# Chapter 9
# Differential Equations

## 9.1 Introduction

A differential equation is an equation involving an unknown function, $y(t)$, and its derivatives. Many differential equations are solved in calculus. For example, if $f(t)$ is continuous on an interval $[\,t_0,\ t_f\,]$, the solution of the equation

$$\frac{dy}{dt} = f(t) \tag{9.1}$$

is given by the fundamental theorem of calculus

$$y(t) = C + \int_{t_0}^{t} f(\tau)d\tau. \tag{9.2}$$

Equation (9.2) describes a family of solutions determined by the constant $C$. A particular solution is computed by requiring that the solution pass through the point $(t_0, y_0)$. In other words

$$y(t_0) = y_0. \tag{9.3}$$

The problem specified by (9.1) and (9.3) is called an *initial value problem* (IVP) and has the solution

$$y(t) = y_0 + \int_{t_0}^{t} f(\tau)d\tau.$$

A general *first order* initial value problem consists of a differential equation and an initial value as follows:

$$y'(t) = f(t, y), \qquad y(t_0) = y_0. \tag{9.4}$$

We assume that the known function $f(t, y)$ is continuous in $t$ and $y$. Introductory textbooks on differential equations provide analytical solution methods for many first order initial value problems along with various theoretical aspects. The solution methods are based on special cases for $f(t, y)$. The following example is called a *separable* initial value problem since $f(t, y)$ $= -2ty + y/t$ may be expressed as a product of a function of $t$ and a function of $y$.

$$y' = (-2t + 1/t)y,\ y(0.25) = 0.6 \tag{9.5}$$

By direct substitution you should be able to verify that $y(t) = 2.4te^{-t^2+0.0625}$ is a solution of the IVP given in (9.5). If you are familiar with differential equations, you will recognize that (9.5) is also a *linear* initial value problem. There are special solution methods for linear problems. Unfortunately, many initial value problems cannot be solved by analytical techniques. This leads to the need for numerical methods.

This chapter will focus on some elementary methods for the numerical solution of initial value problems. This means that we wish to compute a set of points $(t_i, \widehat{y}_i)$, $i = 0, 1, 2, \ldots, n$, to approximate the true solution, $y(t)$. The notation $\widehat{y}_i \approx y(t_i)$ will be used. In many cases we assume that $t_i = t_0 + i \cdot h$ where $h$ is a constant called the *step size*. Obviously, the first point is $(t_0, y_0)$. Note also, the initial slope of the true solution is given by the known value $y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0)$.

To begin our study, consider the linear Taylor polynomial plus remainder centered at $t_0$

$$y(t) = y(t_0) + y'(t_0)(t - t_0) + \frac{1}{2}y''(c_0)(t - t_0)^2$$

or

$$y(t) = y_0 + f(t_0, y_0)(t - t_0) + \frac{1}{2}y''(c_0)(t - t_0)^2 \tag{9.6}$$

where $c_0$ is an unknown between $t$ and $t_0$. With $t = t_1 = t_0 + h$, (9.6) becomes

$$y(t_1) = y_0 + hf(t_0, y_0) + \frac{1}{2}y''(c_0)h^2 \tag{9.7}$$

The first two terms on the right hand side of (9.7) lead to a simple numerical method to approximate $y(t_1)$. We find $y(t_1) \approx \widehat{y}_1 = y_0 + hf(t_0, y_0)$. Since we have neglected $\frac{1}{2}y''(c_0)h^2$ to form the approximation, the remaining term on the right hand side of (9.7) is called the *local truncation error*. We say that the local truncation error is $\mathcal{O}(h^2)$.

## 9.2 Euler's Method

The approximation given at the end of the last section, $y(t_1) \approx \widehat{y}_1 = y_0 + hf(t_0, y_0)$, may be viewed as a tangent line approximation. From calculus, a tangent line requires a point, $(t_0, y_0)$, and a slope, $y'(t_0) = f(t_0, y_0)$; thus, the point-slope equation of the line is

$$\begin{aligned} y - y_0 &= f(t_0, y_0)(t - t_0) \\ y &= y_0 + f(t_0, y_0)(t - t_0). \end{aligned}$$

With $t$ replaced by $t_1 = t_0 + h$, the value for $\widehat{y}_1$ is given by $y_0 + hf(t_0, y_0)$.

It is tempting to use a second linear Taylor polynomial plus remainder, centered at $t_1$, to compute the next step. Consider

$$y(t) = y(t_1) + y'(t_1)(t - t_1) + \frac{1}{2}y''(c_1)(t - t_1)^2$$

or

$$y(t) = y(t_1) + f(t_1, y(t_1))(t - t_1) + \frac{1}{2}y''(c_1)(t - t_1)^2. \tag{9.8}$$

With $t = t_2 = t_1 + h$, (9.8) becomes

$$y(t_2) = y(t_1) + f(t_1, y(t_1))h + \frac{1}{2}y''(c_1)h^2. \tag{9.9}$$

The computational problem should be obvious: an exact value for $y(t_1)$ is not available. Using the approximate value, $\widehat{y}_1$, the first two terms on the right hand side of (9.9) lead to a formula estimating $y(t_2)$.

$$\widehat{y}_2 = \widehat{y}_1 + hf(t_1, \widehat{y}_1) \tag{9.10}$$

Generalizing equation (9.10) leads to *Euler's* method

$$\widehat{y}_{i+1} = \widehat{y}_i + hf(t_i, \widehat{y}_i), \text{ for } i = 0, 1, 2, \ldots n. \tag{9.11}$$

Note that $\widehat{y}_0 = y_0$. The first three iterations of Euler's method are

$$\begin{aligned} \widehat{y}_1 &= y_0 + hf(t_0, y_0) \\ \widehat{y}_2 &= \widehat{y}_1 + hf(t_1, \widehat{y}_1) \\ \widehat{y}_3 &= \widehat{y}_2 + hf(t_2, \widehat{y}_2) \end{aligned}$$

To fully understand Euler's method, let's write out the details of the first three iterations using the initial value problem $y' = (-2t + 1/t)y$, $y(0.25) = 0.6$. With $h = 0.1$:

$$\begin{aligned} \widehat{y}_1 &= y_0 + hf(t_0, y_0) \\ &= y_0 + h(-2t_0 + 1/t_0)y_0 \\ &= 0.6 + 0.1 \cdot (-2 \cdot 0.25 + 1/0.25) \cdot 0.6 = 0.81 \\ \\ \widehat{y}_2 &= \widehat{y}_1 + hf(t_1, \widehat{y}_1) \\ &= \widehat{y}_1 + h(-2t_1 + 1/t_1)\widehat{y}_1 \\ &= 0.81 + 0.1 \cdot (-2 \cdot 0.35 + 1/0.35) \cdot 0.81 = 0.985 \\ \\ \widehat{y}_3 &= \widehat{y}_2 + hf(t_2, \widehat{y}_2) \\ &= \widehat{y}_2 + h(-2t_2 + 1/t_2)\widehat{y}_2 \\ &= 0.985 + 0.1 \cdot (-2 \cdot 0.45 + 1/0.45) \cdot 0.985 = 1.115 \end{aligned} \tag{9.12}$$

Keep in mind that $\widehat{y}_1 \approx y(0.35)$, $\widehat{y}_2 \approx y(0.45)$ and $\widehat{y}_3 \approx y(0.55)$. Figure 9.1 shows the values computed with Euler's method and a graph of the true solution. The axes setting in the figure have been modified from MATLAB's default choices to provide a consistent window. As you can see, the data points from Euler's method are above the true solution. Why is this the case?

An M-file to compute, display and plot the results of Euler's method for the example initial value problem is given on the next page. The M-file ydot.m contains $y' = f(t, y)$ $= -2ty + y/t$. There are two new MATLAB commands that should be reviewed with the help command, **pause** and **hold on**. For graphical purposes, the exact solution is in the M-file fct1.m.

M-file ydot.m

```
function yp=ydot(t,y)
%YDOT ydot = f(t,y) for initial value problems
yp=-2*t*.y+y/.t;
```

M-file euler.m

```
function [t,y] = euler(t0,y0,h,n)
% EULER Euler's method
% Requires ydot.m and fct1.m
% initial values,to, yo; step size = h; and number of steps = n
t(1) = t0;y(1)=y0;
t = t0:h:t0+h*n;
for i = 1:n
   y(i+1) = y(i)+h*ydot(t(i),y(i));
end
% display results and pause
disp('      t                    y')
disp([t',y']),pause
% plot results and pause
plot(t,y,'k*'),xlabel('t'),ylabel('y')
hold on,pause
% plot exact solution
x=t0:h/10:t0+h*n;
z = fct1(x);
plot(x,z,'k')
```

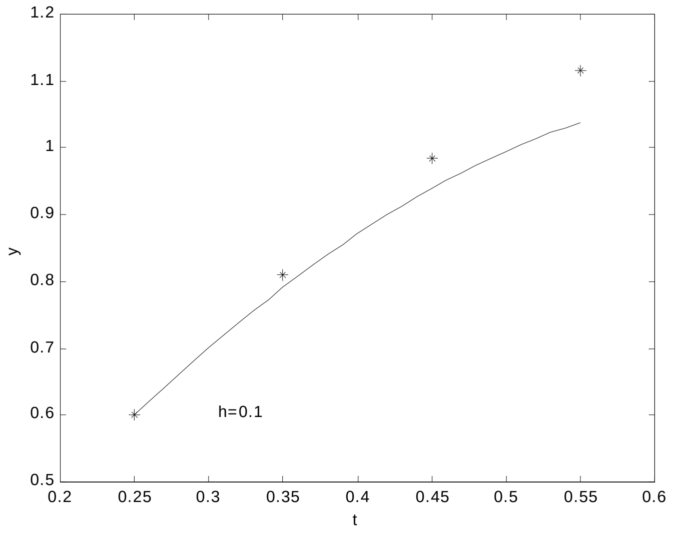

Figure 9.1 Euler's Method, $h = 0.1,$ and $y(t) = 2.4te^{-t^2+0.0625}$

Numerical results will confirm the values computed in (9.12).

```
>> [t,y] = euler(.25,.6,.1,3);
      t                 y
  2.5000e-001  6.0000e-001
  3.5000e-001  8.1000e-001
```

```
4.5000e-001  9.8473e-001
5.5000e-001  1.1149e+000
```

Decreasing the step size will improve the accuracy of Euler's method. The following table summarizes the error at $t = 0.55$ for three different step sizes. Note $y(0.55) = 1.0383$.

| Step Size | Iterations | Approximation | Global Error |
|---|---|---|---|
| $h$ | $i$ | $\widehat{y}_i$ | $\mid y(0.55) - \widehat{y}_i \mid$ |
| $0.1$ | $3$ | $1.1149$ | $0.0766$ |
| $0.05$ | $6$ | $1.0783$ | $0.0400$ |
| $0.025$ | $12$ | $1.0587$ | $0.0204$ |

The errors computed in the table reflect the accumulation of error in each of the previous iterations. The data suggest, imperfectly, that decreasing the step size by half also decreases the error by half. In other words it appears that the error is $\mathcal{O}(h)$. The *global error* at $t_i$ is defined as $|\, y(t_i) - \widehat{y}_i \,|$. Advanced texts on numerical analysis contain proofs showing that the global error of Euler's method is $\mathcal{O}(h)$, neglecting roundoff errors.

Figure 9.2 shows the result of a smaller step size for our example initial value problem.

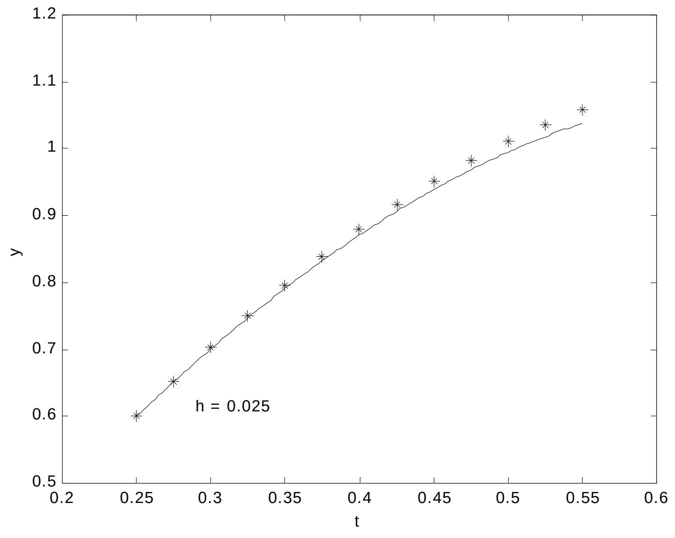

Figure 9.2 Euler's Method, $h = 0.025,$ and $y(t) = 2.4te^{-t^2+0.0625}$

Note that the Euler values remain above the true solution for the interval shown. Since the global error is $\mathcal{O}(h)$, Euler's method is often called a first order numerical method. Figures 9.1 and 9.2 suggest that the maximum of the global error occurs at the final time, $t = 0.55$. As we shall see, this is not always the case. Although Euler's method is simple to derive and easy to analyze, the global error performance is not satisfactory for practical use in the numerical solution of initial value problems. In other words, numerical schemes with better error characteristics are the methods of choice.

## 9.3 The Improved Euler Method

Euler's method is an example of a *one-step* method. In a one-step method only information on the interval $[\,t_{i-1}, t_i\,]$ is used in the computation of $\widehat{y}_i$. Various one-step methods may be derived by integrating $y' = f(t, y)$ from $t_i$ to $t_{i+1}$ as follows:

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f(t, y(t))\,dt$$

or

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t))\,dt \tag{9.13}$$

Since $y(t)$ is unknown, the integral in (9.13) cannot be evaluated; however, using integration techniques discussed in Chapter 8, two one-step methods will be developed in this section. For example, if the integrand $f(t, y(t))$ is replaced with the constant $f(t_i, \widehat{y}_i)$, (9.13) becomes

$$y(t_{i+1}) \approx y(t_i) + \int_{t_i}^{t_{i+1}} f(t_i, \widehat{y}_i)dt$$

or

$$y(t_{i+1}) \approx y(t_i) + hf(t_i, \widehat{y}_i). \tag{9.14}$$

Equation (9.14) leads directly to Euler's method, $\widehat{y}_{i+1} = \widehat{y}_i + hf(t_i, \widehat{y}_i)$.

Another approach for estimating the integral in (9.13) employs a linear interpolating polynomial using the points $(t_i, \widehat{y}_i)$ and $(t_{i+1}, \widehat{y}_{i+1})$. Actually, we are using the Trapezoidal Rule to arrive at

$$y(t_{i+1}) \approx y(t_i) + \frac{h}{2}\Big[f(t_i, \widehat{y}_i) + f(t_{i+1}, \widehat{y}_{i+1})\Big].$$

This suggests the formula

$$\widehat{y}_{i+1} = \widehat{y}_i + \frac{h}{2}\Big[f(t_i, \widehat{y}_i) + f(t_{i+1}, \widehat{y}_{i+1})\Big]. \tag{9.15}$$

Unfortunately, the unknown value $\widehat{y}_{i+1}$ now appears on both sides of (9.15) since it appears as an argument of $f$ on the right hand side. Instead of using a root finding method (recall Chapter 4), to find $\widehat{y}_{i+1}$, the so-called *improved Euler* method is obtained from (9.15) by replacing the right hand $\widehat{y}_{i+1}$ with a value from Euler's method, $\widehat{y}_i + hf(t_i, \widehat{y}_i)$. Using this preliminary estimate of $\widehat{y}_{i+1}$we find

$$\widehat{y}_{i+1} = \widehat{y}_i + \frac{h}{2}\Big[f(t_i, \widehat{y}_i) + f(t_{i+1},\ \widehat{y}_i + hf(t_i, \widehat{y}_i))\Big]. \tag{9.16}$$

The complicated structure of (9.16) can be simplified by introducing additional notation. Define $k_1$and $k_2$ as follows:

$$\begin{aligned} k_1 &= f(t_i, \widehat{y}_i) \\ k_2 &= f(t_i + h,\ \widehat{y}_i + hk_1) \end{aligned}$$

Using these definitions, the improved Euler method becomes

$$\widehat{y}_{i+1} = \widehat{y}_i + \frac{h}{2}\Big[k_1 + k_2\Big]. \tag{9.17}$$

Note that two function evaluations are needed to compute $\widehat{y}_{i+1}$ and that $k_1$ must be computed before $k_2$. On a positive note, it can be shown that the global error of the improved Euler method is $\mathcal{O}(h^2)$.

Details of the computations for the first two iterations of the improved Euler method with $h = 0.1$ are provided below. You will note the similarity to some of the computations for Euler's method found in equations (9.12).

$$\begin{aligned} k_1 &= f(t_0, y_0) \\ &= (-2t_0 + 1/t_0)y_0 \\ &= (-2 \cdot 0.25 + 1/0.25) \cdot 0.600 = 2.10 \\[1em] k_2 &= f(t_0 + h, y_0 + hk_1) \\ &= (-2(t_0 + h) + 1/(t_0 + h))(y_0 + hk_1) \\ &= (-2 \cdot 0.35 + 1/0.35) \cdot (0.600 + 0.210) = 1.747 \\[1em] \widehat{y}_1 &= y_0 + \tfrac{h}{2}(k_1 + k_2) \\ &= 0.6 + \tfrac{0.1}{2}(2.10 + 1.747) = 0.792 \end{aligned} \tag{9.18}$$

$$\begin{aligned} k_1 &= f(t_1, \widehat{y}_1) \\ &= (-2t_1 + 1/t_1)\widehat{y}_1 \\ &= (-2 \cdot 0.35 + 1/0.35) \cdot 0.792 = 1.71 \\[1em] k_2 &= f(t_1 + h, \widehat{y}_1 + hk_1) \\ &= (-2(t_1 + h) + 1/(t_1 + h))(\widehat{y}_1 + hk_1) \\ &= (-2 \cdot 0.45 + 1/0.45) \cdot (0.792 + 0.171) = 1.27 \\[1em] \widehat{y}_2 &= \widehat{y}_1 + \tfrac{h}{2}(k_1 + k_2) \\ &= 0.792 + \tfrac{0.1}{2}(1.71 + 1.27) = 0.941 \end{aligned} \tag{9.19}$$

The approximate values for $y(0.35)$ and $y(0.45)$, found in (9.18) and (9.19), are slightly lower than the corresponding values computed with Euler's method, (9.12), and closer to the actual values of the solution.

The following M-file, implementing the improved Euler method, is very similar to euler.m found in Section 9.2.

M-file impeuler.m

```
function [t,y] = impeuler(t0,y0,h,n)
% IMPEULER Improved Euler method
% requires ydot.m and fct1.m
% initial values,to, yo; step size = h; and number of steps = n
t(1) = t0;y(1)=y0;
t = t0:h:t0+h*n;
for i = 1:n
  k1 = ydot(t(i),   y(i)     );
  k2 = ydot(t(i)+h, y(i)+h*k1);
  y(i+1) = y(i)+h*(k1+k2)/2;
end
% display results and pause
disp('       t                    y')
disp([t',y']),pause
% plot results and pause
plot(t,y,'k*'),xlabel('t'),ylabel('y')
hold on,pause
% plot exact solution
x=t0:h/10:t0+h*n;
z = fct1(x);
plot(x,z,'k')
```

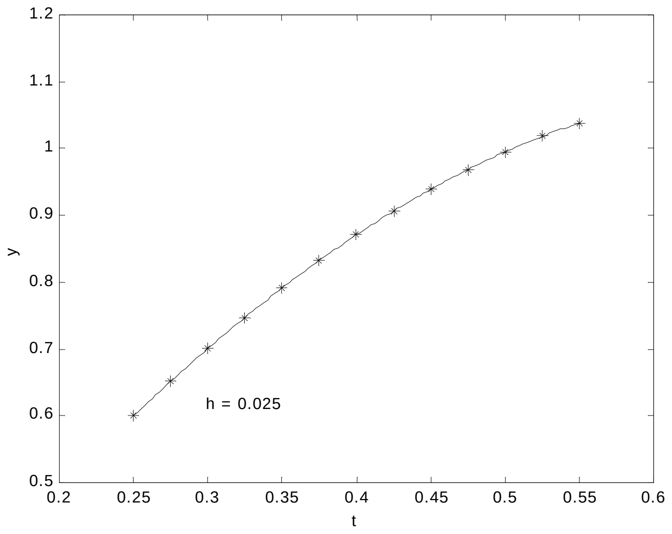

Figure 9.3 Improved Euler Method, $h = 0.025,$ and $y(t) = 2.4te^{-t^2+0.0625}$

Figure 9.3 shows the graphical results of >> **[t,y]=impeuler(.25,.6,.025,12);** . Comparing Figures 9.2 and 9.3 we note the improved global error.

As with Euler's method, decreasing the step size will improve the accuracy of the improved method. The following table summarizes the global error at $t = 0.55$ for four step sizes. Recall $y(0.55) = 1.0383$.

| Step Size $h$ | Iterations $i$ | Approximation $\widehat{y}_i$ | Global Error $\lvert y(0.55) - \widehat{y}_i \rvert$ |
|---|---|---|---|
| $0.1$ | $3$ | $1.0420$ | $0.0037$ |
| $0.05$ | $6$ | $1.0395$ | $0.0012$ |
| $0.025$ | $12$ | $1.0387$ | $0.0004$ |
| $0.0125$ | $24$ | $1.0384$ | $0.0001$ |

With a global error of $\mathcal{O}(h^2)$ we expect the error to drop by a factor of approximately four when the step size is cut in half. Our results seem close to this expected behavior as $h$ becomes smaller. Figure 9.4 shows both Euler methods for our sample initial value problem over a longer time interval, $[\,0.25, 2.25\,]$.

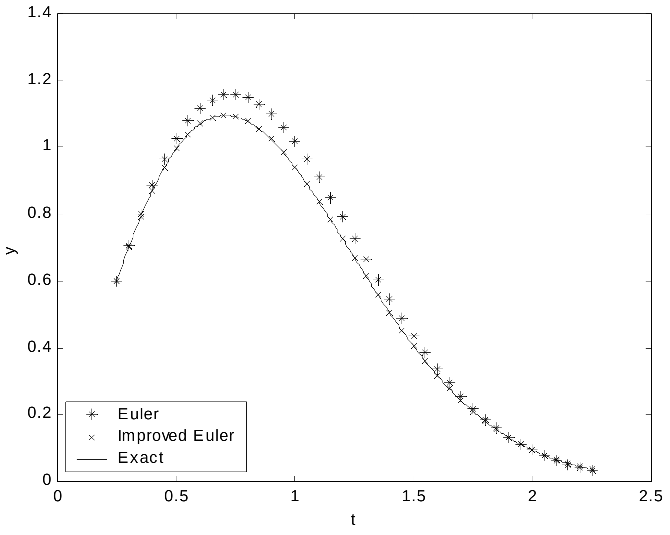

Figure 9.4 Euler Methods, $h\ =\ 0.05$

Based on this figure, the accuracy of a higher order numerical method is well worth the increase in computational effort. The data from >> **[t,y] = euler(.25,.6,.05,40);** is contained in the vectors **t** and **y**; thus, the following commands will locate the maximum of the global error.

```
>> yexact = fct1(t);
>> err = abs(yexact-y);
>> [maxerr,n] = max(err)
maxerr =
  7.6881e-002
n =
    15
```

In other words, the maximum global error for Euler's method is $0.0769$ at $t_{14} = 0.95$. (Remember that MATLAB vectors begin with position one.) Similar computations for the improved Euler method yield a maximum global error of $0.00154$ at $t_{11} = 0.80$. In both cases the maximum error occurs after the peak of the exact solution.

## 9.4 Higher Order One-Step Methods

With a global error of $\mathcal{O}(h^2)$ the improved Euler method is called a second order method. Higher order methods can be derived by including additional terms in a Taylor expression. For example, a quadratic Taylor polynomial plus remainder centered at $t_0$ is

$$y(t) = y(t_0) + y'(t_0)(t-t_0) + \frac{1}{2}y''(t_0)(t-t_0)^2 + \frac{1}{6}y'''(c_0)(t-t_0)^3$$

or

$$y(t) = y_0 + f(t_0,y_0)(t-t_0) + \frac{1}{2}\frac{d}{dt}\Big[f(t,y)\Big]\bigg|_{(t_0,y_0)}(t-t_0)^2 + \frac{1}{6}y'''(c_0)(t-t_0)^3. \quad (9.20)$$

With $t = t_1 = t_0 + h$, equation (9.20) becomes

$$y(t_1) = y_0 + hf(t_0,y_0) + \frac{h^2}{2}\frac{d}{dt}\Big[f(t,y)\Big]\bigg|_{(t_0,y_0)} + \frac{1}{6}y'''(c_0)h^3. \quad (9.21)$$

The derivative in the third term on the right hand side of (9.21) is expressed in terms of partial derivatives as follows:

$$\frac{d}{dt}\Big[f(t,y)\Big] = f_t(t,y) + f_y(t,y)\cdot\frac{dy}{dt} = f_t(t,y) + f_y(t,y)\cdot f(t,y).$$

Substituting into (9.21) gives the lengthy expression

$$y(t_1) = y_0 + hf(t_0,y_0) + \frac{h^2}{2}\Big[f_t(t_0,y_0) + f_y(t_0,y_0)\cdot f(t_0,y_0)\Big] + \frac{1}{6}y'''(c_0)h^3$$

that leads to the Taylor approximation with local truncation error of $\mathcal{O}(h^3)$

$$\widehat{y}_1 = y_0 + hf(t_0,y_0) + \frac{h^2}{2}\Big[f_t(t_0,y_0) + f_y(t_0,y_0)\cdot f(t_0,y_0)\Big]. \quad (9.22)$$

Generalizing (9.22) gives the second order Taylor method

$$\widehat{y}_{i+1} = \widehat{y}_i + hf(t_i,\widehat{y}_i) + \frac{h^2}{2}\Big[f_t(t_i,\widehat{y}_i) + f_y(t_i,\widehat{y}_i)\cdot f(t_i,\widehat{y}_i)\Big]. \quad (9.23)$$

The complexity of (9.23) is apparent. First of all we must compute the partial derivatives, $f_t$ and $f_y$, followed by three function evaluations at the point $(t_i,\widehat{y}_i)$. For these reasons Taylor methods are seldom used in practice; however, they are extremely important in the derivation of numerical methods that do not require computation of partial derivatives of $f(t,y)$. One such method is discussed in the next section.

## 9.5 Runge-Kutta Methods

*Runge-Kutta* methods are designed to match the local truncation error of a Taylor method while avoiding the need for partial derivatives of $f(t,y)$. The general form of a Runge-Kutta method is

$$\widehat{y}_{i+1} = \widehat{y}_i + h\Phi(t_i, \widehat{y}_i, h, f) \tag{9.24}$$

where $\Phi$ is a linear combination of values of $f(t,y)$. Runge-Kutta methods are derived by first specifying a $\Phi$ of your choosing containing numerous unknown parameters. The parameters are determined by attempting to make $\Phi$ resemble the terms in an appropriate Taylor method. An example will clarify the procedure.

Suppose $\Phi$ contains only one value of $f$ with three parameters:

$$\Phi(t_i,\, \widehat{y}_i, h, f) = a_1 f(t_i + \alpha_1 h,\, \widehat{y}_i + h\beta_1)$$

With this choice (9.24) becomes

$$\widehat{y}_{i+1} = \widehat{y}_i + h \cdot a_1 f(t_i + \alpha_1 h,\, \widehat{y}_i + h\beta_1). \tag{9.25}$$

We now attempt to find the parameters, $a_1, \alpha_1$and $\beta_1$, so that (9.25) resembles the three term Taylor method in (9.23). At first glance this seems to be an impossible task. There are no partial derivatives in (9.25).

The remedy is found in the Taylor expansion of a function of two variables. For our purposes we simply state the result. The Taylor expansion of $f(t,y)$ centered at the point $(t_i, \widehat{y}_i)$ is

$$\begin{aligned} f(t,y) &= f(t_i,\widehat{y}_i) + (t-t_i)f_t(t_i,\widehat{y}_i) + (y-\widehat{y}_i)f_y(t_i,\widehat{y}_i) \\ &\quad + \tfrac{1}{2!}\Big[(t-t_i)^2 f_{tt}(t_i,\widehat{y}_i) + 2(t-t_i)(y-\widehat{y}_i)f_{ty}(t_i,\widehat{y}_i) + (y-\widehat{y}_i)^2 f_{yy}(t_i,\widehat{y}_i)\Big] \\ &\quad + \text{ Higher Order Terms} \end{aligned}$$

With this result

$$f(t_i + \alpha_1 h,\, \widehat{y}_i + h\beta_1) = f(t_i,\widehat{y}_i) + \alpha_1 h f_t(t_i,\widehat{y}_i) + h\beta_1 f_y(t_i,\widehat{y}_i) + \cdots. \tag{9.26}$$

Substituting into (9.25) gives the expanded form of the Runge-Kutta method.

$$\widehat{y}_{i+1} = \widehat{y}_i + h \cdot a_1\Big[f(t_i,\widehat{y}_i) + \alpha_1 h f_t(t_i,\widehat{y}_i) + h\beta_1 f_y(t_i,\widehat{y}_i) + \cdots\Big]. \tag{9.27}$$

Dropping the higher order terms in (9.27) gives

$$\widehat{y}_{i+1} \approx \widehat{y}_i + h \cdot a_1\Big[f(t_i,\widehat{y}_i) + \alpha_1 h f_t(t_i,\widehat{y}_i) + h\beta_1 f_y(t_i,\widehat{y}_i)\Big]. \tag{9.28}$$

The coefficients in (9.28) are determined by comparison with the second order Taylor method, (9.23).

$$\widehat{y}_{i+1} = \widehat{y}_i + hf(t_i, \widehat{y}_i) + \frac{h^2}{2}\Big[f_t(t_i, \widehat{y}_i) + f_y(t_i, \widehat{y}_i) \cdot f(t_i, \widehat{y}_i)\Big].$$

The terms of both expressions may be lined up for easy comparison.

| | | | | | |
|---|---|---|---|---|---|
| R-K (9.27) | $\widehat{y}_{i+1} \approx$ | $\widehat{y}_i +$ | $a_1 h f(t_i, \widehat{y}_i) +$ | $a_1\alpha_1 h^2 f_t(t_i, \widehat{y}_i) +$ | $a_1\beta_1 \; h^2 f_y(t_i, \widehat{y}_i)$ |
| Taylor (9.23) | $\widehat{y}_{i+1} =$ | $\widehat{y}_i +$ | $h f(t_i, \widehat{y}_i) +$ | $\frac{1}{2} \; h^2 f_t(t_i, \widehat{y}_i) +$ | $\frac{1}{2} f(t_i, \widehat{y}_i) h^2 f_y(t_i, \widehat{y}_i)$ |

Equating corresponding terms it follows that $a_1 = 1, \alpha_1 = \frac{1}{2}$, and $\beta_1 = \frac{1}{2} f(t_i, \widehat{y}_i)$.

These values can now be substituted into (9.25) to produce what is often called the *Midpoint* method.

$$\widehat{y}_{i+1} = \widehat{y}_i + hf(t_i + \frac{1}{2}h,\, \widehat{y}_i + h\frac{1}{2}f(t_i, \widehat{y}_i)). \tag{9.29}$$

Since the midpoint method has matched all terms in the Taylor method through $h^2$ the local truncation error is $\mathcal{O}(h^3)$ and the global error is $\mathcal{O}(h^2)$. Thus, the midpoint method is comparable to the improved Euler method. Although both methods have the same global error performance, this does not mean that the numerical results generated will be the same. It is easy to modify the M-file impeuler.m to implement the midpoint method. This task is left to the problems at the end of this chapter. Figure 9.5 shows the numerical solution of our example initial value problem. The graphical result is similar to the improved Euler method shown in Figure 9.4.

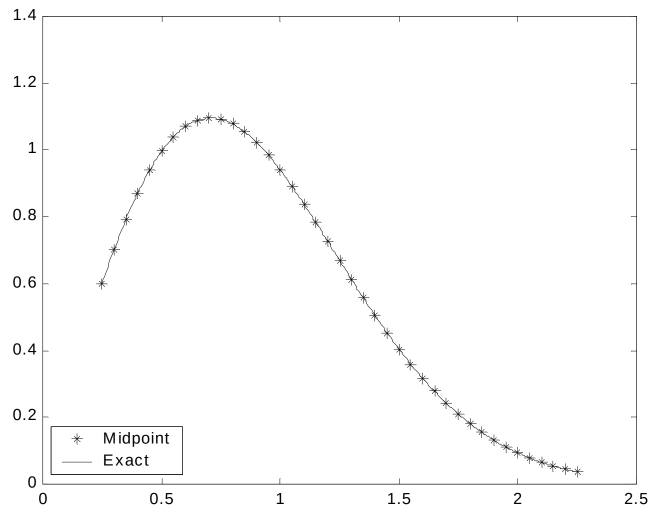

Figure 9.5 Midpoint Method, $h = 0.05$

The midpoint method as well as the improved Euler method are second order Runge-Kutta methods. Using a different form for $\Phi$,

$$\Phi(t_i,\, \widehat{y}_i, h, f) = a_1 f(t_i,\, \widehat{y}_i) + a_2 f(t_i + \alpha_1 h,\, \widehat{y}_i + h\beta_1),$$

many advanced texts derive the improved Euler method as another example of the Runge-Kutta procedure. With other choices for $\Phi$, higher order Runge-Kutta methods with improved global error performance can be derived. The most frequently used Runge-Kutta method is the fourth order scheme where

$$\Phi(t_i, \widehat{y}_i, h, f) = a_1k_1 + a_2k_2 + a_3k_3 + a_4k_4$$

and

$$\begin{aligned}
k_1 &= f(t_i, \widehat{y}_i)\\
k_2 &= f(t_i + \alpha_1 h, \widehat{y}_i + \beta_1 hk_1)\\
k_3 &= f(t_i + \alpha_2 h, \widehat{y}_i + \beta_2 hk_1 + \beta_3 hk_2)\\
k_4 &= f(t_i + \alpha_3 h, \widehat{y}_i + \beta_4 hk_1 + \beta_5 hk_2 + \beta_6 hk_3)
\end{aligned}$$

Using a Taylor polynomial of degree four will give $11$ equations in $13$ unknowns. For example, one of the $11$ equations is $a_1 + a_2 + a_3 + a_4 = 1$. Since $\Phi$ is a weighted sum of slopes (i.e., $f(t, y)$ values), this equation seems reasonable in view of the form of the one-step method (9.24). Note that in the improved Euler method, (9.17), each of the slopes is weighted with $\frac{1}{2}$.

A common choice for the $13$ parameters results in

$$\Phi(t_i, \widehat{y}_i, h, f) = \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4$$

$$\begin{aligned}
k_1 &= f(t_i, \widehat{y}_i)\\
k_2 &= f(t_i + h/2, \widehat{y}_i + hk_1/2)\\
k_3 &= f(t_i + h/2, \widehat{y}_i + hk_2/2)\\
k_4 &= f(t_i + h, \quad \widehat{y}_i + hk_3)
\end{aligned}$$

The Runge-Kutta method, with global error $\mathcal{O}(h^4)$, is

$$\widehat{y}_{i+1} = \widehat{y}_i + \frac{h}{6}[k_1 + 2k_2 + 2k_3 + k_4]. \tag{9.30}$$

As we now know, decreasing the step size will reduce the global error at a particular time. The following table presents data for the Runge-Kutta method using three step sizes and a final time of $t = 2.25$. The true solution is $y(2.25) = 0.0363849$.

| Step Size | Iterations | Approximation | Global Error |
|---|---|---|---|
| $h$ | $i$ | $\widehat{y}_i$ | $\lvert y(2.25) - \widehat{y}_i \rvert$ |
| 0.2 | 10 | 0.036663 | 0.00028 |
| 0.1 | 20 | 0.036399 | 0.000014 |
| 0.05 | 40 | 0.036386 | 0.000001 |

In theory the global error should drop by a factor of approximately $16$ when the step size is cut in half. The results seem consistent with the theoretical predictions.

Figure 9.6 shows the Euler and Runge-Kutta results for our example initial value problem on the interval $[\,0.25,\ 2.25\,]$ with $h = 0.1$. The maximum global error for the Runge-Kutta method for our example is $0.000024$ at $t_5 = 0.75$, again just after the peak of the exact solution.

Based on Figure 9.6 it appears that both methods eventually approach the exact solution, $2.4te^{-t^2+0.0625}$, as $t$ gets large; however, the fourth order Runge-Kutta method is superior. The graphs in Figure 9.6, and others, do not give the complete picture when it comes to comparing methods. For our example initial value problem the exact solution is known; thus, we have the luxury of computing the global errors for all of the methods we have discussed. The economy of higher order method is easy to document.

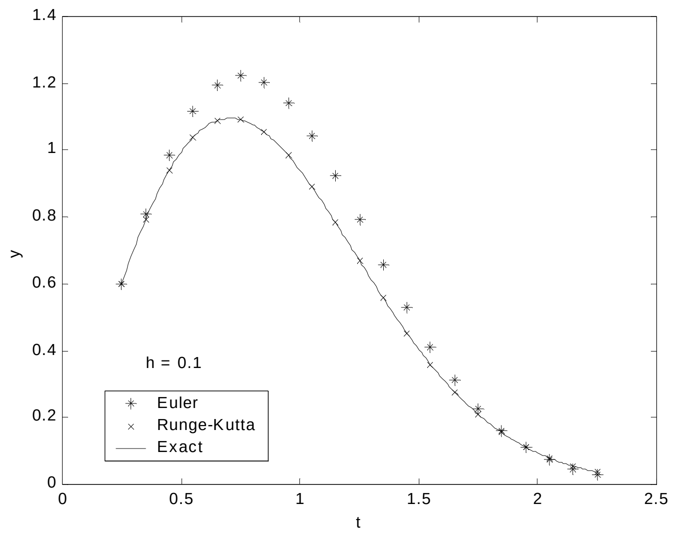

Figure 9.6 Euler and Runge-Kutta, $h = 0.1$

A tabulation of *percent* global error at $t = 2.25$ shows the performance of the four methods we have discussed. Percent error, relative error times $100\%$, is a practical way to compare methods.

Percent Global Error at $t = 2.25$

| Step Size | Iterations | Euler | Improved Euler | Midpoint | Runge-Kutta |
|---|---|---|---|---|---|
| $h$ | $i$ | $\mathcal{O}(h)$ | $\mathcal{O}(h^2)$ | $\mathcal{O}(h^2)$ | $\mathcal{O}(h^4)$ |
| 0.2 | 10 | 55.5 | 36.7 | 19.8 | 0.8 |
| 0.1 | 20 | 21.5 | 6.9 | 3.9 | 0.04 |
| 0.05 | 40 | 9.3 | 1.6 | 0.9 | 0.003 |

The data may be used to make comparisons for our initial value problem. For example, using the midpoint method with $h = 0.05$ is comparable, in terms of percent error, to the Runge-Kutta method with $h = 0.2$. Function evaluation is the time consuming aspect of numerical methods. The midpoint method will require $80$ evaluations, two per step, versus $40$ evaluations, four per step, for the Runge-Kutta method to attain comparable percent error. Once again we see the advantage of higher order methods.

## 9.6 Initial Value Problems and MATLAB

In Section 8.8 we discussed adaptive quadrature wherein the length of a subinterval was adjusted depending on error estimates. Furthermore, in Section 9.3 we saw how integration methods were used to derive numerical methods for the solution of initial value problems. It should be clear that the concept of changing the length of a subinterval while computing a definite integral corresponds to the idea of adjusting the step size, at each step, in the solution of an initial value problem.

Typically, an adaptive step size procedure uses two different methods: one with local truncation error of $\mathcal{O}(h^n)$ and another of $\mathcal{O}(h^{n+1})$. To describe the procedure, assume we have computed an approximation the point $(t_i, \widehat{y}_i)$ and now wish to take the next step to $t_{i+1}$. Using the two methods we compute the approximations, say $\widehat{y}_{i+1}^{(n)}$ and $\widehat{y}_{i+1}^{(n+1)}$, where the superscripts denote the order of the method. If the low order method generates values very close to the higher order method, $\widehat{y}_{i+1}^{(n)} \approx \widehat{y}_{i+1}^{(n+1)}$, there is no need to reduce the step size; however, if there is a major difference between the two values the step size should be reduced and values recomputed. A detailed explanation of how the step size may be adjusted (lower) is discussed in more advanced texts.

MATLAB contains several commands which implement the solution of initial value problems. For example, **ode23** uses Runge-Kutta methods of $\mathcal{O}(h^2)$ and $\mathcal{O}(h^3)$ and **ode45** employs Runge-Kutta methods of $\mathcal{O}(h^4)$ and $\mathcal{O}(h^5)$ in adaptive algorithms. Both functions use a sophisticated procedure called the *Runge-Kutta-Fehlberg* method. Details may be found in many advanced texts. For most problems **ode45** will be satisfactory. See >> **help ode23** or >> **help ode45**.

To illustrate the basic use of **ode45** let's compute the solution of our example initial value problem on the interval $[\,0.25,\ 2.25\,]$. Since ***ode45*** automatically computes an initial step there is no need to specify a step size nor the number of intervals as we did in euler.m and impeuler.m. With $f(t, y)$ in ydot.m, the following MATLAB commands will compute the numerical solution with the output values in the column vectors **ti** and **yi**.

```
>> tint = [0.25, 2.25];
>> y0 = 0.6;
>> [ti, yi ] = ode45('ydot',tint,y0);
```

The results are plotted in Figure 9.7. The values of $t_i$ where the solution is actually computed are shown on the horizontal axis. **diff(ti)** will reveal the step sizes used by **ode45**. For our example problem only three step sizes are used $h = 0.014354,\ 0.05$ and $0.035646$. The smallest step size is used at the beginning of the interval where the slope of the solution curve is the largest. As you can see, the step size of $0.05$ is used for most of the interval. The maximum global error occurs at $t_i = 2.0074$ and equals 0.000012.

If necessary, MATLAB does allow the user to specify numerous parameters in the solution process. Parameter values may be passed to **ode45** by adding a fourth argument, say **options**, to the calling list as follows:

```
>> [ti, yi ] = ode45('ydot',tint,y0,options);
```

The argument **options** is constructed using MATLAB's **odeset**. See >> **help odeset**. In most cases, the default values automatically specified are more than adequate.

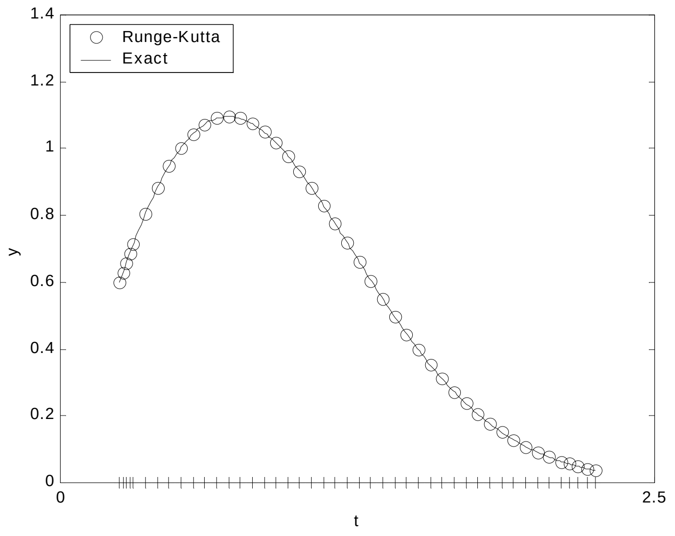

Figure 9.7 Solution from **ode45**

## Problems

9-1. a. Consider the initial value problem $y' = y + 3t^2$, $y(0) = 1$. Using hand/calculator computations with $h = 0.1$, approximate $y(0.3)$ using each of the following methods: Euler, Improved Euler, Midpoint, Runge-Kutta $4^{th}$ order. Construct a table to display your results.

b. The solution of the IVP in part a is $y(t) = 7e^t - 3(t^2 + 2t + 2)$. Construct a table to display the errors at each step for all four methods.

9-2. Modify the M-file impeuler.m to implement the midpoint method, (9.29), and the Runge-Kutta method, (9.30). Suggested M-file names: midpoint.m and rkutta.m. Test your modifications on the text example.

9-3. Use MATLAB and the M-files of problem 9-2 to repeat problem 9-1 with $h = 0.05$.

9-4. Use a forward difference approximation from Section 8.2 to derive Euler's method.

9-5. Show that the improved Euler method, (9.16), and the second order Taylor method, (9.23), are identical if $f(t,y) = at + by$ where $a$ and $b$ are constants.

9-6. Consider the example IVP in the text: $y' = (-2t + 1/t)$, $y(0.25) = 0.6$. Using $h = 0.1$ on the interval $[0.25,\ 2.25]$, determine when the midpoint method provides a more accurate solution compared to the improved Euler method by computing the errors at each step. Use absolute values.

9-7. Use **ode23** to compute the solution of the example IVP. Graph the solution, see Figure 9.7. Determine the maximum global error on the interval $[0.25,\ 2.25]$ based on the data from **ode23**.

9-8. Consider the initial value problem: $y' = 0.5e^{(-t)}/y$ with $y(0) = 2$. The exact solution is $y(t) = \sqrt{5 - e^{(-t)}}$.
a. Use the improved Euler method with $h = 0.2,\ 0.1$ and $0.05$ to estimate $y(4)$.
b. Compute the errors at $t = 4$ and verify that the improved Euler method is a second order method.

For problems 9-9 through 9-12 consider the *logistic* initial value problem: $y' = 25y - y^2$ with $y(0) = 1$. The solution is $y(t) = 25/(1 + 24e^{-25t})$.

9-9. Analyze the logistic IVP as follows:
a. Use Euler's method with $h = 0.1$ to estimate $y(0.3)$ and determine the maximum global error.
b. Repeat part a with $h = 0.01$
c. Explain why $\widehat{y}_1$ is below the exact solution.

9-10. Using four methods, Euler, improved Euler, midpoint and Runge-Kutta, each with the same step size, $h = 0.1$, graph the solutions on the interval $[0,\ 3.0]$. Comment on the results of this numerical experiment. Compute the global errors at $t = 3.0$.

9-11. Repeat problem 9-10 using **ode23** and **ode45**.

9-12. Differential equations whose solution contains a rapidly decaying term such as $e^{-25t}$ are often called *stiff* problems. In other words, $e^{-25t}$ is an important factor in the solution for a very short time. MATLAB contains special commands to solve stiff initial value problems. The algorithms for treating stiff problems are beyond the scope of our efforts. See >> **help ode23s**. Repeat problem 9-10 using **ode23s**.

9-13. Consider the differential equation $y' = (2t + y)/(t + 2y + 0.1)$.

a. Using two different numerical methods of different order, compute the solution beginning at the point $(-2, -3)$. Plot your data.
b. Repeat part a using the initial point $(-2, -2)$.
c. Discuss the results of your numerical computations.
Hint: The slope is undefined whenever $t + 2y + 0.1 = 0$.

9-14. An example of a *Riccati* initial value problem is: $y' = t^2 + y^2$, with $y(1) = 1$. Although the Riccati equation appears simple, the factor of $t^2$ will cause the slope to grow very rapidly as $t$ increases. Compute the solution at $t = 1.7$ using several different methods. Discuss the results of your numerical computations.