General Signal Generator

Signals and Systems Final Project

Lara Hossam Eldine Mostafa

ID: 6853

Table of Contents

Abstract

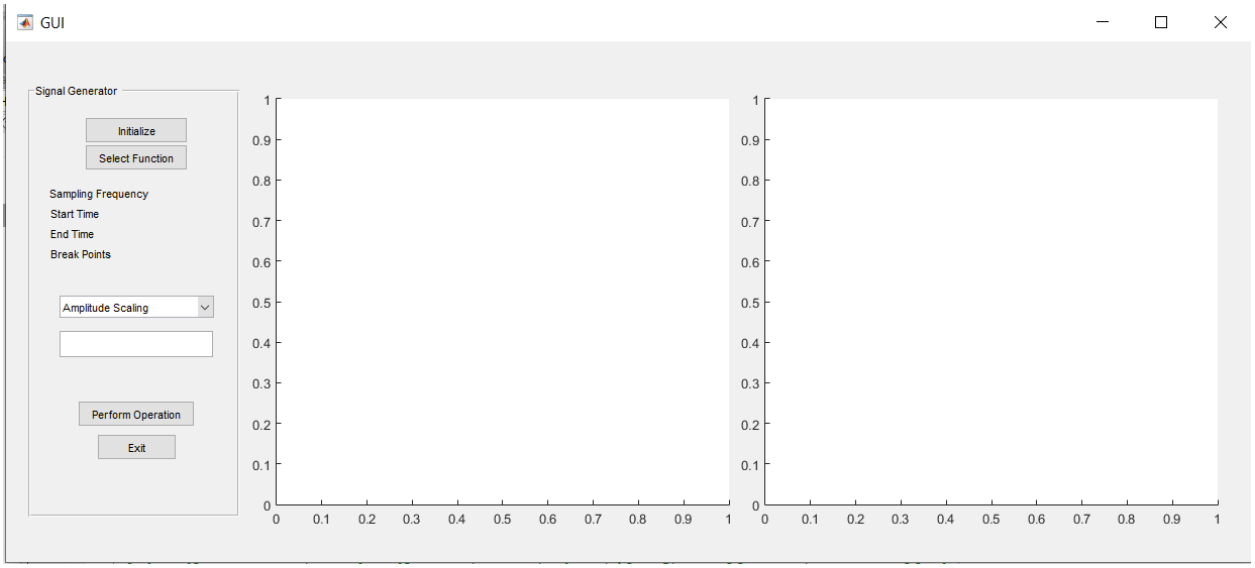
General Signal Generator is a user-friendly program that generates any possible combination of DC, ramp, exponential, polynomial and sinusoidal signals.

The user is also able to perform multiple operations on the signal he generated, he can choose whether he wants his modifications to be on the original signal or on the previous modification.

User Manual

The General Signal Generator is meant to be done in the easiest and most user friendly way possible, that's why it has a graphical user interface **GUI. [BONUS]**

The user interface when launching the program is as follows:

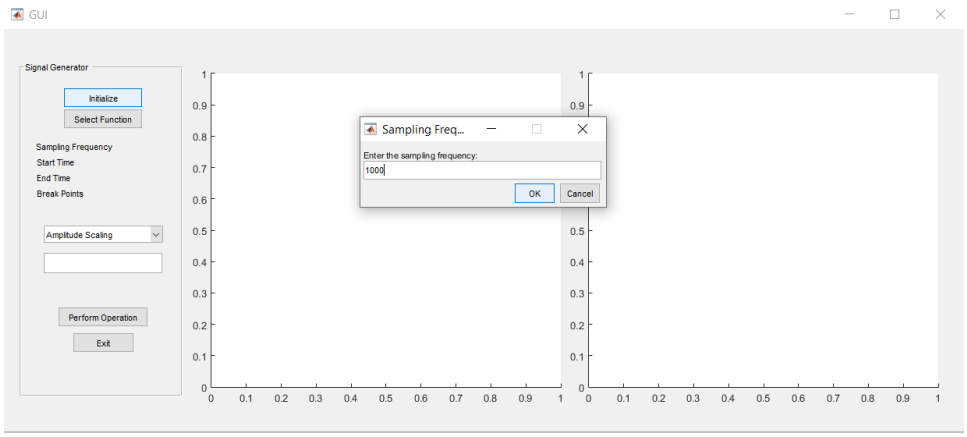


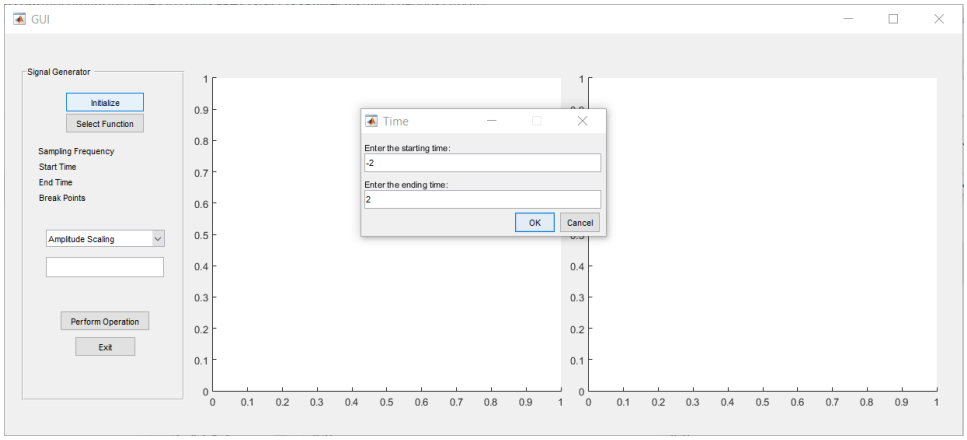
Each button prompts the user to make choices that define his desired signal:

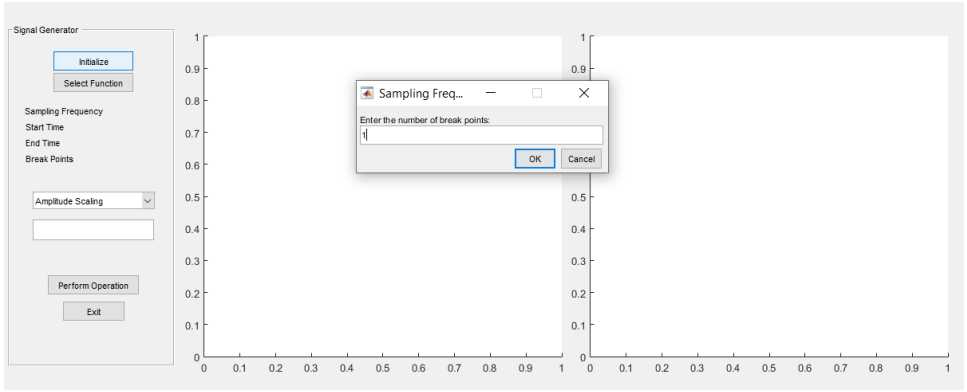
- **Initialize button**

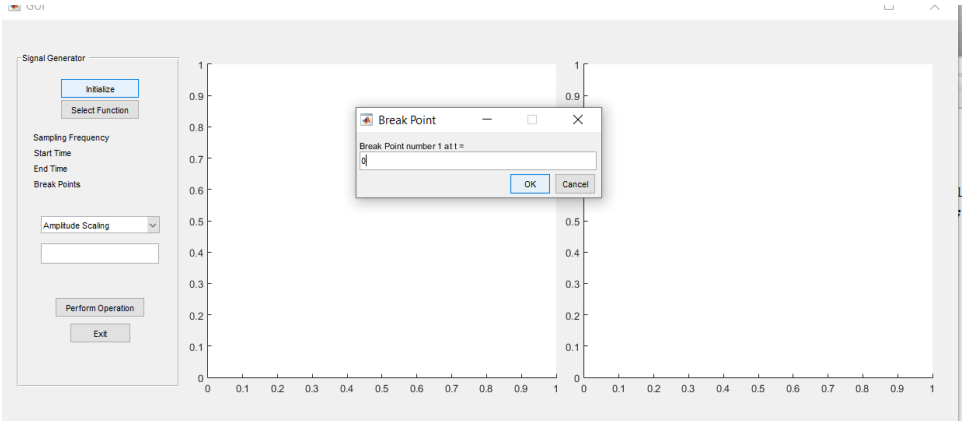
User is prompted to enter the following data:

- Sampling Frequency
- Start Time
- End Time
- Number of breakpoints and their values in time domain

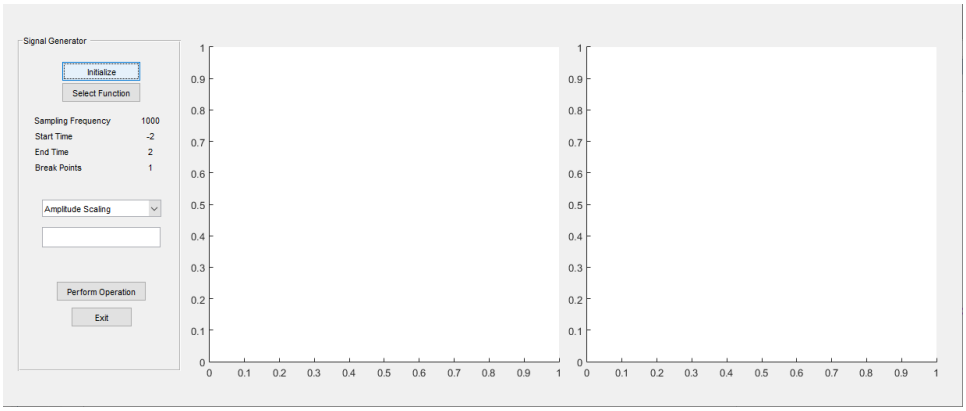

GUI
Signal Generator
Initialize
Select Function
Sampling Frequency
Start Time
End Time
Break Points
Amplitude Scaling
Perform Operation
Exit
Sampling Freq...
Enter the sampling frequency:
1000
OK
Cancel


GUI
Signal Generator
Initialize
Select Function
Sampling Frequency
Start Time
End Time
Break Points
Amplitude Scaling
Perform Operation
Exit
Time
Enter the starting time:
-2
Enter the ending time:
2
OK
Cancel


Signal Generator
Initialize
Select Function
Sampling Frequency
Start Time
End Time
Break Points
Amplitude Scaling
Perform Operation
Exit
Sampling Freq...
Enter the number of break points:
1
OK
Cancel



Data is then updated in the panel and user is prompted to click on the **select function** button to choose his signal:



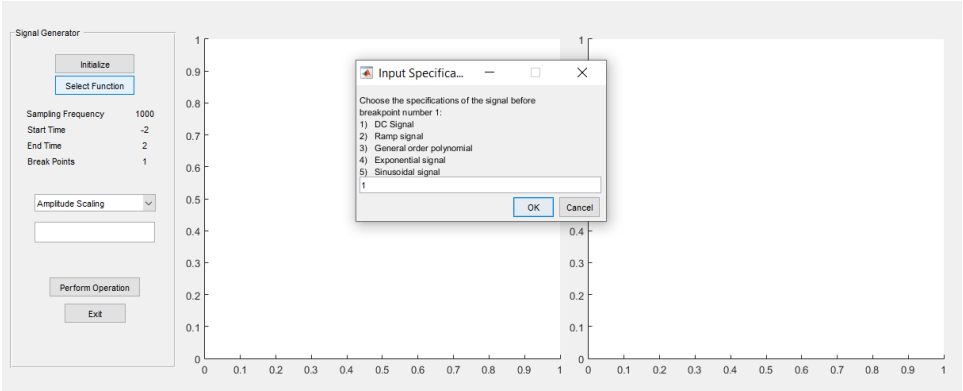
- **Select Function**

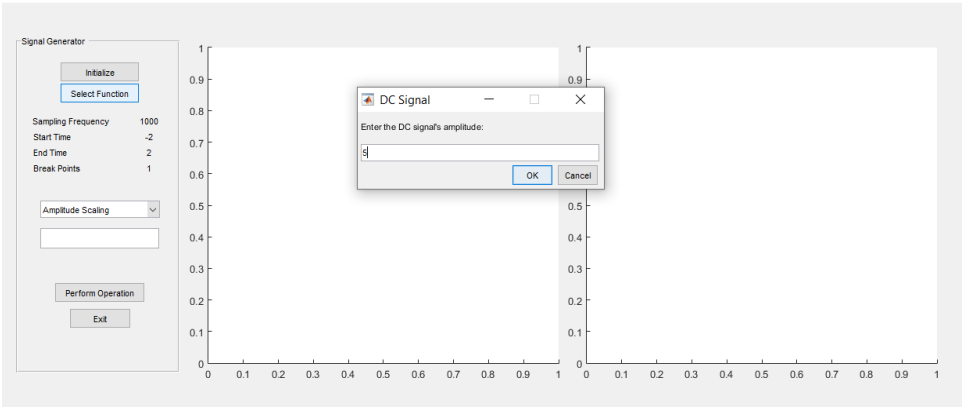
The user gets to choose from multiple options for each breakpoint he entered [only one signal if 0 breakpoints]:

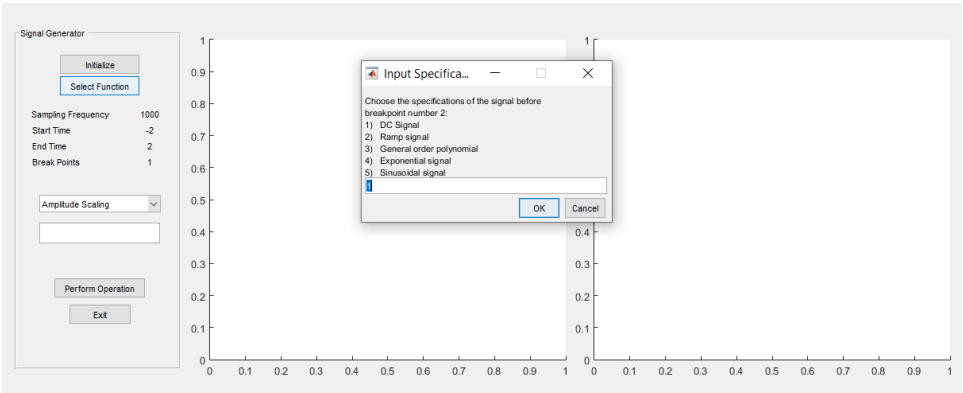
1) DC signal
2) Ramp signal
3) Polynomial Signal
4) Exponential Signal
5) Sinusoidal Signal

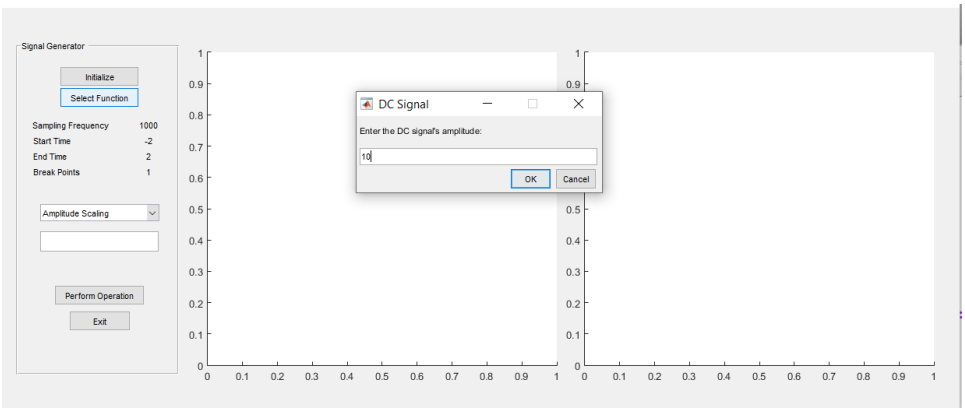
The user then decides all attributes to the signal chosen:

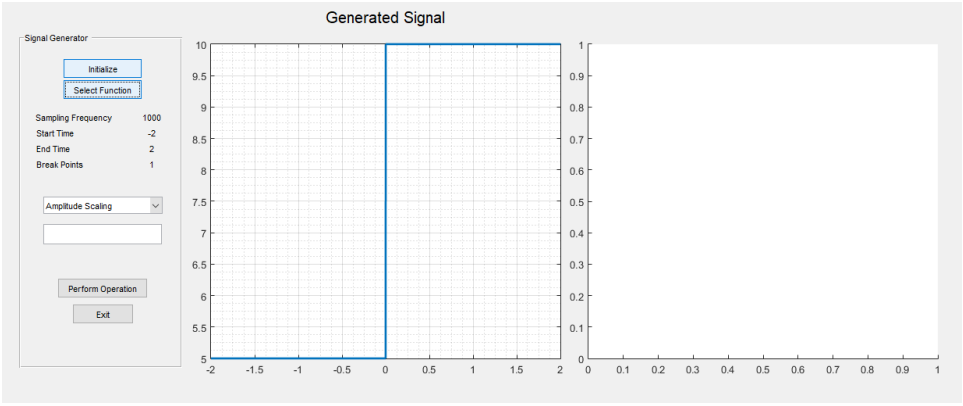
1) DC signal - amplitude
2) Ramp signal - slope - intercept
3) General order polynomial: amplitude - power - intercept
4) Exponential signal: amplitude - exponent
5) Sinusoidal signal: amplitude - frequency - phase


Signal Generator
Initialize
Select Function
Sampling Frequency 1000
Start Time -2
End Time 2
Break Points 1
Amplitude Scaling
Perform Operation
Exit
Input Specifica...
Choose the specifications of the signal before breakpoint number 1:
1) DC Signal
2) Ramp signal
3) General order polynomial
4) Exponential signal
5) Sinusoidal signal
1
OK
Cancel


Signal Generator
Initialize
Select Function
Sampling Frequency 1000
Start Time -2
End Time 2
Break Points 1
Amplitude Scaling
Perform Operation
Exit
DC Signal
Enter the DC signal's amplitude:
5
OK
Cancel

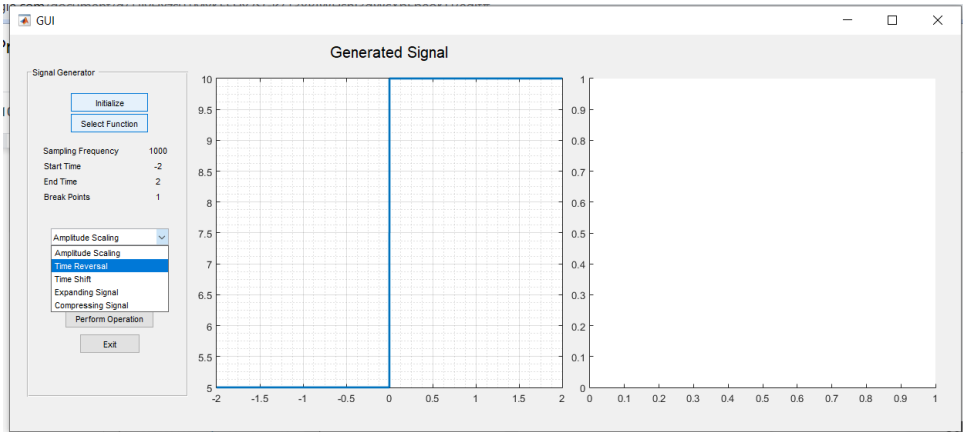

Signal Generator
Initialize
Select Function
Sampling Frequency 1000
Start Time -2
End Time 2
Break Points 1
Amplitude Scaling
Perform Operation
Exit
Input Specifica...
Choose the specifications of the signal before breakpoint number 2:
1) DC Signal
2) Ramp signal
3) General order polynomial
4) Exponential signal
5) Sinusoidal signal
4
OK
Cancel

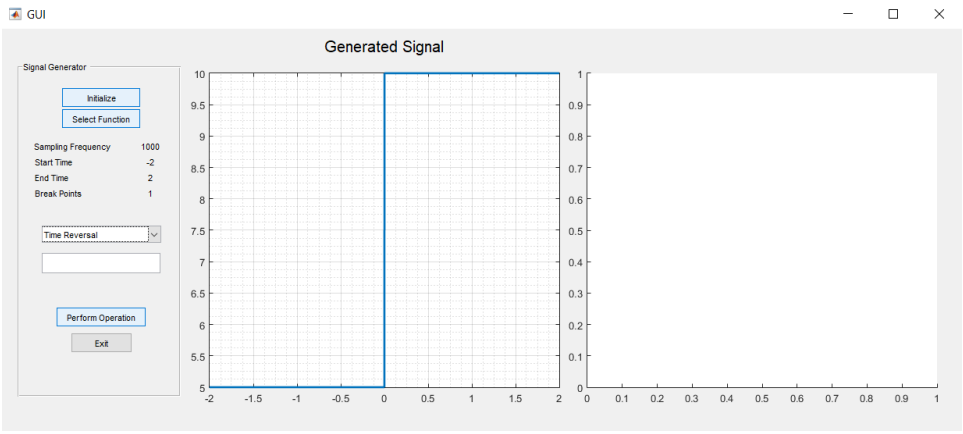

Signal Generator
Initialize
Select Function
Sampling Frequency 1000
Start Time -2
End Time 2
Break Points 1
Amplitude Scaling
Perform Operation
Exit
DC Signal
Enter the DC signal's amplitude:
10
OK
Cancel

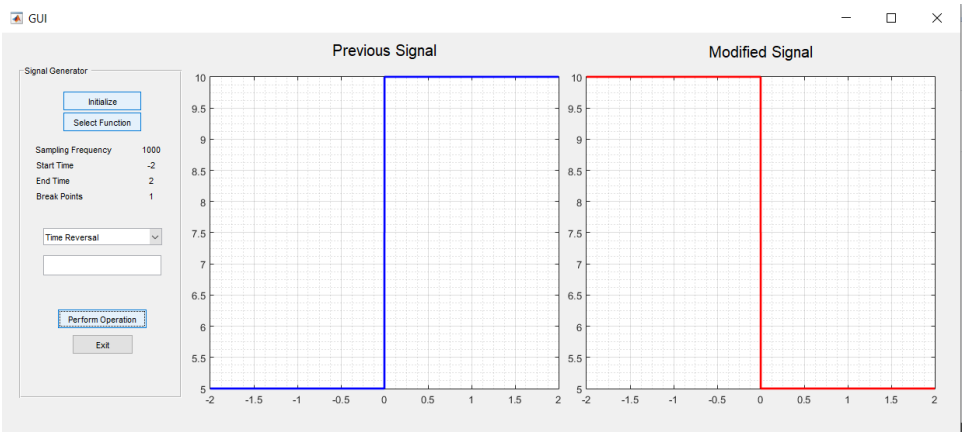


- **To perform any operation, the user chooses from the drop down menu of operations**

After the user chooses the operation he desires from the drop down menu, he enters the suitable value of shift/scale/compression or expansion and then clicks **perform operation.**



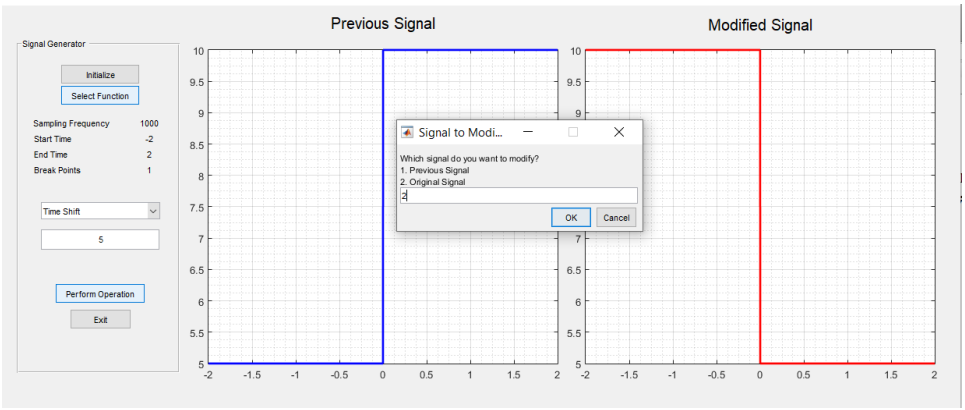


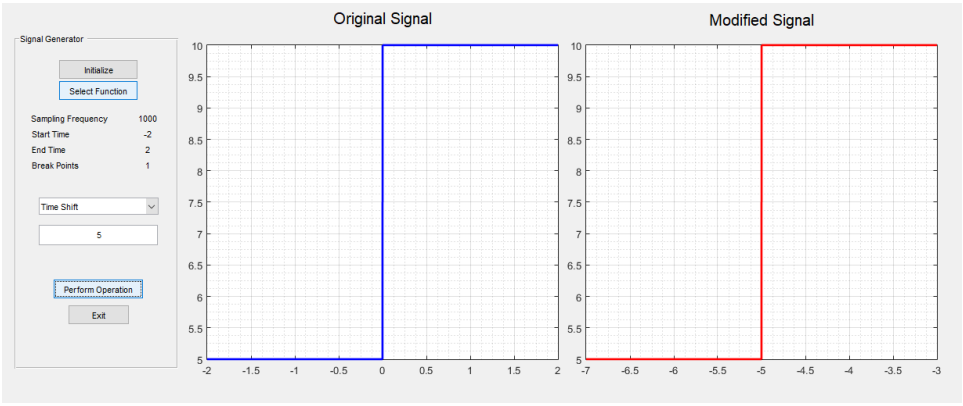


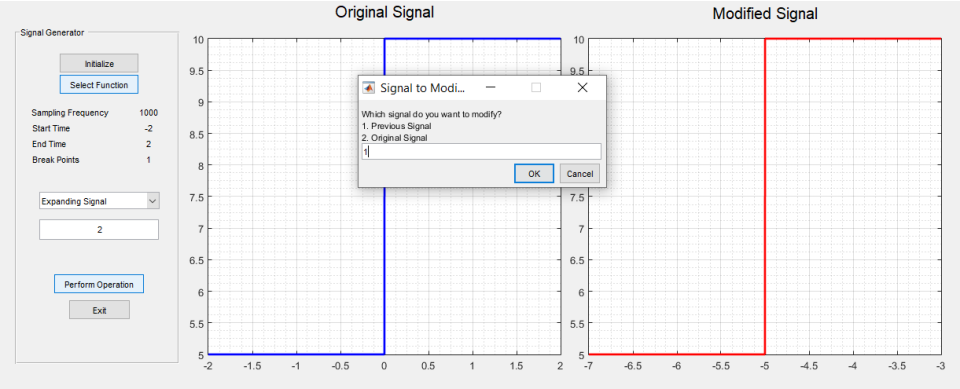
- **<u>For more operations,</u> the user gets to choose the operation he likes from the drop down menu**

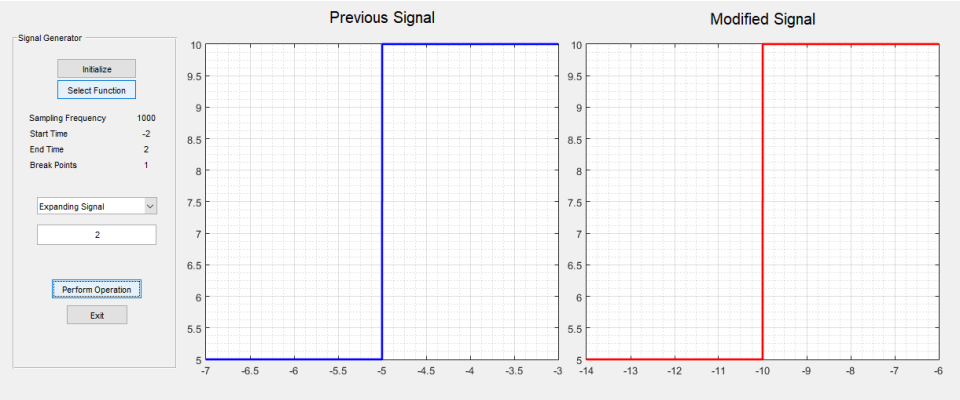
  User gets to choose if he wants to modify the original signal or the modified signal (the most recent plot)

  The user can modify the original function at each point of the program, if he desires to see each change individually, he can also perform all his operations in a row, and plot the final signal with all the changes combined.







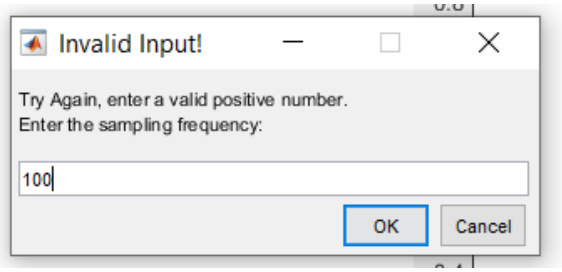


- User can **exit** at any point.

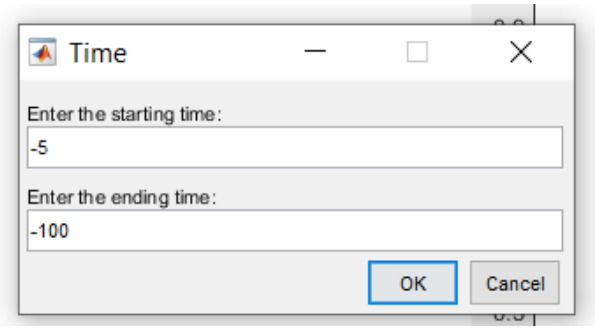
Error Handling

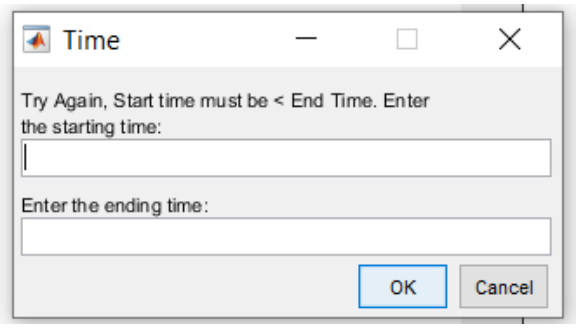
To provide the best user experience possible, user is warned about the false input and is prompted to correct it:

1. Frequency can't be a string or a negative number:

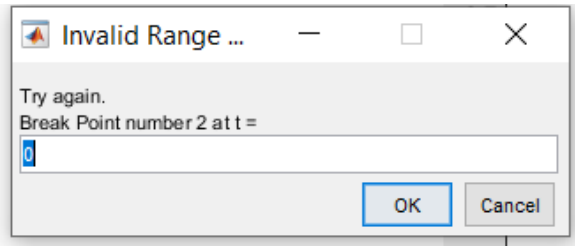


2. If start time is bigger than or equal end time an error is generated and user is prompt to enter the times again:

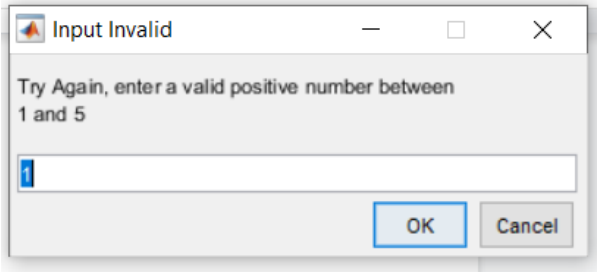




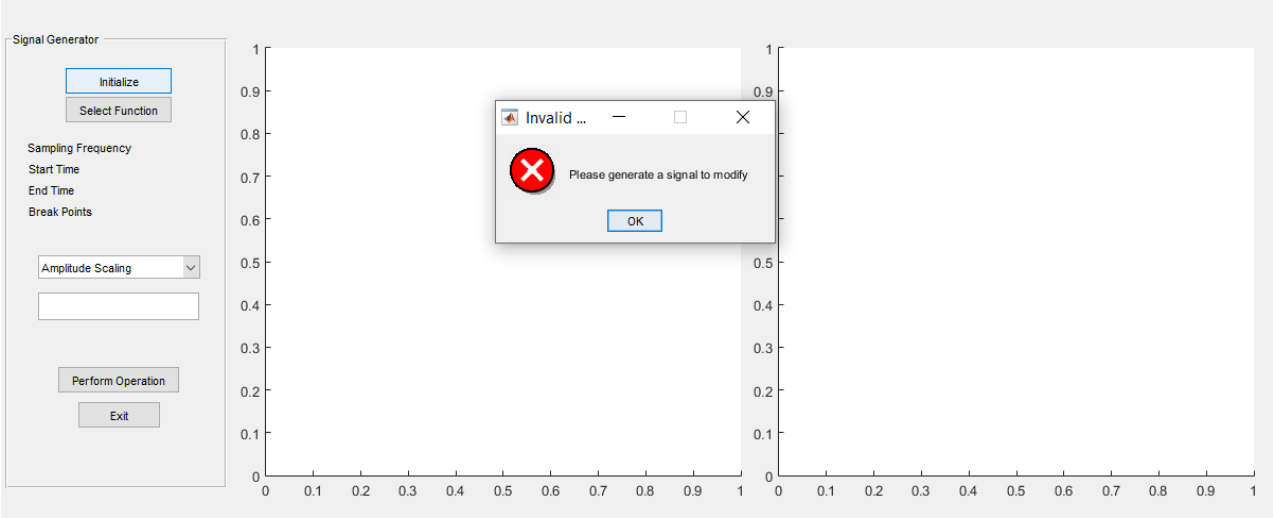
3. Breakpoints must be entered in chronological order



4. User can’t enter a choice outside of the range when choosing the signal type.



5. User can’t perform operations if he hasn't chosen a signal before.

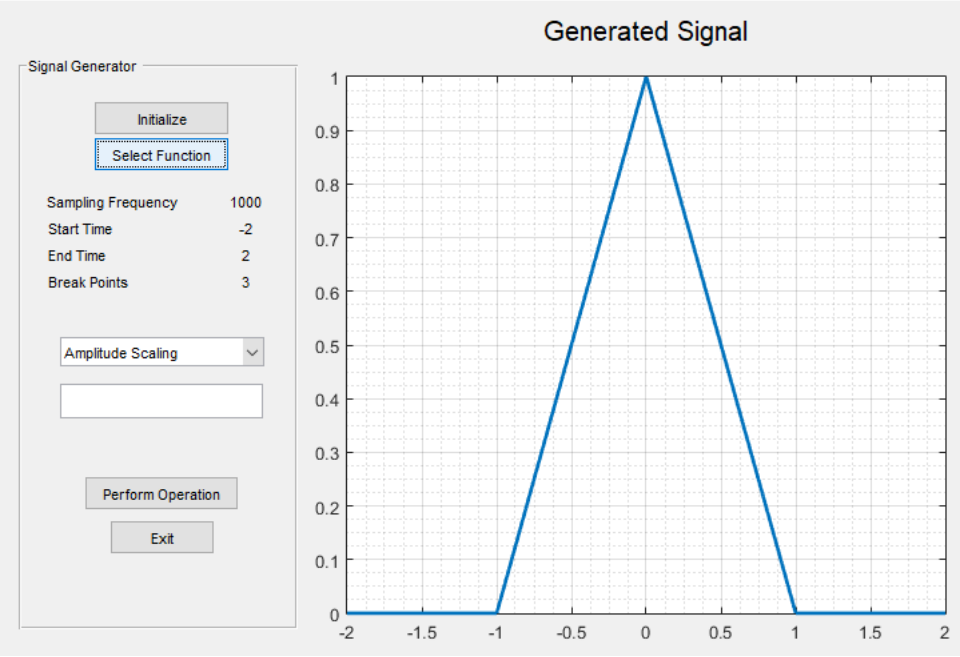


Test Cases

Case 1:

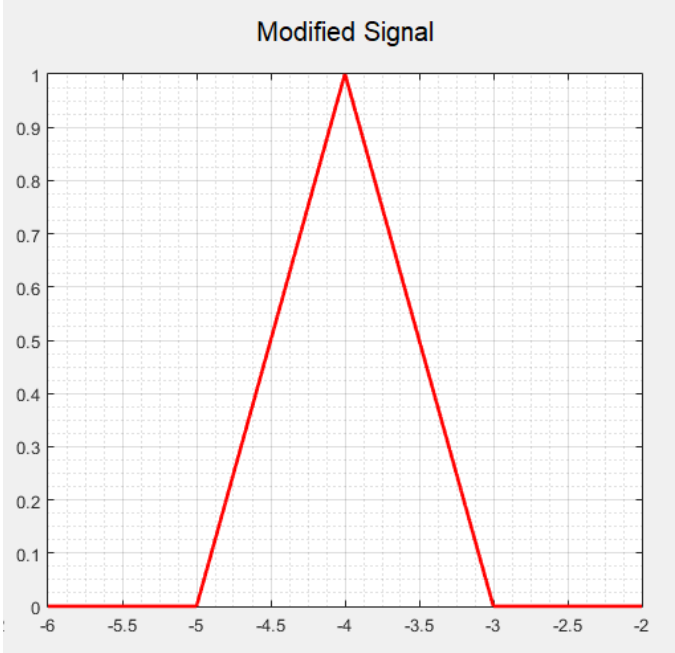
$$x(t) = r(t-1) - 2r(t) + r(t-1)$$

Sampling frequency	1000
Start time	-2
End time	2
Number of breakpoints	3 at t=-1, t=0, t=1
Signal before breakpoint 1	DC signal, amplitude 0
Signal before breakpoint 2	Ramp signal, slope 1, intercept 1
Signal before breakpoint 3	Ramp signal, slope -1, intercept 1
Signal before end	DC signal, amplitude 0

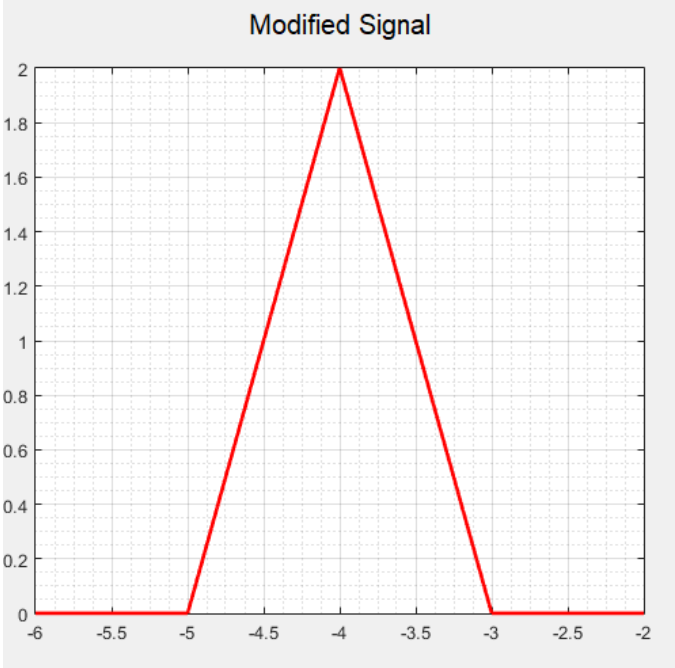


Operations on signal: [each modification is done on the previous one, modifications are made to reach a final function of **x(4-2t)**

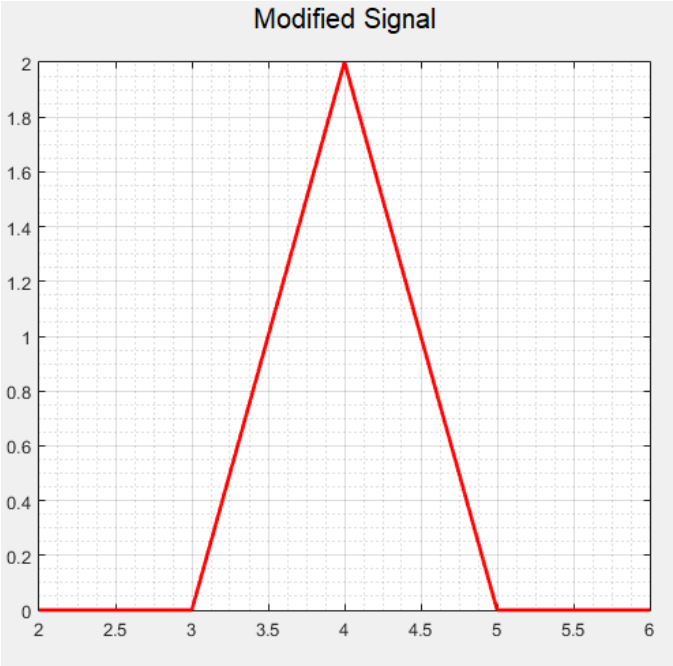
1. Time shifting: value 4



2. Amplitude scaling: 2



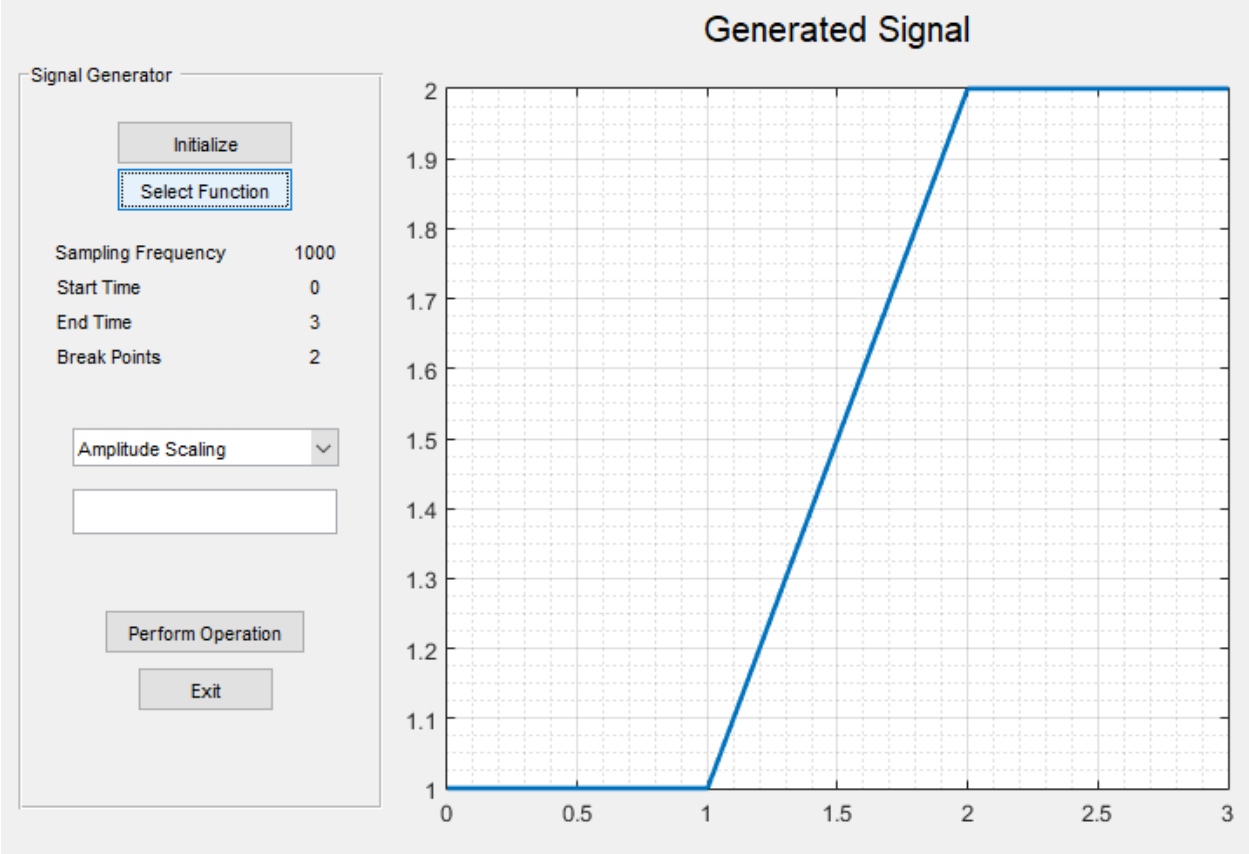
3. Time reversal



Case 2:

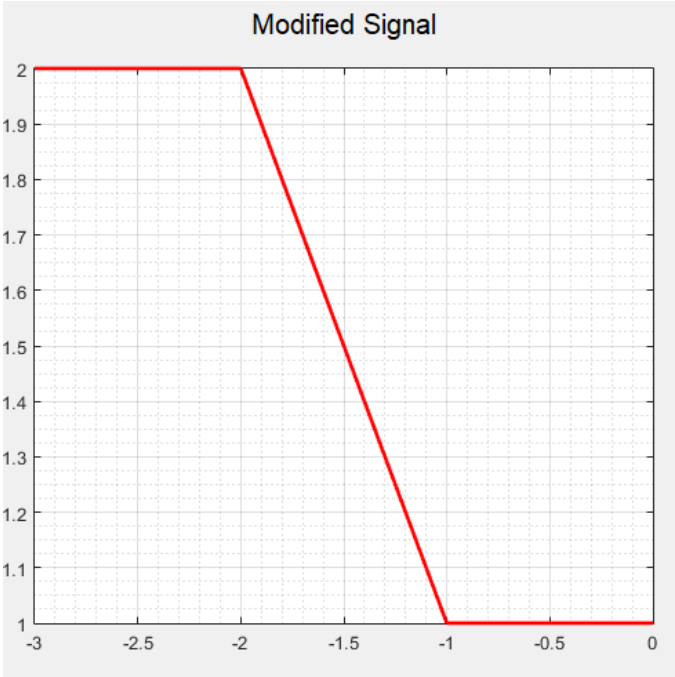
$$x(t) = u(t) + r(t-1) - r(t-2) - 2u(t-3)$$

Sampling frequency	1000
Start time	0
End time	3
Number of breakpoints	2 at t=1, t=2
Signal before breakpoint 1	DC signal, amplitude 1
Signal before breakpoint 2	Ramp signal, slope 1, intercept 0
Signal before end	DC signal, amplitude 2

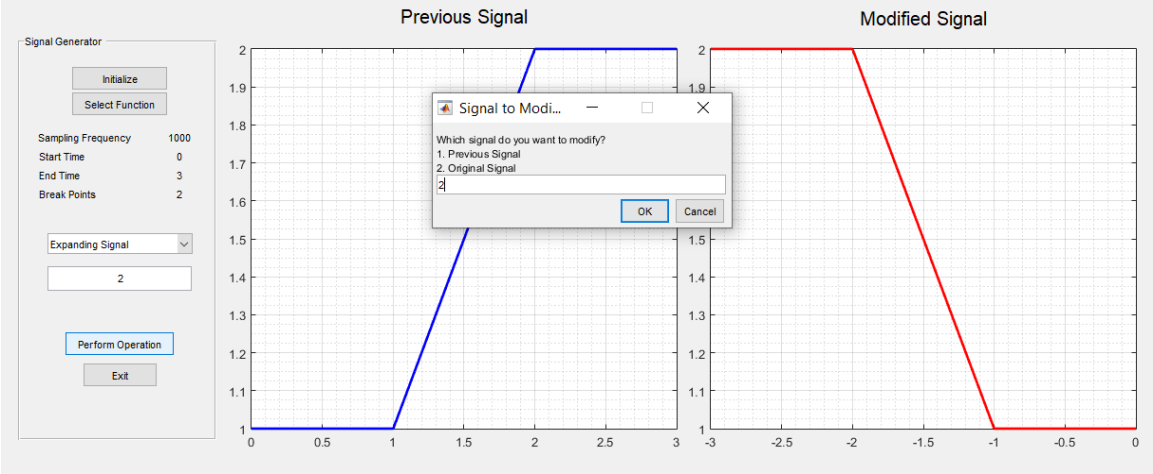


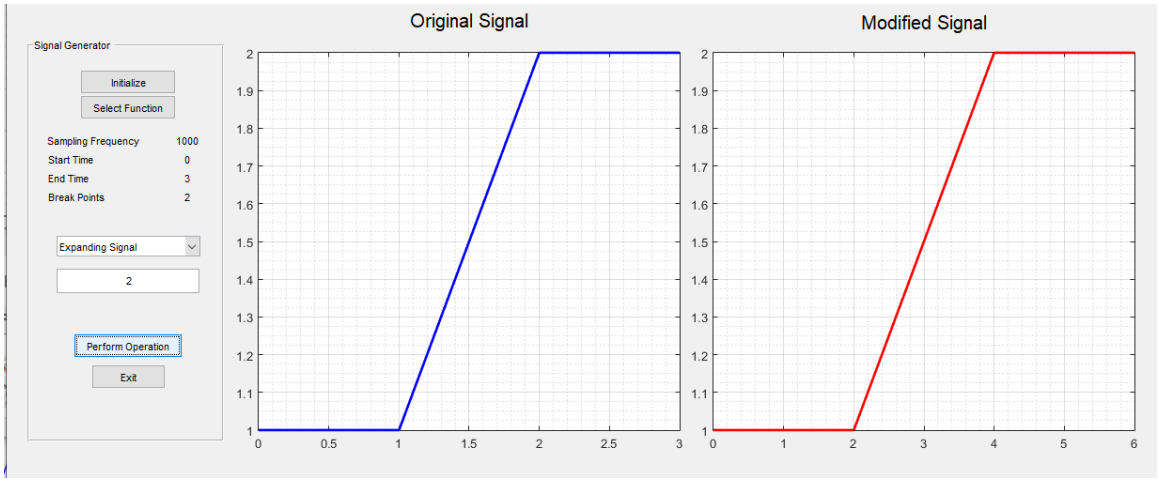
Operations:

1. Time reversal

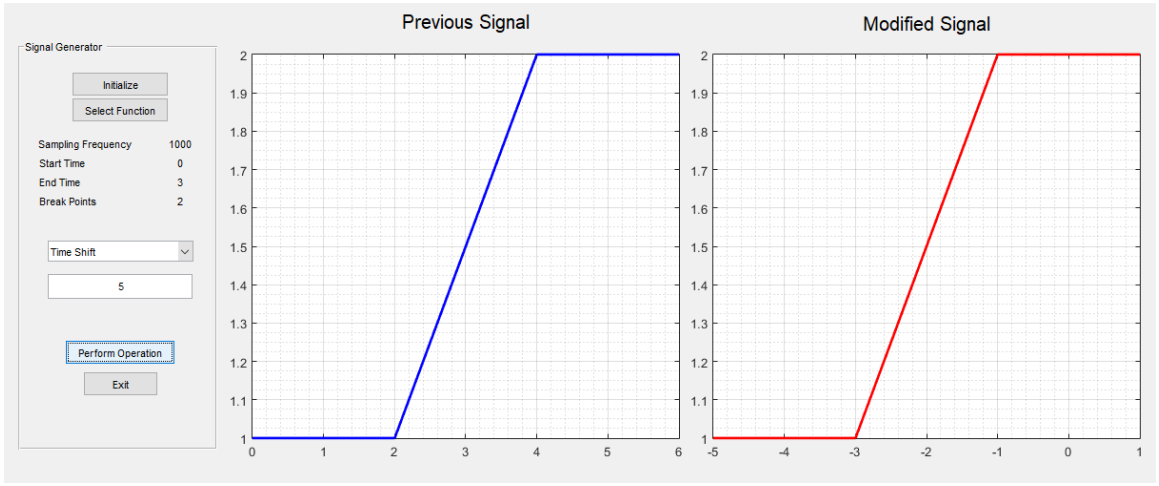


2. Expanding **ORIGINAL** signal by 4





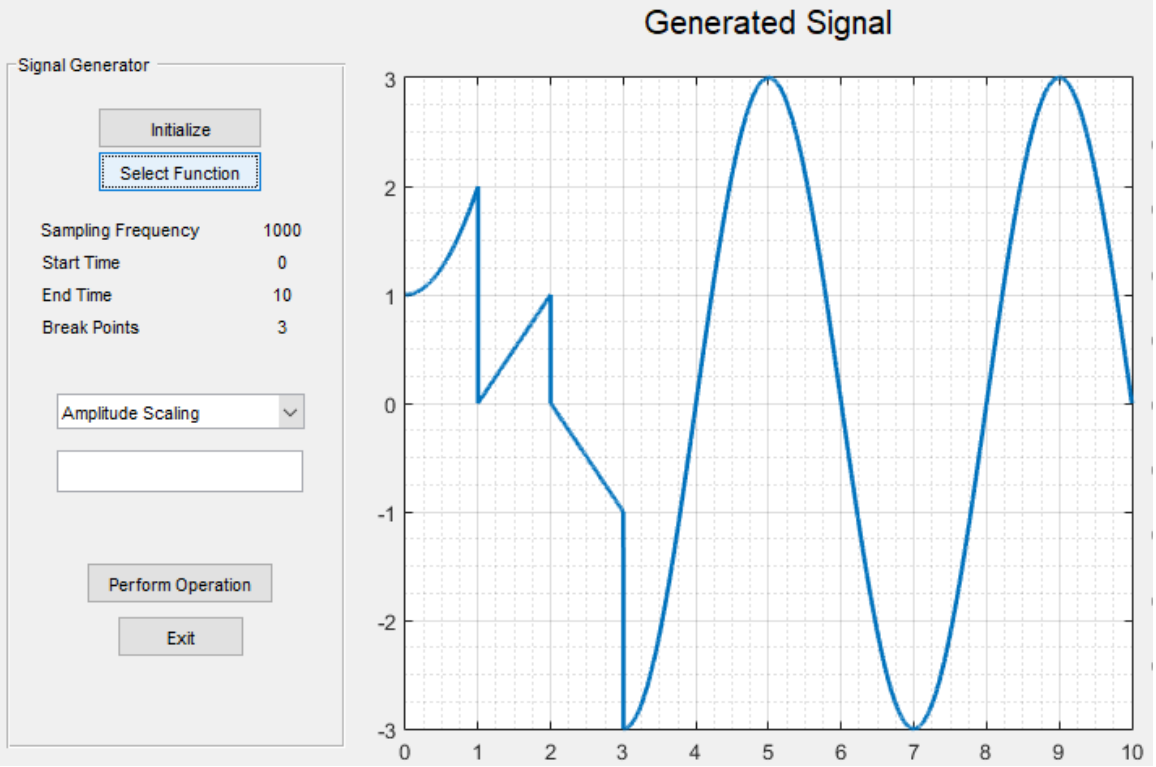
3. Time shift by 5



Case 3:

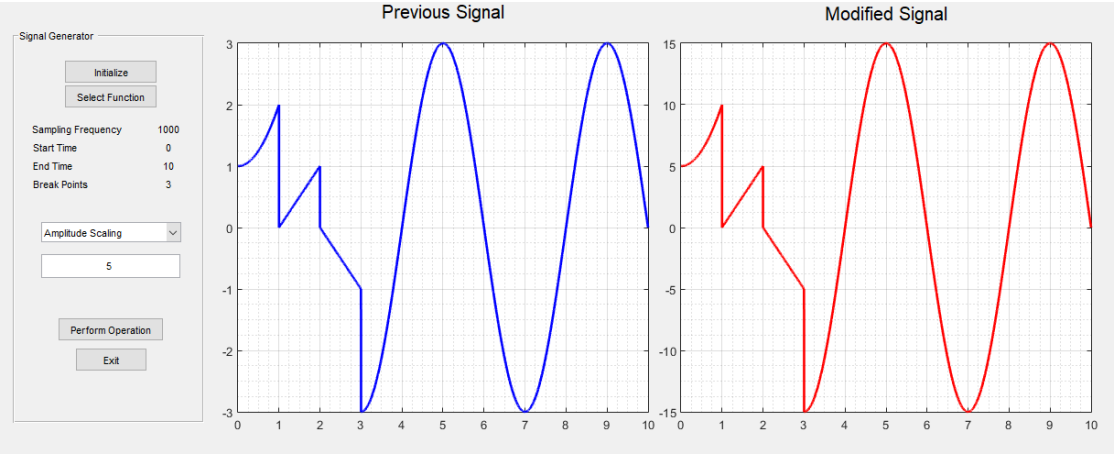
$$x(t) = (x^2 + 1)(u(t) - u(t-1)) + r(t)(u(t-1) - (t-2)) - r(t)(u(t-2) - y(t-3)) + 3sin(2 * 0.25 * \pi)$$

Sampling frequency	1000
Start time	0
End time	10
Number of breakpoints	3 at t=1, t=2, t=3
Signal before breakpoint 1	Polynomial with power 2, coefficients 1,0, intercept 1
Signal before breakpoint 2	Ramp signal, slope 1, intercept -1
Signal before breakpoint 3	Ramp signal, slope -1, intercept 2
Signal before end	Sinusoidal signal, amplitude 3, frequency 0.25 and phase 0

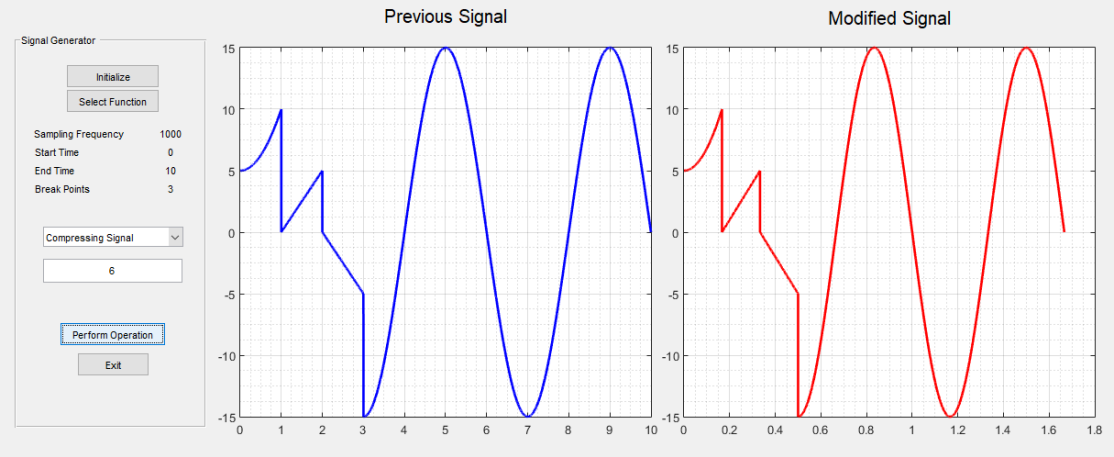


Operations:

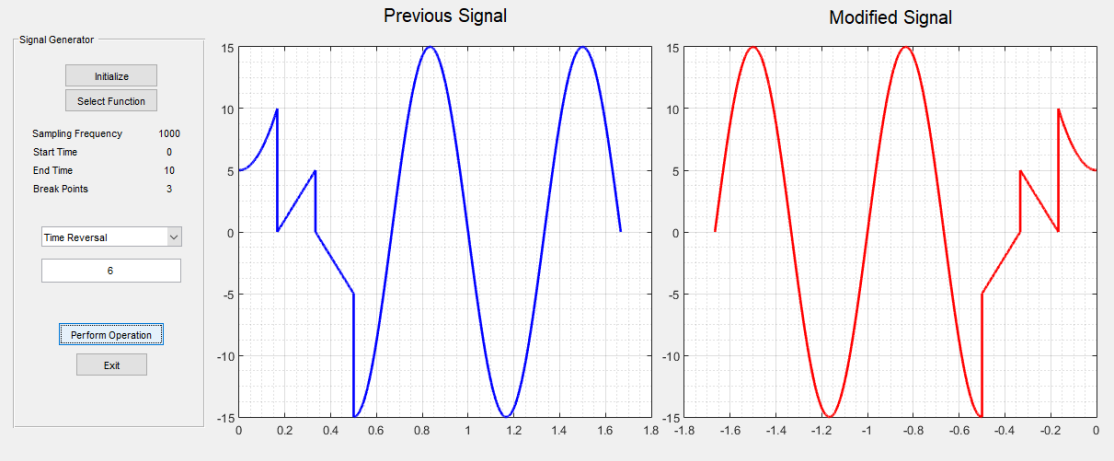
1. Amplitude scaling by 5



2. Compression by 6



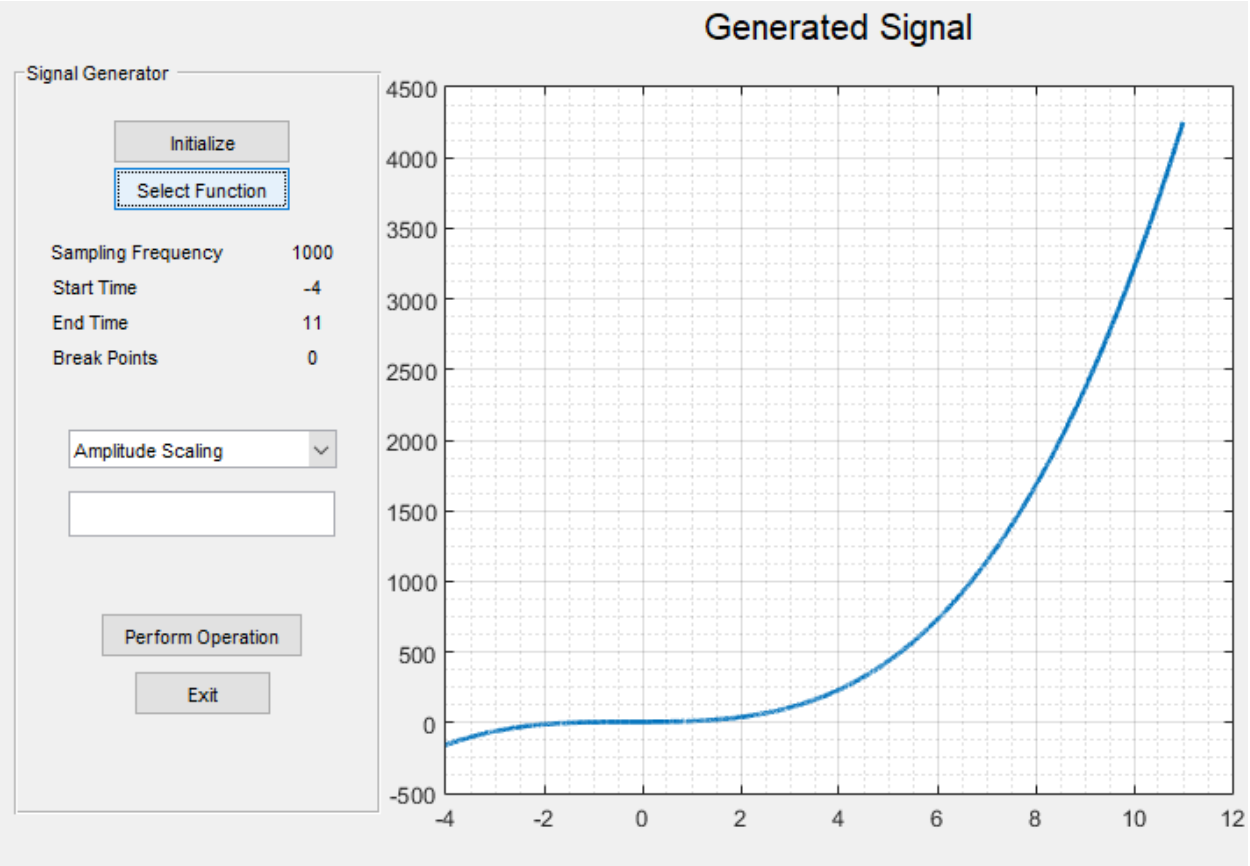
3. Time reversal



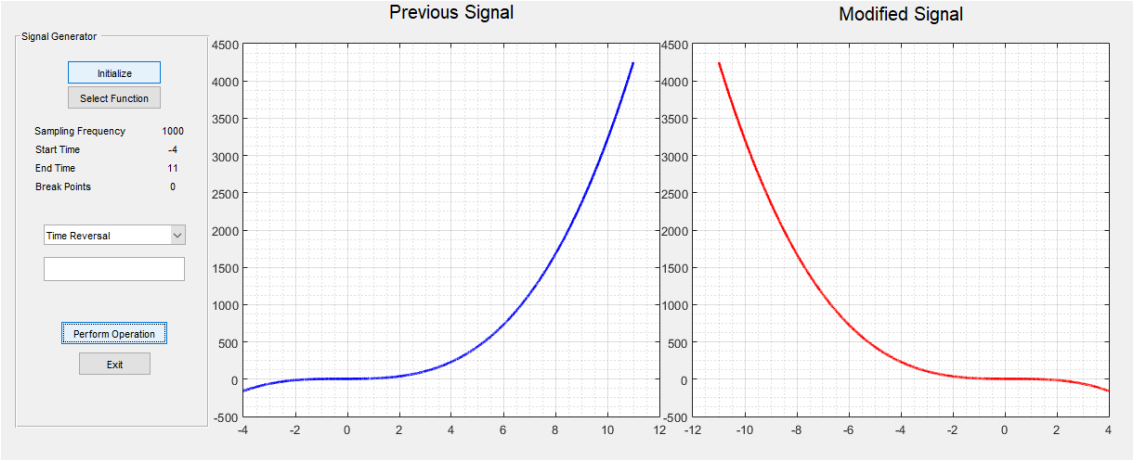
Case 4:

$$x(t) = (3t^3 + 2t^2 + t + 5)(u(t + 4) - u(t - 11))$$

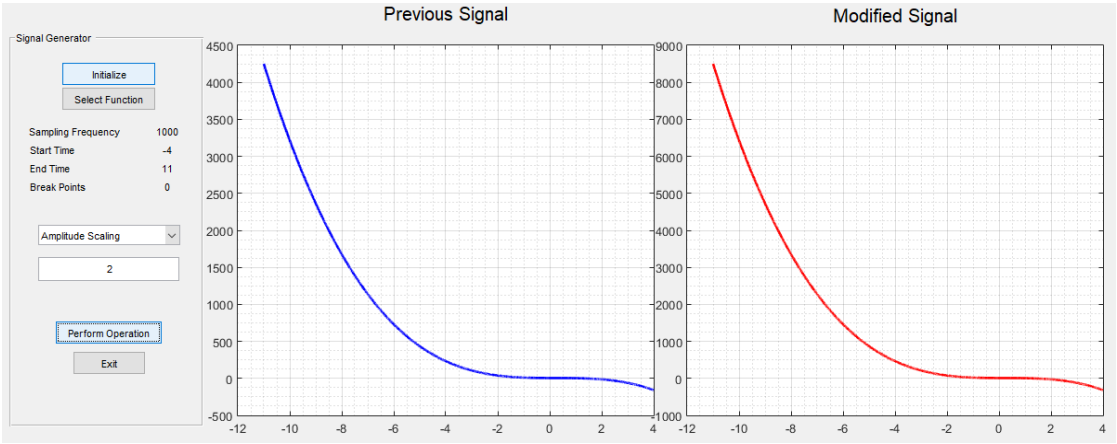
Sampling frequency	1000
Start time	-4
End time	11
Number of breakpoints	0



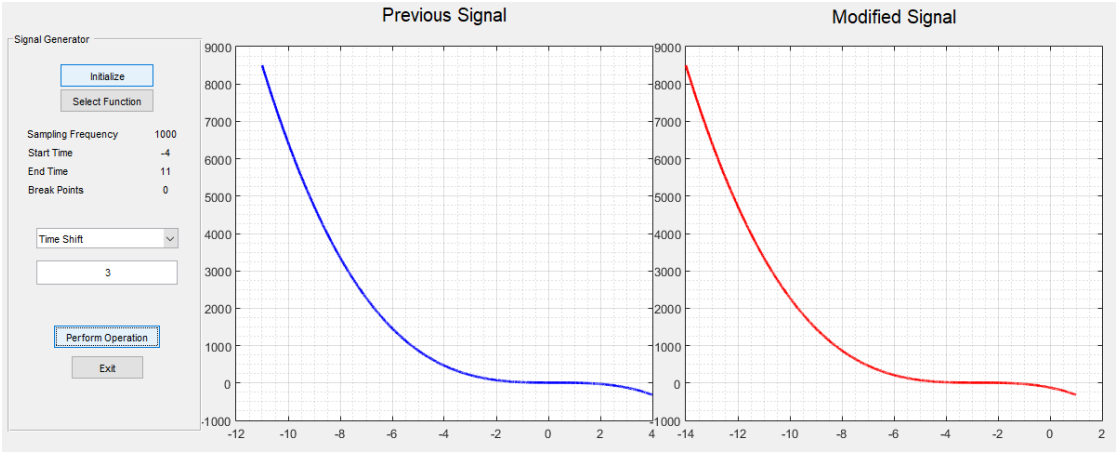
1. Time reversal



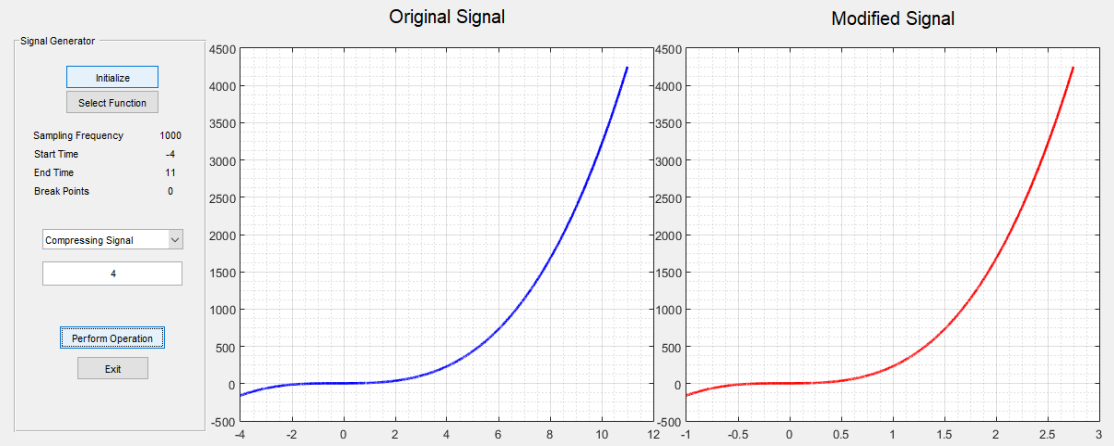
2. Amplitude scaling by 2



3. Shifting by 3

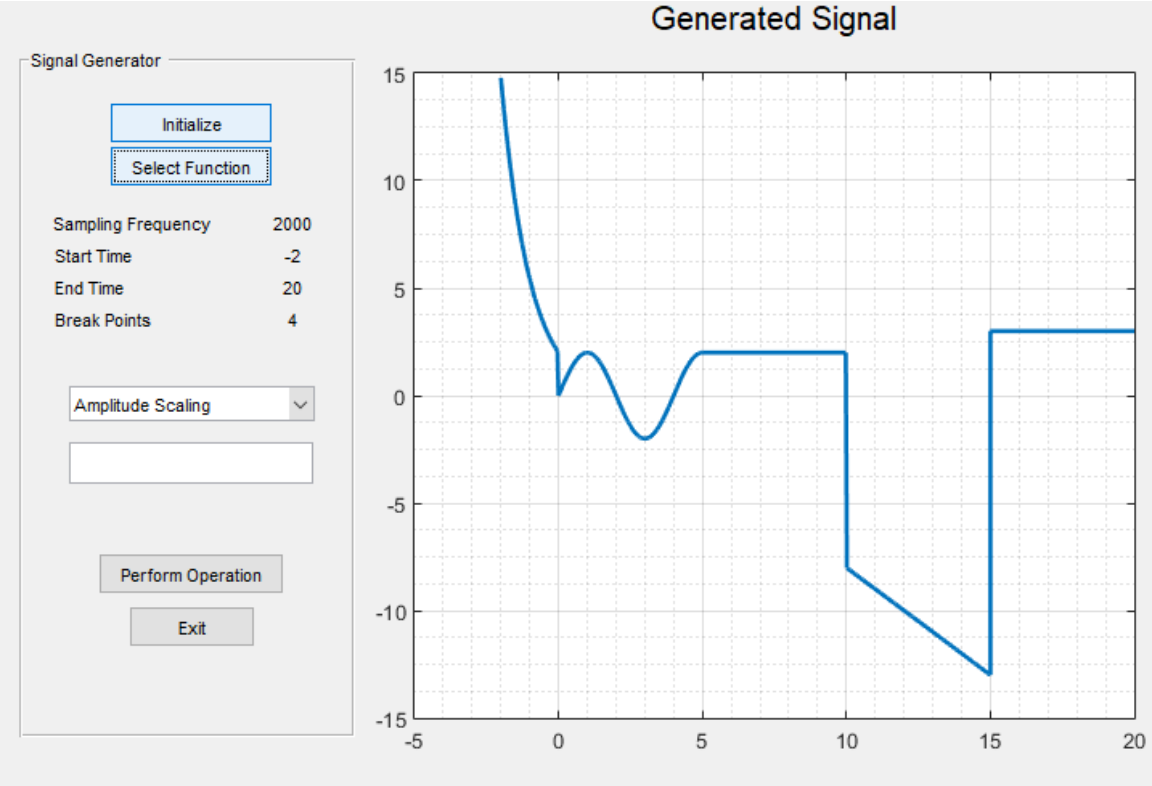


4. Compressing by 4 **THE ORIGINAL SIGNAL**



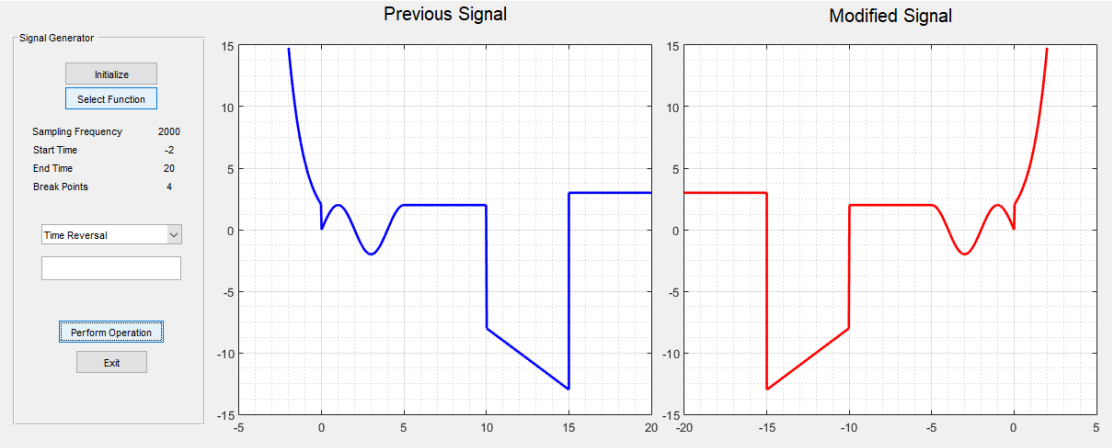
Case 5:

Sampling frequency	2000
Start time	-2
End time	20
Number of breakpoints	4 at t=0, t=5, t=10, t=15
Signal before breakpoint 1	Exponential with amplitude 2 and power -1
Signal before breakpoint 2	Sinusoidal with amplitude 2, frequency 0.3, phase 3.14/2
Signal before breakpoint 3	DC signal with amplitude 2
Signal before breakpoint	Ramp signal with slope -1 and intercept 2
Signal before end	Dc with amplitude 3

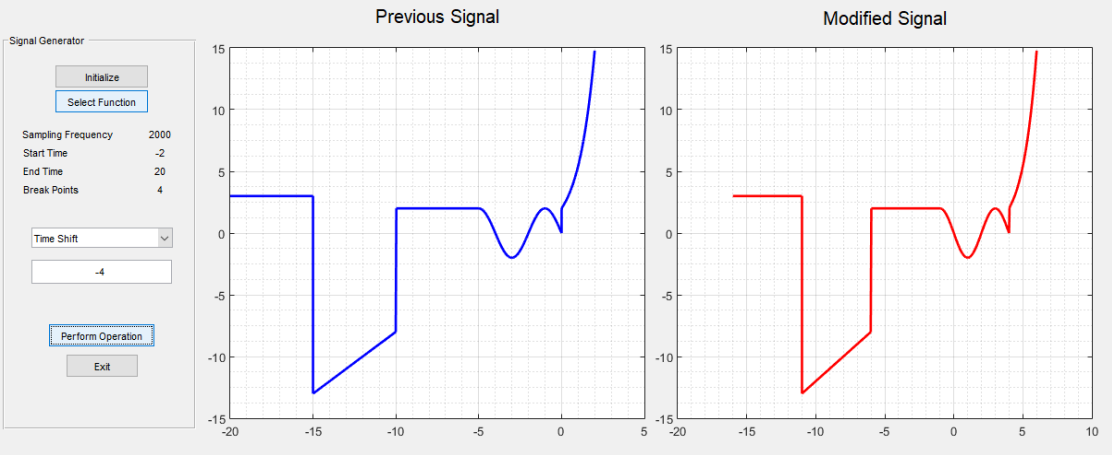


Operations:

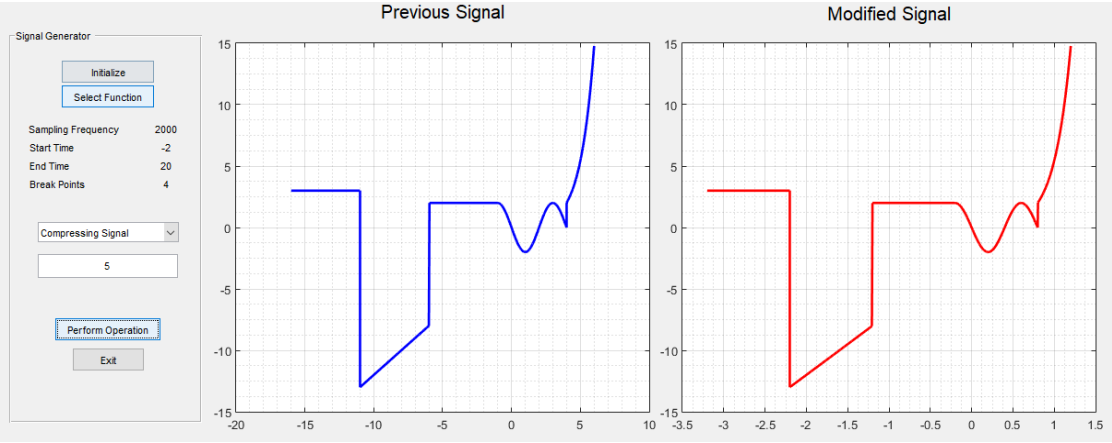
1. Time reversal



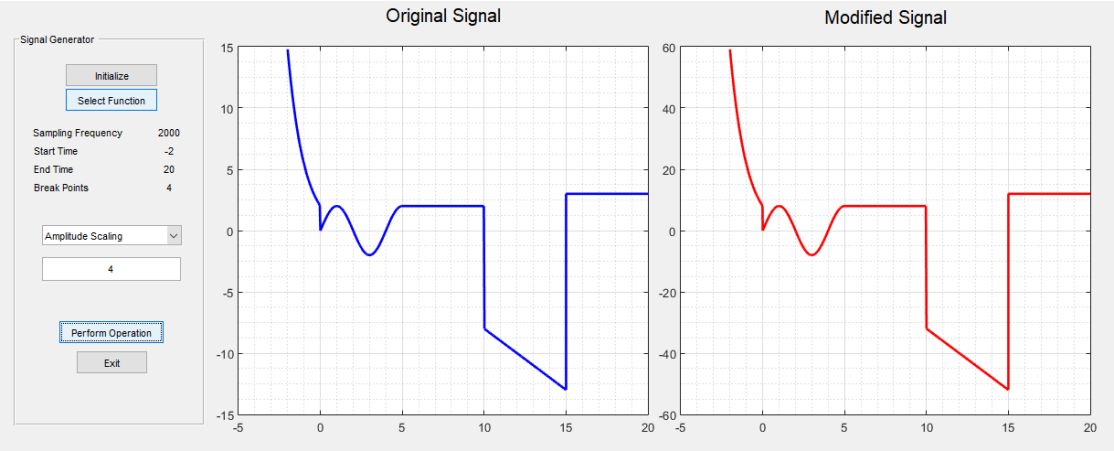
2. Time shift by -4



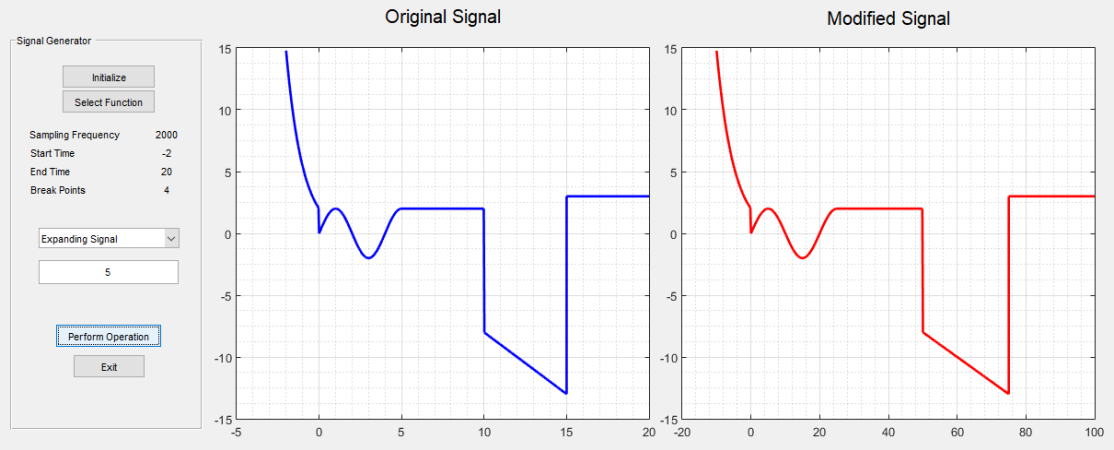
3. Time compression by 5



4. Amplitude scaling by 5 for **ORIGINAL FUNCTION**



5. **Expansion by 5 for original function**



Source Code

GUI COMMENTS ARE REMOVED

```
function varargout = GUI(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
assignin('base', 'flag', 0);
assignin('base', 'flagModification', 0);
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function popUpMenu_Callback(hObject, eventdata, handles)
function popUpMenu_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function exitButton_Callback(hObject, eventdata, handles)
close all
function pushbutton2_Callback(hObject, eventdata, handles)

function pushbutton3_Callback(hObject, eventdata, handles)
function performOpButton_Callback(hObject, eventdata, handles)
operationChoice = get(handles.popUpMenu, 'Value');
```

```
value = str2double(get(handles.valueText, 'String'));
flag = evalin('base','flag');
flagModification = evalin('base','flagModification');
if flag == 0
    errordlg('Please generate a signal to modify', 'Invalid Function');
    return
end
if isnan(value) && operationChoice ~= 2
    errordlg('Please specify an integer as Value','Invalid Input');
    return
end
if flagModification == 0
    signal = evalin('base','signal');
    t = evalin('base','t');
    set(handles.generatedSignalText, 'String', 'Previous Signal');
set(handles.modifiedSignalText, 'String', 'Modified Signal');
else
    prompt = {['Which signal do you want to modify?' sprintf('\n') '1. Previous
Signal' sprintf('\n') '2. Original Signal']};
    dlgtitle = 'Signal to Modify';
    dims = [1 50];
    definput = {''};
    choice = str2double(inputdlg(prompt, dlgtitle, dims, definput));
       %Error Handling to avoid invalid negative inputs or non numeric values.
    while choice ~= 1 && choice ~= 2
        prompt = {['Try Again, enter a valid option.' sprintf('\n') '1. Previous
Signal' sprintf('\n') '2. Original Signal']};
        dlgtitle = 'Invalid Input!';
        dims = [1 50];
        definput = {'1'};
        choice = str2double(inputdlg(prompt, dlgtitle, dims, definput));
    end
    if choice == 1
        signal = evalin('base','signal');
        t = evalin('base','t');
        set(handles.generatedSignalText, 'String', 'Previous Signal');
        set(handles.modifiedSignalText, 'String', 'Modified Signal');
    else
        signal = evalin('base','oldSignal');
        t = evalin('base','oldInterval');
        set(handles.generatedSignalText, 'String', 'Original Signal');
        set(handles.modifiedSignalText, 'String', 'Modified Signal');
```

```
    end
end
oldSignal = signal;
modifiedSignal = oldSignal;
oldInterval = t;
newInterval = oldInterval;

switch operationChoice
    case 1 %Amplitude Scaling
        scaleValue = value;
        modifiedSignal = oldSignal*scaleValue;
    case 2 %Time Reversal
        newInterval = oldInterval*-1;
    case 3 %Time Shift
        shiftValue = value;
        newInterval = oldInterval - shiftValue;
    case 4 %Expanding the signal
        expandingValue = value;
        newInterval = oldInterval*expandingValue;
    case 5 %Compressing the signal
        compressingValue = value;
        if compressingValue == 0
            errordlg('Not possible to divide by 0.','Mathematical Error');
            return
        end
        newInterval = oldInterval/compressingValue;
end
plot(handles.axes1, oldInterval, oldSignal(1:length(oldInterval)), 'blue',
'linewidth', 2)
grid(handles.axes1, 'on')
grid(handles.axes1,'minor')
plot(handles.axes2, newInterval, modifiedSignal(1:length(newInterval)), 'red',
'linewidth', 2)
grid(handles.axes2, 'on')
grid(handles.axes2,'minor')
assignin('base', 'signal', modifiedSignal);
assignin('base', 't', newInterval);

if flagModification == 0
    assignin('base', 'oldSignal', oldSignal);
    assignin('base', 'oldInterval', oldInterval);
```

```
    assignin('base', 'flagModification', 1);
end

function valueText_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function plotButton_Callback(hObject, eventdata, handles)
function startButton_Callback(hObject, eventdata, handles)
prompt = {'Enter the sampling frequency: '};
dlgtitle = 'Sampling Frequency';
dims = [1 50];
definput = {''};
samplingFreq = str2double(inputdlg(prompt, dlgtitle, dims, definput));

%Error Handling to avoid invalid negative inputs or non numeric values.
while (samplingFreq<=0) || isnan(samplingFreq)
    prompt = {['Try Again, enter a valid positive number.' sprintf('\n') 'Enter the
sampling frequency: ' sprintf('\n')]};
    dlgtitle = 'Invalid Input!';
    dims = [1 50];
    definput = {'20','hsv'};
    samplingFreq = str2double(inputdlg(prompt,dlgtitle,dims,definput));
end
prompt = {'Enter the starting time:', 'Enter the ending time:'};
dlgtitle = 'Time';
dims = [1 50];
definput = {'',''};
time = inputdlg(prompt,dlgtitle,dims,definput);
startTime = str2double(time(1));
endTime = str2double(time(2));

%Error Handling to avoid non numeric values.
while isnan(startTime)
    prompt = {['Try Again, time must be a number.' sprintf('\n') 'Enter the starting
time: ']};
    dlgtitle = 'Invalid Input!';
    dims = [1 50];
    definput = {'-5'};
    startTime = str2double(inputdlg(prompt,dlgtitle,dims,definput));
end
```

```
%Error Handling to avoid non numeric values.
while isnan(endTime)
    prompt = {['Try Again, time must be a number.' sprintf('\n') 'Enter the ending
time: ']};
    dlgtitle = 'Invalid Input!';
    dims = [1 50];
    definput = {'5'};
    endTime = str2double(inputdlg(prompt,dlgtitle,dims,definput));
end
 %Error Handling to avoid illogical times.
while (startTime>=endTime)
prompt = {'Try Again, Start time must be < End Time. Enter the starting time:',
'Enter the ending time:'};
dlgtitle = 'Time';
dims = [1 50];
definput = {'',''};
time = inputdlg(prompt,dlgtitle,dims,definput);
startTime = str2double(time(1));
endTime = str2double(time(2));

    %Error Handling to avoid non numeric values.
    while isnan(startTime)
        prompt = {['Try Again, time must be a number.' sprintf('\n') 'Enter the
starting time: ']};
        dlgtitle = 'Invalid Input!';
        dims = [1 50];
        definput = {'-5'};
        startTime = str2double(inputdlg(prompt,dlgtitle,dims,definput));
    end

    %Error Handling to avoid non numeric values.
    while isnan(endTime)
        prompt = {['Try Again, time must be a number.' sprintf('\n') 'Enter the
ending time: ']};
        dlgtitle = 'Invalid Input!';
        dims = [1 50];
        definput = {'5'};
        endTime = str2double(inputdlg(prompt,dlgtitle,dims,definput));
    end
end
```

```
prompt = {['Enter the number of break points: ']};
dlgtitle = 'Sampling Frequency';
dims = [1 50];
definput = {''};
numberBreakPoints = str2double(inputdlg(prompt,dlgtitle,dims,definput));

%Error Handling to avoid non numeric values.
while isnan(numberBreakPoints)
    prompt = {['Try Again, enter a number.' sprintf('\n') 'Enter the number of break
points: ']};
    dlgtitle = 'Invalid Input';
    dims = [1 50];
    definput = {'1'};
    numberBreakPoints = str2double(inputdlg(prompt,dlgtitle,dims,definput));
end
breakPoints = zeros(1, numberBreakPoints);
%Prompting user to define the exact positions of breakpoints, and
%setting time axis values and number of samples
if numberBreakPoints == 0
    t = linspace(startTime, endTime, (endTime-startTime)*samplingFreq);
else
    for i=0:numberBreakPoints-1
        prompt = {['Break Point number ' num2str(i+1) ' at t = ']};
        dlgtitle = 'Break Point';
        dims = [1 50];
        definput = {'0'};
        breakPoints(i+1) = str2double(inputdlg(prompt,dlgtitle,dims,definput));
        while isnan(breakPoints(i+1)) || breakPoints(i+1)<startTime
||breakPoints(i+1) >endTime
            prompt = {['Try again.' sprintf('\n') 'Break Point number ' num2str(i+1)
' at t = ']};
            dlgtitle = 'Invalid Range or Format';
            dims = [1 50];
            definput = {'0'};
            breakPoints(i+1) = str2double(inputdlg(prompt,dlgtitle,dims,definput));
        end
        if i > 0
            while breakPoints(i+1) <= breakPoints(i) || breakPoints(i+1) >= endTime
|| breakPoints(i+1) <= startTime
                prompt = {['Try again.' sprintf('\n') 'Break Point number '
num2str(i+1) ' at t = ']};
                dlgtitle = 'Please enter breakpoints in chronological order. ';
```

```
                dims = [1 50];
                definput = {'0'};
                breakPoints(i+1) =
str2double(inputdlg(prompt,dlgtitle,dims,definput));
            end
        end
    end

    t = linspace(startTime, breakPoints(1), (breakPoints(1)-startTime)*samplingFreq);
    %First range of t that will be used before the first breakpoint to
    %draw the initial signal
end
assignin('base', 'samplingFreq', samplingFreq);
assignin('base', 'startTime', startTime);
assignin('base', 'endTime', endTime);
assignin('base', 'numberBreakPoints', numberBreakPoints);
assignin('base', 't', t);
assignin('base', 'breakPoints', breakPoints);
set(handles.sampleValue, 'String', samplingFreq);
set(handles.startValue, 'String', startTime);
set(handles.endValue, 'String', endTime);
set(handles.breakValue, 'String', numberBreakPoints);
function selectFunctionButton_Callback(hObject, eventdata, handles)
signal = [];   %vector of the signal that will be plotted
samplingFreq = evalin('base','samplingFreq');
startTime = evalin('base','startTime');
endTime = evalin('base','endTime');
t = evalin('base','t');
numberBreakPoints = evalin('base','numberBreakPoints');
breakPoints = evalin('base','breakPoints');
for i=1:numberBreakPoints+1
    if numberBreakPoints == 0
        prompt = {['Choose the specifications of the signal: ' sprintf('\n') '1)   DC
Signal' sprintf('\n') '2)   Ramp signal' sprintf('\n') '3)   General order
polynomial' sprintf('\n') '4)   Exponential signal' sprintf('\n') '5)   Sinusoidal
signal']};
        dlgtitle = 'Input Specifications';
        dims = [1 50];
        definput = {'1'};
        choice = str2double(inputdlg(prompt,dlgtitle,dims,definput));
    elseif numberBreakPoints>0
        prompt = {['Choose the specifications of the signal before breakpoint number
```

```
' num2str(i) ':' sprintf('\n') '1)   DC Signal' sprintf('\n') '2)   Ramp signal'
sprintf('\n') '3)   General order polynomial' sprintf('\n') '4)   Exponential signal'
sprintf('\n') '5)   Sinusoidal signal']};
        dlgtitle = 'Input Specifications';
        dims = [1 50];
        definput = {'1'};
        choice = str2double(inputdlg(prompt,dlgtitle,dims,definput));
    end
    %Error Handling to avoid invalid inputs or non numeric values.
    while choice<1 || choice>5 || isnan(choice)
        prompt = {['Try Again, enter a valid positive number between 1 and 5'
sprintf('\n')]};
        dlgtitle = 'Input Invalid';
        dims = [1 50];
        definput = {'1'};
        choice = str2double(inputdlg(prompt,dlgtitle,dims,definput));
    End
    switch choice

    case 1      %DC signal
        prompt = {['Enter the DC signal''s amplitude: ' sprintf('\n')]};
        dlgtitle = 'DC Signal';
        dims = [1 50];
        definput = {'0'};
        amplitudeDC = str2double(inputdlg(prompt,dlgtitle,dims,definput));
        f(1:length(t))= amplitudeDC;

    case 2      %Ramp signal
        prompt = {['Enter the Ramps signal''s slope: ' sprintf('\n')], ['Enter the
Ramps signal''s intercept: ' sprintf('\n')]};
        dlgtitle = 'Ramp Signal';
        dims = [1 50];
        definput = {'0', '0'};
        array = str2double(inputdlg(prompt,dlgtitle,dims,definput));
        slope = array(1);
        intercept = array(2);
        f  = (slope.*(t))+ intercept;

    case 3      %General Order Polynomial
        prompt = {['Enter the General Order Polynomial''s power: ' sprintf('\n')]};
        dlgtitle = 'General Order Polynomial';
        dims = [1 50];
```

```
        definput = {'0'};
        power = str2double(inputdlg(prompt,dlgtitle,dims,definput));
        amplitudepolynomial = zeros(1, power+1);
        y = 1;

        for x=power:-1:1
            prompt = {['Enter the t^' num2str(x) ' coefficient:' sprintf('\n')]};
            dlgtitle = 'General Order Polynomial';
            dims = [1 50];
            definput = {'0'};
            amplitudepolynomial(y) =
str2double(inputdlg(prompt,dlgtitle,dims,definput));

            y = y+1;
        end
        prompt = {['Enter the General Order Polynomial''s intercept: '
sprintf('\n')]};
        dlgtitle = 'General Order Polynomial';
        dims = [1 50];
        definput = {'1'};
        intercept = str2double(inputdlg(prompt,dlgtitle,dims,definput));

        amplitudepolynomial(power+1) = intercept;
        f = polyval(amplitudepolynomial, t);

    case 4          %Exponential signal
        prompt = {['Enter the Exponential signal''s amplitude: ' sprintf('\n')],
['Enter the Exponential signal''s exponent: ' sprintf('\n')]};
        dlgtitle = 'Exponential Signal';
        dims = [1 50];
        definput = {'0', '0'};
        array = str2double(inputdlg(prompt,dlgtitle,dims,definput));

        amplitudeExponential = array(1)
        exponent = array(2)

        f  = amplitudeExponential*exp(exponent.*t);

    case 5          %Sinusoidal signal
        prompt = {['Enter the Sinusoidal signal''s amplitude: '],['Enter the
Sinusoidal signal''s frequency: '], ['Enter the Sinusoidal signal''s phase: ']};
        dlgtitle = 'Sinusoidal Signal';
```

```
        dims = [1 50];
        definput = {'0', '0', '0'};
        array = str2double(inputdlg(prompt,dlgtitle,dims,definput));

        amplitudeSinusoidal = array(1);
        frequency = array(2);
        phase = array(3);

        f = amplitudeSinusoidal*sin(2*pi*frequency.*t + phase);
    end

    %Setting time vector for next portions of the signal

    if i+1 <= numberBreakPoints  %end of the signal not reached yet
        t = linspace(breakPoints(i), breakPoints(i+1),
(breakPoints(i+1)-breakPoints(i))*samplingFreq);

    elseif i == numberBreakPoints  %end of the signal reached
        t = linspace(breakPoints(i), endTime,
((endTime)-breakPoints(i))*samplingFreq);

    end
    signal = [signal f];
end
t = linspace(startTime, endTime, (endTime-startTime)*samplingFreq);

set(handles.generatedSignalText, 'String', 'Generated Signal');
set(handles.modifiedSignalText, 'String', ' ');

assignin('base', 'signal', signal);
assignin('base', 'flag', 1);
assignin('base', 't', t);

plot(handles.axes1, t, signal(1:length(t)), 'linewidth', 2)
grid(handles.axes1, 'on')
grid(handles.axes1,'minor')


function valueText_Callback(hObject, eventdata, handles)
```