EE495 HW 1

Problem 01-01 Aliasing with sinusoids

Kevin Morales

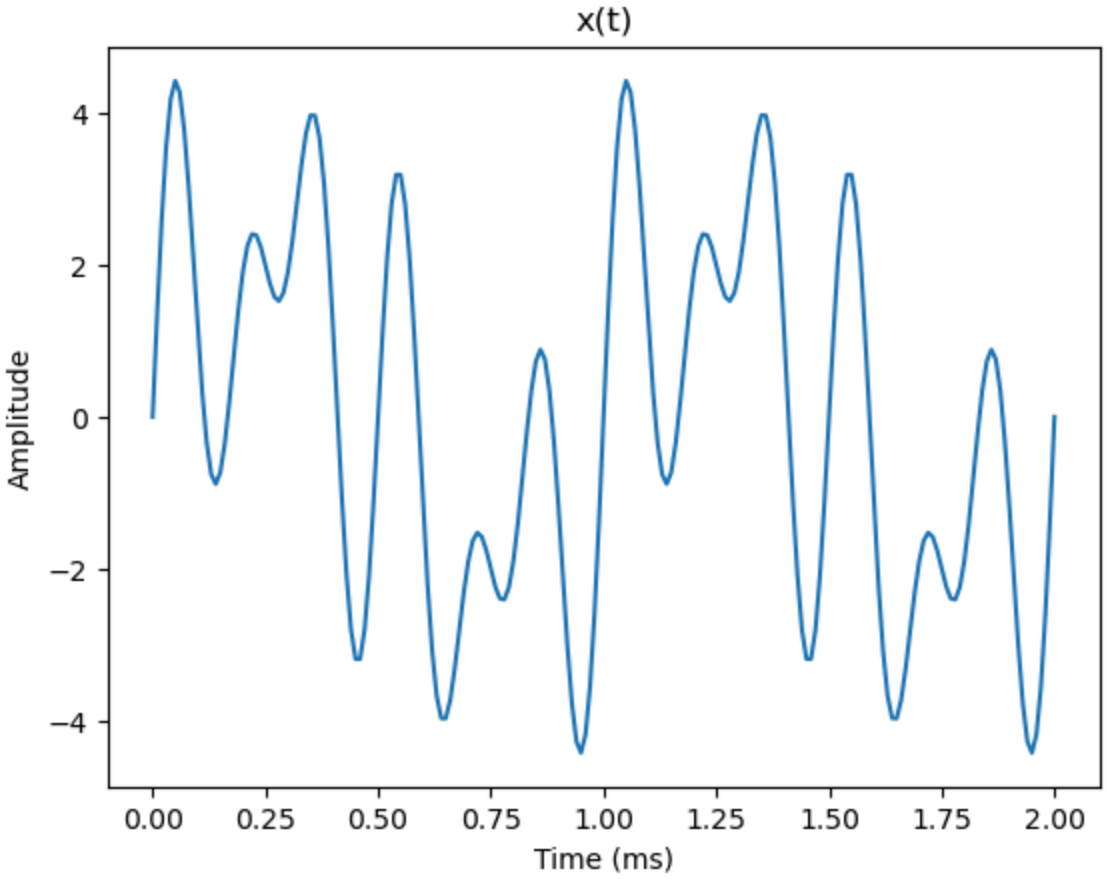
Part A

In part A, we're to plot a sinusoidal signal x(t), using numpy and matplotlib libraries.

In [225…
```
import matplotlib.pyplot as plt
import numpy as np
```

In [227…
```
tc = np.arange(0,2e-3+1e-5,1e-5)
f1 = 1000
f2 = 4000
f3 = 6000
x = 2*np.sin(2*np.pi*f1*tc) + 2*np.sin(2*np.pi*f2*tc) + 2*np.sin(2*np.pi*f3*tc)
plt.plot(tc * 1000,x)
plt.title("x(t)")
plt.ylabel("Amplitude")
plt.xlabel("Time (ms)")
```

Out[227…
```
Text(0.5, 0, 'Time (ms)')
```



Seen above is the expected output, a sinusoid, with 3 different frequencies all of which have an amplitude of 2.

Part B

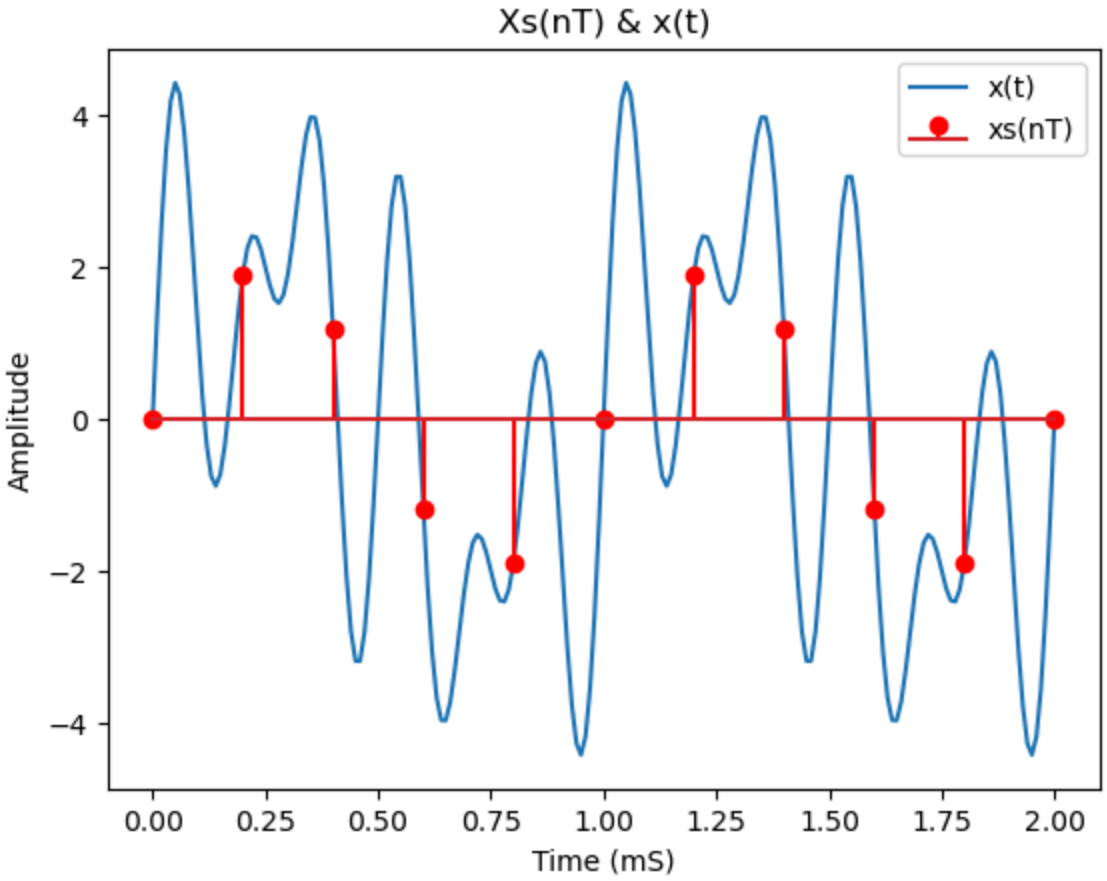
In part two we're tasked with sampling the sinusoidal signal x(t) generated in part A. This sampling will create a new discrete signal xs(nT).

In [232…

```
fs = 5000
Ts = 1 / fs
nT = np.arange(0,2e-3+Ts, Ts)
xs =  2*np.sin(2*np.pi*f1*nT) + 2*np.sin(2*np.pi*f2*nT) + 2*np.sin(2*np.pi*f3*nT)
plt.plot(tc *1000, x)
plt.stem(nT*1000, xs, "r")
plt.title("Xs(nT) & x(t)")
plt.ylabel("Amplitude")
plt.xlabel("Time (mS)")
plt.legend(['x(t)', 'xs(nT)'])
```

Out[232…

```
<matplotlib.legend.Legend at 0x23d3f776c90>
```



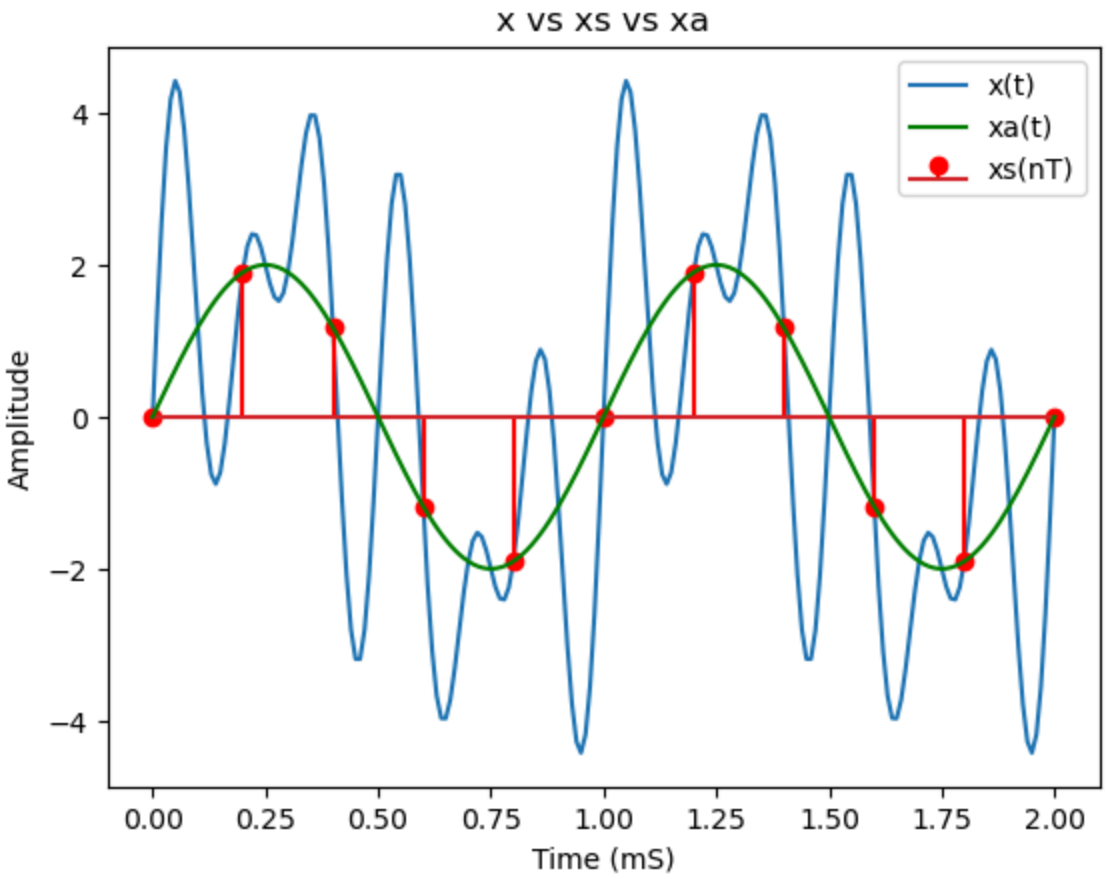
Seen above is the sampled signal xs(nT) overlayed with the original signal x(t). As you can see every point of xs corresponds with a point on x. This leads me to belive that the signal x(t) was sampled properly.

Part C

In this part, we're tasked with generating the signal which will result from aliasing. This signal will be called xa(t).

In [237…
```
fa1 = 1000
fa2 = -1000
fa3 = 1000

xa = 2*np.sin(2*np.pi*fa1*tc) + 2*np.sin(2*np.pi*fa2*tc) + 2*np.sin(2*np.pi*fa3*tc)
plt.plot(tc*1000, x)
plt.stem(nT*1000,xs, "r")
plt.plot(tc*1000, xa,"g")
plt.title("x vs xs vs xa")
plt.xlabel("Time (mS)")
plt.ylabel("Amplitude")
plt.legend(['x(t)', 'xa(t)', 'xs(nT)'])
```

Out[237…
```
<matplotlib.legend.Legend at 0x23d3f6bd3a0>
```



Seen above is the resulting waveform xa(t) overlayed onto the plot from part b. The resulting wave only has one frequency, 1000 Hz. Although the signal xa(t) looks nothing like our original signal x(t). The sampled points seen in xs(nT) all line up with x and xa, showing proper reconstruction from the aliased signal.

Part D

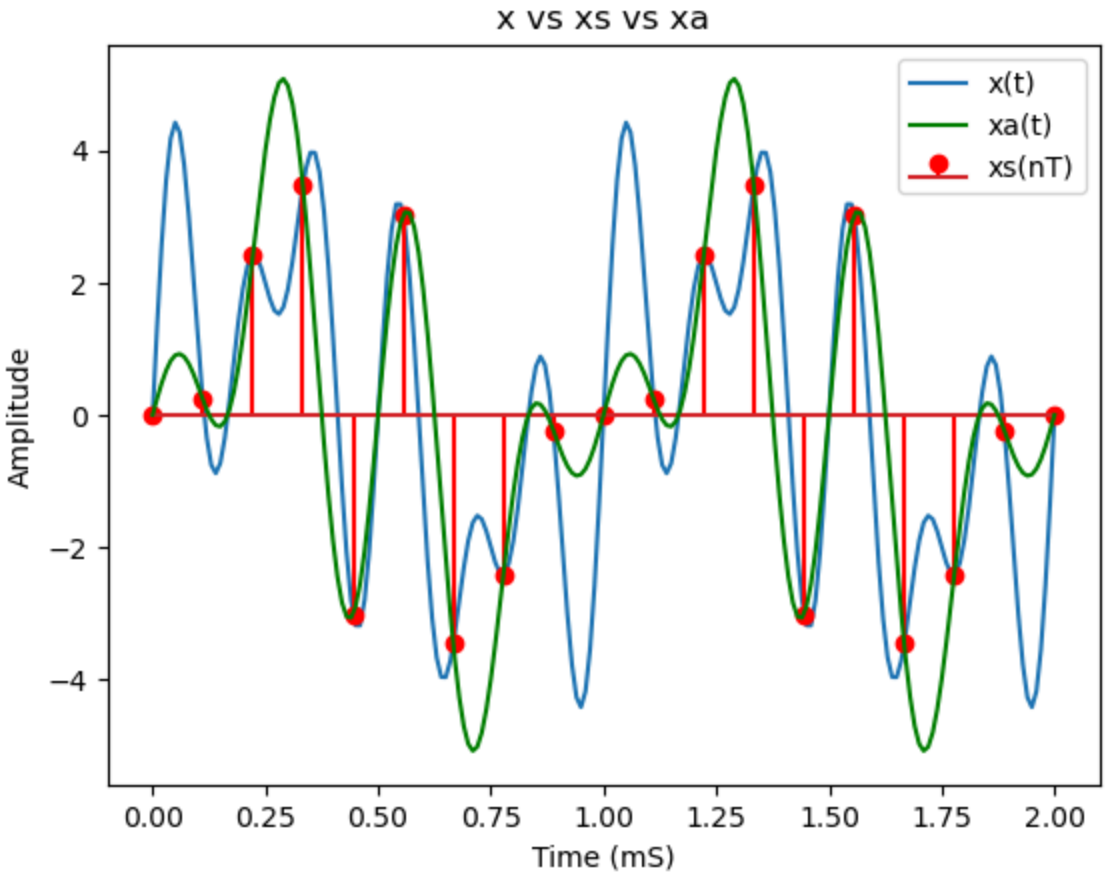
This part will have the same process as part c, with the exception being the sampling frequency increasing to 9000 Hz.

In [245…

```
fs = 9000
Ts = 1 / fs
nT = np.arange(0,2e-3+Ts, Ts)
xs =  2*np.sin(2*np.pi*f1*nT) + 2*np.sin(2*np.pi*f2*nT) + 2*np.sin(2*np.pi*f3*nT)

fa1 = 1000
fa2 = 4000
fa3 = -3000

xa = 2*np.sin(2*np.pi*fa1*tc) + 2*np.sin(2*np.pi*fa2*tc) + 2*np.sin(2*np.pi*fa3*tc)

plt.plot(tc*1000, x)
plt.stem(nT*1000,xs, "r")
plt.plot(tc*1000, xa,"g")
plt.title("x vs xs vs xa")
plt.xlabel("Time (mS)")
```

```
plt.ylabel("Amplitude")
plt.legend(['x(t)', 'xa(t)', 'xs(nT)'])
```

Out[245… `<matplotlib.legend.Legend at 0x23d3fc34170>`



The result of the same process with the higher frequency sampling is similar. The resulting waveform matches all of the sampled points, but the recreated signal doesn't match the original. In this case, we have 3 seperate frequencies, resulting in higher amplitudes, showing a bit more fidelity to the original but not close enough.

Part E

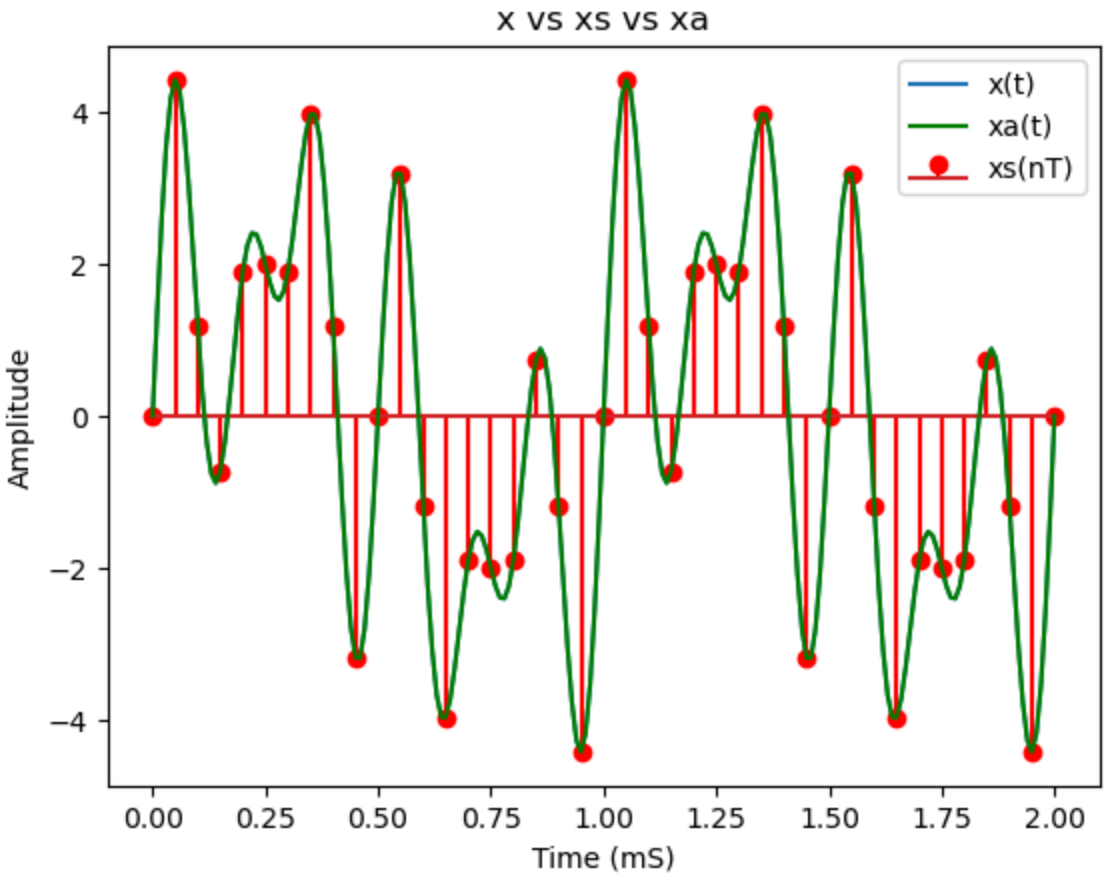
In this final part, we further increase the sampling frequency of 20000 HZ, an extent that it could be considered oversampled.

In [254…

```
fs = 20000
Ts = 1 / fs
nT = np.arange(0,2e-3+Ts, Ts)
xs =  2*np.sin(2*np.pi*f1*nT) + 2*np.sin(2*np.pi*f2*nT) + 2*np.sin(2*np.pi*f3*nT)

fa1 = 1000
fa2 = 4000
fa3 = 6000

xa = 2*np.sin(2*np.pi*fa1*tc) + 2*np.sin(2*np.pi*fa2*tc) + 2*np.sin(2*np.pi*fa3*tc)
```

```
plt.plot(tc*1000, x)
plt.stem(nT*1000,xs, "r")
plt.plot(tc*1000, xa,"g")
plt.title("x vs xs vs xa")
plt.xlabel("Time (mS)")
plt.ylabel("Amplitude")
plt.legend(['x(t)', 'xa(t)', 'xs(nT)'])
```

Out[254… `<matplotlib.legend.Legend at 0x23d3fbee300>`



The result of this attempt is a recreation that looks exactly like the original signal. This is due to the sampling frequency 20 kHz being more than twice the highest frequency of 6 kHz. On top of having the same amplitude at every sampled point, every point in between has the same value as well. This shows the right execution of sampling a signal at a high enough frequency and recreating it without having any aliasing.

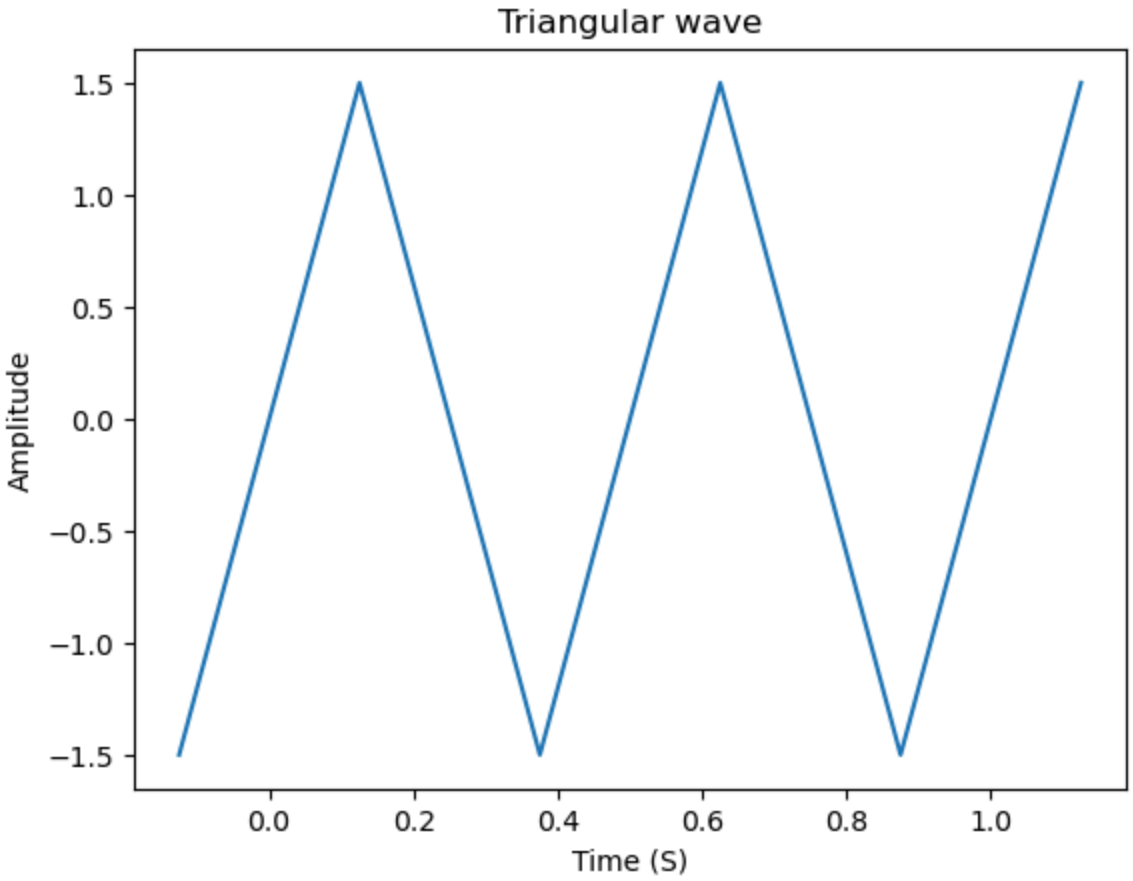
Problem 01-02 Aliasing with a triangular wave

Part A

In this first part, we have to create a triangular wave, exactly like the one seen on problem T01-07. This is to be done by creating the first segment of x, then concatenating it together and flipping it repeatedly until we have the right signal.

In [279…
```
t = np.arange(-0.125, 1.13, 0.01)
x = np.arange(-0.125,0.13,0.01)*12
x = np.concatenate((x,x[-2:0:-1],x,x[-2:0:-1],x))
plt.plot(t,x)
plt.title("Triangular wave")
plt.ylabel("Amplitude")
plt.xlabel("Time (S)")
```

Out[279…
```
Text(0.5, 0, 'Time (S)')
```



Seen above is the signal x(t), which is exaclt the same signal as the one seen on the problem T01-07.

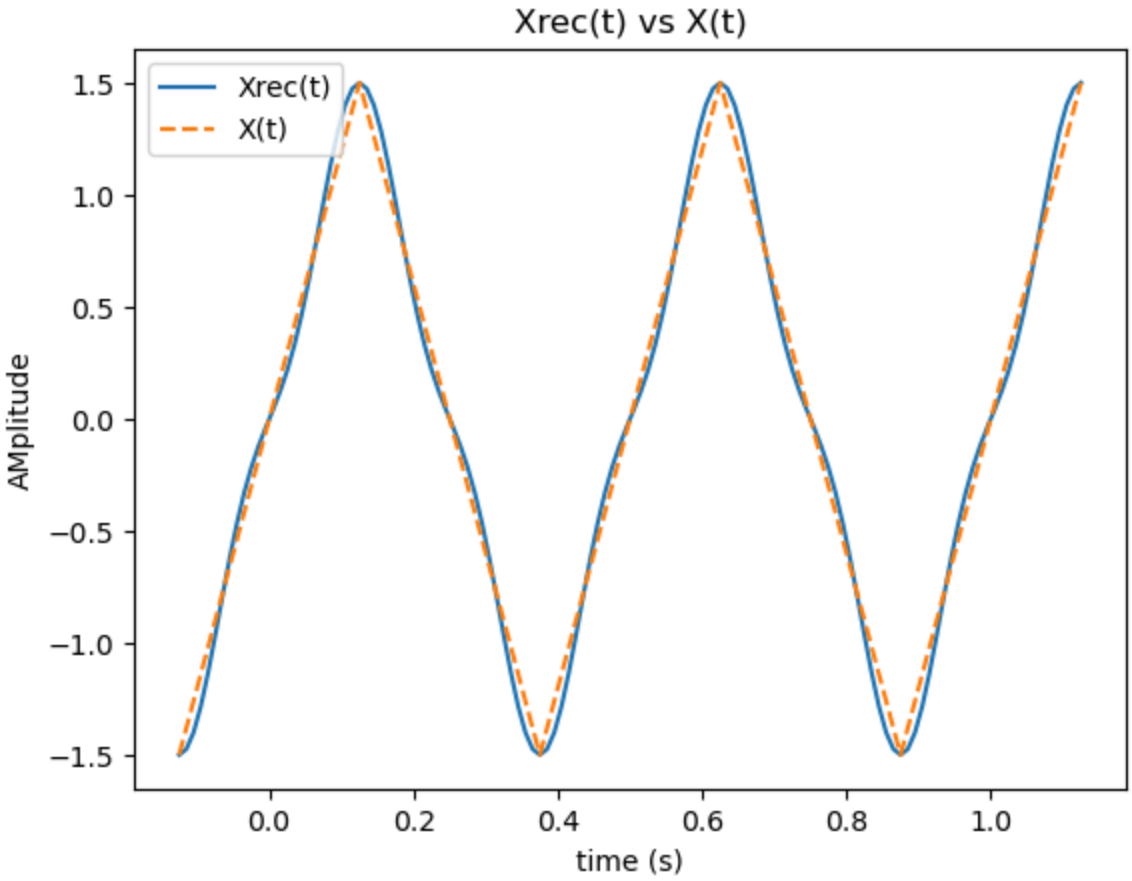
Part B

In this part, we're going to use the results from our calculations in problem T01-07 to plot Xrec(t) the recreation of the signal after sampling at 16 Hz and recreating it.

In [286…
```
A = 1.28
B = -0.220
f1 = 2
f2 = 6

Xrec = A*np.sin(2*np.pi*f1*t)+B*np.sin(2*np.pi*f2*t)
plt.plot(t,Xrec)
plt.plot(t,x,'--')
```

```
plt.title('Xrec(t) vs X(t)')
plt.xlabel('time (s)')
plt.ylabel('AMplitude')
plt.legend(['Xrec(t)', 'X(t)'])
```

Out[286… `<matplotlib.legend.Legend at 0x23d418bbd40>`



As we can see the recreated signal looks very close to the original signal.

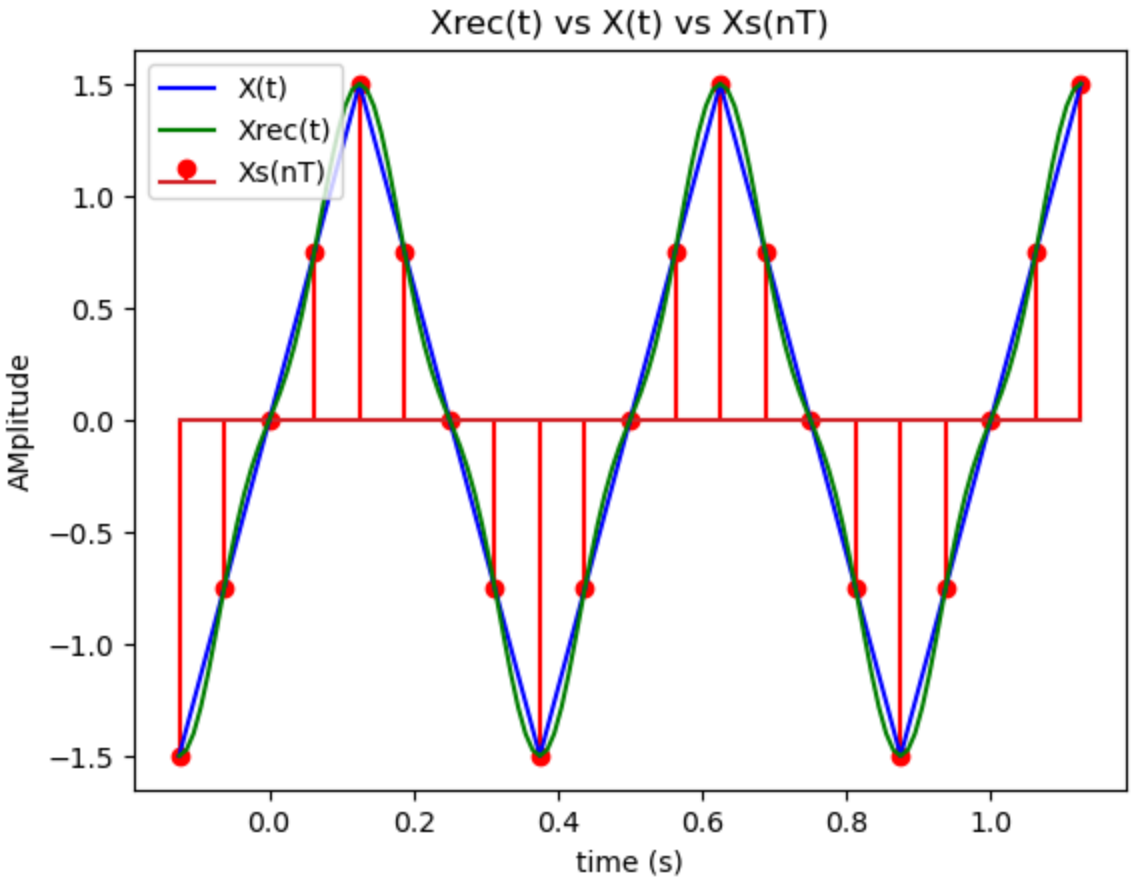
Part c

For the final part, we simply have to verify that the approximation is reasonable. By plotting the sampled points as well we can do a quick sanity check.

In [297…

```
fs = 16
Ts = 1 / fs
nT = np.arange(-0.125,1.125+Ts, Ts)
xs = A*np.sin(2*np.pi*f1*nT)+B*np.sin(2*np.pi*f2*nT)

plt.stem(nT,xs, 'r')
plt.plot(t,x, 'b')
plt.plot(t,Xrec, 'g')
plt.title('Xrec(t) vs X(t) vs Xs(nT)')
plt.xlabel('time (s)')
plt.ylabel('AMplitude')
plt.legend(['X(t)','Xrec(t)', 'Xs(nT)'])
```

Out[297… <matplotlib.legend.Legend at 0x23d4185e300>



As we can see in the above plot, the sampled points correspond to both the original signal and the sampled signal. This proves that our Xrec was generated correctly.