PHYS 391

Day 17

- Aliasing
- Fast Fourier Transform

Lab 4

- 4.5 - Poisson Statistics
- 4.6 - Gaussian Statistics
- 4.7 - Inverse Square Law
- 4.8 - Attenuation Length

Any last questions
on what you are doing?

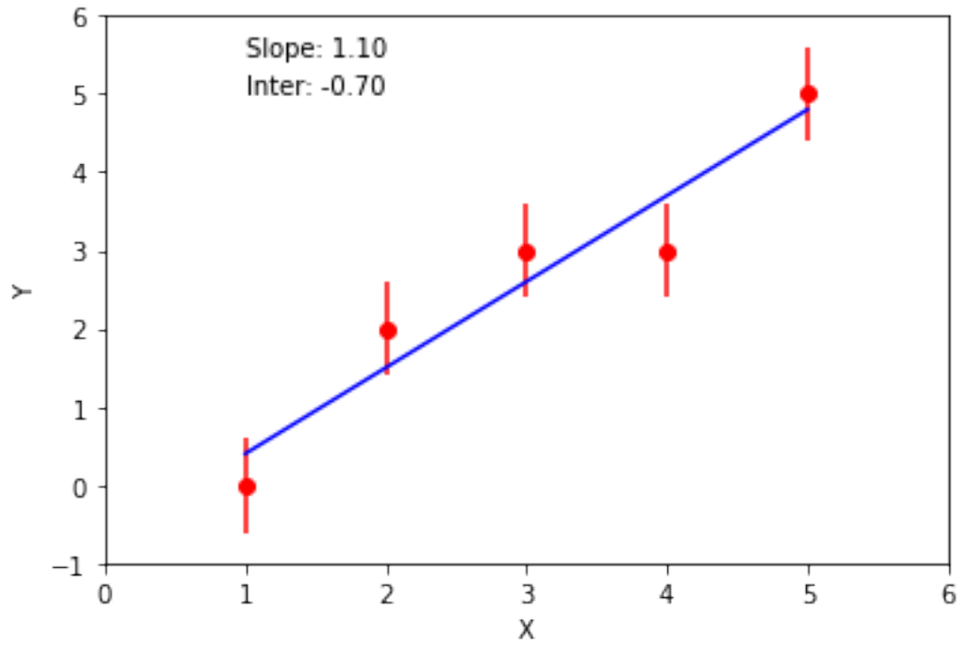
Linear Fits

- Must fit ln(R) vs. distance or thickness to extract the inverse power law or attenuation length - careful with errors

- Showed examples of how to do this using polyfit back on Day 8 - covariance matrix

- Note, by default polyfit will bootstrap the errors for you! So even if you don't pass errors to the fit, you will get reasonable errors returned from your fit. To turn this off, include cov='unscaled' in the arguments.

- Could also use equations in Chapter 8, or in a pinch could just 'eyeball' the slope uncertainty using the method from HW1



Creating Data

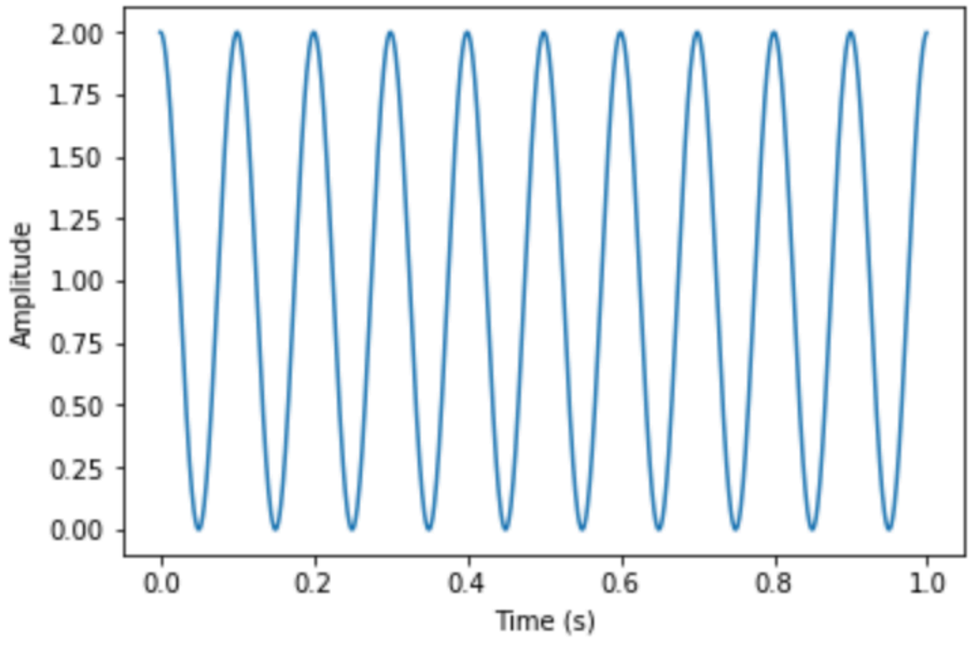
- Normal python setup

```
In [2]: %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
```

- Create simple waveform with $\nu_s >> \nu_0$

$$\cos(2\pi\nu_0 t) + 1$$

```
In [3]: fs = 1000.  # Sampling frequency (in Hz)
        Ts = 1/fs   # delta t (spacing between each time point)
        Range = 1.  # Total range of data (in seconds)

        f0 = 10.0
        t0 = np.arange(0., Range, Ts) # Point every 1 ms
        y0 = np.cos(2*np.pi*f0*t0)+1

        plt.plot(t0,y0)
        #plt.plot(t0,y0,'o')
        plt.xlabel('Time (s)')
        plt.ylabel('Amplitude')
        plt.show()
```



Note arguments to np.arange()

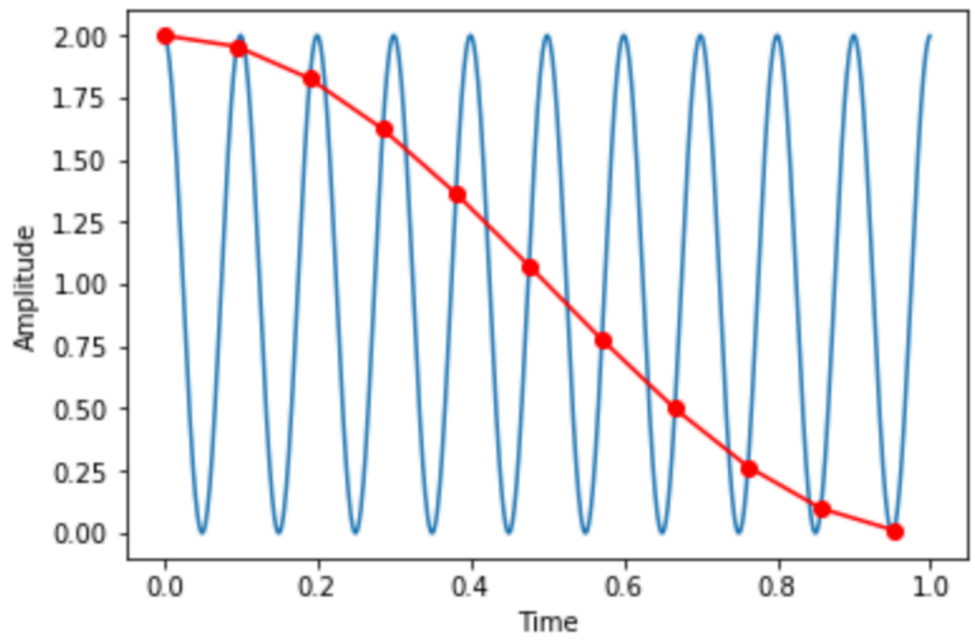
Aliasing

- Nyquist Limit: $v_s > 2\ v_0$

- What happens if we violate this?

$$v_s = 10.5 \text{ Hz}, v_0 = 10 \text{ Hz}$$

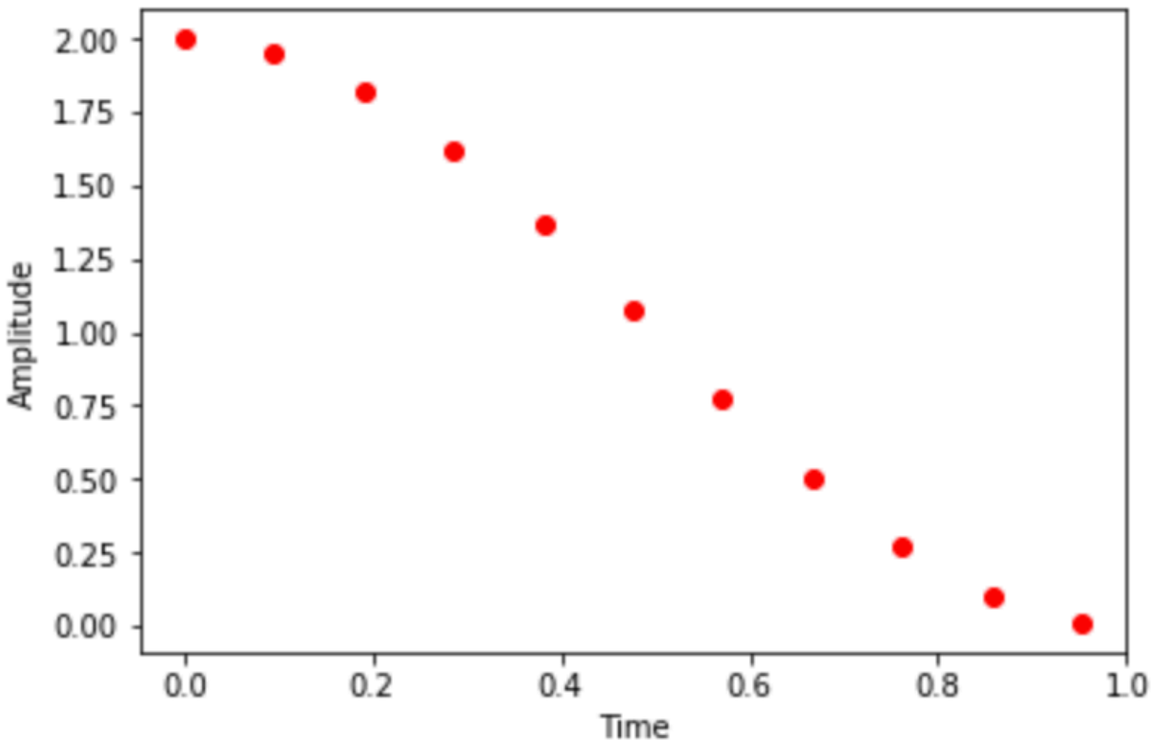
In [4]:
```
# First plot the original function
plt.plot(t0,y0)

fs = 10.5# Reduce sampling frequency a lot
Ts = 1/fs

t1 = np.arange(0., Range, Ts)
y1 = np.cos(2*np.pi*f0*t1)+1
plt.plot(t1,y1,'ro')
plt.plot(t1,y1,'r')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



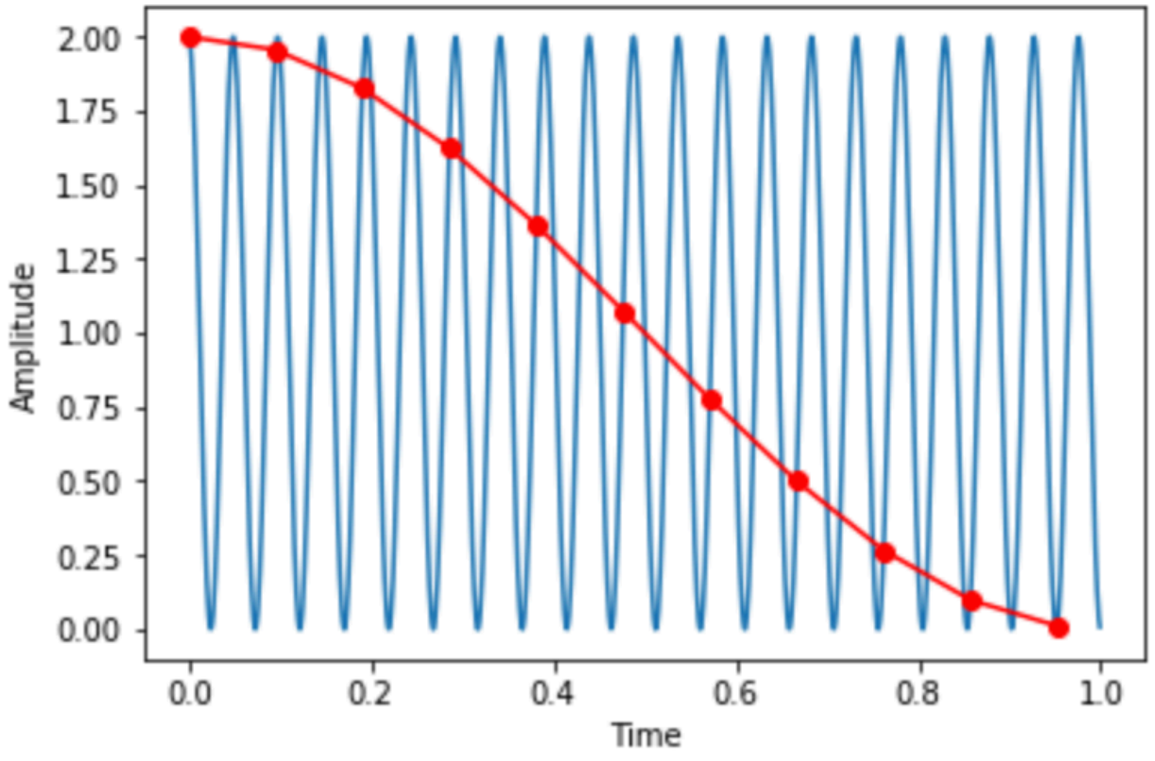
Aliasing

- Really can't tell what the 'real' frequency is

v_{obs} = 0.5 Hz



v_s = 10.5 Hz, v_0 = 20.5 Hz



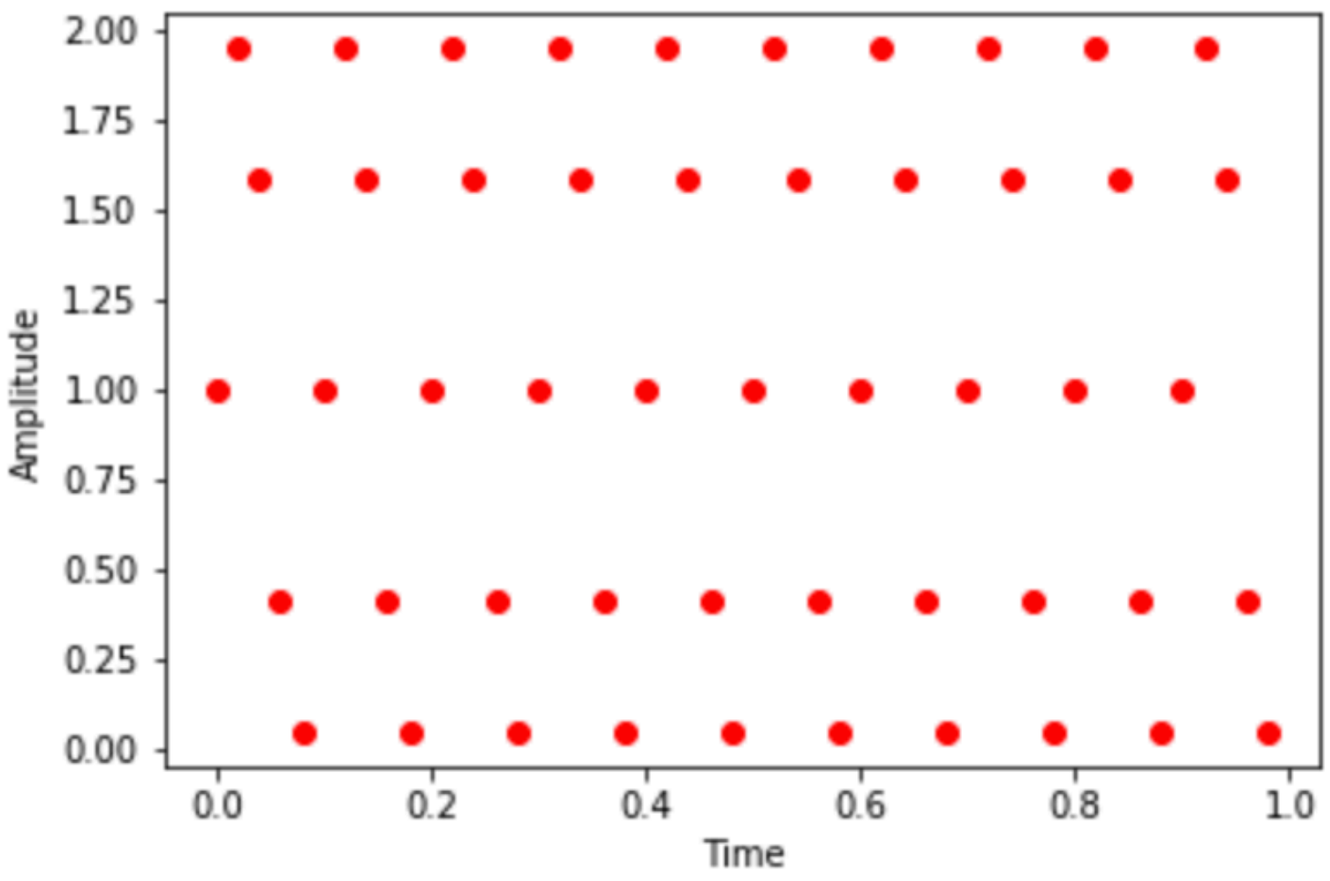
Nothing really to do aside from filter high frequencies: $v > v_s/2$

If you can adjust v_s it is possible to detect aliased frequencies

Fast Fourier Transform

- Example function: $\sin(2\pi\nu_0 t) + 1$ $\quad$ v_s = 50 Hz, v_0 = 10 Hz

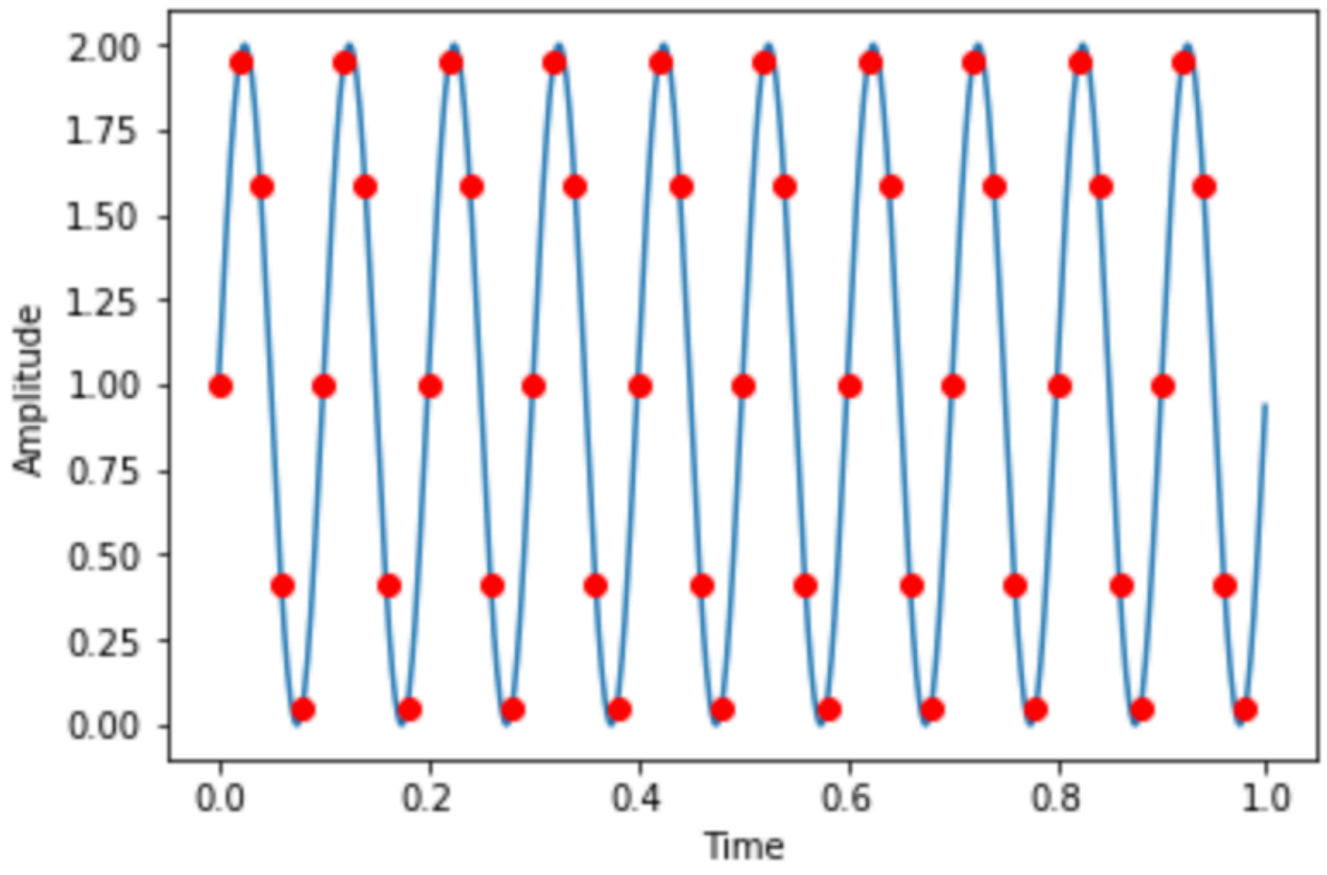
[33]:
```
f0 = 10.0
fs = 50. # Sample this at 50 Hz
Ts = 1/fs

t1 = np.arange(0., Range, Ts)
y1 = np.sin(2*np.pi*f0*t1)+1
plt.plot(t1,y1,'ro')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



Fast Fourier Transform

- Example function: $\sin(2\pi\nu_0 t) + 1$ $\quad$ v_s = 50 Hz, v_0 = 10 Hz

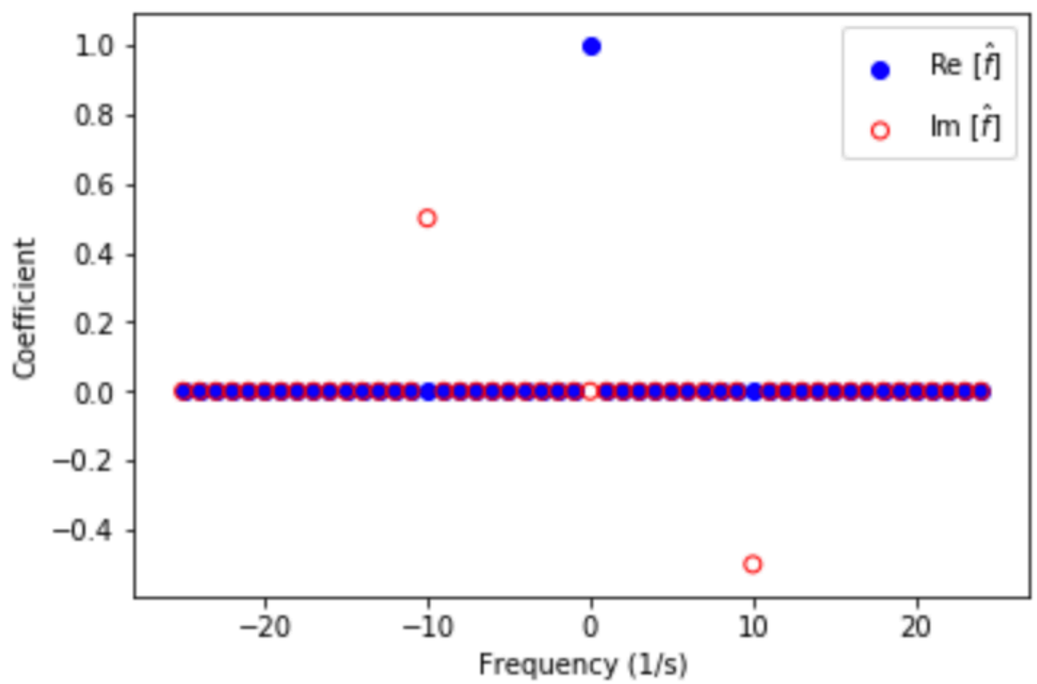
[33]:
```
f0 = 10.0
fs = 50. # Sample this at 50 Hz
Ts = 1/fs

t1 = np.arange(0., Range, Ts)
y1 = np.sin(2*np.pi*f0*t1)+1
plt.plot(t1,y1,'ro')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



FFT Complex Coeff.

[13]:
```
# Discrete Fourier transform
n = len(y1) # length of the signal
ft1 = np.fft.fft(y1)/n # fft computing and normalization

# Create frequency array (depends on sampling time)
frq = np.fft.fftfreq(n, Ts)

plt.scatter(frq,np.real(ft1), facecolors='b', edgecolors='b',label='Re $[ \hat{f} ] $')
plt.scatter(frq,np.imag(ft1), facecolors='none', edgecolors='r', label='Im $[ \hat{f} ] $')
plt.xlabel('Frequency (1/s)')
plt.ylabel('Coefficient')
plt.legend()
plt.show()
```

As shown last week



FFT Complex Coeff.

[13]:
```
# Discrete Fourier transform
n = len(y1) # length of the signal
ft1 = np.fft.fft(y1)/n # fft computing and normalization

# Create frequency array (depends on sampling time)
frq = np.fft.fftfreq(n, Ts)
```

- fft( ) takes as an argument just the array of function values
- Returns same length array of complex coefficients (must divide by n to get amplitudes)
- Order of these coefficients (by index) is 0, 1, …, N/2, -N/2+1, …, -1 !
- Easiest to use fftfreq() to get array of actual frequencies for plotting
- Depends on $\Delta t = 1/\nu_s$ since $\Delta \nu = 1/T = \nu_s/N$
- Δt in time series and $\Delta \nu$ in FFT are linked!

Plotting

```
plt.scatter(frq,np.real(ft1), facecolors='b', edgecolors='b',label='Re $[ \hat{f} ] $')
plt.scatter(frq,np.imag(ft1), facecolors='none', edgecolors='r', label='Im $[ \hat{f} ] $')
plt.xlabel('Frequency (1/s)')
plt.ylabel('Coefficient')
plt.legend()
plt.show()
```

- Plot real/imaginary part of coefficient array vs. frequency array

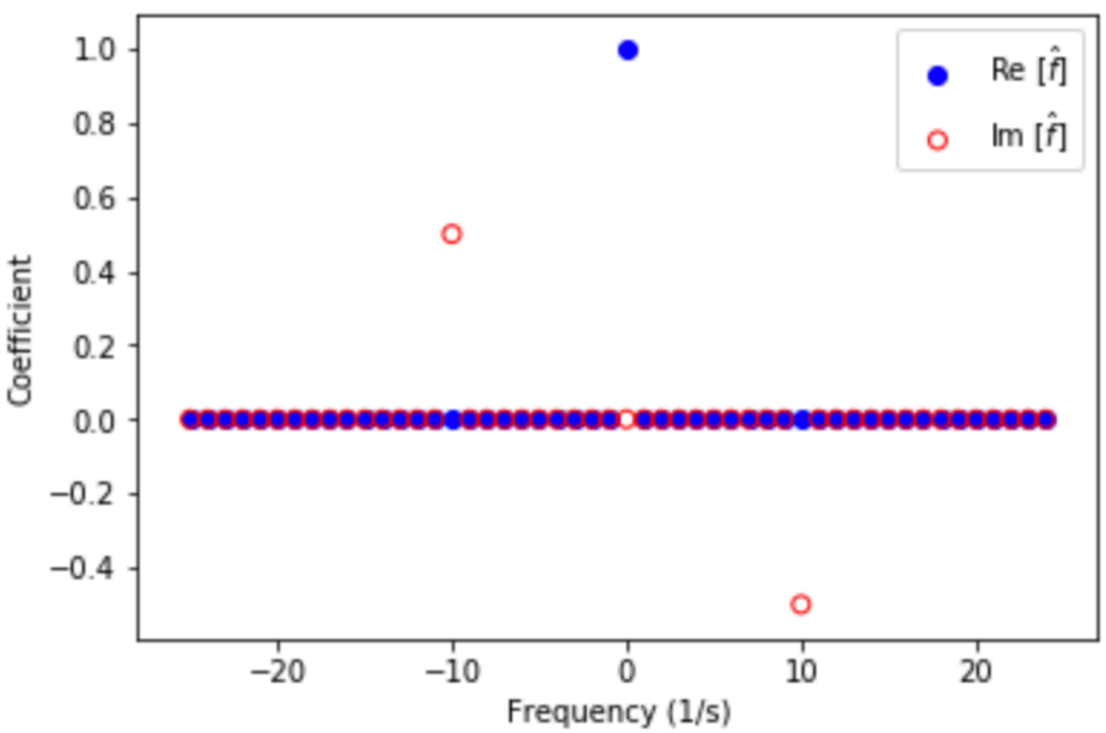
- Note use of np.real() and np.imag() to convert complex number (could also just use ft1.real and ft1.imag)

- Also useful: np.abs() to get magnitude

Magnitude vs. Freq.

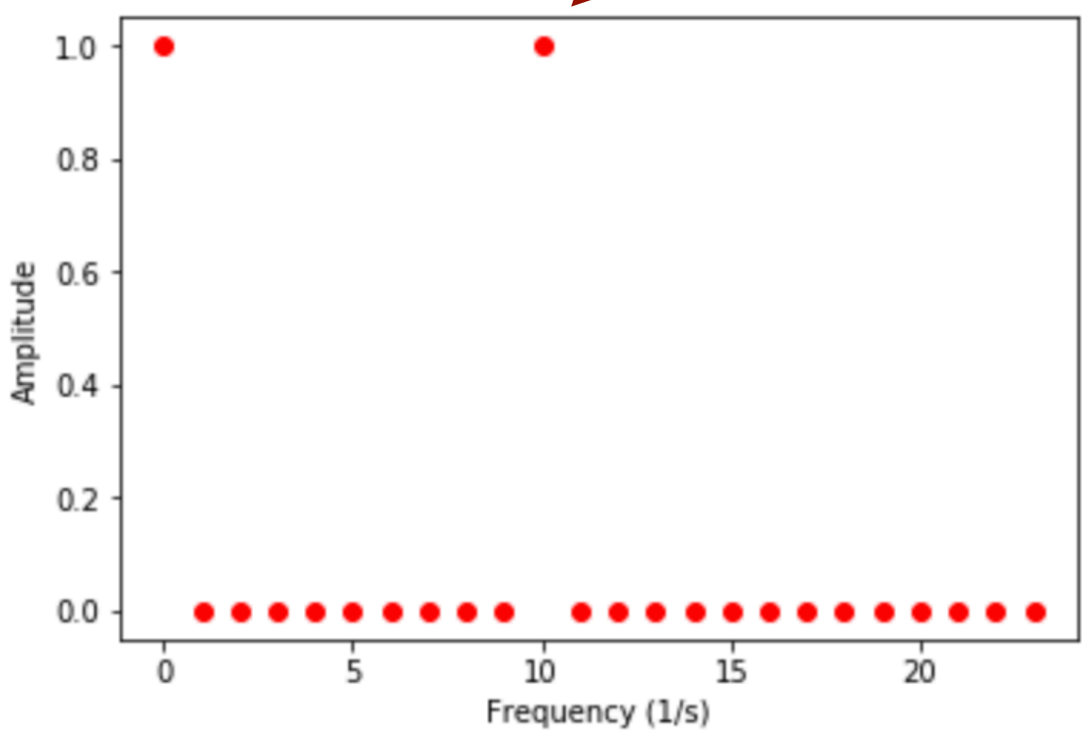
- This is the key part of your last homework

Complex form



Magnitude

Amplitude 1 at 10 Hz



Array indexing is your friend, but must be careful about v = 0 point. Can do this very ‘brute force’…