MATLAB Files for 1D polymer flow problem

This program calculates the velocity profile of a shear-thinning polymer fluid in pressure-driven flow between two infinite, parallel plates separated by a distance B. The bottom plate is stationary and the top plate is moving at a specified velocity in the x-direction. The pressure gradient in the x-direction is specified and is used to drive the flow. Laminar flow conditions are assumed. The constitutive equation employed is that of Yasuda, Armstrong, and Cohen (Rheological Acta, 20, 163-178, 1981).

K. Beers
MIT ChE
10/24/2001

File: polymer_flow_1D.m

```
% polymer_flow_1D\polymer_flow_1D.m
%
% function iflag_main = polymer_flow_1D();
%
% TITLE :
% =======
% Calculation of velocity profile v_x(y) of a shear-thinning
% polymer fluid in pressure-driven laminar flow between two
% infinite, parallel plates separated by a distance B, the top
% one of which is moving at a specified velocity.
%
% AUTHOR :
% ========
% Kenneth Beers
% MIT ChE
% 10/24/2001
%
% PROGRAM SUMMARY :
% =================
% This program calculates the velocity profile of a
% shear-thinning polymer fluid in pressure-driven flow
% between two infinite, parallel plates separated by a
% distance B.  The bottom plate is stationary and the top
% plate is moving at a specified velocity in the x-direction.
% The pressure gradient in the x-direction is specified and is
% used to drive the flow.  Laminar flow conditions are assumed.
% The constitutive equation employed is that of Yasuda, Armstrong,
% and Cohen (Rheological Acta, 20, 163-178, 1981).
%
% PROGRAM IMPLEMENTATION NOTES :
% ==============================
%
```

```
% 1. Discretization of PDE
% Central finite differences will be used to convert the partial
% differential equation for the steady state velocity profile
% into a set of nonlinear algebraic equations.  These are written
% to account for the spatially-varying viscosity due to the
% shear-thinning behavior of the fluid.
%
% 2. The set of nonlinear algebraic equations will be solved
% using the MATLAB routine fsolve that is part of the
% optimization toolkit.  As an initial guess of the velocity profile,
% we will use the analytical solution for a Newtonian fluid under the
% same circumstances.  Since the problem is of large dimension, we will
% use the large-scale algorithm of fsolve, and will provide a matrix
% that specifies the sparsity pattern of the Jacobian.
%
% INPUT :
% =======
%
% System = data structure containing system parameters
% .B
% REAL
% The distance between the two parallel plates.
% .dp_dx
% REAL
% The pressure gradient in the x direction that drives the flow.
% .velocity_up
% REAL
% The velocity of the upper plate in the x-direction.
%
% Fluid = data structure containing fluid parameters
% .visc_0
% REAL
% The zero shear rate viscosity of the fluid.
% .visc_inf
% REAL
% The limiting viscosity as the shear rate approaches infinity.
% .relax_t
% REAL
% The relaxation time of the fluid.
% .n_pow
% REAL
% The power-law exponent of the fluid.
% .a
% REAL
% The coefficient controlling the shape of the viscosity
% curve in the cross-over region.
%
% Grid = data structure containing compuational grid data
% .num_pts
% INT
% The number of grid points in the y-direction.
```

```
% .y
% REAL(num_pts)
% The values of the y coordinate at each grid point.  For
% simplicity, we will use a uniform grid.
%
% OUTPUT :
% ========
%
% v_x
% REAL(num_pts)
% The x-direction velocity profile for the shear
% thinning fluid, calculated numerically.
%
% v_x_Newton
% REAL(num_pts)
% The analytical velocity profile obtained for a
% Newtonian fluid.
%
% viscosity
% REAL(num_pts)
% A plot of the viscosity as a function of position for
% the steady-state velocity profile.


function iflag_main = polymer_flow_1D();

iflag_main = 0;


%PDL> Prompt the user for the name of the input m-file that sets
% the values of the simulation parameters.  Then, use feval to
% run this function indirectly.  See the PDL code for the
% function polymer_flow_1D below for the form that this
% input file should take.

filename = input('Enter input filename (with .m or quotes) : ','s');
[System,Fluid,Grid,iflag] = feval(filename);
if(iflag <= 0)
   imain_flag = iflag;
   error('polymer_flow_1D: input routine failed');
end


%PDL> For these parameters, calculate the analytical velocity
% profile for a Newtonian fluid and use this profile as an
% initial guess for the velocity profile with the shear thinning fluid.

v_x_Newton = (System.velocity_up*System.B).*Grid.y + ...
   (1/2/Fluid.visc_0*System.dp_dx).*(Grid.y.^2 - Grid.y.*System.B);

v_x_guess = v_x_Newton;
```

```
%PDL> Initialize the options for the numerical solver routine
% fsolve and set the matrix that specifies the sparsity pattern
% of the Jacobian.  Use the default large scale algorithm and
% change as needed the level of information that the solver
% routine displays to the screen.

S_Jac = spalloc(Grid.num_pts,Grid.num_pts,5*Grid.num_pts);
S_Jac(1,1) = 1;
for igrid=2:(Grid.num_pts-1)
   S_Jac(igrid,igrid-1) = 1;
   S_Jac(igrid,igrid) = 1;
   S_Jac(igrid,igrid+1) = 1;
   % We now include the weak dependence on the outlying two
   % points through the velocity gradient estimate used to
   % calculate the viscosity at each grid point.
   if(igrid > 2)
      S_Jac(igrid,igrid-2) = 1;
   end
   if(igrid < (Grid.num_pts-1))
      S_Jac(igrid,igrid+2) = 1;
   end
end
S_Jac(Grid.num_pts,Grid.num_pts) = 1;

options = optimset('TolFun',1e-10,'JacobPattern',S_Jac, ...
   'LargeScale','off');


%PDL> Call the MATLAB built-in routine fsolve to numerically
% solve the set of nonlinear algebraic equations for the velocity
% profile of the shear-thinning fluid, where the function that
% calculates the function vector is polymer_flow_1D_calc_f.

i_use_homotopy = input('Try direct calc. (0) or use homotopy (1)? : ');

if(~i_use_homotopy)
   exitflag = 0;
   while(exitflag <= 0)
      [v_x,fval,exitflag,output] = fsolve(@polymer_flow_1D_calc_f, ...
         v_x_guess,options,Grid,System,Fluid);
      if(exitflag <= 0)
         imain_flag = exitflag;
         disp('polymer_flow_1D: fsolver failed to converge');
         ichoose = ...
            input('Stop (0) or Try again (1)? :');
         if(ichoose==0)
            error('polymer_flow_1D: fsolver failed to converge');
         else
            % update the guess with the new results
```

```
            v_x_guess = v_x;
         end
      end
   end

else

   num_homotopy = input('Enter # of homotopy iterations : ');
   if(System.dp_dx < 0)
      ineg = 1;
   else
      ineg = 0;
   end
   dp_dx_min = min(1e3,abs(System.dp_dx)/10);
   dp_dx_vect = logspace(log10(dp_dx_min), ...
      log10(abs(System.dp_dx)),num_homotopy);
   if(ineg)
      dp_dx_vect = -dp_dx_vect;
   end

   % Recalculate velocity profile for the lowest flowrate
   System.dp_dx = dp_dx_vect(1);
   v_x_guess = (System.velocity_up*System.B).*Grid.y + ...
      (1/2/Fluid.visc_0*System.dp_dx).*(Grid.y.^2 - Grid.y.*System.B);

   for i_homotopy = 1:num_homotopy
      System.dp_dx = dp_dx_vect(i_homotopy);
      disp(['Beginning i_homotopy = ', int2str(i_homotopy), ...
            ', System.dp_dx = ', num2str(System.dp_dx)]);
      exitflag = 0;

      % use lower accuracy except for last iteration
      if(i_homotopy < num_homotopy)
         options = optimset(options,'TolFun',1e-4);
      else
         options = optimset(options,'TolFun',1e-10);
      end

      while(exitflag <= 0)
         [v_x,fval,exitflag,output] = ...
            fsolve(@polymer_flow_1D_calc_f, ...
               v_x_guess,options,Grid,System,Fluid);
         if(exitflag <= 0)
            imain_flag = exitflag;
            disp('polymer_flow_1D: fsolver failed to converge');
            ichoose = ...
               input('Stop (0) or Try again (1)? : ');
            if(ichoose==0)
               error('polymer_flow_1D: fsolver failed to converge');
            else
               % update the guess with the new results
```

```
            v_x_guess = v_x;
          end
        end
      end

      % update the guess of the velocity profile with the old results
      v_x_guess = v_x;

   end

end


%PROCEDURE: calc_viscosity_profile
%PDL> For the steady-state velocity profile, calculate the
% viscosity and velocity gradient at each grid point.
%ENDPROCEDURE

[viscosity,dvx_dy] = calc_viscosity_profile(v_x,Grid,Fluid);


%PDL> Plot the Newtonian and shear thinning velocity profiles,
% the shear gradient, and the viscosity as a function of y on
% a master plot.

figure;
subplot(2,2,1);
plot(Grid.y,v_x_Newton);
xlabel('y');
ylabel('v_x (Newtonian)');
axis tight;

subplot(2,2,2);
plot(Grid.y,v_x);
xlabel('y');
ylabel('v_x (polymer)');
axis tight;

subplot(2,2,3);
plot(Grid.y,dvx_dy);
xlabel('y');
ylabel('dv_x/dy');
axis tight;

subplot(2,2,4);
plot(Grid.y,viscosity);
xlabel('y');
ylabel('viscosity (\eta)');
axis tight;

text1 = ['B = ', num2str(System.B)];
```

```
text1 = [text1, ' \Delta p / \Delta x = ',num2str(System.dp_dx)];
text1 = [text1, ' V_{up} = ',num2str(System.velocity_up)];
gtext(text1);


% Save the results to a binary output file.
save polymer_flow_1D_results.mat;


iflag_main = 1;

return;
```

File: polymer_flow_1D_input1.m

```
% \polymer_flow_1D\polymer_flow_1D_input1.m
%
% function [System,Fluid,Grid,iflag] = polymer_flow_1D_input();
%
% This function sets the simulation parameters for the 1D polymer
% flow numerical calculation, and is used as the primary means to
% input data into the simulation.
%
% OUTPUT :
% ========
% System = data structure containing system parameters
% .B
% REAL
% The distance between the two parallel plates.
% .dp_dx
% REAL
% The pressure gradient in the x direction that drives the flow.
% .velocity_up
% REAL
% The velocity of the upper plate in the x-direction.
%
% Fluid = data structure containing fluid parameters
% .visc_0
% REAL
% The zero shear rate viscosity of the fluid.
% .visc_inf
% REAL
% The limiting viscosity as the shear rate approaches infinity.
% .relax_t
% REAL
% The relaxation time of the fluid.
% .n_pow
% REAL
% The power-law exponent of the fluid.
% .a
% REAL
% The coefficient controlling the shape of the viscosity
% curve in the cross-over region.
%
% Grid = data structure containing compuational grid data
% .num_pts
% INT
% The number of grid points in the y-direction.
% .y
% REAL(num_pts)
% The values of the y coordinate at each grid point.  For
% simplicity, we will use a uniform grid.
```

```
function [System,Fluid,Grid,iflag] = polymer_flow_1D_input1();

iflag = 0;


%PDL> Set the value of the distance between the plates, the pressure
% gradient, and the velocity of the upper plate.

System.B = 0.01;
System.dp_dx = -1e7;
System.velocity_up = 0;


%PDL> Set the parameters that define the constitutive equation
% relating shear rate to viscosity for the polymer fluid.

Fluid.visc_0 = 1180;
Fluid.visc_inf = 0;
Fluid.relax_t = 9.24e-2;
Fluid.n_pow = 0.441;
Fluid.a = 2;


%PDL> Specify the number of points in the computational grid, and
% set their locations based on a uniform grid spacing.

Grid.num_pts = 50;
Grid.y = linspace(0,System.B,Grid.num_pts)';


iflag = 1;

return;
```

File: polymer_flow_calc_f.m

```
% polymer_flow_1D\polymer_flow_1D_calc_f.m
%
% function [fval,iflag] = polymer_flow_1D_calc_f(v_x, ...
%    Grid,System,Fluid);
%
% This function takes as input an estimate of the velocity
% profile and calculates the value of each algebraic equation.
%
% INPUT :
% =======
%
% v_x
% REAL(num_pts)
% The x-direction velocity profile for the shear
% thinning fluid, calculated numerically.
%
% Grid = data structure containing compuational grid data
% .num_pts
% INT
% The number of grid points in the y-direction.
% .y
% REAL(num_pts)
% The values of the y coordinate at each grid point.  For
% simplicity, we will use a uniform grid.
%
% System = data structure containing system parameters
% .B
% REAL
% The distance between the two parallel plates.
% .dp_dx
% REAL
% The pressure gradient in the x direction that drives the flow.
% .velocity_up
% REAL
% The velocity of the upper plate in the x-direction.
%
% Fluid = data structure containing fluid parameters
% .visc_0
% REAL
% The zero shear rate viscosity of the fluid.
% .visc_inf
% REAL
% The limiting viscosity as the shear rate approaches infinity.
% .relax_t
% REAL
% The relaxation time of the fluid.
% .n_pow
```

```
% REAL
% The power-law exponent of the fluid.
% .a
% REAL
% The coefficient controlling the shape of the viscosity
% curve in the cross-over region.
%
% OUTPUT :
% ========
%
% fval
% REAL(Grid.num_pts)
% This is a vector containing the function values for each
% nonlinear algebraic equation.

function [fval,iflag] = polymer_flow_1D_calc_f(v_x, ...
   Grid,System,Fluid);

iflag = 0;


% Allocate memory for the function vector.
fval = linspace(0,0,Grid.num_pts)';


%PROCEDURE: calc_viscosity_profile
%PDL> First, use finite differences to calculate the
% viscosity at each grid point.
%ENDPROCEDURE

[viscosity,dvx_dy,iflag] = calc_viscosity_profile(v_x,Grid,Fluid);


%PDL> For each interior grid point
%   FOR igrid = 2: (Grid.num_pts-1)

for igrid=2:(Grid.num_pts-1)


%***PDL> Calculate the value of the function for this grid point.

   fval(igrid) = (viscosity(igrid+1)+viscosity(igrid))* ...
            (Grid.y(igrid)-Grid.y(igrid-1))* ...
            (v_x(igrid+1)-v_x(igrid)) - ...
            (viscosity(igrid)+viscosity(igrid-1))* ...
            (Grid.y(igrid+1)-Grid.y(igrid))* ...
            (v_x(igrid)-v_x(igrid-1)) ...
         - System.dp_dx* ...
            (Grid.y(igrid+1)-Grid.y(igrid-1))* ...
            (Grid.y(igrid+1)-Grid.y(igrid))* ...
            (Grid.y(igrid)-Grid.y(igrid-1));
```

%PDL> ENDFOR

end

%PDL> For each wall grid point, calculate the values of the
% function enforcing the boundary condition.

% lower wall at y = 0
fval(1) = v_x(1);

% upper wall at y = B
fval(Grid.num_pts) = v_x(Grid.num_pts) - System.velocity_up;

iflag = 1;

return;

File: calc_viscosity_profile.m

```
% polymer_flow_1D\calc_viscosity_profile.m
%
% function [viscosity,dvx_dy,iflag] = ...
%   calc_viscosity_profile(v_x,Grid,Fluid);
%
% This function uses second order finite difference
% approximations to obtain the velocity gradient at each
% interior point and at the walls (where 1-sided equations
% based on Lagrange interpolation must be used).  This is
% written to be compatible with a grid that has non-uniform
% spacing.
%
% INPUT :
% =======
%
% v_x
% REAL(num_pts)
% The x-direction velocity profile for the shear
% thinning fluid, calculated numerically.
%
% Grid = data structure containing compuational grid data
% .num_pts
% INT
% The number of grid points in the y-direction.
% .y
% REAL(num_pts)
% The values of the y coordinate at each grid point.  For
% simplicity, we will use a uniform grid.
%
% Fluid = data structure containing fluid parameters
% .visc_0
% REAL
% The zero shear rate viscosity of the fluid.
% .visc_inf
% REAL
% The limiting viscosity as the shear rate approaches infinity.
% .relax_t
% REAL
% The relaxation time of the fluid.
% .n_pow
% REAL
% The power-law exponent of the fluid.
% .a
% REAL
% The coefficient controlling the shape of the viscosity
% curve in the cross-over region.
%
```

```
% OUTPUT :
% ========
%
% viscosity
% REAL(num_pts)
% A plot of the viscosity as a function of position for
% the steady-state velocity profile.
%
% dvx_dy
% REAL(num_pts)
% A vector of the velocity gradient at each grid point.
%
% K. Beers
% MIT ChE
% 10/23/2001


function [viscosity,dvx_dy,iflag] = ...
   calc_viscosity_profile(v_x,Grid,Fluid);

iflag = 0;


% Allocate memory for output.
dvx_dy = linspace(0,0,Grid.num_pts)';
viscosity = linspace(0,0,Grid.num_pts)';


%PDL>  For every interior point
%       FOR igrid = 2 : (Grid.num_pts-1)

for igrid = 2:(Grid.num_pts-1)


%***PDL> Use central finite differences to calculate the
% velocity gradient and store the value

   dvx_dy(igrid) = (v_x(igrid+1)-v_x(igrid-1)) / ...
      (Grid.y(igrid+1)-Grid.y(igrid-1));


%PRODCEDURE: calc_viscosity_polymer
%***PDL> Pass the absolute value of this velocity gradient to
% a function that returns the value of the viscosity at this
% point based on the constitutive relation and store the
% value in the appropriate location.
%ENDPROCEDURE

   gamma_dot = abs(dvx_dy(igrid));
   viscosity(igrid) = calc_viscosity_polymer(gamma_dot,Fluid);
```

```
%PDL> ENDFOR

end


%PDL> Repeat this calculation for each wall grid point using
% one-sided Lagrange interpolation to estimate the velocity
% gradient at the wall.  Use the pre-existing Lagrange
% interpolation routine from the program TR_1D_model1_SS.

% bottom wall at y = 0
[c1,c2,c3,iflag] = discretize_boundary_deriv(...
   Grid.y(1),Grid.y(2),Grid.y(3));
dvx_dy(1) = c1*v_x(1) + c2*v_x(2) + c3*v_x(3);
gamma_dot = abs(dvx_dy(1));
viscosity(1) = calc_viscosity_polymer(gamma_dot,Fluid);

% upper wall at y = B
[c1,c2,c3,iflag] = discretize_boundary_deriv(...
   Grid.y(Grid.num_pts),Grid.y(Grid.num_pts-1),Grid.y(Grid.num_pts-2));
dvx_dy(Grid.num_pts) = c1*v_x(Grid.num_pts) + ...
   c2*v_x(Grid.num_pts-1) + c3*v_x(Grid.num_pts-2);
gamma_dot = abs(dvx_dy(Grid.num_pts));
viscosity(Grid.num_pts) = calc_viscosity_polymer(gamma_dot,Fluid);


iflag = 1;

return;
```

File: discretize_boundary_deriv.m

```
% polymer_flow_1D\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';

i_error = 2;


% 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;

File: calc_viscosity_polymer.m

```
% polymer_flow_1D\calc_viscosity_polymer.m
%
% function [viscosity,iflag] = ...
%    calc_viscosity_polymer(gamma_dot,Fluid);
%
% This function calculates the viscosity of a
% shear-thinning polymer fluid using the constitutive
% model of Yasuda, Armstrong, and Cohen, Rheol. Acta,
% v20, 163-178, 1981.
%
% INPUT :
% =======
%
% gamma_dot
% REAL
% The shear rate of the fluid.
%
% REAL
% The zero shear rate viscosity of the fluid.
% .visc_inf
% REAL
% The limiting viscosity as the shear rate approaches infinity.
% .relax_t
% REAL
% The relaxation time of the fluid.
% .n_pow
% REAL
% The power-law exponent of the fluid.
% .a
% REAL
% The coefficient controlling the shape of the viscosity
% curve in the cross-over region.
%
% OUTPUT :
% ========
%
% viscosity
% REAL
% The value of the fluid viscosity at the input shear rate.
%
% K Beers
% MIT ChE
% 10/23/2001


function [viscosity,iflag] = ...
   calc_viscosity_polymer(gamma_dot,Fluid);
```

```
iflag = 0;
```

%PDL> Check to make sure that the input shear rate is non-negative.

```
if(gamma_dot < 0)
   gamma_dot = -gamma_dot;
end
```

%PDL> Calculate the viscosity for this shear rate
% according to the constitutive law.

```
viscosity = Fluid.visc_inf + ...
   (Fluid.visc_0 - Fluid.visc_inf)* ...
   (1 + (Fluid.relax_t*gamma_dot)^Fluid.a)^ ...
      ((Fluid.n_pow-1)/Fluid.a);

iflag = 1;

return;
```