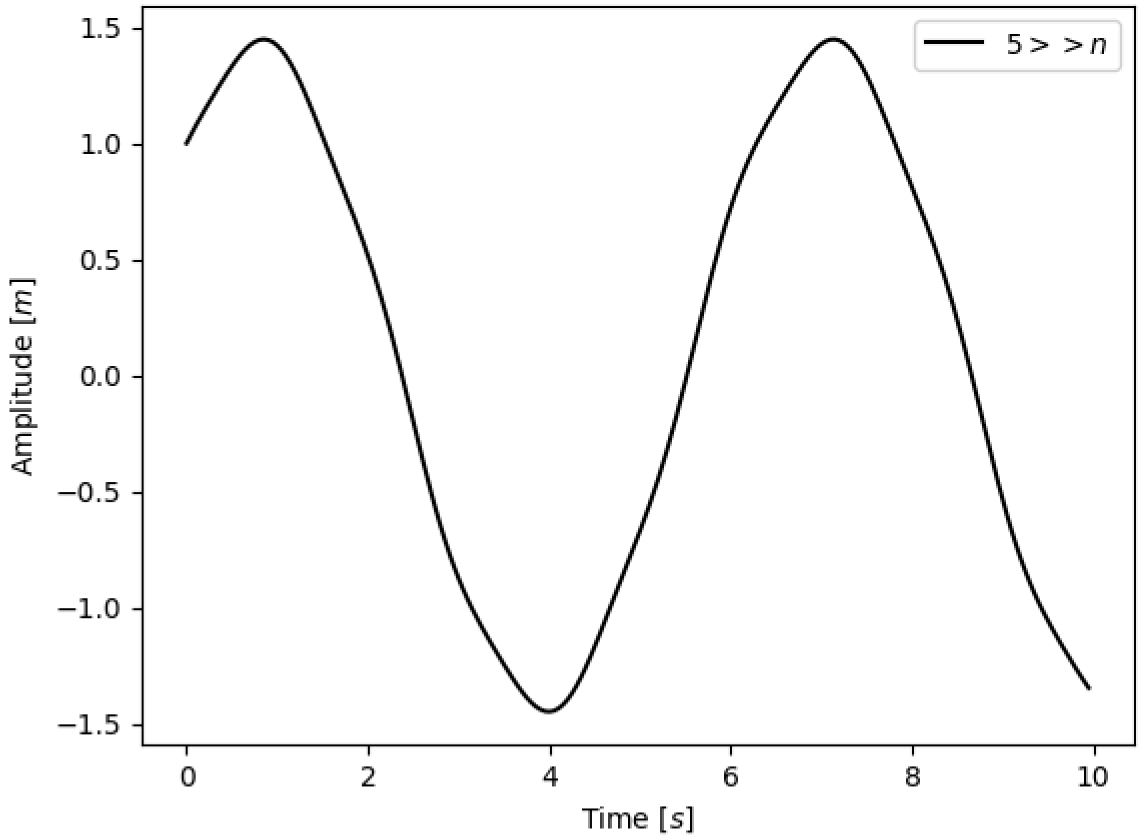
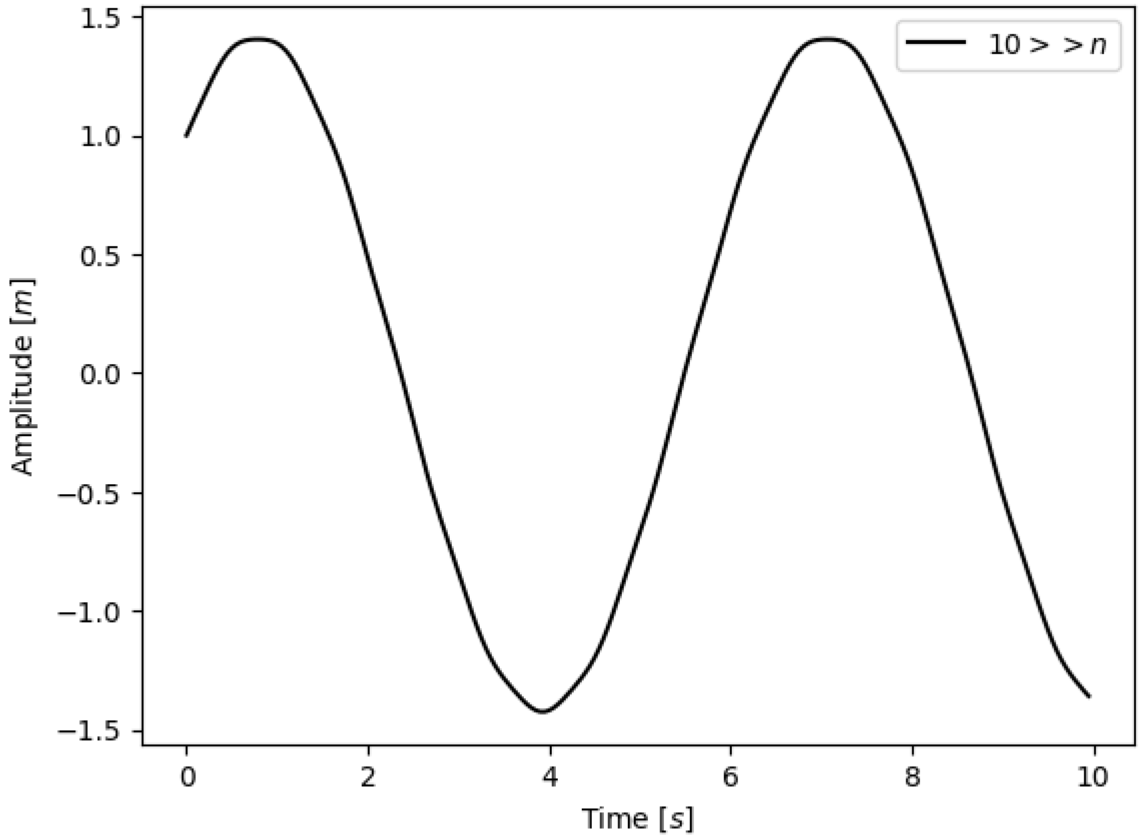
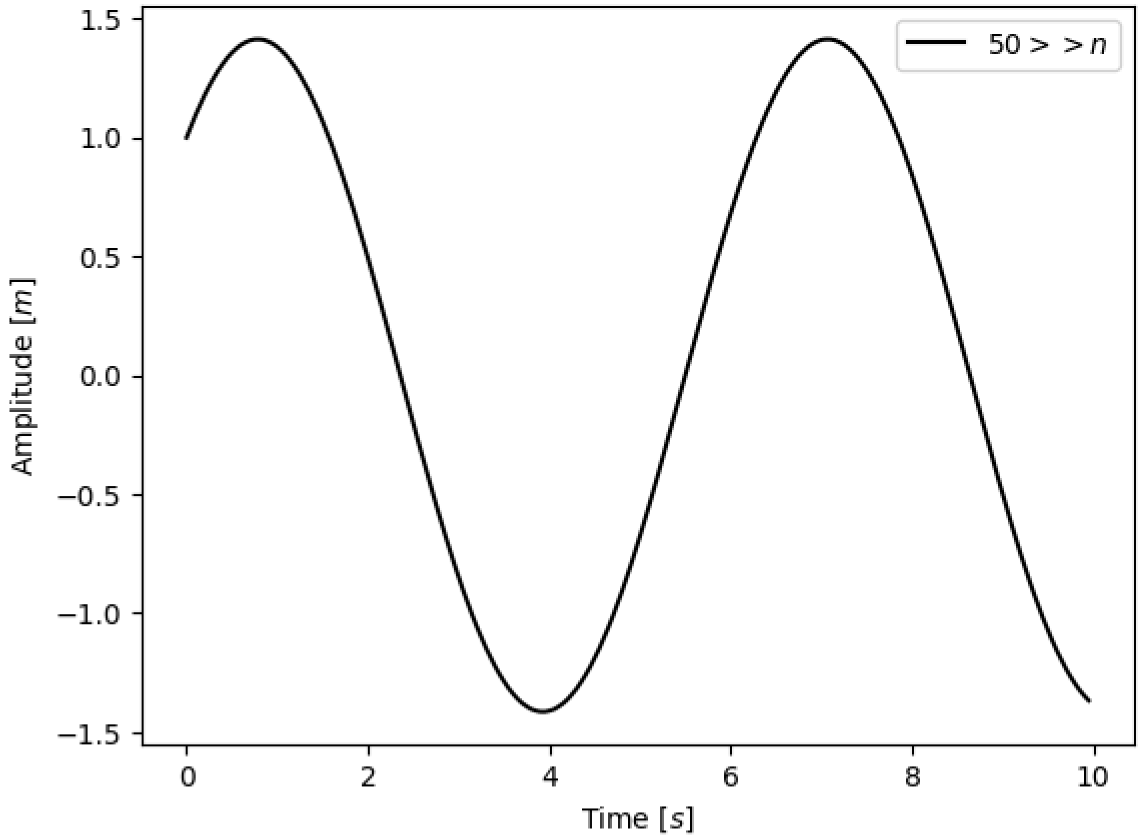
In [ ]:

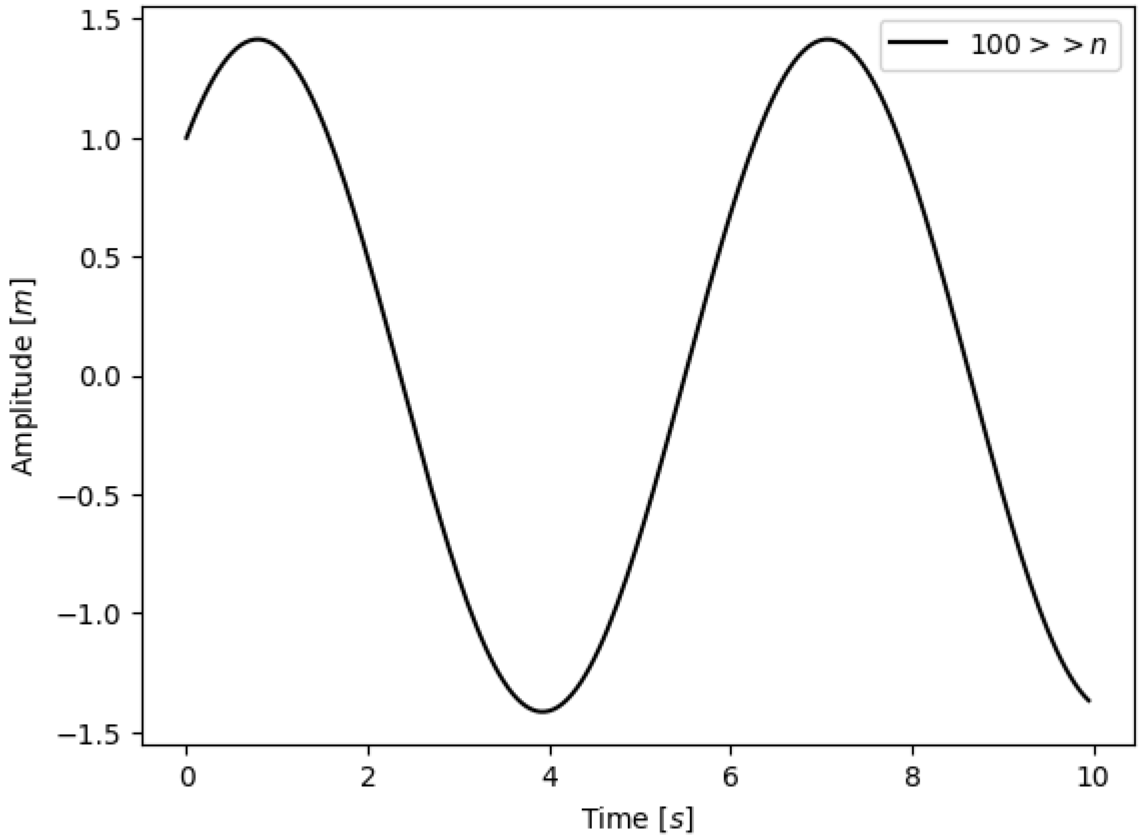
```python
"""
June 22, 2025
@author Luke Voinov

Numerical exploration of a resonance equation

2nd order differential equation: d^2x/dt^2 + (n^2)x = csin(wt) where n = sqrt(k/m) and is the natural freq. of a spring oscillator

Solution: x(t) = a*cos(nt) + b*sin(nt) + ( c / (n^2 - w^2) ) * sin(wt)
"""
import matplotlib.pyplot as plt
import numpy as np

# Case I: w >> n. Let c = 1 (high freq. forcing), n = 1, dx(0)/dt = 0, w varies.

#set constants
a = 1
b = 1
c = 1
n = 1
dx0_dt = 0
W = np.array([5,10,50,100]) #Hz
smooth = np.arange(0,10,0.05)

for w in W:
    x = lambda t : a*np.cos(n*t) + b*np.sin(n*t) + (c / (n**2 - w**2)) * np.sin(w*t)
    time = []
    f = []
    for t in smooth: # t in seconds
        time = np.append(time, t) # 1xm
        f = np.append(f, x(t)) # 1xm

    plt.figure()
    plt.plot(time, f, 'k-', label=f"${w} >> n$")
    plt.legend()
    plt.xlabel('Time $[s]$')
    plt.ylabel('Amplitude $[m]$')

"""
Notes:

The graphs generally look similar. Only at w = 2 do we see some sort of distortion, but 2 isn't that much larger than 1. Once we g
This is due to the forcing averaging out at higher frequencies, thus no longer affecting the intrinsic frequency.
"""
```


5 > > n
Amplitude [m]
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
2
4
6
8
10
Time [s]



10 > > n
Amplitude [m]
Time [s]
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
2
4
6
8
10


50 > > n
Amplitude [m]
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
2
4
6
8
10
Time [s]

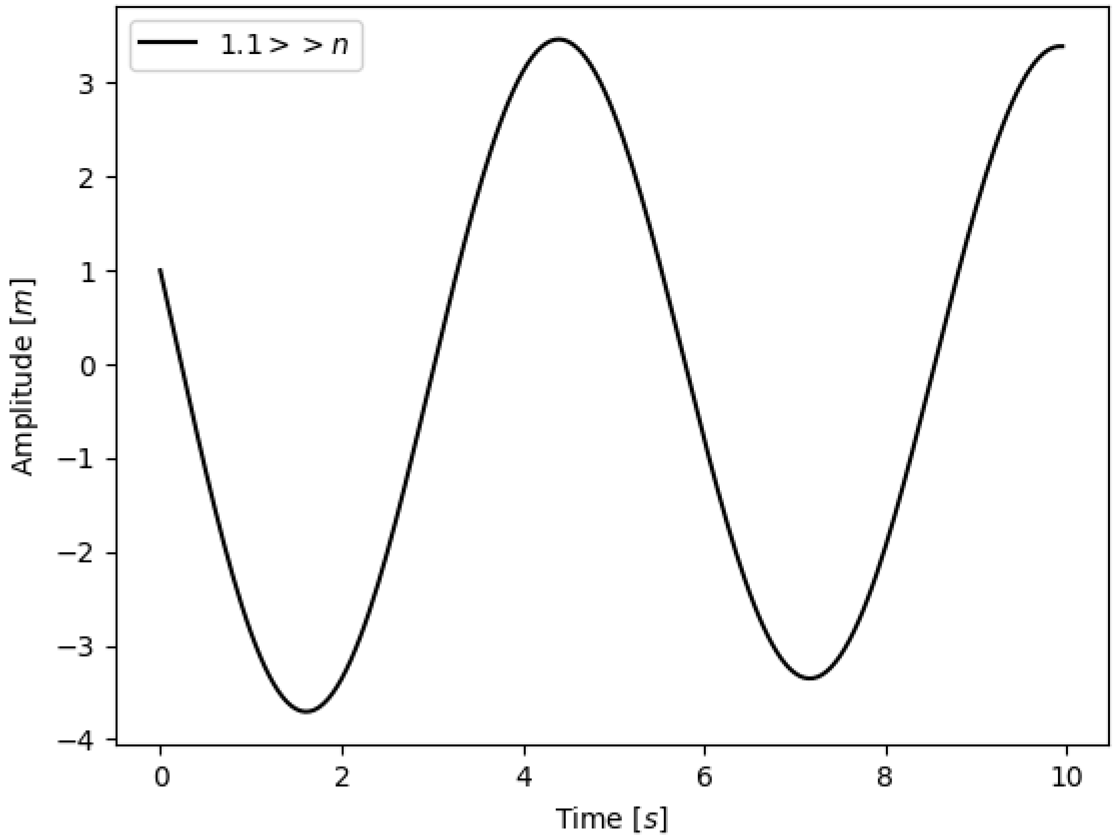


In [ ]:

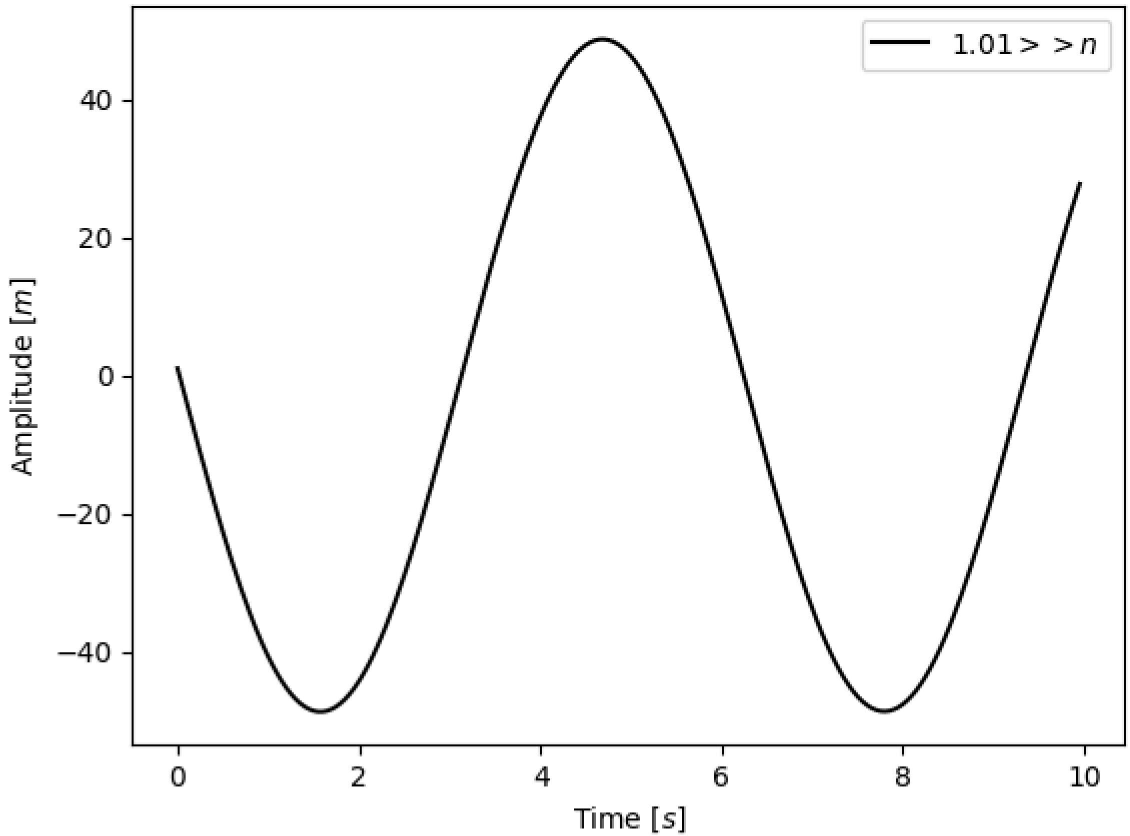
```
# Case II; Resonance: w = n = 1
"""
June 22, 2025
@author Luke Voinov

Numerical exploration of a resonance equation

2nd order differential equation: d^2x/dt^2 + (n^2)x = csin(wt) where n = sqrt(k/m) and is the natural freq. of a spring oscillator

Solution: x(t) = a*cos(nt) + b*sin(nt) + ( c / (n^2 - w^2) ) * sin(wt)
"""
import matplotlib.pyplot as plt
import numpy as np

#set constants
a = 1
b = 1
c = 1
n = 1
dx0_dt = 0
W = np.array([1.1, 1.01, 1.001, n]) #Hz
```

```python
smooth = np.arange(0,10,0.05)

for w in W:
    x = lambda t : a*np.cos(n*t) + b*np.sin(n*t) + (c / (n**2 - w**2)) * np.sin(w*t)
    time = []
    f = []
    for t in smooth: # t in seconds
        time = np.append(time, t) # 1xm
        f = np.append(f, x(t)) # 1xm

    plt.figure()
    plt.plot(time, f, 'k-', label=f"${w} >> n$")
    plt.legend()
    plt.xlabel('Time $[s]$')
    plt.ylabel('Amplitude $[m]$')


"""
Notes:

The amplitude approached infinity as w approaches n. At n = w, the plot DNE (because we would be divind by 0). This is because res

For n = w = 1:
RuntimeWarning: divide by zero encountered in scalar divide
"""
```

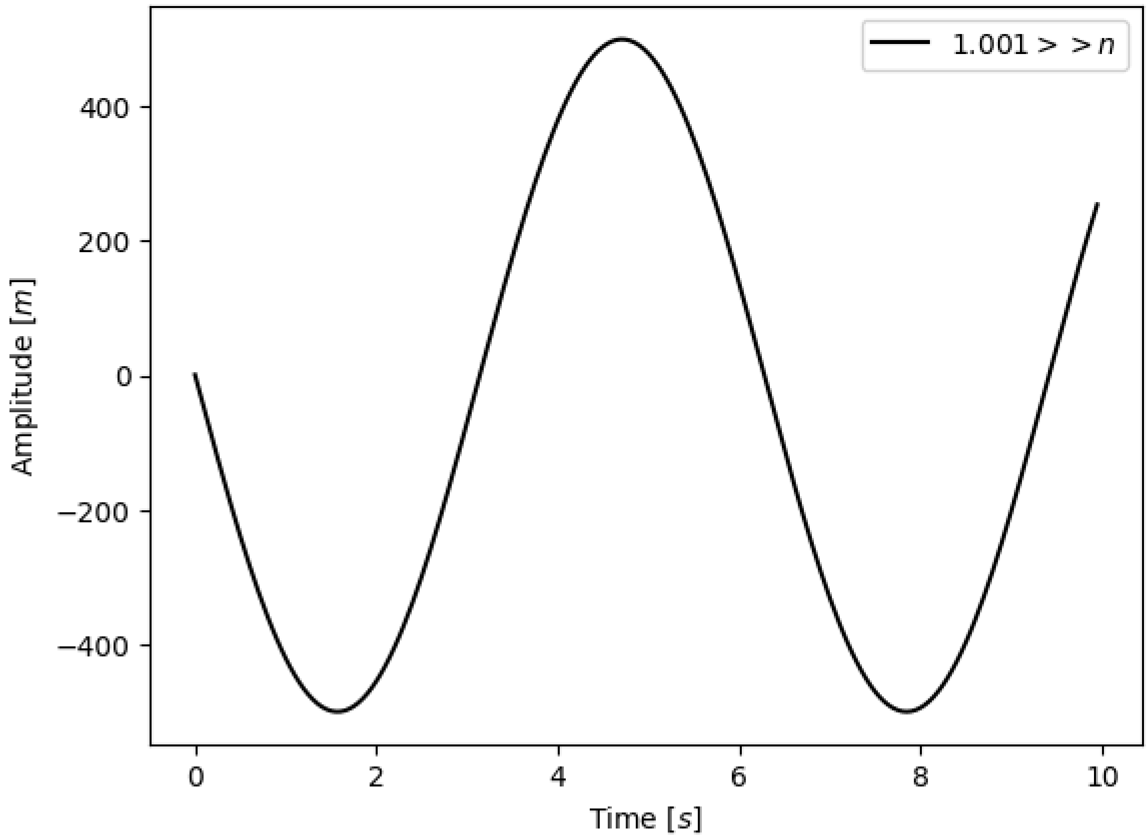
```
C:\Users\lukev\AppData\Local\Temp\ipykernel_29016\2344595110.py:25: RuntimeWarning: divide by zero encountered in scalar divide
  x = lambda t : a*np.cos(n*t) + b*np.sin(n*t) + (c / (n**2 - w**2)) * np.sin(w*t)
C:\Users\lukev\AppData\Local\Temp\ipykernel_29016\2344595110.py:25: RuntimeWarning: invalid value encountered in scalar multiply
  x = lambda t : a*np.cos(n*t) + b*np.sin(n*t) + (c / (n**2 - w**2)) * np.sin(w*t)
```



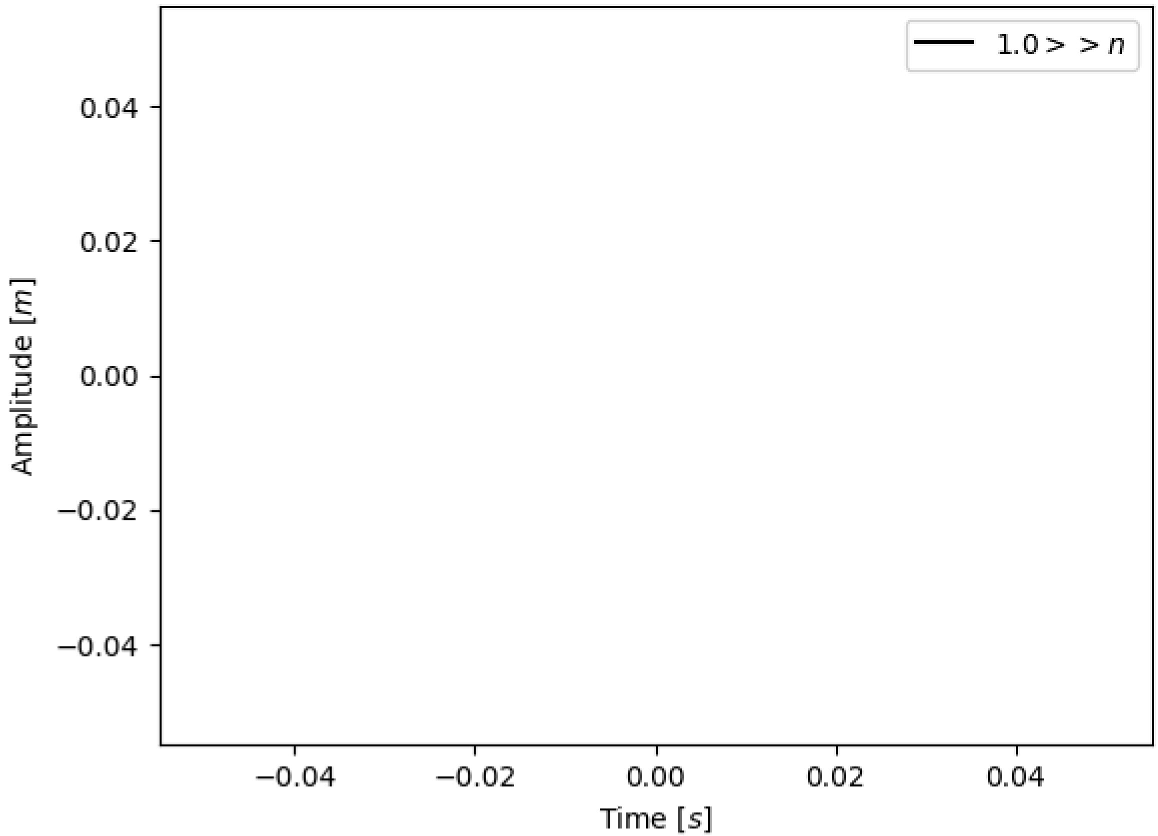
$1.1 >> n$
Amplitude $[m]$
Time $[s]$



$1.01 >> n$
40
20
0
−20
−40
Amplitude $[m]$
0
2
4
6
8
10
Time $[s]$



1.001 > > n
Amplitude [m]
Time [s]


$1.0 >> n$
0.04
0.02
0.00
−0.02
−0.04
Amplitude $[m]$
−0.04
−0.02
0.00
0.02
0.04
Time $[s]$