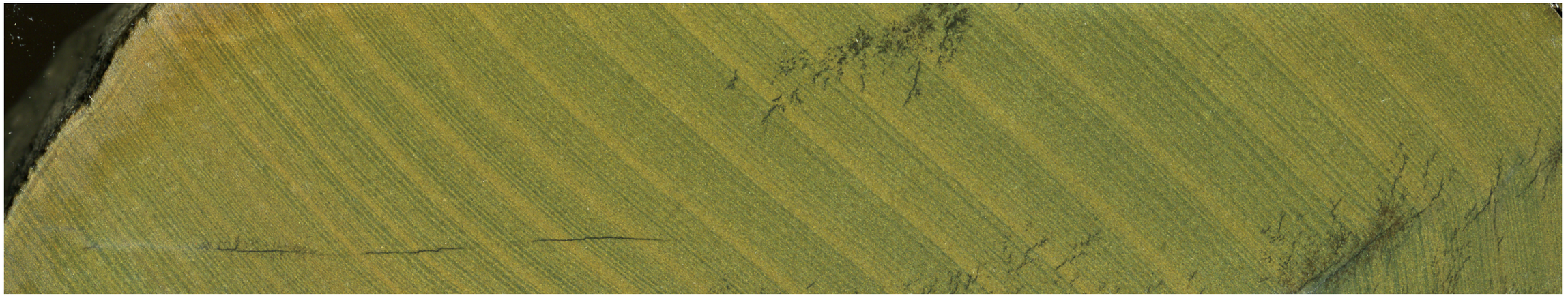


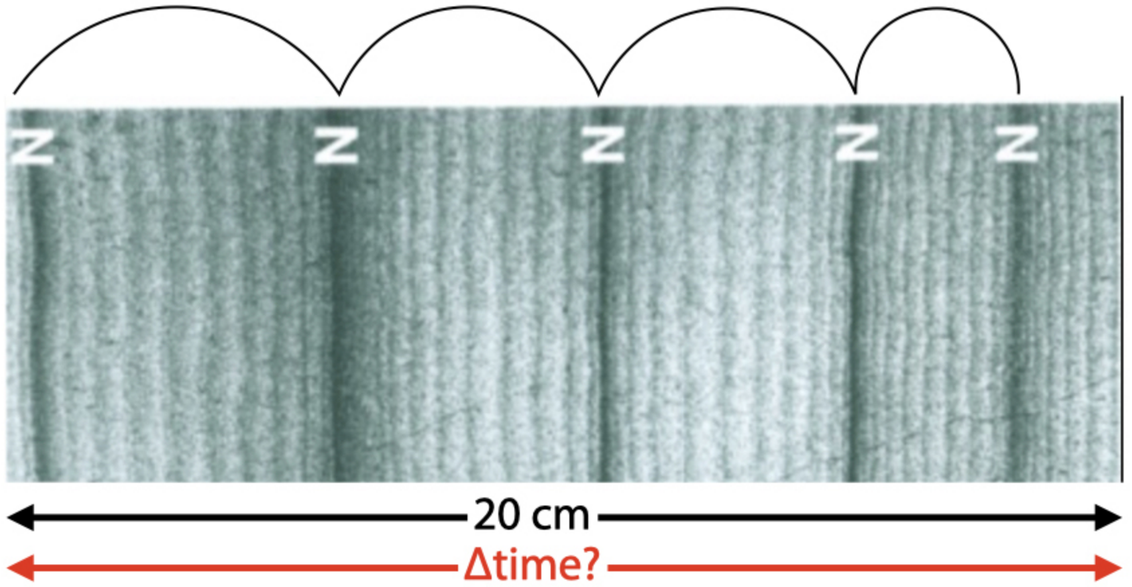
Lecture 10: Cycles

1. Cyclostratigraphy
2. How to identify a cycle
3. Fourier Transforms

We acknowledge and respect the lək^{w}əŋən peoples on whose traditional territory the university stands and the Songhees, Esquimalt and W̱SÁNEĆ peoples whose historical relationships with the land continue to this day.

Cyclostratigraphy

- **Cyclostratigraphy** is a sub-discipline of stratigraphy that seeks to identify, characterize and interpret cyclic variations in the stratigraphic record
    - identify cycles → interpret timing of cycles → age models and correlations
    - fundamentally, it is about explaining the processes *behind* a record



Identifying cycles

In [1]:

```
from scipy.fftpack import fft
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
%matplotlib inline

def plot_data_res(x,y):
    fig1=plt.figure(1)
    fig1.clf()
    ax1=fig1.add_subplot(121)

    ax1.plot(x,y,'-',color='#1E90FF')

    z=np.polyfit(x,y,1)
    p=np.poly1d(z)

    ax1.plot(x,p(x),'--',color='#B22222')
    ax1.text(0.05,0.91,r'r$^2$ = %2.2f' %(np.corrcoef(x,y)[0,1]**2),
             horizontalalignment='left',transform=ax1.transAxes)

    ax2=fig1.add_subplot(122)
    res=y-p(x)

    ax2.hist(res,color='#808080')

    ax2.text(0.05,0.91,r'$\mu$ = %2.1f' %(np.mean(res)),
             horizontalalignment='left',transform=ax2.transAxes)
    ax2.text(0.05,0.85,r'$\sigma$ = %2.1f' %(np.std(res)),
             horizontalalignment='left',transform=ax2.transAxes)
```

Identifying cycles

In [2]:

```
def plot_data_res_fft(x,y,yf,xf):
    fig1=plt.figure(1)
    fig1.clf()
    ax1=fig1.add_subplot(221)
    ax1.plot(x,y,'-',color='#1E90FF')
    z=np.polyfit(x,y,1)
    p=np.poly1d(z)
    ax1.plot(x,p(x),'--',color='#B22222')
    ax1.text(0.05,0.91,r'r$^2$ = %2.2f' %(np.corrcoef(x,y)[0,1]**2),
             horizontalalignment='left',transform=ax1.transAxes)
    ax2=fig1.add_subplot(222)
    res=y-p(x)
    ax2.hist(res,color='#808080')
    ax2.text(0.05,0.91,r'$\mu$ = %2.1f' %(np.mean(res)),
             horizontalalignment='left',transform=ax2.transAxes)
    ax2.text(0.05,0.85,r'$\sigma$ = %2.1f' %(np.std(res)),
             horizontalalignment='left',transform=ax2.transAxes)
    ax3=fig1.add_subplot(212)
    ax3.plot(xf,2.0/N *np.abs(yf[0:N//2]))
    ax3.text(0.99,0.9,'a periodogram',transform=ax3.transAxes,horizontalalignment='right')

    power=2.0/N *np.abs(yf[0:N//2])

    max_power=np.max(2.0/N *np.abs(yf[0:N//2]))
    freq_idx=np.where(power==max_power)[0]
    freq=np.round(xf[freq_idx],2)

    ax3.plot(xf[freq_idx],max_power,'ro')

    return [ax1,ax2,ax3],max_power,freq
```

Identifying cycles

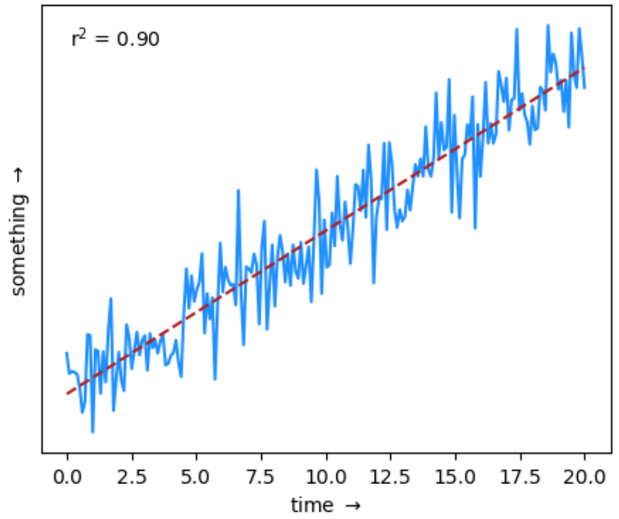
In [3]:

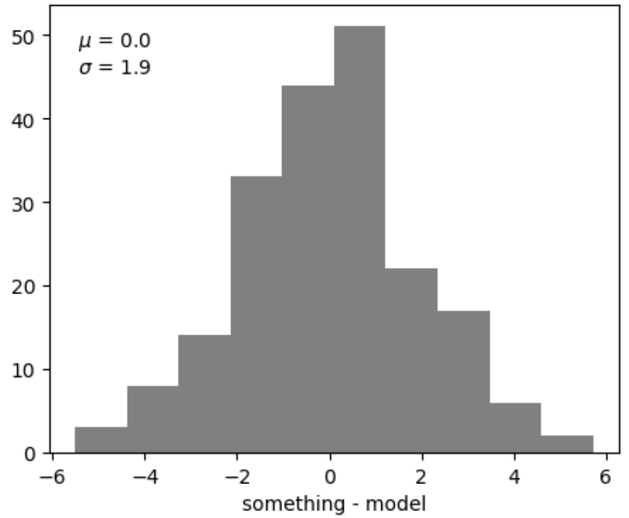
```
t = 20 # total time series
N=200 # sample spacing
x = np.linspace(0.0, t, N) #noise amplitude
a=2
noise=np.random.normal(0,a,N) #total signal
y=x+noise # plot_data_res(x,y)
```

Identifying cycles

In [3]:

```
t = 20 # total time series
N=200 # sample spacing
x = np.linspace(0.0, t, N) #noise amplitude
a=2
noise=np.random.normal(0,a,N) #total signal
y=x+noise # plot_data_res(x,y)
```





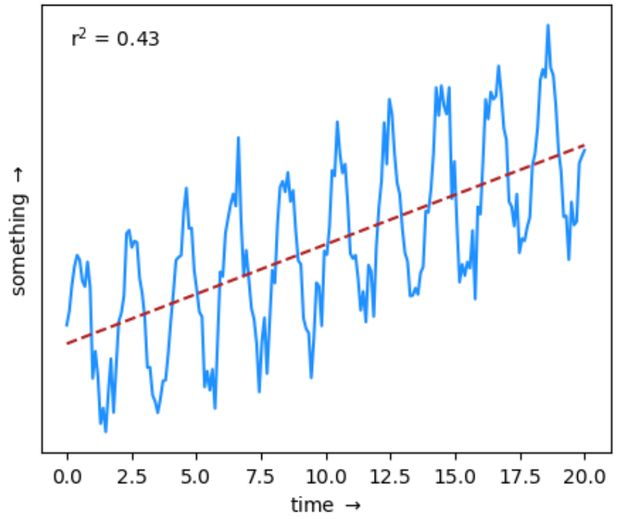
Identifying cycles

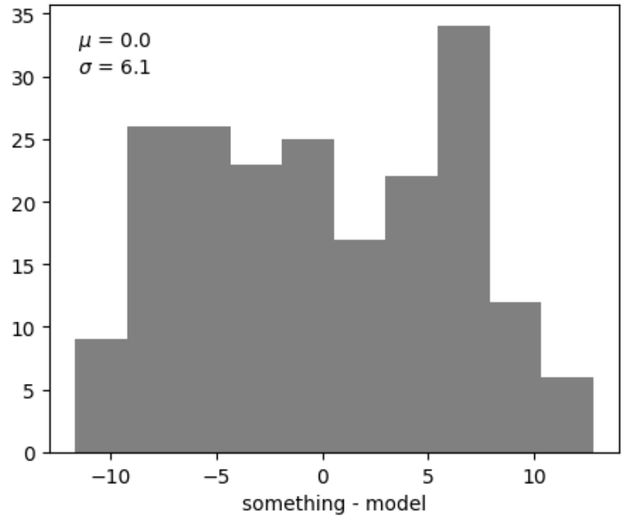
In [4]:

```
#add a sine wave
y=y+8*np.sin(0.5*2*np.pi*x)

# plot_data_res(x,y)
```

Identifying cycles

In [4]:
```
#add a sine wave
y=y+8*np.sin(0.5*2*np.pi*x)

# plot_data_res(x,y)
```





Identifying cycles

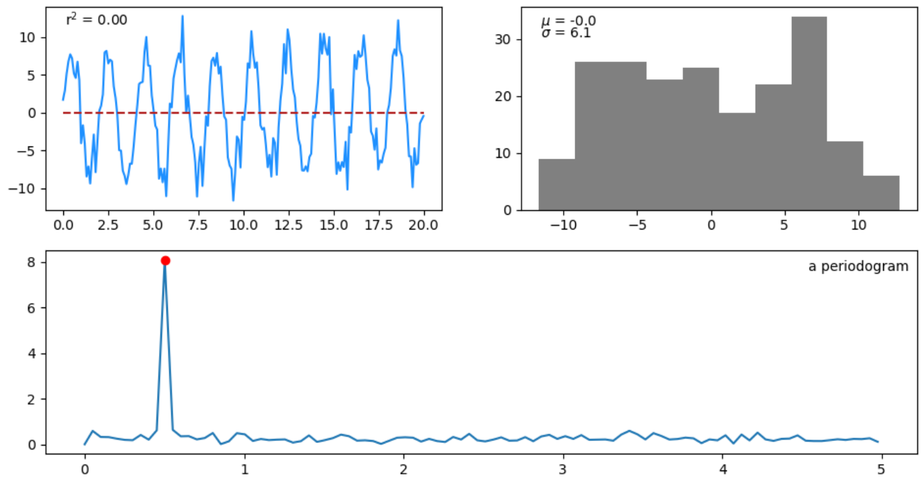
In [5]:

```
#remove the linear rise
y=signal.detrend(y)
#define sampling interval for frequency axis
dt=np.round(np.diff(x),10)[0]
#perform Fast Fourier transform
yf = fft(y)
xf = np.linspace(0.0, 1.0/(2.0*dt), N//2)
#plot results
# axes,max_power,freq=plot_data_res_fft(x,y,yf,xf)
```

Identifying cycles

In [5]:

```
#remove the linear rise
y=signal.detrend(y)
#define sampling interval for frequency axis
dt=np.round(np.diff(x),10)[0]
#perform Fast Fourier transform
yf = fft(y)
xf = np.linspace(0.0, 1.0/(2.0*dt), N//2)
#plot results
# axes,max_power,freq=plot_data_res_fft(x,y,yf,xf)
```



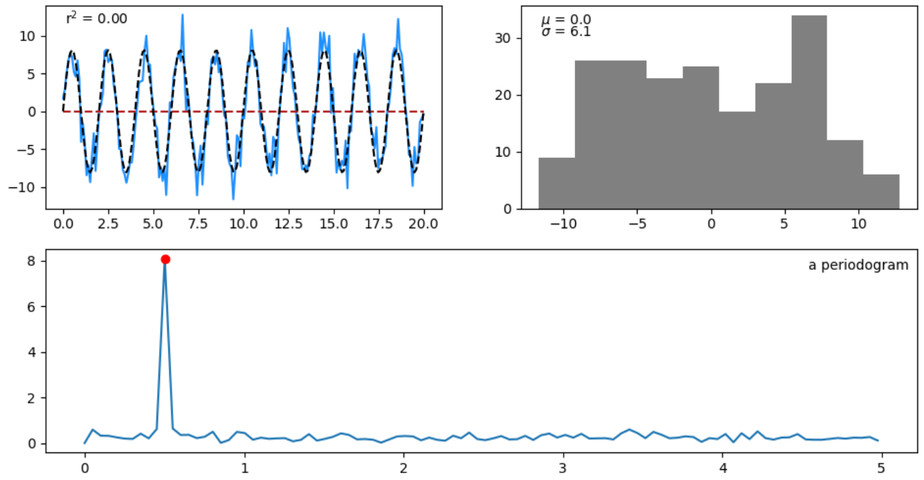
Identifying cycles

In [8]:

```
#create and plot the recovered sinusoid
amp=max_power
ang_freq=freq * 2*np.pi
phase=0 * np.pi/180
cycle=amp*np.sin(ang_freq*x + phase)
#axes,max_power,freq=plot_data_res_fft(x,y,yf,xf)
#axes[0].plot(x,cycle,'k--')
```

Identifying cycles

In [8]:
```
#create and plot the recovered sinusoid
amp=max_power
ang_freq=freq * 2*np.pi
phase=0 * np.pi/180
cycle=amp*np.sin(ang_freq*x + phase)
#axes,max_power,freq=plot_data_res_fft(x,y,yf,xf)
#axes[0].plot(x,cycle,'k--')
```



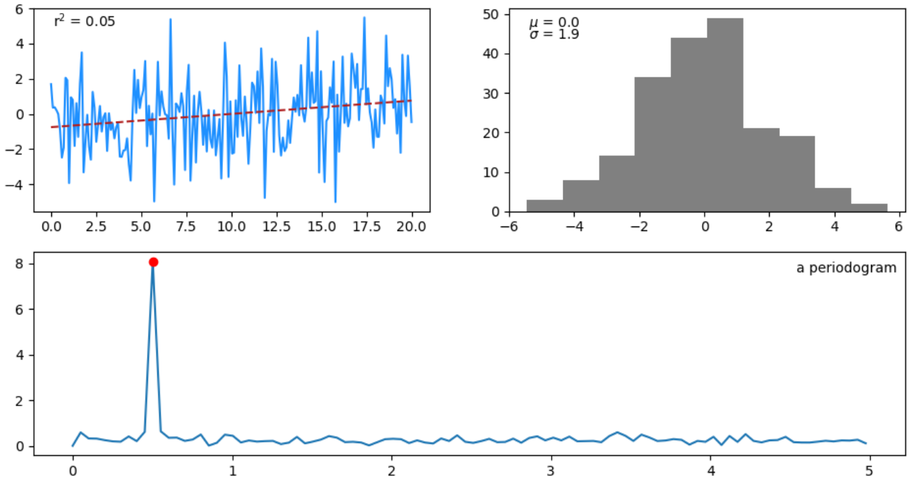
Identifying cycles

In [18]:

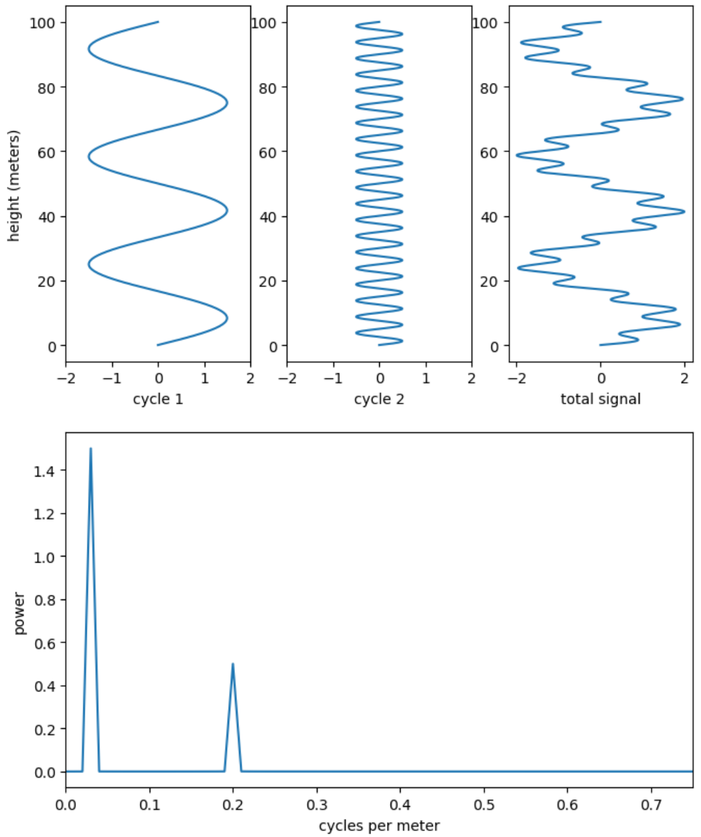
```
##recalculate residuals
#axes=plot_data_res_fft(x,y-cycle,yf,xf)
```

Identifying cycles

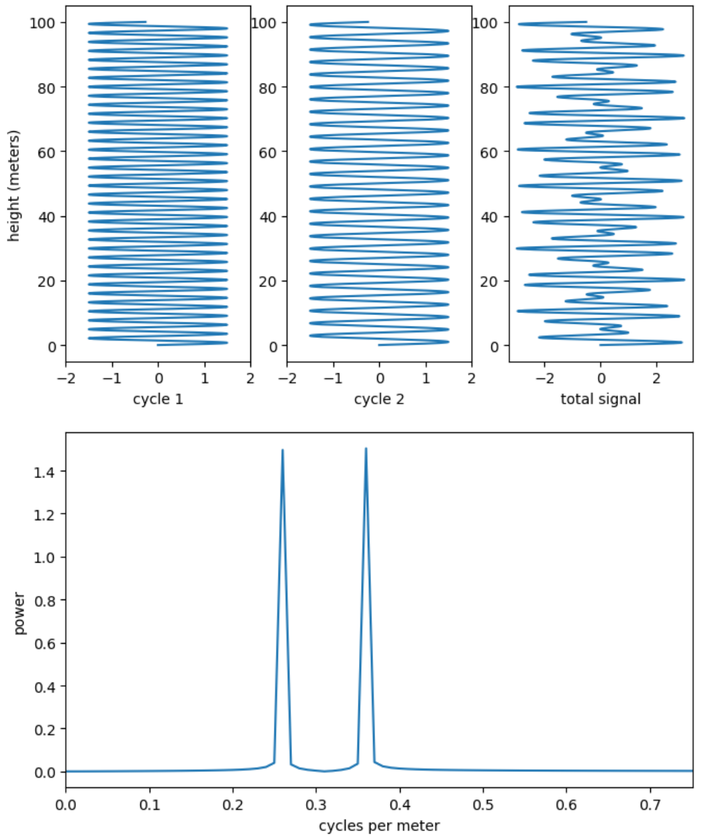
In [18]:
```
##recalculate residuals
#axes=plot_data_res_fft(x,y-cycle,yf,xf)
```



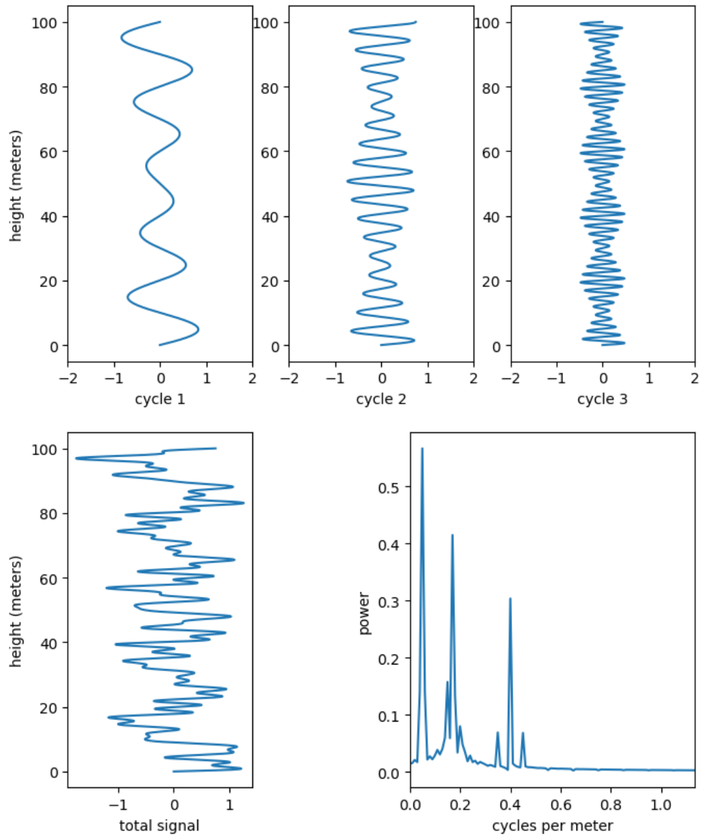
quickly we cannot rely on our eyes …



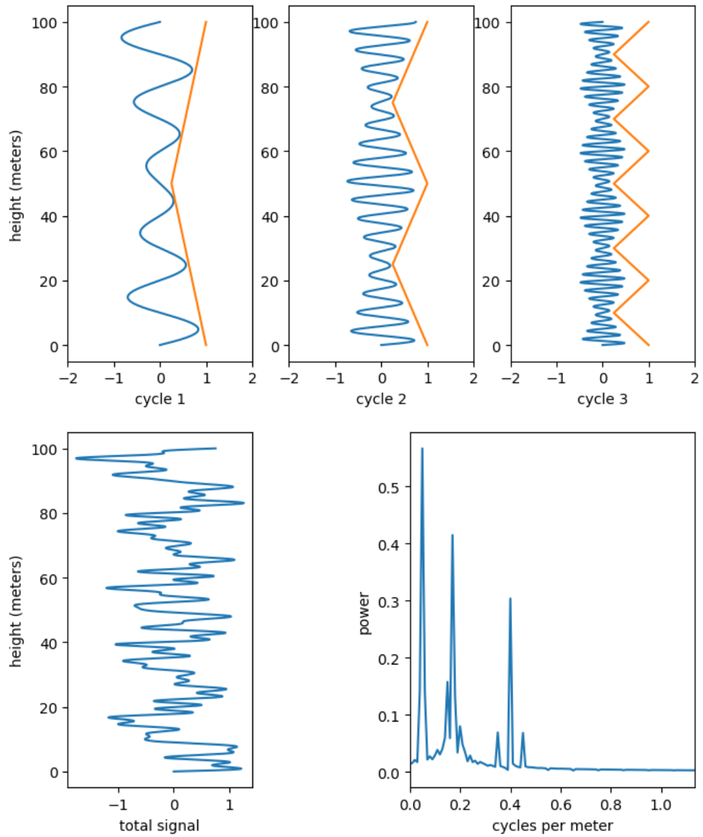
quickly we cannot rely on our eyes …



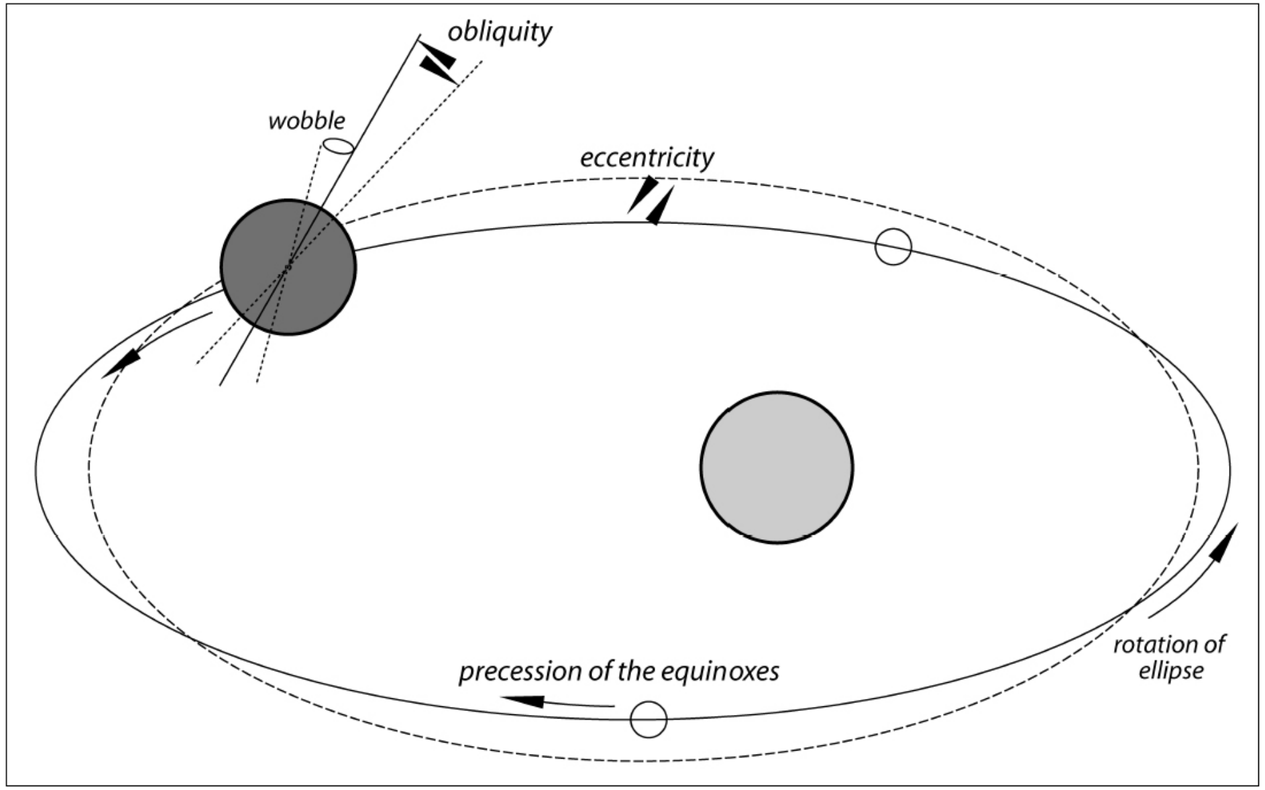
quickly we cannot rely on our eyes …



quickly we cannot rely on our eyes …



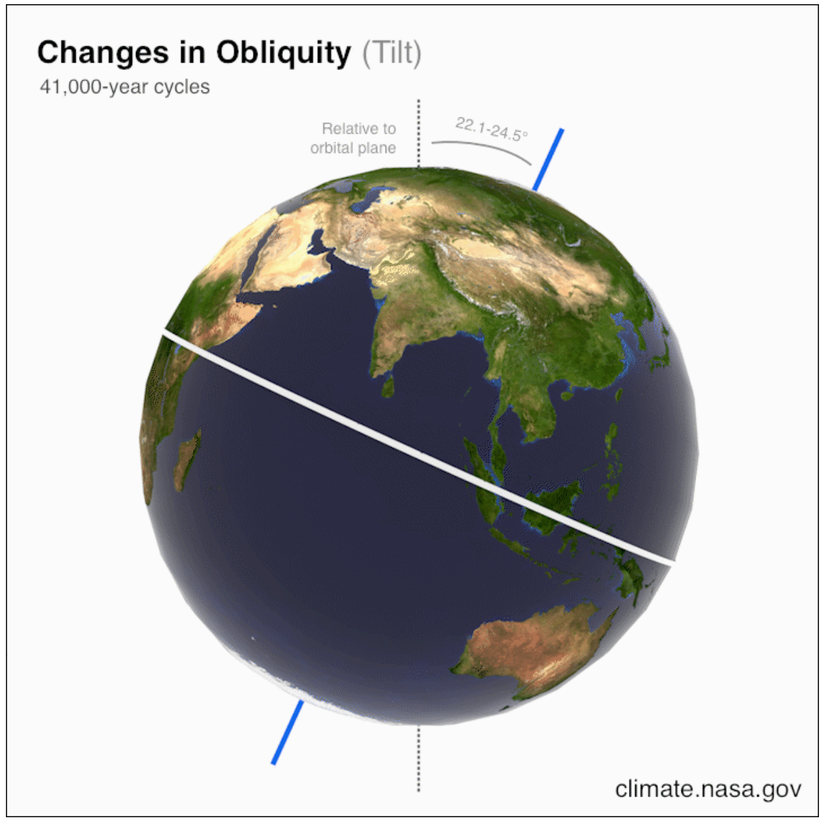
(Draw cycles in sediments)

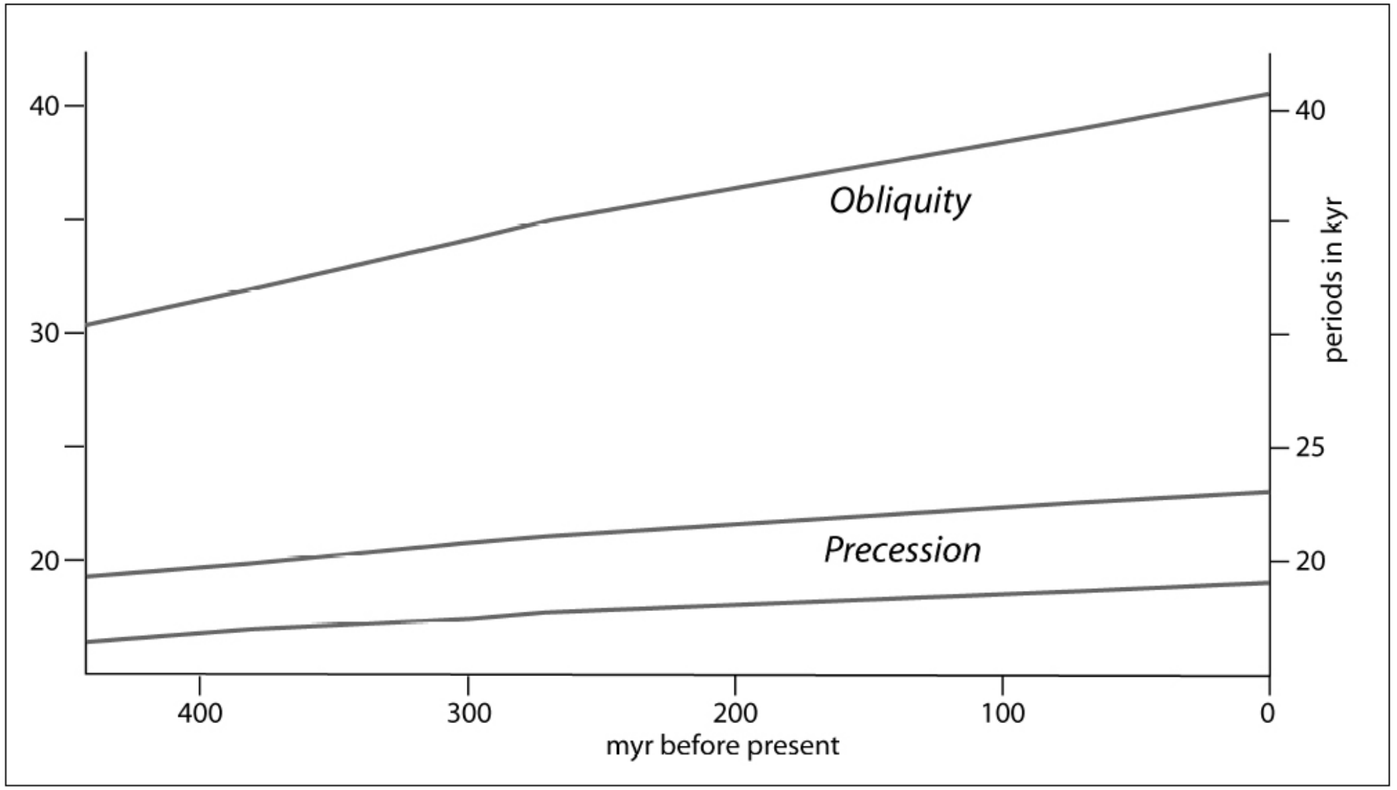

obliquity
wobble
eccentricity
precession of the equinoxes
rotation of ellipse

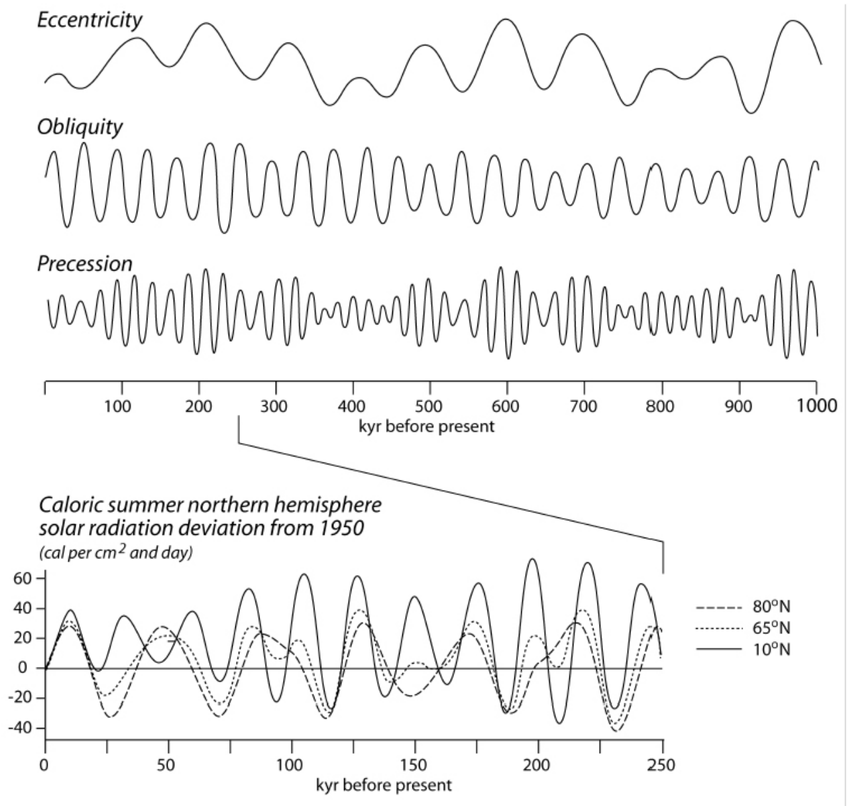
Axial Precession (Wobble)

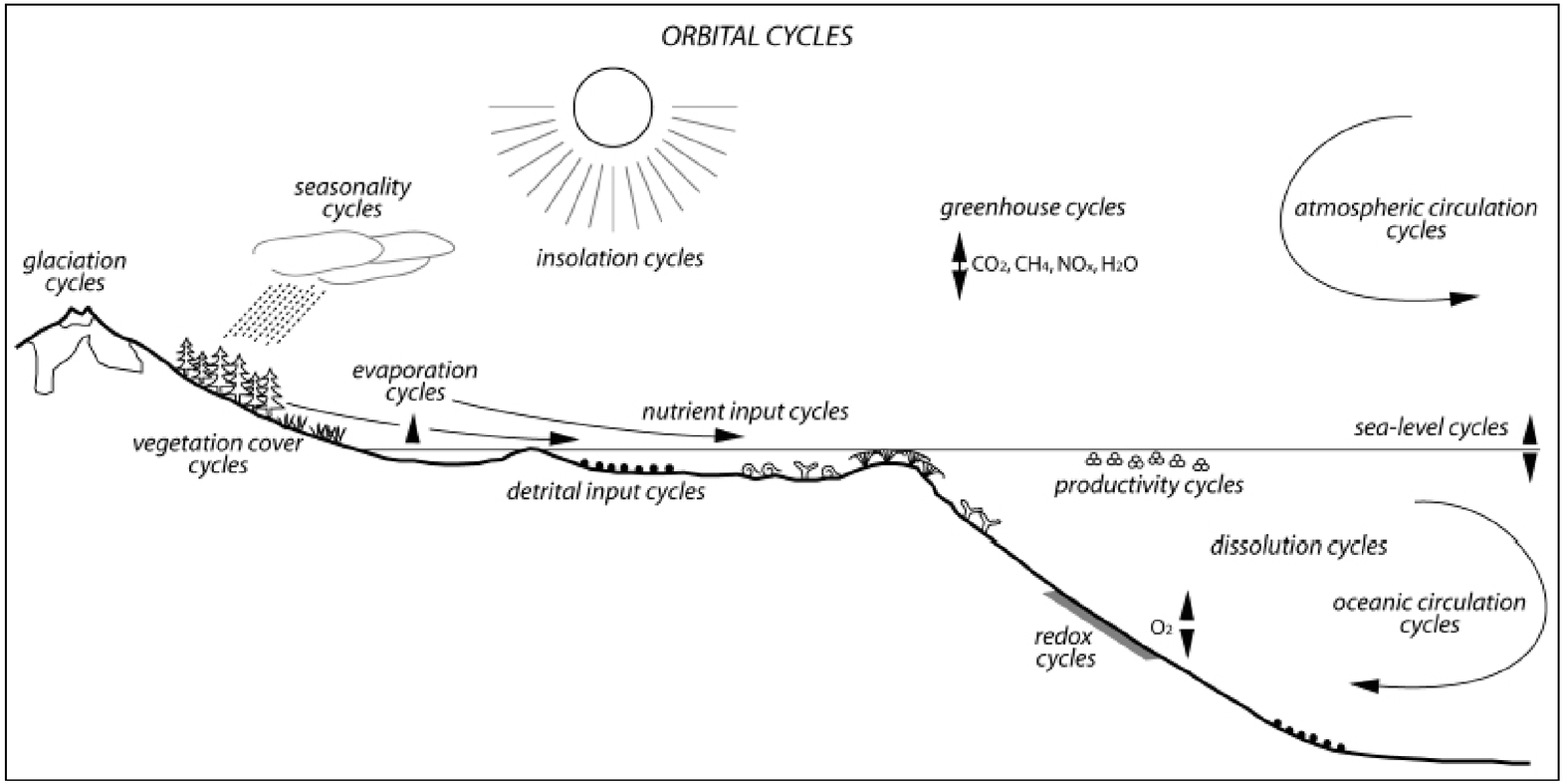
26,000-year cycles

climate.nasa.gov


Changes in Obliquity (Tilt)
41,000-year cycles
Relative to orbital plane
22.1-24.5°
climate.nasa.gov

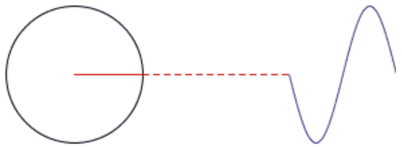

40
30
20
Obliquity
Precession
40
25
20
periods in kyr
400
300
200
100
0
myr before present


Eccentricity
Obliquity
Precession
100
200
300
400
500
600
700
800
900
1000
kyr before present
Caloric summer northern hemisphere
solar radiation deviation from 1950
(cal per cm2 and day)
60
40
20
0
-20
-40
0
50
100
150
200
250
kyr before present
80°N
65°N
10°N

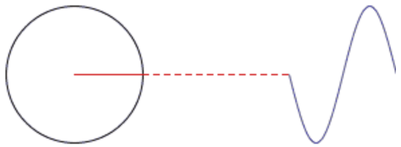

ORBITAL CYCLES
seasonality cycles
insolation cycles
greenhouse cycles
CO_2, CH_4, NO_x, H_2O
atmospheric circulation cycles
glaciation cycles
evaporation cycles
nutrient input cycles
sea-level cycles
vegetation cover cycles
detrital input cycles
productivity cycles
dissolution cycles
oceanic circulation cycles
redox cycles
O_2

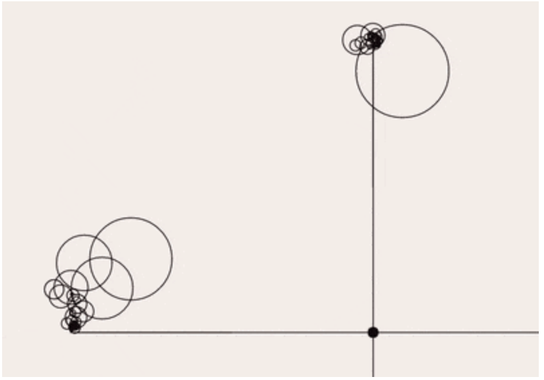
Information can be described as an infinite series of sin and cos waves of differing frequencies and magnitudes.

Information can be described as an infinite series of sin and cos waves of differing frequencies and magnitudes.



Information can be described as an infinite series of sin and cos waves of differing frequencies and magnitudes.





Euler's formula establishes the fundamental relationship between the trigonometric functions and the complex exponential function.

$$e^{ix} = cosx + i\ sinx$$

"the most remarkable formula in mathematics" - Richard Feynman

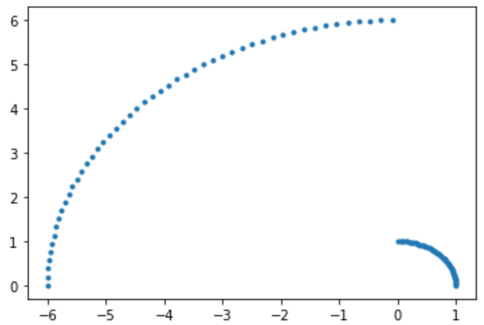
In [9]:
```
np.exp(1j*3)
```

Out[9]:
```
(-0.9899924966004454+0.1411200080598672j)
```

In [20]:

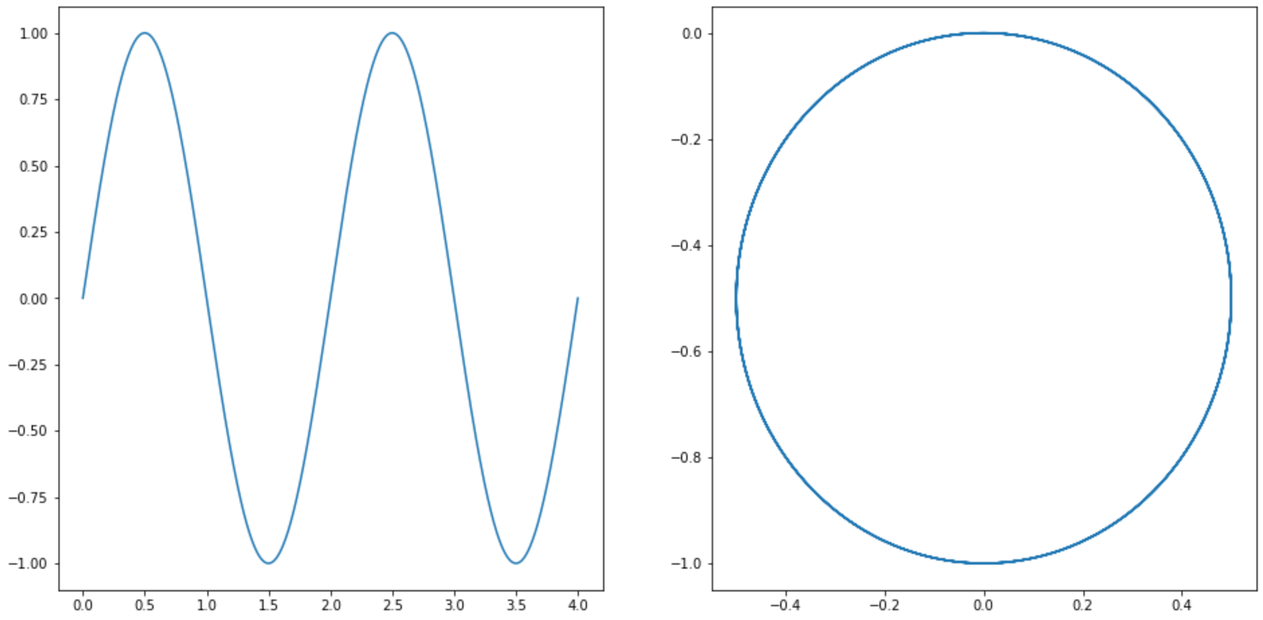
```
x=np.linspace(0,2*np.pi,100)
y=np.ones_like(x)
y[x>np.pi]=6
C=y*np.exp(1j*x/2)
plt.plot(C.real,C.imag,'.')
```

Out[20]: [<matplotlib.lines.Line2D at 0x7ffa97a744f0>]



In [25]:
```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
xt = np.linspace(0, 4, 1000)
y = np.sin(xt*np.pi)
plt.plot(xt, y)
plt.subplot(1,2,2)
freq=1/2
complex_y = y * np.exp(-2 * np.pi * 1j * freq    * xt) #using imaginary j  and exp to convert to polar
plt.plot(complex_y.real, complex_y.imag)
```

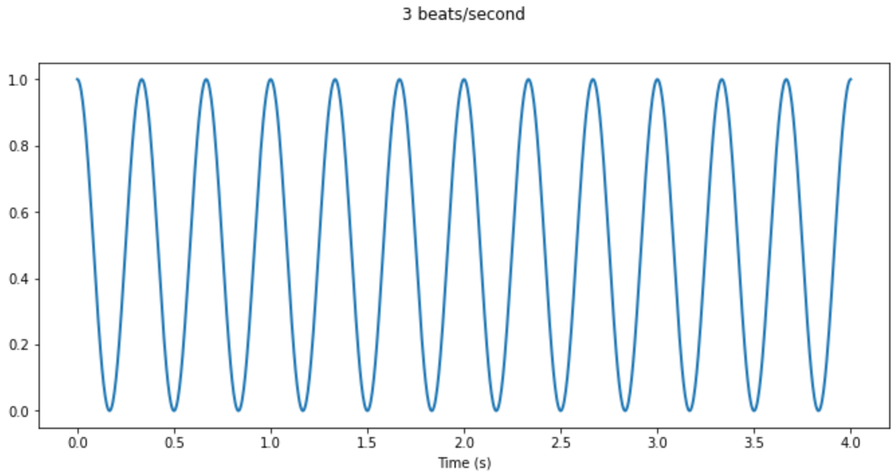
Out[25]:
```
[<matplotlib.lines.Line2D at 0x7ffa95c7fb80>]
```

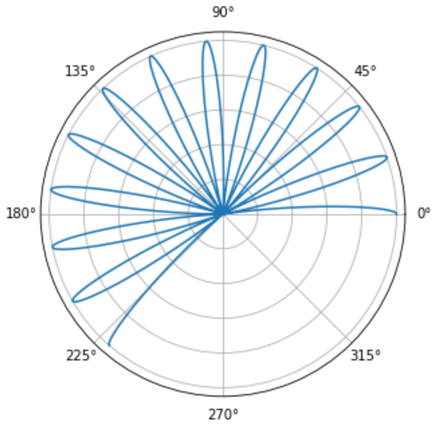


In [27]:

```
plt.figure(figsize=(25,5))
plt.subplot(1,2,1)
xt = np.linspace(0,4,1000)
y = (np.cos(xt*3*2*np.pi)+1)/2
plt.plot(xt,y,lw=2)
plt.gca().set_title('3 beats/second',y=1.1)
plt.gca().set_xlabel('Time (s)')
plt.subplot(1,2,2, polar=True)
_=plt.gca().set_yticklabels([])
plt.plot(xt,y)
```

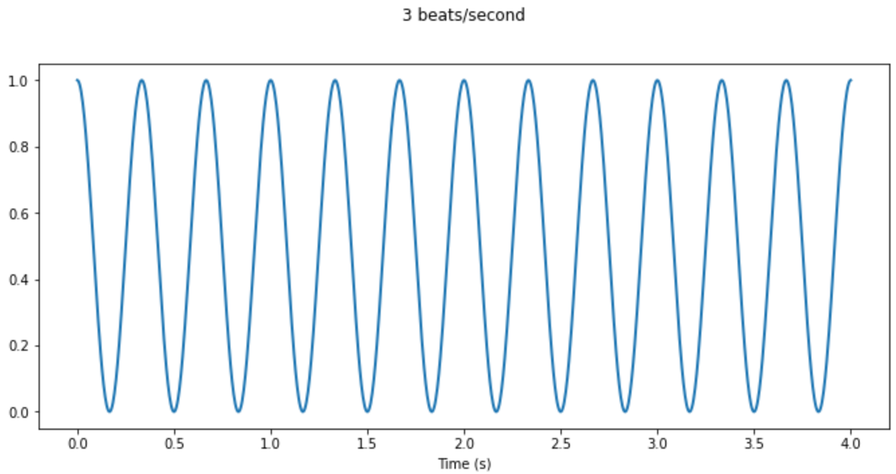
Out[27]: [<matplotlib.lines.Line2D at 0x7ffa95b0d370>]

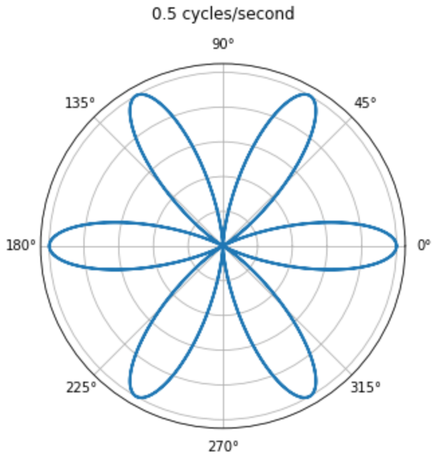




In [28]:

```
plt.figure(figsize=(25,5))
plt.subplot(1,2,1)
xt = np.linspace(0,4,1000)
y = (np.cos(xt*3*2*np.pi)+1)/2
plt.plot(xt,y,lw=2)
plt.gca().set_title('3 beats/second',y=1.1)
plt.gca().set_xlabel('Time (s)')
plt.subplot(1,2,2, polar=True)
plt.plot(xt*.5*2*np.pi,y,alpha=1,lw=2)
plt.gca().set_yticklabels([])
_=plt.gca().set_title('0.5 cycles/second',y=1.1)
```





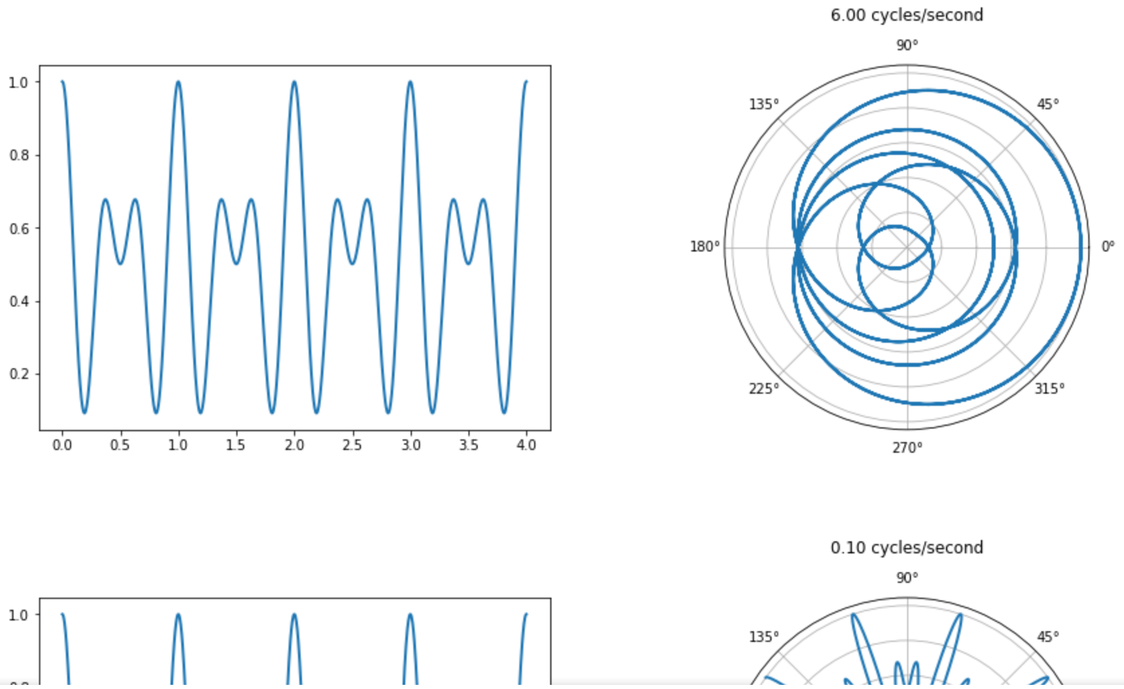
Some set up to cycle through different frequencies:

In [29]:

```
%%capture
import time
import pylab as pl
from IPython import display
xt = np.linspace(0,4,1000)
y = (np.cos(xt*3*2*np.pi)+1)/2
y = (np.cos(xt*3*2*np.pi)+1)/4+(np.cos(xt*2*2*np.pi)+1)/4
plt.gca().set_title('3 beats/second',y=1.1)
plt.gca().set_xlabel('Time (s)')
```

Wrapping the signal (left) around the unit circle (right) at different frequencies:

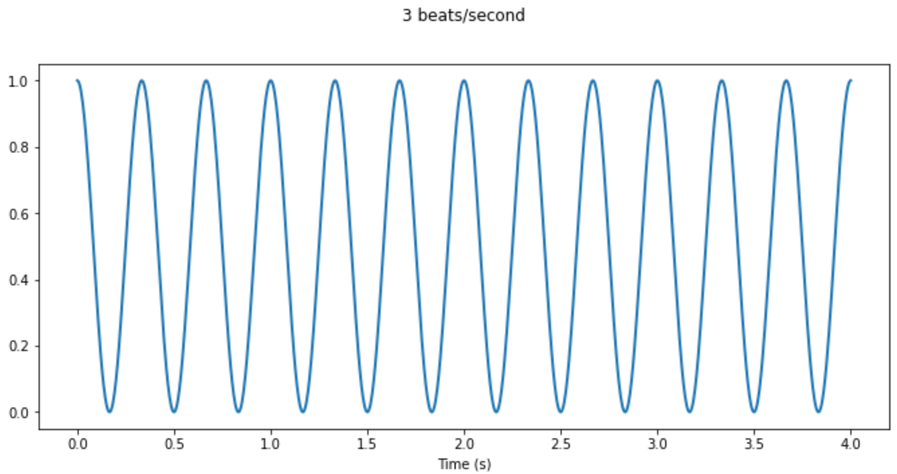
In [32]:
```
for i in np.linspace(.1,6,100):
    fig=plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    plt.plot(xt,y,lw=2)
    plt.subplot(1,2,2, polar=True)
    display.clear_output(wait=True)
    plt.plot(xt*i*2*np.pi,y,alpha=1,lw=2)
    plt.gca().set_yticklabels([])
    _=plt.gca().set_title(f'{i:.2f} cycles/second',y=1.1)
    display.display(pl.gcf())
    time.sleep(.25)
```

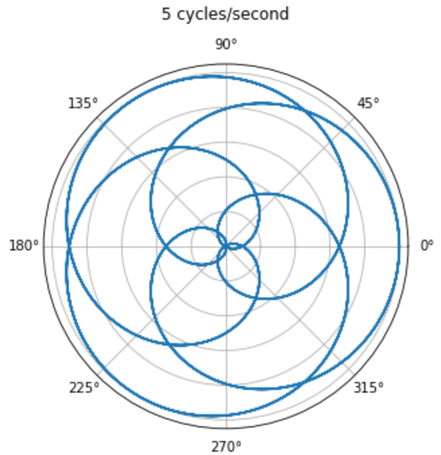


In [78]:

```
plt.figure(figsize=(25,5))
plt.subplot(1,2,1)
xt = np.linspace(0,4,1000)
y = (np.cos(xt*3*2*np.pi)+1)/2
plt.plot(xt,y,lw=2)
plt.gca().set_title('3 beats/second',y=1.1)
plt.gca().set_xlabel('Time (s)')
plt.subplot(1,2,2, polar=True)
# plt.plot(xt*.5*2*np.pi,y,alpha=1)
plt.plot(xt*5*2*np.pi,y)
plt.gca().set_yticklabels([])
plt.gca().set_title('5 cycles/second',y=1.1)
```

Out[78]: Text(0.5, 1.1, '5 cycles/second')

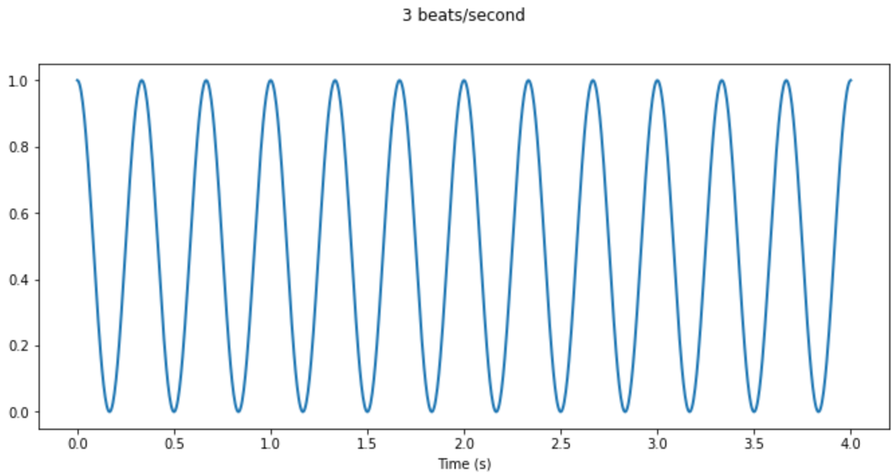


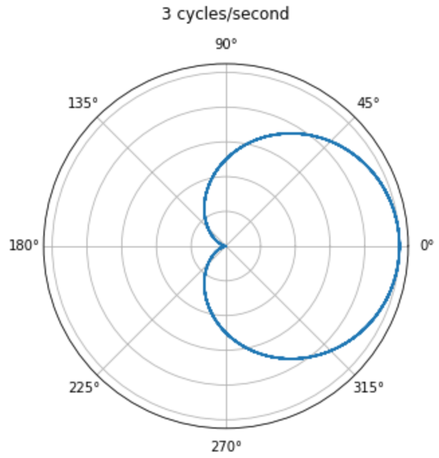


In [79]:

```
plt.figure(figsize=(25,5))
plt.subplot(1,2,1)
xt = np.linspace(0,4,1000)
y = (np.cos(xt*3*2*np.pi)+1)/2
plt.plot(xt,y,lw=2)
plt.gca().set_title('3 beats/second',y=1.1)
plt.gca().set_xlabel('Time (s)')
plt.subplot(1,2,2, polar=True)
# plt.plot(xt*.5*2*np.pi,y,alpha=1) #add back
# plt.plot(xt*5*2*np.pi,y,alpha=1)
plt.plot(xt*3*2*np.pi,y)
plt.gca().set_yticklabels([])
plt.gca().set_title('3 cycles/second',y=1.1)
```

Out[79]: Text(0.5, 1.1, '3 cycles/second')

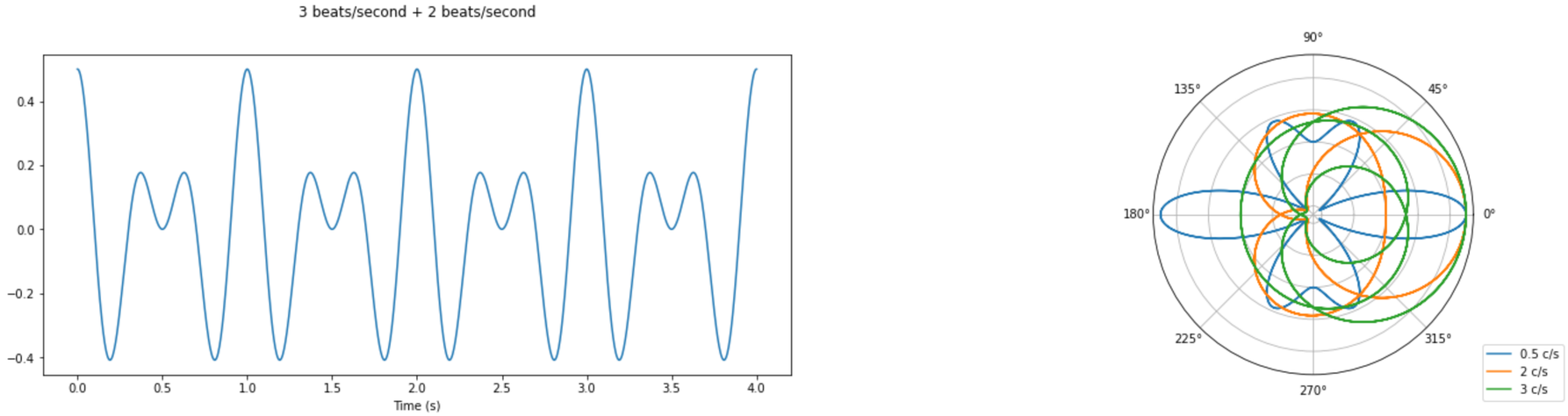




In [103]:

```
plt.figure(figsize=(25,5))
plt.subplot(1,2,1)
xt = np.linspace(0,4,1000)
y = (np.cos(xt * 3 * 2 * np.pi) + 1) / 4 + (np.cos(xt * 2 * 2 * np.pi) + 1) / 4 - .5
plt.plot(xt,y)
plt.gca().set_title('3 beats/second + 2 beats/second',y=1.1)
plt.gca().set_xlabel('Time (s)')
plt.subplot(1,2,2, polar=True)
plt.plot(2 * np.pi * 0.5 * xt,y,label='0.5 c/s')
plt.plot(2 * np.pi * 2 * xt,y,label='2 c/s')
plt.plot(2 * np.pi * 3 * xt,y,label='3 c/s')
plt.gca().set_yticklabels([])
plt.legend(loc='lower right',fontsize=10,bbox_to_anchor=(1.3, -.1))
# plt.gca().set_title('3 cycles/second',y=1.1)
```

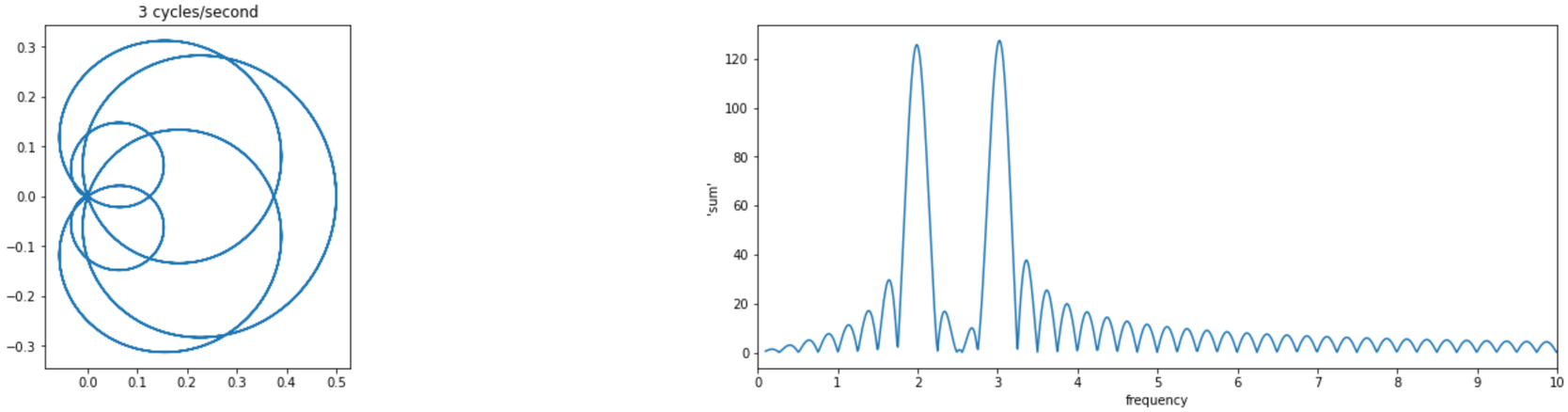
Out[103]: <matplotlib.legend.Legend at 0x7f780e399fd0>



In [104]:

```
plt.figure(figsize=(25, 5))
plt.subplot(1, 2, 1)
xt = np.linspace(0, 4, 1000)
y = (np.cos(xt * 3 * 2 * np.pi) + 1) / 4 + (np.cos(xt * 2 * 2 * np.pi) + 1) / 4 - .5
complex_y = y * np.exp(-2 * np.pi * 1j * 3 * xt) #using imaginary j  and exp to convert to polar
plt.plot(complex_y.real, complex_y.imag)
plt.gca().set_aspect(1); plt.gca().set_title("3 cycles/second"); plt.subplot(1, 2, 2)
x_COM = []
freqs = np.linspace(0.1, 10, 1000)
# freqs = np.arange(0,1000-1)
for i in freqs:
    complex_y = y * np.exp(-2 * np.pi * 1j * i * xt)
    x_COM.append((np.sum(complex_y.real)**2+np.sum(complex_y.imag)**2)**(1/2))

plt.plot(freqs, x_COM,alpha=1,zorder=2); _ = plt.gca().set_xticks(range(0, 11)); plt.gca().set_ylabel("'sum'"); plt.gca().set_xlab
plt.gca().set_xlim([0,10])
```

Out[104]: (0.0, 10.0)



In [ ]: