Delay Differential Equations

Delay differential equation initial value problem solvers

Functions

dde23	Solve delay differential equations (DDEs) with constant delays
ddesd	Solve delay differential equations (DDEs) with general delays
ddensd	Solve delay differential equations (DDEs) of neutral type
ddeget	Extract properties from delay differential equations options structure
ddeset	Create or alter delay differential equations options structure
deval	Evaluate differential equation solution structure

Examples and How To

DDE with Constant Delays

This example shows how to use dde23 to solve a system of DDEs with constant delays.

State-Dependent Delay Problem

This example shows how to use ddesd to solve a system of two DDEs with a state-dependent delay.

Cardiovascular Model with Discontinuities

This example shows how to use dde23 to solve a cardiovascular model that has a discontinuous derivative.

DDE of Neutral Type

This example shows how to use ddensd to solve a neutral DDE.

Initial Value DDE of Neutral Type

This example shows how to use ddensd to solve an initial value DDE.

Concepts

Types of DDEs

Solve delay differential equations.

Discontinuities in DDEs

Communicate discontinuities to the solver using an options structure.

DDE with Constant Delays

This example shows how to use `dde23` to solve a system of DDEs with constant delays.

Click `ddex1.m` or type `edit ddex1.m` in a command window to view the code for this example in an editor.

The differential equations are:

$$y_1'(t) = y_1(t-1)$$
$$y_2'(t) = y_1(t-1) + y_2(t-0.2)$$
$$y_3'(t) = y_2(t).$$

The history of this problem is constant:

$$y_1(t) = 1$$
$$y_2(t) = 1$$
$$y_3(t) = 1$$

for $t \le 0$.

1. Create a new program file in the editor. This file will contain a main function and two local functions.
2. Define the first-order DDE as a local function.

```
function dydt = ddex1de(t,y,Z)
  ylag1 = Z(:,1);
  ylag2 = Z(:,2);
  dydt = [ylag1(1); ylag1(1)+ylag2(2); y(2)];
end
```

3. Define the solution history as a local function.

```
function S = ddex1hist(t)
  S = ones(3,1);
end
```

4. Define the delays, $\tau_1,\ldots,\tau_k$ in the main function.

```
lags = [1,0.2];
```

5. Solve the DDE by calling `dde23` in the main function. Pass the DDE function, the delays, the solution history, and interval of integration, `[0,5]`, as inputs.

```
sol = dde23(@ddex1de,lags,@ddex1hist,[0,5]);
```

The `dde23` function produces a continuous solution over the whole interval of integration $[t_0,t_f]$.

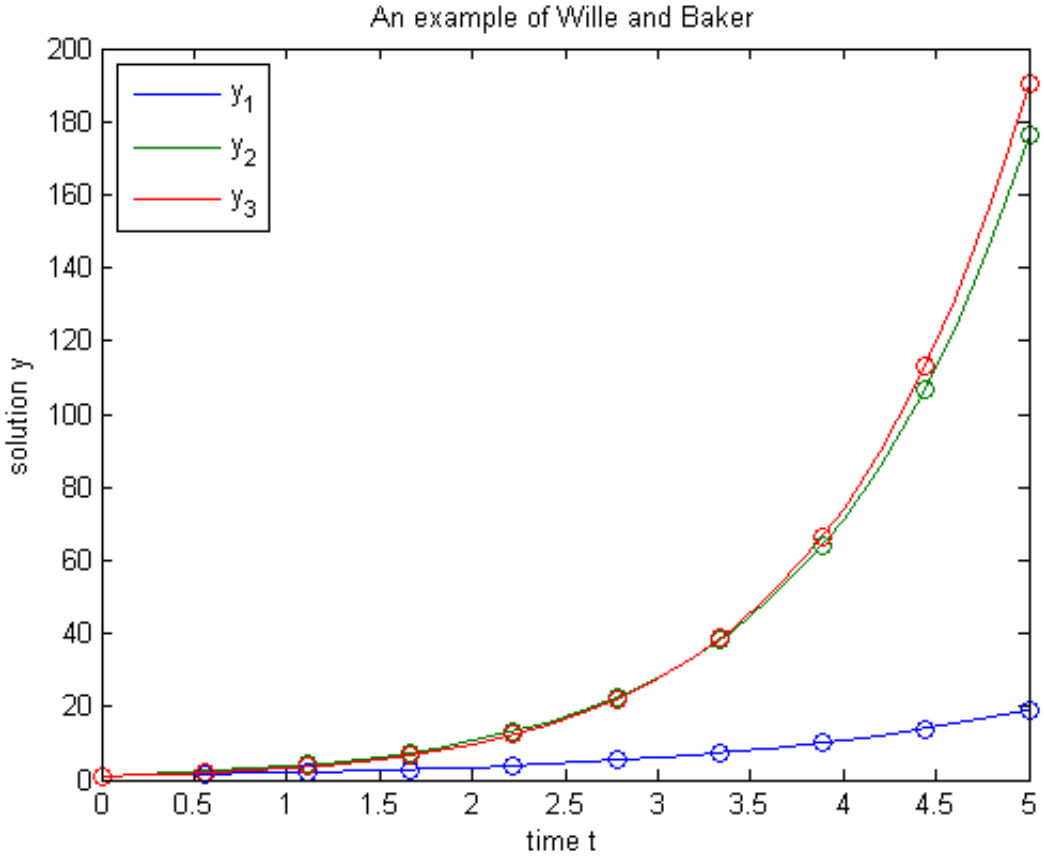
6. Plot the solution returned by `dde23`. Add this code to your main function.

```
plot(sol.x,sol.y);
title('An example of Wille and Baker');
xlabel('time t');
ylabel('solution y');
legend('y_1','y_2','y_3','Location','NorthWest');
```

7. Evaluate the solution at 10 equally spaced points over the interval of integration. Then plot the results on the same axes as `sol.y`. Add this code to the main function.

```
tint = linspace(0,5,10);
Sint = deval(sol,tint)
hold on
plot(tint,Sint,'o');
```

8. Run your program to generate and plot the results.



Related Examples

- Create Functions in Files
- Run Functions in the Editor

More About

- Create Function Handle

State-Dependent Delay Problem

This example shows how to use `ddesd` to solve a system of two DDEs with a state-dependent delay. This system of DDEs was used as a test problem by Enright and Hayashi [1].

Click `ddex3.m` or type `edit ddex3.m` in a command window to view the complete code for this example in an editor.

The equations for this system are:

$$y_1'(t) = y_2(t)$$
$$y_2'(t) = -y_2(e^{(1-y_2(t))}) \cdot y_2(t)^2 \cdot e^{(1-y_2(t))}.$$

The analytical solution

$$y_1(t) = \log(t)$$
$$y_2(t) = 1/t$$

is used as the history for $t \le 0.1$ and the equations are solved on `[0.1, 5]` with `ddesd` rather than `dde23`. The `ddesd` function is appropriate in this case because the first factor in the second equation has the form $y_2(d(y))$ with a delay that depends on the second component of the solution.

1. Create a new program file in the editor. This file will contain a main function and three local functions.
2. Code the system of DDEs as a local function.

```
function dydt = ddex3de(t,y,Z)
  dydt = [y(2); -Z(2)*y(2)^2*exp(1 - y(2))];
end
```

3. Define the delay as a local function.

```
function d = ddex3delay(t,y)
  d = exp(1 - y(2));
end
```

4. Define the solution history as a local function.

```
function v = ddex3hist(t)
  v = [log(t); 1./t];
end
```

5. Define the interval of integration and solve the system. Add this code to the main function in your program file.

```
tspan = [0.1 5];
sol = ddesd(@ddex3de,@ddex3delay,@ddex3hist,tspan);
```

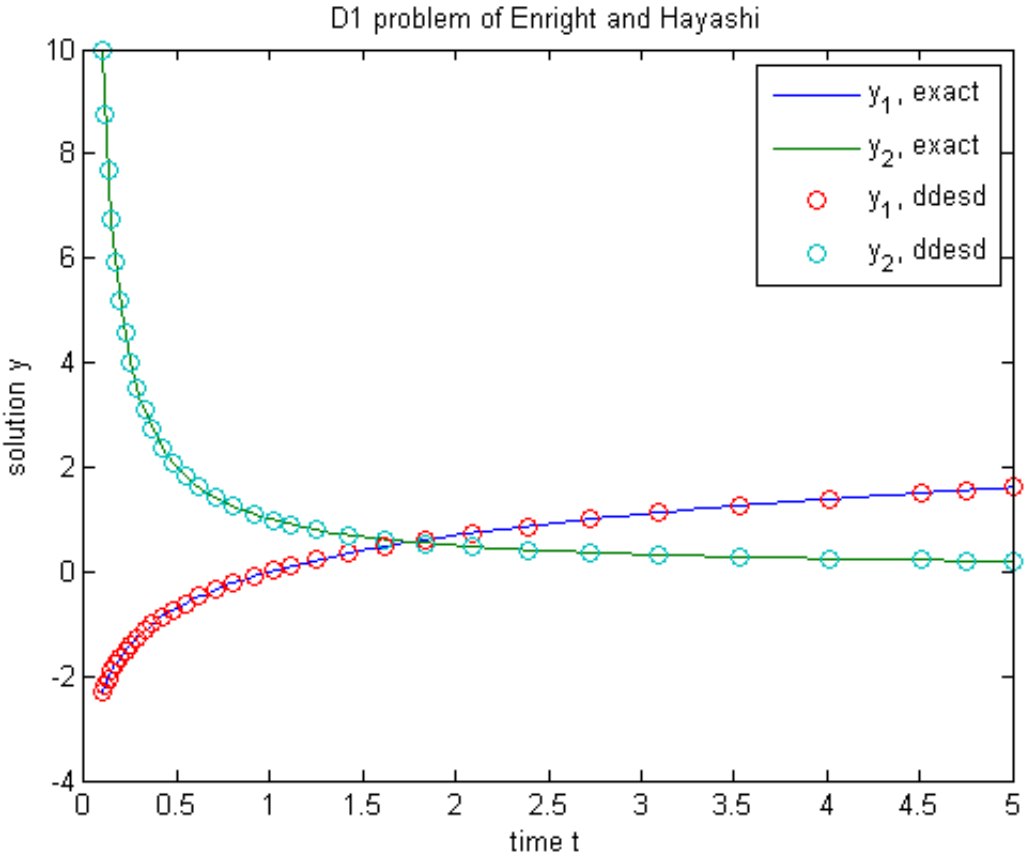
6. Use the history function to calculate the analytical solution within the integration interval. Add this code to the main function.

```
texact = linspace(0.1,5);
yexact = ddex3hist(texact);
```

7. Plot the numerical solution on the same axes as the analytical solution. Add this code to the main function.

```
figure
plot(texact,yexact,sol.x,sol.y,'o')
legend('y_1, exact','y_2, exact','y_1, ddesd','y_2, ddesd')
xlabel('time t')
ylabel('solution y')
title('D1 problem of Enright and Hayashi')
```

8. Run your program to generate and plot the results.



References

[1] Enright, W.H. and H. Hayashi. "The Evaluation of Numerical Software for Delay Differential Equations." In *Proceedings of the IFIP TC2/WG2.5 working conference on Quality of numerical software: assessment and enhancement*. (R.F. Boisvert, ed.). London, UK: Chapman & Hall, Ltd., pp. 179-193.

Related Examples

- Create Functions in Files
- Run Functions in the Editor

More About

- Create Function Handle

Cardiovascular Model with Discontinuities

This example shows how to use `dde23` to solve a cardiovascular model that has a discontinuous derivative as presented by Ottesen [1].

Click `ddex2.m` or type `edit ddex2.m` in a command window to view the code for this example in an editor.

This is a problem with 1 delay, constant history, and 3 differential equations with 14 physical parameters. The system is heavily influenced by peripheral pressure, *R*, which decreases exponentially from 1.05 to 0.84, beginning at *t* = 600. As a result, the system has a discontinuity in a low-order derivative at *t* = 600.

1. Create a new program file in the editor. This file will contain a main function and a nested function. The main function accepts no inputs and returns no outputs.
2. Define the physical parameters. Add this code to the main function.

```
p.ca     = 1.55;
p.cv     = 519;
p.R      = 1.05;
p.r      = 0.068;
p.Vstr   = 67.9;
p.alpha0 = 93;
p.alphas = 93;
p.alphap = 93;
p.alphaH = 0.84;
p.beta0  = 7;
p.betas  = 7;
p.betap  = 7;
p.betaH  = 1.17;
p.gammaH = 0;
```

3. Define the solution history. Add this code to the main function.

```
P0 = 93;
Paval = P0;
Pvval = (1 / (1 + p.R/p.r)) * P0;
Hval = (1 / (p.R * p.Vstr)) * (1 / (1 + p.r/p.R)) * P0;
history = [Paval; Pvval; Hval];
```

4. Define the delay, `tau`. Add this code to the main function.

```
tau = 4;
```

5. Define the location of discontinuity, which occurs at *t* = 600. Add this code to the main function.

```
options = ddeset('Jumps',600);
```

When your DDE has discontinuities in low-order derivatives, and you know the locations in advance, it is better to use `ddeset` with the `Jumps` property.

6. Solve the DDE over the interval [0, 1000]. Add this code to the main function.

```
sol = dde23(@ddex2de,tau,history,[0,1000],options);
```

The function, `@ddex2de`, which defines the system of DDEs, is the first input argument. You define this function in 8.

7. Plot the solution. Add this code to the main function.

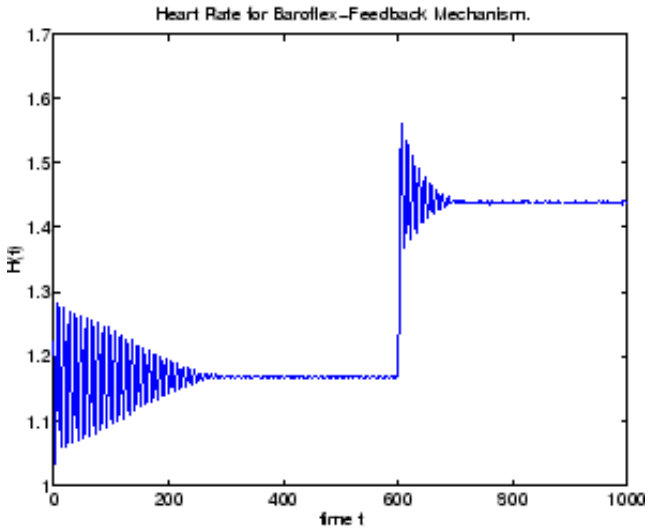
```
figure
plot(sol.x,sol.y(3,:))
title('Heart Rate for Baroflex-Feedback Mechanism.')
xlabel('time t')
ylabel('H(t)')
```

8. Define the system of DDEs as a nested function inside the main function.

```
function dydt = ddex2de(t,y,Z)
    if t <= 600
      p.R = 1.05;
    else
      p.R = 0.21 * exp(600-t) + 0.84;
    end
    ylag = Z(:,1);
    Patau = ylag(1);
    Paoft = y(1);
    Pvoft = y(2);
    Hoft  = y(3);

    dPadt = - (1 / (p.ca * p.R)) * Paoft ...
            + (1/(p.ca * p.R)) * Pvoft ...
            + (1/p.ca) * p.Vstr * Hoft;

    dPvdt = (1 / (p.cv * p.R)) * Paoft...
            - ( 1 / (p.cv * p.R)...
            + 1 / (p.cv * p.r) ) * Pvoft;

    Ts = 1 / ( 1 + (Patau / p.alphas)^p.betas );
    Tp = 1 / ( 1 + (p.alphap / Paoft)^p.betap );

    dHdt = (p.alphaH * Ts) / (1 + p.gammaH * Tp) ...
           - p.betaH * Tp;

    dydt = [ dPadt; dPvdt; dHdt];
  end
```

This function is nested so that the main function can access the 14 parameters defined in 2.

9. Run your program to calculate the solution and display the plot.



References

[1] Ottesen, J. T. "Modelling of the Baroflex-Feedback Mechanism with Time-Delay." *J. Math. Biol.* Vol. 36, Number 1, 1997, pp. 41–63.

Related Examples

- Create Functions in Files
- Run Functions in the Editor

More About

- Create Function Handle

DDE of Neutral Type

This example shows how to use `ddensd` to solve the neutral DDE presented by Paul [1] for $0 \le t \le \pi$.

Click `ddex4.m` or type `edit ddex4.m` in a command window to view the code for this example in an editor.

The equation is

$$y'(t) = 1 + y(t) - 2y(t/2)^2 - y'(t - \pi)$$

with history:

$$y(t) = \cos(t) \text{ for } t \le 0.$$

1. Create a new program file in the editor. This file will contain a main function and four local functions.
2. Define the first-order DDE as a local function.

```
function yp = ddefun(t,y,ydel,ypdel)
    yp = 1 + y - 2*ydel^2 - ypdel;
end
```

3. Define the solution delay as a local function.

```
function dy = dely(t,y)
    dy = t/2;
end
```

4. Define the derivative delay as a local function.

```
function dyp = delyp(t,y)
    dyp = t-pi;
end
```

5. Define the solution history as a local function.

```
function y = history(t)
    y = cos(t);
end
```

6. Define the interval of integration and solve the DDE using the `ddensd` function. Add this code to the main function.

```
tspan = [0 pi];
sol = ddensd(@ddefun,@dely,@delyp,@history,tspan);
```

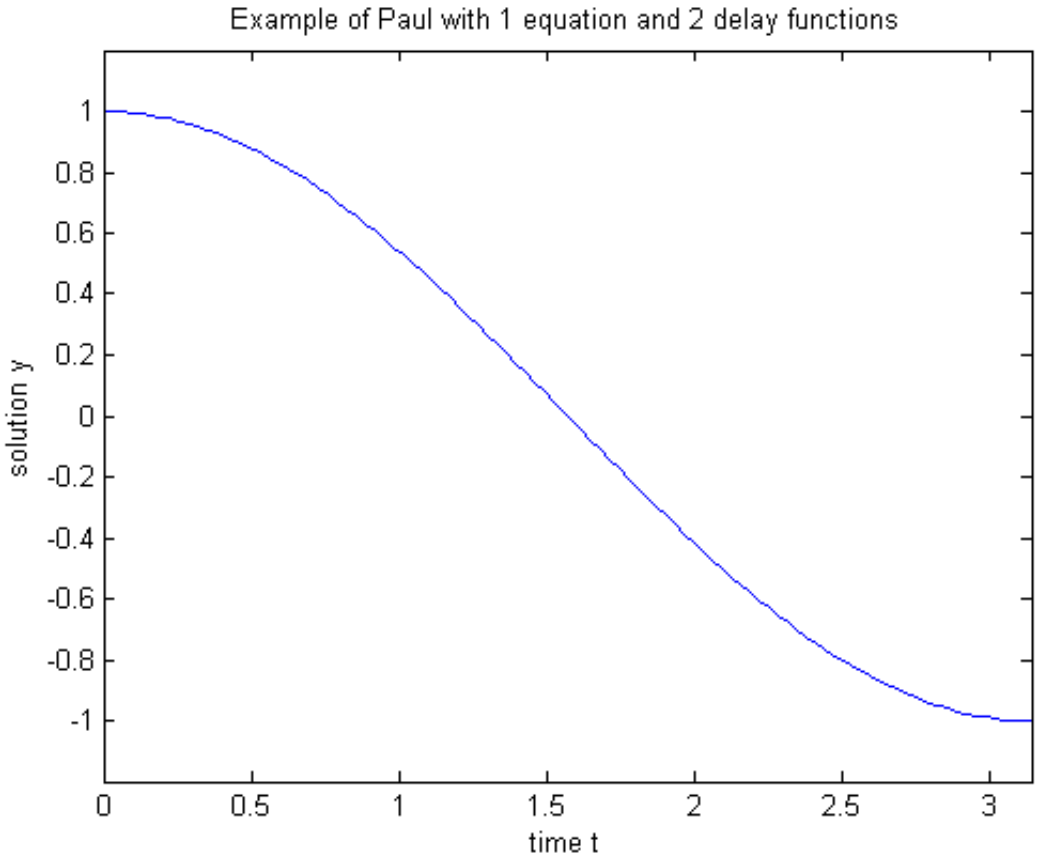
7. Evaluate the solution at 100 equally spaced points between 0 and π. Add this code to the main function.

```
tn = linspace(0,pi);
yn = deval(sol,tn);
```

8. Plot the results. Add this code to the main function.

```
figure
plot(tn,yn);
xlim([0 pi]);
ylim([-1.2 1.2])
xlabel('time t');
ylabel('solution y');
title('Example of Paul with 1 equation and 2 delay functions')
```

9. Run your program to calculate the solution and display the plot.



References

[1] Paul, C.A.H. "A Test Set of Functional Differential Equations." *Numerical Analysis Reports*. No. 243. Manchester, UK: Math Department, University of Manchester, 1994.

Related Examples

- Create Functions in Files
- Run Functions in the Editor

More About

- Create Function Handle

Initial Value DDE of Neutral Type

This example shows how to use `ddensd` to solve the initial value DDE presented by Jackiewicz [1] for $0 \le t \le 0.1$.

Click `ddex5.m` or type `edit ddex5.m` in a command window to view the code for this example in an editor.

The equation is

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

This is an initial value DDE because the delays are zero at t_0. The initial conditions are:

$$y(0) = 1$$

$$y'(0) = s,$$

where s is the solution of:

$$2 + \log(s) - \log(2) = 0.$$

This equation is satisfied by $s_1 = 2$ and $s_2 = 0.4063757399599599$.

1. Create a new program file in the editor. This file will contain a main function and one local function.
2. Define the DDE as a local function.

```
function yp = ddefun(t,y,ydel,ypdel)
yp=2*cos(2*t)*ydel^(2*cos(t))+log(ypdel)-log(2*cos(t))-sin(t);
end
```

3. Define the solution delay and derivative delay. Add this line to the main function.

```
delay = @(t,y) t/2;
```

   You can use one anonymous function to handle both delays since they are the same in the equation.
4. Define the initial conditions, `y0` and `s1`, and the interval of integration, `tspan`. Add this code to the main function.

```
y0 = 1;
s1 = 2;
tspan = [0 0.1];
```

5. Solve the DDE for $0 \le t \le 0.1$, with initial conditions $y(0) = 1$, and $y'(0) = 2$. Add this code to the main function.

```
sol1 = ddensd(@ddefun,delay,delay,{y0,s1},tspan);
```

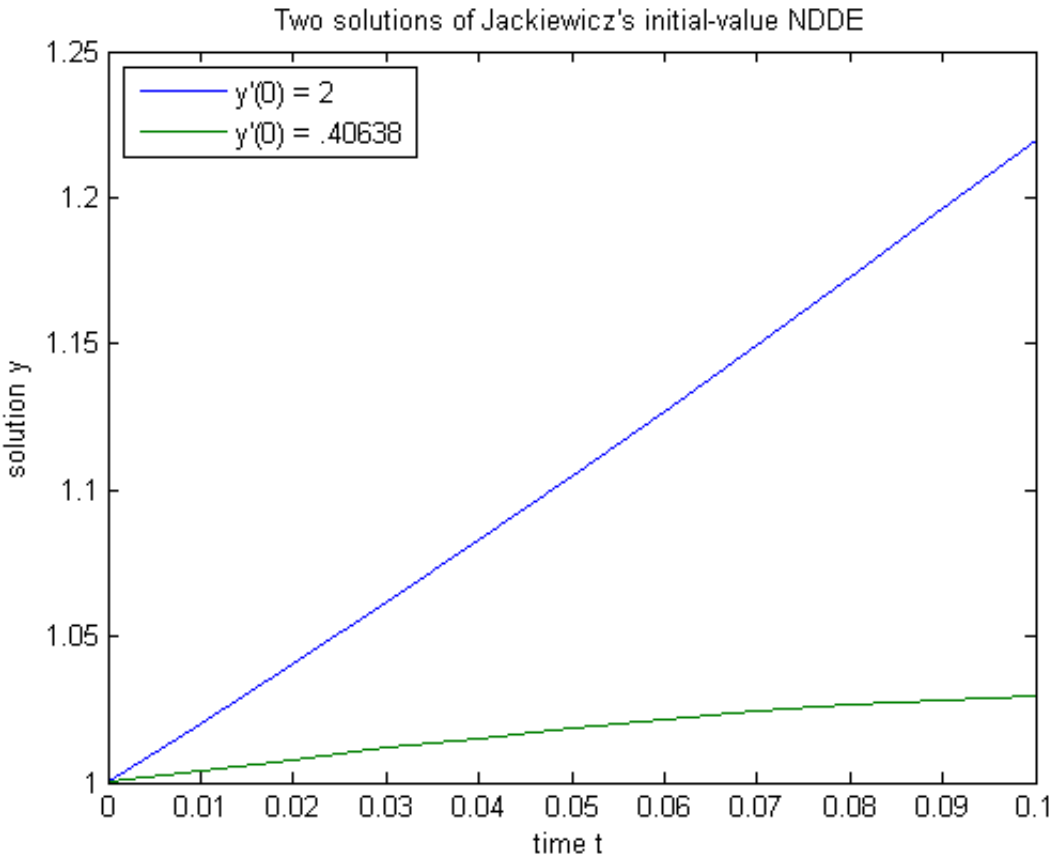
6. Solve the equation again, this time using $y'(0) = 0.4063757399599599$. Add this code to the main function.

```
s2 = 0.4063757399599599;
sol2 = ddensd(@ddefun,delay,delay,{y0,s2},tspan);
```

7. Plot the results. Add this code to the main function.

```
figure
plot(sol1.x,sol1.y,sol2.x,sol2.y);
legend('y''(0) = 2','y''(0) = .40638','Location','NorthWest');
xlabel('time t');
ylabel('solution y');
title('Two solutions of Jackiewicz''s initial-value NDDE');
```

8. Run your program to calculate and plot the solutions for each value of *s*.



References

[1] Jackiewicz, Z. "One step Methods of any Order for Neutral Functional Differential Equations." *SIAM J. Numer. Anal.* Vol. 21, Number 3. 1984. pp. 486–511.

Related Examples

- Create Functions in Files
- Run Functions in the Editor

More About

- Create Function Handle