МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Университет ИТМО»

УТВЕРЖДАЮ
Доцент
_____ И. Е. Бочарова
«_____» _____ 2014 г.

ОТЧЕТ
О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ
по курсу «Мультимедиа технологии»

по теме:
Кодирование речевых сигналов на основе линейного предсказания

Студент гр. 3511 _____ Трофимов В.А.
Студент гр. 3511 _____ Шобей А.В.

Санкт-Петербург
2014

Вариант 4

Задание

Задан речевой сигнал (каждый отсчет представлен 16 битовым целым числом). Разделить сигнал на кадры по 240 отсчетов каждый.
Для каждого кадра:

- Вычислите коэффициенты уравнений Юла-Уокера, описывающих фильтр 10 порядка, автокорреляционным методом.
- Найдите решение уравнений Юла-Уокера методом Левинсона-Дарбина.
- Запишите рекуррентное уравнение предсказывающего фильтра.
- Запишите рекуррентное уравнение синтезирующего фильтра.
- Запишите передаточную функцию предсказывающего фильтра и его амплитудно-частотную характеристику.
- Запишите передаточную функцию синтезирующего фильтра и его амплитудно-частотную характеристику.
- Вычислите сигнал ошибки предсказания.
- Выполните равномерное скалярное квантование коэффициентов фильтра и квантованного сигнала ошибки.
- Оцените число битов на представление квантованных коэффициентов фильтра и квантованного сигнала ошибки.
- Оцените коэффициент сжатия.
- Восстановите речевой сигнал из квантованных данных
- Оцените относительную среднеквадратичную ошибку, возникающую при аппроксимации исходного сигнала восстановленным.

Программа

```
function [] = lab2_04_trofiv_shobey()

    clc;

    n = 10;
    frameSize = 240;
    inputFileName = 'INST.WAV';
    step = 0.00306;
    initinalStepE = 0.001;
    repeats = 30;

    compression = zeros(repeats, 1);
    relativeError = zeros(repeats, 1);

    for i = 1 : repeats

        stepE = initinalStepE * i;
        traceFrame = i;

        fprintf('Input file: %s\n', inputFileName);
        fprintf('Step: %f\n', step);
        fprintf('StepE: %f\n', stepE);

        [compression(i), relativeError(i)] = process(inputFileName, n, step,
stepE, frameSize, traceFrame);

        fprintf('\n\n');

    end
```

```
    plot(compression, relativeError);

end

function [header, data] = readFile(inputFileName)
    file = fopen(inputFileName, 'r');
    stream = fread(file, 'short');
    fclose(file);
    header = stream(1 : 44);
    data = stream(45 : length(stream));
end

function [] = writeFile(header, data, outputFileName)
    file = fopen(outputFileName, 'w');
    header = header';
    buffer = [header data];
    fwrite(file, buffer, 'short');
    fclose(file);
end

function[result] = entropy(x)

    maxX = max(x);
    minX = min(x);
    result = 0;
    p = zeros(maxX - minX + 1, 1);

    for k = 1 : length(x)
        p(x(k) - minX + 1) = p(x(k) - minX + 1) + 1;
    end

    for k = 1 : length(p)
        if p(k) ~= 0
            result = result - (p(k) / length(x) * log2(p(k) / length(x)));
        end
    end

end

function [compression, relativeError] = process(inputFileName, m, step,
stepE, frameSize, traceFrame)

    [header, data] = readFile(inputFileName);

    frameCount = floor(length(data) / frameSize);
    fprintf('Total frames: %d\n\n', frameCount);

    input = zeros(frameCount * frameSize, 1);

    quantizedA = [];
    quantizedE = [];

    restoredA = [];
    restoredE = [];

    restoredData = [];

    squareDifferencefForD = 0;
    squareForD = 0;
```

```
p = m + 1;

for i = 1 : frameCount

    if (i == traceFrame)
        fprintf('Trace for frame %d:\n', i);
    end

    currentFrame = data((i - 1) * frameSize + 1 : i * frameSize);
    input((i - 1) * frameSize + 1 : i * frameSize) = currentFrame;

    % Autocorrelation method
    c = zeros(p);

    for j = 1 : p
        for k = 1 : p
            l = currentFrame(1 : frameSize - abs(j - k));
            r = currentFrame(abs(j - k) + 1 : frameSize);
            c(j, k) = l' * r / frameSize;
        end
    end

    r = zeros(1, p);
    for j = 1 : p
        r(j) = c(1, j) / c(1, 1);
    end

    if (i == traceFrame)
        fprintf('Yule-Walker coefficients:\n');
        for j = 1 : p
            fprintf('R(%d) = %9.6f\n', j, r(j));
        end
    end

    % Levinson-Durbin
    E = zeros(1, p);
    E(1) = r(1);
    a = 0;

    for j = 2 : p

        if (j == 2)
            currentA = (r(j) - (0 * r(j - 1))) / E(j - 1);
            a = currentA;
        else
            currentA = (r(j) - sum(a .* r(j - 1 : -1 : 2))) / E(j - 1);
            a(1 : j - 2) = a(1 : j - 2) - currentA * a(j - 2 : -1 : 1);
            a = [a currentA]; %#ok<AGROW>
        end

        currentE = E(j - 1) * (1 - a(j - 1) ^ 2);
        E(j) = currentE;

    end

    if (i == traceFrame)
        fprintf('\nLevinson-Durbin solution:\n');
        for j = 1 : m
            fprintf('a(%d) = %9.6f\n', j, a(j));
        end
```

```
            fprintf('E = %9.6f\n\n', E(m));
        end

        tempCurrent = zeros(1, frameSize + p - 1);
        tempCurrent(p : length(tempCurrent)) = currentFrame;
        e = zeros(frameSize + p - 1);

        for j = p : frameSize + p - 1
            e(j) = tempCurrent(j) - sum(a .* tempCurrent(j - 1 : -1 : j - p +
1));
        end

        quantizedA = [quantizedA round(a / step)]; %#ok<AGROW>
        temp = round(e / stepE);
        quantizedE = [quantizedE temp(p : frameSize + p - 1)];  %#ok<AGROW>

        indexStart = (i - 1) * frameSize + 1;
        indexEnd = (i - 1) * frameSize + frameSize;
        quantizedEBlock = (quantizedE(indexStart : indexEnd));

        restoredE = [restoredE stepE * quantizedEBlock]; %#ok<AGROW>

        indexStart = (i - 1) * m + 1;
        indexEnd = (i - 1) * m + m;
        quantizedABlock = quantizedA(indexStart : indexEnd);

        restoredA = [restoredA step * quantizedABlock]; %#ok<AGROW>

        indexStart = (i - 1) * frameSize + 1;
        indexEnd = (i - 1) * frameSize + frameSize;
        tempE = zeros(1, frameSize + p - 1);
        tempE(p : frameSize + p - 1) = restoredE(indexStart : indexEnd);

        temp = zeros(1, frameSize + p - 1);

        for j = p : frameSize + p - 1
            l = restoredA((i - 1) * m + 1 : (i - 1) * m + m);
            r = temp(j - 1 : -1 : j - m);
            summ = sum(l .* r);
            temp(j) = tempE(j) + summ;
        end

        restoredData = [restoredData temp(p : frameSize + p - 1)];
%#ok<AGROW>

        encoded = temp(p : p + frameSize - 1);
        startIndex = (i - 1) * frameSize + 1;
        endIndex = (i - 1) * frameSize + frameSize;
        original = data(startIndex : endIndex);

        for j = 1 : frameSize
            squareForD = squareForD + (encoded(j)) ^ 2;
            squareDifferencefForD = squareDifferencefForD + (encoded(j) -
original(j)) ^ 2;
        end

        if (i == traceFrame)

            fprintf('Amplitude function\n');
            fprintf('A(w)=sqrt( (1 ');
```

```
            for alpha = 1 : p - 1
                if (a(alpha) > 0)
                    fprintf('- %4.3f * cos(%d wT) ', a(alpha), alpha);
                else
                    fprintf('+ %4.3f * cos(%d wT) ', -a(alpha), alpha);
                end
            end

            fprintf(')^2 + ( ');

            for alpha = 1 : p - 1
                if (a(alpha) < 0)
                    fprintf('- %4.3f * sin(%d wT) ', -a(alpha), alpha);
                else
                    fprintf('+ %4.3f * sin(%d wT) ', a(alpha), alpha);
                end;
            end;

            fprintf(')^2)\n\n');

            fprintf('Prediction filter: \ne(n) = x(n)-');
            for j = 1 : m
                fprintf('(%.3f)*x(n-%i)', a(j), j);
                if (j ~= m)
                    fprintf('-');
                end
            end
            fprintf('\n\n');

            fprintf('Synthesis filter: \nxs(n) = e(n)+');
            for j = 1 : m
                fprintf('(%.3f)*x(n-%i)', a(j), j);
                if (j ~= m)
                    fprintf('+');
                end
            end;
            fprintf('\n\n');

            fprintf('Transfer function of prediction filter: \nA(z)=1-
((%.3f)*(1/z)+', a(1));
            for j = 2 : m
                fprintf('(%.3f)*(Z^-%i)', a(j),j);
            end;
            fprintf(')\n\n');

            fprintf('Transfer function of synthesis filter: \nH(Z)=1/(1 -
((%.3f)*(1/z)', a(1));
            for j = 2 : p - 1
                fprintf('+(%.3f)*(Z^-%i)', a(j),j);
            end;
            fprintf('))\n\n');

        end

    end

    name = strcat('restore', int2str(stepE), '.wav');

    writeFile(header, restoredData, name);

    D = squareDifferencefForD / (frameCount * frameSize);
```

```
    Drelative = squareDifferencefForD / squareForD;
    fprintf('D: %f\nD relative: %f\n\n', D, Drelative);

    fprintf('Entropy for a: %f\n', entropy(quantizedA));
    fprintf('Average bit count for a: %f\n', entropy(quantizedA) *
(length(quantizedA)));

    fprintf('Entropy for e: %f\n', entropy(quantizedE));
    fprintf('Average bit count for a: %f\n\n', entropy(quantizedE) *
(length(quantizedE)));

    he = entropy(quantizedE);
    ha = entropy(quantizedA);
    originalSize = 16 * frameSize * frameCount;
    compressedSize = (ha * m * frameCount + he * frameSize * frameCount);
    compressionLevel = originalSize / compressedSize;
    fprintf('Compression level: %f\n\n', compressionLevel);

    relativeError = Drelative;
    compression = compressionLevel;

end
```

Вывод программы для step=0.02 и StepE = 100

Input file: INST.WAV
Step: 0.020000
StepE: 100.000000
Total frames: 103

Trace for frame 1:
Yule–Walker coefficients:
R(1) = 1.000000
R(2) = 0.661093
R(3) = 0.320301
R(4) = 0.160450
R(5) = 0.086376
R(6) = -0.010229
R(7) = -0.127930
R(8) = -0.080858
R(9) = -0.168862
R(10) = -0.296428
R(11) = -0.294176

Levinson-Durbin solution:
a(1) = 0.852512
a(2) = -0.294970
a(3) = 0.036557
a(4) = 0.047141
a(5) = 0.056005
a(6) = -0.327546
a(7) = 0.412626
a(8) = -0.258013
a(9) = -0.107623
a(10) = 0.034159
E = 0.448087

Amplitude function

A(w)=sqrt((1 - 0.853 * cos(1 wT) + 0.295 * cos(2 wT) - 0.037 * cos(3 wT) - 0.047 * cos(4 wT) - 0.056 * cos(5 wT) + 0.328 * cos(6 wT) - 0.413 * cos(7 wT) + 0.258 * cos(8 wT) + 0.108 * cos(9 wT) - 0.034 * cos(10 wT))^2 + (+ 0.853 * sin(1 wT) - 0.295 * sin(2 wT) + 0.037 * sin(3 wT) + 0.047 * sin(4 wT) + 0.056 * sin(5 wT) - 0.328 * sin(6 wT) + 0.413 * sin(7 wT) - 0.258 * sin(8 wT) - 0.108 * sin(9 wT) + 0.034 * sin(10 wT))^2)

Prediction filter:
e(n) = x(n)-(0.853)*x(n-1)-(-0.295)*x(n-2)-(0.037)*x(n-3)-(0.047)*x(n-4)-(0.056)*x(n-5)-(-0.328)*x(n-6)-(0.413)*x(n-7)-(-0.258)*x(n-8)-(-0.108)*x(n-9)-(0.034)*x(n-10)

Synthesis filter:
xs(n) = e(n)+(0.853)*x(n-1)+(-0.295)*x(n-2)+(0.037)*x(n-3)+(0.047)*x(n-4)+(0.056)*x(n-5)+(-0.328)*x(n-6)+(0.413)*x(n-7)+(-0.258)*x(n-8)+(-0.108)*x(n-9)+(0.034)*x(n-10)

Transfer function of prediction filter:
A(z)=1-((0.853)*(1/z)+(-0.295)*(Z^-2)(0.037)*(Z^-3)(0.047)*(Z^-4)(0.056)*(Z^-5)(-0.328)*(Z^-6)(0.413)*(Z^-7)(-0.258)*(Z^-8)(-0.108)*(Z^-9)(0.034)*(Z^-10))

Transfer function of synthesis filter:
H(Z)=1/(1 - ((0.853)*(1/z)+(-0.295)*(Z^-2)+(0.037)*(Z^-3)+(0.047)*(Z^-4)+(0.056)*(Z^-5)+(-0.328)*(Z^-6)+(0.413)*(Z^-7)+(-0.258)*(Z^-8)+(-0.108)*(Z^-9)+(0.034)*(Z^-10)))

D: 259686.353713
D relative: 0.030801

Entropy for a: 6.416572
Average bit count for a: 6609.068666
Entropy for e: 4.142095
Average bit count for a: 102392.589660

Compression level: 3.628569

График зависимости относительной среднеквадратичной ошибки от степени сжатия при переборе StepE от 15 до 900 и step = 0.02

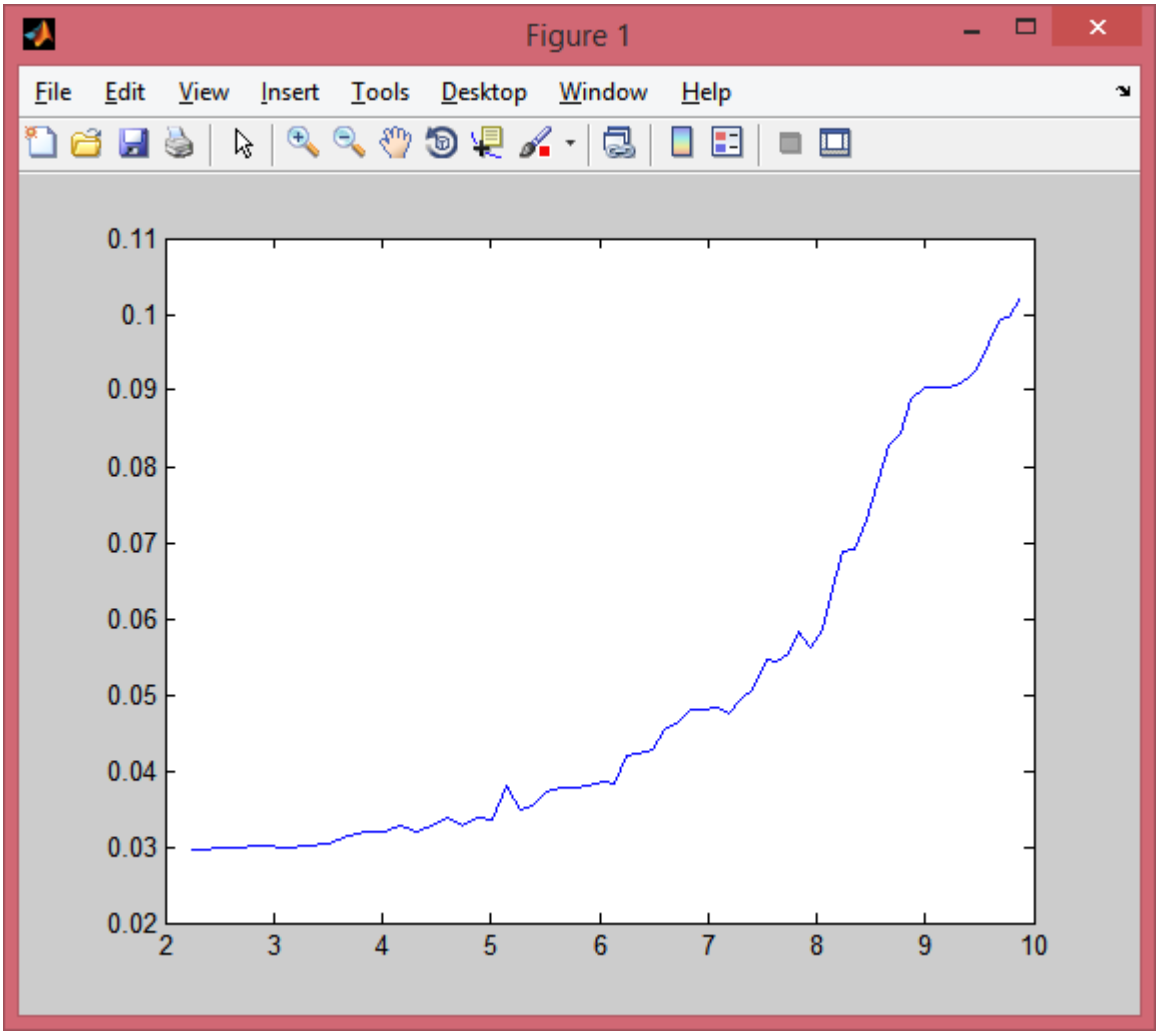


График зависимости относительной среднеквадратичной ошибки от степени сжатия при переборе StepE от 1 до 30 и step = 0.02

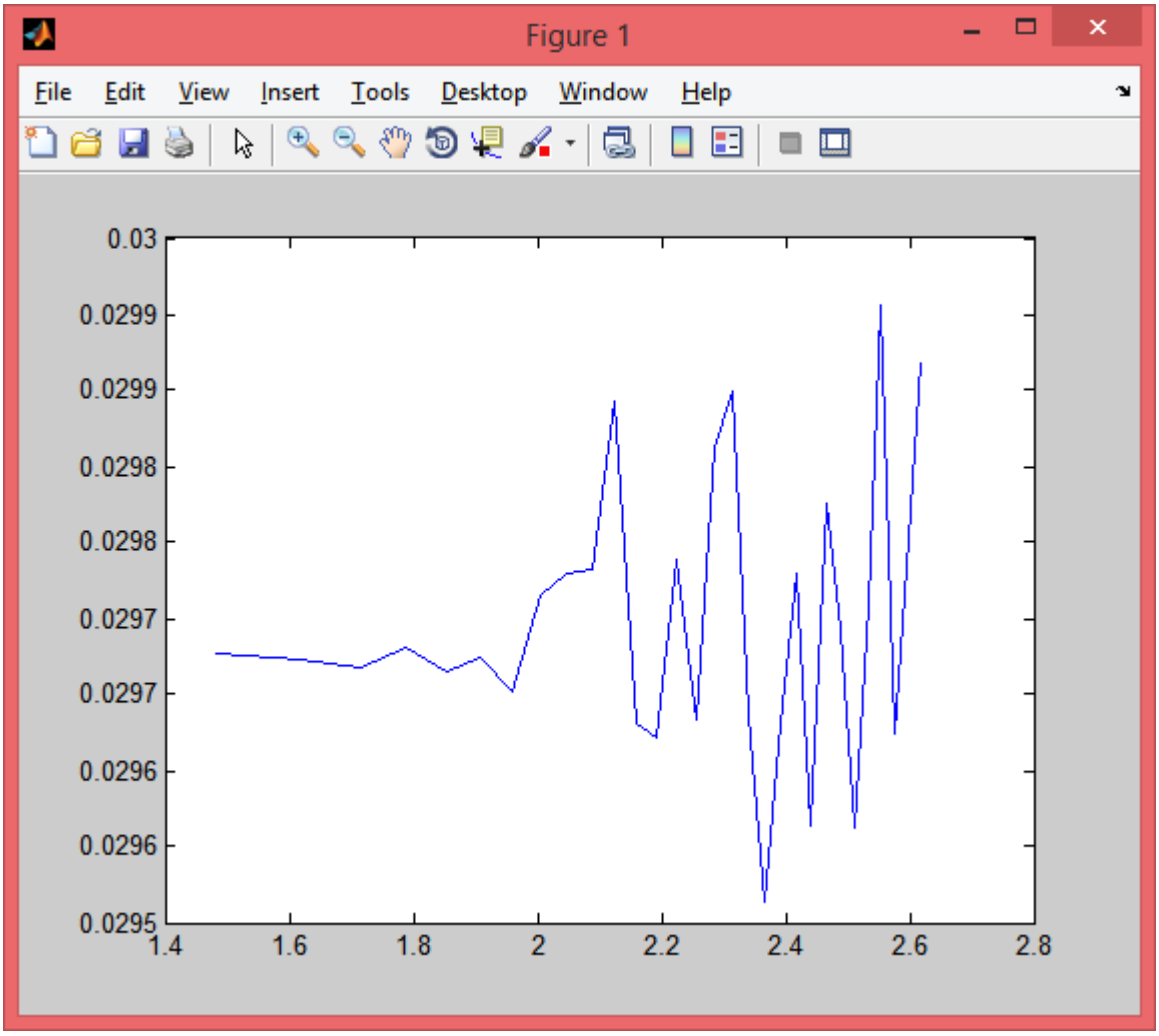
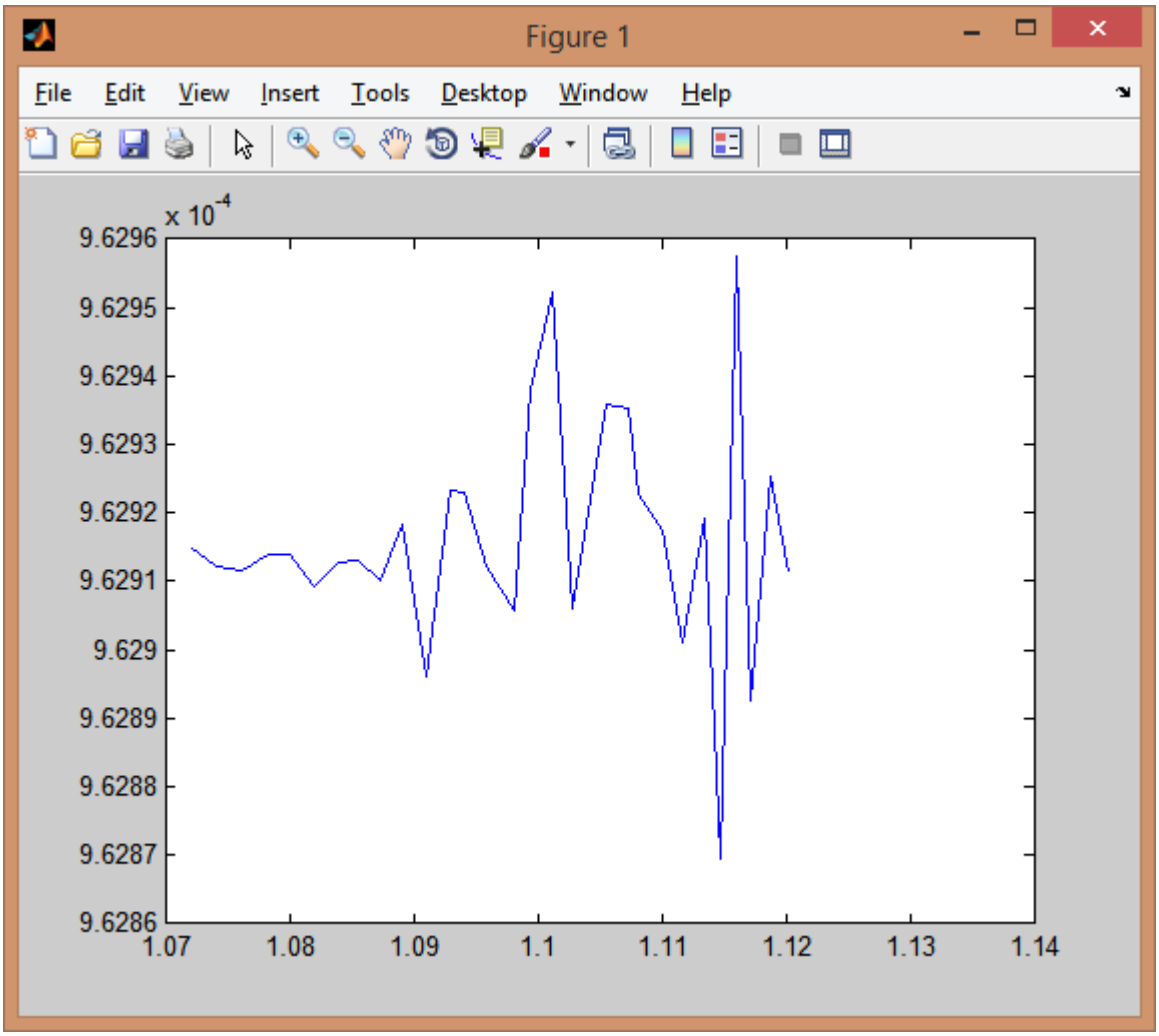


График зависимости относительной среднеквадратичной ошибки от степени сжатия при переборе StepE от 0.001 до 0,03 и step = 0.00306



Вывод

Подбирая коэффициенты step и stepE, можно регулировать степень сжатия исходного сигнала, а также качество восстановленной последовательности. При сжатии 16x (step = 0.02; stepE = 1800), исходное сообщение различимо, но с заметными помехами. При сжатии 5.5x (step = 0.02; stepE = 300), исходное сообщение мало отличимо от оригинала.