Contents

-*- mode: org-mode

1 Exercise 2. Solving ODEs.

1.1 Structure of the Solution

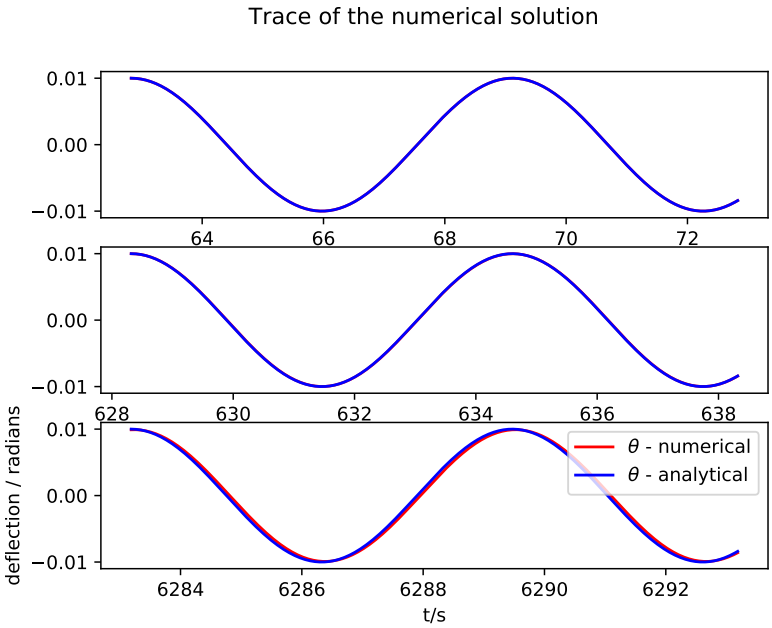
The solution is split between in two. `MechanicalSystem.py`, defines a class `MechanicalSystem`, which can be used to describe a general system described by a single 2-nd order ODE. `Pendulum` is a subclass that describes the exact system given in the practical booklet.

`investigator.py` contains all of the solution code, i.e. one that instantiates the simulations and generates the respective plots, which can be found in the sub-directory `figures/`.

1.2 Comparing analytical and Numerical answers.

1.2.1 Eventual divergence of values.

Consider the plot `comparison with analytial - deflection.pdf`



Notice, that the solutions are near indistinguishable for the first few cycles of simulation, and only in the last two cycles can we see a noticeable difference.

As expected, the simulated solution falls behind the expected analytical one. This may have two causes:

1. The expected analytical solution is only valid for small deflections, hence the numerical solution contains more terms that over time contribute to a difference.

2. The numerical solution accumulates roundoff errors that over time create the discrepancy.

To distinguish between the two, one can simply reduce the initial deflection. If the real cause is the former, the solutions will diverge later, as the higher order terms will become even smaller. However if it's the latter, the solutions will diverge sooner, as the reduced dynamic range will exasperate the floating-point errors. My analysis has shown that the first is the more likely culprit.

To convince ourselves, let's estimate the errors in replacing the sine with the argument and the average roundoff error in each step.

The error in the first approximation is:

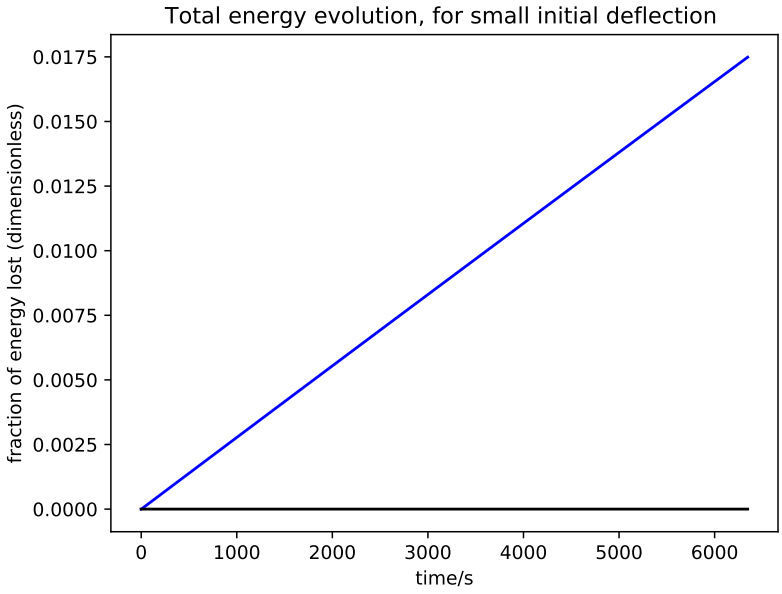
$$\sin(0.01) - 0.01 = 1.67 \times 10^{-7}$$

The roundoff error stems from having to evaluate sines and cosines of of small angles. We would expect that given a sampling rate of 500Hz, that the roundoff error would be of the order of

$$\sin(0.01 + \frac{1}{500}) - \sin(0.01)\cos(\frac{1}{500}) - \sin(\frac{1/500}{\cos}(0.01)) \approx 1.7 \times 10^{-18}$$

1.2.2 Loss of energy.

The next plot: `comparison with analytical - energy.pdf` showcases the energy evolution of our numerical solution, and the one we would expect analytically.



The comparison is normalised to the initial energy: it shows the losses due to discretisation errors as a fraction. One can see that the losses are significant: $\approx 2 lengthofthesimulation.$

One thing to note, is that the comparison is made to the initial value of energy and not the energy computed from the analytical solution for small angles. (see aside)

1. Aside: comparing with initial energy. A naive guess would be to assume that there is no distinction, as the analytical solution conserves energy. However that is not true due to the limited precision with which the mantissa of floating-point numbers is represented. thus the identity

$$\sin^2(\theta) + \cos^2(\theta) = 1$$

   Will not hold in general, and we will have energy losses due to the roundoff errors. Furthermore, the definition of total energy in the case of the analytical approximate solution isn't well-defined.

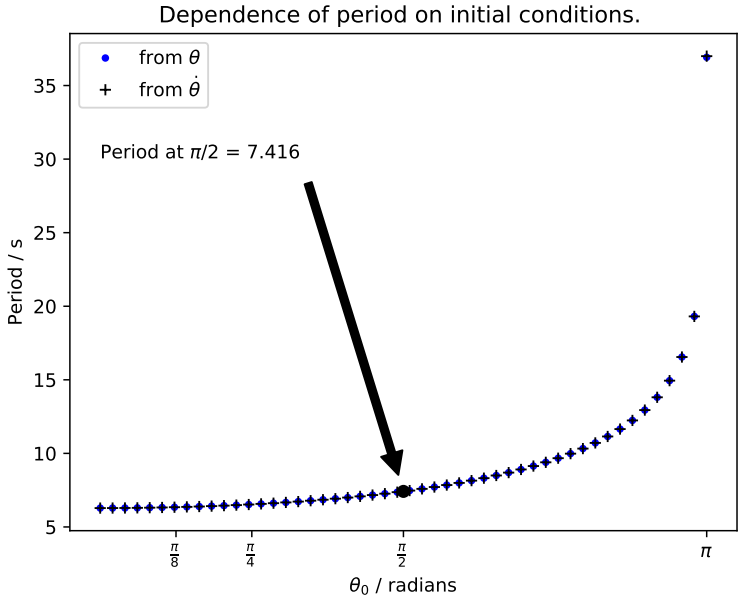
   We could use the exact expression for potential energy: $ V = 1 - cos θ $, but then the analytical approximate solution is paired with the exact expression that ignores the approximation.

   While we could then argue that using an approximate expression for energy would be more consistent, it is an even less accurate representation of the system's energy.

1.3 Estimating the Period as a function of initial amplitude

The Period of oscillations is measured by indexing the points where the value of deflection changes sign. This is done on the angular velocity and not the deflection, since the deflection itself can wrap around the pendulum, and overwrapping - i.e. subtracting a number of factors of π to bring it into the range needed for the algorithm will introduce additional roundoff errors.

This produces the plot `period_vs_amplitude.pdf`.



One can see a predictable divergence when the initial deflection reaches π, as the system would then be in unstable equilibrium.

We then obtain the value:

$$T\left(\frac{\pi}{2}\right) = 7.416\ s$$

1.4 Damping behaviour.

The values of $q = \frac{2}{}$

Quality factor correspond to light, critical and heavy damping, in the small angle approximation.

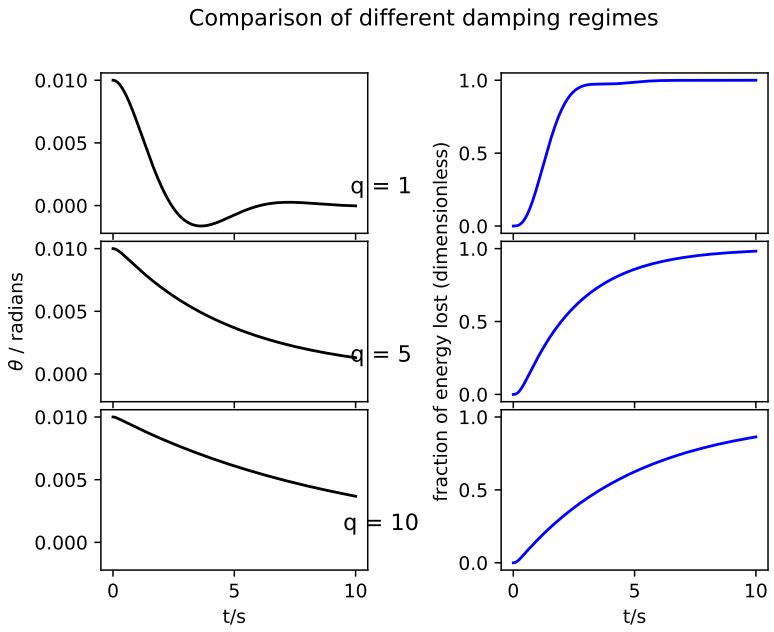
I use the term light critical and heavy damping loosely, as they don't necessarily mean the same for this system.

1. The system is described by a non-linear second order ODE. In this case, there might not be a special case for a particular value of Q-factor. Moreover, the critical to heavy damped solutions might not even be the fastest decaying.

2. Due to roundoff errors the system is always damped, even if the predicted quality factor should have been infinite. The correction is small,

but it has amplitude dependence making the quality factor also, amplitude dependent in a non-trivial way.

We can see that reflected in the plots in `damping.pdf`.

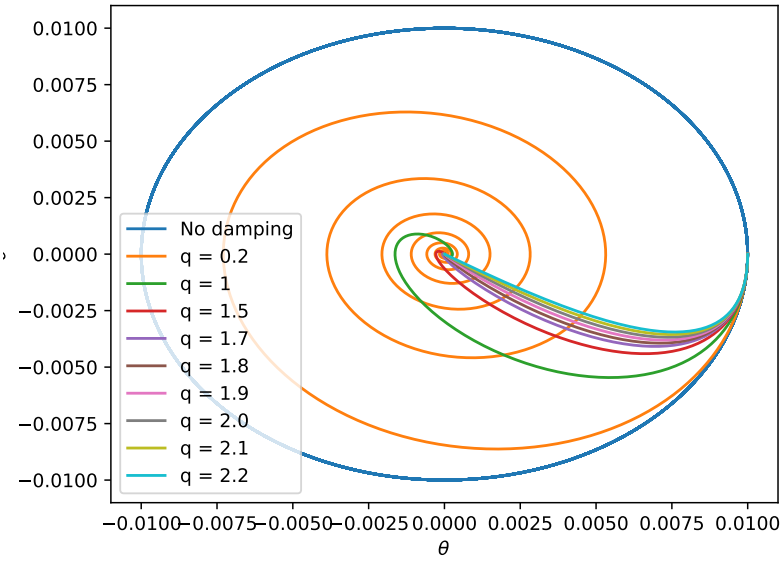


The first case, shows a decaying oscillation, with a quality factor of roughly 1. Interestingly, this regime also maximises the rate of energy dissipation. This further supports the points outlined above.

The second and third plot show a characteristic exponential decay, for Heavy and critical damping. As in the small angle case, the second plot decays to 0 much faster than the heavier-damped oscillator.

Another notable feature is the curvature in the energy plots. In the small angle approximation, we would expect there to be no points of inflection, and for the decay to be exponential at least in the heavy damped case.

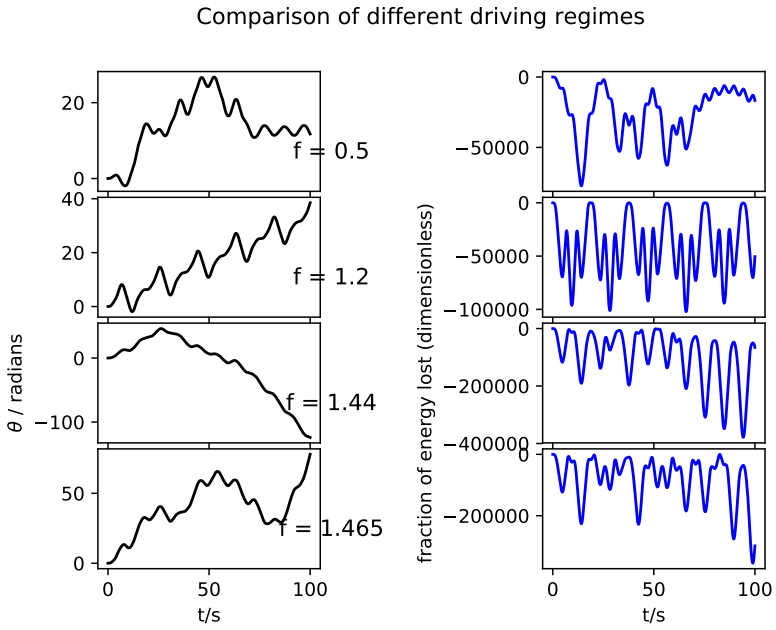
The same phenomena in phase space show nothing remarkable. (see `phase_space-damping.pdf`).

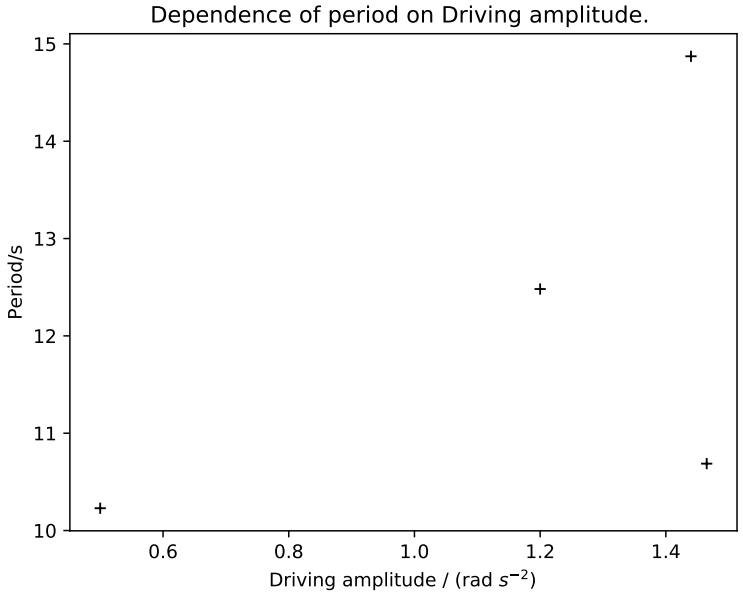


As in the small angle approximation, the difference between heavy and light damping is that the light damping wraps around the origin, while heavy (and critical) damped oscillators decay directly.

1.5 Driving behaviour.

The results of simulations including a sinusoidal driving force are given in the plots `driving - deflections and energies` and `driving - period vs amplitude`.


Comparison of different driving regimes
f = 0.5
f = 1.2
f = 1.44
f = 1.465
θ / radians
t/s
fraction of energy lost (dimensionless)
t/s


Dependence of period on Driving amplitude.
Period/s
Driving amplitude / (rad s^{-2})

1.5.1 Period

The principle linear superposition of solutions to inhomogeneous ODEs leads us expect that the system would oscillate with the frequency of the force: the homogeneous solution being damped out.

Thus the main period of oscillations should be $3/2\pi = 9.424$, This is not what we see, however. For weak damping, the period is less, indicating that there is still a higher frequency contribution from the natural oscillations.

For strong driving the period is greater than the theoretical prediction. This is due to the fact, that for stronger driving the object can wrap around and rotate for a brief time.

We also see that the maximum period is achieved for f =1.44

1.5.2 Deflection

The plots clearly follow the picture outlined above. The weaker driving forces produce an oscillation at a frequency close to that of the driving.

For the cases we were asked to investigate, the driving is sufficient to cause the system to wrap around the top.

For 1.2 thus produces a periodic behaviour, causing the system rotate albeit irregularly.

1.5.3 Energy.

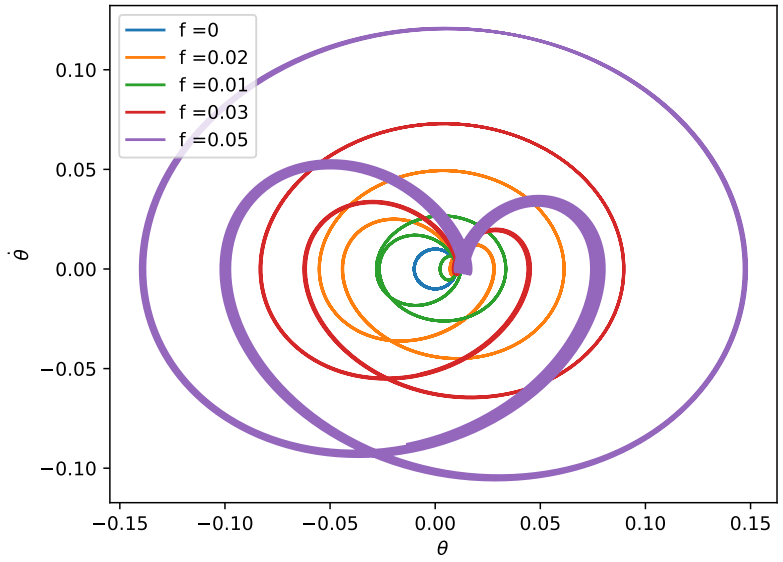
The weak driving case is limited mostly by the off-resonance frequency of the driving.

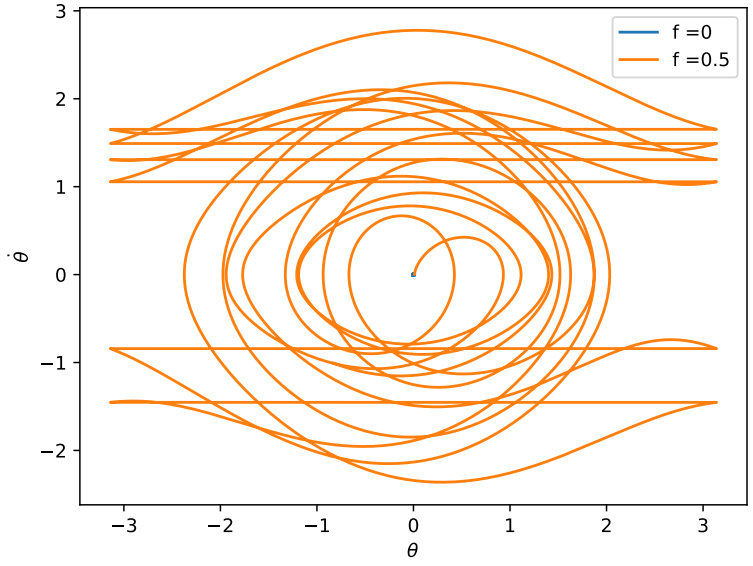
The energy starts exponentially increasing as soon as the system passes unstable equilibrium for the first time. Then the damping term, begins to dominate the evolution, bringing the kinetic energy to almost it's original value, when the system, again, starts gaining energy.

It's interesting to note that for strong driving (f>1.44) the gained energy drifts upwards, suggesting that the true resonant amplitude hadn't been reached during simulation.

1.5.4 Phase space.

Further insight into the system can be gleaned from the phase-space plots: `phase space - heavy driving.pdf` and `phase space - light driving.pdf`.

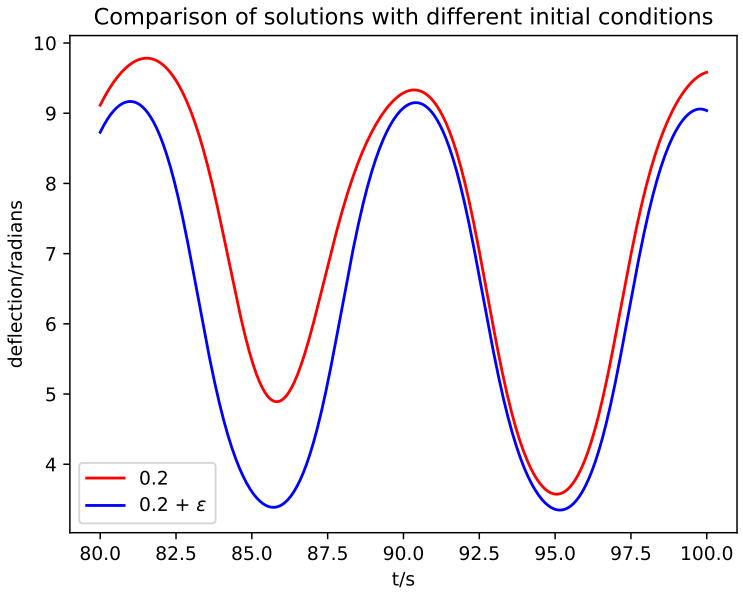




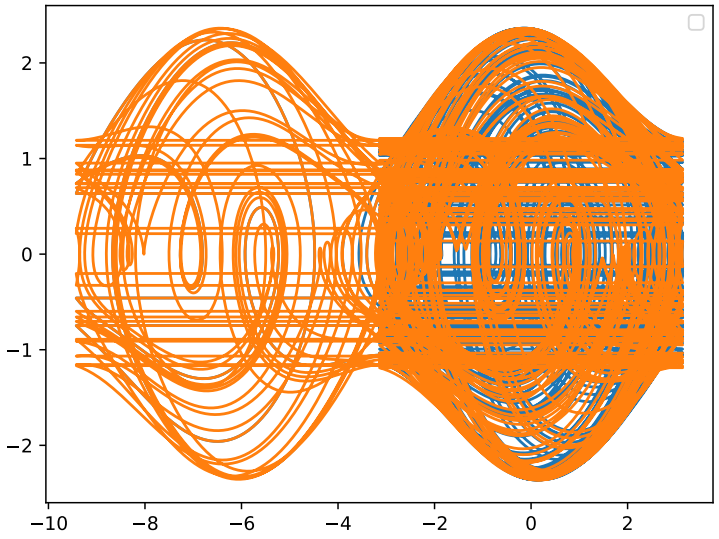
This showcases the wide gamut of different behaviour that can be obtained from this seemingly simple system. Some behaviours are periodic,

some are not. In fact the driving regimes that we were asked to consider, (the two lowest driving amplitudes) already produce a phase space picture too difficult to keep track of.

1.6 Sensitivity to initial conditions.



We start of by noting that the plots diverge rather quickly: within the first ten cycles.



The difference becomes even more apparent, when we consider the phase space plots.

One can clearly see that the larger value of the initial deflection causes the system to explore an entire new region of phase space.

1.7 A note on optimisations

The Library version of `odeint` is the most time consuming function to run, and is also the one that is probably optimised as far as possible. This function is single threaded, as the algorithm is strongly sequential.

The frequency estimate uses a linear interpolation method of finding the crossings. A much better result could have been achieved, had the `odeint` method been able to dynamically adjust the sampling rate to increase near such crossings. However to do that, the function would need a complete rewrite, which is too big of a time investment.

A possible speedup can be achieved by multi-threading the investigator functions, however the file is already large and difficult to read. Besides, A bigger consideration is the memory consumption of the program, and making it multi-process would only make it worse.

One solution would be to *down-sample* the `simulation_data`, but that would reduce the accuracy of the Frequency estimate.

A way to mitigate it, is to scope the objects inside for loops and functions. Thus python's garbage collector will remove the objects as soon as the loop or the function have exited.

A few extra milliseconds could be gained by removing the class inheritance and hard coding some of the parameters, e.g. ω_0, but that can be done at any time if need be.