Delay Differential Equations

Part II: Time– and State–dependent Lags

L.F. Shampine
Department of Mathematics
Southern Methodist University
Dallas, Texas 75275
shampine@smu.edu
www.faculty.smu.edu/shampine
Time– and State–dependent Lags

Now we consider DDEs of the form

\[ y'(t) = f(t, y(t), y(d_1), \ldots, y(d_k)) \]

with delays \( d_j(t, y(t)) \leq t \). This is the form used by the programs \texttt{ddesd} and \texttt{dde_solver}.

Some prefer the equivalent form

\[ y'(t) = f(t, y(t), y(t - \tau_1), \ldots, y(t - \tau_k)) \]

with lags \( \tau_j(t, y(t)) \geq 0 \). The driver \texttt{ezddesd} extends the design of \texttt{ezdde23} to these problems.
Method of Steps

If the lags are bounded away from zero, $\tau_j \geq \alpha > 0$, the method of steps can be applied much as with constant lags to prove existence and uniqueness. If some lag vanishes, the problem is said to be **singular**. Existence and uniqueness are then problematic.

If **all** the delay functions $d_j(t, y(t)) \geq a$, we say this is an initial value DDE (IVDDE) because $y(a)$ alone is enough to specify the solution.

The IVDDE $y'(t) = y(t^2)$ for $0 \leq t \leq 1$ is interesting. There is a second singular point at $t = 1$ where $t^2 = t$. This is not a DDE for $t > 1$ because the “delay” $t^2 > t$. 
“Short” Lags

With multiple delays, the effects of short lags last longer than you might think. Suppose we start at $t = 0$ and there are two lags, 1 and 0.001. The jump in $y'(0)$ propagates to 0.001, 0.002, 0.003, ... The numerical method is affected only by the first few discontinuities. However, there are again low-order discontinuities at 1.001, 1.002, 1.003, ... 

Handling short lags efficiently is critical to solving singular and near-singular problems. In particular, it is simply not possible to solve an IVDDE without special provision for short lags.
Propagation of Discontinuities

As with constant lags, there is generally a jump in the first derivative at the initial point that is propagated by the lags. For retarded DDEs, the order of the discontinuity increases each time it is propagated. For neutral DDEs jumps in the first derivative persist.

A discontinuity at $t^*$ has an effect in

$$y'(t) = f(t, y(t), y(t - \tau_1), \ldots, y(t - \tau_k))$$

at times $t > t^*$ such that

$$t - \tau_j(t, y(t)) = t^*$$
Ignore Discontinuities

Some codes do not track discontinuities because it is so expensive. This is an option in ARCHI and DKLAG6 (and its successor dde_solver).

The codes assume that a low-order discontinuity will result in a failed step. This is dangerous because error estimators assume $y(t)$ is sufficiently smooth—the code may accept an inaccurate result.

The codes assume that the step size selection algorithms will locate discontinuities well enough to take a step. This is usually a good assumption because quality algorithms assume little smoothness. But, discontinuities can result in a lot of failed steps.
Control of Defect

Some codes control the defect (residual), the amount by which an approximation $S(t)$ fails to satisfy the DDE. Estimates of this error may not be affected much by the smoothness of $y(t)$.

- DDVERK monitors step sizes to spot a discontinuity. If it finds one, it uses bisection to place a mesh point there.

- ddesd uses an integral measure of size to get a robust estimate of the residual. The step size algorithm of ode23 deals well with discontinuities of low order.

These codes are pretty reliable, but discontinuities can result in a lot of failed steps.
The system

\[
y'_1(t) = y_2(t) \\
y'_2(t) = -y_2(e^{1-y_2(t)}) y_2(t)^2 + e^{1-y_2(t)}
\]

has the analytical solution \( y_1(t) = \log(t), \ y_2(t) = 1/t. \)
Integrate over \([0.1, 5]\) with analytical solution as history.

There is a state–dependent delay. The DDE is singular at \( t = 1 \) because \( d_1(t) = \exp(1 - y_2(1)) = \exp(0) = 1 = t. \)

```matlab
sol = ddesd(@ddex3de,@ddex3delay,...
            @ddex3hist,[0.1, 5]);
```
Only One Singular Point

\[ d(t, y) \]

\[ t \]
function v = ddex3hist(t)
% History function for DDEX3.
    v = [ log(t); 1./t];

function d = ddex3delay(t,y)
% State dependent delay function for DDEX3.
    d = exp(1 - y(2));

function dydt = ddex3de(t,y,Z)
% Differential equations function for DDEX3.
    dydt = [ y(2); -Z(2)*y(2)^2*exp(1 - y(2))];
D1 problem of Enright and Hayashi

- Blue line with circles: $y_1$, exact
- Green line with circles: $y_2$, exact
- Red circles: $y_1$, ddesd
- Cyan circles: $y_2$, ddesd

Axes:
- Y-axis: solution $y$
- X-axis: time $t$
Locating Discontinuities, $\tau_j(t)$

For each lag $\tau_j(t)$ and each point of discontinuity $t^*$, we have to solve numerically the algebraic equation

$$t - \tau_j(t) = t^*$$

Typically codes solve these equations about as accurately as possible. It may be necessary to solve a great many equations.

In principle we can work out the discontinuity tree in advance and then integrate much as with constant delays. Hardly any codes do this. Instead they treat this as a special case of lags $\tau_j(t, y(t))$. 
Locating Discontinuities, $\tau_j(t, y(t))$

ARCHI and dde_solver track discontinuities. For each $\tau_j(t, y(t))$ and each point of discontinuity $t^*$, we have to solve numerically the algebraic equation

$$t - \tau_j(t, y(t)) = t^*$$

This is complicated and expensive because it depends on $y(t)$, which we do not know.

At each step, first check for discontinuities in $[t_n, t_{n+1}]$ using extrapolation for $y(t)$. If there is a change of sign, locate the first discontinuity. Should iterate on approximation to $y(t)$, but may or may not do this. Iteration is an option in ARCHI.
Neutral DDEs

The numerical solution of neutral DDEs like

\[ y'(t) = f(t, y(t), y(d_1), \ldots, y(d_k), y'(d_{k+1}), \ldots, y'(d_m)) \]

is very much harder because discontinuities do not smooth out. ARCHI, DDVERK, and dde_solver attempt to solve neutral DDEs, but the solution of neutral DDEs is still a research question.

We discuss briefly a new approach implemented in an extension of ddesd called ddeNsd. First, however, we discuss a new issue for IVDDEs of neutral type.
Consistent Initial Conditions

On the interval $[0, 1]$ the neutral DDE

$$y'(t) = 2 \cos(2t) y(t/2)^2 \cos(t) + \log(y'(t/2)) - \log(2 \cos(t)) - \sin(t)$$

is an IVDDE because $d(t, y(t)) = t/2 \geq 0$. There is a solution $y(t) = e^{\sin(2t)}$ with $y(0) = 1$ and $y'(0) = 2$.

Suppose that $y(0) = 1$. Let $t \to 0$ in the DDE to see that $y'(0)$ must then satisfy

$$y'(0) = 2 + \log(y'(0)) - \log(2)$$

Such a $y'(0)$ is consistent with the given $y(0)$. 
User Interface

This IVDDE has another solution with
\[ y'(0) = -LambertW(-2e^{-2}) = 0.406 \ldots \]

We must provide consistent initial conditions to define which solution the solver is to compute.

Unlike typical codes for neutral DDEs, \texttt{ddeNsd} does not ask for \( \phi'(t) \). But, for an IVDDE you must supply \( y(a) \) and \( y'(a) \) in the history argument as a cell array \{ya,ypa\}.

With default tolerances, the example is solved with
\[
\text{sol} = \text{ddeNsd}(\text{@dde},\text{@delays},{1,2},[0,1]);
\]
Only 13 steps were needed to compute an approximation with maximum error of \( 4.7 \times 10^{-3} \).
For a “small” $\delta > 0$, we approximate a neutral DDE (NDDE) like

$$y'(t) = f(t, y(t), y(t - \tau), y'(t - \tau))$$

with a retarded DDE (RDDE),

$$u'(t) = f\left(t, u(t), u(t - \tau), \frac{u(t - \tau) - u(t - \tau - \delta)}{\delta}\right)$$

The underlying code for RDDEs must be able to cope with discontinuities that are very close.

This is somewhat like using artificial dissipation for shocks when solving hyperbolic PDEs: The NDDE has jumps in $y'(t)$ and after the initial one, the RDDE does not.
Accuracy and Choice of $\delta$

The solutions of the NDDE and RDDE differ by $O(\delta)$.
Because the solution of the NDDE has jumps in its first derivative and the solution of the RDDE does not, we can prove only that the first derivatives differ by $O(\delta)$ except in short intervals that contain the jumps.

To get an accurate difference approximation, the lags $\delta_j$ must depend on the delays $t - \tau_j$, so we must use ddesd rather than dde23. (Thompson has experimented quite successfully with the Fortran 90 dde_solver.) There are difficulties with IVDDEs and precision.

For the convenience of users, a quantity has been hard-coded that restricts ddeNsd to moderate accuracy.
Example P2p1p4

Solve on $[0, 4]$ with constant history $y(t) = 1$ the NDDE

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

With default tolerances and using options for constant lag and constant history, this is done with

```matlab
sol = ddeNsd(@dde,1,1,[0,4]);
```

In an obvious extension of the design of `ddesd`,

```matlab
function dydt = dde(t,y,Z,ZP)
    dydt = y + ZP;
end % dde
Output of Program
Comments on Output

• The discontinuity in \( y'(0) \) propagates to 1, 2, 3, . . . , which explains the clustering of mesh points at the integers.

• The clustering is visibly less tight as the integration proceeds because for the approximating RDDE, the discontinuities increase in order at successive integers.

• With default tolerances, the solver took 40 steps and the maximum \textit{relative} error at mesh points was \( 2.5 \times 10^{-4} \).
Further Reading

Some Useful URLs

- A list of publicly available software for DDEs in several computer languages

- Material about the MATLAB DDE solvers and the Fortran 90 DDE solver dde_solver
  www.radford.edu/ thompson/webddes/
  www.radford.edu/ thompson/ffddes/index.html

- Material about ezdde23/ezddesd and the ddeNsd program for neutral DDEs
  www.faculty.smu.edu/shampine/current.html
Fortran 77 Solvers Cited

