

**Second Year Engineering Mathematics Laboratory**  
**Hilary Term 2000**

Ian Reid, 20/12/1999


**Exercise 4: ODEs and dynamical simulation**


The purpose of this exercise is to reinforce your understanding of algorithms for solving ODEs (Ordinary Differential Equations). In particular, different methods have differing orders of accuracy. You will look at Euler (1st order) and Modified Euler (2nd order). The makers of **matlab** have gone to a great deal of trouble to build in reliable functions, so you will eventually throw away your home made Modified Euler function, and replace it with the more sophisticated **matlab**, built-in version. This is used to model a practically important physical model — too hard to solve on paper — the “van Der Pohl” oscillator.

## 1 Preparation

Reread relevant sections of the lecture notes.

### A. Euler simulation of SHM

 What are the exact solutions for  $y$  and  $\dot{y}$  in the second order ODE  $\ddot{y} + y = 0$  (SHM) with the boundary conditions  $y(0) = 1$ ,  $\dot{y}(0) = 0$ ? Make a sketch of  $y$  versus  $\dot{y}$  – this known as a phase-plane portrait.

 Show that the equation can be rewritten as a first order vector ODE of the form  $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$ , where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y \\ \dot{y} \end{bmatrix}, \quad \mathbf{f}(\mathbf{y}) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{y}$$

and hence write down the Euler update equation,  $\mathbf{y}_{n+1} = ??$ .

### B. Van der Pohl oscillator

As with other topics in Engineering Computation, the point is that realistic problems in Engineering tend to be too hard to solve on paper. For instance, Simple Harmonic Motion is easy to describe mathematically ( $\ddot{x} + \omega^2 x = 0$ ) and you might think this would be a sensible model for an electronic oscillator. In fact it is highly unrealistic. This is because it is really a special case of damped SHM,


$$\ddot{x} + 2\lambda\dot{x} + \omega^2 x = 0,$$


in which the damping factor  $\lambda$  happens to be exactly  $\lambda = 0$ . In practice, of course, given the tolerances in component values, we can only achieve  $\lambda \approx 0$ . Suppose it turns out that  $\lambda < 0$ : this gives an oscillation whose amplitude grows exponentially as  $\exp|\lambda|t$  so that, even though  $|\lambda|$  may be very small, the amplitude of oscillation will grow and grow, steadily, without bound. This is highly unrealistic, physically.

A real electronic oscillator has to have some form of saturation in its gain so that the amplitude of oscillation is stabilised (see the Horowitz and Hill section on the Wien bridge, section 4.14 in the 1st edition). For instance, the gain of an operational amplifier saturates as the magnitude of its output approaches limits dictated by the power supply voltage. One mathematical model that embodies this automatic stabilisation of gain is the “van der Pohl” oscillator:

$$\ddot{x} + 2\mu(x^2 - A^2/4)\dot{x} + \omega^2 x = 0,$$

in which  $A$  is an amplitude constant and  $\mu > 0$  is a constant that determines how aggressively the gain stabilisation mechanism acts.

 Write the van der Pohl equation as a first order vector ODE by letting  $x_1 = x$  and  $x_2 = \dot{x}$  (note that the matrix coefficients will not be constant with time!).

 When  $\mu = 0$  this equation describes SHM, and hence its solution in the  $x, \dot{x}$  plane is the same as for A above (i.e. a circle). If the initial conditions are  $x(0) = 2, \dot{x}(0) = 0$ , sketch the phase-plane portrait of the solution for  $\mu = 0$ . Now suppose  $\mu$  is non-zero (but small) and  $A = 2$ . Consider (qualitatively) how the damping coefficient  $\lambda = \mu(x^2 - A^2/4)$  varies over the cycle and in particular indicate on your phase-plane diagram where damping will be positive and where it will be negative.

## 2 Lab Work


Copy the functions files `Euler.m` and `SHMF.m` from the directory `/packages/demo/comp/lab4` to a directory called `lab4` in your own area.

### A. Euler simulation of SHM


Try out the function `Euler` with the function `SHMF` to simulate simple harmonic motion,  $\ddot{y} + y = 0$  with initial conditions  $y(0) = 1, \dot{y}(0) = 0$ , by typing

```
>> [ta, Ya] = Euler('SHMF', [0 2*pi], [1 0], 100);
```

Make sure you understand how the `Euler` function works, including the use of `feval` (see `help feval`) to pass the name of the function (e.g. `SHMF`) that describes the differential equation.

 The solution `Ya` has two columns. What does each column represent?

Obtain estimates of  $y(t)$ , using 100, 50 and 25 steps in turn, and superimpose the graphs onto one plot (use `hold on`), together with the exact solution of the differential equation. You can ensure your plots are different colours by using an optional third argument to `plot` (see `help plot`).

 Note down your comments on how error varies as the step-size is altered.

### B. Modified Euler

The lines in `Euler.m`

```
Yadot = feval(FnName, t, Ya(n-1,:));
Ya(n,:) = Ya(n-1,:) + h * Yadot;
```

implement the Euler method, which is of course the first stage in the modified Euler method.


Modify the `Euler` function to produce a new function `EulerM` for the modified Euler method by adding the extra calculation required and saving the new function in `EulerM.m`:

```
Yadot = feval(FnName, t, Ya(n-1,:));
YaEuler = ???; % unmodified Euler 'prediction'
YaEulerdot = ???; % derivative at predicted value
Ya(n,:) = ???; % modified Euler update
```

Check graphically that, for a given sample interval, the modified Euler method is producing more accurate results than the Euler method above.

Errors may be too small now to see clearly how error varies with step size  $h$ . Instead therefore, write a function `EMerror` to determine the  $\infty$ -norm of the absolute error in the estimation of  $y(t)$  solution as a function of step-size. Use the infinity norm over one period of SHM; i.e.  $\epsilon(h) = \max_i |y_h(x_i) - y(x_i)|$ , where  $y_h$  are the Modified Euler estimates of the true solution  $y$ , and  $0 \leq x_i \leq 2\pi$ .

Plot (using `loglog`) the value of for a suitable range values of  $h$  between 0 and 1. [Remember: you can calculate the  $\infty$ -norm of a vector `a` in `matlab` as `norm(a, inf)`.]

 Sketch the graph in your log and work out the the order of convergence of  $\epsilon(h)$ , ie the value of  $M$  in  $\epsilon(h) \sim h^M$  and comment.


Finally, note that **matlab** has a superior version of modified Euler which you might prefer to use henceforward in place of your home-made **EulerM**. It is known as **ode23** and you should try it out as follows:

```
>> [ta,Ya]=ode23('SHMF',[0 2*pi],[1 0]);
```

then plot as usual. Note that you did not need to specify a step size — **ode23** selects it own step-sizes in order to maintain a given level of accuracy (0.1% by default). This means that it samples unevenly. Try

```
>> plot(ta, ta*0, '*','ta', Ya);
```

to see the distribution of samples together with  $y$  and  $\dot{y}$ .

 Note down where the greatest density of samples is used, and attempt to explain this.

### C. Van der Pohl Oscillator

Recall the definition of the van der Pohl oscillator from the preparation, section C. We now investigate the tradeoff mediated by  $\mu$  between rapid stabilisation (large  $\mu$ ) and low harmonic distortion (small  $\mu$ ).

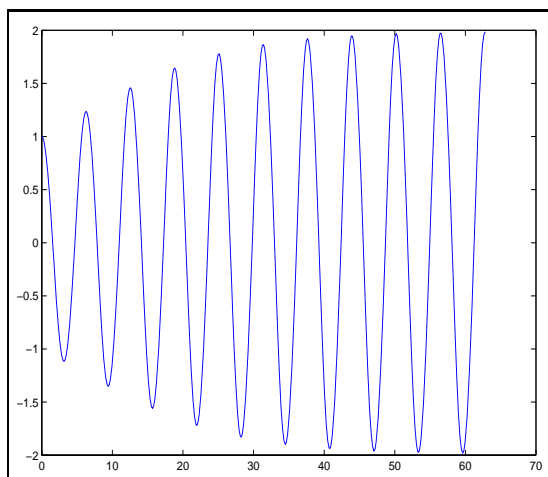
To try out the van der Pohl model, first write a function for use with **matlab ode23**.

Try

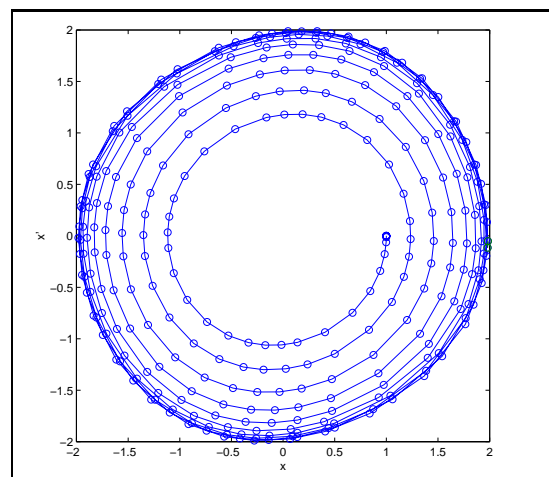
```
>> [ta, Xa] = ode23('vdpol',[0 20*pi],[1 0]);
```

```
>> plot(ta, Xa(:,1));
```

which plots out the oscillating signal. You should see 10 periods of a rapidly stabilising oscillation, as in figure 1(a). You might like to try



(a)




(b)

Figure 1: Simulated time-course of a van der Pohl oscillator, with parameters  $A = 2$  and  $\mu = 0.05$ :  $x, \dot{x}$  versus time (a) and phase-plane portrait (b).

```
>> ode23('vdpol',[0 20*pi],[1 0]);
```

(i.e. with out any output variables) which has the rather nice effect of plotting both the signal and its derivative, as an animation.

Now experiment with various values of parameters, namely  $A = 1, 2$  and  $\mu = 0.1, 0.5$ .

 Sketch the output signals and describe the effect of each of the parameters  $A$ ,  $\mu$  on the amplitude of the signal obtained, and the maximum error compared with an ideal cosine.

To see a clearer view of the rate of amplitude stabilisation and the degree of distortion, the signal can be plotted in the phase-plane with axes  $x, \dot{x}$ . To do this, type the following strange incantation

```
>> options = odeset('OutputFcn','odephas2');
```

(try `help odephas2` if you want to know more about how this works). After setting  $A = 2, \mu = 0.05$  in `vdpol.m`, type

```
>> ode23('vdpol', [0 20*pi], [1 0], options);
```

and the phase-plane portrait will evolve before your eyes. Type


```
>> axis('square');  
>> axis([-2.5 2.5 -2.5 2.5]);
```

to see how the phase-plane trajectory tends towards a circle (approximately), representing an oscillation of stable amplitude, close to a sinusoid, as in figure 1(b). Repeat this yourself also with parameters  $A = 2, \mu = 0.5$  and note the increased distortion and rapid stabilisation of amplitude. In each case, try also an initial value  $x(0) = 4$ , in place of the  $x(0) = 1$  used above.

Type

```
>> [ta,Xa] = ode23('vdpol', [0 2*pi], [2 0]);
```

and use **plot** to display exactly one period of stable oscillation. Note the initial conditions chosen here to jump straight into the steady-state oscillation or “limit cycle”. Superimpose a circle in phase-space, representing a perfect sinusoid of amplitude 2 (i.e. `plot(2*cos(ta), 2*sin(ta))`).

 Compare the result to your prediction of how the damping coefficient varies.