EEE 407/591 PROJECT
DUE: NOVEMBER 21, 2001

# DATA COMPRESSION USING THE FFT

INSTRUCTOR: DR. ANDREAS SPANIAS

TEAM MEMBERS:

IMTIAZ NIZAMI - 993 21 6600

HASSAN MANSOOR - 993 69 3137

Contents

**TECHNICAL BACKGROUND** ..... **4**

**DETAILS OF THE PROGRAM** ..... **5**

RETAINING THE FIRST N-COMPONENTS ..... 5
RETAINING DOMINANT N-COMPONENTS ..... 6

**RESULTS** ..... **7**

**REMARKS** ..... **19**

**APPENDIX** ..... **21**

Code to implement the method with first n-components: ..... 21
Code to implement the method with dominant n-components: ..... 23

Figures

Figure 1: Sliding rectangular window ..... 5

Figure 2: Data compression process ..... 5

Figure 3: SNR vs. percentage of components for first n-component method 13

Figure 4: SNR vs. percentage of components for first n-component method - Scaled Version 14

Figure 5: SNR vs. percentage of components for dominant n-component method 15

Figure 6: SNR vs. percentage of components for dominant n-component method - Scaled Version ..... 16

Figure 7: Comparison of the two methods for the case of N=256 17

Figure 8: Comparison of the two methods for the case of N=256 - Scaled Version 18

Tables

Table 1A: Simulation with N=64 (Rectangular window) ... 7

Table 2A: Simulation with N=128 (Rectangular window). 8

Table 3A: Simulation with N=256 (Rectangular window) 10

## INTRODUCTION

Data compression is one of the necessities of modern day. For instance, with the explosive growth of the Internet there is a growing need for audio compression, or data compression in general. One goal of such compressions is to minimize the storage space. Nowadays a 40GB hard drive can be bought within hundred dollars that makes the storage less of a problem. However, compression is greatly needed to reduce transmission bandwidth requirements, which can be achieved by data compression. Today all kind of audio/video is preferred in digital domain. Almost every computer user keeps audio files, either as MP3s, or in some other format on his/her computer's hard drive. It is very often that people upload/download music of various kinds, which requires a huge amount of bandwidth. This creates a need for better and better speech compression algorithms that reduces the size of the audio file significantly without sacrificing quality. Due to the increasing demand for better speech algorithms, several standards were developed, including MPEG, MP3, etc.

Data compression using transformations such as the DCT and the DFT are the basis for many coding standards such as JPEG, MP3 and AC-3. In this project FFT (IFFT) is used for the compression (decompression) of a speech signal. This data compression scheme is simulated using Matlab. Simulations are performed for different FFT sizes and different number of components chosen. Two different methods used for the purpose are:

- By retaining the first n-components
- By retaining dominant n-components

The SNR's (signal to noise ratios) are computed for all the simulations and used to study the behavior of the compression scheme using FFT. Also the noise introduced in the signal (for various cases) is studied both by listening to the recovered signal and by the calculated SNR's.

## TECHNICAL BACKGROUND

Fourier Transform (FT) can be very simply defined to be a mathematical technique to resolve a given signal into the sum of sines and cosines. The Fourier transform is an invaluable tool in science and engineering. The main features that make Fourier transform attractive are:

- Its symmetry and computational properties.
- Significance of time (space) vs. frequency (spectral) domain.

The Discrete Fourier Transform (DFT) is used to produce frequency analysis of discrete non-periodic signals. If we look at the equation for the Discrete Fourier Transform we will see that it is quite complicated to work out as it involves many additions and multiplications involving complex numbers. Even a simple eight-sample signal would require 49 complex multiplications and 56 complex additions to work out the DFT. At this level it is still manageable, however a realistic signal could have 1024 samples, which requires over 20,000,000 complex multiplications and additions. Obviously, this suggests that this technique becomes very time consuming with a slight increase in the number of samples.

The Fast Fourier Transform (FFT) is a discrete Fourier Transform (DFT) algorithm which reduces the number of computations from something on the order of N^2 to N*log (N). The Fast Fourier Transform greatly simplifies the computations for large values of N, where N is the number of samples in the sequence. The idea behind the FFT is the divide and conquer approach, to break up the original N point sample into two (N/2) sequences. This is because a series of smaller problems is easier to solve than one large one. The DFT requires (N-1)^2 complex multiplications and N (N-1) complex additions as opposed to the FFT's approach of breaking it down into a series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal. Two types of FFT algorithms are in use: decimation-in-time and decimation-in-frequency. The algorithm is simplified if N is chosen to be a power of 2, but it is not a requirement.

## DETAILS OF THE PROGRAM

Two main methods are implemented in the Matlab programs.

- By retaining the first n-components
- By retaining dominant n-components

### RETAINING THE FIRST N-COMPONENTS

In this method we start by reading the wave file "cleanspeech" in Matlab, and saving the speech in vector 's'. The data set (in the vector) to be compressed is then segmented into N-point segments (or frames) using a sliding window. This is done in the program by dividing the vector 's' into segments. This is shown in the figure below.

*Figure 1: Sliding rectangular window*

The FFT of each segment is taken one by one by passing it into the loop N (the number of frames) times. Thus we get the magnitude spectrum of the signal. The result of the N point FFT has (1+N/2) independent components. This is due to the symmetry property of the DFT. These (1+N/2) points are retained as they are sufficient to get back the all the information in the original signal. From this set of about half the points first 'n' points are chosen to reconstruct the original signal. In out simulation this is done for all possible n values from 1 to (1+N/2). The rest of the (1+N/2-n) points are padded with zeros. Now, before taking the IFFT, we have to give the vector of 'n' components its conjugate symmetry back. Otherwise, we will get back an imaginary signal. In order to rebuild the symmetry in the signal, the conjugate of the "first n-component" vector is taken. The first and last components in this new vector are disregarded as they are dc values which does not take part in the symmetry building before taking IFFT. The conjugate vector is flipped and added to the original "first n-component" vector. Hence, we have got the signal with symmetrical properties and we are ready to get back real values after taking the IFFT. The whole process is shown in the figure below.

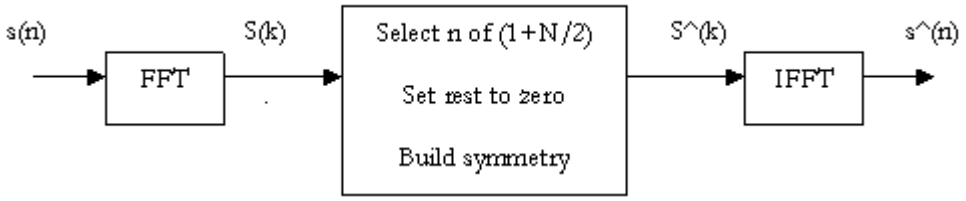


*Figure 2: Data compression process*

The Matlab code for this method is provided in the appendix.

### RETAINING DOMINANT N-COMPONENTS

This method is very similar to the one we have discussed above. The only difference is in the way we select the ‘n’ points for the signal. In the previous case we chose the first ‘n’ components and set the rest to zero. In this case we will choose the dominant n points, i.e., the points with maximum magnitude. The rest of the (1+N/2-n) points are set to zero. Special care is taken to make the chosen dominant ‘n’ points lie at the indices they previously were (in the signal with 1+N/2 components). Also, in our program we have chosen our dominant signal to be at the minimum of the indices in case two components with the same indices are encountered. In this case the dominant point at the next index will be chosen in picking the following component (in case the n points are already not exhausted). The Matlab code for this method is provided in the appendix.

## RESULTS

During the simulations we collected three sets of data, for 64, 128, and 256 point FFT. For each of the three sets 'n' (components selected) is varied from 1 to (1+N/2). The tables summarizing these results follows.

*Table 1A: Simulation with N=64 (Rectangular window)*

| n | N | Method 2 | Method1 |
|---|---|---|---|
| 1 | 64 | 2.2038 | 0.0135 |
| 2 | 64 | 3.8719 | 0.0673 |
| 3 | 64 | 5.2410 | 0.1857 |
| 4 | 64 | 6.4056 | 0.4547 |
| 5 | 64 | 7.4867 | 1.4146 |
| 6 | 64 | 8.4827 | 3.9178 |
| 7 | 64 | 9.4121 | 5.3314 |
| 8 | 64 | 10.3473 | 6.1776 |
| 9 | 64 | 11.1988 | 6.6386 |
| 10 | 64 | 11.9796 | 7.1455 |
| 11 | 64 | 12.7710 | 7.9797 |
| 12 | 64 | 13.6107 | 9.0914 |
| 13 | 64 | 14.4537 | 11.2083 |
| 14 | 64 | 15.3109 | 11.9309 |
| 15 | 64 | 16.1816 | 12.2747 |
| 16 | 64 | 17.1226 | 12.5261 |
| 17 | 64 | 18.0293 | 12.7449 |
| 18 | 64 | 18.8967 | 13.0037 |
| 19 | 64 | 19.8869 | 13.4986 |
| 20 | 64 | 20.9640 | 14.6117 |
| 21 | 64 | 21.9970 | 16.2962 |
| 22 | 64 | 23.0627 | 17.6447 |
| 23 | 64 | 24.1988 | 18.9805 |
| 24 | 64 | 25.3514 | 20.0984 |
| 25 | 64 | 26.4813 | 21.1365 |
| 26 | 64 | 27.6716 | 22.1367 |
| 27 | 64 | 28.8479 | 23.2699 |
| 28 | 64 | 30.0733 | 24.5377 |
| 29 | 64 | 31.6307 | 26.0917 |
| 30 | 64 | 33.4667 | 28.7562 |
| 31 | 64 | 36.2471 | 32.2514 |
| 32 | 64 | 40.7178 | 37.5274 |
| 33 | 64 | 305.5805 | 305.5805 |

*Table 2A: Simulation with N=128 (Rectangular window)*

| n | N | Method 2 | Method 1 |
|---|---|---|---|
| 1 | 128 | 2.3913 | 0.0032 |
| 2 | 128 | 3.7275 | 0.0108 |
| 3 | 128 | 4.7824 | 0.0370 |
| 4 | 128 | 5.6328 | 0.0556 |
| 5 | 128 | 6.3384 | 0.1102 |
| 6 | 128 | 6.9822 | 0.1977 |
| 7 | 128 | 7.5690 | 0.3129 |
| 8 | 128 | 8.1433 | 0.5590 |
| 9 | 128 | 8.6590 | 0.9857 |
| 10 | 128 | 9.1438 | 1.4231 |
| 11 | 128 | 9.6066 | 2.6956 |
| 12 | 128 | 10.0627 | 5.0373 |
| 13 | 128 | 10.5114 | 5.5290 |
| 14 | 128 | 10.9527 | 6.0540 |
| 15 | 128 | 11.3898 | 6.3331 |
| 16 | 128 | 11.8284 | 6.6376 |
| 17 | 128 | 12.2646 | 6.8445 |
| 18 | 128 | 12.7085 | 6.9992 |
| 19 | 128 | 13.1520 | 7.2733 |
| 20 | 128 | 13.5974 | 7.4969 |
| 21 | 128 | 14.0454 | 8.0543 |
| 22 | 128 | 14.4963 | 8.4776 |
| 23 | 128 | 14.9498 | 9.2602 |
| 24 | 128 | 15.3851 | 9.8516 |
| 25 | 128 | 15.7994 | 11.3802 |
| 26 | 128 | 16.2404 | 11.9881 |
| 27 | 128 | 16.6766 | 12.3644 |
| 28 | 128 | 17.0972 | 12.5158 |
| 29 | 128 | 17.5069 | 12.6674 |
| 30 | 128 | 17.9277 | 12.7595 |
| 31 | 128 | 18.3603 | 12.8306 |
| 32 | 128 | 18.8118 | 12.9309 |
| 33 | 128 | 19.2901 | 13.0164 |
| 34 | 128 | 19.7282 | 13.1321 |
| 35 | 128 | 20.2335 | 13.2215 |
| 36 | 128 | 20.7371 | 13.3550 |
| 37 | 128 | 21.2537 | 13.5309 |
| 38 | 128 | 21.7844 | 13.8250 |
| 39 | 128 | 22.3514 | 14.2770 |
| 40 | 128 | 22.8919 | 14.9846 |

| | | | |
|---|---|---|---|
| 41 | 128 | 23.4484 | 15.8839 |
| 42 | 128 | 23.9792 | 16.8314 |
| 43 | 128 | 24.5233 | 17.3377 |
| 44 | 128 | 25.0928 | 18.3420 |
| 45 | 128 | 25.6533 | 18.9407 |
| 46 | 128 | 26.2268 | 19.8718 |
| 47 | 128 | 26.8236 | 20.4231 |
| 48 | 128 | 27.4267 | 21.1078 |
| 49 | 128 | 28.0442 | 21.6144 |
| 50 | 128 | 28.6434 | 22.0653 |
| 51 | 128 | 29.2684 | 22.6249 |
| 52 | 128 | 29.9260 | 23.0933 |
| 53 | 128 | 30.6057 | 23.7241 |
| 54 | 128 | 31.2762 | 24.3549 |
| 55 | 128 | 31.9887 | 24.9883 |
| 56 | 128 | 32.7668 | 25.7346 |
| 57 | 128 | 33.6086 | 26.6174 |
| 58 | 128 | 34.5904 | 27.8736 |
| 59 | 128 | 35.7188 | 29.4067 |
| 60 | 128 | 36.8773 | 30.9742 |
| 61 | 128 | 38.5528 | 33.6740 |
| 62 | 128 | 40.2846 | 35.6910 |
| 63 | 128 | 43.1389 | 38.2483 |
| 64 | 128 | 47.9142 | 43.6936 |
| 65 | 128 | 305.6275 | 305.6275 |

*Table 3A: Simulation with N=256 (Rectangular window)*

| n | N | Method 2 | Method1 |
|---|---|---|---|
| 1 | 256 | 2.0090 | 0.0007 |
| 2 | 256 | 3.0876 | 0.0023 |
| 3 | 256 | 3.7826 | 0.0041 |
| 4 | 256 | 4.3842 | 0.0096 |
| 5 | 256 | 4.9358 | 0.0247 |
| 6 | 256 | 5.4214 | 0.0280 |
| 7 | 256 | 5.8774 | 0.0323 |
| 8 | 256 | 6.3125 | 0.0590 |
| 9 | 256 | 6.6906 | 0.0859 |
| 10 | 256 | 7.0639 | 0.0987 |
| 11 | 256 | 7.4164 | 0.1197 |
| 12 | 256 | 7.7503 | 0.2151 |
| 13 | 256 | 8.0743 | 0.2684 |
| 14 | 256 | 8.3987 | 0.3077 |
| 15 | 256 | 8.7058 | 0.4391 |
| 16 | 256 | 9.0108 | 0.5822 |
| 17 | 256 | 9.3069 | 0.7697 |
| 18 | 256 | 9.5969 | 1.0126 |
| 19 | 256 | 9.8838 | 1.3183 |
| 20 | 256 | 10.1666 | 1.4826 |
| 21 | 256 | 10.4380 | 2.1441 |
| 22 | 256 | 10.6978 | 3.7475 |
| 23 | 256 | 10.9532 | 5.0288 |
| 24 | 256 | 11.1954 | 5.3707 |
| 25 | 256 | 11.4338 | 5.6791 |
| 26 | 256 | 11.6740 | 5.8293 |
| 27 | 256 | 11.9063 | 6.1157 |
| 28 | 256 | 12.1392 | 6.2348 |
| 29 | 256 | 12.3682 | 6.3834 |
| 30 | 256 | 12.5943 | 6.5169 |
| 31 | 256 | 12.8191 | 6.6682 |
| 32 | 256 | 13.0451 | 6.7548 |
| 33 | 256 | 13.2739 | 6.8624 |
| 34 | 256 | 13.4944 | 6.9606 |
| 35 | 256 | 13.7158 | 7.0365 |
| 36 | 256 | 13.9376 | 7.1189 |
| 37 | 256 | 14.1608 | 7.2260 |
| 38 | 256 | 14.3847 | 7.3879 |
| 39 | 256 | 14.6045 | 7.4739 |
| 40 | 256 | 14.8281 | 7.5760 |

| | | | |
|---|---|---|---|
| 41 | 256 | 15.0547 | 7.8927 |
| 42 | 256 | 15.2846 | 8.1763 |
| 43 | 256 | 15.5161 | 8.3581 |
| 44 | 256 | 15.7496 | 8.6720 |
| 45 | 256 | 15.9831 | 9.2201 |
| 46 | 256 | 16.2144 | 9.4267 |
| 47 | 256 | 16.4470 | 9.7869 |
| 48 | 256 | 16.6832 | 10.4543 |
| 49 | 256 | 16.9019 | 11.3228 |
| 50 | 256 | 17.1241 | 11.8626 |
| 51 | 256 | 17.3494 | 12.2138 |
| 52 | 256 | 17.5810 | 12.4037 |
| 53 | 256 | 17.8179 | 12.5534 |
| 54 | 256 | 18.0524 | 12.6431 |
| 55 | 256 | 18.2872 | 12.6970 |
| 56 | 256 | 18.5182 | 12.7878 |
| 57 | 256 | 18.7557 | 12.8467 |
| 58 | 256 | 18.9949 | 12.8774 |
| 59 | 256 | 19.2341 | 12.9083 |
| 60 | 256 | 19.4669 | 12.9584 |
| 61 | 256 | 19.7036 | 12.9878 |
| 62 | 256 | 19.9424 | 13.0258 |
| 63 | 256 | 20.1850 | 13.0690 |
| 64 | 256 | 20.4324 | 13.1144 |
| 65 | 256 | 20.6850 | 13.1563 |
| 66 | 256 | 20.9433 | 13.1994 |
| 67 | 256 | 21.2036 | 13.2470 |
| 68 | 256 | 21.4697 | 13.2825 |
| 69 | 256 | 21.7379 | 13.3318 |
| 70 | 256 | 22.0034 | 13.3786 |
| 71 | 256 | 22.2702 | 13.4342 |
| 72 | 256 | 22.5378 | 13.5005 |
| 73 | 256 | 22.8134 | 13.5983 |
| 74 | 256 | 23.0756 | 13.6950 |
| 75 | 256 | 23.3495 | 13.7923 |
| 76 | 256 | 23.6154 | 14.0204 |
| 77 | 256 | 23.8754 | 14.2565 |
| 78 | 256 | 24.1376 | 14.6320 |
| 79 | 256 | 24.4030 | 15.0195 |
| 80 | 256 | 24.6443 | 15.3790 |
| 81 | 256 | 24.9161 | 15.7558 |
| 82 | 256 | 25.1887 | 16.5182 |
| 83 | 256 | 25.4604 | 17.0393 |

| | | | |
|---|---|---|---|
| 84 | 256 | 25.7420 | 17.3047 |
| 85 | 256 | 26.0281 | 17.6261 |
| 86 | 256 | 26.3232 | 18.0332 |
| 87 | 256 | 26.6199 | 18.6680 |
| 88 | 256 | 26.9221 | 18.8803 |
| 89 | 256 | 27.2302 | 19.2169 |
| 90 | 256 | 27.5420 | 19.6178 |
| 91 | 256 | 27.8606 | 20.0707 |
| 92 | 256 | 28.1778 | 20.5241 |
| 93 | 256 | 28.5036 | 20.8405 |
| 94 | 256 | 28.8349 | 21.0330 |
| 95 | 256 | 29.1717 | 21.3219 |
| 96 | 256 | 29.5078 | 21.7062 |
| 97 | 256 | 29.8547 | 22.0363 |
| 98 | 256 | 30.2057 | 22.1975 |
| 99 | 256 | 30.5597 | 22.4879 |
| 100 | 256 | 30.9237 | 22.7819 |
| 101 | 256 | 31.2899 | 23.0213 |
| 102 | 256 | 31.6606 | 23.2471 |
| 103 | 256 | 32.0555 | 23.4681 |
| 104 | 256 | 32.4606 | 23.8325 |
| 105 | 256 | 32.8491 | 24.0880 |
| 106 | 256 | 33.2798 | 24.4810 |
| 107 | 256 | 33.7327 | 24.7880 |
| 108 | 256 | 34.1531 | 25.2040 |
| 109 | 256 | 34.5739 | 25.4550 |
| 110 | 256 | 35.0137 | 25.7486 |
| 111 | 256 | 35.4791 | 26.2008 |
| 112 | 256 | 35.9586 | 26.5935 |
| 113 | 256 | 36.4592 | 27.0083 |
| 114 | 256 | 36.9715 | 27.4299 |
| 115 | 256 | 37.4865 | 28.0088 |
| 116 | 256 | 38.0395 | 29.1043 |
| 117 | 256 | 38.6156 | 29.8402 |
| 118 | 256 | 39.2317 | 31.0556 |
| 119 | 256 | 39.8956 | 31.8653 |
| 120 | 256 | 40.5950 | 33.0205 |
| 121 | 256 | 41.3442 | 34.9470 |
| 122 | 256 | 42.2583 | 36.4348 |
| 123 | 256 | 43.3169 | 37.4570 |
| 124 | 256 | 44.5666 | 38.7509 |
| 125 | 256 | 46.1363 | 40.1556 |
| 126 | 256 | 48.1689 | 42.2606 |
| 127 | 256 | 50.8224 | 45.3212 |

| | | | |
|---|---|---|---|
| 128 | 256 | 54.7673 | 50.2150 |
| 129 | 256 | 304.3307 | 304.3307 |

The data in the above tables was also plotted in three different ways. In the following figure the SNR curves for 64, 128, and 256 point FFT's are plotted on the same graph. The graph is for the method in which first 'n' components are selected is given in figure below.

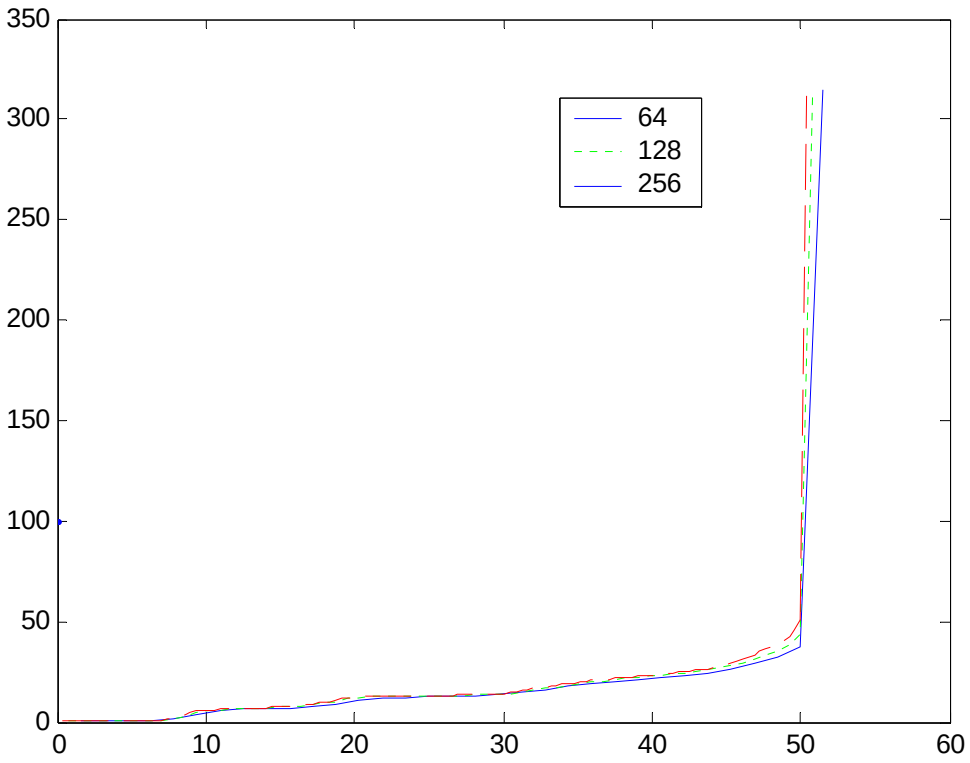


*Figure 3: SNR vs. percentage of components for first n-component method*

A second version of the same graph with scaled y-axis is given below.

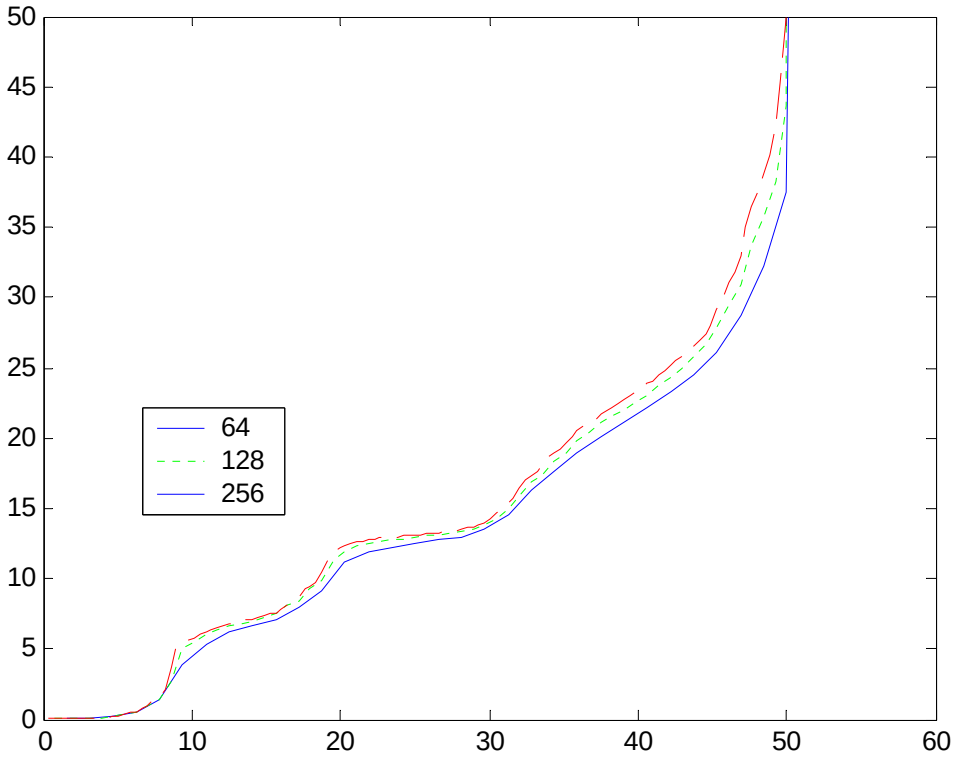


*Figure 4: SNR vs. percentage of components for first n-component method - Scaled Version*

In the following two figures the SNR curves for 64, 128, and 256 point FFT's are plotted on the same graph. The graphs are for the method in which dominant 'n' components are selected. The plot in second figure is a scaled version of the first to visualize the plotin a better way.

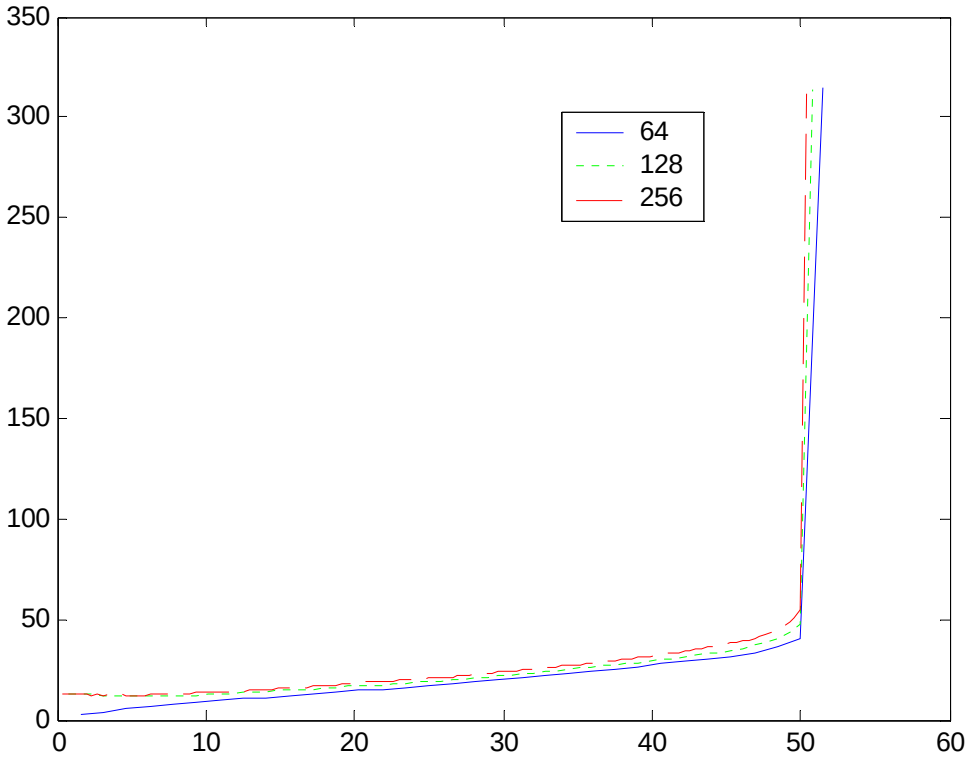


*Figure 5: SNR vs. percentage of components for dominant n-component method*

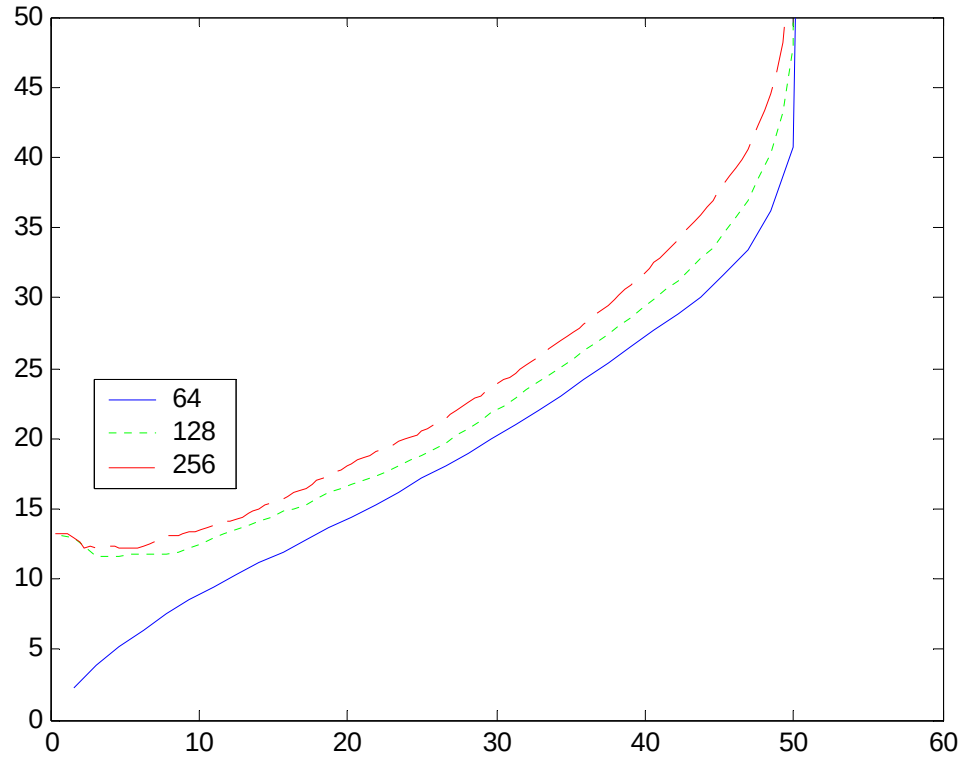


*Figure 6: SNR vs. percentage of components for dominant n-component method - Scaled Version*

In the following two plots SNR's (for the 256 point FFT) for first 'n' and dominant 'n' components are compared. The second plot is the scaled version of first.

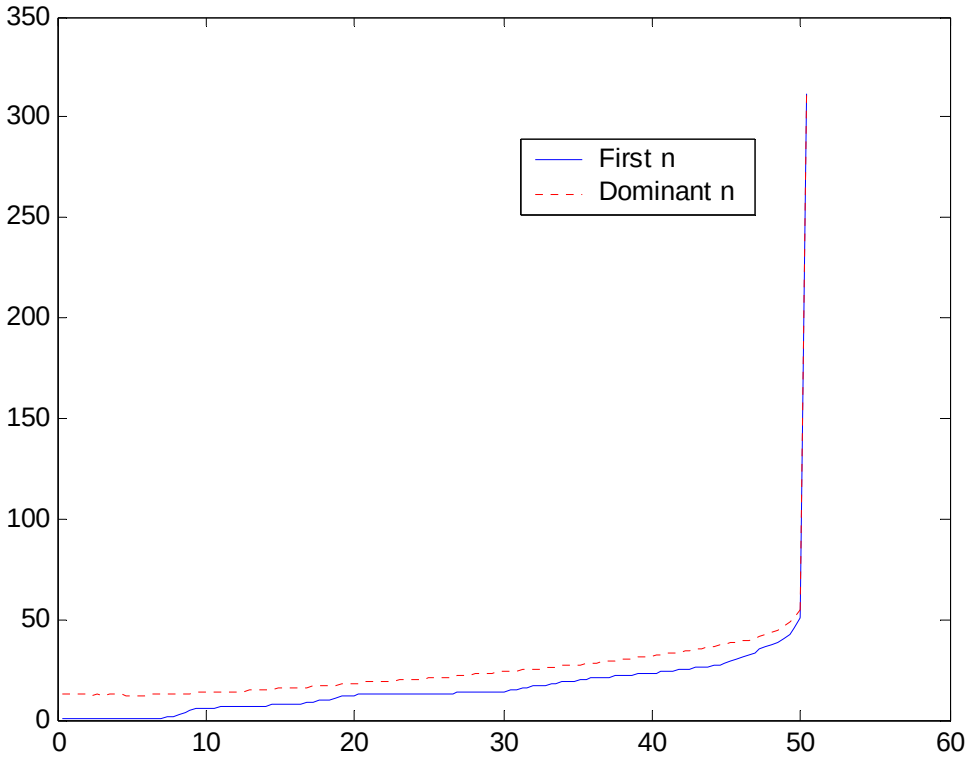


*Figure 7: Comparison of the two methods for the case of N=256*

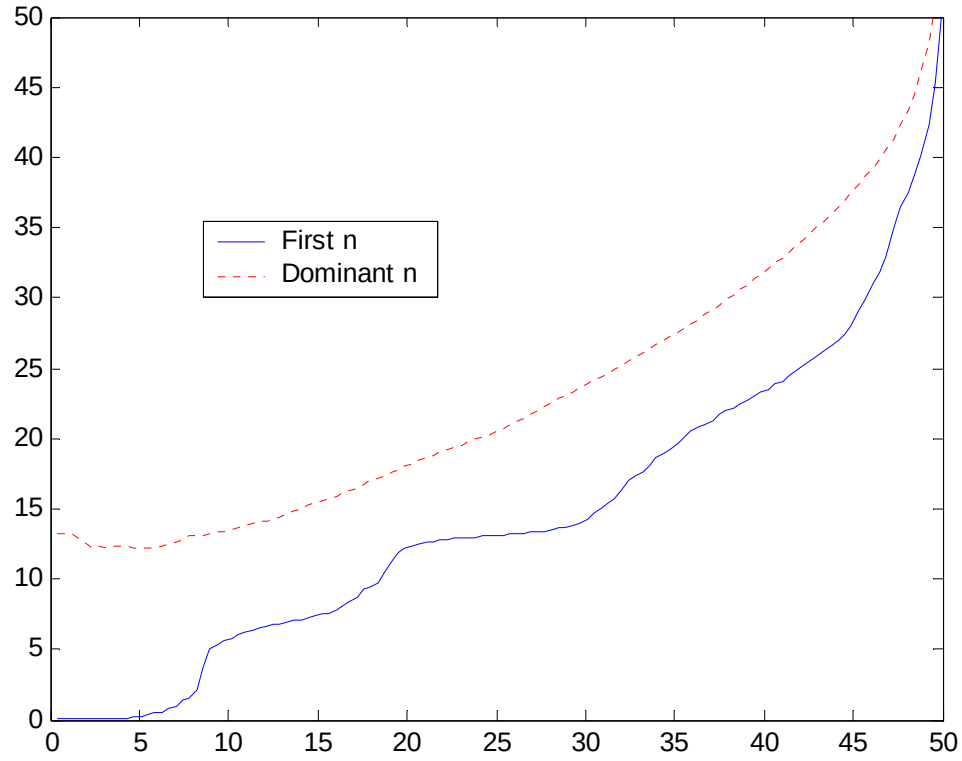


*Figure 8: Comparison of the two methods for the case of N=256 - Scaled Version*

## REMARKS

In this section we have answered the analysis questions.

1. What is the effect of the parameter n (N=fixed) on the SNR? Explain.

N corresponds to the number of components of the signal chosen. This means that by increasing n value we are increasing the resolution of the signal, as we get closer to the original signal. This suggests that we should have an improvement in quality of sound as we increase n, both in the case of first n and dominant n methods. This is indeed the case and is supported by the audio signal created by the process. A signal with increased n gives a better quality audio signal. This can also be seen from the SNR values. The SNR values increase as we increase the n value from 1 to (1+ N/2). The drawback of choosing large n is that the size of the file starts getting bigger as we increase n.

2. What is the effect of N (n/N=fixed) on the SNR? Explain.

N in our program refers to the size of FFT used. This is in fact also the length of the window used for the simulation of a particular N sized FFT. We have done simulations for three values of N, namely 64, 128, and 256. As we increase the value N, we increase the resolution of our signal, by increasing the number of samples. This will increase the quality of the audio signal. In our case the quality of the audio signal simultaneously depends on N and n values. So if N value is increased but n value is chosen to be very low, the overall signal will not be a high quality signal. Choosing N to be 64, our best quality compressed signal will be composed of 33 non-zero components. From best quality we mean that n is chosen to be at its peak value. In the case of N=128, our best quality signal will be composed of 65 non-zero components. In the case of N being 256, the best quality compressed signal will have 129 non-zero components.

3. Explain the differences in the results obtained with method 1 as opposed to method 2.

Using first n component method usually provides a relatively poor result compared to the results provided by the method of choosing n-dominant components. This can be seen from the SNR plots. This should also be our intuitive answer, as by choosing the n dominant points we are in fact taking account of a wider range of values. Since these values are picked so that they have high magnitudes, the quality if audio is relatively better. Choosing the first n components might provide us with non-useful information cutting out the important part of the signal. Using the dominant n-component method we have a smaller chance of getting into such situations. We also note that the SNR values turn out to be the same for a particular N and maximum possible n. This should indeed be the case as when we choose maximum possible n, the components from n dominant and first n methods should be identical.

4. In order to implement an actual data compression scheme then the retained transform components must be encoded in binary format. Assuming that n and N are the same for method 1 and method 2, which method will produce the lowest bit-rate (bits/second)?

The lowest bit-rate will be provided by the method in which we choose the first n-components. This is because we have higher magnitudes in the case of dominant n-component method. On average each component of "n dominant component" method will have greater magnitude than the component of "first n-component" method. This suggests that a greater bit-rate is needed for dominant n-component method. In changing the signal components to binary format we will have lower bit rates for "first n-component" method. For example, we can we can represent '1' as '01' in binary, but to represent 8 we have to have at least 3 bits, that is, '111'.

5. Try to listen to the processed files using the MATLAB sound command and give some comments regarding the subjective quality of the processed record.

For low values of n, keeping the N constant, the quality of voice obtained with n dominant component method is much better. The actual voice (information bearing) part of the signal is clearer in this case. In the case of first n-component method it is difficult to distinguish between noise and voice. It seemed that the voice signal obtained by the first n, and dominant n components can be compared to AM and FM radio respectively. When choosing n to be in the mid-range values, the "first n-component" method produced a voice signal which has more noise than the other case, but it sounded more smooth that the other case. This is because we found sort of "clipping" in the signal produced by choosing dominant components. At high values of n the voice signals generated by using the two different methods sounded almost identical.

This has been a wonderful learning experience. Specially listening to the compressed file and figuring out what effects does the two methods on the quality of sound was particularly interesting. We learnt how to do some serious work in Matlab. We learnt and were amazed by the possibilities Matlab programming provides us with (in order to do mathematical operations).

I think that providing students with such examples of code and letting them play around with the code can be useful for the students. A lab can be made where this or a similar kind of code example can be given to the students. It will be useful and fun to answer questions similar to the ones asked in this project. I think that it can be a very good learning experience because it is not something that is purely mathematical, but the students will in fact be able to experience a worldly example or application of DSP.

Same amount of time and effort has been put into this project by the two team members. We both came up with a code for first n-component method separately. The only problem was that one of us was not getting the correct result at the value when $n=1+N/2$. We figured the dominant n-component method by sitting together and discussing what can be changed in the first code to make it work for the dominant case. Introduction, technical background, and the data tables are written and prepared by Hassan Mansoor. The details of the program, plots, and remarks section is prepared by Imtiaz Nizami.

## APPENDIX

**Code to implement the method with first n-components:**

```
clear,clc;
hold off
tic
for fftPoints= 1 : 3
   switch fftPoints
   case 1
      N=64;
      M=64;
   case 2
      N=128;
      M=128;
   case 3
      N=256;
      M=256;
   end

   s=wavread('cleanspeech');              % Load wave file into matlab as vector
   L=length(s);                           % Scalar representing length of wave-file vector
   S=zeros(N,1);

   S1=zeros(1+N/2,1);
   S2=zeros(1+N/2,1);
   S3=zeros(1,N);
```

```
S4=zeros(1,N);
S5=zeros(L,1);

for i0 = 1:1+N/2
    N1=1:i0;

    for i = 1:K
        S1=zeros(1+N/2,1);
        k=(1:M)+((i-1)*M);
        k=min(k):(min(max(k),L));
        S=fft(s(k),N);
        S1(N1)=S(N1);
        S2=flipud(conj(S1));
        S3=[S1;S2(2:N/2)];
        S4=ifft(S3,N);
        S5(k)=S4;
    end

    S6=real(S5);
    index=1:L;
    S11=s(index);
    S12=S6(index);
    sum(S11.^2);
    sum(S11-S12).^2;
    SNR(i0)=10*log10(sum(S11.^2)./sum((S11-S12).^2));
end
```

```
    switch fftPoints
    case 1
        SNR_64_1=SNR;
    case 2
        SNR_128_1=SNR;
    case 3
        SNR_256_1=SNR;
    end
end

n1=1:33;
n2=1:65;
n3=1:129;
plot(n1*100/64,SNR_64_1)
hold on;
plot(n2*100/128,SNR_128_1,'g--')
plot(n3*100/256,SNR_256_1,'r-')
toc
```

**Code to implement the method with dominant n-components:**

```
clear,clc;
hold off
tic

for fftPoints= 1 : 3
```

```
switch fftPoints
case 1
   N=64;
   M=64;
case 2
   N=128;
   M=128;
case 3
   N=256;
   M=256;
end

s=wavread('cleanspeech');              % Load wave file into matlab as vector
L=length(s);                           % Scalar representing length of wave-file vector
K=round(L/M);                          % Total number of frames
S_old=zeros(N,1);
S=zeros(1+N/2,1);
abs_S=zeros(1+N/2,1);

S1=zeros(1+N/2,1);
S2=zeros(1+N/2,1);
S3=zeros(1,N);
S4=zeros(1,N);
S5=zeros(L,1);

for i0 = 1:1+N/2
```

```
for i = 1:K
   k=(1:M)+((i-1)*M);
   k=min(k):(min(max(k),L));
   S_old=fft(s(k),N);
   S=S_old(1:1+N/2);
   abs_S=abs(S);
   min_abs_S=0;

   S1=zeros(1+N/2,1);

   for count1 = 1:i0
      max_index=find(abs_S==max(abs_S));
      index_value(count1)=min(max_index);
      abs_S(min(max_index))=min_abs_S;
   end

   S1(index_value)=S(index_value);
   S2=flipud(conj(S1));
   S3=[S1;S2(2:N/2)];
   S4=ifft(S3,N);
   S5(k)=S4;
end

S6=real(S5);
index=1:L;
S11=s(index);
```

```
        S12=S6(index);
        sum(S11.^2);
        sum(S11-S12).^2;
        SNR(i0)=10*log10(sum(S11.^2)./sum((S11-S12).^2));
    end

    switch fftPoints
    case 1
        SNR_64_2=SNR;
    case 2
        SNR_128_2=SNR;
    case 3
        SNR_256_2=SNR;
    end
end
n1=1:33;
n2=1:65;
n3=1:129;
plot(n1*100/64,SNR_64_2)
hold on;
plot(n2*100/128,SNR_128_2,'g--')
plot(n3*100/256,SNR_256_2,'r-')
toc
```