TR_1D_model1_SS\discretize_boundary_deriv.m

```
% TR_1D_model1_SS\discretize_boundary_deriv.m
%
% function [c1,c2,c3,iflag] = ...
%   discretize_boundary_deriv(z1,z2,z3);
%
% This procedure uses Lagrange interpolation to
% calculate the coefficients multiplying the values
% of a field at a boundary point and two interior
% points normal to the boundary that discretize
% the normal first derivative operator at the boundary.
%
% INPUT :
% =======
% z1                    REAL
%       the value of the normal coordinate (z) at the boundary
% z2                    REAL
%       the value of the normal coordinate at the first point
%       within the interior
% z3                    REAL
%       the value of the normal coordinate at the second point
%       within the interior
%-----------------  * = z1
%                    * = z2
%                    * = z3
%
% OUTPUT :
% ========
% c1                    REAL
%       the coefficient multiplying the field value at z1 in
%       the discretized form of the normal derivative operator
% c2                    REAL
%       the coefficient multiplying the field value at z2 in
%       the discretized form of the normal derivative operator
% c3                    REAL
%       the coefficient multiplying the field value at z3 in
%       the discretized form of the normal derivative operator
% iflag       INT
%       this integer flag tells how the routine has performed
%       its job :
%       iflag < 0, exit with error
%       iflag = 0, incomplete
%       iflag = 1, successful completion
%
% Kenneth Beers
% Massachusetts Institute of Technology
% Department of Chemical Engineering
% 7/2/2001
%
```

```
% Version as of 7/23/2001


function [c1,c2,c3,iflag] = ...
   discretize_boundary_deriv(z1,z2,z3);


iflag = 0;

func_name = 'discretize_boundary_deriv';


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


% check input

% z1, z2, z3
check_real=1; check_sign=0; check_int=0;
assert_scalar(i_error,z1,'z1', ...
   func_name,check_real,check_sign,check_int);
assert_scalar(i_error,z2,'z2', ...
   func_name,check_real,check_sign,check_int);
assert_scalar(i_error,z3,'z3', ...
   func_name,check_real,check_sign,check_int);


% check that these z values are distinct
i_distinct = 1;
if(z1==z2)
   i_distinct = 0;
end
if(z1==z3)
   i_distinct = 0;
end
if(z2==z3)
   i_distinct = 0;
end
if(i_distinct == 0)
   iflag = -1;
   message = [func_name, ': ', ...
         'Input z1,z2,z3 are not distinct'];
   if(i_error ~= 0)
      if(i_error > 1)
         save dump_error.mat;
      end
```

```
      error(message);
   else
      return;
   end
end
```

%PDL> c1 = (2*z1 - z2 - z3) / ((z1-z2)*(z1-z3))

```
c1 = (2*z1 - z2 - z3) / ( (z1-z2)*(z1-z3) );
```

%PDL> c2 = (2*z1 - z1 - z3) / ((z2-z1)*(z2-z3))

```
c2 = (2*z1 - z1 - z3) / ( (z2-z1)*(z2-z3) );
```

%PDL> c3 = (2*z1 - z1 - z2) / ((z3-z1)*(z3-z2))

```
c3 = (2*z1 - z1 - z2) / ( (z3-z1)*(z3-z2) );


iflag = 1;

return;
```