% function [RxnRate, iflag] = reaction_network_model(num_species, num_rxn, ... conc, Temp, Rxn, density, Cp);

% This procedure evaluates the rates of each reaction and the derivatives of the rates with respect to the concentrations and temperature for a general reaction network. The rate laws are characterized by the product of each concentration raised to an exponential power. The rate constants are temperature dependent, according to an Arrhenius expression based on an activation energy and the value of the rate constant at a specified reference temperature. Also, the contributions to the time derivatives of the concentrations and the temperature due to the total effect of reaction are returned.

% INPUT:
% ========
% num_species INT
% The number of species
% num_rxn INT
% The number of reactions
% conc REAL(num_species)
% This is a column vector of the concentrations of each species at a single point
% Temp REAL
% This is the temperature at a single point
% Rxn This structure contains the kinetic data for the general reaction network. The fields are:
% .stoich_coeff REAL(num_rxn,num_species)
% the stoichiometric coefficients possibly fractional) of each species in each reaction.
% .ratelaw_exp REAL(num_rxn,num_species)
% the exponential power (possibly fractional) to which the concentration of each species is raised each reaction's rate law.
% .is_rxn_elementary INT(num_rxn)
% if a reaction is elementary, then the rate law exponents are zero for the product species and the negative of the stoichiometric coefficient for the
% reactant species. In this case, we need not enter the corresponding components of ratelaw_exp since these are determined by the corresponding values in stoich_coeff. We specify that reaction number irxn is elementary by setting
% is_rxn_elementary(irxn) = 1.
% Otherwise (default = 0), we assume that the reaction is not elementary and require the user to input the values of ratelaw_exp for reaction # irxn.

% .k_ref REAL(num_rxn)
% the rate constants of each reaction at a specified reference temperature
% .T_ref REAL(num_rxn)
% This is the value of the reference temperature used to specify the temperature dependence of each rate constant.
% .E_activ REAL(num_rxn)
% the constant activation energies of each reaction divided by the ideal gas constant
% .delta_H REAL(num_rxn)
% the constant heats of reaction
%
% density REAL
% the density of the medium
% Cp REAL
% the heat capacity of the medium
%
% OUTPUT :
% ========
% 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 \([A][B]^2\).
%
% Kenneth Beers
% Massachusetts Institute of Technology
% Department of Chemical Engineering
% 7/2/2001
%
% Version as of 7/25/2001

function [RxnRate, iflag] = ...
reaction_network_model(num_species,num_rxn, ...
conc_loc,Temp_loc,Rxn,density,Cp);

iflag = 0;

% this integer flag controls the action taken
% when an assertion fails. See the assertion
% routines for a description of its use.
i_error = 1;

func_name = 'reaction_network_model';

% Check input

% num_species
check_real=1; check_sign=1; check_int=1;
assert_scalar(i_error,num_species,'num_species', ...
    func_name,check_real,check_sign,check_int);

% num_rxn
check_real=1; check_sign=1; check_int=1;
assert_scalar(i_error,num_rxn,'num_rxn', ...
    func_name,check_real,check_sign,check_int);

% conc_loc
dim = num_species; check_column=0;
check_real=1; check_sign=0; check_int=0;
assert_vector(i_error,conc_loc,'conc_loc', ...
    func_name,dim,check_real,check_sign, ...
    check_int,check_column);
% now, make sure all concentrations are non-negative
list_neg = find(conc_loc < 0);
for count=1:length(list_neg)
    ispecies = list_neg(count);
    conc_loc(ispecies) = 0;
end

% Temp_loc
check_real=1; check_sign=0; check_int=0;
assert_scalar(i_error,Temp_loc,'Temp_loc', ...
    func_name,check_real,check_sign,check_int);
% make sure the temperature is positive
trace = 1e-20;
if(Temp_loc <= trace)
    Temp_loc = trace;
end

% Rxn
RxnType.struct_name = 'Rxn';
RxnType.num_fields = 7;
% Now set the assertion properties of each field.
% .stoich_coeff
ifield = 1;
FieldType.name = 'stoich_coeff';
FieldType.is_numeric = 1;
FieldType.num_rows = num_rxn;
FieldType.num_columns = num_species;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnType.field(ifield) = FieldType;
% .ratelaw_exp
ifield = 2;
FieldType.name = 'ratelaw_exp';
FieldType.is_numeric = 2;
FieldType.num_rows = num_rxn;
FieldType.num_columns = num_species;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnType.field(ifield) = FieldType;
% .is_rxn_elementary
ifield = 3;
FieldType.name = 'is_rxn_elementary';
FieldType.is_numeric = 1;
FieldType.num_rows = num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 2;
FieldType.check_int = 1;
RxnType.field(ifield) = FieldType;
% .k_ref
ifield = 4;
FieldType.name = 'k_ref';
FieldType.is_numeric = 1;
FieldType.num_rows = num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 2;
FieldType.check_int = 0;
RxnType.field(ifield) = FieldType;

ifo1d  = 5;
FieldType.name = 'T_ref';
FieldType.is_numeric = 1;
FieldType.num_rows = num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 1;
FieldType.check_int = 0;
RxnType.field(ifield) = FieldType;

ifo1d  = 6;
FieldType.name = 'E_activ';
FieldType.is_numeric = 1;
FieldType.num_rows = num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 2;
FieldType.check_int = 0;
RxnType.field(ifield) = FieldType;

ifo1d  = 7;
FieldType.name = 'delta_H';
FieldType.is_numeric = 1;
FieldType.num_rows = num_rxn;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
RxnType.field(ifield) = FieldType;

% call assertion routine for structure
assert_structure(i_error,Rxn,'Rxn',func_name,RxnType);

% density
check_real=1; check_sign=1; check_int=0;
assert_scalar(i_error,density,'density', ...     
     func_name,check_real,check_sign,check_int);

% heat capacity
check_real=1; check_sign=1; check_int=0;
assert_scalar(i_error,Cp,'Cp', ...     
     func_name,check_real,check_sign,check_int);
%PDL> Initialize all output variables to zeros

```matlab
RxnRate.time_deriv_c = linspace(0,0,num_species)';
RxnRate.time_deriv_T = 0;
RxnRate.rate = linspace(0,0,num_rxn)';
RxnRate.rate_deriv_c = zeros(num_rxn,num_species);
RxnRate.rate_deriv_T = linspace(0,0,num_rxn)';
RxnRate.k = linspace(0,0,num_rxn)';
RxnRate.source_term = linspace(0,0,num_rxn)';
```

%PDL> For every reaction, calculate the rates and their derivatives with respect to the concentrations and temperatures
% FOR irxn FROM 1 TO num_rxn

```matlab
for irxn = 1:num_rxn

%PDL> Calculate rate constant at the current temperature
```

```matlab
factor_T = exp(-Rxn.E_activ(irxn) * ...
    (1/Temp_loc - 1/Rxn.T_ref(irxn)));
RxnRate.k(irxn) = Rxn.k_ref(irxn)*factor_T;
```

%PDL> Calculate the derivative of the rate constant with respect to temperature

```matlab
d_rate_k_d_Temp = RxnRate.k(irxn) * ...
    Rxn.E_activ(irxn)/(Temp_loc^2);
```

%PDL> Set ratelaw_vector to be of length num_species whose elements are the concentrations of each species raised to the power ratelaw_exp(irxn,ispecies). If the exponent is 0, automatically set corresponding element to 1.

```matlab
ratelaw_vector = linspace(1,1,num_species)';
list_species = find(Rxn.ratelaw_exp(irxn,:) ~= 0);
for count=1:length(list_species)
    ispecies = list_species(count);
    ratelaw_vector(ispecies) = ...
        conc_loc(ispecies) ^ Rxn.ratelaw_exp(irxn,ispecies);
end
```

%PDL> Calculate the ratelaw source term that is the product of all elements of ratelaw_vector
RxnRate.source_term(irxn) = prod(ratelaw_vector);

%PDL> The rate of reaction # irxn is equal to the product of 
% the ratelaw source term with the value of the rate constant

RxnRate.rate(irxn) = RxnRate.k(irxn) * ... 
RxnRate.source_term(irxn);

%PDL> Set rxn_rate_deriv_T(irxn) to be equal to the product of 
% the temperature derivative of the rate constant times the 
% ratelaw source term

RxnRate.rate_deriv_T(irxn) = ... 
d_rate_k_d_Temp * RxnRate.source_term(irxn);

%PDL> FOR EVERY ispecies WHERE
% ratelaw_exp(irxn,ispecies) IS non-zero

for count=1:length(list_species)
    ispecies = list_species(count);

%PDL> Set vector_work = ratelaw_vector and replace the 
% ispecies element with 
% ratelaw_exp(irxn,ispecies)* 
% conc(ispecies)^(ratelaw_exp(irxn,ispecies)-1) 
% If ratelaw_exp(irxn,ispecies) is exactly 1, then do 
% special case where replace element with 1

vector_work = ratelaw_vector;
if(Rxn.ratelaw_exp(irxn,ispecies) == 1)
    vector_work(ispecies) = 1;
else
    exponent = Rxn.ratelaw_exp(irxn,ispecies);
    vector_work(ispecies) = exponent * ...
        (conc_loc(ispecies) ^ (exponent-1));
end

% PDL> Set rxn_rate_deriv_c(irxn,ispecies) equal to the 
% product of all components of this vector 
% multiplied by the rate constant

RxnRate.rate_deriv_c(irxn,ispecies) = ... 
RxnRate.k(irxn) * prod(vector_work);
% PDL> ENDFOR for sum over participating species

end

% PDL> FOR EVERY ispecies WHERE
% Rxn.stoich_coeff(irxn,ispecies) IS non-zero

list_species = find(Rxn.stoich_coeff(irxn,:) ~= 0);
for count=1:length(list_species)
    ispecies = list_species(count);

% PDL> Increment rxn_time_deriv_c(ispecies) by
% Rxn.stoich_coeff(irxn,ispecies)
% multiplied with the rxn_rate(irxn)

    RxnRate.time_deriv_c(ispecies) = ...
    RxnRate.time_deriv_c(ispecies) + ...
    Rxn.stoich_coeff(irxn,ispecies) * ...
    RxnRate.rate(irxn);

% PDL> ENDFOR over participating species
end

% PDL> Increment rxn_time_deriv_T by the negative of
% Rxn.delta_H divided by the product
% of density and heat capacity
% and then multiply by rxn_rate(irxn)

    RxnRate.time_deriv_T = RxnRate.time_deriv_T - ...
    (Rxn.delta_H(irxn)/density/Cp)*RxnRate.rate(irxn);

%PDL> ENDFOR over reactions

end

iflag = 1;
return;