Table 4.6 Listing of the M-file *misstimv.m*

misstimv.m

```
function [A,B,C,D]=misstimv(t);
% Linear, time-varying state coefficient matrices for a missile
% x = [u v w p q r]';
% A= state dynamics matrix; B= state input coefficient matrix
% C= state output coefficient matrix; D= direct transmission
matrix
% t= time after launch
% copyright(c)2000 by Ashish Tewari
% Thrust as a function of time:-
if t<=20
     Th=1.2717e+005;
else
     Th=0;
end
% inertia properties as functions of time:-
m=983.6716-29.4208*t-0.5812*t^2+0.0338*t^3;
iy=1000*(2.5475-0.0942*t+0.0022*t^2);ix=iy/10;
xcg=3.6356-0.0217*t-0.0008*t^2;
% aerodynamic and propulsive properties as functions of time:-
Zw=-(246.44+21.9038*t-1.5996*t*t+0.0244*t^3);
Mw=-(872.95-52.7448*t-0.0006*t^2+0.0368*t^3);
Mq=(-3-1.39*t+0.08*t^2)/10;
Lr=0.0134+0.0029*t-0.0001*t^2;
Lp=-0.0672-0.0143*t+0.0006*t^2;
Lv=-0.1159-0.0317*t+0.0015*t^2;
Zu=-9.5383-2.592*t+0.1209*t^2-0.0011*t^3;
Xw=1.9067+0.5186*t-0.0242*t^2+0.0002*t^3;
Md=1e5*(1.3425-2.3946*t+0.1278*t^2-0.0017*t^3);
Zd=1e4*(-2.0143-3.6649*t+0.1854*t^2-0.0023*t^3);
Yv=Zw;Nv=Mw;Nr=Mq;Np=Lp/10;Xu=Th/400+Zu/10;ci=(iy-ix)/iy;
% the state coefficient matrices:-
A=[Xu/m 0 Xw/m 0 0 0;0 Yv/m 0 0 0 0;Zu/m 0 Zw/m 0 0 0;
  0 Lv/ix 0 Lp/ix 0 Lr/ix;0 0 Mw/iy 0 Mq/iy 0;
  0 Nv/iy 0 Np/iy 0 Nr/iy];
B=[0 0 0;Th/m 0 0;0 Th/m 0;0 0 1;0 Th*xcg/iy 0;Th*xcg/iy 0 0];
C=eye(6);D=zeros(6,3);
```

4.6 Numerical Solution of Nonlinear State-Equations

The nonlinear state-equations are the most difficult to solve. As for time-varying systems, there is no analytical solution for a set of general nonlinear state-equations. There are several numerical schemes available for solving a set of nonlinear, first-order differential equations using the digital approximation to the continuous-time differential equations. Since the state-equations of a nonlinear system are also a set of nonlinear, first-order differential equations, we can use such numerical schemes to solve the state-equations of nonlinear systems. Due to the nonlinear nature of the differential equations, the solution procedure often is more complicated than merely marching forward in time, as we did

for linear systems in the previous two sections. Instead, an *iterative* solution procedure may be required at each time step, which means that we assume a starting solution, and then go back and keep on changing the assumed solution until the solution *converges* (i.e. stops changing appreciably with the steps of the iteration).

A general set of nonlinear state-equations can be expressed as follows:

$$\mathbf{x}^{(1)}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t) \tag{4.84}$$

where $\mathbf{x}(t)$ is the state-vector, $\mathbf{u}(t)$ is the input vector, and $f(\mathbf{x}(t), \mathbf{u}(t), t)$ denotes a nonlinear function involving the state variables, the inputs, and time, t. The solution, $\mathbf{x}(t)$, of Eq. (4.84) with the initial condition, $\mathbf{x}(t_0) = \mathbf{x}_0$ may not always exist. The existence of solution of nonlinear differential equations requires that the nonlinear function, $f(\mathbf{x}(t), \mathbf{u}(t), t)$, should be defined and *continuous* for all *finite* times, $t \geq t_0$. Also, it is required that $f(\mathbf{x}(t), \mathbf{u}(t), t)$ must satisfy the following condition, known as the *Lipschitz condition*:

$$|f(\mathbf{x}(t), \mathbf{u}(t), t) - f(\mathbf{x}^*(t), \mathbf{u}(t), t)| \leq K|\mathbf{x}(t) - \mathbf{x}^*(t)| \tag{4.85}$$

where $\mathbf{x}^*(t)$ is a vector different from $\mathbf{x}(t)$, K is a constant, and $|\mathbf{V}|$ denotes a vector consisting of the absolute value of each element of the vector $\mathbf{V}$. For greater details on the existence of solution of ordinary nonlinear differential equations see a textbook on ordinary differential equations, such as Henrici [4]. In this book, we will assume that we are dealing with nonlinear system which have a solution to their state-equations. Owing to the nonlinear nature of the differential equations, the numerical procedure cannot be a one-shot (i.e. an open-loop) process, such as that used for linear differential equations. Instead, an *iterative* solution procedure is required for nonlinear systems, which consists of repeatedly evaluating the solution in a *loop* (such as the feedback loop) at each time step, until the solution meets certain desirable conditions. Hence, a nonlinear solution procedure itself is a *closed-loop system.*

The digital solution procedures for Eq. (4.84) can be divided into *single-step methods*, *multi-step methods*, and *hybrid methods*. The single-step methods obtain the approximate solution vector by using the state-vector and input vector *only at the previous time step.* The time marching solution methods of the previous two sections are single-step methods for linear state-equations. For nonlinear systems, examples of single-step methods are *Runge–Kutta* and *Adams* methods. The multi-step methods use information from *more than one* previous time steps to obtain the approximate solution at a given time step. The *predictor-corrector* methods are examples of multi-step methods. The *hybrid methods* are those that either do not fall into the categories of single- and multi-step methods, or those that use information from previous time steps as well as future (extrapolated) time steps. The *Euler method* falls in the hybrid category. For more information on the numerical solution methods for nonlinear differential equations see Ralston and Rabinowitz [5].

While choosing which method to use for solving a nonlinear set of differential equations one should consider *numerical accuracy* (how large is the digital approximation error), *efficiency* (how fast is the algorithm when implemented on a computer), *numerical stability* (whether the algorithm *converges* to a solution), and *starting problem* (how the algorithm can be started). While the multi-step and hybrid methods offer a greater efficiency for a comparable accuracy than the single-step methods, they are usually very difficult to

start, and special attention must be paid in changing the time steps to avoid stability problems. Such issues make multi-step or hybrid methods more complicated than the single-step methods. Complexity of an algorithm often results in a reduced efficiency when implemented in a computer program. A single-step method which is simple to implement, and which provides good accuracy in a wide variety of problems is the *Runge–Kutta* method. The Runge–Kutta method uses the following digital approximation to Eq. (4.84):

$$\mathbf{x}(t_n) - \mathbf{x}(t_{n-1}) = \sum_{i=1}^{p} w_i \mathbf{k}_i \tag{4.86}$$

where t_n and t_{n-1} are the nth and $(n-1)$th time steps, respectively, w_i are constants, and

$$\mathbf{k}_i = \Delta t_n f(\mathbf{x}(t_{n-1}) + \sum_{j=1}^{i=1} \beta_{ij} \mathbf{k}_j, \mathbf{u}(t_{n-1}), t_{n-1} + \alpha_i \Delta t_n) \tag{4.87}$$

where $\Delta t_n = t_n - t_{n-1}$, α_i and β_{ij} are constants, with $\alpha_1 = 0$. The time step size, Δt_n, can be variable. The constants α_i and β_{ij} are evaluated by equating the right-hand side of Eq. (4.86), with the following Taylor series expansion:

$$\mathbf{x}(t_n) - \mathbf{x}(t_{n-1}) = \sum_{k=1}^{\infty} \Delta t_n \mathbf{x}^{(k)}(t_{n-1})/k! \tag{4.88}$$

However, since we cannot numerically evaluate an infinite series, the right-hand side of Eq. (4.88) is approximated by a finite series of m terms as follows:

$$\mathbf{x}(t_n) - \mathbf{x}(t_{n-1}) \approx \sum_{k=1}^{m} \Delta t_n \mathbf{x}^{(k)}(t_{n-1})/k! \tag{4.89}$$

The approximation given by Eq. (4.89) leads to a Runge–Kutta method of *order m*. The higher the number of terms in the series of Eq. (4.89), the greater will be the accuracy of the approximation. Comparing Eqs. (4.86) and (4.89), it can be shown that the *largest number* of terms that can be retained in the series of Eq. (4.89) is $m = p$. Usually, when $m = 4$, the resulting *fourth order Runge–Kutta method* is accurate enough for most practical purposes. It can be shown [5] that substituting Eq. (4.89) into Eq. (4.86), and making use of the exact differential equation, Eq. (4.84), results in the following relationships for the parameters of the fourth order Runge-Kutta method:

$$\alpha_i = \sum_{j=1}^{i-1} \beta_{ij}; \quad (i = 2, 3, 4) \tag{4.90}$$

$$\sum_{i=1}^{4} w_i = 1; \quad \sum_{i=1}^{4} w_i \alpha_i = 1/2; \quad \sum_{i=1}^{4} w_i \alpha_i^2 = 1/3; \quad \sum_{i=1}^{4} w_i \alpha_i^3 = 1/4;$$

$$w_3 \alpha_2 \beta_{32} + w_4(\alpha_2 \beta_{42} + \alpha_3 \beta_{43}) = 1/6; \quad w_3 \alpha_2^2 \beta_{32} + w_4(\alpha_2^2 \beta_{42} + \alpha_3^2 \beta_{43}) = 1/12;$$

$$w_3 \alpha_2 \alpha_3 \beta_{32} + w_4 \alpha_4(\alpha_2 \beta_{42} + \alpha_3 \beta_{43}) = 1/8; \quad w_4 \alpha_2 \beta_{32} \beta_{43} = 1/24 \tag{4.91}$$

Equations (4.90) and (4.91) represent 11 equations and 13 unknowns. Hence, we can obtain the solution of any 11 unknowns in terms of the remaining two unknowns, which we choose to be α_2 and α_3. The Runge–Kutta parameters are thus the following:

$$w_1 = 1/2 + [1 - 2(\alpha_2 + \alpha_3)]/(12\alpha_2\alpha_3); \quad w_2 = (2\alpha_3 - 1)/[12\alpha_2(\alpha_3 - \alpha_2)(1 - \alpha_2)]$$
$$w_3 = (1 - 2\alpha_2)/[12\alpha_3(\alpha_3 - \alpha_2)(1 - \alpha_3)]; \quad w_4 = 1/2 + [2(\alpha_2 + \alpha_3) - 3]/[12(1 - \alpha_2)(1 - \alpha_3)]$$
$$\beta_{32} = \beta_{23} = \alpha_3(\alpha_3 - \alpha_2)/[2\alpha_2(1 - \alpha_2)]; \quad \alpha_4 = 1 \tag{4.92}$$
$$\beta_{42} = \beta_{24} = (1 - \alpha_2)[\alpha_2 + \alpha_3 - 1 - (2\alpha_3 - 1)^2]/\{2\alpha_2(\alpha_3 - \alpha_2)[6\alpha_2\alpha_3 - 4(\alpha_2 + \alpha_3) + 3]\}$$
$$\beta_{43} = \beta_{34} = (1 - 2\alpha_2)(1 - \alpha_2)(1 - \alpha_3)/\{\alpha_3(\alpha_3 - \alpha_2)[6\alpha_2\alpha_3 - 4(\alpha_2 + \alpha_3) + 3]\}$$

Obviously, we should take care to avoid selecting those values for the two parameters, α_2 and α_3, which lead to the denominators of the expressions in Eq. (4.92) becoming zero. A popular choice of these two parameters is $\alpha_2 = \alpha_3 = 1/2$. However, the choice which *minimizes* the *approximation error* in the fourth order Runge-Kutta method is $\alpha_2 = 0.4$, and $\alpha_3 = 7/8 - (3/16)\sqrt{5}$.

The Runge–Kutta algorithm consists of marching in time using Eq. (4.86) with a variable time step size, Δt_n. The *truncation-error* (i.e. error of approximating Eq. (4.88) by Eq. (4.89)) is estimated using the *matrix norm* (such as the one defined in Section 4.5) after each time step. If the error is acceptable, then the solution is updated; if not, the time step size is reduced, and the error re-calculated until the error becomes acceptable. This process is repeated for the next time step, using the solution from the previous step, and so on until the final time is reached. Using MATLAB, a Runge–Kutta algorithm can be easily programmed. Fortunately, MATLAB comes with intrinsic nonlinear functions *ode23* and *ode45*, which are based on third and fifth order Runge–Kutta algorithms, respectively. Other MATLAB functions for solving nonlinear equations are *ode113*, *ode15s*, *ode23s*, *ode23t*, and *ode23tb*. The function *ode113* uses a variable order integration method for nonlinear equations. The functions with names ending with the letters *s*, *t*, or *tb* are specially suited for solving *stiff equations*. Stiff equations [5] are a set of first-order nonlinear equations with a large difference in their *time scales* (e.g. solution to each equation may have a significantly different time for reaching a *steady state*). The normal solution procedure that takes into account only the shortest time scale of stiff equations may either fail to converge, or may require very large number of time steps to arrive at a steady state. Hence, stiff equations require special solution procedures [5]. We will consider the more common variety of nonlinear equations (i.e. non-stiff equations) that can be solved using *ode23*, *ode113*, and *ode45*. These functions are used as follows to obtain a solution to a set of nonlinear state-equations:

```
>>[t,X] = ode23(@fun,tspan,X0,options); <enter>
```

where *@fun* denotes a user supplied M-file, *fun.m*, in which the time derivative of the state-vector, $\mathbf{x}^{(1)}(t)$, is evaluated using Eq. (4.84), *tspan* = [*ti t1 t2 t3* ... *tf*] is a row vector containing the initial time, *ti*, at which the initial condition vector, **x0**, is specified, any intermediate times, *t1, t2, t3*, ..., at which the solution is desired (optional), and the final time, *tf*, and **t** is a vector containing the time points at which the returned solution,

x, is obtained. The returned matrix **x** contains as many rows as there are the time points, and each column of **x** corresponds to a state variable. The first executable statement of the M-file *fun.m* should be the following:

```
>>function xdot = (t,x)
```

and the remaining statements of *fun.m* should evaluate the derivative of the state-vector, **xdot**, based on the state-vector, **x**, and time, t. Note that the input vector, $\mathbf{u}(t)$, is internal to the M-file *fun.m*, i.e. it is not used directly in *ode23*, *ode113* or *ode45*, but only indirectly through **xdot**. The fourth input argument, *options*, can be used to specify *relative error* and *absolute error tolerances*, *Reltol* (a scalar) and *Abstol* (a vector of the same size as **x**), respectively, for convergence through the function *odeset*. This ensures that the error in the ith component of the solution vector, x_i, does not exceed the greater number between $Reltol|x_i|$ and $Abstol(i)$. If options are not specified, then the default values of relative tolerance of 0.001 and absolute tolerance of 10^{-6} are used. For more information on the *ode* functions use the MATLAB's help command.

Example 4.12

Consider a double-pendulum (Figure 1.5). A choice of the state variables for this fourth order system is $x_1(t) = \theta_1(t)$, $x_2(t) = \theta_2(t)$, $x_3(t) = \theta_1^{(1)}(t)$; $x_4(t) = \theta_2^{(1)}(t)$, which results in the following state-equations:

$$
\begin{aligned}
x_1^{(1)}(t) &= x_3(t) \\
x_2^{(1)}(t) &= x_4(t) \\
x_3^{(1)}(t) &= [m_2 L_1 x_3^2(t) \sin(x_2(t) - x_1(t)) \cos(x_2(t) - x_1(t)) \\
&\quad + m_2 L_2 x_4^2(t) \sin(x_2(t) - x_1(t)) + m_2 g \sin(x_2(t)) \cos(x_2(t) - x_1(t)) \\
&\quad - (m_1 + m_2) g \sin(x_1(t))]/[L_1(m_1 + m_2) - m_2 L_1 \cos^2(x_2(t) - x_1(t))] \\
x_4^{(1)}(t) &= -[g \sin(x_2(t)) + L_1 x_3^2(t) \sin(x_2(t) - x_1(t)) + L_1 x_3^{(1)}(t) \cos(x_2(t) \\
&\quad - x_1(t))]/L_2 + u(t)/(m_2 L_2^2)
\end{aligned}
\tag{4.93}
$$

where $u(t)$ is the input torque applied on the mass, m_2. Note that the last state-equation has a term involving $x_3^{(1)}(t)$ on the right-hand side. This has been done for the sake of brevity (you can substitute $x_3^{(1)}(t)$ from the previous state-equation into the last state-equation to obtain the state-equations in explicit form). It is desired to obtain the solution of Eq. (4.93) for the initial condition $\mathbf{x}(0) = [0.7 \text{ rad.}; 1.4 \text{ rad.}; 0 \text{ rad./s}; 0 \text{ rad./s}]^T$ and input, $u(t) = 0.01 \sin(5t) N\text{-}m$. The function M-file for evaluating the time derivative of the state-vector, $\mathbf{x}^{(1)}(t)$, is called *doub.m* and is tabulated in Table 4.7. Note that the input, $u(t)$, must be specified within the function file *doub.m*, while the initial condition is specified in the call to the Runge–Kutta solver (either *ode23* or *ode45*).

Table 4.7 Listing of the M-file *doub.m*

doub.m

```
function xp=doub(t,x)
% Nonlinear state-equations for a double-pendulum, excited
% by input, u(t), torque acting on mass m2.
% x=[theta1 theta2 thetadot1 thetadot2]
% xp is the time derivative of the state-vector, x.
% copyright(c)2000 by Ashish Tewari
m1=1;m2=2;l1=1;l2=2;g=9.8;
u=0.01*sin(5*t);
xp(1,1)=x(3);
xp(2,1)=x(4);
x21=x(2)-x(1);
xp(3,1)=(m2*l1*x(3)*x(3)*sin(x21)*cos(x21)+m2*l2*x(4)*x(4)*sin(x21)...
  + m2*g*sin(x(2))*cos(x21)-(m1+m2)*g*sin(x(1)))/((m1+m2)*l1-...
  m2*l1*cos(x21)*cos(x21));
xp(4,1)=-(g*sin(x(2))+l1*x(3)*x(3)*sin(x21)+l1*xp(3)*cos(x21))/l2
+u/(m2*l2*l2);
```

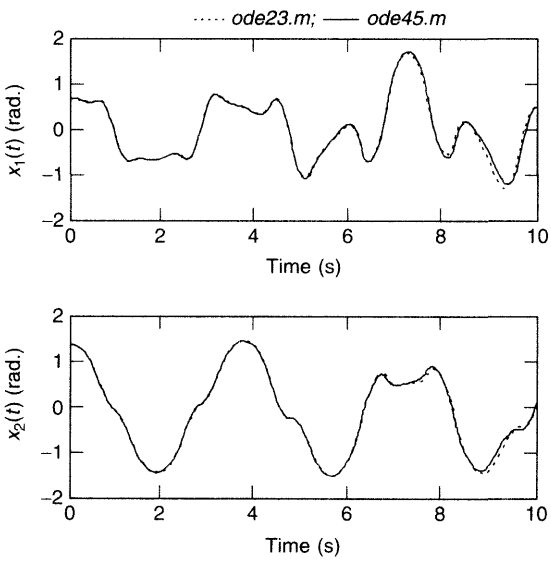

Figure 4.15 The calculated state variables, $x_1(t) = \theta_1(t)$ and $x_2(t) = \theta_2(t)$ for the double-pendulum, Example 4.12

Let us compare the solutions obtained using *ode23* and *ode45*, as follows:

```
>>[t1,x1]= ode23(@doub, [0 10], [0.7 1.4 0 0]'); <enter>

>>[t2,x2]= ode45(@doub, [0 10], [0.7 1.4 0 0]'); <enter>

>>subplot(211), plot(t1,x1(:,1),t2,x2(:,1)), hold on, subplot(212),
  plot(t1,x1(:,2),t2,x2(:,2)) <enter>
```

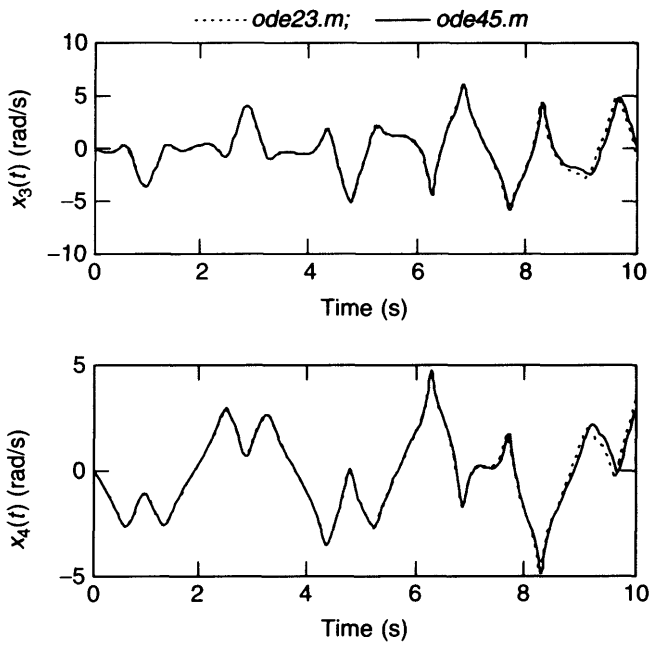

Figure 4.16 The calculated state variables, $x_3(t) = \theta_1^{(1)}(t)$, and $x_4(t) = \theta_2^{(1)}(t)$ for the double-pendulum, Example 4.12

The resulting plots of $x_1(t)$ and $x_2(t)$ are shown in Figure 4.15. Figure 4.16 plotting $x_3(t)$ and $x_4(t)$ is obtained by the following commands:

```
>>newfig <enter>

>>subplot(211), plot(t1,x1(:,3),t2,x2(:,3)), hold on, subplot(212),
   plot(t1,x1(:,4),t2,x2(:,4)) <enter>
```

Note that in Figures 4.15 and 4.16, a very small difference is observed between the state variables calculated by *ode23.m* and those calculated by *ode45.m.* This difference is seen to increase with time, indicating a larger truncation error for the third order Runge–Kutta method of *ode23.m* when compared to the fifth order Runge–Kutta method of *ode45.* Since the truncation error is added up after each time step, there is an error accumulation as time increases. The double-pendulum falls into a special category of nonlinear systems, called *chaotic systems*, which were discussed in Section 1.3. Figure 1.6 compared the state variable, $x_2(t)$, calculated for two very slightly different initial conditions and a zero input, and was generated using *ode45.* Figure 1.6 showed a large difference in the response, $x_2(t)$, when the initial conditions differed by a very small amount, which is the hallmark of a chaotic system.

Example 4.13

Let us consider another interesting nonlinear system, called the *wing-rock phenomenon*. *Wing-rock* is a special kind of *rolling* and *yawing* motion observed in modern fighter type aircraft when operating at a large angle of attack (defined as the angle made by the *longitudinal axis* of the aircraft and the *direction of flight*). The nonlinear state-equations modeling the wing-rock dynamics of a fighter aircraft are the following [6]:

$$\begin{aligned}
x_1^{(1)}(t) &= x_2(t) \\
x_2^{(1)}(t) &= -\omega^2 x_1(t) + \mu_1 x_2(t) + \mu_2 x_1^2(t) x_2(t) + b_1 x_2^3(t) + b_2 x_1(t) x_2^2(t) \\
&\quad + L_\delta x_3(t) + L_\beta x_4(t) - L_r x_5(t) \\
x_3^{(1)}(t) &= -k x_3(t) + k u(t) \\
x_4^{(1)}(t) &= x_5(t) \\
x_5^{(1)}(t) &= -N_p x_2(t) - N_\beta x_4(t) - N_r x_5(t)
\end{aligned} \tag{4.94}$$

where the state variables are, $x_1(t)$: *bank angle* (rad.), $x_2(t)$: *roll-rate* (rad./s), $x_3(t)$: *aileron deflection angle* (rad.), $x_4(t)$: *sideslip angle* (rad.), and $x_5(t)$: *sideslip-rate* (rad./s). The input, $u(t)$, is the *desired aileron deflection* (rad.). The constants ω, μ_1, μ_2, b_1, b_2, L_δ, L_β, L_r, N_p, N_β, and N_r depend on the *inertial* and *aerodynamic* properties of the aircraft, while the constant k is the aileron-actuator's *time-constant* (the aileron is an aerodynamic *control surface* which is deployed using a first order actuator). Let us obtain the solution to Eq. (4.94) when the initial condition is $\mathbf{x}(0) = [1.0 \text{ rad.}; 0.5 \text{ rad./s}; 0 \text{ rad.}; 0 \text{ rad.}; 0 \text{ rad./s}]^T$ and the input is zero. The time derivative of state-vector, $\mathbf{x}^{(1)}(t)$, is evaluated using the M-file called *wrock.m*, which is tabulated in Table 4.8.

Using *ode45*, the initial response (i.e. response when $u(t) = 0$) is obtained as follows:

```
>>[t,x]= ode45(@wrock, [0 700], [0.2 0 0 0 0]'); <enter>
```

The plot of the bank angle, $x_1(t)$, from $t = 0$ s to $t = 700$ s is shown in Figure 4.17. Note that instead of decaying to zero in the limit $t \to \infty$, the initial response keeps on oscillating with a constant time period, and an amplitude which becomes constant in the limit $t \to \infty$. Such a motion is called a *limit cycle motion*. Note that while the system is not unstable (i.e. the response *does not* tend to infinity in the limit $t \to \infty$), a limit cycle response is undesirable from weapons aiming and delivery considerations, and also because it may lead to structural fatigue in the aircraft (or other mechanical systems) thereby causing the wings to come-off. Figure 4.18 shows a plot of $x_2(t)$ against $x_1(t)$. Such a plot in which the time derivative of a variable ($x_2(t) = x_1^{(1)}(t)$ is plotted against the variable itself ($x_1(t)$) is called a *phase-plane plot*. Figure 4.18 shows that the limit cycle motion corresponds to a limiting outer boundary in the phase-plane plot, indicating that the amplitude

Table 4.8 Listing of the M-file *wrock.m*

wrock.m

```
function xdot=wrock(t,x)
% nonlinear state-equations for the wing-rock problem;
% including first order aileron actuator
% xdot is the time derivative of the state-vector, x.
% copyright(c)2000 by Ashish Tewari
a=[-0.05686 0.03254 0.07334 -0.3597 1.4681];
% pure-rolling mode natural-frequency squared:-
w=-0.354*a(1);
% aileron-actuator time-constant:-
k=1/0.0495;
% linear aerodynamic coefficients:-
lbet=-0.02822;lr=0.1517;np=-0.0629;nbet=1.3214;nr=-0.2491;ldelt=1;
% nonlinear inertial and aerodynamic coefficients:-
u(1)=0.354*a(2)-0.001;
u(2)=0.354*a(4);
b(1)=0.354*a(3);
b(2)=0.354*a(5);
% desired aileron deflection as the input, 'f':-
f=0;
% the nonlinear state-equations:-
xdot(1,1)=x(2);
xdot(2,1)=-w*x(1)+u(1)*x(2)+b(1)*x(2)^3+u(2)*x(2)*x(1)^2
      +b(2)*x(1)*x(2)^2...+ldelt*x(3)+lbet*x(4)-lr*x(5);
xdot(3,1)=-k*x(3)+k*f;
xdot(4,1)=x(5);
xdot(5,1)=-nbet*x(4)+nr*x(5)-np*x(2);
```

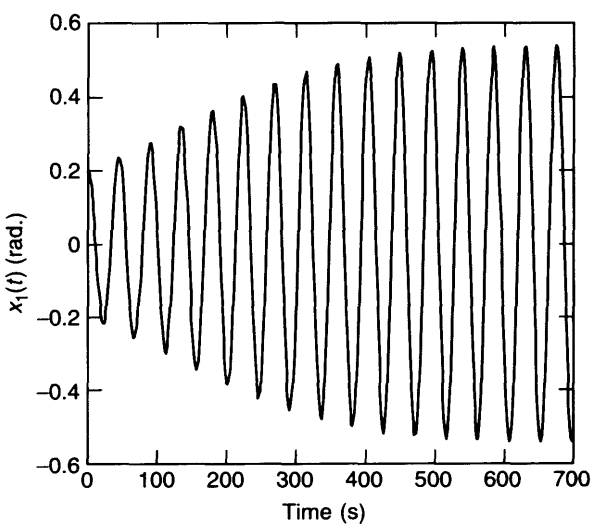

Figure 4.17 Initial response of bank angle, $x_1(t)$, for the wing-rock problem of Example 4.13 showing a *limit cycle motion* (i.e. constant amplitude oscillation in the limit $t \to \infty$)