Problem Sheet 0: Numerical integration and Euler's method

If you find any typos/errors in this problem sheet please email jk208@ic.ac.uk.

The material in this problem sheet is not examinable. It is merely designed to illustrate what is meant by "simulating an ODE model" and to give a rough idea of what MATLAB actually does when we use an ode solver command (e.g., ode45(...)).

1. We often cannot solve analytically a nonlinear Ordinary Differential Equation (ODE), that is find a function $x(t)$ that satisfies the ODE: $\dot{x}(t) = f(x(t))$, $x(0) = x_0$. Note that this does not mean that such a function does not exist, it merely means we cannot find an analytical formula for it. Thus, to figure out how the model behaves we must resort to other analytical tools (e.g., finding fixed points, checking the stability of the fixed points, $\ldots$, etc.) or, alternatively, we can *integrate numerically*, *solve numerically* or simply *simulate* the model. This consists in using some algorithm to find discrete approximations of the solutions (that is an approximate solution defined only at certain discrete instances of time). With modern computing power the approximate solutions can often be made more than sufficiently accurate for the purposes that they are required.

   In the lecture notes we claimed that one algorithm that can be used to this end is Euler's numerical integration algorithm. It is the simplest such algorithm and it dates back to Leonhard Euler in the 2^{nd} half of the 18^{th} century. Consider the general model

   $$\dot{x}(t) = f(x(t)), \qquad x(0) = x_0 \tag{1}$$

   where $f : \mathbb{R} \to \mathbb{R}$ is a function and $x_0 \in \mathbb{R}$ is the initial condition. The algorithm yielded from Euler's method for the above model is obtained by making the following approximation:

   $$\dot{x}(t) = \frac{dx(t)}{dt} = \lim_{\delta \to 0} \frac{x(t+\delta) - x(t)}{\delta} \approx \frac{x(t+h) - x(t)}{h},$$

   where h is a small positive parameter called the *step size.*

   (a) Obtain the *difference equation* describing the Euler's method for models of the type (1). In other words, find an expression relating $x(t+h)$ to $x(t)$.

   (b) Consider the affine model

   $$\dot{x}(t) = kx(t) + a, \qquad x(0) = x_0 \tag{2}$$

   where $k \in \mathbb{R}$, $a \in \mathbb{R}$ are parameters. Use your answer to the previous part to show that in the case of affine models, i.e., those of the form (2), Euler's method is described by

   $$x(t+h) = (1 + hk)x(t) + ah. \tag{3}$$

   (c) We can now use MATLAB to solve the difference equation you obtained. We do this by iterating it forward, i.e., plugging in $x(0) = x_0$ (which we know!) to work out $x(h)$, then using $x(h)$ to work out $x(2h)$ $\ldots$, etc. Write a function in MATLAB titled 'function [xnext] = euleraff(x,h,k,a)' such that it takes in $x(ih)$, h, k and a as arguments and returns $x((i+1)h)$ where i denotes any non-negative integer.

   (d) Now write a second function titled 'function [sol,t] = numint(h,k,a,x0,tf)' that takes in h, k, a, x_0 and the final simulation time, t_f, (a positive scalar) and returns two vectors t and sol, where t $= [0, h, 2h, \ldots, nh]$ with n being the greatest integer such that $nh \leq t_f$ and sol $= [x(0), x(h), x(2h), \ldots, x(nh)]$.

(e) We can now just write a quick MATLAB script that calls numint.m, feed it the desired parameter values, the duration of the simulation and a small step size and, in theory, the vector sol should contain a good approximation of $x(0), x(h), x(2h), \ldots, x(nh)$. However how should small should we choose h and how good of an approximation does it give? We can answer these questions by testing different values of h and comparing with the actual analytical solution of (2)[1]. Assuming that $k \neq 0$, use an integrating factor to find the analytic solution of (2). In addition, write a third Matlab function titled 'function [sol,t] = analytic(h,k,a,x0,tf)' such that sol contains the values $x(0), x(h), x(2h), \ldots, x(nh)$ given by the analytical solution evaluated at the discrete-time points $0, h, 2h, \ldots, nh$.

(f) Lastly, we need some way to check how 'close' is the approximation is to the actual answer. Write two final MATLAB functions. The first titled 'function error = AE(soln,sola)' such that it works out the average absolute error of the approximation given by

$$\frac{1}{n}\sum_{i=0}^{n} |\hat{x}(ih) - x(ih)|,$$

where n is defined as in part (d) above, $\hat{x}$ denotes the approximate solution obtained from Euler's method and x denotes the actual solution given by the derived analytical expression. The second function should be titled 'function error = APE(soln,sola)' and it should work out the average percentage error given by,

$$\frac{1}{n}\sum_{i=0}^{n} \frac{|\hat{x}(ih) - x(ih)|}{|x(ih)|},$$

(g) Test out the functions you wrote using them to plot on the same figure both the analytical and approximated solutions for different values of the parameters, initial conditions and simulations time. In particular, try $t_f = 1$, $x_0 = 3$, $h = 0.1$, $k = 1$ and $a = 2$. You should be getting an average absolute error of 0.2387 and an average percentage error of 2.92% (or something close to that).

(h) Use the MATLAB function surf.m (check MATLAB help) to investigate how the average error and average percentage errors change with the simulation time and the step size. In particular, using the parameter values and the initial conditions above vary the step size between 0.001 and 0.1 and the simulation time between 1 and 10. How does the error change?

(i) (Optional) Can you derive the analytic solution to the difference equation (3), that is, solve $x(nh)$ where n is a non-negative integer? *Hint: iterate forward* (3) *from* $x(0) = x_0$ *and use the result regarding geometric series,* $\sum_{j=m}^{n} ar^j = a\frac{r^m - r^{n+1}}{1-r}$.

[1]As previously discussed, most models do not have analytic solutions, however this one is one of the few that does (it was picked specifically for this exercise).

Solutions

1. (a)
$$\frac{x(t+h)-x(t)}{h} = f(x(t)) \Leftrightarrow x(t+h) = x(t) + hf(x(t)).$$

   (b) Plug in $f(x(t)) = kx(t) + a$ to obtain $x(t+h) = x(t) + h(kx(t)+a) = (1+hk)x(t) + ha$.

   (c) Should look like

```
function [xnext] = euleraff(x,h,k,a)

xnext = (1+h*k)*x+a*h;

end
```

   (d) Something like

```
function [sol,t] = numint(h,k,a,x0,tf)

sol(1) = x0; t(1) = 0;

for i = 2:floor(tf/h)+1
    t(i) = (i-1)*h;
    sol(i) = euleraff(sol(i-1),h,k,a);
end

end
```

   (e)
$$\dot{x}(t) = \frac{dx}{dt}(t) = kx(t) + a \Leftrightarrow \frac{dx}{dt}(t) - kx(t) = a \Leftrightarrow \frac{dx}{dt}(t)e^{-kt} - kx(t)e^{-kt} = ae^{-kt}$$
$$\Leftrightarrow \frac{d}{dt}(x(t)e^{-kt}) = ae^{-kt} \Leftrightarrow \int \frac{d}{dt}(x(t)e^{-kt})dt = \int ae^{-kt}dt \Leftrightarrow \boxed{x(t)e^{-kt} = -\frac{a}{k}e^{-kt} + b}$$

   where $b \in \mathbb{R}$ is an integration constant. Thus
$$x(t) = be^{kt} - \frac{a}{k}.$$

   In addition we can solve for b because we know that $x(0) = x_0$. Hence
$$x(0) = x_0 = b - \frac{a}{k} \Leftrightarrow b = x_0 + \frac{a}{k}.$$

   So we get the solution
$$\boxed{x(t) = \left(x_0 + \frac{a}{k}\right)e^{kt} - \frac{a}{k}} \tag{4}$$

   Then, the MATLAB function should look something like

```
function [sol,t] = analytic(h,k,a,x0,tf)

sol(1) = x0; t(1) = 0;

for i = 2:floor(tf/h)+1
    t(i) = (i-1)*h;
    if k ≠0
```

```
        sol(i) = (x0+(a/k))*exp(k*t(i))-(a/k);
    else
        sol(i) = a*t(i)+x0;
    end
end

end
```

(f) Something like the following for the average error

```
function error = ae(soln,sola)

error = 0;

for i = 1:length(soln)
    error = error + abs(soln(i)-sola(i));
end

error = error/length(soln);

end
```

and something like the following for the average percentage error

```
function error = ape(soln,sola)

error = 0;

for i = 1:length(soln)
    error = error + abs(soln(i)-sola(i))/abs(sola(i));
end

error = error/length(soln);

end
```

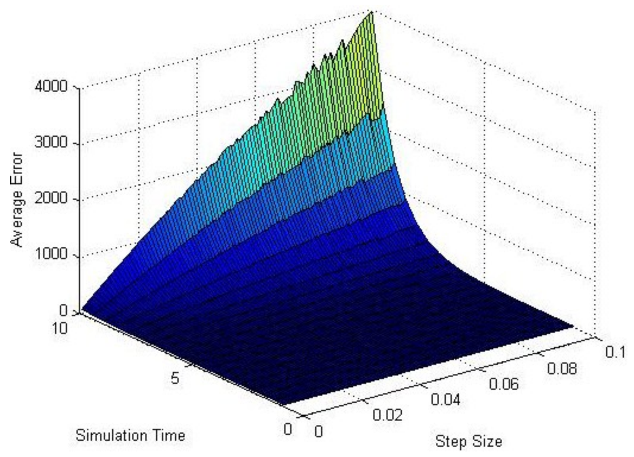
(g) See the following.

(h) Using something along the lines of

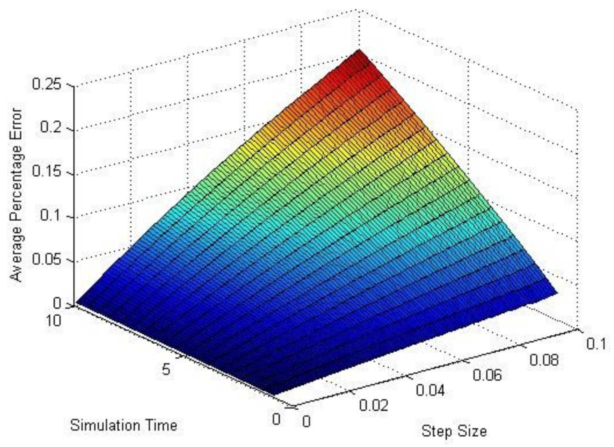
```
tf = 1:0.5:10; x0 = 3; h = 0.001:0.001:0.1; k = 1; a = 2;

error1 = zeros(length(tf),length(h)); error2 = error1;

tic
for i = 1:length(tf)
    for j = 1:length(h)
        [soln,t] = numint(h(j),k,a,x0,tf(i)); [sola,ta] = ...
            analytic(h(j),k,a,x0,tf(i));
        error1(i,j) = ae(soln,sola); error2(i,j) = ape(soln,sola);
        clear soln sola
    end
end
toc

surf(h,tf,error1);
xlabel('Step Size'); ylabel('Simulation Time'); zlabel('Average Error');

figure;
surf(h,tf,error2);
```

```
xlabel('Step Size'); ylabel('Simulation Time'); ...
    zlabel('Average Percentage Error');
```

we obtain



(a) Average absolute error.



(b) Average percentage error.

From the above plots we can note several things. First, the error decreases as the step size decreases. This is not surprising; the algorithm is taking smaller 'jumps' so it is more accurate. Furthermore, if the simulation time is kept constant, the error decreases roughly linearly with the step size. Second, the error increases as the simulation time increases. Earlier errors in the simulation accumulate and 'snowball' as the simulation proceeds making it less and less accurate.

Indeed, one can show that when integrating, using Euler's method, any model of the type $\dot{x} = f(x)$, $x(0) = x_0$, the error will be bounded above by some function of the shape $h\phi(f(\cdot), t_f, x_0)$, in other words, a function that is linear with the step size. For this reason Euler's method is said to be a first order numerical integrator. Unfortunately, as it is in our case (see the average error), $\phi(\cdot)$ can be an unbounded function. This is not very good, especially if we want run long simulations. For example, in our case the error seems to be growing exponentially with the simulation time. Hence if we want to run slightly longer simulations with the same accuracy we will have to decrease the step size by a large amount.

For this reason Euler's method is actually not a very good numerical integrator. There exists much better integration routines in the sense that the reduction in error per reduction in step size is defined by terms of higher orders. For example the integration error of the famous 4^{th} order Runge-Kutta algorithm mentioned in the notes is proportional to h^4 (that is why it is said to be of 4^{th} order). Hence it is considered much better than Euler's because you get a lot more "bang for your buck" in terms of reducing the error by reducing the step size. For instance, in Euler's method a tenfold reduction in step size lead to a tenfold reduction in error, however in the 4^{th} order Runge-Kutta a tenfold reduction in step size leads to a $10,000$-fold reduction in error.

The above said, with modern computing power Euler's method can still often be successfully be used to numerically solve a model. In particular, even for the longest running time of 10 and using a step size of just 4 orders of magnitude less than the simulation time ($h = 0.001$), on average, the approximation was only 0.25% off from the actual value.

(i)

$$x(0) = x_0, \quad x(h) = (1+hk)x_0 + ha, \quad x(2h) = (1+hk)x(1) + ha = (1+hk)^2 x_0 + (1+hk)ha + ha, \quad \ldots,$$

$$x(nh) = (1+hk)^n x_0 + \sum_{i=0}^{n-1} ha(1+hk)^i = (1+hk)^n x_0 + ha\frac{1-(1+hk)^n}{1-(1+hk)} = (1+hk)^n x_0 - \frac{a}{k}(1-(1+hk)^n)$$

$$\Leftrightarrow x(nh) = \left(x_0 + \frac{a}{k}\right)(1+hk)^n - \frac{a}{k}.$$

Note that, just as with ODEs, it is not possible to solve analytically the vast majority of difference equations. In the above case we can do it because the difference equation is affine, but it is a rare exception. However, generally there is absolutely no problem with solving them numerically on a computer, just as we did with numint.m to solve (3).