Chapter 6:
Newton's Laws: Sums, Integrals, and Orbits

Goals:

- To use numerical methods to characterize the motion of particles governed by Newton's laws.
- To understand inheritance of classes in Java.

In this chapter we examine systems described by Newton's laws. In large part, the goal of this chapter is to teach some numerical techniques needed to solve the resulting equations—evaluating infinite sums and integrals, and also solving simple ordinary differential equations. However, there is another element on the agenda here. We would like you to learn something about hypothesis testing on the computer. Computational physics is aimed at asking questions about the behavior of some mathematical model of the real world, and thereby questions about the world itself. Hence, computational physicists are forever defining techniques for answering questions like "If one assumes xxx of our model, then is yyy true?". This chapter will involve a little of this kind of questioning.

A. Some Classic Mechanics Problems.

A.1. The Pendulum. At one level, classical mechanics started with Galileo and his observations of the simple pendulum. We too start with the pendulum. Let a mass M be held by a rod of length L, and let θ be the angle that the rod makes with the vertical direction (see figure 6.1). Then the kinetic and potential energies are given respectively by $M (L\, d\theta/dt)^2/2$ and $MgL(1 - \cos\theta)$, where g is the

* * *

gravitational acceleration. If the pendulum swings freely (no damping), its total energy, E, is constant:[1]

$$E = \frac{1}{2} M\left(L \frac{d\theta}{dt}\right)^2 + MgL(1-\cos\theta) = \text{constant} \ . \qquad (6.1)$$

One can integrate equation (6.1) to obtain the time that the pendulum must take in going from θ-value θ_0 to a larger value θ_1 by having θ increase, i.e. by having dθ/dt be greater than zero. Start by solving eq. (6.1) for $\frac{d\theta}{dt}$:

$$\frac{d\theta}{dt} = \sqrt{\frac{2}{ML^2}\left(E - MgL(1-\cos\theta)\right)} = \sqrt{\frac{2g}{L}\left(\frac{E}{MgL} + \cos\theta - 1\right)} \qquad (6.2)$$

(we can take the positive square root because we assume θ is increasing). Integrating this gives the time to get from one angle to another:

$$t = \sqrt{\frac{L}{2g}} \int_{\theta_0}^{\theta_1} \frac{d\theta}{\sqrt{\frac{E}{MgL} + \cos\theta - 1}} \ . \qquad (6.3)$$

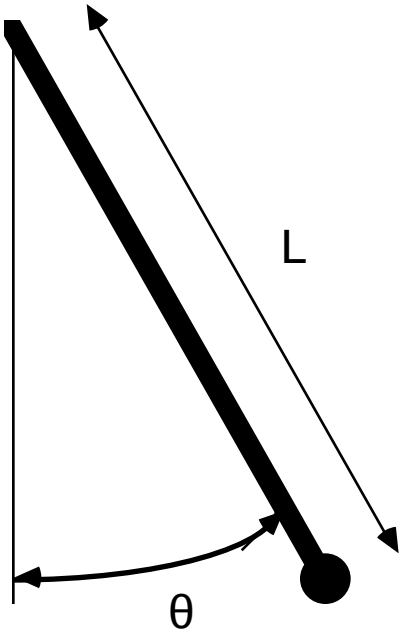
The pendulum must have energy $E > 0$. For $E < 2MgL$, the pendulum oscillates back and forth; for $E > 2MgL$, it turns over and over with $\frac{d\theta}{dt}$ never going through zero.

---

[1]Differentiating equation (6.1) with respect to time yields the familiar equation of motion for a pendulum:

$$ML^2 \frac{d^2\theta}{dt^2} + MgL\sin\theta = 0 \ .$$

* * *



Pendulum

Figure 6.1 : The Pendulum

For now, we look at the case $E < 2MgL$ and notice that the maximum possible value of θ is θ_{max} which satisfies

$$\cos\theta_{max} = 1 - \frac{E}{MgL} \quad . \tag{6.4}$$

The period of oscillation is the time it takes for the motion to go from θ_{max} through zero to its minimum value through zero again and back to θ_{max} once more. This period is four times the time that it takes to go from zero to θ_{max}. Hence if we call the period τ, we have:

$$\tau = \sqrt{\frac{8L}{g}} \int_0^{\theta_{max}} \frac{d\theta}{\sqrt{\frac{E}{MgL} + \cos\theta - 1}} \quad . \tag{6.5}$$

Many of you may recognize that the 'natural frequency' of the pendulum, $\omega_0 \equiv \sqrt{\frac{g}{L}}$, sets the timescale for the period. It is said that Galileo believed that the period of the pendulum to be independent of E (or more likely θ_{max}, since Galileo probably predates the concept of energy). We wish to develop numerical methods for calculating the

* * *

integral (6.5) and seeing how τ depends upon E. Hence we must describe how to calculate integrals numerically.

A.2. Central Forces. In another view, Classical Mechanics started with Kepler and Newton and the Newtonian formulation of central force motion. In modern terms, one would say that a point particle moving through a potential V(r) which depends only upon the magnitude of r would have an energy:

$$E = M (dr/dt)^2 / 2 + L^2 / (2 M r^2) + V(r) . \qquad (6.6)$$

Here, L is the angular momentum given by

$$L = M r^2 d\phi /dt \qquad (6.7)$$

where ϕ is the azimuthal angle which describes the position of the particle. (See Figure 6.2). The coordinate r is measured from the 'center' of the potential V, which is spherically symmetric about the point r=0. The motion is confined to a plane (determined by the particle's momentum), so only one angular coordinate is necessary. In an isolated system, both E and L are conserved, i.e. they are independent of time. We wish to ask, once more, about the nature of the particle orbit. In particular, we would like to know whether the orbit is closed. (The particle eventually returns to its starting point).

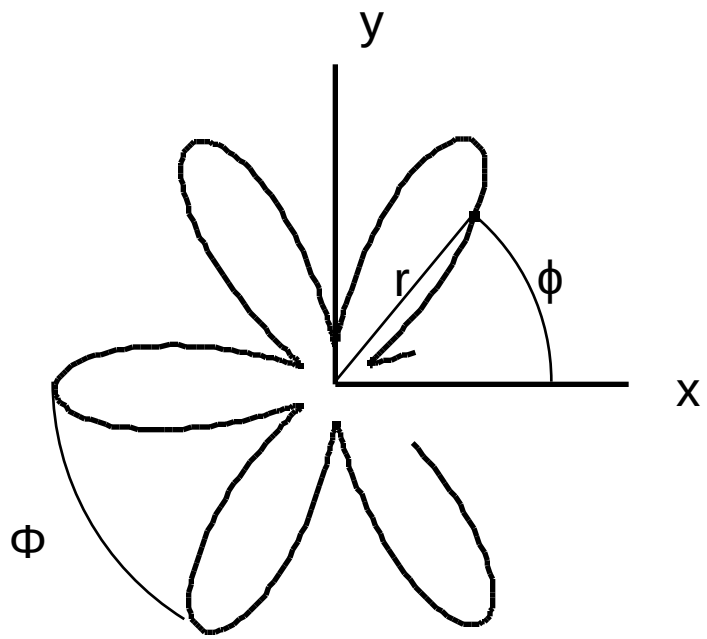


Figure 6.2. Orbit in Central Force

* * *

In a typical orbit, the particle moves in and out radially as it moves around, so that both r and ϕ are time-dependent. The orbit has a period for radial motion, which we denote T_r, and a period for angular motion, which we call $T\phi$. If T_r/T_ϕ is rational ($T_r/T_\phi = P/Q$, where P and Q are integers), then the orbit will close (because after a time $PT_\phi = QT_r$ the configuration repeats); if it is irrational it will not. Equivalently, one can consider the amount Φ by which ϕ advances during one period of the radial motion (T_r). If $\Phi/2\pi$ is rational the orbit will close; if it is irrational it will not. Consequently, we wish to use numerical methods to determine whether $\Phi/2\pi$ is rational.

To calculate Φ, we use a technique essentially similar to the one we employed in the pendulum problem. First, notice that $\frac{dr}{dt}\Big/\frac{d\phi}{dt}$ is the rate of change of r with ϕ, i.e. $\frac{dr}{d\phi}$. Hence, from Eq (6.7),

$$\frac{dr}{dt} = \frac{d\phi}{dt}\frac{dr}{d\phi} = \frac{L}{Mr^2}\frac{dr}{d\phi}.$$

With this substitution, Eq. (6.6) becomes:

$$E = \frac{L^2}{2Mr^4}\left(\frac{dr}{d\phi}\right)^2 + \frac{L^2}{2Mr^2} + V(r). \qquad (6.8)$$

By precisely the same logic as before, one finds the advance in ϕ as r increases from r_0 to r_1 by writing

$$\phi(r_1) - \phi(r_0) = \int_{r_0}^{r_1} dr\left[-r^2 + 2Mr^4\left(E - V(r)\right)/L^2\right]^{-1/2} \qquad (6.9)$$

Hence if one takes as the minimum and maximum values of r, r_{min} and r_{max}, one has the formula for the advance per cycle

$$\Delta\varphi = 2\int_{r_{min}}^{r_{max}} \frac{dr}{\sqrt{X(r)}} \qquad (6.10a)$$

where the integrand is given in terms of X(r) defined as

$$X(r) \equiv 2mr^4\left(E - V(r)\right)/L^2 - r^2 \qquad (6.10b)$$

and the extremal values of r are given by

* * *

$$X(r_{\min}) = X(r_{\max}) = 0 . \qquad (6.10c)$$

Our problem, then, is to do the integral (6.10) for some potential V(r) and see whether the result is a rational multiple of 2π. (See the Menu Project in this chapter for more on orbits in central potentials).

A.3. Damped motion in a potential. Now consider the motion of a particle of mass m in a potential well V(x) subject to a viscous damping force -γv, where v is the particle velocity, and a periodic driving force $F(t) = F_0 \cos(\omega t)$. The position of the particle is governed by Newton's equation of motion:

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} = -\frac{dV}{dx} + F_o\cos(\omega t) \qquad (6.11)$$

If the well is parabolic ($V(x) = kx^2/2$), then this system is just a damped harmonic oscillator, which is covered in every freshman mechanics course. We will consider the motion when the potential has the form $V(x) = -k_1x^2/2 + k_2x^4/4$ (shown in figure 6.3).

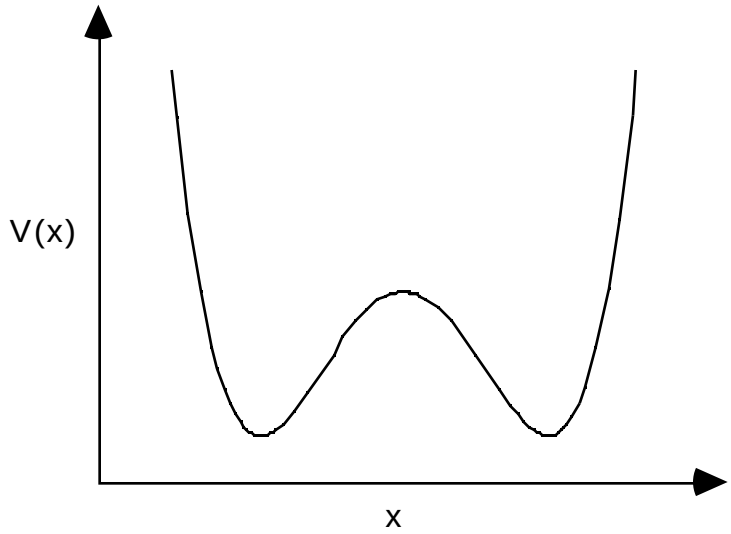


Figure 6.3. Double-Well (Quartic) Potential

For a given F_0 we would like to know whether at long times (after any transients have decayed) the motion of the particle is periodic.

* * *

In other words, we would like to know whether the orbits of this system close.

Exercise 6.1. Orbits of a simple harmonic oscillator . Show that when the potential is quadratic, the orbits of equation (6.11) eventually close for any non-zero F_0.

When the potential is quartic, answering this question is not nearly so simple. We cannot use conservation of energy to simplify the problem as we did in A.1 and A.2, because here energy is not conserved. So, below we will attack this problem by direct numerical integration of the differential equation (6.11).

B. Sums. We have already seen something about how to calculate derivatives numerically (see Chapter 3). Integrals, which are only a little tougher, are defined as limits of sums. Hence it is logical for us to begin with summation.

It is very easy to use the computer to find finite sums (and products) like:

$$Exp(x,N) \equiv \sum_{j=0}^{N} \frac{x^j}{j!} \qquad (6.12a)$$

(Here j! = j (j-1) (j-2)...(2) (1))

$$Gau(x,N) \equiv \sum_{j=-N}^{N} \exp\left(-\frac{1}{2} x j^2\right) \qquad (6.12b)$$

$$zeta(p,N) \equiv \sum_{j=1}^{N} j^{-p} \qquad (6.12c)$$

$$\sin(x,N) \equiv x \prod_{j=1}^{N} \left(1 - \left(\frac{x}{\pi j}\right)^2\right) \qquad (6.12d)$$

$$Geo(r,N) \equiv \sum_{j=1}^{N} r^j = \frac{1-r^{N+1}}{1-r} \qquad (6.12e)$$

The last of these can be recognized as the simple geometric series and the last equals sign gives the formula for summing such a series. Given enough time, a computer can always evaluate expressions such as these.

Exercise 6.2. Simple finite sums. Write functions which perform the sums in (6.12c) and (6.12d) for any value of the

* * *

arguments. (To do this you will need to use the method Math.pow.) Calculate zeta(1.5,N) and sin(1.5,N) for N equal to 1, 3, 10, 30, and 100. Do these quantities appear to converge to a limit as N goes to infinity? The notation suggests that in the limit as N goes to infinity sin(x,N) goes to sin(x). Find some evidence for or against this proposition.

Problem 6.1. Simple infinite sums . A function, which we wish to call Gau(x), is defined as the large-N limit of expression (6.12b)

$$Gau(x) \equiv \sum_{j=-\infty}^{\infty} \exp\left(-\frac{1}{2} x j^2\right) \qquad \text{for } x > 0 \tag{6.13}$$

As we have already seen, for reasonably large values of x, this sum converges extremely rapidly. When x is a small positive number, the sum converges slowly. However, there is a formula which enables one to calculate Gau(x) for positive x close to zero (the formula is exact for all x):

$$Gau(x) = \left(\frac{2\pi}{x}\right)^{q} Gau\left(\frac{4\pi^2}{x}\right) \tag{6.14}$$

Find some evidence (numerical or analytic) that would convince a reasonable person that equation (6.14) is correct, and determine q. Write a program which plots Gau(x) for x between zero and five. How big are the errors that you make in this calculation? Under what conditions is this error insignificant?

C. Integrals. A slightly harder case is provided by the calculation of an integral, for example,

$$I(c, c+\Delta) \equiv \int_{c}^{c+\Delta} f(x)\,dx \tag{6.15}$$

Consider first the case in which Δ is quite small so that we can assume that the function f(x) varies very little in the integration interval. In that case, one can replace f(x) by is average value in the interval, i.e. $f(x) \to (f(c) + f(c+\Delta))/2$ so that the integral is approximately evaluated as:

$$I(c, c+\Delta) = \Delta(f(c) + f(c+\Delta))/2 + O(f''(c)\Delta^3) \tag{6.16}$$

Here, O is a symbol meaning that the term in question (usually a correction to a useful result) is of the order of magnitude of the

* * *

value stated. This result for the error magnitude can be derived by expanding f(x) in a Taylor series about x=c.

Starting from the result (6.16), one can easily calculate an integral over a more extended region.

$$I(a,b) \equiv \int_a^b f(x)dx \qquad (6.17)$$

Divide the region between a and b in Eq (6.17) into N pieces each of size Δ, where Δ is again small. We represent the function f(x) by the values

$$f_j = f(a + j\Delta) \quad , \quad \text{where } N\Delta = b - a \qquad (6.18)$$

A summation of the N contributions to the integrand (each like (6.15) and (6.16)) coming from subintervals of size Δ gives

$$I(a,b) = \Delta \sum_{j=1}^{N-1} f_j + \Delta(f_0 + f_N)/2 + O(f''(x)N\Delta^3).$$

(trapezoidal integration formula) (6.19)

The correction term in Equation (6.19) involves an error which is proportional to $f''(x)$, i.e. to the second derivative of f someplace in our interval.

Exercise 6.3. Definite Integrals. To check out this method, calculate numerically the integral between 0 and 1 of the following simple functions: x^3 , cos(x) , and exp(x). Use several different values of N and see if you can get some idea of how the error diminishes as N increases.

Problem 6.2 . Modify the applet you used to do the integrals in exercise 6.3, so that it integrates $(x(1-x))^{-1/2}$ from 0 to 1 using Eq. (6.19). How does the error depend upon the choice of Δ (please be quantitative)?

(Note: we expect you to do the integral in the form written. No fancy substitutions, please. You can, however, check your result by considering the substitution

$$x = (1 + \cos\theta)/2 \qquad (6.20)$$

which does reduce the integral to a trivial form.)

Exercise 6.4. The zeta function . The quantity

* * *

$$\varsigma(p) = \lim_{N \to \infty} zeta(p, N) \equiv \sum_{j=1}^{\infty} j^{-p} \qquad (6.21)$$

is called the Riemann zeta function. It is of fundamental importance in analytic number theory. The summation (6.21) is far, far less well-behaved than the one in Equation (6.13). To see this one can use the finite-N expressions which you generated in Exercise 6.2 to estimate the zeta function itself. In this way, please estimate zeta(p), for p equal to 1.25, 2, 2.5, and 4. To four decimal places these sums are equal to 4.5951..., 1.6449..., 1.3415..., 1.0823.... How well can you do in calculating the values of these sums? Which of your estimates is worst? Why?

You can improve your estimates by including in a rough fashion the left-out terms in the summation. For large N, it is true that the sum $\sum_{j=N}^{\infty} j^{-p}$ is closely approximated by the integral $\int_{N}^{\infty} j^{-p} dj$:

$$\sum_{j=N}^{\infty} j^{-p} \approx \int_{N}^{\infty} j^{-p} dj = \frac{1}{(p-1)N^{p-1}} \ . \qquad (6.22)$$

This is because the sum has contributions from many terms, and these terms change slowly with j. Given Eq. (6.22), one can improve the convergence of the calculation of expression (6.21) in a marked fashion by summing explicitly up to some N and using the integral to approximate the rest of the terms.

Problem 6.3. It is true that for each integer m, ζ(2m) is equal to a simple rational number times an integral power of π. Write an applet that computes ζ(2), ζ(4), and ζ(6) and thus figure out which power and which rational number for the values m = 1, 2, and 3.

Problem 6.4. The period of the pendulum. Create an applet to calculate the period of the pendulum (Eq. 6.4 and 6.5) for $\frac{E}{mgL}$ = 0.01, 0.1, 1, 1.5 and 1.99. Express your results in terms of the 'natural timescale' of the pendulum (i.e. compute $\tau\sqrt{g/L}$). To get accurate results, you should not use the variable θ as the integration variable, but instead make a transformation analogous to the one in Eq (6.20)

$$\theta = \theta_{max} \cos\varphi .$$

What do you gain from this change of variables?

* * *

Menu Project. Periodic and Non-periodic Orbits in a Central Potential. By doing the integral in Eq (6.10) one can see whether for a given potential and given values of E and L Φ is a rational number. If it is rational, then the orbit closes; if not the orbit never repeats itself. One might then ask the question about whether there are any forms of the potential V(r) which permit the orbit to be closed for all E and L. However, once V(r) is fixed we know that Φ is a function of E and L, which we then write as Φ(E,L). Some applications of theorems derived from calculus indicate that the function is continuous. Hence it can only be rational by being constant, independent of E and L.

It turns out that Φ(E,L) is constant only for very special potentials, those which vary as a power of the distance:

$$V(r) = -\alpha r^{-\alpha} . \qquad (6.23)$$

One can recognize two familiar special cases, α=1, which is the attractive gravitational force, and α=-2, which is the harmonic oscillator. Compute the value of Φ(E,L) for these two cases and find out whether the orbit closes. Then do the same for α=3. Plot up a few orbits to show what is going on.

Notice that the integrals involved must be computed carefully because of their singularities at the two endpoints. To do them accurately, one should first compute r_{min} and r_{max} and then make a transformation like that in Problem 6.4 to eliminate the singularities at the endpoints.

D. Integrating First Order Differential equations. The integration of differential equations is not so different from just doing simple integrals. Consider an ordinary differential equation problem, which is stated in the form:

$$\frac{dx}{dt} = G(x,t); \quad x(t=0) = 0 \qquad (6.24)$$

* * *

Equation (6.24) can be integrated numerically by replacing the 1st derivative with a finite difference: $\frac{dx}{dt} \approx \frac{x(t+\Delta) - x(t)}{\Delta}$, where Δ is a small step in t. So our algorithm looks like

$$x(t+\Delta) = x(t) + \Delta G(x,t) \qquad (6.25).$$

It is common practice to define the discrete variables $t_j \equiv j\Delta$ and $x_j \equiv x(t_j)$. Then the algorithm of eq. (6.25) turns into

$$x_{j+1} = x_j + \Delta G(x_j, j\Delta) \ . \qquad (6.26)$$

At this point we have basically replaced our differential equation with a discrete (but time-dependent) mapping function $G(x_j, j\Delta)$.

Exercise 6.5. Solving a first-order differential equation. Use numerical methods to solve the differential equation

$$\frac{dx}{dt} = -t^2 x^2,$$

with the initial condition $x(t=0) = 10$. What happens to x as t approaches infinity? Can you find a large t limiting form? (Hint: Make a log-log plot of x versus t. Look for a straight line behavior. What data does the slope give you?) Now can you compute a numerical solution for t < 0. At some point there is a disaster. When? What happens near the disaster? Can you now figure out an exact solution? How well does the exact solution agree with the numerical one? How does the accuracy of your result depend upon the value of Δ which you use in your numerical solution?

E. Integrating Differential equations: higher order equations. Higher order differential equations can also be attacked by very much the same method. Consider a second order equation like the one for a particle in a quartic potential:

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} = k_1 x - k_2 x^3 + F_0 \cos\omega t. \qquad (6.27)$$

One can write Equation (6.27) in terms of a pair of first order equations by using the momentum, p, as an additional variable, and then writing two first order equations, namely

$$\frac{dx}{dt} = \frac{p}{m}$$

* * *

$$\frac{dp}{dt} + \frac{\gamma}{m} p = k_1 x - k_2 x^3 + F_0 \cos \omega t \ . \qquad (6.28)$$

Exercise 6.6. Solving a second-order differential equation.
Please write an algorithm (but not a program) which will solve the pair of first order differential equations:

$$\frac{dx}{dt} = U(x, y; t)$$
$$\frac{dy}{dt} = V(x, y; t) \qquad (6.29)$$

Assume that x and y are given at t = 0 and that the step-length, Δ, is also given.

We did not ask you to write a program that implements your algorithm for solving equations (6.29) because we would like you to use a more accurate method of solution called the Runge-Kutta method, in which the functions U and V are evaluated at several places chosen to minimize the errors. We illustrate the process for the second order Runge-Kutta method, which is the simplest one. Let

$$\mathbf{z} \equiv \begin{pmatrix} x \\ y \end{pmatrix}$$

stand for the two coordinates x and y. Then equation (6.29) is in the form

$$\frac{d}{dt}\mathbf{z} = \mathbf{f}(\mathbf{z}, t), \qquad (6.30)$$

with $\mathbf{f}(\mathbf{z}, t) \equiv \begin{pmatrix} U(\mathbf{z}, t) \\ V(\mathbf{z}, t) \end{pmatrix}$.

If Δ is a small step size, then:

$$\mathbf{z}(t + \Delta) - \mathbf{z}(t) = \Delta \mathbf{f}(\mathbf{z}(t + \Delta/2), t + \Delta/2) + O(\Delta^3) \ . \qquad (6.31)$$

Note that if the Δ/2 were lacking on the right hand side the approximation would have an error of order Δ^2. As (6.31) stands it has an error of order Δ^3.

Since the right side of equation (6.31) is already multiplied by a Δ, we can evaluate $\mathbf{z}(t + \Delta/2)$ to within an error of order Δ and still get

* * *

a result good to order Δ^2. Hence an appropriate algorithm for using (6.31) is to let $\mathbf{z}_j = \mathbf{z}(t_o + j\Delta)$ and write

$$\mathbf{h}_j = \mathbf{f}(\mathbf{z}_j, t) \tag{6.32a}$$

$$\mathbf{z}_{j+1} = \mathbf{z}_j + \Delta\mathbf{f}(\mathbf{z}_j + \Delta\mathbf{h}_j/2, t + \Delta/2) \tag{6.32b}$$

Equations (6.32a) and (6.32b) (evaluated in that order) comprise the second order Runge-Kutta algorithm for determining the solution to equation (6.29). This method can be extended straightforwardly to higher order. It is popular to truncate it at fourth order (for reasons described in Numerical Recipes), so that the error is $O(\Delta^5)$. More sophisticated techniques, such as varying the step size Δ, can be applied to pathological equations for which straightforward Runge-Kutta methods are insufficient. You can find a good exposition on methods for solving differential equations at http://csep1.phy.ornl.gov/ode/ode.html; the section on the Runge-Kutta method is at http://csep1.phy.ornl.gov/ode/ode.html.

F. More on classes: Inheritance. Every dynamical system is a set of rules for going from one combination (z, t) to another combination $(\mathbf{z}(t+\Delta), t+\Delta)$. This definition applies both to discrete equations like the logistic map and to continuous differential equations like (6.27). For the logistic map, the rule is so simple ($f(x) = x(t+\Delta) = rx(1-x)$) that we just wrote a method to calculate f(x) and didn't bother to make a separate class. However, solving a system of differential equations requires several steps, so we will write a class `DiffEqSystem` to do it. Our class uses the Runge-Kutta method to integrate differential equation systems of the form:

$$\frac{d}{dt}\mathbf{z} = \mathbf{f}(\mathbf{z}, t), \tag{6.33}$$

where the function f and the number of components in the vector z are not known in advance. (Note that the Runge-Kutta formulas (6.32) work for any number of components of z.) That way we can use it for any equations of form (6.33). Now of course eventually we will need to know the actual equations that we wish to solve, but we'll get to that later.

Our first step is to define a class called a `VariableSet`, which bundles the combination (z, t) so that we don't need to explicitly keep track of them separately. This class is very simple; its methods mostly just set and get z and t.

* * *

```
class VariableSet  {

    private double t;
    private double[] x;

    VariableSet(double time, double[] xvars)       {       //  constructor
        setvars(time, xvars);
    }

    public void settime(double time){t = time;}

    public double gettime()  {return t;}

    public void setx(double[] x1)    {x = x1;}

    public double[] getx()    {return x;}

    public void setvars(double time, double[] xvars)       {
        settime(time);
        setx(xvars);
    }

    public void setvars(VariableSet v)  {
        settime(v.gettime());
        setx(v.getx());
    }

    public String toString() {
        return "t = "+", x = "+x[0]+", p = "+x[1];
    }
}                                  //  end of class VariableSet
```

Program 6.1 The VariableSet Class

Next we write a class DiffEqSystem, which contains a method nextvars, which calculates $(\mathbf{z}(t+\Delta),t+\Delta)$, given $(\mathbf{z}(t),t)$ and the vector function f in (6.30).

Because we don't know the form of f or even how many components z has, we do not have enough information to make a DiffEqSystem object. Therefore, DiffEqSystem is designated as an abstract class. We also designate the method timederiv (which defines f) as an

abstract method. An object of a subclass of DiffEqSystem can be created only if this method is defined in the subclass.

```
abstract class DiffEqSystem       {    //  abstract means that we can't
//  make any instances of this class--not surprising since we don't
//  yet know what differential equations we are going to solve

    protected VariableSet myvars;
    protected int n_var;           //  # of degrees of freedom
    protected double dt, dt2;      //  dt = time step; dt2 = dt/2

//  constructor
    DiffEqSystem()   {
    //  don't know about variables yet, so just set time step:
        setdt(0.3);
    }

    public VariableSet nextvars()   {    //  method that returns next VariableSet
        return new VariableSet(myvars.gettime()+dt, newx_rk4());
                          //  4th order Runge-Kutta
    }

    public abstract double[] timederiv(double t, double[] x) ;      //  abstract method
                       //  actual equations vary--defined only in subclass

    //  2nd order Runge-Kutta for time evolution
    private double[] newx_rk2()    {
        double x[] = myvars.getx();
        double t = myvars.gettime();
        double [] xtemp2 = new double[ n_var];
        //  xtemp2 is estimate of x halfway along interval where taking
        //  derivative leads to higher order error
        xtemp2 = arraysum(x, mult_by_cst(timederiv(t, x), dt2));

        return arraysum(x, mult_by_cst(timederiv(t + dt2, xtemp2), dt));
    }

    //  4th order Runge-Kutta for time evolution
    //  formula from p 551 of Numerical Recipes, or
    //      at http://csep1.phy.ornl.gov/ode/node7.html
    private double[] newx_rk4()    {

        double t = myvars.gettime();
        double x[] = myvars.getx();
```

* * *

```
        double[] k1 = new double[ n_var ];
        double[] k2 = new double[ n_var ];
        double[] k3 = new double[ n_var ];
        double[] k4 = new double[ n_var ];
        double[] xsum = new double[ n_var ];

        k1 = mult_by_cst(dt, timederiv(t, x));
        k2 = mult_by_cst(dt,
               timederiv(t + dt2, arraysum(x, mult_by_cst(k1, .5))));
        k3 = mult_by_cst(dt,
               timederiv(t + dt2, arraysum(x, mult_by_cst(k2, .5))));
        k4 = mult_by_cst(dt,
               timederiv(t + dt, arraysum(x, k3)));

        for(int i=0; i<n_var; i++)  {
           xsum[i] = x[i] + (1./6.)*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i]);
        }
        return xsum;
    }

//  a bunch of set and get methods:
    public void setdt(double deltat)  {
        dt = deltat;
        dt2 = dt/2.;
    }

    public double getdt() {return dt;}

    public void setvars(VariableSet v)  {
        if (v.getx().length == n_var) myvars = v;
        else System.out.println("Wrong number of variables");
    }

    public VariableSet getvars() {
        return myvars;
    }

    public void settime(double time){
        myvars.settime(time);
    }

    public double gettime()  { return myvars.gettime();  }
```

* * *

```
    //  some useful utility methods, used in Runge-Kutta methods:

    public double[] mult_by_cst(double constant, double array[])        {
        double[] temp = new double[ n_var ];
        for(int i=0; i< n_var; i++) {
            temp[i] = array[i]*constant;
        }
        return temp;
    }                           //  end of mult_by_cst

    public double[] mult_by_cst(double array[], double constant)        {
        return mult_by_cst(constant, array);
    }                           //  end of mult_by_cst

    public double[] arraysum(double array1[], double array2[]) {
                                //  add two arrays of length n_var
        double[] temp = new double[ n_var ];
        for (int i=0; i< n_var; i++) {
            temp[i] = array1[i] + array2[i];
        }
        return temp;
    }                           //  end of arraysum


}   //  end of class DiffEqSystem
```

Program 6.2 The `DiffEqSystem` Class

Now, finally, the class `DoubleWell`, a subclass of `DiffEqSystem`, specializes to the motion of a particle in a quartic potential. The subclass relationship, denoted by the declaration that `DoubleWell` "extends DiffEqSystem," means that `DoubleWell` automatically inherits the methods from the `DiffEqSystem` class (such as `nextvars`). The `DoubleWell` class then defines methods to set and get the system parameters, and to calculate the potential in which the particle moves. The constructor for `DoubleWell` sets the parameters of the motion, initializes the variables, and defines the time derivatives in `f`, so we can make instances of the `DoubleWell` class.

```
/*
    file DoubleWell.java
    class for calculating motion in double well potential
    extends class DiffEqSystem
*/
import java.awt.*;
```

* * *

```
public class DoubleWell extends DiffEqSystem       {
    private double A;       // parameter in double-well potential: V(x) = x*x(x*x-A*A)
    private double force;
    private double omega;
    private double mass;
    private double damping;

    public VariableSet resetvars;

//  constructor
    DoubleWell() {
        super();              // first call constructor of DiffEqSystem class
        A = 1.;               //  set default parameters
        force = 1.;
        omega = 2.*Math.PI*(0.2);
        mass = 1.;
        damping = 1.;

    //  set # of degrees of freedom and initial conditions
        n_var = 2;
        double resetarray[] = {-A, 0};
        resetvars = new VariableSet(0, resetarray);
        myvars = resetvars;
    }

//  constructor when initial conditions are given
    DoubleWell(VariableSet v)   {
        this();               //  first call no-argument constructor
        setvars(v);               //  set variables to those given
    }

    public double[] timederiv(double t, double[] xvec)     {       // equations of motion
        double x = xvec[0];                  // x = position
        double p = xvec[1];                  // p = momentum
        double[] temp = new double[n_var];// array [dx/dt, dp/dt]

        temp[0] = p/mass;                          //  dx/dt = p/mass
        temp[1] = ( force*Math.cos(omega*t) - damping * p/mass
                  -4.*x*x*x + 2*A*A*x );

        return temp;
    }
```

* * *

```
    //  methods for calculating potential in which particle moves
    public double potential(double x)     {  return x*x*(x*x - A*A);  }

    public double potential(double x, double t)       {
        return x*x*(x*x - A*A) - x*force*Math.cos(omega*t);
    }

    public double maxpotential(double xmin, double xmax){
            //  maximum value of potential in range [xmin, xmax]
        if(xmax < A)  { return 0.; }
        else    {
            if (Math.abs(xmax) > Math.abs(xmin)) {
                return potential(xmax);
            }
            else    { return potential(xmin); }
        }
    }

    public double minpotential(double xmin, double xmax){
        return potential(A/Math.sqrt(2));
    }

//  set and get methods for system parameters
    public void setA(double aparam) { A = aparam; }

    public double getA()  {return A;}

    public void setforce(double f)    { force = f; }

    public double getforce()  { return force;         }

    public void setomega( double o )  { omega = o; }

    public double getomega() { return omega; }
}
```

Program 6.3 The DoubleWell Class

Thus, we have constructed a very general class DiffEqSystem and then used inheritance to create the more specific subclass DoubleWell tailored to describe the motion of a particle in a double-well potential.

* * *

A sub-class of a super-class inherits all the public and protected methods and variables; that is, it has access to the methods and variables as if they were defined in the sub-class. In our example, the sub-class `DoubleWell` can access a method such as `getdt()`, which is defined in the super-class `DiffEqSystem`, as if it were defined in `DoubleWell` itself.

Often, methods in the sub-class overwrite existing methods in the super-class. For example, in the class `DiffEqSystem`, `timederiv` does nothing because we don't yet know what function we want to differentiate. In `DoubleWell`, it is overwritten, because now the equations of motion are known.

Problem 6.5. Motion of particle in a quartic potential. Write an applet that calculates the position x as a function of time for a particle in a double-well potential, with motion governed by:

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} = k_1 x - k_2 x^3 + F_0\cos\omega t. \qquad (6.34)$$

Assume that x and $\frac{dx}{dt}$ are given at t = 0 and that the step-length, Δ, is also given. Have your applet plot x(t) versus t. Feel free to use the classes `VariableSet`, `DiffEqSystem`, and `DoubleWell`, which are all in the file "DoubleWell.java" that you can find by going down the folder sequence "Chapter_6:Programs:DoubleWell."

Apply this program to the particular case of Eq. (6.34) in which $m = \gamma = 1$, $k_1 = 2$, $k_2 = 4$, and $\omega = \pi$. Use the initial conditions $\left.\frac{dx}{dt}\right|_{t=0} = 0$ and $x(0) = -1.2$. Determine whether at long times the motion appears to be periodic when $F_0 = 0.5$. How about when $F_0 = 1.0$? When $F_0 = 2.0$?

* * *