ode45 (Calls: 140, Time: 3.116 s)

Generated 04-Feb-2022 20:56:02 using performance time.
Function in file E:\Program Files\toolbox\matlab\funfun\ode45.m
Copy to new window for comparing multiple runs

Parents (calling functions)

Function Name	Function Type	Calls
Main>OptimizationFunction	Subfunction	139
Main	Script	1

Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
406	`odezero(@ntrp45,eventFcn,eventArgs,valt,t,y,tnew…`	15189	1.238	39.7%	
451	`yntrp45 = ntrp45split(tref,t,y,h,f1,f3,f4,f5,f6,…`	15189	0.240	7.7%	
299	`f2 = odeFcn_main(t2, y2);`	15200	0.215	6.9%	
311	`f5 = odeFcn_main(t5, y5);`	15200	0.162	5.2%	
303	`f3 = odeFcn_main(t3, y3);`	15200	0.160	5.1%	
All other lines			1.102	35.4%	
Totals			3.116	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time (s)	% Time	Time Plot
funfun\private\odezero	Function	15189	1.078	34.6%	
Main>@(t,input)ODEFUN(t,input)	Anonymous function	91200	0.814	26.1%	
funfun\private\ntrp45split	Function	15189	0.182	5.8%	
funfun\private\odearguments	Function	140	0.060	1.9%	
funfun\private\odefinalize	Function	140	0.027	0.9%	
funfun\private\odeevents	Function	140	0.025	0.8%	
odeget	Function	560	0.014	0.4%	
funfun\private\ntrp45	Function	140	0.011	0.4%	
funfun\private\odemass	Function	140	0.009	0.3%	
funfun\private\odefcncleanup	Function	140	0.002	0.1%	
Self time (built-ins, overhead, etc.)			0.895	28.7%	
Totals			3.116	100%	

Code Analyzer results

Line Number	Message
409	`The variable 'teout' appears to change size on every loop iteration. Consider preallocating…`
410	`The variable 'yeout' appears to change size on every loop iteration. Consider preallocating…`
411	`The variable 'ieout' appears to change size on every loop iteration. Consider preallocating…`
432	`The variable 'tout' appears to change size on every loop iteration. Consider preallocating …`
433	`The variable 'yout' appears to change size on every loop iteration. Consider preallocating …`
436	`The variable 'tout' appears to change size on every loop iteration. Consider preallocating …`
437	`The variable 'yout' appears to change size on every loop iteration. Consider preallocating …`
438	`The variable 'f3d' appears to change size on every loop iteration. Consider preallocating f…`
461	`The variable 'tout_new' appears to change size on every loop iteration. Consider preallocat…`
462	`The variable 'yout_new' appears to change size on every loop iteration. Consider preallocat…`
467	`The variable 'tout_new' appears to change size on every loop iteration. Consider preallocat…`
469	`The variable 'yout_new' appears to change size on every loop iteration. Consider preallocat…`
472	`The variable 'yout_new' appears to change size on every loop iteration. Consider preallocat…`

483	The variable 'tout' appears to change size on every loop iteration. Consider preallocating …
484	The variable 'yout' appears to change size on every loop iteration. Consider preallocating …
487	The variable 'tout' appears to change size on every loop iteration. Consider preallocating …
488	The variable 'yout' appears to change size on every loop iteration. Consider preallocating …

Coverage results

Show coverage for parent folder

Total lines in function	538
Non-code lines (comments, blank lines)	168
Code lines (lines that can run)	370
Code lines that did run	242
Code lines that did not run	128
Coverage (did run/can run)	65.41 %

Function listing

```
Time    Calls    Line
                   1  function varargout = ode45(ode,tspan,y0,options,varargin)
                   2  %ODE45  Solve non-stiff differential equations, medium order method.
                   3  %   [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates
                   4  %   the system of differential equations y' = f(t,y) from time T0 to TFINAL
                   5  %   with initial conditions Y0. ODEFUN is a function handle. For a scalar T
                   6  %   and a vector Y, ODEFUN(T,Y) must return a column vector corresponding
                   7  %   to f(t,y). Each row in the solution array YOUT corresponds to a time
                   8  %   returned in the column vector TOUT.  To obtain solutions at specific
                   9  %   times T0,T1,...,TFINAL (all increasing or all decreasing), use TSPAN =
                  10  %   [T0 T1 ... TFINAL].
                  11  %
                  12  %   [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) solves as above with default
                  13  %   integration properties replaced by values in OPTIONS, an argument created
                  14  %   with the ODESET function. See ODESET for details. Commonly used options
                  15  %   are scalar relative error tolerance 'RelTol' (1e-3 by default) and vector
                  16  %   of absolute error tolerances 'AbsTol' (all components 1e-6 by default).
                  17  %   If certain components of the solution must be non-negative, use
                  18  %   ODESET to set the 'NonNegative' property to the indices of these
                  19  %   components.
                  20  %
                  21  %   ODE45 can solve problems M(t,y)*y' = f(t,y) with mass matrix M that is
                  22  %   nonsingular. Use ODESET to set the 'Mass' property to a function handle
                  23  %   MASS if MASS(T,Y) returns the value of the mass matrix. If the mass matrix
                  24  %   is constant, the matrix can be used as the value of the 'Mass' option. If
                  25  %   the mass matrix does not depend on the state variable Y and the function
                  26  %   MASS is to be called with one input argument T, set 'MStateDependence' to
                  27  %   'none'. ODE15S and ODE23T can solve problems with singular mass matrices.
                  28  %
                  29  %   [TOUT,YOUT,TE,YE,IE] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) with the 'Events'
                  30  %   property in OPTIONS set to a function handle EVENTS, solves as above
                  31  %   while also finding where functions of (T,Y), called event functions,
                  32  %   are zero. For each function you specify whether the integration is
                  33  %   to terminate at a zero and whether the direction of the zero crossing
                  34  %   matters. These are the three column vectors returned by EVENTS:
                  35  %   [VALUE,ISTERMINAL,DIRECTION] = EVENTS(T,Y). For the I-th event function:
                  36  %   VALUE(I) is the value of the function, ISTERMINAL(I)=1 if the integration
                  37  %   is to terminate at a zero of this event function and 0 otherwise.
                  38  %   DIRECTION(I)=0 if all zeros are to be computed (the default), +1 if only
                  39  %   zeros where the event function is increasing, and -1 if only zeros where
                  40  %   the event function is decreasing. Output TE is a column vector of times
                  41  %   at which events occur. Rows of YE are the corresponding solutions, and
                  42  %   indices in vector IE specify which event occurred.
                  43  %
                  44  %   SOL = ODE45(ODEFUN,[T0 TFINAL],Y0...) returns a structure that can be
                  45  %   used with DEVAL to evaluate the solution or its first derivative at
                  46  %   any point between T0 and TFINAL. The steps chosen by ODE45 are returned
```

```
                47  %   in a row vector SOL.x.  For each I, the column SOL.y(:,I) contains
                48  %   the solution at SOL.x(I). If events were detected, SOL.xe is a row vector
                49  %   of points at which events occurred. Columns of SOL.ye are the corresponding
                50  %   solutions, and indices in vector SOL.ie specify which event occurred.
                51  %
                52  %   Example
                53  %         [t,y]=ode45(@vdp1,[0 20],[2 0]);
                54  %         plot(t,y(:,1));
                55  %     solves the system y' = vdp1(t,y), using the default relative error
                56  %     tolerance 1e-3 and the default absolute tolerance of 1e-6 for each
                57  %     component, and plots the first component of the solution.
                58  %
                59  %   Class support for inputs TSPAN, Y0, and the result of ODEFUN(T,Y):
                60  %     float: double, single
                61  %
                62  %   See also ODE23, ODE113, ODE15S, ODE23S, ODE23T, ODE23TB, ODE15I,
                63  %            ODESET, ODEPLOT, ODEPHAS2, ODEPHAS3, ODEPRINT, DEVAL,
                64  %            ODEEXAMPLES, RIGIDODE, BALLODE, ORBITODE, FUNCTION_HANDLE.
                65
                66  %   ODE45 is an implementation of the explicit Runge-Kutta (4,5) pair of
                67  %   Dormand and Prince called variously RK5(4)7FM, DOPRI5, DP(4,5) and DP54.
                68  %   It uses a "free" interpolant of order 4 communicated privately by
                69  %   Dormand and Prince.  Local extrapolation is done.
                70
                71  %   Details are to be found in The MATLAB ODE Suite, L. F. Shampine and
                72  %   M. W. Reichelt, SIAM Journal on Scientific Computing, 18-1, 1997.
                73
                74  %   Mark W. Reichelt and Lawrence F. Shampine, 6-14-94
                75  %   Copyright 1984-2017 The MathWorks, Inc.
                76
< 0.001     140 77  solver_name = 'ode45';
                78
                79  % Check inputs
< 0.001     140 80  if nargin < 4
                81    options = [];
                82    if nargin < 3
                83      y0 = [];
                84      if nargin < 2
                85        tspan = [];
                86        if nargin < 1
                87          error(message('MATLAB:ode45:NotEnoughInputs'));
                88        end
                89      end
                90    end
< 0.001     140 91  end
                92
                93  % Stats
< 0.001     140 94  nsteps  = 0;
< 0.001     140 95  nfailed = 0;
< 0.001     140 96  nfevals = 0;
                97
                98  % Output
< 0.001     140 99  FcnHandlesUsed  = isa(ode,'function_handle');
< 0.001     140 100 output_sol = (FcnHandlesUsed && (nargout==1));      % sol = odeXX(...)
  0.001     140 101 output_ty  = (~output_sol && (nargout > 0));  % [t,y,...] = odeXX(...)
               102  % There might be no output requested...
               103
< 0.001     140 104 sol = []; f3d = [];
< 0.001     140 105 if output_sol
               106    sol.solver = solver_name;
               107    sol.extdata.odefun = ode;
               108    sol.extdata.options = options;
               109    sol.extdata.varargin = varargin;
< 0.001     140 110 end
               111
               112  % Handle solver arguments
```

```
 0.070   140  113  [neq, tspan, ntspan, next, t0, tfinal, tdir, y0, f0, odeArgs, odeFcn, ...
         140  114    options, threshold, rtol, normcontrol, normy, hmax, htry, htspan, dataType] = ...
         140  115    odearguments(FcnHandlesUsed, solver_name, ode, tspan, y0, options, varargin);
< 0.001  140  116  nfevals = nfevals + 1;
              117
              118  % Handle the output
< 0.001  140  119  if nargout > 0
 0.004   140  120    outputFcn = odeget(options,'OutputFcn',[],'fast');
              121  else
              122    outputFcn = odeget(options,'OutputFcn',@odeplot,'fast');
< 0.001  140  123  end
< 0.001  140  124  outputArgs = {};
< 0.001  140  125  if isempty(outputFcn)
< 0.001  140  126    haveOutputFcn = false;
              127  else
              128    haveOutputFcn = true;
              129    outputs = odeget(options,'OutputSel',1:neq,'fast');
              130    if isa(outputFcn,'function_handle')
              131      % With MATLAB 6 syntax pass additional input arguments to outputFcn.
              132      outputArgs = varargin;
              133    end
< 0.001  140  134  end
 0.005   140  135  refine = max(1,odeget(options,'Refine',4,'fast'));
< 0.001  140  136  if ntspan > 2
              137    outputAt = 1;          % output only at tspan points
< 0.001  140  138  elseif refine <= 1
              139    outputAt = 2;          % computed points, no refinement
< 0.001  140  140  else
< 0.001  140  141    outputAt = 3;          % computed points, with refinement
 0.001   140  142    S = (1:refine-1) / refine;
< 0.001  140  143  end
 0.006   140  144  printstats = strcmp(odeget(options,'Stats','off','fast'),'on');
              145
              146  % Handle the event function
 0.030   140  147  [haveEventFcn,eventFcn,eventArgs,valt,teout,yeout,ieout] = ...
         140  148    odeevents(FcnHandlesUsed,odeFcn,t0,y0,options,varargin);
              149
              150  % Handle the mass matrix
 0.014   140  151  [Mtype, M, Mfun] =  odemass(FcnHandlesUsed,odeFcn,t0,y0,options,varargin);
< 0.001  140  152  if Mtype > 0  % non-trivial mass matrix
              153    Msingular = odeget(options,'MassSingular','no','fast');
              154    if strcmp(Msingular,'maybe')
              155      warning(message('MATLAB:ode45:MassSingularAssumedNo'));
              156    elseif strcmp(Msingular,'yes')
              157      error(message('MATLAB:ode45:MassSingularYes'));
              158    end
              159    % Incorporate the mass matrix into odeFcn and odeArgs.
              160    [odeFcn,odeArgs] = odemassexplicit(FcnHandlesUsed,Mtype,odeFcn,odeArgs,Mfun,M);
              161    f0 = feval(odeFcn,t0,y0,odeArgs{:});
              162    nfevals = nfevals + 1;
< 0.001  140  163  end
              164
              165  % Non-negative solution components
 0.004   140  166  idxNonNegative = odeget(options,'NonNegative',[],'fast');
< 0.001  140  167  nonNegative = ~isempty(idxNonNegative);
< 0.001  140  168  if nonNegative  % modify the derivative function
              169    [odeFcn,thresholdNonNegative] = odenonnegative(odeFcn,y0,threshold,idxNonNegative);
              170    f0 = feval(odeFcn,t0,y0,odeArgs{:});
              171    nfevals = nfevals + 1;
< 0.001  140  172  end
              173
< 0.001  140  174  t = t0;
< 0.001  140  175  y = y0;
              176
              177  % Allocate memory if we're generating output.
< 0.001
```

```
< 0.001   140   179  tout = []; yout = [];
< 0.001   140   180  if nargout > 0
< 0.001   140   181    if output_sol
                182      chunk = min(max(100,50*refine), refine+floor((2^11)/neq));
                183      tout = zeros(1,chunk,dataType);
                184      yout = zeros(neq,chunk,dataType);
                185      f3d  = zeros(neq,7,chunk,dataType);
< 0.001   140   186    else
< 0.001   140   187      if ntspan > 2                         % output only at tspan points
                188        tout = zeros(1,ntspan,dataType);
                189        yout = zeros(neq,ntspan,dataType);
< 0.001   140   190      else                                  % alloc in chunks
< 0.001   140   191        chunk = min(max(100,50*refine), refine+floor((2^13)/neq));
  0.002   140   192        tout = zeros(1,chunk,dataType);
  0.002   140   193        yout = zeros(neq,chunk,dataType);
< 0.001   140   194      end
< 0.001   140   195    end
< 0.001   140   196    nout = 1;
< 0.001   140   197    tout(nout) = t;
< 0.001   140   198    yout(:,nout) = y;
< 0.001   140   199  end
                200
                201  % Initialize method parameters.
< 0.001   140   202  pow = 1/5;
< 0.001   140   203  A = [1/5, 3/10, 4/5, 8/9, 1, 1]; % Still used by restarting criteria
                204  % B = [
                205  %     1/5          3/40     44/45    19372/6561      9017/3168       35/384
                206  %     0            9/40     -56/15   -25360/2187     -355/33         0
                207  %     0            0        32/9     64448/6561      46732/5247      500/1113
                208  %     0            0        0        -212/729        49/176          125/192
                209  %     0            0        0        0               -5103/18656     -2187/6784
                210  %     0            0        0        0               0               11/84
                211  %     0            0        0        0               0               0
                212  %     ];
                213  % E = [71/57600; 0; -71/16695; 71/1920; -17253/339200; 22/525; -1/40];
                214
                215  % Same values as above extracted as scalars (1 and 0 values ommitted)
  0.001   140   216  a2=cast(1/5,dataType);
< 0.001   140   217  a3=cast(3/10,dataType);
< 0.001   140   218  a4=cast(4/5,dataType);
< 0.001   140   219  a5=cast(8/9,dataType);
                220
< 0.001   140   221  b11=cast(1/5,dataType);
< 0.001   140   222  b21=cast(3/40,dataType);
< 0.001   140   223  b31=cast(44/45,dataType);
< 0.001   140   224  b41=cast(19372/6561,dataType);
< 0.001   140   225  b51=cast(9017/3168,dataType);
< 0.001   140   226  b61=cast(35/384,dataType);
< 0.001   140   227  b22=cast(9/40,dataType);
< 0.001   140   228  b32=cast(-56/15,dataType);
< 0.001   140   229  b42=cast(-25360/2187,dataType);
< 0.001   140   230  b52=cast(-355/33,dataType);
< 0.001   140   231  b33=cast(32/9,dataType);
< 0.001   140   232  b43=cast(64448/6561,dataType);
< 0.001   140   233  b53=cast(46732/5247,dataType);
< 0.001   140   234  b63=cast(500/1113,dataType);
< 0.001   140   235  b44=cast(-212/729,dataType);
< 0.001   140   236  b54=cast(49/176,dataType);
< 0.001   140   237  b64=cast(125/192,dataType);
< 0.001   140   238  b55=cast(-5103/18656,dataType);
< 0.001   140   239  b65=cast(-2187/6784,dataType);
< 0.001   140   240  b66=cast(11/84,dataType);
                241
< 0.001   140   242  e1=cast(71/57600,dataType);
< 0.001   140   243  e3=cast(-71/16695,dataType);
```

```
< 0.001     140   245  e5=cast(-17253/339200,dataType);
< 0.001     140   246  e6=cast(22/525,dataType);
< 0.001     140   247  e7=cast(-1/40,dataType);
                  248
< 0.001     140   249  hmin = 16*eps(t);
< 0.001     140   250  if isempty(htry)
                  251    % Compute an initial step size h using y'(t).
< 0.001     140   252    absh = min(hmax, htspan);
< 0.001     140   253    if normcontrol
                  254      rh = (norm(f0) / max(normy,threshold)) / (0.8 * rtol^pow);
< 0.001     140   255    else
  0.001     140   256      rh = norm(f0 ./ max(abs(y),threshold),inf) / (0.8 * rtol^pow);
< 0.001     140   257    end
  0.002     140   258    if absh * rh > 1
< 0.001     140   259      absh = 1 / rh;
< 0.001     140   260    end
< 0.001     140   261    absh = max(absh, hmin);
                  262  else
                  263    absh = min(hmax, max(hmin, htry));
< 0.001     140   264  end
< 0.001     140   265  f1 = f0;
                  266
                  267  % Initialize the output function.
< 0.001     140   268  if haveOutputFcn
                  269    feval(outputFcn,[t tfinal],y(outputs),'init',outputArgs{:});
< 0.001     140   270  end
                  271
                  272  % Cleanup the main ode function call
< 0.001     140   273  FcnUsed = isa(odeFcn,'function_handle');
  0.005     140   274  odeFcn_main = odefcncleanup(FcnUsed,odeFcn,odeArgs);
                  275
                  276  % THE MAIN LOOP
                  277
< 0.001     140   278  done = false;
< 0.001     140   279  while ~done
                  280
                  281    % By default, hmin is a small number such that t+hmin is only slightly
                  282    % different than t.  It might be 0 if t is 0.
  0.003   15189   283    hmin = 16*eps(t);
  0.001   15189   284    absh = min(hmax, max(hmin, absh));    % couldn't limit absh until new hmin
< 0.001   15189   285    h = tdir * absh;
                  286
                  287    % Stretch the step if within 10% of tfinal-t.
  0.001   15189   288    if 1.1*absh >= abs(tfinal - t)
                  289      h = tfinal - t;
                  290      absh = abs(h);
                  291      done = true;
< 0.001   15189   292    end
                  293
                  294    % LOOP FOR ADVANCING ONE STEP.
  0.001   15189   295    nofailed = true;                      % no failed attempts
  0.003   15189   296    while true
  0.008   15200   297      y2 = y + h .* (b11.*f1 );
  0.001   15200   298      t2 = t + h .* a2;
  0.215   15200   299      f2 = odeFcn_main(t2, y2);
                  300
  0.008   15200   301      y3 = y + h .* (b21.*f1 + b22.*f2 );
  0.001   15200   302      t3 = t + h .* a3;
  0.160   15200   303      f3 = odeFcn_main(t3, y3);
                  304
  0.008   15200   305      y4 = y + h .* (b31.*f1 + b32.*f2 + b33.*f3 );
  0.001   15200   306      t4 = t + h .* a4;
  0.156   15200   307      f4 = odeFcn_main(t4, y4);
                  308
  0.009   15200   309      y5 = y + h .* (b41.*f1 + b42.*f2 + b43.*f3 + b44.*f4 );
```

```
0.001   15200  310       t5 = t + h .* a5;
0.162   15200  311       f5 = odeFcn_main(t5, y5);
               312
0.009   15200  313       y6 = y + h .* (b51.*f1 + b52.*f2 + b53.*f3 + b54.*f4 + b55.*f5 );
0.001   15200  314       t6 = t + h;
0.156   15200  315       f6 = odeFcn_main(t6, y6);
               316
0.001   15200  317       tnew = t + h;
0.001   15200  318       if done
               319         tnew = tfinal;   % Hit end point exactly.
0.001   15200  320       end
0.001   15200  321       h = tnew - t;      % Purify h.
               322
0.019   15200  323       ynew = y + h.* ( b61.*f1 + b63.*f3 + b64.*f4 + b65.*f5 + b66.*f6 );
0.157   15200  324       f7 = odeFcn_main(tnew,ynew);
               325
0.001   15200  326       nfevals = nfevals + 6;
               327
               328       % Estimate the error.
< 0.001 15200  329       NNrejectStep = false;
0.009   15200  330       fE = f1*e1 + f3*e3 + f4*e4 + f5*e5 + f6*e6 + f7*e7;
0.001   15200  331       if normcontrol
               332         normynew = norm(ynew);
               333         errwt = max(max(normy,normynew),threshold);
               334         err = absh * (norm(fE) / errwt);
               335         if nonNegative && (err <= rtol) && any(ynew(idxNonNegative)<0)
               336           errNN = norm( max(0,-ynew(idxNonNegative)) ) / errwt ;
               337           if errNN > rtol
               338             err = errNN;
               339             NNrejectStep = true;
               340           end
               341         end
0.001   15200  342       else
0.026   15200  343         err = absh * norm((fE) ./ max(max(abs(y),abs(ynew)),threshold),inf);
0.001   15200  344         if nonNegative && (err <= rtol) && any(ynew(idxNonNegative)<0)
               345           errNN = norm( max(0,-ynew(idxNonNegative)) ./ thresholdNonNegative, inf);
               346           if errNN > rtol
               347             err = errNN;
               348             NNrejectStep = true;
               349           end
< 0.001 15200  350         end
0.001   15200  351       end
               352
               353       % Accept the solution only if the weighted error is no more than the
               354       % tolerance rtol.  Estimate an h that will yield an error of rtol on
               355       % the next step or the next try at taking this step, as the case may be,
               356       % and use 0.8 of this value to avoid failures.
0.002   15200  357       if err > rtol                       % Failed step
< 0.001    11  358         nfailed = nfailed + 1;
< 0.001    11  359         if absh <= hmin
               360           warning(message('MATLAB:ode45:IntegrationTolNotMet', sprintf( '%e', t ), sprintf( '%e', hmin )));
               361           solver_output = odefinalize(solver_name, sol,...
               362                                       outputFcn, outputArgs,...
               363                                       printstats, [nsteps, nfailed, nfevals],...
               364                                       nout, tout, yout,...
               365                                       haveEventFcn, teout, yeout, ieout,...
               366                                       {f3d,idxNonNegative});
               367           if nargout > 0
               368             varargout = solver_output;
               369           end
               370           return;
< 0.001    11  371         end
               372
< 0.001    11  373         if nofailed
< 0.001    11  374           nofailed = false;
< 0.001    11  375           if NNrejectStep
```

```
                  376             absh = max(hmin, 0.5*absh);
< 0.001       11  377           else
< 0.001       11  378             absh = max(hmin, absh * max(0.1, 0.8*(rtol/err)^pow));
< 0.001       11  379           end
                  380         else
                  381           absh = max(hmin, 0.5 * absh);
< 0.001       11  382         end
< 0.001       11  383         h = tdir * absh;
< 0.001       11  384         done = false;
                  385
< 0.001    15189  386       else                                % Successful step
                  387
< 0.001    15189  388         NNreset_f7 = false;
  0.001    15189  389         if nonNegative && any(ynew(idxNonNegative)<0)
                  390           ynew(idxNonNegative) = max(ynew(idxNonNegative),0);
                  391           if normcontrol
                  392             normynew = norm(ynew);
                  393           end
                  394           NNreset_f7 = true;
  0.001    15189  395         end
                  396
  0.002    15189  397         break;
                  398
< 0.001       11  399       end
< 0.001       11  400     end
  0.001    15189  401     nsteps = nsteps + 1;
                  402
  0.002    15189  403     if haveEventFcn
  0.019    15189  404       f = [f1 f2 f3 f4 f5 f6 f7];
  1.239    15189  405       [te,ye,ie,valt,stop] = ...
           15189  406           odezero(@ntrp45,eventFcn,eventArgs,valt,t,y,tnew,ynew,t0,h,f,idxNonNegative);
  0.001    15189  407       if ~isempty(te)
< 0.001      140  408         if output_sol || (nargout > 2)
< 0.001      139  409           teout = [teout, te];
< 0.001      139  410           yeout = [yeout, ye];
< 0.001      139  411           ieout = [ieout, ie];
< 0.001      140  412         end
< 0.001      140  413         if stop               % Stop on a terminal event.
                  414           % Adjust the interpolation data to [t te(end)].
                  415
                  416           % Update the derivatives using the interpolating polynomial.
< 0.001      140  417           taux = t + (te(end) - t)*A;
  0.016      140  418           [~,f(:,2:7)] = ntrp45(taux,t,y,[],[],h,f,idxNonNegative);
  0.002      140  419           f2 = f(:,2); f3 = f(:,3); f4 = f(:,4); f5 = f(:,5); f6 = f(:,6); f7 = f(:,7);
                  420
< 0.001      140  421           tnew = te(end);
< 0.001      140  422           ynew = ye(:,end);
< 0.001      140  423           h = tnew - t;
< 0.001      140  424           done = true;
< 0.001      140  425         end
  0.001    15189  426       end
< 0.001    15189  427     end
                  428
  0.001    15189  429     if output_sol
                  430       nout = nout + 1;
                  431       if nout > length(tout)
                  432         tout = [tout, zeros(1,chunk,dataType)];  % requires chunk >= refine
                  433         yout = [yout, zeros(neq,chunk,dataType)];
                  434         f3d  = cat(3,f3d,zeros(neq,7,chunk,dataType));
                  435       end
                  436       tout(nout) = tnew;
                  437       yout(:,nout) = ynew;
                  438       f3d(:,:,nout) = [f1 f2 f3 f4 f5 f6 f7];
  0.001    15189  439     end
                  440
  0.003    15189  441     if output_ty || haveOutputFcn
```

```
0.001   15189  442       switch outputAt
0.001   15189  443        case 2      % computed points, no refinement
               444         nout_new = 1;
               445         tout_new = tnew;
               446         yout_new = ynew;
0.003   15189  447        case 3      % computed points, with refinement
0.007   15189  448         tref = t + (tnew-t)*S;
0.001   15189  449         nout_new = refine;
0.047   15189  450         tout_new = [tref, tnew];
0.240   15189  451         yntrp45 = ntrp45split(tref,t,y,h,f1,f3,f4,f5,f6,f7,idxNonNegative);
0.030   15189  452         yout_new = [yntrp45, ynew];
               453        case 1      % output only at tspan points
               454         nout_new =  0;
               455         tout_new = [];
               456         yout_new = [];
               457         while next <= ntspan
               458           if tdir * (tnew - tspan(next)) < 0
               459             if haveEventFcn && stop     % output tstop,ystop
               460               nout_new = nout_new + 1;
               461               tout_new = [tout_new, tnew];
               462               yout_new = [yout_new, ynew];
               463             end
               464             break;
               465           end
               466           nout_new = nout_new + 1;
               467           tout_new = [tout_new, tspan(next)];
               468           if tspan(next) == tnew
               469             yout_new = [yout_new, ynew];
               470           else
               471             yntrp45 = ntrp45split(tspan(next),t,y,h,f1,f3,f4,f5,f6,f7,idxNonNegative);
               472             yout_new = [yout_new, yntrp45];
               473           end
               474           next = next + 1;
               475         end
0.001   15189  476       end
               477
0.002   15189  478       if nout_new > 0
0.003   15189  479         if output_ty
< 0.001 15189  480           oldnout = nout;
0.001   15189  481           nout = nout + nout_new;
0.002   15189  482           if nout > length(tout)
0.004     227  483             tout = [tout, zeros(1,chunk,dataType)];  % requires chunk >= refine
0.008     227  484             yout = [yout, zeros(neq,chunk,dataType)];
< 0.001 15189  485           end
0.020   15189  486           idx = oldnout+1:nout;
0.028   15189  487           tout(idx) = tout_new;
0.021   15189  488           yout(:,idx) = yout_new;
< 0.001 15189  489         end
0.001   15189  490         if haveOutputFcn
               491           stop = feval(outputFcn,tout_new,yout_new(outputs,:),'',outputArgs{:});
               492           if stop
               493             done = true;
               494           end
0.001   15189  495         end
< 0.001 15189  496       end
0.001   15189  497     end
               498
0.002   15189  499     if done
0.001     140  500       break
< 0.001 15049  501     end
               502
               503     % If there were no failures compute a new h.
0.003   15049  504     if nofailed
               505       % Note that absh may shrink by 0.8, and that err may be 0.
0.005   15038  506       temp = 1.25*(err/rtol)^pow;
0.003   15038  507       if temp > 0.2
```

< 0.001	15038	508	absh = absh / temp;
		509	else
		510	absh = 5.0*absh;
< 0.001	15038	511	end
< 0.001	15049	512	end
		513	
		514	% Advance the integration one step.
0.001	15049	515	t = tnew;
0.003	15049	516	y = ynew;
0.001	15049	517	if normcontrol
		518	normy = normynew;
0.001	15049	519	end
0.001	15049	520	if NNreset_f7
		521	% Used f7 for unperturbed solution to interpolate.
		522	% Now reset f7 to move along constraint.
		523	f7 = odeFcn_main(tnew,ynew);
		524	nfevals = nfevals + 1;
< 0.001	15049	525	end
0.004	15049	526	f1 = f7; % Already have f(tnew,ynew)
		527	
0.003	15049	528	end
		529	
0.035	140	530	solver_output = odefinalize(solver_name, sol,...
	140	531	outputFcn, outputArgs,...
	140	532	printstats, [nsteps, nfailed, nfevals],...
	140	533	nout, tout, yout,...
	140	534	haveEventFcn, teout, yeout, ieout,...
	140	535	{f3d,idxNonNegative});
0.004	140	536	if nargout > 0
< 0.001	140	537	varargout = solver_output;
0.005	140	538	end

Local functions in this file are not included in this listing.