Supplement of Solid Earth, 13, 1045–1063, 2022
https://doi.org/10.5194/se-13-1045-2022-supplement

Solid Earth Open Access EGU

Supplement of

Thermal non-equilibrium of porous flow in a resting matrix applicable to melt migration: a parametric study

Laure Chevalier and Harro Schmeling

Correspondence to: Harro Schmeling (schmeling@geophysik.uni-frankfurt.de)

1 Analytical solution

Eq. (21) (in the main paper) is a second order differential equation for $T_f - T_s$ with constant parameters. Its solution is composed of the sum of a particular solution and of the solution of the corresponding homogeneous differential equation.

1.1 Particular solution

The right hand side of Eq. (21) is a constant. In this special case with constant parameters, a simple way to find a particular solution consists in assuming that for this solution, $T_f - T_s$ is a constant. Its derivative is equal to zero, and the constant value of $T_f - T_s$ is:

$$T_f - T_s = PeG \tag{S1}$$

1.2 Homogeneous solution

We now need to find the general solution of the homogeneous equation

$$\frac{\partial^2(T_f - T_s)}{\partial z^2} - Pe\frac{\partial(T_f - T_s)}{\partial z} - \left(T_f - T_s\right) = 0 \tag{S2}$$

The second order algebraic equation associated with Eq. (S2) is

$$r^2 - Pe\, r - 1 = 0 \tag{S3}$$

Its determinant and roots are

$$\Delta = Pe^2 + 4 \quad , \; r_1 = \frac{1}{2}\left(Pe - \sqrt{Pe^2 + 4}\right), \; r_2 = \frac{1}{2}\left(Pe + \sqrt{Pe^2 + 4}\right)$$

and the complete solution of Eq. (21) is of the form:

$$T_f - T_s = \alpha e^{r_1 z} + \beta e^{r_2 z} + PeG \tag{S4}$$

We now need to determine α and β from boundary conditions.

1.3 Constraints from boundary conditions

At z = 0, both T_f and T_s values are set constant, to the same value. Therefore we have the condition $T_f - T_s = 0$ at z = 0. We then have the following relationship for α and β:

$$\alpha + \beta = -PeG \tag{S5}$$

At $z = H$, both T_f and T_s gradients are constant and equal to each other. Thus we have the condition $\frac{\partial(T_f - T_s)}{\partial z} = 0$ at $z = H = 1/G$. This gives the following relationship for α and β :

$$\alpha r_1 e^{r_1/G} + \beta r_2 e^{r_2/G} = 0 \tag{S6}$$

From Eq. (S5) and (S6) we get

$$\alpha = PeG\frac{r_2}{r_1 e^{(r_1 - r_2)/G} - r_2}, \quad \beta = PeG\frac{r_1}{r_2 e^{(r_2 - r_1)/G} - r_1} \tag{S7}$$

2 Limits determination

2.1 Limit $Pe \to 0$

When *Pe* tends to 0, Pe^2 becomes negligible with respect to 4. We then get

$r_1 \to -1, \qquad r_2 \to 1$

and obtain after few manipulations the following limit for $(T_f - T_s)$ using Eq. (22)

$$T_f - T_s = PeG\left(1 - \frac{e^{-z}}{1+e^{-2/G}} - \frac{e^{z}}{1+e^{2/G}}\right) = PeG\left(1 - \frac{e^{-z}+e^{2/G-z}+e^{z}+e^{-2/G+z}}{2+e^{-2/G}+e^{2/G}}\right) = PeG(1-M) \tag{S8}$$

with

$$M = \frac{\cosh(z)+\cosh\left(\frac{2}{G}-z\right)}{1+\cosh\left(\frac{2}{G}\right)} \tag{S9}$$

An example is shown in Fig. S1 demonstrating that this limit is valid for $Pe < 1$.

2.2 Limit $Pe \to \infty$

To obtain the limit of Eq. (22) for $Pe \to \infty$ we write Eq. (22) in terms of $C = 4/Pe^2$ and determine the limit for $C \to 0$. The terms α, β in Eq. (22) can be linearized with respect to *C*. Inserting them into Eq. (22) gives

$$T_f - T_s = GPe\left[-\left(1 - \frac{1}{4}C\, e^{-Pe\left(1+\frac{1}{2}C\right)/G}\right)e^{-Pe\frac{1}{4}Cz} + \left(-\frac{1}{4}C\left(1-\frac{1}{4}C\right)e^{-Pe\left(1+\frac{1}{2}C\right)/G}\right)e^{Pe\left(1+\frac{1}{4}C\right)z} + 1\right] \tag{S10}$$

Allowing still for a finite term (*C Pe*), in the limit of $C \to 0$ Eq. (S10) turns into

$$T_f - T_s = G\frac{1-e^{-\frac{z}{Pe}}}{1/Pe} \tag{S11}$$

Substituting x = 1/*Pe* and applying the rule of L'Hôpital we get

$$\lim_{x\to 0}\left(T_f - T_s\right) = \lim_{x\to 0} Gze^{-zx} = Gz \tag{S12}$$

One can see in Fig. (S1) that this limit predicts ΔT_{max} values in very good agreement with Eq. (22) for $Pe > 100$.

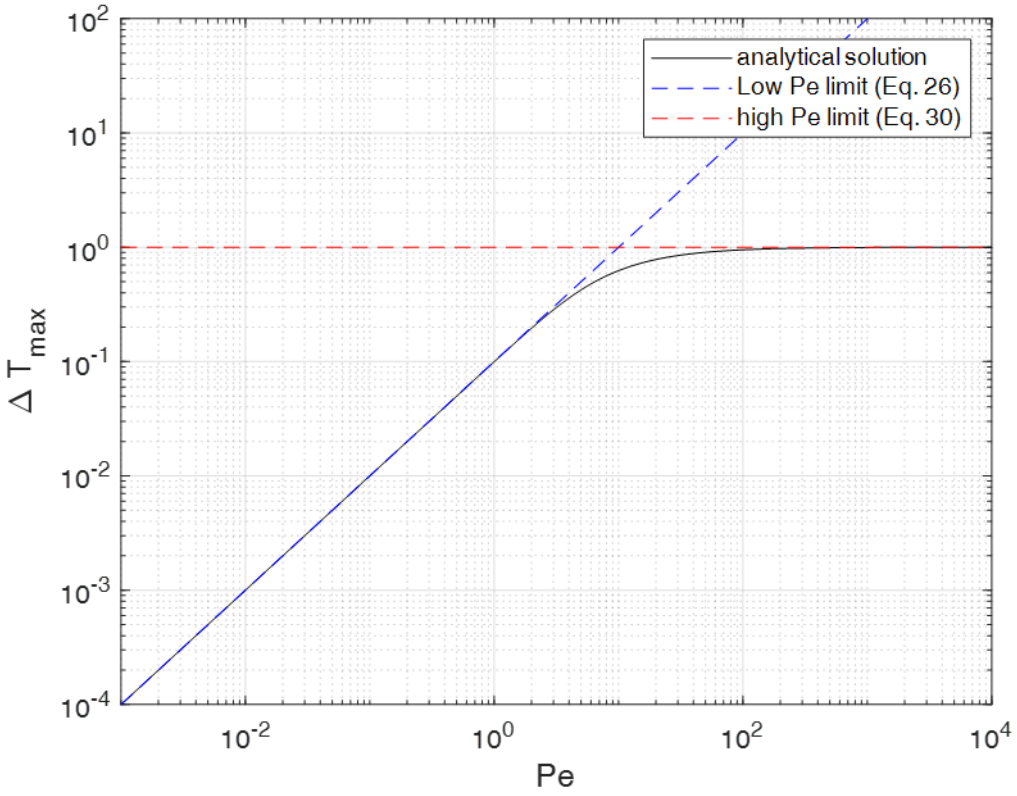

Figure S1. Comparison of the analytic solution Eq. (22) with the different limits derived in section 2.1 and 2.2. The black curve represents the analytic solution, the colored straight lines show the results in the high or low value limits of Eq. (26) to (30), respectively. G=0.1 was assumed.

3 Numerical programs

Here we give the Matlab routines used to calculate fluid to solid temperatures differences to generate figures like Fig.5.

Main program:

```
% Parent mfile for runing models. For every model it creates a
% new directory, in which key temperature results and outputfiles from the
% model solving are written. This code uses the function
% LTNEbasicdtpaper. This function solves non-thermal equilibrium two-phase
% flow (static matrix) and returns the fluid, solid temperatures and the
% temperature difference at top at the end of the run, as well as the
% maximum temperature difference at top recorded during the system
% evolution. Input parameters are :
% (Pe,phi0,H,delT,modelname,tmax,outputfactor)
% Pe: Péclet number
```

```
% phi0: Porosity (fluid volume fraction)
% H: Height of the domain
% delT: Initial temperature difference between top and bottom
% modelname: Name of the model
% tmax : Ending time
% outputfactor: Basic outputs come every multpl1*outputfactor steps for the first
% 1000*outputfactor time steps, and every multpl2*outputfactor steps
% afterwards. multpl1 and multpl2 have to be defined in the routine
% LTNEbasicdtpaper.m

clearvars
close all

nmodel = 1; % number of models to be run. If > 1 a loop has to be inserted
Pe = 1;
phi = 0.1;
H = 10;
delT = 1;
model = 'model1';
tmax = 100;
outputfactor = 1;

Results = zeros(nmodel,6);

mkdir('model1')
[Tftop,Tstop,dTtop,dTtopmax,kmax]=LTNEbasicdtpaper(Pe,phi,H,delT,model,tmax,outputf
actor);
Results(1,:) = [1,Tftop,Tstop,dTtop,dTtopmax,kmax];
save(['1' '_' 'values' '.txt'],'Results','-ascii')
```

Routine which is called by the above program:

```
% Function LTNEbasicdt to be used with a parent mfile in which the model to
% be run is defined. This function solves non-thermal equilibrium two-phase
% flow (static matrix) and returns the fluid, solid temperatures and the
% temperature difference at top at the end of the run, as well as the
% maximum temperature difference at top recorded during the system
% evolution. Input parameters are :
% Pe: Péclet number
% phi0: Porosity (fluid volume fraction)
% H: Height of the domain
% delT: Initial temperature difference between top and bottom
% modelname: Name of the model
% tmax : Ending time
% outputfactor: Basic outputs come every multpl1*outputfactor steps for the first
```

```
% 1000*outputfactor time steps, and every multpl2*outputfactor steps
% afterwards. multpl1 and multpl2 have to be defined in this routine

function
[Tftop,Tstop,dTtop,dTtopmax,tkmax]=LTNEbasicdtpaper(Pe,phi0,H,delT,modelname,tmax,o
utputfactor)

clf
ncolor=8;
cmap = parula(ncolor);
kcol =1;
dx = 1e-1; % Grid size
if H < 10; dx = min(dx,H/100);end
output = 1; %1=outputs, other : no output files

dt = 0.25*min(dx/Pe, dx^2)

% Prepare other parameters
prefixe = strcat('./',modelname,'/',modelname);
x = 0:dx:H;
nx = length(x);
t    = dt:dt:tmax;
nmax=tmax/dt+1;
tkmax = 0;


Tf = zeros(nx,1);
difTtop = [];

%Initial condition
for i = 1:nx
    Tf(i) = delT           -x(i)/H; % constant gradient
%     Tf(i) = 0;
end
Tm=Tf;
Tmnew =Tm;
Tfnew=Tf;

%Boundary conditions at start
Tm0 = delT;
Tf0 = delT;
Tm1 = 0;
Tf1 = 0;

Tmnew(1) = Tm0;
Tm(1)    = Tm0;

Tmnew(nx) = Tm1;
Tm(nx)    = Tm1;

Tfnew(1) = Tf0;
```

```
Tf(1) = Tf0;

Tfnew(nx) = Tf1;
Tf(nx) = Tf1;

%figure initial conditions
% figure;
%plot(x,Tfnew,'k',x,Tmnew,'k')
hold on

if output == 1
    Qsave = [x' Tfnew Tmnew];
    save([prefixe '_' num2str(0) '.txt'],'Qsave','-ascii')
end

dTtopmax=0;
kfirst = 0;
for k =2:length(t)
    for i = 2:nx
        % FTCS with upwind
        if i <nx
            Tmnew(i)=Tm(i)+dt*((Tm(i+1)-2*Tm(i)+Tm(i-1))/dx^2+...
                phi0*(Tf(i)-Tm(i)));
            Tfnew(i)= Tf(i) + dt*(-Pe*(Tf(i)-Tf(i-1))/dx +...
                (Tf(i+1)-2*Tf(i)+Tf(i-1))/dx^2  - (1-phi0)*(Tf(i)-Tm(i)));
        end
        if i == nx
% Top boundatry condition:
% finite flux as in paper
            Tmnew(i) = Tmnew(i-1)-dx/H; %constant flux condition at top (Neumann)
            Tfnew(i) = Tfnew(i-1)-dx/H;
% Zero flux
%             Tmnew(i) = Tmnew(i-1)-0*dx/H; %constant flux condition at top
(Neumann)
%             Tfnew(i) = Tfnew(i-1)-0*dx/H;
% Zero T
%             Tmnew(i) = 0;
%             Tfnew(i) = 0;
% Open with one-sided boundary condition for diffusion
%             Tmnew(i)=Tm(i)+dt*((1/A)*(Tm(i)-2*Tm(i-1)+Tm(i-2))/dx^2+...
%                 (phi0/(1-phi0))*(Tf(i)-Tm(i)));
%             Tfnew(i)= Tf(i) + dt*(-(Pe/A)*(Tf(i)-Tf(i-1))/dx +...
%                 (1/A)*(Tf(i)-2*Tf(i-1)+Tf(i-2))/dx^2  - (Tf(i)-Tm(i)));
% open with advection but diffusion switched off
%             Tmnew(i)=Tm(i)+dt*((1/A)*(0)/dx^2+...
%                 (phi0/(1-phi0))*(Tf(i)-Tm(i)));
%             Tfnew(i)= Tf(i) + dt*(-(Pe/A)*(Tf(i)-Tf(i-1))/dx +...
%                 (1/A)*(0)/dx^2  - (Tf(i)-Tm(i)));
% Robin
%             qtotf = Pe;
%             qtotm = 1/H;
```

```
%              Tmnew(i) = -(dx*qtotm - Tmnew(i-1));
%              Tfnew(i) = (dx*qtotf - Tfnew(i-1))/(dx*Pe-1);
        end
        dTtopmaxlast = dTtopmax;
        dTtopmax = max(dTtopmax,Tfnew(nx)-Tmnew(nx));
        if dTtopmax > dTtopmaxlast ; kmax = k; tkmax=t(k); end
    end
    if Pe > 99; multpl1 = 100; else; multpl1 = 200;end
    if Pe > 99; multpl2 = 4000; else; multpl2 = 2000;end
    if k < (1000*outputfactor+1)
        kmod = mod(k,multpl1*outputfactor);
        if kmod == 0
            if kfirst == 0;linw = 2;end
%            plot(x,Tfnew,'r',x,Tmnew,'b')
            plo = plot(x,Tfnew-Tmnew,'Color',cmap(kcol,:),'linewidth',linw);
            kfirst = 1;
            linw=1;
            kcol=kcol+1;
            if kcol>ncolor;kcol=1;end
            difTtop=[difTtop,Tfnew(nx)-Tmnew(nx)];
            if output ==1
                Qsave = [x' Tfnew Tmnew];
                save([prefixe '_' num2str(k*dt) '.txt'],'Qsave','-ascii')
            end
            t(k)
        end

    else
        kmod = mod(k,multpl2*outputfactor);
        if kmod == 0
%            plot(x,Tfnew,'r',x,Tmnew,'b')
            plo= plot(x,Tfnew-Tmnew,'Color',cmap(kcol,:));
                        kcol=kcol+1;
            if kcol>ncolor;kcol=1;end

           difTtop=[difTtop,Tfnew(nx)-Tmnew(nx)];
            if output == 1
                Qsave = [x' Tfnew Tmnew];
                save([prefixe '_' num2str(k*dt) '.txt'],'Qsave','-ascii')
            end
            t(k)

        end
    end

    drawnow

    Tf = Tfnew;
    Tm = Tmnew;
end
```

```
xlabel('z')
% ylabel('Tm, Tf')
ylabel('T_f - T_s')
grid on
box on

if output == 1
    save([prefixe '_' 'outputdata.txt'],'x','Tfnew','Tmnew','-ascii')
    print('-f1','-dpng',prefixe)
end

% close (1)

%Key values returned from function
Tftop = Tfnew(nx);
Tstop = Tmnew(nx);
dTtop = Tfnew(nx)-Tmnew(nx);
```

Main program for analytical solution. It can also be run directly after the numerical time evolution has been plotted by the above code. Then the analytical and numerical solutions are plotted into the same graph:

```
% program for drawing dTmax as a function of z for a given set of parameters.
% Parameters : Pe, dT, H (=1/G). Used to plot the analytic solution
% into the previousy plotted temporal evolution of (Tf - Ts) -curves
% This program uses the analytical solution for dTmax, and some
% limits, where the analytical solution is not solvable by matlab (large
% exponential exponents for example). The domains where limits have to be
% used instead of the analytical solution must be precised by the user
% (lines marked with %%%%%%%%%). In this version, only the limits for
% high Pe were used. Others can be added if needed.

clearvars

% Here the previously plotted figure must be open and the same parameters
% have to be given (phi not needed)
hold on
Pe = 1;
dT = 1;
H = 10;

z = [0:0.01*H:H];
Pestr = num2str(Pe);
Gstr = num2str(1/H);

for k = 1:length(z)
 if Pe < 30
        dTmax(k) = dTmaxcalc(Pe,dT,H,z(k));
 else
```

```
            [dTmax0(k),dTmax(k)] = dTmaxcalchighPe(Pe,dT,H,z(k));
 end
end

plot(z,dTmax,'r--','linewidth',2)
xlabel('z')
ylabel('dTmax')
box on
ylabel('T_f - T_s')
title(['Pe =' ,Pestr,',  G = ',Gstr])

```

Routines which is called by the above program:

```
% function for calculating the analytical value of dTmax. Parameters are
% the Péclet number Pe, the inital temperature difference between bottom and
% top dT, the domain size H (distance at which top boundary conditions are
% applied) and the distance from bottom z.

function [dTmax]=dTmaxcalc(Pe,dT,H,z)

r1 = (Pe-sqrt(Pe^2+4))/2;
r2 = (Pe+sqrt(Pe^2+4))/2;
PeG = Pe*dT/H;
alpha = PeG/(r1/r2*exp((r1-r2)*H) - 1);
beta = PeG/(r2/r1*exp((r2-r1)*H) - 1);
dTmax = alpha*exp(r1*z) + beta*exp(r2*z) + PeG;


% function for calculating the limit of dTmax analytical solution (see
% function dTmaxcalc), when Pe tends to infinity (high Pe values).
% dTmaxhighPe2 is mostly another way of writing dTmax analytical function,
% that allows easier numerical calculation of dTmax (smaller exponential
% exponents), and can be used for smaller Pe values than dTmaxhighPe.

function [dTmaxhighPe, dTmaxhighPe2]=dTmaxcalchighPe(Pe,dT,H,z)

r1 = (Pe-sqrt(Pe^2+4))/2;
r2 = (Pe+sqrt(Pe^2+4))/2;
dTmaxhighA = Pe*dT/H;
alpha = dTmaxhighA/(r1/r2*exp((r1-r2)*H) - 1);
dTmaxhighPe = dT*z/H;
dTmaxhighPe2 = alpha*exp(r1*z) +dTmaxhighA/(r2/r1)*exp(r2*(z-H)+r1*H)+ dTmaxhighA;
```