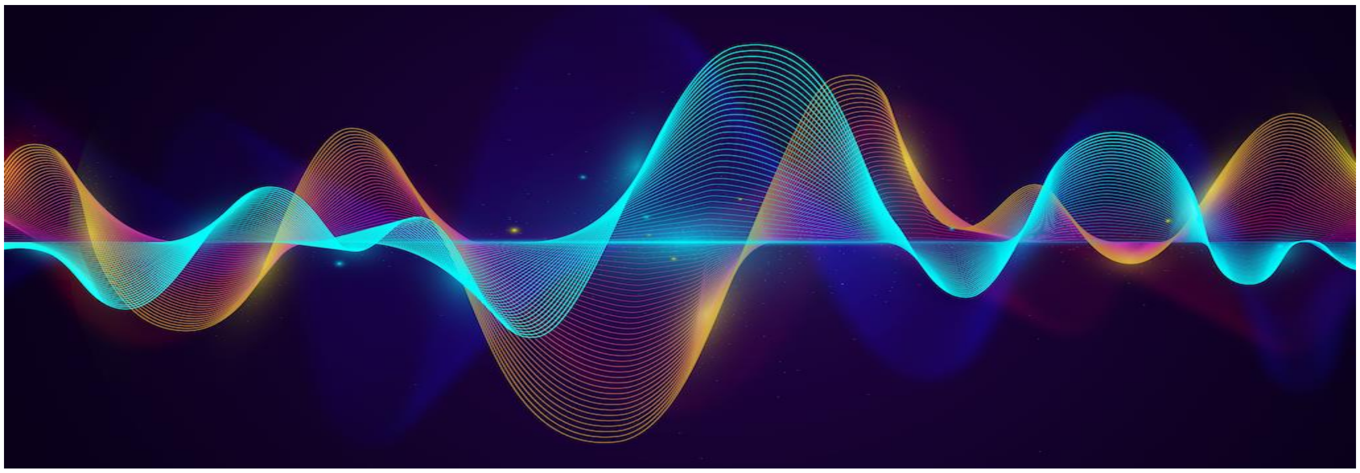
Faculty of Engineering Alexandria University

EEC271 – Signals and Systems

Electronics and Communication Engineering Major

Signal Processing using MATLAB

Term Project I: General Signal Generator

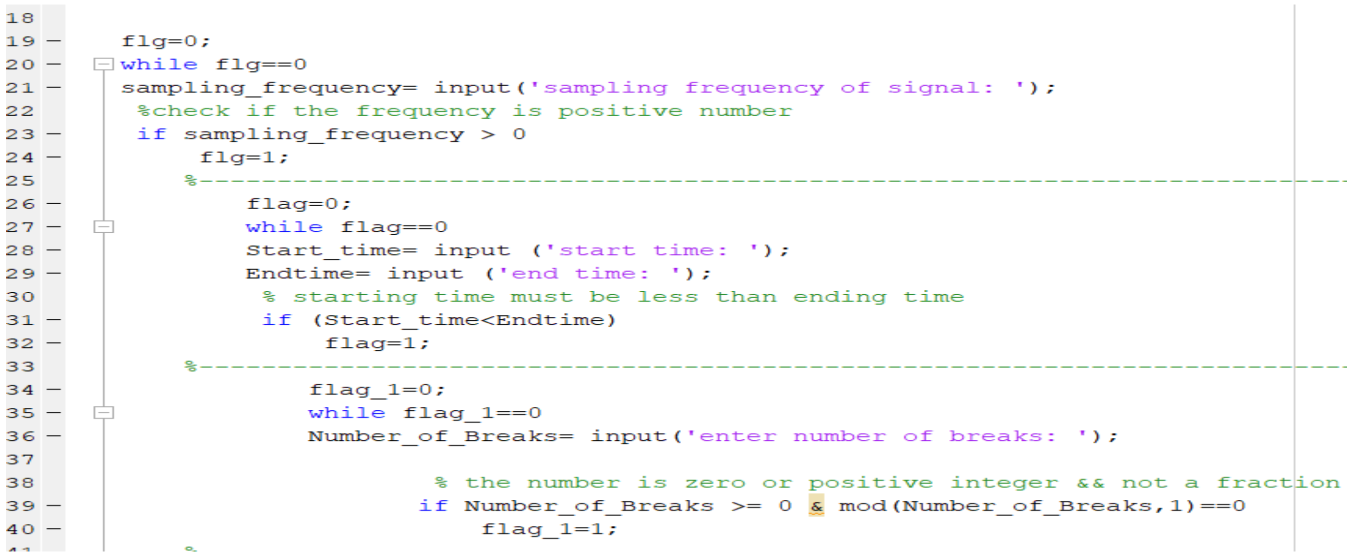


Introduction

The aim of the program is to generate different types of signals, display it, then perform some operations on it.

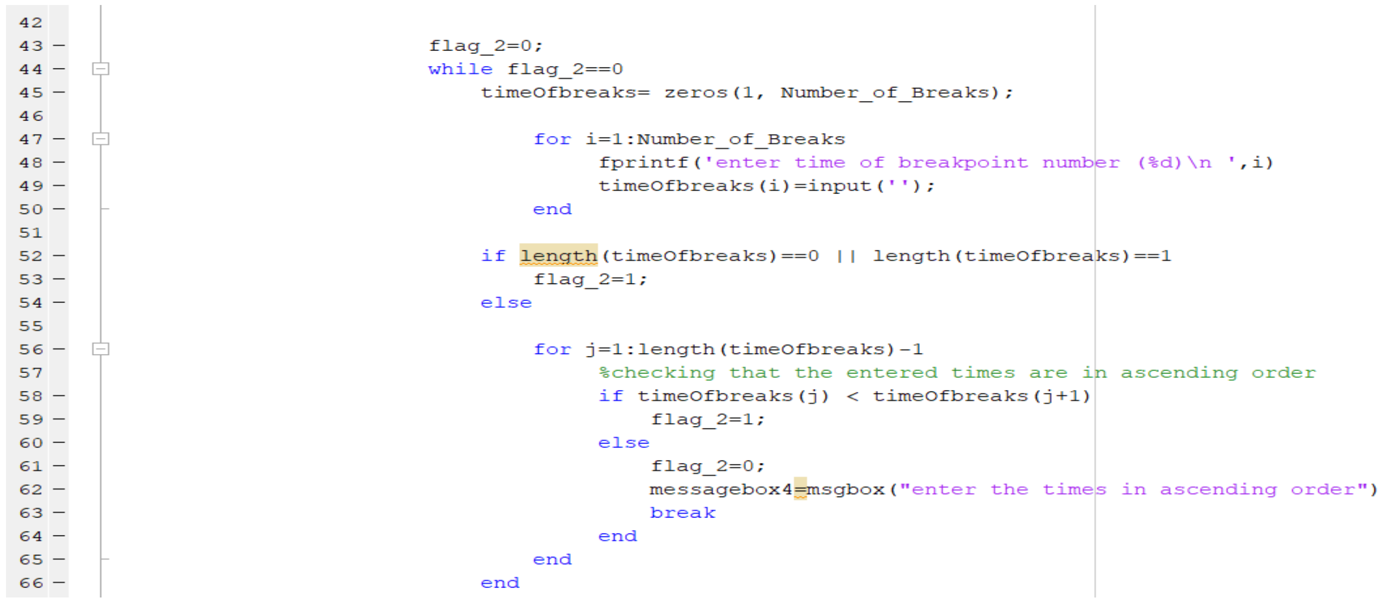
Procedures

1. **The program asks the user for the following parameters**
   - Sampling frequency of signal.
   - Start and end of time scale
   - Number of the break points and their positions (i.e., the points that the signal definition rule changes).



```
  flg=0;
while flg==0
sampling_frequency= input('sampling frequency of signal: ');
 %check if the frequency is positive number
 if sampling_frequency > 0
     flg=1;
    %-----------------------------------------------------------------------
        flag=0;
        while flag==0
        Start_time= input ('start time: ');
        Endtime= input ('end time: ');
         % starting time must be less than ending time
         if (Start_time<Endtime)
             flag=1;
    %-----------------------------------------------------------------------
            flag_1=0;
            while flag_1==0
            Number_of_Breaks= input('enter number of breaks: ');

                       % the number is zero or positive integer && not a fraction
                      if Number_of_Breaks >= 0 & mod(Number_of_Breaks,1)==0
                          flag_1=1;
    %
```

Fig. 1. Inputs (frequency, start and end time, no. of breaks) code block.



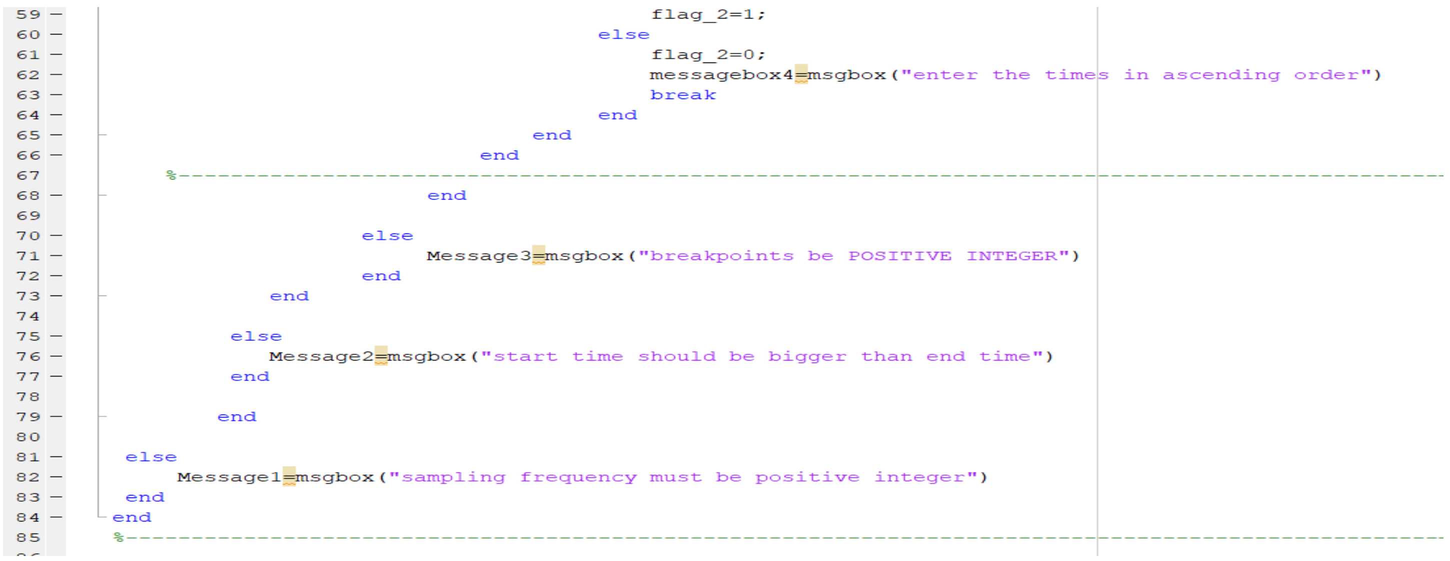
```
                         flag_2=0;
                         while flag_2==0
                             timeOfbreaks= zeros(1, Number_of_Breaks);

                                 for i=1:Number_of_Breaks
                                     fprintf('enter time of breakpoint number (%d)\n ',i)
                                     timeOfbreaks(i)=input('');
                                 end

                             if length(timeOfbreaks)==0 || length(timeOfbreaks)==1
                                 flag_2=1;
                             else

                                 for j=1:length(timeOfbreaks)-1
                                     %checking that the entered times are in ascending order
                                     if timeOfbreaks(j) < timeOfbreaks(j+1)
                                         flag_2=1;
                                     else
                                         flag_2=0;
                                         messagebox4=msgbox("enter the times in ascending order")
                                         break
                                     end
                                 end
                             end
```

Fig. 2. Inputs (positions of breakpoints) code block.

This block of code shown in figures 1,2 is for the inputs part. By using flags, the program handles the worst-case scenarios that may be entered by the user.

- **flg**: flag for checking that the frequency is correct "it should be a positive number”.
- **flag**: flag for checking that start and end times are correct "the start time must be less than the end time".
- **flag_1**: flag for checking that number of break points is correct "the number is zero or positive integer and not a fraction".
- **flag_2**: flag for checking that the entered times are in ascending order which means each break point in order has its correct time position.

If any of these are not achieved, the program appears a message box shown in figure 3 that indicates there is an error, then keep asking the user to reenter the correct inputs.

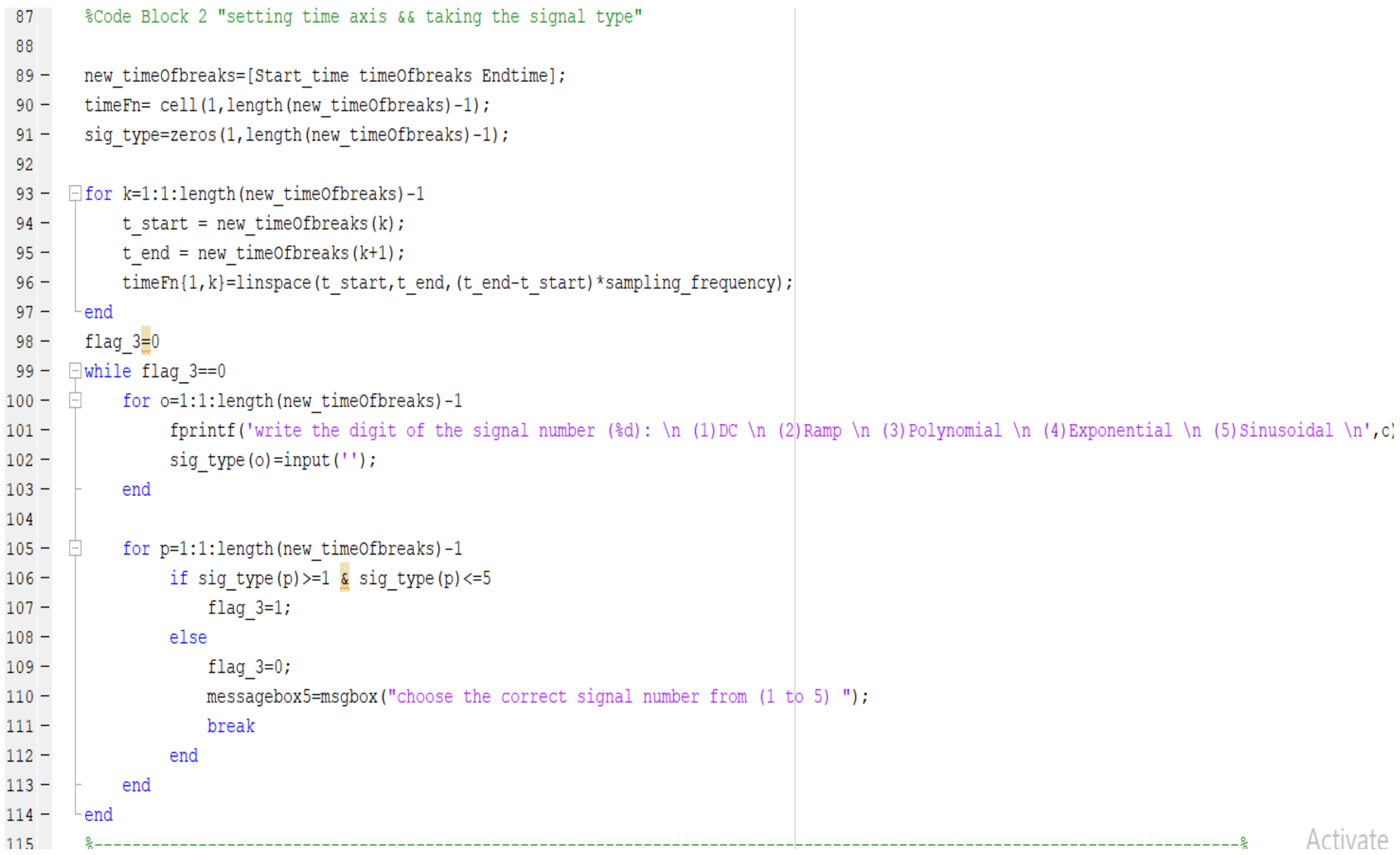


```
                                         flag_2=1;
                                     else
                                         flag_2=0;
                                         messagebox4=msgbox("enter the times in ascending order")
                                         break
                                     end
                                end
                            end
    %-------------------------------------------------------------------------------------------------
                        end

                   else
                        Message3=msgbox("breakpoints be POSITIVE INTEGER")
                   end
            end

         else
            Message2=msgbox("start time should be bigger than end time")
         end

        end

 else
     Message1=msgbox("sampling frequency must be positive integer")
 end
end
%-----------------------------------------------------------------------------------------------------
```

Fig. 3. The code block of message boxes.

2. **According to the number of breakpoints the program asks the user at each region to enter the specifications of the signal at this region Which are**:
   a. DC signal: Amplitude.
   b. Ramp signal: slope – intercept.
   c. General order polynomial: Amplitude-power – intercept.
   d. Exponential signal: Amplitude – exponent.
   e. Sinusoidal signal: Amplitude – frequency – phase.

3. Display the resulting signal in time domain.



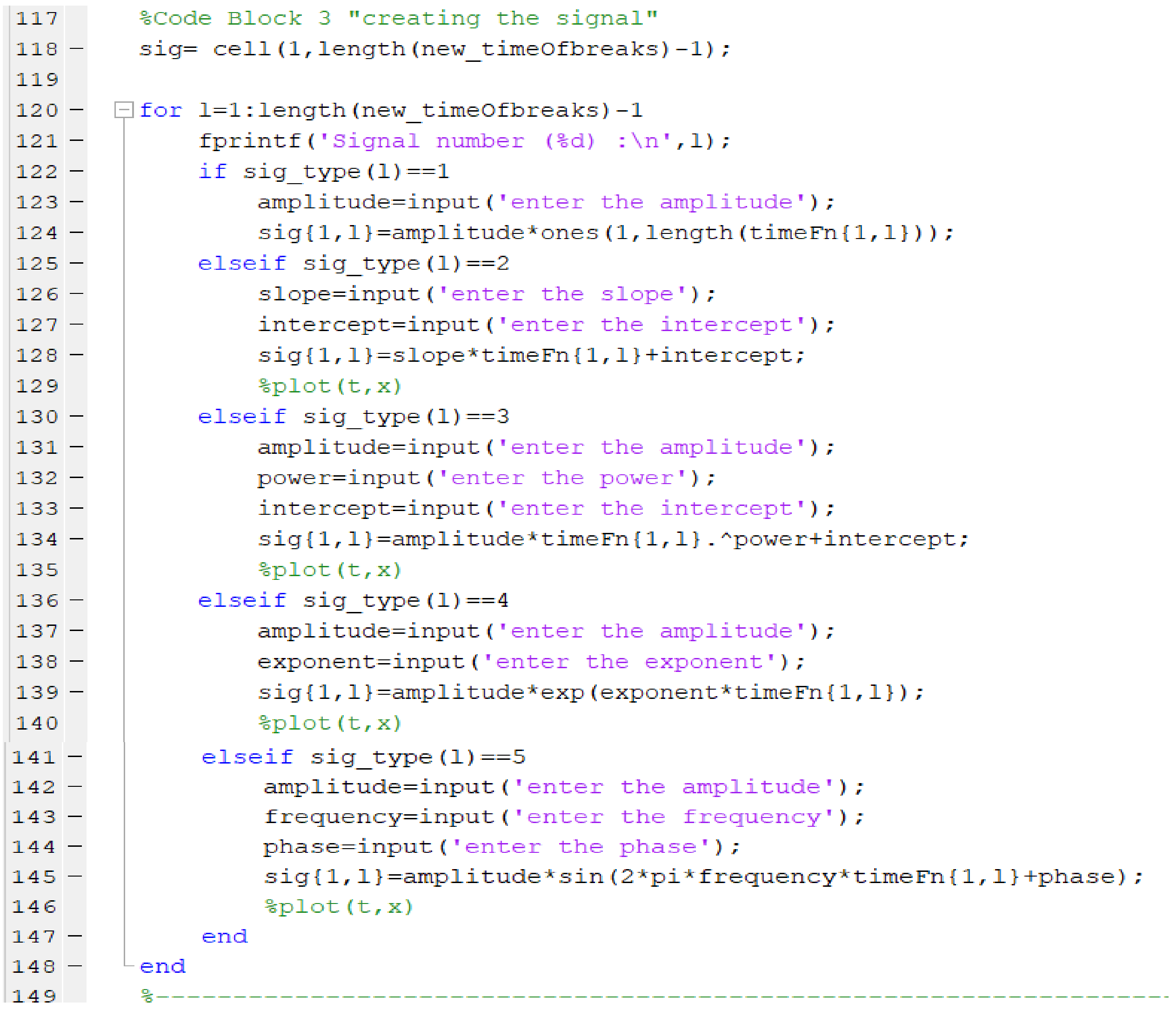
```
%Code Block 2 "setting time axis && taking the signal type"

new_timeOfbreaks=[Start_time timeOfbreaks Endtime];
timeFn= cell(1,length(new_timeOfbreaks)-1);
sig_type=zeros(1,length(new_timeOfbreaks)-1);

for k=1:1:length(new_timeOfbreaks)-1
    t_start = new_timeOfbreaks(k);
    t_end = new_timeOfbreaks(k+1);
    timeFn{1,k}=linspace(t_start,t_end,(t_end-t_start)*sampling_frequency);
end
flag_3=0
while flag_3==0
    for o=1:1:length(new_timeOfbreaks)-1
        fprintf('write the digit of the signal number (%d): \n (1)DC \n (2)Ramp \n (3)Polynomial \n (4)Exponential \n (5)Sinusoidal \n',c
        sig_type(o)=input('');
    end

    for p=1:1:length(new_timeOfbreaks)-1
        if sig_type(p)>=1 & sig_type(p)<=5
            flag_3=1;
        else
            flag_3=0;
            messagebox5=msgbox("choose the correct signal number from (1 to 5) ");
            break
        end
    end
end
%------------------------------------------------------------------------------------------------------------------------%
```

Fig. 4. The code block for setting time axis and the signal type.

The second block of code shown in figure 4 is for setting the time axis by putting start time, end time and all the breakpoints in an array, the **'cell function'** is used due to the existence of different sizes of samples in each region of the linspace, the **'zeroes function'** defines an empty array the will be used later to store the different signals.

Then, the 'for' loop is used to set the time range of the function between each 2 points, using 'linspace' to compute the signal at a set of values of t corresponding to the time interval of interest. In the 'while' loop the program ask the user to define the type of each signal from 1 to 5 different types.

flag_3: flag for checking that the entered signal number is correct and exists in the range between (1 and 5).

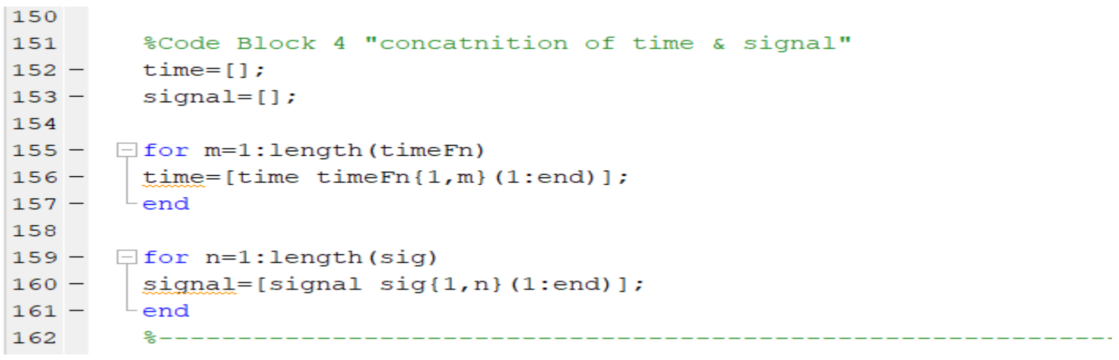


```matlab
%Code Block 3 "creating the signal"
sig= cell(1,length(new_timeOfbreaks)-1);

for l=1:length(new_timeOfbreaks)-1
    fprintf('Signal number (%d) :\n',l);
    if sig_type(l)==1
        amplitude=input('enter the amplitude');
        sig{1,l}=amplitude*ones(1,length(timeFn{1,l}));
    elseif sig_type(l)==2
        slope=input('enter the slope');
        intercept=input('enter the intercept');
        sig{1,l}=slope*timeFn{1,l}+intercept;
        %plot(t,x)
    elseif sig_type(l)==3
        amplitude=input('enter the amplitude');
        power=input('enter the power');
        intercept=input('enter the intercept');
        sig{1,l}=amplitude*timeFn{1,l}.^power+intercept;
        %plot(t,x)
    elseif sig_type(l)==4
        amplitude=input('enter the amplitude');
        exponent=input('enter the exponent');
        sig{1,l}=amplitude*exp(exponent*timeFn{1,l});
        %plot(t,x)
    elseif sig_type(l)==5
        amplitude=input('enter the amplitude');
        frequency=input('enter the frequency');
        phase=input('enter the phase');
        sig{1,l}=amplitude*sin(2*pi*frequency*timeFn{1,l}+phase);
        %plot(t,x)
    end
end
%-----------------------------------------------------------------------
```

Fig. 5. The parameter inputs for each function.

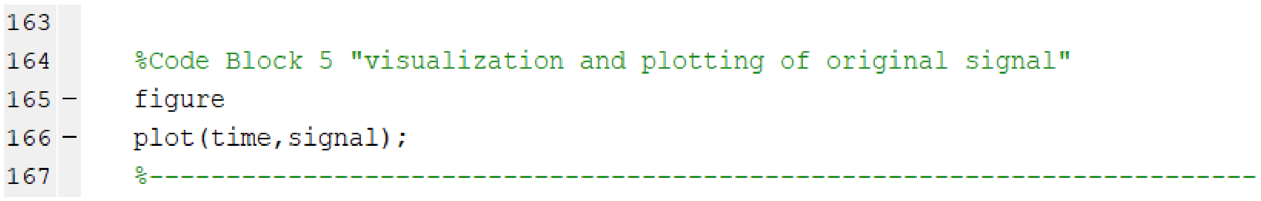
The third block of code in figure 5 is for asking the user for the parameters of each entered function and putting all of these in an array, again, due to different types of signals in the array, the **‘cell function’** is used.

Then concatenate all these functions as shown in figure 6 and its time regions in one array to be used in a visualization part in figure 7.



```matlab

%Code Block 4 "concatnition of time & signal"
time=[];
signal=[];

for m=1:length(timeFn)
time=[time timeFn{1,m}(1:end)];
end

for n=1:length(sig)
signal=[signal sig{1,n}(1:end)];
end
%--------------------------------------------------------------
```

Fig. 6. Concatenating the different types of signals in one array.



```
%Code Block 5 "visualization and plotting of original signal"
figure
plot(time,signal);
%-------------------------------------------------------------------------
```

Fig. 7. Plotting the original signal.

4. The program asks the user if he wants to perform any operation on the signal

a. **Amplitude Scaling**: scale value.

b. **Time reversal**.

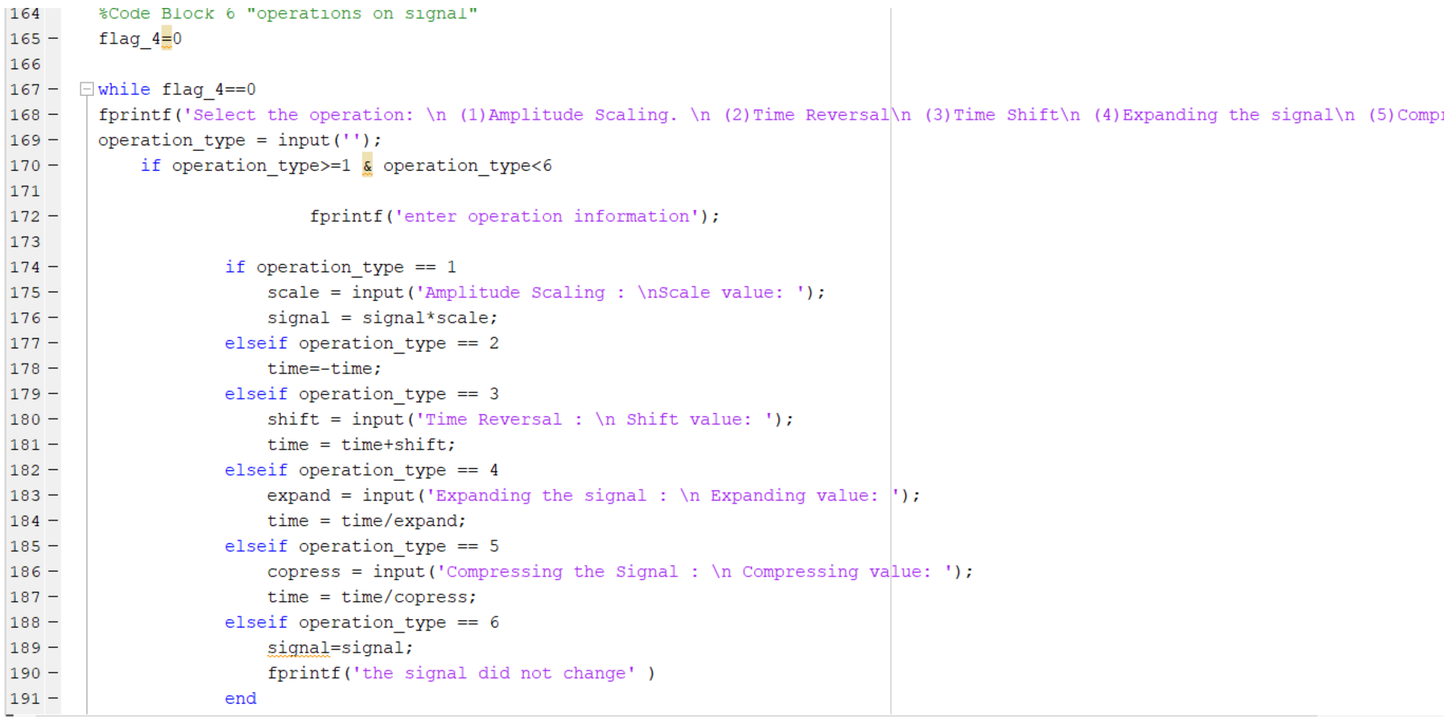
c. **Time shift**: shift value.

d. **Expanding the signal**: expanding value.

e. **Compressing the signal**: compressing value.

f. **None.**

5. Display the new signal in time domain.

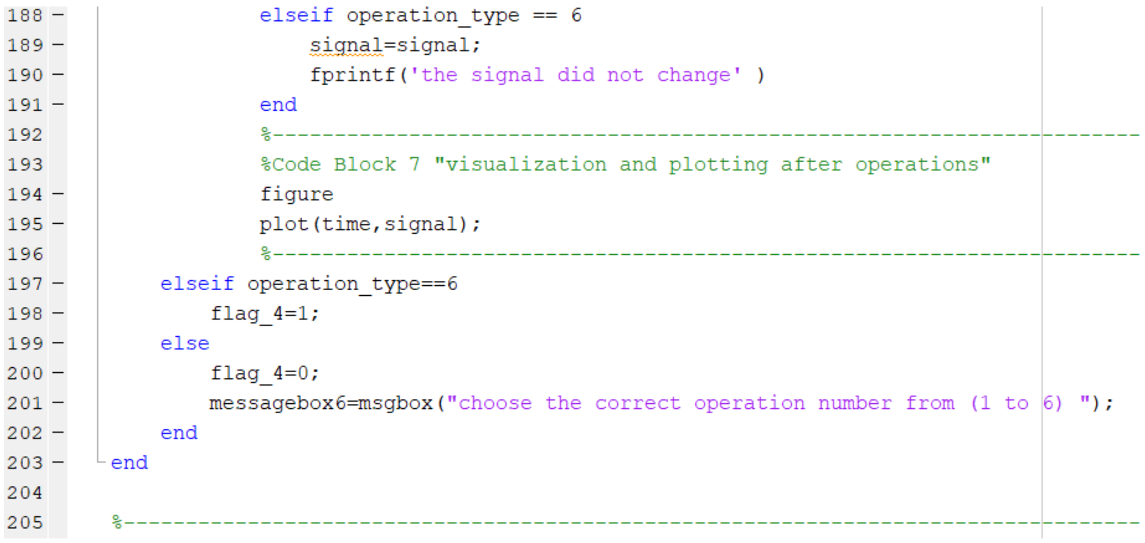


```
%Code Block 6 "operations on signal"
flag_4=0

while flag_4==0
fprintf('Select the operation: \n (1)Amplitude Scaling. \n (2)Time Reversal\n (3)Time Shift\n (4)Expanding the signal\n (5)Compr
operation_type = input('');
    if operation_type>=1 & operation_type<6

                    fprintf('enter operation information');

            if operation_type == 1
                scale = input('Amplitude Scaling : \nScale value: ');
                signal = signal*scale;
            elseif operation_type == 2
                time=-time;
            elseif operation_type == 3
                shift = input('Time Reversal : \n Shift value: ');
                time = time+shift;
            elseif operation_type == 4
                expand = input('Expanding the signal : \n Expanding value: ');
                time = time/expand;
            elseif operation_type == 5
                copress = input('Compressing the Signal : \n Compressing value: ');
                time = time/copress;
            elseif operation_type == 6
                signal=signal;
                fprintf('the signal did not change' )
            end
```

Fig. 8. The code block of the required operations.

The fourth block of code in figure 8 to perform different operations on the main signal, first the program checks the correctness of the user input by **flag_4:** The input must be from 1 to 6. It returns a message box if it invalid.

Then in figure 9 the program plots the modified signal after the chosen operation and asks the user to choose the operation again, until the user enters none, the program stops.



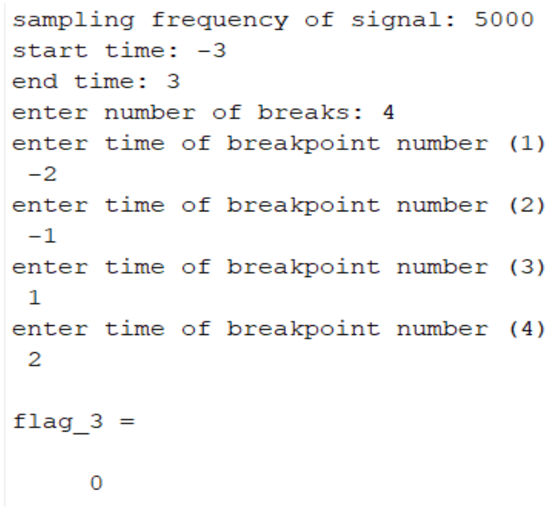
```
                elseif operation_type == 6
                    signal=signal;
                    fprintf('the signal did not change' )
                end
                %------------------------------------------------------------------
                %Code Block 7 "visualization and plotting after operations"
                figure
                plot(time,signal);
                %------------------------------------------------------------------
        elseif operation_type==6
            flag_4=1;
        else
            flag_4=0;
            messagebox6=msgbox("choose the correct operation number from (1 to 6) ");
        end
end

%------------------------------------------------------------------------------
```

Fig. 9. Plotting the signal after the operation.

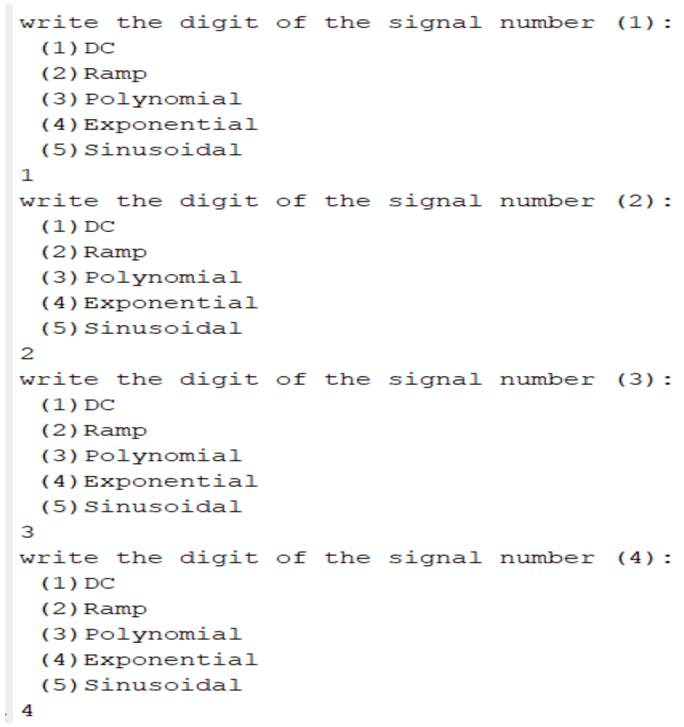
Running the program

It is a simple simulation of the program, the sampling frequency = 5000, the time duration [-3,3], the number of breakpoints =4, and its positions in -2, -1, 1, 2.

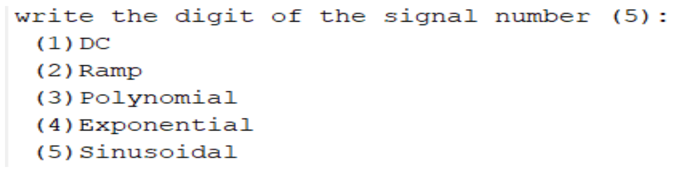
Then the used functions are as in the same appeared sequence DC, Ramp, Polynomial, Exponential and sinusoidal.



```
sampling frequency of signal: 5000
start time: -3
end time: 3
enter number of breaks: 4
enter time of breakpoint number (1)
 -2
enter time of breakpoint number (2)
 -1
enter time of breakpoint number (3)
 1
enter time of breakpoint number (4)
 2

flag_3 =

     0
```

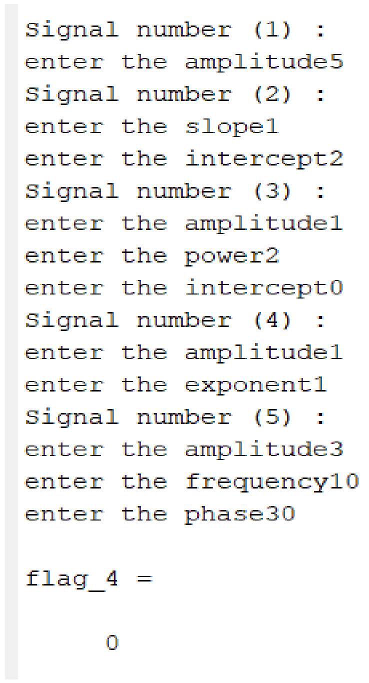


```
write the digit of the signal number (1):
 (1)DC
 (2)Ramp
 (3)Polynomial
 (4)Exponential
 (5)Sinusoidal
1
write the digit of the signal number (2):
 (1)DC
 (2)Ramp
 (3)Polynomial
 (4)Exponential
 (5)Sinusoidal
2
write the digit of the signal number (3):
 (1)DC
 (2)Ramp
 (3)Polynomial
 (4)Exponential
 (5)Sinusoidal
3
write the digit of the signal number (4):
 (1)DC
 (2)Ramp
 (3)Polynomial
 (4)Exponential
 (5)Sinusoidal
4
```



```
write the digit of the signal number (5):
 (1)DC
 (2)Ramp
 (3)Polynomial
 (4)Exponential
 (5)Sinusoidal
```

The entered functions are: 5, $t+2$, t^2, e^t, $3\sin(20\pi+30)$.



```
Signal number (1) :
enter the amplitude5
Signal number (2) :
enter the slope1
enter the intercept2
Signal number (3) :
enter the amplitude1
enter the power2
enter the intercept0
Signal number (4) :
enter the amplitude1
enter the exponent1
Signal number (5) :
enter the amplitude3
enter the frequency10
enter the phase30

flag_4 =

     0
```

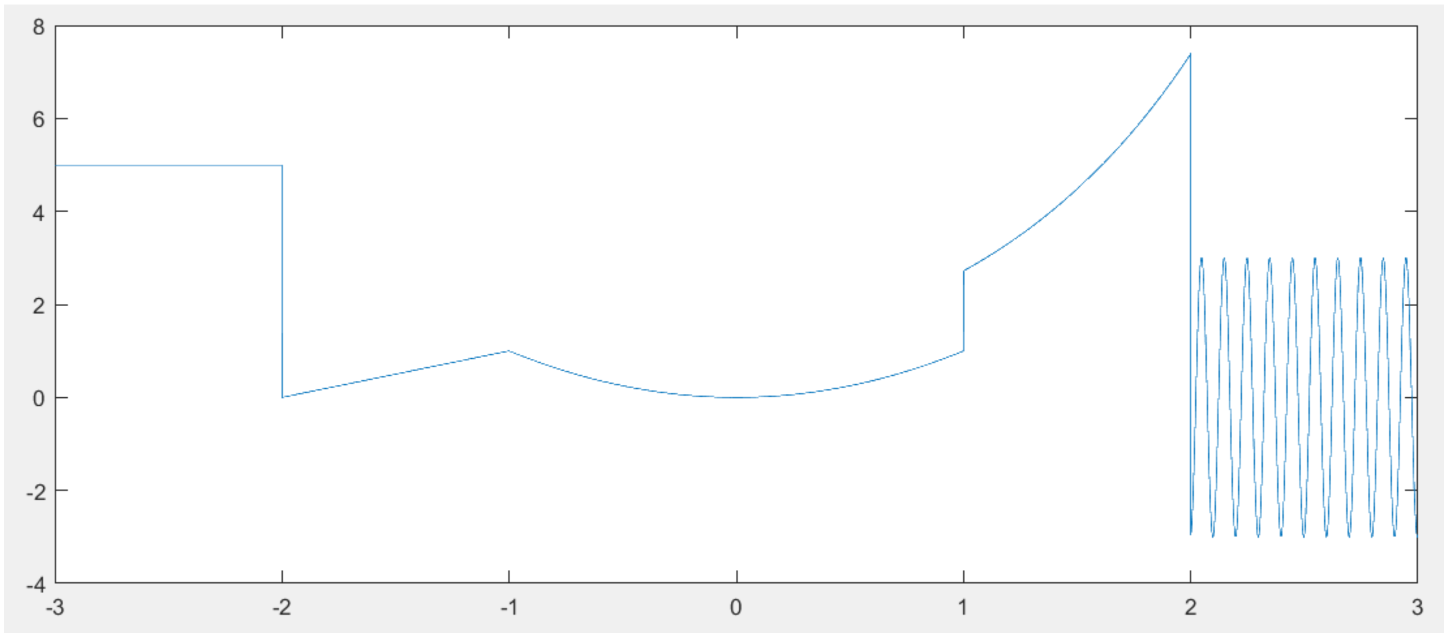


Fig. 10 The original signal.

```
Select the operation:
 (1)Amplitude Scaling.
 (2)Time Reversal
 (3)Time Shift
 (4)Expanding the signal
 (5)Compressing the Signal
 (6)None
 2
```

Perform time reversal on the original signal then repeat asking the user for different operation, and exit the program if the user enters 6.

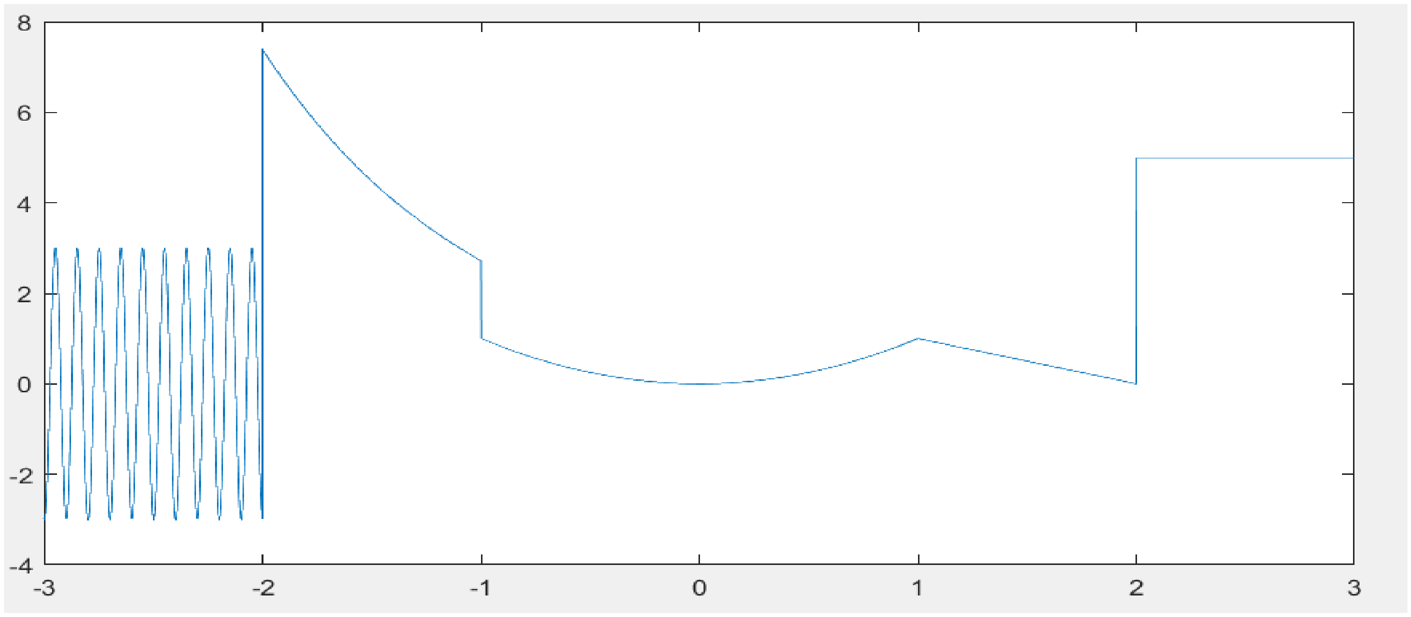
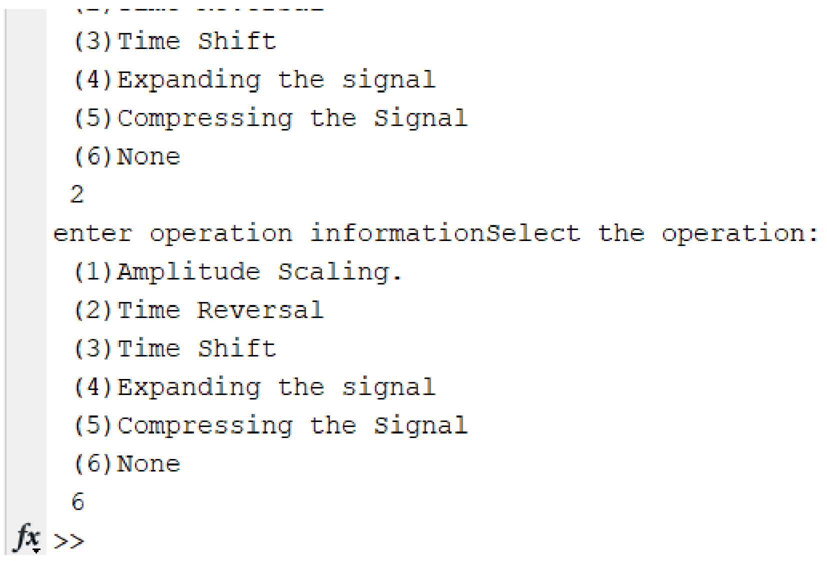


Fig. 11. The signal after the time reversal.



```
 (3)Time Shift
 (4)Expanding the signal
 (5)Compressing the Signal
 (6)None
 2
enter operation informationSelect the operation:
 (1)Amplitude Scaling.
 (2)Time Reversal
 (3)Time Shift
 (4)Expanding the signal
 (5)Compressing the Signal
 (6)None
 6
>>
```

Fig. 12. Exit the program