Computational Methods Junior Honours

Dr Peter Boyle, 2007. Room 4401. paboyle@ph.ed.ac.uk

1 Practicalities

- Labs are on Tuesdays and Fridays in the CPLab from 3-5pm throughout term in CPlab and 1206.
  There are 45 slots on Tuesday and 55 on Friday.
  You may attend only one session per week.
  Sign up is through WebCT
- Labs run from weeks 1-11, starting today
- All the teaching materials, including this document, are at
  `www.ph.ed.ac.uk/~paboyle/Teaching/CompMeth2008/`
- Course is continuously assessed and computationally based
- It is expected that you will need to do additional work outwith the contact hours to complete the course

2 Introduction

The idea that physics is about solving equations analytically is a major misapprehension in undergraduate education created by teaching you the things we can solve instead of the things we can't.

The mathematics underlying most problems of interest today cannot be solved analytically, (e.g. non-linear systems, many particle systems, or bodies with complex shapes and/or no simplifying symmetries) and the contribution of computing in 20th century physics has been to solve otherwise intractable equations by using numerical algorithms to integrate the various forces in a simulated time, just as nature integrates its complex forces in real time.

In the 21st century, computer simulation will leave complete mathematical solution further behind and physical phenomena will be deemed understood when numerical simulation of the governing equations of motion for the underlying processes are shown to describe the real world.

3 Continuous and discrete time

Nature, at least in the classical Newtonian picture, works with with a continuously flowing time and classical objects accelerate at each instant in response to the forces to which they are exposed.

Classical mechanics is therefore mathematically an initial value problem, where if all potentials, particle coordinates and velocities are specified at some initial time t_0, we can integrate the Newtonian equations of motion to obtain all information about any later time $t > t_0$. Most of this course will be focus on solving various initial value problems using numerical (i.e. computer) integration.

On a digital computer only a finite number of arithmetic operations can be carried out before it breaks and it is time to buy a new one, so clearly to move a simulated world over a non-zero simulated time period T you need to approximate nature with large number N of time steps of length $\delta = \frac{T}{N}$.

4 Approximations to derivatives

This discrete approach should not seem to unnatural - after all the Newtonian derivative is defined via a limiting process that uses finite differences δ that are taken to zero.

$$\frac{dF(x)}{dx} = \lim_{\delta \to 0} \frac{F(x+\delta) - F(x)}{\delta} \tag{1}$$

For non-zero δ we can use the Taylor expansion of F(x) to define numerical difference schemes.

$$F(x+\delta) = F(x) + \sum_{n=1}^{\infty} \frac{\delta^n}{n!} \frac{d^n}{dx^n} F(x) \tag{2}$$

or

$$F(x+\delta) = F(x) + \delta F'(x) + \frac{\delta^2}{2}F''(x) + \frac{\delta^3}{6}F'''(x) + O(\delta^4) \tag{3}$$

A naive (Euler) numerical derivative is obtained by simply taking right hand side of Equation 1 at fixed δ.

$$\frac{F(x+\delta) - F(x)}{\delta} = F'(x) + \frac{\delta}{2}F''(x) + \frac{\delta^2}{6}F'''(x) + O(\delta^4) \tag{4}$$

This approximation would have a leading relative error of $O(\delta)$ for small δ.

This should give you a clue about how to produce better numerical approximations. The key thing is to note that the series in Equation 3 consists of terms that are alternately odd and even in δ.

$$F(x-\delta) = F(x) - \delta F'(x) + \frac{\delta^2}{2}F''(x) - \frac{\delta^3}{6}F'''(x) + O(\delta^4) \tag{5}$$

Subtracting and adding (3) and (5) results in cancellations of different powers of δ:

$$\frac{F(x+\delta) - F(x-\delta)}{2\delta} = F'(x) + O(\delta^2) \tag{6}$$

$$\frac{F(x+\delta) + F(x-\delta) - 2F(x)}{\delta^2} = F''(x) + O(\delta^2) \tag{7}$$

This gives better approximations to the true continuous derivatives.

In numerical simulations by many scientists a small value for δ and a good approximation scheme is used so that any numerical errors should be small.

In numerical simulations by *reputable* scientists the same simulation is run with a number of values for δ. An extrapolation for $\delta \to 0$ can be made using so that approximation is both shown to be good and is removed from the final answer.

5 Leap-frog integration algorithm

Much of this course will use the leap-frog integration algorithm to evolve our simulated system.

In this scheme a particle has associated with it a position, velocity, and force.

The force is often dependent only on the particles location within some potential, and thus the accelaration $\vec{a}_\tau$ is known at all the discrete time instants τ at which we update and store the particle's position x_τ.

$$\vec{a}_\tau = -\frac{1}{m}\vec{\nabla}V(\vec{x}_\tau) \tag{8}$$

In order to evolve that particle in our simulated time we wish eliminate step size errors as illustrated above.

The key to the leap-frog scheme is that to update the velocity (whose derivative is $\vec{a}_\tau$) with the most accurate way of (6) we should update $\vec{v}$ according to a time shifted grid at times $\tau + \frac{\delta}{2}$ that is interleaved evenly between the time grid for $\vec{x}$ and $\vec{a}$.

Thus $\vec{v}$ and $\vec{x}$ leap-frog over each other in time and we update $\vec{v}$ at the intermediate time instants $\tau - \frac{\delta}{2}$ and $\tau + \frac{\delta}{2}$ such that the best approximation to the derivative of $\vec{v}$ from $\vec{v}(\tau - \frac{\delta}{2})$ and $\vec{v}(\tau + \frac{\delta}{2})$ is the accelaration $\vec{a}(\tau)$.

The leap frog algorithm is one of a particularly stable class call *symplectic* integrators that are commonly used in numerical science.

6 Updates for the leap-frog algorithm

`Setup:`

$$\begin{aligned} \tau &= 0 && (9)\\ \vec{x}(0) &= \text{position} && (10)\\ \vec{v}(0) &= \text{velocity} && (11)\\ \vec{v}(0+\frac{\delta}{2}) &= \vec{v}(0) + \vec{a}(\vec{x}(0)) \times \frac{\delta}{2} && (12) \end{aligned}$$

`Loop:`

$$\begin{aligned} \tau &= \tau + \delta && (13)\\ \vec{x}(\tau) &= \vec{x}(\tau - \delta) + \vec{v}(\tau - \frac{\delta}{2}) \times \delta && (14)\\ \vec{v}(\tau) &= \vec{v}(\tau - \frac{\delta}{2}) + \vec{a}(\vec{x}(\tau)) \times \frac{\delta}{2} && (15)\\ \vec{v}(\tau + \frac{\delta}{2}) &= \vec{v}(\tau) + \vec{a}(\vec{x}(\tau)) \times \frac{\delta}{2} && (16) \end{aligned}$$

Note that if the force is not purely a function of the position $\vec{x}$, but also depends on the velocity $\vec{v}$, as is the case with a magnetic field, then the insertion of an "old" velocity is required to evaluate the acceleration.

7 Applications

During this course you will develop an object oriented code to integrate Newton's laws for one, two, three and also many particle systems.

The programming required will draw on the Physics 2 Scientific Programming class, and where additional knowledge is required it will be highlighted in the course material.

Specifically the systems looked at will include

7.1 Driven Simple Harmonic Motion

$$m\ddot{x} = -kx - b\dot{x} + F\cos\omega t \tag{17}$$

$$\ddot{x} = -\omega_0^2 x - \tilde{b}\dot{x} + \tilde{F}\cos\omega t \tag{18}$$

where $\tilde{b} = \frac{b}{m}$ and $\tilde{F} = \frac{F}{m}$. Subsituting for steady state solutions

$$x = A\cos(\omega t + \phi) \tag{19}$$

yields

$$\tan\phi = \frac{\tilde{b}\omega}{\omega^2 - \omega_0^2} \tag{20}$$

and

$$A = \frac{\tilde{F}}{\sqrt{(\omega^2 - \omega_0^2)^2 + \omega^2 b^2}} \tag{21}$$

For non-zero damping b, all solutions of the undriven case (i.e. transients, or, complementary functions of the ODE) can be written as sums of terms of the form $Ae^{\alpha t}$ where α is complex, and satisfies:

$$\alpha = \frac{-\tilde{b} \pm \sqrt{\tilde{b}^2 - 4\omega_0^2}}{2} \tag{22}$$

Thus with damping, the transients die out exponentially. There are three cases for α

- $b < 2\omega_0$: α has a oscillatory imaginary component and the system is underdamped
- $b = 2\omega_0$: $\alpha = -\omega$ and the system is critically damped
- $b > 2\omega_0$: α has two real roots.
  Long time behaviour is determined by the smaller root and the system is overdamped.

Usually (initial conditions $x \neq 0$, $\dot{x} = 0$) critical damping will yield no overshoot and the (typically) fastest approach to zero of all of the above cases.

It is left as an interesting exercise for the student to find the (contrived) case that allows a faster return to equilibrium (hint: it is in the heavily damped region, and can be seen by looking at α particle stopping).

7.2 Molecular dynamics

Simulations involving many interacting atoms or molecules are often called *molecular dynamics* simulations.

They model the system as a set of classical point particles with some inter-particle long-range potential (Coulomb force would be the potential for charged particles, and models of screened potentials such as the Lennard-Jones potential used in the un-charged case).

This classical model is integrated in simulated computer time, and statistical properties of the medium extracted (e.g. finding pressure vs T for a non-ideal gas).

The same algorithm can be used in a number modern simulations both including traditional MD (e.g. protein folding in solution), through to diverse simulations such as colliding galaxies and even in Quantum Chromo-dynamics. The name has stuck, and now is often when the algorithm is applied in all of the above classes of simulation even though out of context it might be a misnomer.

In this course you will apply molecular dynamics by calculating the inter-particle forces for a number of potentials. To keep the computer numerics in the range of a finite precision computer, we will *rescale* our units to keep the numbers of order 1.

- Gravity. Here we will take units of distance and time such that $G = 1$ and the solar mass is 1

$$F = -\frac{Gm_1m_2}{r^2}\hat{\mathbf{r}}$$

- Coulomb's law. Here we will take units where $E = 1$, and the electron charge is 1

$$F = \frac{Eq_1q_2}{r^2}\hat{\mathbf{r}}$$

- Lennard-Jones potential. Here we will take a potential where

$$V(r) = U\left[\left(\frac{a}{r}\right)^{12} - \left(\frac{a}{r}\right)^{6}\right]$$

$$F = \frac{U}{a^2}\left[12\left(\frac{a}{r}\right)^{14} - 6\left(\frac{a}{r}\right)^{8}\right]\vec{r}$$

When implementing the Coulomb and Gravitational force on a computer it is typically easier to evaluate the force *without* computing the unit vector via

$$\frac{1}{r^2}\hat{r} = \frac{\vec{r}}{r^3}$$

This sort of force calculation when dealing with many particles has been behind the IBM BlueGene supercomputer line for protein folding. Also, the Gordon Bell prize winning Japanese (academic) GRAPE (GRAvity PipelinE) special purpose supercomputer line has been designed to hardware accelerate precisely this radial force and position velocity update calculation in galactic dynamics and protein folding.

The leap frog integration algorithm is also central to the QCD simulation algorithms run on the 14,000 processor, 10TFlop/s QCDOC supercomputer designed and built by the Edinburgh PPT group in collaboration with Columbia University, NY and IBM Watson.

8 Java

This course builds on the Physics 2 Scientific Programming course.

As such it is expected that you will be familiar with the basics of the Java programming language.

In addition to teaching you computational methods, this course is also intended to expand your base of knowledge of modern object oriented programming in general and Java in particular. Another key aim is to give you the programming vocabulary habits required to successfully carry out a computational Honours project.

The key areas in which this course is intended to enhance your skills are

- Careful design of code to minimise the amount that has to be written.
- Multi-file compilation and code reuse.
- File I/O.
  It is particularly important for projects that you be able to write programmes that can read in parameters and save their results.
  This makes for much more flexible programmes and the preparation of graphs of publishable quality requires the use of graphing packages such as XMGRACE.
- Input parsing is required for most complex forms of file I/O.
- Advanced students will learn the use of JavaBeans XML serialisation.

- Inheritance (whereby a class can extend and reuse another without either copying or modifying the original code).
- Alternative model solutions will be made available highlighting more advanced constructs (function objects, virtual methods etc..)

Due to the nature of programming it is not overly beneficial to give pedagogical notes on language features. Throughout the course we attempt to teach by giving simple relevant examples that encapsulate the point and serve as a starting point for the students own code development.