Visvesvaraya Technological University, Belagavi.

PROJECT REPORT

on

“Low Bit-Rate Speech Codec”

Project Report submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Engineering

in

Electronics and Communication Engineering

For the academic year 2019-20

Submitted by

USN	Name
1CR16EC009	AKANKSHA C HOWALE
1CR16EC096	NETHRAVATHI G

Under the guidance of

Internal
Mrs. Sutapa Sarkar
Assistant Professor
Department of ECE
CMRIT, Bengaluru

External
Dr. R. Muralishankar
Professor
School of Engineering and Technology
CMR University, Bengaluru

Department of Electronics and Communication Engineering

CMR Institute of Technology, Bengaluru – 560 037

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CELEBRATING 25 YEARS
CMR INSTITUTE OF TECHNOLOGY

CERTIFICATE

This is to Certify that the dissertation work **"Low Bit-Rate Speech Codec"** carried out by **Akanksha C Howale**, USN: 1CR16EC009, **Nethravathi G,** 1CR16EC096, bonafide students of **CMRIT** in partial fulfillment for the award of **Bachelor of Engineering** in **Electronics and Communication Engineering** of the **Visvesvaraya Technological University, Belagavi,** during the academic year **2019-20**. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said degree.

Signature of Guide	Signature of HOD	Signature of Principal
________________	________________	________________
Mrs. Sutapa Sarkar,	Dr. R. Elumalai	Dr. Sanjay Jain
Assistant Professor,	Head of the Department,	Principal,
Dept. of ECE.,	Dept. of ECE.,	CMRIT,
CMRIT,Bengaluru.	CMRIT,Bengaluru.	Bengaluru.

External Viva

Name of Examiners	Signature & date
1.	
2.	

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose consistent guidance and encouragement crowned our efforts with success.

We consider it as our privilege to express the gratitude to all those who guided in the completion of the project.

We express our gratitude to Principal, **Dr. Sanjay Jain,** for having provided us the golden opportunity to undertake this project work in their esteemed organization.

We sincerely thank Dr. R. Elumalai**,** HOD**,** Department of Electronics and Communication Engineering, CMR Institute of Technology for the immense support given to us.

We express our gratitude to our project guide Dr. R. Muralishankar, Professor and Mrs. Sutapa Sarkar, Assistant Professor, for their support, guidance and suggestions throughout the project work.

Last but not the least, heartful thanks to our parents and friends for their support.

Above all, We thank the Lord Almighty for His grace on us to succeed in this endeavor.

Akanksha C Howale	Nethravathi G
USN: 1CR16EC009	USN: 1CR16EC096

ABSTRACT

Speech signals have a well-defined structure in frequency domain. We utilize this feature of speech for reconstruction, i.e. if we have a sufficient portion of speech signal in a frequency domain, we can reconstruct the remaining parts of the speech with reasonable accuracy using only magnitude spectrum. Military tanks are subjected to extreme conditions of the warzones and the establishment and maintenance of communication between military tanks and the ground station or between tanks is a difficult task. Resources such as bandwidth available for communication pose a severe restriction on the communication systems used. Since there is a transition from analog communication systems to their digital counter parts in these tanks, there is a need for designing a very low bit-rate speech codec with acceptable fidelity and intelligibility. In our project, total three steps of compression happens which will give us a low Bit-Rate speech codec and which is also secured because we use codebooks that will be encrypted.

Hence, we achieve a secure communication at low bit-rate.

Table of Contents

List of Figures

List of Tables

Chapter1

INTRODUCTION

Speech coding containing speech is an application of digital audio signals data compression. Speech coding uses speech-specific parameter estimation to model the speech signal using audio signal processing techniques, combined with generic data compression algorithms to represent the resulting modeled parameters in a compact bitstream.

In order to efficiently store and transmit speech signals, speech codecs create a minimally redundant representation of the input which is then decoded with the simplest possible perceptual quality at the receiver.

Military tanks are subjected to extreme conditions in the war zones and the establishment and maintenance of communication between military tanks and the ground station or between tanks is a difficult task. Resources such as bandwidth available for communication pose a severe restriction on the communication systems used. Since there is a transition from analog communication systems to their digital counter parts in these tanks, there is a need for designing a very low bit-rate speech codec with acceptable fidelity and intelligibility.

Speech signals have a well-defined structure in frequency domain. We utilize this feature of speech for reconstruction, i.e. if we have a sufficient portion of speech signal in a frequency domain, we can reconstruct the remaining parts of the speech with reasonable accuracy using only magnitude spectrum.

We exploit the limitations of human hearing and perception that it cannot resolve closely spaced frequency components. These frequency components are characterized by both magnitude and phase. Therefore, an intelligible speech signal can be reconstructed back from the magnitude spectrum. We convert the time-domain speech signal into frequency domain speech using short time Fourier transform.

Hence, we achieve a secure communication at low bit-rate with encrypted data.

Chapter 2

LITERATURE SURVEY

In [1], the algorithm is computationally simple and is obtained by minimizing the mean squared error between the STFT of the estimated signal and the modified STFT. Using this algorithm, we also develop an iterative algorithm to estimate a signal from its modified STFT magnitude.

The iterative algorithm is shown to decrease, in each iteration, the mean squared error between the STFT magnitude of the estimated signal and the modified STFT magnitude. The major computation involved in the iterative algorithm is the discrete Fourier transform (DFT) computation, and the algorithm appears to be real-time implementable with current hardware technology.

The algorithm developed in [1] has been applied to the time-scale modification of speech. The resulting system generates very high-quality speech, and appears to be better in performance than any existing method.

In [2], the authors propose two variations on the G&L algorithm, the “real-time iterative spectrogram inversion” (RTISI) and its improved version, the “RTISI with look ahead” (RTISI-LA), which can perform real-time spectrogram inversion, and does so faster (i.e., in fewer iterations) than the original offline algorithm.

An algorithm for estimating signals from short-time magnitude spectra is introduced offering a significant improvement in quality and efficiency over current methods. The key issue is how to invert a sequence of overlapping magnitude spectra (a “spectrogram”) containing no phase information to generate a real-valued signal free of audible artifacts.

Also important is that the algorithm performs in real-time, both structurally and computationally. In the context of spectrogram inversion, structurally real-time means that the audio signal at any given point in time only depends on transform frames at local or prior points in time. Computationally, real-time means that the algorithm is efficient enough to run in less time than the reconstructed audio takes to play on the available hardware.

The spectrogram inversion algorithm is parameterized to allow tradeoffs between computational demands and the quality of signal reconstruction. The algorithm is applied to audio time-scale and pitch modification and compared to classical algorithms for these takes on a variety of signal types including both monophonic and polyphonic audio signals such as speech and music.

The RTISI-LA method at eight transform iterations per frame generally approaches or exceeds the performance of the classic G&L algorithm with 80 iterations per frame in terms of the SER. The reduction in computation and consistently high quality across a wide variety of signal types makes RTISI-LA an appropriate choice for many signal processing applications.

Drawback is Even though it is giving best performance with less iterations comparable with G&L algorithm, but still it is not a real time algorithm.

The common problem of the iterative state-of-the-art algorithms is that they require many relatively expensive iterations in order to produce acceptable results.

In [3], the focus of a continuous speech recognition process is to match an input signal with a set of words or sentences according to some optimality criteria. The first step of this process is parameterization, whose major task is data reduction by converting the input signal into parameters while preserving virtually all of the speech signal information dealing with the text message.

This contribution presents a detailed analysis of a widely used set of parameters, the Mel frequency cepstral coefficients (MFCC’s), and suggests a new parameterization approach taking into account the whole energy zone in the spectrum. Results obtained with the proposed new coefficients give a confidence interval about their use in a large-vocabulary speaker-independent continuous-speech recognition system.

To communicate with each other, humans generally use language that is based on a finite set of articulated sounds. The meaning of the transmitted message depends on the organization of the elements in this set; these elementary sounds are usually called phonemes. Many aspects of language have been studied over the years.

The basic interest seems to be on the acoustic level, interest that is aimed at two particular applications: automatic speech synthesis and recognition. In the case of synthesis, the research emphasizes the identification of phoneme characteristics in the frequency domain for a better representation of the system that reproduces them. In the case of recognition, the priority is to extract from the acoustic signal enough information such that the recognition of phonemes, words or sentences that contain these phonemes can be possible.

In [3], their attention is devoted to this last application, that is, extracting from the input signal the acoustic information of the word or the sentence pronounced by the speaker. This operation often allows us to obtain a set of parameters or coefficients fewer in number than the initial input samples.

Then the conclusion is that the filtering process occurring in the evaluation of the MFCC does not alter the initial frequency resolution obtained after the fast Fourier transform (FFT).

Common parameters used in many recognition systems are LPC (linear predictive coding) coefficients, and Mel frequency cepstral coefficients (MFCC’s). Some recent works show how the variability of the statistical components of LPC can be reduced using bandpass filtering others are focused on finding an optimal linear transformation of Mel-warped short-time discrete Fourier transform (DFT) information to minimize the errors occurring at the back-end classifier.

In [4], Quantization, the process of approximating continuous-amplitude signals by digital (discrete amplitude) signals, is an important aspect of data compression or coding, the field concerned with the reduction of the number of bits necessary to transmit or store analog data, subject to a distortion or fidelity criterion.

The independent quantization of each signal value or parameter is termed scalar quantization, while the joint quantization of a block of parameters are termed block or vector quantization. This tutorial review presents the basic concepts employed in vector quantization and gives a realistic assessment of its benefits and costs when compared to scalar quantization.

Vector quantization is presented as a process of redundancy removal that makes effective use of four interrelated properties of vector parameters: linear dependency (correlation), nonlinear dependency, shape of the probability density function (pdf), and vector dimensionality itself. In contrast, scalar quantization can utilize effectively only linear dependency and pdf shape.

The basic concepts are illustrated by means of simple examples and the theoretical limits of vector quantizer performance are reviewed, based on results from rate-distortion theory. Practical issues relating to quantizer design, implementation, and performance in actual applications are explored. While many of the methods presented are quite general and can be used for the coding of arbitrary signals, this paper focuses primarily on the coding of speech signals and parameters.

The main purpose of [4], is to present the reader with information that can be used in making a realistic assessment of the benefits and costs of vector quantization relative to scalar quantization, especially in speech coding applications. The emphasis is on the exposition of basic principles rather than the elaboration of various techniques and their variations for which references to the literature are provided.

Vector quantization is presented as a process of redundancy removal that makes effective use of four interrelated properties of vector parameters: linear dependency (correlation), nonlinear dependency, shape of the probability density function (pdf), and vector dimensionality itself.

We shall see that linear dependency and pdf shape can be employed quite effectively with scalar quantization while the other two properties cannot. Nonlinear dependency plays a significant role in the quantization of speech spectral parameters, while dimensionality is important for waveform quantization. Because of the relatively large cost of vector quantization (generally exponential in the number of dimensions and the number of bits per dimension), given today’s computation and storage technologies, the major benefits of vector quantization are realized largely at transmission rates of about 1 bit per parameter or less, which is exactly the range where the performance of scalar quantizers degrades sharply. While the concepts presented are quite general, we shall focus in this paper on the low-rate coding of speech (below 8 kbits/s) as an application.

Chapter 3

SOFTWARE

3.1 MATLAB

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research. MATLAB has many advantages compared to conventional computer languages (e.g.,C, FORTRAN) for solving technical problems.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries world wide. It has powerful built-in routines that enable a very wide variety of computations.

It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

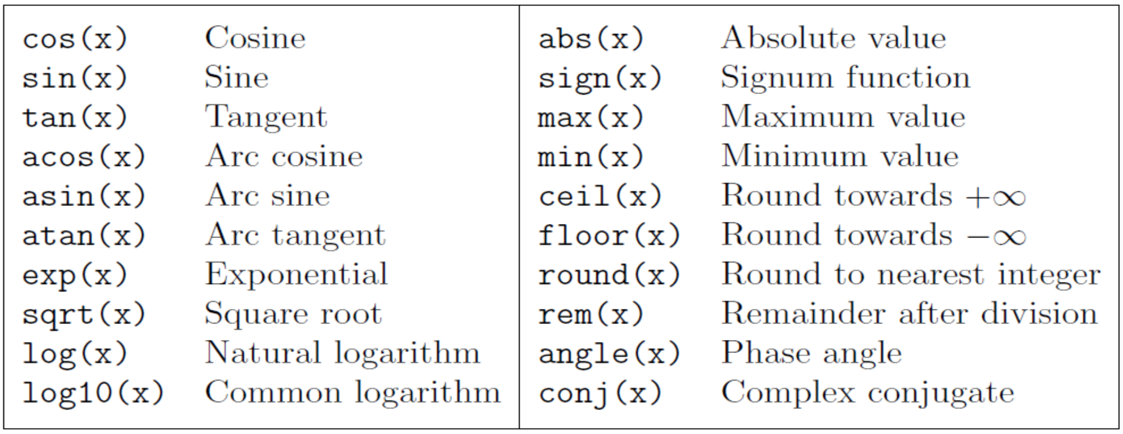
3.1.1 Mathematical functions:

MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions. Typing help elfun and help spec fun calls up full lists of elementary and special functions respectively.

There is a long list of mathematical functions that are built into MATLAB. These functions are called built-ins. Many standard mathematical functions, such as sin(x), cos(x), tan(x), ex, ln(x), are evaluated by the functions sin, cos, tan, exp, and log respectively in MATLAB.

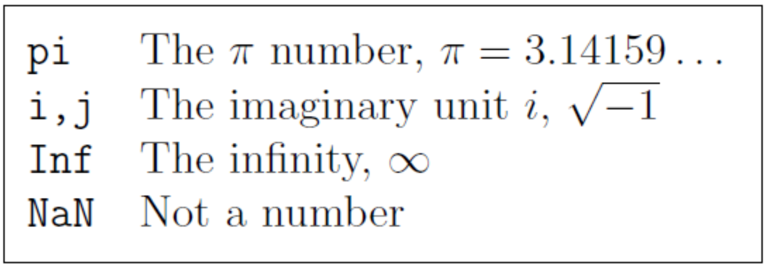
Table 3.1 lists some commonly used functions, where variables x and y can be numbers, vectors, or matrices.

In addition to the elementary functions, MATLAB includes a number of predefined constant values. A list of the most common values is given in Table 3.2.



`cos(x)`	Cosine	`abs(x)`	Absolute value
`sin(x)`	Sine	`sign(x)`	Signum function
`tan(x)`	Tangent	`max(x)`	Maximum value
`acos(x)`	Arc cosine	`min(x)`	Minimum value
`asin(x)`	Arc sine	`ceil(x)`	Round towards $+\infty$
`atan(x)`	Arc tangent	`floor(x)`	Round towards $-\infty$
`exp(x)`	Exponential	`round(x)`	Round to nearest integer
`sqrt(x)`	Square root	`rem(x)`	Remainder after division
`log(x)`	Natural logarithm	`angle(x)`	Phase angle
`log10(x)`	Common logarithm	`conj(x)`	Complex conjugate

Table 3.1 Elementary functions



`pi`	The π number, $\pi = 3.14159\ldots$
`i,j`	The imaginary unit i, $\sqrt{-1}$
`Inf`	The infinity, ∞
`NaN`	Not a number

Table 3.2 predefined constant values

3.1.2 Basic plotting

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible. Trying to understand mathematical equations with graphics is an enjoyable and very efficient way of learning mathematics. Being able to plot mathematical functions and data freely is the most important step, and this section is written to assist you to do just that.

Creating basic plot:

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of x-coordinates, x = (x1; : : : ; xN), and a vector of y-coordinates, y = (y1; : : : ; yN), locate the points (xi; yi), with i = 1; 2; : : : ; n and then join them by straight lines. You need to prepare x and y in an identical array form; namely, x and y are both row arrays or column arrays of the same length.

The MATLAB command to plot a graph is plot(x,y). The vectors x = (1; 2; 3; 4; 5; 6)and y = (3; -1; 2; 4; 5; 1) produce the picture shown in Figure 4.1.

```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```

Note: The plot functions have different forms depending on the input arguments. If y is a vector plot(y)produces a piecewise linear graph of the elements of y versus the index of the elements of y. If we specify two vectors, as mentioned above, plot(x, y) produces a graph of y versus x.

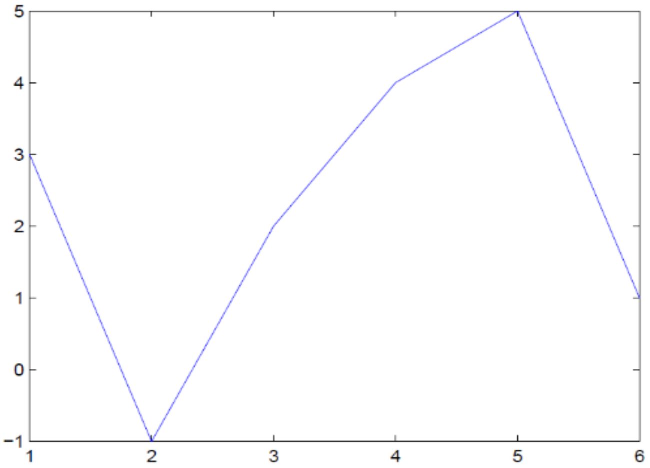


Figure 3.1.2 : Plot for vectors x and y

3.1.3 Matrix generation

Matrices are fundamental to MATLAB. Therefore, we need to become familiar with matrix generation and manipulation. Matrices can be generated in several ways.

Entering a vector:

A vector is a special case of a matrix. The purpose of this section is to show how to create vectors and matrices in MATLAB. As discussed earlier, an array of dimension 1Xn is called a row vector, whereas an array of dimension mX1 is called a column vector. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, v, type

```
>> v = [1 4 7 10 13]
              v =
                   1 4 7 10 13
```

Column vectors are created in a similar way, however, semicolon (;) must separate the components of a column vector,

```
>> w = [1;4;7;10;13]
             w =
                   1
                   4
                   7
                   10
                   13
```

On the other hand, a row vector is converted to a column vector using the transpose operator. The transpose operation is denoted by an apostrophe or a single quote (').

```
>> w = v'
                    w =
                         1
                         4
                         7
                         10
                         13
```

Thus, v (1) is the first element of vector v, v (2) its second element, and so forth. Furthermore, to access blocks of elements, we use MATLAB's colon notation (:). For example, to access the first three elements of v, we write,

```
>> v (1:3)
ans =
                       1 4 7
```

Or, all elements from the third through the last elements,

```
>> v (3, end)
ans =
                    7 10 13
```

where end signifies the last element in the vector. If v is a vector, writing

```
>> v (:)
```

produces a column vector, whereas writing

```
>> v (1: end)
```

produces a row vector.

Entering a matrix:

A matrix is an array of numbers. To type a matrix into MATLAB you must

- begin with a square bracket, [
- separate elements in a row with spaces or commas (,)
- use a semicolon (;) to separate rows
- end the matrix with another square bracket,].

Here is a typical example. To enter a matrix A, such as,

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

type,

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

MATLAB then displays the 3 X 3 matrix as follows,

```
A =
1 2 3
4 5 6
7 8 9
```

Note that the use of semicolons (;) here is different from their use mentioned earlier to suppress output or to write multiple commands in a single line.

Once we have entered the matrix, it is automatically stored and remembered in the Workspace. We can refer to it simply as matrix A. We can then view a particular element in a matrix by specifying its location. We write,

```
>> A(2,1)
ans =
4
```

A(2,1) is an element located in the second row and first column. Its value is 4.

3.2 Simulink in MATLAB

Simulink is a graphical extension to MATLAB for modeling and simulation of systems. One of the main advantages of Simulink is the ability to model a nonlinear system, which a transfer function is unable to do. Another advantage of Simulink is the ability to take on initial conditions. When a transfer function is built, the initial conditions are assumed to be zero.

In Simulink, systems are drawn on screen as block diagrams. Many elements of block diagrams are available, such as transfer functions, summing junctions, etc., as well as virtual input and output devices such as function generators and oscilloscopes. Simulink is integrated with MATLAB and data can be easily transferred between the programs.

In these tutorials, we will apply Simulink to the examples from the MATLAB tutorials to model the systems, build controllers, and simulate the systems. Simulink is supported on UNIX, Macintosh, and Windows environments; and is included in the student version of MATLAB for personal computers. For more information on Simulink, please visit the Math works link at the top of the page.

The idea behind these tutorials is that you can view them in one window while running Simulink in another window. System model files can be downloaded from the tutorials and opened in Simulink. You will modify and extend these systems while learning to use Simulink for system modeling, control, and simulation. Do not confuse the windows, icons, and menus in the tutorials for your actual Simulink windows. Most images in these tutorials are not live - they simply display what you should see in your own Simulink windows. All Simulink operations should be done in your Simulink windows.

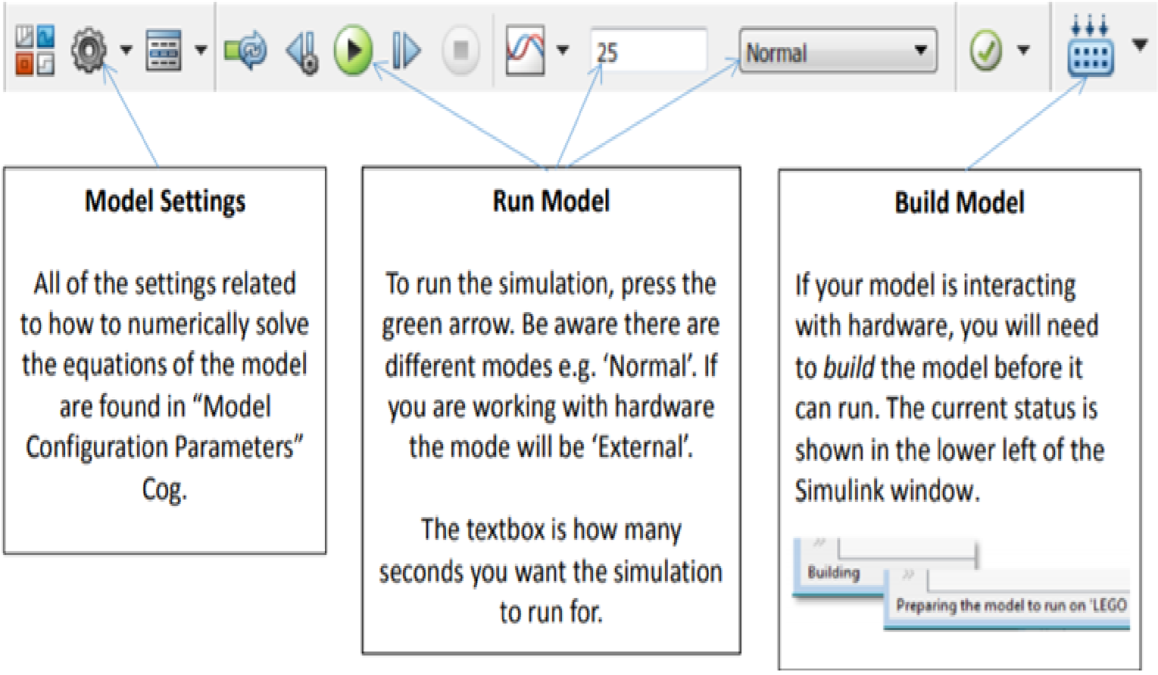
Models contain blocks, signals and annotation on a background:

- **Blocks** are mathematical functions; they can have varying numbers of inputs and outputs.
- **Signals** are lines connecting blocks, transferring values between them. Signals are different data types, for example numbers, vectors or matrices. Signals can be labelled.

- **Annotations** of text or images can be added to the model, and while not used in the calculations they can make it easier for others to understand design decisions in the model.

The Simulink toolbar

Above the main canvas of a Simulink model, is the toolbar



Model Settings

All of the settings related to how to numerically solve the equations of the model are found in "Model Configuration Parameters" Cog.

Run Model

To run the simulation, press the green arrow. Be aware there are different modes e.g. 'Normal'. If you are working with hardware the mode will be 'External'.

The textbox is how many seconds you want the simulation to run for.

Build Model

If your model is interacting with hardware, you will need to *build* the model before it can run. The current status is shown in the lower left of the Simulink window.

Figure 3.2.1: The Simulink toolbar

Working with blocks

Adding Blocks to Model

There are two ways to add blocks to a model: The Library Browser or the Quick Search:

- **Library Browser** Shows all blocks available in Simulink, sorted by folders such as 'Math Operations' or 'Signal Routing'. There is a search bar on the top left. Drag blocks from the library straight onto your model canvas.
- **Quick Search** Directly search for blocks by single clicking on the background of your model and typing in a search term. Select a block from the search results to quickly add it to your model.

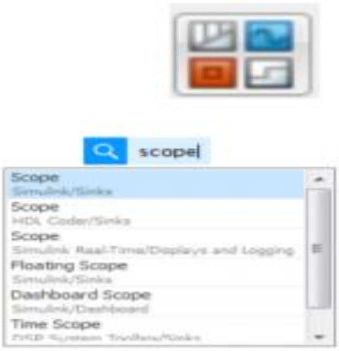


Figure 3.2.2: Adding Blocks to Model

Automatic Block Input Box

When adding a block to a model for the first time, the most common parameter will often pop up automatically for a value to be specified. e.g. If you add a Gain Block it will ask you to specify the gain value. Interacting with this can save time opening the Block Parameters menu.

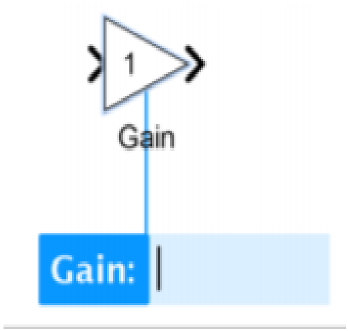
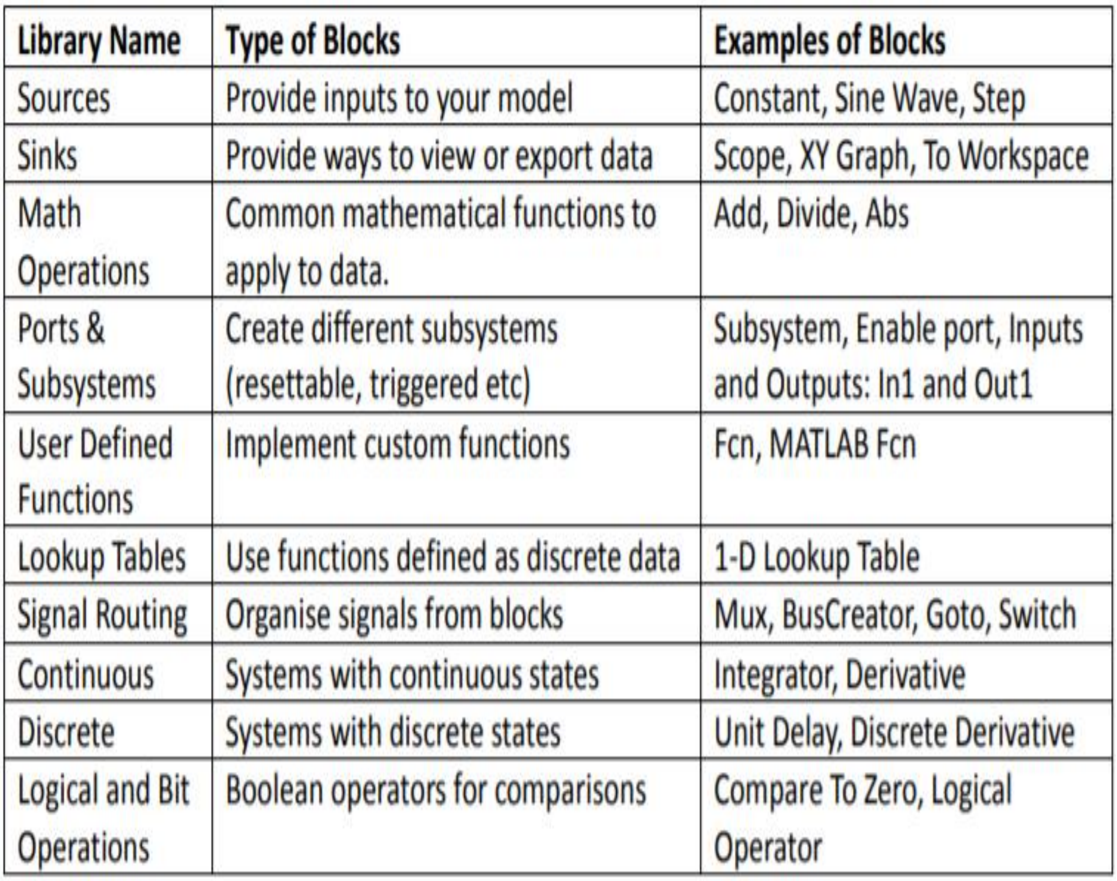


Figure 3.2.3 : Automatic block input box

Positioning Blocks

- Blocks can be moved by simply clicking and dragging.
- Connect blocks by clicking output of one block and dragging it to an input of another block.
- Once a signal connects two blocks, it can be clicked and dragged to be repositioned.
- To create a branch from an existing signal, hold ctrl while clicking and dragging.

- Blocks can be rotated/flipped for better positioning: Right click block, select "Rotate & Flip".



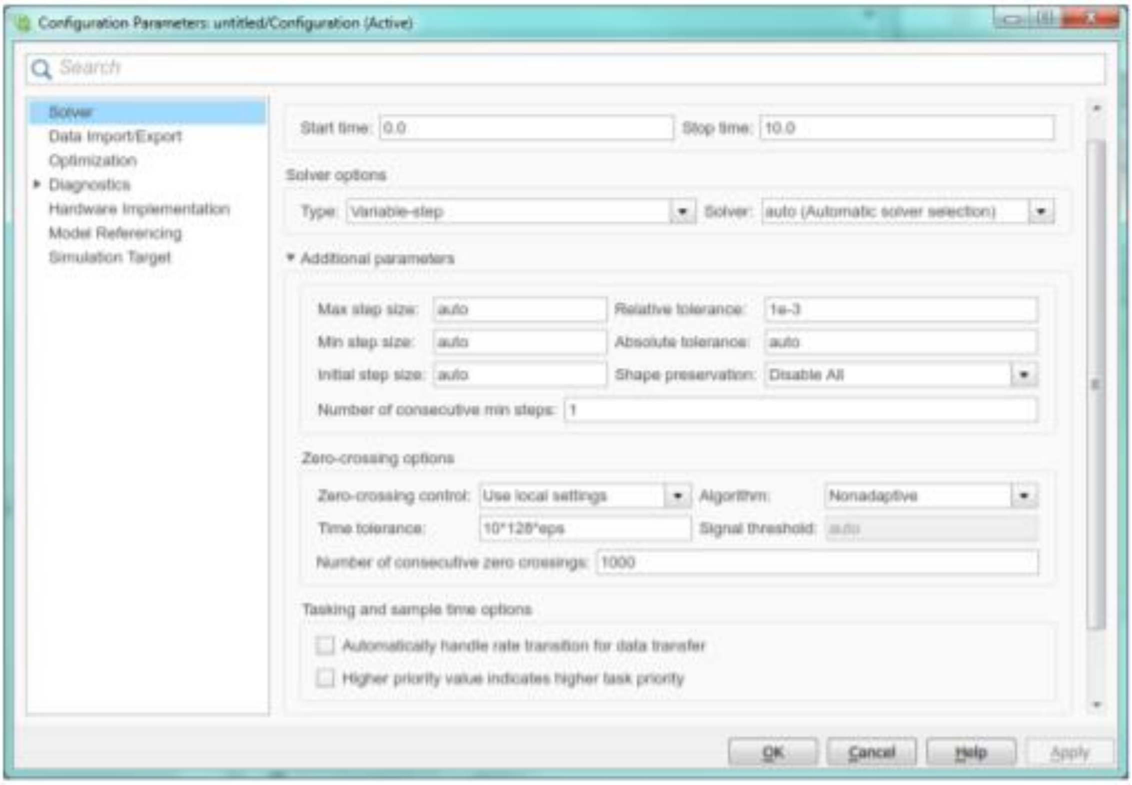
Library Name	Type of Blocks	Examples of Blocks
Sources	Provide inputs to your model	Constant, Sine Wave, Step
Sinks	Provide ways to view or export data	Scope, XY Graph, To Workspace
Math Operations	Common mathematical functions to apply to data.	Add, Divide, Abs
Ports & Subsystems	Create different subsystems (resettable, triggered etc)	Subsystem, Enable port, Inputs and Outputs: In1 and Out1
User Defined Functions	Implement custom functions	Fcn, MATLAB Fcn
Lookup Tables	Use functions defined as discrete data	1-D Lookup Table
Signal Routing	Organise signals from blocks	Mux, BusCreator, Goto, Switch
Continuous	Systems with continuous states	Integrator, Derivative
Discrete	Systems with discrete states	Unit Delay, Discrete Derivative
Logical and Bit Operations	Boolean operators for comparisons	Compare To Zero, Logical Operator

Table 3.3 Over view of libraries

The Solver

Most of the time, you can just use the default settings to run your model. However, you will sometimes find that you will want the model to use smaller steps, or fixed width steps. This is all configurable on the Solver page of the Configuration Parameters

From the menu bar on your model select **Simulation ► Model Configuration Parameters** Or simply use ⚙▾ the shortcut on the toolbar



Start time: 0.0
Stop time: 10.0

Solver options

Type: Variable-step
Solver: auto (Automatic solver selection)

Additional parameters

Max step size: auto
Relative tolerance: 1e-3
Min step size: auto
Absolute tolerance: auto
Initial step size: auto
Shape preservation: Disable All
Number of consecutive min steps: 1

Zero-crossing options

Zero-crossing control: Use local settings
Algorithm: Nonadaptive
Time tolerance: 10*128*eps
Signal threshold: auto
Number of consecutive zero crossings: 1000

Tasking and sample time options

□ Automatically handle rate transition for data transfer
□ Higher priority value indicates higher task priority

Figure 3.2.4: Configuration parameters Tab

Once opened, select **Solver.** There are many configurable settings.

- **Start/Stop** time: It is suggested that you leave the start time as zero. The stop time is same as in the toolbar at the top of your model.
- There are two types of solver:
    - A variable step solver (the default): This will automatically adjust the step size as the model runs. If you are using variable step generally keep the default solver (ode45). Set the Max step size to a small fixed value to improve the smoothness of any plots.
    - A fixed step solver will be necessary for models with discrete components. If it also has no continuous components, change the solver to Discrete (no continuous states) and set the step size to a known value. The fixed solvers are numbered in order of simplicity, ode1 being the simplest.

- **Zero-crossing options**: Under “Additional Parameters” you will find the Zero-crossing options. Further details about zero-crossing are found on page 27. For more information about solvers, click on the Help button at the bottom of the configuration parameters window, while you are viewing the solver section.

For more information about solvers, click on the **Help** button at the bottom of the configuration parameters window, while you are viewing the solver section.

3.2.1 Math Operations Library

The blocks in this library relate to common mathematical functions.

- **Add, Subtract** and **Sum** Blocks

  The Add, Subtract and Sum blocks are all essentially the same. By changing the Icon shape and List of signs in the block parameter you can convert one into the other.
- **Product** and **Divide** Blocks

  The Product and Divide blocks are also interchangeable. You can use a list of asterisks and forward slashes in the number of inputs block parameter to define operations needed e.g. Block to the right was created by number of inputs = **/*/****
- **Gain Block**

  The Gain block can be used to multiply a signal by a constant value. You must configure the Block Parameters to perform matrix or element-wise (array) multiplication.

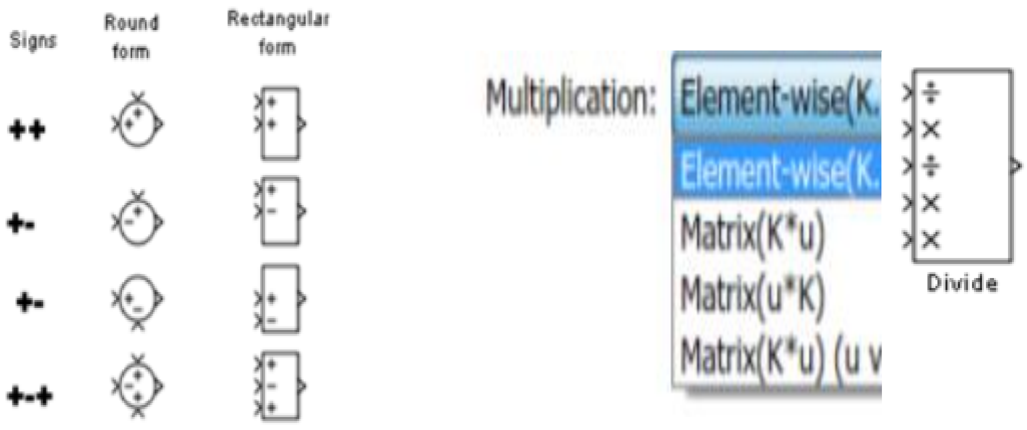


Figure 3.2.5: Mathematical functions

There are lots of blocks for **specific mathematical functions**:

- The **Abs** block finds the magnitude or absolute value of a signal,
- **Unary Minus** block negates a signal and
- **Bias** block adds a constant to a signal. The **Sign** block outputs +/- 1 depending on the sign of the input.

There are also blocks with **multiple functions to choose from**:

- The **Math Function** block can perform many different functions: square, square root, log, reciprocal etc. A block parameter allows you to select which particular function you want.
- The **Trigonometric** Function block operates in a similar manner but for cos, sin, tan-1 etc.

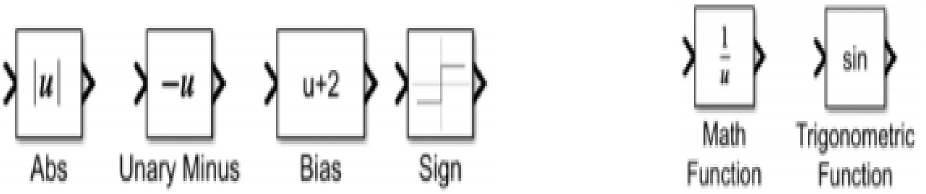
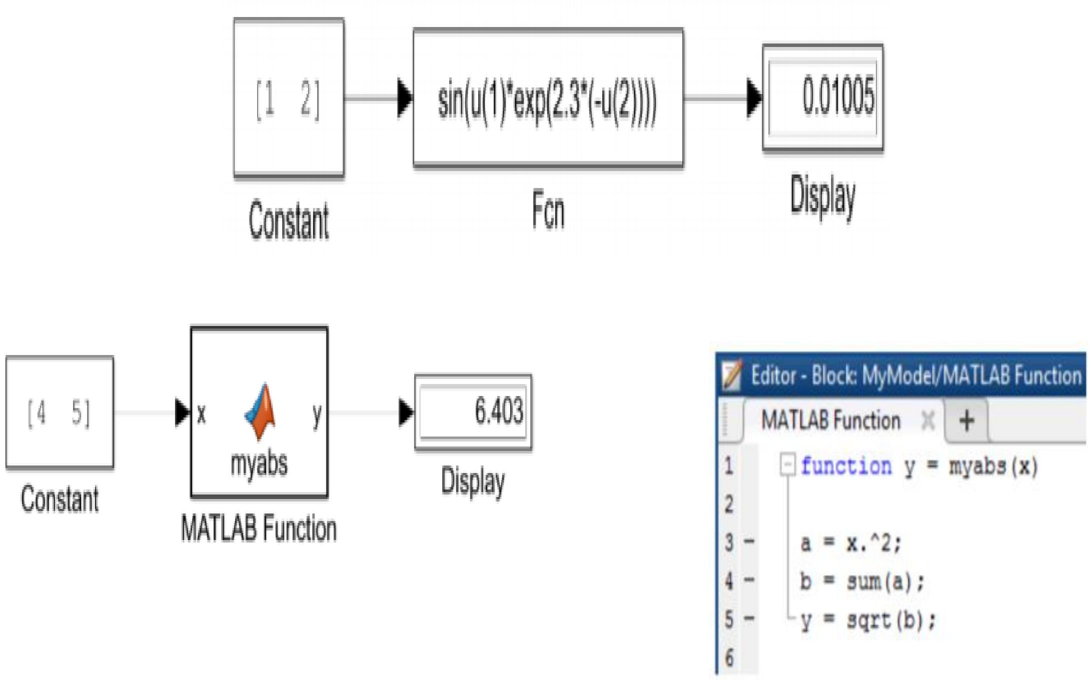


Figure 3.2.6 : Other mathematical functions

3.2.2 User defined functions

- User Defined Functions Sometimes you cannot find the exact function that you want. If this is the case, then try the block called **Fcn**. This allows you to enter a mathematical expression using a restricted set of operators and functions.
- To see what is allowed, click on the Help in the block parameters. If you need something a bit more complicated then you can write your own MATLAB function and use the block **MATLAB Fcn.**
- This block has been configured to call the MATLAB function myabs, shown left. x is the input and y is the output. Notice that the MATLAB function is not stored as a .m file in the working directory, instead you are editing code stored within the block itself. This can be seen from Block: MyModel/MATLAB Function. If you are using Simulink to create a program using the Real Time Workshop, then you will need to use the Embedded Matlab Function or an S Function. S functions are used to embedded Matlab, C or Fortran into your model.



```
function y = myabs(x)

a = x.^2;
b = sum(a);
y = sqrt(b);

```

Figure 3.2.7: User defined functions

3.2.3 Ports and Subsystems

A subsystem is a collection of blocks grouped together to carry out a particular task. They help to keep models organized and easier to understand. The example subsystem on right takes two inputs. It doubles the first input, quadruples the second input and outputs the sum.

Creating a subsystem

- ❖ Drag in a Subsystem block from the **Ports and Subsystems library**

OR

- ❖ Select the blocks that you want to put into a subsystem and right click then select **Create Subsystem from Selection** in the menu that appears.

You can have many nested subsystem (i.e. a subsystem in a subsystem).

Once you have a subsystem, you can open it by **double clicking** the block. This opens a new tab showing the blocks of the subsystem. Exit the subsystem and return to the top page by:

- ❖ Selecting the **'Up to Parent'arrow** ⇧

OR

- ❖ Selecting the home tab, e.g. 'MyModel'

Input and output ports are called In and Out. If you need extra input/output ports, they can be found in the Ports and Subsystems library or the relevant Source / Sink library.

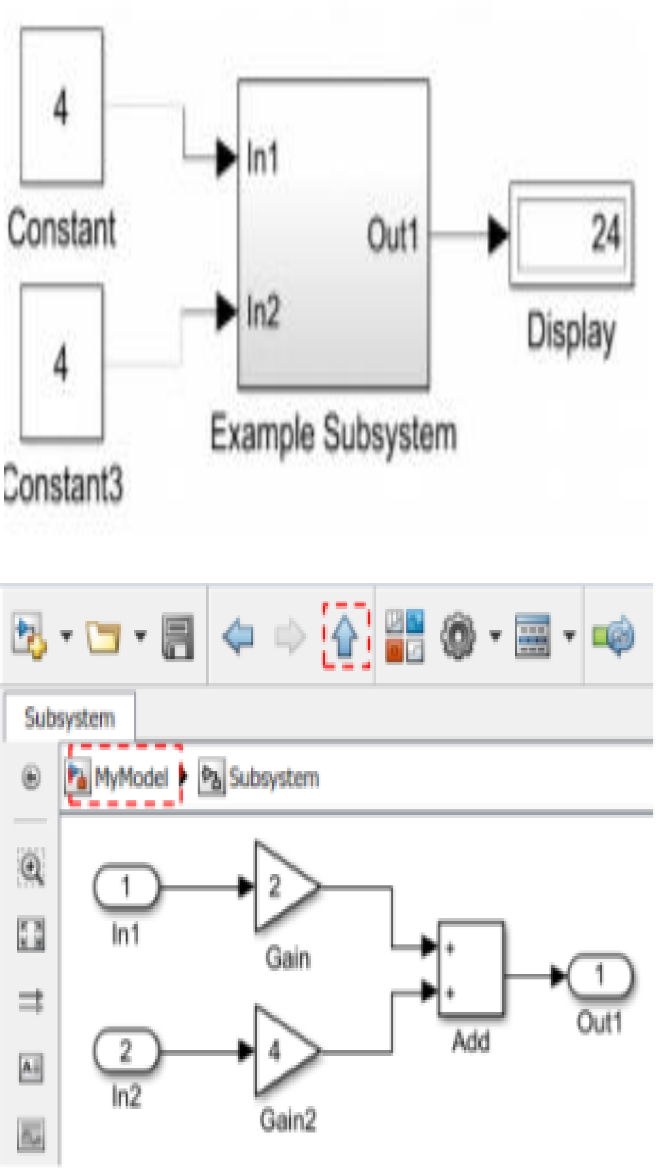


Figure 3.2.8 : Creating a subsystem

3.3 Methodology

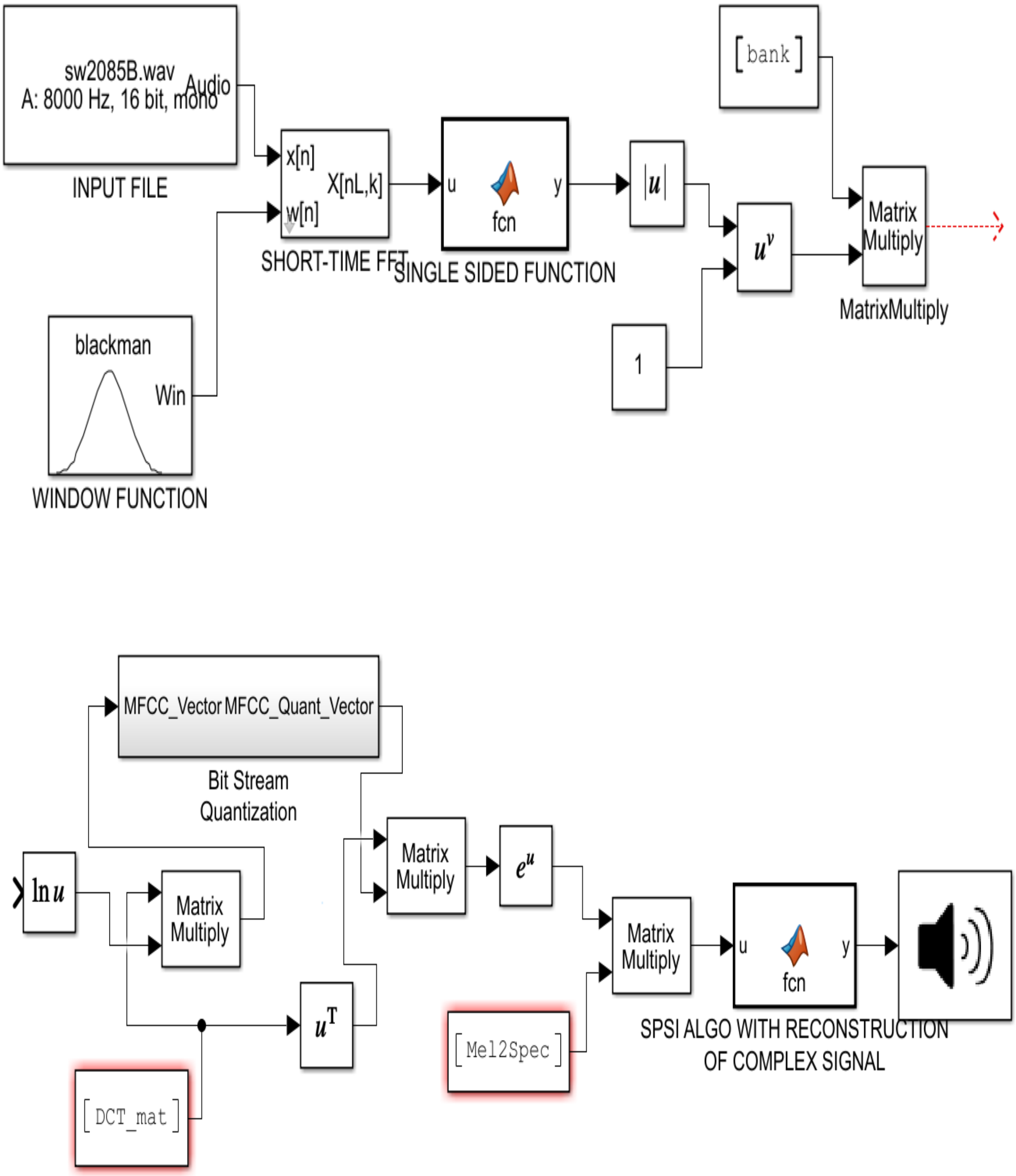


Figure 3.3.1 : Block diagram of Low bit-rate codec

3.3.1General algorithm for the complete process

- Take a wav file with ‘n’ no of samples/sec
- Apply ‘X’ point STFT for x samples after windowing it with window function.

- We get X FFT coefficients which will be in frequency domain.
- As FFT is complex conjugate, consider only X/2=Y coefficients.
- From these coefficients we consider only magnitude and drop out phase.
- Then the first compression happens where Y coefficients will further be reduced (ex: to some Y/5 coefficients) by applying Mel filter and DCT.
- Mel filter filters out in such a way that the remaining coefficients will be enough for retrieving back the input signal
- DCT is also applied in order to further compress it into with significant coefficients.
- These coefficients are called as Mel Frequency Cepstral Coefficients (MFCC)
- Then this MFCC are further compressed using codebook where we use vector quantizer.
- In codebook the input MFCC coefficients are grouped and each group is mapped with unique address.
- This address is converted into binary bits and transmitted.
- In receiver side we have same codebook which decodes it.
- Then the reverse process will happen and 'Y' fft coefficients will be retrieved back.
- For retrieving back phase Single Pass Spectrogram Inversion (SPSI) algorithm is applied.
- We reconstruct back the complex signal by converting it into time domain and get the output which will be of less than 2kbps and is perceptual.

3.3.2 Working principle:

As per our algorithm mentioned above, in our project at first, we are sampling audio signal at 8k Hz i.e. at 8000 samples/sec.

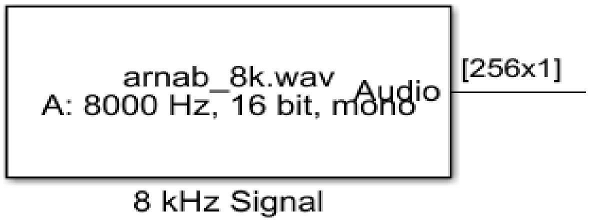


Figure 3.3.2 : Input File

But here we are processing as a frame with 256 samples/frame. And then apply **window function** of size 1024 samples for four frames of 256 samples each. Here we are using Blackman window based on experimental results as it was giving a better perceptibility and sampling is periodic.

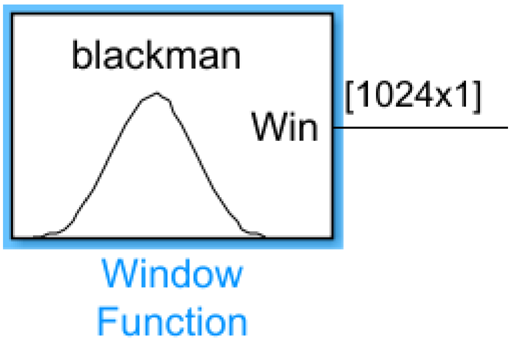


Figure 3.3.3 : Window Function

Blackman window is defined as:

$$w[n] = a_0 - a_1 \cos\left(\frac{2\pi n}{N}\right) + a_2 \cos\left(\frac{4\pi n}{N}\right)$$

$$a_0 = \frac{1-\alpha}{2}; \quad a_1 = \frac{1}{2}; \quad a_2 = \frac{\alpha}{2}.$$

Where, α=0.16,

N= number of Samples

And 0< n<=N

Then we are applying 1024-point STFT for 1024 samples i.e four frames with an overlap between two consecutive window of (1024-256) = 768. STFT module has a buffer of size 1024 for the input samples.

The **Short-time Fourier transform (STFT)**, is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. This reveals the Fourier spectrum on each shorter segment.

The Fourier transform (a one-dimensional function) of the resulting signal is taken as the 1024 sized windowed FFT coefficients is slided along the time axis, resulting in a two-dimensional representation of the signal. Mathematically, this is written as:

$$\mathbf{STFT}\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-i\omega t}\,dt$$

By using this windowing functions, we can further enhance the ability of an FFT to extract spectral data from signals. Windowing functions act on raw data to reduce the effects of the leakage that occurs during an FFT of the data. Leakage amounts to spectral information from an FFT showing up at the wrong frequencies. After windowed STFT, 1024 input time domain signal will be converted into frequency domain signal.

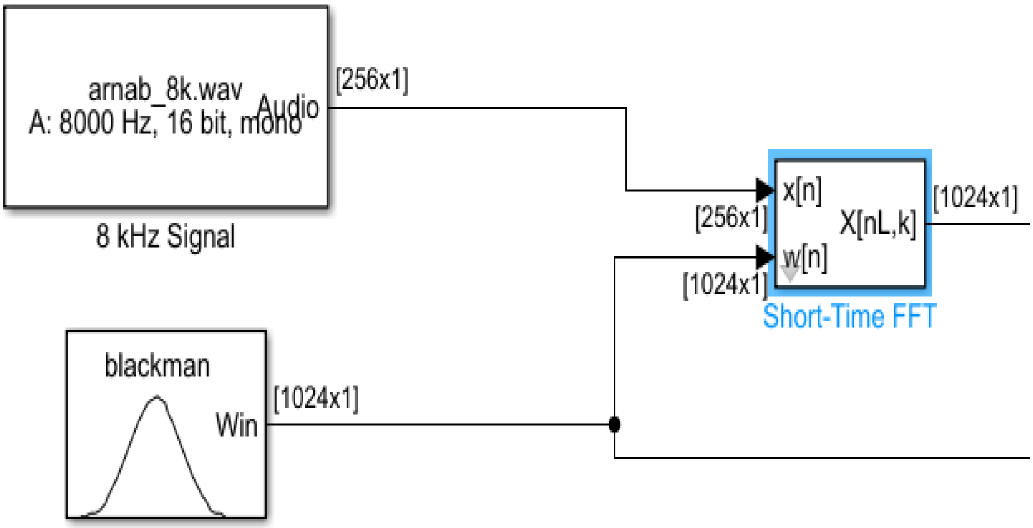


Figure 3.3.4 : Short-Time FFT

Now we have corresponding 1024 complex numbers in frequency domain i.e. frequency coefficient. As these 1024 FFTs are complex conjugate, so we consider (1024/2)+1 = 513 coefficients. Here we are including the 0th index value as it is a dc value.

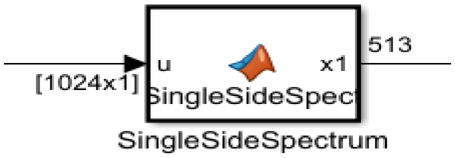


Figure 3.3.5 : Single Side Spectrum

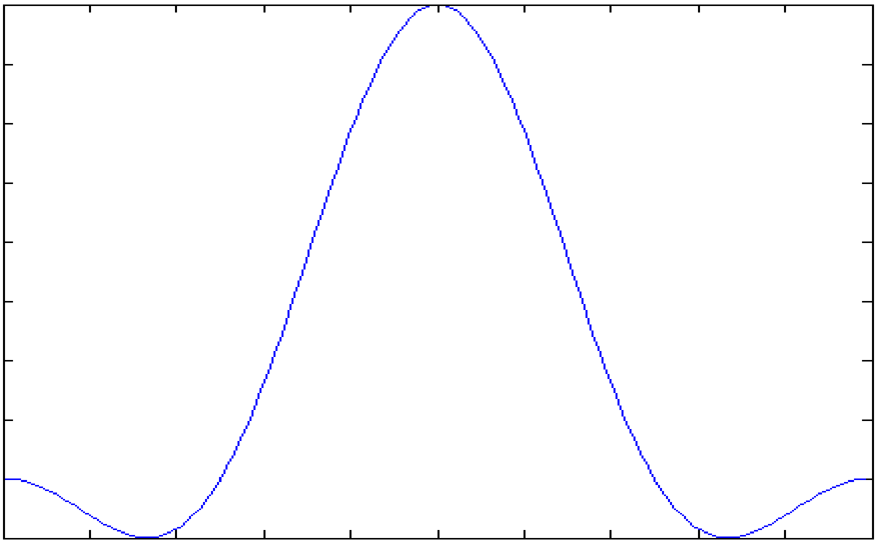


Figure 3.3.6 : Graph of FFT coefficients with x axis of total number of coefficients and y axis corresponds to their amplitudes

So now we have total of 513 FFT coefficients with both magnitude and phase components. In order to reduce the Bit-Rate while transmitting, we drop out the phase of all the coefficients and only consider **magnitude** so that we can generate the phase back in receiver side to reconstruct the signal back. Finally, we have 513 FFT coefficients with only magnitude spectrum. This is the first step to achieve low Bit-Rate.

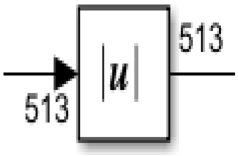


Figure 3.3.7 : Considering only Magnitude

The second step for achieving low Bit-Rate is to convert these 513 FFT coefficients into 90 MFCC i.e, **Mel Frequency Cepstral Coefficients**.

The mel frequency cepstral coefficients (MFCC) is that parameter , whose major task is data reduction by converting the input signal into parameters while preserving virtually all of the speech signal information. MFCC takes into account human perception for sensitivity at appropriate frequencies by converting the conventional frequency to **Mel Scale**, and are thus suitable for speech recognition tasks.

It has three steps, they are

1) Reduce the 513 FFT coefficients into total of 90 coefficients using **Mel Filter**.
2) To apply **log** on those 90 coefficients.
3) Taking the **DCT**.

After all these three steps the coefficients we get is called as MFCCs.

In first step, we use Mel Filters in order to reduce the coefficients to 90. Mel Filter consists of 90 filters which are overlapped and are in triangular shape. The reason for the triangular shape is to balance the energy.

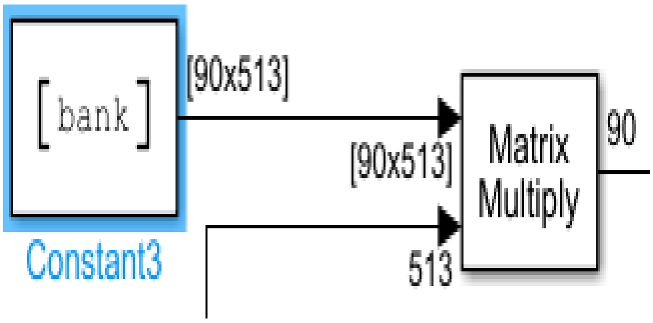


Figure 3.3.8 : Mel Filter Bank Multiplied with Input FFT Coefficients

The **mel filter** is designed based on perceptual model of human ear where human ear will be more sensitive in and around 1kHz. The sensitivity reduces as we move towards higher frequencies i.e, from 2kHz to 4kHz as our sampling rate is 8000.

The chief feature of the mel frequency cepstral coefficients, comes from the distribution of filters in the frequency domain. They are equally spaced from 0–1000 Hz, and their spacing increases continuously beyond 1000 Hz. There is a great similarity between the spacing of these filters and the human perception scale.

The mel scale, which maps an acoustic frequency f to a "perceptual" frequency scale as follows:

$$\text{mel frequency} = 2595 \log_{10}\left(1 + \frac{f}{700}\right).$$

After this process we get 90 mel frequencies.

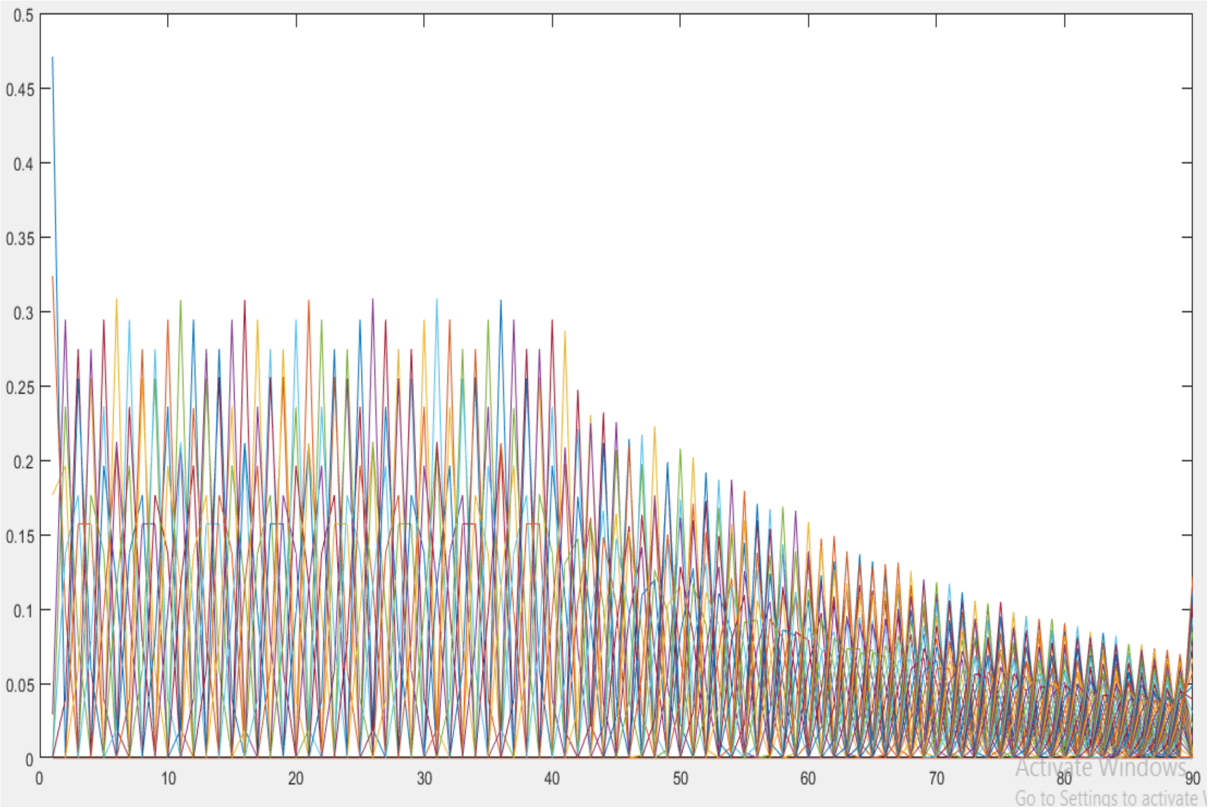


Figure 3.3.9 : Mel Filters

In the second step, we apply **log** to all those frequencies in order to maintain dynamic range in terms of amplitude.

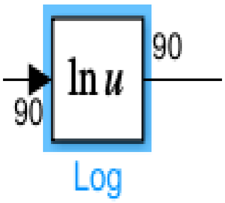


Figure 3.3.10 : Taking the Log

The third step is to take the DCT i.e, **Discrete Cosine Transform** of those 90 mel frequencies. A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies such that we can retrieve back the original signal back.

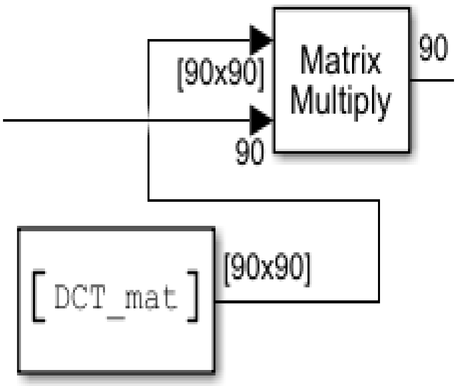


Figure 3.3.11 : DCT matrix multiplied with output of Log

In our project this helps in further compressions by making the lower mel frequencies more significant and the rest of the frequencies amplitude will be negligible which needs less bits to transmit by grouping all of them into one. This way DCT helps in compression in terms of amplitudes.

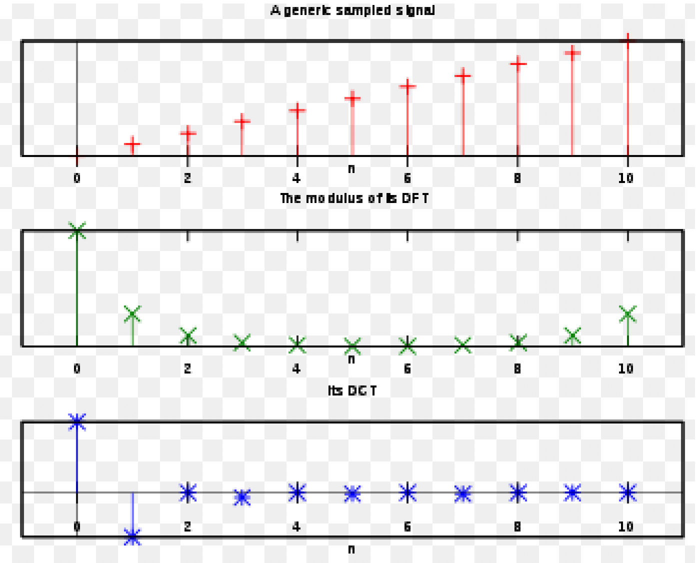


Figure 3.3.12 : DCT

Now we have 90 MFCCs after the second compression.

The next step of compression which is the final compression is done by using **Codebooks**. First we divide the 90 MFCCs into four groups in order to feed it into one scalar quantizer and three vector quantizers.

Codebook comprises of encoders and decoders of three **VQs (Vector Quantizers)** and one **SQ (Scalar Quantizer)** in transmitter and receiver side respectively.

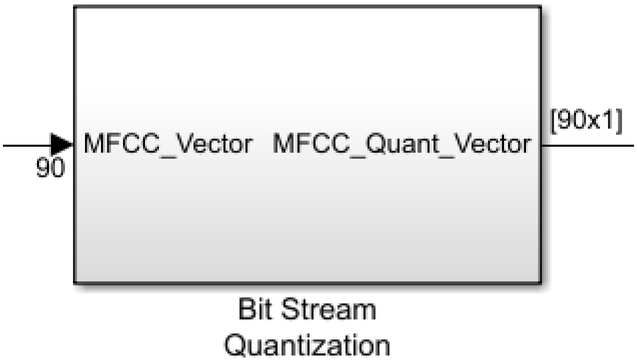


Figure 3.3.13 : Codebook

Vector quantization (**VQ**) is a classical quantization technique from signal processing that allows the modeling of probability density functions by the distribution of prototype vectors. It was originally used for data compression. It works by dividing a large set of points (vectors) into groups having approximately the same number of points closest to them. Each group is represented by its centroid point, as in k-means and some other clustering algorithms.

The simplest training algorithm for vector quantization is:

1. Pick a sample point at random
2. Move the nearest quantization vector centroid towards this sample point, by a small fraction of the distance
3. Repeat

After this training the sample points will reduce to the required number of centroids. If there is any sample that should be sent will be mapped to the centroids. Instead of that sample centroid will be choosen for transmission. However we will not get the exact original signal back in the receiver side but the approximated signal will be retrieved back that will be enough for achieving the perceptibility and intelligibility of speech.

In **Scalar Quantization (SQ)**, each input symbol is treated separately in producing the output, while in vector **quantization** the input symbols are clubbed together in groups called vectors, and processed to give the output.

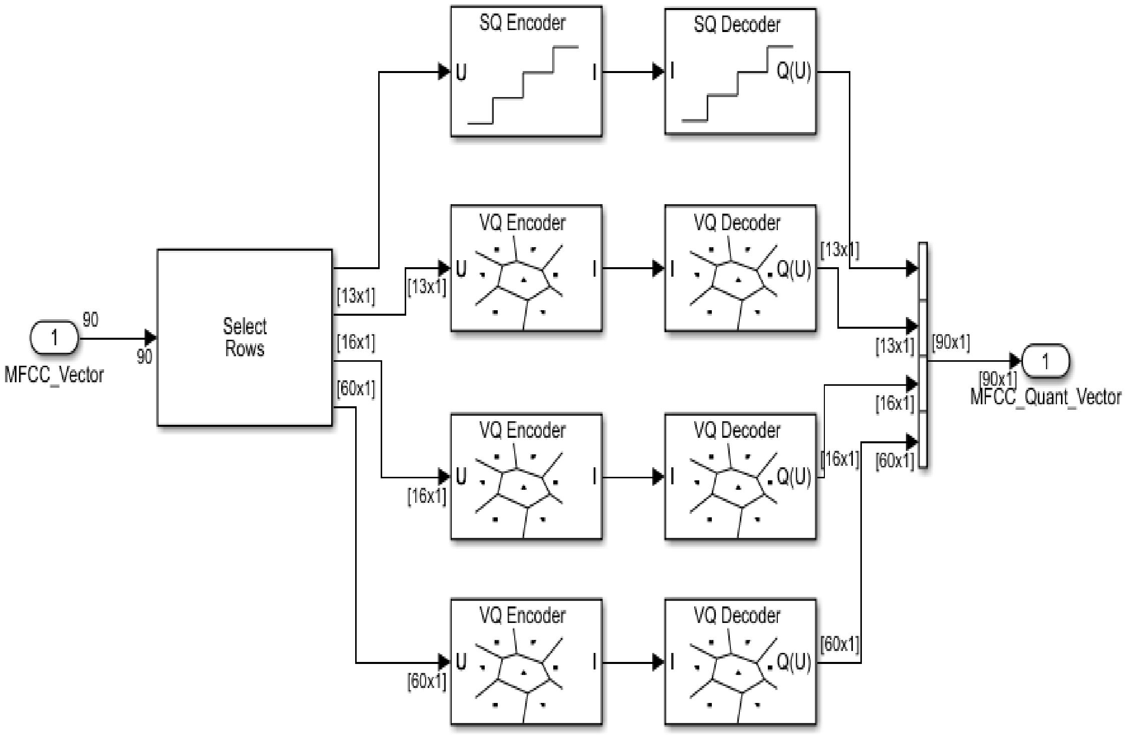


Figure 3.3.14 : Vector Quantizers and Scalar Quantizers

We are using scalar quantizer for the first MFCC as the value is very important for accurately reconstructing the signal back as it determines the DC component of the signal.

In Codebook, the VQs and SQ will be in matrix format in Matlab. As the 90 MFCCs are divided into four groups, we will calculate the squared error of the number of MFCCs present in each group with each column of the VQs or SQ by using squared error. The column with less error will act as address and will be coded into binary bits and transmitted. At the receiver side it will be decoded to get those MFCCs back.

By this method, we managed to reduce the bits for transmitting. For example: consider a group of 13 MFCCs, so the input matrix will be of 13x1 which will be fed to a VQ encoder of matrix 13x16384 where 16384 i.e 2^{14} will be the size of the VQ. Then, squared error is calculated between the 13 input coefficients and each columns of VQ. Therefore we will get 16,384 errors out of which we choose the column with less error. Now this column number will act as an address which will be encoded into 14 binary bits and transmitted. So we can come to an inference that the bits required for transmitting 13

MFCCs directly will require 13x32 or even 13x64 bits based on the value of MFCCs but using this method we require just 14 bits.

So, there will be almost 60 times compression. The same process will happen in scalar quantizer but with only one MFCC value.

After all these compressions, the data will be transmitted and retrieved back by following the inverse process i.e, using VQ and SQ decoder, transpose of DCT, inverse of log and the transpose of Mel Filter. Now, we have 513 FFT coefficients with only magnitude components.

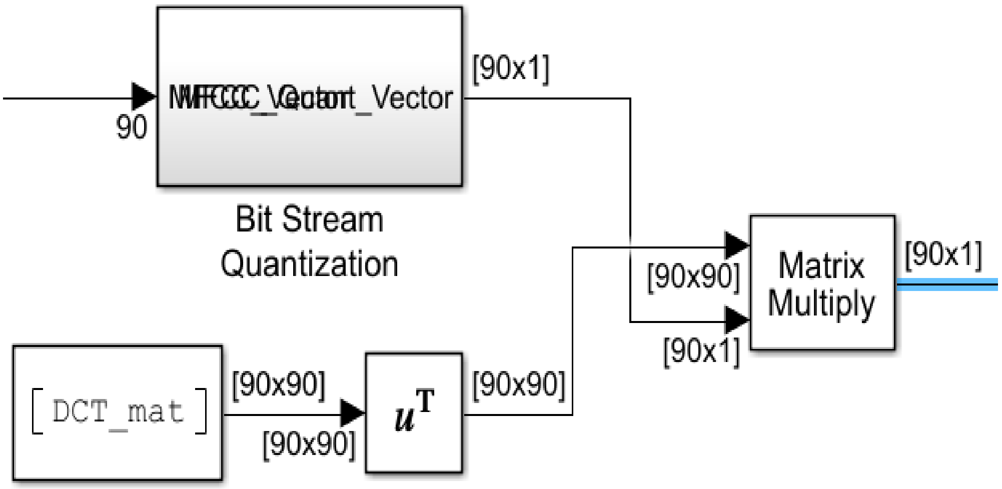


Figure 3.3.15 : Inverse DCT

Exponential of all the 90 MFCCs will be taken which retrieves back the original values before applying Log to it.

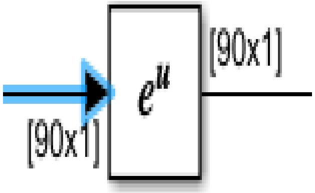


Figure 3.3.16 : Inverse of Log

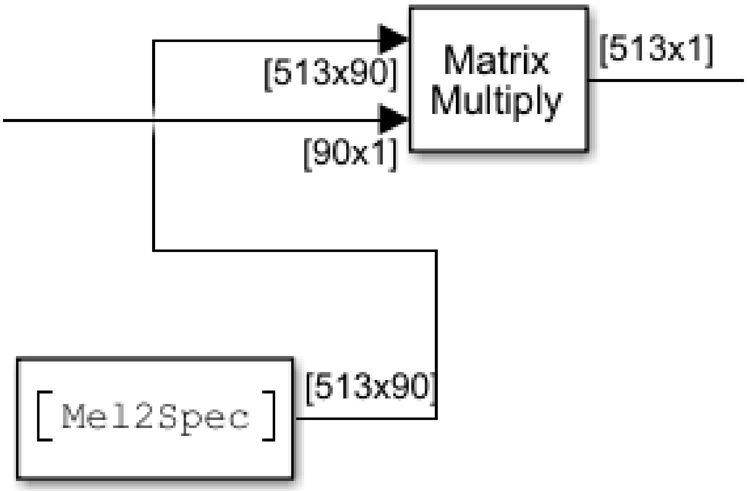


Figure 3.3.17 : MFCC to FFT coefficients

The next process is to generate the phase for corresponding magnitudes by using an algorithm i.e, **Single Pass Spectrogram inversion (SPSI)**.

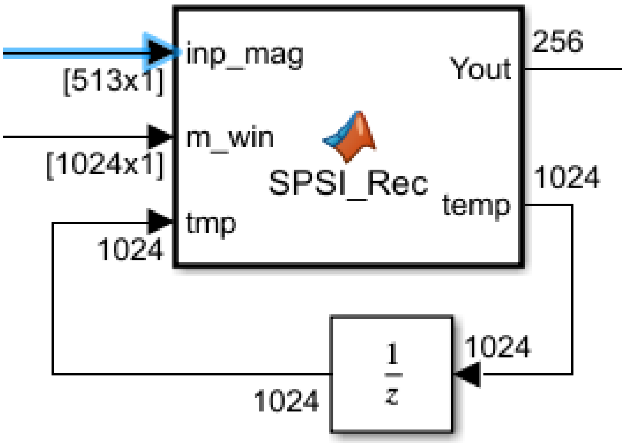


Figure 3.3.18 : Matlab Function For SPSI Algorithm

The SPSI method begins with a series of magnitude spectra assumed to represent overlapping windowed frames of an actual time-domain signal. The frames are assumed to be windowed using a blackman window, with an analysis step size *S* which is one quarter the frame length *L*.

The steps of the algorithm for each frame are:

1.First, from the magnitude spectrum, we identify the bins that represent peaks in the spectrum by comparing the magnitude of each bin j with that of neighbors, *j* + 1 and *j*−1

2. If magnitude of present bin > magnitude of previous bin and magnitude of present bin > magnitude of future bin, then bin j is considered a peak. The frequency of bin j is

$$\omega_j = \frac{2\pi j}{L}$$

Where L = Frame Length

3. Quadratic interpolation is then used as to identify the true peak position based on the magnitude of the peak bin and its neighbors using the formula:

$$p = 0.5\frac{\alpha - \gamma}{\alpha - 2\beta + \gamma}$$

Where, alpha - magnitude of previous bin

Beta - magnitude of present bin

Gamma - magnitude of future bin

The value p has a value in the range [-0.5,0.5]. It represents the deviation from the true peak from the peak bin. This is important since each peak corresponds to a sinusoid whose frequency does not necessarily line up precisely with a bin center frequency. This interpolation yields a better estimate for the true peak frequency. Then the frequency of true peak bin is,

$$\omega_j = \frac{2\pi(j+p)}{L}$$

4. Then using phase accumulator Phi, which represents the phase values to use for bin j at the current frame m,

$$\phi_{m,j} = \phi_{(m-1),j} + S\omega_j$$

where *S* refers to the synthesis step size (which is the same as the analysis step size). The phase accumulator is updated according to this formula only for the peak bins.

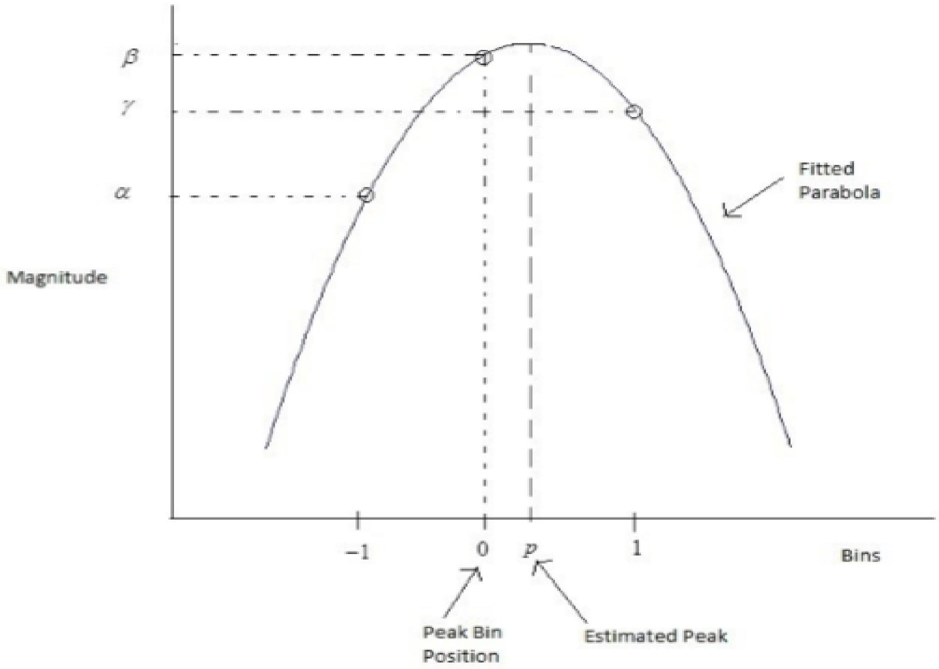


Figure 3.3.19 : Quadratic Interpolation to Estimate the Value of the True Peak

5. The phase at the peaks has been determined, the phases at the remaining bin positions can be determined depending on the sign of p.

The two immediate neighboring bins of true peak bin j will take the phase of the peak bin shifted by pi. For all the bins other than the peak bin and its immediate neighboring bins will have the following conditions, they are:

Case (i): $p < 0$,

If $p < 0$, All the bins to the right of the peak bin will have the phase of the peak bin with a shift of pi until the next 'trough' or 'valley.' Beyond the trough, the bins are locked to the next peak. The bins to the left of the peak bin will have the same phase as that of the peak bin. Again this value is propagated until a trough is encountered.

Case (ii): $p \geq 0$,

If $p \geq 0$, the same procedure is followed as above, but in reverse. The bins to the right of the peak bin will take the phase of the peak bin, and the bins to the left of the peak bin will take the phase of the peak bin with a shift of pi. In both cases the procedure will carried out as far as the next trough comes.

Now that phases for every bin have been computed, they can be combined with the frequency component magnitudes providing the information necessary to reconstruct the time domain signal back using IFFT i.e, Inverse FFT.

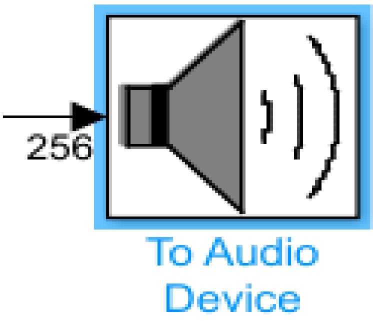


Figure 3.3.20 : Audio Output

Chapter 4

RESULTS

The project will be evaluated based on both standard test methodology for automated assessment of the speech quality that is Perceptual Evaluation of Speech Quality (PESQ) value and also perceptibility which involves experimentation. After the **PESQ** analysis, a **score** is given ranging from -0.5 to 4.5. A higher **score** means a better speech quality.

4.1 Table of Evaluation:

	PESQ
Model with only SPSI Algorithm (For all 513 coefficients)	2.5115
Model with only SPSI Algorithm (For 256 coefficients)	2.4062
Model with only SPSI Algorithm (For 128 coefficients)	2.4186
Model with SPSI Algorithm and Mel Bank of 70 MFCCs as output	2.2269
Model with SPSI Algorithm and Mel Bank of 90 MFCCs as output	2.4335
Model with SPSI Algorithm and Mel Bank of 90 MFCCs as output and Codebook	2.2320

In the above table, we observed that the model with only SPSI Algorithm where the phase for all the 513 coefficients is calculated using Quadratic Interpolation(QI) is giving the PESQ value of 2.5115 which is of average because we have dropped the phase component in transmitter side.

The model with SPSI Algorithm where the phases of 256 out 513 coefficients are calculated using QI and the for the rest of them phase is randomly generated. Here we are getting a PESQ of 2.4062 and the perceptibility was not better compare to the previous model.

The model with SPSI and Mel Bank which gives out 70 coefficients out of 513 is giving PESQ of 2.2269 and the Mel Bank which gives out 90 coefficients out of 513 is giving PESQ of 2.4335. The model with 90 coefficients as output is also better intelligible.

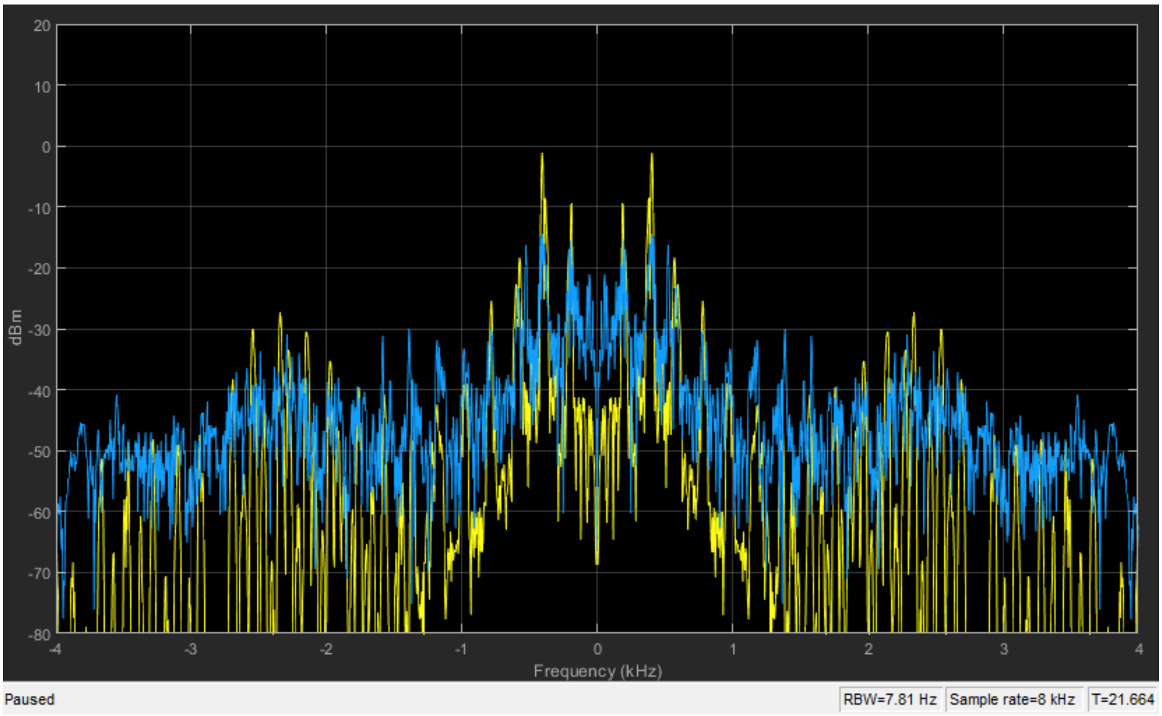
So, our final model consists of SPSI Algorithm with Mel Bank which gives out 90 coefficients and the codebook of SQ size 128 i.e, 2^{7},VQ1 size 16384 i.e, 2^{14},VQ2 size 16384 i.e, 2^{14} and VQ3 size 512 i.e, 2^{9} is giving us PESQ of 2.2320 which is perceptual.

The Bits that is required for the codebook to transmit the data is

7 + 14 + 14 + 9 = 44 bits

The Bit-Rate that we have achieved is 1.375kbps which is less than 2kbps.

4.2 The spectrum of input and output audio signal



Paused

RBW=7.81 Hz | Sample rate=8 kHz | T=21.664

Figure 4.2.1: spectrogram of input and output signal

In this figure the input signal is in Yellow colour and the output is in blue colour. As we can observe the output is almost following the input signal pattern.

Chapter 5

APPLICATIONS AND ADVANTAGES

5.1 Applications

The applications of Low Bit-Rate Speech codec are:

1) Military Voice Communications : Communication between Ground stations and war tankers or between two war tankers.

   Military tanks are subjected to extreme conditions in the war zones and the establishment and maintenance of communication between military tanks and the ground station or between tanks is a difficult task. Here they can use the model.

2) Mobile Radio Communication : In order to provide the opportunity to transmit the digital speech over analog channels.
3) For Multimedia Communications

6.2 Advantages

Some of the advantages of our model are

1) Low Bit-Rate i.e, 1.3kbps, however the models which are now existing is of4kbps.
2) Better fidelity
3) Better Intelligibility
4) Better perceptibility
5) It can fit into Raspberry pi which means it can fit into a memory size of 1GB.

Chapter 6

CONCLUSIONS AND SCOPE FOR FUTURE WORK

As Low Bit-Rate speech codec is one of the trending topic, many of them are trying to develop a model with less than 2kbps as the existing model is having a minimum of 4kbps. In this project we have achieved a Bit-Rate of 1.3kbps which is less than 2kbps by dropping out the phase, reduce the coefficients further using Mel Filter and final compression done by using codebook where very less bits are required to transmit the data. And the PESQ that we have achieved is 2.2320 which is an average value which means the model is intelligible.

However the perceptibility is less compared to the model with 4kbps as we have dropped a lot of information in the transmitter side and it will be difficult to construct the signal back with very less information. But this model is perceptual and can be improved using better algorithms instead of SPSI in order improve its quality. This could be our future work.

REFERENCES

[1] DANIEL W. GRIFFIN AND JAE S. LIM, SENIOR MEMBER, IEEE, "Signal Estimation from Modified Short-Time Fourier Transform", IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. ASSP-32, NO. 2, APRIL 1984

[2] X. Zhu, G. T. Beauregard, and L. Wyse, "Real-time signal estimation from modified short-time Fourier transform magnitude spectra," Audio, Speech, and Language Processing, IEEE Transactions on, vol. 15, no. 5, pp. 1645–1653, July 2007

[3] Rivarol Vergin, Douglas O'Shaughnessy, Senior Member, IEEE, and Azarshid Farhat "Generalized Mel Frequency Cepstral Coefficients" For Large-Vocabulary Speaker-Independent Continuous-Speech Recognition

[4] JOHN MAKHOUL, Fellow, IEEE, SALIM ROUCOS, MEMBER, IEEE, ANDAND HERBERT GISH, MEMBER, IEEE. "Vector Quantization in Speech Coding".

[5] JOHN MAKHOUL, Fellow, IEEE, SALIM ROUCOS, MEMBER, IEEE, ANDAND HERBERT GISH, MEMBER, IEEE. "Vector Quantization in Speech Coding".