DSB-SC with no noise

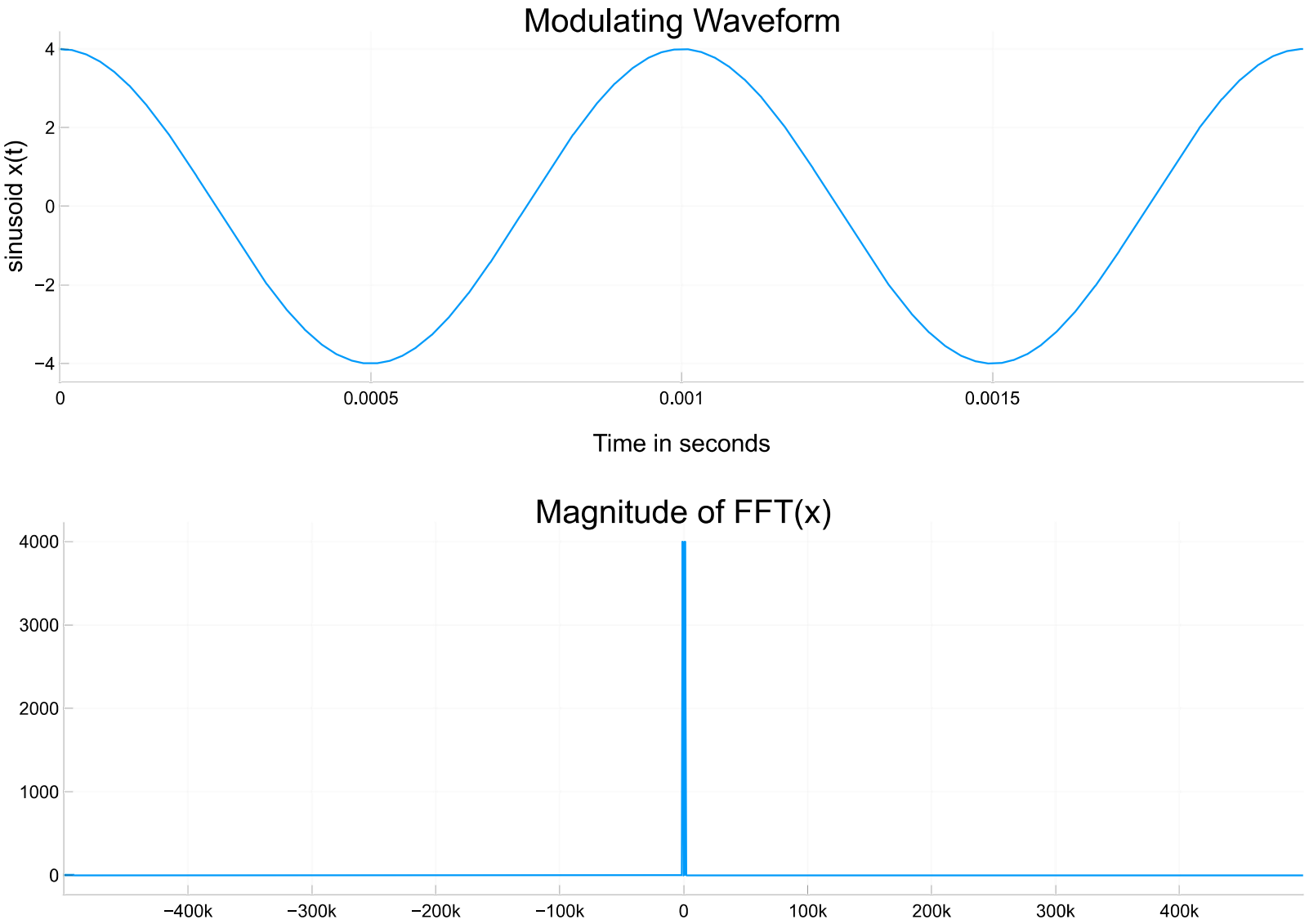
In [81]:

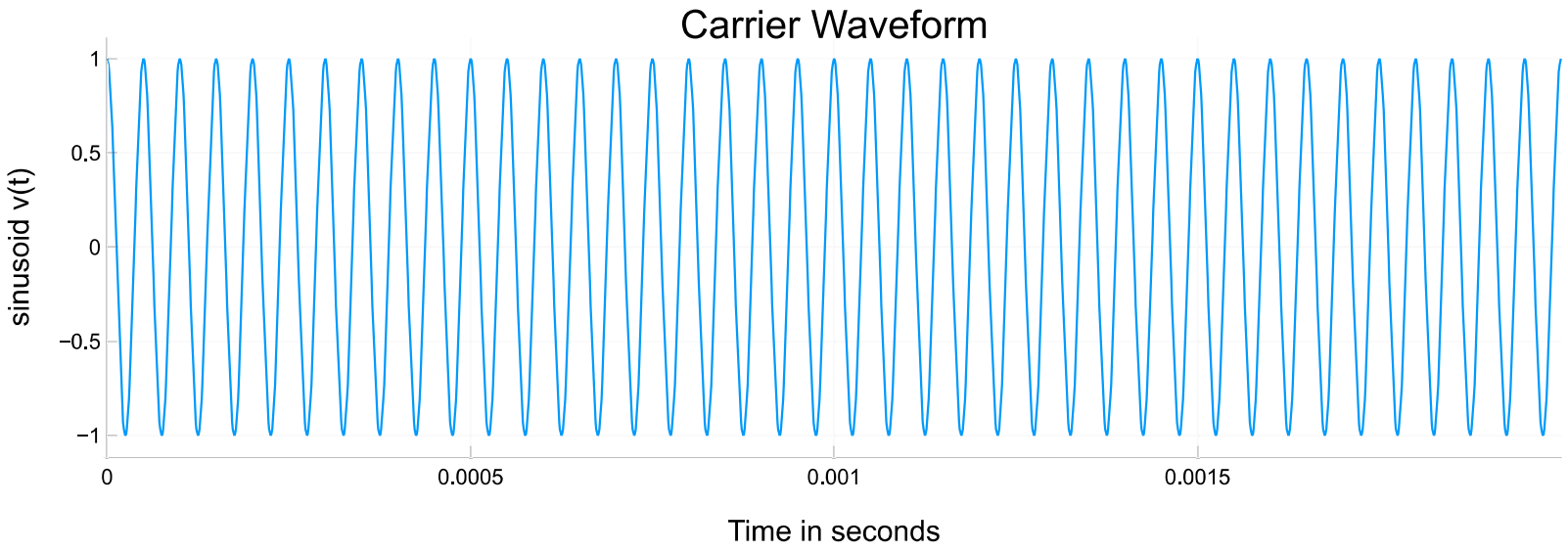
```
using FFTW
using Plots

Plots.plotly();    # Specify Plotly backend which allows zooming with a mouse in Jupyter Notebook.

Plots.default(size=(800,300)); # Set default plot canvas size

Plots.default(label=""); # Turn off legends by default

Plots.default(ticks=:native);  # Ticks on x-axis are labelled properly when zooming in.




Δt = 0.000001   # time step  (to get Greek sybmol, type \Delta <tab>)
t = 0:Δt:0.002;  #Define time from 0 to 1s in steps of 0.01s

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



#Setting up modulating Waveform

fm = 1000 # 10 Hz
ωm = 2*pi*fm;   # rad/s   ( Greek symbol \omega <tab>  )
A = 4

x = A*cos.(ωm*t);   # Create an array holding the sinusoid values
```

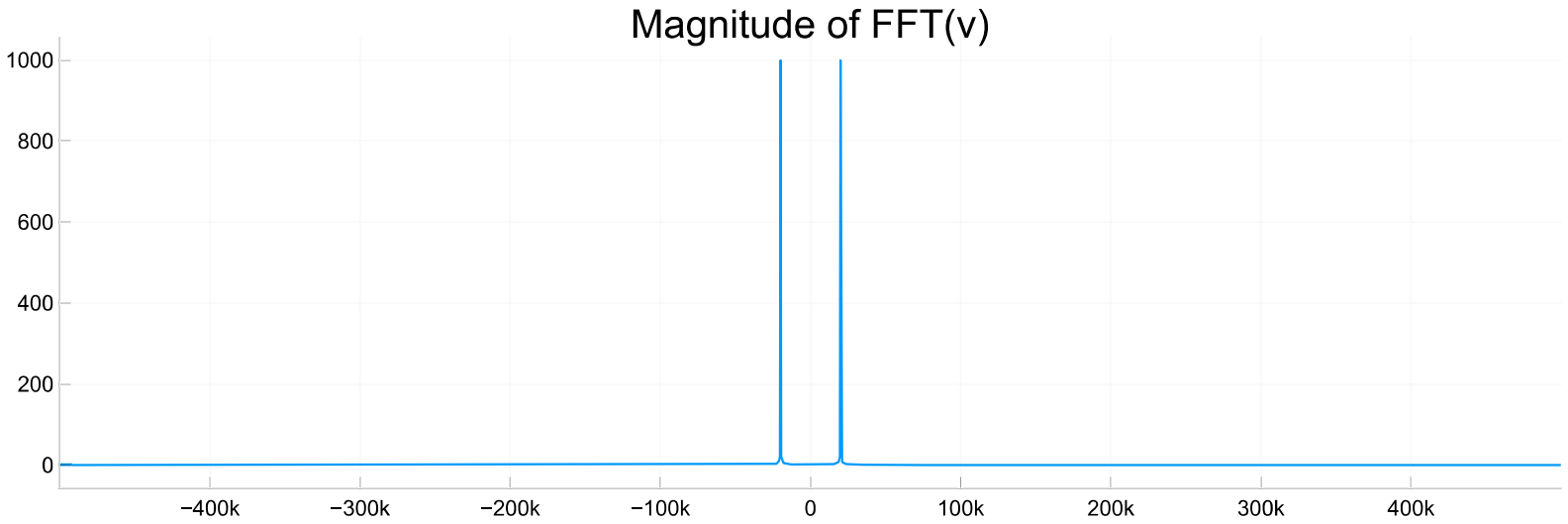
```
fig = Plots.plot(t,x)
xlabel!("Time in seconds");
ylabel!("sinusoid x(t)")
title!("Modulating Waveform")
display(fig)

X = fft(x)

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

fig = Plots.plot( f_axis, fftshift(abs.(X)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(x)")
display(fig);









#Setting up Carrier Waveform

fc = 20000  # 10 Hz
ωc = 2*pi*fc;   # rad/s   ( Greek symbol \omega <tab>  )
A = 1

v = A*cos.(ωc*t);   # Create an array holding the sinusoid values


fig = Plots.plot(t,v)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("Carrier Waveform")
display(fig)

V = fft(v)

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

fig = Plots.plot( f_axis, fftshift(abs.(V)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(v)")
display(fig);
```

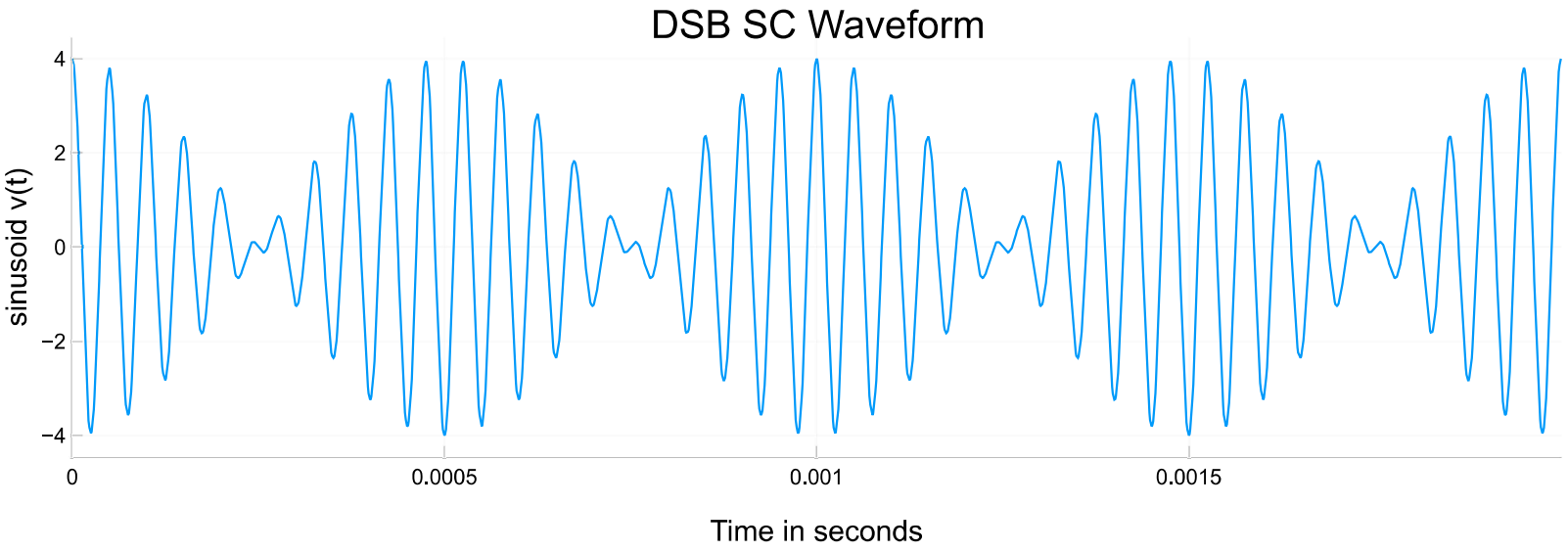
```
#Setting up modulated Waveform

z= x.*v

fig = Plots.plot(t,z)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Z = fft(z)

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

fig = Plots.plot( f_axis, fftshift(abs.(Z)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);









#Setting up demodulated Waveform (Before LPF)

y = z.*v

fig = Plots.plot(t,y)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)
```

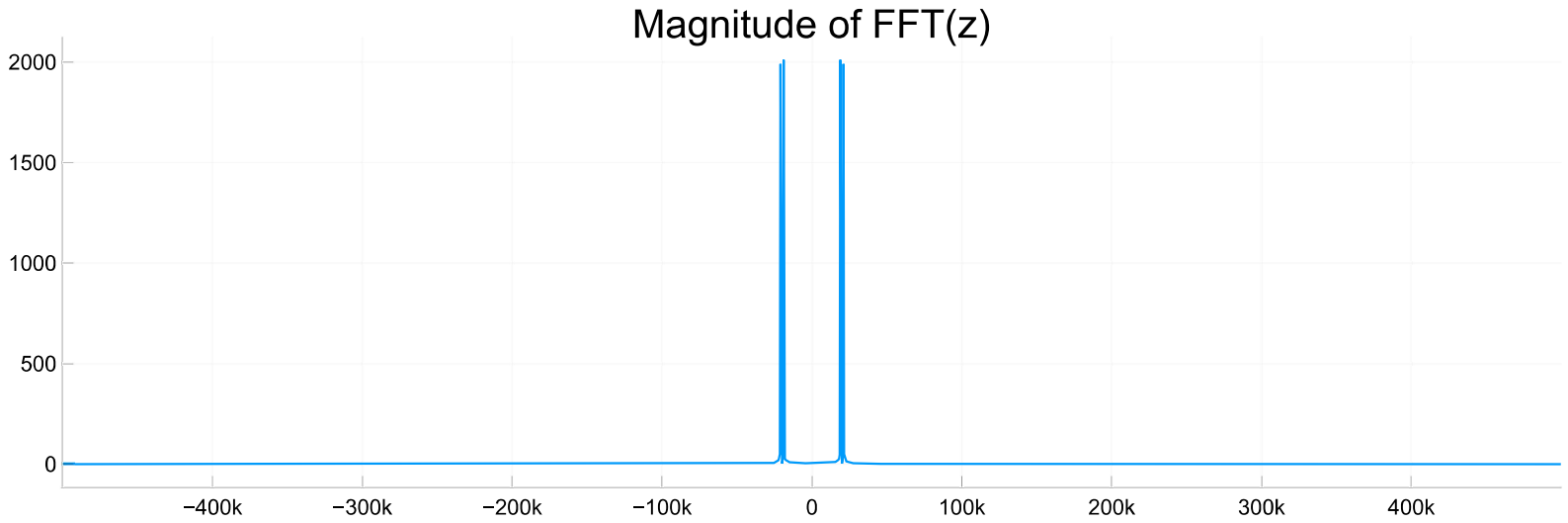
```
Y = fft(y)

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

fig = Plots.plot( f_axis, fftshift(abs.(Y)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(z)")
display(fig);









#Setting up demodulating Waveform (After LPF)

R=1000
C=0.00000007957

# Create a discrete impulse response to model an RC LPF
h = 1/(R*C).*exp.(-t/(R*C)) * Δt;   # with additional factor Δt

H = fft(h);

D = Y.*H

d = ifft(D)
d = real(d)

fig = Plots.plot(t, d)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)

fig = Plots.plot( f_axis, fftshift(abs.(D)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(d)")
display(fig);
```

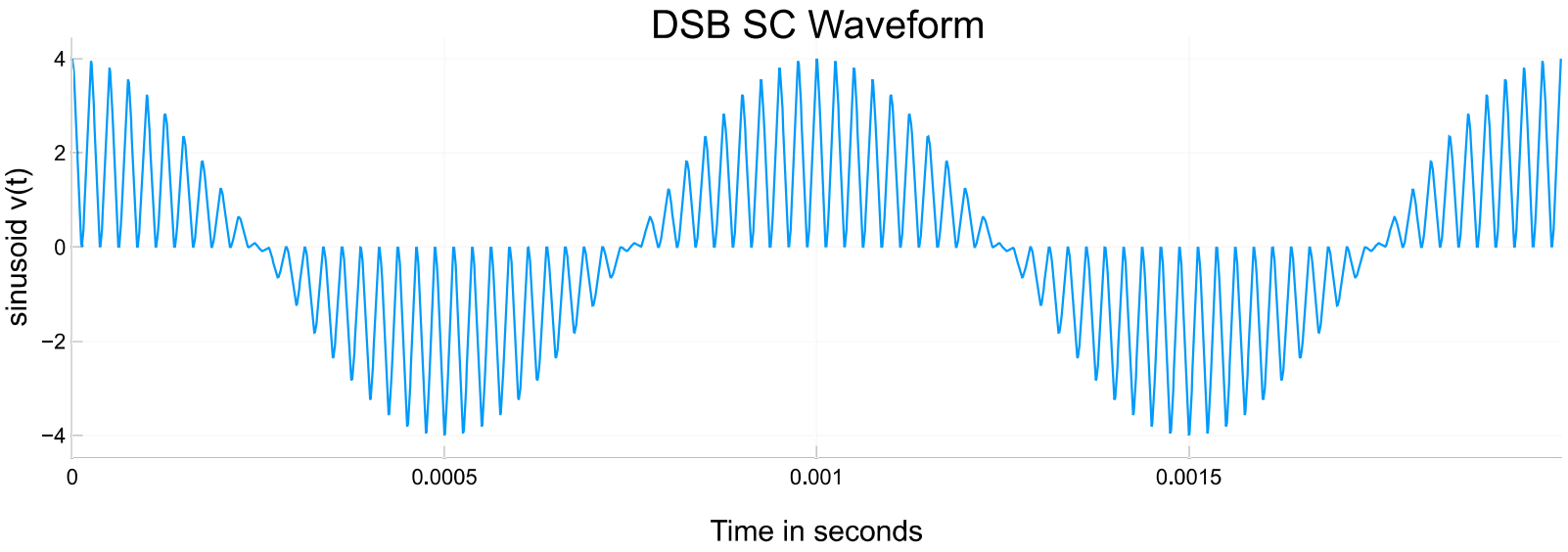

Modulating Waveform
sinusoid x(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds
Magnitude of FFT(x)
4000
3000
2000
1000
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

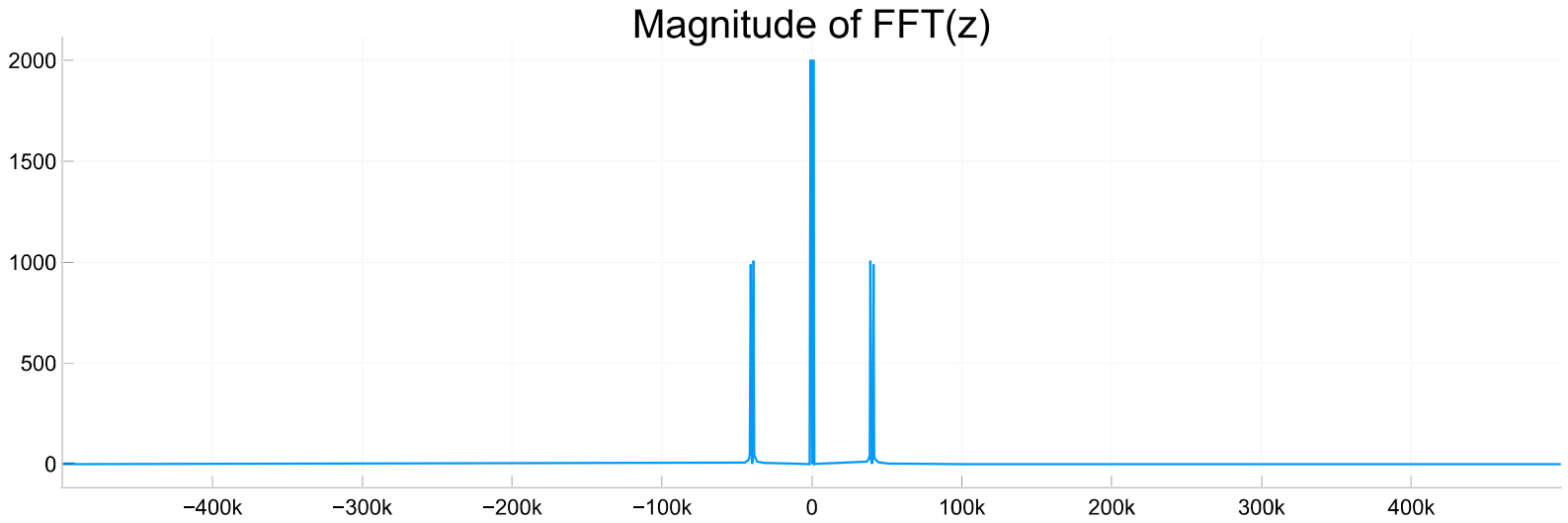

Carrier Waveform
sinusoid v(t)
1
0.5
0
−0.5
−1
0
0.0005
0.001
0.0015
Time in seconds

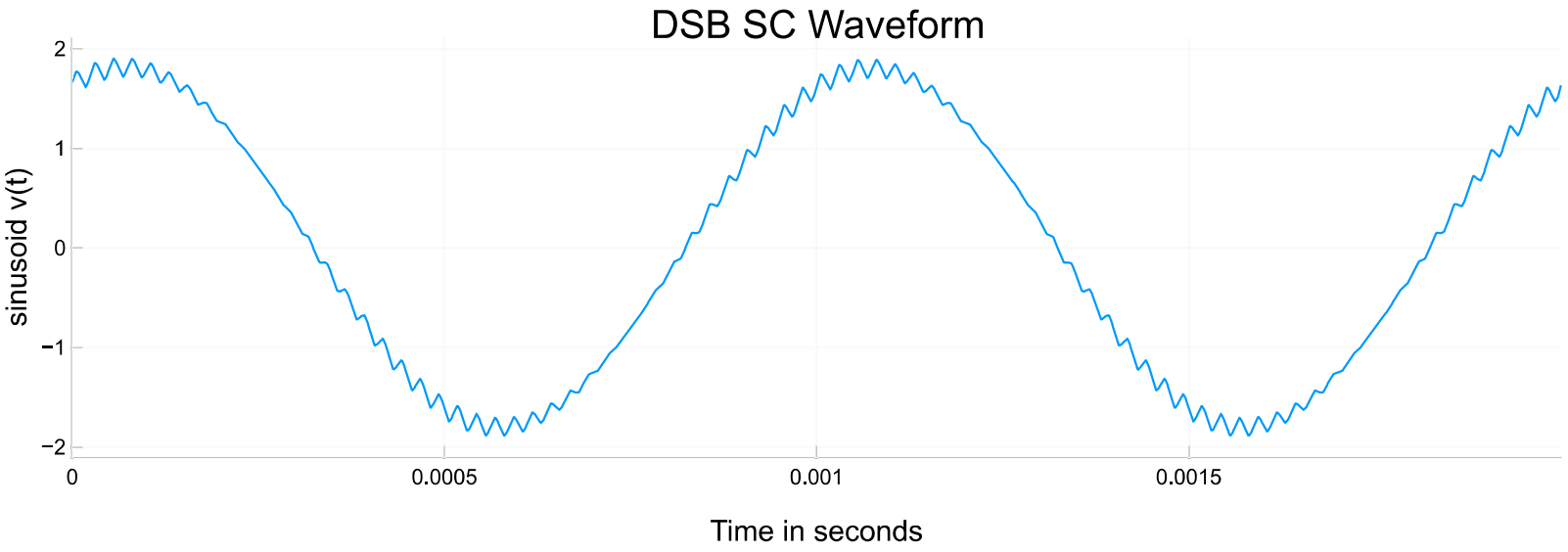

Magnitude of FFT(v)
1000
800
600
400
200
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

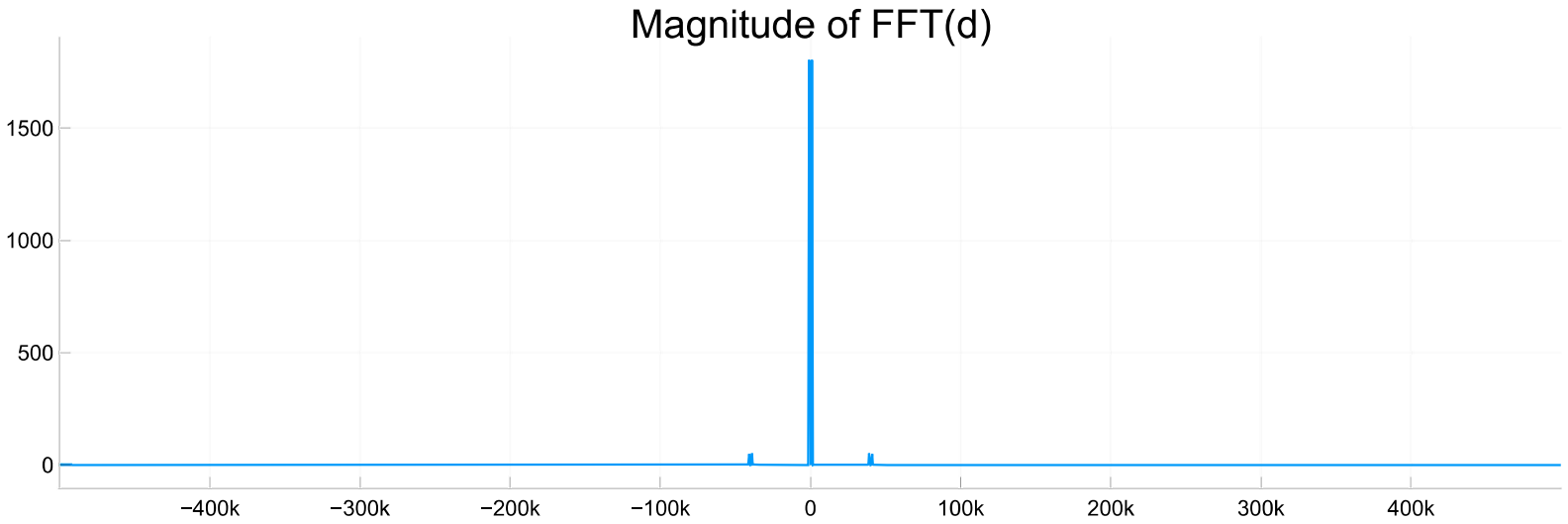

DSB SC Waveform
sinusoid v(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
2000
1500
1000
500
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
2000
1500
1000
500
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k





DSB-SC with a low level of noise

In [89]:
```
using FFTW
```

```julia
using Plots

Plots.plotly();    # Specify Plotly backend which allows zooming with a mouse in Jupyter Notebook.

Plots.default(size=(800,300)); # Set default plot canvas size

Plots.default(label=""); # Turn off legends by default

Plots.default(ticks=:native);  # Ticks on x-axis are labelled properly when zooming in.




Δt = 0.000001   # time step  (to get Greek sybmol, type \Delta <tab>)
t = 0:Δt:0.002;  #Define time from 0 to 1s in steps of 0.01s

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



#Setting up modulating Waveform

fm = 1000 # 10 Hz
ωm = 2*pi*fm;   # rad/s   ( Greek symbol \omega <tab>  )
A = 4

x = A*cos.(ωm*t);   # Create an array holding the sinusoid values



fig = Plots.plot(t,x)
xlabel!("Time in seconds");
ylabel!("sinusoid x(t)")
title!("Modulating Waveform")
```

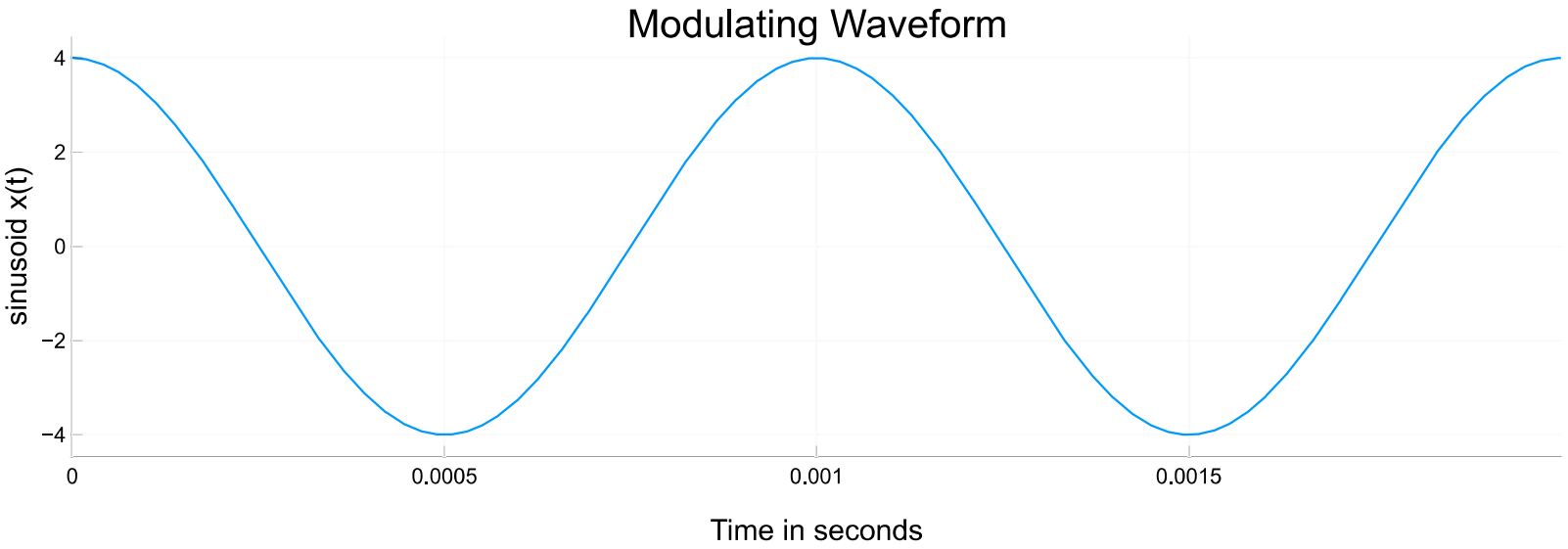
```
display(fig)

X = fft(x)

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

fig = Plots.plot( f_axis, fftshift(abs.(X)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(x)")
display(fig);









#Setting up Carrier Waveform

fc = 20000  # 10 Hz
ωc = 2*pi*fc;   # rad/s   ( Greek symbol \omega <tab>  )
A = 1

v = A*cos.(ωc*t) ;   # Create an array holding the sinusoid values


fig = Plots.plot(t,v)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("Carrier Waveform")
display(fig)

V = fft(v)

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

fig = Plots.plot( f_axis, fftshift(abs.(V)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(v)")
display(fig);
```

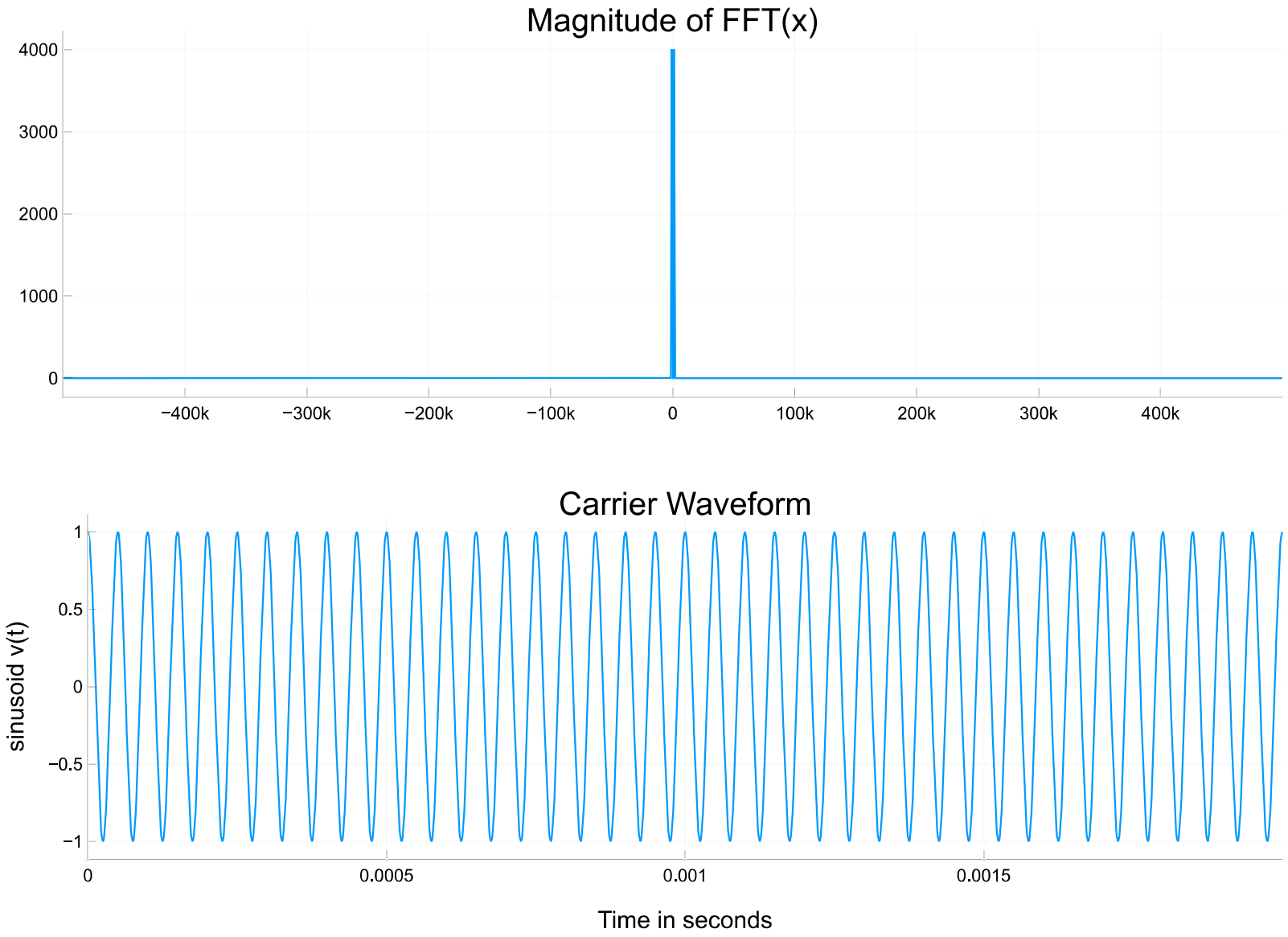
```
# Now generate some noise

# Very compact one-line definition of a rect function. Value is 0 or 1.
rect(t) = (abs.(t).<=0.5)*1.0

Δω = 2*pi/(N*Δt)   # Sample spacing in freq domain in rad/s

ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 6000 # filter bandwidth in Hz

H = rect((ω .- ωc)/(2*π*B)) + rect( (ω .+ (ωc .- 2*π/Δt) )/(2*π*B) )


N = length(t);

noise = randn(N);  # Create an array of N zero-mean Gaussian random number of std dev = 1.

μ = 0.0     # desired mean
σ = 1 # desired standard deviation  NOTE: to create Greek symbol, \sigma<tab>
noise = noise*σ .+ μ


N = fft(noise)
N = real(N)

Noise = H.*N

noise = ifft(Noise)
noise = real(noise)


# Calculate statistics:  mean and standard deviation
using Statistics  #The Statistics module contains basic statistics functionality (mean, std, var etc.)

noise_var = var(noise)
noise_std = std(noise)
```

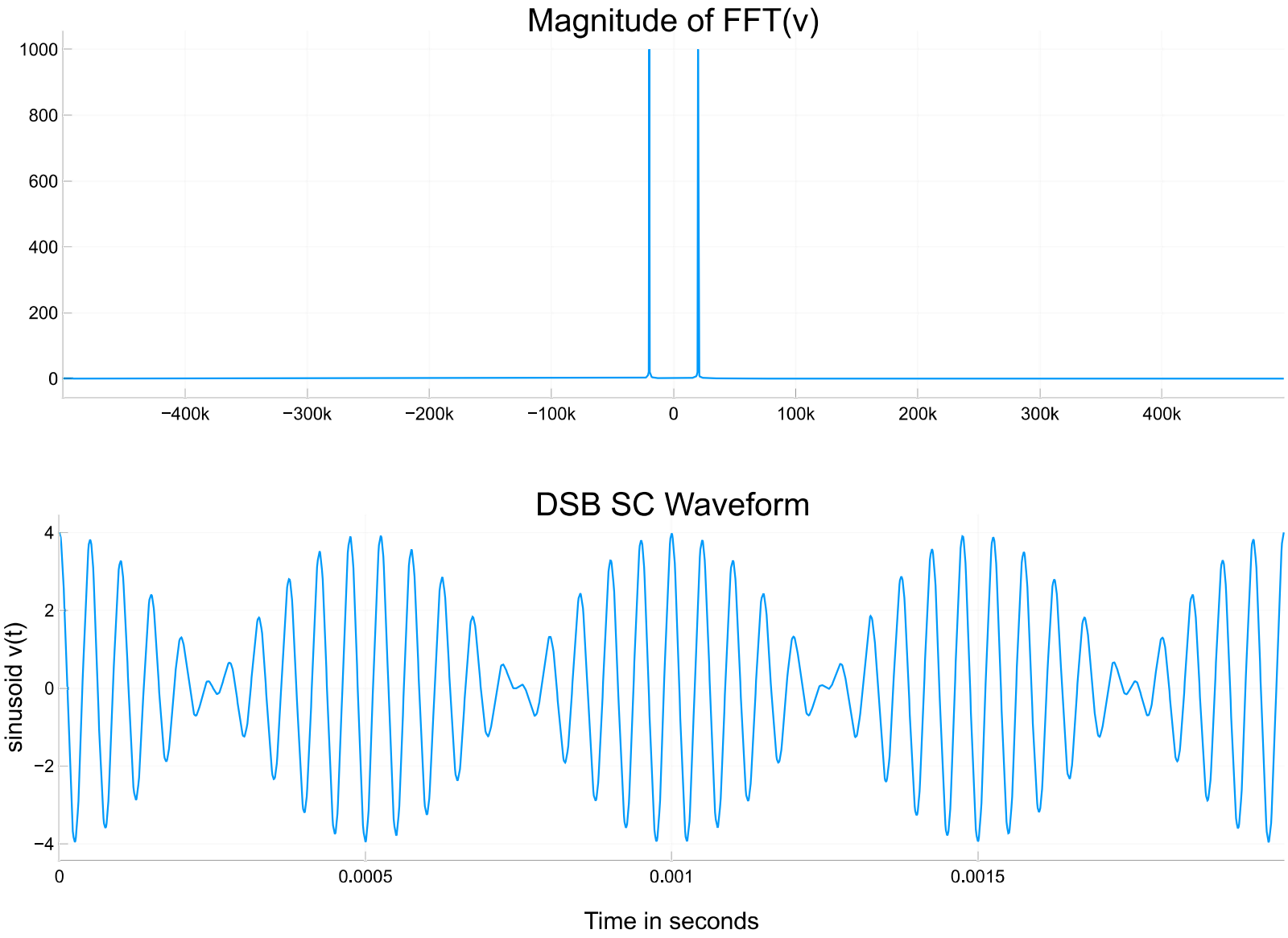
```
println("Noise Statistics:")
println("Variance: $(noise_var)")
println("Standard Deviation: $(noise_std)")









#Setting up modulated Waveform

z= x.*v + noise

fig = Plots.plot(t,z)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Z = fft(z)

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

fig = Plots.plot( f_axis, fftshift(abs.(Z)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);









#Setting up demodulated Waveform (Before LPF)

y = z.*v

fig = Plots.plot(t,y)
xlabel!("Time in seconds");
```

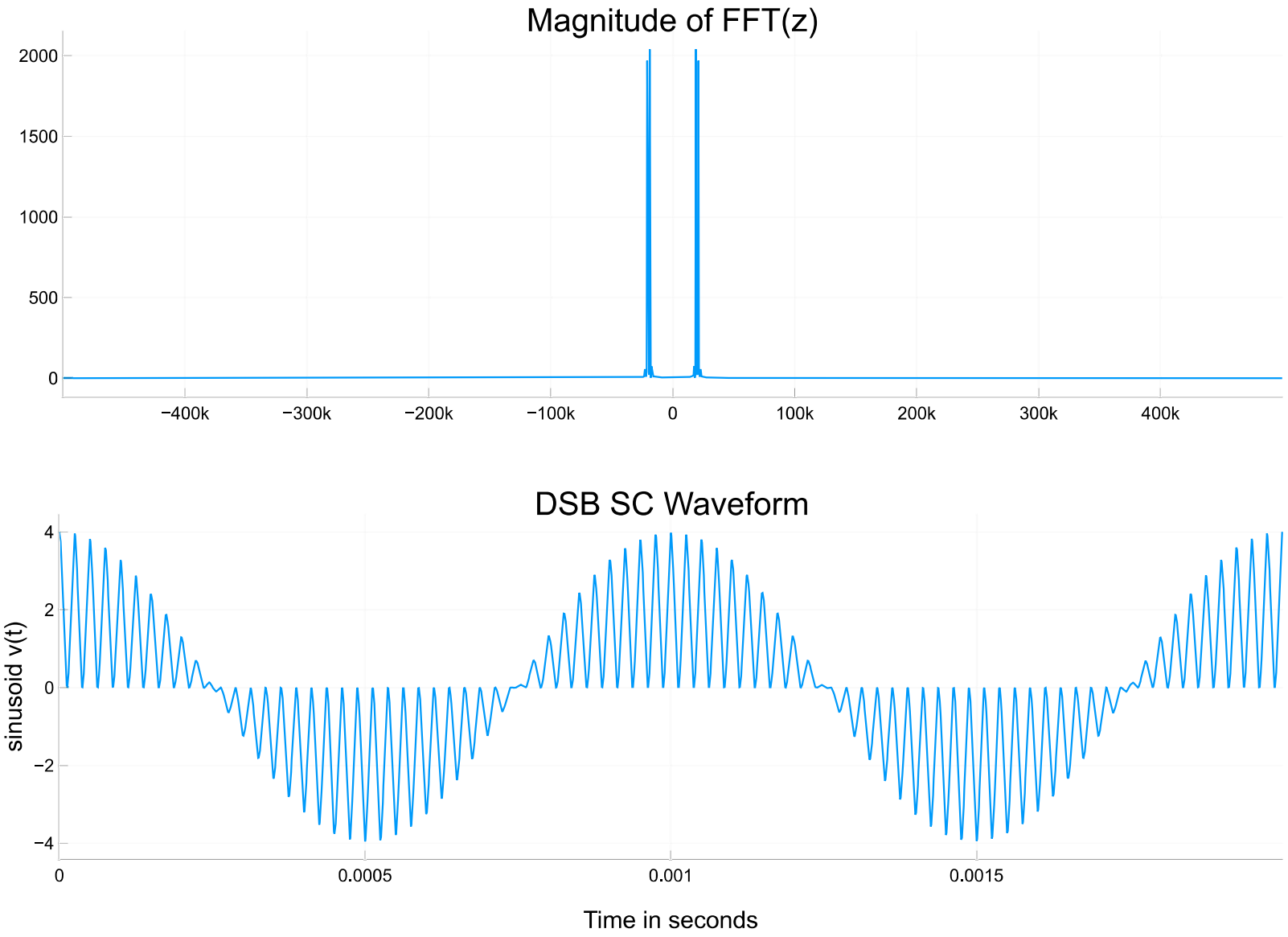
```
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Y = fft(y)

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

fig = Plots.plot( f_axis, fftshift(abs.(Y)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);

y_var = var(y)

SNR_before = 10*log(y_var/noise_var)


println()
println("SNR before LPF = $(SNR_before) dB")









#Setting up demodulating Waveform (After LPF)

R=1000
C=0.00000007957

# Create a discrete impulse response to model an RC LPF
h = 1/(R*C).*exp.(-t/(R*C)) * Δt;   # with additional factor Δt

H = fft(h);

D = Y.*H

d = ifft(D)
d = real(d)

fig = Plots.plot(t, d)
xlabel!("Time in seconds");
```

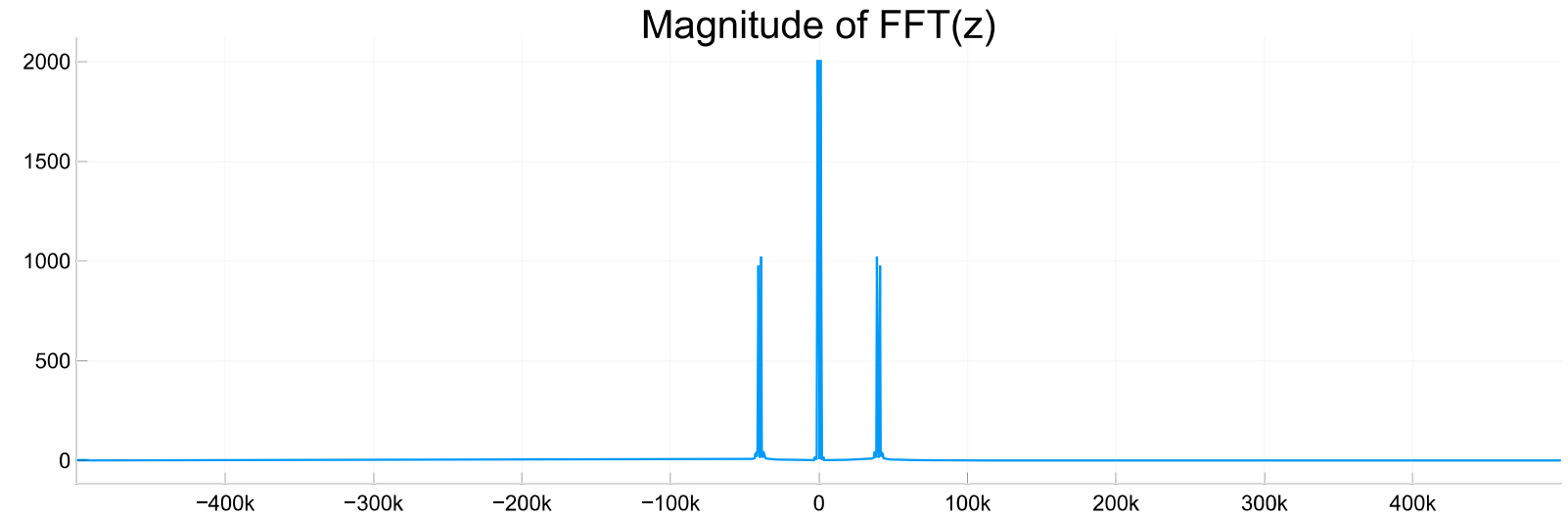
```
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)

fig = Plots.plot( f_axis, fftshift(abs.(D)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(d)")
display(fig);

d_var = var(d)

SNR_after = 10*log(d_var/noise_var)

println("SNR after LPF = $(SNR_after) dB")

SNR_factor = SNR_after/SNR_before

println()
println("The SNR has decreased by a factor of $(SNR_factor)")
```

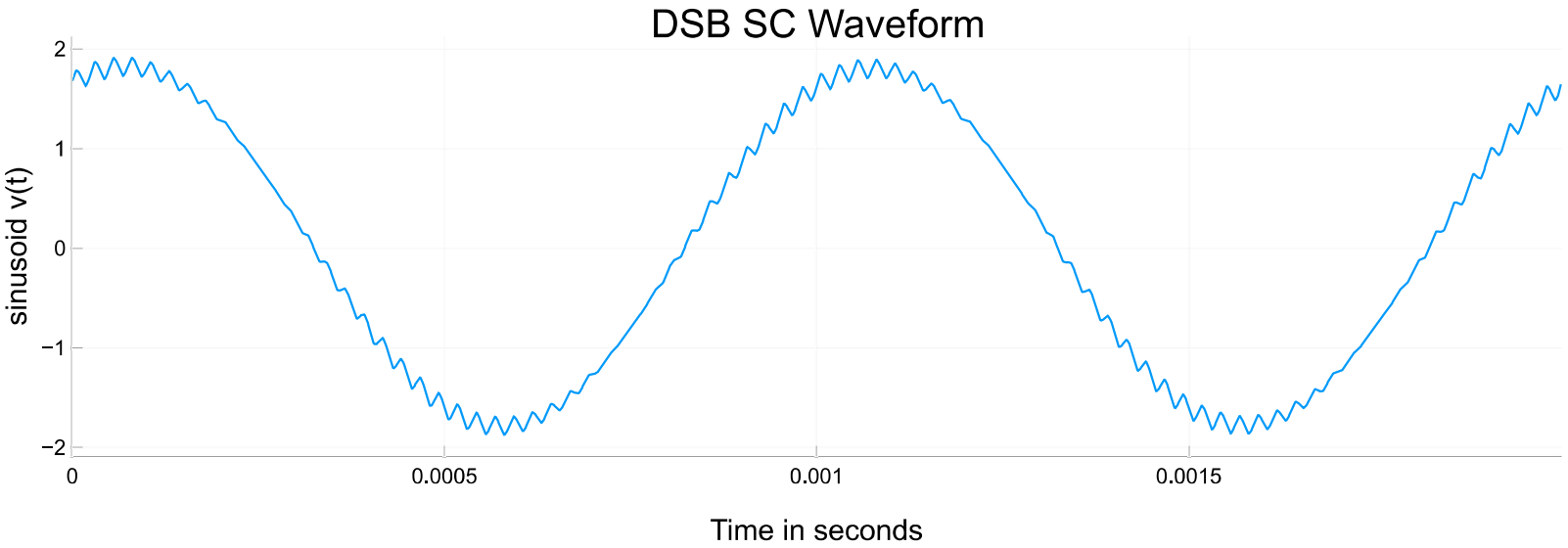


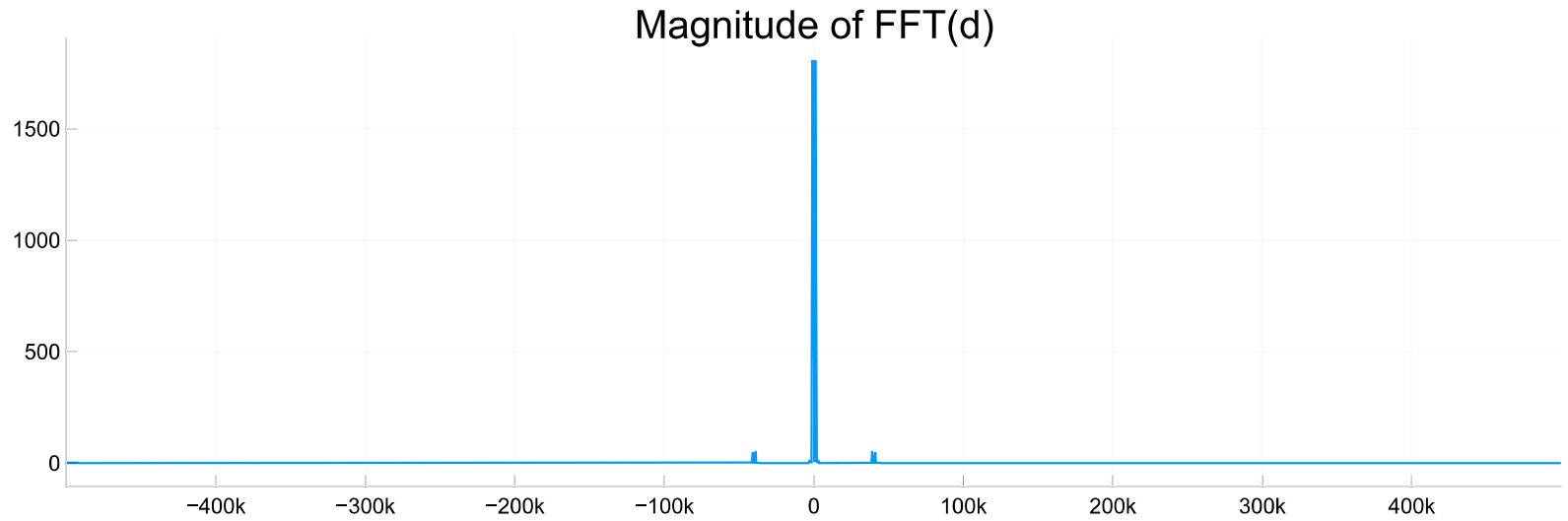

Magnitude of FFT(x)
4000
3000
2000
1000
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k
Carrier Waveform
sinusoid v(t)
1
0.5
0
−0.5
−1
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(v)
1000
800
600
400
200
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k
DSB SC Waveform
sinusoid v(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
2000
1500
1000
500
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k
DSB SC Waveform
sinusoid v(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
2000
1500
1000
500
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
2
1
0
−1
−2
0
0.0005
0.001
0.0015
Time in seconds




```
Noise Statistics:
Variance: 0.00438427166065477
Standard Deviation: 0.06621383284975103

SNR before LPF = 65.34752689810857 dB
SNR after LPF = 59.204640850748255 dB

The SNR has decreased by a factor of 0.905996656048844
```

DSB-SC with a high level of noise

In [90]:

```
using FFTW
using Plots

Plots.plotly();    # Specify Plotly backend which allows zooming with a mouse in Jupyter Notebook.

Plots.default(size=(800,300)); # Set default plot canvas size

Plots.default(label=""); # Turn off legends by default

Plots.default(ticks=:native);  # Ticks on x-axis are labelled properly when zooming in.
```

```
Δt = 0.000001   # time step  (to get Greek sybmol, type \Delta <tab>)
t = 0:Δt:0.002;  #Define time from 0 to 1s in steps of 0.01s

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



#Setting up modulating Waveform

fm = 1000 # 10 Hz
ωm = 2*pi*fm;   # rad/s   ( Greek symbol \omega <tab>  )
A = 4

x = A*cos.(ωm*t);   # Create an array holding the sinusoid values



fig = Plots.plot(t,x)
xlabel!("Time in seconds");
ylabel!("sinusoid x(t)")
title!("Modulating Waveform")
display(fig)

X = fft(x)

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

fig = Plots.plot( f_axis, fftshift(abs.(X)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(x)")
display(fig);
```

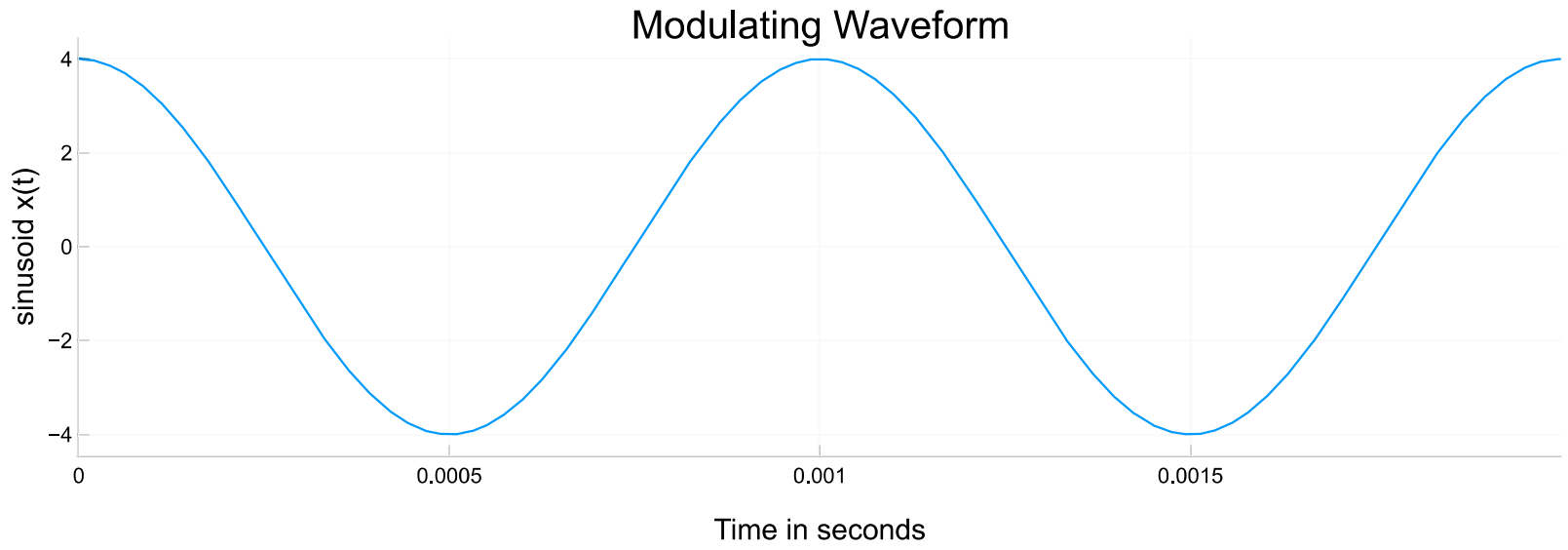
```
#Setting up Carrier Waveform

fc = 20000  # 10 Hz
ωc = 2*pi*fc;   # rad/s   ( Greek symbol \omega <tab>  )
A = 1

v = A*cos.(ωc*t) ;   # Create an array holding the sinusoid values


fig = Plots.plot(t,v)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("Carrier Waveform")
display(fig)

V = fft(v)

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

fig = Plots.plot( f_axis, fftshift(abs.(V)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(v)")
display(fig);









# Now generate some noise

# Very compact one-line definition of a rect function. Value is 0 or 1.
rect(t) = (abs.(t).<=0.5)*1.0

Δω = 2*pi/(N*Δt)   # Sample spacing in freq domain in rad/s
```

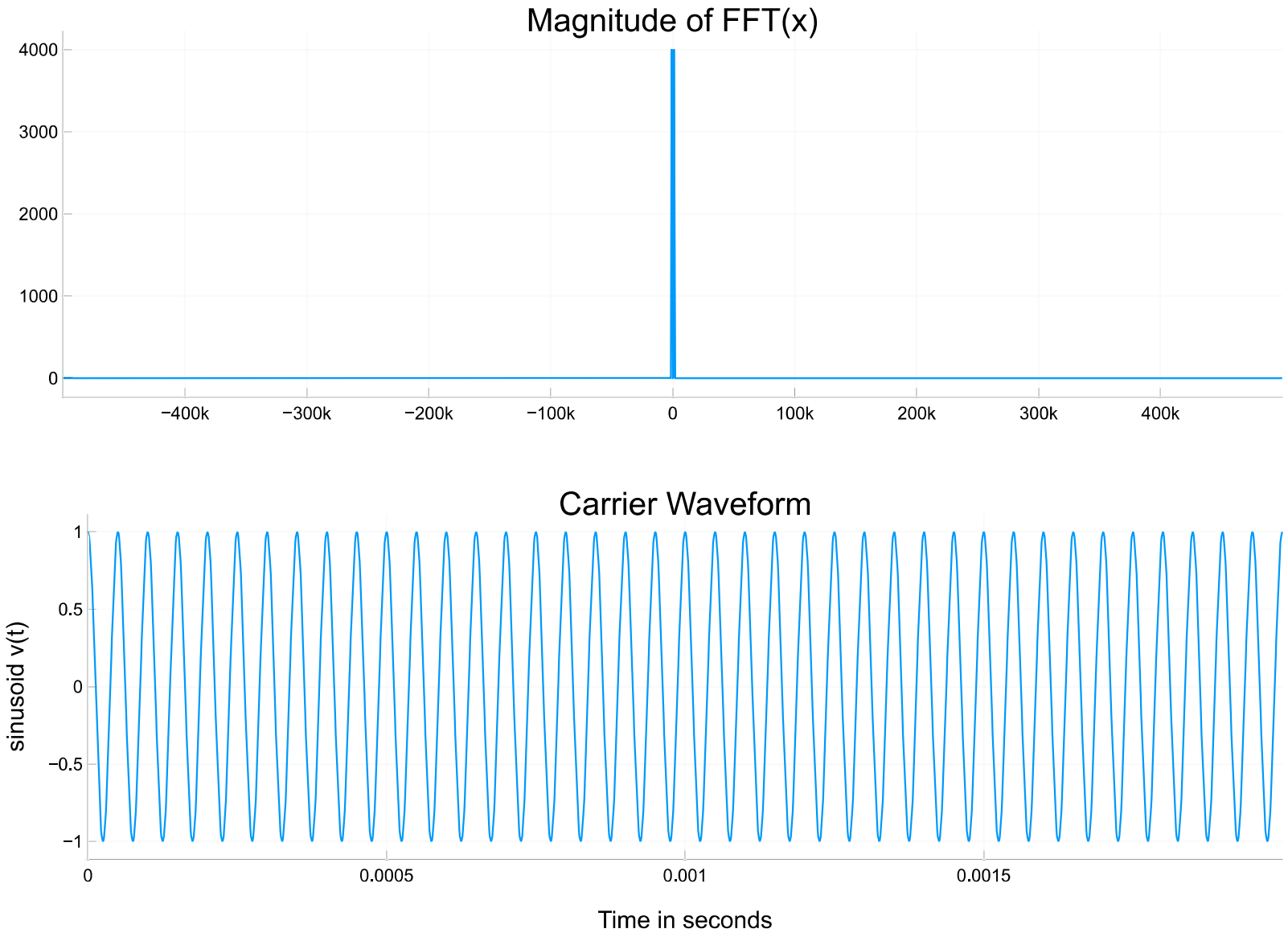
```
ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 6000 # filter bandwidth in Hz

H = rect((ω .- ωc)/(2*π*B)) + rect( (ω .+ (ωc .- 2*π/Δt) )/(2*π*B) )


N = length(t);

noise = randn(N);  # Create an array of N zero-mean Gaussian random number of std dev = 1.

μ = 0.0     # desired mean
σ = 5 # desired standard deviation  NOTE: to create Greek symbol, \sigma<tab>
noise = noise*σ .+ μ


N = fft(noise)
N = real(N)

Noise = H.*N

noise = ifft(Noise)
noise = real(noise)


# Calculate statistics:  mean and standard deviation
using Statistics  #The Statistics module contains basic statistics functionality (mean, std, var etc.)

noise_var = var(noise)
noise_std = std(noise)


println("Noise Statistics:")
println("Variance: $(noise_var)")
println("Standard Deviation: $(noise_std)")
```

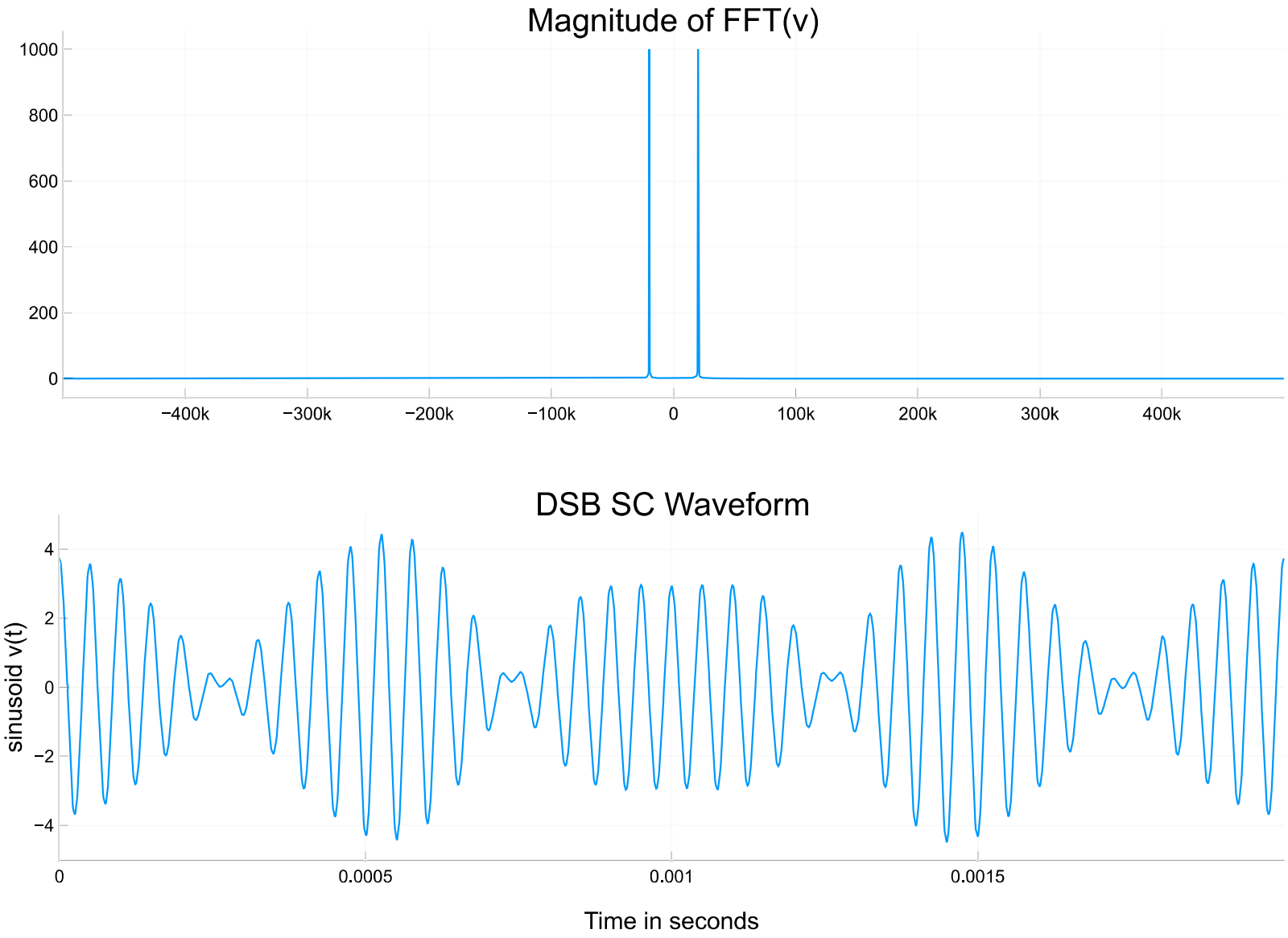
```
#Setting up modulated Waveform

z= x.*v + noise

fig = Plots.plot(t,z)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Z = fft(z)

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

fig = Plots.plot( f_axis, fftshift(abs.(Z)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(z)")
display(fig);









#Setting up demodulated Waveform (Before LPF)

y = z.*v

fig = Plots.plot(t,y)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Y = fft(y)

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

fig = Plots.plot( f_axis, fftshift(abs.(Y)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(z)")
```

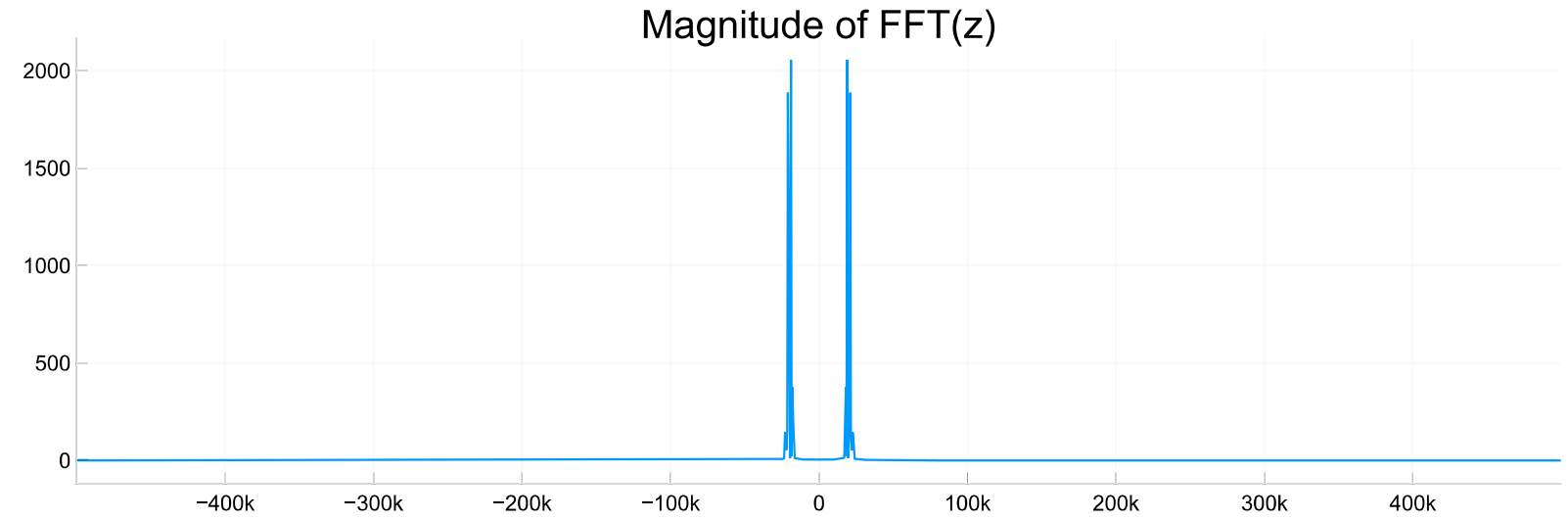
```
display(fig);

y_var = var(y)

SNR_before = 10*log(y_var/noise_var)


println()
println("SNR before LPF = $(SNR_before) dB")









#Setting up demodulating Waveform (After LPF)

R=1000
C=0.00000007957

# Create a discrete impulse response to model an RC LPF
h = 1/(R*C).*exp.(-t/(R*C)) * Δt;   # with additional factor Δt

H = fft(h);

D = Y.*H

d = ifft(D)
d = real(d)

fig = Plots.plot(t, d)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)

fig = Plots.plot( f_axis, fftshift(abs.(D)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(d)")
display(fig);

d_var = var(d)

SNR_after = 10*log(d_var/noise_var)
```

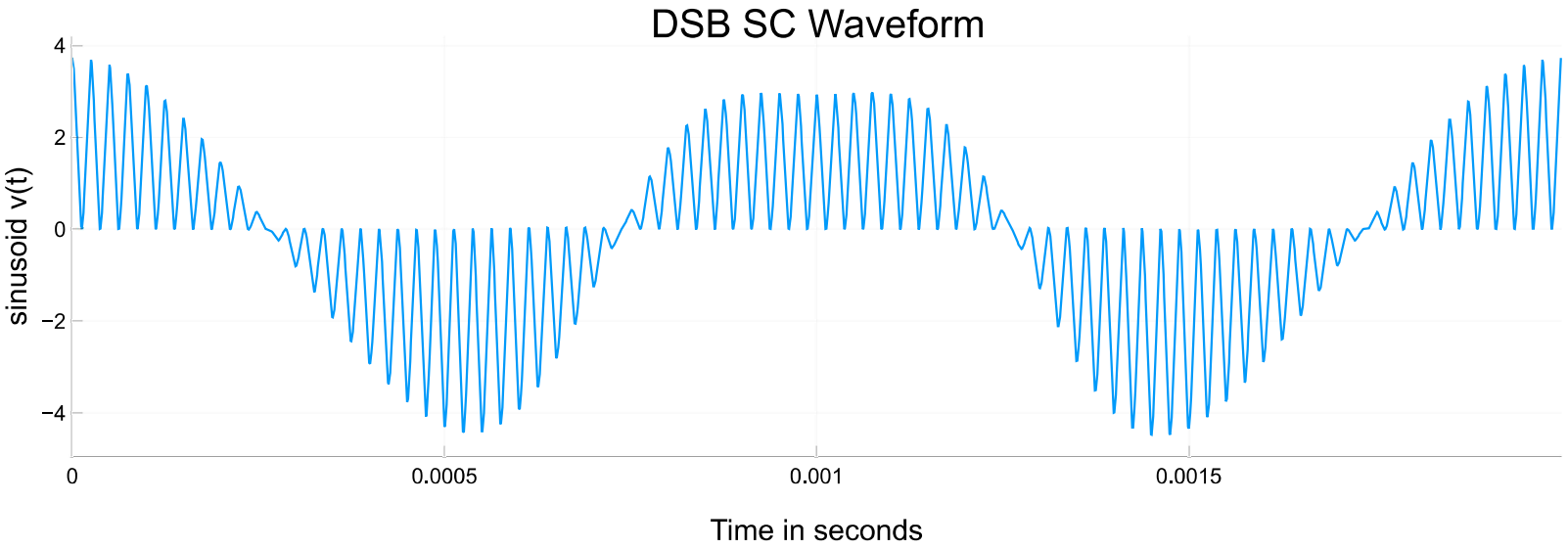
```
println("SNR after LPF = $(SNR_after) dB")

SNR_factor = SNR_after/SNR_before

println()
println("The SNR has decreased by a factor of $(SNR_factor)")
```

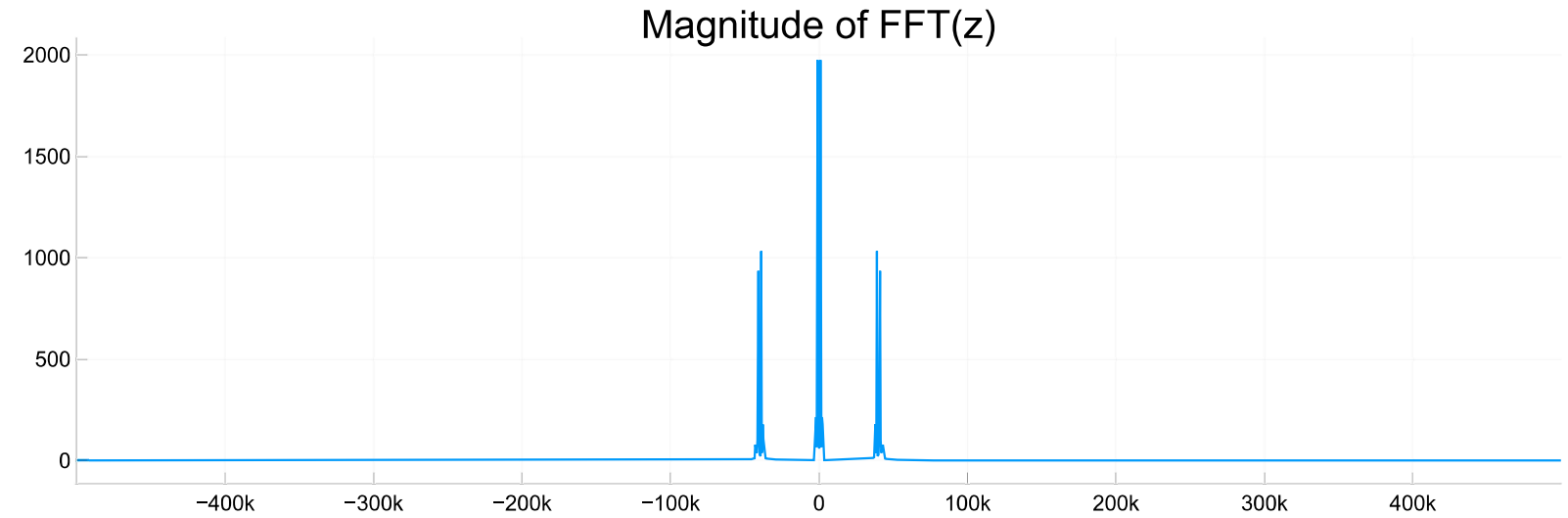


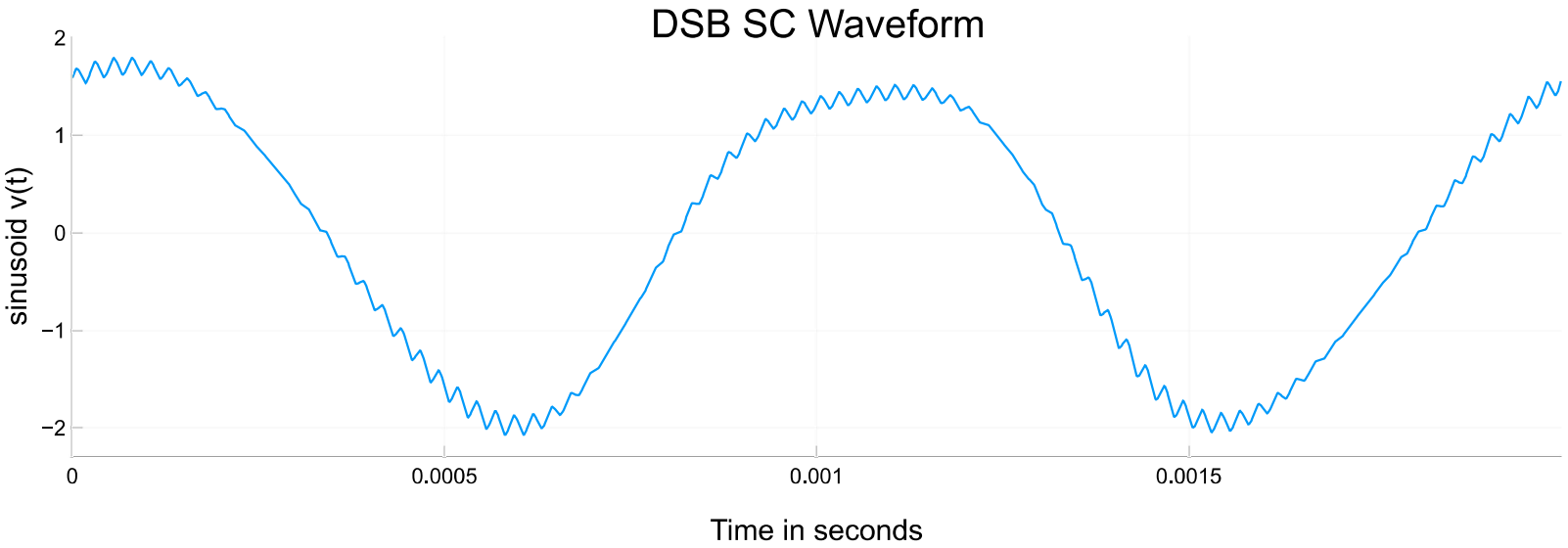

Magnitude of FFT(x)
4000
3000
2000
1000
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k
Carrier Waveform
sinusoid v(t)
1
0.5
0
−0.5
−1
0
0.0005
0.001
0.0015
Time in seconds

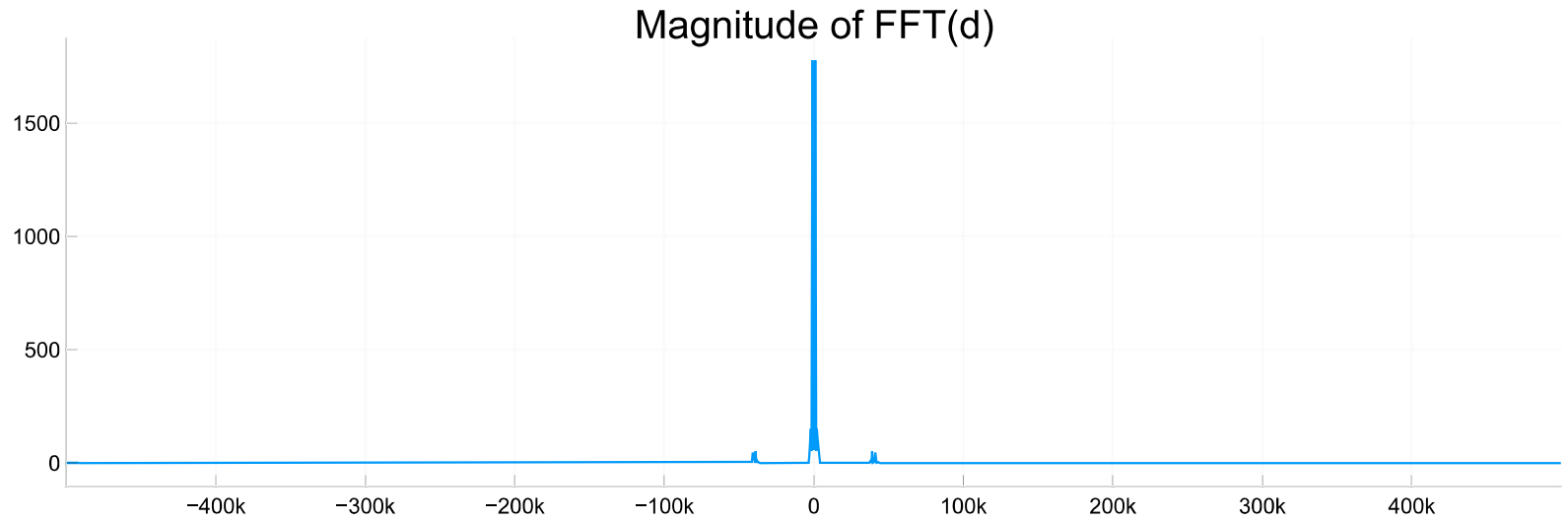

Magnitude of FFT(v)
1000
800
600
400
200
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k
DSB SC Waveform
sinusoid v(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
2000
1500
1000
500
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
2000
1500
1000
500
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
2
1
0
−1
−2
0
0.0005
0.001
0.0015
Time in seconds




```
Noise Statistics:
Variance: 0.14731032254778187
Standard Deviation: 0.38381026894519366

SNR before LPF = 30.152838432236535 dB
SNR after LPF = 23.88251766213079 dB

The SNR has decreased by a factor of 0.7920487391527918
```

DSB-LC with no noise

In [126…

```
using FFTW
using Plots

Plots.plotly();    # Specify Plotly backend which allows zooming with a mouse in Jupyter Notebook.

Plots.default(size=(800,300)); # Set default plot canvas size

Plots.default(label=""); # Turn off legends by default

Plots.default(ticks=:native);  # Ticks on x-axis are labelled properly when zooming in.
```

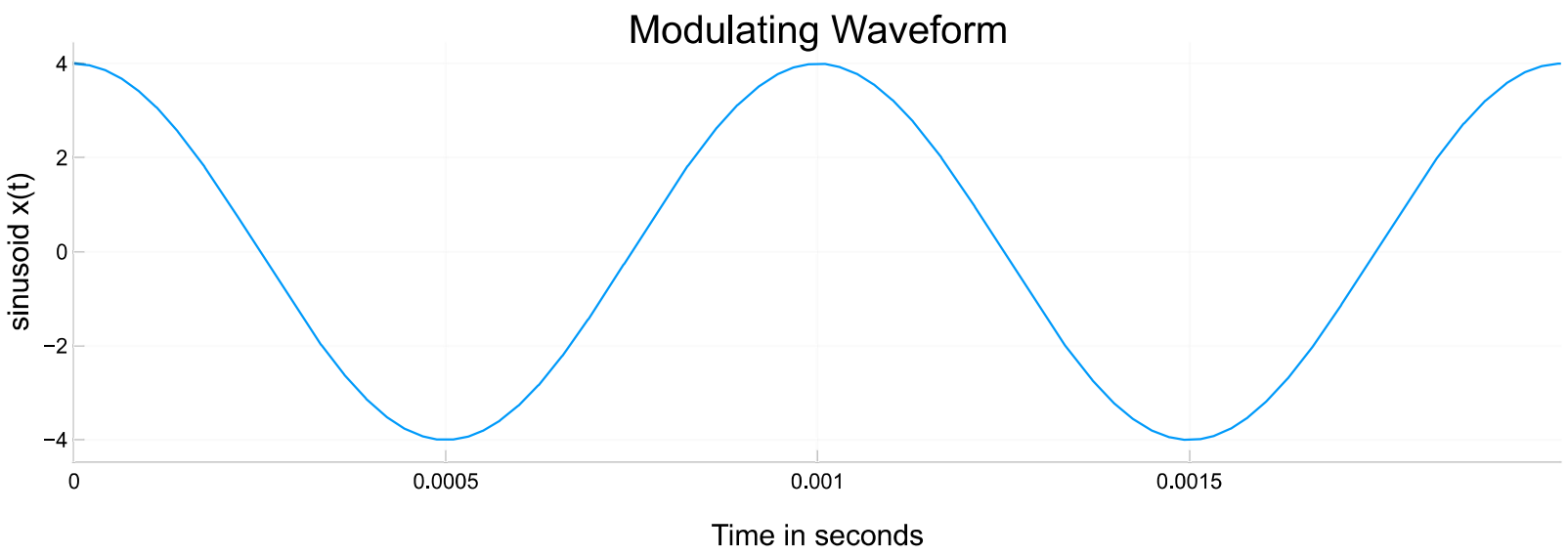
```
Δt = 0.000001   # time step  (to get Greek sybmol, type \Delta <tab>)
t = 0:Δt:0.002;  #Define time from 0 to 1s in steps of 0.01s

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



#Setting up modulating Waveform

fm = 1000 # 10 Hz
ωm = 2*pi*fm;   # rad/s   ( Greek symbol \omega <tab>  )
A = 4

x = A*cos.(ωm*t);   # Create an array holding the sinusoid values



fig = Plots.plot(t,x)
xlabel!("Time in seconds");
ylabel!("sinusoid x(t)")
title!("Modulating Waveform")
display(fig)

X = fft(x)

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

fig = Plots.plot( f_axis, fftshift(abs.(X)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(x)")
display(fig);

M = 10
```

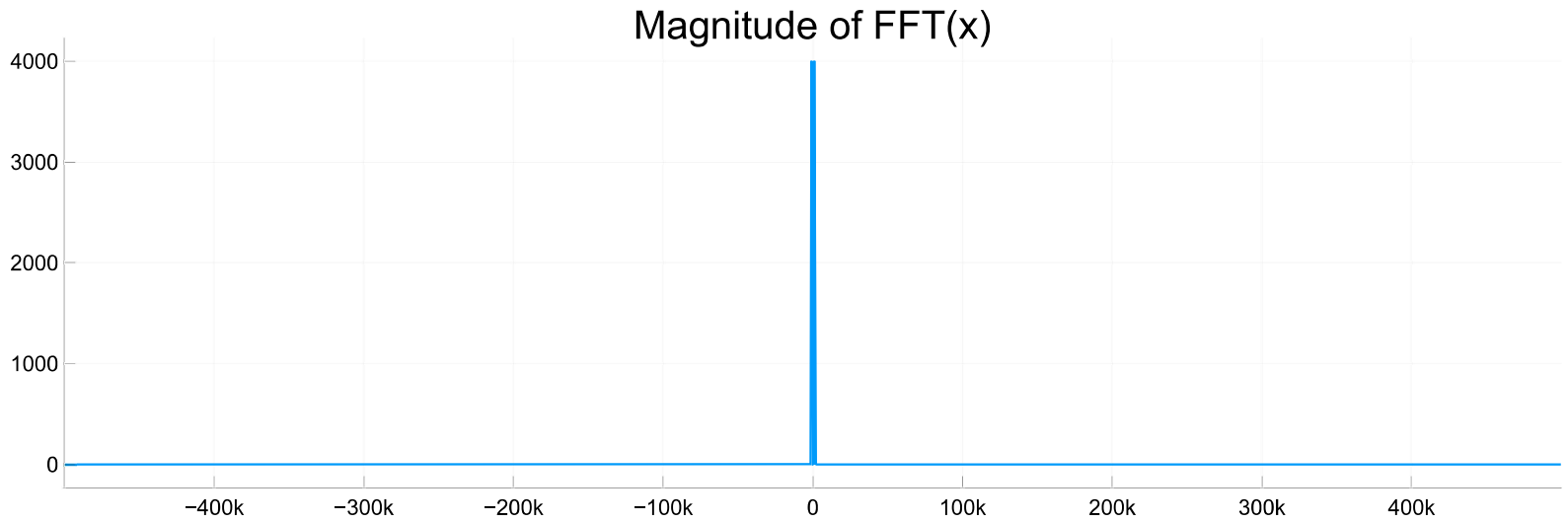
```
x = x .+ M




#Setting up Carrier Waveform

fc = 20000  # 10 Hz
ωc = 2*pi*fc;   # rad/s   ( Greek symbol \omega <tab>  )
A = 1

v = A*cos.(ωc*t) ;   # Create an array holding the sinusoid values


fig = Plots.plot(t,v)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("Carrier Waveform")
display(fig)

V = fft(v)

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

fig = Plots.plot( f_axis, fftshift(abs.(V)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(v)")
display(fig);









#Setting up modulated Waveform

z= x.*v

fig = Plots.plot(t,z)
```

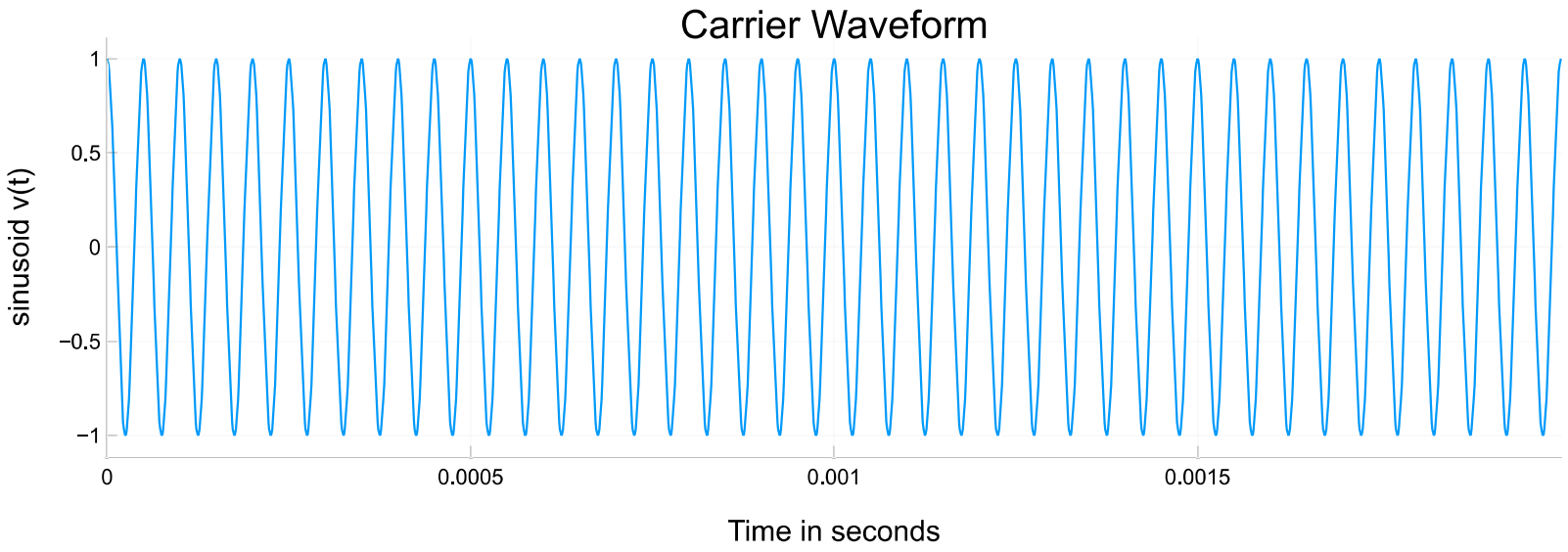
```
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Z = fft(z)

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

fig = Plots.plot( f_axis, fftshift(abs.(Z)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);









#Setting up demodulated Waveform (Before BPF)

y = z.*v

fig = Plots.plot(t,y)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Y = fft(y)

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

fig = Plots.plot( f_axis, fftshift(abs.(Y)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);

y_var = var(y)
```

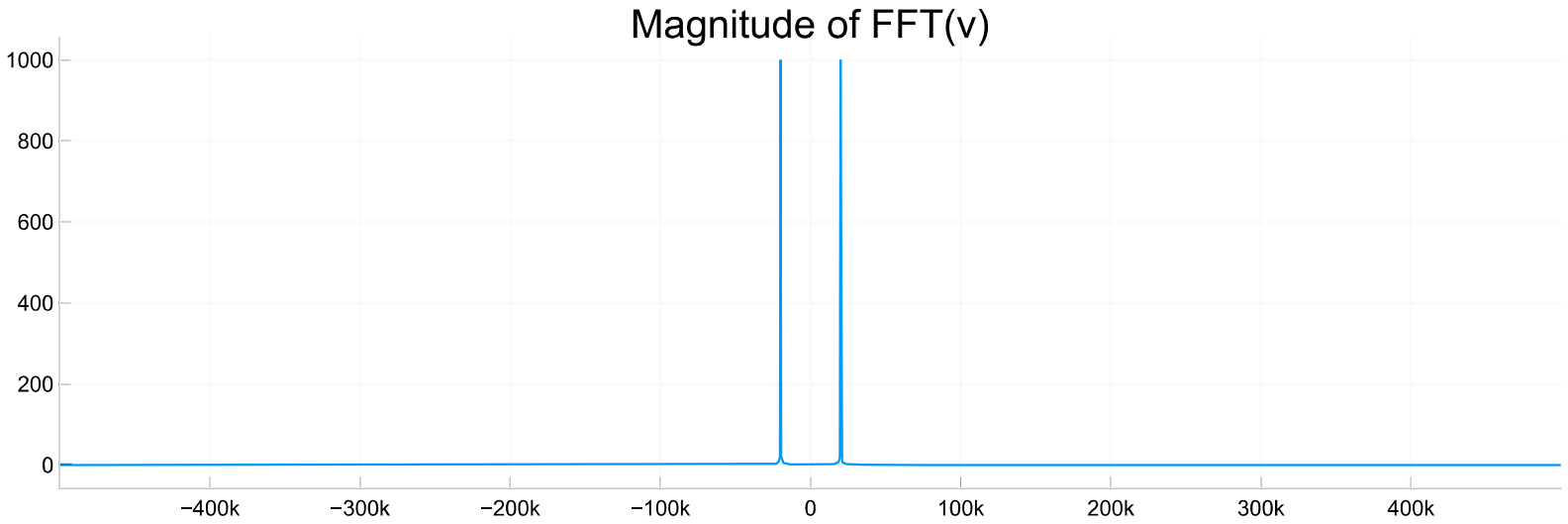
```
#Setting up demodulating Waveform (After BPF)


y = y.*(y.>0)
Y = fft(y)


# creating a BPF
# Very compact one-line definition of a rect function. Value is 0 or 1.
rect(t) = (abs.(t).<=0.5)*1.0

Δω = 2*pi/(N*Δt)   # Sample spacing in freq domain in rad/s

ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 1000/2*π # filter bandwidth in Hz

b = 10000

H = rect((ω .- b)/(2*π*B)) + rect( (ω .+ (b .- 2*π/Δt) )/(2*π*B) )


#H = fft(h);

D = Y.*H

d = ifft(D)
d = real(d)

fig = Plots.plot(t, d)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)

fig = Plots.plot( f_axis, fftshift(abs.(D)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(d)")
display(fig);
```

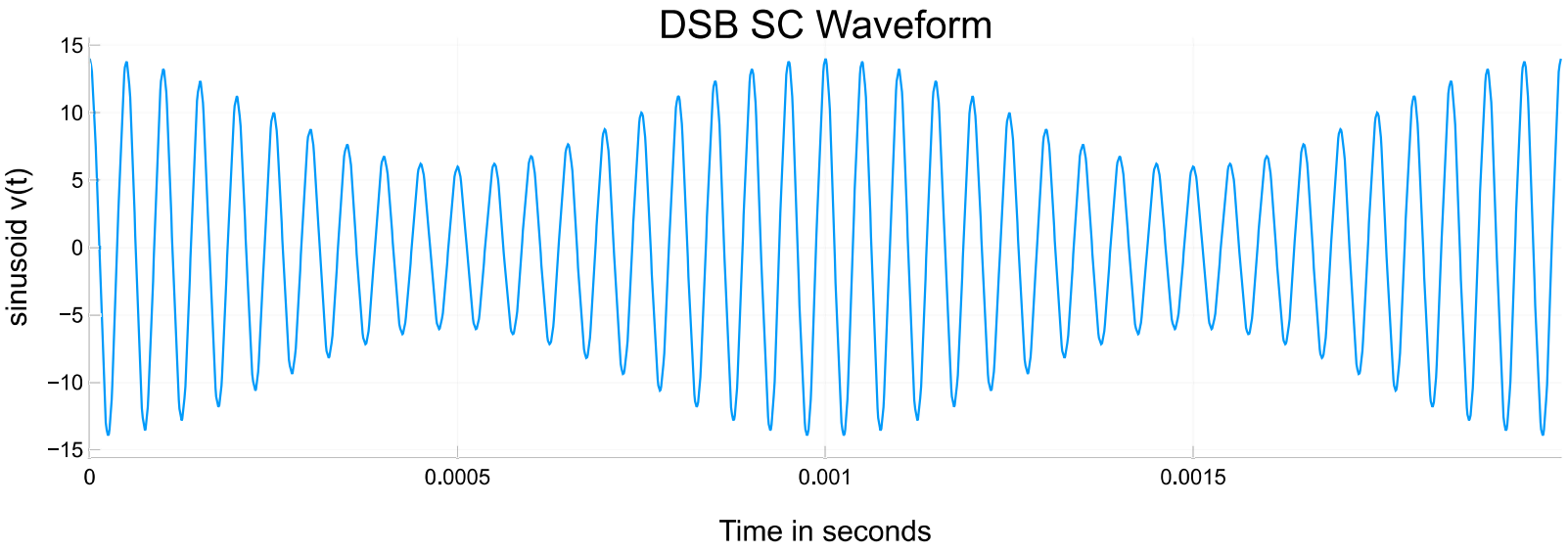
```
d_var = var(d)
```

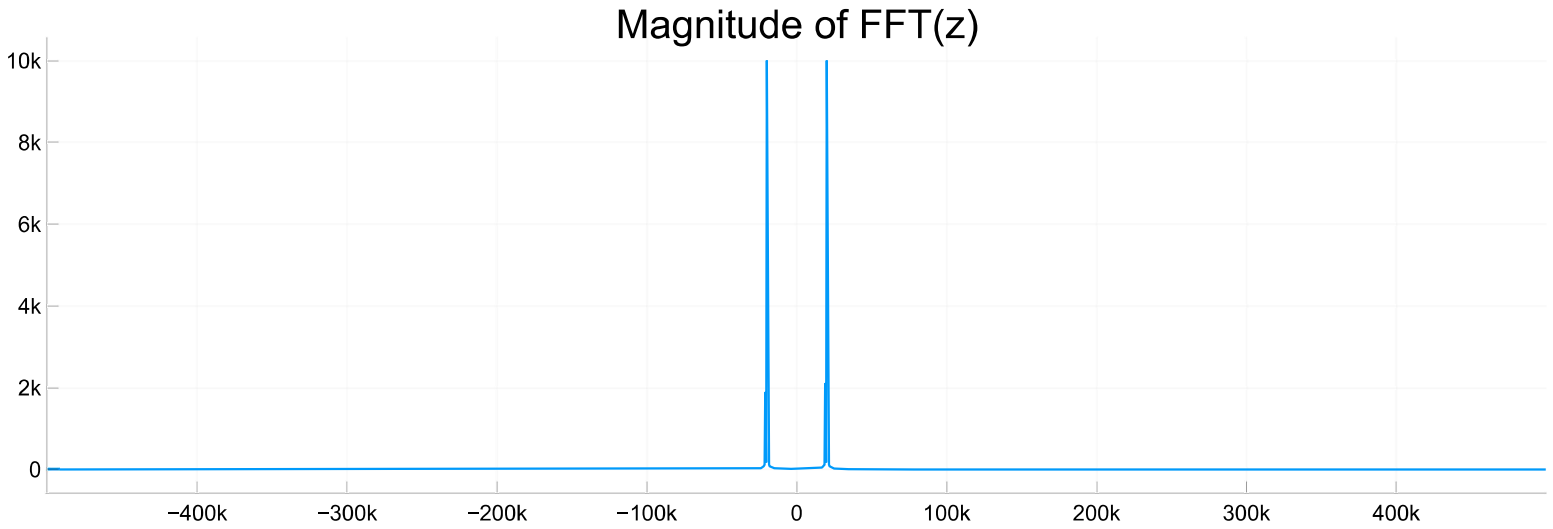

Modulating Waveform
sinusoid x(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds

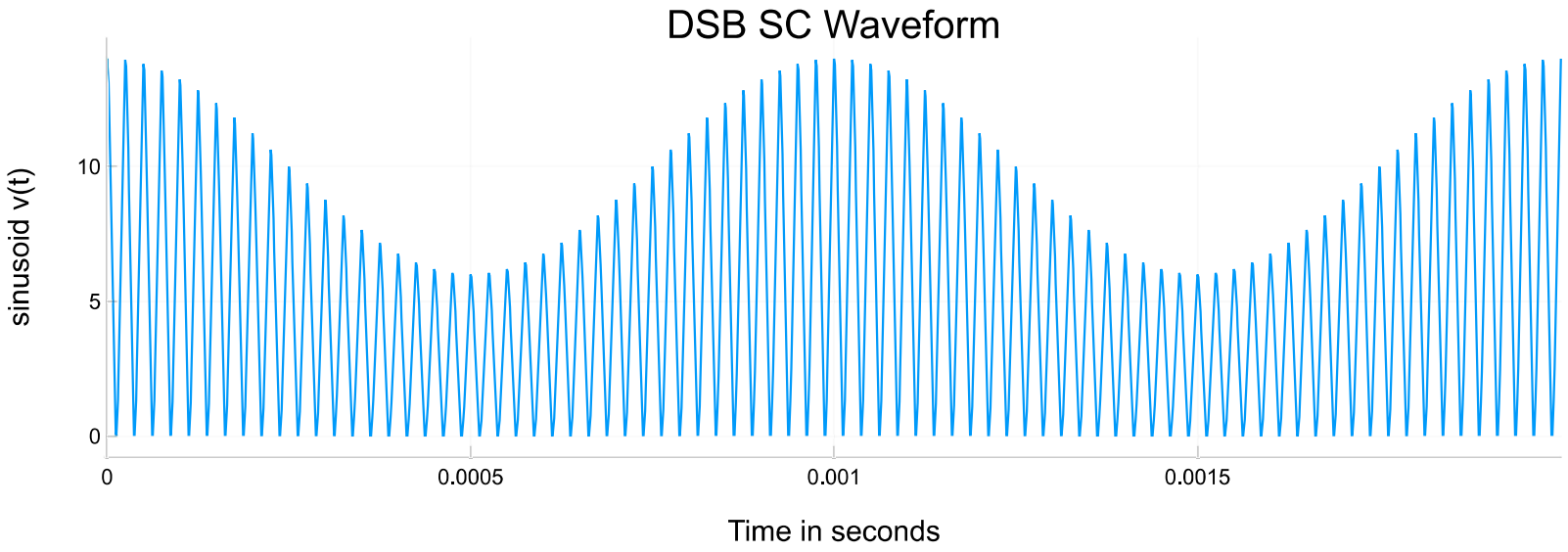

Magnitude of FFT(x)
4000
3000
2000
1000
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

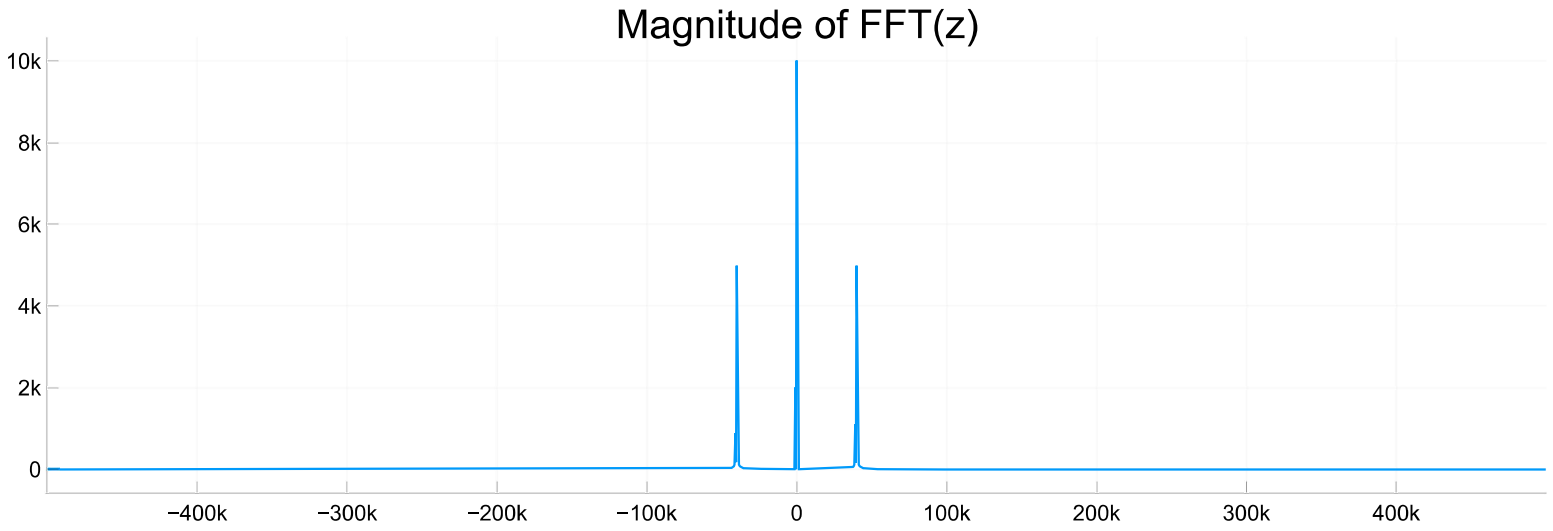

Carrier Waveform
sinusoid v(t)
1
0.5
0
−0.5
−1
0
0.0005
0.001
0.0015
Time in seconds

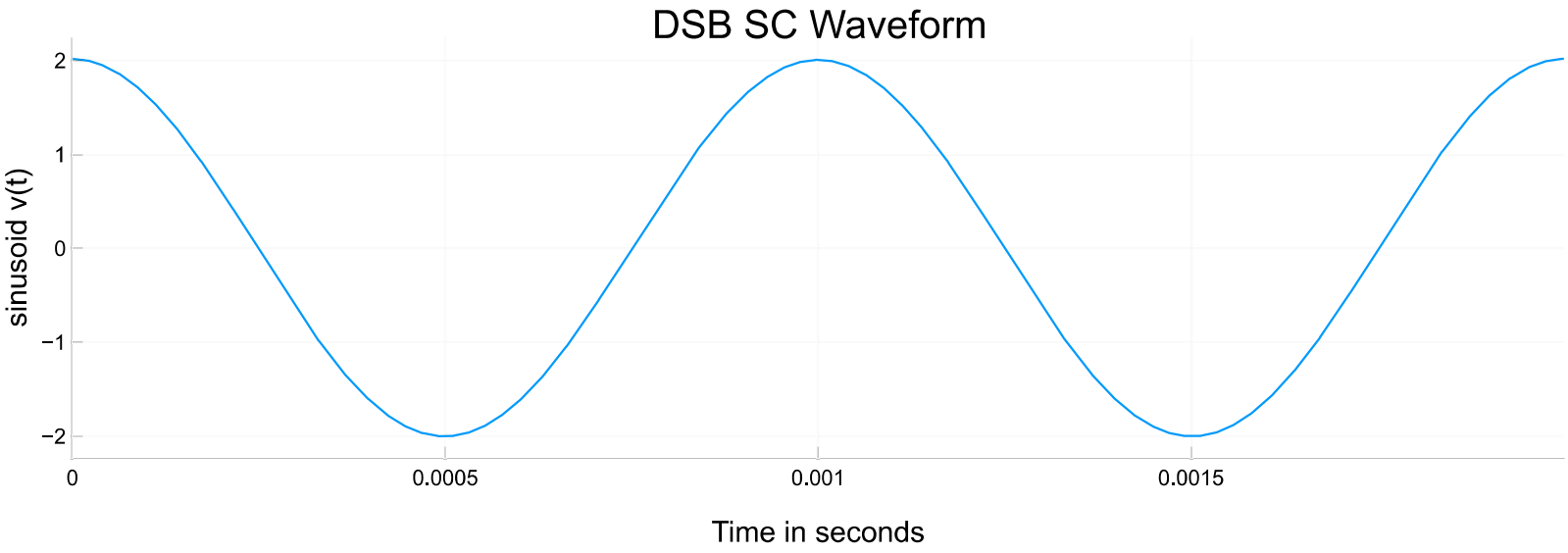

Magnitude of FFT(v)
1000
800
600
400
200
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

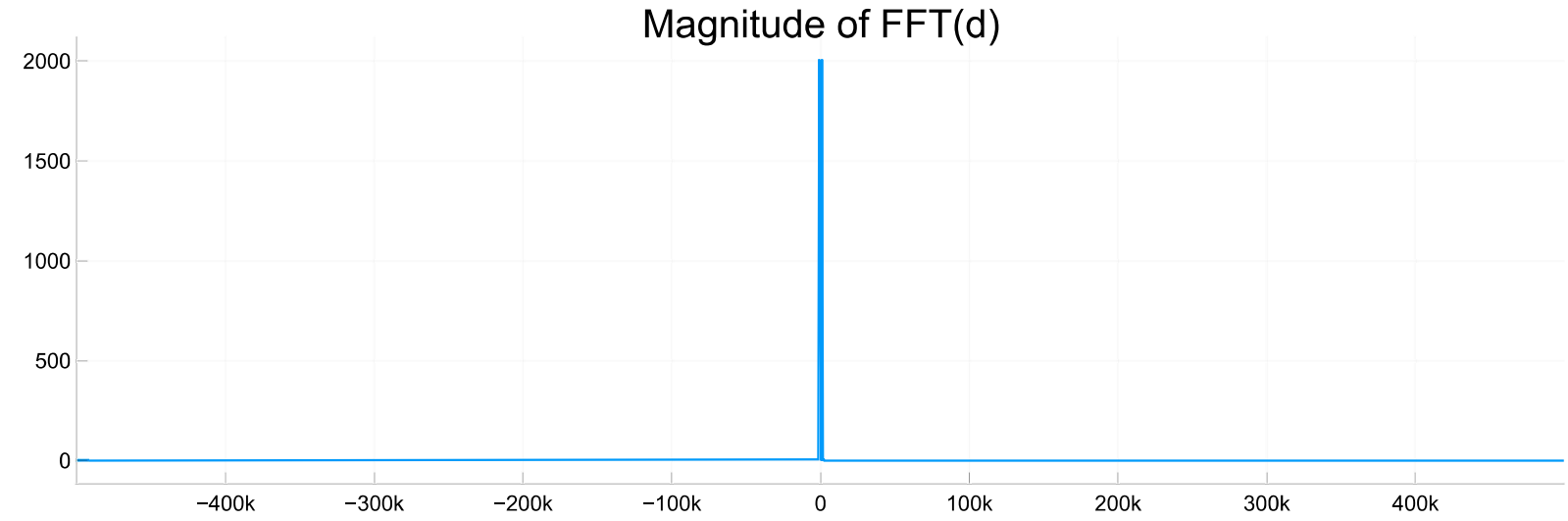

DSB SC Waveform
sinusoid v(t)
15
10
5
0
−5
−10
−15
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
10k
8k
6k
4k
2k
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
10
5
0
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
10k
8k
6k
4k
2k
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k





Out[126]: 2.0160652813641486

DSB-LC with a low level of noise

In [124…

```
using FFTW
using Plots

Plots.plotly();    # Specify Plotly backend which allows zooming with a mouse in Jupyter Notebook.

Plots.default(size=(800,300)); # Set default plot canvas size

Plots.default(label=""); # Turn off legends by default

Plots.default(ticks=:native);  # Ticks on x-axis are labelled properly when zooming in.




Δt = 0.000001   # time step  (to get Greek sybmol, type \Delta <tab>)
t = 0:Δt:0.002;  #Define time from 0 to 1s in steps of 0.01s

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



#Setting up modulating Waveform

fm = 1000 # 10 Hz
ωm = 2*pi*fm;   # rad/s   ( Greek symbol \omega <tab>  )
A = 4

x = A*cos.(ωm*t);   # Create an array holding the sinusoid values



fig = Plots.plot(t,x)
xlabel!("Time in seconds");
ylabel!("sinusoid x(t)")
```

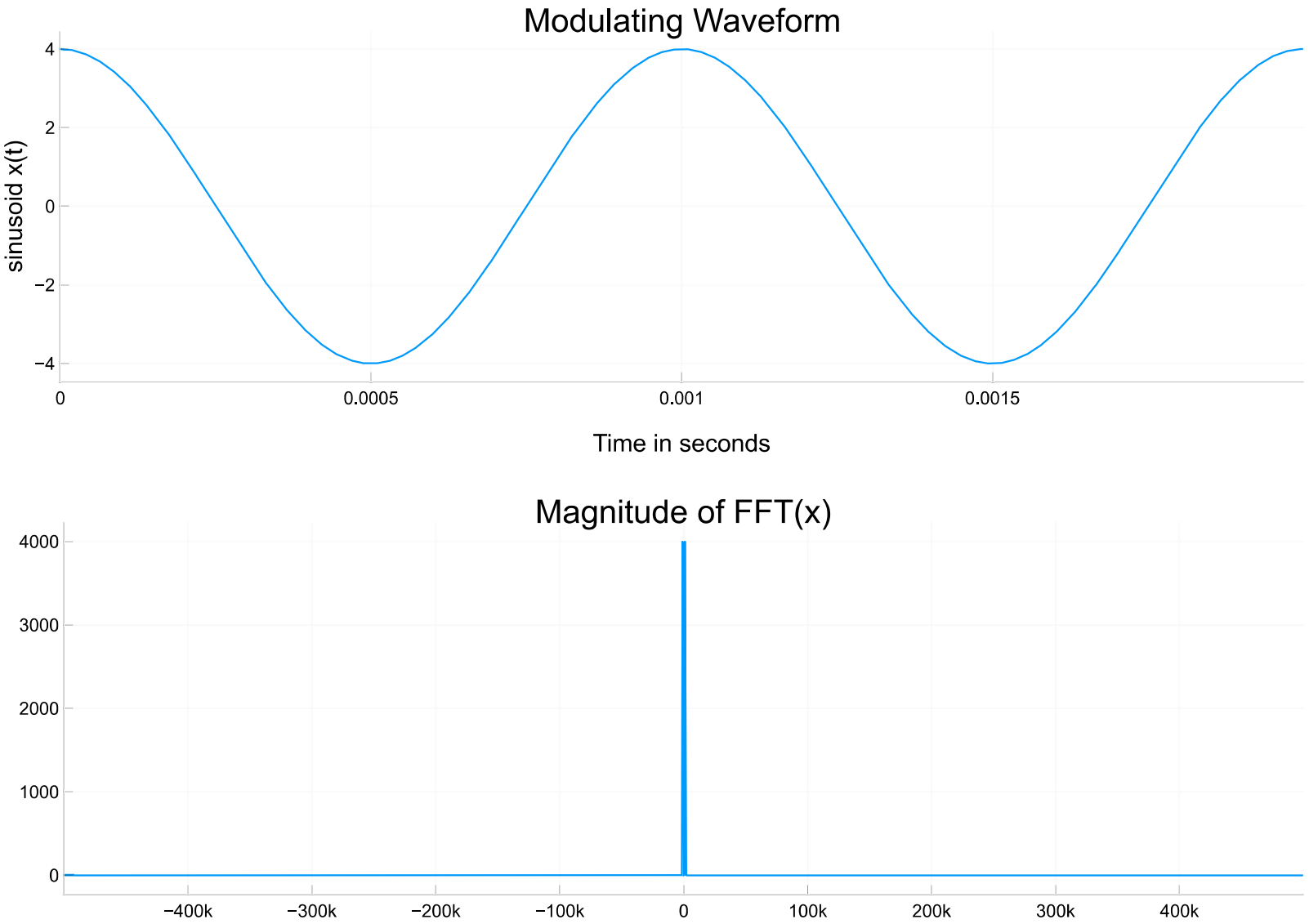
```
title!("Modulating Waveform")
display(fig)

X = fft(x)

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

fig = Plots.plot( f_axis, fftshift(abs.(X)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(x)")
display(fig);

M = 10

x = x .+ M




#Setting up Carrier Waveform

fc = 20000  # 10 Hz
ωc = 2*pi*fc;   # rad/s   ( Greek symbol \omega <tab>  )
A = 1

v = A*cos.(ωc*t) ;   # Create an array holding the sinusoid values


fig = Plots.plot(t,v)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("Carrier Waveform")
display(fig)

V = fft(v)

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

fig = Plots.plot( f_axis, fftshift(abs.(V)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(v)")
display(fig);
```

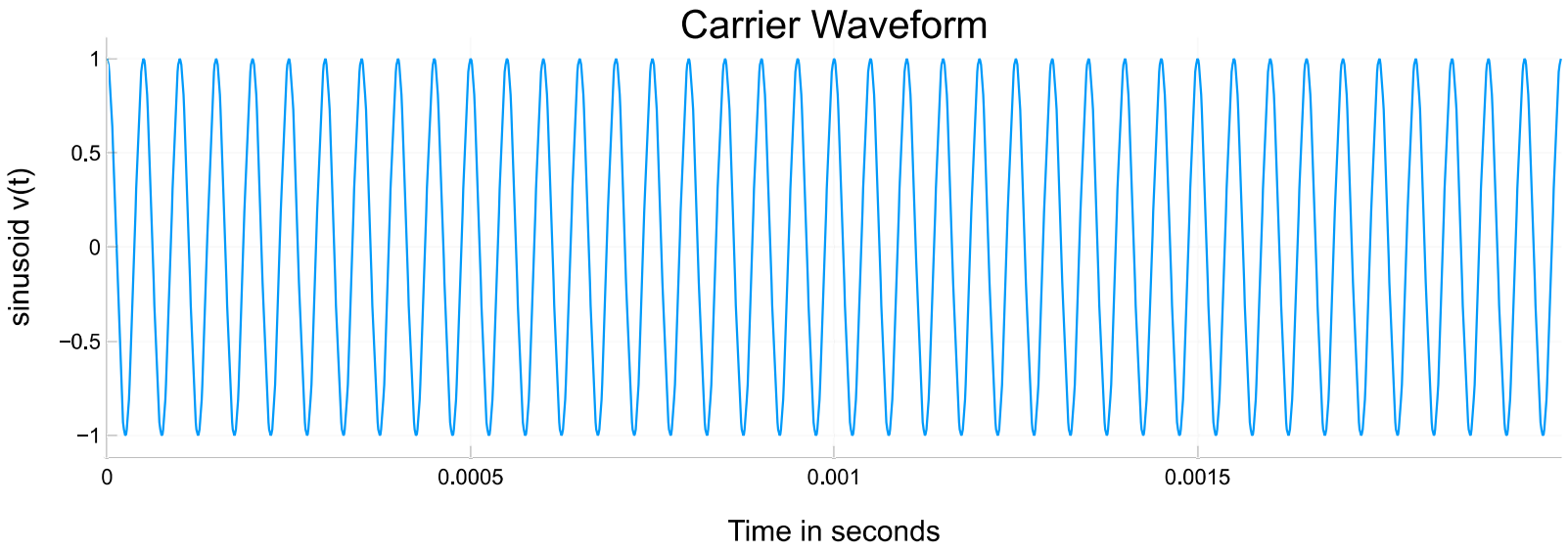
```
# Now generate some noise

# Very compact one-line definition of a rect function. Value is 0 or 1.
rect(t) = (abs.(t).<=0.5)*1.0

Δω = 2*pi/(N*Δt)   # Sample spacing in freq domain in rad/s

ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 6000 # filter bandwidth in Hz

H = rect((ω .- ωc)/(2*π*B)) + rect( (ω .+ (ωc .- 2*π/Δt) )/(2*π*B) )


N = length(t);

noise = randn(N);  # Create an array of N zero-mean Gaussian random number of std dev = 1.

μ = 0.0     # desired mean
σ = 1 # desired standard deviation  NOTE: to create Greek symbol, \sigma<tab>
noise = noise*σ .+ μ


Ns = fft(noise)
Ns = real(Ns)

Noise = H.*Ns

noise = ifft(Noise)
noise = real(noise)


# Calculate statistics:  mean and standard deviation
using Statistics  #The Statistics module contains basic statistics functionality (mean, std, var etc.)

noise_var = var(noise)
```

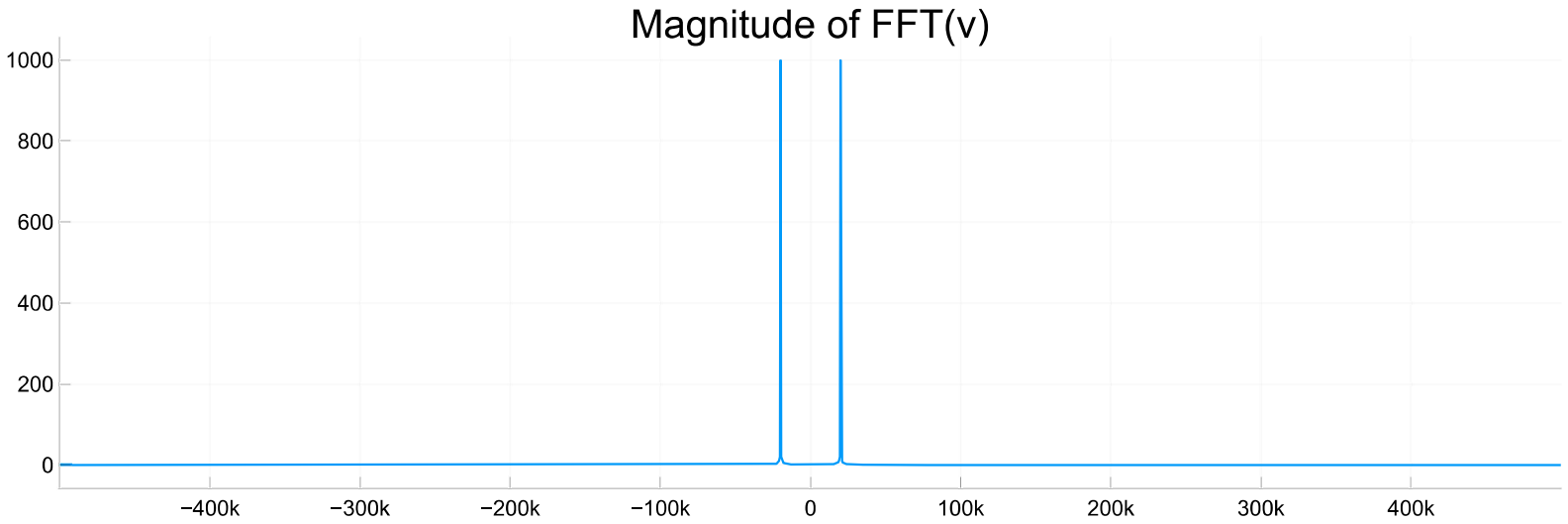
```
noise_std = std(noise)


println("Noise Statistics:")
println("Variance: $(noise_var)")
println("Standard Deviation: $(noise_std)")









#Setting up modulated Waveform

z= x.*v + noise

fig = Plots.plot(t,z)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Z = fft(z)

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

fig = Plots.plot( f_axis, fftshift(abs.(Z)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);









#Setting up demodulated Waveform (Before BPF)
```

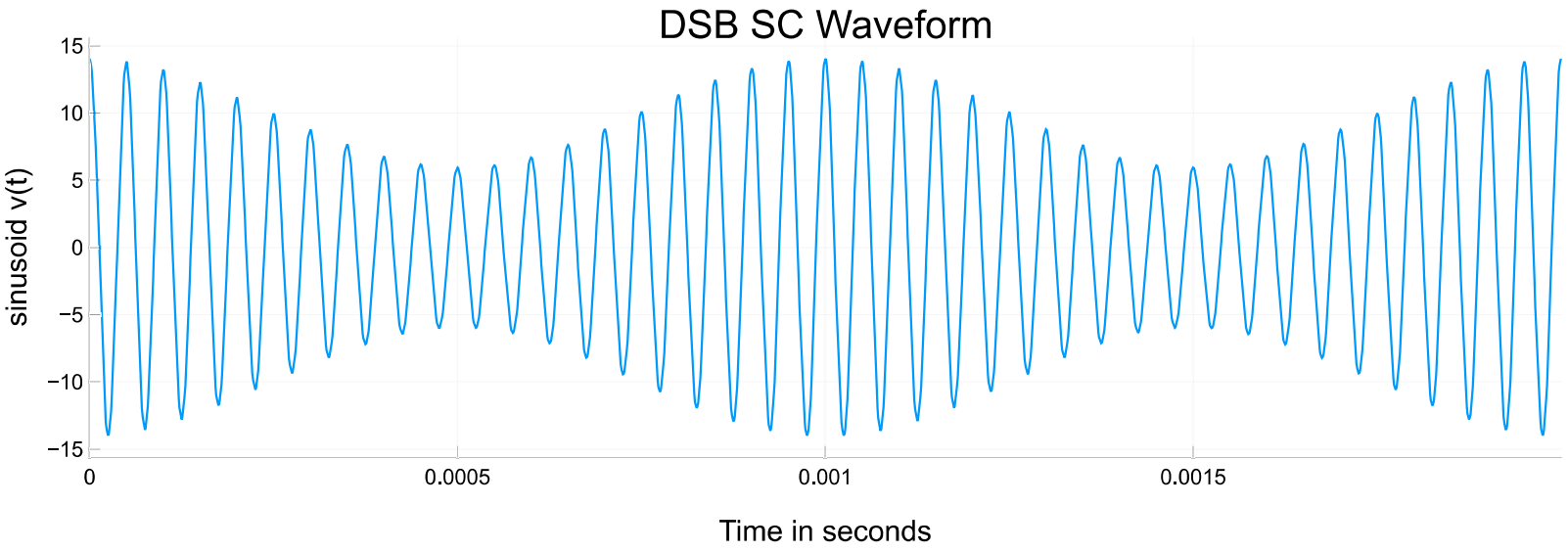
```
y = z.*v

fig = Plots.plot(t,y)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Y = fft(y)

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

fig = Plots.plot( f_axis, fftshift(abs.(Y)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);

y_var = var(y)

SNR_before = 10*log(y_var/noise_var)


println()
println("SNR before BPF = $(SNR_before) dB")









#Setting up demodulating Waveform (After BPF)


y = y.*(y.>0)
Y = fft(y)


# creating a BPF
# Very compact one-line definition of a rect function. Value is 0 or 1.
rect(t) = (abs.(t).<=0.5)*1.0

Δω = 2*pi/(N*Δt)   # Sample spacing in freq domain in rad/s
```

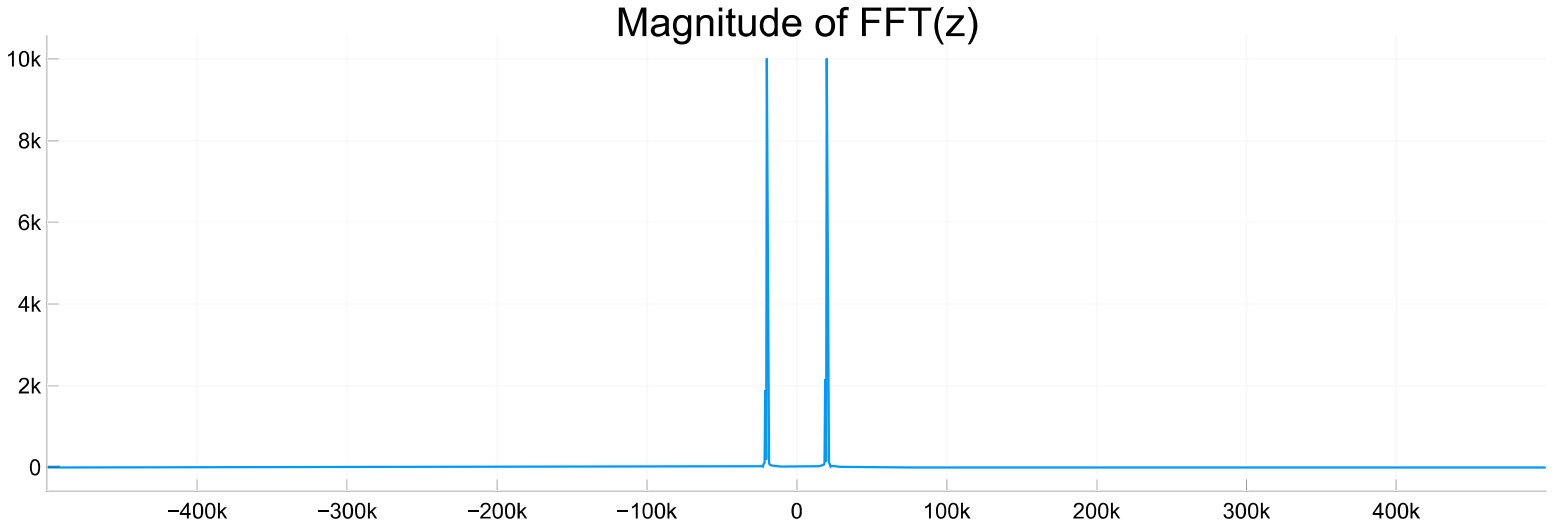
```
ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 1000/2*π # filter bandwidth in Hz

b = 10000

H = rect((ω .- b)/(2*π*B)) + rect( (ω .+ (b .- 2*π/Δt) )/(2*π*B) )


#H = fft(h);

D = Y.*H

d = ifft(D)
d = real(d)

fig = Plots.plot(t, d)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)

fig = Plots.plot( f_axis, fftshift(abs.(D)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(d)")
display(fig);

d_var = var(d)

SNR_after = 10*log(d_var/noise_var)

println("SNR after LPF = $(SNR_after) dB")

SNR_factor = SNR_after/SNR_before

println()
println("The SNR has decreased by a factor of $(SNR_factor)")
```

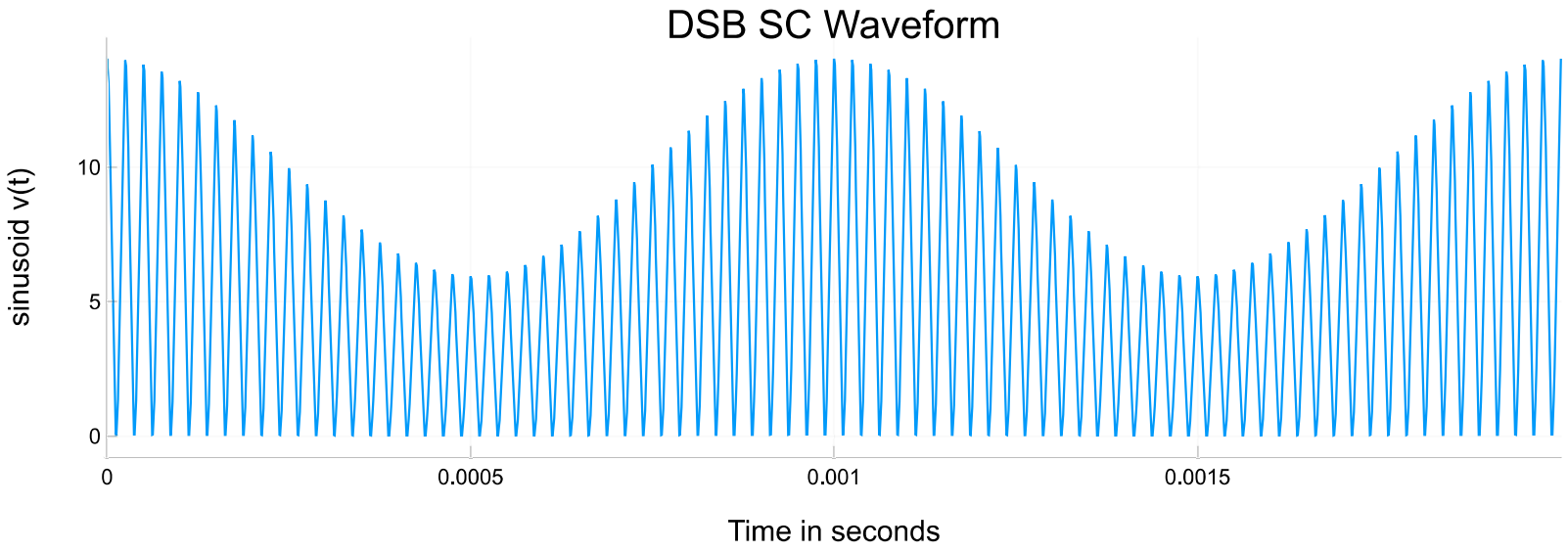

Modulating Waveform
sinusoid x(t)
4
2
0
−2
−4
0
0.0005
0.001
0.0015
Time in seconds
Magnitude of FFT(x)
4000
3000
2000
1000
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

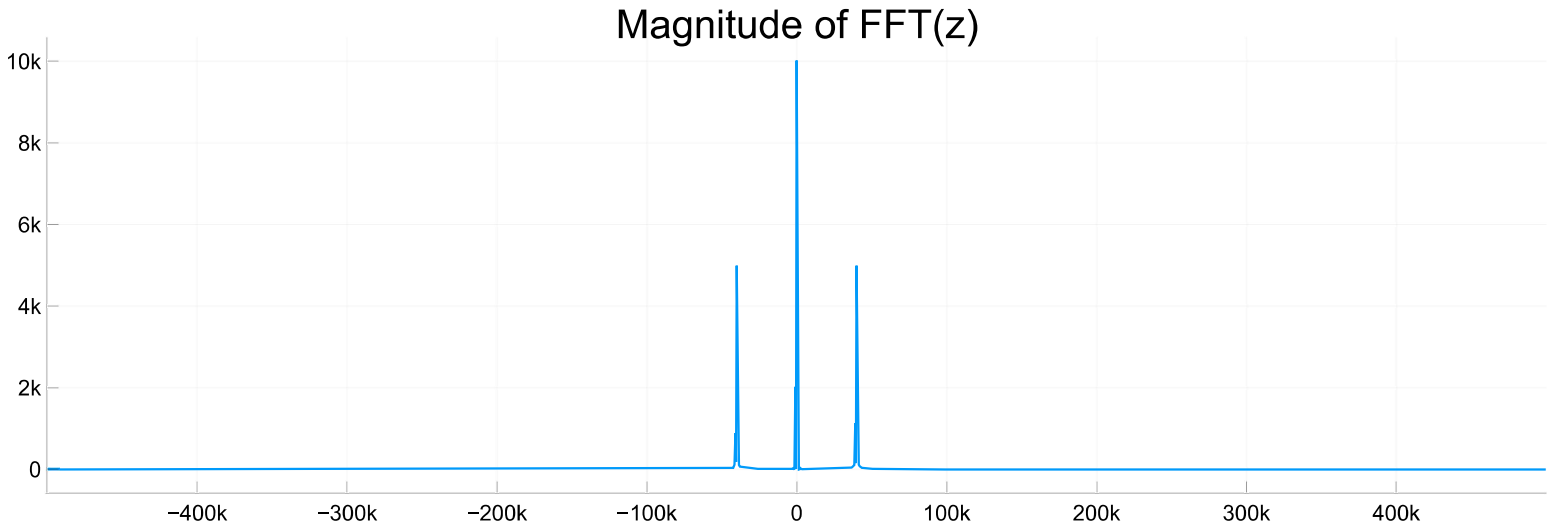

Carrier Waveform
sinusoid v(t)
1
0.5
0
−0.5
−1
0
0.0005
0.001
0.0015
Time in seconds

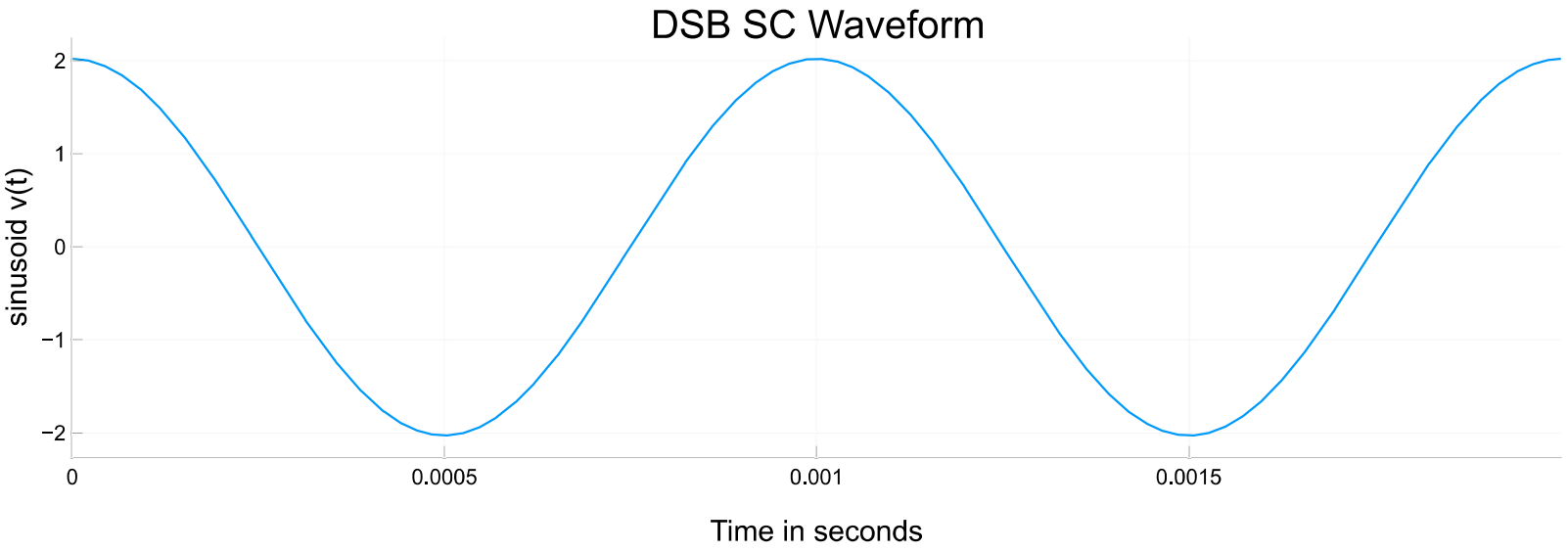

Magnitude of FFT(v)
1000
800
600
400
200
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

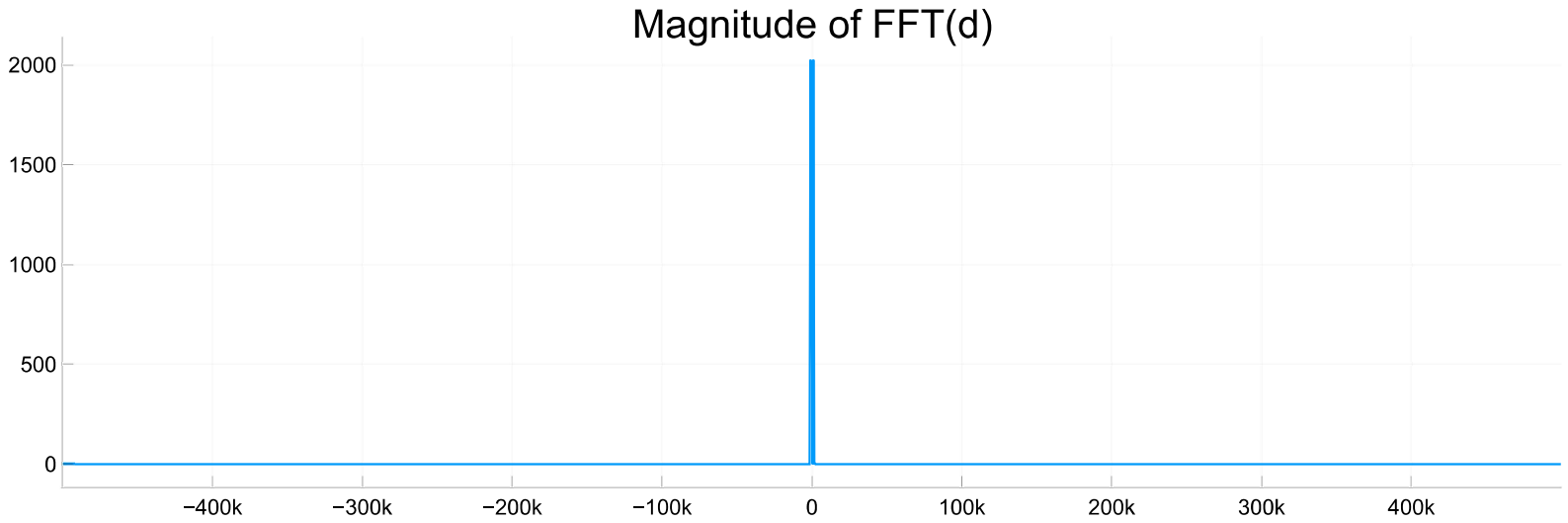

DSB SC Waveform
sinusoid v(t)
15
10
5
0
−5
−10
−15
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
10k
8k
6k
4k
2k
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
10
5
0
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
10k
8k
6k
4k
2k
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
2
1
0
−1
−2
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(d)
2000
1500
1000
500
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

```
Noise Statistics:
Variance: 0.002350671358070221
Standard Deviation: 0.048483722609451316

SNR before BPF = 88.00943139235486 dB
SNR after LPF = 67.70935305777046 dB

The SNR has decreased by a factor of 0.769342012402232
```

DSB-LC with a high level of noise

In [125…

```
using FFTW
using Plots

Plots.plotly();    # Specify Plotly backend which allows zooming with a mouse in Jupyter Notebook.

Plots.default(size=(800,300)); # Set default plot canvas size

Plots.default(label=""); # Turn off legends by default

Plots.default(ticks=:native);  # Ticks on x-axis are labelled properly when zooming in.




Δt = 0.000001   # time step  (to get Greek sybmol, type \Delta <tab>)
t = 0:Δt:0.002;  #Define time from 0 to 1s in steps of 0.01s

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
```

```
#Setting up modulating Waveform

fm = 1000 # 10 Hz
ωm = 2*pi*fm;   # rad/s   ( Greek symbol \omega <tab>  )
A = 4

x = A*cos.(ωm*t);   # Create an array holding the sinusoid values



fig = Plots.plot(t,x)
xlabel!("Time in seconds");
ylabel!("sinusoid x(t)")
title!("Modulating Waveform")
display(fig)

X = fft(x)

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

fig = Plots.plot( f_axis, fftshift(abs.(X)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(x)")
display(fig);

M = 10

x = x .+ M




#Setting up Carrier Waveform

fc = 20000  # 10 Hz
ωc = 2*pi*fc;   # rad/s   ( Greek symbol \omega <tab>  )
A = 1

v = A*cos.(ωc*t) ;   # Create an array holding the sinusoid values


fig = Plots.plot(t,v)
xlabel!("Time in seconds");
```

```
ylabel!("sinusoid v(t)")
title!("Carrier Waveform")
display(fig)

V = fft(v)

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

fig = Plots.plot( f_axis, fftshift(abs.(V)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(v)")
display(fig);









# Now generate some noise

# Very compact one-line definition of a rect function. Value is 0 or 1.
rect(t) = (abs.(t).<=0.5)*1.0

Δω = 2*pi/(N*Δt)   # Sample spacing in freq domain in rad/s

ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 6000 # filter bandwidth in Hz

H = rect((ω .- ωc)/(2*π*B)) + rect( (ω .+ (ωc .- 2*π/Δt) )/(2*π*B) )


N = length(t);

noise = randn(N);  # Create an array of N zero-mean Gaussian random number of std dev = 1.

μ = 0.0     # desired mean
σ = 5 # desired standard deviation  NOTE: to create Greek symbol, \sigma<tab>
noise = noise*σ .+ μ
```

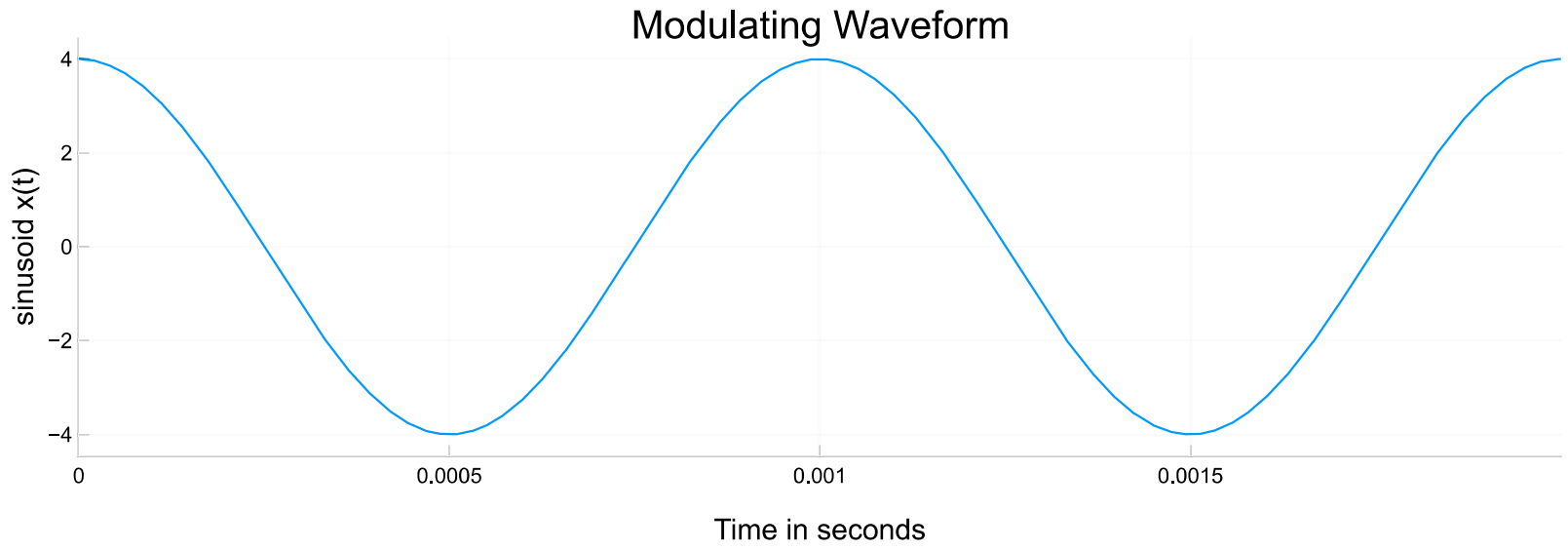
```
Ns = fft(noise)
Ns = real(Ns)

Noise = H.*Ns

noise = ifft(Noise)
noise = real(noise)


# Calculate statistics:  mean and standard deviation
using Statistics  #The Statistics module contains basic statistics functionality (mean, std, var etc.)

noise_var = var(noise)
noise_std = std(noise)


println("Noise Statistics:")
println("Variance: $(noise_var)")
println("Standard Deviation: $(noise_std)")









#Setting up modulated Waveform

z= x.*v + noise

fig = Plots.plot(t,z)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Z = fft(z)

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

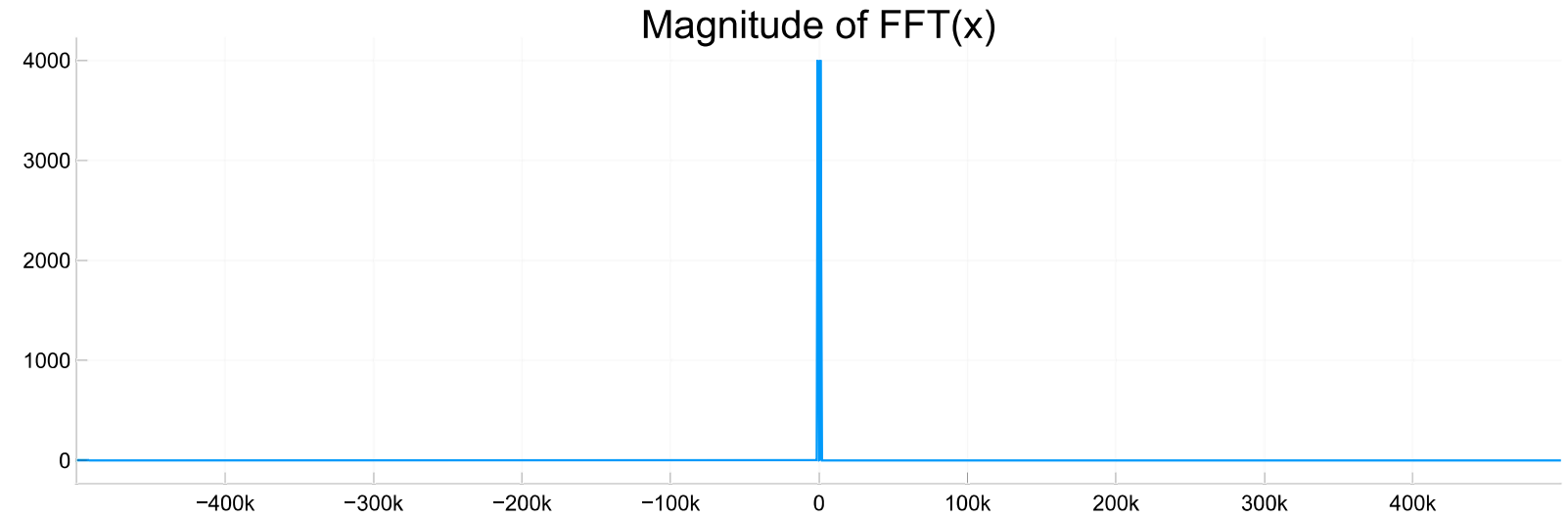
```
fig = Plots.plot( f_axis, fftshift(abs.(Z)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);
```

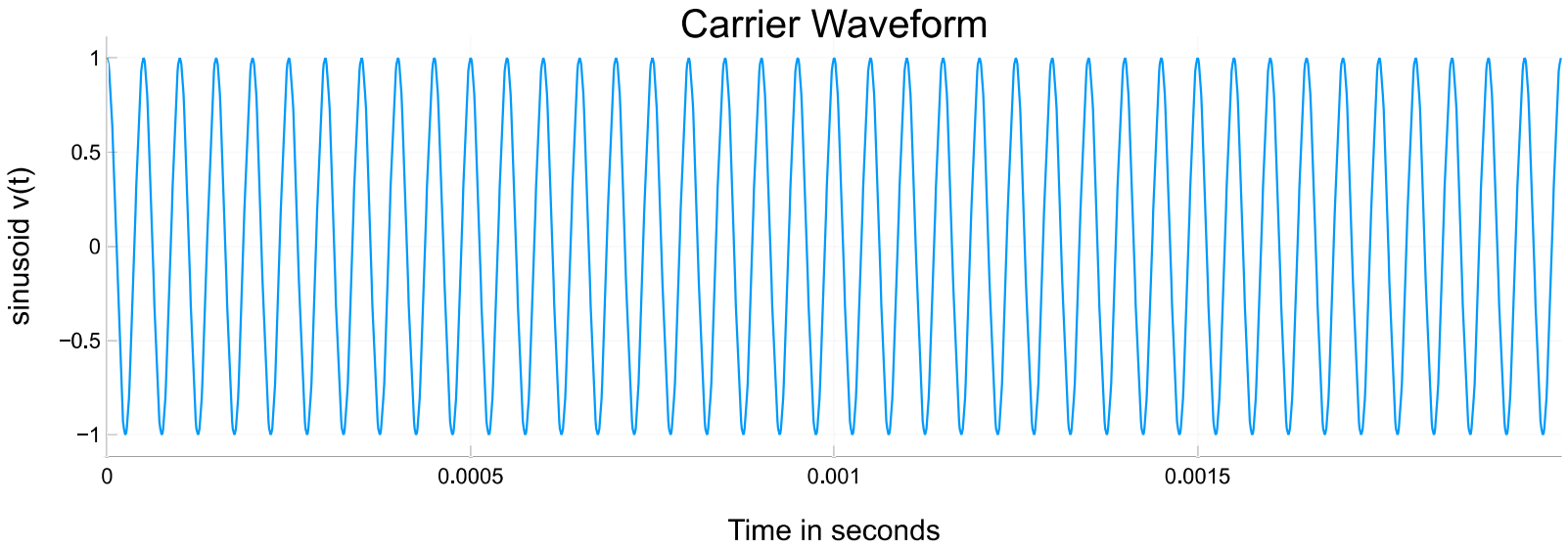
```
#Setting up demodulated Waveform (Before BPF)

y = z.*v

fig = Plots.plot(t,y)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)


Y = fft(y)

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

fig = Plots.plot( f_axis, fftshift(abs.(Y)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/2
title!("Magnitude of FFT(z)")
display(fig);

y_var = var(y)

SNR_before = 10*log(y_var/noise_var)


println()
println("SNR before BPF = $(SNR_before) dB")
```

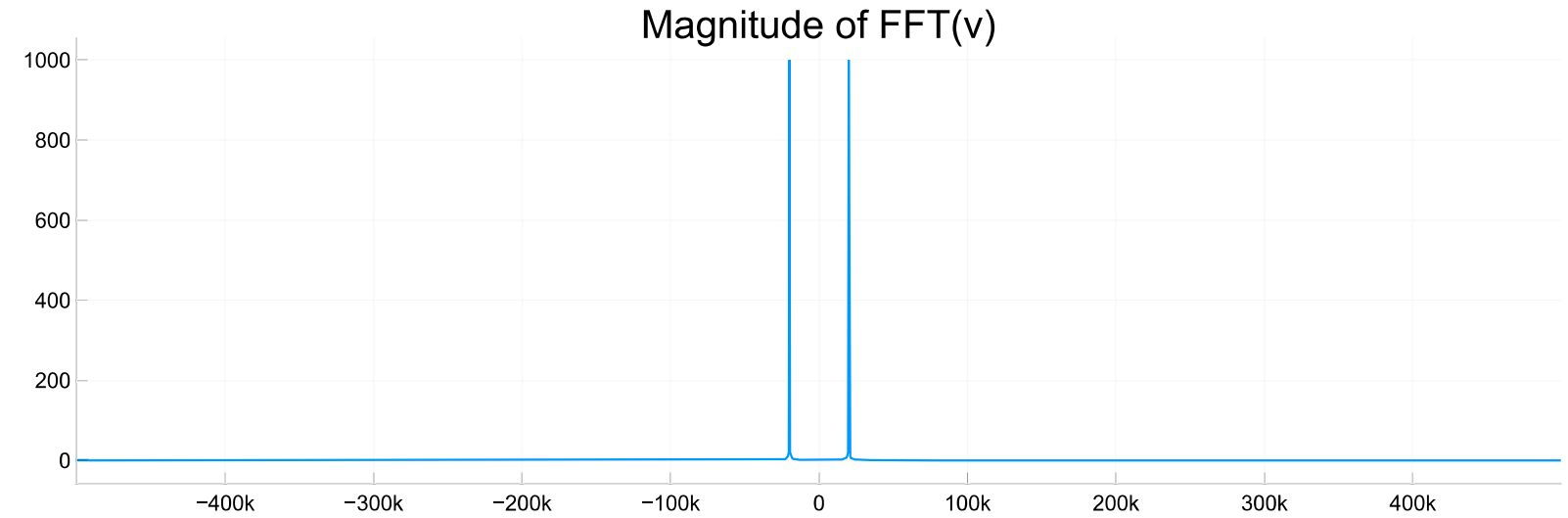
```
#Setting up demodulating Waveform (After BPF)


y = y.*(y.>0)
Y = fft(y)


# creating a BPF
# Very compact one-line definition of a rect function. Value is 0 or 1.
rect(t) = (abs.(t).<=0.5)*1.0

Δω = 2*pi/(N*Δt)   # Sample spacing in freq domain in rad/s

ω = 0:Δω:(N-1)*Δω
f = ω/(2*π)
B = 1000/2*π # filter bandwidth in Hz

b = 10000

H = rect((ω .- b)/(2*π*B)) + rect( (ω .+ (b .- 2*π/Δt) )/(2*π*B) )


#H = fft(h);

D = Y.*H

d = ifft(D)
d = real(d)

fig = Plots.plot(t, d)
xlabel!("Time in seconds");
ylabel!("sinusoid v(t)")
title!("DSB SC Waveform")
display(fig)

fig = Plots.plot( f_axis, fftshift(abs.(D)), markersize = 2, markercolor=:lightblue, markerstrokecolor=:blue)   # Note elts 1:N/
title!("Magnitude of FFT(d)")
display(fig);

d_var = var(d)

SNR_after = 10*log(d_var/noise_var)
```

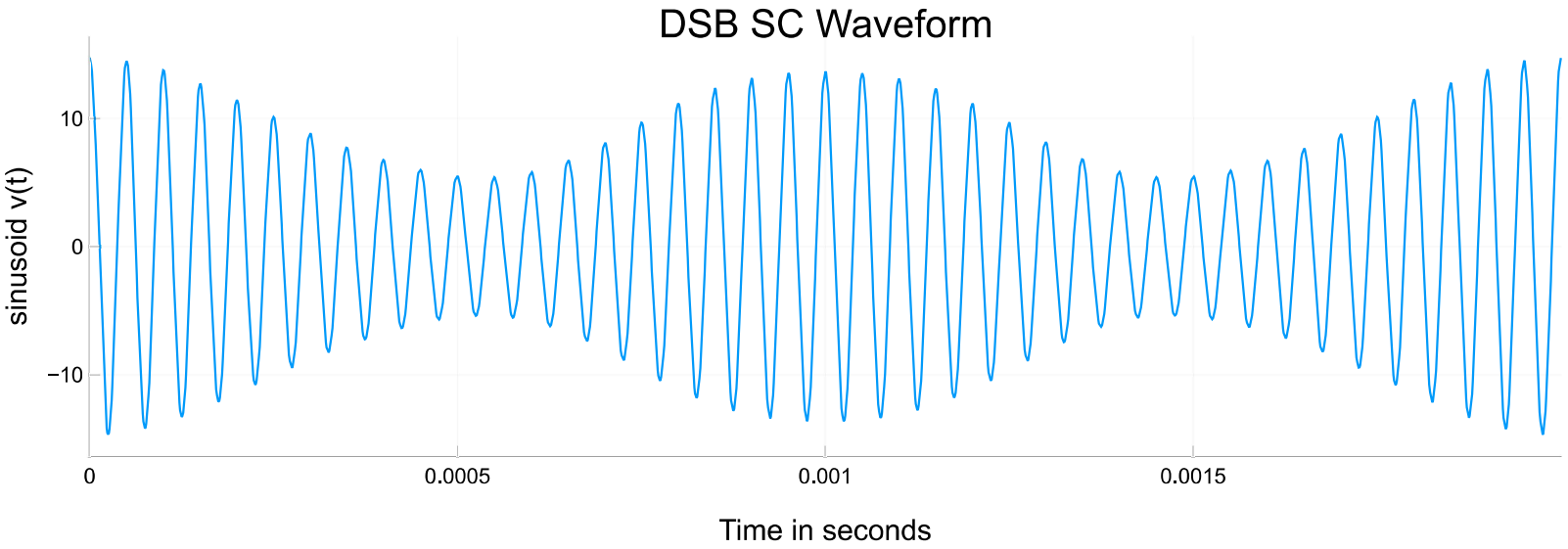
```
println("SNR after LPF = $(SNR_after) dB")

SNR_factor = SNR_after/SNR_before

println()
println("The SNR has decreased by a factor of $(SNR_factor)")
```

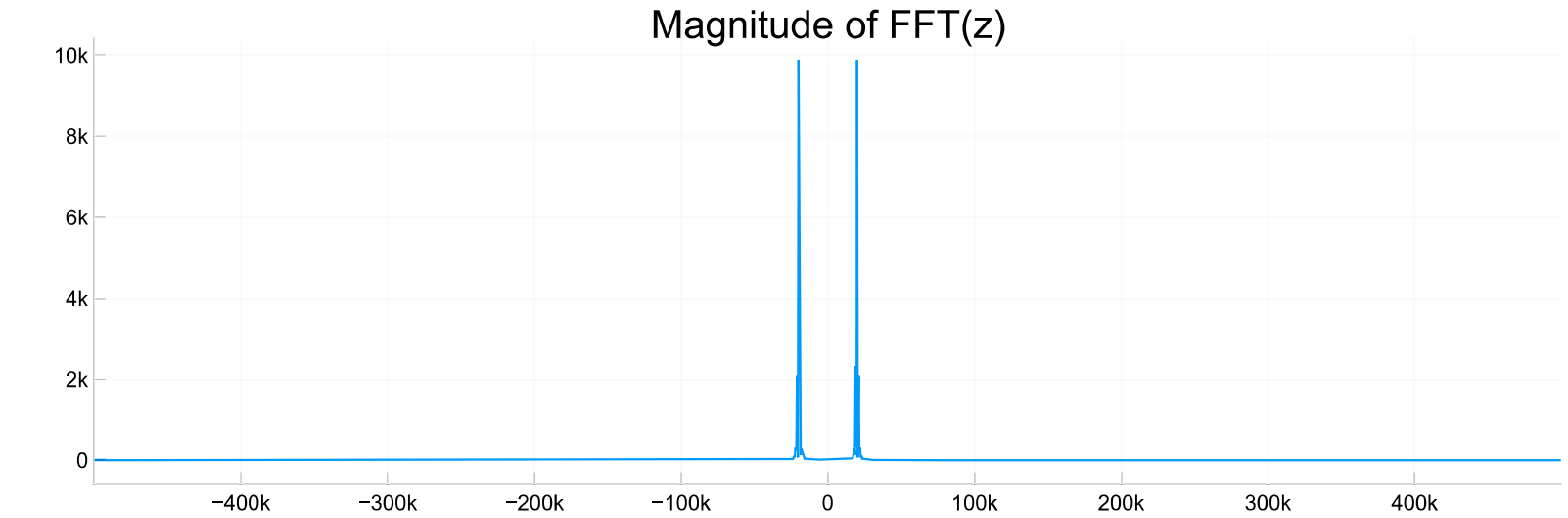


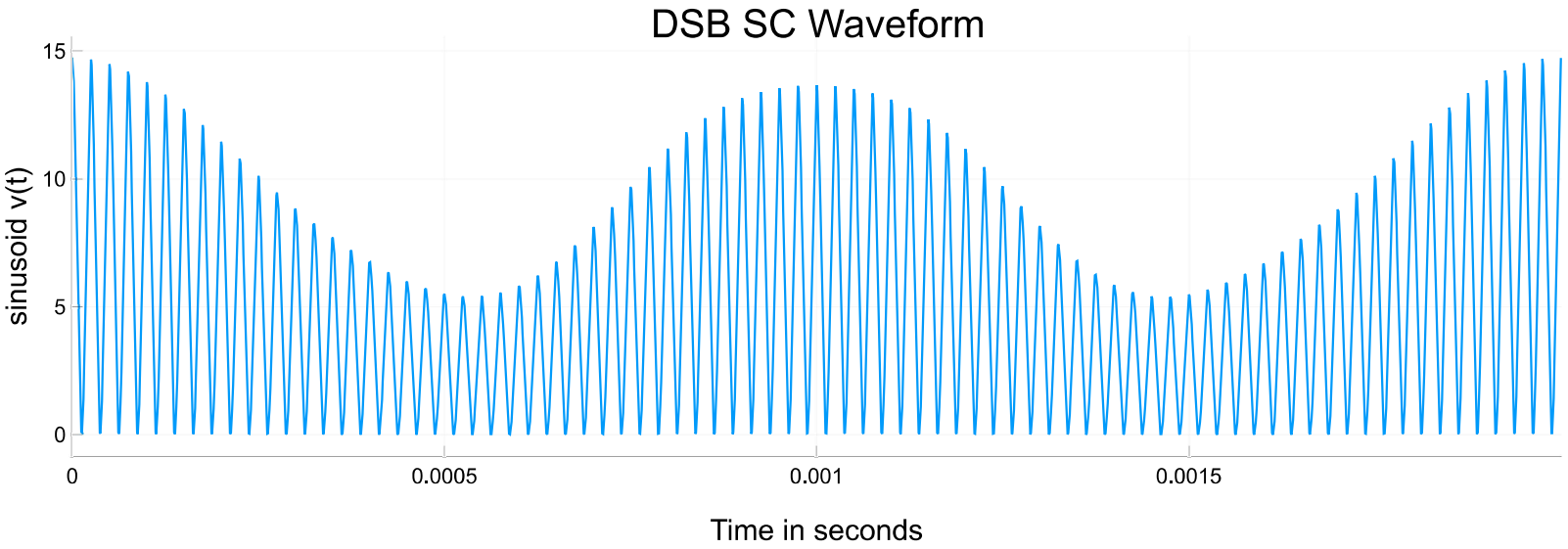

Magnitude of FFT(x)
4000
3000
2000
1000
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

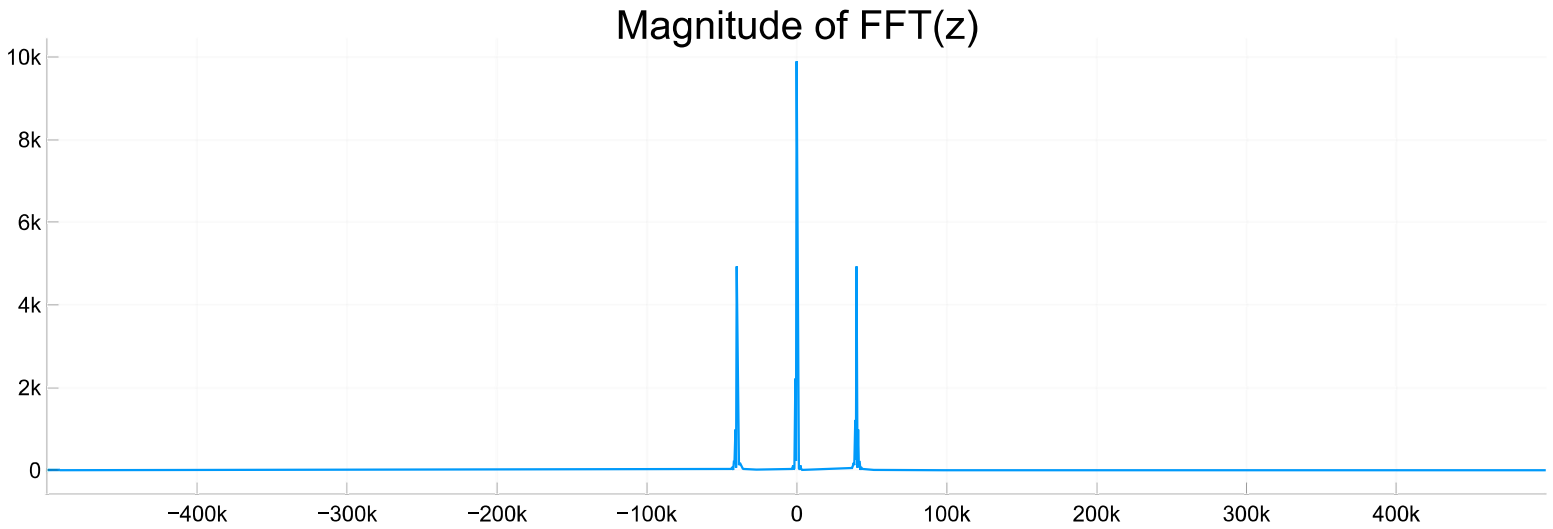

Carrier Waveform
sinusoid v(t)
1
0.5
0
−0.5
−1
0
0.0005
0.001
0.0015
Time in seconds

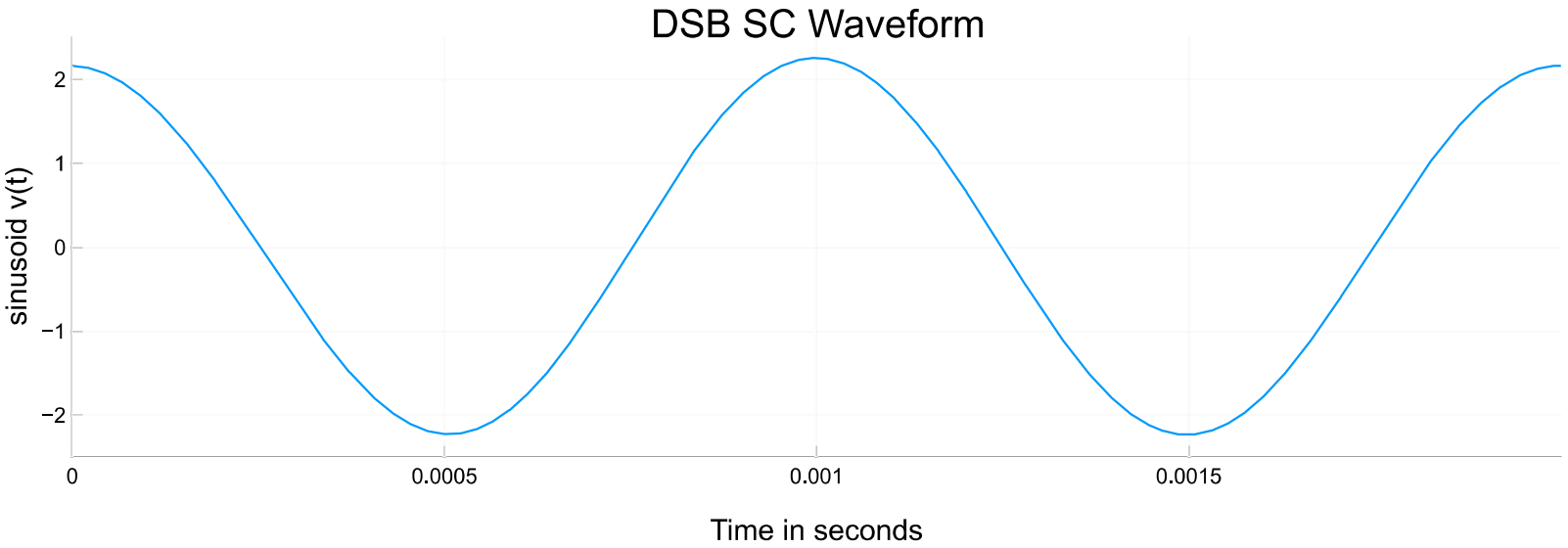

Magnitude of FFT(v)
1000
800
600
400
200
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k

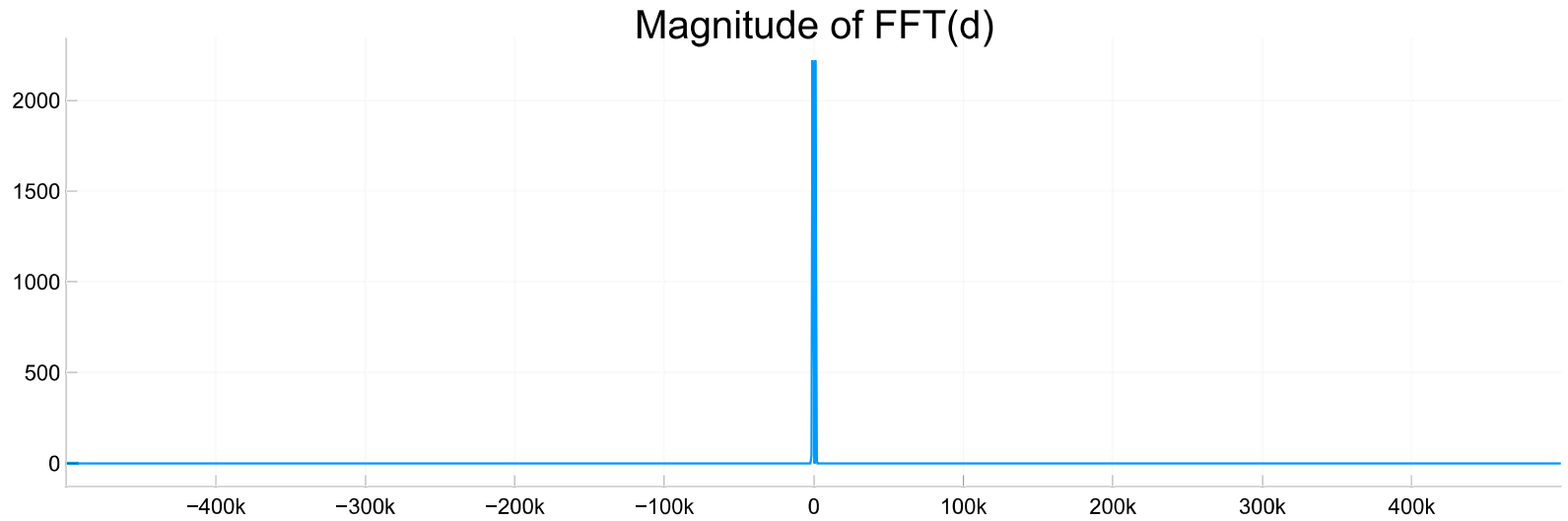

DSB SC Waveform
sinusoid v(t)
10
0
−10
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
10k
8k
6k
4k
2k
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
15
10
5
0
sinusoid v(t)
0
0.0005
0.001
0.0015
Time in seconds


Magnitude of FFT(z)
10k
8k
6k
4k
2k
0
−400k
−300k
−200k
−100k
0
100k
200k
300k
400k


DSB SC Waveform
sinusoid v(t)
2
1
0
−1
−2
0
0.0005
0.001
0.0015
Time in seconds




```
Noise Statistics:
Variance: 0.1687229424478152
Standard Deviation: 0.4107589834048857

SNR before BPF = 45.5052058024829 dB
SNR after LPF = 26.815942117452618 dB

The SNR has decreased by a factor of 0.589293942188686
```

In [ ]: