function varargout = ode45(ode,tspan,y0,options,varargin)

%ODE45 Solve non-stiff differential equations, medium order method.

% [TOUT,YOUT] = ODE45(ODEFUN,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. ODEFUN is a function handle. For a scalar T

% and a vector Y, ODEFUN(T,Y) must return a column vector corresponding

% to f(t,y). Each row in the solution array YOUT corresponds to a time

% returned in the column vector TOUT. To obtain solutions at specific

% times T0,T1,...,TFINAL (all increasing or all decreasing), use TSPAN =

% [T0 T1 ... TFINAL].

%

% [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) solves as above with default

% integration properties 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).

% If certain components of the solution must be non-negative, use

% ODESET to set the 'NonNegative' property to the indices of these

% components.

%

% ODE45 can solve problems M(t,y)\*y' = f(t,y) with mass matrix M that is

% nonsingular. Use ODESET to set the 'Mass' property to a function handle

% MASS if MASS(T,Y) returns the value of the mass matrix. If the mass matrix

% is constant, the matrix can be used as the value of the 'Mass' option. If

% the mass matrix does not depend on the state variable Y and the function

% MASS is to be called with one input argument T, set 'MStateDependence' to

% 'none'. ODE15S and ODE23T can solve problems with singular mass matrices.

%

% [TOUT,YOUT,TE,YE,IE] = ODE45(ODEFUN,TSPAN,Y0,OPTIONS) with the 'Events'

% property in OPTIONS set to a function handle EVENTS, solves as above

% while also finding where functions of (T,Y), called event functions,

% are zero. For each function you specify whether the integration is

% to terminate at a zero and whether the direction of the zero crossing

% matters. These are the three column vectors returned by EVENTS:

% [VALUE,ISTERMINAL,DIRECTION] = EVENTS(T,Y). For the I-th event function:

% VALUE(I) is the value of the function, ISTERMINAL(I)=1 if the integration

% is to terminate at a zero of this event function and 0 otherwise.

% DIRECTION(I)=0 if all zeros are to be computed (the default), +1 if only

% zeros where the event function is increasing, and -1 if only zeros where

% the event function is decreasing. 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.

%

% SOL = ODE45(ODEFUN,[T0 TFINAL],Y0...) returns a structure that can be

% used with DEVAL to evaluate the solution or its first derivative at

% any point between T0 and TFINAL. The steps chosen by ODE45 are returned

% in a row vector SOL.x. For each I, the column SOL.y(:,I) contains

% the solution at SOL.x(I). If events were detected, SOL.xe is a row vector

% of points at which events occurred. Columns of SOL.ye are the corresponding

% solutions, and indices in vector SOL.ie specify which event occurred.

%

% Example

% [t,y]=ode45(@vdp1,[0 20],[2 0]);

% plot(t,y(:,1));

% solves the system y' = vdp1(t,y), using the default relative error

% tolerance 1e-3 and the default absolute tolerance of 1e-6 for each

% component, and plots the first component of the solution.

%

% Class support for inputs TSPAN, Y0, and the result of ODEFUN(T,Y):

% float: double, single

%

% See also ODE23, ODE113, ODE15S, ODE23S, ODE23T, ODE23TB, ODE15I,

% ODESET, ODEPLOT, ODEPHAS2, ODEPHAS3, ODEPRINT, DEVAL,

% ODEEXAMPLES, RIGIDODE, BALLODE, ORBITODE, FUNCTION\_HANDLE.

% ODE45 is an implementation of the explicit Runge-Kutta (4,5) pair of

% Dormand and Prince called variously RK5(4)7FM, DOPRI5, DP(4,5) and DP54.

% It uses a "free" interpolant of order 4 communicated privately by

% Dormand and Prince. 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 1984-2020 The MathWorks, Inc.

solver\_name = 'ode45';

% Check inputs

if nargin < 4

options = [];

if nargin < 3

y0 = [];

if nargin < 2

tspan = [];

if nargin < 1

error(message('MATLAB:ode45:NotEnoughInputs'));

end

end

end

end

% Stats

nsteps = 0;

nfailed = 0;

nfevals = 0;

% Output

FcnHandlesUsed = isa(ode,'function\_handle');

output\_sol = (FcnHandlesUsed && (nargout==1)); % sol = odeXX(...)

output\_ty = (~output\_sol && (nargout > 0)); % [t,y,...] = odeXX(...)

% There might be no output requested...

sol = []; f3d = [];

if output\_sol

sol.solver = solver\_name;

sol.extdata.odefun = ode;

sol.extdata.options = options;

sol.extdata.varargin = varargin;

end

% Handle solver arguments

[neq, tspan, ntspan, next, t0, tfinal, tdir, y0, f0, odeArgs, odeFcn, ...

options, threshold, rtol, normcontrol, normy, hmax, htry, htspan, dataType] = ...

odearguments(FcnHandlesUsed, solver\_name, ode, tspan, y0, options, varargin);

nfevals = nfevals + 1;

% Handle the output

if nargout > 0

outputFcn = odeget(options,'OutputFcn',[],'fast');

else

outputFcn = odeget(options,'OutputFcn',@odeplot,'fast');

end

outputArgs = {};

if isempty(outputFcn)

haveOutputFcn = false;

else

haveOutputFcn = true;

outputs = odeget(options,'OutputSel',1:neq,'fast');

if isa(outputFcn,'function\_handle')

% With MATLAB 6 syntax pass additional input arguments to outputFcn.

outputArgs = varargin;

end

end

refine = max(1,odeget(options,'Refine',4,'fast'));

if ntspan > 2

outputAt = 1; % output only at tspan points

elseif refine <= 1

outputAt = 2; % computed points, no refinement

else

outputAt = 3; % computed points, with refinement

S = (1:refine-1) / refine;

end

printstats = strcmp(odeget(options,'Stats','off','fast'),'on');

% Handle the event function

[haveEventFcn,eventFcn,eventArgs,valt,teout,yeout,ieout] = ...

odeevents(FcnHandlesUsed,odeFcn,t0,y0,options,varargin);

% Handle the mass matrix

[Mtype, M, Mfun] = odemass(FcnHandlesUsed,odeFcn,t0,y0,options,varargin);

if Mtype > 0 % non-trivial mass matrix

Msingular = odeget(options,'MassSingular','no','fast');

if strcmp(Msingular,'maybe')

warning(message('MATLAB:ode45:MassSingularAssumedNo'));

elseif strcmp(Msingular,'yes')

error(message('MATLAB:ode45:MassSingularYes'));

end

% Incorporate the mass matrix into odeFcn and odeArgs.

[odeFcn,odeArgs] = odemassexplicit(FcnHandlesUsed,Mtype,odeFcn,odeArgs,Mfun,M);

f0 = feval(odeFcn,t0,y0,odeArgs{:});

nfevals = nfevals + 1;

end

% Non-negative solution components

idxNonNegative = odeget(options,'NonNegative',[],'fast');

nonNegative = ~isempty(idxNonNegative);

if nonNegative % modify the derivative function

[odeFcn,thresholdNonNegative] = odenonnegative(odeFcn,y0,threshold,idxNonNegative);

f0 = feval(odeFcn,t0,y0,odeArgs{:});

nfevals = nfevals + 1;

end

t = t0;

y = y0;

% Allocate memory if we're generating output.

nout = 0;

tout = []; yout = [];

if nargout > 0

if output\_sol

chunk = min(max(100,50\*refine), refine+floor((2^11)/neq));

tout = zeros(1,chunk,dataType);

yout = zeros(neq,chunk,dataType);

f3d = zeros(neq,7,chunk,dataType);

else

if ntspan > 2 % output only at tspan points

tout = zeros(1,ntspan,dataType);

yout = zeros(neq,ntspan,dataType);

else % alloc in chunks

chunk = min(max(100,50\*refine), refine+floor((2^13)/neq));

tout = zeros(1,chunk,dataType);

yout = zeros(neq,chunk,dataType);

end

end

nout = 1;

tout(nout) = t;

yout(:,nout) = y;

end

% Initialize method parameters.

pow = 1/5;

A = [1/5, 3/10, 4/5, 8/9, 1, 1]; % Still used by restarting criteria

% B = [

% 1/5 3/40 44/45 19372/6561 9017/3168 35/384

% 0 9/40 -56/15 -25360/2187 -355/33 0

% 0 0 32/9 64448/6561 46732/5247 500/1113

% 0 0 0 -212/729 49/176 125/192

% 0 0 0 0 -5103/18656 -2187/6784

% 0 0 0 0 0 11/84

% 0 0 0 0 0 0

% ];

% E = [71/57600; 0; -71/16695; 71/1920; -17253/339200; 22/525; -1/40];

% Same values as above extracted as scalars (1 and 0 values ommitted)

a2=cast(1/5,dataType);

a3=cast(3/10,dataType);

a4=cast(4/5,dataType);

a5=cast(8/9,dataType);

b11=cast(1/5,dataType);

b21=cast(3/40,dataType);

b31=cast(44/45,dataType);

b41=cast(19372/6561,dataType);

b51=cast(9017/3168,dataType);

b61=cast(35/384,dataType);

b22=cast(9/40,dataType);

b32=cast(-56/15,dataType);

b42=cast(-25360/2187,dataType);

b52=cast(-355/33,dataType);

b33=cast(32/9,dataType);

b43=cast(64448/6561,dataType);

b53=cast(46732/5247,dataType);

b63=cast(500/1113,dataType);

b44=cast(-212/729,dataType);

b54=cast(49/176,dataType);

b64=cast(125/192,dataType);

b55=cast(-5103/18656,dataType);

b65=cast(-2187/6784,dataType);

b66=cast(11/84,dataType);

e1=cast(71/57600,dataType);

e3=cast(-71/16695,dataType);

e4=cast(71/1920,dataType);

e5=cast(-17253/339200,dataType);

e6=cast(22/525,dataType);

e7=cast(-1/40,dataType);

hmin = 16\*eps(t);

if isempty(htry)

% Compute an initial step size h using y'(t).

absh = min(hmax, htspan);

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

f1 = f0;

% Initialize the output function.

if haveOutputFcn

feval(outputFcn,[t tfinal],y(outputs),'init',outputArgs{:});

end

% Cleanup the main ode function call

FcnUsed = isa(odeFcn,'function\_handle');

odeFcn\_main = odefcncleanup(FcnUsed,odeFcn,odeArgs);

% 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(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

y2 = y + h .\* (b11.\*f1 );

t2 = t + h .\* a2;

f2 = odeFcn\_main(t2, y2);

y3 = y + h .\* (b21.\*f1 + b22.\*f2 );

t3 = t + h .\* a3;

f3 = odeFcn\_main(t3, y3);

y4 = y + h .\* (b31.\*f1 + b32.\*f2 + b33.\*f3 );

t4 = t + h .\* a4;

f4 = odeFcn\_main(t4, y4);

y5 = y + h .\* (b41.\*f1 + b42.\*f2 + b43.\*f3 + b44.\*f4 );

t5 = t + h .\* a5;

f5 = odeFcn\_main(t5, y5);

y6 = y + h .\* (b51.\*f1 + b52.\*f2 + b53.\*f3 + b54.\*f4 + b55.\*f5 );

t6 = t + h;

f6 = odeFcn\_main(t6, y6);

tnew = t + h;

if done

tnew = tfinal; % Hit end point exactly.

end

h = tnew - t; % Purify h.

ynew = y + h.\* ( b61.\*f1 + b63.\*f3 + b64.\*f4 + b65.\*f5 + b66.\*f6 );

f7 = odeFcn\_main(tnew,ynew);

nfevals = nfevals + 6;

% Estimate the error.

NNrejectStep = false;

fE = f1\*e1 + f3\*e3 + f4\*e4 + f5\*e5 + f6\*e6 + f7\*e7;

if normcontrol

normynew = norm(ynew);

errwt = max(max(normy,normynew),threshold);

err = absh \* (norm(fE) / errwt);

if nonNegative && (err <= rtol) && any(ynew(idxNonNegative)<0)

errNN = norm( max(0,-ynew(idxNonNegative)) ) / errwt ;

if errNN > rtol

err = errNN;

NNrejectStep = true;

end

end

else

err = absh \* norm((fE) ./ max(max(abs(y),abs(ynew)),threshold),inf);

if nonNegative && (err <= rtol) && any(ynew(idxNonNegative)<0)

errNN = norm( max(0,-ynew(idxNonNegative)) ./ thresholdNonNegative, inf);

if errNN > rtol

err = errNN;

NNrejectStep = true;

end

end

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;

if absh <= hmin

warning(message('MATLAB:ode45:IntegrationTolNotMet', sprintf( '%e', t ), sprintf( '%e', hmin )));

solver\_output = odefinalize(solver\_name, sol,...

outputFcn, outputArgs,...

printstats, [nsteps, nfailed, nfevals],...

nout, tout, yout,...

haveEventFcn, teout, yeout, ieout,...

{f3d,idxNonNegative});

if nargout > 0

varargout = solver\_output;

end

return;

end

if nofailed

nofailed = false;

if NNrejectStep

absh = max(hmin, 0.5\*absh);

else

absh = max(hmin, absh \* max(0.1, 0.8\*(rtol/err)^pow));

end

else

absh = max(hmin, 0.5 \* absh);

end

h = tdir \* absh;

done = false;

else % Successful step

NNreset\_f7 = false;

if nonNegative && any(ynew(idxNonNegative)<0)

ynew(idxNonNegative) = max(ynew(idxNonNegative),0);

if normcontrol

normynew = norm(ynew);

end

NNreset\_f7 = true;

end

break;

end

end

nsteps = nsteps + 1;

if haveEventFcn

f = [f1 f2 f3 f4 f5 f6 f7];

[te,ye,ie,valt,stop] = ...

odezero(@ntrp45,eventFcn,eventArgs,valt,t,y,tnew,ynew,t0,h,f,idxNonNegative);

if ~isempty(te)

if output\_sol || (nargout > 2)

teout = [teout, te]; %#ok<AGROW>

yeout = [yeout, ye]; %#ok<AGROW>

ieout = [ieout, ie]; %#ok<AGROW>

end

if stop % Stop on a terminal event.

% Adjust the interpolation data to [t te(end)].

% Update the derivatives using the interpolating polynomial.

taux = t + (te(end) - t)\*A;

[~,f(:,2:7)] = ntrp45(taux,t,y,[],[],h,f,idxNonNegative);

f2 = f(:,2); f3 = f(:,3); f4 = f(:,4); f5 = f(:,5); f6 = f(:,6); f7 = f(:,7);

tnew = te(end);

ynew = ye(:,end);

h = tnew - t;

done = true;

end

end

end

if output\_sol

nout = nout + 1;

if nout > length(tout)

tout = [tout, zeros(1,chunk,dataType)]; %#ok<AGROW> % requires chunk >= refine

yout = [yout, zeros(neq,chunk,dataType)]; %#ok<AGROW>

f3d = cat(3,f3d,zeros(neq,7,chunk,dataType));

end

tout(nout) = tnew; %#ok<AGROW>

yout(:,nout) = ynew; %#ok<AGROW>

f3d(:,:,nout) = [f1 f2 f3 f4 f5 f6 f7]; %#ok<AGROW>

end

if output\_ty || haveOutputFcn

switch outputAt

case 2 % computed points, no refinement

nout\_new = 1;

tout\_new = tnew;

yout\_new = ynew;

case 3 % computed points, with refinement

tref = t + (tnew-t)\*S;

nout\_new = refine;

tout\_new = [tref, tnew];

yntrp45 = ntrp45split(tref,t,y,h,f1,f3,f4,f5,f6,f7,idxNonNegative);

yout\_new = [yntrp45, ynew];

case 1 % output only at tspan points

nout\_new = 0;

tout\_new = [];

yout\_new = [];

while next <= ntspan

if tdir \* (tnew - tspan(next)) < 0

if haveEventFcn && stop % output tstop,ystop

nout\_new = nout\_new + 1;

tout\_new = [tout\_new, tnew]; %#ok<AGROW>

yout\_new = [yout\_new, ynew]; %#ok<AGROW>

end

break;

end

nout\_new = nout\_new + 1;

tout\_new = [tout\_new, tspan(next)]; %#ok<AGROW>

if tspan(next) == tnew

yout\_new = [yout\_new, ynew]; %#ok<AGROW>

else

yntrp45 = ntrp45split(tspan(next),t,y,h,f1,f3,f4,f5,f6,f7,idxNonNegative);

yout\_new = [yout\_new, yntrp45]; %#ok<AGROW>

end

next = next + 1;

end

end

if nout\_new > 0

if output\_ty

oldnout = nout;

nout = nout + nout\_new;

if nout > length(tout)

tout = [tout, zeros(1,chunk,dataType)]; %#ok<AGROW> requires chunk >= refine

yout = [yout, zeros(neq,chunk,dataType)]; %#ok<AGROW>

end

idx = oldnout+1:nout;

tout(idx) = tout\_new; %#ok<AGROW>

yout(:,idx) = yout\_new; %#ok<AGROW>

end

if haveOutputFcn

stop = feval(outputFcn,tout\_new,yout\_new(outputs,:),'',outputArgs{:});

if stop

done = true;

end

end

end

end

if done

break

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

if NNreset\_f7

% Used f7 for unperturbed solution to interpolate.

% Now reset f7 to move along constraint.

f7 = odeFcn\_main(tnew,ynew);

nfevals = nfevals + 1;

end

f1 = f7; % Already have f(tnew,ynew)

end

solver\_output = odefinalize(solver\_name, sol,...

outputFcn, outputArgs,...

printstats, [nsteps, nfailed, nfevals],...

nout, tout, yout,...

haveEventFcn, teout, yeout, ieout,...

{f3d,idxNonNegative});

if nargout > 0

varargout = solver\_output;

end