```
function [tout,yout,varargout] = ode23(odefile,tspan,y0,options,varargin)
%ODE23  Solve non-stiff differential equations, low order method.
%   [T,Y] = ODE23('F',TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the
%   system of differential equations y' = F(t,y) from time T0 to TFINAL with
%   initial conditions Y0.  'F' is a string containing the name of an ODE
%   file.  Function F(T,Y) must return a column vector.  Each row in
%   solution array Y corresponds to a time returned in column vector T.  To
%   obtain solutions at specific times T0, T1, ..., TFINAL (all increasing
%   or all decreasing), use TSPAN = [T0 T1 ... TFINAL].
%
%   [T,Y] = ODE23('F',TSPAN,Y0,OPTIONS) solves as above with default
%   integration parameters replaced by values in OPTIONS, an argument
%   created with the ODESET function.  See ODESET for details.  Commonly
%   used options are scalar relative error tolerance 'RelTol' (1e-3 by
%   default) and vector of absolute error tolerances 'AbsTol' (all
%   components 1e-6 by default).
%
%   [T,Y] = ODE23('F',TSPAN,Y0,OPTIONS,P1,P2,...) passes the additional
%   parameters P1,P2,... to the ODE file as F(T,Y,FLAG,P1,P2,...) (see
%   ODEFILE).  Use OPTIONS = [] as a place holder if no options are set.
%
%   It is possible to specify TSPAN, Y0 and OPTIONS in the ODE file (see
%   ODEFILE).  If TSPAN or Y0 is empty, then ODE23 calls the ODE file
%   [TSPAN,Y0,OPTIONS] = F([],[],'init') to obtain any values not supplied
%   in the ODE23 argument list.  Empty arguments at the end of the call list
%   may be omitted, e.g. ODE23('F').
%
%   As an example, the commands
%
%       options = odeset('RelTol',1e-4,'AbsTol',[1e-4 1e-4 1e-5]);
%       ode23('rigidode',[0 12],[0 1 1],options);
%
%   solve the system y' = rigidode(t,y) with relative error tolerance 1e-4
%   and absolute tolerances of 1e-4 for the first two components and 1e-5
%   for the third.  When called with no output arguments, as in this
%   example, ODE23 calls the default output function ODEPLOT to plot the
%   solution as it is computed.
%
%   ODE23 can solve problems M(t,y)*y' = F(t,y) with a mass matrix M that is
%   nonsingular.  Use ODESET to set Mass to 'M', 'M(t)', or 'M(t,y)' if the
%   ODE file is coded so that F(T,Y,'mass') returns a constant,
%   time-dependent, or time- and state-dependent mass matrix, respectively.
%   The default value of Mass is 'none'.  See FEM1ODE, FEM2ODE, or BATONODE.
%   ODE15S and ODE23T can solve problems with singular mass matrices.
%
%   [T,Y,TE,YE,IE] = ODE23('F',TSPAN,Y0,OPTIONS) with the Events property in
%   OPTIONS set to 'on', solves as above while also locating zero crossings
%   of an event function defined in the ODE file.  The ODE file must be
%   coded so that F(T,Y,'events') returns appropriate information.  See
%   ODEFILE for details.  Output TE is a column vector of times at which
%   events occur, rows of YE are the corresponding solutions, and indices in
%   vector IE specify which event occurred.
%
%   See also ODEFILE and
%       other ODE solvers:  ODE45, ODE113, ODE15S, ODE23S, ODE23T, ODE23TB
%       options handling:   ODESET, ODEGET
%       output functions:   ODEPLOT, ODEPHAS2, ODEPHAS3, ODEPRINT
%       odefile examples:   ORBITODE, ORBT2ODE, RIGIDODE, VDPODE
```

```
%   ODE23 is an implementation of the explicit Runge-Kutta (2,3) pair of
%   Bogacki and Shampine called BS23.  It uses a "free" interpolant of order
%   3.  Local extrapolation is done.

%   Details are to be found in The MATLAB ODE Suite, L. F. Shampine and
%   M. W. Reichelt, SIAM Journal on Scientific Computing, 18-1, 1997.

%   Mark W. Reichelt and Lawrence F. Shampine, 6-14-94
%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.56 $  $Date: 1998/05/14 11:28:38 $

true = 1;
false = ~true;

nsteps = 0;                             % stats
nfailed = 0;                            % stats
nfevals = 0;                            % stats
npds = 0;                               % stats
ndecomps = 0;                           % stats
nsolves = 0;                            % stats

if nargin == 0
  error('Not enough input arguments.  See ODE23.');
elseif ~isstr(odefile) & ~isa(odefile, 'inline')
  error('First argument must be a single-quoted string.  See ODE23.');
end

if nargin == 1
  tspan = []; y0 = []; options = [];
elseif nargin == 2
  y0 = []; options = [];
elseif nargin == 3
  options = [];
elseif ~isempty(options) & ~isa(options,'struct')
  if (length(tspan) == 1) & (length(y0) == 1) & (min(size(options)) == 1)
    tspan = [tspan; y0];
    y0 = options;
    options = [];
    varargin = {};
    msg = sprintf('Use ode23(''%s'',tspan,y0,...) instead.',odefile);
    warning(['Obsolete syntax.  ' msg]);
  else
    error('Correct syntax is ode23(''odefile'',tspan,y0,options).');
  end
end

% Get default tspan and y0 from odefile if none are specified.
if isempty(tspan) | isempty(y0)
  if (nargout(odefile) < 3) & (nargout(odefile) ~= -1)
    msg = sprintf('Use ode23(''%s'',tspan,y0,...) instead.',odefile);
    error(['No default parameters in ' upper(odefile) '.  ' msg]);
  end
  [def_tspan,def_y0,def_options] = feval(odefile,[],[],'init',varargin{:});
  if isempty(tspan)
    tspan = def_tspan;
  end
  if isempty(y0)
    y0 = def_y0;
  end
```

```
  if isempty(options)
    options = def_options;
  else
    options = odeset(def_options,options);
  end
end

% Test that tspan is internally consistent.
tspan = tspan(:);
ntspan = length(tspan);
if ntspan == 1
  t0 = 0;
  next = 1;
else
  t0 = tspan(1);
  next = 2;
end
tfinal = tspan(ntspan);
if t0 == tfinal
  error('The last entry in tspan must be different from the first entry.');
end
tdir = sign(tfinal - t0);
if any(tdir * (tspan(2:ntspan) - tspan(1:ntspan-1)) <= 0)
  error('The entries in tspan must strictly increase or decrease.');
end

t = t0;
y = y0(:);
neq = length(y);

% Get options, and set defaults.
rtol = odeget(options,'RelTol',1e-3);
if (length(rtol) ~= 1) | (rtol <= 0)
  error('RelTol must be a positive scalar.');
end
if rtol < 100 * eps
  rtol = 100 * eps;
  warning(['RelTol has been increased to ' num2str(rtol) '.']);
end

atol = odeget(options,'AbsTol',1e-6);
if any(atol <= 0)
  error('AbsTol must be positive.');
end

normcontrol = strcmp(odeget(options,'NormControl','off'),'on');
if normcontrol
  if length(atol) ~= 1
    error('Solving with NormControl ''on'' requires a scalar AbsTol.');
  end
  normy = norm(y);
else
  if (length(atol) ~= 1) & (length(atol) ~= neq)
    error(sprintf(['Solving %s requires a scalar AbsTol, ' ...
                   'or a vector AbsTol of length %d'],upper(odefile),neq));
  end
  atol = atol(:);
end
threshold = atol / rtol;
```

```
% By default, hmax is 1/10 of the interval.
hmax = min(abs(tfinal-t), abs(odeget(options,'MaxStep',0.1*(tfinal-t))));
if hmax <= 0
  error('Option ''MaxStep'' must be greater than zero.');
end
htry = abs(odeget(options,'InitialStep'));
if htry <= 0
  error('Option ''InitialStep'' must be greater than zero.');
end

haveeventfun = strcmp(odeget(options,'Events','off'),'on');
if haveeventfun
  valt = feval(odefile,t,y,'events',varargin{:});
  teout = [];
  yeout = [];
  ieout = [];
end

if nargout > 0
  outfun = odeget(options,'OutputFcn');
else
  outfun = odeget(options,'OutputFcn','odeplot');
end
if isempty(outfun)
  haveoutfun = false;
else
  haveoutfun = true;
  outputs = odeget(options,'OutputSel',1:neq);
end
refine = odeget(options,'Refine',1);
printstats = strcmp(odeget(options,'Stats','off'),'on');

mass = lower(odeget(options,'Mass','none'));

switch(mass)
  case 'none', Mtype = 0;
  case 'm', Mtype = 1;
  case 'm(t)', Mtype = 2;
  case 'm(t,y)', Mtype = 3;
  otherwise, error('Unrecognized Mass property value.  See ODESET.');
end

Msingular = odeget(options,'MassSingular');
if strcmp(Msingular,'maybe')
  warning(['This solver assumes MassSingular is ''no''.  See ODE15S or ' ...
        'ODE23T.']);
elseif strcmp(Msingular,'yes')
  error(['MassSingular cannot be ''yes'' for this solver.  See ODE15S or ' ...
        'ODE23T.']);
end

if Mtype == 1
  M = feval(odefile,t,y,'mass',varargin{:});
  [L,U] = lu(M);
else
  L = [];
  U = [];
end
```

```
% Set the output flag.
if ntspan > 2
  outflag = 1;                          % output only at tspan points
elseif refine <= 1
  outflag = 2;                          % computed points, no refinement
else
  outflag = 3;                          % computed points, with refinement
  S = (1:refine-1)' / refine;
end

% Allocate memory if we're generating output.
if nargout > 0
  if ntspan > 2                         % output only at tspan points
    tout = zeros(ntspan,1);
    yout = zeros(ntspan,neq);
  else                                  % alloc in chunks
    chunk = max(ceil(128 / neq),refine);
    tout = zeros(chunk,1);
    yout = zeros(chunk,neq);
  end
  nout = 1;
  tout(nout) = t;
  yout(nout,:) = y.';
end

% Initialize method parameters.
pow = 1/3;
A = [1/2; 3/4; 1];
B = [
    1/2         0               2/9
    0           3/4             1/3
    0           0               4/9
    0           0               0
    ];
E = [-5/72; 1/12; 1/9; -1/8];
f = zeros(neq,4);

% The input arguments of odefile determine the args to use to evaluate f.
if (exist(odefile) == 3) | (nargin(odefile) == 2) % don't nargin MEX files
  args = {};                            % odefile accepts only (t,y)
else
  args = [{''} varargin];               % use (t,y,'',p1,p2,...)
end

f0 = evalODEfile(odefile,t,y,Mtype,L,U,varargin,args);
nfevals = nfevals + 1;                  % stats
[m,n] = size(f0);
if n > 1
  error([upper(odefile) ' must return a column vector.'])
elseif m ~= neq
  msg = sprintf('an initial condition vector of length %d.',m);
  error(['Solving ' upper(odefile) ' requires ' msg]);
end

hmin = 16*eps*abs(t);
if isempty(htry)
  % Compute an initial step size h using y'(t).
  absh = min(hmax, abs(tspan(next) - t));
```

```
  if normcontrol
    rh = (norm(f0) / max(normy,threshold)) / (0.8 * rtol^pow);
  else
    rh = norm(f0 ./ max(abs(y),threshold),inf) / (0.8 * rtol^pow);
  end
  if absh * rh > 1
    absh = 1 / rh;
  end
  absh = max(absh, hmin);
else
  absh = min(hmax, max(hmin, htry));
end
f(:,1) = f0;

% Initialize the output function.
if haveoutfun
  feval(outfun,[t tfinal],y(outputs),'init');
end

% THE MAIN LOOP

done = false;
while ~done

  % By default, hmin is a small number such that t+hmin is only slightly
  % different than t.  It might be 0 if t is 0.
  hmin = 16*eps*abs(t);
  absh = min(hmax, max(hmin, absh));    % couldn't limit absh until new hmin
  h = tdir * absh;

  % Stretch the step if within 10% of tfinal-t.
  if 1.1*absh >= abs(tfinal - t)
    h = tfinal - t;
    absh = abs(h);
    done = true;
  end

  % LOOP FOR ADVANCING ONE STEP.
  nofailed = true;                      % no failed attempts
  while true
    hB = h * B;
    f(:,2) = evalODEfile(odefile,t+h*A(1),y+f*hB(:,1),Mtype,L,U,varargin,args);
    f(:,3) = evalODEfile(odefile,t+h*A(2),y+f*hB(:,2),Mtype,L,U,varargin,args);
    tnew = t + h*A(3);
    ynew = y + f*hB(:,3);
    f(:,4) = evalODEfile(odefile,tnew,ynew,Mtype,L,U,varargin,args);
    nfevals = nfevals + 3;              % stats

    % Estimate the error.
    if normcontrol
      normynew = norm(ynew);
      err = absh * (norm(f * E) / max(max(normy,normynew),threshold));
    else
      err = absh * norm((f * E) ./ max(max(abs(y),abs(ynew)),threshold),inf);
    end

    % Accept the solution only if the weighted error is no more than the
    % tolerance rtol.  Estimate an h that will yield an error of rtol on
    % the next step or the next try at taking this step, as the case may be,
```

```
    % and use 0.8 of this value to avoid failures.
    if err > rtol                       % Failed step
      nfailed = nfailed + 1;            % stats
      if absh <= hmin
        msg = sprintf(['Failure at t=%e.  Unable to meet integration ' ...
                       'tolerances without reducing the step size below ' ...
                       'the smallest value allowed (%e) at time t.\n'],t,hmin);
        warning(msg);
        if haveoutfun
          feval(outfun,[],[],'done');
        end
        if printstats                   % print cost statistics
          fprintf('%g successful steps\n', nsteps);
          fprintf('%g failed attempts\n', nfailed);
          fprintf('%g function evaluations\n', nfevals);
          fprintf('%g partial derivatives\n', npds);
          fprintf('%g LU decompositions\n', ndecomps);
          fprintf('%g solutions of linear systems\n', nsolves);
        end
        if nargout > 0
          tout = tout(1:nout);
          yout = yout(1:nout,:);
          if haveeventfun
            varargout{1} = teout;
            varargout{2} = yeout;
            varargout{3} = ieout;
            varargout{4} = [nsteps; nfailed; nfevals; npds; ndecomps; nsolves];
          else
            varargout{1} = [nsteps; nfailed; nfevals; npds; ndecomps; nsolves];
          end
        end
        return;
      end

      if nofailed
        nofailed = false;
        absh = max(hmin, absh * max(0.5, 0.8*(rtol/err)^pow));
      else
        absh = max(hmin, 0.5 * absh);
      end
      h = tdir * absh;
      done = false;

    else                                % Successful step
      break;

    end
  end
  nsteps = nsteps + 1;                  % stats

  if haveeventfun
    [te,ye,ie,valt,stop] = ...
        odezero('ntrp23',odefile,valt,t,y,tnew,ynew,t0,varargin,h,f);
    nte = length(te);
    if nte > 0
      if nargout > 2
        teout = [teout; te];
        yeout = [yeout; ye.'];
        ieout = [ieout; ie];
```

```
      end
      if stop                             % stop on a terminal event
        tnew = te(nte);
        ynew = ye(:,nte);
        done = true;
      end
    end
  end

  if nargout > 0
    oldnout = nout;
    if outflag == 2                       % computed points, no refinement
      nout = nout + 1;
      if nout > length(tout)
        tout = [tout; zeros(chunk,1)];
        yout = [yout; zeros(chunk,neq)];
      end
      tout(nout) = tnew;
      yout(nout,:) = ynew.';
    elseif outflag == 3                   % computed points, with refinement
      nout = nout + refine;
      if nout > length(tout)
        tout = [tout; zeros(chunk,1)];  % requires chunk >= refine
        yout = [yout; zeros(chunk,neq)];
      end
      i = oldnout+1:nout-1;
      tout(i) = t + (tnew-t)*S;
      yout(i,:) = ntrp23(tout(i),t,y,[],[],h,f).';
      tout(nout) = tnew;
      yout(nout,:) = ynew.';
    elseif outflag == 1                   % output only at tspan points
      while next <= ntspan
        if tdir * (tnew - tspan(next)) < 0
          if haveeventfun & done
            nout = nout + 1;
            tout(nout) = tnew;
            yout(nout,:) = ynew.';
          end
          break;
        elseif tnew == tspan(next)
          nout = nout + 1;
          tout(nout) = tnew;
          yout(nout,:) = ynew.';
          next = next + 1;
          break;
        end
        nout = nout + 1;                  % tout and yout are already allocated
        tout(nout) = tspan(next);
        yout(nout,:) = ntrp23(tspan(next),t,y,[],[],h,f).';
        next = next + 1;
      end
    end

    if haveoutfun
      i = oldnout+1:nout;
      if ~isempty(i) & (feval(outfun,tout(i),yout(i,outputs).') == 1)
        tout = tout(1:nout);
        yout = yout(1:nout,:);
        if haveeventfun
```

```
        varargout{1} = teout;
        varargout{2} = yeout;
        varargout{3} = ieout;
        varargout{4} = [nsteps; nfailed; nfevals; npds; ndecomps; nsolves];
      else
        varargout{1} = [nsteps; nfailed; nfevals; npds; ndecomps; nsolves];
      end
      return;
    end
  end

elseif haveoutfun
  if outflag == 2
    if feval(outfun,tnew,ynew(outputs)) == 1
      return;
    end
  elseif outflag == 3                 % computed points, with refinement
    tinterp = t + (tnew-t)*S;
    yinterp = ntrp23(tinterp,t,y,[],[],h,f);
    if feval(outfun,[tinterp; tnew],[yinterp(outputs,:), ynew(outputs)]) == 1
      return;
    end
  elseif outflag == 1                 % output only at tspan points
    ninterp = 0;
    while next <= ntspan
      if tdir * (tnew - tspan(next)) < 0
        if haveeventfun & done
          ninterp = ninterp + 1;
          tinterp(ninterp,1) = tnew;
          yinterp(:,ninterp) = ynew;
        end
        break;
      elseif tnew == tspan(next)
        ninterp = ninterp + 1;
        tinterp(ninterp,1) = tnew;
        yinterp(:,ninterp) = ynew;
        next = next + 1;
        break;
      end
      ninterp = ninterp + 1;
      tinterp(ninterp,1) = tspan(next);
      yinterp(:,ninterp) = ntrp23(tspan(next),t,y,[],[],h,f);
      next = next + 1;
    end
    if ninterp > 0
      if feval(outfun,tinterp(1:ninterp),yinterp(outputs,1:ninterp)) == 1
        return;
      end
    end
  end
end

% If there were no failures compute a new h.
if nofailed
  % Note that absh may shrink by 0.8, and that err may be 0.
  temp = 1.25*(err/rtol)^pow;
  if temp > 0.2
    absh = absh / temp;
  else
```

```
      absh = 5.0*absh;
    end
  end

  % Advance the integration one step.
  t = tnew;
  y = ynew;
  if normcontrol
    normy = normynew;
  end
  f(:,1) = f(:,4);                         % Already evaluated odefile(tnew,ynew)

end

if haveoutfun
  feval(outfun,[],[],'done');
end

if printstats                              % print cost statistics
  fprintf('%g successful steps\n', nsteps);
  fprintf('%g failed attempts\n', nfailed);
  fprintf('%g function evaluations\n', nfevals);
  fprintf('%g partial derivatives\n', npds);
  fprintf('%g LU decompositions\n', ndecomps);
  fprintf('%g solutions of linear systems\n', nsolves);
end

if nargout > 0
  tout = tout(1:nout);
  yout = yout(1:nout,:);
  if haveeventfun
    varargout{1} = teout;
    varargout{2} = yeout;
    varargout{3} = ieout;
    varargout{4} = [nsteps; nfailed; nfevals; npds; ndecomps; nsolves];
  else
    varargout{1} = [nsteps; nfailed; nfevals; npds; ndecomps; nsolves];
  end
end

% ----------------------------------------------------------------------------

function f = evalODEfile(odefile,t,y,Mtype,L,U,args1,args2)
%EVALODEFILE Evaluate the ODE file and deal with mass matrix if present.

if Mtype == 0                              % no mass matrix
  f = feval(odefile,t,y,args2{:});
elseif Mtype == 1                          % constant mass matrix
  f = U \ (L \ feval(odefile,t,y,args2{:}));
else                                       % M(t) or M(t,y)
  M = feval(odefile,t,y,'mass',args1{:});
  f = M \ feval(odefile,t,y,args2{:});
end
```