MA50174 Advanced Numerical Methods

Assignment 3 Initial value ordinary differential equations

MATLAB has several facilities for finding the numerical solution of initial value differential equation problems. In this assignment we will look at both the inbuilt MATLAB routines and also some other routines, for both stiff and non-stiff problems.

Non-stiff problems

We start the assignment by looking at the performance of some integrators on non-stiff initial value ordinary differential equations. These are problems for which all of the components evolve simultaneously on comparable time-scales. Non-stiff problems are often solved by using *explicit* methods usually with some error control. The MATLAB routine `ode45` is such a routine and we will look at this, together with the Euler method and the geometrically motivated symplectic methods. We will investigate the problem of the angle of swing θ of a *simple pendulum* of unit length, which in a unit gravitational field, without damping, satisfies the differential equation

$$\frac{d^2\theta}{dt^2} + \sin(\theta) = 0, \quad \theta(0) = \alpha < \pi, \quad \frac{d\theta}{dt}(0) = 0. \tag{1}$$

1. Show (analytically) that the energy $E(t)$ defined by

$$E(t) = \frac{1}{2}\left(\frac{d\theta}{dt}\right)^2 - \cos(\theta), \tag{2}$$

is a constant of the motion, and evaluate this constant. [1]

(b) Express the pendulum equation with the initial conditions given in (1), as a vector differential equation in the form

$$d\mathbf{u}/dt = \mathbf{f}(\mathbf{u}), \quad \mathbf{u}(0) = \mathbf{u_0}. \tag{3}$$

where $\mathbf{u}(t) = (u_1(t), u_2(t))^T$, and u_1, u_2 are suitably chosen. This differential equation can be solved by using the MATLAB command `ode45`. This routine integrates (3) by using an embedded Runge-Kutta routine. This is a variable step size method which chooses the step size at each step so that the estimated error made at each such step is less than a given (default or user specified) tolerance. Using the default parameters, calculate and plot the solution in the form $(\theta, d\theta/dt)$ in the case. $\alpha = 1, \quad 0 \le t \le 100$. [1]

(c) As a measure of the accuracy of the solution we can look at the numerical values of the error $err(t) = |E(t) - E(0)|$. Calculate and plot (on the same graph) the numerical approximation to $err(t)$ for the problem in (b), setting both the absolute and relative tolerance of the `ode45` routine firstly to 10^{-6} and then to 10^{-8}. Comment on, and give a brief explanation of your results. [1]

+(d) Setting $\alpha = 0.99\pi, \quad 0 \le t \le 100$, solve the ODE using the default tolerances and plot $(\theta, d\theta/dt)$. By considering the level curves of the function $E(t)$ and the results in *(c)* comment on, and explain, your answers. [2]

2. *(a)* The pendulum equation (1) is an example of a Hamiltonian system. This can be solved by using a *symplectic integrator* with a constant time step $h \ll 1$. Such integrators preserve area in phase space and have excellent properties when used to solve equations over long time intervals. Suppose that U_1^n, U_2^n and E^n are the numerical approximations of $u_1(t), u_2(t)$ and $E(t)$ at the time $t = (n-1)h$. Two explicit symplectic integrators are:

$$\text{Symplectic-Euler:} \quad U_1^{n+1} = U_1^n + hU_2^n, \quad U_2^{n+1} = U_2^n - h\sin(U_1^{n+1}),$$

$$\text{Störmer-Verlet:} \quad U_1^* = U_1^n + \frac{h}{2}U_2^n, \quad U_2^{n+1} = U_2^n - h\sin(U_1^*), \quad U_1^{n+1} = U_1^* + \frac{h}{2}U_2^{n+1}.$$

Write a MATLAB routine which implements (on request) either the *forward Euler* method, the *Symplectic Euler* method or the *Störmer-Verlet* method for solving the pendulum equation (3) with a requested h and α over a time interval $[0, T]$. This routine should generate sequences U_1^n, U_2^n and E^n. [1]

(b) Apply this routine in the case of $T = 100$, $h = 0.1$ and $\alpha = 0.25$, comparing the results of the forward Euler, symplectic Euler and Störmer-Verlet method by plotting the points (U_1^n, U_2^n). Which do you think is the better method for this problem and why? [1]

(c) Suppose that α and hence U_1 and U_2 are *small*. By using a suitable approximation of the functions in (3), find a matrix A so that if $\mathbf{U^n} = (U_1^n, U_2^n)^T$ are the solutions of the Forward Euler method, then

$$\mathbf{U}^{n+1} = A\mathbf{U}^n.$$

Hence, or otherwise, show that in this case

$$\left((U_1^{n+1})^2 + (U_2^{n+1})^2\right) = \left(1 + h^2\right)\left((U_1^n)^2 + (U_2^n)^2\right).$$

[1]

+(d) Find a similar matrix A for the small solutions of the Symplectic Euler method. Hence, or otherwise, show that if (U_1^n, U_2^n) is a small solution of the Symplectic Euler method then there is a constant C so that

$$(U_1^n)^2 + hU_1^nU_2^n + (U_2^n)^2 = C.$$

Using this result and that in (c) give a brief explanation of the results of your computations in *(b)*. [2]

Stiff ordinary differential equations

We now look at differential equations which have widely differing timescales. These are generally called *stiff* differential equations. When solving a stiff equation a numerical method has a step size which is restricted *by considerations of stability rather than accuracy*. MATLAB uses the routines `ode23t` and `ode15s` to solve stiff problems. Both of these routines usually involve solving nonlinear equations. Stiff equations are very common in applications; examples arise in the study of chemical reactions. We start with some theoretical investigations before applying the results we have derived to a problem arising in chemical engineering.

3. Suppose that $\mathbf{u}$ satisfies the differential equation

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}) \equiv \Lambda\mathbf{u}, \qquad \Lambda = \begin{pmatrix} -100 & 1 \\ 0 & -1 \end{pmatrix} \quad \mathbf{u}(0) = (2, 2)^T.$$

(a) Show, by differentiating, that the solution of this system is given by

$$\mathbf{u}(t) = ae^{-100t}\begin{pmatrix} 1 \\ 0 \end{pmatrix} + be^{-t}\begin{pmatrix} 1/99 \\ 1 \end{pmatrix}$$

for constants a, b which you should determine by setting $t = 0$. [1]
(b) Show that if the Forward Euler method is used to solve this differential equation, with $\mathbf{U^n}$ approximating $\mathbf{u}((n-1)h)$ then

$$\mathbf{U}^n = (1 + h\Lambda)^{n-1}\,\mathbf{U}^1.$$

Let $A_h = I + h\Lambda$. Find the eigenvalues of A_h. Hence show that $\mathbf{U^n}$ will grow without bound if $h > 2/100$. This method is *unstable* if $h > 2/100 = 0.02$ and this imposes a restriction on the step size that we can use. [1]

(c) The *absolute truncation error* $\mathcal{E}$ that is made at each step of the Forward Euler method between t and $t + h$ is given by

$$\mathcal{E} = \frac{h^2}{2}|\mathbf{u}''(\xi)|, \quad \xi \in [t, t+h].$$

The relative error R is given by $R = \mathcal{E}/|\mathbf{u}|$. If we want a relative error of 0.5×10^{-2} at each step when calculating the solution, show that we could *in principle* use a step size of $h = 0.1$ in the latter part of the calculation. What step size do we in principle need to use in the initial stages of the calculation? [1]

This shows that in the initial stages of the calculation (when $\mathbf{u}$ is varying rapidly), the step size is restricted by considerations of *accuracy* and in the latter stages by consideration of *stability*. To avoid the growth of small errors in the latter stages of the calculation the method is working too hard. This is the hallmark of using a non-stiff method on a stiff problem. The routine `ode45` behaves in a very similar manner.

4. The *Trapezoidal rule* is widely used to solve stiff ordinary differential equations. It combines stability with accuracy and ease of use. It is also a *symmetric* and *self-adjoint* method which makes it especially suitable for solving differential equations which remain unchanged when time is reversed. MATLAB has a trapezoidal rule method `ode23t`. The Trapezoidal rule is defined by

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \frac{h}{2}\left(\mathbf{f}(\mathbf{U^n}) + \mathbf{f}(\mathbf{U^{n+1}})\right).$$

If $\mathbf{f}(\mathbf{u})$ is a nonlinear function, then at each step of the Trapezoidal method a nonlinear system must be solved.

(a) Show that for the problem in Q.4. we have

$$\mathbf{U}^{n+1} = B_h\mathbf{U}^n,$$

for some matrix B_h which you should identify. Hence show that this method is *stable* for *all* (positive) h i.e. $\mathbf{U}_n$ does not grow without bound. [1]

(b) The absolute truncation error at each stage of the trapezoidal rule is given by

$$\mathcal{E} = \frac{h^3}{12}|\mathbf{u}'''(\xi)|, \quad \xi \in [t, t+h].$$

Find choices of step size h for the initial and final stages of the calculation so that the relative error $R = E/|\mathbf{u}|$ is less than 10^{-2}. Compare these with the step sizes for the Forward Euler method and comment. [1]

5. The HIRES problem is a stiff system of 8 nonlinear ordinary differential equations. HIRES stands for the High Irradiation Responses of photomorphogenesis on the basis of phytochrome, by means of a chemical reaction involving eight reactants. The concentration of the reactants is given by the vector $u(i)$ with $i = 1, 2, .., 8$ and each differential equation describes a chemical reaction as part of an overall process which is influenced by the presence of enzymes. It is a typical example of the sort of problems that arise in chemical engineering and/or biochemistry. Details of the problem are given in:

E. Schäfer, *A new approach to explain the 'high irradiance responses' of photomorphogenesis on the basis of phytochrome*, J. Math. Biology, **2**, (1975), 41–56.

We now integrate this system, using two routines, the explicit, non-stiff routine `ode45` which behaves like the Forward Euler method (in that this is an explicit method in which the step size is chosen in a similar manner to that of the Forward Euler method and is restricted by considerations of stability) and the variable order stiff solver `ode15s` which behaves like the trapezoidal rule (in that this is an implicit method which requires the solution of nonlinear equations at each step and in which the step size is determined by considerations of accuracy rather than stability). The `ode15s` algorithm is a very powerful routine which should be used for most stiff problems.

The HIRES system takes the form

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}), \quad \mathbf{u}(0) = \mathbf{u}_0.$$

Here

$$\mathbf{f}(\mathbf{u}) = \begin{pmatrix} -1.71u_1 + 0.43u_2 + 8.32u_3 + 0.0007 \\ 1.71u_1 - 8.75u_2 \\ -10.03u_3 + 0.43u_4 + 0.035u_5 \\ 8.32u_2 + 1.71u_3 - 1.12u_4 \\ -1.745u_5 + 0.43u_6 + 0.43u_7 \\ -280u_6u_8 + 0.69u_4 + 1.71u_5 - 0.43u_6 + 0.69u_7 \\ 280u_6u_8 - 1.81u_7 \\ -280u_6u_8 + 1.81u_7 \end{pmatrix}$$

and the initial values are given by $\mathbf{u}_0 = (1, 0, 0, 0, 0, 0, 0, 0.0057)^T$.

(a) (i) Write a MATLAB script file which integrates this problem over the range `t=[0 321.8122]` using either `ode45` or `ode15s` with the default parameters. Calculate the solution `[t,u]` in each case (you will have to wait for the `ode45` calculation) and plot (on the same graph) $\log(h(i))$ as a function of $t(i)$ in both cases, where h the vector of time-steps $h(i) = t(i+1) - t(i)$. [1]

(ii) Modify the script file so that it uses the `options = odeset(..)` routine to set up a relative tolerance `'RelTol'` equal to $10^{-(4+m/4)}$ for $m = 0, 1, 2, .., 24$ and an absolute tolerance `'AbsTol'` equal to `'RelTol'`. For each of these values of m determine the *cputime* that it takes to run the calculation when using either `ode45` or `ode15s`. (See `help cputime` for details of this command). Plot (separate) graphs in each of these two cases of the cputime as a function of m. [1]

+(b) Using the results of Q.4 and Q.5 give a careful qualitative and quantitative explanation of the form of each of the graphs in *(a)(i)* and *(a)(ii)*, comparing `ode45` and `ode15s`, and giving a mathematical justification where necessary. [3]

CJB