TR_1D_model1_SS\TR_1D_model1_func_calc_b_int.m

```
% TR_1D_model1_SS\TR_1D_model1_func_calc_b_int.m
%
% function [b_int,bJac_int,iflag] =  ...
%    TR_1D_model1_calc_b_int(x_state,epsilon,Param);
%
% This function calculates the b vector and its
% Jacobian bJac of a DAE system of the form :
%
%   epsilon(k) * df_dt(k) = b(k) -
%          sum_{j} {A(k,j)*x_state(j)}
%
% for a reaction system that is comprised of an
% arbitrary number of species and an arbitrary number
% of reactions.  The vector epsilon is used as an
% integer mask to tell the solver where to calculate
% the elements of b and bJac from the reaction model
% for the interior points.  The reaction system is
% comprised of an arbitrary number of reactions with
% specified stoichiometric coefficients and exponential
% rate law powers for each species.  This procedure
% goes through each interior grid point and passes the
% concentration and temperature values at this point
% to a general subroutine that evaluates the source
% term and its derivatives at this point.  Then, a
% shifting routine takes this data and places it in the
% appropriate spots in the b vector and bJac matrix.
% Since the reaction rate depends on the values of the
% concentration and temperature fields at a single
% point, this routine may  be used in any geometry.
%
% INPUT :
% =======
% x_state           REAL(num_DOF) where
%                    num_DOF = Grid.num_pts *
%               (ProbDim.num_species + 1)
%                    This is the column vector of the state
%                    variables containing the concentration
%                    and temperature profiles.
% epsilon             INT(num_DOF)
%                    This is a vector of integers that has
%                    1's at the interior point equations
%                    and 0's at the boundary point
%                    equations.
% Param                This data structure contains the
%                    system parameters stored in the
%                    fields :
%                    .ProbDim
%                    .Reactor
%                    .Rxn
%                    .Physical
%                    .Grid
%                    see TR_1D_model1_SS.m for details of
%                    these data structures
%
% OUTPUT :
% ========
% b_int                REAL(num_DOF)
%                    This column vector contains the source
%                    terms at the interior points of the
%                    concentration and temperature PDE's
% bJac_int             REAL(num_DOF,num_DOF)
%                    This is the Jacobian element of b_int
%                    whose (m,n) element is defined as the
%                    partial derivative of b_int(m) with
%                    respect to x_state(n).
%
% Kenneth Beers
% Massachusetts Institute of Technology
% Department of Chemical Engineering
% 7/2/2001
%
% Version as of 7/25/2001



function [b_int,bJac_int,iflag] =  ...
   TR_1D_model1_calc_b_int(x_state,epsilon,Param);

iflag = 0;

func_name = 'TR_1D_model1_calc_b_int';

% This sets what to do in case an assertion fails.
i_error = 2;


% First, check input.

% First, extact parameter structures.

if(~isstruct(Param))
   iflag = -1;
   message = [func_name, ': ', ...
         'Paramis not a structure'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      b_int = 0; bJac_int=0;
      return;
   end
end


% ProbDim
if(~isfield(Param,'ProbDim'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain ProbDim'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      b_int=0; bJac_int=0;
      return;
   end
end
ProbDim = Param.ProbDim;

% Reactor
if(~isfield(Param,'Reactor'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain Reactor'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      b_int=0; bJac_int=0;
      return;
   end
end
Reactor = Param.Reactor;

% Rxn
if(~isfield(Param,'Rxn'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain Rxn'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end
Rxn = Param.Rxn;

% Physical
if(~isfield(Param,'Physical'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain Physical'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end
Physical = Param.Physical;

% Grid
if(~isfield(Param,'Grid'))
   iflag = -1;
   message = [func_name, ': ', ...
         'Param does not contain Grid'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end
Grid = Param.Grid;



% Extract the proper dimension of x_state and epsilon.

num_DOF = (ProbDim.num_species+1)*Grid.num_pts;


% check x_state and epsilon to be vectors of proper type

% x_state
dim = num_DOF; check_column = 1;
check_real = 1; check_sign = 0; check_int = 0;
```

```
assert_vector(i_error,x_state,'x_state',func_name, ...
   dim,check_real,check_sign,check_int,check_column);

% epsilon
dim = num_DOF; check_column = 1;
check_real = 1; check_sign = 2; check_int = 1;
assert_vector(i_error,epsilon,'epsilon', func_name, ...
   dim,check_real,check_sign,check_int,check_column);


%PDL> Initialize b, bJac to zeros

b_int = linspace(0,0,num_DOF)';

max_nonzero = (ProbDim.num_species+1)*Grid.num_pts;
bJac_int = spalloc(num_DOF,num_DOF,max_nonzero);


%PROCEDURE: unstack_state
%PDL> Unstack x_state to state_data variable format
%ENDPROCEDURE

[State,iflag_func] = unstack_state(x_state, ...
   ProbDim.num_species,Grid.num_pts);
if(iflag_func <= 0)
   iflag = -2;
   message = [func_name, ': ', ...
         'Error (', int2str(iflag_func), ') ', ...
         'returned from unstack_state'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end


%PDL> FOR EVERY interior grid point ipoint
%       FROM 1 TO Grid.num_pts
%       WHERE epsilon(ipoint) is non-zero

for ipoint = 1:Grid.num_pts
if(epsilon(ipoint) ~= 0)


%           PROCEDURE: reaction_nework_model
%         PDL> Pass the concentrations and temperature
%       at this point along with the density and
%       heat_capacity to a general routine that
%       implements the general reaction network model
%       and returns the reaction rates, the partial
%       derivatives of the rates with respect to each
%       concentration and temperature.
%           ENDPROCEDURE

   conc_loc = State.conc(ipoint,:);
   Temp_loc = State.Temp(ipoint);


   % the local reaction is characterized by the data
   % in a structure RxnRate that contains the following
   % fields :
   % .rxn_time_deriv_c - the total time derivative of each
   %                  concentration due to all reactions
   % .rxn_time_deriv_T - the total time derivative of the
   %                  the temperature due to all reactions
   % .rxn_rate - the rates of each reaction
   % .rxn_rate_deriv_c - the derivative of each reaction rate
   %                  with respect to each concentration
   % .rxn_rate_deriv_T - the derivative of each reaction rate
   %                  with respect to the temperature

   [RxnRate_loc, iflag_func] = ...
      reaction_network_model(...
      ProbDim.num_species,ProbDim.num_rxn, ...
      conc_loc,Temp_loc,Rxn,Physical.density,Physical.Cp);
   if(iflag_func <= 0)
      iflag = -3;
      message = [func_name, ': ', ...
            'Error (', int2str(iflag_func), ') ', ...
            'returned from reaction_network_model'];
      if(i_error ~= 0)
         if(i_error > 1)
            save dump_error.mat;
         end
         error(message);
      else
         return;
      end
   end


%           PROCEDURE: shift_rxn_source_term
%         PDL> Next, b and bJac are incremented by a
%       procedure that takes as input the grid point
%       and the reaction rates and their derivatives,
%       and adds these results to the appropriate b
%       and bJac locations.
%           ENDPROCEDURE
```

```
   [b_loc,bJac_loc,iflag_func] = shift_rxn_source_term(...
      ProbDim,Grid,Rxn,Physical,ipoint,RxnRate_loc);
   if(iflag_func <= 0)
      iflag = -4;
      message = [func_name, ': ', ...
            'Error (', int2str(iflag_func), ') ', ...
            'returned from shift_rxn_source_term'];
      if(i_error ~= 0)
         if(i_error > 1)
            save dump_error.mat;
         end
         error(message);
      else
         return;
      end
   end

   b_int = b_int + b_loc;
   bJac_int = bJac_int + bJac_loc;


%PDL> ENDFOR for loop over interior points

end
end


%PDL> Add contribution from heat transfer to the
%   jacket

[b_HT,bJac_HT,iflag_func] = jacket_heat_transfer( ...
   State,epsilon,ProbDim,Reactor,Physical,Grid);
if(iflag_func <= 0)
   iflag = -5;
   message = [func_name, ': ', ...
         'Error (', int2str(iflag_func), ') ', ...
         'returned from jacket_heat_transfer'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end

b_int = b_int + b_HT;
bJac_int = bJac_int + bJac_HT;
```

```
iflag = 1;

return;
```