

## Handout Six Part II

November 3, 2011

### 1 Supplementary Notes : Solving an Initial Value Problem for an Ordinary Differential Equation

It is not the aim of these notes to teach a course on the numerical solution of ordinary differential equations (ODE's). However you do need to be able to understand the methodology behind the subject so you are able to 'code up' a simple method. A numerical method for solving an ordinary differential equation is a procedure that produces approximate solutions to the differential equation at progressive points along the independent variable. It does this using the given differential equation and an accompanying initial condition as a starting point.

You will only need to write Fortran programs for problems of the form;

$$y' = \frac{dy}{dx} = f(x, y) \rightarrow \text{differential equation} \quad (1)$$

$$y_0 = y(x_0) \rightarrow \text{initial condition} \quad (2)$$

The first equation is simply the differential equation, it gives us the 'rate of change' of the dependent variable  $y$  with respect to the independent variable  $x$ . The second equation tells us that at a known point of the independent variable  $x$  the value of the function  $y(x)$  is known and is  $y_0$ . With these initial conditions we can attempt to produce a numerical solution to the ODE over a range of values in  $x$ . This is done by starting at  $x_0$  where we know the solution  $y_0$  as it is given, and making an approximation for  $y$  at  $x = x_0 + h$ . The differential equation is then used, along with a small step  $h$  in the  $x$  direction to give an estimate  $\delta y$  of the change in  $y$  that the small step  $h$  in  $x$  would produce. There are many different methods for solving ODE's, some very much more complicated than others. In this handout we will look at seven of the simpler methods around. The Euler method, Heun's method, Nystrom's and a third order Runge-Kutta method.

#### 1.1 The Order of a Numerical Method

In general the higher the order of a numerical method the more accurate it is. The order of a numerical method is given as a positive integer. Euler's method is of order one, so it the lowest possible order you can have. If the exact solution of an ordinary differential equation is a polynomial of order  $k$  then the numerical solution and the exact solution will be identical (except for computational truncation error) if the numerical method is of order  $k$  or above.

#### 1.2 Simple Numerical Methods

First we will look at a few of the simplest of numerical methods for ODE's. Then a couple of methods well known group of 'Runge-Kutta' based solutions. Finally a couple of 'predictor-corrector' methods.

##### 1.2.1 Euler's Method

Euler's method is probably the most simple of all numerical methods for solving ordinary differential equations numerically. Given the point  $(x_n, y_n)$  and the gradient  $f_n$  at that point, where  $f_n = f(x_n, y_n)$ ,

Euler's method makes the assumption that over the sufficiently small step,  $h$ , in  $x$  the function is approximately linear. This allows us to write down an approximation for  $y_{n+1}$  at  $x_{n+1} = x + h$ .

$$y_{n+1} = y_n + hf(x_n, y_n) \quad (3)$$

$$x_{n+1} = x_n + h \quad (4)$$

The top plot in figure (1.2.1) shows geometrically the first step using the Euler method. Initially both  $x_0$  and  $y_0$  are known. The derivative function  $y' = f(x, y)$  is given and calculated for  $x = x_0$  and  $y = y_0$  to give us,  $f_0$ , the derivative of  $y$  w.r.t.  $x$  at the initial point  $(x_0, y_0)$ . This derivative is then multiplied by our chosen step size,  $h$ , to give us an estimate of  $\delta y$  the corresponding increment in  $y$  given the increment  $h$  in  $x$ . This  $\delta y$  is then added to  $y_0$  to give us  $y_1$ , our numerical approximation to  $y(x + h)$ . So for each new estimate using the Euler method **only** information from the **previous** single time step is required.

This process can be repeated using  $(x_1, y_1)$  to give us  $f_1$  and then used to calculate  $y_2 = y_1 + hf_1$ , and so on. The lower plot in figure (1.2.1) shows a series of six Euler steps, note how the numerical solution is made up from a series of straight line segments. The plot also demonstrates how the error builds up with each iteration. The choice of the step-size  $h$  is arbitrary, as a general rule, the smaller  $h$  the more accurate the numerical solution will be.

### 1.2.2 Heun's Method

Heun's method is often called the 'modified Euler method' or the 'implicit Euler method'. The difference between Heun's method and Euler's method is that Heun's method uses information from two time-steps, not the single previous time-step like Euler's method. The equations for Heun's method are,

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{2} [f(x_n, y_n) + f(x_{n+1}, y_{n+1})] \\ x_{n+1} &= x_n + h \end{aligned} \quad (5)$$

So in Heun's method we are using as our estimate of the gradient between two successive points, the average of the gradients at each of the points. There is an obvious problem here and it is in equation (6), can you spot it? The problem is that  $y_{n+1}$  appears on both sides of the equation so how can we calculate the RHS of (6) when we do not know  $y_{n+1}$ ? This is done by writing down Heun's method in the form,

$$\begin{aligned} \tilde{y}_{n+1} &= y_n + hf_n \\ y_{n+1} &= y_n + \frac{h}{2} [f(x_n, y_n) + f(x_{n+1}, \tilde{y}_{n+1})] \\ x_{n+1} &= x_n + h \end{aligned} \quad (6)$$

Therefore the problem in (6) is solved by approximating the  $y_{n+1}$  on the RHS with  $\tilde{y}_{n+1}$ . The  $\tilde{y}_{n+1}$  is simply an approximation obtained from an Euler step. Heun's method is of order two so is more accurate for a given step size than Euler's method.

### 1.2.3 Nystrom's Method

Nystrom's method is given by the formula,

$$\begin{aligned} y_{n+1} &= y_{n-1} + 2hf_n \\ x_{n+1} &= x_n + h \end{aligned} \quad (8)$$

Again we have a problem here in equation (8). Can you spot what it is? The first term on the RHS is ' $y_{n-1}$ ', this presents a problem when trying to execute the first time step! To be more precise, in order to calculate  $y_1$  using Nystrom's method we would need  $y_0$  and also  $y_{-1}$ , however  $y_{-1}$  we do not have!

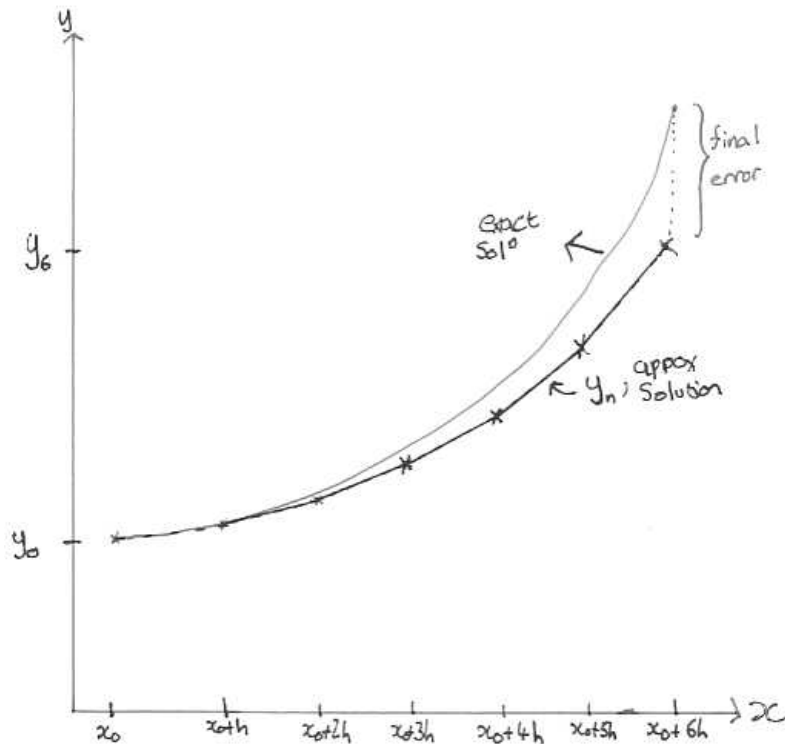
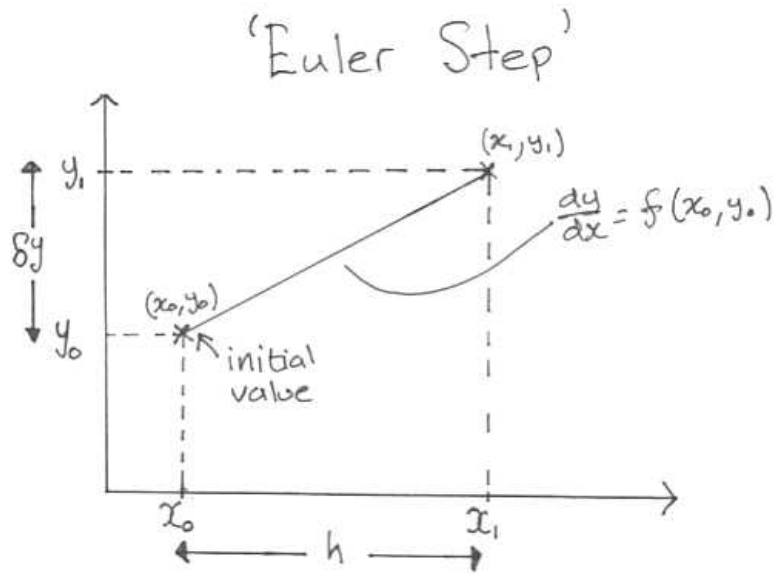


Figure 1: The first Euler Step (top) and a series of six Euler steps (bottom).

So to get around this problem the first step ( $y_1$ ) is calculated using Heun's method and all the following steps can then be calculated using Nystrom's method. Nystrom's method, like Heun's method, is also a second order method. This is why we would use Heun's method to do the first step and not Euler's. Nystrom's method is an example of a **'non-self-starting method'** because it needs information from

two previous time steps in order to proceed and therefore requires a ‘**self-starting method**’ to calculate the first time-step. This method is also second order.

### 1.3 Runge-Kutta Methods

Runge-Kutta methods are a well known set of ‘self-starting’ methods. In fact Euler’s method is a first order Runge-Kutta method and Heun’s method is a second order Runge-Kutta method!

#### 1.3.1 A Third order Runge-Kutta Method

The following method is a third order Runge-Kutta method;

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 4k_2 + k_3) \quad (9)$$

where,

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\ k_3 &= hf(x_n + h, y_n - k_1 + 2k_2) \\ x_{n+1} &= x_n + h \end{aligned}$$

So first one computes the various  $k$  values  $1 \rightarrow 3$  and then  $y_{n+1}$ .

#### 1.3.2 A Fourth order Runge-Kutta Method

The following method is a fourth order Runge-Kutta method;

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (10)$$

where,

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\ k_3 &= hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right) \\ k_4 &= hf(x_n + h, y_n + k_3) \\ x_{n+1} &= x_n + h \end{aligned}$$

So first one computes the various  $k$  values  $1 \rightarrow 4$  and then  $y_{n+1}$ .

### 1.4 Multi-step Methods

The methods discussed previously only require information from one previous timestep and are therefore single-step methods. Multi-step methods require information from two or more timesteps. If the information is not available in the form of initial conditions then in order for a multi-step method to proceed the previous timestep values must be calculated by a self starting single-step method

#### 1.4.1 Adams-Bashforth Two-Step Method

The following method is the two step Adams-Bashforth and needs the solution value at the previous time-step in order to work. The method is second order.

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{2}(3f_n - f_{n-1}) \\ x_{n+1} &= x_n + h. \end{aligned}$$

### 1.4.2 Adams-Bashforth Three-Step Method

The following method is the three step Adams-Bashforth and needs the solution value from the two previous time-steps in order to work. The method is third order.

$$\begin{aligned}y_{n+1} &= y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}) \\x_{n+1} &= x_n + h.\end{aligned}$$

### 1.4.3 Adams-Bashforth Four-Step Method

The following method is the four step Adams-Bashforth and needs the solution value from the three previous time-steps in order to work. The method is fourth order.

$$\begin{aligned}y_{n+1} &= y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \\x_{n+1} &= x_n + h.\end{aligned}$$

## 1.5 Predictor-Corrector Methods

A ‘predictor-corrector’ method uses a set of two equations to calculate  $y_{n+1}$ . The first equation is called the predictor step and is used to give an initial approximation to  $y_{n+1}$  denoted by  $py_{n+1}$ . The second equation is the corrector step and makes use of the predicted value of  $y_{n+1}$  to give a better corrected approximation to  $y_{n+1}$ .

### 1.5.1 A Simple Second-Order Method

The trapezoidal method for ordinary differential equations is,

$$\begin{aligned}y_{n+1} &= y_n + \frac{h}{2}(f_n + f_{n+1}) \\x_{n+1} &= x_n + h.\end{aligned}$$

Note that if we let  $f_{n+1}$  be predicted by using Euler’s method,  $y_{n+1} = y_n + hf_n$ , to compute  $f_{n+1}$  the above then becomes Heun’s method. If we combine ‘Nystroms’ method with the trapezoidal method then we have a second order method given by

$$\begin{aligned}py_{n+1} &= y_{n-1} + 2hf_n \\y_{n+1} &= y_n + \frac{h}{2}(f_n + pf_{n+1}) \\x_{n+1} &= x_n + h.\end{aligned}$$

Note the predicted value of  $y_{n+1}$  is denoted by  $py_{n+1}$  and it then follows that  $pf_{n+1} = f(x_{n+1}, py_{n+1})$ . Also remember that the term  $y_{n-1}$  is not known initially so another self starting method is required to do the first step.

### 1.5.2 Milne’s Method

Milne’s method is a fourth order predictor-corrector method and is defined as follows.

$$\begin{aligned}py_{n+1} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\y_{n+1} &= y_{n-1} + \frac{h}{3}(pf_{n+1} + 4f_n + f_{n-1}) \\x_{n+1} &= x_n + h.\end{aligned}$$

### 1.5.3 Hamming's Method

Hamming's method is also a fourth order predictor-corrector method and uses the same predictor as Milne's but a different corrector.

$$\begin{aligned}py_{n+1} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\y_{n+1} &= \frac{1}{8}(9y_n - y_{n-2}) + \frac{3h}{8}(pf_{n+1} + 2f_n - f_{n-1}) \\x_{n+1} &= x_n + h.\end{aligned}$$

### 1.5.4 Starting Values

The methods presented in this section all require more starting values than are given in a first order initial value problem. In addition to  $y_0$  the simple second-order method requires  $y_1$  and both Milne's method and Hamming's method require  $y_1, y_2$  and  $y_3$ . To generate a starting value for the second order method another second order method is used, for example Heun's method. A fourth order Runge-Kutta method is often used to generate starting values for both Milne's method and Hamming's method.

## 1.6 Modified Predictor-Corrector Methods

From an error analysis of the fourth order predictor-corrector methods the predicted value of  $y_n$  can be modified (that is, improved). The modified value of  $y_n$  can then be used in the corrector instead of the originally calculated predicted value. So we let  $py_n$  represent the predicted value of  $y_n$  and denote the modified value of  $y_n$  by  $my_n$ .

### 1.6.1 Modified Milne's Method

$$\begin{aligned}py_{n+1} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\my_{n+1} &= py_{n+1} + \frac{28}{29}(y_n - py_n) \\y_{n+1} &= y_{n-1} + \frac{h}{3}(mf_{n+1} + 4f_n + f_{n-1}) \\x_{n+1} &= x_n + h.\end{aligned}$$

### 1.6.2 Modified Hamming's Method

$$\begin{aligned}py_{n+1} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\my_{n+1} &= py_{n+1} + \frac{112}{121}(y_n - py_n) \\y_{n+1} &= \frac{1}{8}(9y_n - y_{n-2}) + \frac{3h}{8}(mf_{n+1} + 2f_n - f_{n-1}) \\x_{n+1} &= x_n + h.\end{aligned}$$

### 1.6.3 Starting Values

Neither of the above two modified methods can be started until  $y_0, y_1, y_2, y_3$  and  $y_4$  are known. The value of  $y_0$  being prescribed by the initial condition. The values  $y_1, y_2$  and  $y_3$  are, as with the non-modified methods, calculated using the fourth order Runge-Kutta method. The standard Milne or Hamming method can then be used to calculate  $y_4$ . The reason that the modified version can not be used to calculate  $y_4$  is the dependence of  $my_{n+1}$  on  $py_n$  and in order to calculate  $py_n$  we need to know  $y_{n-4}$ .

## Assignment Two

Deadline Tuesday 15<sup>th</sup> November 11:00pm

An initial value problem can be written in the form,

$$\begin{aligned}y' &= \frac{dy}{dx} = f(x, y) \\ y_0 &= y(x_0)\end{aligned}$$

In your `'assign2'` directory create a file called `'odemod.f90'`. In the file `'odemod.f90'` write a module that contains five **subroutines** and one **function** as follows.

1. A subroutine to perform a single step of the Runge-Kutta fourth order method. Call this subroutine `'rk4'` and it should accept as its arguments `'(y,x,h)'` and update `'y'` and `'x'` with their new values.
2. A subroutine to perform a single step of the Adams-Bashforth two step method. Call this subroutine `'adam2'` and it should accept as its arguments `'(y,x,h)'` and update `'y'` and `'x'` with their new values. This method is not self starting so when necessary you should call your `'rk4'` subroutine from 'inside' your `'adam2'` subroutine. The updated `'y'` value from `'RK4'` is taken as the updated `'y'` value for this step of the `'adam2'` method. [HINT : Create a variable, called `'oy1'`, with the `'SAVE'` attribute to hold the *'old y-value'* inside your subroutine `'adam2'`].
3. A subroutine to perform a single step of the Adams-Bashforth three step method. Call this subroutine `'adam3'` and it should accept as its arguments `'(y,x,h)'` and update `'y'` and `'x'` with their new values. Your `'adam3'` subroutine should call your `'rk4'` subroutine when required as instructed for the `'adam2'` subroutine. [HINT : Create variables, called `'oy1, oy2'`, with the `'SAVE'` attribute to hold the *'old y-values'* inside your subroutine `'adam3'`].
4. A subroutine to perform a single step of the Adams-Bashforth four step method. Call this subroutine `'adam4'` and it should accept as its arguments `'(y,x,h)'` and update `'y'` and `'x'` with their new values. Your `'adam4'` subroutine should call your `'rk4'` subroutine when required as instructed for the `'adam2'` subroutine. [HINT : Create variables, called `'oy1, oy2, oy3'`, with the `'SAVE'` attribute to hold the *'old y-values'* inside your subroutine `'adam4'`].
5. A subroutine to perform a single step of the **modified** Hamming's fourth order predictor-corrector method. Call this subroutine `'modhamming'` and it should accept as its arguments `'(y,x,h)'` and update `'y'` and `'x'` with their new values. Remember this method is not self starting so when necessary, to advance the single required step, your code should call your `'rk4'` subroutine from 'inside' your `'modhamming'` subroutine. The updated `'y'` value from `'RK4'` is taken as the updated `'y'` value for this step of the `'modhamming'` method. When you have accumulated enough previous values, use the standard Hamming's method for one step. Then, for all further steps, you can now use the **modified** Hamming's method.
6. A function, to be used by all the above methods, which will calculate the differential equation for given values  $y$  and  $x$ . Call this function `'ode'` and it should accept as its arguments `'(x,y)'`.

Write a main program unit `'main.f90'` that solves the differential equation,  $y' = -y + x + 2$ , with the initial condition  $y(0) = 2$ . The exact solution to this equation is  $y = e^{-x} + x + 1$ . For each method edit your main program to call the different methods and produce respective text files `'rk4.txt'`, `'adam2.txt'`, `'adam3.txt'`, `'adam4.txt'` and `'modhamming.txt'`. Remember that you were shown how to create text files from your code in assignment one. These results text files should hold a table with four columns. The first column holding the value  $x_n$ , the second column the approximation  $y_n$  and the third column the true solution  $y(x_n)$  and the fourth column the absolute value of the error in the approximation at that step. For each method take ten steps with a step-size of  $h = 0.1$ . Each column should have an

appropriate title and all but the first column should be formatted to hold six decimal places. The first column should be formatted appropriately.

All your code should follow the methods of good programming practice that you have learnt so far.

---