Solving Delay Differential Equations with `dde23`

L.F. Shampine
Mathematics Department
Southern Methodist University
Dallas, TX 75275
lshampin@mail.smu.edu

S. Thompson
Department of Mathematics & Statistics
Radford University
Radford, VA 24142
thompson@runet.edu

J. Kierzenka
The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760
jkierzenka@mathworks.com

May 2, 2002

1 Introduction

Ordinary differential equations (ODEs) and delay differential equations (DDEs) are used to describe many phenomena of physical interest. While ODEs con-

tain derivatives which depend on the solution at the present value of the independent variable ("time"), DDEs contain in addition derivatives which depend on the solution at previous times. DDEs arise in models throughout the sciences [1]. Despite the obvious similarities between ODEs and DDEs, solutions of DDE problems can differ from solutions for ODE problems in several striking, and significant, ways [2] [18]. This accounts in part for the lack of much general-purpose software for solving DDEs.

We consider here only systems of delay differential equations of the form

$$y'(t) = f(t, y(t), y(t-\tau_1), y(t-\tau_2), \ldots, y(t-\tau_k)) \tag{1}$$

that are solved on $a \le t \le b$ with given history $y(t) = S(t)$ for $t \le a$. The constant delays are such that $\tau = \min(\tau_1, \ldots, \tau_k) > 0$. Although DDEs with delays (lags) of more general form are important, this is a large and useful class of DDEs. Indeed, Baker, Paul, and Willé [2] write that "The lag functions that arise most frequently in the modelling literature are constants."

Although the effective solution of DDEs has benefited a great deal from the advances made in ODE technology during the past several years, the state-of-the-art for DDE software is not at the level of ODE software. The few FORTRAN codes for solving DDEs are considerably more difficult to use than the popular ODE codes. We have developed a MATLAB program `dde23` [18] with the goal of making it as easy as possible to solve the wide range of DDEs with constant delays encountered in practice.

This tutorial shows how to solve DDEs with `dde23`. It is organized as follows. Important differences between DDEs and ODEs are discussed briefly in §2. In §3 there is a brief discussion of how numerical methods for ODEs can be extended to solve DDEs. The most important part of this tutorial is the collection of examples in §4. As the first few show, anyone familiar with solving ODEs using `ode23` [16] will find it easy to solve routine DDEs with `dde23`. Several examples then illustrate the powerful capabilities of `dde23` for solving DDEs that are far from routine. Most of the examples have an exercise that provides some practice with the techniques illustrated by the example. The tutorial ends with some problems that serve as practice for solving DDEs with constant delays in general. The complete solutions for all examples, exercises, and problems that accompany the tutorial can be used as templates. They show that interesting delay differential equation problems can be solved easily in MATLAB with `dde23`.

2 Delay Differential Equations

In this section we describe briefly some important differences between DDEs and ODEs. More detailed discussions of the various issues are found in [18].

The most obvious difference between ODEs and DDEs is the initial data. The solution of an ODE is determined by its value at the initial point $t = a$. In evaluating the DDEs (1) for $a \le t \le b$, a term like $y(t-\tau_j)$ may represent values of the solution at points prior to the initial point. For example, at $t = a$ we must have the solution at $a - \tau_j$. It is easy to see that if T is the longest delay, the equations generally require us to provide the solution $S(t)$ for $a - T \le t \le a$. For DDEs we must provide not just the value of the solution at the initial point, but also the "history", the solution at times prior to the initial point.

Because numerical methods for both ODEs and DDEs are intended for problems with solutions that have several continuous derivatives, discontinuities in low-order derivatives require special attention. This is a much more serious matter for DDEs. For one thing, such discontinuities are not unusual for ODEs, but they are almost always present for DDEs: Generally there is a discontinuity in the first derivative of the solution at the initial point because generally $S'(a-) \ne y'(a+) = f(a, S(a-\tau_1), \ldots, S(a-\tau_k))$. There can also be discontinuities at times both before and after the initial point. Some problems have histories with discontinuities in low-order derivatives. Some models involve equations that change when the solution satisfies a given relation, e.g., when a solution component has a given value. These changes often cause discontinuities in low-order derivatives of the solution.

Another reason why discontinuities are much more serious for DDEs is that they propagate. If the solution has a discontinuity in a derivative somewhere, there are discontinuities in the rest of the interval at a spacing given by the delays. In reasonably general circumstances, the propagated discontinuities are smoothed: If there is a discontinuity at t^* of order k, i.e., there is a jump in $y^{(k)}$ at t^*, then the discontinuity at $t^* + \tau_j$ is of order at least $k+1$, the discontinuity at $t^* + 2\tau_j$ is of order at least $k+2$, and so on. This is very important for numerical solution of the DDE because once the orders are high enough, the discontinuities will not interfere with the numerical method and we can stop tracking them.

To see how discontinuities propagate and smooth out, let us solve

$$y'(t) = y(t-1) \tag{2}$$

for $0 \leq t$ with history $S(t) = 1$ for $t \leq 0$. With this history, the problem reduces on the interval $0 \leq t \leq 1$ to the ODE $y'(t) = 1$ with initial value $y(0) = 1$. Solving this problem we find that $y(t) = t+1$ for $0 \leq t \leq 1$. Notice that the solution has a discontinuity in its first derivative at $t = 0$. In the same way we find that $y(t) = (t^2 + 3)/2$ for $1 \leq t \leq 2$. The first derivative is continuous at $t = 1$, but there is a discontinuity in the second derivative. In general the solution on the interval $[k, k + 1]$ is a polynomial of degree $k + 1$ and there is a discontinuity of order $k + 1$ at $t = k$.

3 Numerical Methods for DDEs

In this section we discuss a few aspects of the numerical solution of DDEs. A detailed discussion of the methods used by `dde23` can be found in [18].

A popular approach to solving DDEs is to extend one of the methods used to solve ODEs. Most of the codes are based on explicit Runge-Kutta methods. `dde23` takes this approach by extending the method of the MATLAB ODE solver `ode23`. The idea is the same as the so-called "method of steps" for solving DDEs that was used to solve an example in the last section. To be concrete, we describe the idea as applied to this example. In solving (2) for $0 \leq t \leq 1$, the DDE reduces to an initial value problem for an ODE with $y(t - 1)$ equal to the given history $S(t - 1)$ and initial value $y(0) = 1$. We can solve this ODE numerically using any of the popular methods for the purpose. Analytical solution of the DDE on the next interval $1 \leq t \leq 2$ is handled the same way as the first interval, but the numerical solution is somewhat complicated, and the complications are present for each of the subsequent intervals. The first complication is that we must keep track of how the discontinuity at the initial point propagates because of the delays. Another is that at each discontinuity we start the solution of an initial value problem for an ODE. Runge-Kutta methods are attractive because they are much easier to start than other popular numerical methods for ODEs. Still another issue is the term $y(t - 1)$ that is in principle known because we have already found $y(t)$ for $0 \leq t \leq 1$. This has been a serious obstacle to applying Runge-Kutta methods to DDEs, so we need to discuss the matter more fully.

Runge-Kutta methods, like all discrete variable methods for ODEs, produce approximations y_n to $y(x_n)$ on a mesh $\{x_n\}$ in the interval of interest, here $[0, 1]$. They do this by starting with the given initial value, $y_0 = y(a)$ at $x_0 = a$, and stepping from $y_n \approx y(x_n)$ a distance of h_n to $y_{n+1} \approx y(x_{n+1})$ at

$x_{n+1} = x_n + h_n$. The step size h_n is chosen as small as necessary to get an accurate approximation. It is chosen as big as possible so as to reach the end of the interval in as few steps as possible, which is to say, as cheaply as possible. In the case of solving (2) on the interval $[1, 2]$, we have values of the solution only on a mesh in $[0, 1]$. So, where do the values $y(t-1)$ come from? In their original form Runge-Kutta methods produce answers only at mesh points, but it is now known how to obtain "continuous extensions" that yield an approximate solution between mesh points. The trick is to get values between mesh points that are just as accurate and to do this cheaply. In some cases the continuous extensions can be viewed as interpolants. As an example, the first widely available FORTRAN DDE solver, DMRODE [10], is based on a standard Runge-Kutta formula and Hermite interpolants of various orders. The BS(2,3) Runge-Kutta method used by `ode23` was derived along with a continuous extension based on cubic Hermite interpolation. Besides the other good qualities of this method, cubic Hermite interpolation between mesh points provides a numerical solution just as accurate as the solution at mesh points. Furthermore, the data needed for the interpolation is available as a byproduct of the step itself. With such a method, when we solve (2) on $[0, 1]$ we obtain $y(t)$ everywhere in the interval, not just mesh points. In this way we obtain inexpensively the accurate values for $y(t-1)$ needed when integrating the ODE on $[1, 2]$, and similarly for all the subsequent intervals.

The Runge-Kutta methods mentioned are all explicit recipes for computing y_{n+1} given y_n and the ability to evaluate the equation. For reasons of efficiency, a solver tries to use the biggest step size h_n that will yield the specified accuracy, but what if it is bigger than the shortest delay τ? In taking a step to $x_n + h_n$, we would then need values of the solution at points in the span of the step, but we are trying to compute the solution at the end of the step and do not yet know these values. A good many solvers restrict the step size to avoid this issue. Some solvers, including `dde23`, use whatever step size appears appropriate and iterate to evaluate the implicit formula that arises in this way.

4 Examples

In this section we use problems from the literature to show how to solve DDEs with `dde23`. Solving a DDE with `dde23` is much like solving an ODE with `ode23`, but there are some notable differences. Examples 1 through 3

show how to solve typical problems. They should be read in order. `dde23` has a powerful event location capability that is quite similar to that of `ode23`. Example 4 illustrates the capability by finding local maxima of the solution. ODE and DDE solvers are intended for problems with solutions that have several continuous derivatives. However, it is not unusual for equations to have different forms in different circumstances, which leads to discontinuities in low-order derivatives of the solution when the circumstances change. This matter is more serious for DDEs because discontinuities propagate and discontinuities can occur in the history. Examples 5 through 8 show how to deal with discontinuities in low-order derivatives, including jumps in the solution itself. They consider situations in order of difficulty and some require familiarity with a previous example. `dde23` is limited to problems with constant delays, but the examples/exercises/problems of this section show that for this class of problems, it is both easy to use and powerful.

Complete solutions are provided for all the examples that can be used as templates. Some of the examples have exercises that are solved in a similar way. It is worth trying them for practice. Complete solutions are provided as a check and as further templates. This tutorial ends with some additional problems that serve as exercises for all the examples. Again, complete solutions are provided as a check and as further templates.

A naming convention is used throughout this section. For example, `exam1.m` is the M–file for solving the problem of Example 1. The equations of this problem are evaluated in the subfunction `exam1f` of `exam1.m`. Some problems involve additional functions, specifically a history function and/or an event function. The corresponding subfunctions have the names `exam1h` and `exam1e`, respectively. The M–files and functions for the exercises follow the same convention with `exam` replaced by `exer`. Finally, the M–files and functions for the additional problems are similarly named with `exam` replaced by `prob`.

Example 1

We illustrate the straightforward solution of a DDE by computing and plotting the solution of Example 3 of [21]. The equations

$$\begin{aligned} y_1'(t) &= y_1(t-1) \\ y_2'(t) &= y_1(t-1) + y_2(t-0.2) \\ y_3'(t) &= y_2(t) \end{aligned}$$

are to be solved on $[0, 5]$ with history $y_1(t) = 1, y_2(t) = 1, y_3(t) = 1$ for $t \leq 0$.

A typical invocation of `dde23` has the form

```
sol = dde23(ddefun,lags,history,tspan);
```

The input argument `tspan` is the interval of integration, here `[0, 5]`. The `history` argument is a function that evaluates the solution at the input value of `t` and returns it as a column vector. Here `exam1h` can be coded as

```
function v = exam1h(t)
v = ones(3,1);
```

Quite often the history is a constant vector. A simpler way to provide the history then is to supply the vector itself as the `history` argument. The delays are provided as a vector `lags`, here `[1, 0.2]`. `ddefun` is a function for evaluating the DDEs. Here `exam1f` can be coded as

```
function v = exam1f(t,y,Z)
ylag1 = Z(:,1);
ylag2 = Z(:,2);
v = zeros(3,1);

v(1) = ylag1(1);
v(2) = ylag1(1) + ylag2(2);
v(3) = y(2);
```

The input `t` is the current t and `y`, an approximation to $y(t)$. The input array `Z` contains approximations to the solution at all the delayed arguments. Specifically, `Z(:,j)` approximates $y(t - \tau_j)$ for τ_j given as `lags(j)`. It is not necessary to define local vectors `ylag1, ylag2` as we have done here, but often this makes the coding of the DDEs clearer. The `ddefun` must return a column vector.

This is perhaps a good place to point out that `dde23` does not assume that terms like $y(t - \tau_j)$ actually appear in the equations. Because of this, you can use `dde23` to solve ODEs. If you do, it is best to input an empty array, `[]`, for `lags` because any delay specified affects the computation even when it does not appear in the equations.

The input arguments of `dde23` are much like those of `ode23`, but the output differs formally in that it is one structure, here called `sol`, rather than several arrays

```
[t,y,...] = ode23(...
```

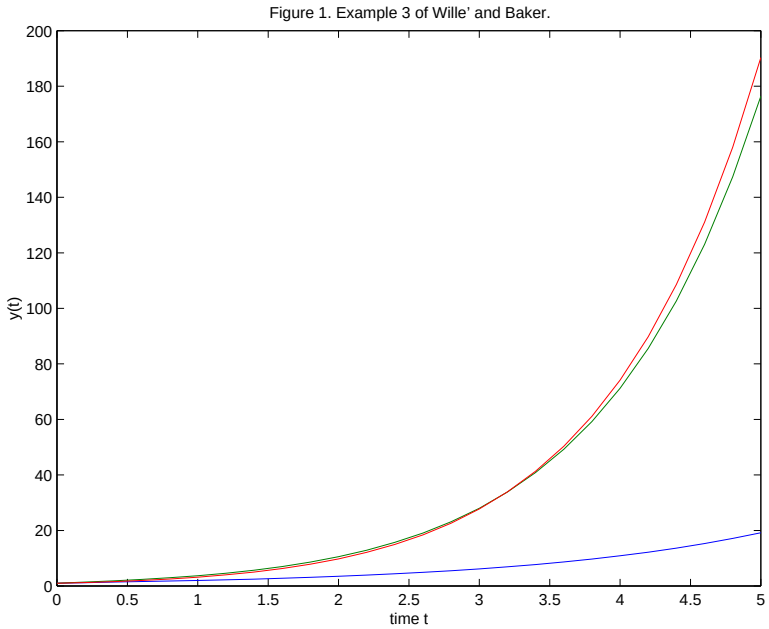
The field `sol.x` corresponds to the array `t` of values of the independent variable returned by `ode23` and the field `sol.y`, to the array `y` of solution values. So, one way to plot the solution is

```
plot(sol.x,sol.y);
```

After defining the equations in `exam1f`, the program `exam1.m` to compute and plot the solution is

```
sol = dde23(@exam1f,[1, 0.2],ones(3,1),[0, 5]);

plot(sol.x,sol.y);
title('Figure 1. Example 3 of Wille'' and Baker.')
xlabel('time t');
ylabel('y(t)');
```

Note that we must supply the *function handle* of the `ddefun` to the solver, i.e., `@exam1f` rather than `exam1f`. Also, we have taken advantage of the easy way to specify a constant history.



Exercise 1

To gain experience with `dde23`, compute and plot the solution of the following problem from [10]. Solve

$$\begin{aligned} y_1'(t) &= y_5(t-1) + y_3(t-1) \\ y_2'(t) &= y_1(t-1) + y_2(t-0.5) \\ y_3'(t) &= y_3(t-1) + y_1(t-0.5) \\ y_4'(t) &= y_5(t-1)y_4(t-1) \\ y_5'(t) &= y_1(t-1) \end{aligned}$$

on $[0, 1]$ with history $y_1(t) = \exp(t+1), y_2(t) = \exp(t+0.5), y_3(t) = \sin(t+1), y_4(t) = y_1(t), y_5(t) = y_1(t)$ for $t \leq 0$.

In this you will have to evaluate the history in a function and supply its handle, say `@exer1h`, as the history argument of `dde23`. Remember that both the `ddefun` and the history function must return *column* vectors. In [10] this problem is used to show how to prepare a class of DDEs for solution with DMRODE. You might find it interesting to compare this preparation to what you had to do.

Example 2

We show how to get output at specific points with Example 5 of [21], a scalar equation that exhibits chaotic behavior. We solve the equation

$$y'(t) = \frac{2\,y(t-2)}{1 + y(t-2)^{9.65}} - y(t) \tag{3}$$

on $[0, 100]$ with history $y(t) = 0.5$ for $t \leq 0$.

Output from `dde23` is not just formally different from that of `ode23`. `dde23` computes an approximate solution $S(t)$ valid throughout `tspan` and places in `sol` the information necessary to evaluate it. This evaluation is done with `deval`. All you have to do is supply the solution structure and an array `t` of points where you want the values of $S(t)$:

```
S = deval(sol,t);
```

With this form of output, you can solve a DDE just once and then obtain inexpensively as many solution values as you like, anywhere you like. The

numerical solution itself is continuous and has a continuous derivative, so you can always get a smooth graph by evaluating it at enough points with `deval`.

The example of [21] plots $y(t-2)$ against $y(t)$. This is quite a common task in nonlinear dynamics, but we cannot proceed as in Example 1. That is because the entries of `sol.x` are not equally spaced: If t^* appears in `sol.x`, we have an approximation to $y(t^*)$ in `sol.y`, but generally t^*-2 does not appear in `sol.x`, so we do not have an approximation to $y(t^*-2)$. `deval` makes such plots easy. In `exam2.m` we first define an array `t` of 1000 equally spaced points in $[2, 100]$ and obtain solution values at these points with `deval`. We then use `deval` a second time to evaluate the solution at the entries of `t-2`. In this way we obtain values approximating both $y(t)$ and $y(t-2)$ for the same t. This might seem like a lot of plot points, but `deval` is just evaluating a piecewise-polynomial function and is coded to take advantage of fast builtin functions and vectorization, so this is not expensive and results in a smooth graph.

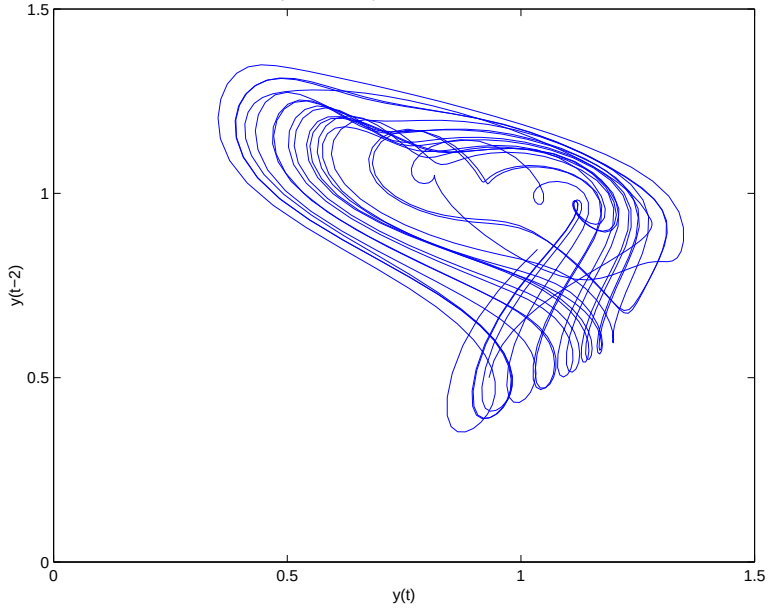
Because MATLAB does not distinguish scalars and vectors of one component, the single DDE can be coded as

```
function v = exam2f(t,y,Z)
v = 2*Z/(1 + Z^9.65) - y;
```

The program `exam2.m` to compute and plot $y(t-2)$ against $y(t)$ is

```
sol = dde23(@exam2f,2,0.5,[0, 100]);

t = linspace(2,100,1000);
y = deval(sol,t);
ylag = deval(sol,t - 2);
plot(y,ylag);
title('Figure 2. Example 5 of Wille'' and Baker.')
xlabel('y(t)');
ylabel('y(t-2)');
axis([0 1.5 0 1.5])
```

Figure 2. Example 5 of Wille' and Baker.



Exercise 2

Farmer [5] gives plots of various Poincaré sections for the Mackey–Glass equation, a scalar DDE that exhibits chaotic behavior. Reproduce Fig. 2a of the paper by solving

$$y'(t) \;=\; \frac{0.2\,y(t-14)}{1+y(t-14)^{10}} \;-\; 0.1y(t) \tag{4}$$

on $[0, 300]$ with history $y(t) = 0.5$ for $t \le 0$ and plotting $y(t-14)$ against $y(t)$. The figure begins with $t = 50$ to allow an initial transient time to settle down. To reproduce it, form an array of 1000 equally spaced points in $[50, 300]$, evaluate $y(t)$ at these points, and then evaluate $y(t-14)$.

Example 3

We show how to set options and deal with parameters by solving Example 4.2 of [12]. The equation

$$y'(t) \;=\; -\lambda y(t-1)\,(1+y(t)) \tag{5}$$

is solved on $[0, 20]$ with history $y(t) = t$ for $t \le 0$ for four values of the parameter λ, namely 1.5, 2, 2.5, and 3.

Often default error tolerances are perfectly satisfactory, but here more stringent tolerances are needed for the larger values of λ. Options are set with `ddeset` exactly as they are set for `ode23` with `odeset`. When options are used, a call to `dde23` has the form

```
sol = dde23(ddefun,lags,history,tspan,options);
```

Options like relative and absolute error tolerances are the same in the two solvers. In particular, both have a default relative error tolerance of 10^{-3} and default absolute error tolerance of 10^{-6}. The tolerances imposed for the larger λ in `exam3.m` are relatively stringent for this solver, but this is a price that must be paid to obtain a satisfactory solution. To see this for yourself, try solving the problem with $\lambda = 3$ and default tolerances.

Parameters can always be communicated as `global` variables, but as is common with MATLAB solvers, they can also be passed through `dde23` as arguments following the `options` argument. For two values of λ we use default tolerances, so must use an empty array, `[]`, as a placeholder for the `options` argument. When parameters are passed through `dde23`, they must appear as arguments of the `ddefun` and if present, the history function, *even if they are not used*. Accordingly, `exam3f` can be coded as

```
function v = exam3f(t,y,Z,lambda)
v = -lambda*Z*(1 + y);
```

and `exam3h` as

```
function v = exam3h(t,lambda)
v = t;
```

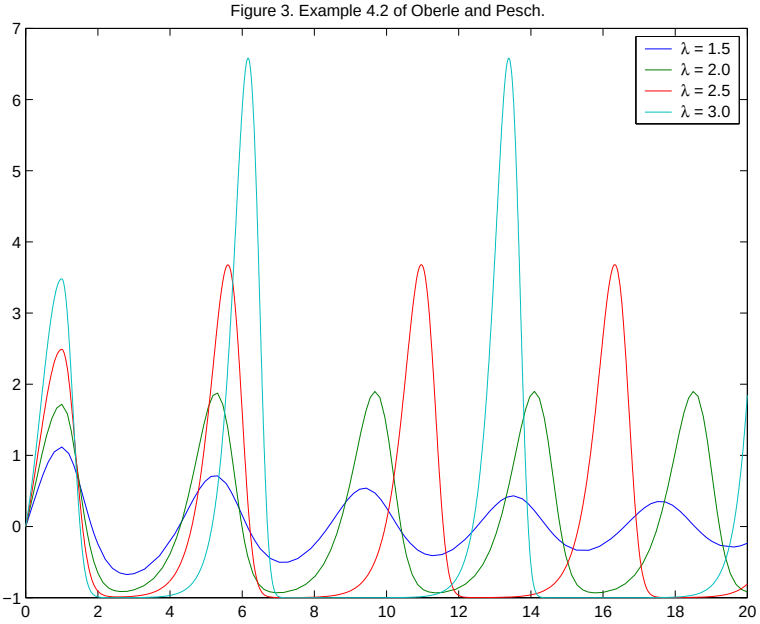
After defining the equation in `exam3f` and the history in `exam3h`, the program `exam3.m` to compute and plot the four solutions as in [12] is

```
sol1 = dde23(@exam3f,1,@exam3h,[0, 20],[],1.5);

sol2 = dde23(@exam3f,1,@exam3h,[0, 20],[],2);

opts = ddeset('RelTol',1e-5,'AbsTol',1e-8);
sol3 = dde23(@exam3f,1,@exam3h,[0, 20],opts,2.5);

opts = ddeset('RelTol',1e-6,'AbsTol',1e-10);
```

```
sol4 = dde23(@exam3f,1,@exam3h,[0, 20],opts,3);

plot(sol1.x,sol1.y,sol2.x,sol2.y,...
     sol3.x,sol3.y,sol4.x,sol4.y);
legend('\lambda = 1.5','\lambda = 2.0',...
       '\lambda = 2.5','\lambda = 3.0')
title('Figure 3. Example 4.2 of Oberle and Pesch.')
```

This has been coded in a very straightforward manner to make clear that we are solving four problems and using different tolerances.

Figure 3. Example 4.2 of Oberle and Pesch.



Exercise 3

Wheldon's model of chronic granuloctic leukemia [8] has the form

$$\begin{aligned} y_1'(t) &= \frac{\alpha}{1+\beta\, y_1(t-\tau)^\gamma} - \frac{\lambda\, y_1(t)}{1+\mu\, y_2(t)^\delta} \\ y_2'(t) &= \frac{\lambda\, y_1(t)}{1+\mu\, y_2(t)^\delta} - \omega\, y_2(t) \end{aligned}$$

Code the equations for general values of the parameters to make it easy to experiment with the model. Remember that if you do not set any options, you

must use a placeholder of `[]` for the `options` argument. Solve the problem on $[0, 200]$ with history $y_1(t) = 100, y_2(t) = 100$ for $t \leq 0$ and parameter values $\alpha = 1.1 \times 10^{10}, \beta = 10^{-12}, \gamma = 1.25, \delta = 1, \lambda = 10, \mu = 4 \times 10^{-8}, \omega = 2.43$ that you set in the main program. Compare the solutions you obtain with $\tau = 7$ and $\tau = 20$. You could code this as

```
for tau = [7, 20]
   sol = dde23(@exer3f,tau,...
   ...
end
```

You should find that the solution is oscillatory in both cases. In the first, the oscillations are damped quickly and in the second, they are not.

Example 4

It is often necessary to find when a solution satisfies a certain relation, e.g., when a component has a specific value. An event is said to occur when a function of the solution, $g(t, y(t), y(t-\tau_1), \ldots, y(t-\tau_k))$, vanishes. Some problems involve many of these "event functions". This example shows how to use the powerful event location capability of `dde23`.

Figure 15.6 of [6] displays the solution of an infectious disease model. The equations

$$\begin{aligned} y_1'(x) &= -y_1(x)y_2(x-1) + y_2(x-10) \\ y_2'(x) &= y_1(x)y_2(x-1) - y_2(x) \\ y_3'(x) &= y_2(x) - y_2(x-10) \end{aligned}$$

are solved on $[0, 40]$ with history $y_1(x) = 5, y_2(x) = 0.1, y_3(x) = 1$ for $x \leq 0$. To illustrate event location, we compute the local maxima of all three solution components.

We compute the maxima by finding where the first derivatives vanish. The three event functions come from the DDEs: $y_1'(x) = -y_1(x)y_2(x-1) + y_2(x-10)$, and so forth. All event functions are evaluated in a single MATLAB function that returns the values as a column vector. The name of this function is passed to the solver as the value of the `'Events'` option. For this example we evaluate the three functions in `exam4e` by a call to `exam4f`. Because event location is used for a variety of purposes, we have to tell `dde23` more about what we want to do. Sometimes we just want to know that an

event has occurred and other times we want to terminate the integration then. We tell the solver about this by returning a vector `isterminal` from `exam4e`. To terminate the integration when event function k vanishes, we set component k of `isterminal` to 1 (true), and otherwise to 0 (false). For this example none of the events is terminal. There is an annoying matter of some importance: Sometimes we want to start an integration with an event function that vanishes at the initial point. Imagine, for example, that we fire a model rocket into the air and we want to know when it hits the ground. It is natural to use the height of the rocket as a terminal event function, but it vanishes at the initial time as well as the final time. `dde23` treats an event at the initial point in a special way. The solver locates such an event and reports it, but does not treat it as terminal, no matter how `isterminal` is set. The example shows that how an event function vanishes may be important: To distinguish maxima from minima, we want the solver to report that a derivative vanished only when it changes from positive to negative values. This is done using `direction`. If we are interested only in events for which event function k is increasing through 0, we set component k of `direction` to $+1$. Correspondingly, we set it to -1 if we are interested only in those events for which the event function is decreasing, and 0 if we are interested in all events. Once we understand what information must be provided, it is easy to code the event functions of this example as

```
function [value,isterminal,direction] = exam4e(x,y,Z)
value = exam4f(x,y,Z);
isterminal = zeros(3,1);
direction = -ones(3,1);
```

Now that we have discussed how to tell the solver what we want it to do, we have to discuss how it reports what happened. The locations of events are returned as the field `sol.xe` and the values of the solution at these points are returned as the field `sol.ye`. If there are no events, `sol.xe = []`. The field `sol.ie` reports which event occurred. A value of k indicates that event function k vanished at the corresponding entry of `sol.xe`.

It is straightforward to code the equations as

```
function v = exam4f(x,y,Z)
ylag1 = Z(:,1);
ylag2 = Z(:,2);
v = zeros(3,1);
```

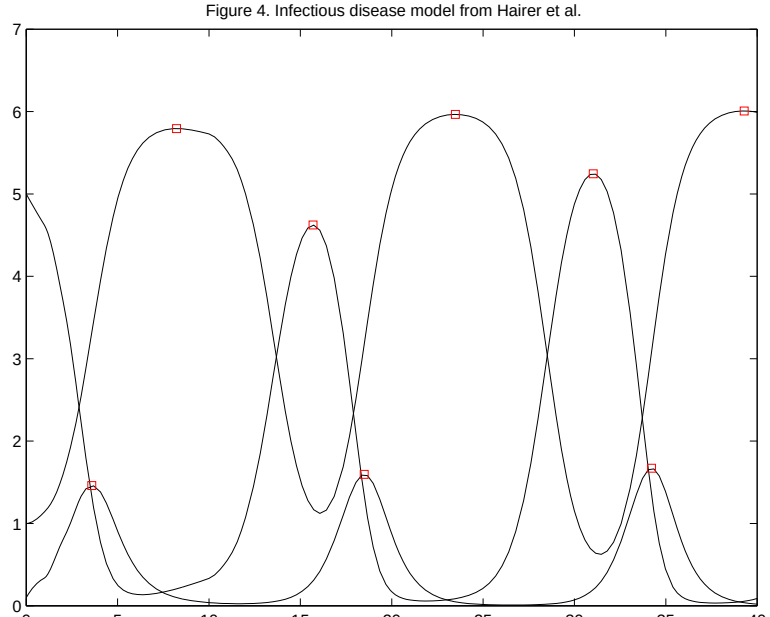
```
v(1) = -y(1)*ylag1(2) + ylag2(2);
v(2) =  y(1)*ylag1(2) - y(2);
v(3) =  y(2) - ylag2(2);
```

With `exam4e` and `exam4f`, it is also straightforward to code the solution of the problem as the first two lines of the solution `exam4.m` that follows:

```
options = ddeset('Events',@exam4e);
sol = dde23(@exam4f,[1, 10],[5; 0.1; 1],[0, 40],options);

xe = sol.xe;
ye = sol.ye;
ie = sol.ie;

n1 = find(ie == 1);
x1 = xe(n1);
y1 = ye(1,n1);
n2 = find(ie == 2);
x2 = xe(n2);
y2 = ye(2,n2);
n3 = find(ie == 3);
x3 = xe(n3);
y3 = ye(3,n3);

plot(sol.x,sol.y,'k',x1,y1,'rs',x2,y2,'rs',x3,y3,'rs')
title('Figure 4. Infectious disease model from Hairer et al.')
xlabel('Maxima are indicated by red squares.')
```

The only complication in this program is separating the various kinds of events. It is not necessary, but perhaps clearer, to introduce local variables for the fields that return the results of the event location. The command `n1 = find(ie == 1)` finds the indices corresponding to the first event function. These indices allow us to extract the information that $y_1(x)$ has its maxima at `xe(n1)` and its values there are `ye(1,n1)`. The second and third event functions are handled in the same way and then all the results are plotted.

Figure 4. Infectious disease model from Hairer et al.



Exercise 4

To gain some experience with event location, try two experiments:

- Terminate the integration when $y_1(x)$ has its first maximum.
- Compute local minima instead of maxima.

Each can be done by changing only one line in `exam4e`.

Example 5

ODE and DDE solvers are intended for problems with solutions that have several continuous derivatives. It is not unusual for equations to have different forms in different circumstances, which leads to discontinuities in low-order derivatives of the solution, or even in the solution itself, when the circumstances change. Although a robust solver may be able to produce an acceptable solution, it is better practice to account for the changes and it can be necessary. There are two issues: Do we know in advance where the changes occur? Is the solution itself continuous? In this example we show how to solve problems that have a continuous solution with discontinuities

in a low-order derivative at points known in advance. The history is the solution prior to the initial point and its discontinuities must also be taken into account because they propagate into the interval of integration. Discontinuities in the history are handled in the same way, but are a little simpler because discontinuities in the solution itself are permitted.

Example 4.4 of [12] is an infection model due to Hoppensteadt and Waltman. The equation

$$y'(t) = \begin{cases} -r\,y(t)\,0.4\,(1-t) & \text{if } 0 \le t \le 1-c, \\ -r\,y(t)\,(0.4(1-t)+10-e^{\mu}y(t)) & \text{if } 1-c < t \le 1, \\ -r\,y(t)\,(10-e^{\mu}y(t)) & \text{if } 1 < t \le 2-c, \\ -r\,e^{\mu}\,y(t)\,(y(t-1)-y(t)) & \text{if } 2-c < t. \end{cases}$$

is solved on $[0, 10]$ with history $y(t) = 10$ for $t \le 0$. Here $c = 1/\sqrt{2}$ and $\mu = r/10$. Oberle and Pesch solve this problem for several values of the parameter r, but we solve it only for $r = 0.5$. The different phases of the spread of the disease are described by different equations. In this example the phases change at times known in advance. The model requires the solution to be continuous, but the changes in the equation lead to jumps in low order derivatives. In addition to $y(t)$, an approximation to $I(t) = -y'(t)/(r\,y(t))$ is required.

`dde23` deals easily with problems that have a continuous solution and discontinuities in low-order derivatives at known points. All you have to do is tell the solver where the discontinuities are by providing them as the value of the `'Jumps'` option. However, you need to keep in mind that the history is the solution prior to the initial point, so you must also account for its discontinuities. For instance, the Marchuk immunology model discussed in [6, pp. 297–298] has the history $\max(0, t + 10^{-6})$ for $t \le 0$. Its solution has a jump in the first derivative at $t = -10^{-6}$ which propagates into the interval of integration. Discontinuities in the history are handled like discontinuities at known points during the integration. In one respect they are simpler; a jump in the solution itself is treated the same as a jump in one of its low order derivatives. Low-order discontinuities in the history have an effect in the interval of integration because of the delays. If the initial point is a and the longest delay is T, discontinuities that occur before $a - T$ have no effect on the integration, so there is no need to include them in `'Jumps'`.

Having discussed how to deal with the discontinuities, it is straightforward to solve the problem. We compute an approximation to $y(10)$ and compare

it to an accurate value reported in [12]. This illustrates the computation of an approximation at a specific point and confirms the accuracy of the computation. We compute and plot $I(t)$ at the points of $sol.x$ using the fields $sol.y$ and $sol.yp$. If we should want values at other t or should want a smoother graph, we would compute the necessary values with `deval`. If we treat r as a parameter, the equation can be coded as

```
function v = exam5f(t,y,Z,r)
c = 1/sqrt(2);
mu = r/10;
if t <= 1 - c
   v = -r*y*0.4*(1 - t);
elseif t <= 1
   v = -r*y*(0.4*(1 - t) + 10 - exp(mu)*y);
elseif t <= 2 - c
   v = -r*y*(10 - exp(mu)*y);
else
   v = -r*exp(mu)*y*(Z - y);
end
```

The program `exam5.m` is then

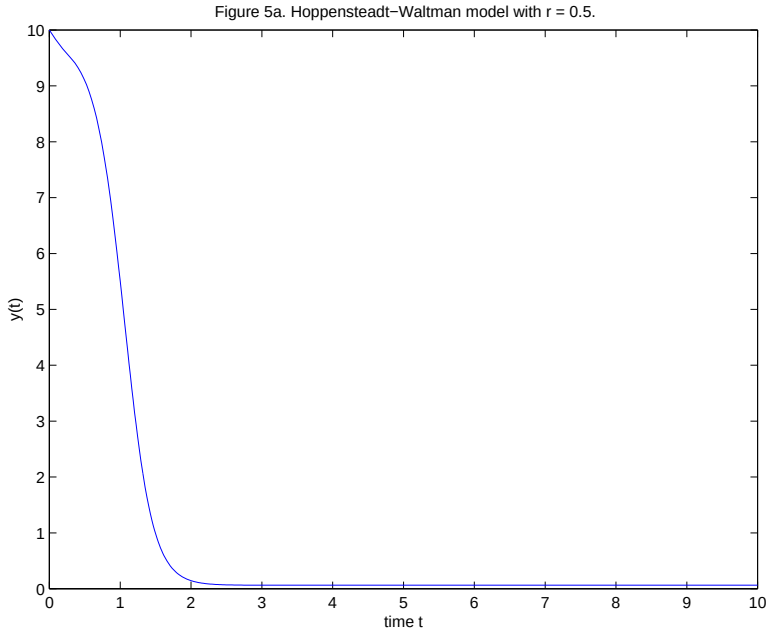
```
r = 0.5;
c = 1/sqrt(2);
opts = ddeset('Jumps',[(1-c), 1, (2-c)],...
              'RelTol',1e-5,'AbsTol',1e-8);
sol = dde23(@exam5f,1,10,[0, 10],opts,r);

y10 = deval(sol,10);
fprintf('DDE23 computed     y(10) =%15.11f.\n',y10);
fprintf('Reference solution y(10) =%15.11f.\n',0.06302089869);

plot(sol.x,sol.y)
title(['Figure 5a. Hoppensteadt-Waltman model with r = ',...
       num2str(r),'.'])
xlabel('time t')
ylabel('y(t)')

Ioft = -(1/r)*(sol.yp ./ sol.y);
```

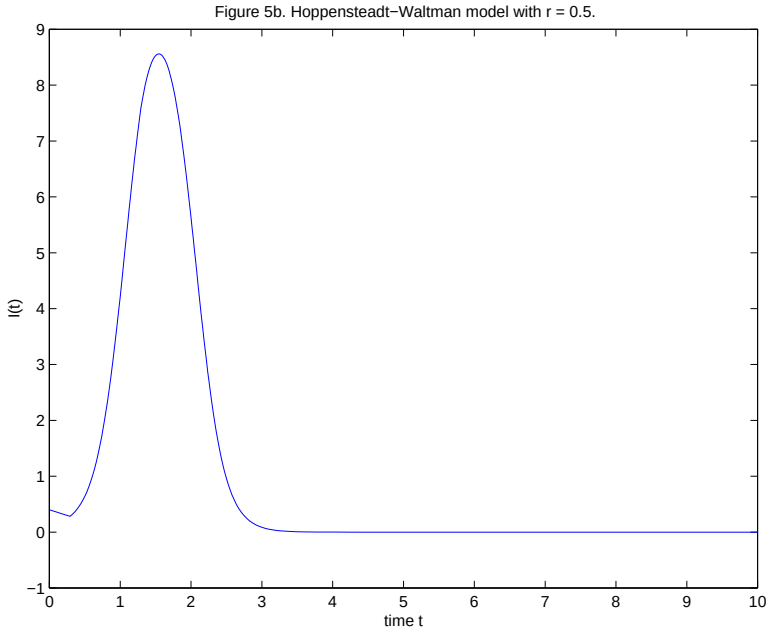
```
figure
plot(sol.x,Ioft)
title(['Figure 5b. Hoppensteadt-Waltman model with r = ',...
       num2str(r),'.'])
xlabel('time t')
ylabel('I(t)')
```

This program results in the output

```
DDE23 computed      y(10) =  0.06301980845.
Reference solution y(10) =  0.06302089869.
```

and the two figures displayed. The accuracy of the computed result is what we might expect for the specified error tolerances.





Exercise 5

Example 4 of [21] solves the equation

$$y'(t) = y(t-1)$$

on $[0,1]$ with history $y(t) = (-1)^{[-5t]}$ for $t \leq 0$. For $s > 0$, the function $[s]$ is `floor(s)` in MATLAB. The history has jump discontinuities prior to $t = 0$ that must be set in `'Jumps'`. With a delay of 1, only jumps that occur at $t \geq -1$ can have an effect in $[0,1]$.

Example 6

Discontinuities in the solution itself complicate matters. If nothing else, we must specify the jumps. In this example we show how to deal with the special case of a jump in the solution at the initial point. We also show how to plot the solution in a phase plane.

We solve a model of the infamous four year life cycle of a population of lemmings [20]. The equation

$$y'(t) \quad = \quad ry(t)\left(1 - \frac{y(t-0.74)}{m}\right)$$

is solved on $[0, 40]$ with history $y(t) = 19$ for $t < 0$. The parameters r and m have the values 3.5 and 19, respectively. Notice that with these values, the equation has a constant (steady-state) solution, $y(t) = 19$. Tavernini uses this solution as history and perturbs the initial value to $y(0) = 19.00001$ so that $y(t)$ will move away from the steady state. Here we use $y(0) = 19.001$ so as to see the cyclic behavior sooner.

Because the solution settles into a cyclic behavior, it is interesting to plot $y'(t)$ against $y(t)$. This is easily done because in addition to `sol.y`, `dde23` also returns a field `sol.yp` with values of the first derivative.

Most DDE problems have solutions that are continuous at the initial point, so there is no need to supply the solver with an initial value in addition to a history function. However, if you should want to use a different initial value, all you have to do is provide it as the value of the `'InitialY'` option. The solver deals automatically with the discontinuity in the first derivative that is ordinarily present at the initial point, so you need act only if the solution itself is discontinuous. Here the solution has a small jump at the initial point, indeed small enough that we must use error tolerances smaller than the default values so that the solver "sees" the jump.

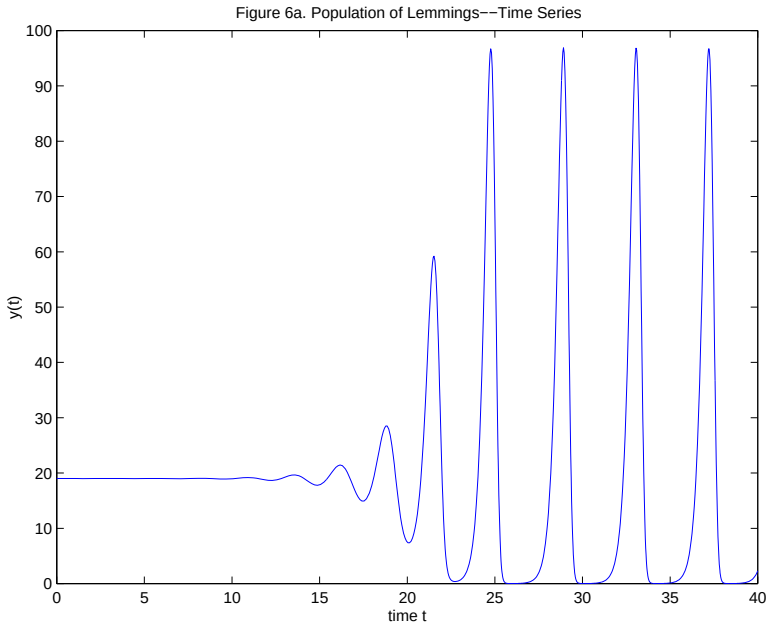
Using the capability of passing parameters through `dde23`, `exam6f` can be coded as

```
function v = exam6f(t,y,Z,r,m)
v = r*y*(1 - Z/m);
```

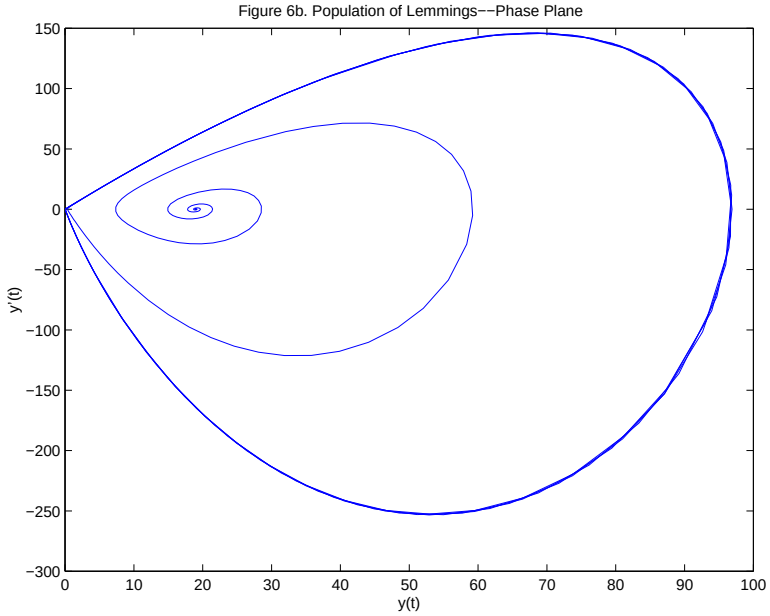
The program `exam6.m` to compute and plot the solution is then

```
r = 3.5;
m = 19;
options = ddeset('RelTol',1e-4,'AbsTol',1e-7,...
                 'InitialY',19.001);
sol = dde23(@exam6f,0.74,19,[0 40],options,r,m);

plot(sol.x,sol.y);
title('Figure 6a. Population of Lemmings--Time Series')
xlabel('time t');
ylabel('y(t)');
figure
plot(sol.y,sol.yp);
title('Figure 6b. Population of Lemmings--Phase Plane')
```

```
xlabel('y(t)');
ylabel('y''(t)');
```



Clearly the normalized population gets quite small, but how small? A reasonably accurate answer is obtained easily: The smallest value of $y(t)$ is approximately `min(sol.y)`, namely `0.0116`. If we wanted a better answer, we could obtain it by introducing event functions as in the last example.



Exercise 6

The ARCHI manual [15] provides a sample program for solving

$$\begin{aligned} y_1'(t) &= y_1(t-1)\, y_2(t-2) \\ y_2'(t) &= -y_1(t)\, y_2(t-2) \end{aligned}$$

on $[0, 4]$ with history $y_1(t) = \cos(t), y_2(t) = \sin(t)$ for $t < 0$ and initial values $y_1(0) = 0, y_2(0) = 0$. Notice that $y_1(t)$ is discontinuous at the initial point. For practice, compute and plot the solution. The sample program specifies a pure absolute error of 10^{-9}. `dde23` does not permit a pure absolute error, but for practice with options, use the default relative error tolerance and set `'AbsTol'` to `1e-9`. You might find it interesting to compare your program to the sample in [15].

Example 7

For some problems the changes in the equations occur at times that are not known in advance. The event location capability is used to determine when there is a change and the integration is terminated. It is then restarted with the new definition of the equation. The role of the history and the possibility

of a jump discontinuity in the solution itself complicate this, but `dde23` was designed to make it as painless as possible. This example and the next show how to proceed.

Marriott and DeLisle [9] solve a DDE that involves a step function of the history term. With $\Delta = y(t-12) - x_b$, the equation is

$$y'(t) = \left(-y(t) + \pi\left(a + \epsilon\,\mathrm{sign}(\Delta) - u\sin^2(\Delta)\right)\right)/\tau.$$

It is solved on $[0, 120]$ with history $y(t) = 0.6$ for $t \le 0$ and parameter values $x_b = -0.427, a = 0.16, \epsilon = 0.02, u = 0.5, \tau = 1$.

The term $\mathrm{sign}(\Delta)$ is an idealization of a sharp change that we do not expect to resolve in detail. `dde23` is sufficiently robust that it can solve this problem in a straightforward way. However, ignoring discontinuities can get you into trouble even when solving an ODE [17], much less a DDE. You can be much more confident of your numerical results if you arrange that the solver is always integrating an equation with smooth coefficients. This can be done here by letting the parameter `state` be the value the solver is to use for $\mathrm{sign}(\Delta)$. With the given history, it is initialized to $+1$. We integrate until the event function $y(t-12) - x_b$ vanishes. The `'Events'` option is used to locate this event and terminate the integration then. The sign of `state` is changed and the integration restarted. This continues until the end of the interval is reached. It is straightforward to code the event location as

```
function [value,isterminal,direction] = exam7e(t,y,Z,state)
xb = -0.427;
value = Z - xb;
isterminal = 1;
direction = 0;
```

and with `state` defined in `exam7.m` as described, the evaluation of the DDE is coded as

```
function v = exam7f(t,y,Z,state)
xb = -0.427;
a = 0.16;
epsilon = 0.02;
u = 0.5;
tau = 1;
Delta = Z - xb;
v = (-y + pi * (a + epsilon*state - u*sin(Delta)^2)) / tau;
```

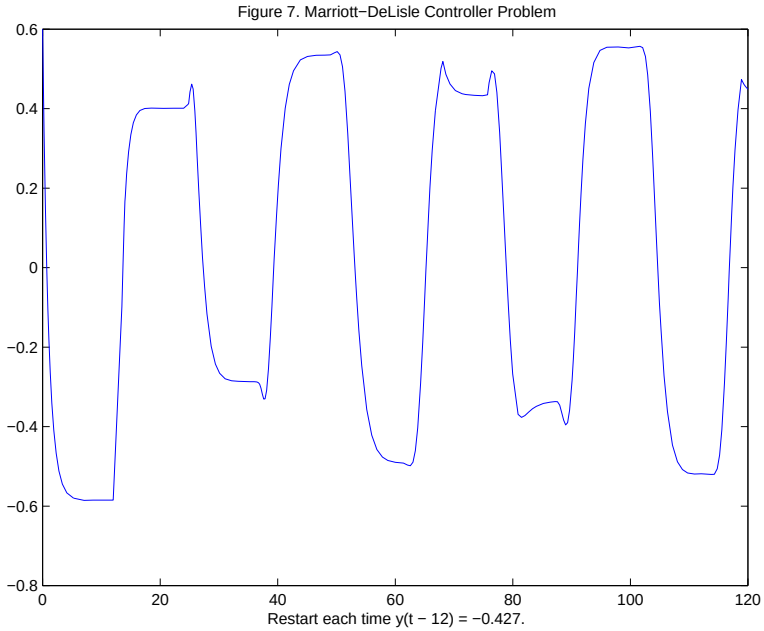
The event location capability deals with the issue of finding when the equations change, but there is a matter special to DDEs on restarting, the history. On a restart, `dde23` accepts the previously computed solution structure as history. `dde23` updates the information in the solution structure each time it is called, so the solution returned is always valid from the initial to the last point reached in the integration, namely `sol.x(end)`. It is convenient to compare this point to the end of the interval of integration and perform the restarts in a `while` loop. In this loop the solution structure of one integration is used as the history for the next until the run is completed.

With `exam7e` and `exam7f`, the program `exam7.m` to compute and plot the solution is

```
state = +1;
opts = ddeset('Events',@exam7e);
sol = dde23(@exam7f,12,0.6,[0, 120],opts,state);
while sol.x(end) < 120
  fprintf('Restart at %5.1f.\n',sol.x(end));
  state = - state;
  sol = dde23(@exam7f,12,sol,[sol.x(end), 120],opts,state);
end

plot(sol.x,sol.y);
title('Figure 7. Marriott-DeLisle Controller Problem')
xlabel('Restart each time y(t - 12) = -0.427.');
```

The program prints out a message about restarts and where they occur. The resulting output is

```
Restart at  14.0.
Restart at  24.9.
Restart at  68.1.
Restart at  75.7.
Restart at 118.9.
```



Exercise 7

The equations of the Marchuk immunology model discussed in [6] are

$$\begin{aligned} y_1'(t) &= (h_1 - h_2 y_3(t))\, y_1(t) \\ y_2'(t) &= \xi(y_4)\, h_3\, y_3(t-\tau) y_1(t-\tau) - h_5(y_2(t) - 1) \\ y_3'(t) &= h_4(y_2(t) - y_3(t)) - h_8 y_3(t) y_1(t) \\ y_4'(t) &= h_6 y_1(t) - h_7\, y_4(t) \end{aligned}$$

Here the coefficient

$$\xi(y_4) = \begin{cases} 1 & \text{if } y_4 \le 0.1 \\ (1 - y_4)\, 10/9 & \text{if } 0.1 < y_4 \le 1 \end{cases}$$

is continuous, but has a jump in its first derivative when $y_4(t) = 0.1$, which leads to a jump in a low order derivative of $y_2(t)$. The value of the delay is $\tau = 0.5$. The problem is solved on $[0, 60]$ with history $y_1(t) = \max(0, t + 10^{-6}), y_2(t) = 1, y_3(t) = 1, y_4(t) = 0$ for $t \le 0$. As was noted in Example 5, $y_1(t)$ has a jump in its first derivative at $t = -10^{-6}$ that propagates into the interval of integration. Figure 15.8 of [6] presents plots for $h_1 = 2, h_2 = 0.8, h_3 = 10^4, h_4 = 0.17, h_5 = 0.5, h_7 = 0.12, h_8 = 8$ and two values of h_6,

namely 10 and 300. Treat h_6 as a parameter in your program and try to reproduce the figure for $h_6 = 300$. For this you will have to plot the scaled components $10^4 y_1, y_2/2, y_3, 10y_4$ with `axis([0 60 -1 15.5])` . An array `yplot` of scaled values for plotting can be formed easily by

```
yplot = sol.y;
yplot(1,:) = 1e4*yplot(1,:);
```

and so forth. To solve this problem accurately over the whole interval, you will need to reduce the tolerances to, say, a relative tolerance of 10^{-5} and an absolute tolerance of 10^{-8}.

`dde23` is sufficiently robust that it can solve this problem in a straightforward way. However, you can be much more confident of the results if you ensure that the solver is always working with equations that have smooth coefficients. There are two kinds of discontinuities in this problem. As in Example 5, use the `'Jumps'` option to tell the solver about the discontinuity at $t = -10^{-6}$. As in Example 7, terminate the integration when the event function $y_4(t) - 0.1$ vanishes. Use a parameter `state` with value $+1$ if $y_4(t) \leq 0.1$ and -1 otherwise. The problem is to be solved with $y_4(0) = 0$, so initialize `state` to $+1$. Thereafter, each time that the solver returns, check whether you have reached the end of the interval. If `sol.x(end) < 60`, change the sign of `state` and call `dde23` again with the previous solution as history. In the function for evaluating the DDEs, set $\xi(y_4) = 1$ if `state` is $+1$ and $\xi(y_4) = (1 - y_4)\, 10/9$ otherwise.

Example 8

This example is much like Example 7 except that the solution itself is discontinuous. We restart at discontinuities, so the jump in the solution occurs at the initial point of an integration and can be handled as in Example 6.

A two-wheeled suitcase may begin to rock from side to side as it is pulled. When this happens, the person pulling it attempts to return it to the vertical by applying a restoring moment to the handle. There is a delay in this response that can affect significantly the stability of the motion. This is modeled by Suherman et alia [19] with the DDE

$$\theta''(t) + \mathrm{sign}(\theta(t))\gamma\cos(\theta(t)) - \sin(\theta(t)) + \beta\theta(t-\tau) = A\sin(\Omega t + \eta)$$

The equation is solved on $[0, 12]$ as a pair of first order equations with $y_1(t) = \theta(t), y_2(t) = \theta'(t)$. Figure 3 of [19] shows a plot of $y_1(t)$ against t and a plot

of $y_2(t)$ against $y_1(t)$ when $\gamma = 2.48, \beta = 1, \tau = 0.1, A = 0.75, \Omega = 1.37, \eta = \arcsin(\gamma/A)$ and the initial history is the constant vector zero.

A wheel hits the ground (the suitcase is vertical) when $y_1(t) = 0$ and the suitcase has fallen over when $|y_1(t)| = \pi/2$. The events are terminal and all are to be reported. The event function can be coded as

```
function [value,isterminal,direction] = exam8e(t,y,Z,state)
value = [y(1); abs(y(1))-pi/2];
isterminal = [1; 1];
direction = [0; 0];
```

As in Example 7, the parameter `state` seen in the event function is used in `exam8.m` to evaluate properly the discontinuous coefficient $\text{sign}(y_1(t))$ in the DDE. We initialize it to $+1$ and change its sign when `dde23` returns because $y_1(t)$ vanished. However, there are two event functions, so we must check the last entry in `sol.ie` to see if we should change the sign of `state`. With this, the DDEs can be coded as

```
function v = exam8f(t,y,Z,state)
ylag = Z(:,1);
v = zeros(2,1);

gamma = 0.248;
beta  = 1;
A = 0.75;
omega = 1.37;
eta = asin(gamma/A);

v(1) = y(2);
v(2) = sin(y(1)) - state*gamma*cos(y(1)) - beta*ylag(1) ...
      + A*sin(omega*t + eta);
```

When a wheel hits the ground, the integration is to be restarted with $y_1(t) = 0$ and $y_2(t)$ multiplied by the coefficient of restitution 0.913. The `'InitialY'` option is used for this purpose. The solution at all the mesh points is available as the field `sol.y` and in particular, the solution at the time of the event is the last column of this array, `sol.y(:,end)`. If the suitcase falls over, the run is terminated, so again we must check which event occurred. With `exam8e` and `exam8`, the program `exam8.m` to solve the problem and plot the solution in the phase plane is

```
state = +1;
opts = ddeset('RelTol',1e-5,'AbsTol',1e-5,'Events',@exam8e);
sol = dde23(@exam8f,0.1,[0; 0],[0 12],opts,state);

% Reference values for event locations:
ref = [4.516757065, 9.751053145, 11.670393497];
fprintf('\nKind of Event:                dde23   reference\n');
event = 0;
while sol.x(end) < 12
  event = event + 1;
  if sol.ie(end) == 1
     fprintf('A wheel hit the ground.   %10.4f  %10.6f\n',...
             sol.x(end),ref(event));
     state = - state;
     opts = ddeset(opts,'InitialY',[ 0; 0.913*sol.y(2,end)]);
     sol = dde23(@exam8f,0.1,sol,[sol.x(end) 12],opts,state);
  else
     fprintf('The suitcase fell over.   %10.4f  %10.6f\n',...
             sol.x(end),ref(event));
     break;
  end
end
plot(sol.y(1,:),sol.y(2,:))
title('Figure 8.  Suitcase problem.')
xlabel('\theta(t)')
ylabel('\theta''(t)')
```

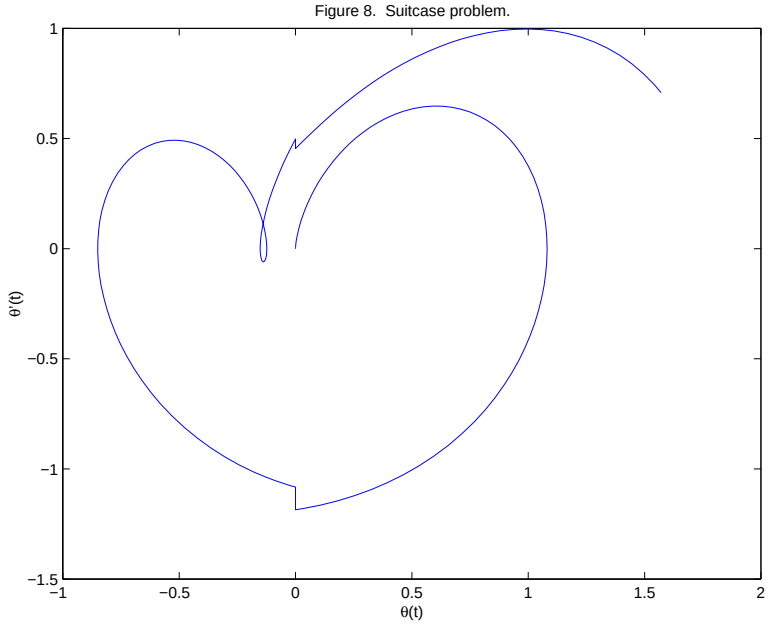
Note that `ddeset` can be used to change the value of an option or add an option, just as with `odeset`. The program reproduces the phase plane plot of Figure 3 in [19]. It also reports what kind of event occurred and the location of the event. Reference values were computed with the FORTRAN 77 code DKLAG5 [11] used in [19] and verified with its successor DKLAG6 [4]. Having written the three solvers, we can fairly say that it is very much easier to solve this problem in MATLAB with `dde23`. The program results in the output

```
Kind of Event:                dde23   reference
A wheel hit the ground.      4.5168    4.516757
A wheel hit the ground.      9.7511    9.751053
```

```
The suitcase fell over.       11.6704    11.670393
```

The accuracy of the computed results is what we might expect for the specified error tolerances.

Figure 8. Suitcase problem.



After running this program, `sol.xe` is

```
0.0000     4.5168     4.5168     9.7511     9.7511    11.6704
```

This does not seem to agree with the event locations reported by the program. For instance, why is there an event at 0? That is because one of the event functions is y_1 and this component of the solution has initial value 0. As explained in Example 4, `dde23` locates this event, but does not terminate the integration because the terminal event occurs at the initial point. The first integration terminates at the first point after the initial point where $y_1(t^*) = 0$, namely $t^* = 4.5168$. The second appearance of 4.5168 in `sol.xe` is the same event at the initial point of the second integration. The same thing happens at 9.7511 and finally the event at 11.6704 tells us that the suitcase fell over and we are finished.

5 Additional Problems

This section presents a few problems for practice. They are taken from the literature and some are quite recent. A familiarity with the examples of

the previous section is assumed. Some hints are given and some output is provided both as a check and to show how the solutions behave. Complete solutions are provided as additional templates. These problems and their solutions show that interesting problems can be solved easily in MATLAB with `dde23`.

Problem 1

Hale [7] cites predator–prey models obtained by introducing a resource limitation on the prey and assuming the birth rate of predators responds to changes in the magnitude of the population y_1 of prey and the population y_2 of predators only after a time delay τ. Starting with the system of ODEs [13]

$$\begin{aligned} y_1'(t) &= a\,y_1(t) + b\,y_1(t)\,y_2(t) \\ y_2'(t) &= c\,y_2(t) + d\,y_1(t)\,y_2(t) \end{aligned}$$

we arrive in this way at a system of DDEs

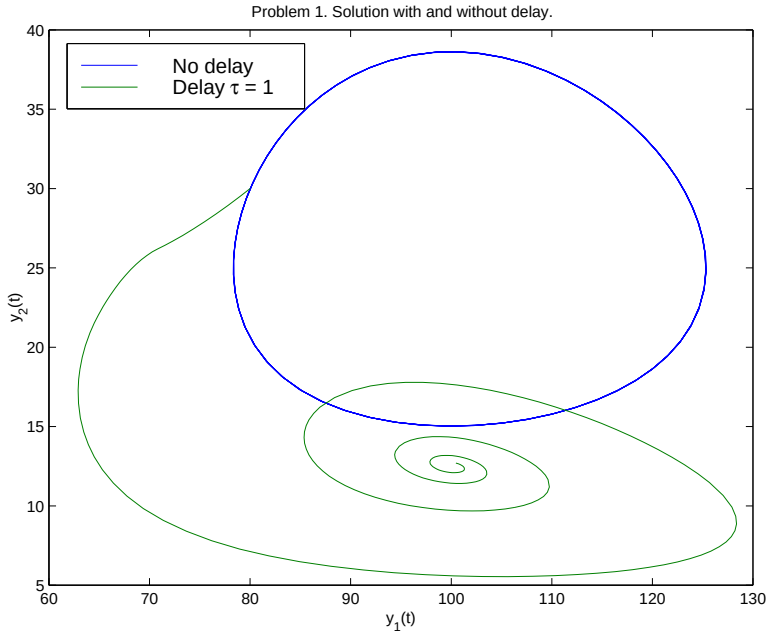
$$\begin{aligned} y_1'(t) &= a\,y_1(t)\left(1 - \frac{y_1(t)}{m}\right) + b\,y_1(t)\,y_2(t) \\ y_2'(t) &= c\,y_2(t) + d\,y_1(t-\tau)\,y_2(t-\tau) \end{aligned}$$

It is interesting to explore the effect of the delay, so let us solve both systems on $[0, 100]$ with initial value $y_1(0) = 80, y_2(0) = 30$ for the ODEs and the same vector as constant history for the DDEs. Suppose that the parameters $a = 0.25, b = -0.01, c = -1.00, d = 0.01$, and $m = 200$.

Recall that you solve ODEs with `dde23` by setting `lags` to `[]`. When this is done, the argument `Z` that `dde23` supplies to the functions it calls is the empty array. You can use this to code the evaluation of both sets of equations in the same function by testing `isempty(Z)` to find out which set to evaluate. A more straightforward approach is to use two functions for the two sets of equations. Solve the DDE with $\tau = 1$. Plot in one figure $y_2(t)$ against $y_1(t)$ for the two solutions. This phase plane plot of the solution of the ODEs should be a closed curve corresponding to a limit cycle. To achieve this you will need to tighten the error tolerances with a command like

```
opts = ddeset('RelTol',1e-5,'AbsTol',1e-8);
```

The figure makes the point that introducing a delay into an ODE model can have a profound effect on the solution. If you experiment with τ, you will find this to be true even for small delays. It is also interesting to remove the resource term $1 - y_1(t)/m$ and see how the orbits change as τ is changed.



Problem 2

This problem considers a cardiovascular model due to Ottesen [14] involving the arterial pressure, $P_a(t) = y_1(t)$, the venous pressure, $P_v(t) = y_2(t)$, and the heart rate, $H(t) = y_3(t)$. Ottesen studies conditions under which the delay causes qualitative differences in the solution and in particular, oscillations in $P_a(t)$. Delays $\tau = 1.0, 1.4, 3.9, 5.0, 7.5, 10$ are considered in [14]. There are a number of parameters in the model, so you might wish simply to declare them as global values rather than pass them through `dde23` to the function for evaluating the equations, or even to hard code them. Plot $y_1(t) = P_a(t)$ for several values of τ. You should find that the solutions obtained for different values of τ differ dramatically. Solve on $[0, 350]$ the equations

$$y_1'(t) \;=\; -\frac{1}{c_a R}\, y_1(t) + \frac{1}{c_a R}\, y_2(t) + \frac{1}{c_a}\, V_{str}\, y_3(t)$$

$$
\begin{aligned}
y_2'(t) &= \frac{1}{c_v R}\, y_1(t) - \left(\frac{1}{c_v R} + \frac{1}{c_v r}\right)\, y_2(t) \\
y_3'(t) &= f(T_s, T_p)
\end{aligned}
$$

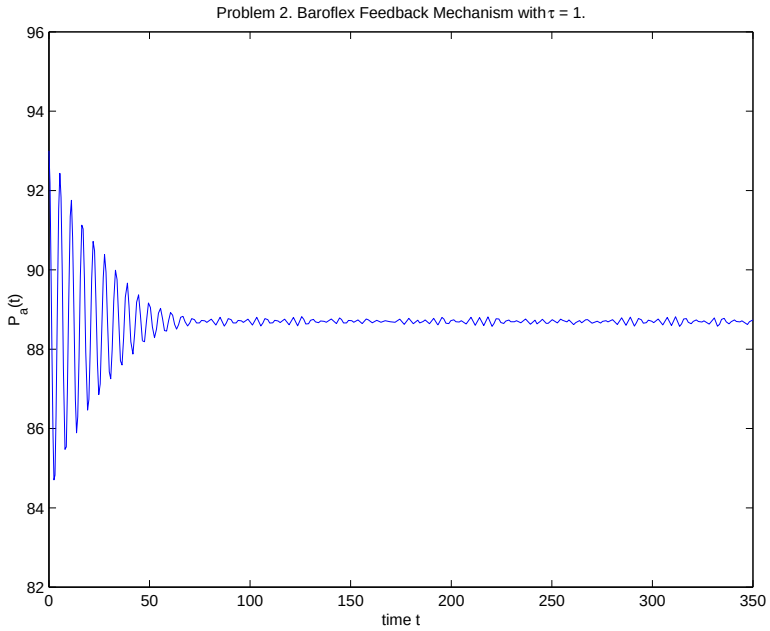
where

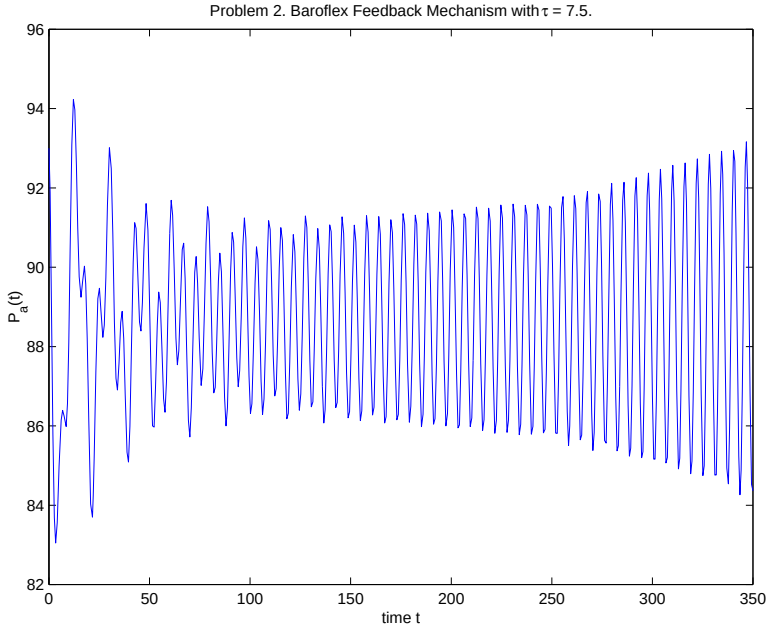
$$
\begin{aligned}
T_s &= \frac{1}{1 + \left(y_1(t-\tau)/\alpha_s\right)^{\beta_s}} \\
T_p &= \frac{1}{1 + \left(\alpha_p / y_1(t)\right)^{\beta_p}} \\
f(T_s, T_p) &= \frac{\alpha_H T_s}{1 + \gamma_H T_p} - \beta_H T_p.
\end{aligned}
$$

For $t \leq 0$, the solution has the constant value

$$
\begin{aligned}
y_1(t) &= P_0 \\
y_2(t) &= \left(\frac{1}{1 + R/r}\right)\, P_0 \\
y_3(t) &= \left(\frac{1}{R\, V_{str}}\right)\left(\frac{1}{1 + r/R}\right)\, P_0
\end{aligned}
$$

As in [14], use $c_a = 1.55, c_v = 519, R = 1.05, r = 0.068, V_{str} = 67.9, \alpha_0 = \alpha_s = \alpha_p = 93, \alpha_H = 0.84, \beta_0 = \beta_s = \beta_p = 7, \beta_H = 1.17, \gamma_H = 0, P_0 = 93$. The following figures for $\tau = 1$ and $\tau = 7.5$ show qualitatively different solutions.



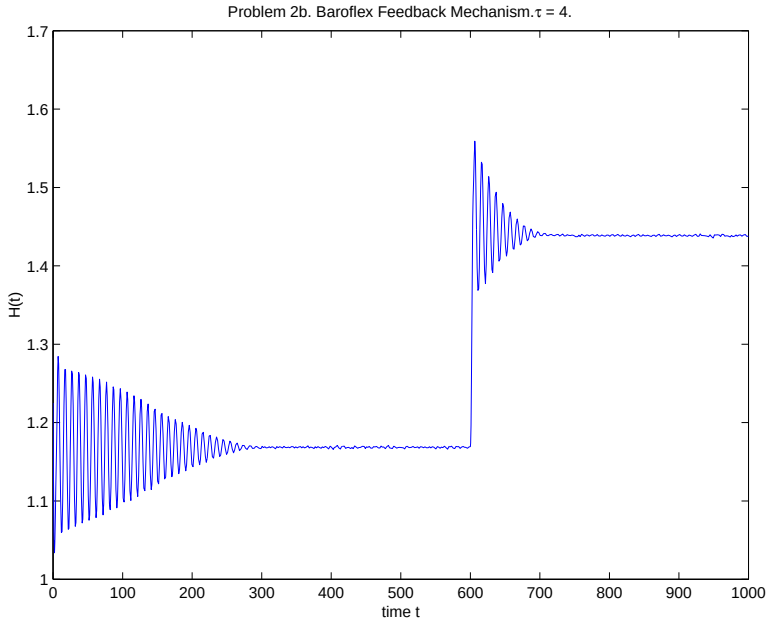


One of the figures of [14] shows the solution components when the peripheral pressure R is reduced exponentially from its value of 1.05 to 0.84 beginning at $t = 600$. For this computation the delay was 4 and the interval $[0, 1000]$. You can easily modify the previous program to solve this problem.

All you have to do is inform the solver of the low-order discontinuity at a known time by setting the value of the `'Jumps'` option to 600, modify the function for evaluating the DDEs to include

```
if t <= 600
   R = 1.05;
else
   R = 0.21 * exp(600-t) + 0.84;
end
```

and use the specified delay and interval. All the solution components are of interest. The figure shows the sharp change in the heart rate due to the change in R at $t = 600$.



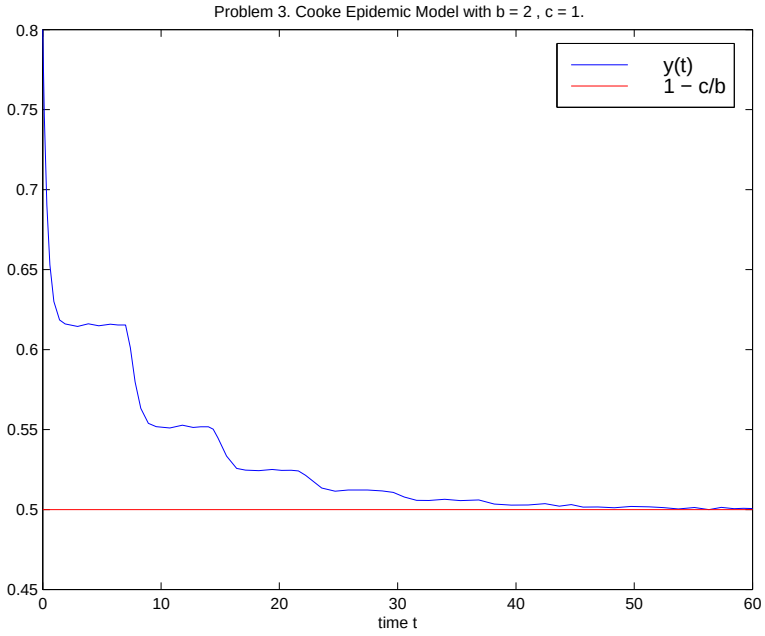
Problem 3

An epidemic model due to Cooke [8] describes the fraction $y(t)$ of a population which is infected at time t by the equation

$$y'(t) = b\,y(t-7)\,(1-y(t)) \;-\; c\,y(t)$$

Here b and c are positive constants. The equation is solved on $[0, 100]$ with history $y(t) = \alpha$ for $t \leq 0$. The constant α satisfies $0 < \alpha < 1$.

For all values of b and c, the solution $y(t) = 0$ is an equilibrium point. For $b > c$, the solution $y(t) = 1 - c/b$ is a second equilibrium point. Solve this DDE for different values of b, c, and α. Verify that if $b > c$, the solution approaches the second equilibrium point, and otherwise it approaches the zero equilibrium point. The long–term behavior of the solution is independent of the delay; you might want to verify this computationally. The figure shows the approach of the solution to the second equilibrium point when $b = 2$, $c = 1$, and $\alpha = 0.8$.



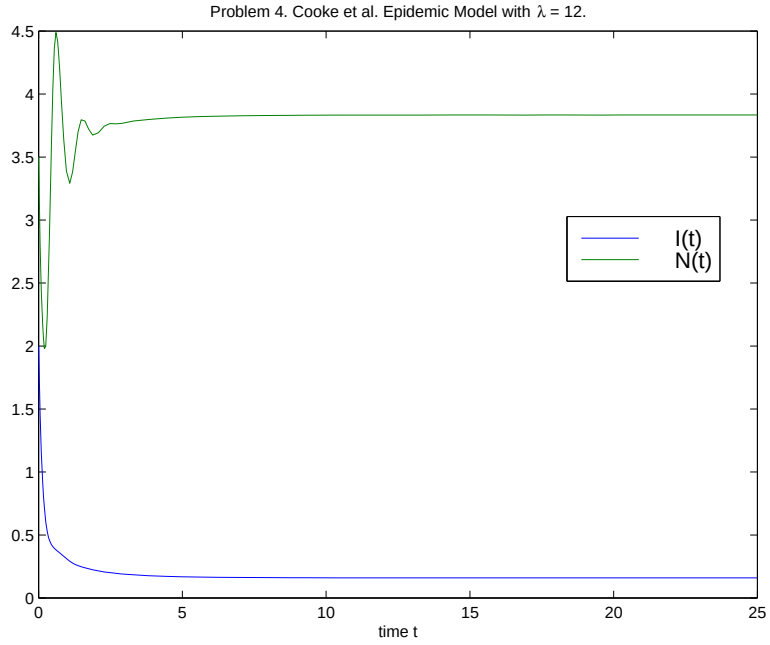
Problem 4

Another epidemic model due to Cooke et alia [3] that is more complicated than Problem 3 involves the size $I(t) = y_1(t)$ of an infected population and the total size $N(t) = y_2(t)$ of the population at time t. The equations are

$$\begin{aligned} y_1'(t) &= \lambda\,(y_2(t) - y_1(t))\,\frac{y_1(t)}{y_2(t)} \;-\; (d + \epsilon + \gamma)\; y_1(t) \\ y_2'(t) &= b\,e^{-a y_2(t-T)}\, y_2(t-T)\, e^{-d_1 T} \;-\; d y_2(t) \;-\; \epsilon y_1(t) \end{aligned}$$

They are solved on $[0, 25]$ with history $y_1(t) = 2, y_2(t) = 3.5$ for $t \le 0$ and parameter values $a = 1, b = 80, d = 1, d_1 = 1, \gamma = 0.5, \epsilon = 10, T = 0.2$.

As in Problem 2, it is convenient to pass the parameters to the function for evaluating the DDEs as global variables or to hard code them. In [3] the solution is investigated for a number of λ, so pass it as a parameter through `dde23`. Values $\lambda = 12, 15, 20, 28$ are of interest. You might find it interesting to compare your plots to those of Figure 4 in [3]. The following figure shows the case $\lambda = 12$.



Problem 5

A population growth model due to Cooke et alia [3] describes the population $y(t)$ at time t by the equation

$$y'(t) = b\, e^{-ay(t-T)}\, y(t-T)\, e^{-d_1 T} - dy(t)$$

Solve the equation on $[0, 25]$ with history $y(t) = 3.5$ for $t \leq 0$ for one or more of the data sets

1. $a = 1, d = 1, d_1 = 1, b = 20$
2. $a = 1, d = 1, d_1 = 1, b = 80$
3. $a = 1, d = 1, d_1 = 0, b = 20$

4. $a = 1, d = 1, d_1 = 0, b = 80$

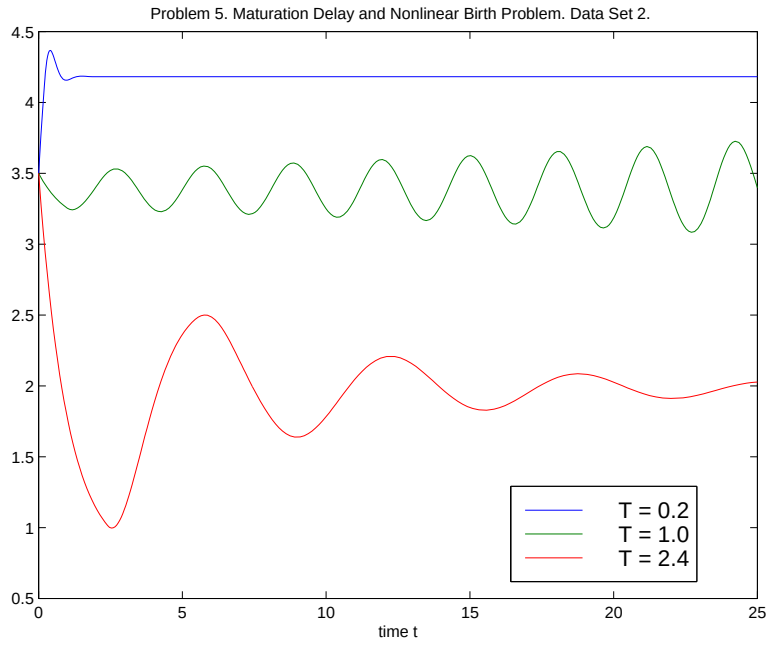
For each set of parameter values, solve the problem using three values of the delay, namely $T = 0.2, 1.0, 2.4$, and plot the solutions on the same figure. Structures can be indexed, so this can be coded as

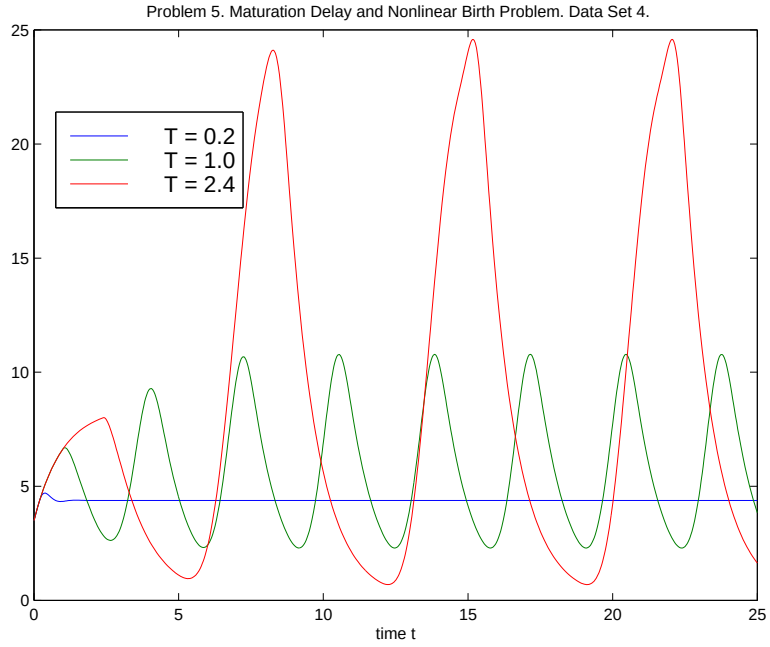
```
for i = 1:3
   T = Delays(i);
   sol(i) = dde23(@prob6f,T,3.5,[0, 25],opts);
end
```

On exit from the loop, the solution for the first delay is `sol(1).x,sol(1).y` and so forth. Note that T must be communicated to `prob6f` as a parameter or global variable because it appears in the equation. In the code fragment it is communicated as a global variable along with the parameters of the data set. You should use tolerances more stringent than the defaults, e.g.,

```
opts = ddeset('RelTol',1e-5,'AbsTol',1e-8);
```

You might find it interesting to compare your solutions with those of Figure 3 in [3]. The following figures show the solutions for two of the data sets. Obviously the delay has a profound effect on the solution.





References

[1] C.T.H. Baker, C.A.H. Paul, and D.R. Willé, A bibliography on the numerical solution of delay differential equations, Numer. Anal. Rept. No. 269, Maths. Dept., Univ. of Manchester, U.K., 1995.

[2] C.T.H. Baker, C.A.H. Paul, and D.R. Willé, Issues in the numerical solution of evolutionary delay differential equations, Adv. Comp. Math., 3 (1995) 171–196.

[3] K. Cooke, P. van den Driessche, and X. Zou, Interaction of maturation delay and nonlinear birth in population and epidemic models, J. Math. Biol., 39 (1999) 332–352.

[4] S.P. Corwin, D. Sarafyan, and S. Thompson, DKLAG6: A code based on continuously imbedded sixth order Runge–Kutta methods for the solution of state dependent functional differential equations, Appl. Num. Math., 24 (1997) 319–333.

[5] J.D. Farmer, Chaotic attractors of an infinite–dimensional dynamical system, Physica D, 4 (1982) 366–393.

[6] E. Hairer, S.P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I*, Springer-Verlag, Berlin, 1987.

[7] J. Hale, *Functional Differential Equations*, Springer-Verlag, Berlin, 1971.

[8] N. MacDonald, *Time Lags in Biological Models*, Springer-Verlag, Berlin, 1978.

[9] C. Marriott and C. DeLisle, Effects of discontinuities in the behavior of a delay differential equation, Physica D, 36 (1989) 198–206.

[10] K.W. Neves, Automatic integration of functional differential equations: an approach, ACM TOMS, 1 (1975), 357–368.

[11] K.W. Neves and S. Thompson, Software for the numerical solution of systems of functional differential equations with state dependent delays, Appl. Num. Math., 9 (1992), 385–401.

[12] H.J. Oberle and H.J. Pesch, Numerical treatment of delay differential equations by Hermite interpolation, Numer. Math., 37 (1981) 235–255.

[13] J.M. Ortega and W.G. Poole, *An Introduction to Numerical Methods for Differential Equations*, Pitman Publishing Inc., Marshfield, Massachusetts, 1981.

[14] J.T. Ottesen, Modelling of the Baroflex–Feedback Mechanism With Time–Delay, J. Math. Biol., 36 (1997), 41–63.

[15] C.A.H. Paul, A user-guide to Archi, Numer. Anal. Rept. No. 283, Maths. Dept., Univ. of Manchester, U.K., 1995.

[16] L.F. Shampine and M.W. Reichelt, The MATLAB ODE suite, SIAM J. Sci. Comput., 18 (1997) 1–22.

[17] L.F. Shampine and S. Thompson, Event location for ordinary differential equations, Comp. & Maths. with Appls., 39 (2000) 43–54.

[18] L.F. Shampine and S. Thompson, Solving DDEs in MATLAB, Appl. Numer. Math., 37 (2001) 441–458.

[19] S. Suherman, R.H. Plaut, L.T. Watson, and S. Thompson, Effect of human response time on rocking instability of a two–wheeled suitcase, J. of Sound and Vibration, 207 (1997) 617–625.

[20] L. Tavernini, *Continuous–Time Modeling and Simulation*, Gordon and Breach, Amsterdam, 1996.

[21] D.R. Willé and C.T.H. Baker, DELSOL – a numerical code for the solution of systems of delay–differential equations, Appl. Numer. Math., 9 (1992) 223–234.