Computational Physics - Project A

David Monk – 00928791

Abstract

This report will look at the use of finite difference methods (FDMs) to solve the motion of both a single and double pendulum in damped and undamped conditions. The stability of four different methods were tested for a single pendulum, while only the Runge-Kutta 4 method was tested for a double pendulum.

Introduction

Finite Difference Methods

Finite difference methods (FDMs) use iterative matrix multiplication to solve series of coupled first order differential equations. This can be generalised by the equation:

$$\frac{d}{dt}\begin{pmatrix} u \\ v \end{pmatrix} = \mathbf{T} \cdot \begin{pmatrix} u \\ v \end{pmatrix} \tag{1}$$

where $\mathbf{T}$ is the operating matrix which couples u and v. This matrix describes how the function evolves with time, as the $n+1$ term will be a function of the previous terms. Four FDMs were implemented in this project.

The simplest of these techniques is the Euler forward method, which implements the first order Taylor expansion to result in the iterative formula:

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{T} \cdot \mathbf{v}_n \tag{2}$$

This method has a global error of $\mathcal{O}(h)$. The leapfrog method has a global accuracy of $\mathcal{O}\left(h^2\right)$ and is the simplest of the multi-step family of procedures. The increase in accuracy comes from the requirement for the $n-1$th term as well as the nth term. Although making it more accurate and stable in certain conditions[1], the extra term can lead to initial value errors.

The Runge-Kutta 4 method has a global error $\mathcal{O}\left(h^4\right)$[3], and is hence much more accurate than the previous two techniques. The Runge-Kutta family use the weighted average of gradients at a number of points between t and $t+h$ to find the $n+1$th term of the function.

The final of the four implemented techniques differs from the others by being an implicit method, compared to the previous three explicit methods. The implicit Euler method is analogous to its explicit cousin, but instead evaluates the gradient at $n+$, rather than n. Although being accurate only to $\mathcal{O}(h)$, it is unconditionally stable[2].

In the case of a single pendulum, equation of motion is given by

$$ml\frac{d^2\theta}{dt^2} = -mg\sin\theta - D\frac{d\theta}{dt} \tag{3}$$

This equation is first scaled, then converted into matrix form under the small and regime (see appendix for full derivation), yielding:

$$\frac{d}{d\hat{t}}\begin{pmatrix}\theta \\ \omega\end{pmatrix} = \begin{pmatrix}0 & 1 \\ -1 & -\hat{D}\end{pmatrix} \cdot \begin{pmatrix}\theta \\ \omega\end{pmatrix} \tag{4}$$

This matrix can then be used in the chosen finite difference method.

Criteria for stability

For an undamped, undriven pendulum, the energy of the system is constant with time. Thus, calculating the total energy of the system (potential + kinetic) for each time step will provide a good test of stability. In order to remove dependency on g, l etc. from the stability test, the relative error in the energy was calculated instead of the absolute energy. This was given by $E_{error} = |\frac{E_i - E_0}{E_0}|$. In the case of a damped pendulum, the total energy should decay with time, thus if the energy is greater than the initial energy at any time $\hat{t} > 1/\hat{D}$ (to allow for numerical errors at times close to $\hat{t} = 0$, then the method will be considered unstable.

Programming Procedure

The algorithms described above were implemented in C, with use of the GNU Scientific Library for certain matrix operations. Compiled C is generally considered to be in the order of 10 times faster than Python, which meant that checking stability over a large number of periods was not time consuming. These algorithms were then wrapped in Python to allow for flexible plotting of the values.

Analysis

The equations of motion for a single pendulum can be solved analytically and thus the results from the FDMs could compared with the actual values. Plots of this comparison can be seen in figures **??**-**??**. Even from the simple amplitude plots, it is clear that the Euler forward method is unstable for oscillating solutions, even for very small time steps.

Both the leapfrog and RK4 methods seem to be stable for these conditions, however both also seem to accumulate error in a linear fashion with time. As seen on the energy stability plots (**??** - **??**), this error cannot not arise from instability of the amplitude, moreover it is a form of numerical dispersion. Both of these methods produce solutions which have an angular frequency very slightly different from the original. This means that the numerical solution becomes out of phase with the analytical, thus producing an error. This can be seen more easily when the number of maximum time is large (see fig **??**). Such a phenomenon is trivial to calculate when the correct solution is know, but much harder for non-analytic problems, such as the double pendulum.

Returning to the energy stability plots, it is clear that leapfrog and RK4 methods become unstable at different lengths of time step. For leapfrog, the maximum energy tends to an asymptotic infinity at $h = 1$, while RK4 is stable up to $h = 2.84$. Furthermore, the RK4 method becomes unstable very sharply, while leapfrog becomes semi-unstable from $h \approx 0.8$. This means that RK4 can be used at large step sizes less than the critical step size, without concern that the result is becoming unstable.

Conclusion

References

[1] Charles K Birdsall and A Bruce Langdon. *Plasma physics via computer simulation.* CRC Press, 2004.

[2] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods.* Wiley-Interscience, 1987.

[3] Antony Jameson, Wolfgang Schmidt, Eli Turkel, et al. Numerical solutions of the euler equations by finite volume methods using runge-kutta time-stepping schemes. *AIAA paper*, 1259:1981, 1981.

Appendix

$$ml\frac{d^2\theta}{dt^2} = -mg\sin\theta - D\frac{d\theta}{dt} \tag{5}$$

This equation is first scaled by introducing a new variable, $\hat{t} = \sqrt{\frac{g}{l}}t$, before being decomposed into a set of coupled first-order differential equations:

$$\frac{d\theta}{d\hat{t}} = \omega \tag{6}$$

$$\frac{d\omega}{d\hat{t}} = -\sin\theta - \hat{D}\omega \tag{7}$$

where $\hat{D} = \frac{D}{m\sqrt{gl}}$ (For full derivation, see appendix). By using the small angle approximation ($\theta \ll 1$, $\sin\theta = \theta$), equations 6 & 7 can be converted to matrix form, yielding:

$$\frac{d}{d\hat{t}}\begin{pmatrix}\theta \\ \omega\end{pmatrix} = \begin{pmatrix}0 & 1 \\ -1 & -\hat{D}\end{pmatrix} \cdot \begin{pmatrix}\theta \\ \omega\end{pmatrix} \tag{8}$$

Figures