TR_1D_model1_SS\FinDiff_1D_SecondDeriv.m

```
% TR_1D_model1_SS\FinDiff_1D_SecondDeriv.m
%
% function [SecondDerivMatrix,iflag] = ...
%    FinDiff_1D_SecondDeriv(Grid,imask);
%
% This procedure returns a matrix that is the
% discretized form of the second spatial derivative
% operator on a non-uniform 1-D grid.  The method of
% central finite differences is used.  The
% discretization is only performed at those grid
% points whose values of an integer mask are non-zero.
%
% INPUT :
% =======
% Grid              This data structure contains a field
%                   .num_pts that has the total number of
%                   grid points, and the field .z that
%                   has the z coordinate of each point.
% imask               INT(Grid.num_pts)
%                   This is a vector that has non-zero
%                   values at the interior points at which
%                   the derivative is to be evaluated.
%
% OUTPUT :
% ========
% SecondDerivMatrix      REAL(num_pts,num_pts) SPARSE
%                   This sparse matrix contains the discretized
%                   form of the second derivative operator at
%                   each of the grid points with non-zero
%                   imask values
%
% Kenneth Beers
% Massachusetts Institute of Technology
% Department of Chemical Engineering
% 7/2/2001
% Version as of 7/23/2001
```

```
function [SecondDerivMatrix,iflag] = ...
   FinDiff_1D_SecondDeriv(Grid,imask);


iflag = 0;

func_name = 'FinDiff_1D_SecondDeriv';
```

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


% First, check the input.

% Grid
GridType.num_fields = 2;
% .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;
% .z
ifield = 2;
FieldType.name = 'z';
FieldType.is_numeric = 1;
FieldType.num_rows = Grid.num_pts;
FieldType.num_columns = 1;
FieldType.check_real = 1;
FieldType.check_sign = 0;
FieldType.check_int = 0;
GridType.field(ifield) = FieldType;
% perform assertion
assert_structure(i_error,Grid,'Grid',func_name,GridType);

% imask
dim=Grid.num_pts; check_column=0;
check_real=1; check_sign=2; check_int=1;
assert_vector(i_error,imask,'imask', ...
   func_name,dim,check_real,check_sign, ...
   check_int,check_column);


%PDL> Initialize SecondDerivMatrix to all zeros

% find all points at which derivative is to be
% discretized
list_points = find(imask ~= 0);
num_eval_points = length(list_points);

max_nonzero = num_eval_points*3;
SecondDerivMatrix = spalloc(Grid.num_pts,Grid.num_pts, ...
   max_nonzero);
```

```
%PDL> FOR every grid point with a non-zero integer mask variable

for count=1:num_eval_points
   ipoint = list_points(count);


%        PDL> Check to ensure grid point is not a boundary point

   if(or((ipoint == 1),(ipoint == Grid.num_pts)))
      iflag = -2;
      message = [func_name, ': ', ...
            'Central FD not allowed for end point'];
      if(i_error ~= 0)
         if(i_error > 1)
            save dump_error.mat;
         end
         error(message);
      else
         return;
      end
   end


%        PDL> Set denom_inv = 1/(
%          0.5*(grid_z(m+1)-grid_z(m-1))*
%                  (grid_z(m+1)-grid_z(m))*(grid_z(m)-grid_z(m-1)) )
%   Set the inverse of the common denominator used for this point
%   in the central finite difference formula (on a non-uniform
%   1-D grid).

   i_center = ipoint;
   i_left =ipoint - 1;
   i_right = ipoint + 1;

   denom = 0.5 * ( Grid.z(i_right)-Grid.z(i_left) ) ...
      * ( Grid.z(i_right) - Grid.z(i_center) ) ...
      * ( Grid.z(i_center) - Grid.z(i_left) );
   denom_inv = 1 / denom;


%        PDL> SecondDerivMatrix(m,m-1) = denom_inv*(grid_z(m+1)-grid_z(m))

   SecondDerivMatrix(i_center,i_left) = denom_inv * ...
      (Grid.z(i_right)-Grid.z(i_center));


%        PDL> SecondDerivMatrix(m,m) = -denom_inv*(grid_z(m+1)-grid_z(m-1))

   SecondDerivMatrix(i_center,i_center) = -denom_inv * ...
```

```
      (Grid.z(i_right)-Grid.z(i_left));


%       PDL> SecondDerivMatrix(m,m+1) = denom_inv*(grid_z(m)-grid_z(m-1))

   SecondDerivMatrix(i_center,i_right) = denom_inv * ...
      (Grid.z(i_center) - Grid.z(i_left));


%PDL> ENDFOR over points with non-zero imask values

end

iflag = 1;

return;
```