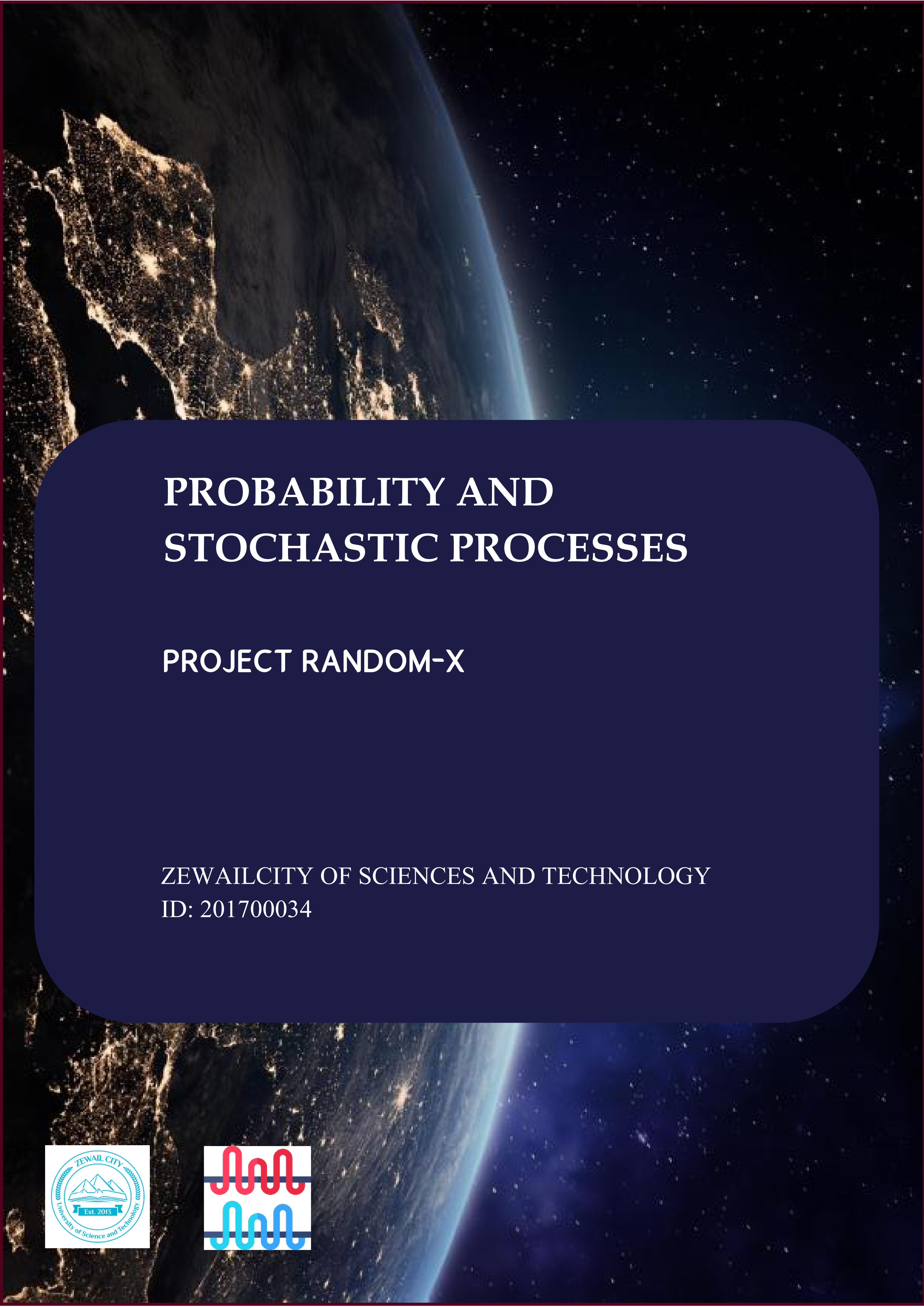


PROBABILITY AND STOCHASTIC PROCESSES

PROJECT RANDOM-X

ZEWAILCITY OF SCIENCES AND TECHNOLOGY
ID: 201700034

RANDOM-X APPLICATION

Dr. Samy S. Soliman, Eng. Amal Mohamed, Eng. MennatAllah Mohamed Hassan, Mustafa Elaraby.

⌘| Guidelines

❶ | Abstract.

This is a comprehensive report about RANDOM-X application. RANDOM-X is a gui-based tool that allows a user to enter any stochastic process and results in the ensemble and the time statistics of such process. It iterates through the random process, check number of repetition of every realization and calculates probabilities of the ensemble, with probabilities in pouch, it can calculate and plot the ensemble mean, with entering some specific two time instances by the user, it can calculate the statistical auto correlation function between the two samples, it also calculates the auto correlation function between all possible time instances, and plot them in a 3d plot, it calculates the time mean of the random process, the time auto correlation function, it also calculates and plots the time auto correlation function for all time differences. It uses the fast Fourier transform technique to evaluate the power spectral density of the process, and plots this psd, and finally it calculates the total average power of the random process. So it can be considered very powerful tool in the stochastic scope and specially signal processing scope.

❷ | Definitions and Basic Concepts.

Definition 1.1: Mean of Random Process.

Mean of Random process can be obtained from the first order PDF as:

$$\overline{x(t)} = \int_{-\infty}^{\infty} x(t)\, p(x;t)\, dx(t)$$

This results in the average of all samples at every time → the average wave form.

Definition 1.2: Statistical Auto Correlation Function.

It is measured by correlating the amplitudes of the signal at two distinct time instants t_i and t_j.

$$R_x(t_i, t_j) = \overline{x(t_i)x(t_j)} = \overline{x_i x_j} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x_i x_j\, p(x_i, x_j)\,dx_i\,dx_j$$

Definition 1.3: The Time average.

$$\widetilde{x(t)} = \lim_{T\to\infty}\frac{1}{T}\int_{-T/2}^{T/2} x(t,\xi_i)\ dt$$

Definition 1.4: The Time Auto Correlation Function.

$$\widetilde{R_x(\tau)} = x(t)\widetilde{x(t+\tau)} = \lim_{T\to\infty}\frac{1}{T}\int_{-T/2}^{T/2} x(t)\,x(t+\tau)\ dt$$

Definition 1.5: The Power Spectral Density.

$$S_x(\omega) = \lim_{T\to\infty}\left[\frac{\overline{|X_T(\omega)|^2}}{T}\right]$$

Definition 1.6: The Total Average Power.

$$P_x = \overline{x^2(t)} = R_x(0) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_x(\omega)\,d\omega$$

❸ | Documentation.

❁ Properties.

Property	Documentation
fileName	file name of the input random process.
M	Number of sample functions to be plotted.
N	index of sample function.
ti	index of sample i.
tj	index of sample j.
time	time array.
RandomProcess	D array of the Random process.
RPS	Random process size.
Ensemble	2D array of the Ensemble.
ES	Ensemble size.
probabilities	probabilities of every sample func.
EnsembleMean	Ensemble Mean.
ACF	Auto corr func between ith and jth samples.
All_ACF	Auto corr func between all samples.
TimeMean	Time Mean of the nth sample func.
TimeACF	Time auto correlation function.
all_TimeACF	Vector of Time ACF for all possible Taus.
PSD	Power spectral density.
ALL_PSD	PSD between all samples.

❁ Functions.

Function	Documentation
getFile(app)	Open a file dialog so that the user can browse and choose a random process file stored any where in the local storage.
readFile(app,file)	Dereference the struct sample process file, and read data of the every variable in it.
getProbabilities(app)	Calculates probability of every sample function and store it in a vector.
plot_M_Sample_Functions(app)	Allow the user to enter a number M, and it plots the first M sample functions of the ensemble.
plotEnsembleMean(app)	Get the average wave form of the ensemble and plots it.
getACF(app)	Calculates the time auto correlation function between the ith and jth samples.

plotACF3D(app)	Calculates the time auto correlation function between all the possible ith and jth samples.
getTimeMean(app)	Calculates the time mean of the random process.
getTimeACF(app)	Calculates the Time ACF for a specific Tau.
plotAll_TimeACF(app)	Calculates and plots the time ACF for all Taus.
plotPSD(app)	Calculated and plot power spectral density.
etInitialStates(app)	Return every object in the app to initial states.
clear(app)	Return every object in the app to initial states, and clears the memory of the application.

❹ | Codes.

❁ Properties.

```
properties (Access = private)
    fileName            % file name of the input random process;
    M                   % Number of sample functions to be plotted.
    N                   % index of sample function.
    ti                  % index of sample i;
    tj                  % index of sample j.
    time                % time array;
    RandomProcess       % 2D array of the Random process.
    RPS                 % Random process size;
    Ensemble            % 2D array of the Ensemble.
    ES                  % Ensemble size.
    probabilities       % probabilities of every sample func.
    EnsembleMean        % Ensemble Mean.
    ACF                 % Auto corr func between ith and jth samples.
    All_ACF             % Auto corr func between all samples.
    TimeMean            % Time Mean of the nth sample func.
    TimeACF             % Time auto correlation function.
    all_TimeACF         % Vector of Time ACF for all possible Taus.
    PSD                 % Power spectral density.
    ALL_PSD             % PSD between all samples.
    Avg_power           % Total average power.
    TotalAveragePower   % Average power of the ensemble.

End
```

❁ Functions.

```
%-----------------------> DISPLAY CHOOSE FILE DIALOG. <--------------------


        function getFile(app)
            [file,path] = uigetfile(".mat");
            app.fileName = string(path)+string(file);
        end


%==========================================================================

%-------------------> READ THE FILE ENTERED BY THE USER. <-----------------

        function readFile(app,file)

%           check if file exist or valid.
            if exist(file,"file")

%               store the file in a variable.
                Input = load(file);

%               convert from struct type to cell type to allow dynamic
%               acess.
                Input= struct2cell(Input);

%               get the Random process 2d array.
                app.RandomProcess = Input{1};
                app.time = Input{2};

%               get the time vector.
                app.Lamp.Color = "green";
            else
%               if file doesn't exist tell the user to enter valid file.
                app.Lamp.Color = "red"; v
                app.HTML.HTMLSource="<html><head><style>p {color:red;font-
family:arial}</style></head><body><p>Enter a valid filename!</p></body></html>";
                app.InputFileName.FontColor = "red";
            end

        end

%==========================================================================

%-------------> CALCULATE PROBABILITIES OF SAMPLE FUNCTIONS. <-------------

        function getProbabilities(app)
%           check if random process matrix and time vecror is not empty.
            if(~isempty(app.RandomProcess) && ~isempty(app.time))

%               get the random process size.
                app.RPS = size(app.RandomProcess);

%               get the Ensemble of the random process.
                app.Ensemble = unique(app.RandomProcess,"rows");
```

```
%               get the size of the Ensemble.
                app.ES = size(app.Ensemble);

%               initiate an array to calculate number of repititions of
%               every sample function.
                C=zeros([app.ES(1)],1);
                for Ni=1:app.ES(1)
                    for Nj=1:app.RPS(1)
                        if(app.Ensemble(Ni,:) == app.RandomProcess(Nj,:))
                            C(Ni) = C(Ni)+1;
                        end
                    end
                end

%               devide by total number of sample func to get probabilities.
                C = C/app.RPS(1);

%               store the probability vector.
                app.probabilities = C;
            end
        end


%===========================================================================

%------------------------> PLOT M SAMPLE FUCTIONS  <-----------------------

        function plot_M_Sample_Functions(app)

%           check if Ensemble matrix and time vecror is not empty.
            if(~isempty(app.Ensemble) && ~isempty(app.time))

%               check if M is valid input.
                if(app.M > 0 && app.M <= app.ES(1))

%                   initiate temp sample function.
                    toPlot = zeros(app.M,[length(app.time)]);

%                   plot every sample function up to M.
                    for k=1:app.M
                        toPlot(k,:) = app.Ensemble(k,:);
                        plot(app.EnsemblePlot,app.time,toPlot)
                    end

%               if the user enters invalid M plot all Ensemble.
                elseif (app.M < 0 ||app.M > app.ES(1))
                    plot(app.EnsemblePlot,app.time,app.Ensemble);
                    app.Lamp.Color = "#20B2AA";
                    app.HTML.HTMLSource="<html><head><style>p
{color:LightSeaGreen;font-family:arial}</style></head><body><p>Recieved Invalid Value
for M. All Ensembl Plotted.</p></body></html>";
                    app.InputFileName.FontColor = "#20B2AA";

%               if the user enters invalid M plot all Ensemble.
                else
                    plot(app.EnsemblePlot,app.time,app.Ensemble);
                    app.Lamp.Color = "#20B2AA";
                    app.HTML.HTMLSource="<html><head><style>p
{color:LightSeaGreen;font-family:arial}</style></head><body><p>Recieved no input for M.
so, all ensemble plotted.</p></body></html>";
                    app.InputFileName.FontColor = "#20B2AA";
                end
            end
        end

%==========================================================================

%-------------------------> PLOT ENSEMBLE MEAN  <-------------------------

        function plotEnsembleMean(app)

%           check if Ensemble matrix and time vecror is not empty.
            if(~isempty(app.Ensemble) && ~isempty(app.time))

%               initiate a matrix to multiply Ensemble by probabilities.
                FRB_P = zeros(app.ES);
                for k=1:app.ES(1)
                    FRB_P(k,:) = app.Ensemble(k,:)*app.probabilities(k);
                end

%               get the average of the ensemble.
                app.EnsembleMean = sum(FRB_P);

%               plot the Ensemble mean.
                plot(app.EnsambleMeanPlot,app.time,app.EnsembleMean);

%              generate 3 random integers between 1 ans width of ensemble.
               if(height(app.Ensemble)>3)
                   rnd = randperm(app.ES(1),3);
    %              plot the three random sample functions.
                   plot(app.SampleFunction1plot,app.time,app.Ensemble(rnd(1),:))
                   plot(app.SampleFunction2plot,app.time,app.Ensemble(rnd(2),:))
                   plot(app.SampleFunction3plot,app.time,app.Ensemble(rnd(3),:))
               elseif(height(app.Ensemble)==2)
                   rnd = randperm(app.ES(1),2);
    %              plot the three random sample functions.
                   plot(app.SampleFunction1plot,app.time,app.Ensemble(rnd(1),:))
                   plot(app.SampleFunction2plot,app.time,app.Ensemble(rnd(2),:))
               else
                   plot(app.SampleFunction1plot,app.time,app.Ensemble(1,:))
               end
            end
        end


%==========================================================================

%------------------------> GET AUTO CORREL FUNC. <------------------------

        function getACF(app)

%           check if Ensemble matrix and time vecror is not empty.
            if(~isempty(app.Ensemble) && ~isempty(app.time))
%               check if tau is valid.
                if(app.ti > 0 && app.tj > 0 && app.ti <= length(app.time) && app.tj <=
length(app.time))
```

```
%                    get sample at ti.
                     Si = app.Ensemble(:,app.ti);

%                    get sample at tj.
                     Sj = app.Ensemble(:,app.tj);

%                    calc ACF.
                     SiSj = Si.*Sj;
                     SiSjP = SiSj.*app.probabilities;
                     tempACF = sum(SiSjP);
                     app.ACF = tempACF;

%                    display ACF value.
                     app.ACF_Display.Value = app.ACF;
                     app.ACF_Display.FontColor = [0 0 0];
                 end
             end
         end


%========================================================================

%------------------------> GET AUTO CORREL FUNC. <------------------------

         function plotACF3D(app)

%            check if Ensemble matrix and time vecror is not empty.
             if(~isempty(app.Ensemble) && ~isempty(app.time))

%                calc ACF.
                 temp_ACF = zeros(app.ES(2));
                 for k=1:app.ES(2)
                     for l=1:app.ES(2)
                         Si = app.Ensemble(:,k);
                         Sj = app.Ensemble(:,l);
                         SiSj = Si.*Sj;
                         SiSjP=SiSj.*app.probabilities;
                         temp_ACF(k,l) = sum(SiSjP);
                     end
                 end
                 app.All_ACF = temp_ACF;

%                x-axis linespace.
                 x = 1:1:width(app.time);

%                y-axis linespace.
                 y = 1:1:width(app.time);

%                3D plot of acf using mesh.
                 mesh(app.ACF3D,x,y,app.All_ACF,XData=x,YData=y);
                 colorbar(app.ACF3D,"east");
                 xlabel(app.ACF3D,"i","Color","red",FontSize=25,FontWeight="bold")
                 ylabel(app.ACF3D,"j","Color","red",FontSize=25,FontWeight="bold")

             end
         end


%========================================================================
%---------------------------> GET TIME MEAN. <----------------------------

         function getTimeMean(app)

%           check if Ensemble matrix and time vecror is not empty.
             if(~isempty(app.Ensemble) && ~isempty(app.time))

%               Check if N is valid.
                 if(app.N > 0 && app.N < length(app.time))
                     SampleFunc_N = app.Ensemble(app.N,:);

%                   integrate with respect to time.
                     integ = trapz(app.time,SampleFunc_N);

                     app.TimeMean = integ/(app.time(app.ES(2))-app.time(1));
                     app.TimeMean_Display.Value = app.TimeMean;
                     app.TimeMean_Display.FontColor = "black";
                 else
                     %
                 end
             end
         end
%==========================================================================

%----------------------> GET TIME AUTO CORREL FUNC. <---------------------

         function getTimeACF(app)

%           check if Ensemble matrix and time vecror is not empty.
             if(~isempty(app.Ensemble) && ~isempty(app.time))

%               Check if N is valid.
                 if(app.N > 0 && app.N < length(app.time))
                     if(abs(app.tj - app.ti)>=0 &&(app.tj - app.ti)<width(app.time))
                         SampleFunc_N = app.Ensemble(app.N,:);
                         Tau = abs(app.tj - app.ti) ;
                         SFS = width(SampleFunc_N)-Tau;
                         d = zeros(1,SFS);
                             for j=1:SFS
                                 d(j) = SampleFunc_N(j)*SampleFunc_N(j+Tau);
                             end
                             Ti = app.time(1);
                             Tf=app.time(width(app.time));
                             d_time = app.time(1:end-Tau);
                         if(width(d)>1)
                             app.TimeACF = trapz(d_time,d)/(Tf-Ti);
                         else
                             app.TimeACF = (d*Ti)/(Tf-Ti);
                         end
                         app.TimeACF_Dispaly.Value = app.TimeACF;
                         app.TimeACF_Dispaly.FontColor = "black";
                     end
                 end
             end
         end


%==========================================================================

%--------------------> PLOT ALL TIME AUTO CORREL FUNC. <------------------
```

```
        function plotAll_TimeACF(app)
            if(~isempty(app.Ensemble) && ~isempty(app.time))
                if(app.N > 0 && app.N < length(app.time))
                    TauMax = width(app.time) - 1;
                    SampleFunctionA = app.Ensemble(app.N,:);

                    TACF = zeros(1,TauMax);
                    for Tau=1:TauMax
                        size = width(SampleFunctionA) - Tau;
                            TACFA = zeros(1,size);
                        for i=1:size
                            TACFA(i) = SampleFunctionA(i)*SampleFunctionA(i+Tau);
                        end
                        Tf = app.time(width(app.time));
                        Ti = app.time(1);
                        d_time = app.time(app.time <= app.time(width(app.time)-Tau));
                        if(width(TACFA)>1)
                             integrated = trapz(d_time,TACFA)/(Tf-Ti);
                             TACF(Tau) = integrated;
                        else
                            TACF(Tau) = (TACFA*Ti)/(Tf-Ti);
                        end
                    end
                    app.all_TimeACF = TACF;
                    plot(app.TimeACF_Plot,app.all_TimeACF);
                end
            end
        end

%===========================================================================

%-----------------------> GET POWER SPECTRAL DENSITY <---------------------

        function plotPSD(app)
            fftEnsemble = zeros(app.ES);
            avg = zeros(app.ES);
            for k=1:height(app.Ensemble)
                fftEnsemble(k,:) =abs(fftshift(fft(app.Ensemble(k,:))));
            end

            fftEnsemble = fftEnsemble.^2;
            for i=1:height(app.Ensemble)
              avg(i,:) = fftEnsemble(i,:).*app.probabilities(i);
            end
            meanfft = sum(avg);
            Tf = max(app.time);
            Ti = min(app.time);
            psd = meanfft./(Tf - Ti);
            disp(psd);
            plot(app.PSD_plot,psd);
        end


%===========================================================================

%------------------------> GET TOTAL AVERAGE POWER <-----------------------

        function getTotalAvgPower(app)
            TAVP = zeros(1,width(app.Ensemble));
            for i=1:width(TAVP)
                TAVP(i) = bandpower(app.Ensemble(:,i));
            end
            app.Avg_power = TAVP;
            app.TotalAveragePower = sum(TAVP);
            plot(app.TAPDisplay,app.time,TAVP);
            app.T_avg_Pow_Display.Value = app.TotalAveragePower;
            app.T_avg_Pow_Display.FontColor = "black";
        end

%==========================================================================

%-----------------------> SET INITIAL STATES <----------------------------

        function setInitialStates(app)
            app.fileName = '';
            app.M = 0;
            app.N = 0;
            app.ti = 0;
            app.tj = 0;
            app.time = [];
            app.RandomProcess = [];
            app.RPS = [];
            app.Ensemble = [];
            app.ES = [];
            app.probabilities = [];
            app.EnsembleMean = [];
            app.ACF = [];
            app.All_ACF = [];
            app.TimeMean = [];
            app.PSD = [];
            app.ALL_PSD = [];
            app.ACF_Display.Value = 0;
            app.ACF_Display.FontColor = "white";
            app.TimeMean_Display.Value = 0;
            app.TimeMean_Display.FontColor = "white";
            app.TimeACF_Dispaly.Value = 0;
            app.TimeACF_Dispaly.FontColor = "white";
            app.T_avg_Pow_Display.Value = 0;
            app.T_avg_Pow_Display.FontColor = "white";
            app.InputFileName.FontColor = "black";
            app.Lamp.Color = [0.8 0.8 0.8];
            app.HTML.HTMLSource = "<p></p>";
            cla(app.EnsemblePlot);
            cla(app.EnsambleMeanPlot);
            cla(app.SampleFunction1plot);
            cla(app.SampleFunction2plot);
            cla(app.SampleFunction3plot);
            cla(app.ACF3D);
            cla(app.TimeACF_Plot);
            cla(app.PSD_plot);
            cla(app.TAPDisplay);
        end


%==========================================================================

%--------------------------------> CLEAR <--------------------------------

        function clear(app)
```

```
            app.InputFileName.Value = '';
            app.fileName = '';
            app.M_Input.Value = 0;
            app.N_Input.Value = 0;
            app.I_Input.Value = 0;
            app.J_Input.Value = 0;
            app.M = 0;
            app.N = 0;
            app.ti = 0;
            app.tj = 0;
            app.time = [];
            app.RandomProcess = [];
            app.RPS = [];
            app.Ensemble = [];
            app.ES = [];
            app.probabilities = [];
            app.EnsembleMean = [];
            app.ACF = [];
            app.All_ACF = [];
            app.TimeMean = [];
            app.PSD = [];
            app.ALL_PSD = [];
            app.ACF_Display.Value = 0;
            app.ACF_Display.FontColor = "white";
            app.TimeMean_Display.Value = 0;
            app.TimeMean_Display.FontColor = "white";
            app.TimeACF_Dispaly.Value = 0;
            app.TimeACF_Dispaly.FontColor = "white";
            app.T_avg_Pow_Display.Value = 0;
            app.T_avg_Pow_Display.FontColor = "white";
            app.InputFileName.FontColor = "black";
            app.Lamp.Color = [0.8 0.8 0.8];
            app.HTML.HTMLSource = "<p></p>";
            cla(app.EnsemblePlot);
            cla(app.EnsambleMeanPlot);
            cla(app.SampleFunction1plot);
            cla(app.SampleFunction2plot);
            cla(app.SampleFunction3plot);
            cla(app.ACF3D);
            cla(app.TimeACF_Plot);
            cla(app.PSD_plot);
            cla(app.TAPDisplay);
        end

    end
```

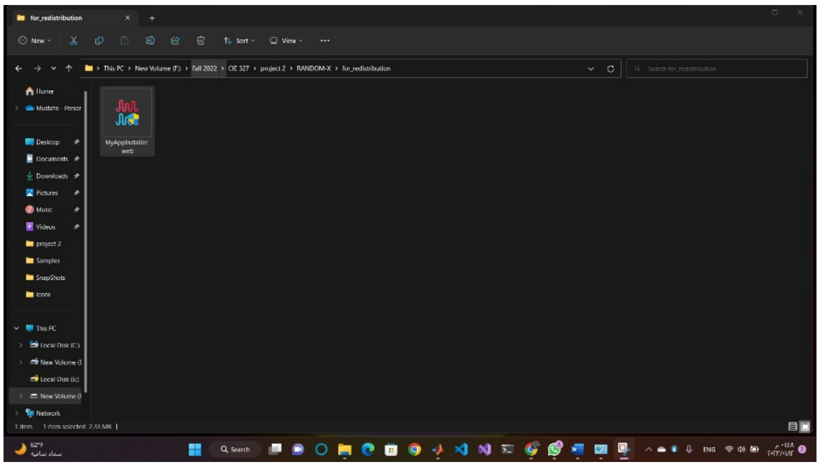
❁ Callbacks.

```
    % Callbacks that handle component events
    methods (Access = private)

        % Callback function
        function HTMLDataChanged(app, event)

        end

        % Image clicked function: Image
        function ImageClicked(app, event)
            app.getFile();
            app.InputFileName.Value = app.fileName;
            app.InputFileName.FontColor = [0 0 0];
            app.InputFileName.FontSize = 18;
        end

        % Button pushed function: RunButton
        function RunButtonPushed(app, event)
            app.setInitialStates();
            app.M = app.M_Input.Value;
            app.N = app.N_Input.Value;
            app.ti = app.I_Input.Value;
            app.tj = app.J_Input.Value;
            app.readFile(app.InputFileName.Value);
            app.getProbabilities();
            app.plot_M_Sample_Functions();
            app.plotEnsembleMean();
            app.plotACF3D();
            app.getTimeMean();
            app.getACF();
            app.getTimeACF();
            app.plotAll_TimeACF();
            app.plotPSD();
            app.getTotalAvgPower();
        end

        % Value changing function: InputFileName
        function InputFileNameValueChanging(app, event)
            app.InputFileName.FontColor = "black";
            app.HTML.HTMLSource = "<p></p>";
        end

        % Button pushed function: ClearButton
        function ClearButtonPushed(app, event)
            app.clear();
        end
    end
```

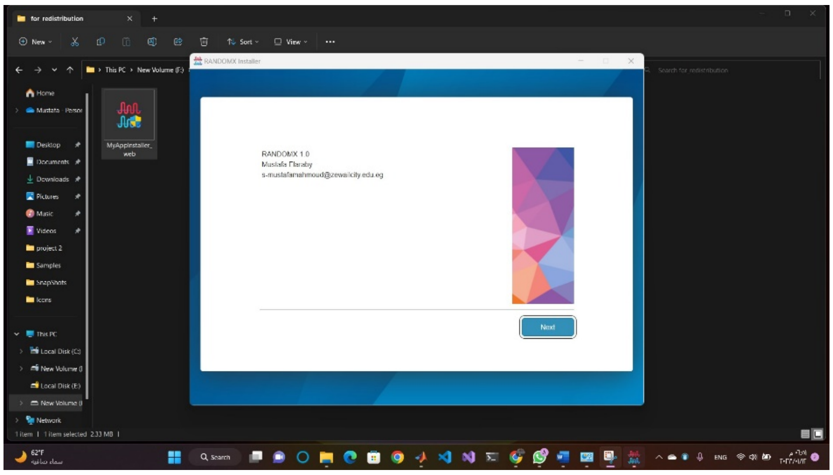
❺ | Setup and Run.

❁ Setup.

1. Right click in the TEXTlab-installer.exe and select open.

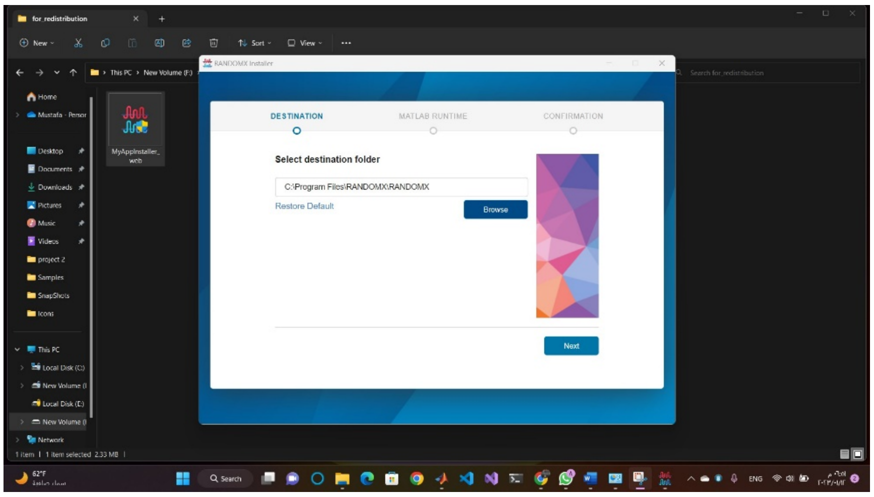


2. Select next.



RANDOMX 1.0
Mustafa Elaraby
s-mustafamahmoud@zewailcity.edu.eg

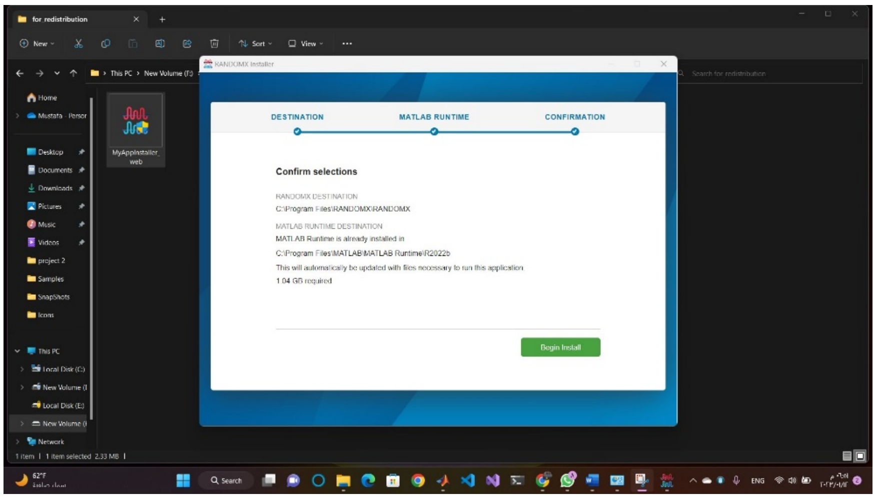
3. Choose the destination where you want setup your app, and select next.



Select destination folder

C:\Program Files\RANDOMX\RANDOMX

4. Select begin install.

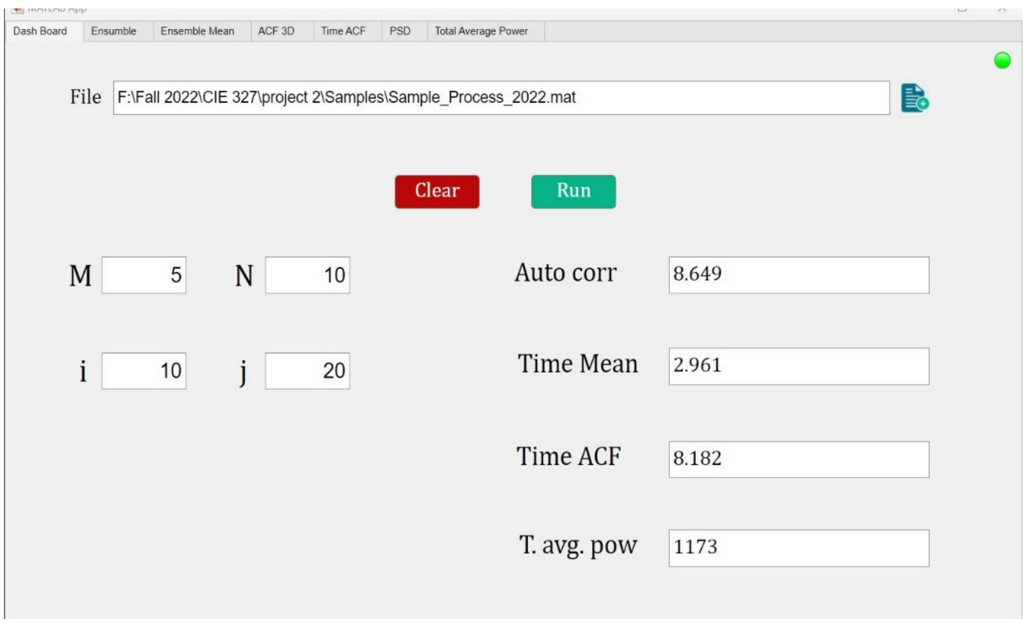


Confirm selections

RANDOMX DESTINATION
C:\Program Files\RANDOMX\RANDOMX
MATLAB RUNTIME DESTINATION
MATLAB Runtime is already installed in
C:\Program Files\MATLAB\MATLAB Runtime\R2022b
This will automatically be updated with files necessary to run this application.
1.04 GB required

❻ | Results of the Samples.

❁ X(t).



File F:\Fall 2022\CIE 327\project 2\Samples\Sample_Process_2022.mat

M 5 N 10

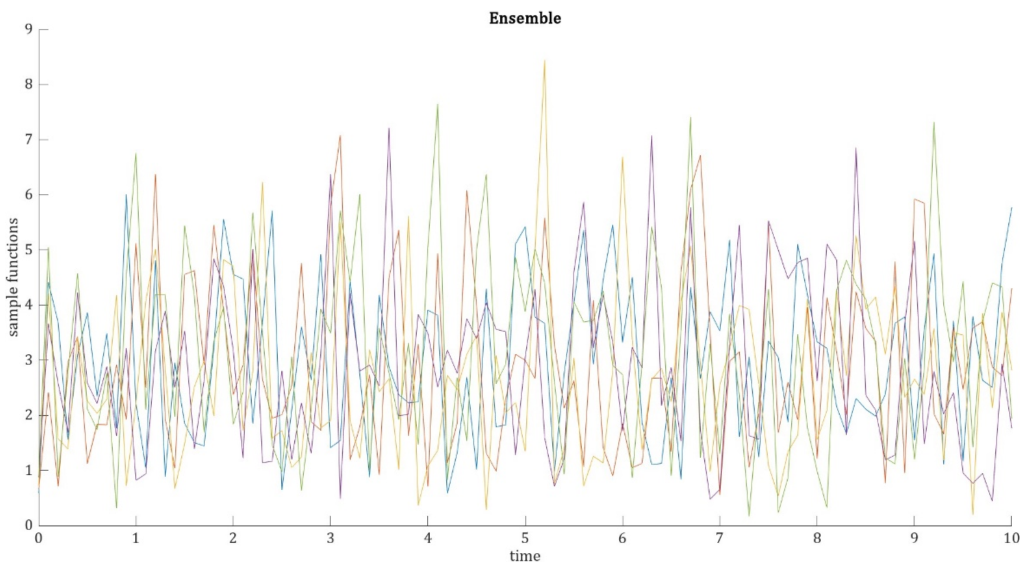
i 10 j 20

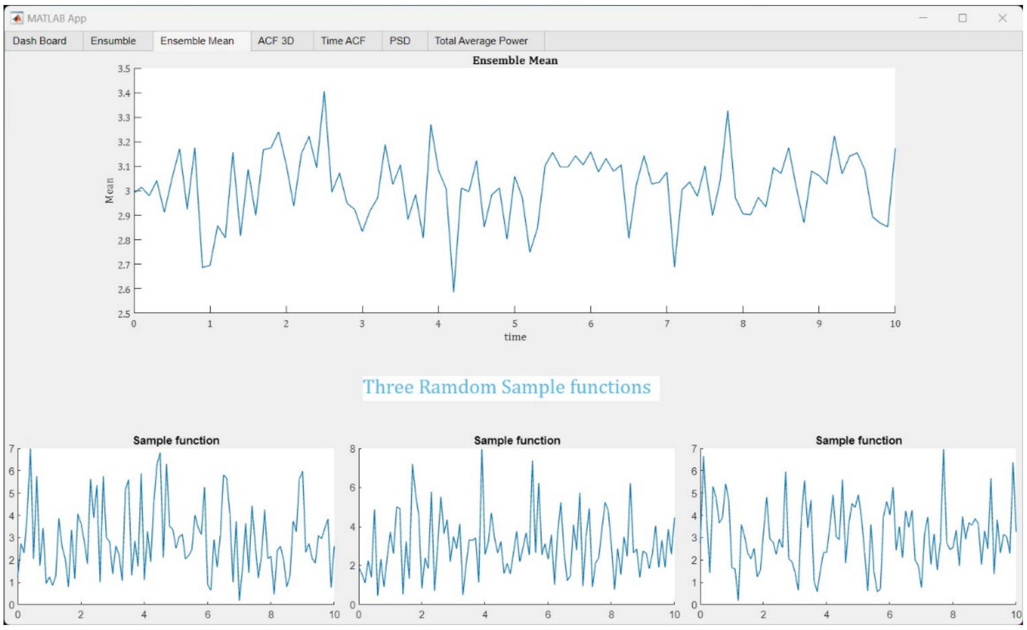
Auto corr 8.649

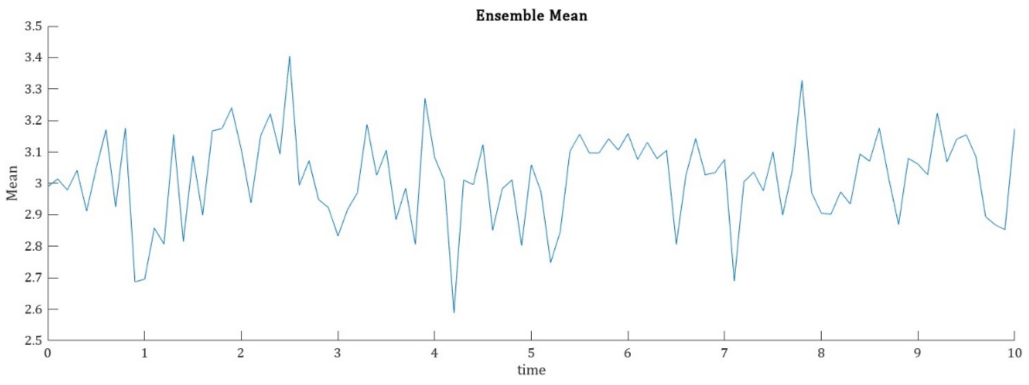
Time Mean 2.961

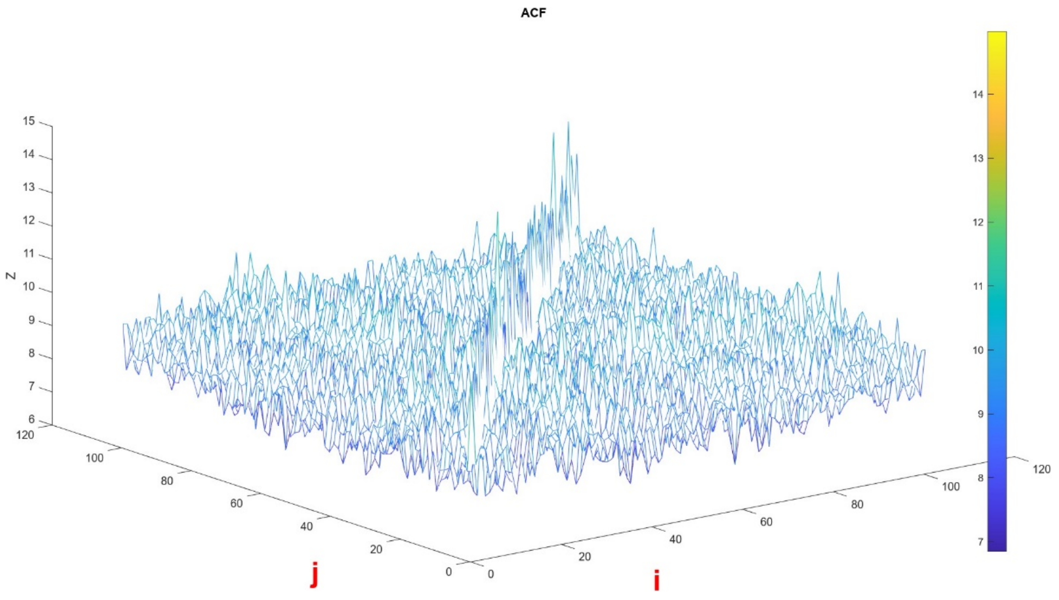
Time ACF 8.182

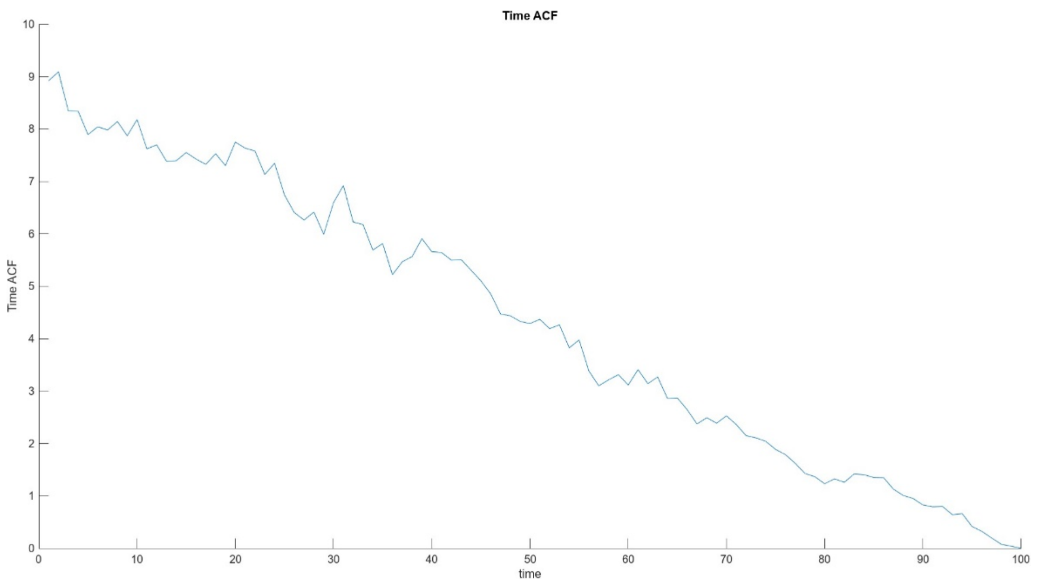
T. avg. pow 1173

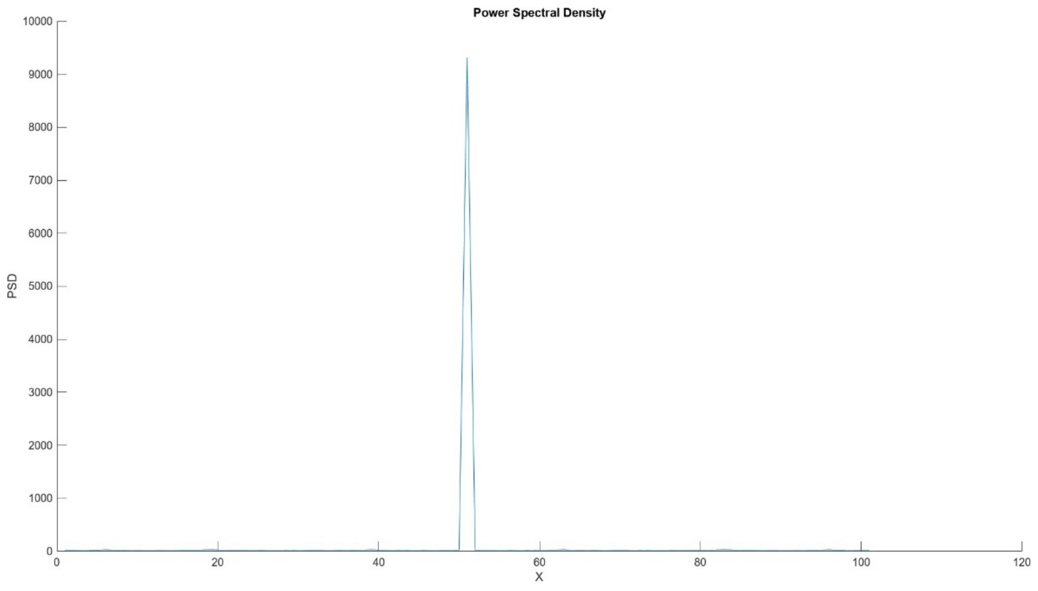


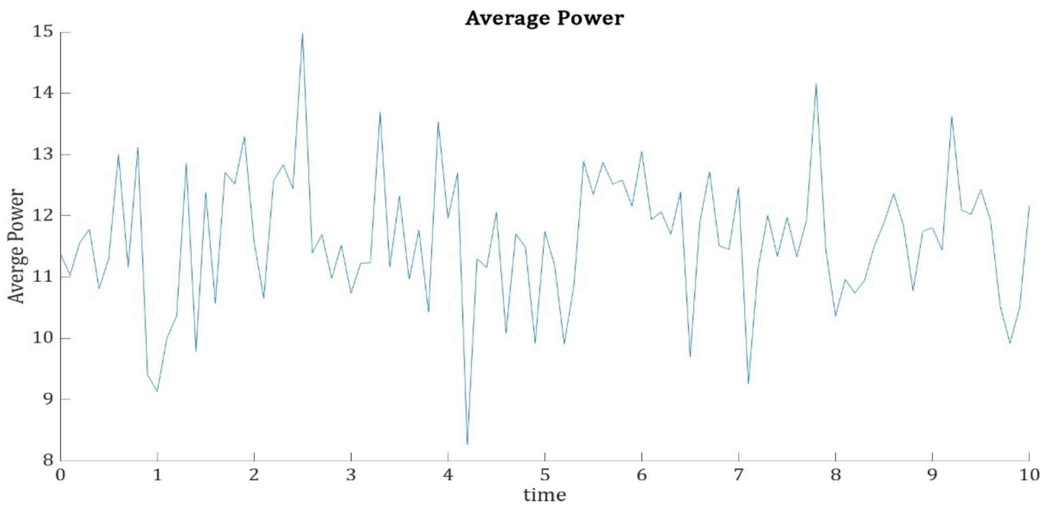

MATLAB App
Dash Board
Ensumble
Ensemble Mean
ACF 3D
Time ACF
PSD
Total Average Power
Ensemble Mean
Mean
time
Three Ramdom Sample functions
Sample function
Sample function
Sample function


Ensemble Mean
Mean
time

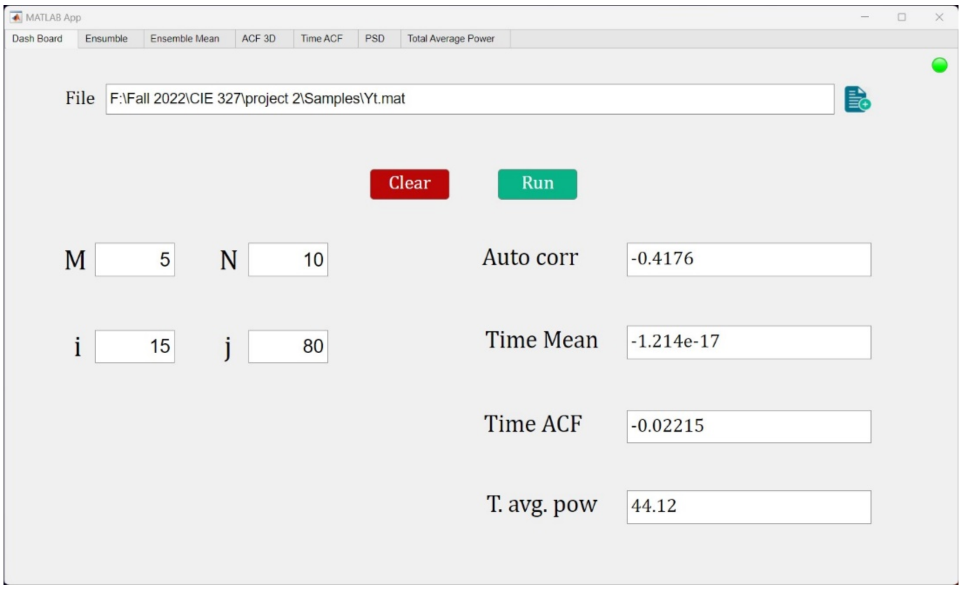

ACF
Z
j
i


Time ACF
Time ACF
time


Power Spectral Density
PSD
X


Average Power
Averge Power
time

- Y(t).



File F:\Fall 2022\CIE 327\project 2\Samples\Yt.mat

M 5 N 10

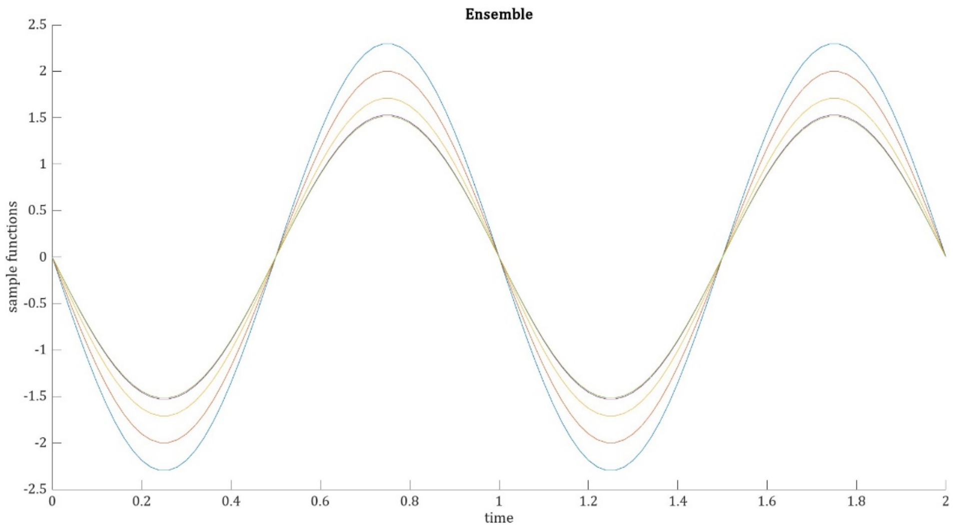
i 15 j 80

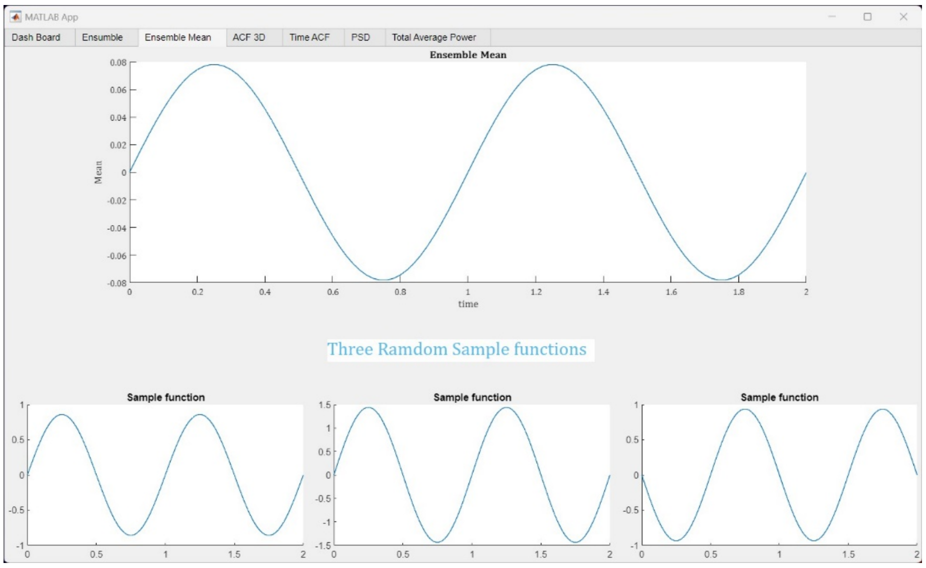
Auto corr -0.4176

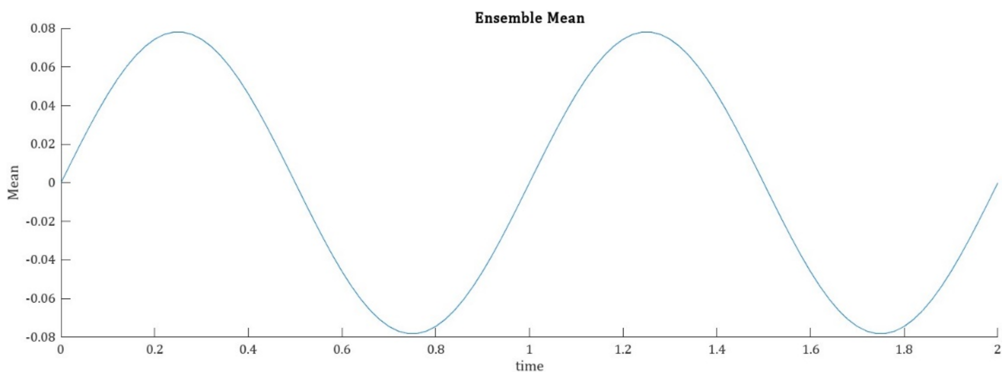
Time Mean -1.214e-17

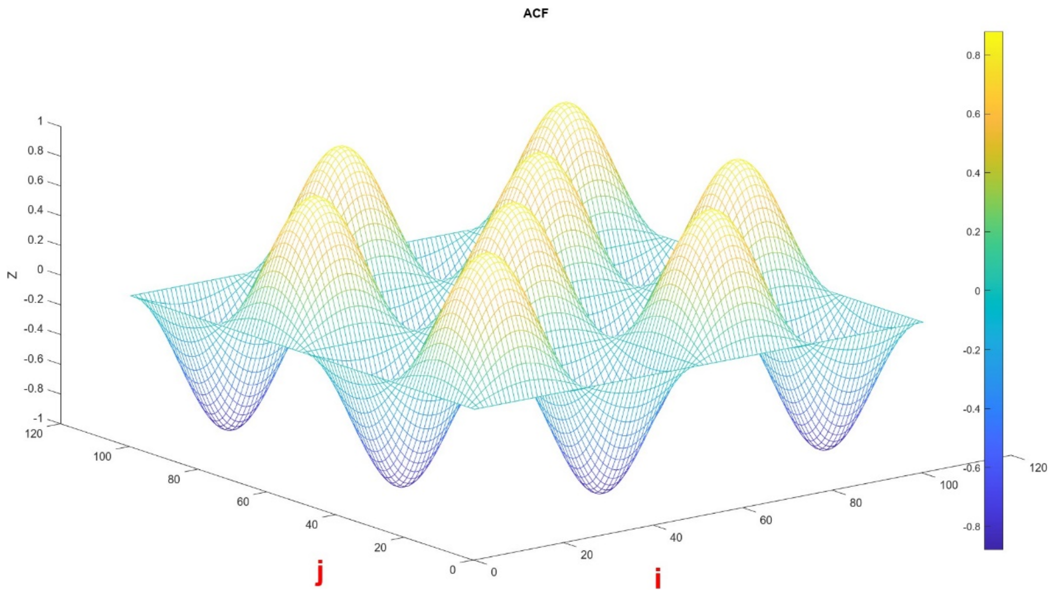
Time ACF -0.02215

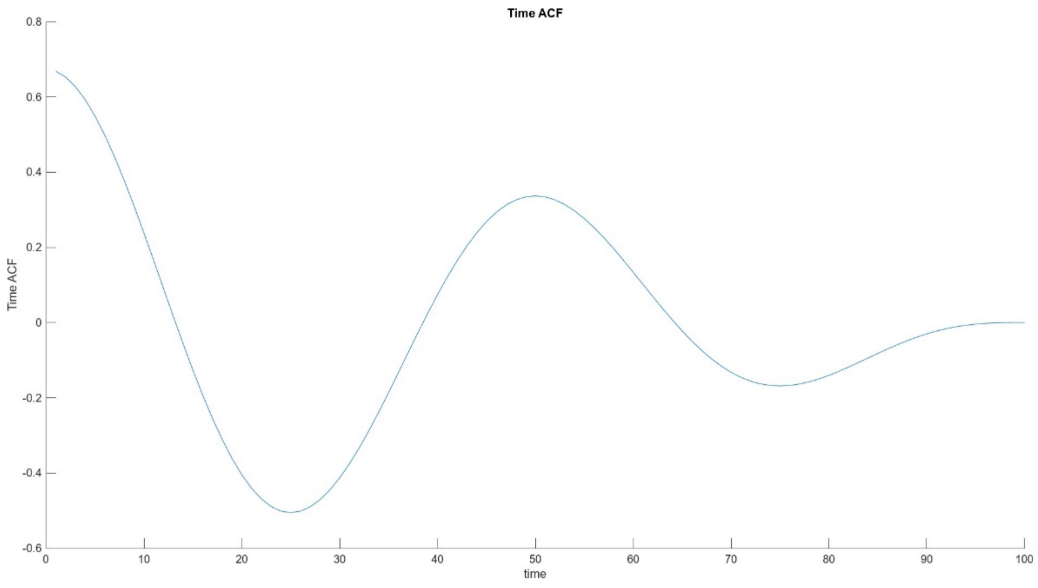
T. avg. pow 44.12

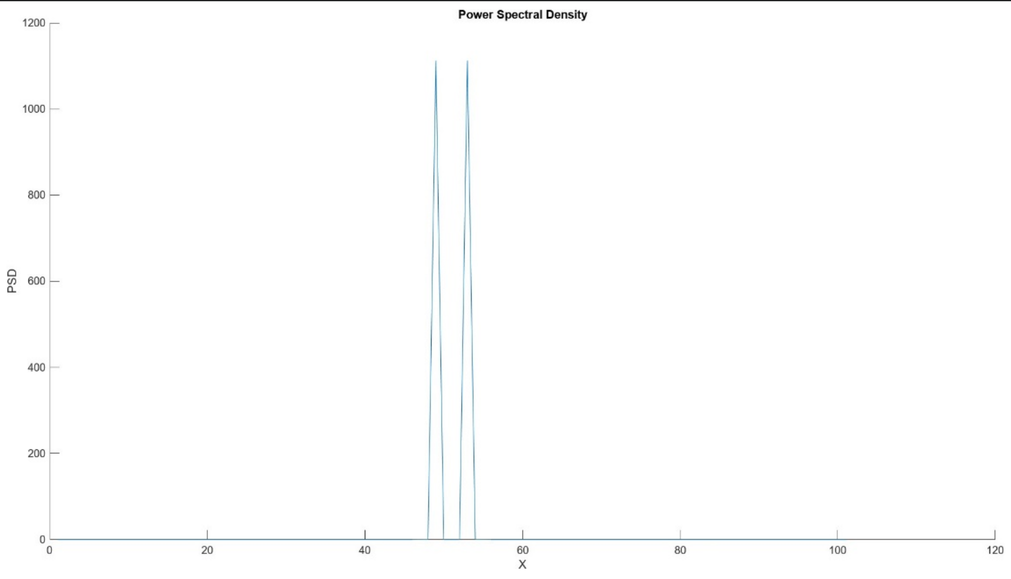


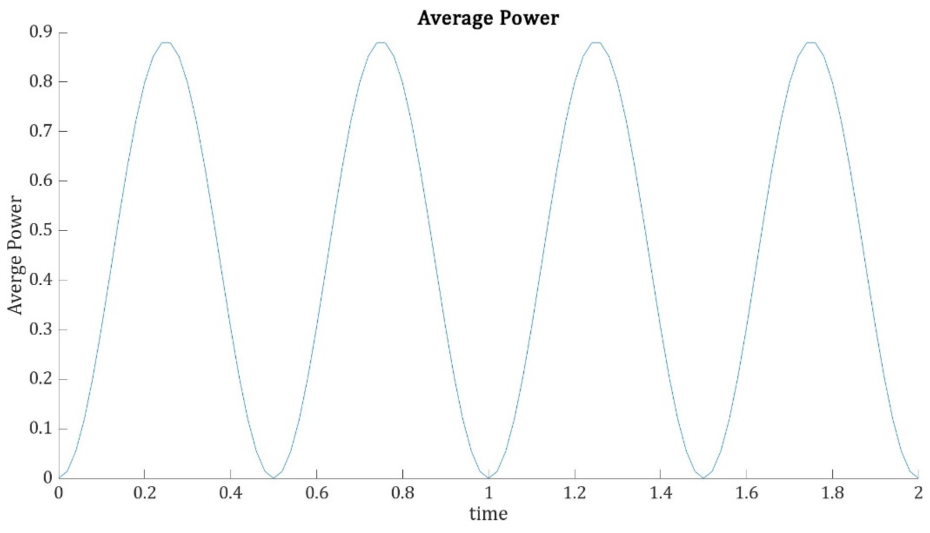



Ensemble Mean
Mean
time

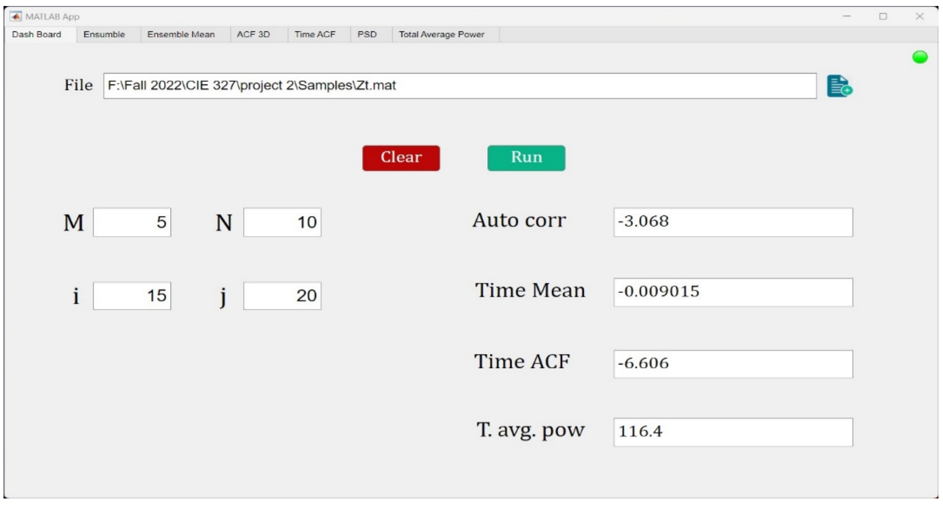

ACF
Z
j
i


Time ACF
Time ACF
time



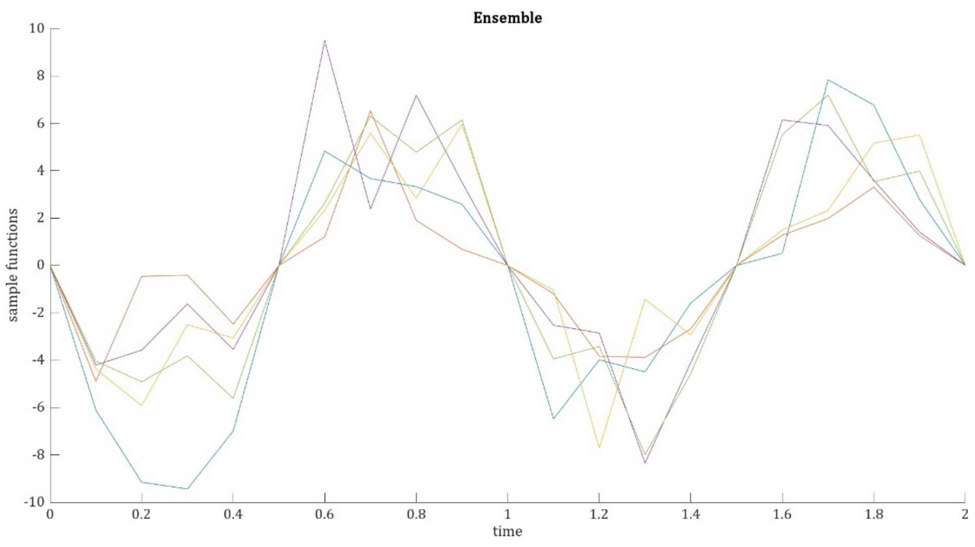


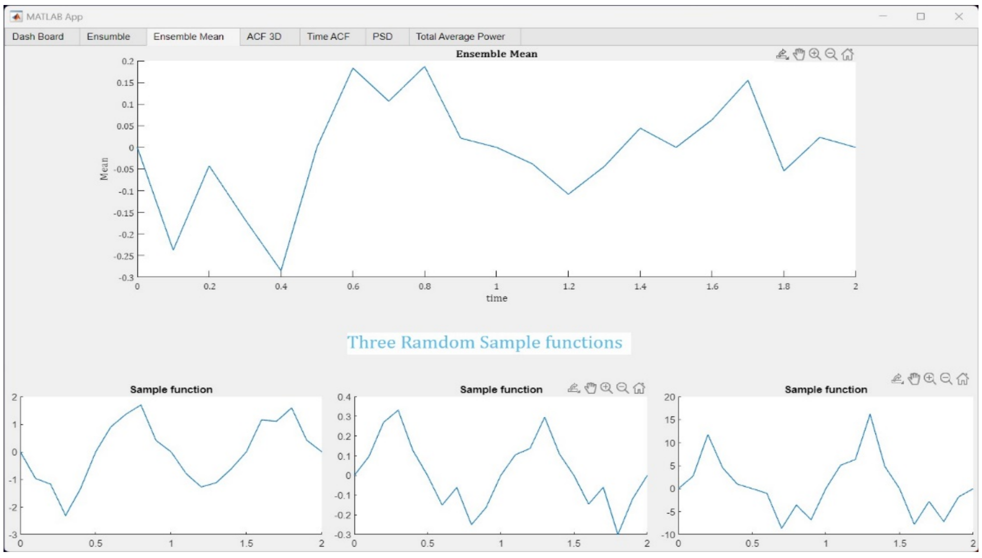
- Z(t).

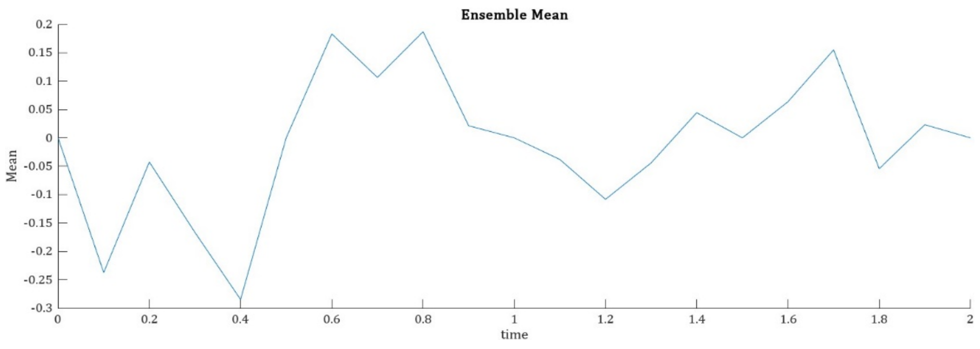


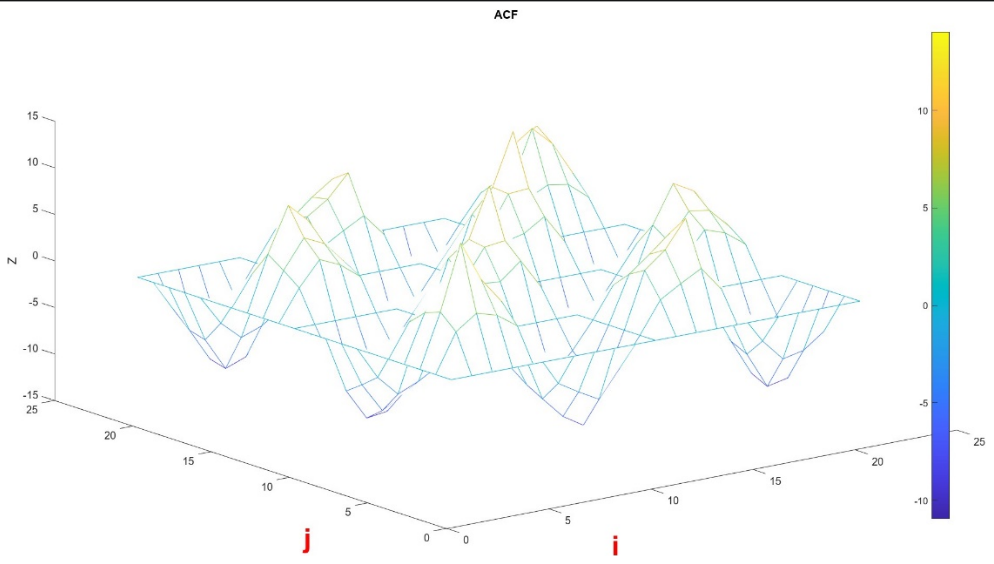
File F:\Fall 2022\CIE 327\project 2\Samples\Zt.mat

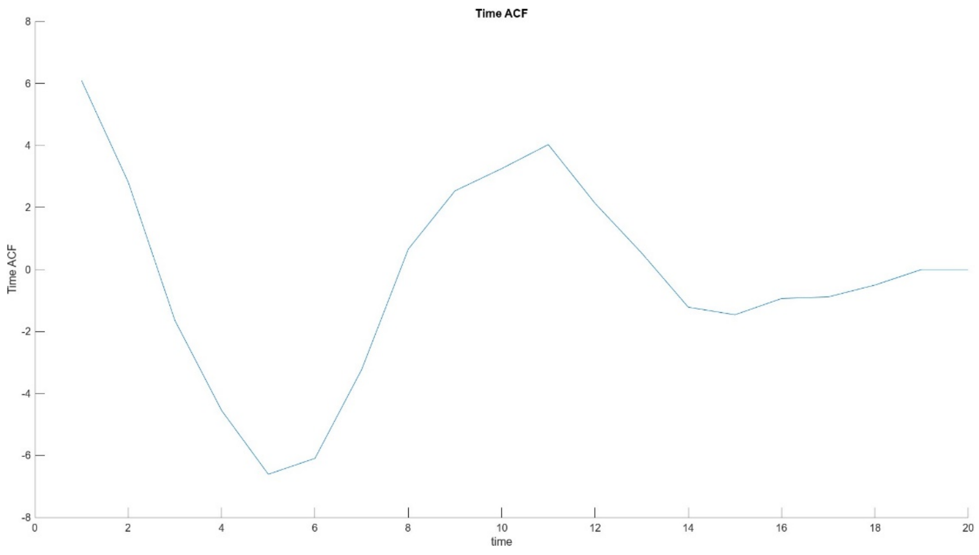
M	5	N	10	Auto corr	-3.068
i	15	j	20	Time Mean	-0.009015
				Time ACF	-6.606
				T. avg. pow	116.4

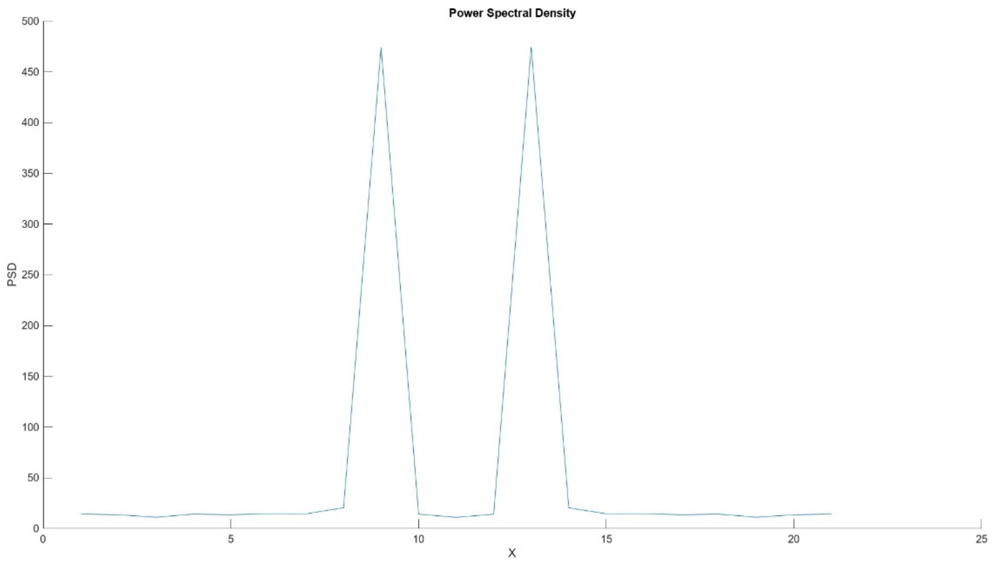

Ensemble
sample functions
time

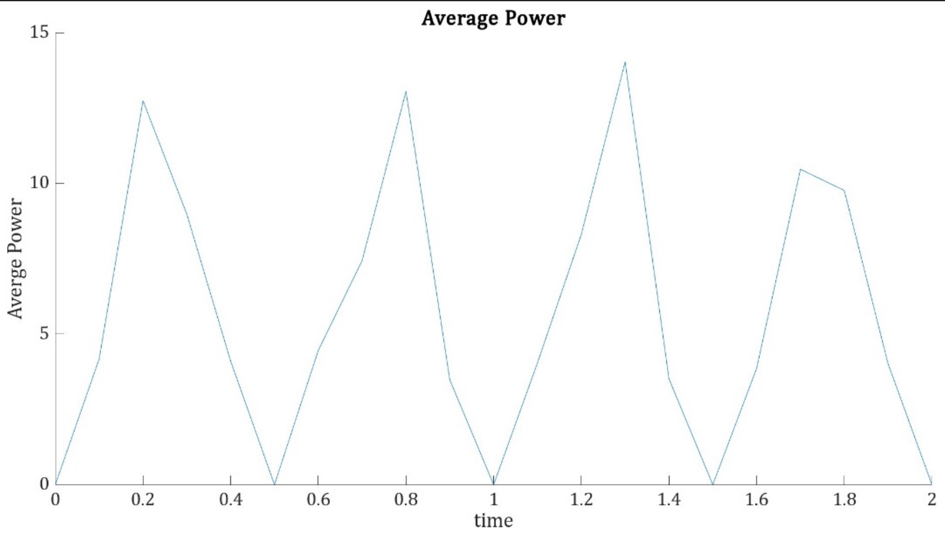

MATLAB App
Dash Board
Ensumble
Ensemble Mean
ACF 3D
Time ACF
PSD
Total Average Power
Ensemble Mean
Mean
time
Three Ramdom Sample functions
Sample function
Sample function
Sample function


Ensemble Mean
Mean
time

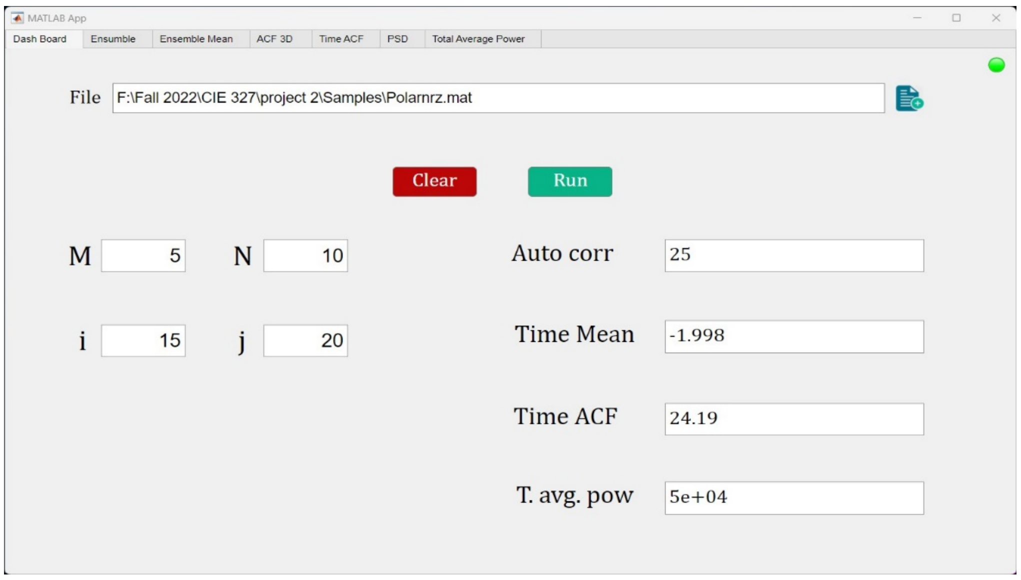

ACF
Z
j
i
15
10
5
0
-5
-10
-15
25
20
15
10
5
0
0
5
10
15
20
25


Time ACF
Time ACF
time
8
6
4
2
0
-2
-4
-6
-8
0
2
4
6
8
10
12
14
16
18
20


Power Spectral Density
PSD
X
500
450
400
350
300
250
200
150
100
50
0
0
5
10
15
20
25

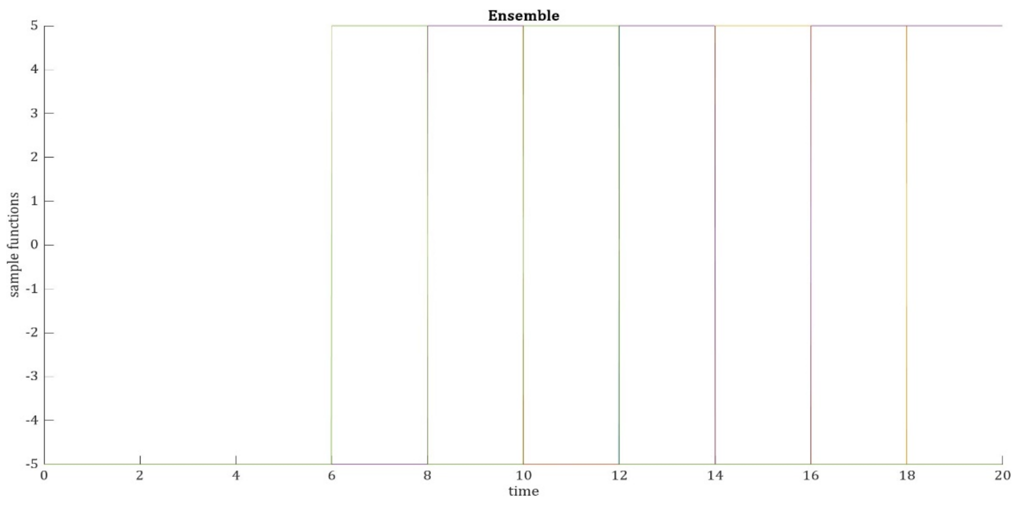


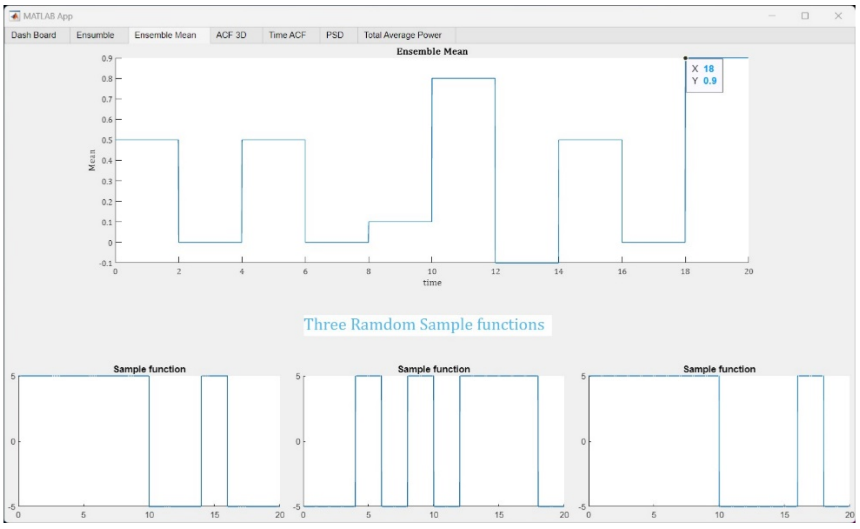
❁ Polar NRZ.

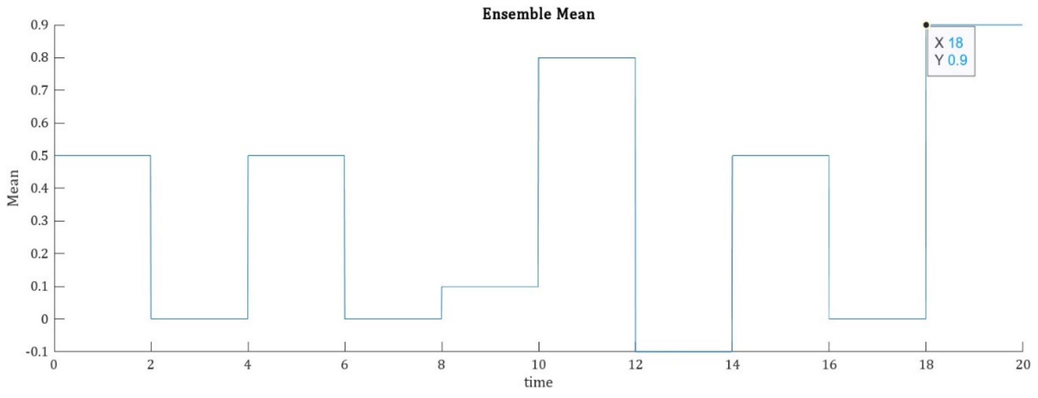


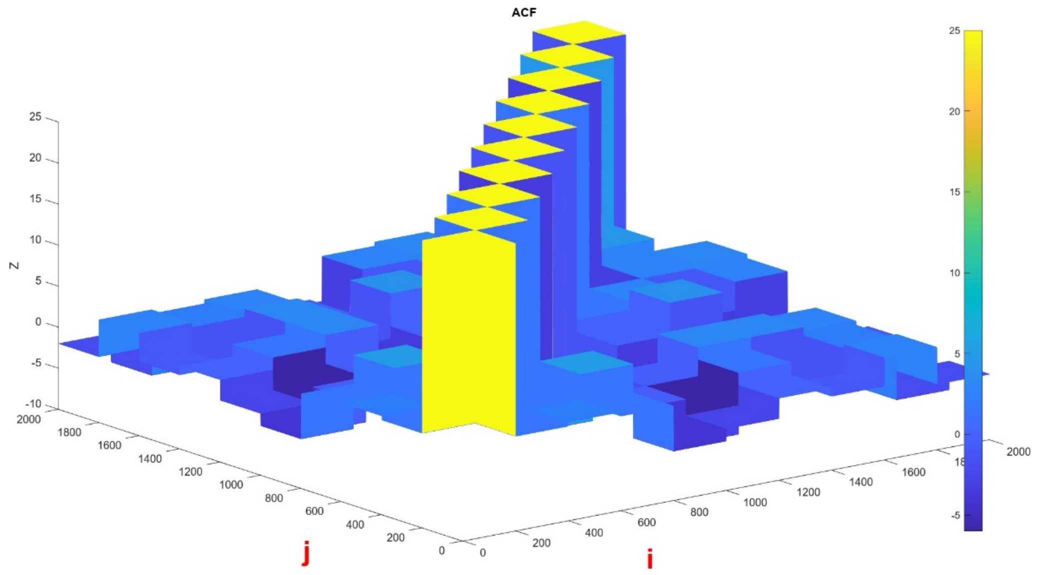
File F:\Fall 2022\CIE 327\project 2\Samples\Polarnrz.mat

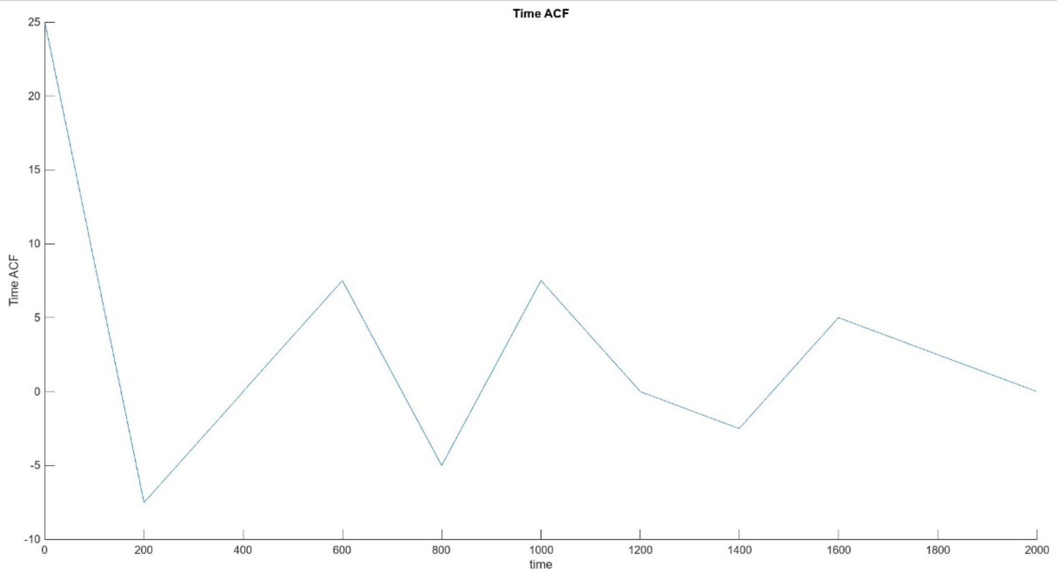
M	5	N	10	Auto corr	25
i	15	j	20	Time Mean	-1.998
				Time ACF	24.19
				T. avg. pow	5e+04

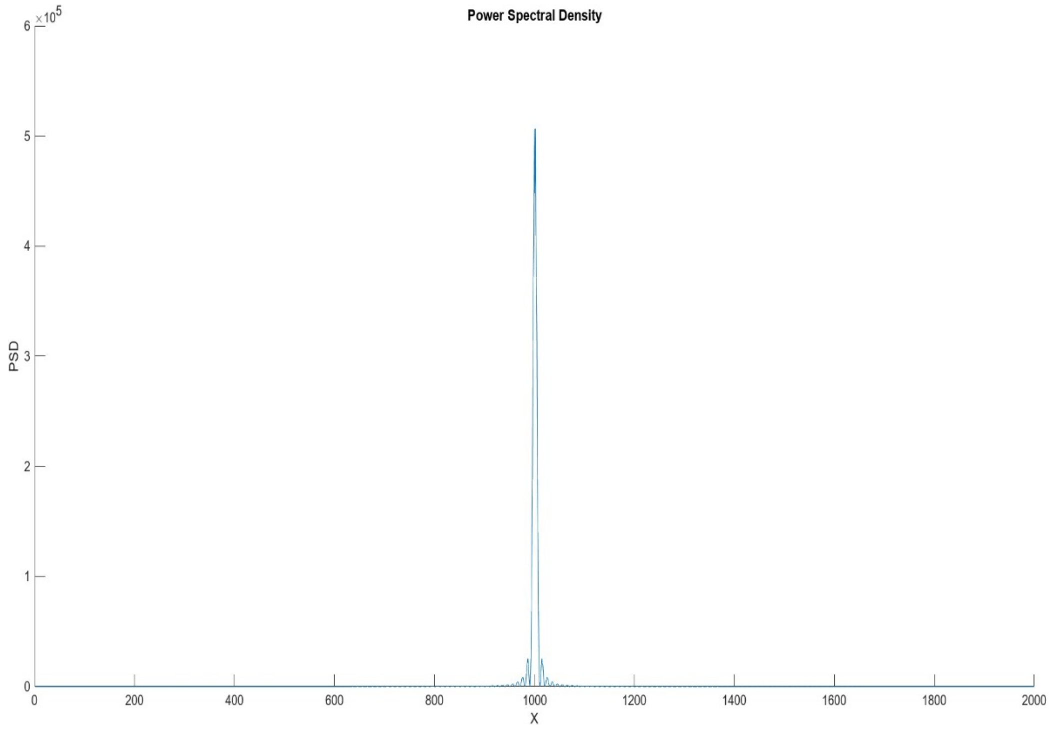


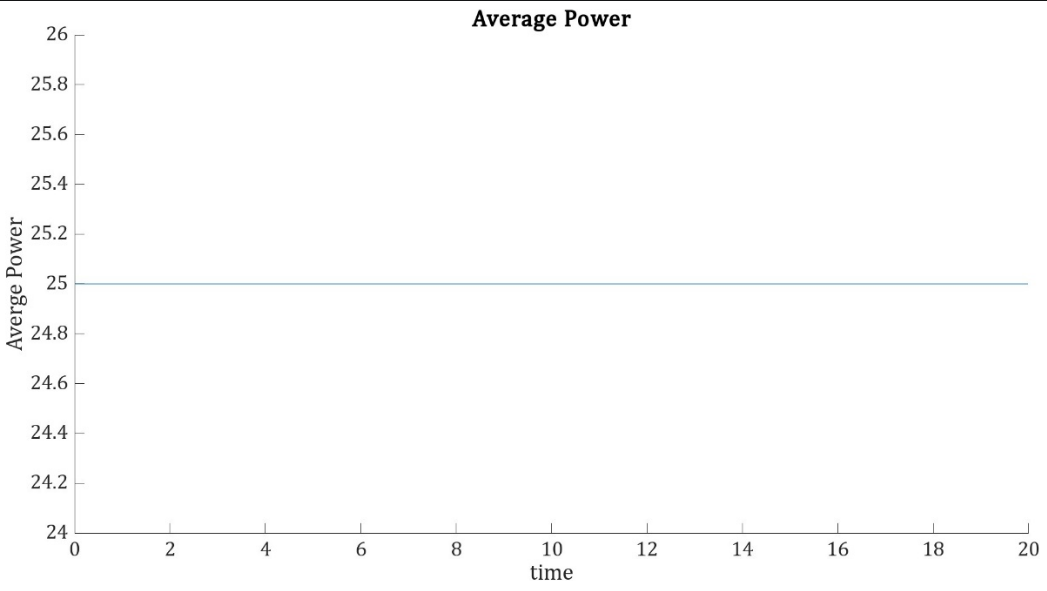

MATLAB App
Dash Board
Ensumble
Ensemble Mean
ACF 3D
Time ACF
PSD
Total Average Power
Ensemble Mean
Mean
time
X 18
Y 0.9
Three Ramdom Sample functions
Sample function
Sample function
Sample function


Ensemble Mean
Mean
time
X 18
Y 0.9

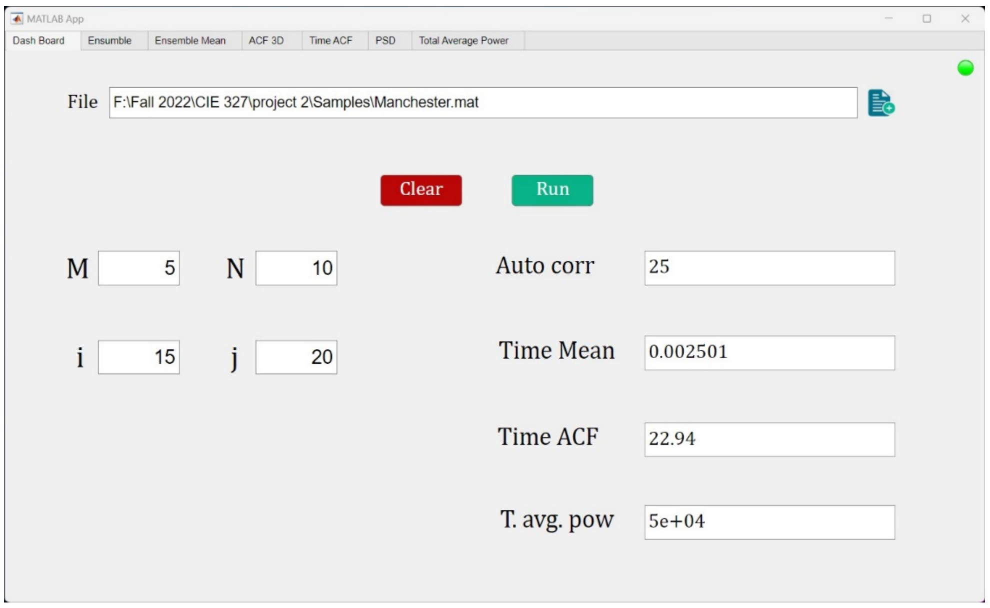

ACF
Z
j
i


Time ACF
Time ACF
time


Power Spectral Density
PSD
X

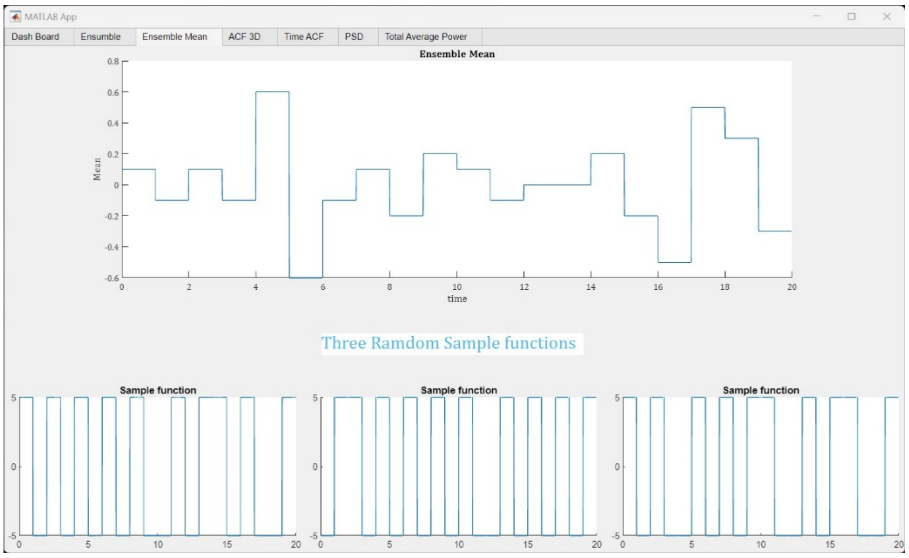


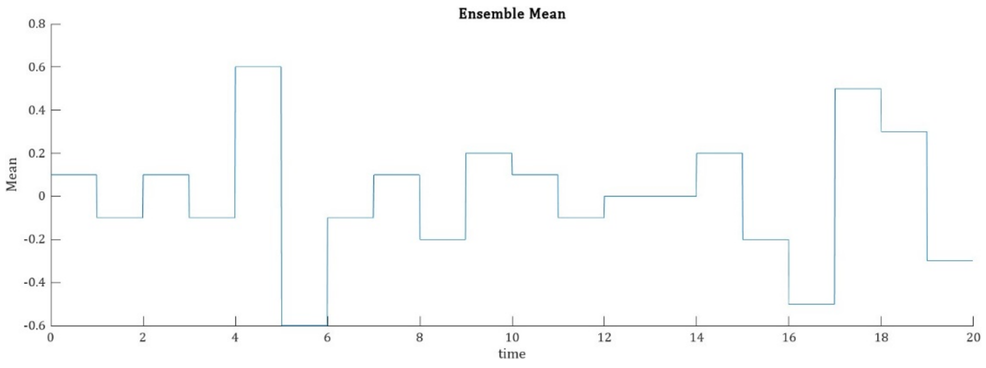
❁ Manchester

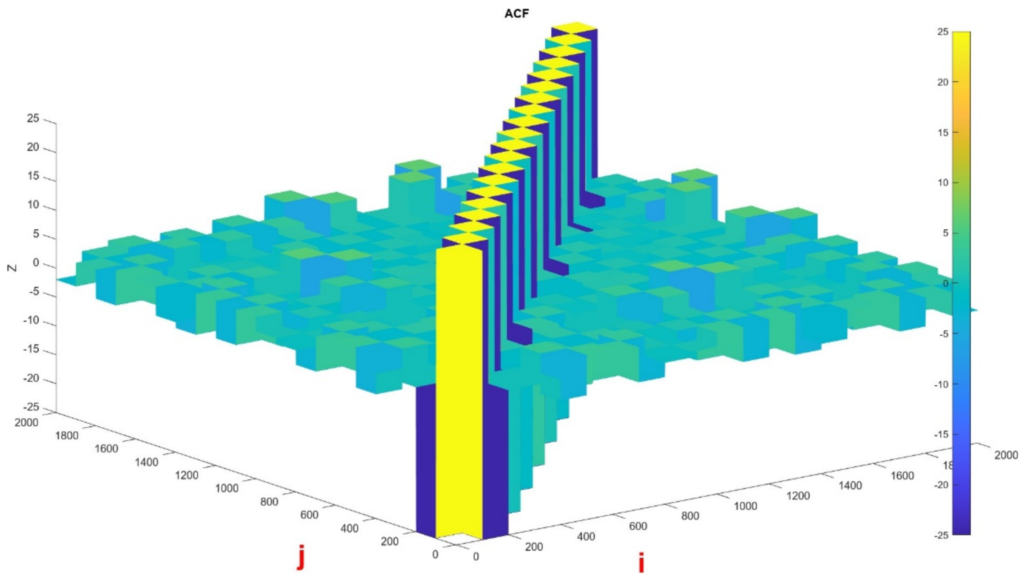


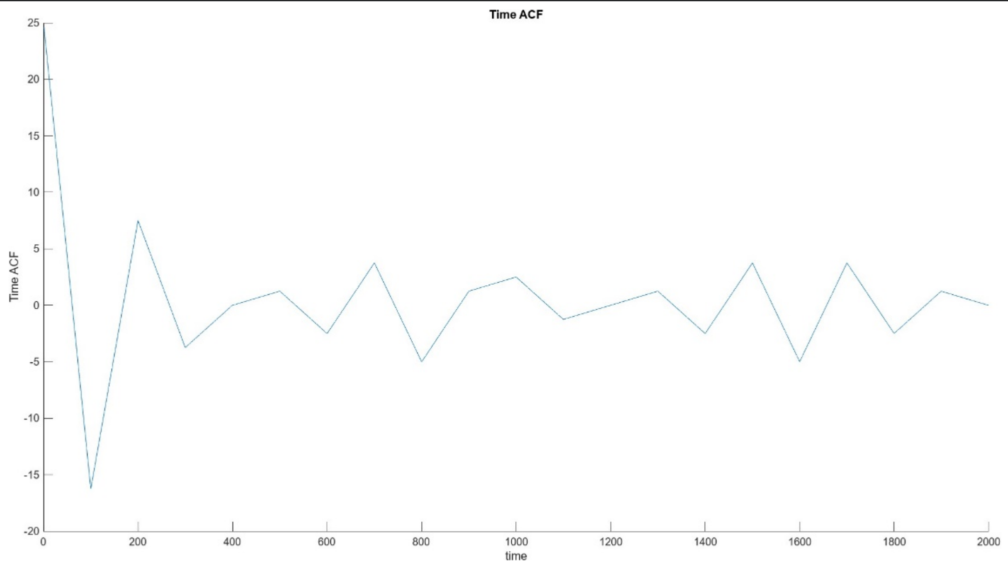
File F:\Fall 2022\CIE 327\project 2\Samples\Manchester.mat

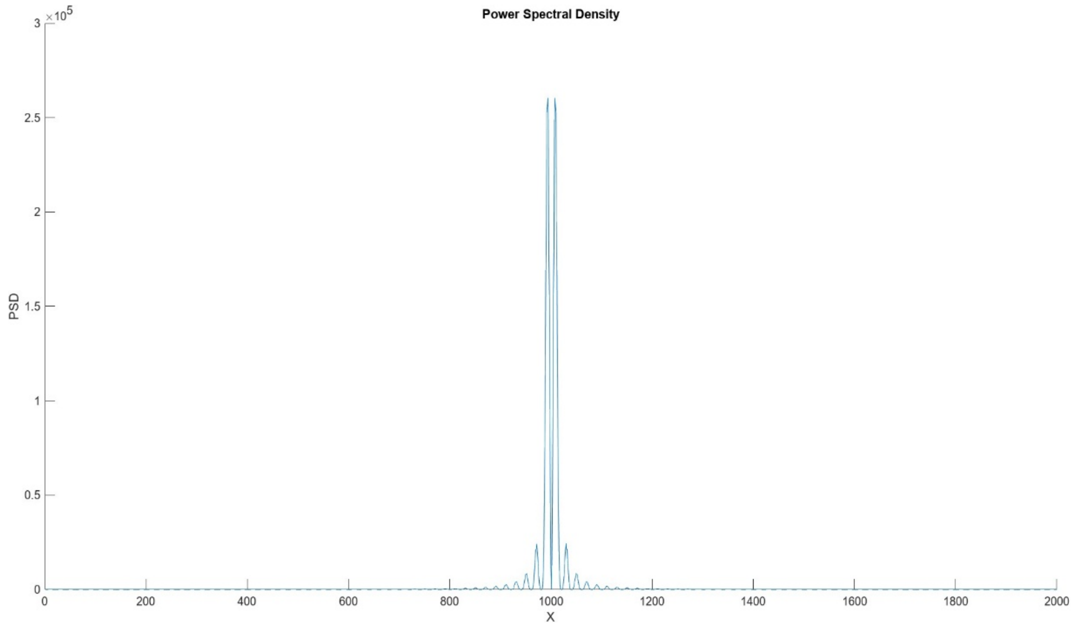
M	5	N	10	Auto corr	25
i	15	j	20	Time Mean	0.002501
				Time ACF	22.94
				T. avg. pow	5e+04

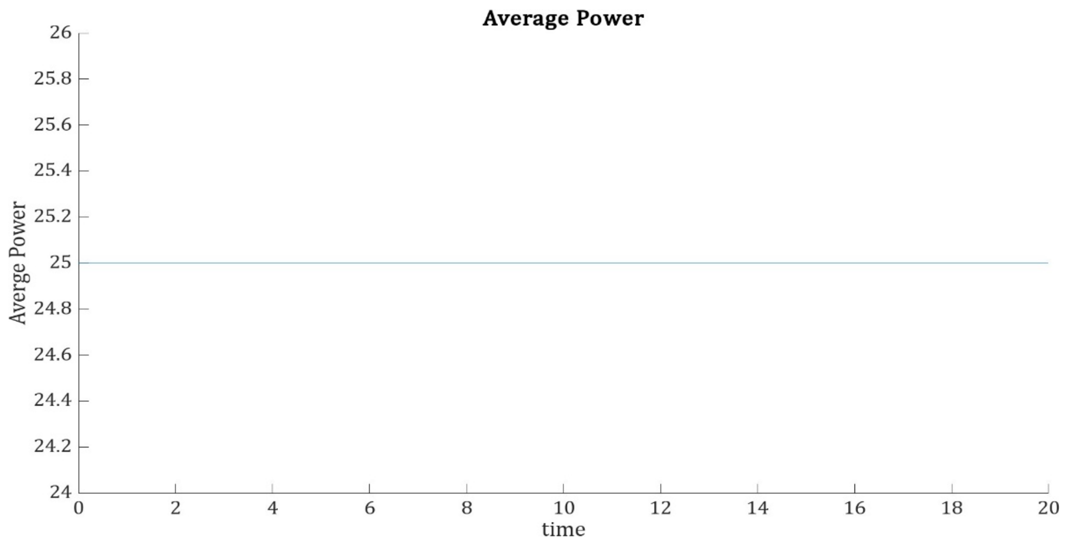

MATLAB App
Dash Board
Ensumble
Ensemble Mean
ACF 3D
Time ACF
PSD
Total Average Power
Ensemble Mean
Mean
time
Three Ramdom Sample functions
Sample function
Sample function
Sample function


Ensemble Mean
Mean
time


ACF
Z
j
i


Time ACF
Time ACF
time


Power Spectral Density
PSD
X
×10^5


Average Power
Averge Power
time

❼ | References.

[1] Walpole, Ronald E._ Myers, Raymond H._ Myers, Sharon L._ Ye, Keying - Probability & statistics for engineers & scientists-Pearson (2017)
[2] https://www.randomservices.org/random/expect/Skew.html

*****