Simulation of Kapitza's Pendulum

Federico Caretti

March 17, 2022

Abstract

Kapitza's pendulum[1] is a simple pendulum whose fulcrum is placed in a periodic motion with negligible amplitude and frequency much higher than the characteristic frequency of the unperturbed simple pendulum. In this presentation, the most interesting property of this system, the stability of the equilibrium point above the fulcrum, which is instead unstable in the simple pendulum, is showed, both with a theoretical approximation (Multi-Scale Method) and a simulation.

1 Theoretical description

Suppose that the motion of the pendulum lays on the plane (x, y) and assume that the fulcrum's position is defined by $y_F(t) = b\cos\omega t$. Hence, the acceleration of the fulcrum can be derived: $a_F = \frac{d^2y}{dt^2} = -b\omega^2\cos\omega t$. In angular coordinates, with θ being the angle between the pendulum and the vertical axis with positive value [2] :

$$\frac{d^2\theta}{dt^2} + \frac{1}{l}\left(g + b\omega^2\cos\omega t\right)\sin\theta = 0 \tag{1}$$

This can be expressed as

$$\frac{d^2\theta}{dt^2} = -\frac{\partial V}{\partial\theta}$$

where the potential is

$$V(\theta) = \frac{1}{l}\left(g + b\omega^2\cos\omega t\right)\cos\theta$$

By replacing $t' = \omega t$, $\sigma = \frac{g}{l\omega^2}$, $a = \frac{b}{l}$:

$$\frac{d^2\theta}{dt'^2} - \sigma\sin\theta - a\cos t'\sin\theta = 0$$

Now it is possible to introduce two "slow times": $t_1 = \varepsilon t$, $t_2 = \varepsilon^2 t$ and expand the total derivative over time in partial derivatives: $\frac{d}{dt} = \frac{\partial}{\partial t_0} + \varepsilon\frac{\partial}{\partial t_1} + \varepsilon^2\frac{\partial}{\partial t_2}$ and the angle θ in $\theta(t_0, t_1, t_2) = \theta_0(t_0, t_1, t_2) + \varepsilon\theta_1(t_0, t_1, t_2) + \varepsilon^2\theta_2(t_0, t_1, t_2) + O(\varepsilon^3)$. Since $\omega >> 1$, it is possible to introduce σ^* and a^* so that $\sigma = \varepsilon^2\sigma^*$ and $a = \varepsilon a^*$. By substitution on the equation above:

$$\left(\frac{\partial}{\partial t_0} + \varepsilon\frac{\partial}{\partial t_1} + \varepsilon^2\frac{\partial}{\partial t_2}\right)^2\left(\theta_0 + \varepsilon\theta_1 + \varepsilon^2\theta_2\right) = \varepsilon^2\sigma^*\sin\left(\theta_0 + \varepsilon\theta_1 + \varepsilon^2\theta_2\right) + \varepsilon a^*\cos t_0\sin\left(\theta_0 + \varepsilon\theta_1 + \varepsilon^2\theta_2\right)$$

We expand the angle around θ_0:

$$\left(\frac{\partial}{\partial t_0} + \varepsilon\frac{\partial}{\partial t_1} + \varepsilon^2\frac{\partial}{\partial t_2}\right)^2\left(\theta_0 + \varepsilon\theta_1 + \varepsilon^2\theta_2\right) = \left(\varepsilon^2\sigma^* + \varepsilon a^*\cos t_0\right)\left(\sin\theta_0 + \varepsilon\theta_1\cos\theta_0\right)$$

It is possible to consider the different orders of ε^2 separately. Starting from $O(\varepsilon^0)$:

$$\frac{\partial^2\theta_0}{\partial t_0^2} = 0$$

[1] https://www.youtube.com/watch?v=5o5eikf7xiY minute 4:20 for experimental example

[2] It is also possible to add a friction coefficient c and add a term $c\frac{d\theta}{dt}$, which will not be considered here

Which has constant or linear solution in t_0. In order to obtain, in the limit for $\lim_{\varepsilon\to\,0}$, the simple pendulum, we take only the linear coefficient, in order to avoid a "secular term": $\theta_0\left(t_0,t_1,t_2\right)=\theta_0\left(t_1,t_2\right)$

At order $O(\varepsilon)$:

$$\frac{\partial^2\theta_1}{\partial t_0^2}+2\frac{\partial}{\partial t_0}\frac{\partial\theta_0}{\partial t_1}=a^*\cos t_0\sin\theta_0$$

which, remembering that θ_0 is constant in t_0, yields:

$$\theta_1\left(t_0,t_1,t_2\right)=-a^*\cos t_0\sin\theta_0\left(t_1,t_2\right)$$

At order $O\left(\varepsilon^2\right)$:

$$\frac{\partial^2\theta_2}{\partial t_0^2}+2\frac{\partial}{\partial t_0}\frac{\partial\theta_1}{\partial t_1}+\frac{\partial^2\theta_0}{\partial t_1^2}=\sigma^*\sin\theta_0+a^*\cos t_0\theta_1\cos\theta_0$$

By replacing θ_0 and θ_1 with the previous results:

$$\frac{\partial\theta_2}{\partial t_0}=-2a^*\sin t_0\frac{\partial}{\partial t_1}\sin\theta_0-\frac{\partial^2\theta_0}{\partial t_1^2}+\sigma^*\sin\theta_0+\frac{a^{*2}}{4}\left(1+\cos 2t_0\right)\sin 2\theta_0$$

Again, in order to avoid secular terms, which in the limit $\lim_{t\to\infty}$ would make the contribution of $\varepsilon\theta_1$ and $\varepsilon^2\theta_2$ of the same order of magnitude of the θ_0 term, we require that they cancel out:

$$-\frac{\partial^2\theta_0}{\partial t_1^2}+\sigma^*\sin\theta_0+\frac{a^{*2}}{4}\sin 2\theta_0=0$$

This equation is very similar to the one of a pendulum: by replacing the values with the initial ones we obtain:

$$\frac{d^2\theta}{dt}-\sigma\sin\theta+\frac{a^2}{4}\sin 2\theta=0 \tag{2}$$

This can be interpreted as $\frac{d^2\theta}{dt^2}=-\frac{\partial V}{\partial\theta}$ where

$$V\left(\theta\right)=\sigma\cos\theta-\frac{a^2}{8}\cos 2\theta \tag{3}$$

is a sort of "average potential" (notice that if we had not considered the slow times we would have obtained the simple pendulum by simply averaging over the contributions). If $a^2>2\sigma$, the potential has a local minimum in the point $0,l$; therefore, the stability condition is $b\omega^2>\frac{2gl}{b}$, from which we define a *critical frequency*:

$$\omega_c=\frac{\sqrt{2gl}}{b} \tag{4}$$

This is the result of some expansions where the limits $\varepsilon\ll 1$ and $w\gg 1$ had to be satisfied. One interesting question is whether this theoretical approach is also applicable in the case of discrete time steps, like the ones that are necessarily used during numerical simulations.

2 Numerical solution

2.1 Methods

In this section the implemented methods will be introduced. Most of the analysis is made with Runge-Kutta 4 and implicit trapezoidal methods.

When we move in angular coordinates, we have a two-dimensional vector in phase space, called $\vec{Y}$, such that $Y_0=\theta$ and $Y_1=\dfrac{d\theta}{dt}$. We use $\vec{R}$, another two-dimensional vector, to denote the vector of the derivatives of the components of $\vec{Y}$. The derivatives are known from the physics of the problem, and the goal is to evaluate $\vec{Y}$ after a time step h. For instance, for the simple pendulum:

$$\begin{cases}Y_0=\theta\\ Y_1=\frac{d\theta}{dt}\\ R_0=\frac{dY_0}{dt}=\frac{d\theta}{dt}=Y_1\\ R_1=\frac{dY_1}{dt}=\frac{d}{dt}\frac{d\theta}{dt}=-\frac{g}{l}\sin\theta=-\frac{g}{l}\sin Y_0\end{cases} \tag{5}$$

The idea behind the Runge-Kutta methods is to evaluate the function in multiple points for each time step, therefore lowering the order of magnitude of the error. The following is the evaluation of $\vec{Y}^{n+1}$ using a Runge-Kutta 2 method:

$$\vec{Y}^{n+1} = \vec{Y}^n + \frac{h}{2}\left[\vec{R}(t^n, \vec{Y}^n) + \vec{R}(t^{n+1}, \vec{Y}_*^{n+1})\right]$$

where $\vec{Y}_*^{n+1}$ is the approximation of $\vec{Y}^{n+1}$ given by a Euler method:

$$\vec{Y}_*^{n+1} = \vec{Y}^n + hR(t^n, \vec{Y}^n)$$

Approximation with Runge-Kutta 2 results in an error of order $O(h^3)$ instead of the order $O(h^2)$ given by the Euler method. Similarly, the Runge-Kutta 4 method[3] requires 4 evaluations of $\vec{R}$ and gives an error of order $O(h^5)$.

Implicit methods are particularly useful when the function is *stiff*. The implicit Euler method follows the rule:

$$\vec{Y}^{n+1} = \vec{Y}^n + h\vec{R}(t^{n+1}, \vec{Y}^{n+1})$$

which looks simple in one-dimension, but in more dimension it requires finding the root of a set of equations. Similarly, a second-order implicit method is the trapezoidal, or Crank-Nicolson, method:

$$\vec{Y}^{n+1} = \vec{Y}^n + \frac{h}{2}\left(\vec{R}(t^n, \vec{Y}^n) + \vec{R}(t^{n+1}, \vec{Y}^{n+1})\right) \tag{6}$$

In this case, instead of estimating the value of $\vec{R}(t^{n+1}, \vec{Y}^{n+1})$ with a lower order method, as done in the RK2 method, the equations are solved through root-finding methods. In general, all the components of $\vec{R}$ depend on all the components of $\vec{Y}$, but in particular cases, like the simple pendulum and also Kapitza's pendulum, the equations in the system can be decoupled, as shown in section 2.5.

2.2 Check of used methods

The numerical analysis mainly revolves around the use of the Runge-Kutta 4 and implicit trapezoidal methods. The plot in Fig.1 shows the evolution of energy in a simple pendulum for the mentioned algorithms. The RK4 and implicit trapezoidal methods perform significantly better than the others; a comparison of these two methods, for higher Δt and number of periods, follows in fig. 2, which shows that the implicit method conserves the energy better in this simple task.

2.3 Simulation of the dynamics

When STAGE is defined to 1, a .dat file to be loaded with gnuplot is created in order to generate a dynamic view of the pendulum. In this phase, a reasonably high ω has been chosen, together with an initial condition suitable to observe the stability of the upper equilibrium point. The method used to simulate the evolution of the pendulum over time was Runge-Kutta 4, with the following right-hand side, which comes from eq.(1):

```
void rhs(double t, double *X, double *R){
  double mu=0.;
  R[0]=X[1];
  R[1]=-g/l*sin(X[0])-b/l*omega*omega*cos(omega*t)*sin(X[0])-mu*X[1];
}
```

The animation is 300 periods long, with a period defined as the time that the system takes to change its speed direction twice. The following settings have been used [4]:

```
double l=10, b=0.5, g=1, omega=sqrt(100/b);
```

Notice that the animation works also for initial conditions that do not allow stability, i.e. for which the dynamics is similar to a simple pendulum.

[3] explicitly written in appendix 4.1

[4] Global variables have been created in order to modify them inside of rhs

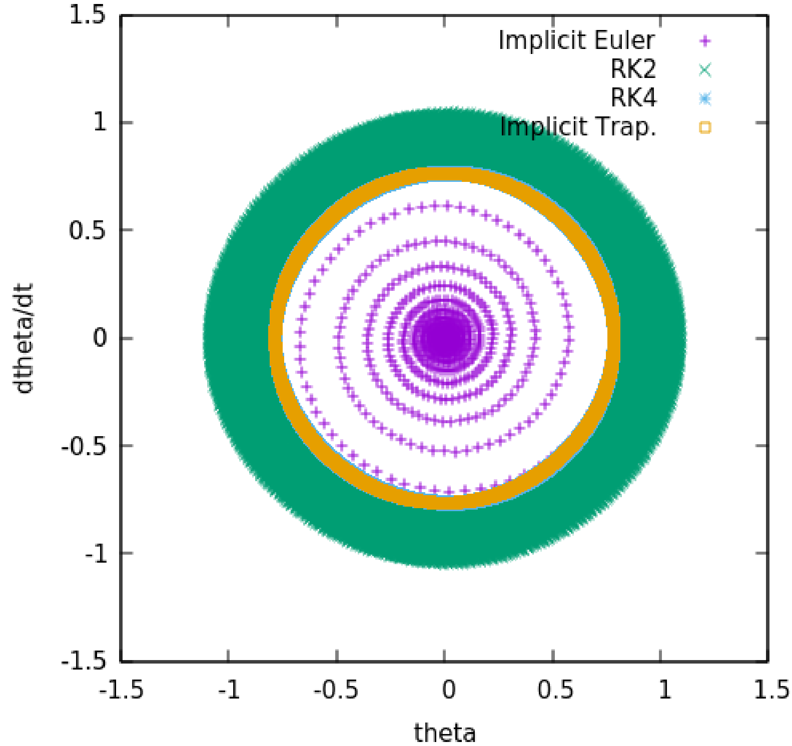

Figure 1: Energy evolution for some common methods. The time step is fixed at 0.1, the initial condition is $\theta = \frac{\pi}{4}$, the number of periods is 500

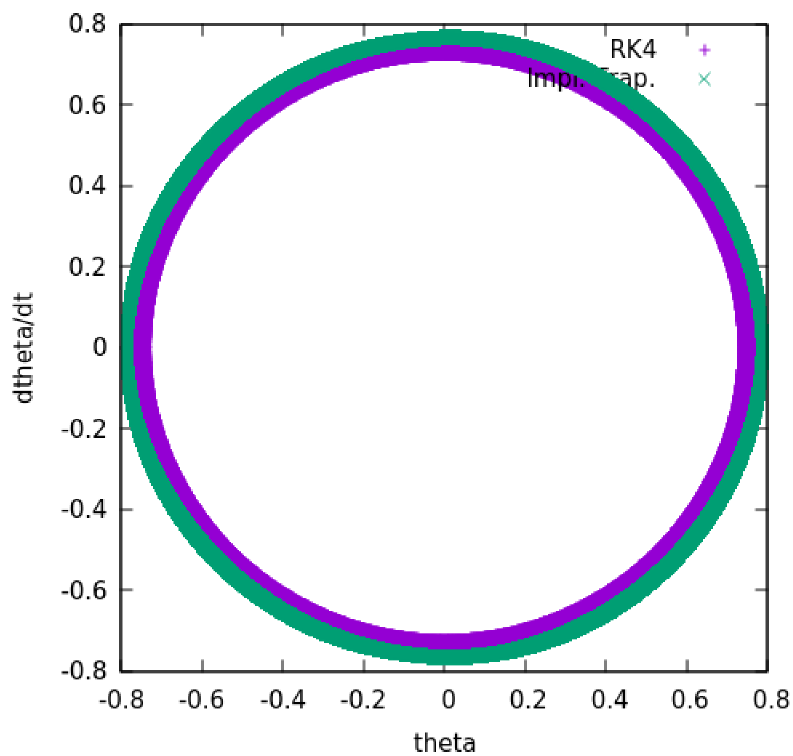

Figure 2: Energy evolution for RK4 and trapezoidal implicit method. The time step is fixed at 0.2, the initial condition is $\theta = \frac{\pi}{4}$, the number of periods is 5000

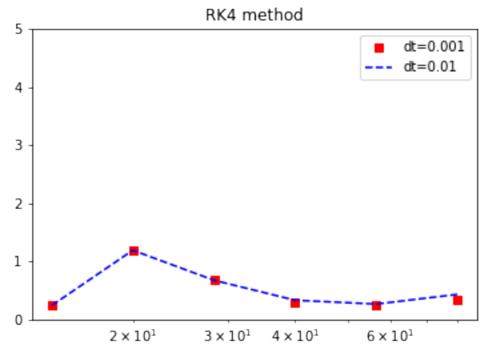

Figure 3: Plot of the difference between the evaluated ω_c and the theoretical value, for $\Delta t = 0.001$ (which gives the same results as $\Delta t = 0.0001$) and $\Delta t = 0.01$

2.4 Critical value of ω

Secondly, the validity of the solution found for the critical value of ω (following: ω_c) at equation 4 is tested. The method used is similar to a bisection method for root finding: two starting values are selected so that the first one is below the critical value (and hence no stability occurs) and the second is above; these has been verified by using the values as inputs for the animation of Part1. After that, at each step, ω is set to be equal to the average between the lower and upper values, and the stability is tested along 3000 periods. If the system never visits the lower part of the space, hence being close to equilibrium point, we assume it to be in a state where the frequency is high enough and replace the upper value with ω; otherwise, the lower value is replaced. The process continues until the difference between the two values is lower than a fixed tolerance.

If STAGE==3, the process above is repeated for some values of b (and hence of ω_c) without printing the intermediate steps. This is done for $\Delta t = 0.1, 0.01, 0.001, 0.0001$ and a comparison is printed (the following uses an initial condition $y[0] = -0.96\pi$, check of stability $fabs(\pi - y[0]) < 0.042\pi$ and 3000 periods), for $\sqrt{\frac{g}{l}}$ fixed to 1 (by setting $g, l = 10$) and values of b variable, starting at 1 and reducing it by a factor of $\sqrt{2}$ at each iteration, therefore multiplying the value of the expected ω_c by $\sqrt{2}$ each time:

```
Critical Omega14.3787   Expected Omega:14.1421   Difference: 0.236551   dt= 0.0001
Critical Omega14.3787   Expected Omega:14.1421   Difference: 0.236551   dt= 0.001
Critical Omega14.3787   Expected Omega:14.1421   Difference: 0.236551   dt= 0.01
Critical Omega14.369   Expected Omega:14.1421   Difference: 0.226847   dt= 0.1
Critical Omega21.1889   Expected Omega:20   Difference: 1.18888   dt= 0.0001
Critical Omega21.1889   Expected Omega:20   Difference: 1.18888   dt= 0.001
Critical Omega21.1883   Expected Omega:20   Difference: 1.18828   dt= 0.01
Critical Omega33.464   Expected Omega:20   Difference: 13.464   dt= 0.1
Critical Omega28.9556   Expected Omega:28.2843   Difference: 0.671316   dt= 0.0001
Critical Omega28.9556   Expected Omega:28.2843   Difference: 0.671316   dt= 0.001
Critical Omega28.9562   Expected Omega:28.2843   Difference: 0.671923   dt= 0.01
Critical Omega32.5548   Expected Omega:28.2843   Difference: 4.27051   dt= 0.1
Critical Omega40.2875   Expected Omega:40   Difference: 0.287516   dt= 0.0001
Critical Omega40.2875   Expected Omega:40   Difference: 0.287516   dt= 0.001
Critical Omega40.3275   Expected Omega:40   Difference: 0.327547   dt= 0.01
Critical Omega34.9682   Expected Omega:40   Difference: −5.03181   dt= 0.1
Critical Omega56.8126   Expected Omega:56.5685   Difference: 0.24407   dt= 0.0001
Critical Omega56.8126   Expected Omega:56.5685   Difference: 0.24407   dt= 0.001
Critical Omega56.8302   Expected Omega:56.5685   Difference: 0.26166   dt= 0.01
Critical Omega40.5641   Expected Omega:56.5685   Difference: −16.0044   dt= 0.1
Critical Omega80.3365   Expected Omega:80   Difference: 0.336538   dt= 0.0001
Critical Omega80.3365   Expected Omega:80   Difference: 0.336538   dt= 0.001
Critical Omega80.4281   Expected Omega:80   Difference: 0.428125   dt= 0.01
Critical Omega80.4997   Expected Omega:80   Difference: 0.499697   dt= 0.1
```

Two phenomena happen: in general, the obtained value is not exactly the expected frequency (especially for $\Delta t = 0.1$) and for some particular values the frequency for Δt is especially wrong also for lower Δts. Two are the possible explanations that came to my mind:

- the number of periods is not high enough. This is particularly true at even higher omega, when the behaviour of the pendulum becomes more chaotic. An increase in the number of periods should (and does) guarantee an improvement, at the cost of computational time. This phenomenon should lead to underestimate the critical value of ω. In previous versions I used to use 30 periods, but if there is no computational need, I suggest 3000. The output file is executed in a few seconds anyway.
- being Δt discrete, the difference with a continuous time could be particularly evident at the "border" of the stability region. Again, lowering the value of Δt would make the output file much slower to run, but should give a better result

Since the difference is always positive (at least for small Δt), hence the system is unstable even when it should be stable, the error could be due to the theoretical approximation or due to the second reason (a higher number of periods would not fix this).

2.5 Implicit Trapezoidal Method

In this section, the stability of the implicit trapezoidal method is tested. By substitution of eq.1 in eq.6:

$$\begin{cases} R_0 = \frac{d\theta}{dt} = Y_1 \\ R_1 = \frac{dY_1}{dt} = \frac{d}{dt}\frac{d\theta}{dt} = -\left(\frac{g}{l} + \frac{\omega^2}{l}\cos(\omega t)\right)\sin\theta = -\left(\frac{g}{l} + \frac{\omega^2}{l}\cos(\omega t)\right)\sin Y_0 \end{cases} \tag{7}$$

$$\begin{cases} Y_0^{n+1} = Y_0^n + \frac{\Delta t}{2}\left(R_0^n + R_0^{n+1}\right) = Y_0^n + \frac{\Delta t}{2}\left(Y_1^n + Y_1^{n+1}\right) \\ Y_1^{n+1} = Y_1^n + \frac{\Delta t}{2}\left(\left(-\frac{b\omega^2}{l}\cos(\omega t) - \frac{g}{l}\right)\sin Y_0^n + \left(-\frac{b\omega^2}{l}\cos(\omega(t+\Delta t)) - \frac{g}{l}\right)\sin Y_0^{n+1}\right) \end{cases} \tag{8}$$

It is important to notice that in the equation for Y_0^{n+1} only Y_1^{n+1} appears and vice versa; it is therefore possible to decouple the equations, by replacing Y_1^{n+1} in the first equation with its value given by the second equation. This results in:

$$Y_0^{n+1} = Y_0^n + \Delta t\left(Y_1^n + \frac{\Delta t}{4}\left(\left(-\frac{b\omega^2}{l}\cos(\omega(t+\Delta t)) - \frac{g}{l}\right)\sin Y_0^{n+1} + \left(-\frac{b\omega^2}{l}\cos(\omega t) - \frac{g}{l}\right)\sin Y_0^n\right)\right) \tag{9}$$

Now it is possible to solve this equation by moving every term on one side and using a one-dimensional root finding method. Then, the result can be inserted into the equation for Y_1^{n+1}. This process is executed by the following functions:

```c
double func_for_trap(double *Y, double x, double h, double t){
  return x-Y[0]-h*Y[1]+h*h/4*((b/l*omega*omega*cos(omega*(t+h))+g/l)*sin(x)
    +(b/l*omega*omega*cos(omega*(t))+g/l)*sin(Y[0]));
}

//Had to rewrite bisection in order to pass more arguments
void I_trap_kapitza(double t, double *Y, double h){
  double Y1[2];
  Y1[0]=Bisection(func_for_trap,Y, Y[0]-1,Y[0]+1,1e-7 ,h,t);
  Y1[1]=Y[1]-h/2*((b/l*omega*omega*cos(omega*t)+g/l)*sin(Y[0])+
    (b/l*omega*omega*cos(omega*(t+h))+g/l)*sin(Y1[0]));
  Y[0]=Y1[0];
  Y[1]=Y1[1];
}
```

The following are the output of the program with STAGE==3 and METHOD==2:

```
Critical Omega14.3787    Expected Omega:14.1421    Difference: 0.236551    dt= 0.0001
Critical Omega14.3793    Expected Omega:14.1421    Difference: 0.237158    dt= 0.001
Critical Omega14.2659    Expected Omega:14.1421    Difference: 0.123735    dt= 0.01
Critical Omega35.7718    Expected Omega:14.1421    Difference: 21.6297    dt= 0.1
Critical Omega21.1889    Expected Omega:20    Difference: 1.18888    dt= 0.0001
Critical Omega21.1889    Expected Omega:20    Difference: 1.18888    dt= 0.001
Critical Omega43.2438    Expected Omega:20    Difference: 23.2438    dt= 0.01
Critical Omega20.8268    Expected Omega:20    Difference: 0.82678    dt= 0.1
```

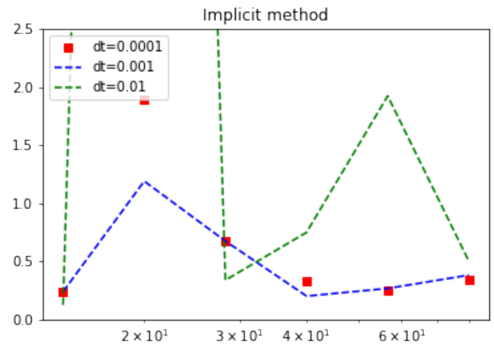

Figure 4: Difference between empirical and theoretical ω_cs at different frequencies, as done in section 2.4

```
Critical Omega28.9544   Expected Omega:28.2843   Difference: 0.670103   dt= 0.0001
Critical Omega28.9574   Expected Omega:28.2843   Difference: 0.673136   dt= 0.001
Critical Omega28.6208   Expected Omega:28.2843   Difference: 0.336508   dt= 0.01
Critical Omega38.6723   Expected Omega:28.2843   Difference: 10.388   dt= 0.1
Critical Omega40.3251   Expected Omega:40   Difference: 0.325121   dt= 0.0001
Critical Omega40.1996   Expected Omega:40   Difference: 0.199568   dt= 0.001
Critical Omega40.7479   Expected Omega:40   Difference: 0.747877   dt= 0.01
Critical Omega40.5174   Expected Omega:40   Difference: 0.517393   dt= 0.1
Critical Omega56.8199   Expected Omega:56.5685   Difference: 0.251348   dt= 0.0001
Critical Omega56.8338   Expected Omega:56.5685   Difference: 0.265299   dt= 0.001
Critical Omega58.4915   Expected Omega:56.5685   Difference: 1.92296   dt= 0.01
Critical Omega80.4336   Expected Omega:56.5685   Difference: 23.865   dt= 0.1
Critical Omega80.3432   Expected Omega:80   Difference: 0.34321   dt= 0.0001
Critical Omega80.3802   Expected Omega:80   Difference: 0.380209   dt= 0.001
Critical Omega80.4997   Expected Omega:80   Difference: 0.499697   dt= 0.01
Critical Omega78.1008   Expected Omega:80   Difference: −1.89916   dt= 0.1
```

With this level of analysis, the RK4 method outperforms the implicit method both in terms of accuracy and time of computation, not requiring to find the zero of a non-linear function. It can be observed in the plots that both methods fail at $expected\omega_c = 20$, but this is probably due to the initial condition of the search (for this iteration, the highest ω_c tried is 81) and whatever the initial condition, I always found some values where the simulation fails to obtain the right result. I think that this is an inherent risk of using discrete time steps, and the problem could be partially solved by lowering the number of periods or Δt. However, for small Δts and frequencies, the results are coherent with those from RK4.

3 Final thoughts

The code provides a good simulation for values of the variables close to one, but as soon as they are too big, errors happen. Just to show an example, this is the result obtained during STAGE==3 with a starting $upperomega = 1000$, both using the implicit trapezoidal method:

```
Critical Omega220.508   Expected Omega:14.1421   Difference: 206.366   dt= 0.0001
Critical Omega14.2423   Expected Omega:14.1421   Difference: 0.100205   dt= 0.001
Critical Omega1000   Expected Omega:14.1421   Difference: 985.857   dt= 0.01
Critical Omega1000   Expected Omega:14.1421   Difference: 985.857   dt= 0.1
Critical Omega16.606   Expected Omega:20   Difference: −3.39396   dt= 0.0001
Critical Omega14.6577   Expected Omega:20   Difference: −5.34227   dt= 0.001
Critical Omega1000   Expected Omega:20   Difference: 980   dt= 0.01
Critical Omega1000   Expected Omega:20   Difference: 980   dt= 0.1
Critical Omega28.4331   Expected Omega:28.2843   Difference: 0.148843   dt= 0.0001
Critical Omega154.648   Expected Omega:28.2843   Difference: 126.363   dt= 0.001
Critical Omega1000   Expected Omega:28.2843   Difference: 971.715   dt= 0.01
Critical Omega1000   Expected Omega:28.2843   Difference: 971.715   dt= 0.1
Critical Omega505.572   Expected Omega:40   Difference: 465.572   dt= 0.0001
Critical Omega40.1925   Expected Omega:40   Difference: 0.192545   dt= 0.001
Critical Omega885.645   Expected Omega:40   Difference: 845.645   dt= 0.01
Critical Omega1000   Expected Omega:40   Difference: 960   dt= 0.1
```

```
Critical Omega56.8299    Expected Omega:56.5685    Difference: 0.261363    dt= 0.0001
Critical Omega314.924    Expected Omega:56.5685    Difference: 258.355    dt= 0.001
Critical Omega808.661    Expected Omega:56.5685    Difference: 752.092    dt= 0.01
Critical Omega1000    Expected Omega:56.5685    Difference: 943.431    dt= 0.1
Critical Omega80.3392    Expected Omega:80    Difference: 0.339241    dt= 0.0001
Critical Omega125.989    Expected Omega:80    Difference: 45.9889    dt= 0.001
Critical Omega86.1261    Expected Omega:80    Difference: 6.12607    dt= 0.01
Critical Omega1000    Expected Omega:80    Difference: 920    dt= 0.1
```

and RK4:

```
Critical Omega14.2423    Expected Omega:14.1421    Difference: 0.100205    dt= 0.0001
Critical Omega14.2423    Expected Omega:14.1421    Difference: 0.100205    dt= 0.001
Critical Omega1000    Expected Omega:14.1421    Difference: 985.857    dt= 0.01
Critical Omega1000    Expected Omega:14.1421    Difference: 985.857    dt= 0.1
Critical Omega14.6577    Expected Omega:20    Difference: -5.34227    dt= 0.0001
Critical Omega14.6577    Expected Omega:20    Difference: -5.34227    dt= 0.001
Critical Omega1000    Expected Omega:20    Difference: 980    dt= 0.01
Critical Omega1000    Expected Omega:20    Difference: 980    dt= 0.1
Critical Omega28.4322    Expected Omega:28.2843    Difference: 0.147891    dt= 0.0001
Critical Omega28.4322    Expected Omega:28.2843    Difference: 0.147891    dt= 0.001
Critical Omega1000    Expected Omega:28.2843    Difference: 971.715    dt= 0.01
Critical Omega1000    Expected Omega:28.2843    Difference: 971.715    dt= 0.1
Critical Omega40.5165    Expected Omega:40    Difference: 0.51647    dt= 0.0001
Critical Omega40.5165    Expected Omega:40    Difference: 0.51647    dt= 0.001
Critical Omega909.586    Expected Omega:40    Difference: 869.586    dt= 0.01
Critical Omega1000    Expected Omega:40    Difference: 960    dt= 0.1
Critical Omega56.8118    Expected Omega:56.5685    Difference: 0.243262    dt= 0.0001
Critical Omega56.8118    Expected Omega:56.5685    Difference: 0.243262    dt= 0.001
Critical Omega835.668    Expected Omega:56.5685    Difference: 779.099    dt= 0.01
Critical Omega1000    Expected Omega:56.5685    Difference: 943.431    dt= 0.1
Critical Omega80.3364    Expected Omega:80    Difference: 0.336382    dt= 0.0001
Critical Omega80.3364    Expected Omega:80    Difference: 0.336382    dt= 0.001
Critical Omega836.648    Expected Omega:80    Difference: 756.648    dt= 0.01
Critical Omega1000    Expected Omega:80    Difference: 920    dt= 0.1
```

Using a ω too high probably makes too evident the discreteness of the time steps. An implicit higher order method could provide an higher range of stability. In Appendix I reported the code both for the C++ file and the gnuplot file for the gifs. I copied the used functions (like RK4) into the code so that it possible to compile it without imports. As previously stated, I suggest to not change too much the variables, and the delay in the .gp file can be setted depending on which value of Δt is chosen. In STAGE==1 with RK4, it is also possible to add a friction coefficient from the rhs function.

4 Appendix

4.1 Runge-Kutta 4

The rule for the RK4 method is the following:

$$\begin{cases} k_1 = \vec{R}(t^n, Y^n) \\ k_2 = \vec{R}(t^n + \frac{h}{2}, Y^n + \frac{h}{2}k_1) \\ k_3 = \vec{R}(t^n + \frac{h}{2}, Y^n + \frac{h}{2}k_2) \\ k_4 = \vec{R}(t^n + h, Y^n + hk_3) \\ Y^{n+1} = Y^n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases} \tag{10}$$

4.2 C++ Code

```
#include <iostream>
#include <cmath>
#include<iomanip>
#include <fstream>


#define STAGE 1
#define METHOD 2
```

```
using namespace std;
void rhs(double t, double *X, double *R);
void RK4Step(double t, double *Y, void (*RHS_Func)(double, double *, double *), double h, int neq);
void I_trap_kapitza(double t, double *Y, double h);
void forward_time(double t, double *y, void (*RHS_Func)(double, double *, double *),
  double dt, int neq, int n_periods, bool verbose);
double Bisection(double (*F)(double *Y, double x, double h, double t),
  double *Y,double a, double b, double tol, double h, double t);
double l=10, b=0.5, g=1, omega=2*sqrt(2*100)/b;


int main(){
  double t=0.0;
  double dt;

  double y[2]={-M_PI*0.80, 0.0};
  int N_periods;
  #if STAGE==1
  N_periods=300;
  dt=0.01;
  bool verbose=true;
  #endif
  #if STAGE==2
  N_periods=3000;
  dt=0.001;
  bool verbose=true;
  #endif
  #if STAGE==1 || STAGE==2
  forward_time(t,y,rhs,dt,2,N_periods, verbose);
  #endif
  #if STAGE==3
  N_periods=3000;
  bool verbose=false;
  g=10;
  l=10;
  for(b=1;b>0.15;b=b/sqrt(2)){
    for(dt=0.0001;dt<1;dt=10*dt){
      t=0.0;
      forward_time(t,y,rhs,dt,2,N_periods,verbose);
    }
  }
  #endif
  return 0;
}



void forward_time(double t, double *y, void (*RHS_Func)(double, double *, double *),
    double dt, int neq, int n_periods, bool verbose){
  #if STAGE==1
  ofstream fdata;
  fdata.open("kapitza.dat");
  int cycle=0;
  double vaux=0.0;
  for(int i=0;(cycle<2*n_periods);i++){
    vaux=y[1];
    #if METHOD==1
    RK4Step(t, y, rhs, dt, 2);
    #endif
    #if METHOD==2
    I_trap_kapitza(t,y,dt);
    #endif
    t+=dt;
    if (verbose==true){
      cout <<t << "    " <<cycle <<endl;
    }
    fdata <<t << "    " << 1 << "    "<<0.0 <<" " <<-b*cos(omega*t) <<endl;
    fdata <<t << "    " << 2 << "    "<< l*sin(y[0]) <<" "
      <<-l*cos(y[0])-b*cos(omega*t)  <<endl;
    if(vaux*y[1]<=0.0){
```

```
        cycle++;
      }
    }
    fdata <<endl <<endl;
    fdata.close();
  #endif
  #if STAGE==2 || STAGE==3
  double tol=0.001, vaux, cycle;
  double low_omega=1.;
  double upper_omega=81;
  bool stable;
  double a,d;
  while(fabs(upper_omega-low_omega)>tol){
    stable=false;
    omega=0.5*(low_omega+upper_omega);
    t=0.0;
    y[0]=M_PI*(-0.96);
    y[1]=0.;
    cycle=0.0;
    for(int i=0;(cycle<2*n_periods)&&(t<300);i++){
      vaux=y[1];
      #if METHOD==1
      RK4Step(t, y, rhs, dt, 2);
      #endif
      #if METHOD==2
      I_trap_kapitza(t,y,dt);
      #endif
      t+=dt;
      //cout <<t << "      " <<cycle <<endl;
      if(vaux*y[1]<=0.0){
        cycle++;
      }
    }
    if (fabs(y[0]+M_PI)<0.041*M_PI){
      stable=true;
    }
    if(stable==true){
      upper_omega=omega;
      if(verbose==true){
        cout <<omega <<": Stable"  <<endl;
      }
    }
    if(stable==false){
      low_omega=omega;
      if(verbose==true){
        cout <<omega <<": Unstable" <<endl;
      }
    }
    if(verbose==true){
      cout <<"Lower:" <<low_omega <<" Upper:" <<upper_omega <<endl;
    }
  }
  a=0.5*(low_omega+upper_omega);
  d=sqrt(2*g*l)/b;
  cout <<"Critical Omega" <<a  <<"   Expected Omega:" <<d
    <<"   Difference: " <<(a-d) <<"   dt= " <<dt <<endl;
  #endif
}


void rhs(double t, double *X, double *R){
  double mu=0.0;
  R[0]=X[1];
  R[1]=-g/l*sin(X[0])-b/l*omega*omega*cos(omega*t)*sin(X[0])-mu*X[1];
}

void RK4Step(double t, double *Y, void (*RHS_Func)
     (double, double *, double *), double h, int neq){
  double Y1[neq], k1[neq], k2[neq], k3[neq], k4[neq];
  int i;
  RHS_Func(t,Y,k1);
```

```
  for(i=0; i<neq; i++){
      Y1[i] = Y[i]+0.5*h*k1[i];
  }
  RHS_Func(t+0.5*h,Y1,k2);
  for(i=0; i<neq; i++){
      Y1[i] = Y[i]+0.5*h*k2[i];
  }
  RHS_Func(t+0.5*h,Y1,k3);
  for(i=0; i<neq; i++){
      Y1[i] = Y[i]+h*k3[i];
  }
  RHS_Func(t+h,Y1,k4);
  for(i=0; i<neq; i++){
      Y[i] += h/6.0*(k1[i]+2*k2[i]+2*k3[i]+k4[i]);
  }
}


double func_for_trap(double *Y, double x, double h, double t){
  return x-Y[0]-h*Y[1]+h*h/4*((b/l*omega*omega*cos(omega*(t+h))+g/l)*sin(x)
    +(b/l*omega*omega*cos(omega*(t))+g/l)*sin(Y[0]));
}

//Had to rewrite bisection in order to pass more arguments
void I_trap_kapitza(double t, double *Y, double h){
  double Y1[2];
  Y1[0]=Bisection(func_for_trap,Y, Y[0]-1,Y[0]+1,1e-7 ,h,t);
  Y1[1]=Y[1]-h/2*((b/l*omega*omega*cos(omega*t)+g/l)*sin(Y[0])+
    (b/l*omega*omega*cos(omega*(t+h))+g/l)*sin(Y1[0]));
  Y[0]=Y1[0];
  Y[1]=Y1[1];
}

double Bisection(double (*F)(double *Y, double x, double h, double t),
    double *Y,double a, double b, double tol, double h, double t){
  double fa=F(Y,a,h,t);
  double fb=F(Y,b,h,t);
  double xm;
  double fm;
  double dx=fabs(b-a);
  for(int k=1;fabs(a-b)>tol; k++){
    xm=(a+b)*0.5;
    fm=F(Y,xm,h,t);
    //cout <<"Bisection:  k=" <<k <<"; [a,b]= [" <<a <<", " <<b <<"]" <<"; xm= "
      //<<xm <<endl;
    if(fm*fa>0){
      a=xm;
      fa=fm;
    }
    else if(fm*fa<0){
      b=xm;
      fb=fm;
    }
    else if(fm*fa==0){
      return xm;
    }
  }
  return xm;
}
```

4.3 Gnuplot

```
reset
set terminal gif animate delay 1
set output 'kapitza.gif'
set xrange [-10:10]
set yrange [-10:10]
set size square 1,1
set pointsize 2
```

```
set style line 2 lc rgb '#0060ad' pt 7

do for [i=0:4000] {
    plot 'kapitza.dat' u (0):(0):3:4 every ::2*i::2*i w vectors lc "grey" nohead notitle, \
         x2=y2=NaN '' u (x1=x2,x2=$3):(y1=y2,y2=$4):(x1-x2):(int($0)%2==0 ? NaN: y1-y2) every ::2*i::2*i
         ''    u 3:4 every ::2*i::2*i     w p pt 7 ps 3 lc "red" title "1", \
         ''    u 3:4 every ::2*i+1::2*i+1 w p pt 7 ps 2 lc "blue" title "2", \
}
```