Data Analysis and Modelling

—

A Practical Guide to Fourier Data Techniques

Joseph Pritchard - Adelaide University

April 11, 2023

Introduction

MATLAB software is used to analyse the benefits of various data manipulation techniques in inter-space mapping. A sample data set of electronic driving signal and corresponding response is provided for analysis. Techniques compared are those of data averaging (interleaved vs. adjacent) and windowing (Gauss, Hann and Blackmann Harris). The effects of data truncation are also displayed for comparison.

Determination of Sample Rate Given Driving Data Signal

Note the following technique is designed requiring the input signal be a single frequency signal over the data points provided. It is however sufficient every successive data point equal in magnitude to the first in the set label a half period.

MATLAB software records the magnitude of the first data point in the input file. Data is then scanned for succesive data points of the same magnitude and used to label the periodicity of the data. A desired number of cycles to average over may be specified by the user. In the given sample it is found 16 cycles provides sufficient accuracy to exactly determine the frequency composition of the 145Hz driving signal [2]. The calculated sample rate is $4.4116 \times 10^4 Hz$.

FFT without Adjustment

As a control, an FFT is computed directly on the supplied output data using the calculated sample rate of 145kHz (Figure 1). Maximum frequency reliently detectable is provided by the Nyquist condition:

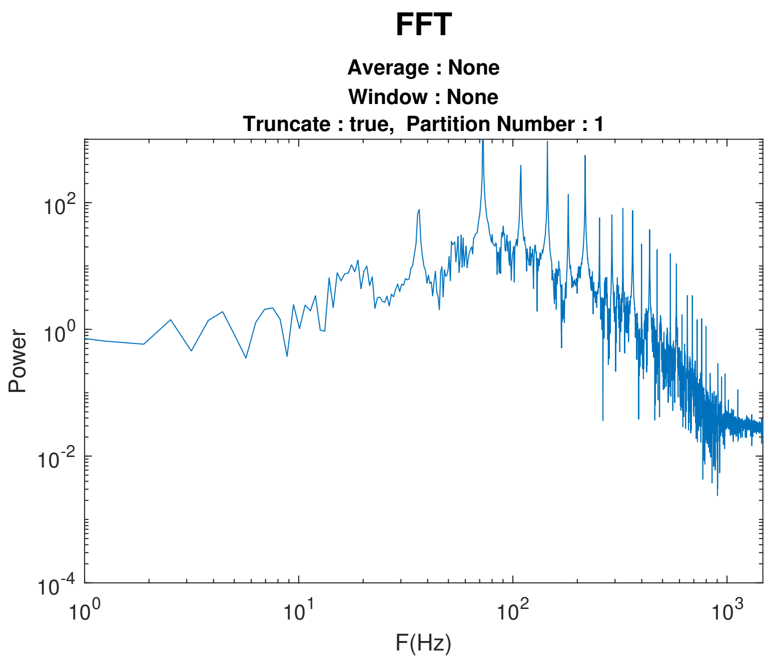
$$f_{Nyquist} = \frac{f_{Sampling}}{2}$$

where $f_{Sampling}$ is the sampling frequency. In the given data this results in a Nyquist frequency of 2.2058×10^4, at which we terminate the frequency axis.

A lower frequency limit is given by the smallest wave able to fit within the total sample window, in our case $1.587s$. Thus

$$f_{Min} = \frac{1}{T_{Max}} = \frac{1}{1.587s} = 0.6302Hz$$

Note the time space data is truncated to a factor of 2 before the FFT is computed. This is to negate the adverse effects of FFT zero-padding incurred when the time signal is not a multiple of 2 [1]. For the effects of direct computation see appendix plots.



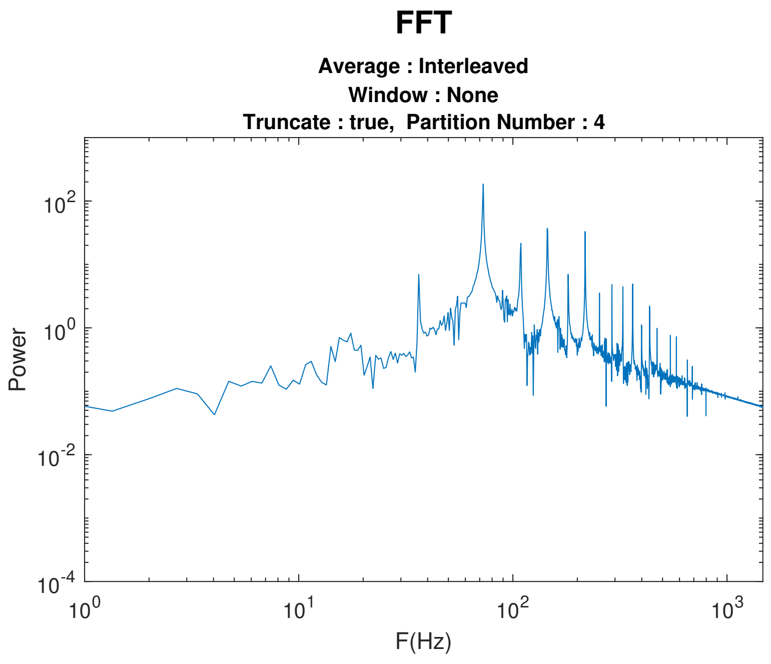
(a) Figure 1

FFT Averaging Strategies

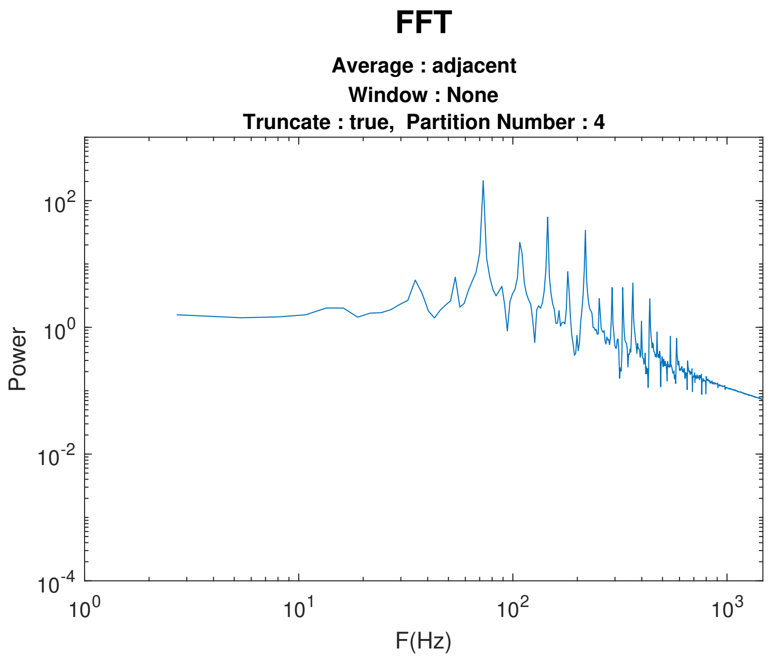
Interleaving and adjacent slicing provide different averaging properties. Interleaving has the effect of creating a sample set with greater sampling period per sample than the original data, decreasing point separation in frequency space, and reducing the maximum frequency detectable by the Nyquist theorem (Figure 2).

The provided plot features a partition number of 4 (4 sample sets), reducing the Nyquist frequency by a factor of 4 to $5.515 \times 10^3 Hz$. A lower limit is provided by the smallest frequency able to fit in total sample window, in this case $\frac{1}{T_{Window}} = 0.6302Hz$.

Adjacent slicing provides a somewhat opposite effect, applying a narrower square pulse window per sample within the set. This has the effect of increasing leakage around frequency peaks while suppressing leakage far from the peaks. Sampling and thus Nyquist frequency is unaffected. Minimum frequency detectable is increased by a factor of 4 due to the narrowed sample duration to $\frac{1}{T_{Window}} = 2.5209Hz$ (Figure 3).



(a) Figure 2

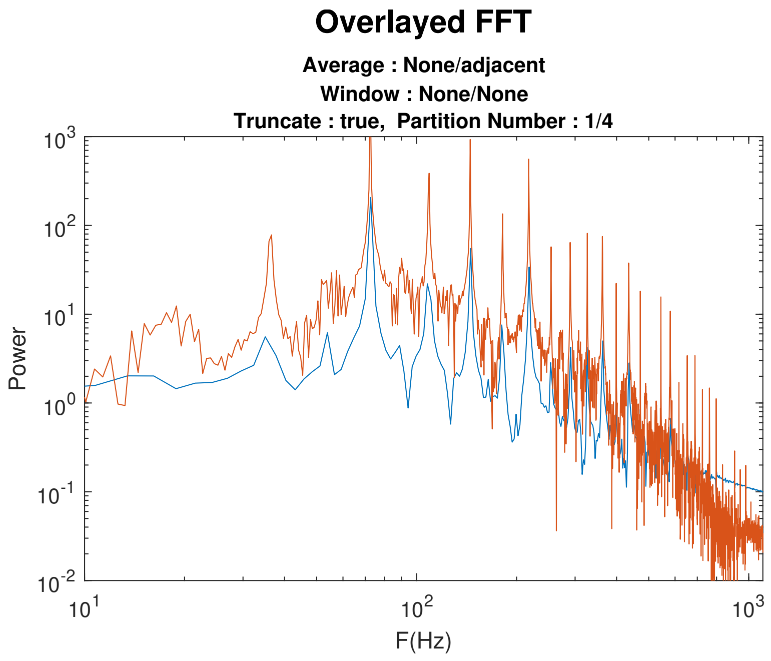


(a) Figure 3

Power Variance

Importance of averaging is gleamed easily when raw data is overlayed with the same data having been placed through an adjacent slicing averaging process (Figure 7). A consistent smoothing of the spectrum is observed across the total frequency range, having the effect to diminish the strength of the dominant peaks but also remove the base level of noise detected.

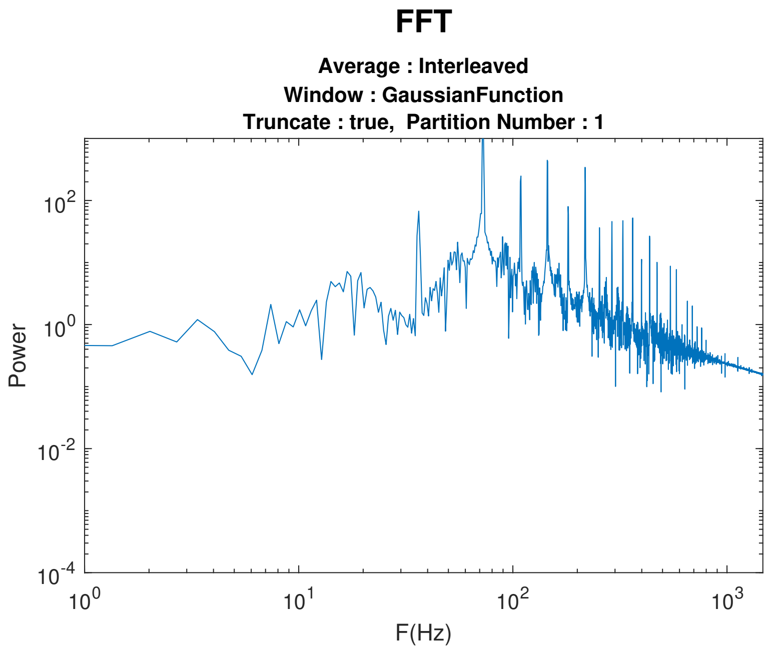
By analysing the variance in averaged and raw data, we are able to immediately discard many erratic features present in the raw frequency spectrum and deem them spurious.



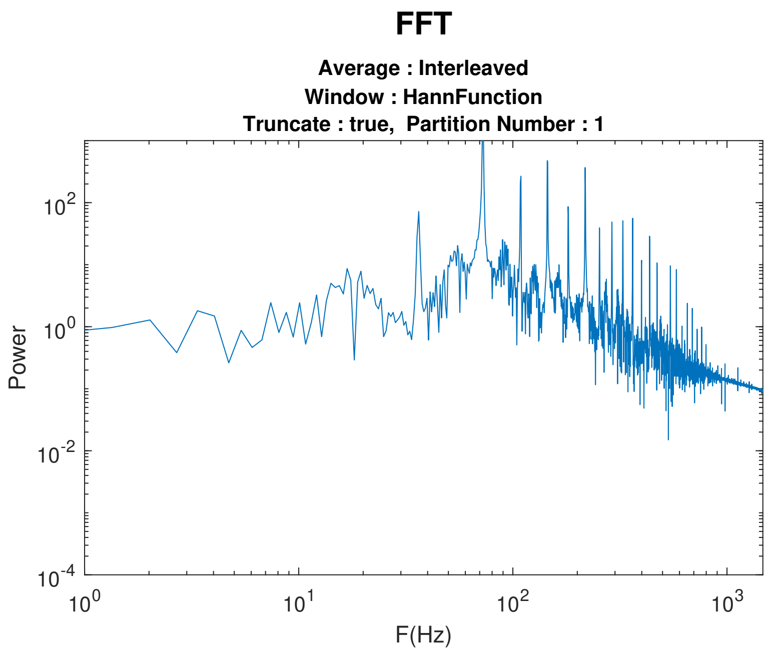
(a) Figure 7, Red: Raw Data, Blue: Adjacent Slice Data

FFT Windowing

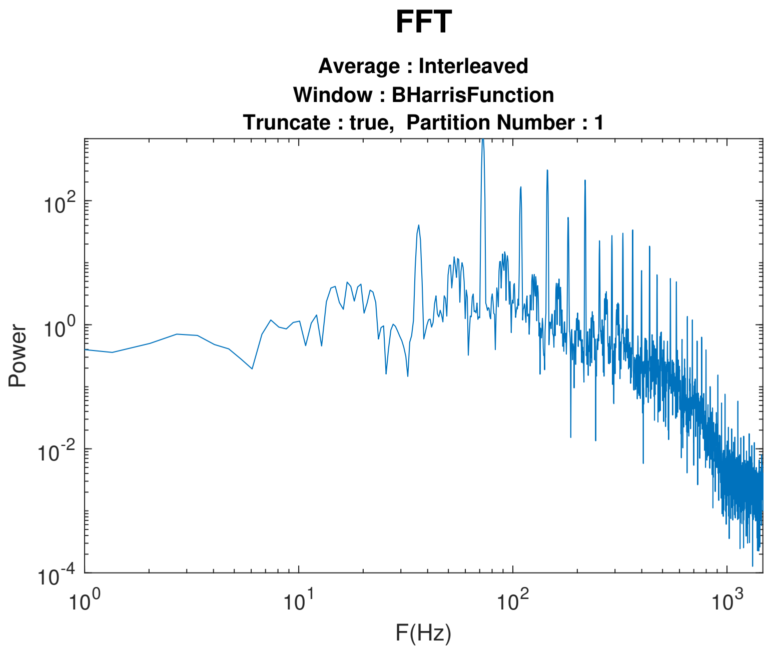
Application of a data window limits leakage attributable to the square pulse nature of a finite data set. A common (though not ubiquitous) trade-off occurs between central peak broadening/narrowing (CPB/CPN) and side peak amplification/suppression (SPA/SPS) [1]. Illustrating this trade-off are three window functions (Gaussian: Figure 4, Hann: Figure 5, Blackmann Harris: Figure 6) with increasing CPB and SPS in the displayed order. The source of this corellation comes from area (power) conservation in the respective fourier space domains [1][3].



(a) Figure 4



(a) Figure 5



(a) Figure 6

Window Advantage and Artifact Comparison

CPB and SPA are consistently visible in a number of areas in the presented windowed data (Figures 1, 2, 3). A progressive drop in amplitude of high frequency components from over 10^0 to below 10^{-2} signifies suppression of peaks far from the dominant values.

A progressive increase in peak width at the first dominant frequency peak around $10^{0.5} Hz$ indicates CPB.

Both features are exhibited in the second dominant peak around $10^{0.9} Hz$. We witness a progressive roundening of the peak tip, indicating CPB, and a consistent narrowing of the peak base width, indicating SPS.

Note that sample rate, total sample time duration and number of data points is unaffected by window application and by the theory discussed previously the minimum and maximum frequencies reliently detectable are unaffected. These remain the same as in the original unadjusted spectrum.

References

[1] Rowell, G 2020, *Data Analysis and Modelling Course Notes*, Data Analysis and Modelling, University of Adelaide

[2] Rowell, G 2020, *Data Analysis and Modelling Assignment 2 Guidelines*, Data Analysis and Modelling, University of Adelaide

[3] National Instruments, *Understanding FFTs and windowing*, National Instruments, viewed 26 October 2019, <http://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf>

Appendix

Included is the MATLAB code used to process, analyse and plot all results included in this report. Following are excess plots produced. These illustrate the effects not truncating the time space data before applying an FFT.

Table of Contents

```
%Note: Code requires input is a pure sine wave
%Note: Method for calculating sampling rate becomes more
%       accurate as # of cross overs is increased, then
%       time divided to give one period
%Note: Averaging is performed on the vector labelled as output by the
 user

close all;
clear all;
```

USER INPUTS

```
%Specify input file
data = importdata('BBChaos145Hz_1.txt');
input_Hz = 145;

%Specify sample_rate estimate properties
number_of_periods_to_average = 16;

%Set the number of partitions to separate the output data into for
%averaging
partition_number = 1;

%Choose wheter to truncate data to 2^N in size before applying fft
truncate = true;


%Specify Window functions
Hann_function.function = @(x, N) (sin(pi.*x./N));
Hann_function.name = "HannFunction";

%B_Harris - generalization of Hamming family
a0 = 0.35875;
a1 = 0.48829;
a2 = 0.14128;
a3 = 0.01168;
B_Harris_function.function = @(x, N) (a0 - a1*cos(2*pi.*x./N) +
 a2*cos(4*pi.*x./N) - a3*cos(6*pi.*x./N));
B_Harris_function.name = "BHarrisFunction";

%Gaussian
d = 0.5; % d <=0.5
```

```
Gaussian_function.function = @(x, N) (exp(-(1/2)*((x-N/2)/
(d*N/2))^2));
Gaussian_function.name = "GaussianFunction";
```

SOFTWARE ANALYSIS

```
%Define i/p and o/p
raw_input = data(:,1);
raw_output = data(:,2);

output_struct.data = raw_output;

%Reset data properties, EG: average, truncate, windowed (y/n for each)
output_struct = reset_labels(output_struct);

%Set input period
input_period = 1/input_Hz;

%Determine sample rate
%Method: Count 2 crossings of the initial input value
sample_rate = Calculate_sample_rate(raw_input, input_period,
 number_of_periods_to_average);


%Split data into N partitions
points_per_partition = Points_per_partition(raw_output,
 partition_number);
```

PLOTTING

```
%No Average No Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Fft(output_struct, truncate);
output_struct.sample_rate = sample_rate;
%Plot_fourier_transform(sample_rate, truncate, output_struct)

Overlay1 = output_struct;


%Adjacent Average No Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Array_to_matrix_adjacent(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)

Overlay2 = output_struct

%Interleaved Average No Window
output_struct = reset_labels(output_struct);
```

```
output_struct.data = raw_output;
output_struct = Array_to_matrix_interleaved(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)

%Adjacent Average Hann Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Array_to_matrix_adjacent(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Window_matrix_rows(output_struct, Hann_function);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)

%Adjacent Average B Harris Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Array_to_matrix_adjacent(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Window_matrix_rows(output_struct, B_Harris_function);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)

%Adjacent Average Gaussian Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Array_to_matrix_adjacent(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Window_matrix_rows(output_struct, Gaussian_function);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)

%Interleaved Average Hann Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Array_to_matrix_interleaved(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Window_matrix_rows(output_struct, Hann_function);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)

%Interleaved Average B Harris Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Array_to_matrix_interleaved(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Window_matrix_rows(output_struct, B_Harris_function);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)

%Interleaved Average Gaussian Window
output_struct = reset_labels(output_struct);
output_struct.data = raw_output;
output_struct = Array_to_matrix_interleaved(output_struct,
 points_per_partition, partition_number, sample_rate);
output_struct = Window_matrix_rows(output_struct, Gaussian_function);
output_struct = Fft(output_struct, truncate);
output_struct = Average_matrix_rows(output_struct);
%Plot_fourier_transform(sample_rate, truncate, output_struct)


%Plot Overlays
%Plot_fourier_transform(sample_rate, truncate, Overlay1, Overlay2)


Overlay2 =

  struct with fields:

              data: [1×65536 double]
           average: "adjacent"
            window: "None"
          truncate: "None"
  partition_number: 1
       sample_rate: 4.4125e+04
```

FUNCTIONS

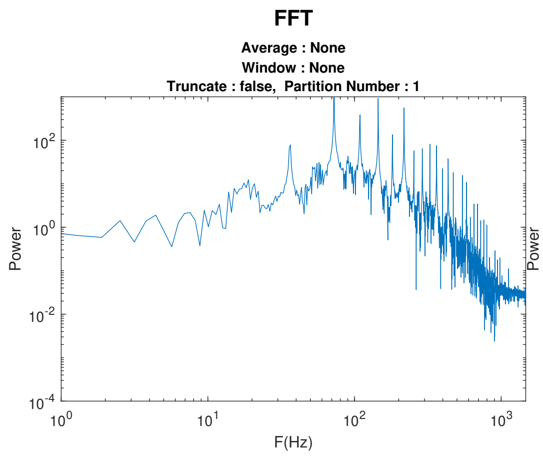
```
%Function calculating if b < a, if a_low
function result = Crossover(a, b, a_low)
    if(a_low)
        result = b < a;
    else
        result = b > a;
    end
end


%If one data_struct supplie plots data_struct.data, saves result
 as .eps and .jpg for viewing
%If two data_structs supplied overlays both on the same plot, saves
 result as .eps and .jpg for viewing
function result = Plot_fourier_transform(sample_rate, truncate,
 data_struct1, data_struct2)


    if (nargin == 3)

        FIG = figure
        samples = length(data_struct1.data);
        f = linspace(0, (data_struct1.sample_rate), samples);
        loglog(f, data_struct1.data);

        heading = {'\fontsize{15pt}FFT','\fontsize{10pt}Average : ' +
data_struct1.average, "Window : " + data_struct1.window, "Truncate :
" + truncate...
            + ",  Partition Number : " +
data_struct1.partition_number};
        title(heading)
        xlabel("F(Hz)")
        ylabel("Power")
        axis([1 sample_rate/30 0.0001 10^3])

        FILENAME = data_struct1.average + "_" + data_struct1.window
+ "_" + truncate + "_" + data_struct1.partition_number;
        saveas(FIG, FILENAME, 'jpg');
        saveas(FIG, FILENAME, 'epsc');
        result = FIG;


    elseif(nargin == 5)
        FIG = figure


        samples = length(data_struct2.data);
        f = linspace(0, (data_struct2.sample_rate), samples);
        loglog(f, data_struct2.data);
                hold on

        samples = length(data_struct1.data);
        f = linspace(0, (data_struct1.sample_rate), samples);
        loglog(f, data_struct1.data);



        heading = {'\fontsize{15pt}Overlayed
FFT','\fontsize{10pt}Average : ' + data_struct1.average + "/" +
data_struct2.average,...
            "Window : " + data_struct1.window + "/" +
data_struct2.window, "Truncate : " + truncate + ",
Partition Number : " + data_struct1.partition_number + "/" +
data_struct2.partition_number};
        title(heading)
        xlabel("F(Hz)")
        ylabel("Power")
        axis([10 sample_rate/40 0.01 10^3])
        %xlim([0 sample_rate/2])
        %ylim([0 10^3])
                hold off

        FILENAME = "Overlay_" + data_struct1.average
+ "_" + data_struct1.window + "_" + truncate + "_" +
data_struct1.partition_number;
```

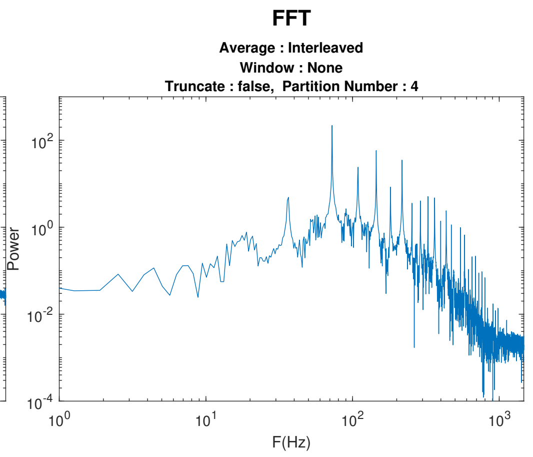
```
        %saveas(FIG, FILENAME, 'jpg');
        saveas(FIG, FILENAME, 'epsc');
        result = FIG;

    end
end

%returns sampling period of data, averaged over a number periods
function result = Calculate_sample_rate(data, input_period,
 number_of_periods_to_average)

    points_per_period = Calculate_points_per_period(data,
 number_of_periods_to_average);

    %Sample rate
    sampling_period = input_period/points_per_period;
    result = 1/sampling_period;
end

%Plots Time Domain of a data sample
function Plot_time_domain(data, sample_rate, heading)
    sampling_period = 1/sample_rate;
    total_time = length(data) * sampling_period;
    disp("asdf: " + total_time);
    samples = length(data);
    t = linspace(0,total_time,samples);

    figure
    plot(t, data);
    %tix=get(gca,'xtick')';
    %set(gca,'xticklabel',num2str(tix*total_time,'%.4f'))
    title(heading);
end

%Returns the number of data points per period data, averaged of num of
 per.
function result = Calculate_points_per_period(data,
 number_of_periods_to_average)
    cross_over_counter = 0;
    extremum = data(1);
    samples = length(data);
    if(data(2) > data(1))
        extremum_is_minimum = true;
    else
        extremum_is_minimum = false;
    end
    for i=2:1:samples
        if(Crossover(extremum, data(i), extremum_is_minimum))
            cross_over_counter = cross_over_counter + 1;
            extremum_is_minimum = not(extremum_is_minimum);
            extremum = data(i);
        end
        if(cross_over_counter == 2*number_of_periods_to_average)
            result = i/number_of_periods_to_average;
```

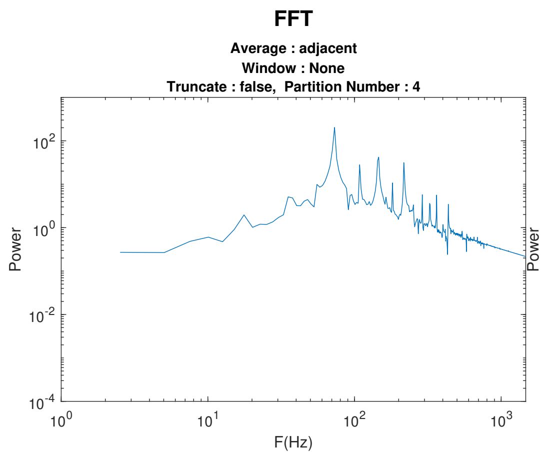
```
                break
            end
        end
    end

    function result = Points_per_partition(data, partition_number)
        total_size = length(data);
        result = floor(total_size/partition_number);
    end

    %Returns an MxN matrix, M = slice_number, N = points_per_slice
    %Data point spacing is sample period
    function result = Array_to_matrix_adjacent(data_struct,
     points_per_slice, slice_number, sample_rate)
        matrix = zeros(slice_number,points_per_slice);
        for i=1:slice_number:1
            start = (i-1)*points_per_slice + 1;
            finish = i*points_per_slice;
            matrix(i,:) = data_struct.data(start:finish);
        end

        data_struct.data = matrix;
        data_struct.average = "adjacent";
        data_struct.partition_number = slice_number;
        data_struct.sample_rate = sample_rate;
        result = data_struct;
    end

    %Returns an MxN matrix, M = slice_number, N = points_per_slice
    %Data point spacing is slice_number * sample period
    function result = Array_to_matrix_interleaved(data_struct,
     points_per_slice, slice_number, sample_rate)
        matrix = zeros(slice_number,points_per_slice);
        for i=1:1:slice_number

            for j=1:1:points_per_slice

                step_size = slice_number;

                %Stagger points at interval step_size, offset by
     slice_number
                matrix(i,j) = data_struct.data( j * step_size + i -
     step_size);
            end
        end

        data_struct.data = matrix;
        data_struct.average = "Interleaved";
        data_struct.partition_number = slice_number;
        data_struct.sample_rate = sample_rate / slice_number;
        result = data_struct;
    end

    %Given MxN matrix returns 1xN collum vector, average of M rows
```

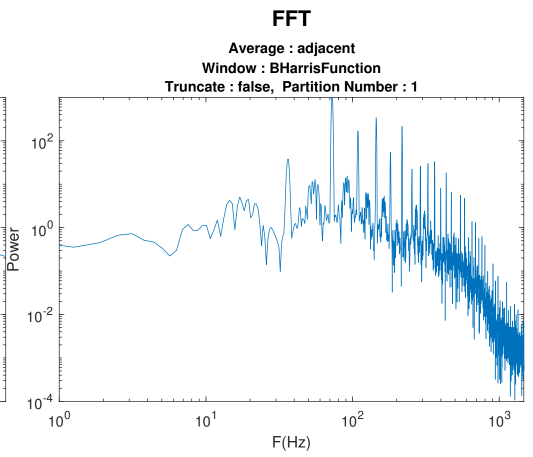
```
function result = Average_matrix_rows(data_struct)
    rows = length(data_struct.data(:,1));
    cols = length(data_struct.data(1,:));

    average = zeros(1,cols);

    for i=1:rows:1
       average = average + data_struct.data(i,:);
    end

    data_struct.data = average;
    result = data_struct;
end

%Given MxN input data, return an MxN matrix with 'window_function'
 applied
%per row
%Window function form f(a,b)
%a == current data point
%b == total number of data points
function result = Window_matrix_rows(data_struct, window_function)

    rows = length(data_struct.data(:,1));
    cols = length(data_struct.data(1,:));

    windowed = zeros(rows, cols);

    for i=1:1:rows
        for j=1:1:cols
            windowed(i,j) =
 window_function.function(j,cols)*data_struct.data(i,j);
        end
    end
    data_struct.data = windowed;


    data_struct.window = window_function.name;
    result = data_struct;
end

%Performs an FFT on the rows of MxN matrix data_struct.data
%Truncates to 2^N if truncate == true
function result = Fft(data_struct, truncate)
%Create power spectrum

    %Truncate data to power of 2 for FFT accuracy
    if(truncate)
        data_power = floor(log2(length(data_struct.data(1,:))));
        data_points = 2 ^ data_power;
        data_struct.data = data_struct.data(:,1:data_points);
    end

    %fft applied per collum hence take transpose
    data_struct.data = fft(data_struct.data');
    data_struct.data = data_struct.data';

    %Calculate Power
    data_struct.data = abs(data_struct.data);

    result = data_struct;

end

%Reset data labels
function result = reset_labels(data)
    data.average = "None";
    data.window = "None";
    data.truncate = "None";
    data.partition_number = "1";
    data.sample_rate = 1;
    result = data;
end


%So far Hanning window seems best choice as it significantly reduces
 signal
%presence at higher frequencies while alse limiting frequency
 bandwidth at
%low frequencies. G Harris window limits high w behaviour but
 maintains a
%wider bandwidth at lower w. Gaussian window narrows bandwidth the
 most but
%has large high w content.
```

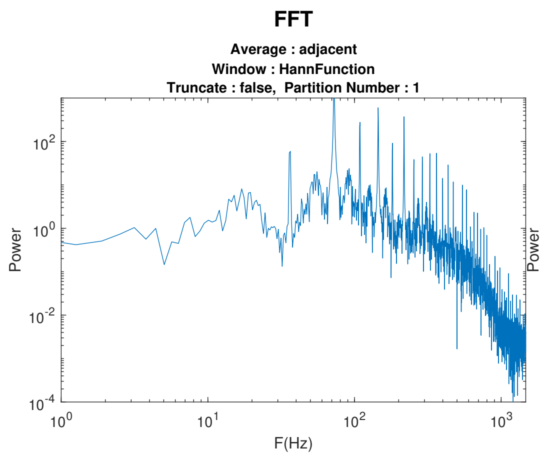
Published with MATLAB® R2018b

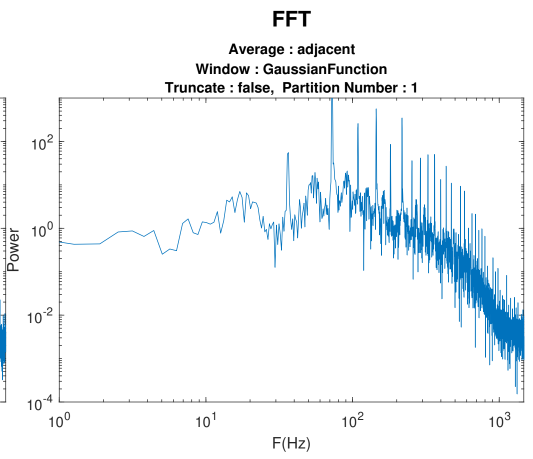

FFT
Average : None
Window : None
Truncate : false, Partition Number : 1
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{1}
10^{3}

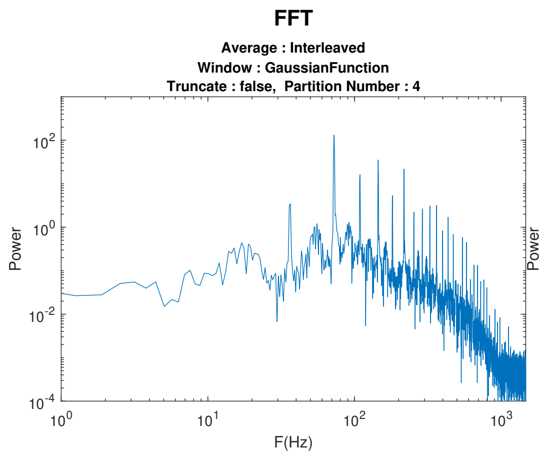

FFT
Average : Interleaved
Window : None
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{1}
10^{3}

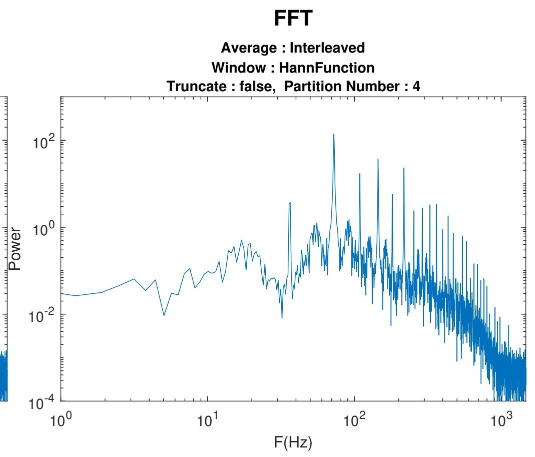

FFT
Average : adjacent
Window : None
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{1}
10^{3}

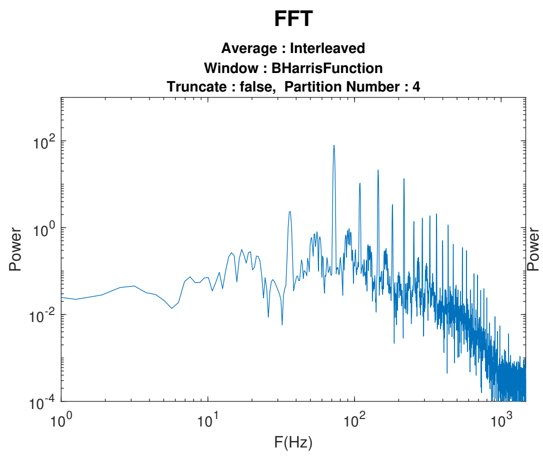

FFT
Average : adjacent
Window : BHarrisFunction
Truncate : false, Partition Number : 1
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{1}
10^{3}

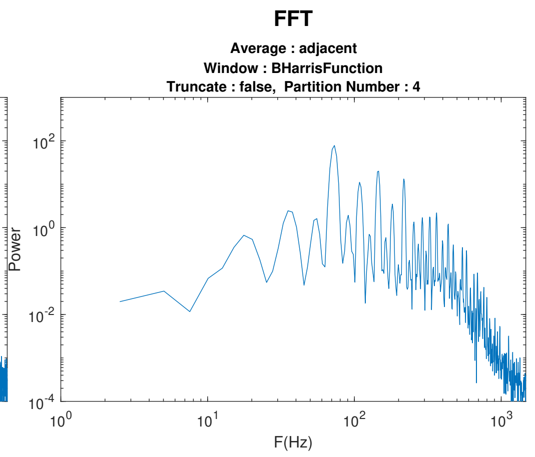

FFT
Average : adjacent
Window : HannFunction
Truncate : false, Partition Number : 1
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{1}
10^{3}

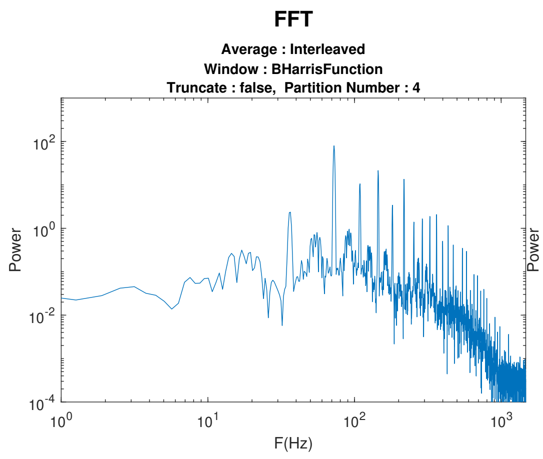

FFT
Average : adjacent
Window : GaussianFunction
Truncate : false, Partition Number : 1
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{1}
10^{3}

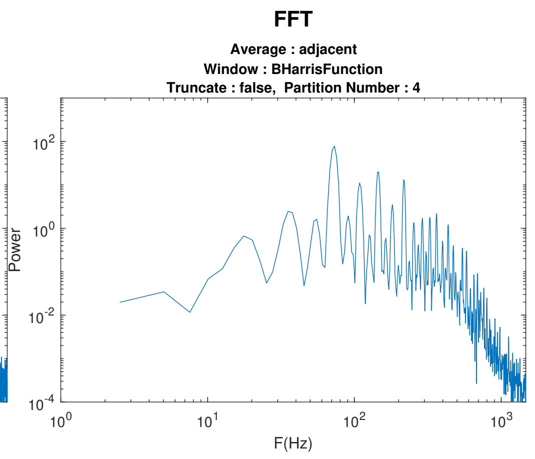

FFT
Average : Interleaved
Window : GaussianFunction
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}


FFT
Average : Interleaved
Window : HannFunction
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}


FFT
Average : Interleaved
Window : BHarrisFunction
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}


FFT
Average : adjacent
Window : BHarrisFunction
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}


FFT
Average : Interleaved
Window : BHarrisFunction
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}


FFT
Average : adjacent
Window : BHarrisFunction
Truncate : false, Partition Number : 4
Power
F(Hz)
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{0}
10^{1}
10^{2}
10^{3}