Name: Kananelo Chabeli

Student Number: CHBKAN001

EEE3092F-Signals & Sytems II

Assignment 1

PLagarism Declaration Statement

I hereby declare that I didn't not recieve help, nor give help in this assignment.

Julia exercise 2.5.1

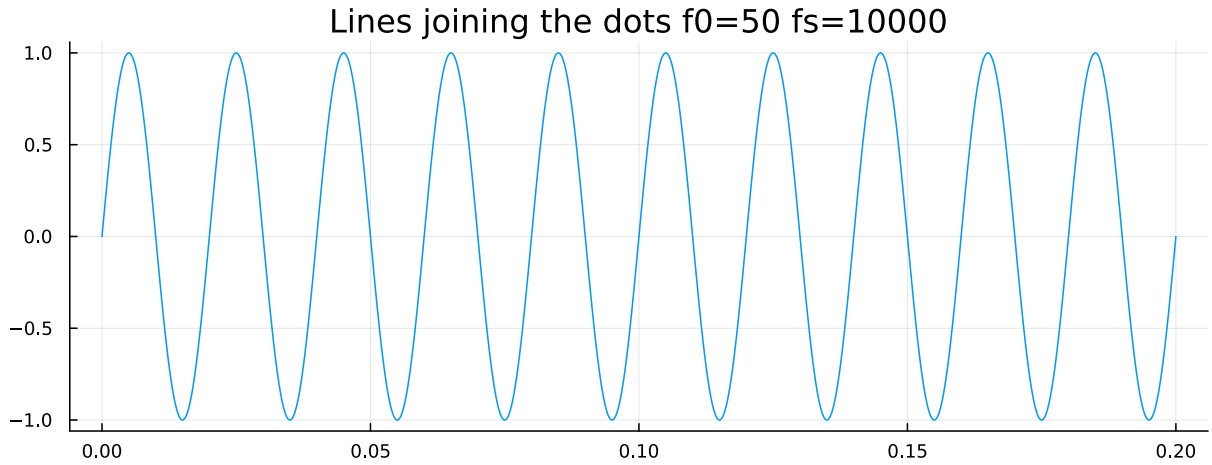
In [1]:

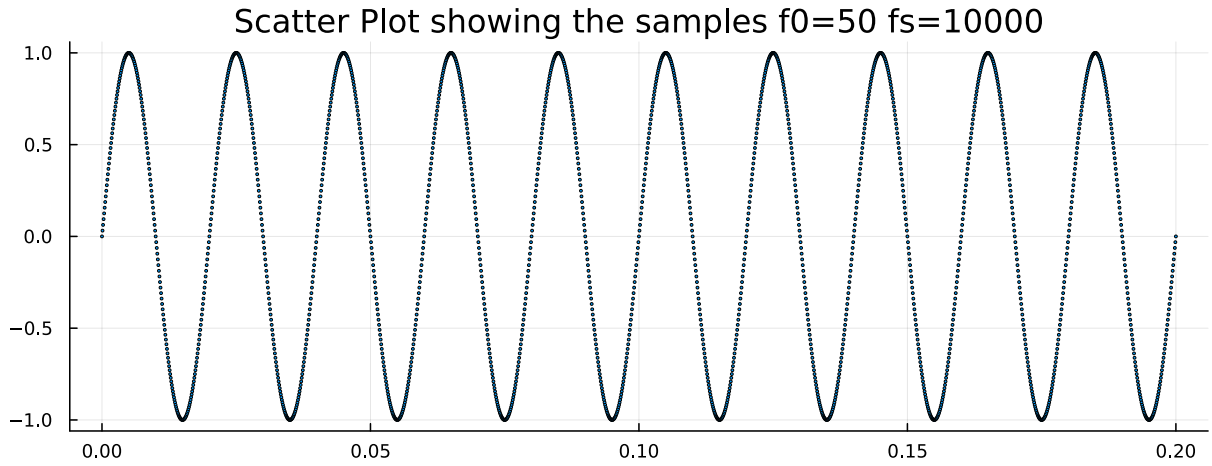
```
using Plots; gr() # Plots with Plotly backend allows mouse zooming
using WAV, FFTW, Statistics
default(size=(800,300)); # Plot canvas size
default(label=""); # Turn off legends
default(ticks=:native); # Ticks on x-axis are labelled properly when zooming in.
function myplot(k)
 f0 =50 # Specify the frequency of the sine wave as 50hz
 T0 = 1/f0 # Calculate the period
 fnyquist = 2*f0 # Calculate Nyquist rate
 fs = k*fnyquist # Define sample rate
 Δt = 1/fs # Calculate time step
 t = 0:Δt:10*T0; # Specify a range of time values
 x = sin.(2*pi*f0*t); # Create array containing function to be plotted
 # Plot statements using Plots + Plotly backend.
 fig = plot(t,x, title = "Lines joining the dots f0=$(f0) fs=$(fs)")
 display(fig)

 fig = scatter(t,x, markersize=1)
 title!("Scatter Plot showing the samples f0=$(f0) fs=$(fs)")
 display(fig)#displaying the plots


    println("Nyquist frequecy-Signal frequency= $(fs-f0)Hz")
end;
```

Plots when smapling frequency equals 100x the Nyquist frequency

In [2]:

```
myplot(100);
```



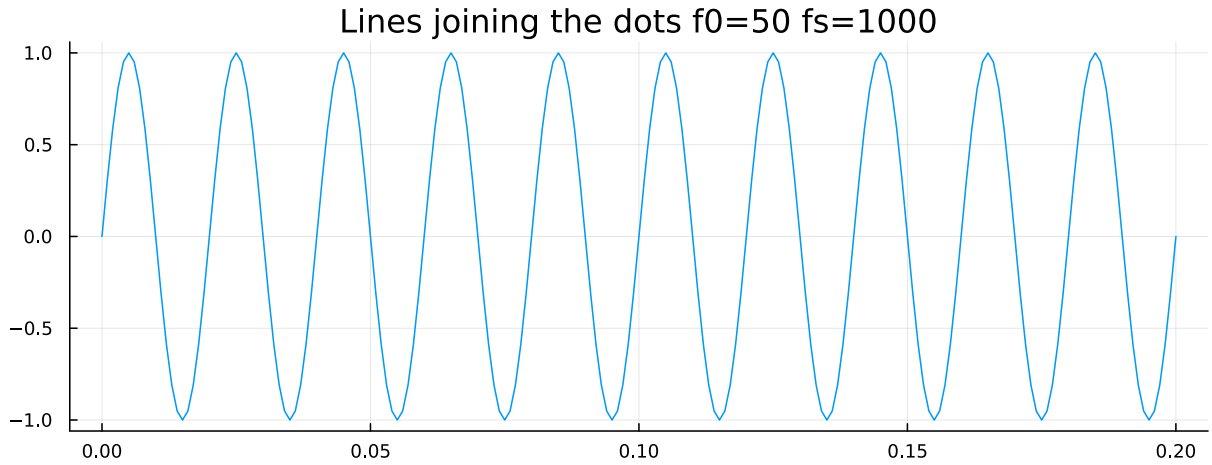


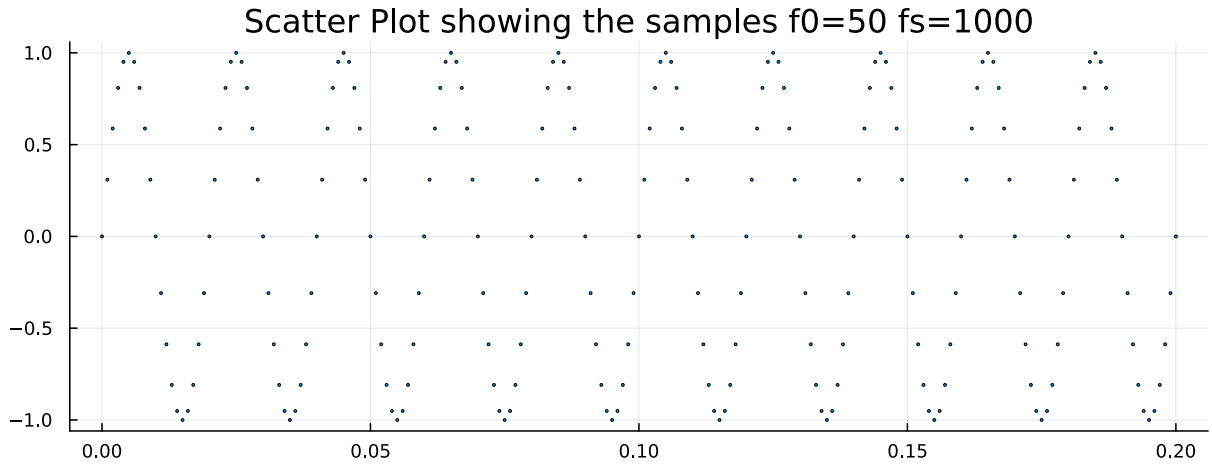

```
Nyquist frequecy-Signal frequency= 9950Hz
```

Plots when smapling frequency equals 10x the Nyquist frequency

In [3]:

```
myplot(10)
```



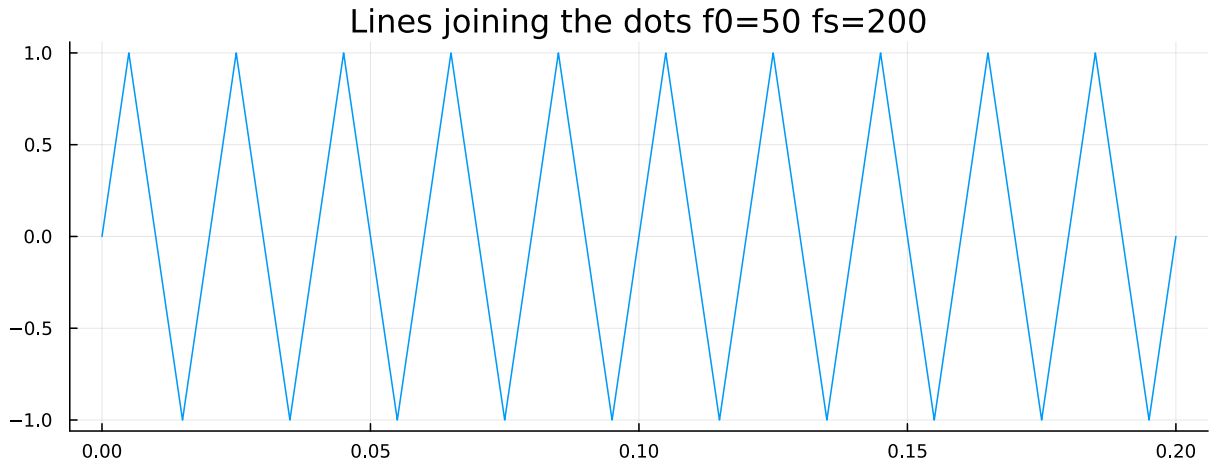


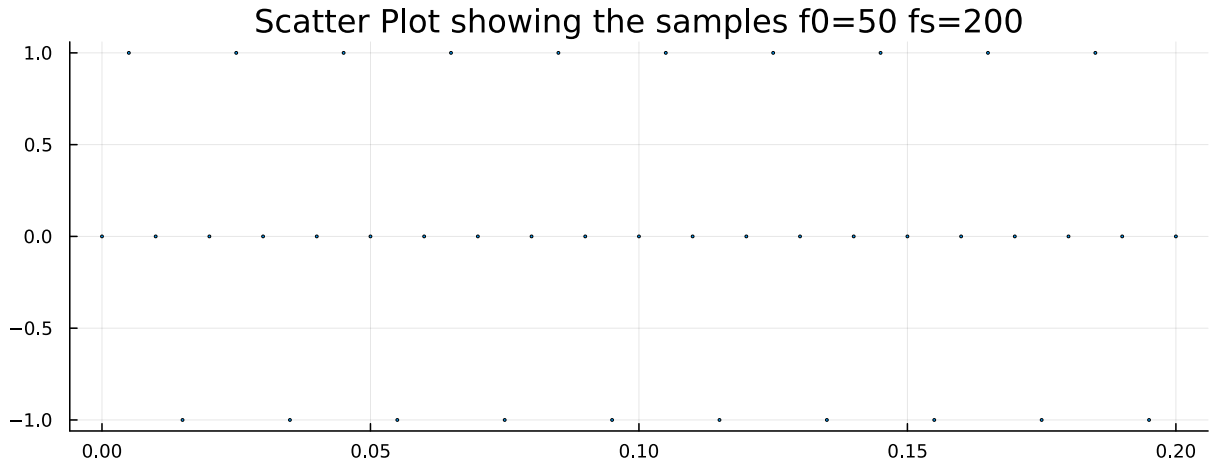

```
Nyquist frequecy-Signal frequency= 950Hz
```

Plots when smapling frequency equals 2x the Nyquist frequency

In [4]:

```
myplot(2)
```



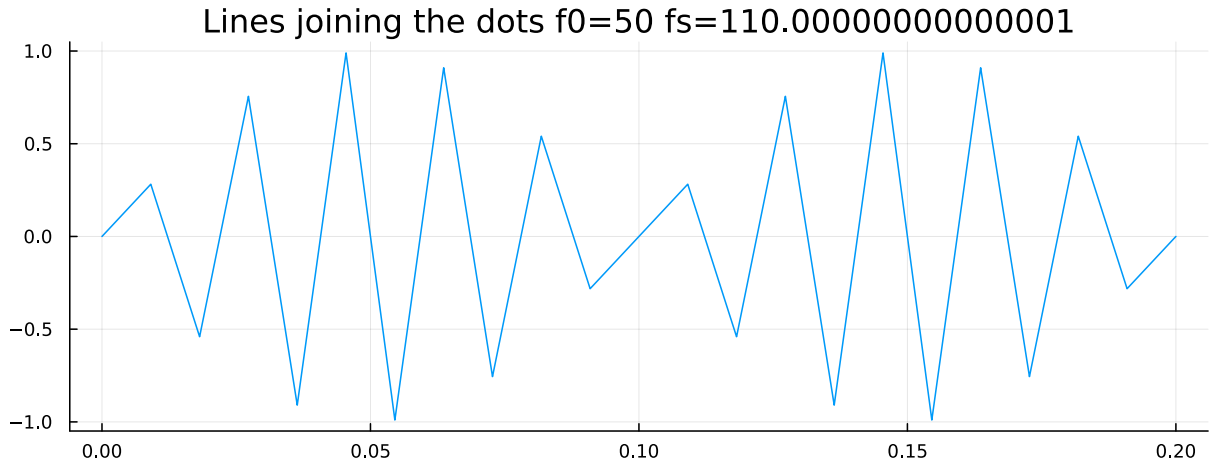


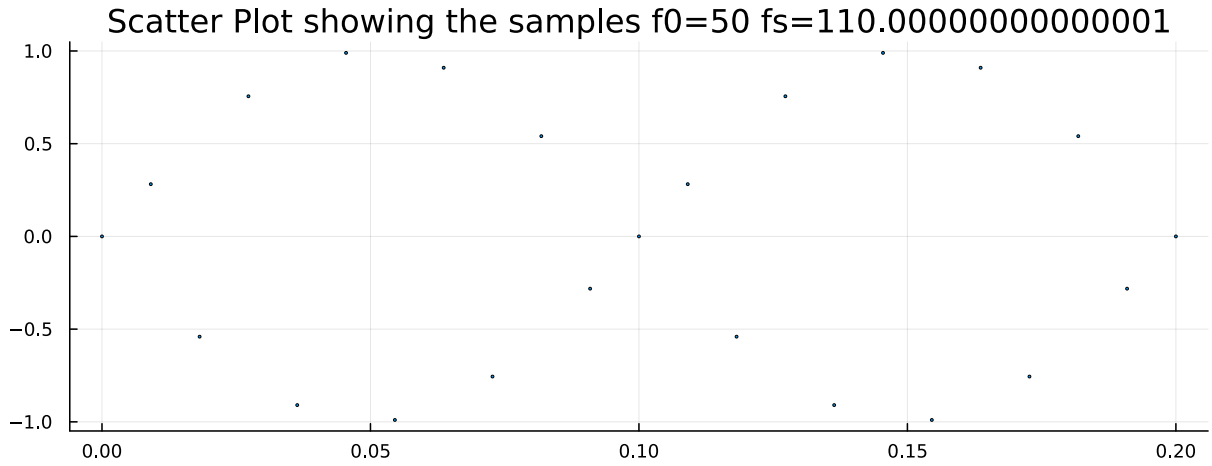

```
Nyquist frequecy-Signal frequency= 150Hz
```

Plots when smapling frequency equals 1.1x the Nyquist frequency

In [5]:

```
myplot(1.1)
```



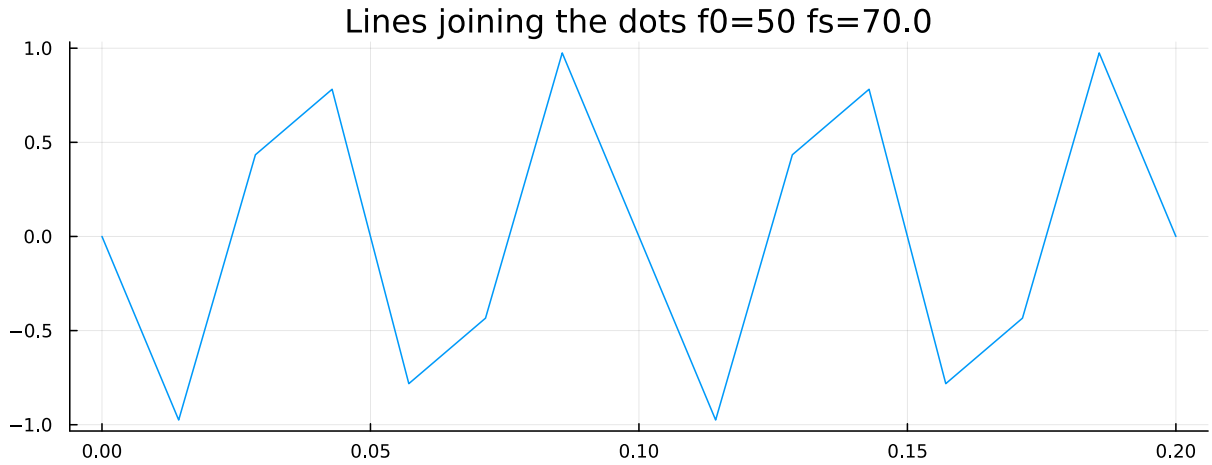


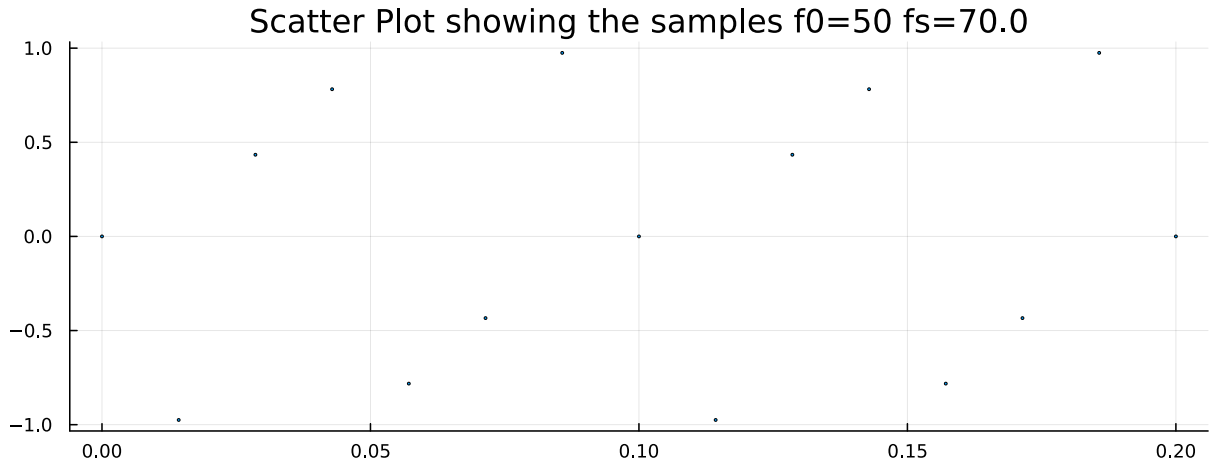

```
Nyquist frequecy-Signal frequency= 60.000000000000014Hz
```

Plots when smapling frequency equals 0.7x the Nyquist frequency

In [6]:

```
myplot(0.7)
```



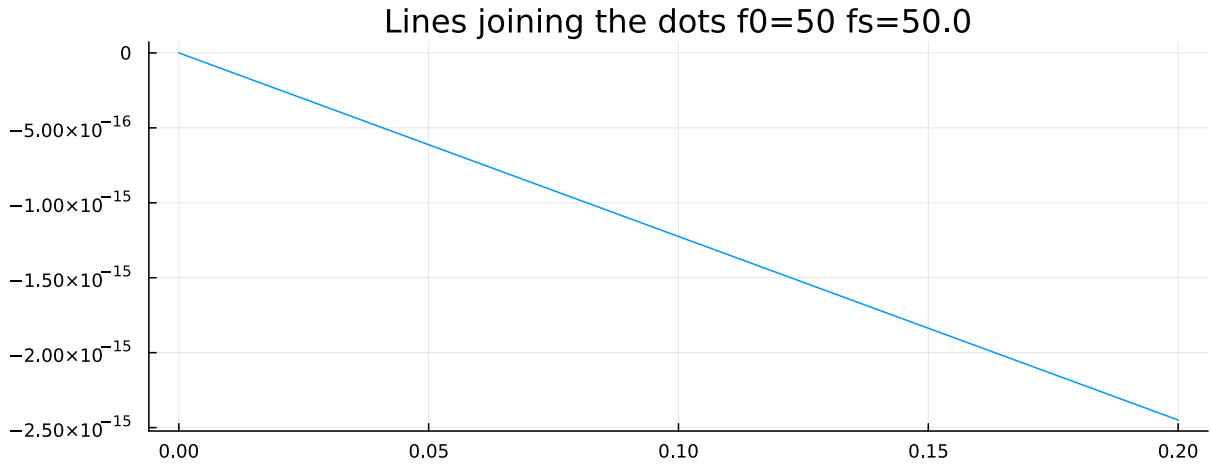


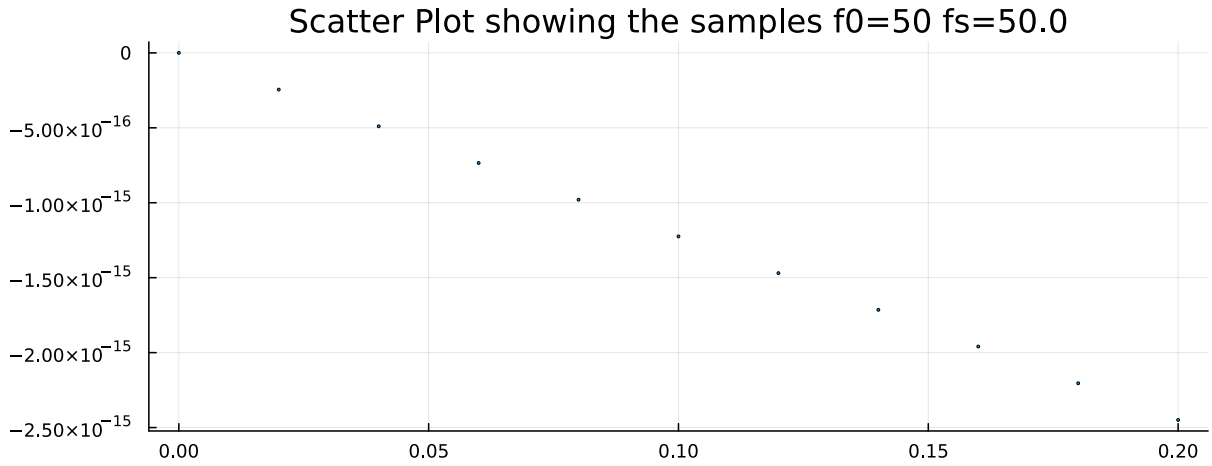

```
Nyquist frequecy-Signal frequency= 20.0Hz
```

Plots when smapling frequency equals 0.5x the Nyquist frequency

In [7]:

```
myplot(0.5)
```






```
Nyquist frequecy-Signal frequency= 0.0Hz
```

###Comments on the plots

The observation made here is that the waveforms of the signal get distorted as the sampling frequecy goes below twice the Nyquist frequency

Julia Exercise 2.5.2 DTF/FFT Introduction

In [54]:

```
#a function that computes DFT Of the sampled signal using the DFT formula

#to be used in comparision with the effecient algorithm
function dft(x)
 N=length(x)
 X = zeros(N)+im*zeros(N) # Complex array of 0+0im; Another method is X=zeros(ComplexF64,N)
 for k=1:N
 for n=1:N
 X[k] = X[k] + x[n]*exp(-im*2*pi*(k-1)*(n-1)/N)
 end
 end
 return X
end;
```

In [ ]:

```

```

q) Comparing fft() to dft() fucntion

In [55]:

```
#using fftw
using FFTW
x = [0,1,1,0,0,0,0,0]
@show fft(x);
println()
@show dft(x);
```

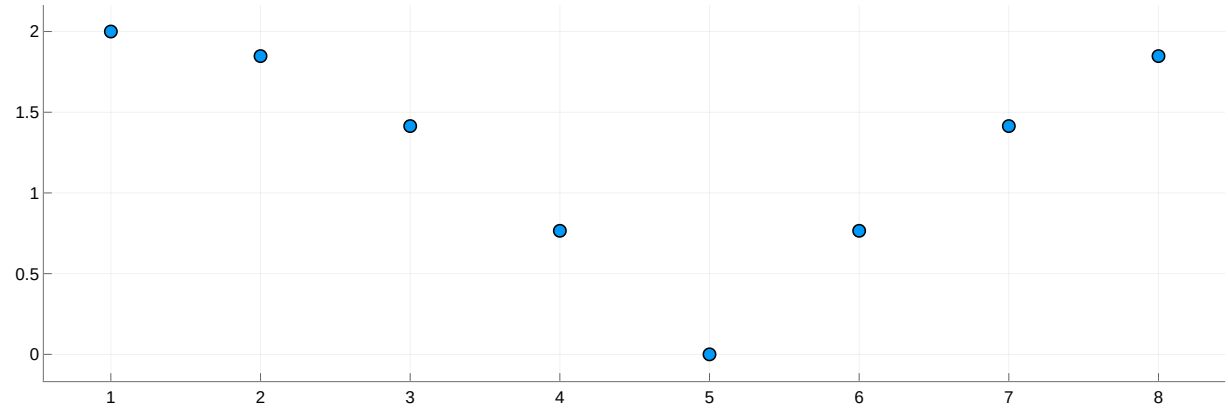
```
fft(x) = ComplexF64[2.0 + 0.0im, 0.7071067811865476 - 1.7071067811865475im, -1.0 - 1.0im, -0.70
71067811865476 + 0.2928932188134524im, 0.0 + 0.0im, -0.7071067811865476 - 0.2928932188134524im,
-1.0 + 1.0im, 0.7071067811865476 + 1.7071067811865475im]

dft(x) = ComplexF64[2.0 + 0.0im, 0.7071067811865477 - 1.7071067811865475im, -0.9999999999999999
- 1.0000000000000002im, -0.7071067811865477 + 0.2928932188134524im, 0.0 + 1.2246467991473532e-1
6im, -0.7071067811865474 - 0.29289321881345254im, -1.0000000000000002 + 0.9999999999999997im,
0.7071067811865469 + 1.7071067811865477im]
```

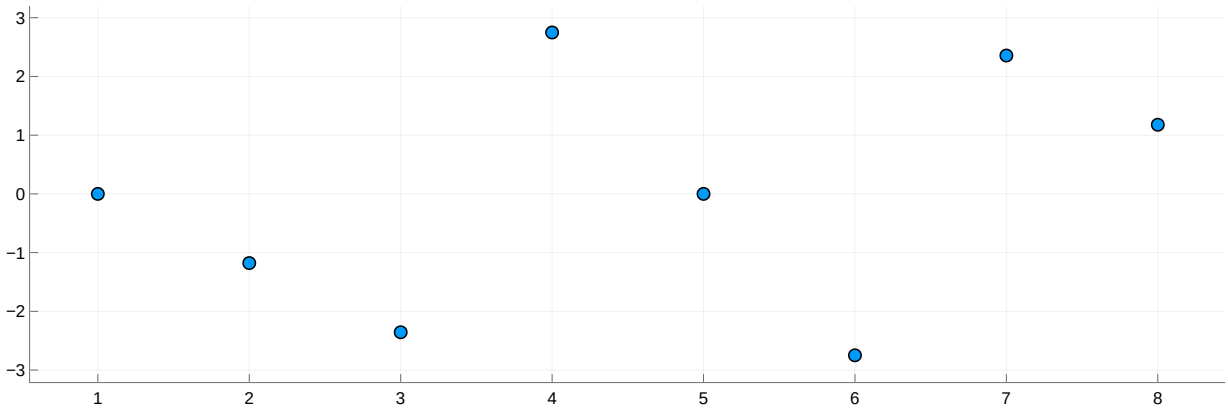
Observation: The two functions produced very similar complex arrays with only minor difference in few decimal places

a)Plotting the Magnitude and Phase of the DFT

In [56]:

```
X = fft(x);#computes DFT

display( scatter(abs.(X)));#displays the magnitude
display(scatter(angle.(X)));#displays the phase
```





c)showing the IFFT

In [57]:

```
@show invX=ifft(X);
```

```
invX = ifft(X) = ComplexF64[0.0 + 0.0im, 1.0 + 0.0im, 1.0 + 0.0im, -3.925231146709438e-17 + 0.0
im, 0.0 + 0.0im, -2.910577510952485e-17 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
```

removing the imaginery parts(which resulted from inperfect numerical computations)

In [58]:

```
@show invX=real(ifft(X));
```

```
invX = real(ifft(X)) = [0.0, 1.0, 1.0, -3.925231146709438e-17, 0.0, -2.910577510952485e-17, 0.
0, 0.0]
```

d) Comparing time complexities of dft(x) and fft() functions

In [59]:

```
#define that will computes the times of each functions for different sizes of N
function test_timing(N)
    x = randn(N);
    t_dft = @elapsed dft(x);
    t_fft = @elapsed fft(x);
    println("dft of length $(N) took $(t_dft) seconds")
    println("fft of length $(N) took $(t_fft) seconds")

end;
```

In [60]:

```
println("For the size N=$(2^8) the time compares and follows:");
test_timing(2^8);

println()
println("For the size N=$(2^10) the time compares and follows:");
test_timing(2^10);
println()
println("For the size N=$(2^11) the time compares and follows:");
test_timing(2^11);
println()
println("For the size N=$(2^12) the time compares and follows:");
test_timing(2^12);
```

```
For the size N=256 the time compares and follows:
dft of length 256 took 0.001970199 seconds
fft of length 256 took 5.4766e-5 seconds

For the size N=1024 the time compares and follows:
dft of length 1024 took 0.034268493 seconds
fft of length 1024 took 7.0125e-5 seconds

For the size N=2048 the time compares and follows:
dft of length 2048 took 0.107692377 seconds
fft of length 2048 took 7.8866e-5 seconds

For the size N=4096 the time compares and follows:
dft of length 4096 took 0.364120762 seconds
fft of length 4096 took 0.000106295 seconds
```

Answer to questions: 1. finding the exact size of input required by an algorithm to take specific time of computation is not easy to find because the big-O notation igores so many important constants in the actaul function describing alorithm's time complexity what one can do is only an approximation. here, we see that and N=1024, the fftw algorithm takes 0.023044739s, now what size will it takes 1s cross multipying the size N appears to be

Exercise 2.5.3 FFT of sine wave

a)plotting signal and analysis in frequency domain

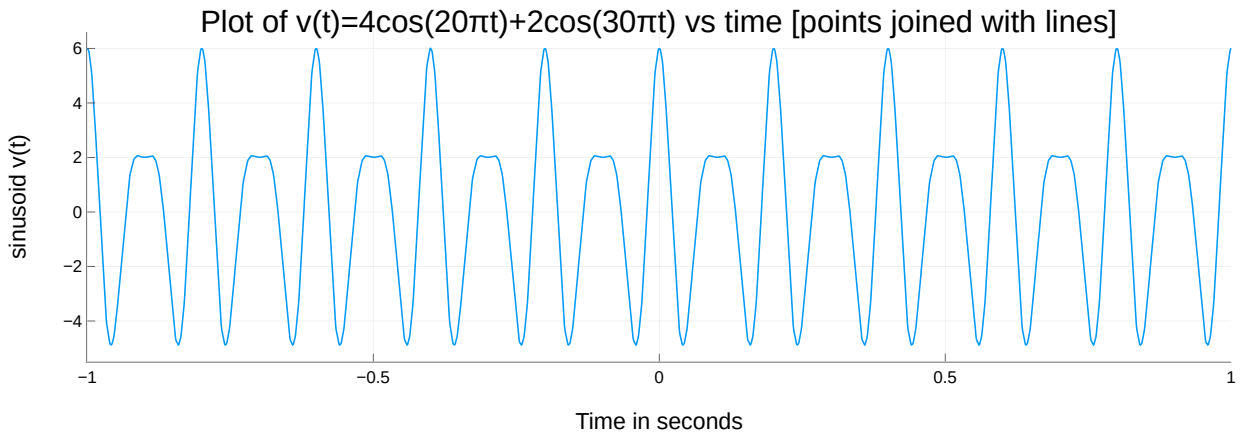
In [61]:

```
Δt = 0.001   # time step
t = -1:Δt:1;  #Define time from 0 to 1s in steps of 0.01s

N = length(t)
println("Number of samples = $(N)")

f0 = 5 # 5 Hz
ω0 = 2*pi*f0;   #

v = 4*cos.(2*ω0*t)+2*cos.(3*ω0*t);   # Create an array holding the sinusoid values

# Show first 10 samples
println("Display first few samples of the array")
@show v[1:10]
;
```

```
Number of samples = 2001
Display first few samples of the array
v[1:10] = [6.0, 5.983230842919242, 5.933033306715286, 5.849736374268631, 5.733885616291015, 5.5
862391135573395, 5.4077617945570235, 5.199618234615434, 4.9631639750182694, 4.699935432655341]
```

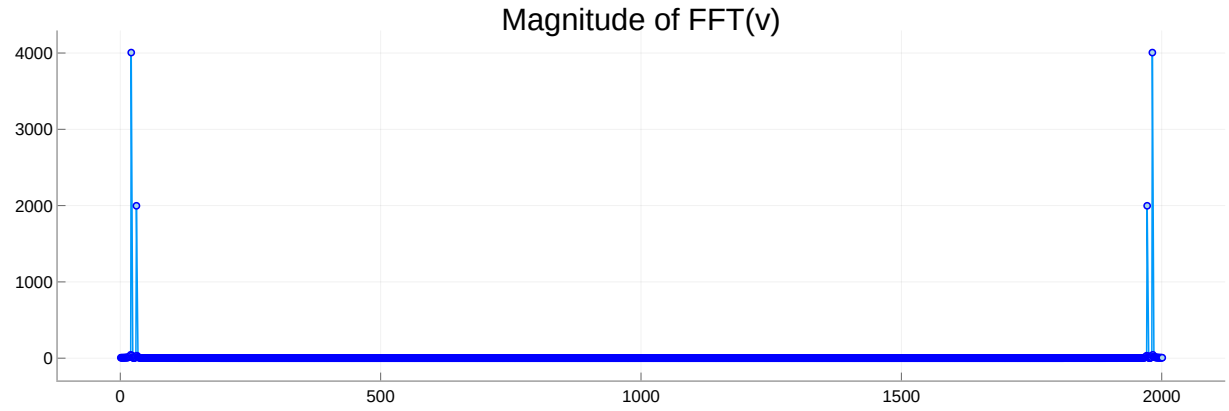
In [62]:

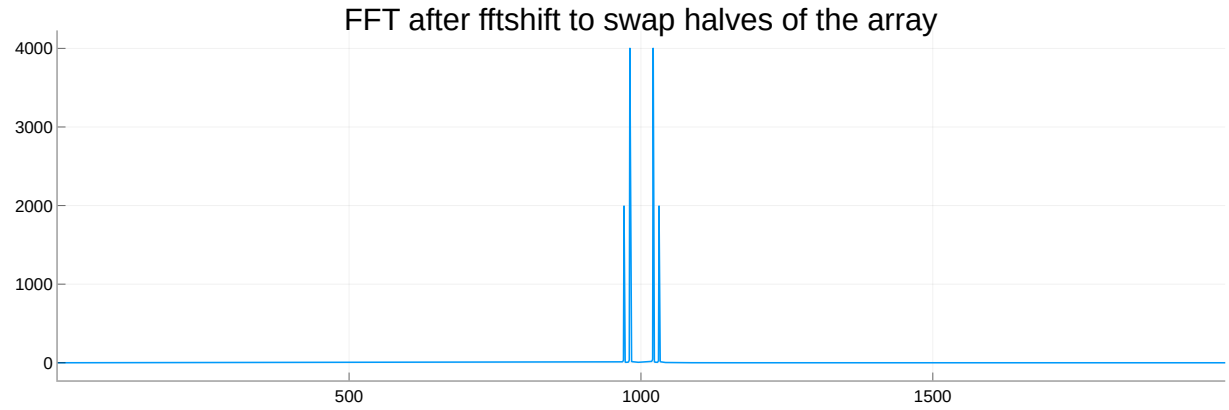
```
#plotting the signal in time domain
fig = plot(t,v)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("Plot of v(t)=4cos(20πt)+2cos(30πt) vs time [points joined with lines]")
display(fig)
```



In [63]:

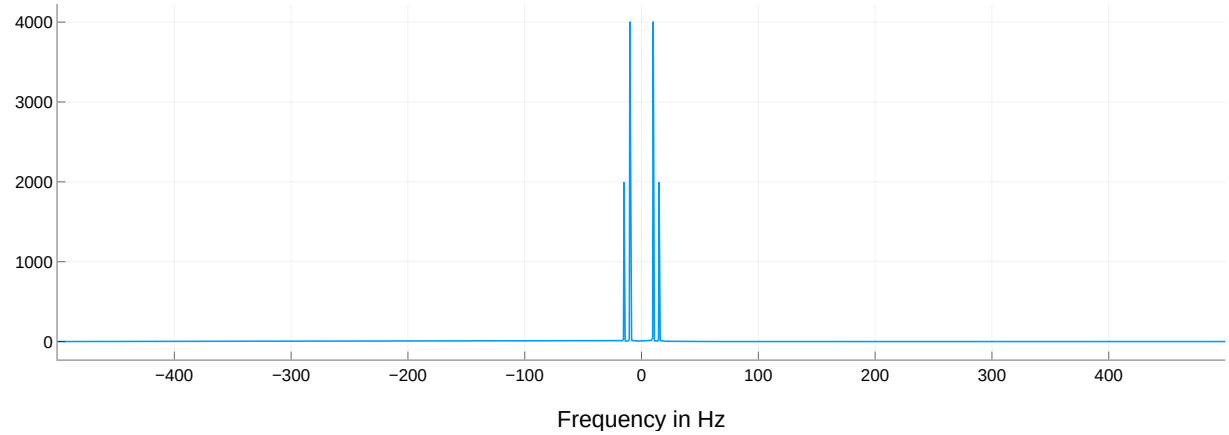
```
# Inspect in frequency domain

using FFTW   # Import Fourier library

V = fft(v)

# This time I will plot both the lines and the points on the same canvas

fig = plot( abs.(V), markershape = :circle, markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)
title!("Magnitude of FFT(v)")
display(fig);

#fig = plot( fftshift(abs.(V)), seriestype = :scatter, markersize=2)   # swap around halves of array so that
#plot!( fftshift(abs.(V))); # Note elts 1:N/2 pos freq components; N/2+1:N contain neg freq components

fig = plot( fftshift(abs.(V)))   # swap around halves of array so that plot show zero Hz in middle
title!("FFT after fftshift to swap halves of the array")
display(fig);
```





In [64]:

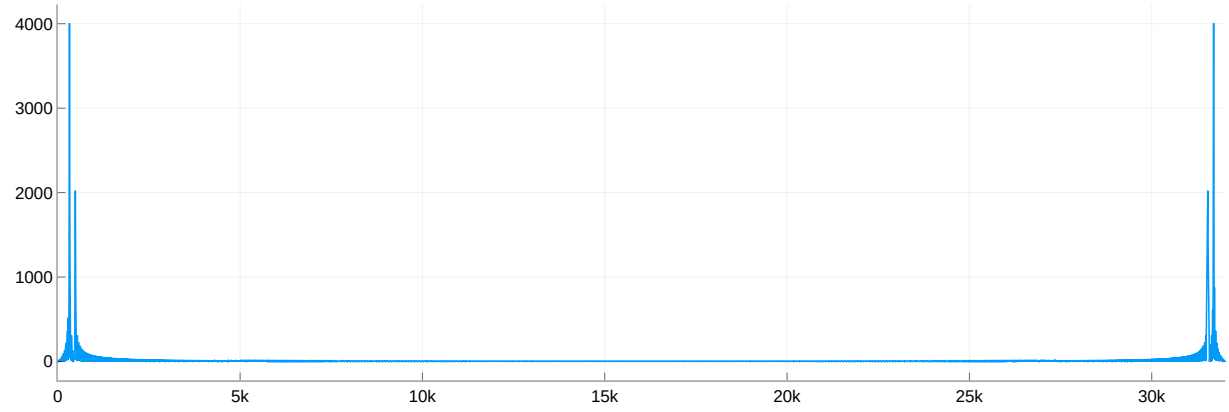
```
#Label the frequency axis in Hz
N = length(t);
Δf = 1/(N*Δt)  # spacing in frequency domain


#create array of freq values stored in f_axis.
if mod(N,2)==0    # case N even
    f_axis = (-N/2:N/2-1)*Δf;
else   # case N odd
    f_axis = (-(N-1)/2 : (N-1)/2)*Δf;
end

fig = plot(f_axis, fftshift(abs.(V)))   # Note elts 1:N/2 pos freq components; N/2+1:N contain neg freq compo
xlabel!("Frequency in Hz");
display(fig);
```



b)Zero padding in time domain

In [65]:

```
y = zeros(16*N) # Make array 16x longer.
y[1:N] = v; # Copy x into first N samples. The rest contains zeros.
Y = fft(y);#compute fft
display( plot(abs.(Y)))
```



In [ ]:

```

```

Answer to question: It is not clearly visible where the sinusoids are placed

Exercise 2.5.4 – Effect of ADC quantization

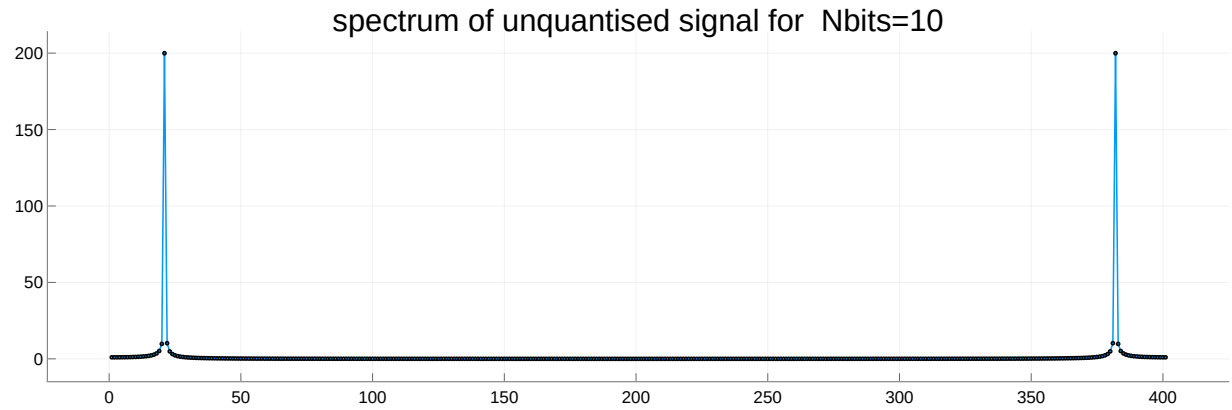
In [66]:

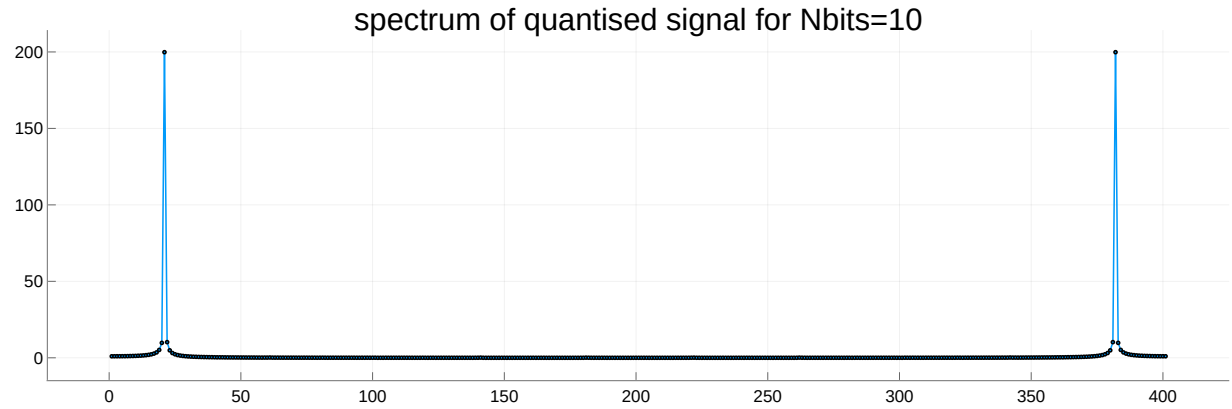
```
##define the fucntion that analyses the quantisation of the  signal given Nbits when is the number of bit
function quantise(Nbits)
    f0=10
    t=-1:0.005:1#fnyquiest is 20, so we want fs=1/delta=10*20=200, when time step must be 0.005
    v=cos.(2*pi*f0*t)
    Nlevels = 2^Nbits
    Amax = 1+0.00001 # Add a small amount to prevent problem at extremes.
    Amin = -1-0.00001
    v_quantized = (round.( (v .- Amin)/(Amax-Amin)*Nlevels .- 0.5) .+0.5) /
    Nlevels*(Amax-Amin) .+ Amin;
    # Display spectrum of the unquantized signal
    display( plot(abs.(fft(v)), markershape = :circle, markersize = 1,title="spectrum of unquantised signal
    # Display spectrum of the quantized signal
    display( plot(abs.(fft(v_quantized)), markershape = :circle, markersize=1,title="spectrum of quantised s
    # Try a dBv scale to see wide dynamic range.
    fig = plot( 20*log10.(abs.(fft(v_quantized))), markershape = :circle,
    markersize = 1);
    title!("Magnitude of FFT(v_quantized) - dBv scale");
    display(fig)
end;
```

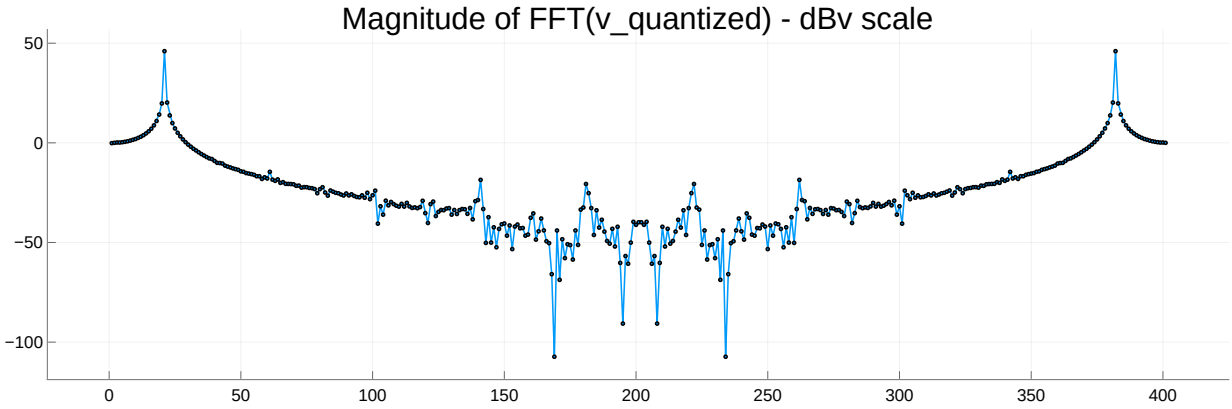
Quantization effect when Nbits=10

In [67]:

```
quantise(10);
```



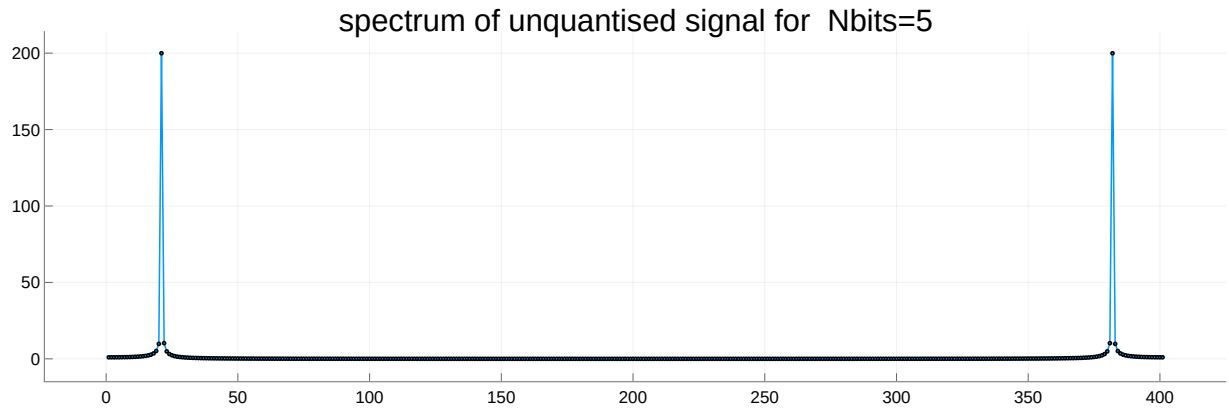


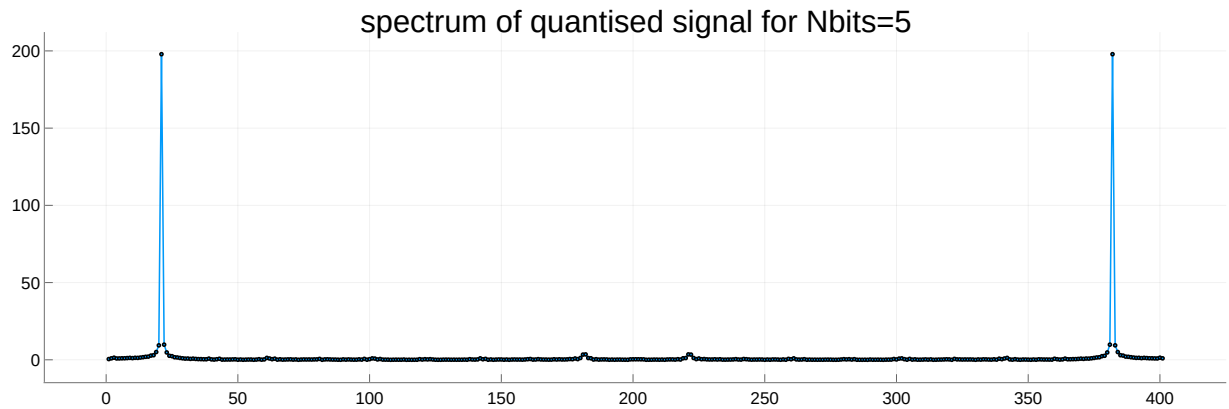


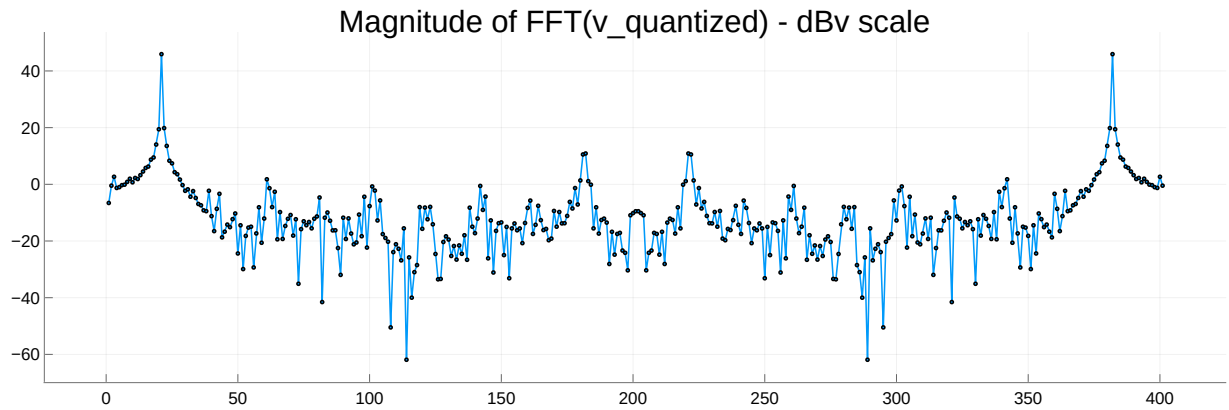
Quantization when Nbits=5

In [68]:

```
quantise(5);
```



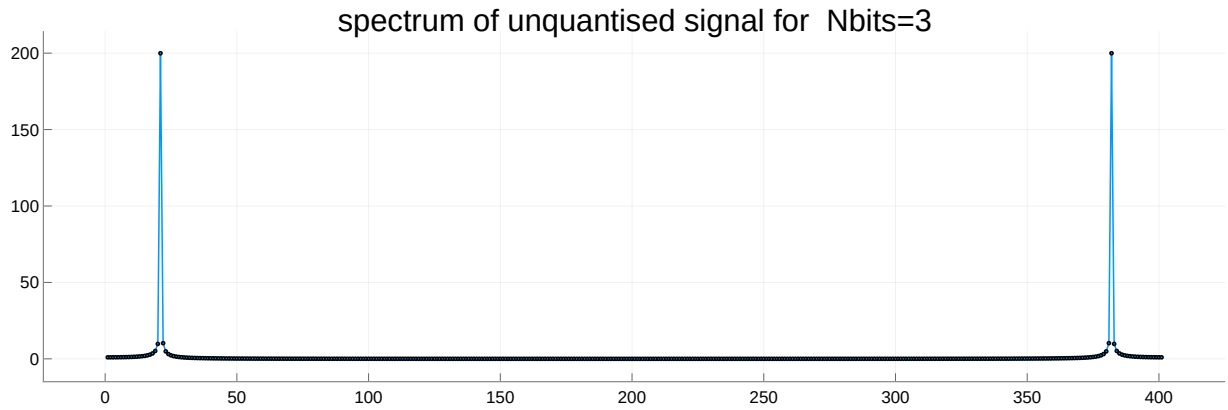


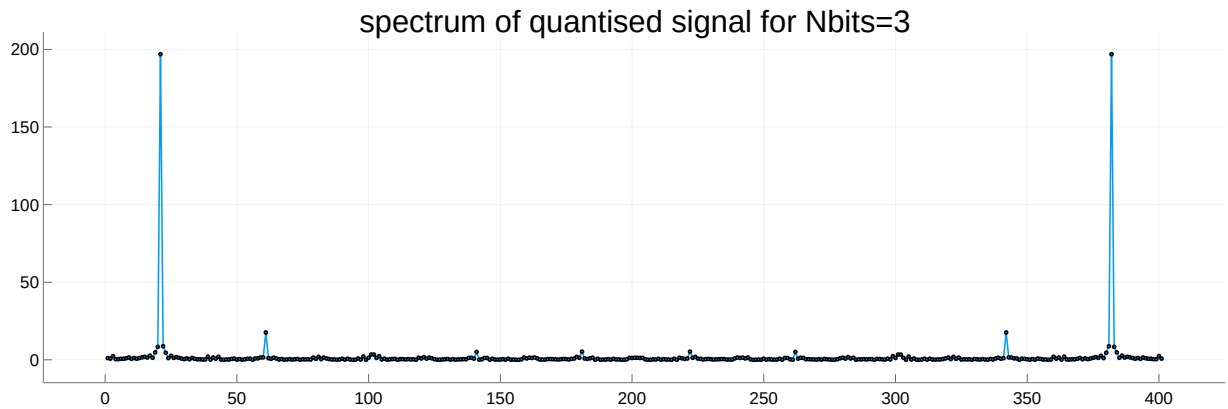


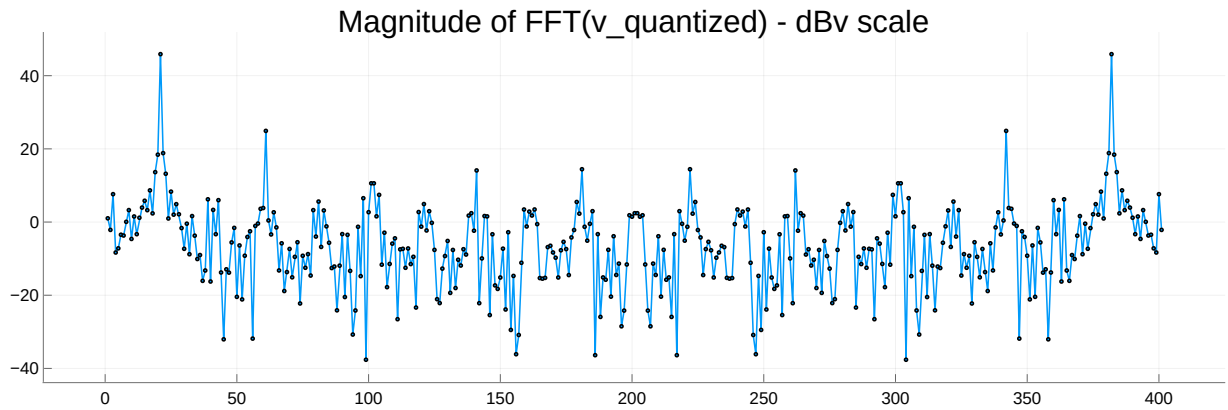
Quantisation when NBits=3

In [69]:

```
quantise(3)
```







Comments on Effect of Quantization

As seen form the plots above, the effect of quantisation on the signal increases with decrese it the number of bits, what we would refer to as resolution of ADC. For higher Resolution, the quantised signal gives good approximation of the of original signal. For small bits(resolution), the frequency domain is highly distorted.

Exercise 2.5.5 Simulating bandlimited noise

In [ ]:

```
```

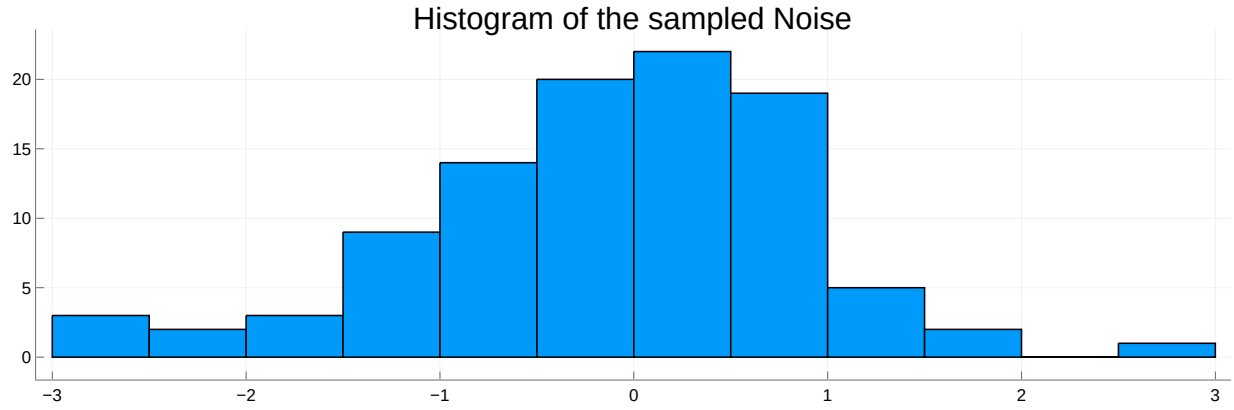
In [ ]:

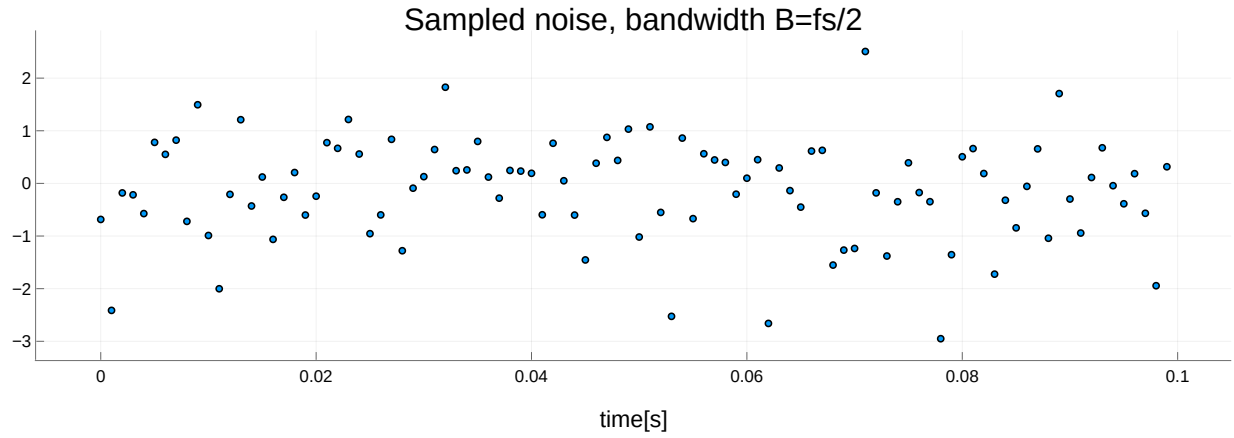
```
```

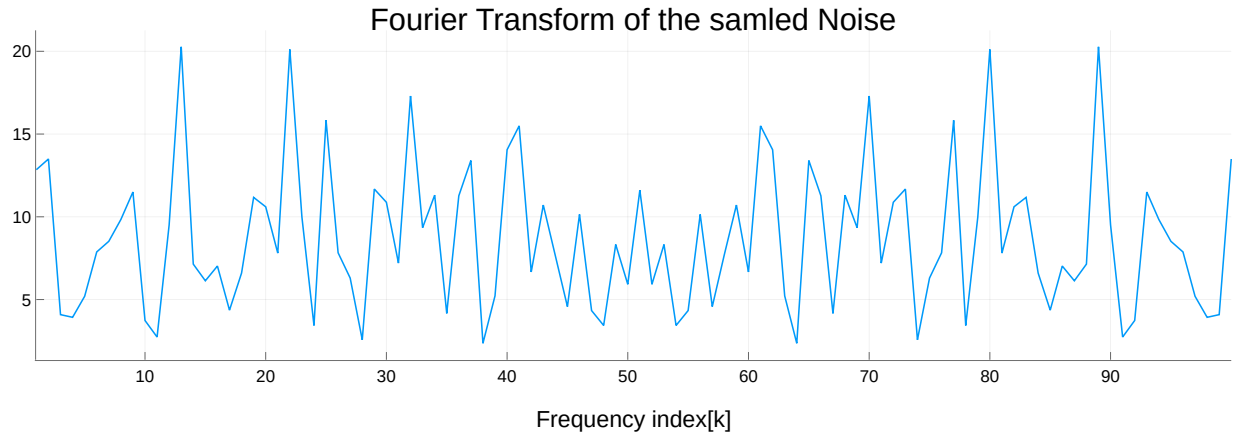
a)

In [70]:

```
fs = 1000 # sample rate in Hz
Δt=1/fs; # sample spacing in s
N = 100 # Choose an even number (makes like easier later)
t = range(0, step=Δt, length=N) # Define time axis
σ = 1
x = σ * randn(N); # Create the random samples with std dev of σ.
display( histogram(x, nbins=10,title="Histogram of the sampled Noise") ) # inspect histogram
# inspect sampled time domain
display( scatter(t,x, markersize=2, title="Sampled noise, bandwidth
B=fs/2",xlabel="time[s]") )
X = fft(x);#compute fft
display( plot(abs.(X),title="Fourier Transform of the samled Noise",xlabel="Frequency index[k]") )
# inspect DFT frequency domain
```

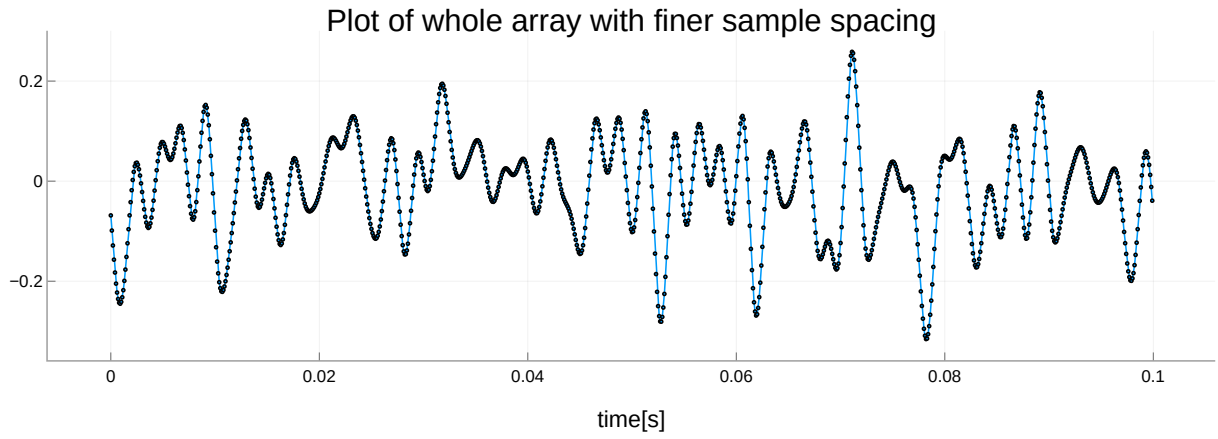


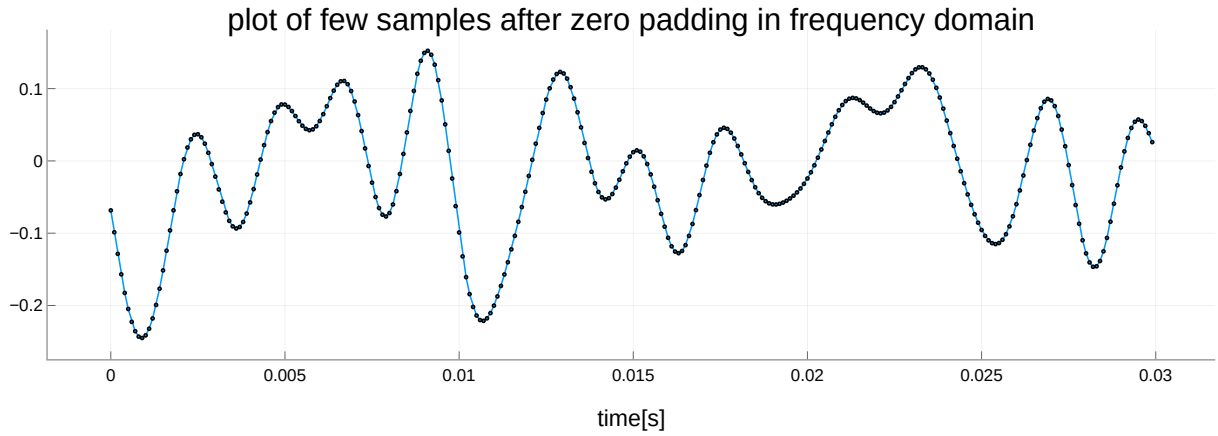




In [71]:

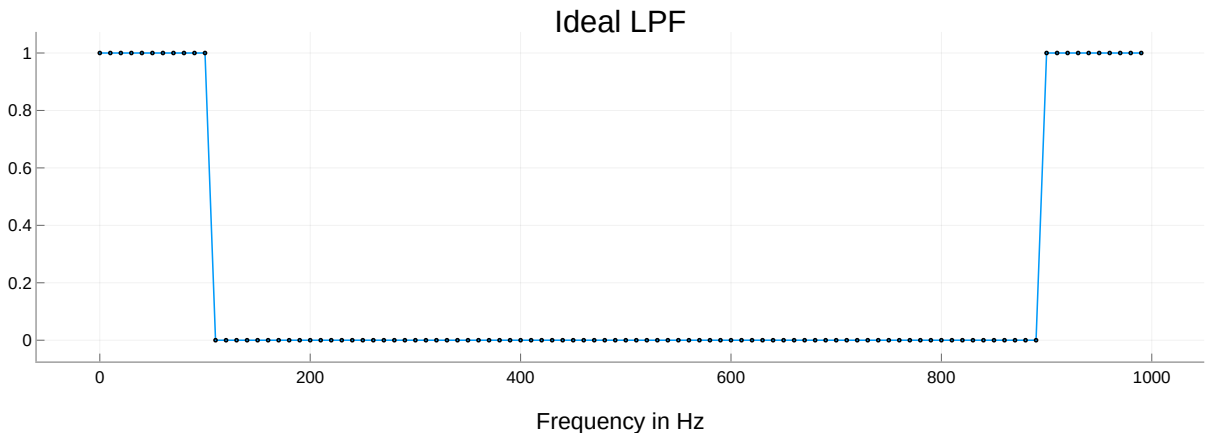
```
pad_factor=10
Ny = pad_factor*N;
Y = zeros(Ny)+im*zeros(Ny) # Create a complex array of zeros
k_mid = Int(N/2)
Y[1:k_mid]=X[1:k_mid]; # Insert the first half of X
Y[Ny-k_mid+1:Ny]=X[k_mid+1:N]; # Insert the 2nd half of X at the end
plot(abs.(Y)); # inspect padded array
y = ifft(Y); # Go back to time domain
y = real(y); # discard the very tiny imaginary components
Ny = length(y)
t_new = range(0, step=Δt/pad_factor, length=Ny) # Define new t-axis
# Plot whole array
display( plot(t_new,y, markershape = :circle, markersize=1,xlabel="time[s]",title="Plot of whole array with
# Plot just first 300 samples
display( plot(t_new[1:300],y[1:300], markershape = :circle,
markersize=1,title="plot of few samples after zero padding in frequency domain",xlabel="time[s]"));
```





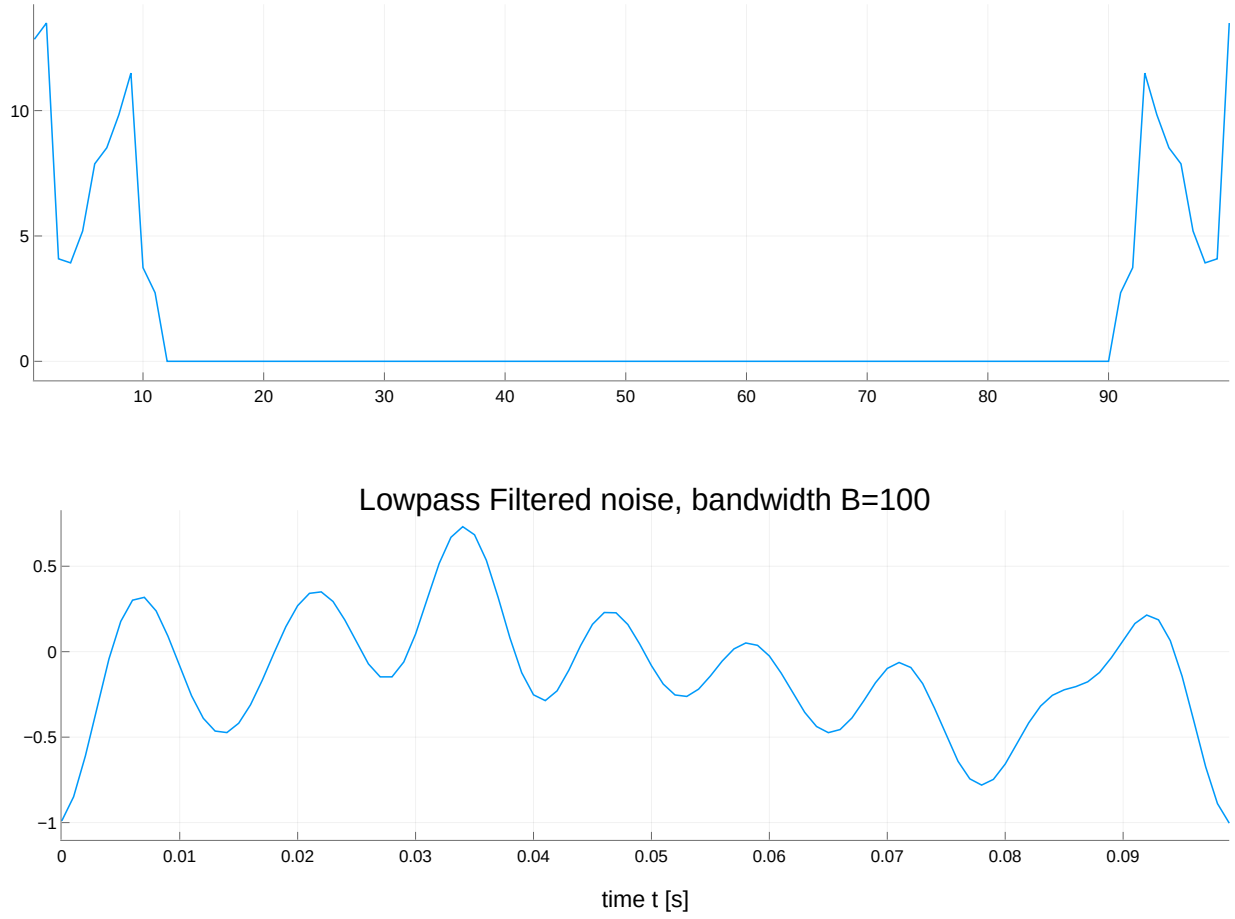
In [77]:

```
# Create a Filter transfer function
Δω = 2*pi/(N*Δt) # Sample spacing in freq domain in rad/s
ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 100 # filter bandwidth in Hz
# In the sampled frequency domain. add a rect centred on zero to one centred
#at the next repeat
# i.e. centred on 0 rad/s an on 2pi/Δt rad/s.
rect(t) = (abs.(t).<=0.5)*1.0;
H = rect(ω/(4*π*B)) + rect( (ω .- 2*π/Δt)/(4*π*B) );
# Note, H in this case is purely real.
fig = plot(f,H, markershape = :circle, markersize=1);
title!("Ideal LPF")
xlabel!("Frequency in Hz")
display(fig)
```



In [74]:

```
X_filtered = X.*H;
display( plot(abs.(X_filtered)) )
x_filtered = ifft(X_filtered)
x_filtered = real(x_filtered)
fig = plot(t,x_filtered)
title!("Lowpass Filtered noise, bandwidth B=$(B)")
xlabel!("time t [s]")
display(fig)
```



In [75]:

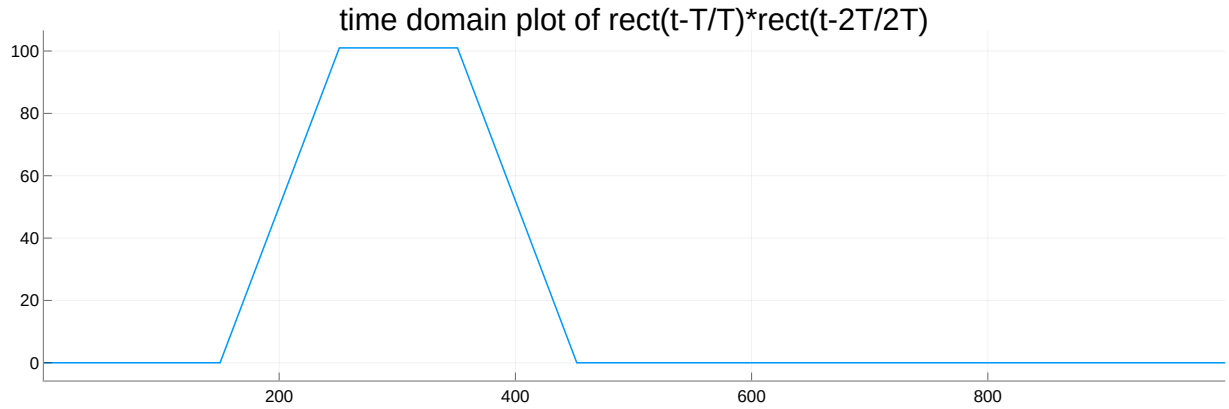
```
println("Standard deviation of the sampled noise: ",std(x));
println("Standard deviation of the sampled Noise after filtering: ",std(x_filtered));
```

```
Standard deviation of the sampled noise: 0.9703111482493276
Standard deviation of the sampled Noise after filtering: 0.3556648703428909
```

Exercise 2.5.6 Discrete Convolution

In [79]:

```
T=1 #specifyig the width of the rectangular pulse
dt=0.01 #specify stem
y = rect( (t.-2T)/(2T) )#rectangular pulse of width 2T
x = rect( (t.-T)/T ) #rectangular plse of width T
t=0:dt:5#time array
z = ifft(fft(x).*fft(y)); #Disrete convolution
fig=plot(real(z)) #plotting the resulting function
title!("time domain plot of rect(t-T/T)*rect(t-2T/2T)");
display(fig)
```



Exercise 2.5.7 Spectral Analysis of Music

Define Functions That read wav file, analyse plot in time and frequency domains*

In [80]:

```
#defining a fucntion that read a wave file and plays the audio,
#depending on the value passed as the second argument and first argument is ethe filename
#the function returns a tuple (data,fs) where 'data' is the 2D array of the wavefile read and 'fs' in the sa

function loadfile(wavefile,play)
    data, fs = wavread(wavefile) # Read stereo wave file
    # wavread() returns 2D data array (matrix) and also the sample rate fs.
    # If stereo, the 1st column holds one channel and the 2nd column holds the other.
    @show Nsamples, Nchannels = size(data); # show size of the array; if stereo, it will be a 2D array.
    fs = Float64(fs) # Convert fs from Float32 to Float64 (standard precision inJulia)
    println("Sample rate fs = ",fs)
    if play==1
         wavplay(data, fs) # play the sound(turn up your volume!).
    end;
    return data, fs;
end;
```

In [81]:

```
###the function takes data and sampling frequency and plots the time domian wave forms of the music
##the array and fs passed to its argument will be values returned by the loadfile function above
## the 3rd and 4th arguments specify the time interval on which the music is to be analysised
# it has been define in this way for easy extraction of singe notes and stuff
#the function returns the extracted portion and sampling period (time spacing)
function timeDomain(data,fs,t1,t2,title)
    channel1 = data[:,1] # extract 1st column (left channel I think)
    # channel2 = data[:,2] # if stereo, then there is a 2nd channel in the 2nd column.
    Nsamples, Nchannels = size(data)
    dt = 1/fs; # calculate sample spacing from sample rate
    #t_axis_all = (0:length(channel1)-1)*dt; # time axis for plotting
  #  fig = plot(t_axis_all[1:10:Nsamples], channel1[1:10:Nsamples]);
  #  title!("Channel1 - every 10th sample", xlabel="Time [seconds]");
  # display(fig);
    i1 = Int(round(t1/dt))+1 # calculate index of sample closest to t1
    i2 = Int(round(t2/dt))+1 # calculate index of sample closest to t2
    y = channel1[i1:i2] # extract relevant portion from channel1
    @show N = length(y) # Get size of array
    wavplay(y, fs) # play the sound; to abort press ctrl-C a few times.
    t=(0:N-1)*dt; # Define a time axis for the extracted portion
    #display(plot(y, xlabel="Sample number",title="Array vs index number for channel1 for first 10seconds"))
    display(plot(t,y, xlabel="Time [seconds]",title=title)) # Plot array vs time
    return y,dt;
end;
```

In [82]:

```
#define a function that takes extracted portion(1D array only one channel will be analysed in frequency
#to  minimise amouts of plots--Julia is killing my computer!) of the music

function freqDomain(y,dt,title) #takes array y and fs and time spacing dt and produce a frequncy domain plot

    N = length(y);#length of time/ array
    Δf = 1/(N*dt)  # spacing in frequency domain
    Y=fft(y)
    #create array of freq values stored in f_axis.
    if mod(N,2)==0    # case N even
        f_axis = (-N/2:N/2-1)*Δf;
    else   # case N odd
        f_axis = (-(N-1)/2 : (N-1)/2)*Δf;
    end
    #computes fftshift and displays in in frequency domain
    plotly();
    fig = plot(f_axis, fftshift(abs.(Y)))
    # Note elts 1:N/2 pos freq components; N/2+1:N contain neg freq components
    xlabel!("Frequency in Hz");
    title!(title)
    display(fig);
    gr()
end;
```

A) Spectral Analysis of Piano Notes

In [83]:

```
data,fs=loadfile("Piano_Scale_20s.wav",0);#read the  piano audio file and play it
```

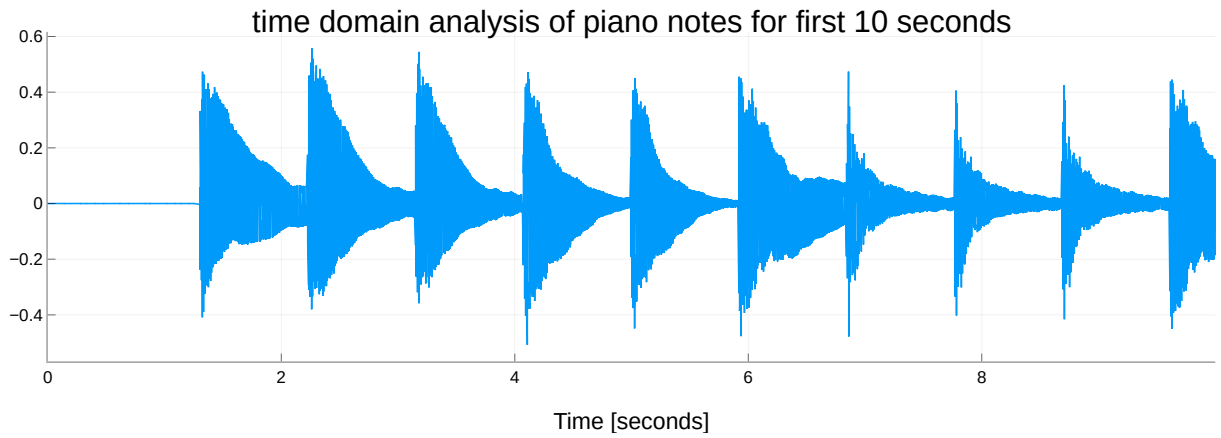
```
(Nsamples, Nchannels) = size(data) = (971495, 2)
Sample rate fs = 48000.0
```

A.1 Time domain analysis of piano notes for first 10 seconds

In [84]:

```
y,dt=timeDomain(data,fs,0,10,"time domain analysis of piano notes for first 10 seconds");
```

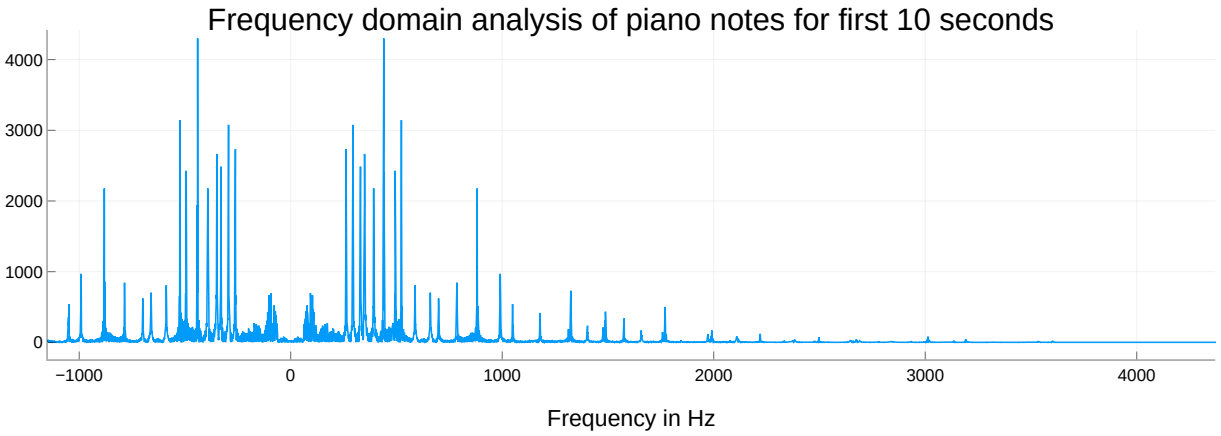
```
N = length(y) = 480001
```



A.2 Frequency domain analysis of piano notes for the first 10 seconds

In [86]:

```
freqDomain(y,dt,"Frequency domain analysis of piano notes for first 10 seconds");
```

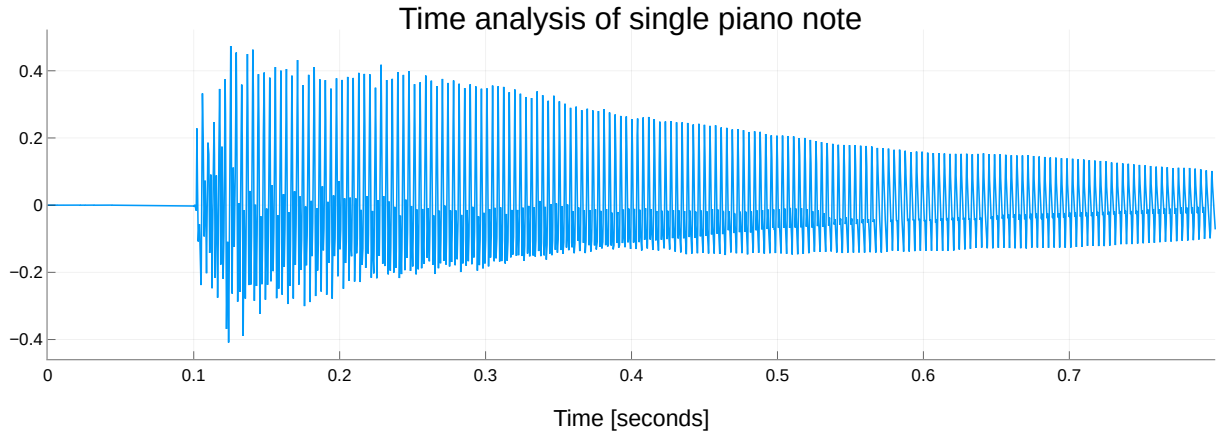


Spectral Analysis of single piano notes

In [36]:

```
y,dt=timeDomain(data,fs,1.2,2,"Time analysis of single piano note");
```

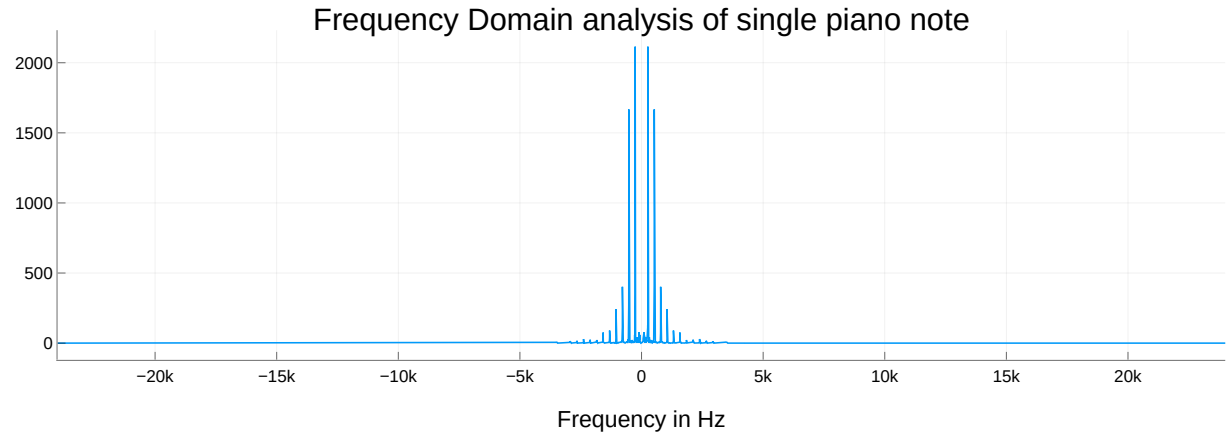
```
N = length(y) = 38401
```



Frequency domain of a single Piano note

In [37]:

```
freqDomain(y,dt,"Frequency Domain analysis of single piano note")
```



Out[37]:

```
Plots.GRBackend()
```

B.Spectral Analysis of Where-Is-The-Love-song

Time domain analysis for first 10 seconds

In [88]:

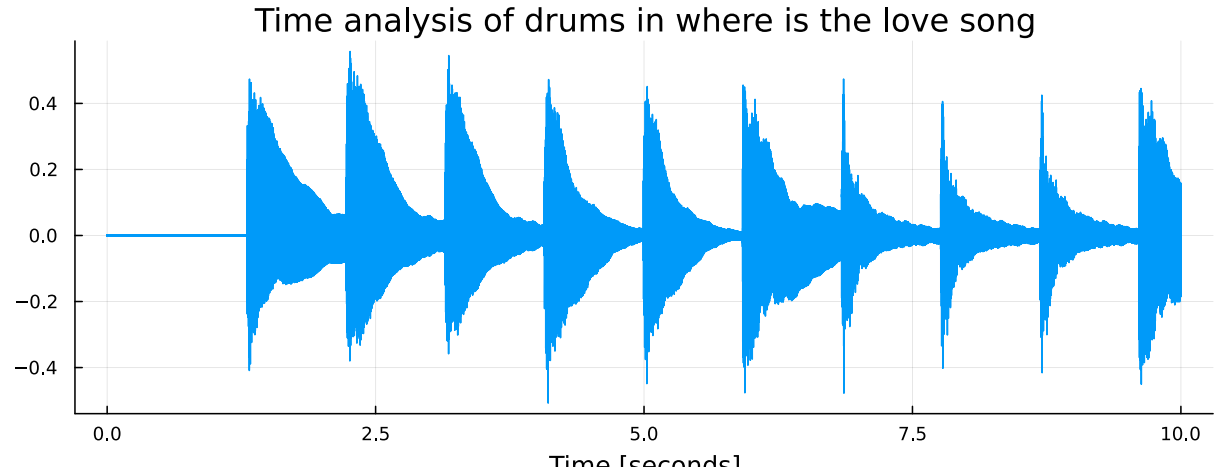
```
data,fs=loadfile("The_Black_Eyed_Peas_-_Where_Is_The_Love_20s.wav",0);#read the  piano audio file and play i
```

```
(Nsamples, Nchannels) = size(data) = (901834, 2)
Sample rate fs = 44100.0
```

In [39]:

```
y,dt=timeDomain(data,fs,0,10,"Time analysis of drums in where is the love song");
```

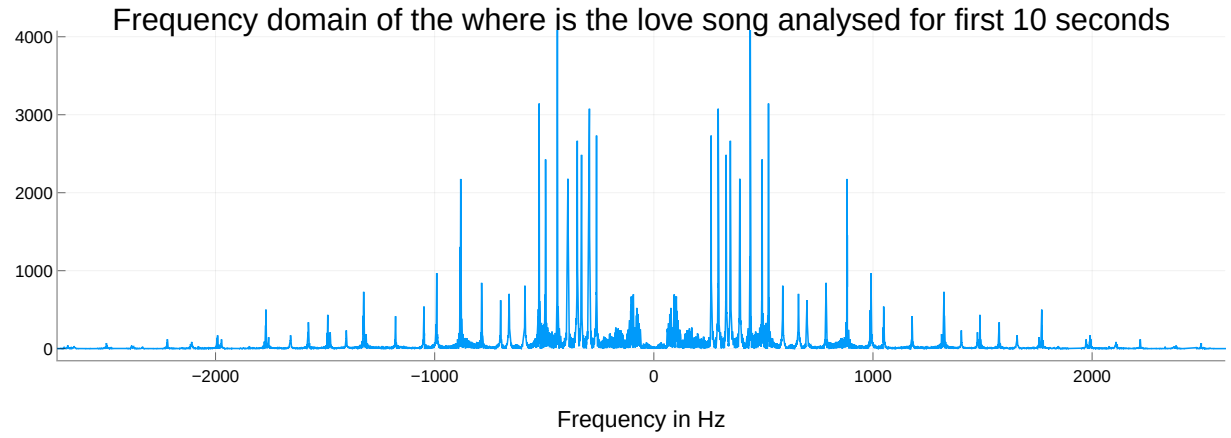
```
N = length(y) = 480001
```



Frequency domain of the where is the Song

In [89]:

```
freqDomain(y,dt,"Frequency domain of the where is the love song analysed for first 10 seconds");
```



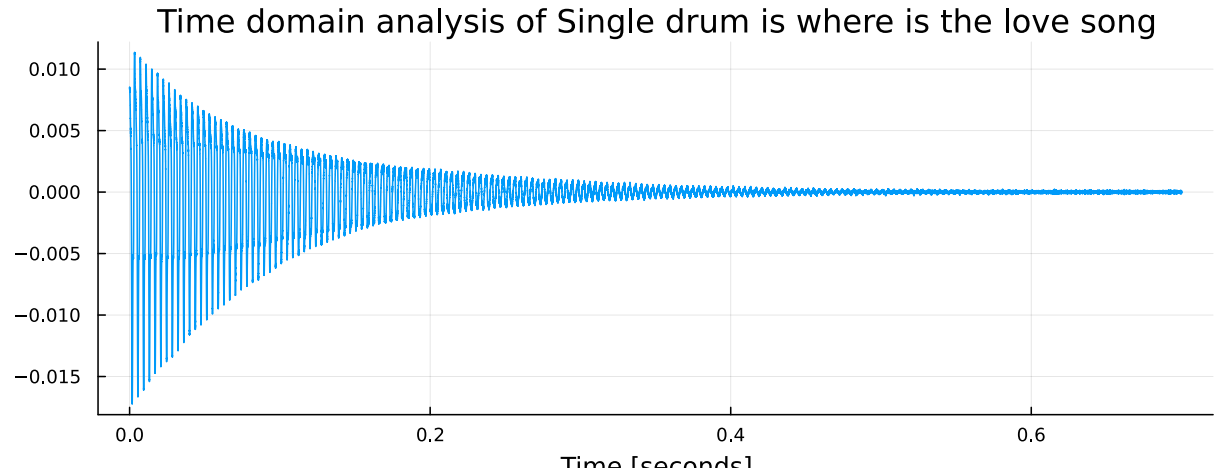
In [41]:

```
#### Extraction and Analysis of single drum
```

In [42]:

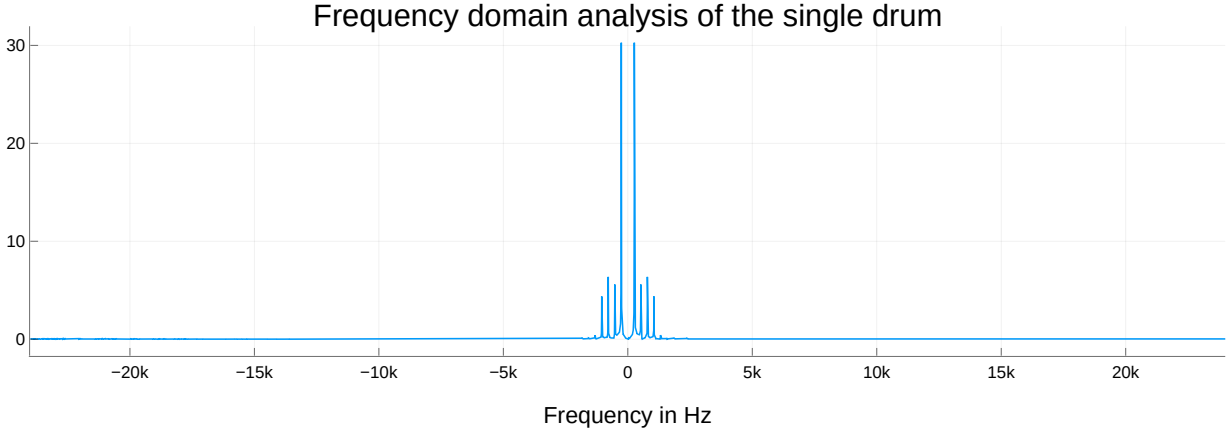
```
y,dt=timeDomain(data,fs,15.3,16,"Time domain analysis of single drum is where is the love song");
```

```
N = length(y) = 33601
```



In [43]:

```
freqDomain(y,dt,"Frequency domain analysis of the single drum");
```



C. Spectral of Voice audio-Vocals

In [44]:

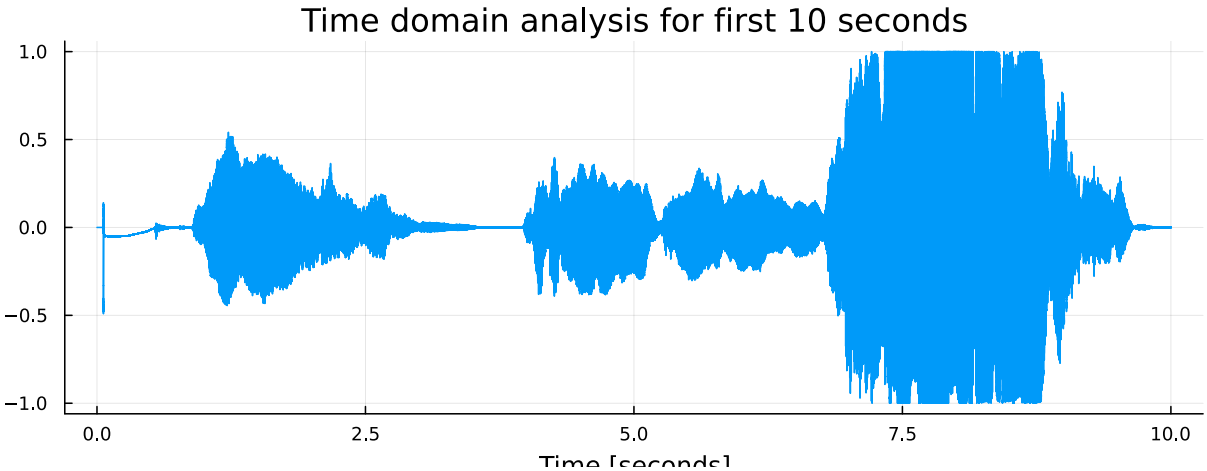
```
data,fs=loadfile("recorded_voice.wav",0);
```

```
(Nsamples, Nchannels) = size(data) = (1232640, 1)
Sample rate fs = 48000.0
```

In [45]:

```
y,dt=timeDomain(data,fs,0,10,"Time domain analysis for first 10 seconds");
```

```
N = length(y) = 480001
```



In [46]:

```
freqDomain(y,dt,"Frequency domain analysis for first 10 seconds");
```

Excercise 2.5.8 Spectrogram Plots

In [95]:

```
function spectrogram(data,fs,T)
    channel1=data[:,1];
    dt=1/fs;
    len=Int(round(0.1/dt)) # length in samples
    Nrows=len;
    Ncolumns=T ;
    matrix=zeros(Nrows,Ncolumns); # Define a 2D array (initially filled with zeros)
    for col=1:Ncolumns
        #@show col
         i1 = 1+(col-1)*len ;
       # @show i1
         i2 = i1+len-1; # Determine start and end time indexes
          X = fft(channel1[i1:i2]) # Calculate the fft of a portion of signal x
          matrix[:,col] = abs.(X) # Assign n'th column of matrix to abs.(X)
    end ;# The following syntax would also work: matrix[1:Nrows,col]=abs.(X[1:len
    Nt=Nrows; Nf=Ncolumns; dt=0.1; df=0.1;
    t=(0:Nt-1)*dt; f=(0:Nf-1)*df;
    # Fill a 2D array with some rand
    #fig = heatmap(t, f, matrix,xlabel="Time [s]", ylabel="Frequency [Hz]", clims=(0,4));
    plotly()
#    fig=heatmap(t,f,matrix,xlabel="Time[s]",ylabel="Frequency [Hz]");
     fig=heatmap(matrix[1:200,1:100].^0.5,xlabel="Time[s]",ylabel="Frequency [Hz]");
     display(fig);
end;
```

In [96]:

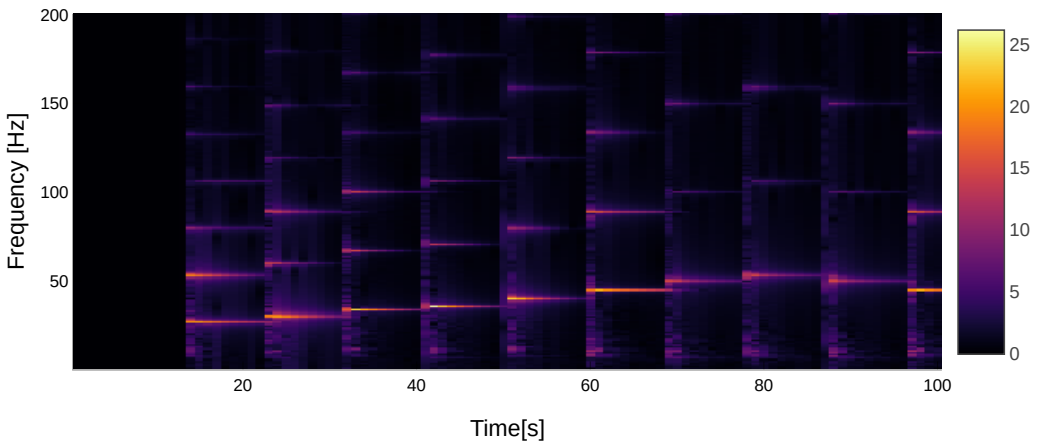
```
data,fs=loadfile("Piano_Scale_20s.wav",0);
```

```
(Nsamples, Nchannels) = size(data) = (971495, 2)
Sample rate fs = 48000.0
```

Spectrogram plot of Piano Notes

In [97]:

```
spectrogram(data,fs,100);
```



Spectrogram plot for Where is the love song

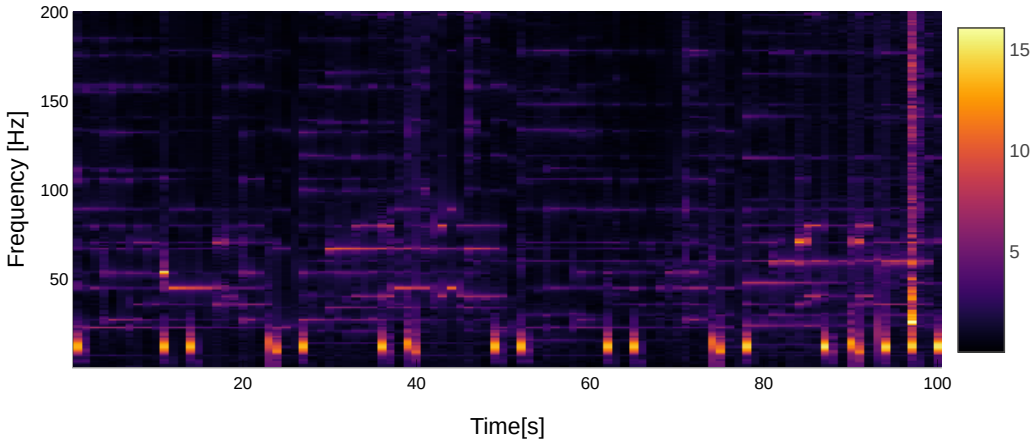
In [98]:

```
data,fs=loadfile("The_Black_Eyed_Peas_-_Where_Is_The_Love_20s.wav",0);#read the  piano audio file and play i
```

```
(Nsamples, Nchannels) = size(data) = (901834, 2)
Sample rate fs = 44100.0
```

In [99]:

```
spectrogram(data,fs,100);
```



Spectrogram of voice

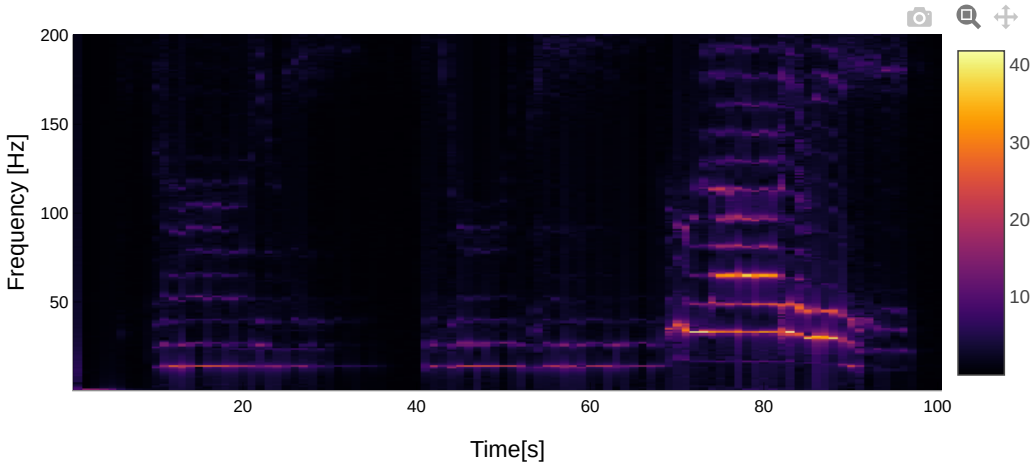
In [93]:

```
data,fs=loadfile("recorded_voice.wav",0);
```

```
(Nsamples, Nchannels) = size(data) = (1232640, 1)
Sample rate fs = 48000.0
```

In [94]:

```
spectrogram(data,fs,100);
```



In [ ]:

```

```