Chapter 3

Numerical Solutions

"The laws of mathematics are not merely human inventions or creations. They simply 'are;' they exist quite independently of the human intellect." - M. C. Escher (1898-1972)

So far we have seen some of the standard methods for solving first and second order differential equations. However, we have had to restrict ourselves to special cases in order to get nice analytical solutions to initial value problems. While these are not the only equations for which we can get exact results, there are many cases in which exact solutions are not possible. In such cases we have to rely on approximation techniques, including the numerical solution of the equation at hand.

The use of numerical methods to obtain approximate solutions of differential equations and systems of differential equations has been known for some time. However, with the advent of powerful computers and desktop computers, we can now solve many of these problems with relative ease. The simple ideas used to solve first order differential equations can be extended to the solutions of more complicated systems of partial differential equations, such as the large scale problems of modeling ocean dynamics, weather systems and even cosmological problems stemming from general relativity.

3.1 Euler's Method

In this section we will look at the simplest method for solving first order equations, Euler's Method. While it is not the most efficient method, it does provide us with a picture of how one proceeds and can be improved by introducing better techniques, which are typically covered in a numerical analysis text.

Let's consider the class of first order initial value problems of the form

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0. \tag{3.1}$$

We are interested in finding the solution $y(x)$ of this equation which passes through the initial point (x_0, y_0) in the xy-plane for values of x in the interval

$[a, b]$, where $a = x_0$. We will seek approximations of the solution at N points, labeled x_n for $n = 1, \ldots, N$. For equally spaced points we have $\Delta x = x_1 - x_0 = x_2 - x_1$, etc. We can write these as

$$x_n = x_0 + n\Delta x.$$

In Figure 3.1 we show three such points on the x-axis.

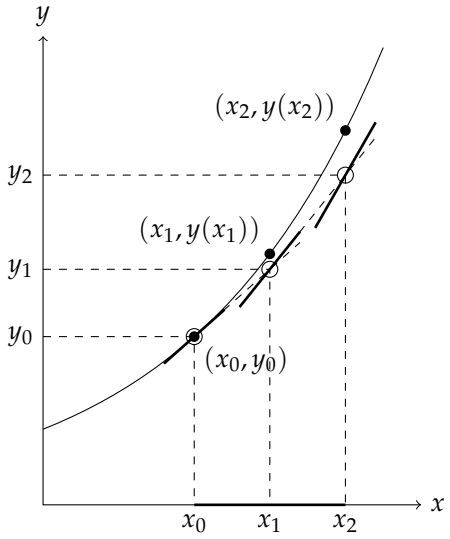


Figure 3.1: The basics of Euler's Method are shown. An interval of the x axis is broken into N subintervals. The approximations to the solutions are found using the slope of the tangent to the solution, given by $f(x, y)$. Knowing previous approximations at (x_{n-1}, y_{n-1}), one can determine the next approximation, y_n.

The first step of Euler's Method is to use the initial condition. We represent this as a point on the solution curve, $(x_0, y(x_0)) = (x_0, y_0)$, as shown in Figure 3.1. The next step is to develop a method for obtaining approximations to the solution for the other x_n's.

We first note that the differential equation gives the slope of the tangent line at $(x, y(x))$ of the solution curve since the slope is the derivative, $y'(x)'$ From the differential equation the slope is $f(x, y(x))$. Referring to Figure 3.1, we see the tangent line drawn at (x_0, y_0). We look now at $x = x_1$. The vertical line $x = x_1$ intersects both the solution curve and the tangent line passing through (x_0, y_0). This is shown by a heavy dashed line.

While we do not know the solution at $x = x_1$, we can determine the tangent line and find the intersection point that it makes with the vertical. As seen in the figure, this intersection point is in theory close to the point on the solution curve. So, we will designate y_1 as the approximation of the solution $y(x_1)$. We just need to determine y_1.

The idea is simple. We approximate the derivative in the differential equation by its difference quotient:

$$\frac{dy}{dx} \approx \frac{y_1 - y_0}{x_1 - x_0} = \frac{y_1 - y_0}{\Delta x}. \tag{3.2}$$

Since the slope of the tangent to the curve at (x_0, y_0) is $y'(x_0) = f(x_0, y_0)$, we can write

$$\frac{y_1 - y_0}{\Delta x} \approx f(x_0, y_0). \tag{3.3}$$

Solving this equation for y_1, we obtain

$$y_1 = y_0 + \Delta x f(x_0, y_0). \tag{3.4}$$

This gives y_1 in terms of quantities that we know.

We now proceed to approximate $y(x_2)$. Referring to Figure 3.1, we see that this can be done by using the slope of the solution curve at (x_1, y_1). The corresponding tangent line is shown passing though (x_1, y_1) and we can then get the value of y_2 from the intersection of the tangent line with a vertical line, $x = x_2$. Following the previous arguments, we find that

$$y_2 = y_1 + \Delta x f(x_1, y_1). \tag{3.5}$$

Continuing this procedure for all x_n, $n = 1, \ldots N$, we arrive at the following scheme for determining a numerical solution to the initial value problem:

$$\begin{aligned} y_0 &= y(x_0), \\ y_n &= y_{n-1} + \Delta x f(x_{n-1}, y_{n-1}), \quad n = 1, \ldots, N. \end{aligned} \tag{3.6}$$

This is referred to as Euler's Method.

Example 3.1. Use Euler's Method to solve the initial value problem $\frac{dy}{dx} = x + y, \quad y(0) = 1$ and obtain an approximation for $y(1)$.

First, we will do this by hand. We break up the interval $[0, 1]$, since we want the solution at $x = 1$ and the initial value is at $x = 0$. Let $\Delta x = 0.50$. Then, $x_0 = 0$, $x_1 = 0.5$ and $x_2 = 1.0$. Note that there are $N = \frac{b-a}{\Delta x} = 2$ subintervals and thus three points.

We next carry out Euler's Method systematically by setting up a table for the needed values. Such a table is shown in Table 3.1. Note how the table is set up. There is a column for each x_n and y_n. The first row is the initial condition. We also made use of the function $f(x, y)$ in computing the y_n's from (3.6). This sometimes makes the computation easier. As a result, we find that the desired approximation is given as $y_2 = 2.5$.

n	x_n	$y_n = y_{n-1} + \Delta x f(x_{n-1}, y_{n-1}) = 0.5x_{n-1} + 1.5y_{n-1}$
0	0	1
1	0.5	$0.5(0) + 1.5(1.0) = 1.5$
2	1.0	$0.5(0.5) + 1.5(1.5) = 2.5$

Table 3.1: Application of Euler's Method for $y' = x + y$, $y(0) = 1$ and $\Delta x = 0.5$.

Is this a good result? Well, we could make the spatial increments smaller. Let's repeat the procedure for $\Delta x = 0.2$, or $N = 5$. The results are in Table 3.2.

Now we see that the approximation is $y_1 = 2.97664$. So, it looks like the value is near 3, but we cannot say much more. Decreasing Δx more shows that we are beginning to converge to a solution. We see this in Table 3.3.

Of course, these values were not done by hand. The last computation would have taken 1000 lines in the table, or at least 40 pages! One could

Table 3.2: Application of Euler's Method for $y' = x + y$, $y(0) = 1$ and $\Delta x = 0.2$.

n	x_n	$y_n = 0.2x_{n-1} + 1.2y_{n-1}$
0	0	1
1	0.2	$0.2(0) + 1.2(1.0) = 1.2$
2	0.4	$0.2(0.2) + 1.2(1.2) = 1.48$
3	0.6	$0.2(0.4) + 1.2(1.48) = 1.856$
4	0.8	$0.2(0.6) + 1.2(1.856) = 2.3472$
5	1.0	$0.2(0.8) + 1.2(2.3472) = 2.97664$

Table 3.3: Results of Euler's Method for $y' = x + y$, $y(0) = 1$ and varying Δx

Δx	$y_N \approx y(1)$
0.5	2.5
0.2	2.97664
0.1	3.187484920
0.01	3.409627659
0.001	3.433847864
0.0001	3.436291854

use a computer to do this. A simple code in Maple would look like the following:

```
> restart:
> f:=(x,y)->y+x;
> a:=0:  b:=1:  N:=100:  h:=(b-a)/N;
> x[0]:=0: y[0]:=1:
  for i from 1 to N do
  y[i]:=y[i-1]+h*f(x[i-1],y[i-1]):
  x[i]:=x[0]+h*(i):
  od:
  evalf(y[N]);
```

In this case we could simply use the exact solution. The exact solution is easily found as

$$y(x) = 2e^x - x - 1.$$

(The reader can verify this.) So, the value we are seeking is

$$y(1) = 2e - 2 = 3.4365636\ldots.$$

Thus, even the last numerical solution was off by about 0.00027.

Adding a few extra lines for plotting, we can visually see how well the approximations compare to the exact solution. The Maple code for doing such a plot is given below.

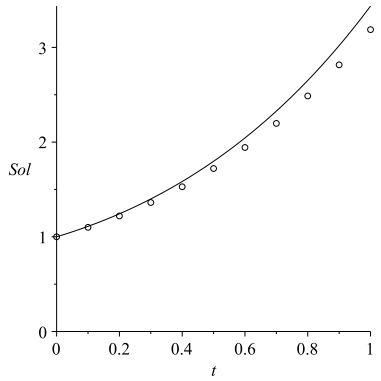


Figure 3.2: A comparison of the results Euler's Method to the exact solution for $y' = x + y$, $y(0) = 1$ and $N = 10$.

```
> with(plots):
> Data:=[seq([x[i],y[i]],i=0..N)]:
> P1:=pointplot(Data,symbol=DIAMOND):
> Sol:=t->-t-1+2*exp(t);
> P2:=plot(Sol(t),t=a..b,Sol=0..Sol(b)):
> display({P1,P2});
```

We show in Figures 3.2-3.3 the results for $N = 10$ and $N = 100$. In Figure 3.2 we can see how quickly the numerical solution diverges from the exact solution. In Figure 3.3 we can see that visually the solutions agree, but we note that from Table 3.3 that for $\Delta x = 0.01$, the solution is still off in the second decimal place with a relative error of about 0.8%.

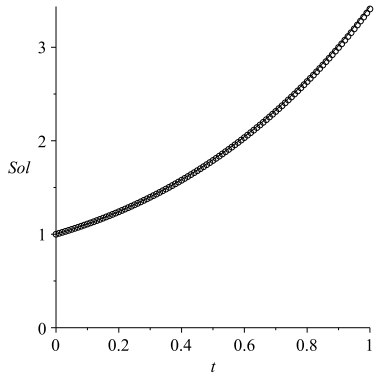


Figure 3.3: A comparison of the results Euler's Method to the exact solution for $y' = x + y$, $y(0) = 1$ and $N = 100$.

Why would we use a numerical method when we have the exact solution? Exact solutions can serve as test cases for our methods. We can make sure our code works before applying them to problems whose solution is not known.

There are many other methods for solving first order equations. One commonly used method is the fourth order Runge-Kutta method. This method has smaller errors at each step as compared to Euler's Method. It is well suited for programming and comes built-in in many packages like Maple and MATLAB. Typically, it is set up to handle systems of first order equations.

In fact, it is well known that nth order equations can be written as a system of n first order equations. Consider the simple second order equation

$$y'' = f(x, y).$$

This is a larger class of equations than the second order constant coefficient equation. We can turn this into a system of two first order differential equations by letting $u = y$ and $v = y' = u'$. Then, $v' = y'' = f(x, u)$. So, we have the first order system

$$\begin{aligned} u' &= v, \\ v' &= f(x, u). \end{aligned} \tag{3.7}$$

We will not go further into higher order methods until later in the chapter. We will discuss in depth higher order Taylor methods in Section 3.3 and Runge-Kutta Methods in Section 3.4. This will be followed by applications of numerical solutions of differential equations leading to interesting behaviors in Section 3.5. However, we will first discuss the numerical solution using built-in routines.

3.2 Implementation of Numerical Packages

3.2.1 First Order ODEs in MATLAB

One can use MATLAB to obtain solutions and plots of solutions of differential equations. This can be done either symbolically, using **dsolve**, or numerically, using numerical solvers like **ode45**. In this section we will provide examples of using these to solve first order differential equations. We will end with the code for drawing direction fields, which are useful for looking at the general behavior of solutions of first order equations without explicitly finding the solutions.

Symbolic Solutions

THE FUNCTION **dsolve** OBTAINS THE SYMBOLIC SOLUTION and **ezplot** is used to quickly plot the symbolic solution. As an example, we apply **dsolve** to solve the

$$x' = 2\sin t - 4x, \quad x(0) = 0 \tag{3.8}$$

At the MATLAB prompt, type the following:

```
sol = dsolve('Dx=2*sin(t)-4*x','x(0)=0','t');
ezplot(sol,[0 10])
xlabel('t'),ylabel('x'), grid
```

The solution is given as

```
sol =

 (2*exp(-4*t))/17 - (2*17^(1/2)*cos(t + atan(4)))/17
```

Figure 3.4 shows the solution plot.

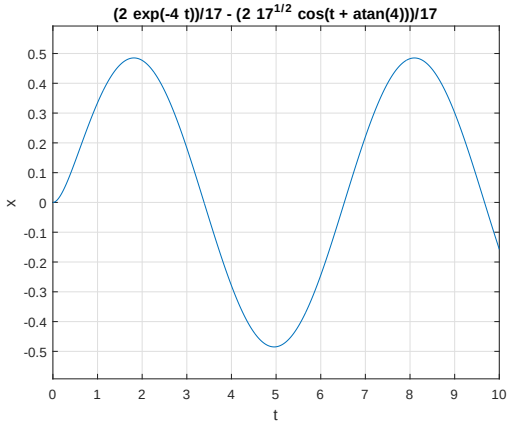


Figure 3.4: The solution of Equation (3.8) with $x(0) = 0$ found using MATLAB's **dsolve** command.

ODE45 and Other Solvers.

THERE ARE SEVERAL ODE SOLVERS IN MATLAB, implementing Runge-Kutta and other numerical schemes. Examples of its use are in the differential equations textbook. For example, one can implement **ode45** to solve the initial value problem

$$\frac{dy}{dt} = -\frac{yt}{\sqrt{2-y^2}}, \quad y(0) = 1,$$

using the following code:

```
[t y]=ode45('func',[0 5],1);
plot(t,y)
xlabel('t'),ylabel('y')
title('y(t) vs t')
```

One can define the function **func** in a file **func.m** such as

```
function f=func(t,y)
f=-t*y/sqrt(2-y.^2);
```

Running the above code produces Figure 3.5.

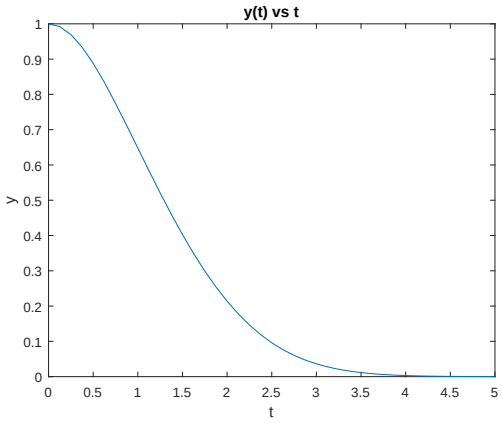


Figure 3.5: A plot of the solution of $\frac{dy}{dt} = -\frac{yt}{\sqrt{2-y^2}}$, $y(0) = 1$, found using MATLAB's **ode45** command.

One can also use **ode45** to solve higher order differential equations. Second order differential equations are discussed in Section 3.2.2. See MATLAB help for other examples and other ODE solvers.

Direction Fields

One can produce direction fields in MATLAB. For the differential equation

$$\frac{dy}{dx} = f(x, y),$$

we note that $f(x, y)$ is the slope of the solution curve passing through the point in the xy=plane. Thus, the direction field is a collection of tangent vectors at points (x, y) indication the slope, $f(x, y)$, at that point.

A sample code for drawing direction fields in MATLAB is given by

```
[x,y]=meshgrid(0:.1:2,0:.1:1.5);
dy=1-y;
dx=ones(size(dy));
quiver(x,y,dx,dy)
axis([0,2,0,1.5])
xlabel('x')
ylabel('y')
```

The mesh command sets up the xy-grid. In this case x is in $[0, 2]$ and y is in $[0, 1.5]$. In each case the grid spacing is 0.1.

We let dy = 1-y and dx =1. Thus,

$$\frac{dy}{dx} = \frac{1-y}{1} = 1 - y.$$

The **quiver** command produces a vector (dx,dy) at (x,y). The slope of each vector isdy/dx. The other commands label the axes and provides a window with xmin=0, xmax=2, ymin=0, ymax=1.5. The result of using the above code is shown in Figure 3.6.

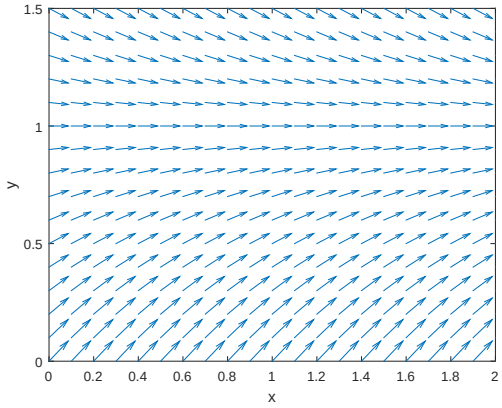


Figure 3.6: A direction field produced using MATLAB's **quiver** function for $y' = 1 - y$.

One can add solution, or integral, curves to the direction field for different initial conditions to further aid in seeing the connection between direction fields and integral curves. One needs to add to the direction field code the following lines:

```
hold on
[t,y] = ode45(@(t,y) 1-y, [0 2], .5);
plot(t,y,'k','LineWidth',2)
[t,y] = ode45(@(t,y) 1-y, [0 2], 1.5);
plot(t,y,'k','LineWidth',2)
hold off
```

Here the function $f(t,y) = 1 - y$ is entered this time using MATLAB's anonymous function, **@(t,y) 1-y**. Before plotting, the **hold** command is invoked to allow plotting several plots on the same figure. The result is shown in Figure 3.7

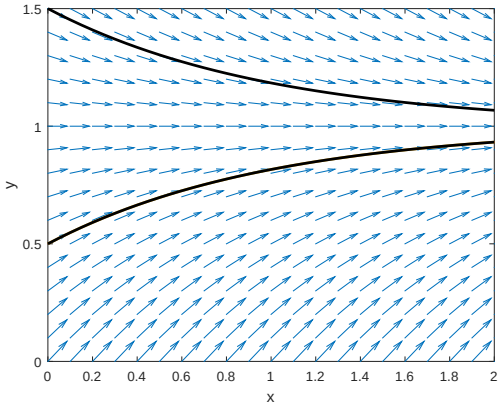


Figure 3.7: A direction field produced using MATLAB's **quiver** function for $y' = 1 - y$ with solution curves added.

3.2.2 *Second Order ODEs in MATLAB*

We can also use **ode45** to solve second and higher order differential equations. The key is to rewrite the single differential equation as a system of first order equations. Consider the simple harmonic oscillator equation, $\ddot{x} + \omega^2 x = 0$. Defining $y_1 = x$ and $y_2 = \dot{x}$, and noting that

$$\ddot{x} + \omega^2 x = \dot{y}_2 + \omega^2 y_1,$$

we have

$$\begin{aligned} \dot{y}_1 &= y_2, \\ \dot{y}_2 &= -\omega^2 y_1. \end{aligned}$$

Furthermore, we can view this system in the form $\dot{\mathbf{y}} = \mathbf{y}$. In particular, we have

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ -\omega^2 y_2 \end{bmatrix}$$

Now, we can use **ode45**. We modify the code slightly from Chapter 1.

```
[t y]=ode45('func',[0 5],[1 0]);
```

Here [0 5] gives the time interval and [1 0] gives the initial conditions

$$y_1(0) = x(0) = 1, \quad y_2(0) = \dot{x}(0) = 1.$$

The function **func** is a set of commands saved to the file **func.m** for computing the righthand side of the system of differential equations. For the simple harmonic oscillator, we enter the function as

```
function dy=func(t,y)
omega=1.0;
dy(1,1) = y(2);
dy(2,1) = -omega^2*y(1);
```

There are a variety of ways to introduce the parameter ω. Here we simply defined it within the function. Furthermore, the output **dy** should be a column vector.

After running the solver, we then need to display the solution. The output should be a column vector with the position as the first element and the velocity as the second element. So, in order to plot the solution as a function of time, we can plot the first column of the solution, y(:,1), vs t:

```
plot(t,y(:,1))
xlabel('t'),ylabel('y')
title('y(t) vs t')
```

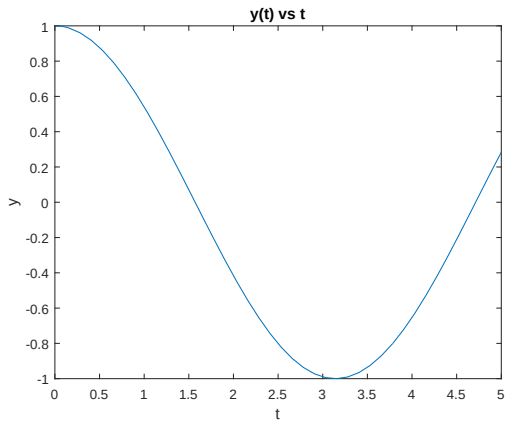


Figure 3.8: Solution plot for the simple harmonic oscillator.

The resulting solution is shown in Figure 3.8.

We can also do a phase plot of velocity vs position. In this case, one can plot the second column, y(:,2), vs the first column, y(:,1):

```
plot(y(:,1),y(:,2))
xlabel('y'),ylabel('v')
title('v(t) vs y(t)')
```

The resulting solution is shown in Figure 3.9.

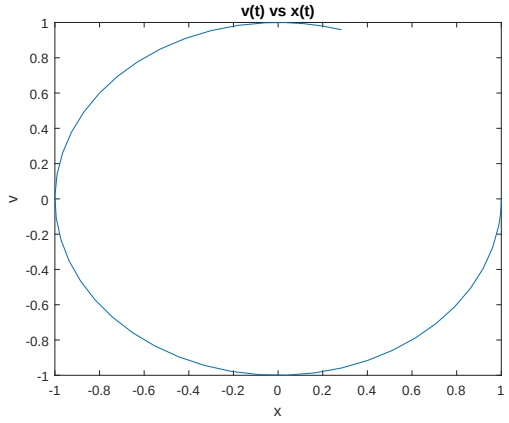


Figure 3.9: Phase plot for the simple harmonic oscillator.

Finally, we can plot a direction field using a quiver plot and add solution curves using **ode45**. The direction field is given for $\omega = 1$ by dx=y and dy=-x.

```
clear
[x,y]=meshgrid(-2:.2:2,-2:.2:2);
dx=y;
dy=-x;
quiver(x,y,dx,dy)
axis([-2,2,-2,2])
xlabel('x')
ylabel('y')
```

```
hold on
[t y]=ode45('func',[0 6.28],[1 0]);
plot(y(:,1),y(:,2))
hold off
```

The resulting plot is given in Figure 3.10.

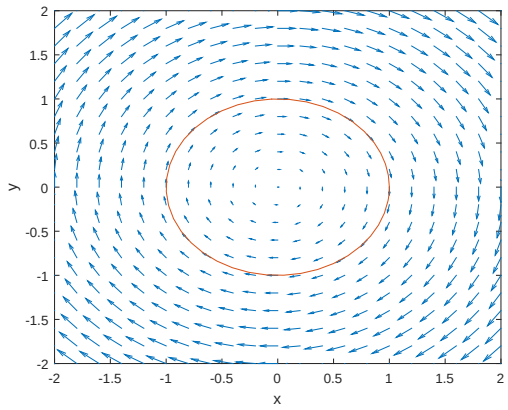


Figure 3.10: Phase plot for the simple harmonic oscillator.

3.2.3 *GNU Octave*

MUCH OF MATLAB'S FUNCTIONALITY CAN BE USED IN GNU OCTAVE. However, a simple solution of a differential equation is not the same. Instead GNU Octave uses the Fortan **lsode** routine. The main code below gives what is needed to solve the system

$$\frac{d}{dt}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ -cy \end{bmatrix}.$$

```
global c
c=1;

y=lsode("oscf",[1,0],(tau=linspace(0,5,100))');

figure(1);
plot(tau,y(:,1));
xlabel('t')
ylabel('x(t)')

figure(2);
plot(y(:,1),y(:,2));
xlabel('x(t)')
ylabel('y(t)')
```

The function called by the **lsode** routine, **oscf**, looks similar to MATLAB code. However, one needs to take care in the syntax and ordering of the input variables. The output from this code is shown in Figure 3.11.

```
function ydot=oscf(y,tau);
global c

ydot(1)=y(2);
ydot(2)=-c*y(1);
```

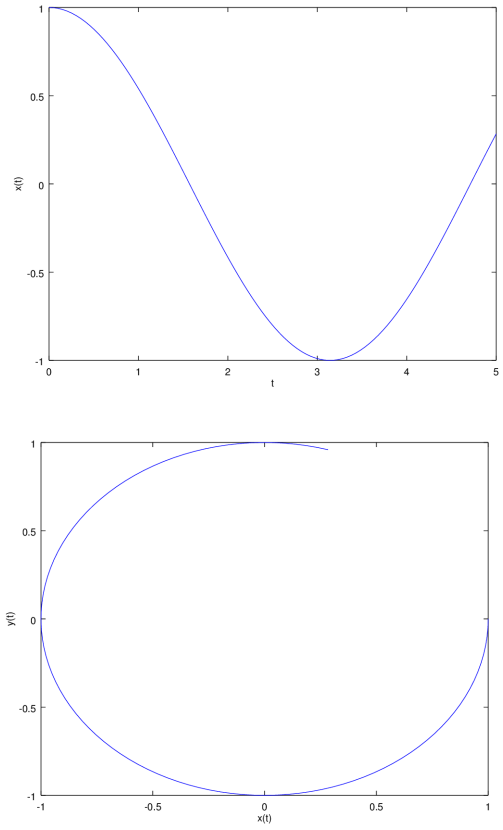


Figure 3.11: Numerical solution of the simple harmonic oscillator using GNU Octave's **lsode** routine. In these plots are the position and velocity vs times plots and a phase plot.

3.2.4 *Python Implementation*

ONE CAN ALSO SOLVE ORDINARY DIFFERENTIAL EQUATIONS using Python. One can use the **odeint** routine from **scipy.inegrate**. This uses a variable step routine based on the Fortan **lsoda** routine. The below code solves a simple harmonic oscillator equation and produces the plot in Figure 3.12.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

```
# Solve dv/dt = [y, - cx] for v = [x,y]
def odefn(v,t, c):
  x,  y = v
  dvdt = [y, -c*x ]
  return dvdt

v0 = [1.0, 0.0]
t = np.arange(0.0, 10.0, 0.1)
c = 5;

sol = odeint(odefn, v0, t,args=(c,))

plt.plot(t, sol[:,0],'b')
plt.xlabel('Time (sec)')
plt.ylabel('Position')
plt.title('Position vs Time')
plt.show()
```

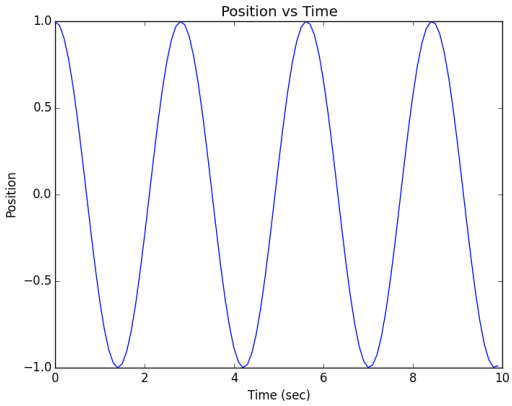


Figure 3.12: Numerical solution of the simple harmonic oscillator using Python's odeint.

If one wants to use something similar to the Runga-Kutta scheme, then the **ode** routine can be used with a specification of ode solver. The below code solves a simple harmonic oscillator equation and produces the plot in Figure 3.13.

```
from scipy import *
from scipy.integrate import ode
from pylab import *


# Solve dv/dt = [y, - cx] for v = [x,y]
def odefn(t,v, c):
  x,  y = v
```

```
  dvdt = [y, -c*x ]
  return dvdt

v0 = [1.0, 0.0]

t0=0;
tf=10;
dt=0.1;
c = 5;

Y=[];
T=[];

r = ode(odefn).set_integrator('dopri5')
r.set_f_params(c).set_initial_value(v0,t0)

while r.successful() and r.t+dt < tf:
    r.integrate(r.t+dt)
    Y.append(r.y)
    T.append(r.t)

Y = array(Y)

subplot(2,1,1)
plot(T,Y)
plt.xlabel('Time (sec)')
plt.ylabel('Position')

subplot(2,1,2)
plot(Y[:,0],Y[:,1])
xlabel('Position')
ylabel('Velocity')
show()
```

3.2.5 *Maple Implementation*

MAPLE ALSO HAS BUILT-IN ROUTINES FOR SOLVING DIFFERENTIAL EQUATIONS. First, we consider the symbolic solutions of a differential equation. An example of a symbolic solution of a first order differential equation, $y' = 1 - y$ with $y(0) - 1.5$, is given by

```
> restart: with(plots):
> EQ:=diff(y(x),x)=1-y(x):
> dsolve({EQ,y(0)=1.5});
```

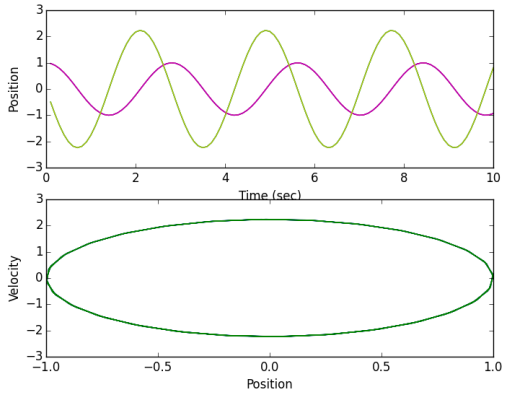


Figure 3.13: Numerical solution of the simple harmonic oscillator using Python's ode routine. In these plots are the position and velocity vs times plots and a phase plot.

The resulting solution from Maple is

$$y(x) = 1 + \frac{1}{2}e^{-x}.$$

One can also plot direction fields for first order equations. An example is given below with the plot shown in Figure 3.14.

```
> restart: with(DEtools):
> ode := diff(y(t),t) = 1-y(t):
> DEplot(ode,y(t),t=0..2,y=0..1.5,color=black);
```

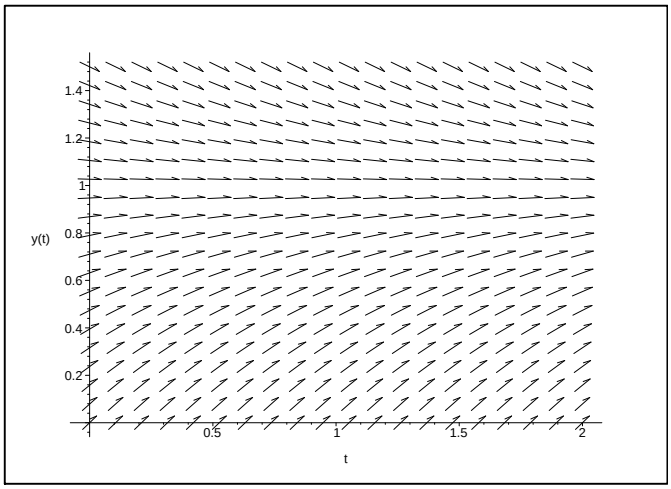


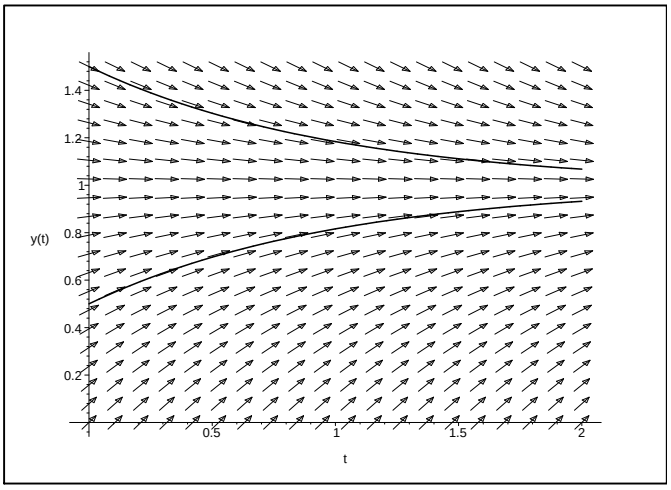
Figure 3.14: Maple direction field plot for first order differential equation.

In order to add solution curves, we specify initial conditions using the following lines as seen in Figure 3.15.

```
> ics:=[y(0)=0.5,y(0)=1.5]:
> DEplot(ode,yt),t=0..2,y=0..1.5,ics,arrows=medium,linecolor=black,color=black);
```

These routines can be used to obtain solutions of a system of differential equations.

Figure 3.15: Maple direction field plot for first order differential equation with solution curves added.

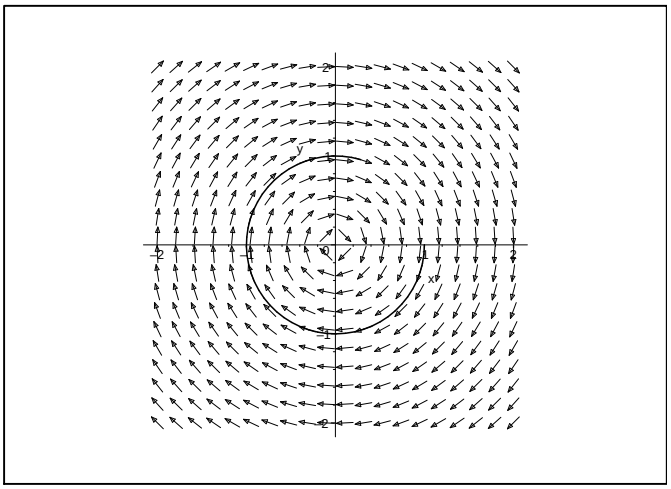



```
> EQ:=diff(x(t),t)=y(t),diff(y(t),t)=-x(t):
> ICs:=x(0)=1,y(0)=0;
> dsolve([EQ, ICs]);
> plot(rhs(%[1]),t=0..5);
```

A phaseportrait with a direction field, as seen in Figure 3.16, is found using the lines

```
> with(DEtools):
> DEplot( [EQ], [x(t),y(t)], t=0..5, x=-2..2, y=-2..2, [[x(0)=1,y(0)=0]],
  arrows=medium,linecolor=black,color=black,scaling=constrained);
```

Figure 3.16: Maple system plot.



3.3 *Higher Order Taylor Methods**

EULER'S METHOD FOR SOLVING DIFFERENTIAL EQUATIONS is easy to understand but is not efficient in the sense that it is what is called a first order

method. The error at each step, the local truncation error, is of order Δx, for x the independent variable. The accumulation of the local truncation errors results in what is called the global error. In order to generalize Euler's Method, we need to rederive it. Also, since these methods are typically used for initial value problems, we will cast the problem to be solved as

$$\frac{dy}{dt} = f(t,y), \quad y(a) = y_0, \quad t \in [a,b]. \tag{3.9}$$

The first step towards obtaining a numerical approximation to the solution of this problem is to divide the t-interval, $[a,b]$, into N subintervals,

$$t_i = a + ih, \quad i = 0,1,\ldots,N, \quad t_0 = a, \quad t_N = b,$$

where

$$h = \frac{b-a}{N}.$$

We then seek the numerical solutions

$$\tilde{y}_i \approx y(t_i), \quad i = 1,2,\ldots,N,$$

with $\tilde{y}_0 = y(t_0) = y_0$. Figure 3.17 graphically shows how these quantities are related.

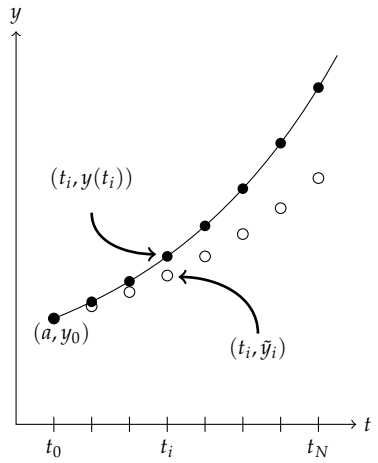


Figure 3.17: The interval $[a,b]$ is divided into N equally spaced subintervals. The exact solution $y(t_i)$ is shown with the numerical solution, $\tilde{y}_i$ with $t_i = a + ih$, $i = 0,1,\ldots,N$.

Euler's Method can be derived using the Taylor series expansion of of the solution $y(t_i + h)$ about $t = t_i$ for $i = 1,2,\ldots,N$. This is given by

$$\begin{aligned} y(t_{i+1}) &= y(t_i + h) \\ &= y(t_i) + y'(t_i)h + \frac{h^2}{2}y''(\xi_i), \quad \xi_i \in (t_i, t_{i+1}). \end{aligned} \tag{3.10}$$

Here the term $\frac{h^2}{2}y''(\xi_i)$ captures all of the higher order terms and represents the error made using a linear approximation to $y(t_i + h)$.

Dropping the remainder term, noting that $y'(t) = f(t,y)$, and defining the resulting numerical approximations by $\tilde{y}_i \approx y(t_i)$, we have

$$\begin{aligned} \tilde{y}_{i+1} &= \tilde{y}_i + hf(t_i, \tilde{y}_i), \quad i = 0,1,\ldots,N-1, \\ \tilde{y}_0 &= y(a) = y_0. \end{aligned} \tag{3.11}$$

This is Euler's Method.

Euler's Method is not used in practice since the error is of order h. However, it is simple enough for understanding the idea of solving differential equations numerically. Also, it is easy to study the numerical error, which we will show next.

The error that results for a single step of the method is called the local truncation error, which is defined by

$$\tau_{i+1}(h) = \frac{y(t_{i+1}) - \tilde{y}_i}{h} - f(t_i, y_i).$$

A simple computation gives

$$\tau_{i+1}(h) = \frac{h}{2}y''(\xi_i), \quad \xi_i \in (t_i, t_{i+1}).$$

Since the local truncation error is of order h, this scheme is said to be of order one. More generally, for a numerical scheme of the form

$$\begin{aligned} \tilde{y}_{i+1} &= \tilde{y}_i + hF(t_i, \tilde{y}_i), \quad i = 0, 1, \ldots, N-1, \\ \tilde{y}_0 &= y(a) = y_0, \end{aligned} \tag{3.12}$$

The local truncation error.

the local truncation error is defined by

$$\tau_{i+1}(h) = \frac{y(t_{i+1}) - \tilde{y}_i}{h} - F(t_i, y_i).$$

The accumulation of these errors leads to the global error. In fact, one can show that if f is continuous, satisfies the Lipschitz condition,

$$|f(t, y_2) - f(t, y_1)| \leq L|y_2 - y_1|$$

for a particular domain $D \subset R^2$, and

$$|y''(t)| \leq M, \quad t \in [a, b],$$

then

$$|y(t_i) - \tilde{y}_|| \leq \frac{hM}{2L}\left(e^{L(t_i - a)} - 1\right), \quad i = 0, 1, \ldots, N.$$

Furthermore, if one introduces round-off errors, bounded by δ, in both the initial condition and at each step, the global error is modified as

$$|y(t_i) - \tilde{y}_|| \leq \frac{1}{L}\left(\frac{hM}{2} + \frac{\delta}{h}\right)\left(e^{L(t_i - a)} - 1\right) + |\delta_0| e^{L(t_i - a)}, \quad i = 0, 1, \ldots, N.$$

Then for small enough steps h, there is a point when the round-off error will dominate the error. [See Burden and Faires, *Numerical Analysis* for the details.]

Can we improve upon Euler's Method? The natural next step towards finding a better scheme would be to keep more terms in the Taylor series expansion. This leads to Taylor series methods of order n.

Taylor series methods of order n take the form

$$\begin{aligned} \tilde{y}_{i+1} &= \tilde{y}_i + hT^{(n)}(t_i, \tilde{y}_i), \quad i = 0, 1, \ldots, N-1, \\ \tilde{y}_0 &= y_0, \end{aligned} \tag{3.13}$$

where we have defined

$$T^{(n)}(t, y) = y'(t) + \frac{h}{2}y''(t) + \cdots + \frac{h^{(n-1)}}{n!}y^{(n)}(t).$$

However, since $y'(t) = f(t, y)$, we can write

$$T^{(n)}(t, y) = f(t, y) + \frac{h}{2}f'(t, y) + \cdots + \frac{h^{(n-1)}}{n!}f^{(n-1)}(t, y).$$

We note that for $n = 1$, we retrieve Euler's Method as a special case. We demonstrate a third order Taylor's Method in the next example.

Example 3.2. Apply the third order Taylor's Method to

$$\frac{dy}{dt} = t + y, \quad y(0) = 1$$

and obtain an approximation for $y(1)$ for $h = 0.1$.

The third order Taylor's Method takes the form

$$\begin{aligned} \tilde{y}_{i+1} &= \tilde{y}_i + hT^{(3)}(t_i, \tilde{y}_i), \quad i = 0, 1, \ldots, N-1, \\ \tilde{y}_0 &= y_0, \end{aligned} \tag{3.14}$$

where

$$T^{(3)}(t, y) = f(t, y) + \frac{h}{2} f'(t, y) + \frac{h^2}{3!} f''(t, y)$$

and $f(t, y) = t + y(t)$.

In order to set up the scheme, we need the first and second derivative of $f(t, y)$:

$$\begin{aligned} f'(t, y) &= \frac{d}{dt}(t + y) \\ &= 1 + y' \\ &= 1 + t + y \end{aligned} \tag{3.15}$$

$$\begin{aligned} f''(t, y) &= \frac{d}{dt}(1 + t + y) \\ &= 1 + y' \\ &= 1 + t + y \end{aligned} \tag{3.16}$$

Inserting these expressions into the scheme, we have

$$\begin{aligned} \tilde{y}_{i+1} &= \tilde{y}_i + h\left[(t_i + y_i) + \frac{h}{2}(1 + t_i + y_i) + \frac{h^2}{3!}(1 + t_i + y_i)\right], \\ &= \tilde{y}_i + h(t_i + y_i) + h^2(\frac{1}{2} + \frac{h}{6})(1 + t_i + y_i), \\ \tilde{y}_0 &= y_0, \end{aligned} \tag{3.17}$$

for $i = 0, 1, \ldots, N-1$.

In Figure 3.2 we show the results comparing Euler's Method, the 3rd Order Taylor's Method, and the exact solution for $N = 10$. In Table 3.4 we provide are the numerical values. The relative error in Euler's method is about 7% and that of the 3rd Order Taylor's Method is about 0.006%. Thus, the 3rd Order Taylor's Method is significantly better than Euler's Method.

In the last section we provided some Maple code for performing Euler's method. A similar code in MATLAB looks like the following:

```
a=0;
b=1;
N=10;
h=(b-a)/N;
```

Table 3.4: Numerical values for Euler's Method, 3rd Order Taylor's Method, and exact solution for solving Example 3.2 with $N = 10$..

Euler	Taylor	Exact
1.0000	1.0000	1.0000
1.1000	1.1103	1.1103
1.2200	1.2428	1.2428
1.3620	1.3997	1.3997
1.5282	1.5836	1.5836
1.7210	1.7974	1.7974
1.9431	2.0442	2.0442
2.1974	2.3274	2.3275
2.4872	2.6509	2.6511
2.8159	3.0190	3.0192
3.1875	3.4364	3.4366

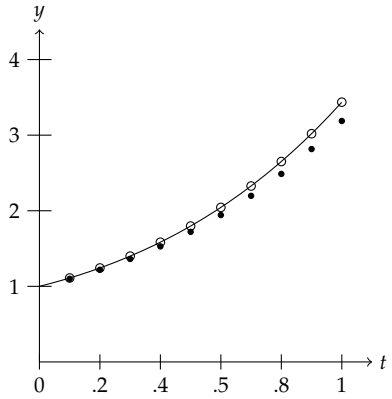


Figure 3.18: Numerical results for Euler's Method (filled circle) and 3rd Order Taylor's Method (open circle) for solving Example 3.2 as compared to exact solution (solid line).

```
% Slope function
  f = inline('t+y','t','y');
  sol = inline('2*exp(t)-t-1','t');

% Initial Condition
  t(1)=0;
  y(1)=1;

% Euler's Method
  for i=2:N+1
      y(i)=y(i-1)+h*f(t(i-1),y(i-1));
      t(i)=t(i-1)+h;
  end
```

A simple modification can be made for the 3rd Order Taylor's Method by replacing the Euler's method part of the preceding code by

```
% Taylor's Method, Order 3
  y(1)=1;
  h3 = h^2*(1/2+h/6);
  for i=2:N+1
      y(i)=y(i-1)+h*f(t(i-1),y(i-1))+h3*(1+t(i-1)+y(i-1));
      t(i)=t(i-1)+h;
  end
```

While the accuracy in the last example seemed sufficient, we have to remember that we only stopped at one unit of time. How can we be confident that the scheme would work as well if we carried out the computation for much longer times. For example, if the time unit were only a second, then one would need 86,400 times longer to predict a day forward. Of course, the scale matters. But, often we need to carry out numerical schemes for long times and we hope that the scheme not only converges to a solution, but that it coverges to the solution to the given problem. Also, the previous

example was relatively easy to program because we could provide a relatively simple form for $T^{(3)}(t,y)$ with a quick computation of the derivatives of $f(t,y)$. This is not always the case and higher order Taylor methods in this form are not typically used. Instead, one can approximate $T^{(n)}(t,y)$ by evaluating the known function $f(t,y)$ at selected values of t and y, leading to Runge-Kutta methods.

3.4 *Runge-Kutta Methods**

As we had seen in the last section, we can use higher order Taylor methods to derive numerical schemes for solving

$$\frac{dy}{dt} = f(t,y), \quad y(a) = y_0, \quad t \in [a,b], \tag{3.18}$$

using a scheme of the form

$$\begin{aligned} \tilde{y}_{i+1} &= \tilde{y}_i + hT^{(n)}(t_i, \tilde{y}_i), \quad i = 0, 1, \ldots, N-1, \\ \tilde{y}_0 &= y_0, \end{aligned} \tag{3.19}$$

where we have defined

$$T^{(n)}(t,y) = y'(t) + \frac{h}{2}y''(t) + \cdots + \frac{h^{(n-1)}}{n!}y^{(n)}(t).$$

In this section we will find approximations of $T^{(n)}(t,y)$ which avoid the need for computing the derivatives.

For example, we could approximate

$$T^{(2)}(t,y) = f(t,y) + \frac{h}{2}\frac{df}{dt}(t,y)$$

by

$$T^{(2)}(t,y) \approx af(t+\alpha, y+\beta)$$

for selected values of a, α, and β. This requires use of a generalization of Taylor's series to functions of two variables. In particular, for small α and β we have

$$\begin{aligned} af(t+\alpha, y+\beta) &= a\left[f(t,y) + \frac{\partial f}{\partial t}(t,y)\alpha + \frac{\partial f}{\partial y}(t,y)\beta \right. \\ &\quad \left. + \frac{1}{2}\left(\frac{\partial^2 f}{\partial t^2}(t,y)\alpha^2 + 2\frac{\partial^2 f}{\partial t \partial y}(t,y)\alpha\beta + \frac{\partial^2 f}{\partial y^2}(t,y)\beta^2\right)\right] \\ &\quad + \quad \text{higher order terms.} \end{aligned} \tag{3.20}$$

Furthermore, we need $\frac{df}{dt}(t,y)$. Since $y = y(t)$, this can be found using a generalization of the Chain Rule from Calculus III:

$$\frac{df}{dt}(t,y) = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{dy}{dt}.$$

Thus,

$$T^{(2)}(t,y) = f(t,y) + \frac{h}{2}\left[\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{dy}{dt}\right].$$

Comparing this expression to the linear (Taylor series) approximation of $af(t+\alpha, y+\beta)$, we have

$$\begin{aligned} T^{(2)} &\approx af(t+\alpha, y+\beta) \\ f + \frac{h}{2}\frac{\partial f}{\partial t} + \frac{h}{2} f \frac{\partial f}{\partial y} &\approx af + a\alpha\frac{\partial f}{\partial t} + \beta\frac{\partial f}{\partial y}. \end{aligned} \tag{3.21}$$

We see that we can choose

$$a = 1, \quad \alpha = \frac{h}{2}, \quad \beta = \frac{h}{2} f.$$

This leads to the numerical scheme

$$\begin{aligned} \tilde{y}_{i+1} &= \tilde{y}_i + hf\left(t_i + \frac{h}{2}, \tilde{y}_i + \frac{h}{2} f(t_i, \tilde{y}_i)\right), \quad i = 0, 1, \ldots, N-1, \\ \tilde{y}_0 &= y_0, \end{aligned} \tag{3.22}$$

The Midpoint or Second Order Runge-Kutta Method.

This Runge-Kutta scheme is called the Midpoint Method, or Second Order Runge-Kutta Method, and it has order 2 if all second order derivatives of $f(t, y)$ are bounded.

Often, in implementing Runge-Kutta schemes, one computes the arguments separately as shown in the following MATLAB code snippet. (This code snippet could replace the Euler's Method section in the code in the last section.)

```
% Midpoint Method
  y(1)=1;
  for i=2:N+1
      k1=h/2*f(t(i-1),y(i-1));
      k2=h*f(t(i-1)+h/2,y(i-1)+k1);
      y(i)=y(i-1)+k2;
      t(i)=t(i-1)+h;
  end
```

Example 3.3. Compare the Midpoint Method with the 2nd Order Taylor's Method for the problem

$$y' = t^2 + y, \quad y(0) = 1, \quad t \in [0, 1]. \tag{3.23}$$

The solution to this problem is $y(t) = 3e^t - 2 - 2t - t^2$. In order to implement the 2nd Order Taylor's Method, we need

$$\begin{aligned} T^{(2)} &= f(t, y) + \frac{h}{2} f'(t, y) \\ &= t^2 + y + \frac{h}{2}(2t + t^2 + y). \end{aligned} \tag{3.24}$$

The results of the implementation are shown in Table 3.3.

There are other way to approximate higher order Taylor polynomials. For example, we can approximate $T^{(3)}(t, y)$ using four parameters by

$$T^{(3)}(t, y) \approx af(t, y) + bf(t + \alpha, y + \beta f(t, y).$$

Exact	Taylor	Error	Midpoint	Error
1.0000	1.0000	0.0000	1.0000	0.0000
1.1055	1.1050	0.0005	1.1053	0.0003
1.2242	1.2231	0.0011	1.2236	0.0006
1.3596	1.3577	0.0019	1.3585	0.0010
1.5155	1.5127	0.0028	1.5139	0.0016
1.6962	1.6923	0.0038	1.6939	0.0023
1.9064	1.9013	0.0051	1.9032	0.0031
2.1513	2.1447	0.0065	2.1471	0.0041
2.4366	2.4284	0.0083	2.4313	0.0053
2.7688	2.7585	0.0103	2.7620	0.0068
3.1548	3.1422	0.0126	3.1463	0.0085

Table 3.5: Numerical values for 2nd Order Taylor's Method, Midpoint Method, exact solution, and errors for solving Example 3.3 with $N = 10$..

Expanding this approximation and using

$$T^{(3)}(t,y) \approx f(t,y) + \frac{h}{2}\frac{df}{dt}(t,y) + \frac{h^2}{6}\frac{df}{dt}(t,y),$$

we find that we cannot get rid of $O(h^2)$ terms. Thus, the best we can do is derive second order schemes. In fact, following a procedure similar to the derivation of the Midpoint Method, we find that

$$a + b = 1, \quad , \alpha b = \frac{h}{2}, \beta = \alpha.$$

There are three equations and four unknowns. Therefore there are many second order methods. Two classic methods are given by the modified Euler method ($a = b = \frac{1}{2}$, $\alpha = \beta = h$) and Huen's method ($a = \frac{1}{4}$, $b = \frac{3}{4}$, $\alpha = \beta = \frac{2}{3}h$).

The Fourth Order Runge-Kutta.

The Fourth Order Runge-Kutta Method, which is most often used, is given by the scheme

$$\begin{aligned}
\tilde{y}_0 &= y_0, \\
k_1 &= hf(t_i, \tilde{y}_i), \\
k_2 &= hf(t_i + \frac{h}{2}, \tilde{y}_i + \frac{1}{2}k_1), \\
k_3 &= hf(t_i + \frac{h}{2}, \tilde{y}_i + \frac{1}{2}k_2), \\
k_4 &= hf(t_i + h, \tilde{y}_i + k_3), \\
\tilde{y}_{i+1} &= \tilde{y}_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad i = 0, 1, \ldots, N-1. \qquad (3.25)
\end{aligned}$$

Again, we can test this on Example 3.3 with $N = 10$. The MATLAB implementation is given by

```
% Runge-Kutta 4th Order to solve dy/dt = f(t,y), y(a)=y0, on [a,b]
  clear

  a=0;
  b=1;
```

```
N=10;
h=(b-a)/N;

% Slope function
f = inline('t^2+y','t','y');
sol = inline('-2-2*t-t^2+3*exp(t)','t');

% Initial Condition
t(1)=0;
y(1)=1;

% RK4 Method
y1(1)=1;
for i=2:N+1
    k1=h*f(t(i-1),y1(i-1));
    k2=h*f(t(i-1)+h/2,y1(i-1)+k1/2);
    k3=h*f(t(i-1)+h/2,y1(i-1)+k2/2);
    k4=h*f(t(i-1)+h,y1(i-1)+k3);
    y1(i)=y1(i-1)+(k1+2*k2+2*k3+k4)/6;
    t(i)=t(i-1)+h;
end
```

MATLAB has built-in ODE solvers, as do other software packages, like Maple and Mathematica. You should also note that there are currently open source packages, such as Python based NumPy and Matplotlib, or Octave, of which some packages are contained within the Sage Project.

MATLAB has built-in ODE solvers, such as **ode45** for a fourth order Runge-Kutta method. Its implementation is given by

```
[t,y]=ode45(f,[0 1],1);
```

In this case f is given by an inline function like in the above RK4 code. The time interval is enetered as $[0, 1]$ and the 1 is the initial condition, $y(0) = 1$.

However, **ode45** is not a straight forward RK4 implementation. It is a hybrid method in which a combination of 4th and 5th order methods are combined allowing for adaptive methods to handled subintervals of the integration region which need more care. In this case, it implements a fourth order Runge-Kutta-Fehlberg method. Running this code for the above example actually results in values for $N = 41$ and not $N = 10$. If we wanted to have the routine output numerical solutions at specific times, then one could use the following form

```
tspan=0:h:1;
[t,y]=ode45(f,tspan,1);
```

In Table 3.6 we show the solutions which results for Example 3.3 comparing the RK4 snippet above with **ode45**. As you can see RK4 is much better than the previous implementation of the second order RK (Midpoint) Method. However, the MATLAB routine is two orders of magnitude better that RK4.

Exact	Taylor	Error	Midpoint	Error
1.0000	1.0000	0.0000	1.0000	0.0000
1.1055	1.1055	4.5894e-08	1.1055	-2.5083e-10
1.2242	1.2242	1.2335e-07	1.2242	-6.0935e-10
1.3596	1.3596	2.3850e-07	1.3596	-1.0954e-09
1.5155	1.5155	3.9843e-07	1.5155	-1.7319e-09
1.6962	1.6962	6.1126e-07	1.6962	-2.5451e-09
1.9064	1.9064	8.8636e-07	1.9064	-3.5651e-09
2.1513	2.1513	1.2345e-06	2.1513	-4.8265e-09
2.4366	2.4366	1.6679e-06	2.4366	-6.3686e-09
2.7688	2.7688	2.2008e-06	2.7688	-8.2366e-09
3.1548	3.1548	2.8492e-06	3.1548	-1.0482e-08

Table 3.6: Numerical values for Fourth Order Runge-Kutta Method, rk45, exact solution, and errors for solving Example 3.3 with $N = 10$.

There are many ODE solvers in MATLAB. These are typically useful if RK4 is having difficulty solving particular problems. For the most part, one is fine using RK4, especially as a starting point. For example, there is **ode23**, which is similar to **ode45** but combining a second and third order scheme. Applying the results to Example 3.3 we obtain the results in Table 3.6. We compare these to the second order Runge-Kutta method. The code snippets are shown below.

```
% Second Order RK Method
  y1(1)=1;
  for i=2:N+1
      k1=h*f(t(i-1),y1(i-1));
      k2=h*f(t(i-1)+h/2,y1(i-1)+k1/2);
      y1(i)=y1(i-1)+k2;
      t(i)=t(i-1)+h;
  end

  tspan=0:h:1;
  [t,y]=ode23(f,tspan,1);
```

Exact	Taylor	Error	Midpoint	Error
1.0000	1.0000	0.0000	1.0000	0.0000
1.1055	1.1053	0.0003	1.1055	2.7409e-06
1.2242	1.2236	0.0006	1.2242	8.7114e-06
1.3596	1.3585	0.0010	1.3596	1.6792e-05
1.5155	1.5139	0.0016	1.5154	2.7361e-05
1.6962	1.6939	0.0023	1.6961	4.0853e-05
1.9064	1.9032	0.0031	1.9063	5.7764e-05
2.1513	2.1471	0.0041	2.1512	7.8665e-05
2.4366	2.4313	0.0053	2.4365	0.0001
2.7688	2.7620	0.0068	2.7687	0.0001
3.1548	3.1463	0.0085	3.1547	0.0002

Table 3.7: Numerical values for Second Order Runge-Kutta Method, rk23, exact solution, and errors for solving Example 3.3 with $N = 10$.

We have seen several numerical schemes for solving initial value problems. There are other methods, or combinations of methods, which aim to refine the numerical approximations efficiently as if the step size in the current methods were taken to be much smaller. Some methods extrapolate solutions to obtain information outside of the solution interval. Others use one scheme to get a guess to the solution while refining, or correcting, this to obtain better solutions as the iteration through time proceeds. Such methods are described in courses in numerical analysis and in the literature. At this point we will apply these methods to several physics problems before continuing with analytical solutions.

3.5 Numerical Applications

In this section we apply various numerical methods to several physics problems after setting them up. We first describe how to work with second order equations, such as the nonlinear pendulum problem. We will see that there is a bit more to numerically solving differential equations than to just running standard routines. As we explore these problems, we will introduce other methods and provide some MATLAB code indicating how one might set up the system.

Other problems covered in these applications are various free fall problems beginning with a falling body from a large distance from the Earth, to flying soccer balls, and falling raindrops. We will also discuss the numerical solution of the two body problem and the Friedmann equation as nonterrestrial applications.

3.5.1 The Nonlinear Pendulum

Now we will investigate the use of numercial methods for solving the nonlinear pendulum problem.

Example 3.4. Nonlinear pendulum Solve

$$\ddot{\theta} = -\frac{g}{L}\sin\theta, \quad \theta(0) = \theta_0, \quad \omega(0) = 0, \quad t \in [0, 8],$$

using Euler's Method. Use the parameter values of $m = 0.005$ kg, $L = 0.500$ m, and $g = 9.8$ m/s^2.

This is a second order differential equation. As describe later, we can write this differential equation as a system of two first order differential equations,

$$\begin{aligned} \dot{\theta} &= \omega, \\ \dot{\omega} &= -\frac{g}{L}\sin\theta. \end{aligned} \tag{3.26}$$

Defining the vector

$$\Theta(t) = \begin{pmatrix} \theta(t) \\ \omega(t) \end{pmatrix},$$

we can write the first order system as

$$\frac{d\Theta}{dt} = \mathbf{F}(t,\Theta), \quad \Theta(0) = \begin{pmatrix} \theta_0 \\ 0 \end{pmatrix},$$

where

$$F(t,\Theta) = \begin{pmatrix} \omega(t) \\ -\frac{g}{L}\sin\theta(t) \end{pmatrix}.$$

This allows us to use the the methods we have discussed on this first order equation for $\Theta(t)$.

For example, Euler's Method for this system becomes

$$\Theta_{i+1} = \Theta_{i+1} + h\mathbf{F}(t_i,\Theta_i)$$

with $\Theta_0 = \Theta(0)$.

We can write this scheme in component form as

$$\begin{pmatrix} \theta_{i+1} \\ \omega_{i+1} \end{pmatrix} = \begin{pmatrix} \theta_i \\ \omega_i \end{pmatrix} + h \begin{pmatrix} \omega_i \\ -\frac{g}{L}\sin\theta_i \end{pmatrix},$$

or

$$\begin{aligned} \theta_{i+1} &= \theta_i + h\omega_i, \\ \omega_{i+1} &= \omega_i - h\frac{g}{L}\sin\theta_i, \end{aligned} \tag{3.27}$$

starting with $\theta_0 = \theta_0$ and $\omega_0 = 0$.

The MATLAB code that can be used to implement this scheme takes the form

```
g=9.8;
L=0.5;
m=0.005;

a=0;
b=8;
N=500;
h=(b-a)/N;

% Initial Condition
  t(1)=0;
  theta(1)=pi/6;
  omega(1)=0;

% Euler's Method
  for i=2:N+1
      omega(i)=omega(i-1)-g/L*h*sin(theta(i-1));
      theta(i)=theta(i-1)+h*omega(i-1);
      t(i)=t(i-1)+h;
  end
```

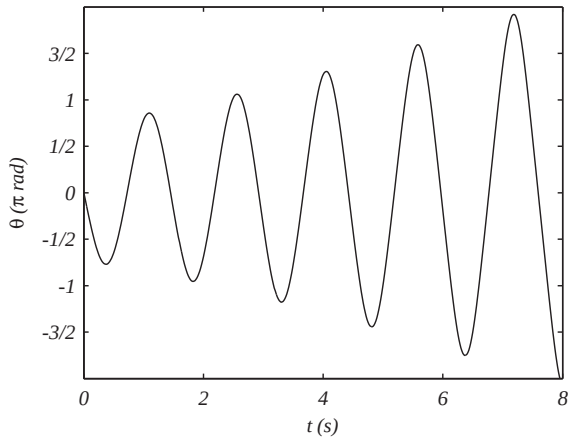


Figure 3.19: Solution for the nonlinear pendulum problem using Euler's Method on $t \in [0, 8]$ with $N = 500$.

In Figure 3.19 we plot the solution for a starting position of 30^o with $N = 500$. Notice that the amplitude of oscillation is increasing, contrary to our experience. So, we increase N and see if that helps. In Figure 3.20 we show the results for $N = 500$, 1000, and 2000 points, or $h = 0.016$, 0.008, and 0.004, respectively. We note that the amplitude is not increasing as much.

The problem with the solution is that Euler's Method is not an energy conserving method. As conservation of energy is important in physics, we would like to be able to seek problems which conserve energy. Such schemes used to solve oscillatory problems in classical mechanics are called symplectic integrators. A simple example is the Euler-Cromer, or semi-implicit Euler Method. We only need to make a small modification of Euler's Method. Namely, in the second equation of the method we use the updated value of the dependent variable as computed in the first line.

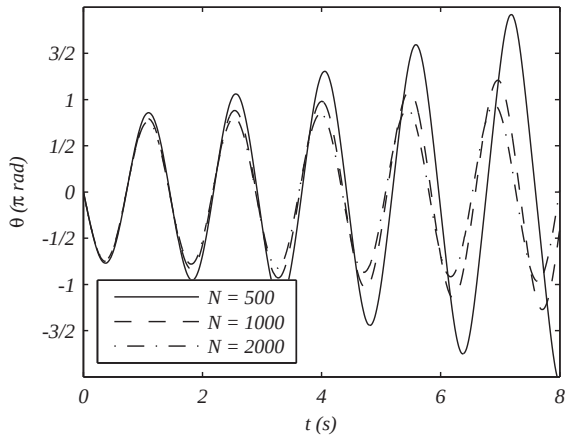


Figure 3.20: Solution for the nonlinear pendulum problem using Euler's Method on $t \in [0, 8]$ with $N = 500, 1000, 2000$.

Let's write the Euler scheme as

$$\omega_{i+1} = \omega_i - h\frac{g}{L}\sin\theta_i,$$

$$\theta_{i+1} = \theta_i + h\omega_i. \tag{3.28}$$

Then, we replace ω_i in the second line by ω_{i+1} to obtain the new scheme

$$\begin{aligned} \omega_{i+1} &= \omega_i - h\frac{g}{L}\sin\theta_i, \\ \theta_{i+1} &= \theta_i + h\omega_{i+1}. \end{aligned} \tag{3.29}$$

The MATLAB code is easily changed as shown below.

```
g=9.8;
L=0.5;
m=0.005;

a=0;
b=8;
N=500;
h=(b-a)/N;

% Initial Condition
  t(1)=0;
  theta(1)=pi/6;
  omega(1)=0;

% Euler-Cromer Method
  for i=2:N+1
      omega(i)=omega(i-1)-g/L*h*sin(theta(i-1));
      theta(i)=theta(i-1)+h*omega(i);
      t(i)=t(i-1)+h;
  end
```

We then run the new scheme for $N = 500$ and compare this with what we obtained before. The results are shown in Figure 3.21. We see that the oscillation amplitude seems to be under control. However, the best test would be to investigate if the energy is conserved.

Recall that the total mechanical energy for a pendulum consists of the kinetic and gravitational potential energies,

$$E = \frac{1}{2}mv^2 + mgh.$$

For the pendulum the tangential velocity is given by $v = L\omega$ and the height of the pendulum mass from the lowest point of the swing is $h = L(1 - \cos\theta)$. Therefore, in terms of the dynamical variables, we have

$$E = \frac{1}{2}mL^2\omega^2 + mgL(1 - \cos\theta).$$

We can compute the energy at each time step in the numerical simulation. In MATLAB it is easy to do using

```
E = 1/2*m*L^2*omega.^2+m*g*L*(1-cos(theta));
```

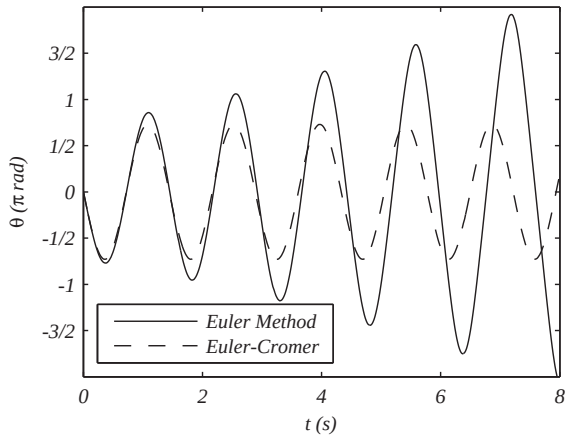


Figure 3.21: Solution for the nonlinear pendulum problem comparing Euler's Method and the Euler-Cromer Method on $t \in [0, 8]$ with $N = 500$.

after implementing the scheme. In other programming environments one needs to loop through the times steps and compute the energy along the way. In Figure 3.22 we shown the results for Euler's Method for $N = 500, 1000, 2000$ and the Euler-Cromer Method for $N = 500$. It is clear that the Euler-Cromer Method does a much better job at maintaining energy conservation.

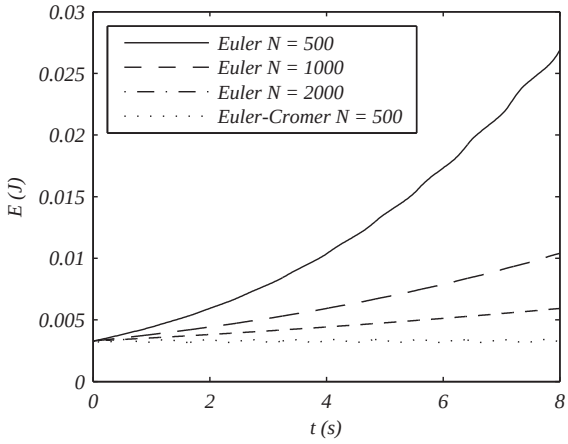


Figure 3.22: Total energy for the nonlinear pendulum problem.

3.5.2 *Extreme Sky Diving**

On October 14, 2012 Felix Baumgartner jumped from a helium balloon at an altitude of 39045 m (24.26 mi or 128100 ft). According preliminary data from the Red Bull Stratos Mission[1], as of November 6, 2012 Baumgartner experienced free fall until he opened his parachute at 1585 m after 4 minutes and 20 seconds. Within the first minute he had broken the record set by Joe Kittinger on August 16, 1960. Kittinger jumped from 102,800 feet (31 km) and fell freely for 4 minutes and 36 seconds to an altitude of 18,000 ft

[1] The original estimated data was found at the Red Bull Stratos site, http://www.redbullstratos.com/. Some of the data has since been updated. The reader can redo the solution using the updated data.

(5,500 m). Both set records for their times. Kittinger reached 614 mph (Mach 0.9) and Baumgartner reached 833.9 mph (Mach 1.24). Another record that was broken was that over 8 million watched the event on YouTube, breaking current live stream viewing events.

This much attention also peaked interest in the physics of free fall. Free fall at constant g through a height of h should take a time of

$$t = \sqrt{\frac{2h}{g}} = \sqrt{\frac{2(36,529)}{9.8}} = 86 \text{ s.}$$

Of course, g is not constant. In fact, at an altitude of 39 km, we have

$$g = \frac{GM}{R+h} = \frac{6.67 \times 10^{-11} \text{ N m}^2\text{kg}^2(5.97 \times 10^{24} \text{ kg})}{6375 + 39 \text{ km}} = 9.68 \text{ m/s}^2.$$

So, g is roughly constant.

Next, we need to consider the drag force as one free falls through the atmosphere, $F_D = \frac{1}{2}CA\rho_a v^2$. One needs some values for the parameters in this problem. Let's take $m = 90$ kg, $A = 1.0$ m^2, and $\rho = 1.29$ kg/m^3, $C = 0.42$. Then, a simple model would give

$$m\dot{v} = -mg + \frac{1}{2}CA\rho v^2,$$

or

$$\dot{v} = -g + .0030v^2.$$

This gives a terminal velocity of 57.2 m/s, or 128 mph. However, we again have assumed that the drag coefficient and air density are constant. Since the Reynolds number is high, we expect C is roughly constant. However, the density of the atmosphere is a function of altitude and we need to take this into account.

The Reynolds number is used several times in this chapter. It is defined as

$$Re = \frac{2rv}{\nu},$$

where ν is the kinematic viscosity. The kinematic viscosity of air at 60^o F is about 1.47×10^{-5} m^2/s.

[2] http://www.grc.nasa.gov/WWW/k-12/rocket/atmos.html

A simple model for $\rho = \rho(h)$ can be found at the NASA site.[2]. Using their data, we have

$$\rho(h) = \begin{cases} \dfrac{101290(1.000 - 0.2253 \times 10^{-4}h)^{5.256}}{83007 - 1.8696h}, & h < 11000, \\ .3629e^{1.73 - 0.157 \times 10^{-3}h}, & h, < 25000 \\ \dfrac{2488}{(.6551 + 0.1380 \times 10^{-4}h)^{11.388}(40876 + .8614h)}, & h > 25000. \end{cases} \tag{3.30}$$

In Figure 3.23 the atmospheric density is shown as a function of altitude.

In order to use the methods for solving first order equations, we write the system of equations in the form

$$\begin{aligned} \frac{dh}{dt} &= v, \\ \frac{dv}{dt} &= -\frac{GM}{(R+h)^2} + \frac{1}{5}\rho(h)CAv^2. \end{aligned} \tag{3.31}$$

This is now in the form of a system of first order differential equations.

Then, we define a function to be called and store in as **gravf.m** as shown below.

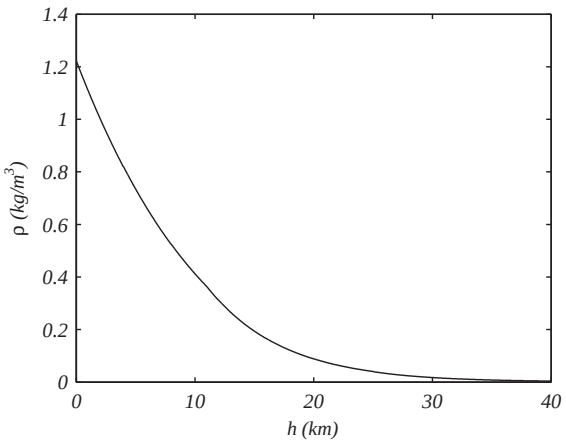


Figure 3.23: Atmospheric density as a function of altitude.

```
function dy=gravf(t,y);

G=6.67E-11;
M=5.97E24;
R=6375000;
m=90;
C=.42;
A=1;

dy(1,1)=y(2);
dy(2,1)=-G*M/(R+y(1)).^2+.5*density2(y(1))*C*A*y(2).^2/m;
```

Now we are ready to call the function in our favorite routine.

```
h0=1000;
tmax=20;
tmin=0;
[t,y]=ode45('dgravf',[tmin tmax],[h0;0]);% Const rho
plot(t,y(:,1),'k--')
```

Here we are simulating free fall from an altitude of one kilometer. In Figure 3.24 we compare different models of free fall with g taken as constant or derived from Newton's Law of Gravitation. We also consider constant density or the density dependence on the altitude as given earlier. We chose to keep the drag coefficient constant at $C = 0.42$.

We can see from these plots that the slight variation in the acceleration due to gravity does not have as much an effect as the variation of density with distance.

Now we can push the model to Baumgartner's jump from 39 km. In Figure 3.25 we compare the general model with that with no air resistance, though both taking into account the variation in g. As a body falls through the atmosphere we see the changing effects of the denser atmosphere on the free fall. For the parameters chosen, we find that it takes 238.8s, or a little

less than four minutes to reach the point where Baumgartner opened his parachute. While not exactly the same as the real fall, it is amazingly close.

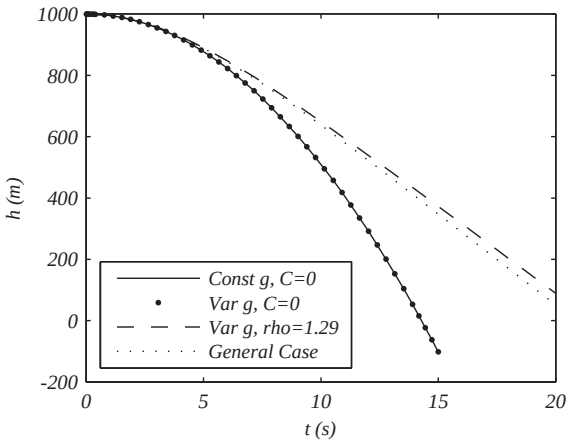


Figure 3.24: Comparison of different models of free fall from one kilometer above the Earth.

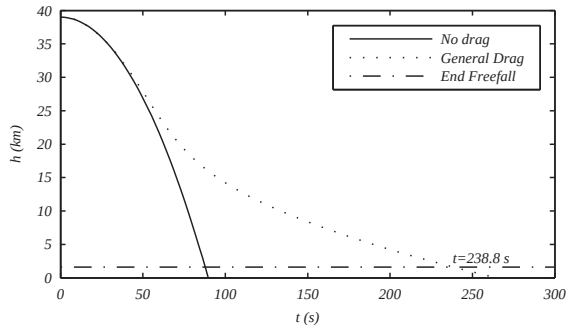


Figure 3.25: Free fall from 39 km at constant g as compared to nonconstant g and nonconstant atmospheric density with drag coefficient $C = .42$.

3.5.3 *The Flight of Sports Balls**

Another interesting problem is the projectile motion of a sports ball. In an introductory physics course, one typically ignores air resistance and the path of the ball is a nice parabolic curve. However, adding air resistance complicates the problem significantly and cannot be solved analytically. Examples in sports are flying soccer balls, golf balls, ping pong balls, baseballs, and other spherical balls.

We will consider a ball moving in the xz-plane spinning about an axis perpendicular to the plane of motion. Such an analysis was reported in Goff and Carré, AJP 77(11) 1020. The typical trajectory of the ball is shown in Figure 3.26. The forces acting on the ball are the drag force, $\mathbf{F}_D$, the lift force, $\mathbf{F}_L$, and the gravitational force, $\mathbf{F}_W$. These are indicated in Figure 3.27. The equation of motion takes the form

$$m\mathbf{a} = \mathbf{F}_W + \mathbf{F}_D + \mathbf{F}_L.$$

Writing out the components, we have

$$ma_x = -F_D \cos\theta - F_L \sin\theta \tag{3.32}$$
$$ma_z = -mg - F_D \sin\theta + F_L \cos\theta. \tag{3.33}$$

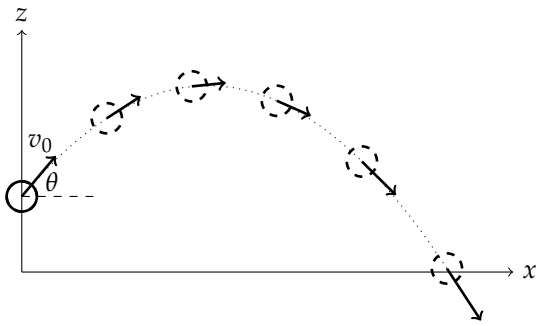


Figure 3.26: Sketch of the path for projectile motion problems.

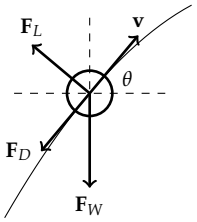


Figure 3.27: Forces acting on ball.

As we had seen before, the magnitude of the damping (drag) force is given by

$$F_D = \frac{1}{2} C_D \rho A v^2.$$

For the case of soccer ball dynamics, Goff and Carré noted that the Reynolds number, $Re = \frac{2rv}{\nu}$, is between 70000 and 490000 by using a kinematic viscosity of $\nu = 1.54 \times 10^{-5}$ m^2/s and typical speeds of $v = 4.5 - 31$ m/s. Their analysis gives $C_D \approx 0.2$. The parameters used for the ball were $m = 0.424$ kg and cross sectional area $A = 0.035$ m^2 and the density of air was taken as 1.2 kg/m^3.

The lift force takes a similar form,

$$F_L = \frac{1}{2} C_L \rho A v^2.$$

The sign of C_L indicates if the ball has top spin ($C_L < 0$) or bottom spin *$C_L > 0$). The lift force is just one component of a more general Magnus force, which is the force on a spinning object in a fluid and is perpendicular to the motion. In this example we assume that the spin axis is perpendicular to the plane of motion. Allowing for spinning balls to veer from this plane would mean that we would also need a component of the Magnus force perpendicular to the plane of motion. This would lead to an additional sideways component (in the **k** direction) leading to a third acceleration equation. We will leave that case for the reader.

The lift coefficient can be related to the spin as

$$C_L = \frac{1}{2 + \frac{v}{v_{spin}}},$$

where $v_{spin} = r\omega$ is the peripheral speed of the ball. Here R is the ball radius and ω is the angular speed in rad/s. If $v = 20$ m/s, $\omega = 200$ rad/s, and $r = 20$ mm, then $C_L = 0.45$.

So far, the problem has been reduced to

$$\frac{dv_x}{dt} = -\frac{\rho A}{2m}(C_D \cos\theta + C_L \sin\theta)v^2, \tag{3.34}$$
$$\frac{dv_z}{dt} = -g - \frac{\rho A}{2m}(C_D \sin\theta - C_L \cos\theta)v^2, \tag{3.35}$$

for v_x and v_z the components of the velocity. Also, $v^2 = v_x^2 + v_z^2$. Furthermore, from Figure 3.27, we can write

$$\cos\theta = \frac{v_x}{v}, \quad \sin\theta = \frac{v_z}{v}.$$

So, the equations can be written entirely as a system of differential equations for the velocity components,

$$\frac{dv_x}{dt} = -\alpha(C_D v_x + C_L v_z)(v_x^2 + v_z^2)^{1/2}, \tag{3.36}$$

$$\frac{dv_z}{dt} = -g - \alpha(C_D v_z - C_L v_x)(v_x^2 + v_z^2)^{1/2}, \tag{3.37}$$

where $\alpha = \rho A/2m = 0.0530\ \text{m}^{-1}$.

Such systems of equations can be solved numerically by thinking of this as a vector differential equation,

$$\frac{d\mathbf{v}}{dt} = \mathbf{F}(t, \mathbf{v}),$$

and applying one of the numerical methods for solving first order equations.

Since we are interested in the trajectory, $z = z(x)$, we would like to determine the parametric form of the path, $(x(t), z(t))$. So, instead of solving two first order equations for the velocity components, we can rewrite the two second order differential equations for $x(t)$ and $z(t)$ as four first order differential equations of the form

$$\frac{d\mathbf{y}}{dt} = \mathbf{F}(t, \mathbf{y}).$$

We first define

$$\mathbf{y} = \begin{bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \\ y_4(t) \end{bmatrix} = \begin{bmatrix} x(t) \\ z(t) \\ v_x(t) \\ v_z(t) \end{bmatrix}$$

Then, the systems of first order differential equations becomes

$$\begin{aligned} \frac{dy_1}{dt} &= y_3, \\ \frac{dy_2}{dt} &= y_4, \\ \frac{dy_3}{dt} &= -\alpha(C_D v_x + C_L v_z)(v_x^2 + v_z^2)^{1/2}, \\ \frac{dy_4}{dt} &= -g - \alpha(C_D v_z - C_L v_x)(v_x^2 + v_z^2)^{1/2}. \end{aligned} \tag{3.38}$$

The system can be placed into a function file which can be called by an ODE solver, such as the MATLAB m-file below.

```
function dy = ballf(t,y)
global g CD CL alpha

dy = zeros(4,1);    % a column vector
v = sqrt(y(3).^2+y(4).^2);   % speed v

dy(1) = y(3);
dy(2) = y(4);
dy(3) = -alpha*v.*(CD*y(3)+CL*y(4));
dy(4) = alpha*v.*(-CD*y(4)+CL*y(3))-g;
```

Then, the solver can be called using

```
[T,Y] = ode45('ballf',[0 2.5],[x0,z0,v0x,v0z]);
```

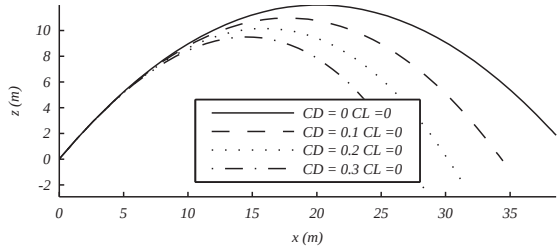


Figure 3.28: Example of soccer ball under the influence of drag.

In Figures 3.28 and 3.29 we indicate what typical solutions would look like for different values of drag and lift coefficients. In the case of nonzero lift coefficients, we indicate positive and negative values leading to flight with top spin, $C_L < 0$, or bottom spin, $C_L > 0$.

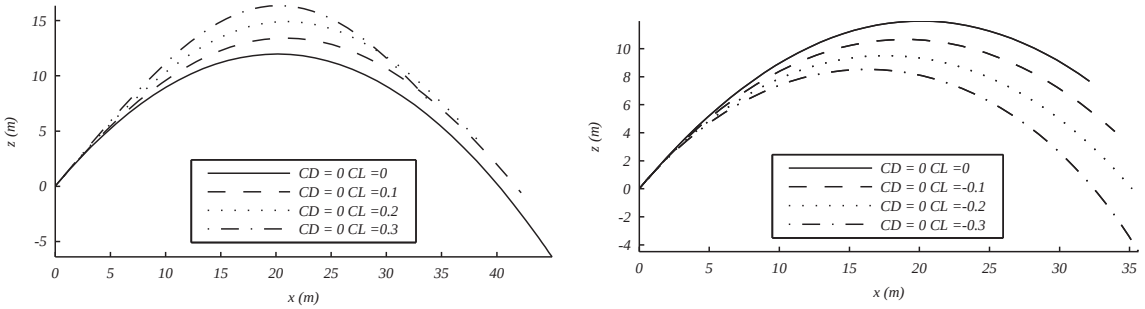


Figure 3.29: Example of soccer ball under the influence of lift with $CL > 0$ and $CL < 0$

3.5.4 *Falling Raindrops**

A SIMPLE PROBLEM THAT APPEARS IN MECHANICS is that of a falling raindrop through a mist. The raindrop not only undergoes free fall, but the mass of the drop grows as it interacts with the mist. There have been several papers written on this problem and it is a nice example to explore using numerical methods. In this section we look at models of a falling raindrop with and without air drag.

First we consider the case in which there is no air drag. A simple model of free fall from Newton's Second Law of Motion is

$$\frac{d(mv)}{dt} = mg.$$

In this discussion we will take downward as positive. Since the mass is not constant. we have

$$m\frac{dv}{dt} = mg - v\frac{dm}{dt}.$$

In order to proceed, we need to specify the rate at which the mass is changing. There are several models one can adapt.We will borrow some of the ideas and in some cases the numerical values from Sokal(2010)[3] and Edwards, Wilder, and Scime (2001).[4] These papers also quote other interesting work on the topic.

[3] A. D. Sokal, *The falling raindrop, revisited*, Am. J. Phys. 78, 643-645, (2010).

[4] B. F. Edwards, J. W. Wilder, and E. E. Scime, *Dynamics of Falling Raindrops*, Eur. J. Phys. 22, 113-118, (2001).

While v and m are functions of time, one can look for a way to eliminate time by assuming the rate of change of mass is an explicit function of m and v alone. For example, Sokal (2010) assumes the form

$$\frac{dm}{dt} = \lambda m^{\sigma} v^{\beta}, \quad \lambda > 0.$$

This contains two commonly assumed models of accretion:

1. $\sigma = 2/3$, $\beta = 0$. This corresponds to growth of the raindrop proportional to the surface area. Since $m \propto r^3$ and $A \propto r^2$, then $\dot{m} \propto A$ implies that $\dot{m} \propto m^{2/3}$.
2. $\sigma = 2/3$, $\beta = 1$. In this case the growth of the raindrop is proportional to the volume swept out along the path. Thus, $\Delta m \propto A(v\Delta t)$, where A is the cross sectional area and $v\Delta t$ is the distance traveled in time Δt.

In both cases, the limiting value of the acceleration is a constant. It is $g/4$ in the first case and $g/7$ in the second case.

Another approach might be to use the effective radius of the drop, assuming that the raindrop remains close to spherical during the fall. According to Edwards, Wilder, and Scime (2001), raindrops with Reynolds number greater than 1000 and with radii larger than 1 mm will flatten. Even larger raindrops will break up when the drag force exceeds the surface tension. Therefore, they take $0.1 \text{ mm} < r < 1 \text{ mm}$ and $10 < Re < 1000$. We will return to a discussion of the drag later.

It might seem more natural to make the radius the dynamic variable, than the mass. In this case, we can assume the accretion rate takes the form

$$\frac{dr}{dt} = \gamma r^{\alpha} v^{\beta}, \quad \gamma > 0.$$

Since, $m = \frac{4}{3}\pi\rho_d r^3$,

$$\frac{dm}{dt} \sim r^2 \frac{dr}{dt} \sim m^{2/3} \frac{dr}{dt}.$$

Therefore, the two special cases become

1. $\alpha = 0$, $\beta = 0$. This corresponds to a growth of the raindrop proportional to the surface area.
2. $\alpha = 0$, $\beta = 1$. In this case the growth of the raindrop is proportional to the volume swept out along the path.

Here ρ_d is the density of the raindrop.

We will also need

$$\begin{aligned}\frac{v}{m}\frac{dm}{dt} &= \frac{4\pi\rho_d r^2}{\frac{4}{3}\pi\rho_d r^3} v\frac{dr}{dt} \\ &= 3\frac{v}{r}\frac{dr}{dt} \\ &= 3\gamma r^{\alpha-1}v^{\beta+1}. \end{aligned} \tag{3.39}$$

Putting this all together, we have a systems of two equations for $v(t)$ and $r(t)$:

$$\begin{aligned}\frac{dv}{dt} &= g - 3\gamma r^{\alpha-1}v^{\beta+1}, \\ \frac{dr}{dt} &= \gamma r^{\alpha}v^{\beta}. \end{aligned} \tag{3.40}$$

Example 3.5. Determine $v = v(r)$ for the case $\alpha = 0$, $\beta = 0$ and the initial conditions $r(0) = 0.1$ mm and $v(0) = 0$ m/s.

In this case Equations (3.40) become

$$\begin{aligned}\frac{dv}{dt} &= g - 3\gamma r^{-1}v, \\ \frac{dr}{dt} &= \gamma. \end{aligned} \tag{3.41}$$

Noting that

$$\frac{dv}{dt} = \frac{dv}{dr}\frac{dr}{dt} = \gamma\frac{dv}{dr},$$

we can convert the problem to one of finding the solution $v(r)$ subject to the equation

$$\frac{dv}{dr} = \frac{g}{\gamma} - 3\frac{v}{r}$$

with the initial condition $v(r_0) = 0$ m/s for $r_0 = 0.0001$ m.

Rearranging the differential equation, we find that it is a linear first order differential equation,

$$\frac{dv}{dr} + \frac{3}{r}v = \frac{g}{\gamma}.$$

This equation can be solved using an integrating factor, $\mu = r^3$, obtaining

$$\frac{d}{dr}(r^3 v) = \frac{g}{\gamma}r^3.$$

Integrating, we obtain the solution

$$v(r) = \frac{g}{4\gamma}r\left(1 - \left(\frac{r_0}{r}\right)^4\right).$$

Note that for large r, $v \sim \frac{g}{4\gamma}r$. Therefore, $\frac{dv}{dt} \sim \frac{g}{4}$.

While this case was easily solved in terms of elementary operations, it is not always easy to generate solutions to Equations (3.40) analytically. Sokal (2010) derived a general solution in terms of incomplete Beta functions,

though this does not help visualize the solution. Also, as we will see, adding air drag will lead to a nonintegrable system. So, we turn to numerical solutions.

In MATLAB, we can use the function in **raindropf.m** to capture the system (3.40). Here we put the velocity in y(1) and the radius in y(2).

```
function dy=raindropf(t,y);
global alpha beta gamma g

dy=[g-3*gamma*y(2)^(alpha-1)*y(1)^(beta+1); ...
    gamma*y(2)^alpha*y(1)^beta];
```

We then use the Runge-Kutta solver, **ode45**, to solve the system. An implementation is shown below which calls the function containing the system. The value $\gamma = 2.5 \times 10^{-7}$ is based on empirical results quoted by Edwards, Wilder, and Scime (2001).

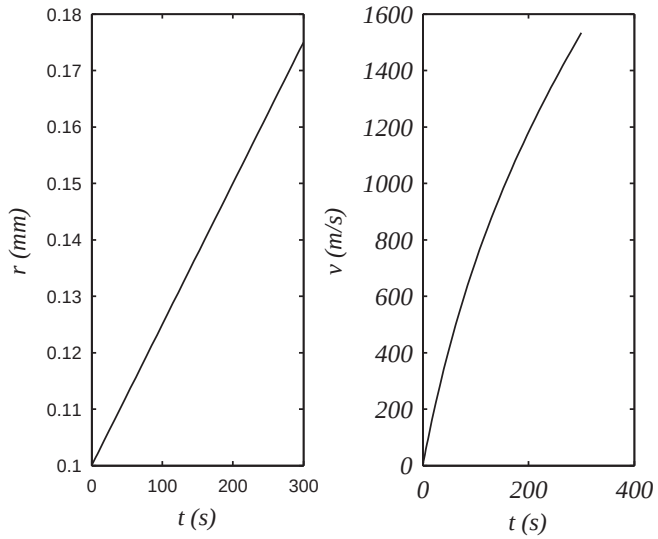


Figure 3.30: The plots of position and velocity as a function of time for $\alpha = \beta = 0$.

```
clear
global alpha beta gamma g

alpha=0;
beta=0;
gamma=2.5e-07;
g=9.81;

r0=0.0001;
v0=0;
y0=[v0;r0];
tspan=[0 1000];
```

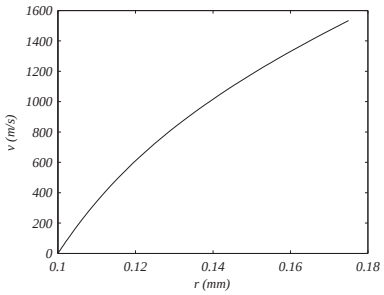


Figure 3.31: The plot the velocity as a function of position for $\alpha = \beta = 0$.

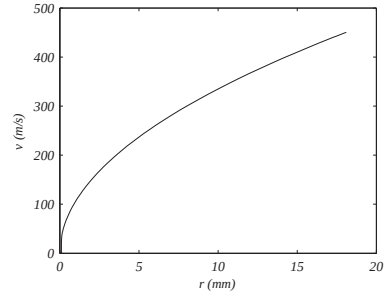


Figure 3.32: The plot the velocity as a function of position for $\alpha = 0, \beta = 1$.

```
[t,y]=ode45(@raindropf,tspan,y0);
plot(1000*y(:,2),y(:,1),'k')
```

The resulting plots are shown in Figures 3.30-3.31. The plot of velocity as a function of position agrees with the exact solution, which we derived in the last example. We note that these drops do not grow much, but they seem to attain large speeds.

For the second case, $\alpha = 0, \beta = 1$, one can also obtain an exact solution. The result is

$$v(r) = \left[\frac{2g}{7\gamma} r \left(1 - \left(\frac{r_0}{r}\right)^7\right)\right]^{\frac{1}{2}}.$$

For large r one can show that $\frac{dv}{dt} \sim \frac{g}{7}$. In Figures 3.33-3.32 we see again large velocities, though about a third as fast over the same time interval. However, we also see that the raindrop has significantly grown well past the point it would break up.

In this simple model of a falling raindrop we have not considered air drag. Earlier in the chapter we discussed the free fall of a body with air resistance and this lead to a terminal velocity. Recall that the drag force given by

$$f_D(v) = -\frac{1}{2} C_D A \rho_a v^2, \tag{3.42}$$

where C_D is the drag coefficient, A is the cross sectional area and ρ_a is the air density. Also, we assume that the body is falling downward and downward is positive, so that $f_D(v) < 0$ so as to oppose the motion.

We would like to incorporate this force into our model (3.40). The first equation came from the force law, which now becomes

$$m\frac{dv}{dt} = mg - v\frac{dm}{dt} - \frac{1}{2} C_D A \rho_a v^2,$$

or

$$\frac{dv}{dt} = g - \frac{v}{m}\frac{dm}{dt} - \frac{1}{2m} C_D A \rho_a v^2.$$

The next step is to eliminate the dependence on the mass, m, in favor of the radius, r. The drag force term can be written as

$$\begin{aligned} \frac{f_D}{m} &= \frac{1}{2m} C_D A \rho_a v^2 \\ &= \frac{1}{2} C_D \frac{\pi r^2}{\frac{4}{3}\pi \rho_d r^3} \rho_a v^2 \\ &= \frac{3}{8}\frac{\rho_a}{\rho_d} C_D \frac{v^2}{r}. \end{aligned} \tag{3.43}$$

We had already done this for the second term; however, Edwards, Wilder, and Scime (2001) point to experimental data and propose that

$$\frac{dm}{dt} = \pi \rho_m r^2 v,$$

where ρ_m is the mist density. So, the second terms leads to

$$\frac{v}{m}\frac{dm}{dt} = \frac{3}{4}\frac{\rho_m}{\rho_d}\frac{v^2}{r}.$$

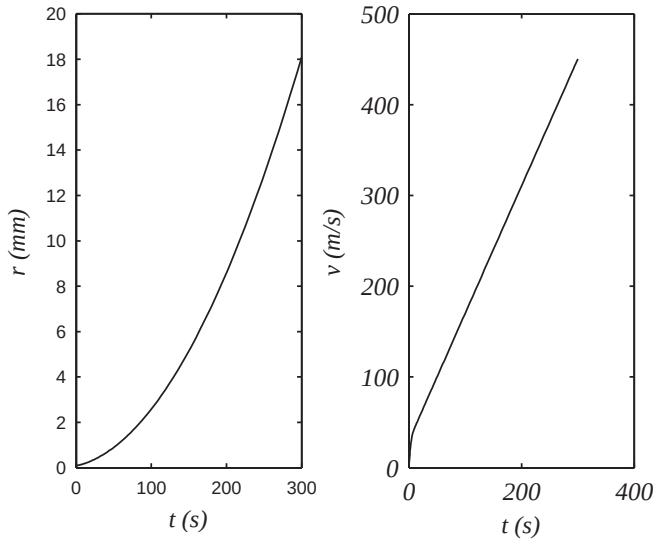


Figure 3.33: The plots of position and velocity as a function of time for $\alpha = 0, \beta = 1$.

But, since $m = \frac{4}{3}\pi\rho_d r^3$,

$$\frac{dm}{dt} = 4\pi\rho_d r^2 \frac{dr}{dt}.$$

So,

$$\frac{dr}{dt} = \frac{\rho_m}{4\rho_d} v.$$

This suggests that their model corresponds to $\alpha = 0$, $\beta = 1$, and $\gamma = \frac{\rho_m}{4\rho_d}$.

Now we can write down the modified system

$$\begin{aligned} \frac{dv}{dt} &= g - 3\gamma r^{\alpha-1} v^{\beta+1} - \frac{3}{8}\frac{\rho_a}{\rho_d} C_D \frac{v^2}{r}, \\ \frac{dr}{dt} &= \gamma r^\alpha v^\beta. \end{aligned} \tag{3.44}$$

Edwards, Wilder, and Scime (2001) assume that the densities are constant with values $\rho_a = .856$ kg/m^3, $\rho_d = 1.000$ kg/m^3, and $\rho_m = 1.00 \times 10^{-3}$ kg/m^3. However, the drag coefficient is not constant. As described later in Section 3.5.7, there are various models indicating the dependence of C_D on the Reynolds number,

$$Re = \frac{2rv}{\nu},$$

where ν is the kinematic viscosity, which Edwards, Wilder, and Scime (2001) set to $\nu = 2.06 \times 10^{-5}$ m^2/s. For raindrops of the range $r = 0.1$ mm to 1 mm, the Reynolds number is below 1000. Edwards, Wilder, and Scime (2001) modeled $C_D = 12Re^{-1/2}$. In the plots in Section 3.5.7 we include this model and see that this is a good approximation for these raindrops. In Chapter 10 we discuss least squares curve fitting and using these methods, one can use the models of Putnam (1961) and Schiller-Naumann (1933) to obtain a power law fit similar to that used here.

So, introducing

$$C_D = 12Re^{-1/2} = 12\left(\frac{2rv}{\nu}\right)^{-1/2}$$

and defining

$$\delta = \frac{9}{2^{3/2}}\frac{\rho_a}{\rho_d}\nu^{1/2},$$

we can write the system of equations (3.44) as

$$\begin{aligned}\frac{dv}{dt} &= g - 3\gamma\frac{v^2}{r} - \delta\left(\frac{v}{r}\right)^{\frac{3}{2}},\\ \frac{dr}{dt} &= \gamma v.\end{aligned} \tag{3.45}$$

Now, we can modify the MATLAB code for the raindrop by adding the extra term to the first equation, setting $\alpha = 0$, $\beta = 1$, and using $\delta = 0.0124$ and $\gamma = 2.5 \times 10^{-7}$ from Edwards, Wilder, and Scime (2001).

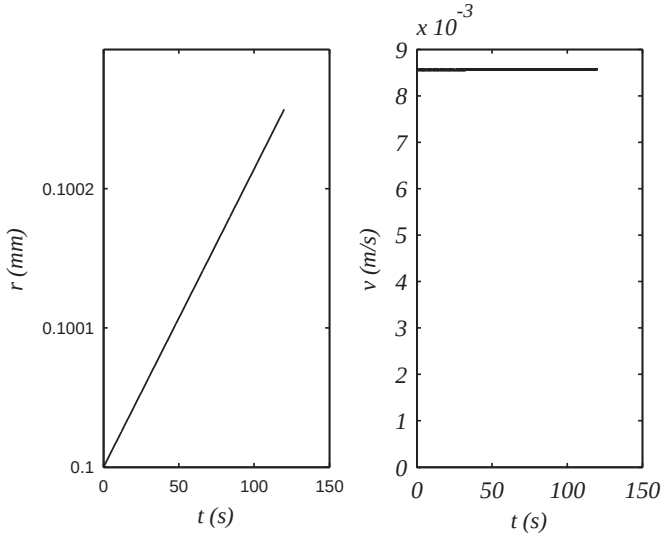


Figure 3.34: The plots of position and velocity as a function of time with air drag included.

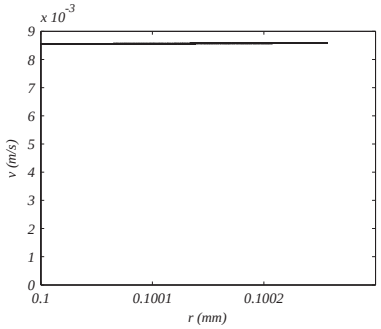


Figure 3.35: The plot the velocity as a function of position with air drag included.

In Figures 3.34-3.35 we see different behaviors as compared to the previous models. It appears that the velocity quickly reaches a terminal velocity and the radius continues to grow linearly in time, though at a slow rate.

We might be able to understand this behavior. Terminal, or constant v, would occur when

$$g - 3\gamma\frac{v^2}{r} - \delta\left(\frac{v}{r}\right)^{\frac{3}{2}} = 0.$$

Looking at these terms, one finds that the second term is significantly smaller than the other terms and thus

$$\delta\left(\frac{v}{r}\right)^{\frac{3}{2}} \approx g,$$

or

$$\frac{v}{r} \approx \left(\frac{g}{\delta}\right)^{2/3} \approx 85.54\ \text{s}^{-1}.$$

This agrees with the numerical data which gives the slope of the v vs r plot as 85.5236 s^{-1}.

3.5.5 *The Two-body Problem**

A STANDARD PROBLEM IN CLASSICAL DYNAMICS is the study of the motion of several bodies under the influence of Newton's Law of Gravitation. The so-called n-body problem is not solvable. However, the two body problem is. Such problems can model the motion of a planet around the sun, the moon around the Earth, or a satellite around the Earth. Further interesting, and more realistic problems, would involve perturbations of these orbits due to additional bodies. For example, one can study problems such as the influence of large planets on the asteroid belt. Since there are no analytic solutions to these problems, we have to resort to finding numerical solutions. We will look at the two body problem since we can compare the numerical methods to the exact solutions.

We consider two masses, m_1 and m_2, located at positions, $\mathbf{r}_1$ and $\mathbf{r}_2$, respectively, as shown in Figure 3.36. Newton's Law of Gravitation for the force between two masses separated by position vector $\mathbf{r}$ is given by

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

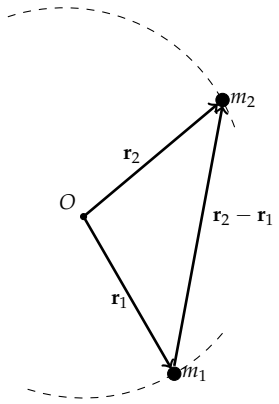


Figure 3.36: Two masses interact under Newton's Law of Gravitation.

Each mass experiences this force due to the other mass. This gives the system of equations

$$m_1\ddot{\mathbf{r}}_1 = -\frac{Gm_1m_2}{|\mathbf{r}_2 - \mathbf{r}_1|^3}(\mathbf{r}_1 - \mathbf{r}_2) \tag{3.46}$$

$$m_2\ddot{\mathbf{r}}_2 = -\frac{Gm_1m_2}{|\mathbf{r}_2 - \mathbf{r}_1|^3}(\mathbf{r}_2 - \mathbf{r}_1). \tag{3.47}$$

Now we seek to set up this system so that we can find numerical solutions for the positions of the masses. From the conservation of angular momentum, we know that the motion takes place in a plane. [Note: The solution of the Kepler Problem is discussed in Chapter 9.] We will choose the orbital plane to be the xy-plane. We define $r_{12} = |\mathbf{r}_2 - \mathbf{r}_1|$, and $(x_i, y_i) = \mathbf{r}_i$, $i = 1, 2$. Furthermore, we write the two second order equations as four first order equations. So, defining the velocity components as $(u_i, v_i) = \mathbf{v}_i$, the system of equations can be written in the form

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ u_1 \\ v_1 \\ u_2 \\ v_2 \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ -\alpha m_2(x_1 - x_2) \\ -\alpha m_2(y_1 - y_2) \\ -\alpha m_1(x_2 - x_1) \\ -\alpha m_1(y_2 - y_1). \end{pmatrix}, \tag{3.48}$$

where $\alpha = \frac{G}{r_{12}^3}$.

This system can be encoded in MATLAB as indicated in the function **twobody**:

```
function dz = twobody(t,z)
dz = zeros(8,1);
G = 1;
m1 = .1;
m2 = 2;
r=((z(1) - z(3)).^2 + (z(2) - z(4)).^2).^(3/2);
alpha=G/r;
dz(1) = z(5);
dz(2) = z(6);
dz(3) = z(7);
dz(4) = z(8);
dz(5) = alpha*m2*(z(3) - z(1));
dz(6) = alpha*m2*(z(4) - z(2));
dz(7) = alpha*m1*(z(1) - z(3));
dz(8) = alpha*m1*(z(2) - z(4));
```

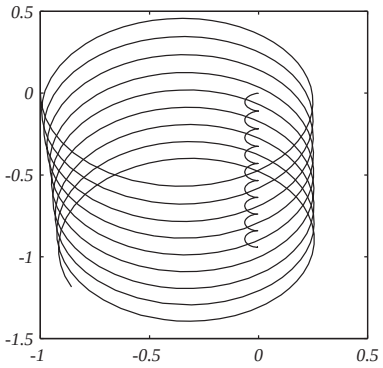


Figure 3.37: Simulation of two bodies under gravitational attraction.

In the above code we picked some seemingly nonphysical numbers for G and the masses. Calling **ode45** with a set of initial conditions,

```
[t,z] = ode45('twobody',[0 20], [-1 0 0 0 0 -1 0 0]);
plot(z(:,1),z(:,2),'k',z(:,3),z(:,4),'k');
```

we obtain the plot shown in Figure 3.37. We see each mass moves along what looks like elliptical helices with the smaller body tracing out a larger orbit.

In the case of a very large body, most of the motion will be due to the smaller body. So, it might be better to plot the relative motion of the small body with respect to the larger body. Actually, an analysis of the two body problem shows that the center of mass

$$\mathbf{R} = \frac{m_1\mathbf{r}_1 + m_2\mathbf{r}_2}{m_1 + m_2}$$

satisfies $\ddot{\mathbf{R}} = 0$. Therefore, the system moves with a constant velocity.

The relative position of the masses is defined through the variable $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$. Dividing the masses from the left hand side of Equations (3.47) and subtracting, we have the motion of m_1 about m_2

$$\ddot{\mathbf{r}} = -G(m_1 + m_2)\frac{\mathbf{r}}{r^3},$$

where $r = |\mathbf{r}| = |\mathbf{r}_1 - \mathbf{r}_2|$. Note that $\mathbf{r} \times \ddot{\mathbf{r}} = 0$. Integrating, this gives $\mathbf{r} \times \dot{\mathbf{r}} =$ constant. This is just a statement of the conservation of angular momentum.

The orbiting body will remain in a plane and, therefore, we can take the z-axis perpendicular to $\mathbf{r} \times \dot{\mathbf{r}}$, the position as $\mathbf{r} = (x(t), y(t))$, and the velocity as $\dot{\mathbf{r}} = (u(t), v(t))$. Then, the equations of motion can be written as four first order equations:

$$\begin{aligned} \dot{x} &= u \\ \dot{y} &= v \end{aligned}$$

$$\begin{aligned} \dot{u} &= -\mu \frac{x}{r^3} \\ \dot{v} &= -\mu \frac{y}{r^3}, \end{aligned} \tag{3.49}$$

where $\mu = G(m_1 + m_2)$ and $r = \sqrt{x^2 + y^2}$.

While we have established a system of equations which can be integrated, we should note a few results from the study of the Kepler problem in classical dynamics which we review in Chapter 9. Kepler's Laws of Planetary Motion state:

1. All planets travel in ellipses.

   The polar equation for the path is given by

   $$r = \frac{a(1-e^2)}{1 + e\cos\phi},$$

   where e is the eccentricity and a is the length of the semimajor axis. For $0 \leq e < 1$, the orbit is an ellipse.

2. A planet sweeps out equal areas in equal times.

3. The square of the period of the orbit is proportional to the cube of the semimajor axis. In particular, one can show that

   $$T^2 = \frac{4\pi^2}{\mu} a^3.$$

   By an appropriate choice of units, we can make $\mu = G(m_1 + m_2)$ a reasonable number. For the Earth-Sun system,

   $$\begin{aligned} \mu &= 6.67 \times 10^{-11} m^3 kg^{-1} s^{-2} (1.99 \times 10^{30} + 5.97 \times 10^{24}) kg \\ &= 1.33 \times 10^{20} m^3 s^{-1}. \end{aligned}$$

   That is a large number and can cause problems in the numerics. However, if one uses astronomical scales, such as putting lengths in astronomical units, 1 AU $= 1.50 \times 10^8$ km, and time in years, then

   $$\mu = \frac{4\pi^2}{T^2} a^3 = 4\pi^2$$

   in units of AU^3/yr^2.

Setting $\phi = 0$, the location of the perigee is given by

$$r = \frac{a(1-e^2)}{1+e} = a(1-e),$$

or

$$\mathbf{r} = (a(1-e), 0).$$

At this point the velocity is given by

$$\dot{\mathbf{r}} = \left(0, \sqrt{\frac{\mu}{a}\frac{1+e}{1-e}}\right).$$

Knowing the position and velocity at $\phi = 0$,, we can set the initial conditions for a bound orbit. The MATLAB code based on the above analysis is given below and the solution can be seen in Figure 3.38.

```
e=0.9;
tspan=[0 100];
z0=[1-e;0;0;sqrt((1+e)/(1-e))];
[t,z] = ode45('twobodyf',tspan, z0);
plot(z(:,1),z(:,2),'k');
axis equal

function dz = twobodyf(t,z)
dz = zeros(4,1);
GM = 1;

r=(z(1).^2 + z(2).^2).^(3/2);
alpha=GM/r;
dz(1) = z(3);
dz(2) = z(4);
dz(3) = -alpha*z(1);
dz(4) = -alpha*z(2);
```

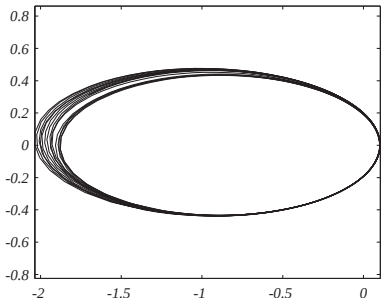


Figure 3.38: Simulation of one body orbiting a larger body under gravitational attraction.

While it is clear that the mass is following an elliptical orbit, we see that it will only do so for a finite period of time partly because the Runge-Kutta code does not conserve energy and it does not conserve the angular momentum. The conservation of energy is found (up to a factor of m_1) as

$$\frac{1}{2}(\dot{x}^2 + \dot{y}^2) - \frac{\mu}{t} = -\frac{\mu}{2a}.$$

Similarly, the conservation of (specific) angular momentum is given by

$$\mathbf{r} \times \mathbf{v} = (x\dot{y} - y\dot{x})\mathbf{k} = \sqrt{\mu a(1-e^2)}\mathbf{k}.$$

As was the case with the nonlinear pendulum example, we saw that an implicit Euler method, or Cromer's method, was better at conserving energy. So, we compare the Euler's Method version with the Implicit-Euler Method. In general, we seek to solve the system

$$\begin{aligned} \dot{\mathbf{r}} &= \mathbf{F}(\mathbf{r}, \mathbf{v}), \\ \dot{\mathbf{v}} &= \mathbf{G}(\mathbf{r}, \mathbf{v}). \end{aligned} \tag{3.50}$$

As we had seen earlier, Euler's Method is given by

$$\begin{aligned} \mathbf{v}_n &= \mathbf{v}_{n-1} + \Delta t * \mathbf{G}(t_{n-1}, \mathbf{x}_{n-1}), \\ \mathbf{r}_n &= \mathbf{r}_{n-1} + \Delta t * \mathbf{F}(t_{n-1}, \mathbf{v}_{n-1}). \end{aligned} \tag{3.51}$$

Euler's Method for the two body problem

For the two body problem, we can write out the Euler Method steps using $\mathbf{v} = (u, v)$, $\mathbf{r} = (x, y)$, $\mathbf{F} = (u, v)$, and $\mathbf{G} = -\frac{\mu}{r^3}(x, y)$. The MATLAB code would use the loop

```
for i=2:N+1
     alpha=mu/(x(i-1).^2 + y(i-1).^2).^(3/2);
     u(i)=u(i-1)-h*alpha*x(i-1);
```

```
    v(i)=v(i-1)-h*alpha*y(i-1);
    x(i)=x(i-1)+h*u(i-1);
    y(i)=y(i-1)+h*v(i-1);
    t(i)=t(i-1)+h;
end
```

Note that more compact forms can be used, but they are not readily adaptable to other packages or programming languages.

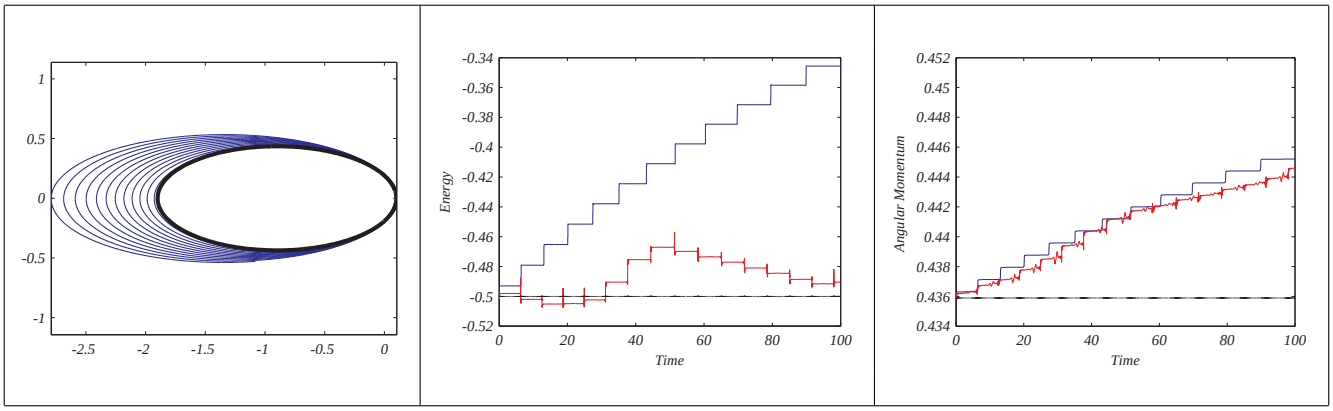


Table 3.8: Results for using Euler method for $N = 4000000$ and $t \in [0, 100]$. The parameters are $\mu = 1$, $e = 0,9$, and $a = 1$.

In Figure 3.8 we show the results along with the energy and angular momentum plots for $N = 4000000$ and $t \in [0, 100]$ for the case of $\mu = 1$, $e = 0,9$, and $a = 1$. The orbit based on the exact solution is in the center of the figure on the left. The energy and angular momentum as a function of time are shown along with the similar plots obtained using **ode45**. In neither case are these two quantities conserved.

```
for i=2:N+1
    alpha=mu/(x(i-1).^2 + y(i-1).^2).^(3/2);
    u(i)=u(i-1)-h*alpha*x(i-1);
    v(i)=v(i-1)-h*alpha*y(i-1);
    x(i)=x(i-1)+h*u(i);
    y(i)=y(i-1)+h*v(i);
    t(i)=t(i-1)+h;
 end
```

Implicit-Euler Method for the two body problem

The Implicit-Euler Method is a slight modification to the Euler Method and has a better chance at handing the conserved quantities as the Implicit-Euler Method is one of many symplectic integrators. The modification uses the new value of the velocities in the updating of the position. Thus, we have

$$\begin{aligned} \mathbf{v}_n &= \mathbf{v}_{n-1} + \Delta t * \mathbf{G}(t_{n-1}, \mathbf{x}_{n-1}), \\ \mathbf{r}_n &= \mathbf{r}_{n-1} + \Delta t * \mathbf{F}(t_{n-1}, \mathbf{v}_n). \end{aligned} \tag{3.52}$$

It is a simple matter to update the MATLAB code. In Figure 3.9 we show the results along with the energy and angular momentum plots for $N =$

200000 and $t \in [0, 100]$ for the case of $\mu = 1$, $e = 0,9$, and $a = 1$. The orbit based on the exact solution coincides with the orbit as seen in the left figure. The energy and angular momentum as functions of time are appear to be conserved. The energy fluctuates about -0.5 and the angular momentum remains constant. Again, the **ode45** results are shown in comparison. The number of time steps has been decreased from the Euler Method by a factor of 20.

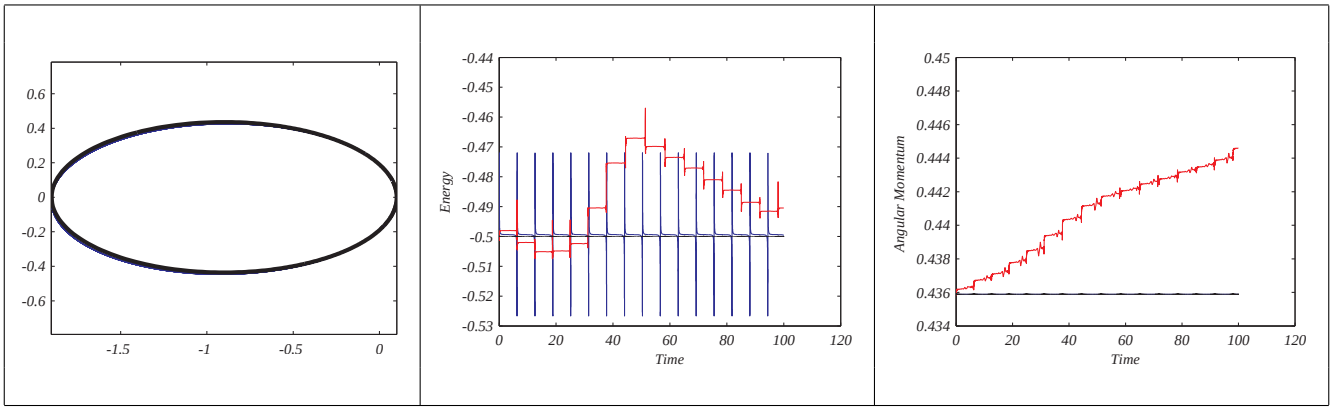


Table 3.9: Results for using the Implicit-Euler method for $N = 200000$ and $t \in [0, 100]$. The parameters are $\mu = 1$, $e = 0,9$, and $a = 1$.

The Euler and Implicit Euler are first order methods. We can attempt a faster and more accurate process which is also a symplectic method. As a final example, we introduce the velocity Verlet method for solving

$$\ddot{\mathbf{r}} = \mathbf{a}(\mathbf{r}(t)).$$

The derivation is based on a simple Taylor expansion:

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(\mathbf{t})\Delta t^2 + \cdots.$$

Replace Δt with $-\Delta t$ to obtain

$$\mathbf{r}(t - \Delta t) = \mathbf{r}(t) - \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(\mathbf{t})\Delta t^2 - \cdots.$$

Now, adding these expressions leads to some cancellations,

$$\mathbf{r}(t + \Delta t) = 2\mathbf{r}(t) - \mathbf{r}(t - \Delta t) + \mathbf{a}(\mathbf{t})\Delta t^2 + O(\Delta t^4).$$

Writing this in a more useful form, we have

$$\mathbf{r}_{n+1} = 2\mathbf{r}_n - \mathbf{r}_{n-1} + \mathbf{a}(\mathbf{r}_n)\Delta t^2.$$

Thus, we can find $\mathbf{r}_{n+1}$ from the previous two values without knowing the velocity. This method is called the Verlet, or Störmer-Verlet Method.

Loup Verlet (1931-) is a physicist who works on molecular dynamics and Fredrik Carl Mülertz Störmer (1874-1957) was a mathematician and physicist who modeled the motion of charged particles in his studies of the aurora borealis.

It is useful to know the velocity so that we can check energy conservation and angular momentum conservation. The Verlet Method can be rewritten in an equivalent form know as the velocity Verlet method. We use

$$\mathbf{r}(t) - \mathbf{r}(t - \Delta t) \approx \mathbf{v}(t)\Delta t - \frac{1}{2}\mathbf{a}\Delta t^2$$

in the Stömer-Verlet Method and write

$$\begin{aligned} \mathbf{r}_n &= \mathbf{r}_{n-1} + \mathbf{v}_{n-1}u + \frac{h^2}{2}\mathbf{a}(\mathbf{r}_{n-1}), \\ \mathbf{v}_{n-1/2} &= \mathbf{v}_{n-1} + \frac{h}{2}\mathbf{a}(\mathbf{r}_{n-1}), \\ \mathbf{a}_n &= \mathbf{a}(\mathbf{r}_n), \\ \mathbf{v}_n &= \mathbf{v}_{n-1/2} + \frac{h}{2}\mathbf{a}_n, \end{aligned} \tag{3.53}$$

where $h = \Delta t$. For the current problem, $\mathbf{a}(\mathbf{r}_n) = -\frac{\mu}{r_n^2}\mathbf{r}_n$.

The MATLAB snippet is given as

Störmer-Verlet Method for the two body problem.

```
for i=2:N+1
     alpha=mu/(x(i-1).^2 + y(i-1).^2).^(3/2);
     x(i)=x(i-1)+h*u(i-1)-h^2/2*alpha*x(i-1);
     y(i)=y(i-1)+h*v(i-1)-h^2/2*alpha*y(i-1);
     u(i)=u(i-1)-h/2*alpha*x(i-1);
     v(i)=v(i-1)-h/2*alpha*y(i-1);
     alpha=mu/(x(i).^2 + y(i).^2).^(3/2);
     u(i)=u(i)-h/2*alpha*x(i);
     v(i)=v(i)-h/2*alpha*y(i);
     t(i)=t(i-1)+h;
 end
```

The results using the velocity Verlet method are shown in Figure 3.10. For only 50,000 steps we have much better results for the conservation laws and the orbit appears stable.

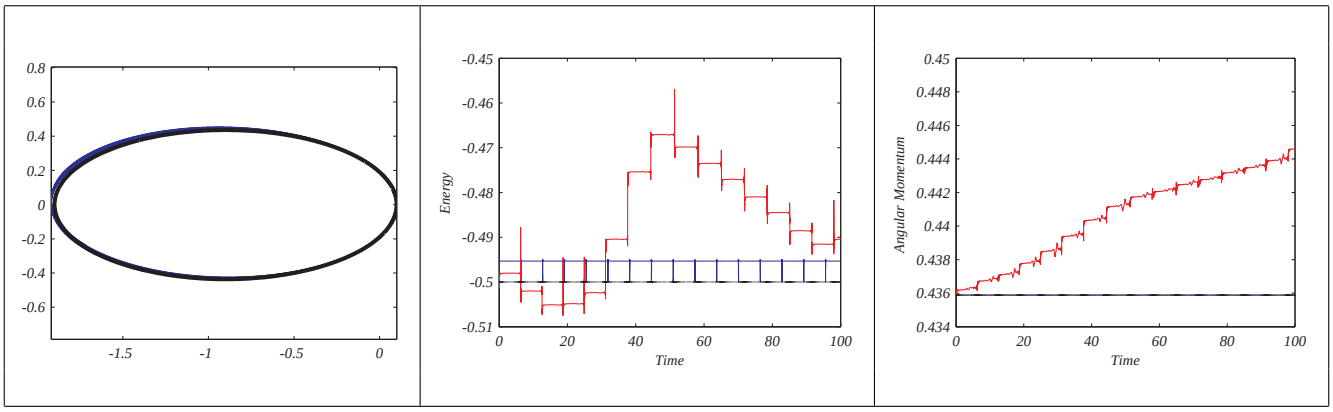


Table 3.10: Results for using velocity Verlet method for $N = 50000$ and $t \in [0, 100]$. The parameters are $\mu = 1$, $e = 0,9$, and $a = 1$.

3.5.6 *The Expanding Universe**

One of the remarkable stories of the twentieth century is the development of both the theory and the experimental data leading to our current understanding of the large scale structure of the universe. In 1916 Albert Einstein (1879-1955) published his general theory of relativity. It is a geometric theory of gravitation which relates the curvature of spacetime to

its energy and momentum content. This relationship is embodied in the Einstein field equations, which are written compactly as

$$G_{\mu\nu} + \Lambda g_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}.$$

The left side contains the curvature of spacetime as determined by the metric $g_{\mu\nu}$. The Einstein tensor, $G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu}$, is determined from the curvature tensor $R_{\mu\nu}$ and the scalar curvature, R. These in turn are obtained from the metric tensor. Λ is the famous cosmological constant, which Einstein originally introduced to maintain a static universe, which has since taken on a different role. The right-hand side of Einstein's equation involves the familiar gravitational constant, the speed of light, and the stress-energy tensor, $T_{\mu\nu}$.

Georges Lemaître (1894-1966) had actually predicted the expansion of the universe in 1927 and proposed what later became known as the big bang theory.

In 1917 Einstein applied general relativity to cosmology. However, it was Alexander Alexandrovich Friedmann (1888-1925) who was the first to provide solutions to Einstein's equation based on the assumptions of homogeneity and isotropy and leading to the expansion of the universe. Unfortunately, Friedmann died in 1925 of typhoid.

In 1929 Edwin Hubble (1889-1953) showed that the radial velocities of galaxies are proportional to their distance, resulting in what is now called Hubble's Law. Hubble's Law takes the form

$$v = H_0 r,$$

where H_0 is the Hubble constant and indicates that the universe is expanding. The current values of the Hubble constant are (70 ± 7) km s^{-1} Mpc $^{-1}$ and some recent WMAP results indicate it could be (71.0 ± 2.5) km s^{-1} Mpc $^{-1}$.[5]

[5] These strange units are in common usage. Mpc stands for 1 megaparsec $= 3.086 \times 10^{22}$ m and 1 km s^{-1} Mpc $^{-1}$ $= 3.24 \times 10^{-20}$ s^{-1}. The recent value was reported at the NASA website on March 25, 2013 http://map.gsfc.nasa.gov/universe/bb_tests_exp.html

In this section we are interested in Friedmann's Equation, which is the simple differential equation

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3c^2}\varepsilon(t) - \frac{\kappa c^2}{R_0^2} + \frac{\Lambda}{3}.$$

Here, $a(t)$ is the scale factor of the universe, which is taken to be one at present time; $\varepsilon(t)$ is the energy density; R_0 is the radius of curvature; and, κ is the curvature constant, ($\kappa = +1$ for positively curved space, $\kappa = 0$ for flat space, $\kappa = -1$ for negatively curved space.) The cosmological constant,Λ, is now added to account for dark energy. The idea is that if we know the right side of Friedmann's equation, then we can say something about the future size of the unverse. This is a simple differential equation which comes from applying Einstein's equation to an isotropic, homogenous, and curved spacetime. Einstein's equation actually gives us a little more than this equation, but we will only focus on the (first) Friedmann equation. The reader can read more in books on cosmology, such as B. Ryden's *Introduction to Cosmology.*

Friedmann's equation can be written in a simpler form by taking into account the different contributions to the energy density. For $\Lambda = 0$ and

zero curvature, one has

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3c^2}\varepsilon(t).$$

We define the Hubble parameter as $H(t) = \dot{a}/a$. At the current time, t_0, $H(t_0) = H_0$, Hubble's constant, and we take $a(t_0) = 1$. The energy density in this case is called the critical density,

$$\varepsilon_c(t) = \frac{3c^2}{8\pi G}H(t)^2.$$

It is typical to introduce the density parameter, $\Omega = varepsilon/\varepsilon_c$. Then, the Friedmann equation can be written as

$$1 - \Omega = -\frac{\kappa c^2}{R_0^2 a(t)^2 H(t)^2}.$$

Evaluating this expression at the current time, then

$$1 - \Omega_0 = -\frac{\kappa c^2}{R_0^2 H_0^2};$$

and, therefore,

$$1 - \Omega = -\frac{H_0^2(1-\Omega_0)}{a^2 H^2}.$$

Solving for H^2, we have the differential equation

$$\left(\frac{\dot{a}}{a}\right)^2 = H_0^2\left[\Omega(t) + \frac{1-\Omega_0}{a^2}\right],$$

where Ω takes into account the contributions to the energy density of the universe. These contributions are due to nonrelativistic matter density, contributions due to photons and neutrinos, and the cosmological constant, which might represent dark energy. This is discussed in Ryden (2003). In particular, Ω is a function of $a(t)$ for certain models. So, we write

$$\Omega = \frac{\Omega_{r,0}}{a^4} + \frac{\Omega_{m,0}}{a^3} + \Omega_{\Lambda,0},$$

where current estimates (Ryden (2003)) are $\Omega_{r,0} = 8.4 \times 10^{-5}$, $\Omega_{m,0} = 0.3$, $\Omega_{\Lambda,0} \approx 0.7$. In general, We require

$$\Omega_{r,0} + \Omega_{m,0} + \Omega_{\Lambda,0} = \Omega_0.$$

So, in later examples, we will take this relationship into account.

The compact form of Friedmann's equation.

Therefore, the Friedmann equation can be written as

$$\left(\frac{\dot{a}}{a}\right)^2 = H_0^2\left[\frac{\Omega_{r,0}}{a^4} + \frac{\Omega_{m,0}}{a^3} + \Omega_{\Lambda,0} + \frac{1-\Omega_0}{a^2}\right]. \tag{3.54}$$

Taking the square root of this expression, we obtain a first order equation for the scale factor,

$$\dot{a} = \pm H_0\sqrt{\frac{\Omega_{r,0}}{a^2} + \frac{\Omega_{m,0}}{a} + \Omega_{\Lambda,0}a^2 + 1 - \Omega_0}.$$

The appropriate sign will be used when the scale factor is increasing or decreasing.

For special universes, by restricting the contributions to Ω, one can get analytic solutions. But, in general one has to solve this equation numerically. We will leave most of these cases to the reader or for homework problems and will consider some simple examples.

Example 3.6. Determine $a(t)$ for a flat universe with nonrelativistic matter only. (This is called an Einstein-de Sitter universe.)

In this case, we have $\Omega_{r,0} = 0$, $\Omega_{\Lambda,0} = 0$, and $\Omega_0 = 1$. Since $\Omega_{r,0} + \Omega_{m,0} + \Omega_{\Lambda,0} = \Omega_0$, $\Omega_{m,0} = 1$ and the Friedman equation takes the form

$$\dot{a} = H_0\sqrt{\frac{1}{a}}.$$

This is a simple separable first order equation. Thus,

$$H_0\,dt = \sqrt{a}da.$$

Integrating, we have

$$H_0 t = \frac{2}{3}a^{3/2} + C.$$

Taking $a(0) = 0$, we have

$$a(t) = \left(\frac{t}{\frac{2}{3}H_0}\right)^{2/3}.$$

Since $a(t_0) = 1$, we find

$$t_0 = \frac{2}{3H_0}.$$

This would give the age of the universe in this model as roughly $t_0 = 9.3$ Gyr.

Example 3.7. Determine $a(t)$ for a curved universe with nonrelativistic matter only.

We will consider $\Omega_0 > 1$. In this case, the Friedman equation takes the form

$$\dot{a} = \pm H_0\sqrt{\frac{\Omega_0}{a} + (1 - \Omega_0)}.$$

Note that there is an extremum a_{max} which occurs for $\dot{a} = 0$. This occurs for

$$a = a_{max} \equiv \frac{\Omega_0}{\Omega_0 - 1}.$$

Analytic solutions are possible for this problem in parametric form. Note that we can write the differential equation in the form

$$\begin{aligned}
\dot{a} &= \pm H_0\sqrt{\frac{\Omega_0}{a} + (1 - \Omega_0)} \\
&= \pm H_0\sqrt{\frac{\Omega_0}{a}}\sqrt{1 + \frac{a(1 - \Omega_0)}{\Omega_0}} \\
&= \pm H_0\sqrt{\frac{\Omega_0}{a}}\sqrt{1 - \frac{a}{a_{max}}}.
\end{aligned} \tag{3.55}$$

A separation of variables gives

$$H_0\sqrt{\Omega_0}\,dt = \pm\frac{\sqrt{a}}{\sqrt{1-\frac{a}{a_{max}}}}\,da.$$

This form suggests a trigonometric substitution,

$$\frac{a}{a_{max}} = \sin^2\theta$$

with $da = 2a_{max}\sin\theta\cos\theta\,d\theta$. Thus, the integration becomes

$$H_0\sqrt{\Omega_0}t = \pm\int\frac{\sqrt{a_{max}\sin^2\theta}}{\sqrt{\cos^2\theta}}\,2a_{max}\sin\theta\cos\theta\,d\theta.$$

In proceeding, we should be careful. Recall that for real numbers $\sqrt{x^2} = |x|$. In order to remove the roots of squares we need to consider the quadrant θ is in. Since $a = 0$ at $t = 0$, and it will vanish again for $\theta = \pi$, we will assume $0 \le \theta \le \pi$. For this range, $\sin\theta \ge 0$. However, $\cos\theta$ is not of one sign for this domain. In fact, a reaches it maximum at $\theta = \pi/2$. So, $\dot{a} > 0$. This corresponds to the upper sign in front of the integral. For $\theta > \pi/2$, $\dot{a} < 0$ and thus we need the lower sign and $\sqrt{\cos^2\theta} = -\cos\theta$ for that part of the domain. Thus, it is safe to simplify the square roots and we obtain

$$\begin{aligned} H_0\sqrt{\Omega_0}t &= 2a_{max}^{3/2}\int\sin^2\theta,d\theta. \\ &= a_{max}^{3/2}\int(1-\cos 2\theta)\,,d\theta. \\ &= a_{max}^{3/2}\left(\theta-\frac{1}{2}\sin 2\theta\right) \end{aligned} \tag{3.56}$$

for $t = 0$ at $\theta = 0$.

We have arrived at a parametric solution to the example,

$$\begin{aligned} a &= = a_{max}\sin^2\theta, \\ t &= \frac{a_{max}^{3/2}}{H_0\sqrt{\Omega_0}}\left(\theta-\frac{1}{2}\sin 2\theta\right), \end{aligned} \tag{3.57}$$

for $0 \le \theta \le \pi$. Letting, $\phi = 2\theta$, this solution can be written as

$$\begin{aligned} a &= \frac{1}{2}a_{max}(1-\cos\phi), \\ t &= \frac{a_{max}^{3/2}}{2H_0\sqrt{\Omega_0}}\left(\phi-\sin\phi\right), \end{aligned} \tag{3.58}$$

for $0 \le \phi \le 2\pi$. As we will see in Chapter 10, the curve described by these equations is a cycloid.

A similar computation can be performed for $\Omega_0 < 1$. This will be left as a homework exercise. The answer takes the form

$$\begin{aligned} a &= \frac{\Omega_0}{2(1-\Omega_0)}(\cosh\eta-1), \\ t &= \frac{\Omega_0}{2H_0(1-\Omega)^{3/2}}\left(\sinh\eta-\eta\right), \end{aligned} \tag{3.59}$$

for $\eta \ge 0$.

Example 3.8. Determine the numerical solution of Friedmann's equation for a curved universe with nonrelativistic matter only.

Since Friedmann's equation is a differential equation, we can use our favorite solver to obtain a solution. Not all universe types are amenable to obtaining an analytic solution as the last example. We can create a function in MATLAB for use in **ode45**:

```
function da=cosmosf(t,a)
global Omega
f=Omega./a+1-Omega;
da=sqrt(f);
end
```

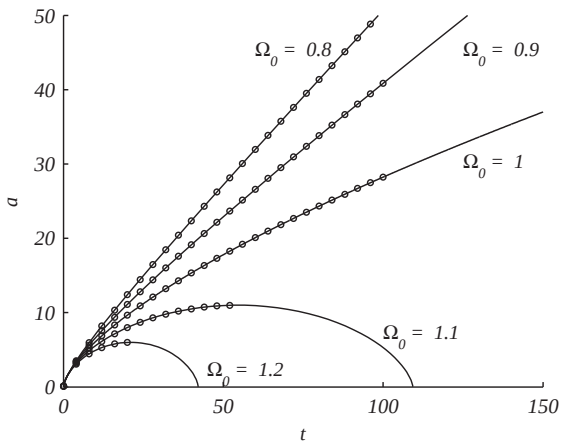


Figure 3.39: Numerical solution (circles) of the Friedmann equation superimposed on the analytic solutions for a matter plus curvature ($\Omega_0 \neq 1$) or no curvature ($\Omega_0 = 1$) universe.

We can then solve the Friedmann equation and compare the solutions to the analytic forms in the last two examples. The code for doing this is given below:

```
clear
global Omega

for Omega=0.8:.1:1.2;
    if Omega<1
        amax=50;
        tmax=100;
    elseif Omega==1
        amax=50;
        tmax=100;
    else
        amax=Omega/(Omega-1);
        tmax=Omega/(Omega-1)^1.5/2*pi;
    end

    tspan=0:4:tmax;
```

```
        a0=.1;
        [t,a]=ode45(@cosmosf,tspan,a0);
        plot(t,a,'ok')
        hold on

        if Omega<1
            eta=0:.1:4;
            a3 = Omega/(1-Omega)/2*(cosh(eta)-1);
            t3 = Omega/(1-Omega)^1.5/2*(sinh(eta)-eta);
            plot(t3,a3,'k')
            axis([0,max(t3),0,max(a3)])
            xlabel('t')
            ylabel('a')
        elseif Omega==1
            t3=0:.1:1.5*tmax;
            a3=(3*t3/2).^(2/3);
            plot(t3,a3,'k')
        else
            phi=0:.1:2*pi;
            a3 = Omega/(Omega-1)/2*(1-cos(phi));
            t3 = Omega/(Omega-1)^1.5/2*(phi-sin(phi));
            plot(t3,a3,'k')
        end
    end
    hold off
    axis([0,150,0,50])
    xlabel('t')
    ylabel('a')
```

In Figure 3.39 we show the results. For $\Omega_0 > 1$ the solutions lie on the first half of the cycloid solution. The other solutions indicate that the universe continues to expand, leading to what is called the Big Chill. The analytic solutions to the $\Omega_0 > 1$ cases eventually collapse to $a = 0$ in finite time. These final states are what Stephen Hawking calls the Big Crunch.

The numerical solutions for $\Omega_0 > 1$ run into difficulty because the radicand in the square root is negative. But, this corresponds to when $\dot{a} < 0$. So, we have to modify the code by estimating the maximum on the curve and run the numerical algorithm with new initial conditions and using the fact that $\dot{a} < 0$ in the function **cosmosf** by setting da=-sqrt(f). The modified code is below and the resulting numerical solutions are shown in Figure 3.40.

```
tspan=0:4:tmax;
    a0=.1;
    [t,a]=ode45(@cosmosf,tspan,a0);
    plot(t,a,'ok','MarkerSize',2)
```

```
hold on
if Omega>1
     tspan=tmax+.0001:4:2*tmax;
     a0=amax-.0001;
     [t2,a2]=ode45(@cosmosf2,tspan,a0);
     plot(t2,a2,'ok','MarkerSize',2)
end
```

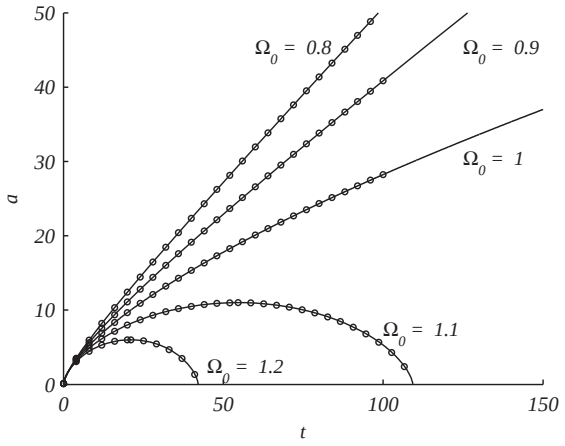


Figure 3.40: Modified numerical solution (circles) of the Friedmann equation superimposed on the analytic solutions for a matter plus curvature ($\Omega_0 \neq 1$) or no curvature ($\Omega_0 = 1$) universe with the extension past the maximum value of a when $\Omega_0 > 1$..

3.5.7 *The Coefficient of Drag**

We have seen that air drag can play a role in interesting physics problems in differential equations. This also is an important concept in fluid flow and fluid mechanics when looking at flows around obstacles, or when the obstacle is moving with respect to the background fluid. The simplest such object is a sphere, such as a baseball, soccer ball, golf ball, or ideal spherical raindrop. The resistive force is characterized by the dimensionless drag coefficient

$$C_D = \frac{F_D}{\frac{1}{2}\rho U^2 L^2},$$

where L and U are the characteristic length and speed of the object moving through the fluid.

The Reynolds number, *Re*, is named after Osborne Reynolds (1842-1912) who first determined it in 1883.

There has been much attention focussed on relating the drag coefficient to the Reynolds number. The Reynolds number is given by

$$Re = \frac{\rho L U}{\eta} = \frac{LV}{\nu},$$

where η is the viscosity and $\nu = \frac{\eta}{\rho}$ is the kinematic viscosity. It is a measure of the size of the kinematic to viscous forces in the problem at hand. There are different ranges of fluid behavior depending on the order of the Reynolds number. These range from laminar flow ($Re < 1000$) to turbulent

flow ($Re > 2.5 \times 10^5$). There are a range of other types of flows such as creeping flow ($Re << 1$) and transitional flows, which are a mix of laminar and turbulent flow.

For low Reynolds number, the inertial forces are small compared to the viscous forces, leading to the Stokes drag force, $C_D = 24Re^{-1}$. This result can be determined analytically. Similarly, for large Reynolds number the drag coefficient is a constant. This is the Newtonian regime. Somewhere in between the form of the drag coefficient is found through empirical studies. There have been many empirical expressions developed and all are within a few percent of the data in the range of applicability. Some of the commonly used expressions are given below.

Models that are useful for $Re < 10^3$:

$$C_2 = \frac{24}{Re} + \frac{4}{Re^{1/3}}, \quad \text{Putnam (1961),} \tag{3.60}$$

$$C_3 = \frac{24}{Re}\left(1 + 0.15Re^{0.687}\right), \quad \text{Schiller-Naumann (1933),} \tag{3.61}$$

$$C_4 = 12Re^{-.5}; \quad \text{Edwards et al. (2000),} \tag{3.62}$$

$$\tag{3.63}$$

Models that are useful for $Re < 2 \times 10^5$ are the White (1991) and Clift-Gavin (1970), respectively,

$$C_1 = \frac{24}{Re} + \frac{6.}{1+\sqrt{Re}} + 0.4, \tag{3.64}$$

$$C_5 = \frac{24}{Re}\left(1 + 0.15Re^{0.687}\right) + \frac{.42}{1 + 42500/Re^{1.16}}. \tag{3.65}$$

A more recent model was proposed by Morrison (2010) for $Re < 10^6$:

$$C_6 = \frac{24}{Re} + \frac{2.6\left(\frac{Re}{5.0}\right)}{1+\left(\frac{Re}{5.0}\right)^{1.52}} + \frac{.411\left(\frac{Re}{263000}\right)^{-7.94}}{1+\left(\frac{Re}{263000}\right)^{-8.00}} + \frac{Re^{0.80}}{461000}. \tag{3.66}$$

Plots for these models are shown in Figures 3.41-3.42. In Figure 3.41 we see that the models differ significantly for large Reynolds numbers.

Figure 3.42 shows a log-log plot of the drag coefficient as a function of Reynolds number. In Figure 3.43 we show a power law fit for Reynolds number less than 1000 confirming the model used by Edwards, Wilder, and Scime (2001) as described in the raindrop problem.

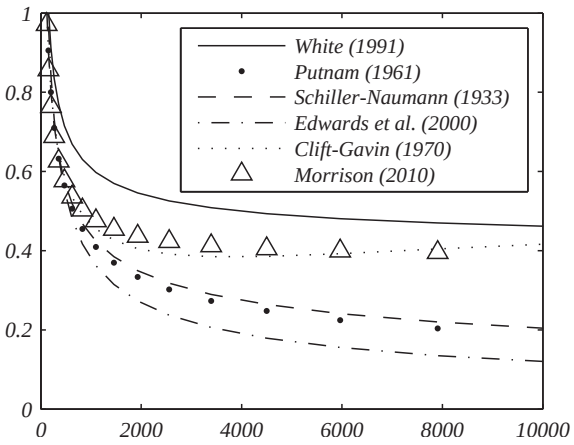


Figure 3.41: Drag coefficient as a function of Reynolds number for spheres.

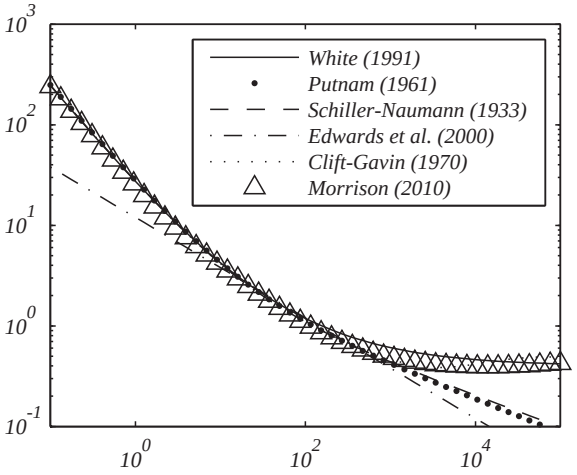


Figure 3.42: Log-log plot of the drag coefficient as a function of Reynolds number for spheres.

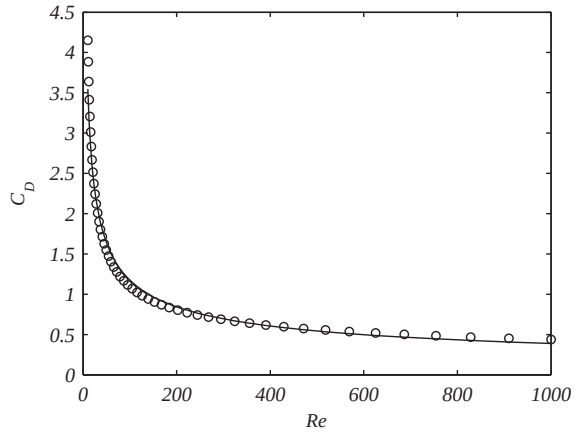


Figure 3.43: A power law fit for the drag coefficient as a function of Reynolds number using linearization and linear regression.

Problems

1. Use Euler's Method to determine the given value of y for the following problems. When possible compare the numerical approximations with the exact solutions.

a. $\frac{dy}{dx} = 2y$, $y(0) = 2$. Find $y(1)$ with $h = 0.1$.

b. $\frac{dy}{dx} = x - y$, $y(0) = 1$. Find $y(2)$ with $h = 0.2$.

c. $\frac{dy}{dx} = x\sqrt{1 - y^2}$, $y(1) = 0$. Find $y(2)$ with $h = 0.2$.

d. $\frac{dy}{dt} = 1 + \frac{y}{t}$, $y(1) = 2$ with $h = 0.25$.

e. $\frac{dy}{dt} = -3y + te^{2t}$, $y(0) = 0$ with $h = 0.25$.

2. Use the Midpoint Method to solve the initial value problems in Problem 1.

3. Numerically solve the nonlinear pendulum problem using the Euler-Cromer code for a pendulum with length $L = 0.5$ m using initial angles of $\theta_0 = 10^o$, and $\theta_0 = 70^o$. In each case run the routines long enough and with an appropriate h such that you can determine the period in each case. Compare your results with the linear pendulum period.

4. For the Baumgartner sky dive we had obtained the results for his position as a function of time. There are other questions which could be asked.

a. Find the velocity as a function of time for the model developed in the text.

b. Find the velocity as a function of altitude for the model developed in the text.

c. What maximum velocity is obtained in the model? At what time and position?

d. Does the model indicate that terminal velocity was reached?

e. What speed is predicted for the point at which the parachute opened?

f. How do these numbers compare with reported data?

5. Consider the flight of a golf ball with mass 46 g and a diameter of 42.7 mm. Assume it is projected at 30^o with a speed of 36 m/s and no spin.

a. Ignoring air resistance, analytically find the path of the ball and determine the range, maximum height, and time of flight for it to land at the height that the ball had started.

b. Now consider a drag force $f_D = \frac{1}{2}C_D\rho\pi r^2 v^2$, with $C_D = 0.42$ and $\rho = 1.21$ kg/m^3. Determine the range, maximum height, and time of flight for the ball to land at the height that it had started.

c. Plot the Reynolds number as a function of time. [Take the kinematic viscosity of air, $\nu = 1.47 \times 10^{-5}$.

d. Based on the plot in part c, create a model to incorporate the change in Reynolds number and repeat part b. Compare the results from parts a, b and d.

6. Consider the flight of a tennis ball with mass 57 g and a diameter of 66.0 mm. Assume the ball is served 6.40 meters from the net at a speed of 50.0 m/s down the center line from a height of 2.8 m. It needs to just clear the net (0.914 m).

a. Ignoring air resistance and spin, analytically find the path of the ball assuming it just clears the net. Determine the angle to clear the net and the time of flight.

b. Find the angle to clear the net assuming the tennis ball is given a topspin with $\omega = 50$ rad/s.

c. Repeat part b assuming the tennis ball is given a bottom spin with $\omega = 50$ rad/s.

d. Repeat parts a, b, and c with a drag force, taking $C_D = 0.55$.

7. In Example 3.7 $a(t)$ was determined for a curved universe with nonrelativistic matter for $\Omega_0 > 1$. Derive the parametric equations for $\Omega_0 < 1$,

$$\begin{aligned} a &= \frac{\Omega_0}{2(1-\Omega_0)}(\cosh\eta - 1), \\ t &= \frac{\Omega_0}{2H_0(1-\Omega)^{3/2}}\left(\sinh\eta - \eta\right), \end{aligned} \tag{3.67}$$

for $\eta \geq 0$.

8. Find numerical solutions for other models of the universe.

a. A flat universe with nonrelativistic matter only with $\Omega_{m,0} = 1$.

b. A curved universe with radiation only with curvature of different types.

c. A flat universe with nonrelativistic matter and radiation with several values of $\Omega_{m,0}$ and $\Omega_{r,0} + \Omega_{m,0} = 1$.

d. Look up the current values of $\Omega_{r,0}$, $\Omega_{m,0}$, $\Omega_{\Lambda,0}$, and κ. Use these values to predict future values of $a(t)$.

e. Investigate other types of universes of your choice, but different from the previous problems and examples.