

2 Numeric Integration.

2.1 Introduction.

Differential equations are equations for an unknown function which contain one or more of the derivatives of the unknown function. Almost all interesting dynamical systems have a differential equations describing them, so finding solutions – the functions which satisfy the given equations – has been the focus of considerable mathematical study for centuries. Differential equations divide naturally into several different categories which correspond nicely to how difficult they look, and how difficult they actually are.

Ordinary differential equations are those that describe a function which has only one independent variable, often written as $y(t)$ or $x(t)$ or $y(x)$. Partial differential equations describe a function with several independent variables, looking like $u(x, y)$ or $v(t, x)$ or so. Typically, ordinary differential equations are the easier to solve.

Differential equations are called *linear* if the unknown function (call it $y(t)$ for convenience) and its derivatives never appear in the differential equation in any way other than raised to the first power, possibly multiplied by a function of t – so that, for example, $y''' - 4y' + 2y = 0$ is linear; $y' = y^2$ is not; $ty' + 2y = t$ is linear; $y'' + \sin(y) = 0$ and $y'' + y'y + y = 0$ are not. Generally linear differential equations are easier to solve than nonlinear ones – usually, nonlinear differential equations cannot be solved, and the few which can be solved require extensive clever reasoning to solve.

A differential equation with *constant coefficients* is as the name suggests – the unknown function y and its derivatives are multiplied only by constants, and not by functions of the independent variable t . $y'' + 4y = \sin(t)$ is a linear differential equation with constant coefficients; $y' - ty = t$ is a linear ordinary differential equation with *variable coefficients*. Many linear ordinary differential equations with constant coefficients are solvable.

An *homogenous* differential equation is one in which the independent variable never appears – everything can be written in terms of the unknown function $y(t)$. The equation $y'' + y = 0$ is homogenous; $y'' + y = \frac{1}{t}$ is not.

A linear, homogenous, ordinary differential equation can *always* be solved, *exactly*. Even better, they can be solved by a predictable, automatable routine. The routine has a few special cases and minor tricks to it, but it could be written down on a single sheet of paper.

Unfortunately, the realm of these equations is very narrow, usually simple

equations that do not get more complex than basic physics courses. The rocket equation is about the most complex and physically real system that can generally be solved exactly.

So one falls back on approximate solutions: if one has an equation which describes the relationship between a function and its derivatives, and one has an initial condition for the system, then one can try to evaluate the system numerically, and find an answer – a trace of the function’s values at a collection of times – which is within a tolerable margin of error of the exact function. (This makes the assumption the independent variable represents time. In the physical systems from which these systems were derived this was true; it does change the work if the independent variable represents something else.)

For the sake of convenience suppose that the differential equation to be solved can be written in the form $y' = f(t, y)$ – that is, the rate of change of the function depends on the current time (or whatever the independent variable represents) and the current value of the function. This may *appear* to be an absurd limitation on the problems that can possibly be solved this way. It is not, for a subtle reason: these techniques apply equally well to the equation $\vec{y}' = f(t, \vec{y})$, with \vec{y} a vector with as many components as needed. Further, incredibly, a differential equation that uses the first n derivatives of a function $y(t)$ can be rewritten as a differential equation involving only the first derivatives for an n -dimensional vector function $\vec{y}(t)$. This simple case solves most of the equations one needs.

Usually one starts with a known initial condition, typically written as $y(t_0) = y_0$ (which makes the assumption that the system being modeled begins at $t = t_0$, but starting at a different value of t changes nothing substantial). One has a differential equation describing the derivative $y'(t) = f(t, y)$, and one would like to know $y(t)$ at a representative variety of values of t , the set of times t_1, t_2, \dots, t_n .

So take a time step – often called Δt or h – representing the difference between one examined time t_i and the next time t_{i+1} . This time step does not need to be the same between every adjacent pair of times – there are some problems best done with a variable time step – but for this project the time step will be supposed to be constant.

Numerical integrators are called *self-starting* if they require only the value of the desired function at one time in order to calculate the next time step. There are *non-self-starting* methods which use some of the function’s history, the value at several recent time steps, which provide some numerical advan-

tages but which require either more initial data or that the program fall back on a second integrator method to generate the needed history. Self-starting methods will be described in this assignment.

2.2 Euler Integrators.

Suppose one has an initial condition $y(t_i) = y_i$, and the differential equation $y' = f(t, y)$. If one wants to approximate the value for $y_{i+1} = y(t_{i+1}) = y(t_i + h)$, the value of the function after one time step, what would be the most first, most straightforward guess one would make for it?

Since

$$y' = \frac{dy}{dt} = f(t, y)$$

and one knows that $y(t_i) = y_i$, then one would expect that

$$\begin{aligned} y(t_i + h) &= y(t_i + \Delta t) = y_i + \left(\frac{dy}{dt} \right)_{(t_i, y_i)} \Delta t \\ y_1 &= y(t_i + h) = y_i + h \cdot f(t_i, y_i) \end{aligned}$$

and this is the *first-order Euler integrator*. It is first-order because the time step h is only raised to the first power (meaning, among other things, that one expects if the time step were cut in half, the error – the difference between the integrator’s approximation to $y(h)$ and its real, exact, value – should be halved); it is an integrator because it solves the differential equation; it is credited to Leonhard Euler, the first person known to have described it.

This describes only how to get from time t_0 to time $t_0 + h$, but this is enough: to get to a much later time T requires only repeating this step, $\frac{T}{h}$ times over, the calculated value for $y_i = y(t_i)$ after a given step used as the starting point to find $y_{i+1} = y(t_{i+1})$.

The method has the advantage of overwhelming simplicity; it takes very little effort to remember or to reinvent it, and it can be coded in any computer language in a very short and completely transparent routine.

Its main disadvantage is that it is not very good – if the function $f(t, y)$ is anything more complex than a straight line is, it will make errors on each time step; and the error will (in general) grow. Numerical errors can be reduced by making the time step shorter, but not very efficiently. A tiny

time step must typically be used to get a very small error, which in turns requires much longer runs on the computer. With some ingenuity one can do better.

2.3 Euler Predictor-Corrector Integrators.

A potential way around one of the flaws of the Euler integrator can be demonstrated by considering how it would solve the differential equation $y' = 2t$, $y(0) = 0$, and thinking about the slope of the solution $y(t)$. The exact solution to this equation and initial condition is $y = t^2$.

Following the Euler integrator method one would find that when $t = 0$, the slope of the curve $y(t)$ is $y' = 0$. With a time step $h = 1$, one then projects that $y(1) = 0$. The slope of the curve $y(t)$ at $t = 1$ is $y' = 2$. In the interval between $t = 0$ and $t = 1$ the slope varied from 0 to 2.

If one supposed the slope of the curve $y(t)$, the rate of change y' , were the arithmetic mean of these two values – that it were $\frac{1}{2}(0 + 2) = 1$ during this interval, one projects the value of $y(1)$ to be 1, the correct value.

This highly promising result is blatantly contrived, of course – generally in interesting problems the slope does not depend only on time, and rarely will it be a simple linear function like that. It nevertheless presents the possibility of the predictor-corrector methods.

These methods involve two steps. In the first, the predictor stage, a projection y_{i+1}^p of where the curve will be at the next time step is made. This prediction may not be very close to where the function will actually be at the next time step, but it will typically be somewhere in the vicinity, and evaluating the slope $f(t_{i+1}, y_{i+1}^p)$ at this predicted value gives some idea of how the slope changes between t_i and t_{i+1} .

In the corrector step, the predicted value of the slope at the next time step is used, averaged with the initial step's value for the slope, and this generates a corrected next time step y_{i+1}^c . This corrected step is typically more accurate than just the predicted step would be. (One may repeat the corrector step, using this value to get another estimate for the slope at y_{i+1} – and one may repeat again, as many times as desired – but usually this will not increase accuracy enough to be worth the extra computation time.) The corrected projection becomes y_{i+1} , the estimated value for $y(t_{i+1})$.

In the Euler Predictor-Corrector, an Euler method is the prediction step.

Suppose $y' = f(t, y)$, and one has an initial value $y(t_i) = y_i$ and time step h .

Prediction Step.

$$y_{i+1}^p = y_i + h \cdot f(t_i, y_i)$$

Correction Step.

$$y_{i+1}^c = y_i + h \cdot \frac{1}{2} (f(t_i, y_i) + f(t_{i+1}, y_{i+1}^p))$$

Note that in both steps the method is that the predicted new value is the value the function had at its start, plus the time step times a slope; the difference is how the estimated slope is calculated.

This method is of order h^2 . One expects that cutting the time step in half will reduce the error to $\frac{1}{4}$ its original value.

An appealing property of Predictor-Corrector methods is that they tend to be self-correcting – small numerical errors are cancelled out and the numerical integration snaps towards the exact value. This comes from the corrector step. As a consequence, for example, one has order h^2 accuracy from a method which superficially only uses the time step h raised to the first power.

2.4 Runge-Kutta Integrators.

“Runge-Kutta” describes a family of integration schemes all built around a similar theme – that the exact solution $y(t)$ will be a continuous function and so can be approximated by a Taylor polynomial in terms of t . To go from that fact to an integrator is a long but not inherently difficult derivation, but the details of it can be spared.

Runge-Kutta methods exist for any order of the time step h desired. Higher order methods give better results – one can use larger time steps without exceeding the acceptable error margin, and so use fewer time steps to get to a desired finish time T – but they also require much more computation on each step, as each time step requires calculating more subsidiary functions, so it is easy for the total computation time to grow larger than the total time a lower-order method would require.

As a compromise most routine work uses a fourth-order Runge-Kutta integrator – the time step is used in powers up to h^4 and halving the time step commonly will reduce the error to $\frac{1}{16}$ of its initial value. One particular method, described below, is so universally favored it is often described as

“the” Runge-Kutta method. There do exist other methods, though, and some work may require one of these other Runge-Kuttas.

Suppose one has the initial condition $y(t_i) = y_i$, and a given time step h ; then one can find an approximate value for $y_{i+1} = y(t_i + h)$ by this calculation.

First, define the quantities

$$\begin{aligned}\delta_1 &= hf(t_i, y_i) \\ \delta_2 &= hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}\delta_1\right) \\ \delta_3 &= hf\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}\delta_2\right) \\ \delta_4 &= hf(t_i + h, y_i + \delta_3)\end{aligned}$$

Then the next time step for y is:

$$y_{i+1} = y_i + \frac{1}{6}(\delta_1 + 2\delta_2 + 2\delta_3 + \delta_4)$$

If one looks carefully and skeptically enough at this, some resemblance to Predictor-Corrector methods may be seen. It is not a trick of the light; the Euler Predictor-Corrector method is also a second-order Runge-Kutta method, and can be derived the same way fourth or other order methods are.

2.5 Symplectic Integrators.

The methods described above are designed to solve any differential equation. As a result they are not optimized to solve dynamics problems – for example, in a physical system, the coordinates of all the particles in the system can only be combinations of values that conserve the energy and the linear and angular momentums. This information is not in the above integrators. If it is considered at all, the physical constants may be measured and the time step adjusted to try keeping the energy and momentums within acceptable error bounds, or possibly to just halt the simulation if they get outside a tolerable error margin. Even in the best of circumstances, this usually means an integrator has a finite interval of time before its predictions become meaningless.

Many of these weaknesses were addressed by the creation of *symplectic* integrators. These are integration techniques for a Hamiltonian function $H(q, p)$ (where, as above, q and p may be vectors – this requires putting arrows over the letters in one’s notes, and turning the variables from single elements to arrays in one’s code, but does not make the work substantially different). If one has an initial condition $q(t_i) = q_i$, $p(t_i) = p_i$, then one will accept a proposed (q_{i+1}, p_{i+1}) only if $H(q_{i+1}, p_{i+1}) = H(q_i, p_i)$.

While this method will still make errors – if nothing else, running it on a computer forces the use of floating point arithmetic, which is an imperfect representation of the real numbers – the genius of the system is that the projected point (q_{i+1}, p_{i+1}) may not be at the point $(q(t_{i+1}), p(t_{i+1}))$ of the exact solution, it will fall somewhere close to it, along the curve $(q(t), p(t))$ of the exact solution. Errors tend to be errors of phase, of position along the trajectory, rather than of the shape of the trajectory. This makes symplectic integrators very well suited to projecting long-term behaviour of systems, such as the motion of the planets in a star system over the course of billions of years (which is the problem for which symplectic integrators were designed – is the solar system stable? Will the perturbative effects of Jupiter and the other planets change the Earth’s orbit significantly over the course of the next five billion years? This question is still not satisfactorily answered).

Unfortunately, the tremendous gain in confidence in the answers one gets by using symplectic integrators comes at the price of computation time: each step takes more work, and to get the most accurate results one must use a function $K(q, p)$ modified from the Hamiltonian $H(q, p)$.

One can write the function $K(q, p)$ as the sum of a series which looks much like a Taylor polynomial built with ever-increasing powers of the time step Δt : $K(q, p) = \sum_{m=0}^{\infty} \frac{\Delta t^m}{m!} K_m(q, p)$. This can be truncated on whatever order of the time step one wants.

Given the terms $K_m(q, p)$ for a given Hamiltonian, and initial condition $q(t_i) = q_i$, $p(t_i) = p_i$, one finds the next time step $q(t_{i+1}) = q_{i+1}$, $p(t_{i+1}) = p_{i+1}$ by solving the equations:

$$p_{i+1} = p_i + \sum_{m=1}^{\infty} \frac{\Delta t^m}{m!} \frac{\partial K_m}{\partial q}(p_i, q_i)$$

$$q_{i+1} = q_i - \sum_{m=1}^{\infty} \frac{\Delta t^m}{m!} \frac{\partial K_m}{\partial p}(p_i, q_{i+1})$$

where

$$\begin{aligned} K_1 &= H \\ K_2 &= -\frac{\partial K_1}{\partial q} \cdot \frac{\partial H}{\partial p} \\ K_3 &= -\frac{\partial K_2}{\partial q} \cdot \frac{\partial H}{\partial p} + \sum_{i,j=1}^n \frac{\partial K_1}{\partial q_i} \frac{\partial K_1}{\partial q_j} \frac{\partial^2 H}{\partial p_i \partial p_j} \end{aligned}$$

and the higher order values for K_m are vastly more complicated functions. Note that this definition, as written, will work for n -dimensional vectors for q and p . The multiplications in K_2 and the first term in K_3 are dot products, if the coordinates q and p are vectors.

Notice a significant problem: the equation for q_{i+1} is *implicit*; the value q_{i+1} one wants (typically) cannot be separated onto one side of the equation.

Solving implicit equations can be done by a variety of techniques. The simplest one to program, and one which will work if the time steps are sufficiently small, is an iterative technique. Let $q_{i+1}^0 = q_i$ be the first guess one makes for the value of q_{i+1} . Then define:

$$q_{i+1}^{j+1} = q_i - \sum_{m=1}^{\infty} \frac{\Delta t^m}{m!} \frac{\partial K_m}{\partial p} (p_i, q_{i+1}^j)$$

and repeat, putting the next refined guess q_{i+1}^j back into the original equation until the numbers converge, until q_{i+1}^j is sufficiently close to q_{i+1}^{j+1} . This will not be perfect, but it can be made close enough to the correct q_{i+1} .

Symplectic integrators work remarkably well. Once the time step gets sufficiently small, the energy error – how much the Hamiltonian varies over the course of the simulator run – stays within bounds through its entire run.

2.6 Assignment.

Due 25 September.

Consider the differential equation

$$y' = \frac{dy}{dt} = y$$

with the initial condition $y(0) = y_0 = 1$. This is an exactly solvable differential equation.

What is the exact values of $y(10)$?

Write a program to numerically integrate this differential equation, with a separate function (or subroutine) set aside to calculate the change between one y_i and y_{i+1} . One should be able to make the same program be an Euler, Euler Predictor-Corrector, or Runge-Kutta method *just* by changing this one subroutine. (One may turn in a single program with the three separate integration functions appended to it; or one may decide to write one program capable of using any of these integration techniques, as selected.) The function must be able to integrate from $t = 0$ to an input time of T .

How small a time step h is needed with each of these three integration techniques to get a value for $y(10)$ accurate to within five percent? To within one percent?

Can a symplectic integrator be used for this function? Does there exist a Hamiltonian for which $q' = \frac{dq}{dt} = q$? If there is, write this Hamiltonian, and program a symplectic integrator to evaluate it. (Note that p does not have to behave like t ; it does not need to increase at a constant rate.)

How many of the terms K_m are needed for this integrator? What justifies the number of terms used? How accurate is the estimated $y(10)$ with $h = 0.1$? With $h = 0.01$? Is this the accuracy expected? How well is the Hamiltonian conserved?

If a symplectic integrator can not be used, explain why not.

Due 9 October.

Consider a Hamiltonian representing a simplified version of the solar system: let (\vec{q}^1, \vec{p}^1) represent the momentum and position of the Earth in the two-dimensional plane; let (\vec{q}^2, \vec{p}^2) represent the momentum and position of Jupiter. Suppose that the Sun is a motionless mass at the origin.

Assume that the Earth has mass 1, and a circular orbit with radius 1 to begin. Assume Jupiter has mass 318, with a circular orbit of radius 5. What should the Hamiltonian describing their motion look like, and what is a sensible initial position and momentum for both planets? (The sun's mass, on this scale, is about 330,000.)

Write a symplectic integrator for this problem. Find a time step h sufficiently small that on a few orbits, the simulated Earth and Jupiter do trace out reasonably exact circles in their orbits. (Jupiter should take approximately 12 times as long as Earth does to complete one orbit.)

How long can this simulator run? Is the Earth in this system stable – after a very long time, will its orbit remain circular, and will the radius of its orbit remain very nearly 1?