МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 52

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Борисовская А.В.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

МЕТОДЫ БЛОКОВОЙ ОБРАБОТКИ ПРИ СЖАТИИ С ПОТЕРЯМИ НА ПРИМЕРЕ СТАНДАРТА JPEG

по курсу: Мультимедиа технологии

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5912		Нам Д.О.
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Цель работы

Изучение алгоритмов, используемых в базовом (baseline) режиме стандарта JPEG, анализ статистических свойств, используемых при сжатии коэффициентов дискретного косинусного преобразования (ДКП), а также получение практических навыков разработки методов блоковой обработки при сжатии изображений с потерями.

1. Дискретное косинусное преобразование (ДКП)

$$y_{k,l} = \sqrt{C_k}\sqrt{C_l} \cdot \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x_{i,j} \cos\left(\frac{(2i+1)\pi}{2N}k\right) \cos\left(\frac{(2j+1)\pi}{2N}l\right),$$

$$k = 0, ..., N-1; \quad l = 0, ..., N-1;$$

Рисунок 1. Формула прямого ДКП

$$x_{i,j} = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} \sqrt{C_k}\sqrt{C_l} \cdot y_{k,l} \cos\left(\frac{(2i+1)\pi}{2N}k\right) \cos\left(\frac{(2j+1)\pi}{2N}l\right),$$

$$i = 0, ..., N-1; \quad j = 0, ..., N-1.$$

Рисунок 2. Формула обратного ДКП

$$C_f = \begin{cases} \frac{1}{N}, \text{если } f = 0 \\ \frac{2}{N}, \text{иначе} \end{cases}.$$

Рисунок 3. Формула нормирующего коэффициента

1.1 ДКП для изображения baboon.bmp

Рисунок 4. Исходное изображение слева и изображение после ДКП справа

PSNR(Y) = 55.894
PSNR(Cr) = 55.922
PSNR(Cb) = 55.911

1.2 ДКП для изображения kodim04.bmp

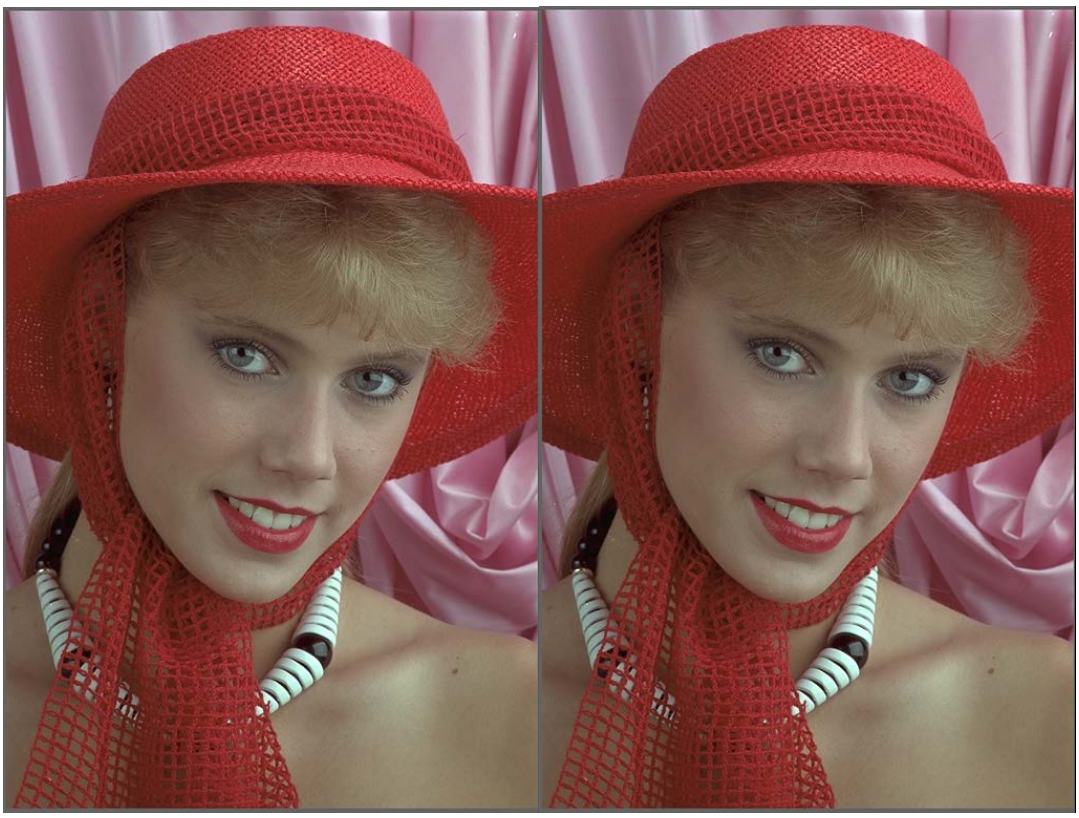

Рисунок 5. Исходное изображение слева и изображение после ДКП справа

PSNR(Y) = 55.928
PSNR(Cr) = 56.107
PSNR(Cb) = 56.067

1.3 ДКП для изображения lena.bmp

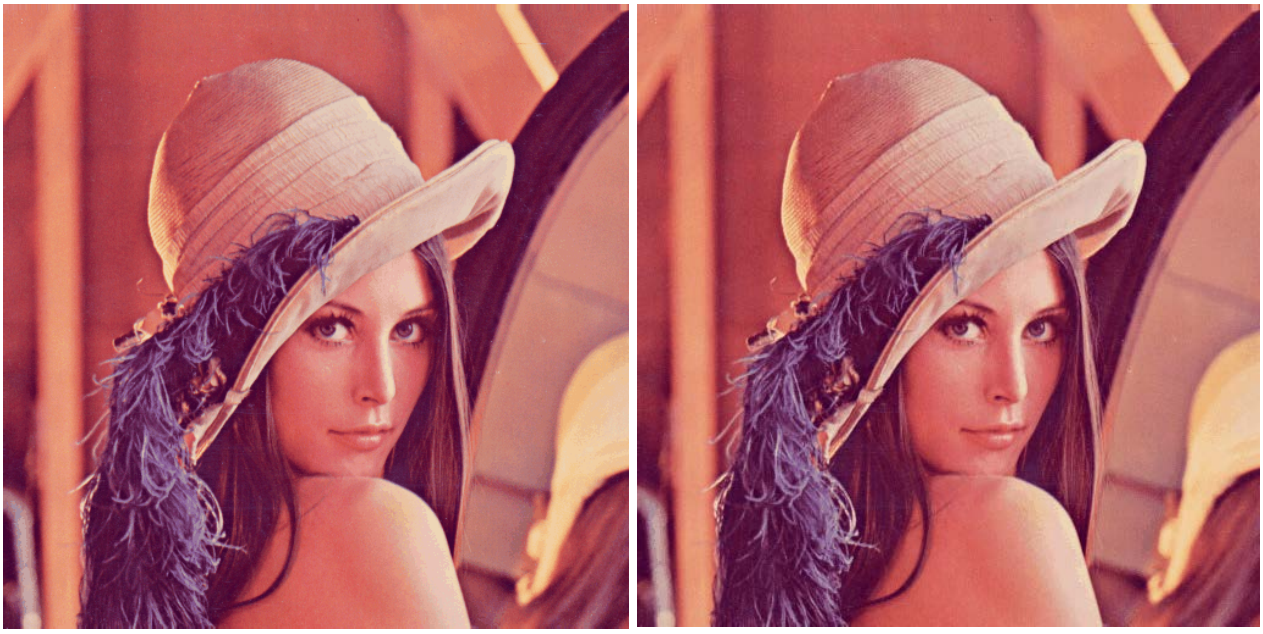

Рисунок 6. Исходное изображение слева и изображение после ДКП справа

PSNR(Y) = 56.020
PSNR(Cr) = 56.050
PSNR(Cb) = 55.834

2. Квантование спектральных коэффициентов

$$Y_{i,j}^{q} = round\left(\frac{Y_{i,j}}{q_{i,j}^{(c)}}\right) \quad , i = 0, ..., 7, \quad j = 0, ..., 7$$

Рисунок 7. Формула процедуры квантования спектральных коэффициентов

$$Y_{i,j}^{dq} = Y_{i,j}^{q} \cdot q_{i,j}^{(c)} \quad , i = 0, ..., 7, \quad j = 0, ..., 7$$

Рисунок 8. Формула процедуры деквантования спектральных коэффициентов

$$q_{i,j}^{Y}(R) = 1 + (i + j) \cdot R, \quad i = 0, ..., 7, \quad j = 0, ..., 7$$

Рисунок 9. Формула процедуры построения матриц квантования

2.1 Реализация процедуры квантования и деквантования

2.1.1 Изображение baboon.bmp

Рисунок 10. Изображение после квантования при R = 1

Рисунок 11. Изображение после квантования при R = 5

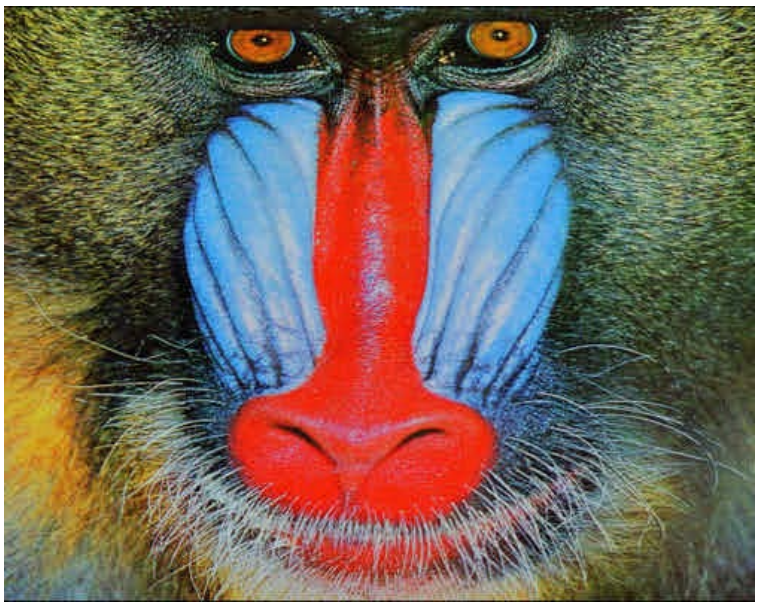

Рисунок 12. Изображение после квантования при R = 10

2.1.2 Изображение kodim04.bmp

Рисунок 13. Изображение после квантования при R = 1

Рисунок 14. Изображение после квантования при R = 5

Рисунок 15. Изображение после квантования при R = 10

2.1.3 Изображение lena.bmp

Рисунок 16. Изображение после квантования при R = 1

Рисунок 17. Изображение после квантования при R = 5

Рисунок 18. Изображение после квантования при R = 10

2.2 Оценка влияния искажений, вносимых квантованием, на компоненты этих изображений

2.2.1 Изображение baboon.bmp

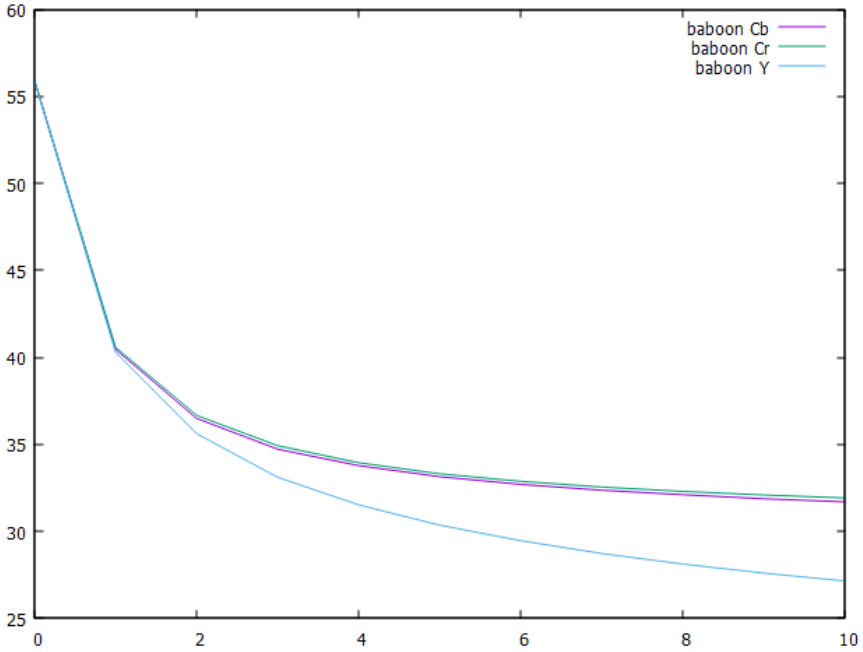

Рисунок 19. График зависимости PSNR от R для различных компонент

Рисунок 20. Сравнение исходного изображения с квантованным при R = 10

2.2.2 Изображение kodim04.bmp

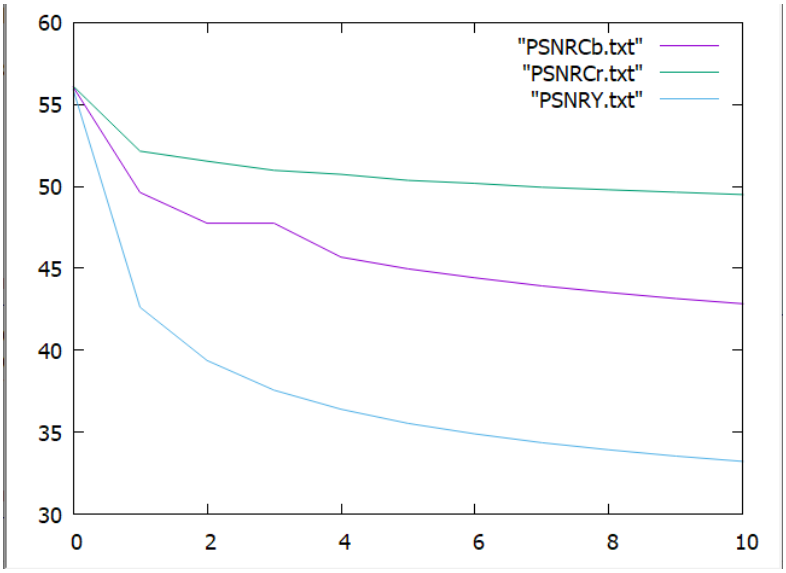

Рисунок 21. График зависимости PSNR от R для различных компонент

Рисунок 22. Сравнение исходного изображения с квантованным при R = 10

2.2.3 Изображение lena.bmp

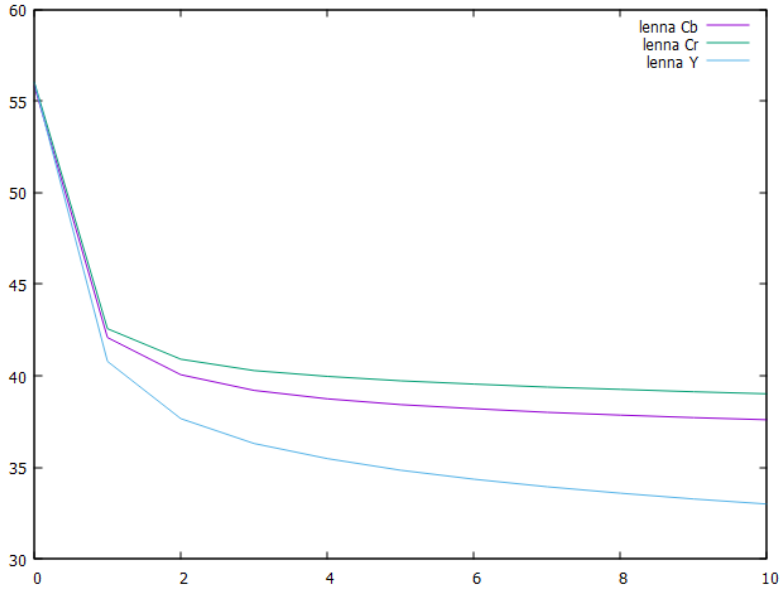

Рисунок 23. График зависимости PSNR от R для различных компонент

Рисунок 24. Сравнение исходного изображения с квантованным при R = 10

На графиках видно, что значение PSNR для яркостной компоненты уменьшается быстрее относительно компонент Cb и Cr. Это связано с тем, что Y содержит больше информации.

3. Сжатие без потерь

3.1 Процедура кодирования квантованных коэффициентов постоянного тока *DC*

$$\Delta_{DC} = DC_i^q - DC_{i-1}^q$$

Рисунок 25. Разностный метод кодирования DC^q

Значение ΔDC представляется в форме битовой категории BC и амплитуды MG, где битовая категория от значения x вычисляется как $BC(x) = \lceil \log_2(|x| + 1) \rceil$, а значение амплитуды – само кодируемое значение.

3.2 Оценка эффективности использования разностного кодирования для коэффициентов постоянного тока

Гистограммы частот для изображения baboon.bmp

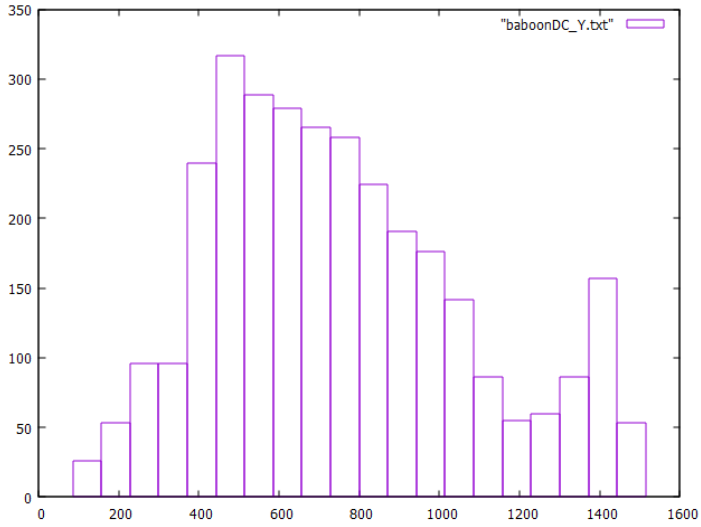

Рисунок 26. Гистограмма частот f(DC) компоненты Y

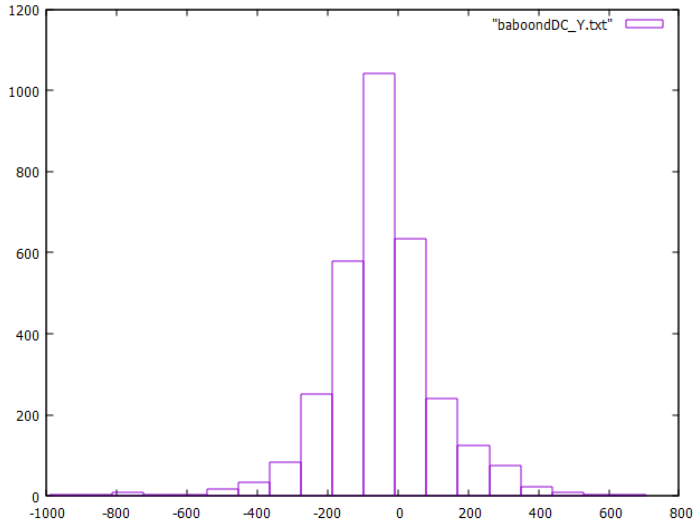

Рисунок 27. Гистограмма частот f(ΔDC) компоненты Y

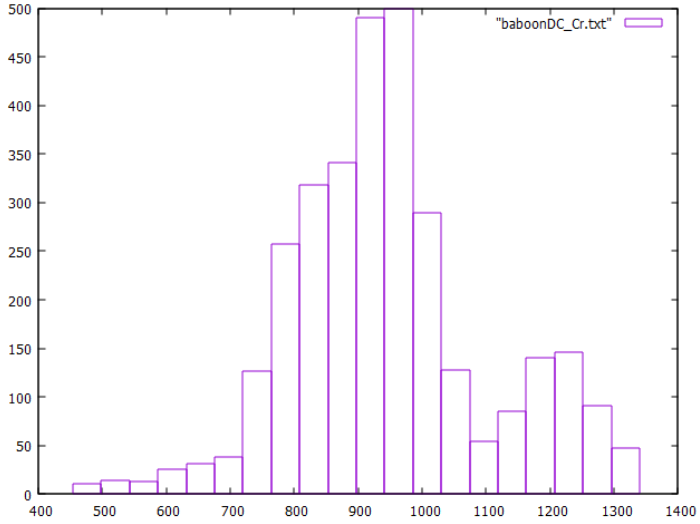

Рисунок 28. Гистограмма частот f(DC) компоненты Cr

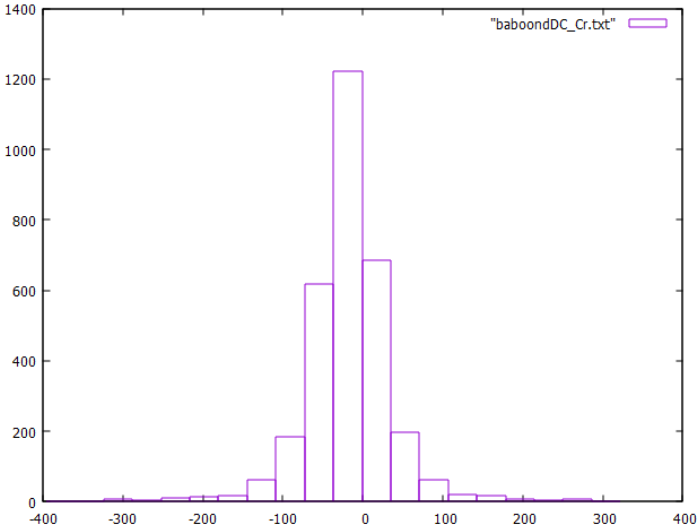

Рисунок 29. Гистограмма частот f(ΔDC) компоненты Cr

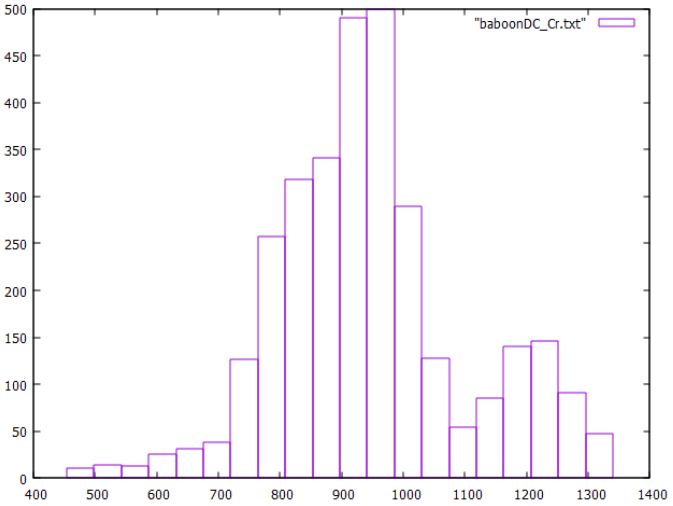

Рисунок 30. Гистограмма частот f(DC) компоненты Cb

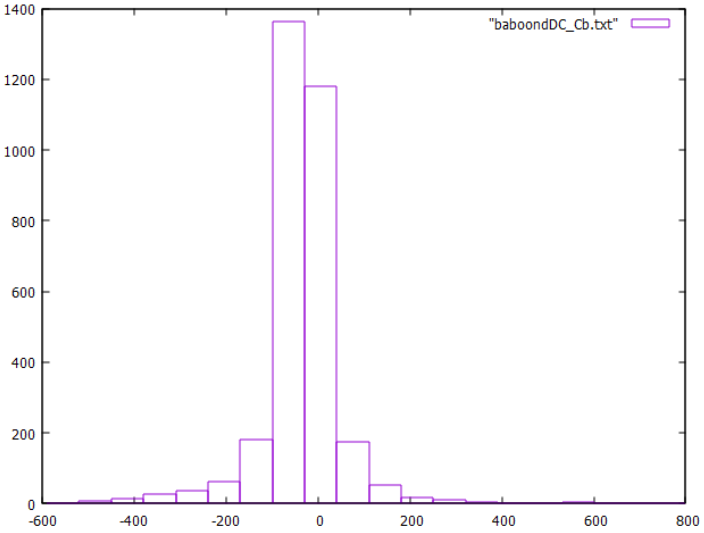

Рисунок 31. Гистограмма частот f(ΔDC) компоненты Cb

Значения энтропии DC^q:

H(Y) = 9.872
H(Cr) = 8.941
H(Cb) = 9.305

Значение энтропии ΔDC:

H(Y) = 8.936
H(Cr) = 7.459
H(Cb) = 7.797

Гистограммы частот для изображения kodim04.bmp

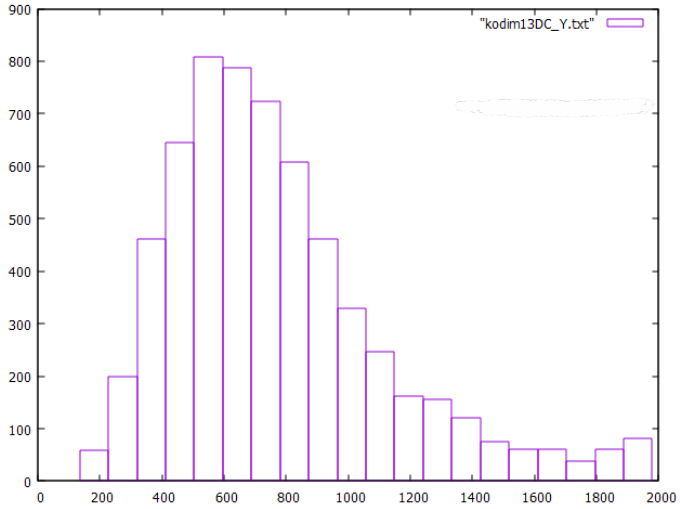

Рисунок 32. Гистограмма частот f(DC) компоненты Y

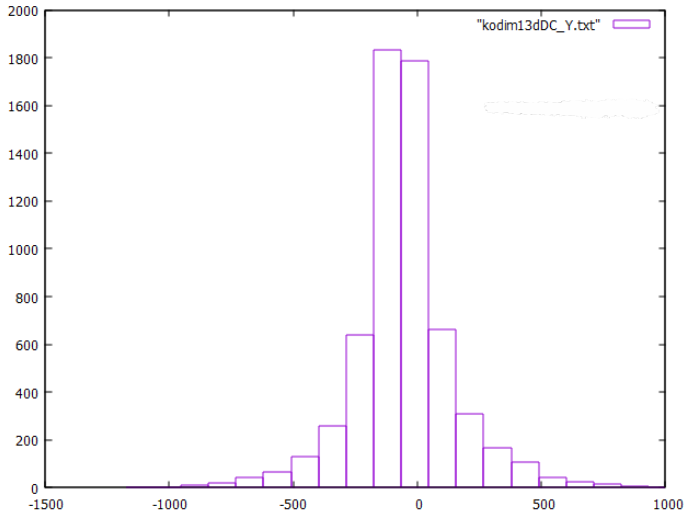

Рисунок 33. Гистограмма частот f(ΔDC) компоненты Y

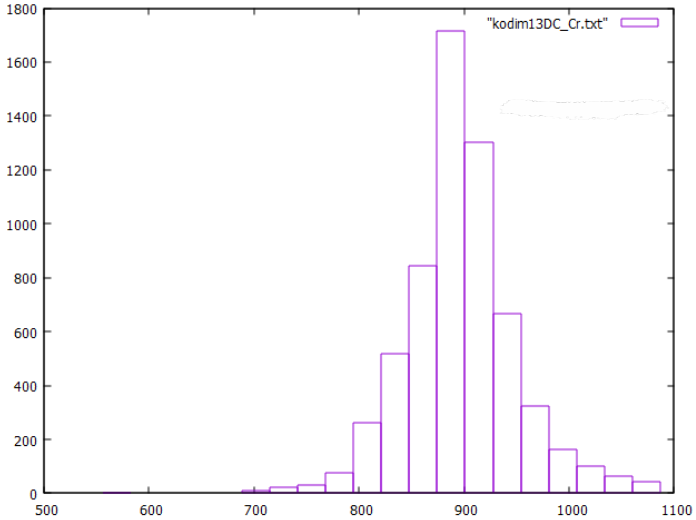

Рисунок 34. Гистограмма частот f(DC) компоненты Cr

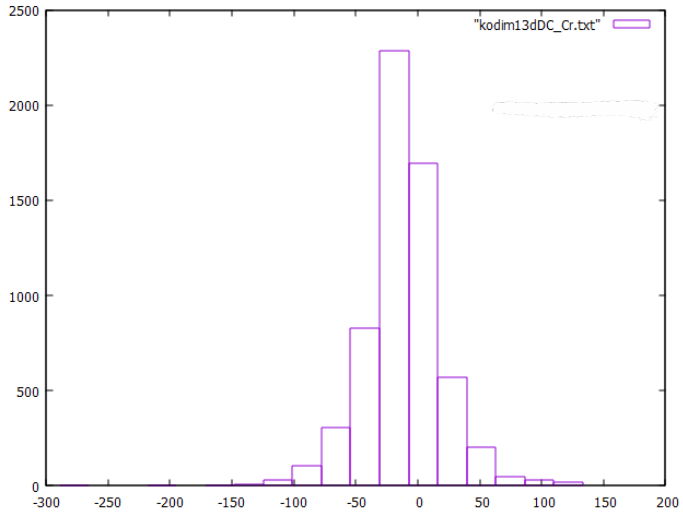

Рисунок 35. Гистограмма частот f(ΔDC) компоненты Cr

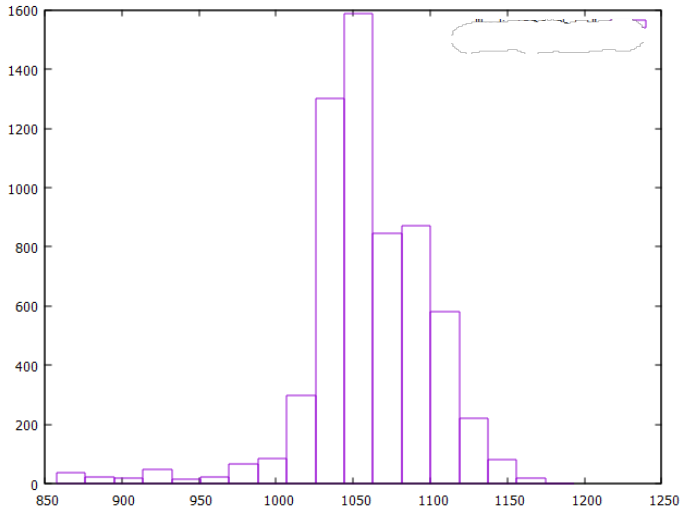

Рисунок 36. Гистограмма частот f(DC) компоненты Cb

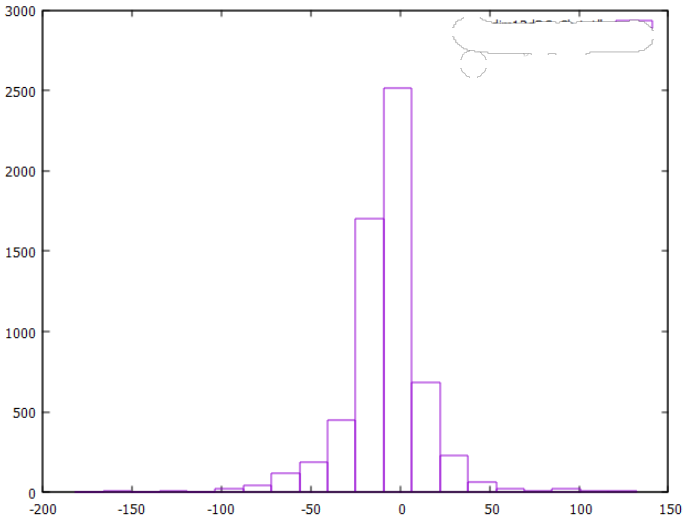

Рисунок 37. Гистограмма частот f(ΔDC) компоненты Cb

Значения энтропии DC^q:

H(Y) = 10.074
H(Cr) = 7.618
H(Cb) = 7.120

Значение энтропии ΔDC:

H(Y) = 9.346
H(Cr) = 6.886
H(Cb) = 6.308

Гистограмма частот для изображения lena.bmp

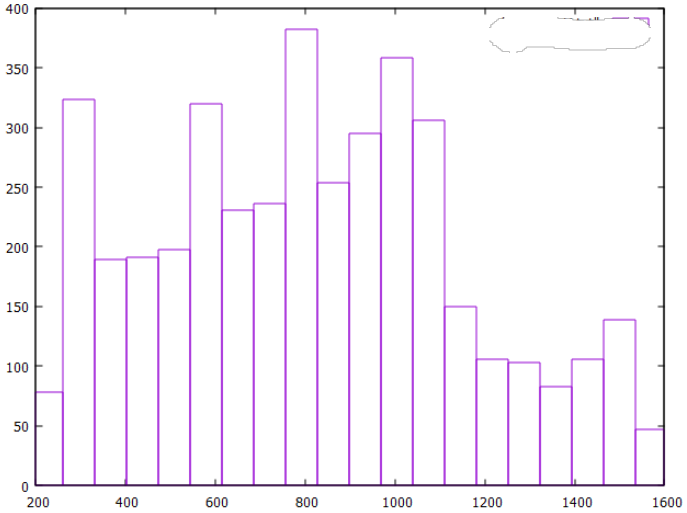

Рисунок 38. Гистограмма частот f(DC) компоненты Y

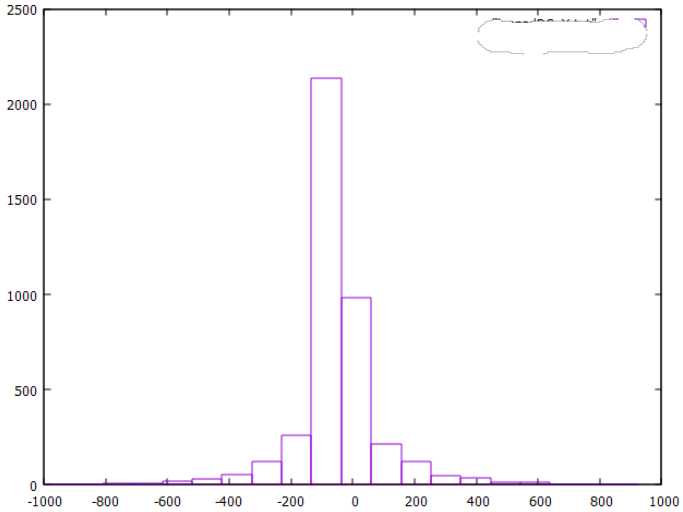

Рисунок 39. Гистограмма частот f(ΔDC) компоненты Y

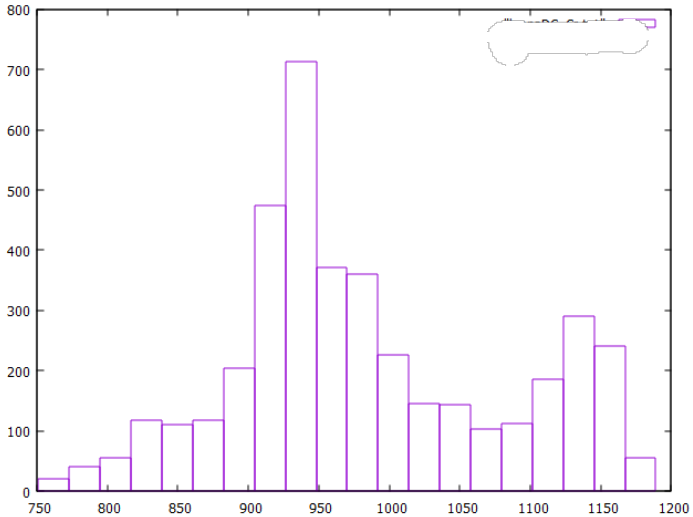

Рисунок 40. Гистограмма частот f(DC) компоненты Cr

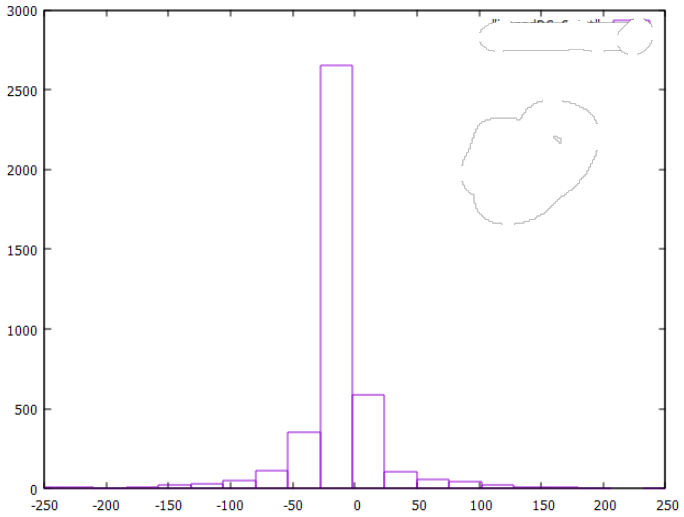

Рисунок 41. Гистограмма частот f(ΔDC) компоненты Cr

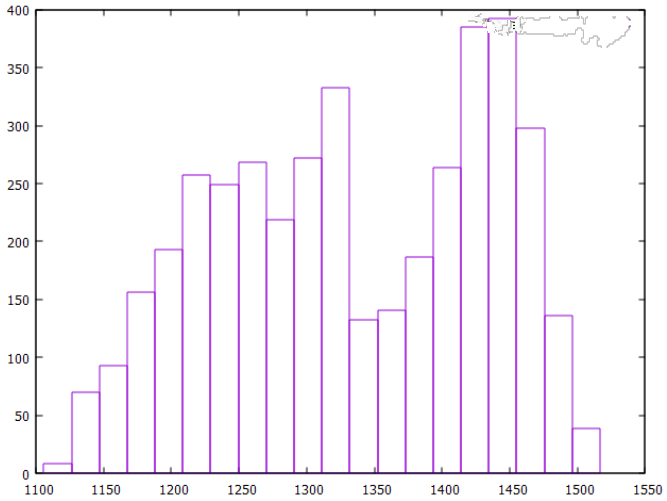

Рисунок 42. Гистограмма частот f(DC) компоненты Cb

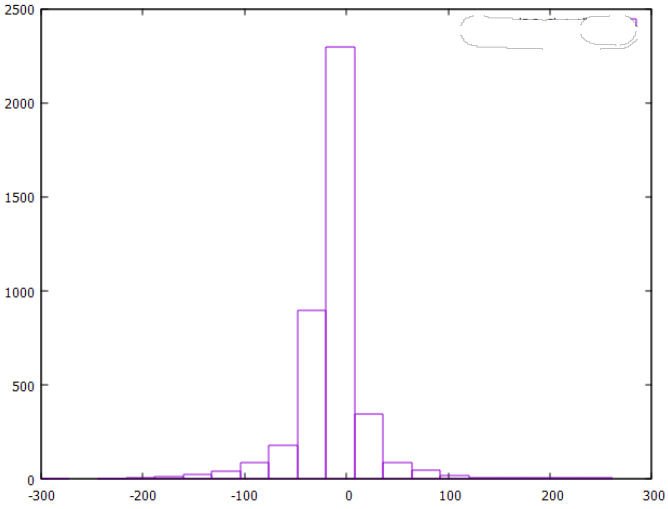

Рисунок 43. Гистограмма частот f(ΔDC) компоненты Cb

Значения энтропии DC^q:

H(Y) = 9.988
H(Cr) = 8.263
H(Cb) = 8.371

Значение энтропии ΔDC:

H(Y) = 8.301
H(Cr) = 6.361
H(Cb) = 6.667

3.3 Реализация процедуры кодирования длинами серий (RLE)

Сформированная в предыдущем пункте последовательность коэффициентов AC q необходимо закодировать длинами серий. Данный этап кодирования состоит из 3 частей:

1) Перегруппировка коэффициентов переменного тока AC^q в соответствии с зигзагообразной последовательность.
2) Этап кодирования длин серии, в результате которого предыдущая последовательность заменяется на новую последовательность, состоящую из пар (Run, Level). Run определяет число нулевых значений в серии, а Level — ненулевой завершающий элемент серии. Если в старой последовательности не остаётся ненулевых значений, то в конце новой последовательности ставится пара (0,0).
3) Значение Level заменяется на пару BC(Level), Magnitude(Level).

3.4 Определение соотношений размеров в сжатом битовом потоке

Необходимо определить соотношение размеров в сжатом битовом потоке для:

— BC(ΔDC)
— Magnitude(ΔDC)
— (Run, BC(Level)
— Magnitude(Level)

Результат работы для изображения baboon.bmp

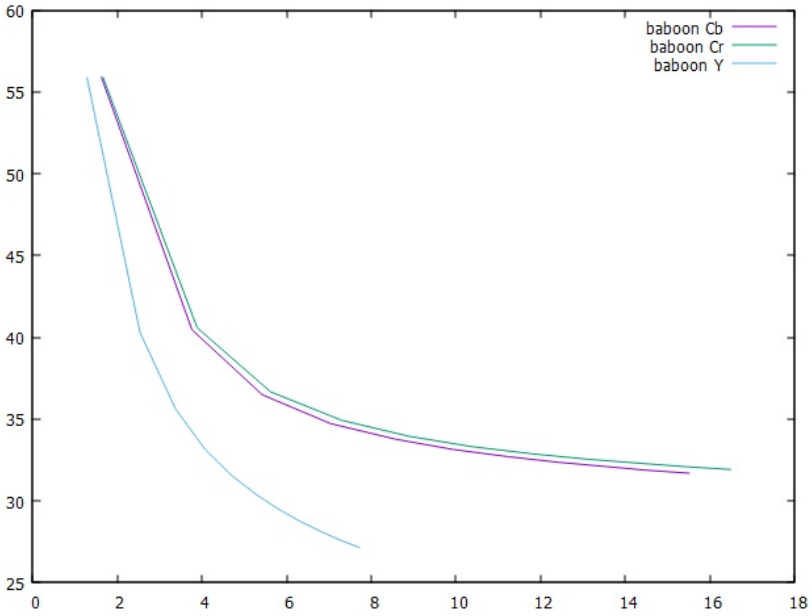

Рисунок 44. Зависимость PSNR от степени сжатия

```
R: 1
Y
BC(DC): 1.3876441374939281%
MG(DC): 3.218985075424744%
BC(AC): 51.87657333346746%
MG(AC): 43.516797453613876%
Cr
BC(DC): 2.076835789032071%
MG(DC): 3.756206960680614%
BC(AC): 58.49216437989512%
MG(AC): 35.67479287039219%
Cb
BC(DC): 2.1146412665042384%
MG(DC): 3.8246976327200235%
BC(AC): 58.07570821777122%
MG(AC): 35.98495288300452%
```

```
R: 10
Y
BC(DC): 4.210664637099743%
MG(DC): 9.767681971345574%
BC(AC): 61.11729552626555%
MG(AC): 24.904357865289136%
Cr
BC(DC): 8.790699259285502%
MG(DC): 15.899035408267526%
BC(AC): 60.86830090696797%
MG(AC): 14.441964425478998%
Cb
BC(DC): 8.701560068582245%
MG(DC): 15.738289383851335%
BC(AC): 60.11982560178728%
MG(AC): 15.440324945779151%
```

Результат работы для изображения kodim04.bmp

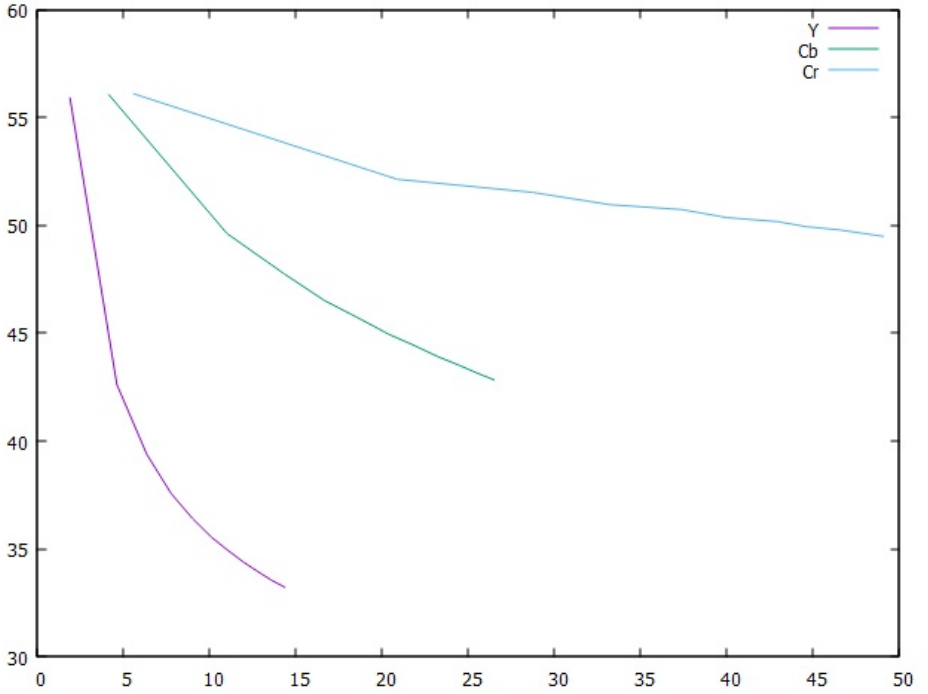

Рисунок 45. График зависимости PSNR от степени сжатия

```
R: 1
Y
BC(DC): 2.73772261887234%
MG(DC): 4.8998624264787125%
BC(AC): 58.07398787861517%
MG(AC): 34.28842707603378%
Cr
BC(DC): 10.775915707634601%
MG(DC): 10.201414104234933%
BC(AC): 64.54919813371019%
MG(AC): 14.473472054420274%
Cb
BC(DC): 6.439800461893277%
MG(DC): 8.382309345244476%
BC(AC): 60.99627847390131%
MG(AC): 24.181611718960927%
```

```
R: 10
Y
BC(DC): 8.477412375670806%
MG(DC): 15.172521163018505%
BC(AC): 58.97040912598096%
MG(AC): 17.37965733532973%
Cr
BC(DC): 25.313060911530723%
MG(DC): 23.96353345834874%
BC(AC): 47.64607866361949%
MG(AC): 3.0773269665010456%
Cb
BC(DC): 15.48130275430152%
MG(DC): 20.15110087988488%
BC(AC): 54.04650623994187%
MG(AC): 10.32109012587173%
```

Результат работы для lena.bmp

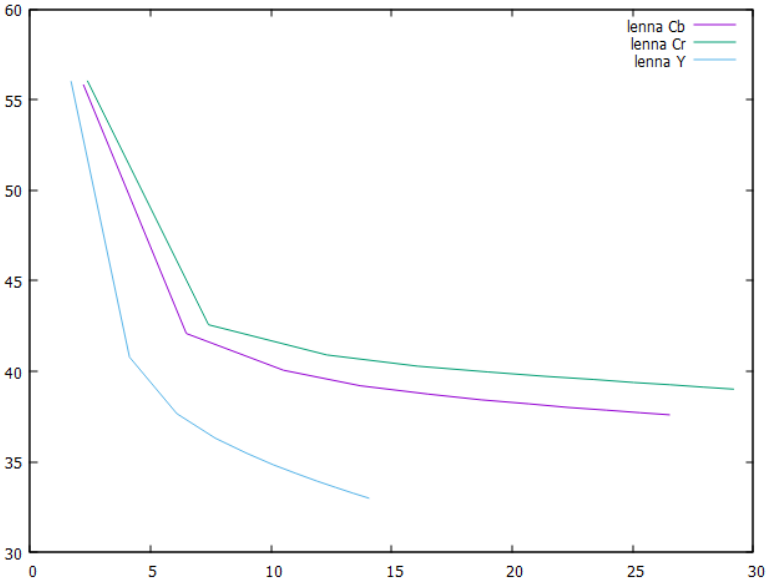

Рисунок 46. График зависимости PSNR от степени сжатия

```
R: 1
Y
BC(DC): 2.504899327899197%
MG(DC): 4.460954003807762%
BC(AC): 58.92727489223388%
MG(AC): 34.10687177605916%
Cr
BC(DC): 4.160759109220273%
MG(DC): 5.241313038507246%
BC(AC): 66.495668501782%
MG(AC): 24.102259350490474%
Cb
BC(DC): 3.635581741203876%
MG(DC): 5.018197173069432%
BC(AC): 65.2161010693088%
MG(AC): 26.130120016417887%
```

```
R: 10
Y
BC(DC): 8.502585399048597%
MG(DC): 15.142182344874 51%
BC(AC): 57.934790217844004%
MG(AC): 18.420442038232895%
Cr
BC(DC): 16.36674017389922%
MG(DC): 20.617201433561032%
BC(AC): 54.13341121620459%
MG(AC): 8.882647176335167%
Cb
BC(DC): 14.840378786234949%
MG(DC): 20.484189924362248%
BC(AC): 55.164101774467774%
MG(AC): 9.511329514935019%
```

Дополнительное задание

Вариант 3.

Постройте на одном графике зависимости Процент BC(Run, Level) в сжатых данных для AC коэффициентов от параметра R. И Процент Magnitude(level) в сжатых данных для AC коэффициентов от параметра R. То есть интересует вклад каждой из этих составляющих в информацию, сохраняемую для AC коэффициентов в сжатом потоке:

100% * Size(BC(Run, Level)) / (Size(BC(Run, Level)) + Size(Magnitude(level)))
100% * Size(Magnitude(level)) / (Size(BC(Run, Level)) + Size(Magnitude(level)))

Постройте данные зависимости от параметра R для Y компонент вашего изображения, изображения lena и peppers на одном графике. Для данного графика используйте значения R в диапазоне от 1 до 60 с шагом 3.
Сделайте вывод какая часть информации, сохраняемой для коэффициентов переменного тока, занимает больше места в сжатом потоке и как меняется соотношение в зависимости от степени сжатия, и почему.

Рисунок 47. Текст задания

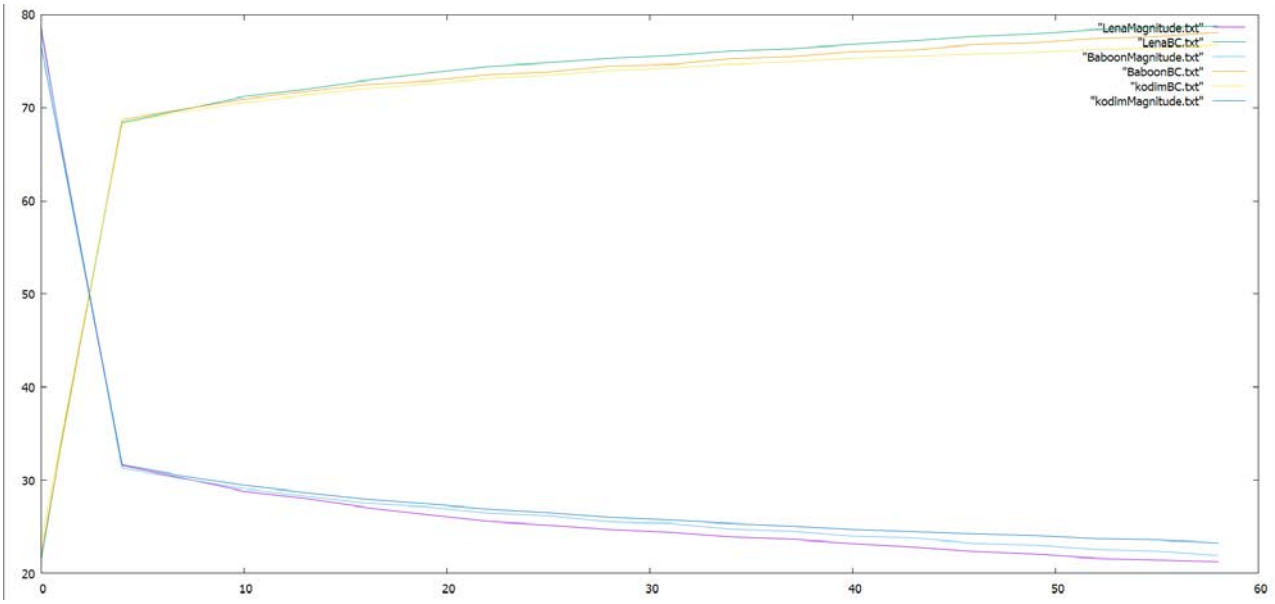

Рисунок 48. Графики зависимости вклада составляющих в информацию от значения R

Из полученных графиков можно сделать вывод, что BC (Run, Level) занимает больше места, чем Magnitude(Level). При увеличении R растёт процент BC (Run, Level), а процент Magnitude(Level) уменьшается, следовательно, при увеличении степени сжатия R число нулевых коэффициентов AC увеличивается, а количество ненулевых уменьшается. Значения пересекаются примерно при R равном 5.

Вывод

В ходе лабораторной работы были изучены методы блоковой обработки при сжатии в стандарте JPEG.

1) Дискретное косинусное преобразование.
   ДКП и обратное ДКП визуально не меняет изображение, это отображают и значения PSNR.

2) Квантование спектральных коэффициентов.
   Из результирующих графиков следует, что значения PSNR уменьшаются при повышении степени сжатия изображения, а компонента Y уменьшается сильнее всех, т.к. эта компонента содержит наибольшее количество информации. Для baboon.bmp компонента Y имеет наихудший показатель PSNR из всех, поскольку в данном изображении часто происходят переходы между значениями яркости.

3) Кодирование без потерь.
   Из результирующих графиков следует, что хуже всего сжимается Y компонента, поскольку она содержит всю основную информацию. При увеличении R, AC сжимается, в то время как DC занимает все больше места в процентном соотношении. Происходит это из-за того, что DC не сжимается, а AC, в свою очередь, занимает все меньше места.

Листинг программы

```
public class Main {

    public static void main(String[] args) throws IOException {
        MT3 mt3 = new MT3("kodim04.bmp");
    }
}
public class Block {
    private final int N;
    private final long[][] block;
    private int DC;
    private int dDC;
    private int[] AC;

    Pair pair_dDC;
    ArrayList<Pair> run_level;
    ArrayList<Triple> run_BC_MG;

    Block(int N) {
        this.N = N;
        block = new long[N][N];
    }

    double C(int f) {
        if (f == 0) return Math.sqrt(1.0 / N);
        return Math.sqrt(2.0 / N);
```

```
    }

    void DKP(double[][] A, int ii, int jj) {
        for (int k = 0; k < N; k++) {
            for (int l = 0; l < N; l++) {
                double sum = 0;
                for (int i = 0; i < N; i++) {
                    for (int j = 0; j < N; j++) {
                        sum += (A[ii + i][jj + j]) * cos(i, k) * cos(j, l);
                    }
                }
                block[k][l] = Math.round(C(k) * C(l) * sum);
            }
        }
    }

    double[][] ODKP(double[][] A, int ii, int jj) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                double tmp = 0;
                for (int k = 0; k < N; k++) {
                    for (int l = 0; l < N; l++) {
                        tmp += (C(k) * C(l) * block[k][l] * cos(i, k) * cos(j,
l));
                    }
                }
                A[ii + i][jj + j] = Math.round(tmp);
            }
        }
        return A;
    }


    int[][] Q(int R) {
        int[][] Q = new int[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                Q[i][j] = 1 + ((i + j) * R);
            }
        }
        return Q;
    }


    void Quantization(int R, boolean de) {
        int[][] Q = Q(R);
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                block[i][j] = (!de) ? Math.round((double) block[i][j] / Q[i][j])
: (block[i][j] * Q[i][j]);
            }
        }
    }


    double cos(int i, int k) {
        return Math.cos((2 * i + 1) * Math.PI * k / (2 * N));
    }


    void ACDC() {
        DC = (int) block[0][0];
        AC = new int[-1 + N * N];
```

```
    int i = 0;
    int j = 0;
    int index = 0;
    while (index < 63) {
        if ((i + j) % 2 == 0) {
            if (i + j < N) {
                if (i != 0) i--;
                j++;
            } else {
                if (j != N - 1) {
                    j++;
                    i--;
                } else i++;
            }
        } else {
            if (i + j < N) {
                if (i == N - 1) j++;
                else {
                    if (j != 0) j--;
                    i++;
                }
            } else {
                if (i != N - 1) {
                    i++;
                    j--;
                } else j++;
            }
        }
        AC[index] = (int) block[i][j];
        index++;
    }
}

public int getdDC() {
    return dDC;
}

void dDC(int DC) {
    dDC = this.DC - DC;
}

int getBC_DC() {
    return pair_dDC.getKey();
}

public int getDC() {
    return DC;
}

public void createPair_dDC() {
    pair_dDC = new Pair(BC(dDC), dDC);
}

public int BC(int x) {
    return (int) Math.ceil(log2(x));
}

public void createPair_runlevel() {
    int Run = 0;
    int Level;
    run_level = new ArrayList<>();
    for(int i = 0; i < AC.length; i++) {
        Level = AC[i];
```

```
            if (Level != 0) {
                run_level.add(new Pair(Run, AC[i]));
                Run = 0;
            } else if (Run == 16) {
                run_level.add(new Pair(15, 0));
                Run = 0;
            }
            Run++;
        }
        run_level.add(new Pair(0, 0));
    }

    public void createTriple() {
        run_BC_MG = new ArrayList<>();
        for (Pair pair : run_level) {
            run_BC_MG.add(new Triple(pair.getKey(), new
Pair(BC(pair.getValue()), pair.getValue())));
        }
    }

    public int getRunLevel(int i) {
        return run_BC_MG.get(i).getKey() + run_BC_MG.get(i).getPair().getKey();
    }

    public int getMGLevel() {
        int res = 0;
        for (Triple triple : run_BC_MG) {
            res += triple.getPair().getKey();
        }
        return res;
    }

    public double log2(int x) {
        return Math.log(Math.abs(x) + 1) / Math.log(2);
    }

}
public class MT3 {
    private String pathImages = "C:\\Users\\Don\\Desktop\\6
семак\\MT\\Images\\";
    private String pathFiles = "C:\\Users\\Don\\Desktop\\6 семак\\MT\\Files\\";
    ArrayList<Double> BC = new ArrayList<>();
    ArrayList<Double> AC = new ArrayList<>();

    public MT3(String filename) throws IOException {
        File file = new File(pathImages + filename);
        BufferedImage img = ImageIO.read(file);
        int W = img.getWidth();
        int H = img.getHeight();

        double[][][] pixels = getPixels(img);
        double[][] RED = new double[W][H];
        double[][] GREEN = new double[W][H];
        double[][] BLUE = new double[W][H];
        for (int i = 0; i < W; i++) {
            for (int j = 0; j < H; j++) {
                RED[i][j] = pixels[i][j][2];
                GREEN[i][j] = pixels[i][j][1];
                BLUE[i][j] = pixels[i][j][0];
            }
        }

        double[][] Y = new double[W][H];
```

```
        double[][] Cb = new double[W][H];
        double[][] Cr = new double[W][H];
        for (int i = 0; i < W; i++) {
            for (int j = 0; j < H; j++) {
                Y[i][j] = 0.299 * RED[i][j] + 0.587 * GREEN[i][j] + 0.114 *
BLUE[i][j];
                Cb[i][j] = 0.5643 * (BLUE[i][j] - Y[i][j]) + 128;
                Cr[i][j] = 0.7132 * (RED[i][j] - Y[i][j]) + 128;
            }
        }

        int N = 8;

        double[] c_ratioY = new double[11];
        double[] c_ratioCr = new double[11];
        double[] c_ratioCb = new double[11];

        for(int R = 0; R <= 10; R += 1) {
            ArrayList<Block> blockY = new ArrayList<>();
            ArrayList<Block> blockCr = new ArrayList<>();
            ArrayList<Block> blockCb = new ArrayList<>();

            double[][] newY = new double[W][H];
            double[][] newCb = new double[W][H];
            double[][] newCr = new double[W][H];
            System.out.println("R: " + R);
            for (int i = 0; i < W; i += N) {
                for (int j = 0; j < H; j += N) {
                    blockY.add(new Block(N));
                    blockCb.add(new Block(N));
                    blockCr.add(new Block(N));

                    blockY.get(blockY.size() - 1).DKP(Y, i, j);
                    blockCb.get(blockCb.size() - 1).DKP(Cb, i, j);
                    blockCr.get(blockCr.size() - 1).DKP(Cr, i, j);

                    blockY.get(blockY.size() - 1).Quantization(R, false);
                    blockCb.get(blockCb.size() - 1).Quantization(R, false);
                    blockCr.get(blockCr.size() - 1).Quantization(R, false);

                    blockY.get(blockY.size() - 1).ACDC();
                    blockCb.get(blockCb.size() - 1).ACDC();
                    blockCr.get(blockCr.size() - 1).ACDC();

                    // Если нужно вывести картинки
                    blockY.get(blockY.size() - 1).Quantization(R, true);
                    blockCb.get(blockCb.size() - 1).Quantization(R, true);
                    blockCr.get(blockCr.size() - 1).Quantization(R, true);

                    newY = blockY.get(blockY.size() - 1).ODKP(newY, i, j);
                    newCb = blockCb.get(blockCb.size() - 1).ODKP(newCb, i, j);
                    newCr = blockCr.get(blockCr.size() - 1).ODKP(newCr, i, j);

                }
            }

            dDC(blockY);
            dDC(blockCb);
            dDC(blockCr);

            BCDC(blockY);
            BCDC(blockCr);
            BCDC(blockCb);
```

```
            RunLevel(blockY);
            RunLevel(blockCr);
            RunLevel(blockCb);

            Triple(blockY);
            Triple(blockCb);
            Triple(blockCr);

            System.out.println("Y");
            Calculate(blockY, W, H);
            System.out.println("Cr");
            Calculate(blockCr, W, H);
            System.out.println("Cb");
            Calculate(blockCb, W, H);

            // Если нужно вывести PSNR
//            System.out.println("PSNR Y: " + PSNR(Y, newY, W, H));
//            System.out.println("PSNR Cr: " + PSNR(Cr, newCr, W, H));
//            System.out.println("PSNR Cb: " + PSNR(Cb, newCb, W, H));

        }
        String name = getName(filename);
        writeIntoFile(name + "DC_Cb.txt", BC);
        writeIntoFile(name + "AC_Cb.txt", AC);

        writeIntoFile(name + "RatioY.txt", c_ratioY);
        writeIntoFile(name + "RatioCr.txt", c_ratioCr);
        writeIntoFile(name + "RatioCb.txt", c_ratioCb);

    }

    public double[][][] getPixels(BufferedImage image) {
        double[][][] pixels = new
double[image.getWidth()][image.getHeight()][3];
        for (int i = 0; i < image.getWidth(); i++) {
            for (int j = 0; j < image.getHeight(); j++) {
                Color color = new Color(image.getRGB(i, j));
                pixels[i][j][0] = color.getRed();
                pixels[i][j][1] = color.getGreen();
                pixels[i][j][2] = color.getBlue();
            }
        }
        return pixels;
    }
    public double PSNR(double[][] A, double[][] newA, int W, int H) {
        double res = 0;
        for (int i = 0; i < W; i++) {
            for (int j = 0; j < H; j++) {
                res += Math.pow(A[i][j] - newA[i][j], 2);
            }
        }
        return 10 * Math.log10((W * H * Math.pow(Math.pow(2, 8) - 1, 2)) / res);
    }
    public int Mean(ArrayList<Block> blocks) {
        int res = 0;
        for(int i = 0; i < blocks.size(); i++) {
            res += ((double) blocks.get(i).getDC() / blocks.size());
        }
        return res;
    }
    public void dDC(ArrayList<Block> blocks) {
        blocks.get(0).dDC(Mean(blocks));
```

```
    for(int i = 1; i < blocks.size(); i++) {
        blocks.get(i).dDC(blocks.get(i-1).getDC());
    }
}
public void BCDC(ArrayList<Block> blocks) {
    for(int i = 0; i < blocks.size(); i++) {
        blocks.get(i).createPair_dDC();
    }
}
public void RunLevel(ArrayList<Block> blocks) {
    for(int i = 0; i < blocks.size(); i++) {
        blocks.get(i).createPair_runlevel();
    }
}
public void Triple(ArrayList<Block> blocks) {
    for(int i = 0; i < blocks.size(); i++) {
        blocks.get(i).createTriple();
    }
}
public double Calculate(ArrayList<Block> blocks, int W, int H) {
    ArrayList<Integer> DC = new ArrayList<>();
    ArrayList<Integer> AC = new ArrayList<>();

    int MG_DC = 0;
    int MG_AC = 0;
    for(int i = 0; i < blocks.size(); i++) {
        DC.add(blocks.get(i).getBC_DC());
        MG_DC += DC.get(i);
        for(int j = 0; j < blocks.get(i).run_BC_MG.size(); j++) {
            AC.add(blocks.get(i).getRunLevel(j));
        }
        MG_AC += blocks.get(i).getMGLevel();
    }
    double BC_DC = Hx(DC) * DC.size();
    double BC_AC = Hx(AC) * AC.size();

    double totalSize = BC_DC + MG_DC + BC_AC + MG_AC;

    System.out.println("BC(DC): " + BC_DC * 100.0 / totalSize + "%");
    System.out.println("MG(DC): " + MG_DC * 100.0 / totalSize + "%");
    System.out.println("BC(AC): " + BC_AC * 100.0 / totalSize + "%");
    System.out.println("MG(AC): " + MG_AC * 100.0 / totalSize + "%");

    this.BC.add((BC_DC + MG_DC) * 100 / totalSize);
    this.AC.add((BC_AC + MG_AC) * 100 / totalSize);
    return (double) (W * H * 8) / totalSize;
}


public double Hx(ArrayList<Integer> A) {
    double res = 0;
    for(int x = 0; x < 256; x++) {
        double p = p(A, x);
        res += p*log2(p);
    }
    return -res;
}
public double log2(double p) {
    if (p == 0) return 1;
    return Math.log(p)/Math.log(2);
}
public double p(ArrayList<Integer> A, int x) {
    double count = 0;
```

```
        for(int i = 0; i < A.size(); i++) {
            if (A.get(i) == x) count++;
        }
        return count/A.size();
    }

    public String getName(String filename) {
        return filename.substring(0, filename.indexOf(".bmp"));
    }
    public void writeIntoFile(String filename, double[] a) throws IOException {
        File file = new File(pathFiles + filename);
        if (!file.exists()) file.createNewFile();
        FileWriter writer = new FileWriter(file);
        for(int i = 0; i < a.length; i++) {
            writer.append(Double.toString(a[i])).append("\n");
            writer.flush();
        }
        writer.close();
    }

    public void writeIntoFile(String filename, ArrayList<Double> a) throws
IOException {
        File file = new File(filename);
        if (!file.exists()) file.createNewFile();
        FileWriter writer = new FileWriter(file);
        int R = 0;
        for (Double aDouble : a) {
            writer.append(Integer.toString(R)).append("
").append(Double.toString(aDouble)).append("\n");
            writer.flush();
            R += 5;
        }
        writer.close();
    }

    double saturation(double x) {
        if (x < 0) return 0;
        if (x > 255) return 255;
        return x;
    }
    double[][][] toRGB(double[][] Y, double[][] Cb, double[][] Cr, int W, int H)
{
        double[][][] RGB = new double[W][H][3];
        for (int i = 0; i < W; i++) {
            for (int j = 0; j < H; j++) {
                RGB[i][j][2] = saturation(Y[i][j] + 1.772 * (Cb[i][j] - 128));
                RGB[i][j][1] = saturation(Y[i][j] - 0.714 * (Cr[i][j] - 128) -
0.334 * (Cb[i][j] - 128));
                RGB[i][j][0] = saturation(Y[i][j] + 1.402 * (Cr[i][j] - 128));
            }
        }
        return RGB;
    }
    public void setPixel(BufferedImage image, File file, double[][][] pixels)
throws IOException {
        for (int i = 0; i < pixels.length; i++) {
            for (int j = 0; j < pixels[i].length; j++) {
                int res = 0;
                for (int k = 0; k < 3; k++) {
                    res += (int) pixels[i][j][k] << (8 * k);
                }
                image.setRGB(i, j, res);
            }
```

```
            }
            ImageIO.write(image, "bmp", file);
        }
    }
```