Selected Solutions for Exercises in

Numerical Methods with MATLAB: Implementations and Applications

Gerald W. Recktenwald

Chapter 12

Numerical Integration of Ordinary Differential Equations

12.3 Manually perform three steps of Euler's method to solve

$$\frac{dy}{dt} = \frac{1}{t+y+1}, \qquad y(0) = 0,$$

with $h = 0.2$.

Partial Solution: Euler's method predicts $y(0.6) = 0.4576112..$ The exact solution is $y(0.6) = 0.426493$.

$\diamondsuit$

12.7 (Stoer and Bulirsch [70]) Use Euler's method with $h = 0.05$ to solve

$$\frac{dy}{dt} = \sqrt{y}, \qquad y(0) = 0, \qquad 0 \le t \le 2.$$

Plot a comparison of the numerical solution with the exact solution. Does the plot indicate that there is an error in `odeEuler`? If there is no error in `odeEuler`, can you explain the peculiar results? Recompute the solution with `odeMidpt` and `odeRK4`. (*Hint*: What happens if the initial condition $y(0) = \varepsilon_m$ is used instead of $y(0) = 0$?)

Solution: The exact solution is $y = (t/2)^2$. Evaluating the numerical solution gives $y_j = 0$ for any h. This result is not due to an error in `odeEuler`. The behavior of Euler's method is the same as the behavior of any one-step method for this ODE: *all* one-step numerical solutions are $y_j(t_j) = 0$ regardless of step size.

The general one-step formula is (cf. Equation (12.23))

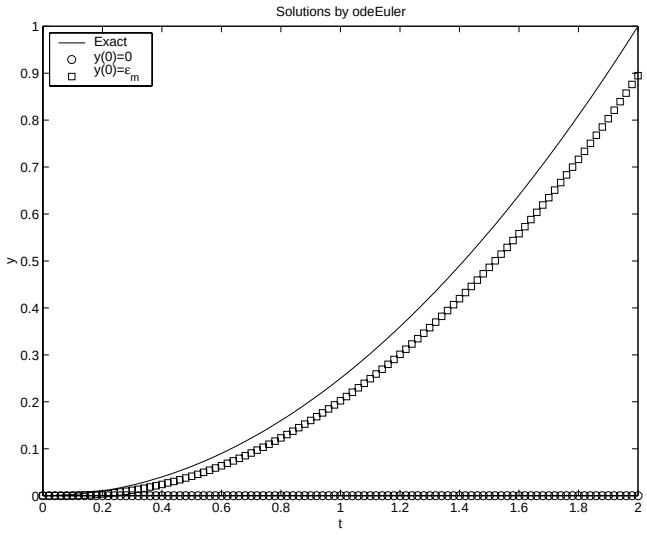
$$y_j = y_{j-1} + h\Phi(t, y, h, f)$$

For this ODE, $\Phi(t, y, h, f) = 0$ at $t = 0$ for any h, so that $y_2 = 0$ (the first step) for any one-step method. Furthermore, $\Phi(t, y, h, f) = 0$ whenever $y = 0$ so the numerical solution will remain stuck at $y_j = 0$ for all j. The trick to solving this problem is to perturb the initial condition, for example, by using $y(0) = \varepsilon_m$ instead of $y(0) = 0$. By intentially introducing this neglible error in the initial condition, the numerical solution will produce $\Phi(t, y, h, f) \neq 0$ for the first and subsequent time steps.

The `odeEulerSqrt` function performs two numerical integrations of the ODE using Euler's method. One numerical solution is with $y(0) = 0$ and the other is with $y(0) = \varepsilon_m$.

```
function demoEulerSqrt
% demoEulerSqrt  Solve y' = sqrt(y),  y(0) = 0;  Exact solution:  y = (t/2)^2

% reference:  Bulirsh and Stoer, Chapter 7, exericse 4

diffeq = inline('sqrt(y)','t','y');
tn = 2;  y0 = 0;

% --- Two solutions differing by the value of y(0)
[t1,y1] = odeEuler(diffeq,tn,tn/100,y0);
[t2,y2] = odeEuler(diffeq,tn,tn/100,y0+eps);
% --- Evaluate exact solution and plot comparison
ye = (t2/2).^2;
plot(t2,ye,'-',t1,y1,'o',t2,y2,'s');
legend('Exact','y(0)=0','y(0)=\epsilon_m',2);
title('Solutions by odeEuler');   xlabel('t');   ylabel('y');
```

Running `odeEulerSqrt` produces the plot on the next page.



Plot of solution to **Exercise 12–7**.

12.10 Using `odeEuler.m` as a guide, write an m-file to implement Heun's method for an arbitrary, first-order ODE. Use your function to solve equation (12.11) for $h = 0.2$, $h = 0.1$, $h = 0.05$, and $h = 0.025$. Compare the global discretization error of your program with the theoretical prediction of the global discretization error for Heun's method.

Partial Solution: The `odeHeun` function is obtained by making small modifications to either the `odeEuler` or `odeMidpt` function. Once `odeHeun` is written, it can be called (for example) by `demoHeun` which is a trivial modification of the `demoEuler` function. The partial output of running `demoHeun` for $h = 0.1$, $h = 0.05$, and $h = 0.025$ is

```
Max error =   3.58e-03 for h = 0.100000
Max error =   8.27e-04 for h = 0.050000
Max error =   1.99e-04 for h = 0.025000
```

12.12 For each of the following initial value problems, verify that the given $y(t)$ is the solution.

(a) $\dfrac{dy}{dt} = \dfrac{t^2}{\alpha} - y; \quad y(0) = 1; \quad \Longrightarrow \quad y(t) = \dfrac{1}{\alpha}\left[t^2 - 2t + 2 + (\alpha - 2)e^{-t}\right]$

Solution (a): The easiest way to verify that the proposed solution is correct is by direct substitution into the initial value problem. The proposed solution is correct only if it satisfies the ODE *and* the initial condition.

Begin by evaluating d/dt of the proposed analytical solution.

$$\frac{d}{dt}\left\{\frac{1}{\alpha}\left[t^2 - 2t + 2 + (\alpha - 2)e^{-t}\right]\right\} = \frac{1}{\alpha}\left[2t - 2 - (\alpha - 2)e^{-t}\right] \qquad (\star)$$

Next, substitute the proposed analytical solution into the right hand side of the ODE

$$\frac{t^2}{\alpha} - y = \frac{t^2}{\alpha} - \frac{1}{\alpha}\left[t^2 - 2t + 2 + (\alpha - 2)e^{-t}\right] = \frac{1}{\alpha}\left[2t - 2 - (\alpha - 2)e^{-t}\right] \qquad (\star\star)$$

Comparing the far right hand side of Equation ($\star$) with the far right hand side of Equation ($\star\star$) we see that the proposed solution satisfies the ODE.

Now, verify the initial condition.

$$y(0) = \frac{1}{\alpha}\left[0^2 - (2)(0) + 2 + (\alpha - 2)e^0\right] = \frac{1}{\alpha}\left[0 - 0 + 2 + \alpha - 2\right] = \frac{\alpha}{\alpha} = 1$$

Therefore, since the proposed solution satisfies both the differential equation an the initial condition, it is a solution to the initial value problem. Since the initial value problem is linear, the solution is unique.

12.15 Repeat Exercise 13 using the built-in `ode23` method and `ode45` methods to solve the equations. Do not attempt to run the sequence of decreasing h values. (Why not?) Instead, compare the solutions from `ode23` and `ode45` using the default convergence parameters.

Solution (a): Use `ode23` and `ode45` to solve

$$\frac{dy}{dt} = \frac{t^2}{\alpha} - y; \quad y(0) = 1;$$

with $\alpha = 3$, $y_0 = 1$, $t_n = 2$. The exact solution is

$$y(t) = \frac{1}{3}\left[t^2 - 2t + 2 + e^{-t}\right]$$

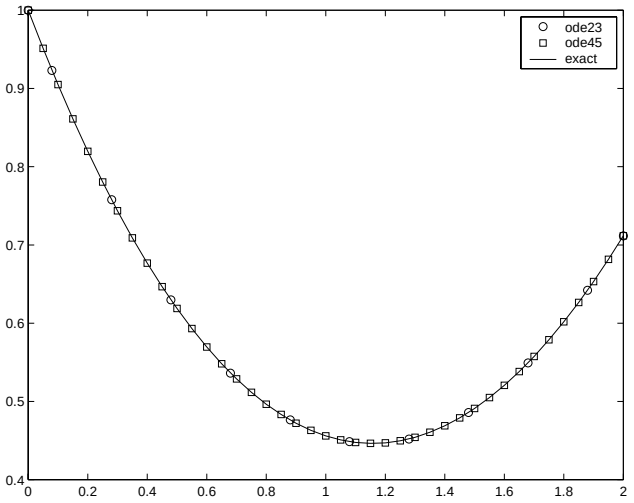
The `demoOde2345_1a` uses `ode23` and `ode45` to obtain the numerical solutions.

```
function demoOde2345_1a
% demoOde2345_1a  Integrate dy/dt = (t^2)/3 - y with ode23 and ode45
%
% Synopsis:   demoOde2345_1a
%
% Input:     none
%
% Output:    A table comparing the numerical and exact solutions

rhs = inline('(t.^2)/3 - y','t','y');               %  rhs of ODE
exact = inline('(t.^2 - 2*t +2 + exp(-t))/3','t');  %  exact solution

tn = 2;  y0 = 1;                  %  stopping time and IC
[t23,y23] = ode23(rhs,tn,1);      %  solution with ode23
[t45,y45] = ode45(rhs,tn,1);      %  solution with ode45
yex = exact(t45);                 %  Exact solution
plot(t23,y23,'o',t45,y45,'s',t45,yex,'-');
fprintf('\nMax error = %10.2e for ode23\n',norm(y23-exact(t23),inf));
fprintf('Max error = %10.2e for ode45\n',norm(y45-yex,inf));
legend('ode23','ode45','exact');
```

Running `odeEulerSqrt` produces

```
>> demoOde2345_1a

Max error =   2.41e-04 for ode23
Max error =   1.97e-07 for ode45
```

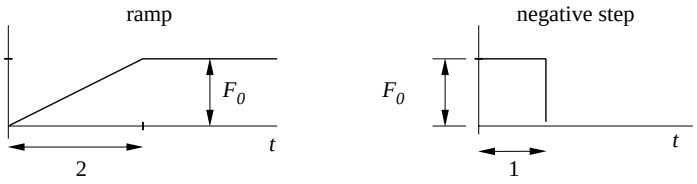
and the plot on the next page. The plot of the solutions by `ode23` and `ode45` appear to be in good agreement with the exact solution. A quantitative comparision of the methods shows that `ode45` produces a more accurate result. The maximum error (which is the same as the global discretization error) produced by `ode45` is much smaller than the maximum error produced by `ode23`.



Plot of solution to **Exercise 12–15**.

◊

12.25 Create modified version of the `rhsSmd` function (see Listing 12.13 on page 723) to implement the following forcing functions for the second order spring–mass–damper system of Example 12.12.



(a) Sinusoidal force input. $F(t) = F_0 \sin(\omega t)$, where $\omega \neq \omega_n$.

(b) Ramp force input from 0 to 2 seconds, then constant at F_0.

(c) Constant force of F_0 for a duration of 1 second, then zero force.

Compare the system responses for $F_0 = 1$. *Hint*: Write a separate m-file function to evaluate each $F(t)$ for parts (a), (b), and (c). A particular $F(t)$ function is then chosen at run-time, not by repeated editing of `rhsSmd`. One way to do this is to pass the *name* of the chosen $F(t)$ m-file through `ode45` to the modified `rhsSmd` function, where the $F(t)$ function is then called with the built-in `feval` function. In this implementation, the name of the $F(t)$ function should also be an input to the modified `demoSmd` function.

Partial Solution: The modified `rhsSmd` function is called `rhsSmdVarInput`. The compatible m-file to evaluate step input forcing function is called `stepFun`. Both m-files are listed on the following page

```
function dydt = rhsSmdVarInput(t,y,flag,zeta,omegan,inFun,u0,tin)
% rhsSmdVarInput  RHS of coupled ODEs for a spring-mass-damper system
%                 Variable forcing functions are specified on input
%
% Synopsis:  dydt = rhsSmdVarInput(t,x,flag,zeta,omegan,a0)
%
% Input:  t      = time, the independent variable
%         y      = vector (length 2) of dependent variables
%                  y(1) = displacement and y(2) = velocity
%         flag   = dummy argument for compatibility with ode45
%         zeta   = damping ratio (dimensionless)
%         omegan = natural frequency (rad/s)
%         inFun  = (string) name of m-file to evaluate forcing function
%                  Currently allowed functions are 'stepFun', 'rampFun'
%                  'stepDownFun', and 'sineFun'
%         u0     = magnitude of force input function
%         tin    = time scale for force input.  Meaning of tin
%                  depends on value of inFun:
%
%                  inFun          meaning of tin
%                  -----       ----------------------------
%                 stepFun      Time at start of step (tin = 0, usually)
%                 rampFun      Time at end of ramp, input is constant for t>tin
%                 stepDownFun  Time at which input is set to zero.
%                              For 0 <= t <= tin, input is u0
%                 sineFun      Time units *per radian* of oscillatory input.
%                              f = u0*sin(t/tin)
%
% Output:   dydt = column vector of dy(i)/dt values

f = feval(inFun,t,u0,tin);
dydt = [ y(2);  f - 2*zeta*omegan*y(2) - omegan*omegan*y(1)];
```

```
function f = stepFun(t,u0,tin)
% stepFun  Step input in force

if t<=tin,  f = 0.0;   %  zero input before the step
else,       f = u0;    %  step input
end
```

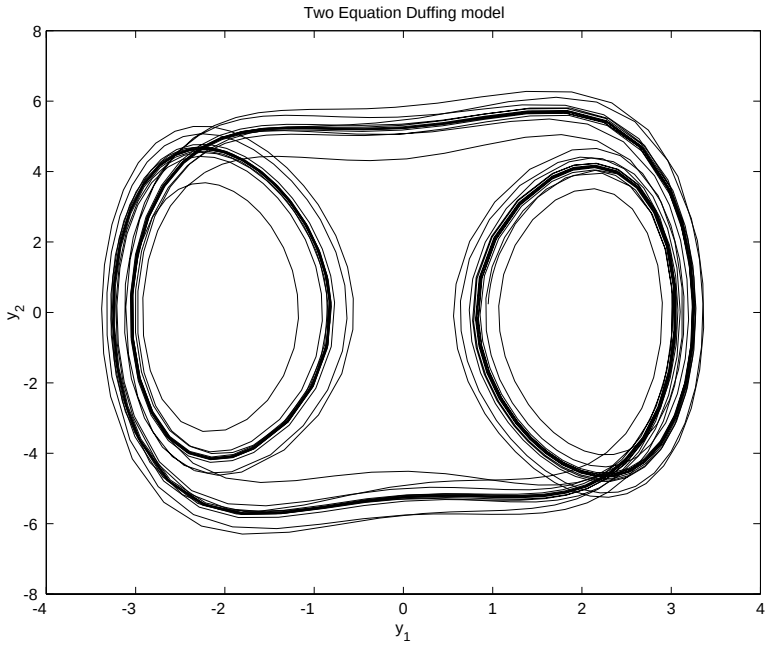
12.27 Duffing's equation

$$\frac{d^2x}{dt^2} + kx + x^3 = B\cos t$$

describes the chaotic dynamics of a circuit with a nonlinear inductor. (See, for example, Moon [57]). Convert this equation to a system of two first-order ODEs, and solve the system for $k = 0.1$ and $B = 12$ and $0 \le t \le 100$. Create the *Poincaré* map of the system by plotting y_2 versus y_1.

Typographical Error: In the first printing of the book, the problem statement incorrectly asserts that the Poincaré map is obtained by plotting dx/dt versus t. Instead, the Poincaré map is a plot of the state variables against each other with time as the parameter. For this problem the map is obtained by plotting dx/dt versus x.

Partial Solution: The Poincaré map is shown below



Plot of solution to **Exercise 12–27**.

◊