TR_1D_model1_SS\shift_rxn_source_term.m

```
% TR_1D_model1_SS\shift_rxn_source_term.m
%
% function [b_loc,bJac_loc,iflag] = ...
%    shift_rxn_source_term(ProbDim, ...
%    Grid,Rxn,Physical,ipoint,RxnRate);
%
% This procedure takes the reaction data calculated
% by the geometry-independent reaction network model
% routine and shifts the results to the appropriate
% locations for the DAE form of a set of PDE's that
% model the concentration and enthalpy balances using
% finite differences.  This routine may be used with
% any geometry as long as the concentrations and
% temperature are stacked into the master state
% vector with the same ordering.
%
% INPUT :
% =======
% ProbDim              This data structure contains the
%                        fields .num_species and .num_rxn
%                        that give the total number of
%                        species and reactions respectively
%                        in the system.
% Grid                 This data structure contains the
%                        field .num_pts that specifies the
%                        total number of grid points.
% ipoint               This is the number of the grid
%                        point at which the reaction rate
%                        terms have been calculated from a
%                        local reaction model.
% Rxn                  This data structure contains the
%                        kinetic data for the reaction
%                        network.
%   .stoich_coeff        REAL(num_rxn,num_species)
%                          the stoichiometric coefficients
%                        (possibly fractional) of each
%                          species in each reaction.
% Physical             This data structure constains the
%                        physical data for the system.
%   .density             REAL
%                        the constant density of the medium
%   .Cp                      REAL
%                        the constant heat capacity of the medium
% RxnRate              data structure containing the following fields :
% .time_deriv_c          REAL(num_species)
%                   this is a column vector of the time derivatives of the
%                   concentration due to all reactions
% .time_deriv_T          REAL
%                  this is the time derivative of the temperature due to
%                    the effect of all the reactions
% .rate                  REAL(num_rxn)
%                    this is a column vector of the rates of each reaction
% .rate_deriv_c          REAL(num_rxn,num_species)
%                    this is a matrix of the partial derivatives of each reaction
%                    rate with respect to the concentrations of each species
% .rate_deriv_T          REAL(num_rxn)
%                    this is a column vector of the partial derivatives of each
%                  reaction rate with respect to the temperature
% k                      REAL(num_rxn)
%                 this is a column vector of the rate constant values at the
%                 current temperature
% .source_term      REAL(num_rxn)
%                 this is a column vector of the values in the rate law expression
%                 that are dependent on concentration.  For example, in the rate law :
%                 R = k*[A]*[B]^2, the source term value is [A]*[B]^2.

% OUTPUT :
% ========
% b_loc              REAL(num_DOF) where
%                 num_DOF = (ProbDim.num_species+1)*Grid.num_pts
%                 This is a column vector of the contribution to
%                 the DAE system vector b from the local reaction
%                 at grid point # ipoint.
% bJac_loc           REAL(num_DOF,num_DOF)
%                 This is the contribution from location reaction at
%                 ipoint to the Jacobian of b.
% iflag              INT
%                   integer flag that has 0 signifying no
%                   completion, a negative value signifying
%                   an exit from an error, and a value of
%                   1 signifying success.
%
% Kenneth Beers
% Massachusetts Institute of Technology
% Department of Chemical Engineering
% 7/2/2001
%
% Version as of 7/24/2001



function [b_loc,bJac_loc,iflag] = ...
   shift_rxn_source_term(ProbDim, ...
   Grid,Rxn,Physical,ipoint,RxnRate);

iflag = 0;

func_name = 'shift_rxn_source_term';
```

```
% This integer flag controls what level of action
% to take in the case of an assertion or called
% routine failure.
i_error = 2;


% check input

%ProbDim
ProbDimType.num_fields=2;
% .num_species
ifield = 1;
FieldType.name = 'num_species';
FieldType.is_numeric = 1;
FieldType.num_rows = 1;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 1;
FieldType.check_int = 1;
ProbDimType.field(ifield) = FieldType;
% .num_rxn
ifield = 2;
FieldType.name = 'num_rxn';
FieldType.is_numeric = 1;
FieldType.num_rows = 1;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 1;
FieldType.check_int = 1;
ProbDimType.field(ifield) = FieldType;
% perform assertion
assert_structure(i_error,ProbDim,'ProbDim', ...
   func_name,ProbDimType);

% Grid
GridType.num_fields = 1;
% .num_pts
ifield = 1;
FieldType.name = 'num_pts';
FieldType.is_numeric = 1;
FieldType.num_rows = 1;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 1;
FieldType.check_int = 1;
GridType.field(ifield) = FieldType;
% perform assertion
assert_structure(i_error,Grid,'Grid', ...
   func_name,GridType);
```

```
% Rxn
RxnType.num_fields = 1;
% .stoich_coeff
ifield = 1;
FieldType.name = 'stoich_coeff';
FieldType.is_numeric = 1;
FieldType.num_rows = ProbDim.num_rxn;
FieldType.num_columns = ProbDim.num_species;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnType.field(ifield) = FieldType;
% perform assertion
assert_structure(i_error,Rxn,'Rxn', ...
   func_name,RxnType);

% Physical
PhysicalType.num_fields = 2;
% .density
ifield = 1;
FieldType.name = 'density';
FieldType.is_numeric = 1;
FieldType.num_rows = 1;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 1;
FieldType.check_int = 0;
PhysicalType.field(ifield) = FieldType;
% .Cp
ifield = 2;
FieldType.name = 'Cp';
FieldType.is_numeric = 1;
FieldType.num_rows = 1;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 1;
FieldType.check_int = 0;
PhysicalType.field(ifield) = FieldType;
% perform assertion
assert_structure(i_error,Physical,'Physical', ...
   func_name,PhysicalType);

% ipoint
check_real=1; check_sign=1; check_int=1;
assert_scalar(i_error,ipoint,'ipoint', ...
   func_name,check_real,check_sign,check_int);
if(ipoint > Grid.num_pts)
   iflag = -1;
   message = [ func_name, ': ', ...
         'Input ipoint > Grid.num_pts'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
   end
end

% RxnRate
RxnRateType.num_fields = 7;
% .time_deriv_c
ifield = 1;
FieldType.name = 'time_deriv_c';
FieldType.is_numeric = 1;
FieldType.num_rows = ProbDim.num_species;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnRateType.field(ifield) = FieldType;
% .time_deriv_T
ifield = 2;
FieldType.name = 'time_deriv_T';
FieldType.is_numeric = 1;
FieldType.num_rows = 1;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnRateType.field(ifield) = FieldType;
% .rate
ifield = 3;
FieldType.name = 'rate';
FieldType.is_numeric = 1;
FieldType.num_rows = ProbDim.num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnRateType.field(ifield) = FieldType;
% .rate_deriv_c
ifield = 4;
FieldType.name = 'rate_deriv_c';
FieldType.is_numeric = 1;
FieldType.num_rows = ProbDim.num_rxn;
FieldType.num_columns = ProbDim.num_species;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
```

```
RxnRateType.field(ifield) = FieldType;
% .rate_deriv_T
ifield = 5;
FieldType.name = 'rate_deriv_T';
FieldType.is_numeric = 1;
FieldType.num_rows = ProbDim.num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnRateType.field(ifield) = FieldType;
% .k
ifield = 6;
FieldType.name = 'k';
FieldType.is_numeric = 1;
FieldType.num_rows = ProbDim.num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 2;
FieldType.check_int = 0;
RxnRateType.field(ifield) = FieldType;
% .source_term
ifield = 7;
FieldType.name = 'source_term';
FieldType.is_numeric = 1;
FieldType.num_rows = ProbDim.num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnRateType.field(ifield) = FieldType;
% perform assertion
assert_structure(i_error,RxnRate,'RxnRate', ...
    func_name,RxnRateType);



% allocate b_loc, bJac_loc and initialize to zeros

num_fields = ProbDim.num_species + 1;

num_DOF = num_fields*Grid.num_pts;

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

max_nonzero = num_DOF*(ProbDim.num_species+1);
bJac_loc = spalloc(num_DOF,num_DOF,max_nonzero);



%PDL> Update the b values for each concentration
```

```
%       FOR ispecies FROM 1 TO ProbDim.num_species

for ispecies = 1:ProbDim.num_species


%       PDL> Set pos_offset = (ispecies-1)*Grid.num_pts
%   Set integer offset for this concentration field location
%   in the master state array.

   pos_offset= (ispecies-1)*Grid.num_pts;


%       PDL> b(pos_offset+ipoint) = rxn_time_deriv_c(ispecies)
%   The b vector for this concentration field at this point
%   is the total time derivative of the concentration due to
%   all reactions.

   iDOF = pos_offset + ipoint;
   b_loc(iDOF) = RxnRate.time_deriv_c(ispecies);


%PDL> ENDFOR

end


%PDL> Update the b value for the temperature
%       PDL> Set pos_offset =
%            ProbDim.num_species*Grid.num_pts
%       Set integer offset to beginning of temperature field
%        in the master state vector.

pos_offset = ProbDim.num_species*Grid.num_pts;


%       PDL> b(pos_offset+kpoint) = rxn_time_deriv_T
%   The b vector element is set to the time derivative of
%   the temprature at that point due to local reaction.

iDOF = pos_offset + ipoint;
b_loc(iDOF) = RxnRate.time_deriv_T;


%PDL> Update the bJac values for each species balance
%       FOR ispecies FROM 1 TO ProbDim.num_species

for ispecies = 1:ProbDim.num_species


%       PDL> Set ipos_offset = (ispecies-1)*Grid.num_pts
%   Set integer offset to the start of the concentration
%   field for species # ispecies.

    ipos_offset = (ispecies-1)*Grid.num_pts;


%         PDL> bJac(ipos_offset+kpoint,:) = 0
%   (ALREADY DONE)

%         PDL> Get Jacobian values for concentration derivatives
%   FOR jspecies FROM 1 TO ProbDim.num_species

    for jspecies = 1:ProbDim.num_species


%   PDL> Set jpos_offset = (jspecies-1)*Grid.num_pts
%   Set integer offset to the start of the concentration field
%   for species # jspecies.

        jpos_offset = (jspecies-1)*Grid.num_pts;


%PDL> FOR every reaction, get contribution to the Jacobian value
%         FOR irxn FROM 1 TO ProbDim.num_rxn

        for irxn = 1:ProbDim.num_rxn


%   PDL> Increment bJac(ipos_offset+ipoint,jpos_offset+ipoint)
%                       by the product of
%           Rxn.stoich_coeff(irxn,ispecies) and
%           RxnRate.rxn_rate_deriv_c(irxn,jspecies)

            update_value = Rxn.stoich_coeff(irxn,ispecies) * ...
                RxnRate.rate_deriv_c(irxn,jspecies);
            iDOF_row = ipos_offset+ipoint;
            iDOF_col = jpos_offset+ipoint;
            bJac_loc(iDOF_row,iDOF_col) = ...
                bJac_loc(iDOF_row,iDOF_col) + ...
                    update_value;


%   PDL> ENDFOR

        end


%         PDL> ENDFOR

    end
```

```
%       PDL> Get Jacobian value for temperature derivative

%       PDL> Set Tpos_offset =
%               ProbDim.num_species*Grid.num_pts
%       Set integer offset for start of the temperature field.

   Tpos_offset = ProbDim.num_species*Grid.num_pts;


%       PDL> FOR every reaction, get contribution to the Jacobian value
%               FOR irxn FROM 1 TO ProbDim.num_rxn

   for irxn = 1:ProbDim.num_rxn


%               PDL> Increment bJac(ipos_offset+ipoint,Tpos_offset+ipoint)
%                       by the product of
%                       Rxn.stoich_coeff(irxn,ispecies) and
%                       RxnRate.rxn_rate_deriv_T(irxn)

      update_value = Rxn.stoich_coeff(irxn,ispecies) * ...
         RxnRate.rate_deriv_T(irxn);
      iDOF_row = ipos_offset + ipoint;
      iDOF_col = Tpos_offset + ipoint;
      bJac_loc(iDOF_row,iDOF_col) = ...
         bJac_loc(iDOF_row,iDOF_col) + ...
            update_value;

%       PDL> ENDFOR

   end

%PDL> ENDFOR

end


%PDL> Update the Jacobian values for the enthalpy balance

%       PDL> Set Tpos_offset =
%         ProbDim.num_species*Grid.num_pts

Tpos_offset = ProbDim.num_species*Grid.num_pts;


%       PDL> Set bJac(Tpos_offset+ipoint,:) = 0
%   (ALREADY DONE)

%       PDL> Get Jacobian values for concentration derivatives
%   FOR jspecies FROM 1 TO ProbDim.num_species
```

```
for jspecies = 1:ProbDim.num_species


%   PDL> Set jpos_offset = (jspecies-1)*Grid.num_pts

   jpos_offset = (jspecies-1)*Grid.num_pts;


%   PDL> FOR every reaction, get contribution to the Jacobian value
%        FOR irxn FROM 1 TO ProbDim.num_rxn

   for irxn = 1:ProbDim.num_rxn


%        PDL> Increment bJac(Tpos_offset+ipoint,jpos_offset+ipoint)
%                 by the product of
%                 (-Rxn.delta_H(irxn) / Physical.density /
%                 Physical.Cp) and
%                 rxn_rate_deriv_c(irxn,jspecies)

      update_value = ...
         (-Rxn.delta_H(irxn) / ...
            Physical.density / Physical.Cp) ...
         * RxnRate.rate_deriv_c(irxn,jspecies);
      iDOF_row = Tpos_offset + ipoint;
      iDOF_col = jpos_offset + ipoint;
      bJac_loc(iDOF_row,iDOF_col) = ...
         bJac_loc(iDOF_row,iDOF_col) + ...
            update_value;


%                 PDL> ENDFOR

   end


%        PDL> ENDFOR

end


%        PDL> Get Jacobian values for the temperature derivative

%        PDL> FOR every reaction, get contribution to the Jacobian value
%                 FOR irxn FROM 1 TO ProbDim.num_rxn

for irxn = 1:ProbDim.num_rxn


%                 PDL> Increment bJac(Tpos_offset+ipoint,Tpos_offset+ipoint)
%                          by the product of
%                              (-Rxn.delta_H(irxn) / Physical.density /
%                                        Physical.Cp) and
%                              rxn_rate_deriv_T(irxn)

      update_value = ...
         (-Rxn.delta_H(irxn) / ...
            Physical.density / Physical.Cp) ...
         * RxnRate.rate_deriv_T(irxn);
      iDOF_row = Tpos_offset + ipoint;
      iDOF_col = Tpos_offset + ipoint;
      bJac_loc(iDOF_row,iDOF_col) = ...
         bJac_loc(iDOF_row,iDOF_col) + ...
            update_value;


%         PDL> ENDFOR

end


iflag = 1;

return;
```