# Appendix E: Sample Code – Matlab Programs

**// Beginnning of Program: DataCompaction_MDWT.m //**

```
/* Matlab Program for Algorithm to perform Data Reduction
using Wavelet Transforms */


%% Program to transform data using Haar Wavelet
Transforms on already reduced nucleotide sequence using
2-bit Binary Indicator %%


tStart=tic;


lookup_seqn = cast(csvread('SRR000675.txt'), 'uint8'); %%
Sample Read of SRR000675

%lookup_seqn = cast(csvread('Cnt46_EC.txt'), 'uint8'); %%
Contig 46 of E.Coli K12 Strain

%lookup_seqn = cast(csvread('ch7_HS_1.txt'), 'uint8'); %%
Chr7 of Homo Sapien


 wavelettype =  'haar';

 maxdec = 4;

 tStart_wavelets = tic;

    for declev = 1:maxdec
```

```
        pdeclev = sprintf('dec %d', declev);

        dec                                                =
mdwtdec('r',lookup_seqn,declev,wavelettype);

    end;

tElapsed_wavelets = toc(tStart_wavelets);


    s_len4(declev) = cast(zeros(), 'uint16');

    for count = 1:declev

        s_len4(count)      =      cast(length(dec.cd{1,4}),
'uint16');

    end;


     rec = mdwtrec(dec);

     sizedec4 = dec.cd{1,4};


%%% Draw Plots %%%

    subplotm=3;

    subplotn=2;

    subplotp=0;


     subplotp=subplotp+1;

     subplot(subplotm,subplotn,subplotp);
```

```
    plot(lookup_seqn); title('original seqn');

    xlabel('time'); ylabel('scale');

    subplotp=subplotp+1;
    subplot(subplotm,subplotn,subplotp);
    plot(rec); title('reconstructed seqn');
    xlabel('time'); ylabel('scale');

   for declev = 1:maxdec
    pdeclev = sprintf('level %d decomposition', declev);
    subplotp=subplotp+1;
    subplot(subplotm,subplotn,subplotp);
    plot(dec.cd{1, declev});
    title(pdeclev); xlabel('time'); ylabel('scale');
   end;
  tElapsed = toc(tStart);
```

**// End of Matlab**

**Program:CompressSequence_DataCompaction_DWT.m //**

**///// Beginning of Program SortedDuplicateReads.m ////**

```
%% Program to Compress (& Decompress) the sequences using
%% functions like mdwtdec & mdwtrec and
%% find the duplicate reads.
```

```
%% The Transformed signals are compared after sorting
them based on lengths and comparing only
%% those sequences whose length are same

tStart=tic;
tStart_dup = tic;
tStart_wavelets = tic;

%% Metagenomics SRA data used in the paper, for finding
duplicate reads

%% To test the program for various data set, uncomment
the following list
%% of filenames one after the other and run the program
in Matlab R2009a release.
%% Various results can be checked from workspace of
matlab

%Afasta_file_reads = 'SRR000907.fna';
%Afasta_file_reads = 'SRR001669.fna';
%Afasta_file_reads = 'SRR001670.fna';
%Afasta_file_reads = 'SRR077225.fna';
%Afasta_file_reads = 'SRR000905.fna';
%Afasta_file_reads = 'SRR000906.fna';
%Afasta_file_reads = 'SRR000675.fna';
%Afasta_file_reads = 'SRR001663.fna';

Afasta_file_reads = 'SRR065619.fna';  % Small fasta file
of bacillus, to be used for testing the program & quick
results

[header,sequence] = fastaread(Afasta_file_reads);

[rows columns] = size(sequence);
seq_len=columns;

wavelettype =  'haar';

total_base_count = 0;
for count = 1:seq_len
        dbs1 = sequence{count};
        ldbs1 = cast(length(dbs1), 'uint16');

%%  To generate frequency string of the given sequence
using, Electron-Ion Interaction Pseudo Potential of
Nucleotides

        fqdbs1 = eiip_submitted(dbs1);

        lseq{count} =  ldbs1;
```

```
        total_base_count = cast(total_base_count,
'uint32') + cast(ldbs1, 'uint32');

%%
%%
%% Compression using multiple transforms :
mdwtdec/mdwtrec %%

        maxdec = 4;

%% Performing each level of transform and storing in an
array of decompositions

        for declev = 1:maxdec
            pdeclev = sprintf('dec %d', declev);
            dec(count) =
mdwtdec('r',fqdbs1,declev,wavelettype);
        end;

%% Performing reconstruction after all 4 levels of
decomposition is completed &
%% plotting it to verify with the original sequence's
plot for exact
%% reconstruction - a proof that wavelet transform based
data reduction is
%% appropriate for doing comparisions for recognizing
duplicate reads

%         rec = mdwtrec(dec(count));
%         recseq{count} = rec;


end;

tElapsed_wavelets = toc(tStart_wavelets );

%%
%% To find the total length of decomposed sequence after
each level of transform is performed. %%

    s_len4(seq_len) = cast(zeros(), 'uint16');
    for count = 1:seq_len
        s_len4(count) = cast(length(dec(count).cd{1,4}),
'uint16');
    end;

%%
%%
%%  This block identifies the duplicate reads, by
checking those reads
```

```
%%  whose decomposed length is same, and if the lengths
are same then, only
%%  compare decomposed values to each other for
similarity check.
%%

    tStart_dup_actual = tic;

    flag = cast(zeros(size(lseq)), 'logical');

    i = 0;
    for row = 1:(seq_len-1)

        if (flag(row) == 1) %% To check if the row is
already identified as a duplicate of earlier row
            continue;
        else

        j = 1;
        for col = (row + 1):seq_len

            if ( seq_len < 255)
                cast_size = 'uint8';
            elseif (seq_len >= 255 && seq_len < 65535)
                cast_size = 'uint16';
            elseif ( seq_len >= 65535 )
                cast_size = 'uint32';
            end;


             if (s_len4(row) == s_len4(col) )
                if( dec(1,row).cd{1,4} ==
dec(1,col).cd{1, 4} )
                    if (j == 1)
                        i = i + 1;
                        dup_read(i,j) =
cast(row,cast_size);
                        j = j + 1;
                        dup_read(i,j) =
cast(col,cast_size);
                        j = j + 1;
                        flag(row) = 1; %% To set the
status, if the row has identified any duplicate
                        flag(col) = 1; %% To set the
status to already compared read, if the row is already
identified as a duplicate of earlier row do not check for
its duplicate again
                    else
                        dup_read(i,j) =
cast(col,cast_size);
                        j = j + 1;
```

```
                        flag(col) = 1;
                    end;
                end;
            end;
        end;
        end;
    end;

tElapsed_dup_actual = toc(tStart_dup_actual);
tElapsed_dup = toc(tStart_dup);
tStart_stats = tic;


%% The following block of code generates the statistical
and more organized and user-friendly output. It is not
compulsory for the code to identify duplicate reads.
%%
%% Generating the Unique list of duplicate reads, unlike
above block which
%% is generating the entire list. This block is based on
the output of the above block.
%% So, do not remove or alter the above block
%%
%% It is also Calculating the total number of duplicate
reads and
%% its percentage ie ratio of duplicate reads to the
total number of reads
%% And the redundant total no. of reads and total number
of bases
%%
%%

%%
%% To find the total max no. of duplicates
%%
    [dup_rows,dup_cols] = size(dup_read);

i=1;
j=1;

total_unique_dup_read = 0;
total_dup_read_count = 0;
total_unique_dup_read_count_bases = 0;
total_redundant_dup_read_count_bases = 0;


    for i=1:dup_rows
        if(dup_read(i,1) ~= 0)
            total_unique_dup_read = total_unique_dup_read
+1;
        for j=2:dup_cols
```

```
            if(dup_read(i,j) ~= 0)

                read_no = cast(dup_read(i,1), cast_size);
                dup_read_details(i,1) =
cast(read_no,cast_size);    %% finds the read no. of a
unique duplicate read
                dup_read_details(i,2) =
cast(lseq{1,read_no},cast_size);   %% finds the no. of bp
in a read
                dup_read_details(i,3) =
cast(j,cast_size);    %% finds total no. of copies of
this duplicate read
                dup_read_details(i,4) =
cast((dup_read_details(i,2) * (dup_read_details(i,3) - 1)
),cast_size); %% finds total redundant bases of this read

            end;
        end;

        total_dup_read_count = total_dup_read_count +
dup_read_details(i,3);
        total_unique_dup_read_count_bases =
total_unique_dup_read_count_bases +
cast(dup_read_details(i,2),'double');
        total_redundant_dup_read_count_bases =
total_redundant_dup_read_count_bases +
cast(dup_read_details(i,4),'double');
        end;
    end;

    total_redundant_dup_read = total_dup_read_count -
total_unique_dup_read;
    total_percentage_redundant_read =
(cast(total_redundant_dup_read, 'double') * 100)/
seq_len;
    total_percentage_redundant_bases =
(cast(total_redundant_dup_read_count_bases,'double') *
100)/ cast(total_base_count, 'double');

    total_dup_read_count_bases =
cast(total_unique_dup_read_count_bases, 'double') +
cast(total_redundant_dup_read_count_bases, 'double');
    total_percentage_dup_read_bases =
(cast(total_dup_read_count_bases, 'double') * 100) /
cast(total_base_count, 'double');

    total_percentage_dup_read =
(cast(total_dup_read_count, 'double') * 100)/ seq_len;

%%
%%
```

```
tElapsed_stats = toc(tStart_stats);
tElapsed = toc(tStart);
```

**//// End of Program : SortedDuplicateReads.m ////**

**FindZeroIndex.m**

**/// Beginning of Program: FindZeroIndex.m ///**

```
/* Matlab Function to find Indices of Zeroes in the given
signal.

This function finds the positions or indices of zero
values in a given signal, so that it can be used to
detect Short Tandem Repeat Regions in a given DNA
sequence.  */


%% Function to find indices of zeros in a given signal %%


function [findzeroindx  d_rle] = findzeroindex( signal,
thresholdval )

    extendsignal = [1, signal, 1];

    convextendsignal          =          conv(extendsignal,
ones(1,thresholdval));

    tempsignal = double(convextendsignal == 0);

    tempconv = conv(tempsignal, ones(1,thresholdval) );

    findzeroindx = find(tempconv) - thresholdval;

    d_rle = runlengthencoding(tempsignal == 0);

end;
```

**///// End of Program: FindZeroIndex.m /////**