```matlab
function [t,x]=solve_nddae(F1,F2,tau,phi,tspan,options)
%solve_nddae
%
%   Solver for strangeness-free delay differential algebraic equations
%   (DDAE) with a constant delays of the form
%
%                           F1(t,x,x',xt) =0,
%                           F2(t,x,   xt) =0,
%
%   where xt(t)=[x(t-tau_1(t));...;x(t-tau_l(t))], t0<=t<=tf for some real
%   numbers t0,tf; x in D1 and x' in D2 with D1 and D2 being open subsets
%   of the R^n.
%
%   For all t<t0 we have x(t)= phi(t).
%   xt is a nxl matrix, where the ith column contains x(t-tau_i).
%
%   The related Radau IIA method was taken from:
%   -----------------------------------------------------------------------
%   P. Kunkel, V. Mehrmann: Differential-Algebraic Equations, p. 243-244
%   -----------------------------------------------------------------------
%
%   INPUT PARAMETERS
%   ----------------
%   F1      the differential part of the DDAE
%   F2      the algebraic part of the DDAE
%   t0      the initial time
%   x0      the initial value in the R^n, not necessarily consistent
%   tau     vector of constant delays, or function for timevarying delays
%   phi     the past function, i.e. x(t)=phi(t) for t in [t0-tau_l,t0]
%   h       the step size of the Runge-Kutta method, must be smaller than
%           tau
%   N       the number of steps for the Runge-Kutta method
%   tol     the tolerance used for testing, if something is equal to zero,
%           i.e. if a<=tol, then we consider a to be (approximately) zero
%
%   OUTPUT PARAMETERS
%   -----------------
%   x       the approximated solution of the DDAE given at the points in t
%   t       the vector [t0,t0+h,t0+2h,...,t0+Nh]

N = 99;
h = diff(tspan)/N;
tolA = 1e-7;
tolR = 1e-7;
x0 = phi(tspan(1));
if exist('options','var')
    if isfield(options,'AbsTol')     tolA=options.AbsTol; end
    if isfield(options,'NGrid')      N=options.NGrid-1; h=diff(tspan)/N; end
    if isfield(options,'RelTol')     tolR=options.RelTol; end
    if isfield(options,'StepSize')   h=options.StepSize; N=floor(diff(tspan)/h); end
    if isfield(options,'x0')         x0=options.x0; end
else
    options = {};
end

%dimension of system
n=length(x0);

%tau has to be a function, so we turn constant delays in a function
if not(isa(tau,'function_handle'))
    tau2=tau;
    tau=@(t) tau2;
end

%numbers of delays
l=length(tau(tspan(1)));

t0 = tspan(1);

% Butcher-tableau of the 3-stage-RadauIIA method
% A=[
%     (88-7*sqrt(6))/360      (296-169*sqrt(6))/1800  (-2+3*sqrt(6))/225
%     (296+169*sqrt(6))/1800  (88+7*sqrt(6))/360      (-2-3*sqrt(6))/225
```

```
%     (16-sqrt(6))/36        (16+sqrt(6))/36        1/9                 ];

% left-hand side of the Butcher tableau or the nodes
c=[
    (4-sqrt(6))/10
    (4+sqrt(6))/10
    1              ];

% The derivatives of the Lagrange polynomials evaluated in the collocation
% points, i.e. V(m,j)=L'_j(c_m), j,m=1,2,3, see p. 244.
% These values are given by the inverse of A in the Butcher-tableau, i.e.
% V=A^-1.
V=[
    3.224744871391589   1.167840084690405  -0.253197264742181
    -3.567840084690405   0.775255128608412   1.053197264742181
    5.531972647421811  -7.531972647421810   5.000000000000000
    ];

% The derivatives of the zero_th Lagrange polynomial evaluated at the
% collocation points, i.e. v0(j)=L'_0(c_m), j=1,2,3, see p. 244.
v0=-V*ones(3,1);

% the container for the approximate solution of the DDAE, the length of
% each column is 3*n, the last n entries of the i-th column form the
% approximation of x(t0+(i-1)*h).
x=nan(3*n,N+1);
x(1:n,1)=phi(t0+(c(1)-1)*h);
x(n+1:2*n,1)=phi(t0+(c(2)-1)*h);
x(2*n+1:3*n,1)=x0;

% the time
t=t0:diff(tspan/N):tspan(2);

% The big nonlinear system, which has to be solved, i.e. find X, such that
% the whole function is zero. All other input parameters will be given.
Fa=    @(t,xi,X,Z)[

    F1(t(1),X(1:n),(v0(1)*xi+V(1,1)*X(1:n)+V(1,2)*X(n+1:2*n)+V(1,3)*X(2*n+1:3*n))/h,Z(:,1));
    F2(t(1),X(1:n),Z(:,1));

    F1(t(2),X(n+1:2*n),(v0(2)*xi+V(2,1)*X(1:n)+V(2,2)*X(n+1:2*n)+V(2,3)*X(2*n+1:3*n))/h,Z(:,2));
    F2(t(2),X(n+1:2*n),Z(:,2));

    F1(t(3),X(2*n+1:3*n),(v0(3)*xi+V(3,1)*X(1:n)+V(3,2)*X(n+1:2*n)+V(3,3)*X(2*n+1:3*n))/h,Z(:,3));
    F2(t(3),X(2*n+1:3*n),Z(:,3))];

% The starting vector of size 3n x 1 for the Newton iteration.
X=[x0;x0;x0];

% Z contains the approximated delayed values x(t_i+c(j)*h-tau), j=1,2,3.
Z=zeros(l*n,3);

for i=1:N
    % calculating Z = x(t_i+c_j*h-tau_k)
    for j=1:3
        TAU = tau(t(i)+c(j)*h);
        for k=1:l
            if TAU(k) <= 0
                error('THE DELAY MUST BE BIGGER THAN ZERO!');
            end
            %check if x(t-tau) is given by Phi or has to be determined by
            % interpolating the current approximate solution
            if t(i)+c(j)*h-TAU(k)<=t0
                Z((k-1)*n+1:k*n,j)=phi(t(i)+c(j)*h-TAU(k));
            else
                % find the biggest time node smaller than t_i+c_j*h-TAU(k)
                t_tau_index = find(t(i)+c(j)*h-TAU(k)<t,1)-1;
                t_tau = t(t_tau_index);
                % if t_i+c_j*h-tau is not a node point, i.e. not in t, then we
                % have to interpolate
```

```
                % prepare some data for the interpolation
                % we use a polynomial of degree 3, so we need 4 data points
                x0_tau = x(2*n+1:3*n,t_tau_index);
                X_tau = reshape(x(:,t_tau_index+1),n,3);
                % interpolate with Neville-Aitken
                Z((k-1)*n+1:k*n,j) = poleval_neville_aitken(t_tau+[0;c]*h,[x0_tau,X_tau],t(i)+c(j)*h-TAU(k));
            end
        end
    end
    % insert all given data into the function Fa defined above, such that
    % we get a function only depending on X
    F=@(X) Fa(t(i)+c'*h,x(2*n+1:3*n,i),X,Z);
    % now solve F(X)=0 with Newton's method
    for newt=1:7
        FX = F(X);
        if max(abs(FX))<=tolR
            break
        end

        DF = jacobian(F,X);

        % If DF does not have full rank, then stop the calculation.ss
        if rank(DF)~=length(DF)
            disp('SINGULAR JACOBIAN IN NEWTON METHOD IN RADAR5!')
            % "cutting out" the solution we have so far
            x = x(2*n+1:3*n,:);
            return
        end
        X = X-DF\FX;
    end
    x(:,i+1) = X;
end
% "cutting out" the approximate solution
x = x(2*n+1:3*n,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   END OF RADAR5.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function px = poleval_neville_aitken(X,F,x)

n=length(X);
%at first px is a container for the values in the Newton scheme
px=F;
% beginning the Newton scheme, see Numerische Mathematik 1
for i=1:n-1
    for j=1:n-i
        px(:,j)=((x-X(j))*px(:,j+1)-(x-X(j+i))*px(:,j))/(X(j+i)-X(j));
    end
end
px=px(:,1);

function J=jacobian(F,X)
n=length(X);
FX=F(X);
J=zeros(n);
for i=1:n
    Xsafe=X(i);
    % preventing delt from becoming too small (cancellation)
    delt=sqrt(eps*max(1e-5,abs(Xsafe)));
    X(i)=X(i)+delt;
    % unfortunately we can not prevent cancellation in F(X)-FX
    J(:,i)=(F(X)-FX)/delt;
    X(i)=Xsafe;
end
```