TR_1D_model1_SS\shift_discretization_matrix.m

```
% TR_1D_model1_SS\shift_discretization_matrix.m
%
% function [A_shifted,iflag] = shift_discretization_matrix(...
%    num_pts,num_fields,A_operator,ifield);
%
% This procedure takes a square dimension num_pts
% matrix that discretizes a transport term in a PDE
% on a given computational domain, and the number of
% the field to which it is to be applied, and returns
% the square dimension num_DOF
% (where num_DOF = num_fields*num_pts)
% matrix containing these elements in the proper
% position for the selected field.  The convention is
% used that with multiple PDE's, first the values of
% field 1 are stored at each grid point, then the
% values of field 2, field 3, etc.
%
% INPUT :
% =======
% num_pts             INT
%                  this is the number of grid points in the
%                  computational domain
% num_fields    INT
%                  this is the number of fields in the PDE system
% A_operator    REAL(num_pts,num_pts)
%                  This is the discretization matrix for a spatial
%                  operator defined on the computational domain
% ifield          INT
%                  This is the number of the field to which the
%                  discretized operator is to be applied.
%
% OUTPUT :
% ========
% A_shifted      REAL(num_DOF,num_DOF) where
%                           num_DOF = num_fields*num_pts
%                  This is the matrix A_operator that is shifted
%                  for application to field # ifield.
%
% Kenneth Beers
% Massachusetts Institute of Technology
% Department of Chemical Engineering
% 7/2/2001
% Version as of 7/23/2001
```

function [A_shifted,iflag] = shift_discretization_matrix(...

```
   num_pts,num_fields,A_operator,ifield);


iflag = 0;

func_name = 'shift_discretization_matrix';


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


% Check the input.

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

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

% A_operator
num_rows=num_pts; num_columns=num_pts;
check_real=1; check_sign=0; check_int=0;
assert_matrix(i_error,A_operator,'A_operator', ...
   func_name,num_rows,num_columns, ...
   check_real,check_sign,check_int);

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


%PDL> Check to ensure that ifield is a valid field number

if(ifield > num_fields)
   iflag = -1;
   message = [ func_name, ': ', ...
         'Input ifield > num_fields'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
      error(message);
   else
      return;
```

```
   end
end
```

```
%PDL> Calculate num_DOF = num_fields*num_pts
% calculate the total number of degrees of freedom
% of the system that sets the dimenension of the
% square output matrix A_shifted.

num_DOF = num_fields*num_pts;


%PDL> Initialize A_shifted to all zeros
% the number of non-zero elements is the same in
% A_shifted as it is in A_operator.

max_nonzero = nnz(A_operator);
A_shifted = spalloc(num_DOF,num_DOF,max_nonzero);


%PDL> Set pos_offset = (ifield-1)*num_pts
% Set the integer offset that tells where the ifield
% values start in the master state vector.

pos_offset = (ifield-1)*num_pts;


%PDL> FOR EVERY non-zero element (m,n) of matrix A_operator
%        PDL> Set A_shifted(pos_offset+m,pos_offset+n) =
%          A_operator(m,n)
%PDL> ENDFOR
% Rather than a FOR loop, we use the MATLAB ability to
% operate with submatrices to use a single assignment
% statement.

A_shifted(pos_offset+1:pos_offset+num_pts, ...
   pos_offset+1:pos_offset+num_pts) = A_operator;


iflag = 1;

return;
```