (ранее вычисленной) суммы. Если четыре самые значимые бита некоторой величины x равны 1111_2 , то вместо этого добавляется 0000_2 . Затем четыре самые значимые бита полученной суммы используются для кодирования значения пикселя обрабатываемого столбца.

```
function y = quantize(x, b, type)
error(nargchk(2, 3, nargin));
if ndims(x) ~= 2 | ~isreal(x) | ...
    ~isnumeric(x) | ~isa(x, 'uint8')
    error('The input must be a UINT8 numeric matrix.');
end
lo = uint8(2 ^ (8 - b) - 1);
hi = uint8(2 ^ 8 - double(lo) - 1);
if nargin < 3 | ~strcmpi(type, 'igs')
    y = bitand(x, hi);
else
    [m, n] = size(x);                                s = zeros(m, 1);
    hitest = double(bitand(x, hi)) ~= hi;           x = double(x);
    for j = 1:n
        s = x(:, j) + hitest(:, j) .* double(bitand(uint8(s), lo));
        y(:, j) = bitand(uint8(s), hi);
    end
end
```

Модифицированное квантование яркости является весьма типичным представителем большого семейства процедур квантования, которые оперируют напрямую со значениями яркости пикселов сжимаемых изображений. Они часто приводят к понижению как пространственного, так и яркостного разрешения изображений. Если изображение сначала отображается с целью сокращения межпиксельной избыточности, то дальнейшее квантование может привести к другим типам искажения в виде размытия тонких контуров (т. е. к потере высокочастотных деталей), когда двумерное преобразование в частотной области используется для декорреляции данных.

Пример 7. Сочетание квантования IGS, кодирования с предсказанием без потерь и кодирования Хаффмана.

Несмотря на то, что квантование, использованное при построении сжатого изображения на рис. 10, ν), удаляет большую часть визуальной избыточности с весьма малым влиянием на воспринимаемое глазом качество изображения, дальнейшее сжатие можно получить, используя методы, изложенные в предыдущих двух параграфах, которые позволяют уменьшить межпиксельную и кодовую избыточности. На самом деле, мы можем более чем вдвое повысить сжатие, достигнутое при квантовании IGS. Следующая последовательность команд выполняет квантование IGS, кодирование с

предсказанием и кодирование Хаффмана, что позволяет сжать изображение на рис. 10a) в четыре раза и более от его исходного размера:

```
>> f = imread(CBrushes.tif);
>> q = quantize(f, 4, 'igs');
>> qs = double(q) / 16;
>> e = mat2lpc(qs);
>> c = mat2huff(e);
>> imratio(f, c)
ans =
    4.1420
```

Закодированное изображение с можно разжать с помощью обратной последовательности операций (только без «обратного квантования»):

```
» ne = huff2mat(c);
>> nqs = lpc2mat(ne); » nq = 16
*nqs;
» compare(q, nq) ans =
    0
» rmse = compare(f, nq) rmse =
    6.8382
```

Отметим, что среднеквадратическое отклонение разжатого изображения от исходного равно приблизительно 7 уровням яркости, что происходит исключительно из-за процедуры квантования.

Стандарты сжатия JPEG

Изложенные методы оперируют непосредственно с пикселями изображения, поэтому они называются *методами квантования в пространственной области*. Рассмотрим семейство популярных стандартов сжатия, которые основаны на модификациях преобразованного изображения. Наша цель заключается в описании использования двумерных преобразований при сжатии изображений, а также в рассмотрении некоторых новых приемов сокращения избыточности. Тогда читатель сможет оценить современное состояние дел в области сжатия изображений. Представленные стандарты (на самом деле, мы рассмотрим лишь некоторые приближения к ним) разработаны для обработки широкого круга изображений, которым требуются весьма различные условия сжатия.

При *трансформационном кодировании* используется некоторое обратимое линейное преобразование, например, дискретное преобразование Фурье DFT или *дискретное косинусное преобразование DCT* (Discrete Cosinus Transform), которое задается уравнениями

$$T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \alpha_M(u) \alpha_N(v) \cos \left[\frac{(2x+1)u\pi}{2M} \right] \cos \left[\frac{(2y+1)v\pi}{2N} \right],$$

где

$$\alpha_M(u) = \begin{cases} \sqrt{\frac{1}{M}} & \text{при } u = 0, \\ \sqrt{\frac{2}{M}} & \text{при } u = 1, 2, \dots, M - 1 \end{cases}$$

(и аналогичная формула для функции $\alpha_N(v)$). При этом изображение $f(x,y)$ отображается в некоторое множество коэффициентов преобразования, которые затем квантуются и кодируются. Для подавляющего числа естественных изображений существенная часть коэффициентов преобразования имеет малую амплитуду, и их можно грубо квантовать (или даже совсем отбросить), внеся тем самым в изображение незначительные искажения, практически незаметные для глаза.

JPEG

Одним из самых известных и широко применяемых универсальных стандартов сжатия изображений с непрерывными тонами является стандарт JPEG (эта аббревиатура образована из слов Joint Photographic Experts Group, объединенная группа экспертов по фотографии). В *базовой системе кодирования JPEG*, основанной на дискретном косинусном преобразовании и годящейся для большинства приложений сжатия, входные и выходные изображения ограничены 8-и битным форматом представления компонент яркости и цветности, а длина представления коэффициентов DCT равна 11 битам. Как видно на упрощенной блок-схеме рис. 11a), само сжатие совершается за четыре шага: извлечение подизображений размерами 8x8, вычисление DCT, квантование и присвоение кодов переменной длины, т. е. кодирование.

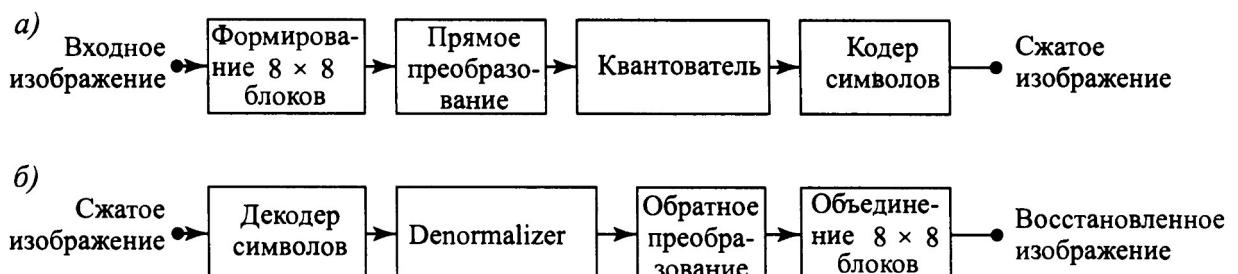


Рисунок 11 - Блок-схема JPEG: а) кодер; б) декодер

Первый шаг процесса сжатия JPEG заключается в разделении входного изображения на непересекающиеся блоки пикселов размерами 8x8. Это выполняется последовательно в направлении слева направо и сверху вниз. Каждый блок 8x8 (подизображение) подвергается определенной обработке. Все его 64 пикселя сдвигаются вычитанием числа 2^{t-1} , где 2^t — это число уровней яркости изображения, после чего вычисляется дискретное косинусное преобразование DCT блока. Полученные коэффициенты нормируются и квантуются по правилу

$$hatT(u, v) = \text{round} \left[\frac{T(u, v)}{Z(u, v)} \right],$$

где $T(u,v)$ при $u,v = 0,1,2,\dots,7$ обозначают нормированные и квантованные коэффициенты, $T(u,v)$ — коэффициенты DCT текущего блока 8×8 изображения $f(x,y)$, а $Z(u,v)$ — нормирующая матрица преобразования, предписанная стандартом и приведенная на рис. 12a). Если менять элементы матрицы $Z(u,v)$ пропорционально некоторому числу, то это позволяет варьировать коэффициент достигаемого сжатия, что отражается на качестве реконструированного изображения.

a)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>16</td><td>11</td><td>10</td><td>16</td><td>24</td><td>40</td><td>51</td><td>61</td></tr> <tr><td>12</td><td>12</td><td>14</td><td>19</td><td>26</td><td>58</td><td>60</td><td>55</td></tr> <tr><td>14</td><td>13</td><td>16</td><td>24</td><td>40</td><td>57</td><td>69</td><td>56</td></tr> <tr><td>14</td><td>17</td><td>22</td><td>29</td><td>51</td><td>87</td><td>80</td><td>62</td></tr> <tr><td>18</td><td>22</td><td>37</td><td>56</td><td>68</td><td>109</td><td>103</td><td>77</td></tr> <tr><td>24</td><td>35</td><td>55</td><td>64</td><td>81</td><td>104</td><td>113</td><td>92</td></tr> <tr><td>49</td><td>64</td><td>78</td><td>87</td><td>103</td><td>121</td><td>120</td><td>101</td></tr> <tr><td>72</td><td>92</td><td>95</td><td>98</td><td>112</td><td>100</td><td>103</td><td>99</td></tr> </table>	16	11	10	16	24	40	51	61	12	12	14	19	26	58	60	55	14	13	16	24	40	57	69	56	14	17	22	29	51	87	80	62	18	22	37	56	68	109	103	77	24	35	55	64	81	104	113	92	49	64	78	87	103	121	120	101	72	92	95	98	112	100	103	99	б)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>0</td><td>1</td><td>5</td><td>6</td><td>14</td><td>15</td><td>27</td><td>28</td></tr> <tr><td>2</td><td>4</td><td>7</td><td>13</td><td>16</td><td>26</td><td>29</td><td>42</td></tr> <tr><td>3</td><td>8</td><td>12</td><td>17</td><td>25</td><td>30</td><td>41</td><td>43</td></tr> <tr><td>9</td><td>11</td><td>18</td><td>24</td><td>31</td><td>40</td><td>44</td><td>53</td></tr> <tr><td>10</td><td>19</td><td>23</td><td>32</td><td>39</td><td>45</td><td>52</td><td>54</td></tr> <tr><td>20</td><td>22</td><td>33</td><td>38</td><td>46</td><td>51</td><td>55</td><td>60</td></tr> <tr><td>21</td><td>34</td><td>37</td><td>47</td><td>50</td><td>56</td><td>59</td><td>61</td></tr> <tr><td>35</td><td>36</td><td>48</td><td>49</td><td>57</td><td>58</td><td>62</td><td>63</td></tr> </table>	0	1	5	6	14	15	27	28	2	4	7	13	16	26	29	42	3	8	12	17	25	30	41	43	9	11	18	24	31	40	44	53	10	19	23	32	39	45	52	54	20	22	33	38	46	51	55	60	21	34	37	47	50	56	59	61	35	36	48	49	57	58	62	63
16	11	10	16	24	40	51	61																																																																																																																												
12	12	14	19	26	58	60	55																																																																																																																												
14	13	16	24	40	57	69	56																																																																																																																												
14	17	22	29	51	87	80	62																																																																																																																												
18	22	37	56	68	109	103	77																																																																																																																												
24	35	55	64	81	104	113	92																																																																																																																												
49	64	78	87	103	121	120	101																																																																																																																												
72	92	95	98	112	100	103	99																																																																																																																												
0	1	5	6	14	15	27	28																																																																																																																												
2	4	7	13	16	26	29	42																																																																																																																												
3	8	12	17	25	30	41	43																																																																																																																												
9	11	18	24	31	40	44	53																																																																																																																												
10	19	23	32	39	45	52	54																																																																																																																												
20	22	33	38	46	51	55	60																																																																																																																												
21	34	37	47	50	56	59	61																																																																																																																												
35	36	48	49	57	58	62	63																																																																																																																												

Рисунок 12 - а) Нормировочная матрица JPEG; б) Последовательность упорядочения коэффициентов по зигзагу в JPEG

После того, как коэффициенты DCT всех блоков будут проквантованы, элементы двумерных матриц $T(u, v)$ переупорядочиваются в зигзагообразном порядке, показанном на рис. 12б) последовательными числами $0,1,2,\dots, 63$, образуя одномерный массив. Поскольку полученные одномерные векторы (квантованных коэффициентов) упорядочены по возрастанию пространственной частоты, то сами они уменьшаются в этом направлении, и кодер символов на рис. 11а) оптимизирован с учетом вероятного появления длинных серий нулей, которые обычно возникают после упорядочения зигзагом. В частности, ненулевые коэффициенты AC [т.е. все $T(u,v)$, кроме случая $u = v = 0$] кодируются с использованием кодов переменной длины, которые определяют величины этих коэффициентов и число следующих за ними нулей. Коэффициенты DC [т. е. $\hat{T}(0,0)$] кодируются разностными кодами по отношению к коэффициентам DC предыдущих под изображений. В стандарте имеются таблицы для кодов Хаффмана коэффициентов DC и AC, которые используются при сжатии по умолчанию, но пользователь может построить свои собственные таблицы, а также задать свою нормировочную матрицу. В последнем случае всю эту информацию придется добавить к сжатым данным в зарезервированную область параметров.

Следующая M-функция моделирует процесс базового кодирования полная реализации стандарта JPEG:

```

function y = im2jpeg(x, quality)
error(nargchk(1, 2, nargin));
if ndims(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
    error('The input must be a UINT8 image.');
end
if nargin < 2
    quality = 1;
end
m = [16 11 10 16 24 40 51 61
      12 12 14 19 26 58 60 55
      14 13 16 24 40 57 69 56
      14 17 22 29 51 87 80 62
      18 22 37 56 68 109 103 77
      24 35 55 64 81 104 113 92
      49 64 78 87 103 121 120 101
      72 92 95 98 112 100 103 99] * quality;

order = [1 9 2 3 10 17 25 18 11 4 5 12 19 26 33 ...
          41 34 27 20 13 6 7 14 21 28 35 42 49 57 50 ...
          43 36 29 22 15 8 16 23 30 37 44 51 58 59 52 ...
          45 38 31 24 32 39 46 53 60 61 54 47 40 48 55 ...
          62 63 56 64];

[xm, xn] = size(x);
x = double(x) - 128;
t = dctmtx(8);
y = blkproc(x, [8 8], 'P1 * x * P2', t, t');
y = blkproc(y, [8 8], 'round(x ./ P1)', m);
y = im2col(y, [8 8], 'distinct');
xb = size(y, 2);
y = y(order, :);

eob = max(x(:)) + 1;
r = zeros(numel(y) + size(y, 2), 1);
count = 0;
for j = 1:xb
    i = max(find(y(:, j)));
    if isempty(i)
        i = 0;
    end
    p = count + 1;
    q = p + i;
    r(p:q) = [y(1:i, j); eob];
    count = count + i + 1;
end
r((count + 1):end) = [];
y.size = uint16([xm xn]);
y.numblocks = uint16(xb);
y.quality = uint16(quality * 100);
y.huffman = mat2huff(r);

```

В соответствии с блок-схемой на рис. 11а), функция `im2jpeg` обрабатывает отдельные подизображения или блоки 8x8 входного изображения `x` (последовательно блок за блоком, но не все изображение сразу). Для упорядочения вычислений используются две функции обработки блоков `blkproc` и `im2col`. Функция `blkproc` имеет стандартный синтаксис

`B = blkproc(A, [M N], FUN, P1, P2, ...).`

Эта функция унифицирует весь процесс поблочной обработки изображения. Она принимает входное изображение `A` вместе с размерами блоков `[M N]`, по которым требуется обрабатывать изображение, а также

функцию FUN, которая будет обрабатывать эти блоки, и некоторый опционный набор аргументов P1, P2, ... функции FUN. Затем функция blkproc разделяет A на блоки MxN (используя заполнение нулями там, где блок выходит за рамки изображения), вызывает функцию FUN для каждого блока с заданными параметрами P1, P2, . . . , и, наконец, помещает результат в выходное изображение B.

Другой функцией поблочной обработки, используемой в процедуре im2jpeg, является функция im2col. Если функция blkproc не годится для реализации специфических поблочных действий, функция im2col может использоваться для переупорядочения входа так, чтобы операции можно было совершить проще и эффективнее (например, применяя векторизацию). Выходом im2col служит матрица, в которой каждый столбец состоит из элементов одного блока входного изображения. Ее стандартизованный формат имеет вид

$$B = \text{im2col}(A, [M N], 'distinct'),$$

где параметры A, B и [M N] имеют тот же смысл, что и в функции blkproc, а строка 'distinct' сообщает im2col о том, что обрабатываемые блоки не пересекаются; другая строка-параметр 'sliding' сигнализирует о том, что необходимо строить один столбец B для каждого пикселя в A (как если бы блок скользил по изображению).

В процедуре im2jpeg функция blkproc применяется для облегчения вычисления DCT, нормирования и квантования, а функция im2col упрощает упорядочение квантованных коэффициентов и отслеживание серий нулей. В отличие от стандарта JPEG, функция im2jpeg обнаруживает только последнюю серию нулей в каждой упорядоченной последовательности коэффициентов квантованного блока, заменяя всю эту серию одним символом eob (End Of Block). Наконец, отметим, что в системе MATLAB имеется эффективная утилита, написанная на базе FFT, которая выполняет DCT для больших изображений. Тем не менее, im2jpeg использует альтернативную матричную

формулу¹

$$T = HFH^T,$$

где F — это блок 8x8 изображения $f(x,y)$, H — матрица преобразования DCT, которая строится командой dctmtx(8), а T — результат применения DCT к F. Здесь верхний индекс T обозначает операцию транспонирования. При отсутствии квантования обратное преобразование DCT от T вычисляется по формуле

$$F = H^T TH.$$

Эта формула особенно эффективна при обработке малых квадратных изображений (подобно DCT 8x8 в JPEG). Таким образом, команда

$$y = \text{blkproc}, [8 8], 'P1*x*P2', h, h')$$

вычисляет преобразование DCT изображения x по блокам 8x8 с использованием матрицы преобразования h и транспонированной матрицы h' в качестве параметров P1 и P2, которые являются множителями в формуле

матричного произведения DCT $P1 * x * P2$, играющего роль функционального параметра FUN.

Аналогичная поблочная обработка и преобразование на основе матричных произведений (см. рис. 11б)) применяются в процессе декодирования изображения, сжатого функцией im2jpeg. Приведенная ниже функция jpeg2im выполняет всю необходимую последовательность обратных операций (за исключением квантования). Она использует общую функцию

$A = \text{col2im}(B, [M N], [MM NN], 'distinct')$

для воссоздания двумерного изображения из столбцов матрицы B, где каждый столбец по 64 элемента является блоком 8x8 реконструируемого изображения. Параметры A, B, [M N] и 'distinct' были определены при описании функции im2col, а массив [MM NN] обозначает размеры выходного изображения A.

```
function x = jpeg2im(y)
error(nargchk(1, 1, nargin));
m = [16 11 10 16 24 40 51 61
      12 12 14 19 26 58 60 55
      14 13 16 24 40 57 69 56
      14 17 22 29 51 87 80 62
      18 22 37 56 68 109 103 77
      24 35 55 64 81 104 113 92
      49 64 78 87 103 121 120 101
      72 92 95 98 112 100 103 99];
order = [1 9 2 3 10 17 25 18 11 4 5 12 19 26 33 ...
          41 34 27 20 13 6 7 14 21 28 35 42 49 57 50 ...
          43 36 29 22 15 8 16 23 30 37 44 51 58 59 52 ...
          45 38 31 24 32 39 46 53 60 61 54 47 40 48 55 ...
          62 63 56 64];
rev = order;
for k = 1:length(order)
    rev(k) = find(order == k);
end
m = double(y.quality) / 100 * m;
xb = double(y.numblocks);
sz = double(y.size);
xn = sz(2);
xm = sz(1);
x = huff2mat(y.huffman);
eob = max(x(:));
z = zeros(64, xb); k = 1;
for j = 1:xb
    for i = 1:64
        if x(k) == eob
            k = k + 1; break;
        else
            z(i, j) = x(k);
            k = k + 1;
        end
    end
end
z = z(rev, :);
x = col2im(z, [8 8], [xm xn], 'distinct');
x = blkproc(x, [8 8], 'x .* P1', m);
t = dctmtx(8);
x = blkproc(x, [8 8], 'P1 * x * P2', t', t);
x = uint8(x + 128);
```

Пример . Сжатие JPEG.

На рис. 13 a) и \bar{b}), приведены два кодированные и декодированные изображения JPEG, которые являются приближениями одного и того же исходного черно-белого изображения на рис. 4 a). Первый результат, имеющий коэффициент сжатия примерно 18 к 1, получен прямым использованием стандартной нормирующей матрицы на рис. 12 a). Второе изображение, имеющее коэффициент сжатия 42 к 1, построено при умножении нормирующей матрицы на 4, что соответствует более грубому квантованию.

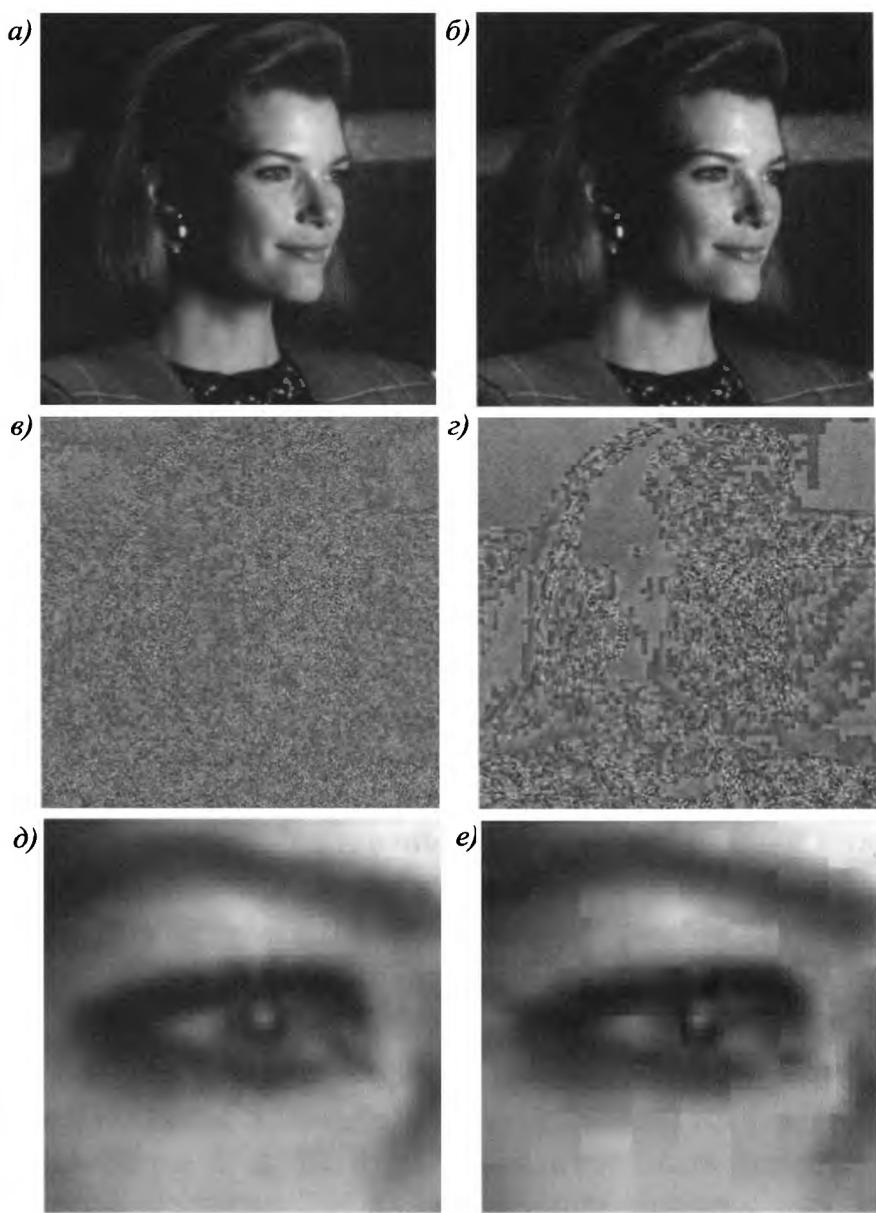


Рисунок 13 – Левая колонка: приближение рис. 4 a), с использованием DCT и нормирующие матрицы рис 12 a). Правая колонка: аналогичные результаты с нормирующей матрицей, увеличенной в 4 раза

Разности между исходным изображением рис. 4 a) и восстановленными изображениями рис. 13 a) и \bar{b}) построены, соответственно, на рис.13 c) и d). Тоны обоих изображений были усилены для лучшей визуализации

имеющихся отклонений. Соответствующие средние ошибки RMS равны 2.5 и 4.4 уровня яркости. Влияние этих ошибок на качество изображений видно более отчетливо на увеличенных участках изображений, приведенных на рис. 13 d) и e). Эти изображения позволяют лучше оценить тонкие различия между обоими восстановленными изображениями [исходное увеличенное изображение имеется на рис. 4 b)]. Обратите внимание на *артефакты блочности*, присутствующие на обоих увеличенных приближениях.

Изображения на рис. 13 и обсуждавшиеся только что численные результаты получены с помощью следующей последовательности команд:

```
>> f = imreadCTracy.tif);
>> cl = im2jpeg(f);
>> fl = jpeg2im(cl);
>> imratio(f, cl)
ans =
    18.2450
>> compare(f, fl, 3)
ans =
    2.4675
>> c4 = im2jpeg(f, 4);
>> f4 = jpeg2im(c4) ;
>> imratio(f, c4)
ans =
    41.7826
>> compare(f, f4, 3)
ans =
    4.4184
```

Эти результаты отличаются от тех, которые получаются при сжатии настоящим стандартом JPEG, поскольку наша программа im2jpeg лишь моделирует процесс кодирования Хаффмана, заложенный в стандарт JPEG. Имеется два принципиальных отличия, заслуживающих внимания:

(1)в стандарте все серии нулевых коэффициентов кодируются по Хаффману, а в im2jpeg кодируется лишь завершающая серия нулей каждого блока;

(2)кодер и декодер стандарта используют (по умолчанию) предписанный код Хаффмана, в то время как im2jpeg получает информацию, необходимую для реконструкции таблицы кодовых слов из самого изображения.

Если использовать полный стандарт JPEG, то коэффициент сжатия повысится еще примерно в два раза.

Сжатие или компрессия цифровых изображений является одной из наиболее часто применяемых к ним операций. Это связано с тем, что цифровые изображения, а особенно подвижные изображения - цифровое видео, имеют очень большие объемы и требуют чрезвычайно большой пропускной способности каналов связи для своей передачи. Эффективное сжатие позволяет существенно снизить требования к пропускной

способности каналов связи при передаче изображений и видео, а также к емкости систем их сравнения.

Наиболее известным и широко распространенным во многих приложениях, в том числе стандарта MPEG передачи видео в системах цифрового телевидения, является алгоритм сжатия JPEG.

Основы алгоритма JPEG

Популярный алгоритм кодирования изображений JPEG (Joint Photographers Experts Group) был утвержден в 1991 году в качестве одного из стандартов сжатия изображений.

Алгоритм базируется на двух основополагающих вещах:

1. Кодировании не самих данных, а некоторого линейного преобразований от этих данных.
2. Квантовании (округлении) коэффициентов линейного преобразования.

В качестве линейного преобразования для JPEG было выбрано Двумерное Дискретное Косинусное Преобразование (ДКП или DCT). обеспечивающее алгоритму достаточную простоту реализации, высокую степень сжатия и достаточно высокое качество сжатого изображения.

Алгоритм JPEG разрабатывался для сжатия цветных 24-битовых изображений, но может использоваться и для сжатия черно-белых изображений в градациях серого. Рассмотрим применение ДКП для сжатия черно-белых изображений.

Двумерное дискретное косинусное преобразование переводит изображение из области пространственных переменных (из представления набором отсчетов или пикселей) в спектральной область (представление набором частотных составляющих).

Дискретное косинусное преобразование от изображения $IMG(x, y)$ записывается следующим образом:

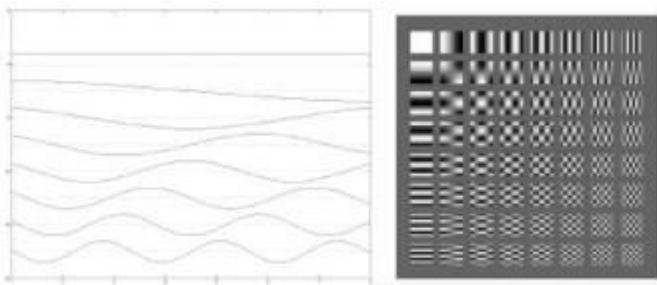
$$DCT(u,v) = \sqrt{\frac{2}{NM}} \sum_{i=0}^N \sum_{j=0}^M IMG(x_i, y_j) \cos((2i+1)\pi u/2N) \cdot \cos((2j+1)\pi v/2M),$$

где i и j – номера пикселей в блоке изображения, $0 < i, j < N, M$; u и v - индексы (номера отсчетов) в блоке спектральных коэффициентов, получаемых в результате ДКП: $N \times M$ -размер блока.

Процедуру ДКП можно записать также в матричной форме:

$$DCT = TC \cdot IMG \cdot TC^T,$$

где TC - матрица базисных (косинусных) коэффициентов преобразования. Для ДКП размером 8×8 вид базисных функций для одномерного (только по одной координате x или y) и двумерного преобразований приведен ниже.



Самая низкочастотная базисная функция (для одномерного ДКП) - соответствующая частотному индексу $u = 0$. изображена слева вверхе. Все ее значения равны единице.

Применительно к двумерному ДКП - $DCT = TC^T \cdot IMG \cdot TC$, (это функция в левом верхнем углу) и ее вид означает, что спектральная составляющая $DCT(0,0)$ получается как сумма всех пикселей блока изображения 8×8 (то есть, как средняя яркость блока).

Следующая по порядку базисная функция $u = 1$ изображена ниже, и представляет собой половину периода косинусоиды по одной координате и константу - по другой. Применительно к двумерному ДКП это означает, что спектральная составляющая с координатами $(1,0)$ представляет собой сумму всех пикселей блока слева от середины блока, минус сумму всех пикселей справа (разность яркостей левой и правой части блока 8×8). И так далее.

При этом, чем ниже и правее в матрице DCT его компонента, тем более высокочастотным деталям изображения она соответствует.

Процедура Двумерного ДКП в MATLAB реализуется с помощью функции

$$J = dct2(I),$$

Функция $J = dct2(I)$ возвращает двумерное ДКП массива изображения I . Матрица ДКП J имеет тот же размер, что и изображение I .

Пример 1:

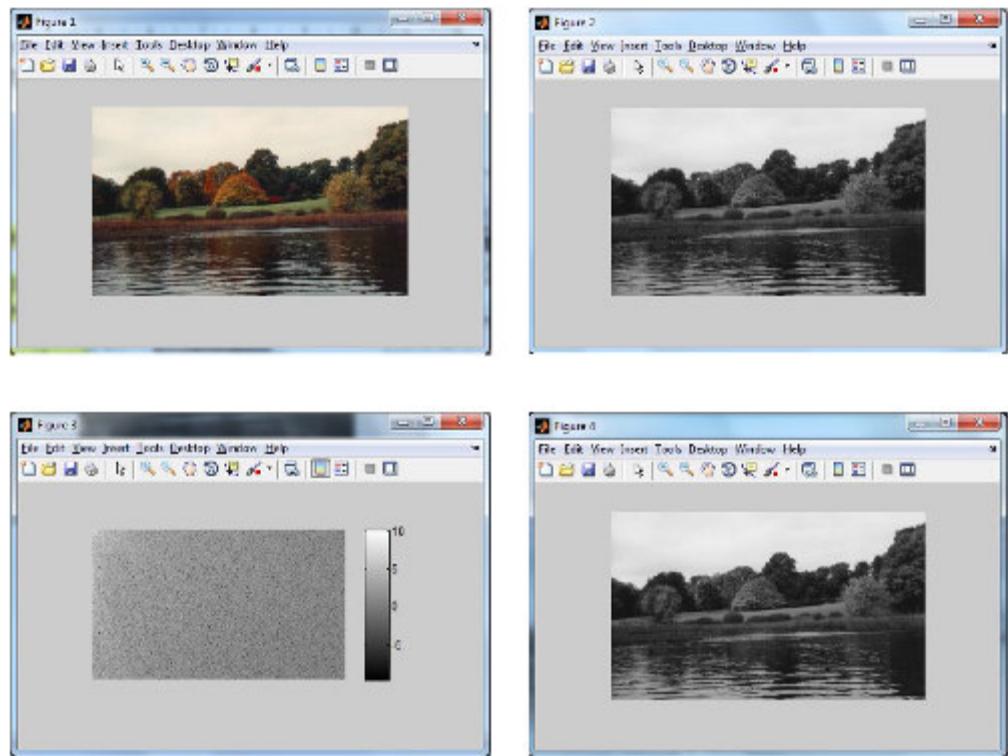
```
close all;
clear;
F = imread('autumn.tif');
imshow(F);
I = rgb2gray(F);
figure; imshow(I);
J = dct2(I);
figure; imshow(log(abs(J)), []), colorbar
```

В этом примере функция $I = rgb2gray(F)$ выполняет преобразование цветного изображения F в черно-белое I .

Отображение результата ДКП производится в логарифмическом масштабе, поскольку его динамический диапазон (от минимального до максимального коэффициента ДКП) гораздо шире динамического диапазона яркостей самого изображения (0...255).

Поскольку ДКП является обратимым линейным преобразованием, то по коэффициентам ДКП можно абсолютно точно восстановить исходное изображение. Обратное преобразование из области спектральных

коэффициентов ДКП в область пространственных переменных производится с использованием функции $K = \text{idct2}(J)$



Обратите внимание на интересную особенность полученного *ДКП* спектра: наибольшие (наиболее яркие) его значения сосредоточены в левом верхнем углу (это низкочастотные составляющие ДКП) вся же остальная часть массива (правая нижняя его часть - высокочастотные составляющие) заполнена относительно небольшими значениями (точками относительно невысокой яркости).

То есть, перевод изображения их пространственной области в спектральную позволил сосредоточить основную часть энергии блока изображения в относительно небольшом количестве низкочастотных спектральных коэффициентов, отвечающих за крупные детали изображения, хорошо различаемые глазом. Высокочастотные же коэффициенты, которые отвечают за мелкие детали изображения, имеют гораздо меньшую величину.

Другими словами - процедура ДКП *устраняет пространственную избыточность* изображения, поскольку чем более похожими друг на друга (коррелированными) будут значения пикселей в блоке, тем большая часть энергии блока будет сконцентрирована в компоненте $DCT(0,0)$ и тем меньшими будут значения высокочастотных компонент. В идеале, когда яркость всех точек блока изображения одинакова, в ДКП останется только один ненулевой элемент - $DCT(0,0)$, все остальные элементы блока будут равны нулю.

Вторым базисным положением кодирования в области преобразований является квантование коэффициентов ДКП.

Вычислив ДКП от исходного изображения просто «обнулим» все коэффициенты ДКП, величина которых по модулю не превышает, например,

10 (или 30). Этую процедуру можно выполнить с использованием функции
 $J(\text{abs}(J) < 10) = 0;$

А затем по оставшимся ненулевым коэффициентам попытаемся восстановить изображение. И посмотрим - что из этого получится.

Пример 2:

```
close all;
clear;
F = imread('autumn.tif');
I = rgb2gray(F);
figure; imshow(I);
J = dct2(I);
figure; imshow(log(abs(J)), []);
J(abs(J)<10) = 0;
figure; imshow(log(abs(J)), []);
K = idct2(J);
figure, imshow(K, [0 255]);
```



Обратите внимание на то, что в результате «обнуления» небольших по величине коэффициентов ДКП почти вся площадь массива ДКП "потемнела". То есть обнулились почти все коэффициенты (<10), кроме расположенных в левой верхней части матрицы (низкочастотных коэффициентов, отвечающих за крупные детали изображения). *Естественно, что результат кодирования квантованных коэффициентов ДКП будет гораздо компактнее, чем для неквантованных.* Результат восстановления изображения по квантованным коэффициентам ДКП приведен рядом, и почти не отличается от исходного изображения.

Если же отбросить все ДКП коэффициенты, по величине <30 (то есть

проквантовать их грубее), то число оставшихся коэффициентов ДКП будет еще меньше, но и качество восстановления изображения будет гораздо хуже.

Блочная обработка. Реализация алгоритма JPEG

Описанные выше основные положения сжатия изображений на основе кодирования квантованных коэффициентов его ДКП являются лишь общей иллюстрацией идеи. В работающем алгоритме JPEG эта идея существенно модифицирована, что позволило упростить ее реализацию, получить больший коэффициент сжатия и обеспечить лучшее качество сжатого изображения, нежели при прямой реализации основных принципов.

Недостатки описанной ранее технологии:

1. Первый недостаток - большой объем вычислений и большой объем необходимой памяти при выполнении прямого (в процессе кодирования) и обратного (в процессе декодирования) двумерного ДКП большого размера (от всего изображения целиком). Гораздо проще в этом смысле было бы разбить изображение на блоки меньшего размера и повторить одинаковую операцию несколько раз, применительно к каждому блоку.
2. Второй недостаток также связан с особенностями выполнения ДКП над массивом большого размера, а затем квантованием коэффициентов ДКП.

Дело в том, что при вычислении каждого коэффициента ДКП суммируются (усредняются с весами базисных функций ДКХ все пиксели изображения (смотри алгоритм ДКП), и вклад каждого отдельного пикселя в этой сумме при большом размере изображения крайне мал. Если какой-либо пиксель (или несколько пикселей) изображения существенно отличаются по яркости от соседних, то их вклад в величины коэффициентов ДКП будет очень небольшим. При квантовании коэффициентов ДКП (при их округлении) это отличие может быть потеряно, и тогда, при обратном ЖП эти пиксели просто теряются.

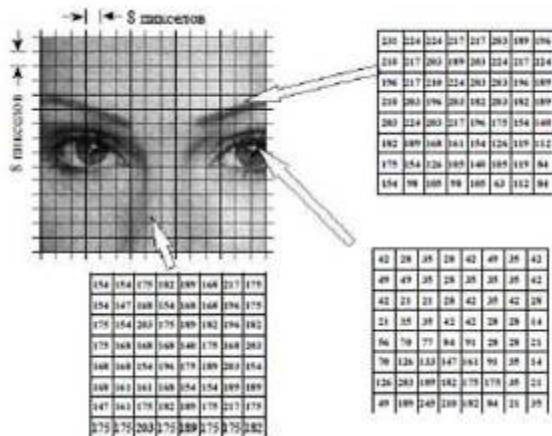
Иными словами, при большом размере ДКП и последующем квантовании коэффициентов ДКП мелкие детали изображения могут быть утеряны.

Все это было учтено при практической реализации алгоритма JPEG.

Работа алгоритма сжатия JPEG начинается с разбиения изображения на квадратные блоки размером 8x8 (64 пикселя).

Почему взят размер именно 8x8, а не 2x2 или 32x32? Выбор такого размера блока обусловлен тем, что при малом размере эффект сжатия будет незначительным, а при большом размере свойства изображения в пределах блока будут сильно изменяться и эффективность кодирования снова снизится. Кроме этого, при увеличении размера блока начнет проявляться эффект утери мелких деталей при квантовании, описанный выше.

Ниже изображено несколько таких блоков (в виде матриц цифровых отсчетов), взятых из различных участков изображения. В дальнейшем эти блоки будут обрабатываться и кодироваться независимо друг от друга.



Второй этап сжатия - Применение к каждому из блоков дискретного косинусного преобразования - ДКП.

Дискретное косинусное преобразование от блока изображения $IMG(x,y)$ может быть записано следующим образом:

$$DCT(u,v) = \sqrt{\frac{2}{N}} \sum_{i=0}^N \sum_{j=0}^N IMG(x_i, y_j) \cos((2i+1)\pi u/2N) \cos((2j+1)\pi v/2N), \quad (2.36)$$

где i, j, u и v индексы (номера отсчетов) в блоках изображения и ДКП спектральных коэффициентов, получаемых в результате ДКП; $N=8$ - размер блока.

Или в матричной форме:

$$DCT = TC \cdot IMG \cdot TC^T.$$

В результате применения к блоку изображения размером 8x8 пикселей дискретного косинусного преобразования получим двумерный ДКП спектр, также имеющий размер 8x8 отсчетов. Иными словами. 64 числа, представляющие отсчеты изображения, превратятся в 64 числа, представляющие отсчеты его ДКП-спектра.

Процедура повторяется для всех блоков изображения.

Для того, чтобы восстановить изображение по набору его ДКП - коэффициентов, нужно, как и ранее, выполнить обратное ДКП от каждого блока коэффициентов, а потом сложить изображение из полученных блоков изображения.

Пример выполнения поблочной процедуры прямого ДКП и обратного ДКП в MATLAB приведен ниже.

Пример 3:

```
I = imread('cameraman.tif');
I = im2double(I);
T = dctmtx(8);
dct = @(block_struct) T * block_struct.data * T';
B = blockproc(I,[8 8],dct);
invdct = @(block_struct) T' * block_struct.data * T;
I2 = blockproc(B,[8 8],invdct);
imshow(I), figure, imshow(I2)
```

Здесь функция:

`I=im2double (I)` – увеличивают точность представления исходных данных для уменьшения ошибок вычисления ДКП;

`T=dctmtx(8)` – задает матрицу коэффициентов ДКП нужного размера;

`B=blockproc(I,[8 8], dct)` – унифицирует процесс поблочной обработки. Она принимает изображение I вместе с размерами блока, который нужно обрабатывать (в данном случае - 8x8)? а также функцию, которая будет обрабатывать эти блоки (в данном случае - dct). Затем функция `blockproc` делит I на блоки 8x8, дополняет эти блоки нулями там, где их размеры выходят за пределы изображения, и выполняет dct.

Процедура `dct=@(block_struct)T*block_struct.data*T`; выполняется поблоно в матричном виде.

Исходное изображение и результат прямого/обратного ДКП выполненного поблоно, приведен ниже.



ЕСЛИ к результату прямого поблочного ДКП применить процедуру квантования с тем же шагом, что и ранее - `B*10*round(B*0.1);`, то получим следующий результат:



Обратите внимание, что эффект потери мелких деталей, в отличие от случая, когда ДКП выполнялось по всему изображению, практически пропал.

При практической реализации алгоритма JPEG процедура квантования выполняется не одинаково для всех коэффициентов ДКП, а с ИСПОЛЬЗОВАНИЕМ специальной матрицы коэффициентов, квантующей коэффициенты ДКП по-разному - более мягко для НЧ коэффициентов (расположенных в левом верхнем углу матрицы ДКП), и более грубо - для ВЧ коэффициентов (расположенных в правом нижнем углу матрицы ДКП).

Пример такой матрицы коэффициентов приведен ниже:

```
mask = [1 1 1 1 0 0 0 0
        1 1 1 0 0 0 0 0
        1 1 0 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];
B2 = blockproc(B,[8 8],@(block_struct) mask .* block_struct.data);
```

JPEG 2000

Как и первоначальная реализация JPEG, рассмотренная в предыдущем параграфе, алгоритм сжатия стандарта JPEG 2000 основан на идее, что коэффициенты преобразования, которое декоррелирует пиксели

изображения, можно кодировать более эффективно, чем сами исходные пиксели. Если базисные функции преобразования — вейвлеты в случае JPEG 2000 — пакуют самую важную часть визуальной информации в относительно малое число коэффициентов, то оставшиеся коэффициенты можно грубо проквантовать или вовсе обнулить, что приведет лишь к малым искажениям сжатого изображения.

Рис. 14 показывает упрощенную систему кодирования JPEG 2000 (в ней отсутствует несколько опционных операций). Первый шаг процесса кодирования, как и в исходном стандарте JPEG, состоит в сдвиге в нуль среднего уровня яркости пикселов, которое достигается вычитанием из всех пикселов числа 2^{m-1} , где 2^t — общее число возможных уровней яркости. Затем вычисляется одномерное дискретное вейвлетное преобразование строк и столбцов изображения. При сжатии без потерь используется биортогональное вейвлетное преобразование 5—3². В приложениях, которые допускают потерю информации, используется вейвлетное преобразование. В обоих случаях первоначальным результатом декомпозиции изображения являются четыре поддиапазона — одно низкочастотное приближение и три частные характеристики (детали) по горизонтали, вертикали и диагонали.

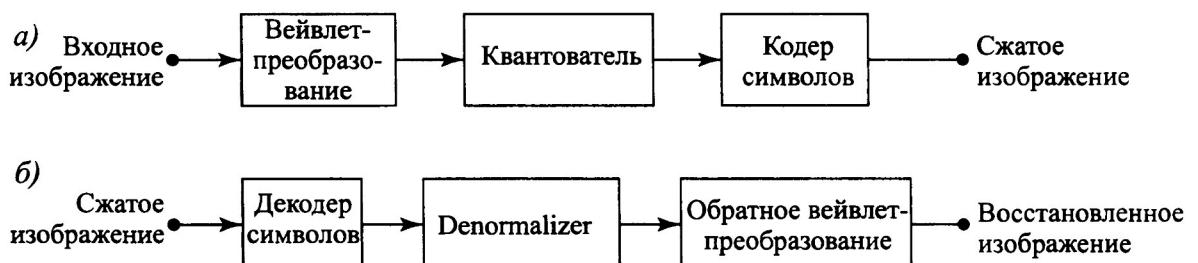


Рисунок 14 – Блок-схема JPEG 2000 а) кодер, б) декодер

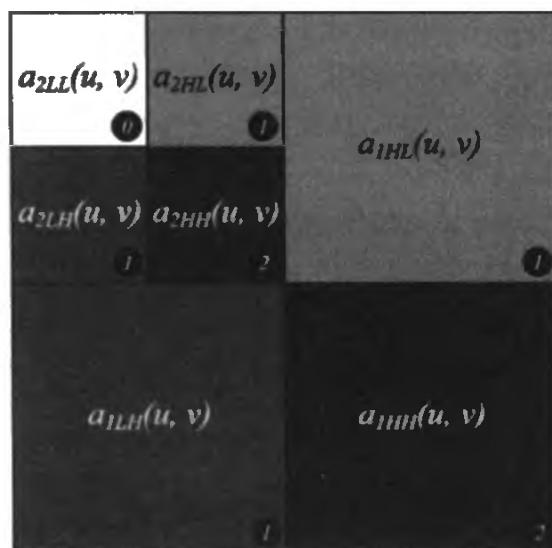


Рисунок 15 – Схема обозначений для коэффициентов двухмасштабного вейвлетного преобразования JPEG 2000 и число дополнительных битов анализа (в черных кружочках)

Пространственное разрешение соседних масштабов различается в два раза, причем самый крупный масштаб образует наиболее точное приближение исходного изображения. Как можно предположить из рис. 15, на котором приведена стандартная схема обозначений для случая масштаба $N_L = 2$, преобразование масштаба N_L общего вида состоит из $3N_L + 1$ поддиапазона, коэффициенты которых обозначаются индексированной буквой a_{ν} , где $\nu = N_{LL}, NLHL, NLH, N_{LL}, \dots, IHL, ILH, IHN$. Стандарт не ограничивает число вычисляемых масштабов. После того, как вейвлетное преобразование масштаба N_L вычислено, общее число коэффициентов преобразования равно числу отсчетов исходного изображения, но главная визуальная информация сконцентрирована только в небольшом числе коэффициентов. Для уменьшения числа битов, требуемых для их представления, коэффициент $a_{\nu}(u, v)$ поддиапазона b квантуется на величину $qb(u, v)$ по формуле

$$qb(u, v) = \text{sign}(a_b(u, v)) \cdot \text{floor} \left[\frac{|a_b(u, v)|}{\Delta_b} \right],$$

где $\text{sign}(\bullet)$ — это знак числа, $\text{floor}[\bullet]$ — целая часть числа (подобно стандартным функциям sign^3 и floor в MATLAB). Шаг квантования Δ_b задается выражением

Здесь R_b — номинальный динамический диапазон поддиапазона b , а E_b и M_b — число битов, отводимых под величины порядка и мантиссы его коэффициентов. Номинальный динамический диапазон b равен сумме числа битов, используемых для представления исходного изображения, и числа дополнительных битов анализа поддиапазона b . Числа дополнительных битов анализа заключены в черные кружочки на рис. 15. Например, для поддиапазона $b = IHN$ отводится 2 дополнительных бита анализа.

Для сжатия без потерь используются следующие значения параметров: $M_b = 0$, $R_b = E_b$, $\Delta_b = 1$. Для необратимого сжатия в стандарте не предусмотрено никакого конкретного шага квантования. Вместо этого декодер должен знать число битов порядка и мантиссы, которое или передается вместе с каждым поддиапазоном (это называется *явным квантованием*), или же только с поддиапазоном N_{LL} (*неявное квантование*). В последнем случае остальные поддиапазоны квантуются с использованием значений параметров, экстраполированных по параметрам поддиапазона N_{LL} . Полагая, что E_0 и M_0 — это известное число битов, отводимых для поддиапазона N_{LL} , параметры для поддиапазона b вычисляются по формулам

$$M_0 = M_b, \quad E_b = E_0 + nsd_b - nsd_0,$$

где nsd_b обозначает число уровней декомпозиции от исходного изображения до поддиапазона b . Финальным шагом процесса сжатия является кодирование квантованных коэффициентов с помощью метода арифметического кодирования на основе битовых плоскостей. Мы здесь не обсуждаем метод арифметического кодирования, который, подобно схеме

Хаффмана, представляет собой эффективный вариант кодирования кодами переменной длины, разработанный для сокращения кодовой избыточности.

Приводимая далее функция `im2jpeg2k` моделирует процесс кодирования JPEG 2000 на рис. 14a) за исключением этапа арифметического символьного кодирования. Как видно из текста этой функции, вместо этого для простоты используется кодирование Хаффмана, расширенное кодированием длин нулевых серий.

```

function y = im2jpeg2k(x, n, q)
global RUNS
error(nargchk(3, 3, nargin));
if ndims(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
    error('The input must be a UINT8 image.');
end
if length(q) ~= 2 & length(q) ~= 3 * n + 1
    error('The quantization step size vector is bad.');
end
x = double(x) - 128;
[c, s] = wavefast(x, n, 'jpeg9.7');
q = stepsize(n, q);
sgn = sign(c);      sgn(find(sgn == 0)) = 1;      c = abs(c);
for k = 1:n
    qi = 3 * k - 2;
    c = wavepaste('h', c, s, k, wavecopy('h', c, s, k) / q(qi));
    c = wavepaste('v', c, s, k, wavecopy('v', c, s, k) / q(qi + 1));
    c = wavepaste('d', c, s, k, wavecopy('d', c, s, k) / q(qi + 2));
end
c = wavepaste('a', c, s, k, wavecopy('a', c, s, k) / q(qi + 3));
c = floor(c);      c = c .* sgn;
zrc = min(c(:)) - 1;      eoc = zrc - 1;      RUNS = [65535];
z = c == 0;           z = z - [0 z(1:end - 1)];
plus = find(z == 1);      minus = find(z == -1);
if length(plus) ~= length(minus)
    c(plus(end):end) = [];      c = [c eoc];
end
for i = length(minus):-1:1
    run = minus(i) - plus(i);
    if run > 10
        ovrflo = floor(run / 65535);      run = run - ovrflo * 65535;
        c = [c(1:plus(i) - 1) repmat([zrc 1], 1, ovrflo) zrc ...
               runcode(run) c(minus(i):end)];
    end
end
y.runs      = uint16(RUNS);
y.s       = uint16(s(:));
y.zrc     = uint16(-zrc);
y.q       = uint16(100 * q');
y.n       = uint16(n);
y.huffman = mat2huff(c);
function y = runcode(x)
global RUNS
y = find(RUNS == x);
if length(y) ~= 1
    RUNS = [RUNS; x];
    y = length(RUNS);
end
function q = stepsize(n, p)
if length(p) == 2
    q = [];
    qn = 2 ^ (8 - p(2) + n) * (1 + p(1) / 2 ^ 11);
    for k = 1:n
        qk = 2 ^ -k * qn;

```

```

    q = [q (2 * qk) (2 * qk) (4 * qk)];
end
q = [q qk];
else
    q = p;
end
q = round(q * 100) / 100;
if any(100 * q > 65535)
    error('The quantizing steps are not UINT16 representable.');
end
if any(q == 0)
    error('A quantizing step of 0 is not allowed.');
end

```

Декодер JPEG 2000 просто обращает описанные выше операции. После декодирования арифметических кодов коэффициентов совершается восстановление выбранного пользователем числа поддиапазонов изображения. Хотя кодер может иметь M_b арифметически закодированных битовых плоскостей для конкретного поддиапазона, пользователь — в силу вложенной природы кодового потока — может выбрать для декодирования только \mathcal{N}_b битовых плоскостей. Это равняется квантованию коэффициентов с использованием размера шага $2^{M_b-N_b} \cdot \Delta_b$. Все недекодируемые биты приравниваются нулю, и полученные коэффициенты, обозначаемые $\bar{q}_b(u, v)$, восстанавливаются по формулам

$$R_{q_b}(u, v) = \begin{cases} (\bar{q}_b(u, v) + 2^{M_b-N_b(u, v)}) \cdot \Delta_b & \text{при } \bar{q}_b(u, v) > 0, \\ (\bar{q}_b(u, v) - 2^{M_b-N_b(u, v)}) \cdot \Delta_b & \text{при } \bar{q}_b(u, v) < 0, \\ 0 & \text{при } \bar{q}_b(u, v) = 0, \end{cases}$$

где $R_{q_b}(u, v)$ обозначает восстановленные значения коэффициентов, а $N_b(u, v)$ — число декодируемых битовых плоскостей для $q_b(u, v)$. Затем полученные коэффициенты подвергаются обратному преобразованию, а к результату прибавляется величина 2^{n-1} , где n — число битов яркости пикселов. Функция jpeg2k2im моделирует весь этот процесс, обращая результат сжатия под действием функции im2jpeg2k.

```

function x = jpeg2k2im(y)
error(nargchk(1, 1, nargin));
n = double(y.n);
q = double(y.q) / 100;
runs = double(y.runs);
rlen = length(runs);
zrc = -double(y.zrc);
eoc = zrc - 1;
s = double(y.s);
s = reshape(s, n + 2, 2);
cl = prod(s(1, :));
for i = 2:n + 1
    cl = cl + 3 * prod(s(i, :));
end
r = huff2mat(y.huffman);
c = []; zi = find(r == zrc); i = 1;
for j = 1:length(zi)
    c = [c r(i:zi(j) - 1) zeros(1, runs(r(zi(j) + 1)))] ;
    i = zi(j) + 2;
end
zi = find(r == eoc);
if length(zi) == 1
    c = [c r(i:zi - 1)];

```

```

c = [c zeros(1, cl - length(c))];
else
    c = [c r(i:end)];
end
c = c + (c > 0) - (c < 0);
for k = 1:n
    qi = 3 * k - 2;
    c = wavepaste('h', c, s, k, wavecopy('h', c, s, k) * q(qi));
    c = wavepaste('v', c, s, k, wavecopy('v', c, s, k) * q(qi + 1));
    c = wavepaste('d', c, s, k, wavecopy('d', c, s, k) * q(qi + 2));
end
c = wavepaste('a', c, s, k, wavecopy('a', c, s, k) * q(qi + 3));
x = waveback(c, s, 'jpeg9.7', n);
x = uint8(x + 128);

```

Принципиальное отличие системы JPEG 2000 на основе вейвлетов (см. рис. 14) от системы JPEG на базе преобразования DCT (см. рис. 11) заключается в отсутствии этапа разделения изображения на маленькие подизображения. Поскольку вейвлетные преобразования одновременно являются вычислительно эффективными и локальными по своей природе (т. е. их базисные функции имеют конечную длительность), нет необходимости делить сжимаемое изображение на подблоки. Как видно в следующем примере, отсутствие стадии деления устраняет артефакты блочности, которые являются неотъемлемыми спутниками приближения изображений на базе преобразования DCT при высокой степени сжатия.

Пример 9. Сжатие JPEG 2000.

На рис. 16 построены два приближения черно-белого изображения на рис. 4a). Рис. 16a) является реконструкцией кодирования, которое сжало исходное изображение в соотношении 42:1, а рис. 16б) получен после сжатия 88:1. Оба результата получены при преобразовании пятого масштаба и с параметрами ($E_o = 8.5$ и 7 соответственно). Поскольку наша функция im2jpeg2k лишь моделирует настоящее арифметическое кодирование по битовым плоскостям JPEG 2000, коэффициент сжатия при использовании настоящего стандарта JPEG 2000 будет отличаться от приведенных выше значений. На самом деле, степень сжатия возрастет почти в 2 раза.

Раз коэффициент сжатия 42:1, отвечающий левой колонке изображений на рис. 16, практически совпадает с коэффициентом сжатия, достигнутым для изображений в правой колонке на рис. 13 (пример 8), то рис. 16a), в) и д) можно сопоставить — качественно и количественно — с результатами сжатия JPEG на рис. 14б), г) и е). Визуальное сравнение обнаруживает заметное снижение ошибок на изображениях JPEG 2000. На самом деле, ошибка RMS для JPEG 2000 [рис. 16a)] равна 3.7 уровня яркости, в отличие от 4.4 уровней для соответствующего изображения JPEG [рис. 13б)]. Помимо уменьшения ошибки реконструкции, кодирование на базе JPEG 2000 значительно улучшает качество изображения (в субъективном смысле). Это становится особенно очевидным при сравнении увеличенных фрагментов на рис. 16д) и 13е). Обратите внимание на то, что на рис. 16д) полностью отсутствуют артефакты блочности, которые отчетливо видны на рис. 13е).

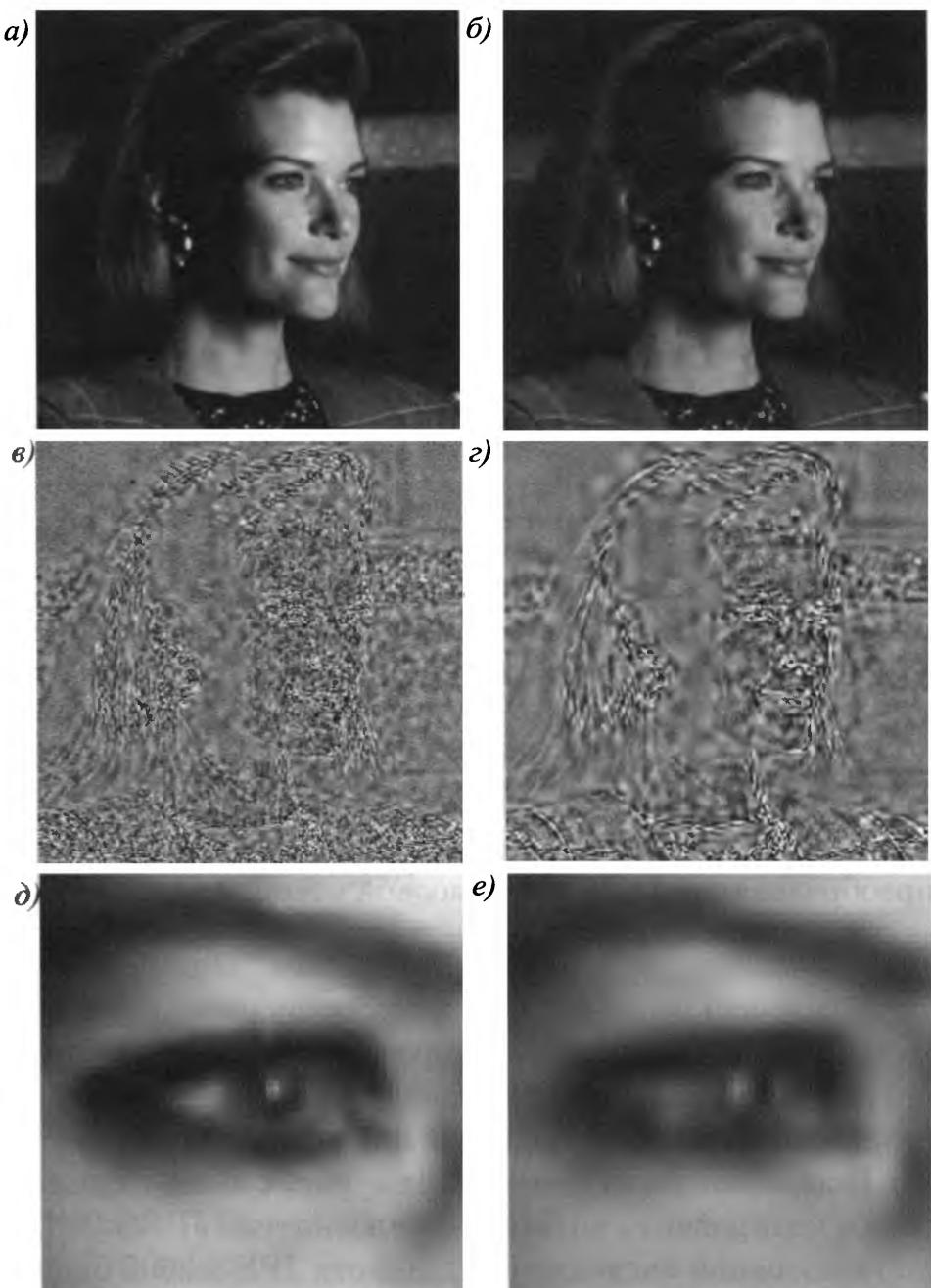


Рисунок 16 – Левая колонка: приближение JPEG 2000 для рис. 4а) с использованием 5 масштабов и при $\mu_0=8$ и $E_0=8.5$. Правая колонка: аналогичные результаты с $E_0=7$

Когда степень сжатия повышается до значения 88: 1 [см. рис.16б)], наблюдается некоторая потеря текстуры на одежде девушки, а также происходит размытие ее глаз. Оба эти эффекта можно обнаружить на рис. 16б) и е). Ошибка RMS для этих реконструкций равна 5.9 уровней яркости. Результаты на рис. 16 были получены с использованием следующей последовательности команд:

```
>> f = imread('Tracy.tif');
>> cl = im2jpeg2k(f, 5, [8 8.5]);
>> fl = jpeg2k2im(cl);
>> rmsl = compare(f, fl)
```

```

rms1 =
    3.6931
>> crl = imratio(f, cl)
crl =
    42.1589
>> c2 = im2jpeg2k(f, 5, [8 7]);
>> f2 = jpeg2k2im(c2);
>> rms2 = compare(f, f2)
rms2 =
    5.9172
>> cr2 = imratio(f, c2) cr2 =
    87.7323

```

Обратите внимание на использование неявного квантования при задании третьего аргумента функции `im2jpeg2k` в виде вектора из двух компонент. Если же число компонент этого вектора больше 2, функция подразумевает явное квантование, и размеры квантования всех $3N + 1$ шагов должны быть заданы в этом векторном аргументе (здесь N обозначает число требуемых масштабов). Все эти числа необходимо упорядочить по уровням декомпозиции (первый, второй, третий, ...) и по типу поддиапазона (горизонтальный, вертикальный, диагональный и приближенный). Например.

```

» c3 = im2jpeg2k(f, 1, [1111]);

```

вычисляет одномасштабное преобразование с явным квантованием, а все шаги квантования всех поддиапазонов равны $\Delta_l = 1$. Значит, все коэффициенты преобразования округляются до ближайшего целого. Это случай минимальной ошибки при реализации `im2jpeg2k`. В этом случае ошибка RMS и коэффициент сжатия равны

```

f3 = jpeg2k2im(c3);
>> rms3 = compare(f, f3) rms3
=
    1.1234
>> cr3 = imratio(f, c3)
cr3 =
    1.6350

```

RLE сжатие

RLE обозначает Run Length Encoding. Это алгоритм сжатия без потерь, который эффективно сжимает только определенные типы данных.

RLE - самый простой алгоритм сжатия. Он заменяет последовательность одинаковых данных в файле образцом и количеством его повторов.

Попробуем сжать следующую последовательность данных (17 байт):

ABBBBBBBBBBCDEEEEF

При использовании RLE сжатия, сжатый файл займет 10 байт и будет выглядеть так:

*A *8B C D *4E F*

Как можно увидеть, повторяющиеся данные были заменены управляющим символом (*), числом повторов и самим символом. Управляющий символ не фиксирован и может отличаться в различных реализациях.

Если символ, соответствующий управляющему, встречается в сжимаемом файле, он кодируется, как отдельный символ.

Как можно видеть, RLE сжатие эффективно для сжатия последовательностей, где одинаковые символы идут 4 и более раз подряд. Дело в том, что сжатие трех символов не приводит к уменьшению файла, а сжатие двух символов увеличивает его.

Важно знать, что существует множество реализаций этого алгоритма. Вышеприведенный пример демонстрирует основные принципы RLE сжатия. Некоторые реализации этого метода адаптируются под определенные виды сжимаемых данных.

Достоинства и недостатки

Этот алгоритм очень легок в реализации и не требует больших процессорных затрат. Но RLE сжатие эффективно только на файлах, содержащих большие объемы повторяющихся данных. Это могут быть текстовые файлы, содержащие большое количество пробелов для отбивок или битмапы, содержащие большие черные или белые области. Цветные битмапы, сгенеренные на компьютере, тоже могут хорошо сжиматься.

Где используется RLE сжатие

RLE сжатие используется в следующих форматах файлов:

- TIFF файлы

Общая постановка задачи

В результате выполнения заданий лабораторной работы студенты **должны уметь** создавать программно-алгоритмическую поддержку для компьютерной реализации требуемых алгоритмов сжатия графической информации в MatLab.

Лабораторные занятия проводятся в компьютерных классах.

Список индивидуальных данных

№ варианта	Значение mask	№ варианта	Значение mask
1.	$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	2.	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

Написать файл-функцию и GUI интерфейс для решения следующих задач.

Задание 1:

1. Выполните чтение нескольких изображений из файла, самостоятельно выбранного студентом. Выберите первую цветовую компоненту изображения в формате RGB и определите размер изображения. Используйте функцию `imshow` для вывода изображений на экран.

2. С использованием функции `rgb2gray` преобразуйте цветные изображения в черно-белые. Отобразите результат.
3. С использованием функции `dct2` выполните Дискретное Косинусное Преобразование изображений. Отобразите результат ДКП в виде изображения с использованием функции `imshow`. При отображении массива ДКП-коэффициентов используйте логарифмический масштаб.
4. С использованием функции `idct2` восстановите изображение по его ДКП-спектру.
5. С использованием процедуры целочисленного деления умножения $J=1/N*\text{round}(J*N)$; выполните квантование результатов ДКП для разных значений шага квантования N . Объясните, как работает эта процедура и что получается в ее результате.
6. Отобразите результаты в виде изображения (как и ранее для ДКП - в логарифмическом масштабе). Объясните вид полученных данных.
7. С использованием функции `idct2` восстановите изображение по квантованному ДКП-спектру при разной величине шага квантования. Объясните полученные результаты.
8. Объясните, какая цель достигается квантованием коэффициентов ДКП.
9. Можно ли добиться этой же цели и такого же результата квантуя исходное изображение, а не коэффициенты его ДКП?
Проверьте это на практике, проквантовав с использованием процедуры $I=(\text{round } (J/n))*n$ исходные изображения и отобразив полученный результат. Объясните результат.
10. Какие недостатки Вы видите в сжатии изображений с использованием его ДКП и квантовании коэффициентов ДКП?
11. Выведите на экран результирующее изображение.
12. Сохраните полученные изображения.

Задание 2:

1. Выполните чтение нескольких изображений из файла, самостоятельно выбранного студентом. Выберите первую цветовую компоненту изображения в формате RGB и определите размер изображения. Используйте функцию `imshow` для вывода изображений на экран.
2. С использованием функции `rgb2gray` преобразуйте цветные изображения в черно-белые. Отобразите результат.
3. С использованием функции `B=blockproc(I,[N N], dct)` и процедуры `dct=@(block_struct)T*block_struct.data*T'`; выполните поблочное Дискретное Косинусное Преобразование изображений.
4. Отобразите результат поблочного ДКП всего исходного изображения с использованием функции `imshow`. При отображении массива ДКП-коэффициентов используйте логарифмический масштаб. Объясните вид полученного изображения.
5. С использованием процедур `invdct=@(block_struct)T'*block_struct.data*T` и `I2=blockproc(B,[8 8], invdct)`; восстановите изображение по его ДКП-спектру.

6. С использованием процедуры целочисленного деления умножения $J = H * \text{round}(J/N)$; выполните квантование результатов ДКП для разных значений шага квантования N. Объясните, как работает эта процедура и что получается в ее результате.
Выполните квантование коэффициентов ДКП, используя процедуру согласно варианту $B2=\text{blockproc}(B,[8\ 8], @(\text{block_struct})\text{masc}.*\text{block_struct}.data);$
7. С использованием процедур $\text{invdct}=@(\text{block_struct})T'*\text{block_struct}.data*T;$ и $I2=\text{blockproc}(B,[8\ 8], \text{invdct});$ восстановите изображение по его квантованному ДКП-спектру.
8. Объясните полученный результат и какая цель достигается квантованием коэффициентов ДКП?
9. Выведите на экран результирующее изображение.
10. Сохраните полученные изображения.

Задание 3:

1. Выполнить чтение нескольких изображений из файла, самостоятельно выбранного студентом. Выбрать первую цветовую компоненту изображения в формате RGB и определить размер изображения. Использовать функцию `imshow` для вывода изображений на экран.
2. Выполнить кодирование с предсказанием без потери информации, показать ошибки предсказания для исходного изображения, построить гистограмму ошибки предсказания.
3. С использованием функции `quantize` выполнить сжатие изображения посредством квантования.
4. Для выбранного изображения выполнить квантование IGS, кодирование с предсказанием без потерь и кодирование Хаффмана. Сравнить полученные результаты.
5. Выполнить сжатие JPEG и рассчитать коэффициент сжатия.
6. Выполнить сжатие JPEG 2000 и рассчитать коэффициент сжатия.
7. Сохранить полученные изображения.

Задание 4:

1. Самостоятельно реализуйте алгоритм сжатия LZW и RLE.

Пример выполнения работы

Примеры реализации в виде программного кода всех алгоритмов, необходимых для выполнения отдельных шагов лабораторной работы, приведены в теоретической части.

Контрольные вопросы к защите

Контрольные вопросы на защите отчета по лабораторной работе заключаются в необходимости объяснить действие того или иного оператора в разработанной программной реализации метода - вопросы формулируются преподавателем после предварительного ознакомления с текстом созданной

программы. При защите отчета по лабораторной работе также необходимо ответить на контрольные вопросы, приведенные после лекционного материала по соответствующей теме.