MTH 244 - Additional Information for §7.2 and §7.3

Section 1 (Merino) and section 3 (Dobrushkin) - March 2003

Sections 7.2 and 7.3 are about numerical procedures used to obtain approximate solutions to the initial-value problems for ordinary differential equations,

$$y' = f(x, y), \qquad y(x_0) = y_0; \tag{1}$$

Numerical methods for solving differential equations produce sequences of points (x_1, y_1), (x_2, y_2), $\ldots$, (x_k, y_k), where y_k is an approximation to $y_{sol}(x_k)$, the actual solution at the point x_k.

Here we shall be considering points the case the x_k's are incremented by a fixed amount, $h > 0$. The general scheme for producing points is the following.

Given a current point (x, y) and a step $h > 0$, generate a new point (x_{next}, y_{next}) as follows:

$$\begin{cases} x_{\texttt{next}} = x + h \\ y_{\texttt{next}} = y + \Phi(x, y, h) \end{cases} \tag{2}$$

Here Φ is known beforehand and it may depend on x, y and h, (that's why we write $\Phi(x, y, h)$). There are many possible choices for Φ, each with advantages and disadvantages.

The value $y_{\texttt{next}}$ in Eq. 2 is intended to approximate $y_{sol}(x+h)$, the value of the actual solution of the initial value problem (1) at the point $x + h$.

Example 1: Euler's method

The simplest choice of Φ that gives a working method is when $\Phi = h\, f(x, y)$, where $f(x, y)$ is the function that appears in equation (1). In this case the resulting method is precisely *Euler's method*, which was discussed in MTH 142. Euler's method is not used much in practice, except for didactical purposes (it is easy to explain, and understanding how it works helps when studying more complicated methods). The formulas for updating x and y with Euler's method are the following:

$$\begin{cases} x_{\texttt{next}} = x + h \\ y_{\texttt{next}} = y + hf(x, y) \end{cases} \tag{3}$$

PROBLEM 1.1

(a) Solve with maple the IVP $y' = x^2 + y^2$, $y(0) = 0$. Plot the solution curve for $0 \le x \le 1.5$.
(b) Write Maple code implementing Euler's method to approximate the solution to the IVP with $h = 0.15$. Calculate 10 points $x_1, x_2, \ldots, x_{10}$. Plot the approximation y_{app} together with the actual solution y_{sol}.
(c) Calculate y_{sol} at the grid points $x_1, x_2, \ldots, x_{10}$. and plot the error $y_{sol}(x_\ell) - y_{app}(x_\ell)$.

PROBLEM 1.2

a) For the IVP given in Problem 1.1, calculate an approximate solution with Euler's method and $h = 0.015$. Calculate 100 points. Plot the approximate solution together with the solution.
b) same as part (a), only now with $h = 0.0015$ and 1000 points.
c) Calculate the error $y_{sol} - y_{app}$ for for the answers (a) and (b). Produce a logplot of the error for both (a) and (b).

Example 2: The Taylor Series Method

The formula for the update y_{next} in Euler's method is based on the linearization of y about x. A more accurate method is obtained if an order 2 Taylor polynomial is used instead. This is the *Taylor Series method of order 2*. The corresponding formulas for solving the initial value problem (1) are shown below.

$$\begin{cases} x_{\texttt{next}} = x + h \\ y_{\texttt{next}} = y + hf(x,y) + \frac{h^2}{2}\,(\ f_x(x,y) + f_y(x,y)f(x,y)\ ) \end{cases} \tag{4}$$

One can also obtain formulas for Taylor Series methods of orders larger than 2, but the formulas become quite complicated.

PROBLEM 2

Redo problems 1.1 and 1.2, but now use the Taylor Series method of order 2 instead. Compare the new results with the ones obtained with Euler's method.

Runge-Kutta Methods

The methods associated with the names of Runge, Kutta and others are among the more popular as well as most accurate numerical procedures used to obtain approximate solutions to the initial-value problems for ordinary differential equations,

$$y' = f(x,y), \qquad y(x_0) = y_0; \tag{5}$$

The main idea of this method is due to Runge (1895). The main features of the Runge-Kutta methods are that the updates $y_{\texttt{next}}$ are calculated with formulas that involve function evaluation of $f(x,y)$ at different points, and that the methods exhibit hight accuracy.

Recall the general scheme for producing a new point (x_{next}, y_{next}):

$$\begin{cases} x_{\texttt{next}} = x + h \\ y_{\texttt{next}} = y + \Phi(x,y,h) \end{cases}$$

In the table given in page 3, you will find the formulas for many special cases of Runge-Kutta.

Error

The (local) *truncation error* of the method at the point (x,y) is defined by

$$T(x,y,h) = y_{sol}(x+h) - y_{\texttt{next}}.$$

The method Φ is said to have *order* p if $|T(x,y,h)| < C\,h^p$, where C is a constant independent of x, y, and h. We express this property in symbols as $T(x,y,h) = O\,(h^p)$. The expression $O(h^n)$ means a function of h that proportional to h^n when $h \to 0$.

It is shown in more advanced courses of Numerical Analysis that that the error in Euler's method is order $p = 2$ (that is, the error is proportional to h^2), and the Taylor Series Method mentioned above is order $p = 3$ (error is proportional to h^3). The Runge Kutta methods, on the other hand, have order $p = 5$. That is, they are at least two orders of magnitude better than Euler's Method and the Taylor Series methods discussed above.

Note: The truncation error requires exact value of (x,y) (it is local). When going from x_0 to x_n there is an accumulation of error, resulting in the **global truncation error**, roughly equal to n times the local error. Since n is inversely proportional to h, the global error is one power of h less than the local error. For example, Euler's method is order 2, but if you plot approximations to the solution corresponding to different values of h on an interval $[a,b]$, you will see an order 1 improvement in the error. (That is, a factor of ten decrease in h results in a factor of 10 decrease in the error).

1. The *improved Euler's method* **or the** *Heun's method*:
$y_{n+1} = y_n + \frac{h}{2} f(x_n, y_n) + \frac{h}{2} f(x_n + h, y_n + hf(x_n, y_n)) + T_n,$

2. The *improved polygon method* **or the** *modified Euler's method*:
$y_{n+1} = y_n + hf(x_n + h/2, y_n + hf(x_n, y_n)/2) + T_n,$

3. The third order Runge-Kutta method:

$$y_{n+1} = y_n + \tfrac{h}{6}(k_1 + 4k_2 + k_3), \text{ where } \begin{array}{rcl} k_1 &=& f(x_n, y_n), \\ k_2 &=& f(x_n + h/2, y_n + k_1 h/2), \\ k_3 &=& f(x_n + h, y_n + 2hk_2 - hk_1). \end{array}$$

4. The fourth order Runge-Kutta method:

$$y_{n+1} = y_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \text{ where } \begin{array}{l} k_1 = f_n = f(x_n, y_n), \\ k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\ k_3 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), \\ k_4 = f(x_n + h, y_n + hk_3). \end{array}$$

5. The fourth order Runge-Kutta method:

$$y_{n+1} = y_n + \tfrac{h}{8}(k_1 + 3k_2 + 3k_3 + k_4), \text{ where } \begin{array}{l} k_1 = f_n = f(x_n, y_n), \\ k_2 = f\left(x_n + \frac{h}{3}, y_n + \frac{h}{3}k_1\right), \\ k_3 = f\left(x_n + \frac{2h}{3}, y_n - \frac{h}{3}k_1 + hk_2\right), \\ k_4 = f(x_n + h, y_n + hk_1 - hk_2 + hk_3). \end{array}$$

6. The Gill's formula (fourth order): $y_{n+1} = y_n + \frac{h}{6}\left[k_1 + 2\left(1 - \frac{1}{\sqrt{2}}\right)k_2 + 2\left(1 + \frac{1}{\sqrt{2}}\right)k_3 + k_4\right],$

$$\text{where } \begin{array}{l} k_1 = f(x_n, y_n), \\ k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\ k_3 = f\left(x_n + \frac{h}{2}, y_n - \left(\frac{1}{2} - \frac{1}{\sqrt{2}}\right)hk_1 + \left(1 - \frac{1}{\sqrt{2}}\right)hk_2\right), \\ k_4 = f(x_n + h, y_n - \frac{1}{\sqrt{2}}hk_2 + \left(1 + \frac{1}{\sqrt{2}}\right)hk_3). \end{array}$$

7. The Runge-Kutta-Fehlberg method: $y_{n+1} = y_n + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6,$

$$\text{where } \begin{array}{l} k_1 = hf(x_n, y_n), \\ k_2 = hf\left(x_n + \frac{h}{4}, y_n + \frac{1}{4}k_1\right), \\ k_3 = hf\left(x_n + \frac{3h}{8}, y_n + \frac{3}{32}k_1 + \frac{9}{32}k_2\right), \\ k_4 = hf\left(x_n + \frac{12h}{13}, y_n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right), \\ k_5 = hf\left(x_n + h, y_n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right), \\ k_6 = hf\left(x_n + \frac{h}{2}, y_n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right). \end{array}$$

8. The fourth order Runge-Kutta-Nystrom method for the second order differential equation $y'' = f(x, y(x), y'(x))$:

$$\begin{array}{ll} F_1 = hf(x_k, y_k, y'_k), & F_a = h\left(y'_k + \frac{1}{2}F_1\right) \\ F_2 = hf\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}F_a, y'_k + \frac{1}{2}F_1\right), & F_3 = hf\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}F_a, y'_k + \frac{1}{2}F_2\right), \\ F_b = h\left(y'_k + \frac{1}{2}F_3\right), & F_4 = hf\left(x_{k+1}, y_k + F_b, y'_k + F_3\right), \\ y_{k+1} = y_k + h\left(y'_k + \frac{1}{6}(F_1 + F_2 + F_3)\right), & y'_{k+1} = y'_k + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4). \end{array}$$

Maple Code Solution of Problems 1.1, 1.2 and 2

```
> restart;
> with(DEtools):
> with(plots):

Problem 1.1.a ------------------------------------------------
> dsolve({diff(y(x),x)=y(x)^2+x^2,y(0)=0},y(x));
> fsol:=x -> -x*(-BesselJ(-3/4,1/2*x^2)+BesselY(-3/4,1/2*x^2))/
              (-BesselJ(1/4,1/2*x^2)+BesselY(1/4,1/2*x^2)):
> psol:=plot(fsol(x),x=0..1.5):
> display(psol);

Problem 1.1.b ------------------------------------------------
> x[0]:=0.:  y[0]:=0: h := 0.15:
> for k from 0 to 9 do
    x[k+1] := x[k]+h:
    y[k+1] := y[k]+(x[k]^2+y[k]^2)*h:
  od:
> ptsapp1 := seq([x[k],y[k]],k=1..10):
> ptssol1 := seq([x[k],fsol(x[k])],k=1..10):
> p1app:=pointplot({ptsapp1}):
> display(psol,p1app);
> errorplot1 := pointplot( {seq([x[k],fsol(x[k])-y[k]],k=1..10)}):
> display(errorplot1);

PROBLEM 1.2.a:-----------------------------------------------
> x[0]:=0.:  y[0]:=0: h := 0.015:
> for k from 0 to 99 do
    x[k+1] := x[k]+h:
    y[k+1] := y[k]+(x[k]^2+y[k]^2)*h:
  od:
> ptsapp2 := seq([x[k],y[k]],k=1..100):
> ptssol2 := seq([x[k],fsol(x[k])],k=1..100):
> p2app:=pointplot({ptsapp2}):
> display(psol,p2app);
> errorplot2 := pointplot( {seq([x[k],fsol(x[k])-y[k]],k=1..100)}):
> display(errorplot2);

PROBLEM 1.2.b ------------------------------------------------
> x[0]:=0.:  y[0]:=0: h := 0.0015:
> for k from 0 to 999 do
    x[k+1] := x[k]+h:
    y[k+1] := y[k]+(x[k]^2+y[k]^2)*h:
  od:
> ptsapp3 := seq([x[k],y[k]],k=1..1000):
> ptssol3 := seq([x[k],fsol(x[k])],k=1..1000):
> p3app:=pointplot({ptsapp3}):
> display(psol,p3app);
> errorplot3 := pointplot( {seq([x[k],fsol(x[k])-y[k]],k=1..1000)}):
> display(errorplot3);

PROBLEM 1.2.c ------------------------------------------------
> display(errorplot1,errorplot2,errorplot3);
> errorplot12c := logplot( [seq([x[k],fsol(x[k])-y[k]],k=1..100)],symbol=box):
> display(errorplot12c);

Problem 2 ----------------------------------------------------
> x[0]:=0.:  y[0]:=0: h := 0.15:
> for k from 0 to 9 do
    x[k+1] := x[k]+h:
    y[k+1] := y[k]+(x[k]^2+y[k]^2)*h + (2*x[k] + 2*y[k]*(x[k]^2+y[k]^2) )*h^2/2:
  od:
> ptsapp4 := seq([x[k],y[k]],k=1..10):
> ptssol4 := seq([x[k],fsol(x[k])],k=1..10):
> p4app:=pointplot({ptsapp4}):
> display(psol,p4app);
```