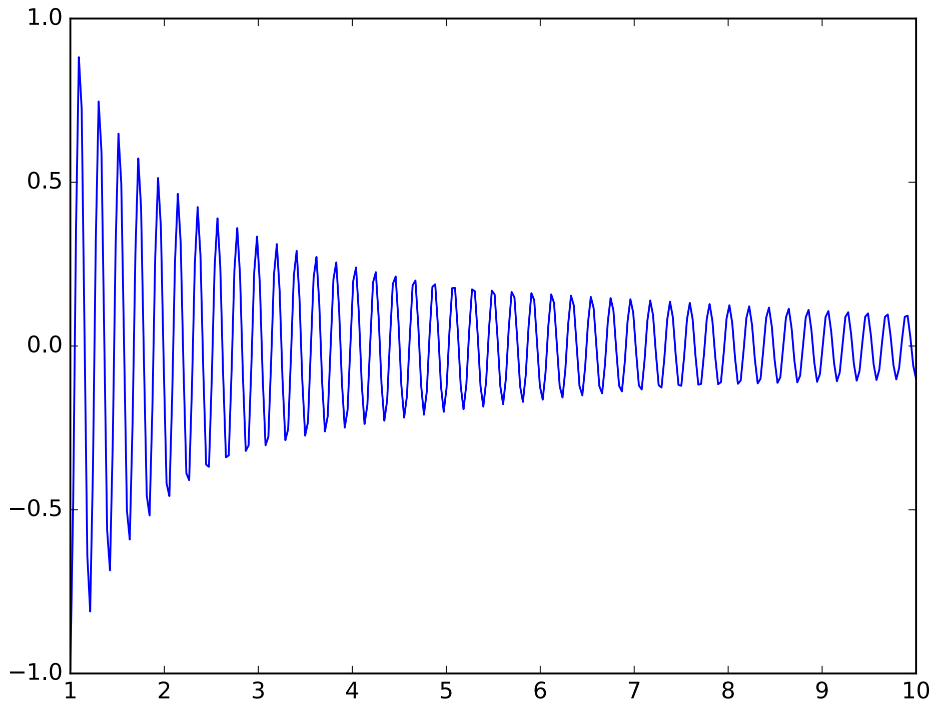
The importance of time sampling with oscillatory phenomena:

Dealing with oscillatory phenomena numerically sometimes leads to unexpected surprises. Problems happen when you are trying to plot an oscillating function with an insufficient number of points where the number of points per period is close to inverse integer of the period of the function. This is a sampling problem, which can take various forms in applied physics, signal analysis, etc…How to properly sample a function is a central question in many (if not all) practical aspect of experimental science. Some general discussion on issues with sampling can be found there:

https://en.wikipedia.org/wiki/Sampling_(signal_processing)

To make this clear with a practical example, imagine you want to plot the function `sin(30x)/x`:

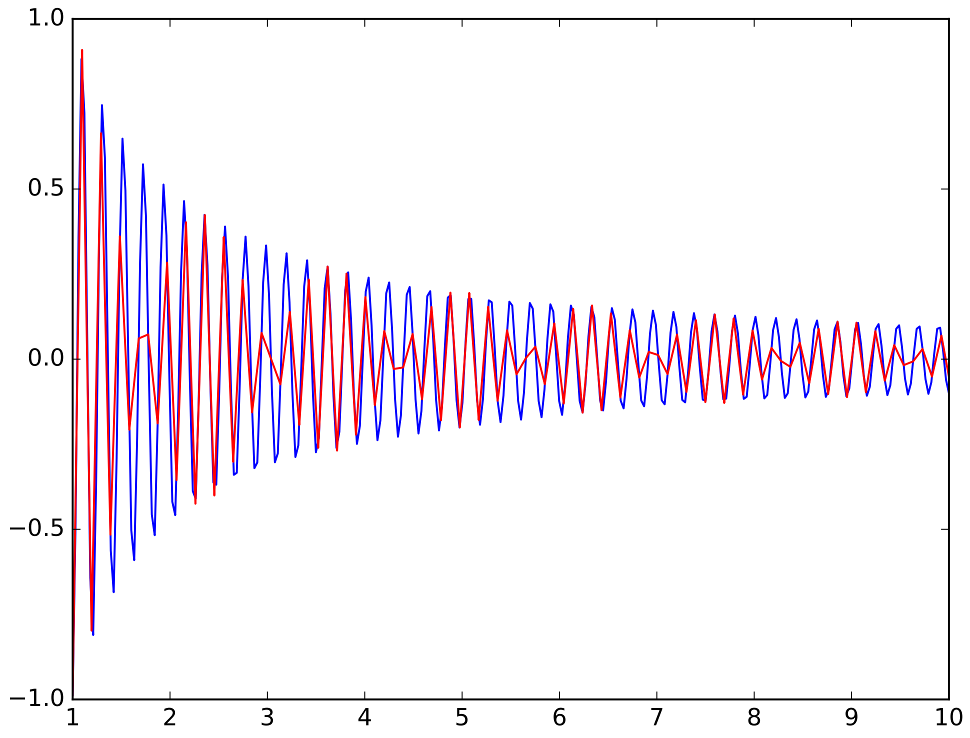
```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(1,10,300)
plt.plot(x,np.sin(30.*x)/x)
plt.show()
```



The period is $2\pi/30\sim0.209$. Now imagine I plot the same function with the following sampling:

```
x2=np.linspace(1,30,300)
plt.plot(x,np.sin(30.*x)/x)
plt.plot(x2,np.sin(30.*x2)/x2,'r')
plt.xlim([1,10])
```

The sampling period in `x2` is `0.1`, that is close to half the function period. You'll get this artificial "beat" (in red)



It is important to be aware of this potential problem when plotting highly oscillating functions. This is also known as "aliasing" problem which occurs when you under-sample a high frequency wave and misinterpret it as a longer wavelength. The concept of Nyquist frequency is particularly important:

https://en.wikipedia.org/wiki/Nyquist_frequency

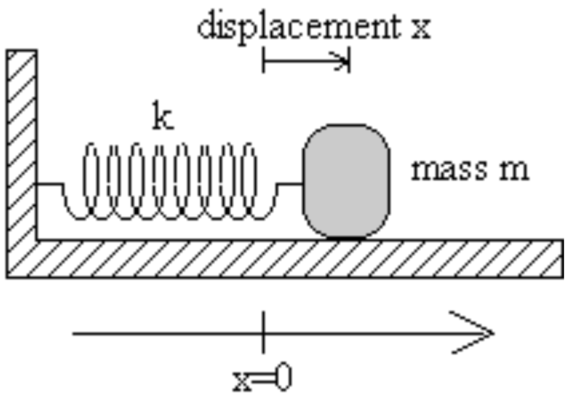
Project 3 (Due date Tuesday November 14th 12p.m.)

For this project you will study the resonance of a damped and forced harmonic oscillator. Some of you may have studied resonance with LRC (electronic) systems and/or simple harmonic oscillators. It is ok if you have not. You can (should!) read more about it on this page:

https://en.wikipedia.org/wiki/Mechanical_resonance

Basically, an oscillator has a "natural" frequency, and when the driven force frequency is close to the natural frequency, the system enters is a state of greater amplitude than for any other driven force frequency. In this project you will reproduce this experiment numerically.

Consider a mass `m` attached to a spring of stiffness `k`:



The motion of the mass `m` is subject to a damping force -c **x** and a harmonic driving force F_0 cos(ωt). The equation of motion is given by:

$$m\frac{\mathrm{d}^2x}{\mathrm{dt}^2} + c\frac{\mathrm{d}x}{\mathrm{dt}} + kx = F_0\cos(\omega t)$$

The system can be described by two characteristic frequencies: the natural angular frequency of the spring ω_0:

$$\omega_0 = \sqrt{\frac{k}{m}}$$

and the driven force angular frequency ω.

Objective: You will write a code **`oscillations.py`** which accepts the values of m, k, c and F_0 as user inputs and solves numerically the differential equation above using **`odeint()`**. Your code should generate a resonance plot called **`resonance.pdf`** and three plots showing the solution `x(t)` for difference oscillation regimes (see below for details) on the same plot called **`oscillations.pdf`**.

The **`resonance.pdf`** plot should show the value of the maximum of the amplitude at large time (i.e. once the system has stabilized) $A=\max[x(t \rightarrow \infty)]$, as function of the driving force angular frequency ω. The curve should look like one of those in the plot from the Wikipedia link above. Remember that the user is free to choose the values of m, k, c and F_0.

The plot **`oscillations.pdf`** should contain 3 panels, each panel will show the solution `x(t)` for a given value of the driven force angular frequency ω. For the values you will choose, one should be *exactly* at the resonance frequency ω_0, one should be twice this value, one should be half this value. The solutions `x(t)` should show at least 20 oscillations.

Tips: no major difficulties in this project, but you have to make sure you choose your time interval and time steps wisely to make sure you capture the whole behavior of the oscillator (sampling). The main difficulty is to write a program which accepts any values of the input parameters given by the user and still generate plots showing the requested features.