```
> des := {delta(t) = 0.01*t, delta(t0) = delta0, diff(phi(t), t) =
tan(delta(t))/l, diff(x(t), t) = V*cos(phi(t)), diff(y(t), t) =
V*sin(phi(t)), phi(t0) = phi0, x(t0) = x0, y(t0) = y0};
> desSol := dsolve(des, numeric, parameters = [V, l, t0, x0, y0, phi0,
delta0]);
> desSol(parameters = [1.3, 5, 0, 0, 0, 0, 0]);
> desSol(90);
> with(plots);
> odeplot(desSol, [[t, x(t)], [t, y(t)], [t, phi(t)]], 0 .. 300, labels =
["t", "Trajectory"], legend = ["x(t) - Position in x", "y(t) - Position
in y", "phi(t) - Heading Angle"]);
> odeplot(desSol, [t, x(t), y(t)], 0 .. 300);
> odeplot(desSol, [x(t), y(t), phi(t)], 0 .. 150, labels = ["x -
Position", "y - Position", "phi - Heading Angle"]);
        /
des := { delta(t) = 0.01 t, delta(t0) = delta0,
        \

   d           tan(delta(t))   d
  --- phi(t) = -------------, --- x(t) = V cos(phi(t)),
   dt                 l        dt

   d
  --- y(t) = V sin(phi(t)), phi(t0) = phi0, x(t0) = x0, y(t0) = y0
   dt

  \
   }
  /
desSol := proc (x_rkf45_dae) local _res, _dat, _vars,

   _solnproc, _xout, _ndsol, _pars, _n, _i; option `Copyright

   (c) 2000 by Waterloo Maple Inc. All rights reserved.`; if 1

   < nargs then error "invalid input: too many arguments" end

   if; _EnvDSNumericSaveDigits := Digits; Digits := 15; if

   _EnvInFsolve = true then _xout := evalf[_EnvDSNumericSaveDigi\

  ts](x_rkf45_dae) else _xout := evalf(x_rkf45_dae) end if;

   _dat := Array(1..4, {(1) = proc (_xin) local _xout, _dtbl,

   _dat, _vmap, _x0, _y0, _val, _dig, _n, _ne, _nd, _nv, _pars,

   _ini, _par, _i, _j, _k, _src; option `Copyright (c) 2002 by

   Waterloo Maple Inc. All rights reserved.`; table( [(

   "complex" ) = false ] ) _xout := _xin; _pars := [V = V, l =

   l, t0 = t0, x0 = x0, y0 = y0, phi0 = phi0, delta0 = delta0];

   _dtbl := array( 1 .. 4, [( 1 ) = (array( 1 .. 26, [( 1 ) =

   (datatype = float[8], order = C_order, storage = rectangular)\
```

, (2) = (datatype = float[8], order = C_order, storage =
rectangular), (3) = ([0, 0, 0, Array(1..0, 1..2, {},
datatype = float[8], order = C_order)]), (4) = (Array(1..63\
, {(1) = 4, (2) = 3, (3) = 0, (4) = 0, (5) = 7, (6) = 0, (7)
= 0, (8) = 0, (9) = 0, (10) = 0, (11) = 0, (12) = 0, (13) =
0, (14) = 0, (15) = 0, (16) = 0, (17) = 0, (18) = 0, (19) =
30000, (20) = 0, (21) = 0, (22) = 1, (23) = 4, (24) = 0,
(25) = 1, (26) = 15, (27) = 1, (28) = 0, (29) = 1, (30) = 3,
(31) = 3, (32) = 0, (33) = 1, (34) = 0, (35) = 0, (36) = 0,
(37) = 0, (38) = 0, (39) = 0, (40) = 0, (41) = 0, (42) = 0,
(43) = 1, (44) = 0, (45) = 0, (46) = 0, (47) = 0, (48) = 0,
(49) = 0, (50) = 50, (51) = 1, (52) = 0, (53) = 0, (54) = 0,
(55) = 0, (56) = 0, (57) = 0, (58) = 0, (59) = 10000, (60) =
0, (61) = 1000, (62) = 0, (63) = 0}, datatype = integer[8])),\
(5) = (Array(1..28, {(1) = undefined, (2) = 0.10e-5, (3) =
.0, (4) = 0.500001e-14, (5) = undefined, (6) = .0, (7) = .0,
(8) = 0.10e-5, (9) = .0, (10) = .0, (11) = .0, (12) = .0,
(13) = 1.0, (14) = .0, (15) = .49999999999999, (16) = .0,
(17) = 1.0, (18) = 1.0, (19) = .0, (20) = .0, (21) = 1.0,
(22) = 1.0, (23) = .0, (24) = .0, (25) = 0.10e-14, (26) =
.0, (27) = .0, (28) = .0}, datatype = float[8], order =
C_order)), (6) = (Array(1..11, {(1) = phi0, (2) = x0, (3)
= y0, (4) = Float(undefined), (5) = Float(undefined), (6) =
Float(undefined), (7) = Float(undefined), (8) = Float(undefin\
ed), (9) = Float(undefined), (10) = Float(undefined), (11) =
Float(undefined)})), (7) = ([Array(1..4, 1..7, {(1, 1) =
.0, (1, 2) = .203125, (1, 3) = .3046875, (1, 4) = .75, (1,
5) = .8125, (1, 6) = .40625, (1, 7) = .8125, (2, 1) =
0.6378173828125e-1, (2, 2) = .0, (2, 3) = .279296875, (2, 4)

```
 = .27237892150878906, (2, 5) = -0.9686851501464844e-1, (2,
 6) = 0.1956939697265625e-1, (2, 7) = .5381584167480469, (3,
 1) = 0.31890869140625e-1, (3, 2) = .0, (3, 3) = -.34375, (3,
 4) = -.335235595703125, (3, 5) = .2296142578125, (3, 6) =
 .41748046875, (3, 7) = 11.480712890625, (4, 1) = 0.9710520505\
905151e-1, (4, 2) = .0, (4, 3) = .40350341796875, (4, 4) =
 0.20297467708587646e-1, (4, 5) = -0.6054282188415527e-2, (4,
 6) = -0.4770040512084961e-1, (4, 7) = .77858567237854},
 datatype = float[8], order = C_order), Array(1..6, 1..6,
 {(1, 1) = .0, (1, 2) = .0, (1, 3) = .0, (1, 4) = .0, (1, 5)
 = .0, (1, 6) = 1.0, (2, 1) = .25, (2, 2) = .0, (2, 3) = .0,
 (2, 4) = .0, (2, 5) = .0, (2, 6) = 1.0, (3, 1) = .1875, (3,
 2) = .5625, (3, 3) = .0, (3, 4) = .0, (3, 5) = .0, (3, 6) =
 2.0, (4, 1) = .23583984375, (4, 2) = -.87890625, (4, 3) =
 .890625, (4, 4) = .0, (4, 5) = .0, (4, 6) = .2681884765625,
 (5, 1) = .1272735595703125, (5, 2) = -.5009765625, (5, 3) =
 .44921875, (5, 4) = -0.128936767578125e-1, (5, 5) = .0, (5,
 6) = 0.626220703125e-1, (6, 1) = -0.927734375e-1, (6, 2) =
 .626220703125, (6, 3) = -.4326171875, (6, 4) = .1418304443359\
375, (6, 5) = -0.861053466796875e-1, (6, 6) = .3131103515625},\
 datatype = float[8], order = C_order), Array(1..6, {(1) =
 .0, (2) = .386, (3) = .21, (4) = .63, (5) = 1.0, (6) = 1.0},
 datatype = float[8], order = C_order), Array(1..6, {(1) =
 .25, (2) = -.1043, (3) = .1035, (4) = -0.362e-1, (5) = .0,
 (6) = .0}, datatype = float[8], order = C_order),
 Array(1..6, 1..5, {(1, 1) = .0, (1, 2) = .0, (1, 3) = .0,
 (1, 4) = .0, (1, 5) = .0, (2, 1) = 1.544, (2, 2) = .0, (2,
 3) = .0, (2, 4) = .0, (2, 5) = .0, (3, 1) = .9466785280815533\
, (3, 2) = .25570116989825814, (3, 3) = .0, (3, 4) = .0, (3,
```

```
 5) = .0, (4, 1) = 3.3148251870684886, (4, 2) = 2.896124015972\
123, (4, 3) = .9986419139977808, (4, 4) = .0, (4, 5) = .0,
 (5, 1) = 1.2212245092262748, (5, 2) = 6.019134481287752, (5,
 3) = 12.537083329320874, (5, 4) = -.687886036105895, (5, 5)
 = .0, (6, 1) = 1.2212245092262748, (6, 2) = 6.019134481287752\
, (6, 3) = 12.537083329320874, (6, 4) = -.687886036105895,
 (6, 5) = 1.0}, datatype = float[8], order = C_order),
 Array(1..6, 1..5, {(1, 1) = .0, (1, 2) = .0, (1, 3) = .0,
 (1, 4) = .0, (1, 5) = .0, (2, 1) = -5.6688, (2, 2) = .0, (2,
 3) = .0, (2, 4) = .0, (2, 5) = .0, (3, 1) = -2.43009335683375\
84, (3, 2) = -.20635991570891224, (3, 3) = .0, (3, 4) = .0,
 (3, 5) = .0, (4, 1) = -.10735290581452621, (4, 2) =
 -9.594562251021896, (4, 3) = -20.470286148096154, (4, 4) =
 .0, (4, 5) = .0, (5, 1) = 7.496443313968615, (5, 2) =
 -10.246804314641219, (5, 3) = -33.99990352819906, (5, 4) =
 11.708908932061595, (5, 5) = .0, (6, 1) = 8.083246795922411,
 (6, 2) = -7.981132988062785, (6, 3) = -31.52159432874373,
 (6, 4) = 16.319305431231363, (6, 5) = -6.0588182388340535},
 datatype = float[8], order = C_order), Array(1..3, 1..5,
 {(1, 1) = .0, (1, 2) = .0, (1, 3) = .0, (1, 4) = .0, (1, 5)
 = .0, (2, 1) = 10.126235083446911, (2, 2) = -7.48799587760763\
3, (2, 3) = -34.800918615557414, (2, 4) = -7.9927717075687275,\
 (2, 5) = 1.0251377232956207, (3, 1) = -.6762803392806898,
 (3, 2) = 6.087714651678606, (3, 3) = 16.43084320892463, (3,
 4) = 24.767225114183653, (3, 5) = -6.5943891257167815},
 datatype = float[8], order = C_order)]), ( 9 ) = ([Array(1..4\
, {(1) = .1, (2) = .1, (3) = .1, (4) = .1}, datatype =
 float[8], order = C_order), Array(1..3, {(1) = .0, (2) = .0,
 (3) = .0}, datatype = float[8], order = C_order),
```

Array(1..3, {(1) = .0, (2) = .0, (3) = .0}, datatype =
float[8], order = C_order), Array(1..3, {(1) = .0, (2) = .0,
(3) = .0}, datatype = float[8], order = C_order),
Array(1..3, {(1) = .0, (2) = .0, (3) = .0}, datatype =
float[8], order = C_order), Array(1..3, 1..3, {(1, 1) = .0,
(1, 2) = .0, (1, 3) = .0, (2, 1) = .0, (2, 2) = .0, (2, 3) =
.0, (3, 1) = .0, (3, 2) = .0, (3, 3) = .0}, datatype =
float[8], order = C_order), Array(1..3, 1..3, {(1, 1) = .0,
(1, 2) = .0, (1, 3) = .0, (2, 1) = .0, (2, 2) = .0, (2, 3) =
.0, (3, 1) = .0, (3, 2) = .0, (3, 3) = .0}, datatype =
float[8], order = C_order), Array(1..3, {(1) = .0, (2) = .0,
(3) = .0}, datatype = float[8], order = C_order),
Array(1..3, 1..3, {(1, 1) = .0, (1, 2) = .0, (1, 3) = .0,
(2, 1) = .0, (2, 2) = .0, (2, 3) = .0, (3, 1) = .0, (3, 2) =
.0, (3, 3) = .0}, datatype = float[8], order = C_order),
Array(1..4, 1..6, {(1, 1) = .0, (1, 2) = .0, (1, 3) = .0,
(1, 4) = .0, (1, 5) = .0, (1, 6) = .0, (2, 1) = .0, (2, 2) =
.0, (2, 3) = .0, (2, 4) = .0, (2, 5) = .0, (2, 6) = .0, (3,
1) = .0, (3, 2) = .0, (3, 3) = .0, (3, 4) = .0, (3, 5) = .0,
(3, 6) = .0, (4, 1) = .0, (4, 2) = .0, (4, 3) = .0, (4, 4) =
.0, (4, 5) = .0, (4, 6) = .0}, datatype = float[8], order =
C_order), Array(1..3, {(1) = 0, (2) = 0, (3) = 0}, datatype
= integer[8]), Array(1..11, {(1) = .0, (2) = .0, (3) = .0,
(4) = .0, (5) = .0, (6) = .0, (7) = .0, (8) = .0, (9) = .0,
(10) = .0, (11) = .0}, datatype = float[8], order =
C_order), Array(1..11, {(1) = .0, (2) = .0, (3) = .0, (4) =
.0, (5) = .0, (6) = .0, (7) = .0, (8) = .0, (9) = .0, (10) =
.0, (11) = .0}, datatype = float[8], order = C_order),
Array(1..11, {(1) = .0, (2) = .0, (3) = .0, (4) = .0, (5) =

.0, (6) = .0, (7) = .0, (8) = .0, (9) = .0, (10) = .0, (11) = .0}, datatype = float[8], order = C_order), Array(1..11, {(1) = .0, (2) = .0, (3) = .0, (4) = .0, (5) = .0, (6) = .0, (7) = .0, (8) = .0, (9) = .0, (10) = .0, (11) = .0}, datatype = float[8], order = C_order), Array(1..3, {(1) = .0, (2) = .0, (3) = .0}, datatype = float[8], order = C_order), Array(1..8, {(1) = .0, (2) = .0, (3) = .0, (4) = .0, (5) = .0, (6) = .0, (7) = .0, (8) = .0}, datatype = float[8], order = C_order), Array(1..3, {(1) = 0, (2) = 0, (3) = 0}, datatype = integer[8])]), (8) = ([Array(1..11, {(1) = .0, (2) = .0, (3) = .0, (4) = .0, (5) = .0, (6) = .0, (7) = .0, (8) = .0, (9) = .0, (10) = .0, (11) = .0}, datatype = float[8], order = C_order), Array(1..11, {(1) = .0, (2) = .0, (3) = .0, (4) = .0, (5) = .0, (6) = .0, (7) = .0, (8) = .0, (9) = .0, (10) = .0, (11) = .0}, datatype = float[8], order = C_order), Array(1..3, {(1) = .0, (2) = .0, (3) = .0}, datatype = float[8], order = C_order), 0, 0]), (11) = (Array(1..6, 0..4, {(1, 1) = .0, (1, 2) = .0, (1, 3) = .0, (1, 4) = .0, (2, 0) = .0, (2, 1) = .0, (2, 2) = .0, (2, 3) = .0, (2, 4) = .0, (3, 0) = .0, (3, 1) = .0, (3, 2) = .0, (3, 3) = .0, (3, 4) = .0, (4, 0) = .0, (4, 1) = .0, (4, 2) = .0, (4, 3) = .0, (4, 4) = .0, (5, 0) = .0, (5, 1) = .0, (5, 2) = .0, (5, 3) = .0, (5, 4) = .0, (6, 0) = .0, (6, 1) = .0, (6, 2) = .0, (6, 3) = .0, (6, 4) = .0}, datatype = float[8], order = C_order)), (10) = ([proc (N, X, Y, YP) Y[4] := (1/100)*X; if N < 1 then return 0 end if; YP[1] := tan(0.10000000000000000000e-1*X)/Y[6]; YP[2] := Y[5]*cos(Y[1]); YP[3] := Y[5]*sin(Y[1]); 0 end proc, -1, 0, 0, 0, 0, 0, 0, 0, 0]), (13) = (), (12) = (), (15) =

("rkf45"), (14) = ([0, 0]), (18) = ([]), (19) = (0), (16) = ([0, 0, 0, 0, 0, []]), (17) = ([proc (N, X, Y, YP) Y[4] := (1/100)*X; if N < 1 then return 0 end if; YP[1] := tan(0.10000000000000000000e-1*X)/Y[6]; YP[2] := Y[5]*cos(Y[1]); YP[3] := Y[5]*sin(Y[1]); 0 end proc, -1, 0, 0, 0, 0, 0, 0, 0, 0]), (22) = (0), (23) = (0), (20) = ([]), (21) = (0), (26) = (Array(1..0, {})), (25) = (Array(1..0, {})), (24) = (0)]))]); _y0 := Array(0..11, {(1) = t0, (2) = phi0, (3) = x0, (4) = y0, (5) = Float(undefined), (6) = undefined, (7) = undefined, (8) = undefined, (9) = undefined, (10) = undefined, (11) = undefined}); _vmap := array(1 .. 4, [(1) = (4), (2) = (1), (3) = (2), (4) = (3)]); _x0 := _dtbl[1][5][5]; _n := _dtbl[1][4][1]; _ne := _dtbl[1][4][3]; _nd := _dtbl[1][4][4]; _nv := _dtbl[1][4][16]; if not type(_xout, 'numeric') then if member(_xout, ["start", "left", "right"]) then if _Env_smart_dsolve_numeric = true or _dtbl[1][4][10] = 1 then if _xout = "left" then if type(_dtbl[2], 'table') then return _dtbl[2][5][1] end if elif _xout = "right" then if type(_dtbl[3], 'table') then return _dtbl[3][5][1] end if end if end if; return _dtbl[1][5][5] elif _xout = "method" then return _dtbl[1][15] elif _xout = "storage" then return evalb(_dtbl[1][4][10] = 1) elif _xout = "leftdata" then if not type(_dtbl[2], 'array') then return NULL else return eval(_dtbl[2]) end if elif _xout = "rightdata" then if not type(_dtbl[3], 'array') then return NULL else return eval(_dtbl[3]) end if elif _xout = "enginedata" then return eval(_dtbl[1]) elif _xout = "enginereset" then _dtbl[2] := evaln(_dtbl[2]); _dtbl[3] := evaln(_dtbl[3]); return NULL

```
 elif _xout = "initial" then return procname(_y0[0]) elif
 _xout = "laxtol" then return _dtbl[`if`(member(_dtbl[4], {2,
 3}), _dtbl[4], 1)][5][18] elif _xout = "numfun" then return
 `if`(member(_dtbl[4], {2, 3}), _dtbl[_dtbl[4]][4][18], 0)
 elif _xout = "parameters" then return [seq(_y0[_n+_i], _i =
 1 .. nops(_pars))] elif _xout = "initial_and_parameters"
 then return procname(_y0[0]), [seq(_y0[_n+_i], _i = 1 ..
 nops(_pars))] elif _xout = "last" then if _dtbl[4] <> 2 and
 _dtbl[4] <> 3 or _x0-_dtbl[_dtbl[4]][5][1] = 0. then error
 "no information is available on last computed point" else
 _xout := _dtbl[_dtbl[4]][5][1] end if elif _xout =
 "function" then if _dtbl[1][4][33]-2. = 0 then return
 eval(_dtbl[1][10], 1) else return eval(_dtbl[1][10][1], 1)
 end if elif _xout = "map" then return copy(_vmap) elif
 type(_xin, `=`) and type(rhs(_xin), 'list') and member(lhs(_x\
in), {"initial", "parameters", "initial_and_parameters"})
 then _ini, _par := [], []; if lhs(_xin) = "initial" then
 _ini := rhs(_xin) elif lhs(_xin) = "parameters" then _par :=
 rhs(_xin) elif select(type, rhs(_xin), `=`) <> [] then _par,
 _ini := selectremove(type, rhs(_xin), `=`) elif nops(rhs(_xin\
)) < nops(_pars)+1 then error "insufficient data for
 specification of initial and parameters" else _par :=
 rhs(_xin)[-nops(_pars) .. -1]; _ini := rhs(_xin)[1 ..
 -nops(_pars)-1] end if; _xout := lhs(_xout); _i := false; if
 _par <> [] then _i := `dsolve/numeric/process_parameters`(_n,\
 _pars, _par, _y0) end if; if _ini <> [] then _i :=
 `dsolve/numeric/process_initial`(_n-_ne, _ini, _y0, _pars,
 _vmap) or _i end if; if _i then `dsolve/numeric/SC/reinitiali\
ze`(_dtbl, _y0, _n, procname, _pars); if _Env_smart_dsolve_num\
```

```
eric = true and type(_y0[0], 'numeric') and _dtbl[1][4][10]
 <> 1 then procname("right") := _y0[0]; procname("left") :=
 _y0[0] end if end if; if _xout = "initial" then return
 [_y0[0], seq(_y0[_vmap[_i]], _i = 1 .. _n-_ne)] elif _xout =
 "parameters" then return [seq(_y0[_n+_i], _i = 1 ..
 nops(_pars))] else return [_y0[0], seq(_y0[_vmap[_i]], _i =
 1 .. _n-_ne)], [seq(_y0[_n+_i], _i = 1 .. nops(_pars))] end
 if elif _xin = "eventstop" then if _nv = 0 then error "this
 solution has no events" end if; _i := _dtbl[4]; if _i <> 2
 and _i <> 3 then return 0 end if; if _dtbl[_i][4][10] = 1
 and assigned(_dtbl[5-_i]) and _dtbl[_i][4][9] < 100 and 100
 <= _dtbl[5-_i][4][9] then _i := 5-_i; _dtbl[4] := _i; _j :=
 round(_dtbl[_i][4][17]); return round(_dtbl[_i][3][1][_j,
 1]) elif 100 <= _dtbl[_i][4][9] then _j := round(_dtbl[_i][4]\
[17]); return round(_dtbl[_i][3][1][_j, 1]) else return 0 end
 if elif _xin = "eventstatus" then if _nv = 0 then error
 "this solution has no events" end if; _i := [selectremove(pro\
c (a) options operator, arrow; _dtbl[1][3][1][a, 7] = 1 end
 proc, {seq(_j, _j = 1 .. round(_dtbl[1][3][1][_nv+1,
 1]))})]; return ':-enabled' = _i[1], ':-disabled' = _i[2]
 elif _xin = "eventclear" then if _nv = 0 then error "this
 solution has no events" end if; _i := _dtbl[4]; if _i <> 2
 and _i <> 3 then error "no events to clear" end if; if
 _dtbl[_i][4][10] = 1 and assigned(_dtbl[5-_i]) and
 _dtbl[_i][4][9] < 100 and 100 < _dtbl[5-_i][4][9] then
 _dtbl[4] := 5-_i; _i := 5-_i end if; if _dtbl[_i][4][9] <
 100 then error "no events to clear" elif _nv < _dtbl[_i][4][9\
]-100 then error "event error condition cannot be cleared"
 else _j := _dtbl[_i][4][9]-100; if irem(round(_dtbl[_i][3][1]\
```

```
[_j, 4]), 2) = 1 then error "retriggerable events cannot be
 cleared" end if; _j := round(_dtbl[_i][3][1][_j, 1]); for _k
 to _nv do if _dtbl[_i][3][1][_k, 1] = _j then if _dtbl[_i][3]\
[1][_k, 2] = 3 then error "range events cannot be cleared"
 end if; _dtbl[_i][3][1][_k, 8] := _dtbl[_i][3][1][_nv+1, 8]
 end if end do; _dtbl[_i][4][17] := 0; _dtbl[_i][4][9] := 0;
 if _dtbl[1][4][10] = 1 then if _i = 2 then try procname(procn\
ame("left")) catch:  end try else try procname(procname("right\
")) catch:  end try end if end if end if; return  elif
 type(_xin, `=`) and member(lhs(_xin), {"eventdisable",
 "eventenable"}) then if _nv = 0 then error "this solution
 has no events" end if; if type(rhs(_xin), {('list')('posint')\
, ('set')('posint')}) then _i := {op(rhs(_xin))} elif
 type(rhs(_xin), 'posint') then _i := {rhs(_xin)} else error
 "event identifiers must be integers in the range 1..%1",
 round(_dtbl[1][3][1][_nv+1, 1]) end if; if select(proc (a)
 options operator, arrow; _nv < a end proc, _i) <> {} then
 error "event identifiers must be integers in the range
 1..%1", round(_dtbl[1][3][1][_nv+1, 1]) end if; _k := {};
 for _j to _nv do if member(round(_dtbl[1][3][1][_j, 1]), _i)
 then _k := _k union {_j} end if end do; _i := _k; if
 lhs(_xin) = "eventdisable" then _dtbl[4] := 0; _j :=
 [evalb(assigned(_dtbl[2]) and member(_dtbl[2][4][17], _i)),
 evalb(assigned(_dtbl[3]) and member(_dtbl[3][4][17], _i))];
 for _k in _i do _dtbl[1][3][1][_k, 7] := 0; if assigned(_dtbl\
[2]) then _dtbl[2][3][1][_k, 7] := 0 end if; if assigned(_dtbl\
[3]) then _dtbl[3][3][1][_k, 7] := 0 end if end do; if _j[1]
 then for _k to _nv+1 do if _k <= _nv and not type(_dtbl[2][3]\
[4][_k, 1], 'undefined') then userinfo(3, {'events',
```

```
 'eventreset'}, `reinit #2, event code `, _k, ` to defined
 init `, _dtbl[2][3][4][_k, 1]); _dtbl[2][3][1][_k, 8] :=
 _dtbl[2][3][4][_k, 1] elif _dtbl[2][3][1][_k, 2] = 0 and
 irem(iquo(round(_dtbl[2][3][1][_k, 4]), 32), 2) = 1 then
 userinfo(3, {'events', 'eventreset'}, `reinit #2, event code
 `, _k, ` to rate hysteresis init `, _dtbl[2][5][24]);
 _dtbl[2][3][1][_k, 8] := _dtbl[2][5][24] elif _dtbl[2][3][1][\
_k, 2] = 0 and irem(iquo(round(_dtbl[2][3][1][_k, 4]), 2), 2)
 = 0 then userinfo(3, {'events', 'eventreset'}, `reinit #2,
 event code `, _k, ` to initial init `, _x0); _dtbl[2][3][1][_\
k, 8] := _x0 else userinfo(3, {'events', 'eventreset'},
 `reinit #2, event code `, _k, ` to fireinitial init `,
 _x0-1); _dtbl[2][3][1][_k, 8] := _x0-1 end if end do;
 _dtbl[2][4][17] := 0; _dtbl[2][4][9] := 0; if _dtbl[1][4][10]\
 = 1 then procname(procname("left")) end if end if; if _j[2]
 then for _k to _nv+1 do if _k <= _nv and not type(_dtbl[3][3]\
[4][_k, 2], 'undefined') then userinfo(3, {'events',
 'eventreset'}, `reinit #3, event code `, _k, ` to defined
 init `, _dtbl[3][3][4][_k, 2]); _dtbl[3][3][1][_k, 8] :=
 _dtbl[3][3][4][_k, 2] elif _dtbl[3][3][1][_k, 2] = 0 and
 irem(iquo(round(_dtbl[3][3][1][_k, 4]), 32), 2) = 1 then
 userinfo(3, {'events', 'eventreset'}, `reinit #3, event code
 `, _k, ` to rate hysteresis init `, _dtbl[3][5][24]);
 _dtbl[3][3][1][_k, 8] := _dtbl[3][5][24] elif _dtbl[3][3][1][\
_k, 2] = 0 and irem(iquo(round(_dtbl[3][3][1][_k, 4]), 2), 2)
 = 0 then userinfo(3, {'events', 'eventreset'}, `reinit #3,
 event code `, _k, ` to initial init `, _x0); _dtbl[3][3][1][_\
k, 8] := _x0 else userinfo(3, {'events', 'eventreset'},
 `reinit #3, event code `, _k, ` to fireinitial init `,
```

```
_x0+1); _dtbl[3][3][1][_k, 8] := _x0+1 end if end do;
_dtbl[3][4][17] := 0; _dtbl[3][4][9] := 0; if _dtbl[1][4][10]\
= 1 then procname(procname("right")) end if end if else for
_k in _i do _dtbl[1][3][1][_k, 7] := 1 end do; _dtbl[2] :=
evaln(_dtbl[2]); _dtbl[3] := evaln(_dtbl[3]); _dtbl[4] := 0;
if _dtbl[1][4][10] = 1 then if _x0 <= procname("right") then
try procname(procname("right")) catch:  end try end if; if
procname("left") <= _x0 then try procname(procname("left"))
catch:  end try end if end if end if; return  elif
type(_xin, `=`) and lhs(_xin) = "eventfired" then if not
type(rhs(_xin), 'list') then error "'eventfired' must be
specified as a list" end if; if _nv = 0 then error "this
solution has no events" end if; if _dtbl[4] <> 2 and
_dtbl[4] <> 3 then error "'direction' must be set prior to
calling/setting 'eventfired'" end if; _i := _dtbl[4]; _val
:= NULL; if not assigned(_EnvEventRetriggerWarned) then
_EnvEventRetriggerWarned := false end if; for _k in
rhs(_xin) do if type(_k, 'integer') then _src := _k elif
type(_k, 'integer' = 'anything') and type(evalf(rhs(_k)),
'numeric') then _k := lhs(_k) = evalf[max(Digits,
18)](rhs(_k)); _src := lhs(_k) else error "'eventfired'
entry is not valid: %1", _k end if; if _src < 1 or
round(_dtbl[1][3][1][_nv+1, 1]) < _src then error "event
identifiers must be integers in the range 1..%1",
round(_dtbl[1][3][1][_nv+1, 1]) end if; _src := {seq(`if`(_dt\
bl[1][3][1][_j, 1]-_src = 0., _j, NULL), _j = 1 .. _nv)}; if
nops(_src) <> 1 then error "'eventfired' can only be
set/queried for root-finding events and time/interval
events" end if; _src := _src[1]; if _dtbl[1][3][1][_src, 2]
```

```
 <> 0. and _dtbl[1][3][1][_src, 2]-2. <> 0. then error
 "'eventfired' can only be set/queried for root-finding
 events and time/interval events" elif irem(round(_dtbl[1][3][\
1][_src, 4]), 2) = 1 then if _EnvEventRetriggerWarned = false
 then WARNING(`'eventfired' has no effect on events that
 retrigger`) end if; _EnvEventRetriggerWarned := true end if;
 if _dtbl[_i][3][1][_src, 2] = 0 and irem(iquo(round(_dtbl[_i]\
[3][1][_src, 4]), 32), 2) = 1 then _val := _val, undefined
 elif type(_dtbl[_i][3][4][_src, _i-1], 'undefined') or _i =
 2 and _dtbl[2][3][1][_src, 8] < _dtbl[2][3][4][_src, 1] or
 _i = 3 and _dtbl[3][3][4][_src, 2] < _dtbl[3][3][1][_src, 8]
 then _val := _val, _dtbl[_i][3][1][_src, 8] else _val :=
 _val, _dtbl[_i][3][4][_src, _i-1] end if; if type(_k, `=`)
 then if _dtbl[_i][3][1][_src, 2] = 0 and irem(iquo(round(_dtb\
l[_i][3][1][_src, 4]), 32), 2) = 1 then error "cannot set
 event code for a rate hysteresis event" end if; userinfo(3,
 {'events', 'eventreset'}, `manual set event code `, _src, `
 to value `, rhs(_k)); _dtbl[_i][3][1][_src, 8] := rhs(_k);
 _dtbl[_i][3][4][_src, _i-1] := rhs(_k) end if end do; return
 [_val] elif type(_xin, `=`) and lhs(_xin) = "direction" then
 if not member(rhs(_xin), {-1, 1, ':-left', ':-right'}) then
 error "'direction' must be specified as either '1' or
 'right' (positive) or '-1' or 'left' (negative)" end if;
 _src := `if`(_dtbl[4] = 2, -1, `if`(_dtbl[4] = 3, 1,
 undefined)); _i := `if`(member(rhs(_xin), {1, ':-right'}),
 3, 2); _dtbl[4] := _i; _dtbl[_i] := `dsolve/numeric/SC/IVPdco\
py`(_dtbl[1], `if`(assigned(_dtbl[_i]), _dtbl[_i], NULL)); if
 0 < _nv then for _j to _nv+1 do if _j <= _nv and not
 type(_dtbl[_i][3][4][_j, _i-1], 'undefined') then
```

```
 userinfo(3, {'events', 'eventreset'}, `reinit #4, event code
 `, _j, ` to defined init `, _dtbl[_i][3][4][_j, _i-1]);
 _dtbl[_i][3][1][_j, 8] := _dtbl[_i][3][4][_j, _i-1] elif
 _dtbl[_i][3][1][_j, 2] = 0 and irem(iquo(round(_dtbl[_i][3][1\
][_j, 4]), 32), 2) = 1 then userinfo(3, {'events',
 'eventreset'}, `reinit #4, event code `, _j, ` to rate
 hysteresis init `, _dtbl[_i][5][24]); _dtbl[_i][3][1][_j, 8]
 := _dtbl[_i][5][24] elif _dtbl[_i][3][1][_j, 2] = 0 and
 irem(iquo(round(_dtbl[_i][3][1][_j, 4]), 2), 2) = 0 then
 userinfo(3, {'events', 'eventreset'}, `reinit #4, event code
 `, _j, ` to initial init `, _x0); _dtbl[_i][3][1][_j, 8] :=
 _x0 else userinfo(3, {'events', 'eventreset'}, `reinit #4,
 event code `, _j, ` to fireinitial init `, _x0-2*_i+5.0);
 _dtbl[_i][3][1][_j, 8] := _x0-2*_i+5.0 end if end do end if;
 return _src elif _xin = "eventcount" then if _dtbl[1][3][1]
 = 0 or _dtbl[4] <> 2 and _dtbl[4] <> 3 then return 0 else
 return round(_dtbl[_dtbl[4]][3][1][_nv+1, 12]) end if else
 return "procname" end if end if; if _xout = _x0 then return
 [_x0, seq(evalf(_dtbl[1][6][_vmap[_i]]), _i = 1 .. _n-_ne)]
 end if; _i := `if`(_x0 <= _xout, 3, 2); if _xin = "last" and
 0 < _dtbl[_i][4][9] and _dtbl[_i][4][9] < 100 then _dat :=
 eval(_dtbl[_i], 2); _j := _dat[4][20]; return [_dat[11][_j,
 0], seq(_dat[11][_j, _vmap[_i]], _i = 1 .. _n-_ne-_nd),
 seq(_dat[8][1][_vmap[_i]], _i = _n-_ne-_nd+1 .. _n-_ne)] end
 if; if not type(_dtbl[_i], 'array') then _dtbl[_i] :=
 `dsolve/numeric/SC/IVPdcopy`(_dtbl[1], `if`(assigned(_dtbl[_i\
]), _dtbl[_i], NULL)); if 0 < _nv then for _j to _nv+1 do if
 _j <= _nv and not type(_dtbl[_i][3][4][_j, _i-1],
 'undefined') then userinfo(3, {'events', 'eventreset'},
```

```
 `reinit #5, event code `, _j, ` to defined init `,
 _dtbl[_i][3][4][_j, _i-1]); _dtbl[_i][3][1][_j, 8] :=
 _dtbl[_i][3][4][_j, _i-1] elif _dtbl[_i][3][1][_j, 2] = 0
 and irem(iquo(round(_dtbl[_i][3][1][_j, 4]), 32), 2) = 1
 then userinfo(3, {'events', 'eventreset'}, `reinit #5, event
 code `, _j, ` to rate hysteresis init `, _dtbl[_i][5][24]);
 _dtbl[_i][3][1][_j, 8] := _dtbl[_i][5][24] elif _dtbl[_i][3][\
1][_j, 2] = 0 and irem(iquo(round(_dtbl[_i][3][1][_j, 4]),
 2), 2) = 0 then userinfo(3, {'events', 'eventreset'},
 `reinit #5, event code `, _j, ` to initial init `, _x0);
 _dtbl[_i][3][1][_j, 8] := _x0 else userinfo(3, {'events',
 'eventreset'}, `reinit #5, event code `, _j, ` to
 fireinitial init `, _x0-2*_i+5.0); _dtbl[_i][3][1][_j, 8] :=
 _x0-2*_i+5.0 end if end do end if end if; if _xin <> "last"
 then if 0 < 0 then if `dsolve/numeric/checkglobals`(op(_dtbl[\
1][14]), _pars, _n, _y0) then `dsolve/numeric/SC/reinitialize`\
(_dtbl, _y0, _n, procname, _pars, _i) end if end if; if
 _dtbl[1][4][7] = 0 then error "parameters must be
 initialized before solution can be computed" end if end if;
 _dat := eval(_dtbl[_i], 2); _dtbl[4] := _i; try _src :=
 `dsolve/numeric/SC/IVPrun`(_dat, _xout) catch: userinfo(2,
 `dsolve/debug`, print(`Exception in solnproc:`, [lastexceptio\
n][2 .. -1])); error  end try; if _dat[17] <> _dtbl[1][17]
 then _dtbl[1][17] := _dat[17]; _dtbl[1][10] := _dat[10] end
 if; if _src = 0 and 100 < _dat[4][9] then _val :=
 _dat[3][1][_nv+1, 8] else _val := _dat[11][_dat[4][20], 0]
 end if; if _src <> 0 or _dat[4][9] <= 0 then _dtbl[1][5][1]
 := _xout else _dtbl[1][5][1] := _val end if; if _i = 3 and
 _val < _xout then Rounding := -infinity; if _dat[4][9] = 1
```

```
then error "cannot evaluate the solution further right of
%1, probably a singularity", evalf[8](_val) elif _dat[4][9]
= 2 then error "cannot evaluate the solution further right
of %1, maxfun limit exceeded (see ?dsolve,maxfun for
details)", evalf[8](_val) elif _dat[4][9] = 3 then if
_dat[4][25] = 3 then error "cannot evaluate the solution
past the initial point, problem may be initially singular or
improperly set up" else error "cannot evaluate the solution
past the initial point, problem may be complex, initially
singular or improperly set up" end if elif _dat[4][9] = 4
then error "cannot evaluate the solution further right of
%1, accuracy goal cannot be achieved with specified
'minstep'", evalf[8](_val) elif _dat[4][9] = 5 then error
"cannot evaluate the solution further right of %1, too many
step failures, tolerances may be too loose for problem",
evalf[8](_val) elif _dat[4][9] = 6 then error "cannot
evaluate the solution further right of %1, cannot downgrade
delay storage for problems with delay derivative order > 1,
try increasing delaypts", evalf[8](_val) elif _dat[4][9] =
10 then error "cannot evaluate the solution further right of
%1, interrupt requested", evalf[8](_val) elif 100 <
_dat[4][9] then if _dat[4][9]-100 = _nv+1 then error
"constraint projection failure on event at t=%1",
evalf[8](_val) elif _dat[4][9]-100 = _nv+2 then error
"index-1 and derivative evaluation failure on event at
t=%1", evalf[8](_val) elif _dat[4][9]-100 = _nv+3 then error
"maximum number of event iterations reached (%1) at t=%2",
round(_dat[3][1][_nv+1, 3]), evalf[8](_val) else if
_Env_dsolve_nowarnstop <> true then `dsolve/numeric/warning`(\
```

```
StringTools:-FormatMessage("cannot evaluate the solution
 further right of %1, event #%2 triggered a halt",
 evalf[8](_val), round(_dat[3][1][_dat[4][9]-100, 1]))) end
 if; Rounding := 'nearest'; _xout := _val end if else error
 "cannot evaluate the solution further right of %1",
 evalf[8](_val) end if elif _i = 2 and _xout < _val then
 Rounding := infinity; if _dat[4][9] = 1 then error "cannot
 evaluate the solution further left of %1, probably a
 singularity", evalf[8](_val) elif _dat[4][9] = 2 then error
 "cannot evaluate the solution further left of %1, maxfun
 limit exceeded (see ?dsolve,maxfun for details)",
 evalf[8](_val) elif _dat[4][9] = 3 then if _dat[4][25] = 3
 then error "cannot evaluate the solution past the initial
 point, problem may be initially singular or improperly set
 up" else error "cannot evaluate the solution past the
 initial point, problem may be complex, initially singular or
 improperly set up" end if elif _dat[4][9] = 4 then error
 "cannot evaluate the solution further left of %1, accuracy
 goal cannot be achieved with specified 'minstep'",
 evalf[8](_val) elif _dat[4][9] = 5 then error "cannot
 evaluate the solution further left of %1, too many step
 failures, tolerances may be too loose for problem",
 evalf[8](_val) elif _dat[4][9] = 6 then error "cannot
 evaluate the solution further left of %1, cannot downgrade
 delay storage for problems with delay derivative order > 1,
 try increasing delaypts", evalf[8](_val) elif _dat[4][9] =
 10 then error "cannot evaluate the solution further right of
 %1, interrupt requested", evalf[8](_val) elif 100 <
 _dat[4][9] then if _dat[4][9]-100 = _nv+1 then error
```

```
 "constraint projection failure on event at t=%1",
 evalf[8](_val) elif _dat[4][9]-100 = _nv+2 then error
 "index-1 and derivative evaluation failure on event at
 t=%1", evalf[8](_val) elif _dat[4][9]-100 = _nv+3 then error
 "maximum number of event iterations reached (%1) at t=%2",
 round(_dat[3][1][_nv+1, 3]), evalf[8](_val) else if
 _Env_dsolve_nowarnstop <> true then `dsolve/numeric/warning`(\
StringTools:-FormatMessage("cannot evaluate the solution
 further left of %1, event #%2 triggered a halt", evalf[8](_va\
l), round(_dat[3][1][_dat[4][9]-100, 1]))) end if; Rounding
 := 'nearest'; _xout := _val end if else error "cannot
 evaluate the solution further left of %1", evalf[8](_val)
 end if end if; if _EnvInFsolve = true then _dig :=
 _dat[4][26]; if type(_EnvDSNumericSaveDigits, 'posint') then
 _dat[4][26] := _EnvDSNumericSaveDigits else _dat[4][26] :=
 Digits end if; _Env_dsolve_SC_native := true; if _dat[4][25]
 = 1 then _i := 1; _dat[4][25] := 2 else _i := _dat[4][25]
 end if; _val := `dsolve/numeric/SC/IVPval`(_dat, _xout,
 _src); _dat[4][25] := _i; _dat[4][26] := _dig; [_xout,
 seq(_val[_vmap[_i]], _i = 1 .. _n-_ne)] else Digits :=
 _dat[4][26]; _val := `dsolve/numeric/SC/IVPval`(eval(_dat,
 2), _xout, _src); [_xout, seq(_val[_vmap[_i]], _i = 1 ..
 _n-_ne)] end if end proc, (2) = Array(0..0, {}), (3) = [t,
 delta(t), phi(t), x(t), y(t)], (4) = [V = V, l = l, t0 = t0,
 x0 = x0, y0 = y0, phi0 = phi0, delta0 = delta0]}); _vars :=
 _dat[3]; _pars := map(rhs, _dat[4]); _n := nops(_vars)-1;
 _solnproc := _dat[1]; if not type(_xout, 'numeric') then if
 member(x_rkf45_dae, ["start", 'start', "method", 'method',
 "left", 'left', "right", 'right', "leftdata", "rightdata",
```

```
 "enginedata", "eventstop", 'eventstop', "eventclear",
 'eventclear', "eventstatus", 'eventstatus', "eventcount",
 'eventcount', "laxtol", 'laxtol', "numfun", 'numfun', NULL])
 then _res := _solnproc(convert(x_rkf45_dae, 'string')); if 1
 < nops([_res]) then return _res elif type(_res, 'array')
 then return eval(_res, 1) elif _res <> "procname" then
 return _res end if elif member(x_rkf45_dae, ["last", 'last',
 "initial", 'initial', "parameters", 'parameters',
 "initial_and_parameters", 'initial_and_parameters', NULL])
 then _xout := convert(x_rkf45_dae, 'string'); _res :=
 _solnproc(_xout); if _xout = "parameters" then return
 [seq(_pars[_i] = _res[_i], _i = 1 .. nops(_pars))] elif
 _xout = "initial_and_parameters" then return [seq(_vars[_i+1]\
 = [_res][1][_i+1], _i = 0 .. _n), seq(_pars[_i] =
 [_res][2][_i], _i = 1 .. nops(_pars))] else return
 [seq(_vars[_i+1] = _res[_i+1], _i = 0 .. _n)] end if elif
 type(_xout, `=`) and member(lhs(_xout), ["initial",
 'initial', "parameters", 'parameters', "initial_and_parameter\
s", 'initial_and_parameters', NULL]) then _xout :=
 convert(lhs(x_rkf45_dae), 'string') = rhs(x_rkf45_dae); if
 type(rhs(_xout), 'list') then _res := _solnproc(_xout) else
 error "initial and/or parameter values must be specified in
 a list" end if; if lhs(_xout) = "initial" then return
 [seq(_vars[_i+1] = _res[_i+1], _i = 0 .. _n)] elif
 lhs(_xout) = "parameters" then return [seq(_pars[_i] =
 _res[_i], _i = 1 .. nops(_pars))] else return [seq(_vars[_i+1\
] = [_res][1][_i+1], _i = 0 .. _n), seq(_pars[_i] =
 [_res][2][_i], _i = 1 .. nops(_pars))] end if elif
 type(_xout, `=`) and member(lhs(_xout), ["eventdisable",
```

```
   'eventdisable', "eventenable", 'eventenable', "eventfired",

   'eventfired', "direction", 'direction', NULL]) then return

   _solnproc(convert(lhs(x_rkf45_dae), 'string') = rhs(x_rkf45_d\

  ae)) elif _xout = "solnprocedure" then return eval(_solnproc)

   elif _xout = "sysvars" then return _vars end if; if procname

   <> unknown then return ('procname')(x_rkf45_dae) else _ndsol

   := 1; _ndsol := _ndsol; _ndsol := pointto(_dat[2][0]);

   return ('_ndsol')(x_rkf45_dae) end if end if; try _res :=

   _solnproc(_xout); [seq(_vars[_i+1] = _res[_i+1], _i = 0 ..

   _n)] catch: error  end try end proc
    [V = 1.3, l = 5., t0 = 0., x0 = 0., y0 = 0., phi0 = 0.,

      delta0 = 0.]
     [t = 90., delta(t) = 0.900000000000000,

       phi(t) = 9.50884887164759, x(t) = 26.0589315281754,

       y(t) = 30.5554882284003]
Warning, cannot evaluate the solution further right of 157.07963,
probably a singularity

Warning, cannot evaluate the solution further right of 157.07963,
probably a singularity


;
>
;
```