

Colloidal Mushy Layers

M. Grae Worster, S.S.L. Peppin and J.S. Wettlaufer

Supplementary Material

Included in this Supplementary Material are two Matlab programs, which are exemplars of the programs to produce the results in this paper.

```

clear all

% Program to find the solidification rate lambda, the base temperature TB
% and the concentration field at marginal equilibrium
% for solidification of a colloidal suspension at a planar interface.
% Uses the concentration as the primary dependent variable.

% Material and universal constants

kB = 1.38e-23;    % Boltzmann's constant
Tm = 273.15;     % Freezing temperature (Kelvin)
rho = 10^3;      % Density of water (kg m^{-3})
L = 3.34e5;      % Latent heat of freezing (J kg^{-1})
mu = 1e-3;       % Dynamic viscosity of water (Pa s)
kappa = 1.43e-7; % Thermal diffusivity of water (m^2 s^{-1})
phic = 0.64;     % Concentration at close packing

% Experimental parameters

R = 1e-9;        % Particle radius
Tinf = 10;       % Far-field temperature
phinf = 0.05;    % Far-field concentration

% Temperature field
T = @(TB, Tinf, eta) Tinf + (TB - Tinf)*erfc(eta);

% Particle volume
vp = @(R) (4/3)*pi*R.^3;

% Diffusivity scale
D0 = @(R) kB*Tm/6/pi/R/mu;

% Liquidus slope at zero concentration
m0 = @(R) Tm/rho/L * kB*Tm./vp(R);

% Osmotic pressure as a function of concentration and radius
Osmotic = @(phi, R) kB*Tm/vp(R)*phic*phi./(phic - phi);

% Liquidus temperature as a function of concentration and radius
Liquidus = @(phi, R) -m0(R)*phic*phi./(phic - phi);

% Concentration as a function of osmotic pressure and radius
InvOsm = @(op, R) phic*op./(op + kB*Tm/vp(R)*phic);

% Liquidus concentration as a function of temperature and radius
PhiLiq = @(TL, R) -phic*TL./(m0(R)*phic - TL);

% Dimensionless permeability as a function of concentration and radius
khat = @(phi, R) (1 - phi).^6;

% Dimensionless diffusivity as a function of concentration and radius
Dhat = @(phi, R) (phic/(phic - phi))^2 * khat(phi, R);

```

```

% The following two statements determine the base temperature TB for a
% given value of lambda as a root of equation (4.9)

marginal = @(TB, Tinf, lam, R) ...
    (Tinf - TB).*khat(PhiLiq(T(TB, Tinf, lam), R), R) - ...
    sqrt(pi)*lam*exp(lam^2) * kappa*m0(R)/D0(R).*PhiLiq(T(TB, Tinf, lam), R);

TBase = @(Tinf, lam, R, TBguess) ...
    fzero(@(TB) marginal(TB, Tinf, lam, R), ...
    [1.2*TBguess, 1.00001*Tinf*(1 - 1/erfc(lam))]);

%----- Main program -----

etainf = 0.4;    % Thickness of colloidal boundary layer

lam = 0.024847; % Initial guess at ice-colloid interface position
TB = -0.44789;  % Initial guess at base temperature

% Differential equations
dydx = @(x, y, lam) [ -2*kappa / D0(R) / Dhat(y(1), R) * y(2); ...
    -2*kappa / D0(R) / Dhat(y(1), R) * (x + lam)*y(2) ];

% Range of integration. Note that integration starts at the phase boundary
% eta = lambda with a new integration variable eta - lambda.
tspan = [0, etainf];

% The solution vector z=[phi, q], with initial values given as a function
% of lambda
z0 = @(lam) [PhiLiq(T(TBase(Tinf, lam, R, TB), Tinf, lam), R), ...
    PhiLiq(T(TBase(Tinf, lam, R, TB), Tinf, lam), R)*lam];

% For given lambda, find the solution of the differential equations
solution = @(lam) ode15s(@(t,z) dydx(t, z, lam), tspan, z0(lam));

% and calculate the residual of the far-field condition phi = phinf
residue = @(lam) deval(solution(lam), etainf, 1) - phinf;

% Determine the value of lambda that makes the residue equal to zero
lam = fzero(residue, [0.99*lam, 1.01*lam]);

```

```

%----- Plot results -----

% With the value of lambda determined above, calculate the base temperature
TB = TBase(Tinf, lam, R, TB);

% and the solution curve
[t,z] = ode15s(@(t,z) dydx(t,z,lam), tspan, z0(lam));

figure(1) % Plot of solution in physical space
plot(z(:, 1), lam + t, 'b-o')
set(gca, 'FontSize', 20);
xlabel('\phi')
ylabel('\eta')
set(get(gca, 'YLabel'), 'Rotation', 0)

figure(2) % Plot of solution in phase space of temperature versus
concentration
plot(z(:, 1), T(TBase(Tinf, lam, R, TB), Tinf, lam + t), 'r', 'LineWidth', 4)
hold on

temp = linspace(-6, 0, 10000);
plot(PhiLiq(temp, R), temp, 'b', 'LineWidth', 2)
set(gca, 'FontSize', 30, 'LineWidth', 1);
xlabel('\phi')
ylabel('T')
set(get(gca, 'YLabel'), 'Rotation', 0)
hold off

```

```

clear all

% Finds the bottom and top positions of the mushy layer
% then finds the distribution of ice fraction in the layer

% Material and universal constants

kB = 1.38e-23;    % Boltzmann's constant
Tm = 273.15;     % Freezing temperature (Kelvin)
rho = 10^3;      % Density of water (kg m-3)
L = 3.34e5;      % Latent heat of freezing (J kg-1)
mu = 1e-3;       % Dynamic viscosity of water (Pa s)
kappa = 1.43e-7; % Thermal diffusivity of water (m2 s-1)
phic = 0.64;     % Concentration at close packing

% Experimental parameters

R = 1e-9;        % Particle radius
Tinf = 10;       % Far-field temperature (Celcius)
TB = -4;         % Fixed base temperature (Celcius)
phinf = 0.05;    % Far-field concentration

% Temperature field and its first and second derivatives
T = @(eta) Tinf + (TB - Tinf).*erfc(eta);
Tp = @(eta) -2/sqrt(pi) * (TB - Tinf).*exp(-eta^2);
Tpp = @(eta) 4/sqrt(pi) * (TB - Tinf).*eta.*exp(-eta^2);
phinf = 0.05;

% Particle volume
vp = @(R) (4/3)*pi*R.^3;

% Diffusivity scale and dimensionless diffusivity
D0 = @(R) kB*Tm/6/pi/R/mu;
epsilon = @(R) D0(R)/kappa;

% Liquidus slope at zero concentration
m0 = @(R) Tm/rho/L * kB*Tm./vp(R);

% Osmotic pressure as a function of concentration and radius
osmotic = @(phi, R) kB*Tm/vp(R)*phi./(1 - phi/phic);

% Derivative of osmotic pressure with respect to concentration
osmotiicp = @(phi, R) kB*Tm/vp(R) / (1 - phi/phic)^2;

% Liquidus temperature as a function of particle concentration
Liquidus = @(phi, R) Tm*(1 - osmotic(phi, R)/rho/L);

% Liquidus slope
LiqSlope = @(phi, R) (Tm/rho/L)*osmotiicp(phi, R);

% Liquidus concentration as a function of
% liquidus temperature and radius

```

```

PhiLiq = @(TL, R) -phic.*TL./(m0(R)*phic - TL);

% Slope of the Liquidus concentration as a function of
% liquidus temperature and radius
PhiLiqPrime = @(TL, R) -m0(R) * phic^2 ./ (m0(R)*phic - TL)^2;

% Dimensionless permeability as a function of concentration and radius
khat = @(phi, R) (1 - phi).^6;

% Permeability as a function of concentration and radius
kp = @(x, R) (2*R^2/9)*(1 - x).^6./x;

% Bulk diffusivity as a function of particle concentration
diffusivity = @(x, R) (kp(x, R)/mu)*(rho*L/Tm) .* x .* LiqSlope(x, R);

% Normalised diffusivity and its gradient with respect of particle
% concentration
Dhat = @(x, R) diffusivity(x, R)/D0(R);
syms z
Dhatp = @(x, R) vpa(subs(diff(Dhat(z, R)), z, x));

%----- Main program -----

% Determine lambda_a as the root of equation (7.7)

la = 0.0001; % Initial guess for lambda_a

residuala = @(TB, Tinf, lam, R) ...
    (Tinf - TB).*(khat(PhiLiq(T(lam), R), R) - ...
    sqrt(pi)*lam*exp(lam^2) * kappa*m0(R)/D0(R).*(PhiLiq(T(lam), R);

lama = @(Tinf, TB, R, guess) ...
    fzero(@(lam) residuala(TB, Tinf, lam, R), guess);

la = lama(Tinf, TB, R, la)

% Determine lambda_b by solving (4.5)
% with boundary conditions (7.4) and (7.5)

lb = 0.25; % Initial guess for lambda_b

% Determines lambda_b given phi so that mush-liquid interface in on liquidus
LiqRes = @(phi, lam, R) Liquidus(phi, R) - Tm - T(lam);
lamb = @(phi, R, guess) fzero(@(lam) LiqRes(phi, lam, R), guess);

% Defines the boundary condition (7.4b)
phibp = @(phib, R, guess) -1/LiqSlope(phib, R) * Tp(lamb(phib, R, guess));

```

```

phib = phinf; % Initial guess for phi_b

% Derivatives. Note that the solution vector here is [phi, phi'] rather
% than [phi, q] reported in the paper
dydx = @(x, y, phib) ...
    [ y(2); -2*(x + lamb(phib, R, lb))*y(2)/epsilon(R)/Dhat(y(1), R) - ...
      Dhatp(y(1), R)*y(2)^2/Dhat(y(1), R) ];

% Implements the boundary conditions (7.4) and (7.5)
res = @(ya, yb, phib) [ ya(1) - phib; ya(2) - phibp(phib, R, lb); ...
                        yb(1) - phinf ];

% Initial guess for the structure of phi(x) and q(x)
yinit = @(x) [ phinf + (phib - phinf)*exp(-x/sqrt(epsilon(R))); ...
               -(phib - phinf)/epsilon(R)*x*exp(-x/sqrt(epsilon(R))) ];
solinit = bvpinit(linspace(0, 20*epsilon(R), 10), yinit, phib);

% Solves the boundary-value problem for phi(x) and q(x)
% with unknown parameter phib
sol = bvp4c(dydx, res, solinit);
phib = sol.parameters;

lb = lamb(phib, R, lb)

% Given values of lambda_a and lambda_b, solve (7.8) for chi

ChiDerivs = @(x, y) ...
    ( 2*R^2/9/mu/kappa * rho*L/Tm * ...
      ( (1 - PhiLiq(T(x), R))^6 * Tpp(x) - ...
        6* (1 - PhiLiq(T(x), R))^5 * PhiLiqPrime(T(x), R) * Tp(x)^2 ) - ...
        2*x*PhiLiqPrime(T(x), R) * Tp(x) ) * y / ...
      ( 2*x*PhiLiq(T(x), R) - ...
        2*R^2/9/mu/kappa * rho*L/Tm * (1 - PhiLiq(T(x), R))^6 * Tp(x) );

xspan = [lb, la+eps];
Chi0 = 1;

[eta, chi] = ode15s(@(x,z) ChiDerivs(x, z), xspan, Chi0);

%----- Plot results -----

plot(chi, eta, 'r', 'LineWidth', 2)
hold on
plot(-chi, eta, 'r', 'LineWidth', 2)
plot([-chi(end), chi(end)], [eta(end), eta(end)], 'r', 'LineWidth', 2)
pbaspect([1, 2, 3])
set(gca, 'FontSize', 20, 'LineWidth', 1);
hold off

```