Journal of Geophysical Research, Earth Surface

Supporting Information for

Convection Cells with accumulating Crust: Models of Continent & Mantle Evolution

J. A. Whitehead

Department of Physical Oceanography, Woods Hole Oceanographic Institution, Woods Hole, MA, USA,

Mail: MS#21, Physical Oceanography Department, Woods Hole Oceanographic Institution, Woods Hole, MA 02543, USA **email: jwhitehead@whoi.edu**

Contents of this file

List of videos

Additional Supporting Information (Files uploaded separately)

The videos are directly connected to the figures and the figure captions can serve as video captions.

Introduction

List of Supplementary material

- Videos are:

2022JB025643-T-sup-0002-Movie SI-S01.mpg 3.7 MB Movie S1
2022JB025643-T-sup-0003-Movie SI-S02.mpg 4.8 MB Movie S2
2022JB025643-T-sup-0004-Movie SI-S03.mpg 4.9 MB Movie S3
2022JB025643-T-sup-0005-Movie SI-S04.mpg 802 KB Movie S4

2022JB025643-T-sup-0006-Movie SI-S05.mpg 7.3 MB Movie S5
2022JB025643-T-sup-0007-Movie SI-S06.mpg 5.3 MB Movie S6
2022JB025643-T-sup-0008-Movie SI-S07.mpg 606 KB Movie S7
2022JB025643-T-sup-0009-Movie SI-S08.mpg 2.8 MB Movie S8

Their captions are the figure captions in the text.

- A Matlab computer code "**CODEDouble.m**" follows. This makes all the Figures with adjustments of the constants and in some cases with more figure panels (In the code we have *CRa* negative unlike the text).
- **A run can be started. You simply set it up by typing "clear;nb=0;nvv=0" & then start the run.**
- To run, a second file **fps2.m** (a Poisson equation solver, enclosed) must be in the same folder as the matlab code.

```
tic
% CODE FOR DOUBLE COMPONENT CONVECTION 2-D FREE SLIP
% This has all the boundary conditions at the sides, top and bottom for
% a rectangular chamber. We use two matrix grids. The extent of the columns
% (depth z) is a multiple of 2 for high speed and the extent of the rows
% (length x) are a multiple of 2 times the depth.

% This grid is nested in a larger grid with an extra column around bottom
% and sides.  The outer grid cells have zero normal velocity through them
% and thus diffusion governs transport of heat and composition through
% them. Therefore there is no flux in the normal direction if the two
% values are set to the same value in that direction.
% The boundary conditions are zero diffusive flux of T and C out of the
% sides and zero diffusive flux of C along the bottom. A value
% of T is imposed along the top and bottom.

% Zero vorticity and zero streamfunction are set at the sides of
% the smaller grid. The streamfunction value is required at the periphery
% of the large grid for the calculation of speed at the edge of the small
% grid so that  value is determined with a Taylor series so speed is
% smooth. % The CODE defining the variables and integrating ahead one step
% in time is in lines 34-227 (123 lines of code). It requires a poisson
% equation solver to find the vorticity and streamfunction.
% Here fps2.m is used.  In my computer, that solver must be stored
% in the same folder.

% Values of parameters for good resolution up to Ra order 10^5 are CIG>0.025
% with a 65x257 grid and dt=2*10^-5. Otherwise fine scale instability occurs.
% See "A note of the supporting information and final code.doc".

% TO INITIATE THE PROGRAM ENTER clear;nvv=0;nb=0 in the command window &run
```

```
% Each step within the next two loops advances by the time dt. The
% innerloop advances nxp and it saves time by not updating the figures,
% which are updated for each step of the outer loop. he outer loop index
% nvv can be advanced again and again, The overall index nb starts and
% simply counts up.

%THIS particular program has internal heating with bottom insulation
% To have instead hot temperature along the bottom modify lines 185-189, 201-205

% Lines 37,38, 279, 280 and 300 produce a video. The are commented here.
% If you want to make the video (with a name NAME.avi), uncomment them.


%vidObj=VideoWriter('NAME.avi');
% open(vidObj);
%for nx=nx:nx
     for nvv=nvv+1:nvv+25
   for nxp=1:100
% time step
       dt=2*10^-5;
%advance of index nb and definition of time
nb=nb+1;
t=nb*dt;
% Lines 38 to 123 assign initial conditions and set up the grids and
% various variables.  It is a one-time loop.

if nb==1
% Value of Rayleigh number and Concentration Ra, CRa is negative unlike the text)
Ra=10000;
CRa=-12000;
% CIG is inverse Lewis number, h is constant internal heat production,
% ho is coefficient of concentration dependent heat production
% timeon is the time that lightness starts, gamma is "lightness" decay rate
CIG=0.01;
H=0;
ho=0;
timeon=0.2;
gamma=0;

% Pr is infinite, now the bottom temperature=1;
TBOT=1;
% box s1ze
N=65;
M=257;
Lz=1;
Lx=(M-1)/(N-1);
mf=1;
```

```
% initial conditions everywhere
% the real temperature concentration and steamfunction fields
T=zeros(N,M);
CT=zeros(N,M);
S=zeros(N,M);
zeta = zeros(N,M);
rhs  = zeros(N,M);
% we use expanded grid fields to have strictly zero side diffusion
TB=zeros(N+1,M+2);
TP=zeros(N+1,M+2);
TM=zeros(N+1,M+2);
TS=zeros(N+1,M+2);
CTP=zeros(N+1,M+2);
CTM=zeros(N+1,M+2);
CTS=zeros(N+1,M+2);
CTB=zeros(N+1,M+2);
SB=zeros(N+1,M+2);
ZS=zeros(N+1,M+2);
ZM=zeros(N+1,M+2);
ZB=zeros(N+1,M+2);
ZP=zeros(N+1,M+2);

% Diagnostic variables
Tflowmax=0;
smax=0;
time(1)=0;
vv=0;
vxx=0;
vxxx=0;
vxxxx=0;

% Index vectors for the regular grid
n=[2:N-1];
m=[2:M-1];
np=n+1;
nm=n-1;
mp=m+1;
mm=m-1;
%index vectors for the bigger grid
nbb=[2:N];
mb=[2:M+1];
nbp=nbb+1;
nbm=nbb-1;
mbp=mb+1;
mbm=mb-1;

% lengths
dx = Lx/(M-1);
```

```
dz = Lz/(N-1);
%running vaariables
Raa=0;
CRaa=0;
Nua=0;
Smax=0;



if dx~= dz
 disp(' grid cells do not have dx=dz ')
 break
end

% INITIAL TEMPERATURE and concentration
TB(:,:)=.5;
TB(10,10)=1;
TB(10,129)=1;
TB(10,246)=1;
TB(N+1,1:M+2)=0;
TM=TB;
CTB(:,:)=1;
CTM=CTB;
end
% End of the loop that sets up the initial conditions

time(nvv+1)=time(nvv)+100*dt;
% STREAMFUNCTION in the outer big grid = inner (real) grid
SB(2:N+1,2:M+1)=S;

% smoothed big grid streamfunction around the edges
SB(1,1:M+2)=(2*SB(2,1:M+2))-SB(3,1:M+2);
SB(1:N+1,1)=(2*(SB(1:N+1,2)))-SB(1:N+1,3);
SB(1:N+1,M+2)=(2*(SB(1:N+1,M+1)))-SB(1:N+1,M);
%STREAMFUNCTION in the outer big grid is the same as the value of the

% streamfunction of inner part of big = streamfunction of the real grid
SB(nbb,mb)=S(nbm,mbm);
% reset internal heating distribution
h=H+ho*(1-CTB);
%Integrate the time-dependent temperature distribution in two steps
%first step in time integration
TS(nbb,mb)=TM(nbb,mb)+(dt*h(nbb,mb))+(2*dt*((((N-1)^2)*(TB(nbm,mb)-
(2*TB(nbb,mb))+TB(nbp,mb)))+(((N-1)^2)*(TB(nbb,mbm)-
(2*TB(nbb,mb))+TB(nbb,mbp)))))+(((2*dt*(N-1)*(N-1))/4)*(((SB(nbb,mbp)-
SB(nbb,mbm)).*(TB(nbp,mb)-TB(nbm,mb)))-((SB(nbp,mb)-SB(nbm,mb)).*(TB(nbb,mbp)-
TB(nbb,mbm)))));
```

```
% insulating bc on TS both sides
TS(2:N+1,1)=TS(2:N+1,2);
TS(2:N+1,M+2)=TS(2:N+1,M+1);

% insulating bc on TS bottom
%TS(1,2:M+1)=TS(2,2:M+1);
%OR---bottom Temp imposed
TS(1,2:M+1)=TBOT;
TS(2,2:M+1)=TBOT;

% temperature along the top
TS(N+1,1:M+2)=0;

% second step in time integration.
TP(nbb,mb)=TB(nbb,mb)+(dt*h(nbb,mb))+(.5*dt*((((N-1)^2)*(TB(nbm,mb)-
(2*TB(nbb,mb))+TB(nbp,mb)))+(((N-1)^2)*(TB(nbb,mbm)-
(2*TB(nbb,mb))+TB(nbb,mbp)))))+(((.5*dt*(N-1)*(N-1))/4)*(((SB(nbb,mbp)-
SB(nbb,mbm)).*(TB(nbp,mb)-TB(nbm,mb)))-((SB(nbp,mb)-SB(nbm,mb)).*(TB(nbb,mbp)-
TB(nbb,mbm)))))+(.5*dt*((((N-1)^2)*(TS(nbm,mb)-(2*TS(nbb,mb))+TS(nbp,mb)))+(((N-
1)^2)*(TS(nbb,mbm)-(2*TS(nbb,mb))+TS(nbb,mbp)))))+(((.5*dt*(N-1)*(N-1))/4)*(((SB(nbb,mbp)-
SB(nbb,mbm)).*(TS(nbp,mb)-TS(nbm,mb)))-((SB(nbp,mb)-SB(nbm,mb)).*(TS(nbb,mbp)-
TS(nbb,mbm)))));

% insulating bc on TS both sides
TP(2:N+1,1)=TP(2:N+1,2);
TP(2:N+1,M+2)=TP(2:N+1,M+1);

% insulating bc on TP bottom
%TP(1,2:M+1)=TP(2,2:M+1);
%OR---bottom Temp imposed
TP(1,2:M+1)=TBOT;
TP(2,2:M+1)=TBOT;

% temperature along the top
TP(N+1,1:M+2)=0;
%iterate
TM=TB;
TB=TP;

%Integrate the Concentration (boundary conditions can differ from T)
%first step in time integration
CTS(nbb,mb)=CTM(nbb,mb)+(dt*gamma*(1-CTB(nbb,mb)))+(2*dt*CIG*((((N-
1)^2)*(CTB(nbm,mb)-(2*CTB(nbb,mb))+CTB(nbp,mb)))+(((N-1)^2)*(CTB(nbb,mbm)-
(2*CTB(nbb,mb))+CTB(nbb,mbp)))))+(((2*dt*(N-1)*(N-1))/4)*(((SB(nbb,mbp)-
SB(nbb,mbm)).*(CTB(nbp,mb)-CTB(nbm,mb)))-((SB(nbp,mb)-SB(nbm,mb)).*(CTB(nbb,mbp)-
CTB(nbb,mbm)))));
%insulating bc on CTS both sides and CTS=1 on bottom
CTS(2:N+1,1)=CTS(2:N+1,2);
```

```
CTS(2:N+1,M+2)=CTS(2:N+1,M+1);
CTS(2,2:M+1)=1;
CTS(1,2:M+1)=1;
% concentration  along the top changed from 1 to 0 after t=timeon
CTS(N+1,1:M+2)=1;
if time(nvv)>timeon
   CTS(N+1,1:M+2)=0;
end

%second step in time integration
CTP(nbb,mb)=CTB(nbb,mb)+(0.5*dt*gamma*(1-CTB(nbb,mb)))+(0.5*dt*gamma*(1-
CTS(nbb,mb)))+(.5*dt*CIG*((((N-1)^2)*(CTB(nbm,mb)-(2*CTB(nbb,mb))+CTB(nbp,mb)))+(((N-
1)^2)*(CTB(nbb,mbm)-(2*CTB(nbb,mb))+CTB(nbb,mbp)))))+(((.5*dt*(N-1)*(N-
1))/4)*(((SB(nbb,mbp)-SB(nbb,mbm)).*(CTB(nbp,mb)-CTB(nbm,mb)))-((SB(nbp,mb)-
SB(nbm,mb)).*(CTB(nbb,mbp)-CTB(nbb,mbm)))))+(.5*dt*CIG*((((N-1)^2)*(CTS(nbm,mb)-
(2*CTS(nbb,mb))+CTS(nbp,mb)))+(((N-1)^2)*(CTS(nbb,mbm)-
(2*CTS(nbb,mb))+CTS(nbb,mbp)))))+(((.5*dt*(N-1)*(N-1))/4)*(((SB(nbb,mbp)-
SB(nbb,mbm)).*(CTS(nbp,mb)-CTS(nbm,mb)))-((SB(nbp,mb)-SB(nbm,mb)).*(CTS(nbb,mbp)-
CTS(nbb,mbm)))));
%insulating bc on CTP both sides and =1 bottom
CTP(2:N+1,1)=CTP(2:N+1,2);
CTP(2:N+1,M+2)=CTP(2:N+1,M+1);
CTP(1,2:M+1)=1;
CTP(2,2:M+1)=1;
% Concentration along the top
CTP(N+1,1:M+2)=1;
if time(nvv)>timeon
   CTP(N+1,1:M+2)=0;
end
%iterate
CTM=CTB;
CTB=CTP;

%Transform T from the core of the big grid to the real grid
T(1:N,1:M)=TB(2:N+1,2:M+1);
%corners
CTB(1,M+2)=CTB(1,M+1);
CTB(1,1)=CTB(1,2);

%Transform C from the core of the big grid to the real grid
CT(1:N,1:M)=CTB(2:N+1,2:M+1);

%Fast integration of vorticity and streamfunction using fps2 program
rhs(n,m)=(-(dx^2)*Ra*(N-1)/2)*(T(n,mp)-T(n,mm))+(-(dx^2)*CRa*(N-1)/2)*(CT(n,mp)-CT(n,mm));
zeta = fps2(rhs);

% solve for streamfunction S
S=-fps2((dx^2)*zeta);
```

```
% SURFACE ELEVATION; EQUIVALENT TO THE TOP ROW PRESSURE pss
% pressure gradient ssm from a Taylor series is integrated

 ssm1(m)=S(64,mp)-2*S(64,m)+S(64,mm)-S(63,mp)-S(63,mm)+S(63,m)+2*S(62,m)-S(61,m);
for msumm=1:M-1
pss1(msumm)=sum(ssm1(1:msumm));
end
pss1m=mean(pss1);
pss1=-10*(pss1m-pss1);
   end
% This is the end of the loop that advances nxp times.
% NOW RESULTS ARE PLOTTED
% PLOTTING RESULTS
v=[1 2 3 4 5 6 7 8 9]/10;
  v5=[1 3 5 7 9]./10;
  v10=[1 3 5 7 9 11 13 1 ]./10;
  vp=[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]./20;
 vy=[.01:.01:.7];
vxx(nvv)=max(max(S));
vxxx(nvv)=min(min(S));
vxxxx=max(max(abs(S)));
vv(nvv)=vxxxx;
Hflow(nvv)=(N-1)*mean((T(N-1,:)));
 subplot(3,1,1)
 plot(pss1,'k','LineWidth',3)
 hold on
 ylabel('ELEVATION (km)','fontsize',20)
 title('TOPOGRAPHY','fontsize',24)
xlabel('Continent yellow; Temperature contours red; Clockwise circulation green,
counterclockwise blue','fontsize',12)
hold off
subplot(3,1,2)
contour(CT,vy,'y','LineWidth',3)
set(gca,'plotboxaspectratio',[Lx,Lz,1])
hold on
contour(CT,[.699 .7],'k','LineWidth',2.5)
contour(S,vxx(nvv)*v5,'b','LineWidth',1)
contour(S,vxxx(nvv)*v5,'g','Linewidth',2)
contour(T,v10,'r','LineWidth',3)
hold off

xlabel('HOT CORE-MANTLE BOUNDARY','fontsize',24)
ylabel('DEPTH','fontsize',24)
title('COLD EARTH SURFACE','fontsize',24)

subplot(3,1,3)
```

```
plot(time(1:nvv),vv(1:nvv),'k','LineWidth',2)
xlabel('t','fontsize',14)
ylabel('MAXIMUM SPEED','fontsize',20)
hold off

%TO MAKE A VIDEO, UNCOMMENT THE NEXT TWO LINES
%   currFrame =  getframe(gcf);
%   writeVideo(vidObj,currFrame);
time(nvv+1)
pause(.000001)
end
hold off

CRa
Ra

toc
   beep
 % close(vidObj);
```

CODE---- fps2.m

```
function u=fps2(p)


q = p;

% stuff below not need for homogeneous boundary conditions
% Fold the boundary into source terms.
% theFactor = -1;    % Boundary-value scheme.
%for i = 1:2
%    q = q.';
%    q(2:end-1, 2:end-3:end-1) = ...
%        q(2:end-1, 2:end-3:end-1) + ...
%            theFactor * q(2:end-1, 1:end-1:end);
%end

% Extract the interior.

q = q(2:end-1, 2:end-1);

theSign = -1;

% Odd-symmetry, sine-transform scheme.

[m, n] = size(q);
q = [zeros(m, 1) q zeros(m, 1) theSign*fliplr(q)];
q = [zeros(1, 2*n+2); q; zeros(1, 2*n+2); theSign*flipud(q)];

% Fast Poisson Transform on square cell grid (dx^2 factor
% already multiplies p).
```

```
[mm, nn] = size(q);
i = (0:mm-1).' * ones(1, nn);
j = ones(mm, 1) * (0:nn-1);
weights = 2 * (cos(2*pi*i/mm) + cos(2*pi*j/nn) - 2);
weights(1, 1) = 1;
pp = ifft2(fft2(q) ./ weights);

if isreal(q), pp = real(pp);

end

% Retain relevant piece.

u = p;
u(2:end-1, 2:end-1) = pp(2:m+1, 2:n+1);
```