To accomplish portions of the research, Alison Margolskee developed a delay differential equations solver that implements a 4th order Runge Kutta method with quartic hermite interpolant to obtain lagged values from stored history. This material is based on work supported by the National Science Foundation under grants DMS-0920927 and DMS-1225607.

Matlab code for the solver is as follows:

```
% ddeRK4
% Last updated 10/12/2012
% Alison Margolskee
%
% delay differential equation solver using explicit 4th order
% fixed-step Runge Kutta integrator and Quartic Hermite
% interpolating polynomial to determine lagged values from stored history

function sol = ddeRK4(ddefun,lags, history, t0, tf, h, options,varargin)

% Output:
% sol
%   .x = time steps
%   .y = solution at time steps .x
%   .yp = derivative at time steps .x
%       allows user to perform hermite interpolation to
%       obtain solution at time points not in .x, if desired
%
% Input:
% ddefun = delay differential equation
%       has the form dydt = ddefun(t, y, z, varargin)
%       where t = time, y = state variable (can be a column vector or a
scalor)
%       and z = [ y(tau1), y(tau2), ... ]
%               row vector of delayed state variables
%               (or matrix if y is a column vector)
% lags = (tau1, tau2, ....) = delays (must be positive)
% history = constant initial history or function of time defined on
%       the interval [t0 - max(lags), t0]
% t0 = initial time
% tf = final time
% h = time step
% options
%   .Nsteps = #steps to skip between writing to output
%       this option allows you to integrate with a
%       small step and save storage by skipping
%       steps in storage (default = 0)
%   .transient = time to run integrator before storing
%       allows integrator to approach a stable attractor before
%       storing the solution (default = t0)
% varargin = extra arguments to feed to ddefun

if min(lags)<=0
    error('Error: delays must be positive');
end
if max(lags) + 2*h > (tf - t0)
    error('Error: maximum delay must be less than tf - t0 by at least two
time steps')
end

%%%%%%%%%%% Set up options %%%%%%%%%%%%%

defaultopt.Nsteps = 0;
defaultopt.transient = t0;

if nargin < 7
    options = defaultopt;
elseif isempty(options)
    options = defaultopt;
else
    allfields = {'Nsteps';'transient'};
    for i = 1:length(allfields)
        try
            options.(cell2mat(allfields(i,:)));
        catch ME
            options.(cell2mat(allfields(i,:))) ...
                = defaultopt.(cell2mat(allfields(i,:)));
        end
    end

end

Nsteps = options.Nsteps;
transient = options.transient;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%% Determine initial y, yp and z
```

```
if isnumeric(history)
  y0 = history;
  sol.history = history;
else
  y0 = feval(history,t0,varargin{:});
  sol.history = history;
end
neq = length(y0);

Z0 = zeros(neq, length(lags));
for j = 1:length(lags)
    if isnumeric(history)
        Z0(:,j) = history;
    else
        Z0(:,j) = feval(history,t0 - lags(j),varargin{:});
    end
end

yp0 = feval(ddefun,t0,y0,Z0,varargin{:});

% total time interval for integrator
tspan=t0:h:tf;

%%%%%%%%%%% Initialize output %%%%%%%%%%
% tout, yout, and ypout
temp = find(transient <= tspan,1,'first');
tout = tspan(temp:(Nsteps+1):end);

yout = zeros(neq,length(tout));
ypout = zeros(neq,length(tout));
nout = 0;
nsteps = 0;

if t0 >= transient
    nout = nout + 1;
    nsteps = nsteps + 1;
    yout(:,1) = y0;
    ypout(:,1) = yp0;
end


%%%%%% Initialize History Storage %%%%%
% Thist, Yhist and YPhist
% determine size of storage for history
Nhist = find( max(lags) >= tspan - t0, 1, 'last') + 2;
```

```
Thist = tspan(1:Nhist) - max(lags) - 2*h;

Yhist = zeros(neq, Nhist);
Yhist(:,:) = y0*ones(1, Nhist);

YPhist = zeros(neq, Nhist);
YPhist(:,:) = yp0*ones(1, Nhist);


%%%%%%  Set up RK4 algorithm %%%%%%%%%
C=[0, 1/2, 1/2, 1, 1];
A=[
    0   1/2 0   0
    0   0   1/2 0
    0   0   0   1
    0   0   0   0];
B=[1/6  1/3 1/3 1/6]';

hA = h*A;
hB = h*B;
hC = h*C;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y = y0;

f=zeros(neq,4);
f(:,1) = yp0;

%%%% Perform RK4 step for each time point t in tspan %%%
for nspan = 2:length(tspan)

    t=tspan(nspan-1);

    %%%%%%%%%%%% Set up RK4 time points %%%%%%%%%%%%%%%
    t2 = t + hC(2);
    t3 = t + hC(3);
    t4 = t + hC(4);
    tnew = t + hC(5);

    %%%%%%%%%% Determine lagged values Z2, Z3, Z4, Znew

    Z2 = zeros(neq, length(lags));
    Z3 = zeros(neq, length(lags));
    Z4 = zeros(neq, length(lags));
    Znew = zeros(neq, length(lags));
    for j = 1:length(lags)
        % If lagged times are <= t0, use user-defined history
        if t4 - lags(j) <= t0
            if isnumeric(history)
                Z2(:,j) = history;
                Z3(:,j) = history;
                Z4(:,j) = history;
                Znew(:,j) = history;
            else
                Z2(:,j) = feval(history,t2 - lags(j),varargin{:});
                Z3(:,j) = feval(history,t3 - lags(j),varargin{:});
                Z4(:,j) = feval(history,t4 - lags(j),varargin{:});
                Znew(:,j) = feval(history,tnew - lags(j),varargin{:});
            end
        elseif t2 - lags(j) <= t0
            if isnumeric(history)
                Z2(:,j) = history;
                Z3(:,j) = history;
            else
                Z2(:,j) = feval(history,t2 - lags(j),varargin{:});
                Z3(:,j) = feval(history,t3 - lags(j),varargin{:});
            end
        else
            indices = find(t2 - lags(j) >= Thist(1:end-1));

            D = indices(end);

            %%%%%%%%%%%%%%%%%%%% Quartic Hermite Interpolation %%%%%%%%%%
            % interpolating polynomial of the form
            %
            %     p= @(t,a,x) a(:,1) + a(:,2)*(t-x(1)) + a(:,3)*(t-x
(1)).^2 ...
            %       + a(:,4)*(t-x(1)).^2.*(t-x(2))+...
            %         a(:,5)*(t-x(1)).^2.*(t-x(2)).^2 ;
            %
            %     dp= @(t,a,x) a(:,2) + 2*a(:,3)*(t-x(1)) ...
            %       + a(:,4)*(2*(t-x(1))*(t-x(2))+(t-x(1))^2) ...
            %       + a(:,5)*(2*(t-x(1))*(t-x(2))^2 + 2*(t-x(1))^2*(t-x
(2))) ;
            %
            alpha = zeros(length(y0),5);

            % a(:,1) = p(x(1),a,x)
            alpha(:,1)=Yhist(:,D-1);
            % a(:,2) = dp(x(1),a,x)
            alpha(:,2)=YPhist(:,D-1);

            % a(:,3) = (y(:,2) - p(x(2),a,x) )/(x(2)-x(1))^2;
            alpha(:,3) = ( (Yhist(:,D) - alpha(:,1) )/(Thist(D)-Thist
(D-1))...
                - alpha(:,2) )/(Thist(D)-Thist(D-1))- alpha(:,3) ;

            % a(:,4) = (dy(:,2) - dp(x(2),a,x) )/(x(2)-x(1))^2;
            alpha(:,4)=( (YPhist(:,D) - alpha(:,2) )/(Thist(D)-Thist
(D-1)) ...
                - 2*alpha(:,3) )/(Thist(D)-Thist(D+1)) - alpha(:,4) ;

            % a(:,5) = ( y(:,3) - p(x(3),a,x) )/((x(3)-x(1))^2*(x(3)-x(2))
^2);
            alpha(:,5)=( ( ( ( Yhist(:,D+1) - alpha(:,1))/(Thist(D+1)-Thist
(D-1)) ...
                - alpha(:,2) )/(Thist(D+1)-Thist(D-1)) - alpha(:,3) )/
(Thist(D+1)-Thist(D)) ...
                - alpha(:,4) )/(Thist(D+1)-Thist(D)) - alpha(:,5);

            %%%%%%% Evaulate lagged values with interpolating polynomial
            Z2(:,j) = alpha(:,1) + (t2 - lags(j) - Thist(D-1))*(alpha(:,
2)...
                + (t2 - lags(j) - Thist(D-1)) *( alpha(:,3) ...
                + (t2 - lags(j) - Thist(D))*( alpha(:,4) ...
                + alpha(:,5)*(t2 - lags(j) - Thist(D))  ) ) );

            Z3(:,j) = alpha(:,1) + (t3 - lags(j) - Thist(D-1))*(alpha(:,
2)...
                + (t3 - lags(j) - Thist(D-1)) *( alpha(:,3) ...
                + (t3 - lags(j) - Thist(D))*( alpha(:,4) ...
                + alpha(:,5)*(t3 - lags(j) - Thist(D))  ) ) );

            Z4(:,j) = alpha(:,1) + (t4 - lags(j) - Thist(D-1))*(alpha(:,
2)...
                + (t4 - lags(j) - Thist(D-1)) *( alpha(:,3) ...
                + (t4 - lags(j) - Thist(D))*( alpha(:,4) ...
                + alpha(:,5)*(t4 - lags(j) - Thist(D))  ) ) );

            Znew(:,j) = alpha(:,1) + (tnew - lags(j) - Thist(D-1))*(alpha
(:,2)...
                + (tnew - lags(j) - Thist(D-1)) *( alpha(:,3) ...
                + (tnew - lags(j) - Thist(D))*( alpha(:,4) ...
                + alpha(:,5)*(tnew - lags(j) - Thist(D))  ) ) );
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        end
    end

    %%%%%%%%%%%%%% Determine ynew %%%%%%%%%%%%%%%%%%%%

    f(:,2) = feval(ddefun,t2,y+f*hA(:,2),Z2,varargin{:});
    f(:,3) = feval(ddefun,t3,y+f*hA(:,3),Z3,varargin{:});
    f(:,4) = feval(ddefun,t4,y+f*hA(:,4),Z4,varargin{:});

    ynew = y + f*hB;

    %%%%%%%%%%%%%%%% Update history %%%%%%%%%%%%%%%%%%
    Yhist(:,1:Nhist-1) = Yhist(:,2:Nhist);
    Yhist(:,Nhist) = ynew;

    YPhist(:,1:Nhist-1) = YPhist(:,2:end);
    YPhist(:,Nhist) = feval(ddefun,tnew,ynew,Znew,varargin{:});

    Thist(1:Nhist-1) = Thist(2:Nhist);
    Thist(Nhist) = tnew;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if tnew >= transient
        if ~mod(nsteps, Nsteps+1)
            % if after transient period, and
            % if Nsteps have been reached, write to yout
            nout = nout + 1;
            yout(:,nout) = Yhist(:,Nhist);
            ypout(:,nout) = YPhist(:,Nhist);
        end
        nsteps = nsteps + 1;
    end

    y = ynew;
    f(:,1) = YPhist(:,Nhist);
end

sol.x = tout;
sol.y = yout;
sol.yp = ypout;
```

Published with MATLAB® 7.12