NUMERICAL MODELS

As discussed in the text, we split the conceptual model for eclogite-driven subsidence into two parts: a mechanical part, and thermo-chemical a part. The mathematical system is similarly split into two sets of coupled equations which we solve with finite difference and finite element methods, respectively. The two models are described below, along with necessary details to reproduce all relevant results and figures.

Note that owing to limitations, the code referenced below is included as pages 6 – 21 of this supplemental pdf. To successfully reproduce the model results, you'll need to copy-paste these sections into text files with the given file names. The files are: 'flex.py', 'matplotlibrc', 'latent_heat.tfml', and latent_heat.shml'. They are separated by comments below. Documentation on each in included in the following text.

FLEXURE

Eclgite-induced subsidence is discussed at length in the text. Mechanically, it is relatively straightforward, being basically an isostatic adjustment filtered through an elastic plate.

Conceptually, the steps are as follows:

- Lava flows into the CCBD.
- Added weight of new basalt is filtered through the elastic plate to some region of lower crust.
- Some rocks that had been close to garnet stability conditions are able to eclogitize.
- Eclogitization densifies, and thus contracts the lower crust.
- Contraction is elastically filtered back through the crust to the surface, where it is expressed as subsidence.

In modeling the system, we treat these steps as follows:

- We assume that the pressure of new deposition is transmitted hydrostatically to the lower crust.
- Thus, a volume of lower crust that is exactly the same dimensions, *V*, as the new lava deposit is converted to eclogite.
- Eclogitization decreases the volume of this lower crustal lens to $V*\rho_b/\rho_e$, where ρ_b and ρ_e are the density of basalt and eclogite, respectively.
- We expect this contraction will likewise create $(1 – \rho_b/\rho_e)*V$ of total subsidence.
- This is equivalent to applying a load of $[\rho_m (1 – \rho_b/\rho_e) g]$ Pascals to the plate, where ρ_m is the density of mantle, and g is gravity.
- We therefore model the contraction as a loading force applied to a buoyantly-supported, thin elastic plate (Equation 1).

We vary elastic thickness in the suite models represented in figure 3. All other properties are held constant. Density of basalt, mantle, and eclogite are fixed at 3000, 3300, and 3370 $kg.m^{-3}$ respectively (Hacker, 2013). Young's modulus is 7e10 Pa (Li et al., 2004), and Poisson's ratio is 0.25 (Stüwe, 2007).

Reproducing the Flexure Model

We solve equations 1 and 2 with a finite difference approximation, written in Python. The model, and all associated code to reproduce Figure 3 is in the file `flex.py`, included with this supplement. In order to run the model, you must first have `Python` installed (tested with versions 2.7 and 3.6), and modules `scipy` (tested with version 1.1.0) and `matplotlib` (tested with version 2.2.2). Other versions are not guaranteed to work, but significant changes in these libraries are infrequent, and other versions very well may work. Both module dependencies come standard with most scientific Python distributions, and are packaged in all major Linux distributions. The best way to install the exact versions is through the cross-platform `pip` package manager:

```
pip install scipy=1.1.0 matplotlib=2.2.2
```

With the above software dependencies satisfied, the model can be run by executing the script:

```
python ./flex.py
```

This will generate Figure 3 as shown in the text, and save it as a file (Figure3.tiff) in the working directory.

LATENT HEAT

Latent heat release during eclogitization inhibits further reaction. The reaction's positive Clapeyron slope at high temperature means that increased temperature tends to favor spinel stability. Over time, the latent heat diffuses away, permitting further eclogitization. The crux of the problem is a competition between the rate of latent heat release and thermal diffusion. The system is illustrated in Figure S1, with variables corresponding to those used in the text.

Our governing equations are defined in the main text, equations 3 and 4. We nondimensionalize the equations before solving to reduce the number of free parameters. We first parameterize temperature as a perturbation to a "reference" geotherm, G: $T = G z + T_0 + T^*$; and describe the Clapeyron slope in pressure, as a function of temperature: $P_c = \rho \Delta S T + \rho g z_0$. We scale the system by a Péclet number, $Pe_r = \boldsymbol{U}^3/[\boldsymbol{\Delta S G \alpha}]$, with variables defined in the main text:

$$\rho' C_p'(\partial T'/\partial t' + u' \partial T'/\partial z') - Pe_r^{-1} \partial/\partial z'(\rho' C_p' \alpha' \partial T'/\partial z') = \rho'(G' z' + T_0' + T') \Delta S' (\partial X'/\partial t' + u' \partial X'/\partial z') \quad \text{(S1)}$$

$$\partial X'/\partial t' = A' \exp(- E_a'/[R' (G' A' + T_0' + T')]) (0.5 + 0.5 \tanh[(z' - z_0' - (G' z' + T_0' + T'))/(\delta_p)] - X') \quad \text{(S2)}$$

where primed variables represent the nondimensional equivalents of those defined in the text.

We solve equations S1 and S2 using the TerraFERMA finite element framework (Wilson et al., 2016), closely based on the FeniCS project (Logg et al., 2012). This allows us to define our entire model in two, self-contained, and portable parameter files that are included as part of this supplement (latent_heat.tfml, and latent_heat.shml). This approach is highly reproducible, and our code may be easily audited by any interested reader. Details on how to do so are described below.

Results are shown in Figure S2 for a range of Péclet numbers, and demonstrate that for Pe_r>>1, the eclogite conversion is spread over a wide range of depths, and rocks may never fully metamorphose within the model domain. However, for realistic geologic scenarios, Pe_r<<1. At low Péclet numbers the model converges to the limiting case where heat diffuses infinitely quickly. That is, latent heat released by eclogitization is diffused away much faster than new basalt is advected through the system.

Reproducing the Latent Heat Model

TerraFERMA is available under a free and open source license at https://terraferma.github.io/. The simplest method to reproduce Figure S2 is to run TerraFERMA with Docker, rather than installing it locally. You can get Docker from https://docs.docker.com/engine/getstarted/step_one/.

Once you have installed Docker, `cd` to the directory containing the `.tfml` and `.shml` files, and run the model with the following command:

```
docker run -v "$PWD:/home/tfuser/shared" --rm terraferma/dev \
    'source /etc/profile; source .profile; cd shared/; tfsimulationharness --test latent_heat.shml' \
    -u tfuser
```

This will start a Docker image with TerraFERMA pre-installed and configured. It will then compile and run the model. It will place the resulting plots in the current directory.

The flags in the above command tell docker to share the current directory from your computer with the container (`-v "$PWD:/home/tfuser/shared"`) and to clean up the container when the model is done running (`--rm`). The argument (`terraferma/dev`) is the name of the container. If the TerraFERMA

container is not already on your system (which it probably isn't if you're following these instructions for the first time), Docker will automatically download it. The long line in single quotes is the actual command that will execute inside the container. It initializes some UNIX environment variables (`'source /etc/profile; source .profile'`), moves to the shared directory (`'cd shared/'`), and finally compiles and executes the TerraFerma model, including postprocessing 'tests' that generate figures (`'tfsimulationharness --test latent_heat.shml'`). The final flag tells docker to run the above command as the user `'tfuser'` in the container environment (`-u tfuser`).

Figure S2 was generated with terraferma/dev container v1.0, build code: bhpnedhtezj48dcagpzsdye.

Modifying the Latent Heat Model

TerraFERMA options files (`.tfml` and `.shml`) are in XML format, and meant to be read by the SPUD options processing system (Ham et al, 2009). Because the files are plain text, it is possible to modify them with a text editor, but for any significant changes, the best way to modify the model is with SPUD's official GUI, *Diamond*. More information can be found on running Diamond on the TerraFERMA wiki (https://github.com/TerraFERMA/TerraFERMA). If one only wants to audit the equations, the relevant parameters can be found in lines 223 – 252 of the file `latent_heat.tfml`, and may be checked with a text editor.

REFERENCES CITED

Hacker, B.R., 2013, Eclogite formation and the Rheology, Buoyancy, Seismicity, and H2O Content of Oceanic Crust: Subduction Top to Bottom, doi: 10.1029/GM096p0337.

Ham, D.A., Farrell, P.E., Gorman, G.J., Maddison, J.R., Wilson, C.R., Kramer, S.C., Shipton, J., Collins, G.S., Cotter, C.J., and Piggott, M.D., 2008, Spud 1.0: generalising and automating the user interfaces of scientific computer models: Geoscientific Model Development Discussions, v. 1, p. 125–146, doi: 10.5194/gmdd-1-125-2008.

Li, F., Dyt, C., and Griffiths, C., 2004, 3D modelling of flexural isostatic deformation: Computers & Geosciences, v. 30, p. 1105–1115, doi: 10.1016/j.cageo.2004.08.005.

Logg, A., Mardal, K.-A., and Wells, G. (Eds.), 2012, Automated Solution of Differential Equations by the Finite Element Method: Berlin, Heidelberg, Springer Berlin Heidelberg, Lecture Notes in Computational Science and Engineering, v. 84, http://link.springer.com/10.1007/978-3-642-23099-8 (accessed August 2016).

Stüwe, K., 2007, Geodynamics of the Lithosphere: An Introduction: Springer Science & Business Media, 497 p.

TerraFERMA: The Transparent Finite Element Rapid Model Assembler, 2018, TerraFERMA, https://github.com/TerraFERMA/TerraFERMA (accessed March 2018).

Wilson, C.R., Spiegelman, M., and van Keken, P.E., 2016, TerraFERMA: The Transparent Finite Element Rapid Model Assembler for multiphysics problems in Earth sciences: Geochemistry, Geophysics, Geosystems, p. n/a-n/a, doi: 10.1002/2016GC006702.

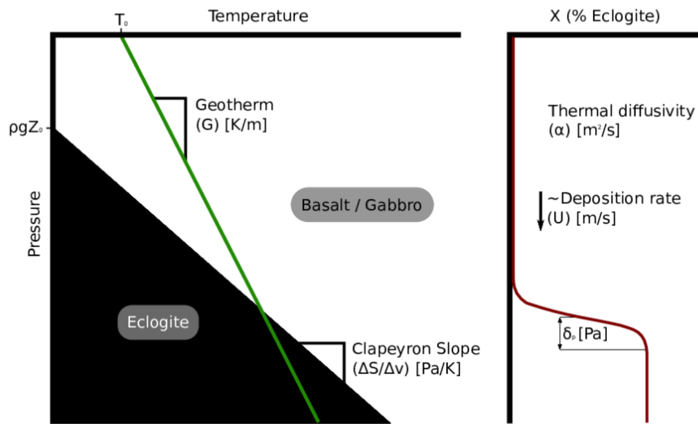

Figure S1. Conceptual 1D model of surface-load induced eclogitization of lower crust. The background geotherm is advected down at the rate of surface loading. Simultaneously minerals crossing the coexistence curve into the eclogite stability field release latent heat, modifying the thermal profile and inhibiting further eclogitization.

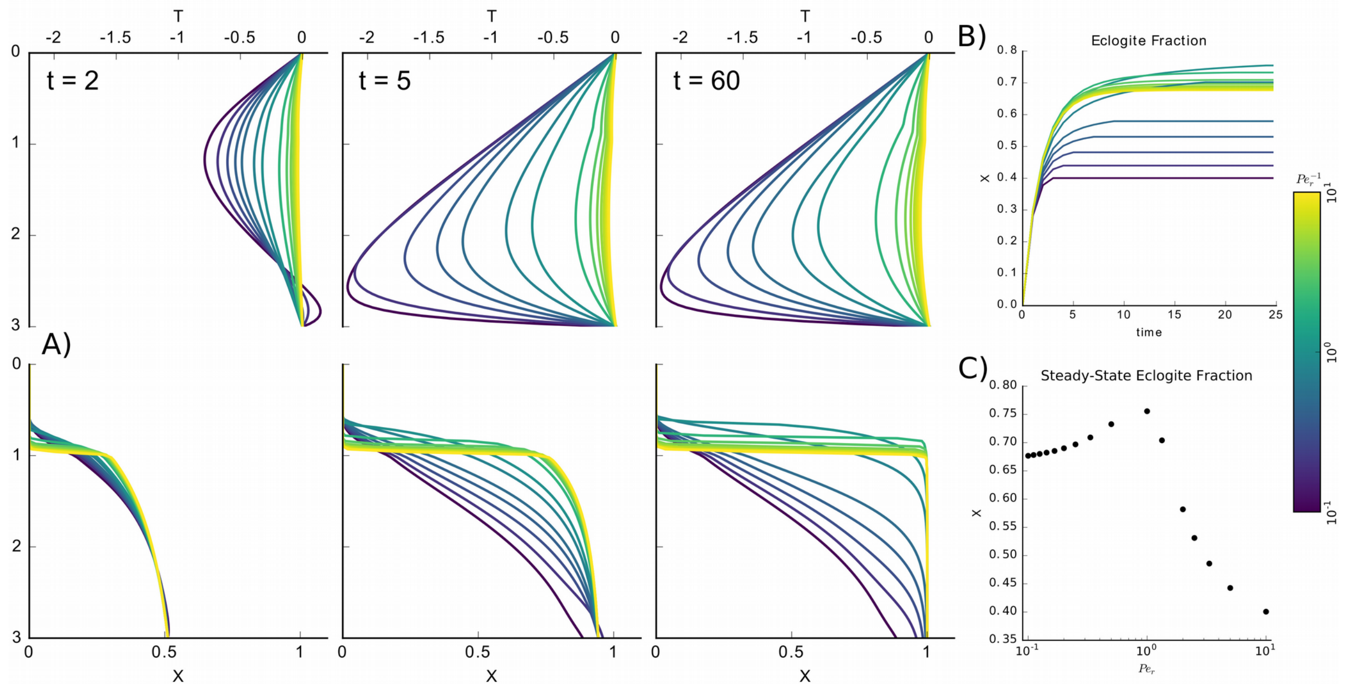

Figure S2. Solutions to equation S1 and S2 for various Péclet numbers over time until steady state. Thermal boundary conditions are imposed at the top and bottom of the model, forcing the temperature to start and end at the reference geotherm. Eclogite is constrained to zero at the top of the model. Temperature is initialized to match the reference geotherm, and eclogite is initialized to zero everywhere.

A: Top row shows nondimensional temperature perturbations over time for models with a range of Péclet numbers. Lighter colors represent low Péclet values. We can see that the low Péclet system stays close to the reference geotherm throughout the model run. High Péclet models diverge from the reference geotherm, cooling due to cold upper crust advecting to depth more quickly than heat can diffuse to reestablish the geotherm. There is a noticeable kink in the temperature solution for the intermediate Péclet systems where eclogite first begins to form.

The bottom row shows eclogite fraction over time for the same models as in the top row. All models start with no eclogite, and develop eclogite over time. The low Péclet systems quickly move to establish a equilibrium eclogite profile, with the deeper, hotter parts achieving equilibrium more quickly owing to the thermal dependence of reaction rate in equation S2.

B: Integrated eclogite fraction for each model over time.

C: Steady-state integrated eclogite fraction as a function of Péclet number. We can see that for very low Pe_r, this value trends towards the equilibrium value of 66%(because the model is set up such that 2/3 of the model domain is below the equilibrium phase-transition depth). As Pe_r increases, the total eclogite fraction increases, because advective cooling shallows the depth to eclogite stability. At very high Pe_r, rocks advect all the way through the model domain without fully reacting, represented by a decrease in total eclogite within the model. There is a peak at Pe_r=1 where these competing effects trade off most efficiently for eclogite formation. In this case the integrated eclogite fraction is >75%. It's worth noting that for any reasonable values of $\boldsymbol{U}$, $\boldsymbol{\Delta S}$, $\boldsymbol{G}$, and $\boldsymbol{\alpha}$, Pe_r is very very low. Only the low Pe_r limit is applicable to geologic problems, including eclogite-driven subsidence.

```
##
## Begin file: flex.py
##

#!/usr/bin/env python

'''
This file is provided as a supplement to the paper Perry-Houts, J., and Humphreys, E., 2018, Eclogite-driven
subsidence of the Columbia Basin caused by deposition of Columbia River Basalt: Geology.

It calculates the subsidence expected for an elastic plate with lower-crustal eclogitization owing to added pressure
of basalt deposition at the surface. See the paper, and supplemental text for more details.

This script has been tested with Python versions 2.7 and 3.6. It requires scipy, and matplotlib, and has been tested
with verisons 1.1.0 and 2.2.2 respectively. These dependencies can be installed with Python's package manager, pip, by
the following:
    pip install scipy=1.1.0 matplotlib=2.2.2
and the script can then be executed with:
    python ./flex.py
This should produce a file called Figure3.tiff in the working directory that corresponds exactly to Figure 3 in the
text.

Note: There should be a file called matplotlibrc also be present in the working directory (included as part of the
supplementary material). If not the formatting of the resulting figure may be a bit off.
'''

import os, sys
pipdir=os.path.expanduser("~")+'/.local/lib/python'+sys.version[:3]+'/site-packages'
if os.path.isdir(pipdir):
    sys.path.insert(0, pipdir)

import pickle
import pylab as pl
import scipy
import scipy.interpolate
import scipy.sparse.linalg
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import matplotlib.patheffects

if pl.mpl.__version__ != '2.2.2':
    print('\n  Warning: This script is only tested with Matplotlib version 2.2.2, but you are using version '
            + pl.mpl.__version__ + '\n  from source ' + pl.mpl.__file__)
    print('  This might produce unexpected results (or might be just fine, who knows).')
    print('  You can install version 2.2.2 with pip:')
    print('    pip install matplotlib=2.2.2\n')

if scipy.__version__ != '1.1.0':
    print('  Warning: This script is only tested with Scipy version 1.1.0, but you are using version '
            + scipy.__version__ + '\n  from source ' + scipy.__file__)
    print('  This might produce unexpected results (or might be just fine, who knows).')
    print('  You can install version 1.1.0 with pip:')
    print('    pip install scipy=1.1.0\n')


'''
Some generally useful functions:
'''
def cat(M,reverse=False):
    ''' Concatenate an array with its self '''
    sign = -1 if reverse else 1
    return pl.concatenate((sign*M[::-1],M))

def dg(k,val,N):
    ''' Construct sparse NxN matrix with non-zeros on diagonals
        defined by k, and values for those diagonals defined
        in val. ex: dg([-1,2],[-7,4],5) returns a 5x5 matrix with
        -7 on the first diagonal to the lower left, and 4 on the second
        diagonal to the top right.'''
    d = (pl.ones((len(k),N)).T*pl.array(val)).T
    return scipy.sparse.csr_matrix(scipy.sparse.dia_matrix((d, pl.array(k)), shape=(N,N)))

'''
A class that represents a flexural model. It keeps the model
execution and parameters and everything self-contained so that
later on all you need is a Model object:
    m = Model()
and you can run models by assembling a new system:
    m.assemble(Te=..., E=..., ...)
and execute it:
    m.run()
You can then re-use the model object with another call to assemble, etc.
This way all of the static matrices don't need to be reinitialized each time.
'''
```

```
class Model:
    # Densities from Hacker, 2013
    rho_e = 3370.0
    rho_m = 3300.0
    rho_b = 3000.0

    def __init__(self, N=50000, rmin=1e-6, rmax=10e6):
        # Discretization
        self.N = N
        self.R = pl.linspace(rmin, rmax, self.N)
        self.dr = abs(self.R[1]-self.R[0])
        # Construct cartesian FD matrices
        self.D0 = dg([0],             [1.0], self.N)
        self.D1 = dg([-1,1],          [-1.0,  1.0], self.N)/(2*self.dr)
        self.D2 = dg([-1, 0, 1],      [ 1.0, -2.0,  1.0], self.N)/(self.dr**2)
        self.D3 = dg([-2,-1, 1, 2],   [-1.0,  2.0, -2.0,  1.0], self.N)/(2*self.dr**3)
        self.D4 = dg([-2,-1, 0, 1, 2], [ 1.0, -4.0,  6.0, -4.0,  1.0], self.N)/(self.dr**4)
        # Construct radial fourth-derivative from cartesian matrices
        R_inv = scipy.sparse.csr_matrix(scipy.sparse.dia_matrix((1.0/self.R, pl.array([0])), shape=(self.N,self.N)))
        self.RD4 = self.D4 + 2*R_inv*self.D3 - (R_inv**2)*self.D2 + (R_inv**3)*self.D1

    # Boundary constraint functions
    def bc(self, kind, k, val, tied=1):
        if kind == 'dirichlet':
            ''' Set row k in LHS = identity,
                and column k in RHS = val.
                tied variable is ignored. '''
            self.A[k,:] = scipy.sparse.csr_matrix(pl.zeros(self.N))
            self.A[k,k] = 1.0
            self.b[k] = val
        elif kind == 'neumann':
            self.A[k,:] = scipy.sparse.csr_matrix(pl.zeros(self.N))
            self.A[k,k] = -1.0
            self.A[k,k+tied] = 1.0
            self.b[k] = val*tied*self.dr
        else:
            assert False, 'Boundary condition type must be of kind "neumann" or "dirichlet"'

    # Assemble stiffness matrix
    def assemble(self, E=0.7e11, v=0.25, Te=10e3):
        self.E = E
        self.v = v
        self.Te = Te
        # Assemble stiffness matrix
        self.A = E*Te**3/(12.0*(1.0-v**2)) * self.RD4 + (self.rho_m - self.rho_b)*9.8*self.D0

    # Run the experiment
    def run(self, num_flows=18, init_basin=(lambda r: -85*pl.exp(-r**2/10e3**2))):
        ## Initial basin geometry
        self.basins = [init_basin(self.R)]

        for flow in range(num_flows):
            # Contracting H meters of basalt to eclogite decreases its thickness to rho_b/rho_e*H,
            # creating a depression (1-rho_b/rho_e)*H meters deep.
            # The same thing could be accomplished by loading the crust with (rho_m*(1-rho_b/rho_e)) kg/m^2.
            load = self.rho_m*(1-self.rho_b/self.rho_e) * self.basins[-1] * 9.8

            self.b = load
            self.b[pl.where(self.b>0)] = 0.0
            # Boundary conditions
            self.bc('dirichlet', -1,0)
            self.bc('neumann', 0,0,1)
            # Solve
            self.basins.append(scipy.sparse.linalg.spsolve(self.A,self.b))

    def get_interfaces(self):
        return [list(zip(cat(self.R/1e3,True),cat(pl.array(self.basins[(-i-1):]).sum(axis=0)/1e3))) \
                for i in range(len(self.basins))]

    def get_radii(self):
        return pl.array([self.R[pl.where(w<pl.exp(-1)*w.min())].max() for w in self.basins])

    def get_depths(self):
        depths=[self.basins[0].ptp()]
        for w in self.basins[1:-1]:
            depths.append(depths[-1]+w.ptp())
        return pl.array(depths)

    def get_flows(self):
        return pl.array([w.ptp() for w in self.basins])

## I want a squiggly broken line in subplot A to represent a gap
## in depth, but plotting a broken axis isn't really supported
```

```
## in matplotlib. This hack isn't pretty, but works. I'm basically
## just making an object that represents a broken line, and
## using it as the path effect for the y axis.
class BrokenPath(matplotlib.patheffects.AbstractPathEffect):
    def __init__(self, offset=(0, 0), **kwargs):
        super(BrokenPath,self).__init__(offset)
        self.gap = 0.03
        self.tick = 0.05
        self.theta = 3*pl.pi/8
        self.aspect = 1200.0/15.75
        self._gc = kwargs

    def draw_path(self, renderer, gc, tpath, affine, rgbFace=None):
        gc0 = renderer.new_gc()
        gc0.copy_properties(gc)
        gc0 = self._update_gc(gc0, self._gc)
        trans = self._offset_transform(renderer, affine)
        v = pl.array(tpath.vertices)
        vec=v[1]-v[0]
        d = pl.sqrt(sum(vec**2))
        theta=pl.arctan2(vec[0],vec[1])+self.theta
        vhat_T = pl.array([pl.sin(theta)/self.aspect,pl.cos(theta)])
        gap = pl.array((v[0]+vec*(1.0-self.gap)/2.0,v[0]+vec*(1.0+self.gap)/2.0))
        renderer.draw_path(gc0, pl.mpl.path.Path([v[0],gap[0]]), trans, rgbFace)
        renderer.draw_path(gc0, pl.mpl.path.Path([gap[1],v[1]]), trans, rgbFace)
        renderer.draw_path(gc0, pl.mpl.path.Path([gap[0]+vhat_T*self.tick*d,gap[0]-vhat_T*self.tick*d]), trans,
rgbFace)
        renderer.draw_path(gc0, pl.mpl.path.Path([gap[1]+vhat_T*self.tick*d,gap[1]-vhat_T*self.tick*d]), trans,
rgbFace)
        gc0.restore()

def single():
    global model
    # Single model run [For subfigures (A) and (B)]
    model.assemble(Te=8e3) # Use default parameters (Te = 10 km)
    model.run(37, (lambda r: -50*pl.exp(-r**2/(2*15e3**2))))
    return model.basins

def montecarlo():
    global model
    # Many flow models [used in both subfigures (C) and (D)]
    nflows = 43

    pl.np.random.seed(42)
    #Tes = pl.array(list(pl.linspace(2e3,13e3,12))+list(pl.rand(16)*11e3+2e3)+list(pl.rand(20)*4e3+6e3))
    #Tes = pl.array(list(pl.linspace(2e3,13e3,12)) + list(pl.arange(4.5e3,10e3,1e3)))
    Tes = pl.linspace(2e3,13e3,23)
    radii = []
    depths = []
    aspects = []
    plausibles = []
    for i, Te in enumerate(Tes):
        sys.stdout.write('\x1b[2K\r%3d %%'%(100.0*i/len(Tes)))
        sys.stdout.flush()
        model.assemble(Te=Te)
        model.run(nflows, (lambda r: -50*pl.exp(-r**2/(2*15e3**2))))
        rdx = model.get_radii()
        dx = model.get_depths()
        radii.append(list(zip(list(range(nflows)), rdx/1e3)))
        depths.append(list(zip(list(range(nflows)), dx)))
        aspects.append(list(zip(list(range(nflows)), rdx[:-1]/dx)))
        locs = (rdx[:-1]>150e3)&(rdx[:-1]<250e3)&(dx>2e3)&(dx<6e3)
        plausibles.append(locs)
    sys.stdout.write('\x1b[2K\r')
    sys.stdout.flush()
    return  (nflows, Tes, radii, depths, aspects, plausibles)

def trycatch(fname, function, title=None):
    '''
        This function tries to load a pickled archive from a file.
        If that fails, then re-run the model from argument: function,
        and cache its results to the requested archive.
    '''
    try:
        res = pickle.load(open(fname, 'rb'))
        if title is not None:
            print('Loaded '+title)
        return res
    except:
        if title is not None:
            print('Running '+title+'...')
        res = function()
        pickle.dump(res, open(fname, 'wb'))
```

```
        return res

##
## "Schematic" diagram (subfig A)
##
def make_schematic(ax):
    W = [list(zip(cat(model.R/1e3,True),cat(pl.array(basins[(-i-1):]).sum(axis=0)/1e3))) \
                for i in range(len(basins)-1)]
    w0 = pl.array(W[0]).T
    w1 = pl.array(W[-1]).T
    ax.fill_between(w0[0],w0[1],w1[1],color='#AAAAFF')
    ax.plot(w0[0], w0[1],             'k-',   linewidth=2)
    ax.plot(w1[0], w1[1],             'k-',   linewidth=2)
    ax.fill_between(w0[0],w1[1]-10,pl.zeros(w1[1].size)-10,color='#FFAAAA')
    ax.plot(w1[0], w1[1]-10,          'k-',   linewidth=2)
    ax.plot(w1[0], pl.zeros(w1[1].size)-10,  'k--',  linewidth=2)
    ax.text(0,-2.6,'CRB',ha='center',va='center',weight='bold',size=9)
    ax.text(0,-5.5,'Crystaline Basement',ha='center',va='center',weight='bold',size=9)
    ax.text(0,-8.5,'Basaltic Underplate',ha='center',va='center',weight='bold',size=9)
    ax.text(0,-12,'Eclogite',ha='center',va='center',weight='bold',size=9)
    ax.set_xlim((-450,450))
    ax.set_ylim((-14.75,1))
    ax.set_yticks((-14,-10,-4,0))
    ax.set_yticklabels(('44','40','4','0'))
    xline=pl.linspace(-450,450,200)
    ax.plot(xline, 0.1*pl.sin(xline/900*20*pl.pi)-7.11125,'k-',linewidth=0.3)
    ax.plot(xline, 0.1*pl.sin(xline/900*20*pl.pi)-6.63875,'k-',linewidth=0.3)
    ax.spines['left'].set_path_effects([BrokenPath()])
    ax.spines['right'].set_path_effects([BrokenPath()])
    ax.set_xlabel('Distance [km]',labelpad=-0.05)
    ax.set_ylabel('Depth [km]',labelpad=0.5)
    ax.set_title('Isostatically and Flexurally Balanced Basin')

##
## Flow sequence (subfig B)
##
def make_sequence_fig(ax):
    cmap=pl.cm.plasma
    cmrange=40
    sequence_flows = len(basins)
    W = [list(zip(cat(model.R/1e3,True),cat(pl.array(basins[(-i-1):]).sum(axis=0)/1e3))) \
                for i in range(len(basins))]
    def plotinterfaces(ax):
        # Discrete colormap
        line_segments = pl.mpl.collections.LineCollection(W, linewidths=[w.ptp()/300 for w in basins[::-1]], \
                cmap=cmap, norm=pl.mpl.colors.BoundaryNorm(pl.arange(cmrange), cmap.N))
        line_segments.set_array(pl.arange(sequence_flows+1)[::-1])
        ax.add_collection(line_segments)
        cbaxes = inset_axes(ax, width="3%", height="80%", loc=3)
        axcb = fig.colorbar(line_segments, format='%1i', cax=cbaxes)
        axcb.set_label('Number of Flows')
    plotinterfaces(ax)
    def mkzoominset(ax):
        zoomax = inset_axes(ax, width="35%", height="35%", loc=4)
        for i, (line,width) in enumerate(zip(W,[w.ptp()/300/0.35 for w in basins[::-1]])):
            x,y=pl.array(line).T
            zoomax.plot(x,y,linewidth=width,color=cmap(max(min((cmrange-i)/float(cmrange),1),0)))
        ymn, ymx = -4.5,-3.3
        xmx = 80
        zoomax.set_xlim((-xmx,xmx))
        zoomax.set_ylim((ymn,ymx))
        zoomax.set_xticks([])
        zoomax.set_yticks([-4])
        zoomax.set_xticklabels([])
        zoomax.set_yticklabels([])
        ax.plot((-xmx,-xmx,xmx,xmx,-xmx),(ymn,ymx,ymx,ymn,ymn),color='k',linewidth=0.5)
    mkzoominset(ax)
    ax.set_xlim((-325,325))
    ax.set_ylim((-4.8,0.25))
    ax.set_yticks(list(range(-4,1)))
    ax.set_yticklabels(list(range(4,-1,-1)))
    ax.set_ylabel('Depth [km]')
    ax.set_xlabel('Distance [km]', labelpad=0)
    ax.set_title('Compounding Subsidence')

##
## Multi-varaible comparison plot (subfig C)
##
def make_comparison_plot(ax):
    cmap = pl.cm.get_cmap('rainbow', 12)
    radii_segments = pl.mpl.collections.LineCollection(radii[::3], linestyles='dashed', linewidths=1.4, \
            cmap=cmap, norm=pl.Normalize(vmin=0,vmax=Tes.max()/1e3))
    radii_segments.set_array(Tes[::3]/1e3)
```

```
    radii_segments.set_label('Basin radius [km]')
    ax.set_xlim((0,nflows-1))
    ax.set_ylim((26.5,25000))
    ax.add_collection(radii_segments)

    depth_segments = pl.mpl.collections.LineCollection(depths[::3], linewidths=1.4, cmap=cmap,
            norm=pl.Normalize(vmin=0,vmax=Tes.max()/1e3))
    depth_segments.set_array(Tes[::3]/1e3)
    depth_segments.set_label('Basin depth [m]')
    ax.add_collection(depth_segments)

    aspects_segments = pl.mpl.collections.LineCollection(aspects[::3], linestyle='dotted', linewidths=1.4, \
            cmap=cmap, norm=pl.Normalize(vmin=0,vmax=Tes.max()/1e3))
    aspects_segments.set_array(Tes[::3]/1e3)
    aspects_segments.set_label('Aspect ratio')
    ax.add_collection(aspects_segments)

    cbaxes21 = inset_axes(ax, width="60%", height="3%", loc=2)
    axcb = fig.colorbar(aspects_segments, format='%1i', cax=cbaxes21, orientation='horizontal')
    axcb.set_label('Elastic Thickness (Te) [km]')
    ax.set_yscale('log')
    ax.legend(loc='lower center')
    ax.set_xticks(range(0,nflows,5))
    ax.set_xlabel('Number of Flows', labelpad=-0.05)
    ax.set_ylabel('See Legend for Units',labelpad=-1)
    ax.set_title('Basin Characteristics')

##
## Parameter space map (subfig D)
##
def make_parameter_space_map(ax):
    P = pl.array(plausibles).flatten()
    W = pl.where(P==True)
    w = pl.where(P==False)
    nflow,depth  = pl.array(depths).reshape((pl.array(depths).size//2,2)).T
    nflow,radius = pl.array(radii).reshape((pl.array(radii).size//2,2)).T
    Te = pl.array([Tes]*nflows).T.reshape((Tes.size*nflows,))
    X,Y = pl.mgrid[0:nflows:100j,Tes.min()/1e3:Tes.max()/1e3:100j]
    RG=scipy.interpolate.griddata((nflow,Te/1e3), radius, (X,Y))
    DG=scipy.interpolate.griddata((nflow,Te/1e3), depth, (X,Y))
    CR=ax.contour(X,Y,RG,colors='k',linestyles='solid',  linewidths=1)
    CD=ax.contour(X,Y,DG,colors='k',linestyles='dashed', linewidths=1,norm=pl.mpl.colors.LogNorm())
    rllocs=[(2,10),(7.5,10),(20,11)]
    dllocs=[(11,9.75),(26,7),(39,6)]
    pl.clabel(CR,inline=1,fontsize=6,manual=rllocs,fmt=(lambda x: '%1i'%x))
    pl.clabel(CD,inline=1,fontsize=6,manual=dllocs,fmt=(lambda x: '%g'%(x/1e3)))
    ax.scatter(nflow[w], Te[w]/1e3, s=10, marker='x', c='#999999',linewidth=0.7)
    ax.scatter(nflow[W], Te[W]/1e3, s=30, marker='o', c='#333333')
    ax.set_ylabel('Elastic Thickness (Te) [km]', labelpad=-0.5)
    ax.set_xlabel('Number of Flows', labelpad=-0.05)
    ax.set_xlim((0,nflows-1.1))
    ax.set_xticks(range(0,nflows,5))
    ax.set_ylim((Tes.min()/1e3,Tes.max()/1e3))
    ax.set_yticks(range(5,int(Tes.max()/1e3),5))
    ax.set_title('Parameter Space')
    SCX=ax.scatter([],[],s=10,marker='x',c='k',linewidth=0.7)
    SCO=ax.scatter([],[],s=30,marker='o',c='k')
    symbols=[CR.collections[0],CD.collections[0],SCX,SCO]
    labels=['Basin Radius [km]','Basin Depth [km]','Poor Fit','Close Fit']
    ax.legend(symbols,labels,loc='lower left')

## Main function
if __name__ == '__main__':

    # Attempt to load prior results. Otherwise run model suite again
    global model, basins, nflows, Tes, radii, depths, aspects, plausibles
    model = Model()
    basins = trycatch('solution.p', single, 'model for subfigs A and B')
    nflows, Tes, radii, depths, aspects, plausibles = \
            trycatch('montecarlo.p', montecarlo, 'monte carlo suite for subfigs C and D')

    ## Set up figure
    fig = pl.figure(figsize=(7.125,4.75))
    ax11 = fig.add_subplot(221)
    ax12 = fig.add_subplot(222)
    ax21 = fig.add_subplot(223)
    ax22 = fig.add_subplot(224)
    fig.subplots_adjust(left=0.05, bottom=0.06, right=0.995, top=0.95, \
            hspace=0.3, wspace=0.13)

    ## Make plots
    make_schematic(ax11)
    make_sequence_fig(ax12)
```

```
    make_comparison_plot(ax21)
    make_parameter_space_map(ax22)

    ## Add subplot labels (A,B,C,D)
    for i, ax in enumerate((ax11, ax12, ax21, ax22)):
        ax.annotate(chr(ord('A')+i)+')', xy=(0,1), xytext=(-7,5), \
                xycoords='axes fraction', textcoords='offset points', \
                ha='right', va='baseline', weight='bold', size=12)

    pl.savefig('Figure3.tiff')

##
## End file: flex.py
##
```

```
##
## Begin file: matplotlibrc
##
## Matplotlib style parameters.
## Read by Python before plotting model results to ensure reproducibility of figures 3.
##
lines.linewidth   : 2
lines.linestyle   : -
lines.solid_joinstyle : miter
lines.antialiased : True
patch.linewidth        : 0.5
font.family         : monospace
font.style          : normal
font.variant        : normal
font.weight         : medium
font.stretch        : normal
font.size           : 8
text.usetex         : False
text.latex.unicode : True
text.hinting : auto
text.antialiased : True
axes.edgecolor      : black
axes.linewidth      : 0.5
axes.grid           : False
axes.titlesize      : 9
axes.labelsize      : 7
axes.labelweight    : normal
axes.labelcolor     : black
axes.formatter.limits : -7, 7
axes.formatter.use_mathtext : True
axes.unicode_minus  : True
axes.xmargin        : 0
axes.ymargin        : 0
xtick.major.width    : 0.3
xtick.minor.width    : 0.3
xtick.labelsize      : 7
xtick.direction      : in
ytick.major.width    : 0.3
ytick.minor.width    : 0.3
ytick.labelsize      : 7
ytick.direction      : in
grid.linewidth   :   0.15
legend.fancybox      : True
legend.fontsize      : 7
figure.figsize   : 3.5,2.75
figure.dpi       : 600
figure.autolayout : False
image.aspect  : equal
image.interpolation  : bilinear
image.cmap    : gray
##
## End file: matplotlibrc
##
```

```
##
## Begin file: latent_heat.shml
##

<?xml version='1.0' encoding='utf-8'?>
<harness_options>
  <length>
    <string_value lines="1">medium</string_value>
  </length>
  <owner>
    <string_value lines="1">JPH</string_value>
  </owner>
  <description>
    <string_value lines="1">Explore \Pi parameter space in nondimensionalized Pasco advection diffusion
problem.</string_value>
  </description>
  <simulations>
    <simulation name="Model1">
      <input_file>
        <string_value lines="1" type="filename">latent_heat.tfml</string_value>
      </input_file>
      <run_when name="input_changed_or_output_missing"/>
      <parameter_sweep>
        <parameter name="Pe">
          <values>
            <string_value lines="1">0.1 0.2 0.3 0.4 0.5 0.75 1.0 2 3 4 5 6 7 8 9 10</string_value>
            <comment></comment>
          </values>
          <update>
            <string_value lines="20" type="code" language="python">import libspud
libspud.set_option("/system::AdvectionDiffusion/coefficient::Pe/type::Constant/rank::Scalar/value::WholeMesh/
constant", float(Pe))
libspud.set_option("timestepping/timestep/coefficient::Timestep/type::Constant/rank::Scalar/value::WholeMesh/
constant", float(Pe)*0.00025)</string_value>
            <comment>Change the nondimensional Pi variable.</comment>
            <single_build/>
          </update>
        </parameter>
        <parameter name="output_base_name">
          <values>
            <string_value lines="1">adv_diffusion_zeroinitialX</string_value>
          </values>
          <update>
            <string_value lines="20" type="code" language="python">import libspud

libspud.set_option("/io/output_base_name", output_base_name)</string_value>
            <single_build/>
          </update>
        </parameter>
        <parameter name="TemperatureBC">
          <values>
            <string_value lines="1">LR</string_value>
          </values>
          <update>
            <string_value lines="20" type="code" language="python">import libspud
if TemperatureBC == 'L':
  boundary_ids = [1]
else:
  boundary_ids = [1, 2]

opt = '/system::AdvectionDiffusion/field::Temperature/type::Function/rank::Scalar/boundary_condition::IC_BC/
boundary_ids'
libspud.set_option(opt, boundary_ids)</string_value>
            <single_build/>
          </update>
        </parameter>
      </parameter_sweep>
      <variables>
        <variable name="Solution">
          <string_value lines="20" type="code" language="python">from buckettools.vtktools import vtu
import re
from numpy import array, concatenate as cat

t = []
z = []
T = []
X = []

f = open(output_base_name+'.pvd', 'r').read()
timesteps = set(re.findall('timestep="(\S*)".*part="(\d+)"', f))
for time, part in sorted(timesteps, key=lambda tp: map(float, tp)):
  regex ='timestep="%s".*part="%s".*file="(%s.*vtu)"' \
            %(time, part, output_base_name)
```

```
  # If there are multiple VTUs saved for the same timestep, and it isn't multi-part,
  # then just choose the last one and throw out the others. That's what Paraview does.
  fname = list(sorted(re.findall(regex, f)))[-1]
  VTU = vtu(fname)
  if int(part) &gt; 0:
    z[-1] = cat((z[-1], VTU.GetLocations().T[0]))
    T[-1] = cat((T[-1], VTU.GetField("AdvectionDiffusion::Temperature").T[0]))
    X[-1] = cat((T[-1], VTU.GetField("AdvectionDiffusion::X").T[0]))
  else:
    t.append(float(time))
    z.append(VTU.GetLocations().T[0])
    T.append(VTU.GetField("AdvectionDiffusion::Temperature").T[0])
    X.append(VTU.GetField("AdvectionDiffusion::X").T[0])

Solution = (array(t), array(z), array(T), array(X))</string_value>
        </variable>
      </variables>
    </simulation>
  </simulations>
  <tests>
    <test name="PlotXAndT">
      <string_value lines="20" type="code" language="python">import matplotlib
matplotlib.use('Agg')
import pylab as pl

slice = {k: Solution.parameters[k][0] for k in Solution.parameters.keys() \
          if len(Solution.parameters[k]) == 1}

log_max_pe = max([pl.log(float(Pe)) for Pe in Solution.parameters['Pe']])

for BCS in Solution.parameters['TemperatureBC']:
  slice['TemperatureBC'] = BCS
  ymin = 0
  ymax = 0
  for Pe in Solution.parameters['Pe']:
    slice['Pe'] = Pe
    tx, zx, Tx, Xx = Solution[slice]
    ymax = max(ymax, Tx.max(), Xx.max())
    ymin = min(ymin, Tx.min(), Xx.min())

  keepgoing = True
  for time in range(1,6)+[60]:
    fig = pl.figure(figsize=(3,6), dpi=100)
    ax1 = fig.add_subplot(211)
    ax2 = fig.add_subplot(212)
    ax1.invert_yaxis()
    ax2.invert_yaxis()
    ymin1 = ymin2 = 1
    ymax1 = ymax2 = 0

    #keepgoing = False
    for Pe in Solution.parameters['Pe']:
      slice['Pe'] = Pe
      Pe = float(slice['Pe'])
      tx, zx, Tx, Xx = Solution[slice]

      step = pl.where(tx &lt;= time)[0][-1]

      #keepgoing |= (tx.max() &gt; time)
      #keepgoing &amp;= time &lt; 60

      z, T, X = zx[step], Tx[step], Xx[step]
      cm = pl.cm.viridis(min(1.0, (pl.log10(Pe)+1)/2.0))
      ax1.plot(T, z, color=cm, linewidth=1.5)
      ax2.plot(X, z, color=cm, linewidth=1.5)
      ymin1 = min(ymin1, T.min())
      ymin2 = min(ymin2, X.min())
      ymax1 = max(ymax1, T.max())
      ymax2 = max(ymax2, X.max())

    ax1.spines['top'].set_visible(True)
    ax1.spines['bottom'].set_visible(False)
    ax1.spines['left'].set_visible(True)
    ax1.spines['right'].set_visible(False)
    ax1.xaxis.set_ticks_position('top')
    ax1.yaxis.set_ticks_position('left')
    ax1.xaxis.set_label_position('top')
    ax1.xaxis.set_ticks([-2, -1.5, -1, -0.5, 0])
    ax1.xaxis.set_ticklabels(['-2', '-1.5', '-1', '-0.5', '0'])
    ax1.yaxis.set_ticks([0, 1, 2, 3])
    ax1.xaxis.tick_top()
    ax1.yaxis.tick_left()
    ax1.set_xlabel('T')
```

```
    ax2.spines['top'].set_visible(False)
    ax2.spines['bottom'].set_visible(True)
    ax2.spines['left'].set_visible(True)
    ax2.spines['right'].set_visible(False)
    ax2.xaxis.set_ticks_position('bottom')
    ax2.yaxis.set_ticks_position('left')
    ax2.xaxis.set_ticks([0, 0.5, 1])
    ax2.xaxis.set_ticklabels(['0', '0.5', '1'])
    ax2.yaxis.set_ticks([0, 1, 2, 3])
    ax2.yaxis.set_ticklabels(['0', '1', '2', '3'])
    ax2.set_xlabel('X')
    ax1.set_xlim((-2.2,0.2))
    ax2.set_xlim((0, 1.1))
    #ax1.set_xlim((ymin1, ymax1))
    #ax2.set_xlim((ymin2, ymax2))
    pl.savefig('time_%05f.eps'%(time))
    #time += 5 #(tx[1]-tx[0])
    pl.close(fig)</string_value>
    </test>
    <test name="IsostasyOverTime">
      <string_value lines="20" type="code" language="python">import matplotlib
matplotlib.use('Agg')
import pylab as pl

slice = {k: Solution.parameters[k][0] for k in Solution.parameters.keys() \
          if len(Solution.parameters[k]) == 1}

log_max_pe = max([pl.log10(float(Pe)) for Pe in Solution.parameters['Pe']])
print log_max_pe
print Solution.parameters['Pe']
print [pl.log10(float(Pe))/10.0 for Pe in Solution.parameters['Pe']]
for BCS in Solution.parameters['TemperatureBC']:
  fig = pl.figure(figsize=(4,4), dpi=100)
  ax = fig.add_subplot(111)
  slice['TemperatureBC'] = BCS
  tmax = 0
  imin = 0
  imax = 0
  isostasy = {}
  time = {}

  for Pe in Solution.parameters['Pe']:
    slice['Pe'] = Pe
    tx, zx, Tx, Xx = Solution[slice]
    time[Pe] = list(tx[pl.where(tx &lt; 60.0)])
    isostasy[Pe] = [Xx[t].sum()/Xx[t].size for t in time[Pe]]
    tmax = max(tmax, max(time[Pe]))
    imin = min(imin, min(isostasy[Pe]))
    imax = max(imax, max(isostasy[Pe]))

  for Pe in isostasy:
    fPe = float(Pe)
    color = pl.cm.viridis(min(1.0, (pl.log10(fPe)+1)/2.0))
    ax.plot(time[Pe]+[tmax], isostasy[Pe]+[isostasy[Pe][-1]], color=color)

  ax.spines['top'].set_visible(False)
  ax.spines['bottom'].set_visible(True)
  ax.spines['left'].set_visible(True)
  ax.spines['right'].set_visible(False)
  ax.xaxis.set_ticks_position('bottom')
  ax.yaxis.set_ticks_position('left')
  ax.xaxis.set_label_position('bottom')
  ax.set_xlabel('time')
  ax.set_ylabel('X')
  ax.set_title('Eclogite Fraction')
  ax.set_ylim([-1e-3, 0.8])
  fig.savefig(BCS+'-topography_vs_time.eps')
  pl.close(fig)</string_value>
    </test>
    <test name="SSIsostasy">
      <string_value lines="20" type="code" language="python">import matplotlib
matplotlib.use('Agg')
import pylab as pl
pl.ticklabel_format(style='sci', axis='y', scilimits=(0,0))

slice = {k: Solution.parameters[k][0] for k in Solution.parameters.keys() \
          if len(Solution.parameters[k]) == 1}

log_max_pe = max([pl.log10(float(Pe)) for Pe in Solution.parameters['Pe']])
print log_max_pe
print Solution.parameters['Pe']
print [pl.log10(float(Pe))/10.0 for Pe in Solution.parameters['Pe']]
for BCS in Solution.parameters['TemperatureBC']:
```

```
  fig = pl.figure(figsize=(4,4), dpi=100)
  ax = fig.add_subplot(111)
  slice['TemperatureBC'] = BCS
  isostasy = {}

  for Pe in Solution.parameters['Pe']:
    slice['Pe'] = Pe
    tx, zx, Tx, Xx = Solution[slice]
    #times = list(tx[pl.where(tx &lt; 60.0)])
    #isostasy[Pe] = -Xx[-1].sum()/Xx.size
    ax.scatter([1./float(Pe)], [Xx[-1].sum()/Xx[-1].size], color='black')

  ax.spines['top'].set_visible(False)
  ax.spines['bottom'].set_visible(True)
  ax.spines['left'].set_visible(True)
  ax.spines['right'].set_visible(False)
  ax.xaxis.set_ticks_position('bottom')
  ax.yaxis.set_ticks_position('left')
  ax.xaxis.set_label_position('bottom')
  #ax.xaxis.set_ticks([-1, -0.5, 0])
  #ax.xaxis.set_ticklabels(['-1', '-0.5', '0'])
  #ax.yaxis.set_ticks([0, 1, 2, 3])

  ax.set_xlabel('$Pe_r$')
  #ax.set_ylabel('$\\frac{1}{L} \\int_\Omega X dz$')
  ax.set_ylabel('X')
  ax.set_xscale('log')
  ax.set_xlim((0.09, 11))
#  ax.set_ylim((-0.05, 0.65))
  ax.set_title('Steady-State Eclogite Fraction')
  fig.savefig(BCS+'-ss_topography_vs_Pe.eps')
  pl.close(fig)</string_value>
    </test>
    <test name="Colorbar">
      <string_value type="code" lines="20" language="python">from pylab import *

fig = figure(figsize=(4,1))
ax1 = fig.add_axes([0.05, 0.5, 0.9, 0.3])

cmap = mpl.cm.viridis
norm = mpl.colors.LogNorm(vmin=0.1, vmax=10)
cb1 = mpl.colorbar.ColorbarBase(ax1, cmap=cmap,
                                norm=norm,
                                orientation='horizontal')
cb1.set_label('$Pe_r^{-1}$')

fig.savefig('colorbar.eps')</string_value>
    </test>
  </tests>
</harness_options>

##
## End file: latent_heat.shml
##
```

```
##
## Begin file: latent_heat.tfml
##
```


```
<?xml version='1.0' encoding='utf-8'?>
<terraferma_options>
  <geometry>
    <dimension>
      <integer_value rank="0">1</integer_value>
    </dimension>
    <mesh name="Mesh">
      <source name="Interval">
        <left>
          <real_value rank="0">0</real_value>
        </left>
        <right>
          <real_value rank="0">3</real_value>
        </right>
        <number_cells>
          <integer_value rank="0">100</integer_value>
          <comment>If I increase this, I need to change the timestep

dt = \alpha *model_width / n_cells</comment>
        </number_cells>
        <cell>
          <string_value lines="1">interval</string_value>
        </cell>
      </source>
    </mesh>
  </geometry>
  <io>
    <output_base_name>
      <string_value lines="1">adv_diffusion</string_value>
    </output_base_name>
    <visualization>
      <element name="P1">
        <family>
          <string_value lines="1">CG</string_value>
        </family>
        <degree>
          <integer_value rank="0">1</integer_value>
        </degree>
      </element>
    </visualization>
    <dump_periods>
      <visualization_period>
        <real_value rank="0">1</real_value>
      </visualization_period>
    </dump_periods>
    <detectors/>
  </io>
  <timestepping>
    <current_time>
      <real_value rank="0">0</real_value>
    </current_time>
    <finish_time>
      <real_value rank="0">200</real_value>
    </finish_time>
    <timestep>
      <coefficient name="Timestep">
        <ufl_symbol name="global">
          <string_value lines="1">dt</string_value>
        </ufl_symbol>
        <type name="Constant">
          <rank name="Scalar" rank="0">
            <value name="WholeMesh">
              <constant>
                <real_value rank="0">0.00025</real_value>
                <comment>Courant condition
dt = C * h/(p|u|) where h is grid spacing, and p is the polynomial degree of the temperature field (2)
For the fully implicit scheme, I think C can be 1.

Von Neumann:
dt &lt; h^2 / 3 * D (diffusivity)

Von Neumann is going to be smaller in all of my cases.
Examples:

Xmax-Xmin                n_cells         Von Neumann      Courant
20                               100                      0.01333                  0.1
20                               1000            0.0001333                0.01</comment>
              </constant>
            </value>
```

```
          </rank>
        </type>
      </coefficient>
    </timestep>
    <steady_state>
      <tolerance>
        <real_value rank="0">1e-6</real_value>
      </tolerance>
    </steady_state>
  </timestepping>
  <global_parameters/>
  <system name="AdvectionDiffusion">
    <mesh name="Mesh"/>
    <ufl_symbol name="global">
      <string_value lines="1">us</string_value>
    </ufl_symbol>
    <field name="Temperature">
      <ufl_symbol name="global">
        <string_value lines="1">T</string_value>
      </ufl_symbol>
      <type name="Function">
        <rank name="Scalar" rank="0">
          <element name="P2">
            <family>
              <string_value lines="1">CG</string_value>
            </family>
            <degree>
              <integer_value rank="0">2</integer_value>
            </degree>
          </element>
          <initial_condition type="initial_condition" name="WholeMesh">
            <constant>
              <real_value rank="0">0</real_value>
            </constant>
          </initial_condition>
          <boundary_condition name="IC_BC">
            <boundary_ids>
              <integer_value shape="2" rank="1">1 2</integer_value>
            </boundary_ids>
            <sub_components name="All">
              <type type="boundary_condition" name="Dirichlet">
                <constant>
                  <real_value rank="0">0</real_value>
                </constant>
              </type>
            </sub_components>
            <comment>Only define boundary condition on left boundary.</comment>
          </boundary_condition>
        </rank>
      </type>
      <diagnostics>
        <include_in_visualization/>
        <include_in_statistics/>
        <include_in_detectors/>
      </diagnostics>
    </field>
    <field name="X">
      <ufl_symbol name="global">
        <string_value lines="1">X</string_value>
      </ufl_symbol>
      <type name="Function">
        <rank name="Scalar" rank="0">
          <element name="P2">
            <family>
              <string_value lines="1">CG</string_value>
            </family>
            <degree>
              <integer_value rank="0">2</integer_value>
            </degree>
          </element>
          <initial_condition type="initial_condition" name="WholeMesh">
            <python rank="0">
              <string_value lines="20" type="code" language="python">def val(x):
  return 0.0
#  from numpy import tanh
#  return 0.5*(1 + tanh((x[0]-1)/0.02)) # - 1./cosh(x[0]-14)**2</string_value>
            </python>
          </initial_condition>
          <boundary_condition name="Left">
            <boundary_ids>
              <integer_value shape="1" rank="1">1</integer_value>
            </boundary_ids>
            <sub_components name="All">
```

```
              <type type="boundary_condition" name="Dirichlet">
                <constant>
                  <real_value rank="0">0</real_value>
                </constant>
              </type>
            </sub_components>
          </boundary_condition>
        </rank>
      </type>
      <diagnostics>
        <include_in_visualization/>
        <include_in_statistics/>
        <include_in_steady_state>
          <norm>
            <string_value lines="1">l2</string_value>
          </norm>
        </include_in_steady_state>
        <include_in_detectors/>
      </diagnostics>
    </field>
    <coefficient name="z">
      <ufl_symbol name="global">
        <string_value lines="1">z</string_value>
      </ufl_symbol>
      <type name="Function">
        <rank name="Scalar" rank="0">
          <element name="P1">
            <family>
              <string_value lines="1">CG</string_value>
            </family>
            <degree>
              <integer_value rank="0">1</integer_value>
            </degree>
          </element>
          <value type="value" name="WholeMesh">
            <cpp rank="0">
              <members>
                <string_value lines="20" type="code" language="cpp">// Do nothing</string_value>
              </members>
              <initialization>
                <string_value lines="20" type="code" language="cpp">// Do nothing</string_value>
              </initialization>
              <eval>
                <string_value lines="20" type="code" language="cpp">values[0] = x[0];</string_value>
              </eval>
              <time_independent/>
            </cpp>
          </value>
        </rank>
      </type>
      <diagnostics/>
    </coefficient>
    <coefficient name="Pe">
      <ufl_symbol name="global">
        <string_value lines="1">Pe</string_value>
      </ufl_symbol>
      <type name="Constant">
        <rank name="Scalar" rank="0">
          <value type="value" name="WholeMesh">
            <constant>
              <real_value rank="0">1</real_value>
            </constant>
          </value>
        </rank>
      </type>
      <diagnostics/>
    </coefficient>
    <nonlinear_solver name="Solver">
      <type name="SNES">
        <form name="Residual" rank="0">
          <string_value lines="20" type="code" language="python"># Theta = 0: Fully explicit first-order Euler scheme
# Theta = 0.5: Second-order Crank-Nicolson (trapezoidal)
# Theta = 1: Fully implicit backwards Euler scheme
theta      = 1.0

T_theta = theta*T_i + (1.0-theta)*T_n
X_theta = theta*X_i + (1.0-theta)*X_n

T0 = 0.0
Z0 = 1.0

FT = T_t * (T_i - T_n) * dx \
    + dt * T_t * grad(T_theta)[0] * dx \
```

```
    + dt * Pe * inner(grad(T_t), grad(T_theta)) * dx \
    + dt * T_t * 1 * dx \
    - T_t * (z + T_theta + T0) * ((X_i - X_n) + dt*grad(X_i)[0]) * dx

Fx = X_t * ((X_i - X_n) - dt*exp(-1.0/(z + T_i + T0))*(0.5 + 0.5*tanh((z - T_i - T0 - Z0)/0.02) - X_theta)) * dx

F = FT + Fx</string_value>
          <ufl_symbol name="solver">
            <string_value lines="1">F</string_value>
          </ufl_symbol>
        </form>
        <form name="Jacobian" rank="1">
          <string_value lines="20" type="code" language="python">J = derivative(F, us_i, us_a)</string_value>
          <ufl_symbol name="solver">
            <string_value lines="1">J</string_value>
          </ufl_symbol>
        </form>
        <form_representation name="quadrature"/>
        <quadrature_rule name="default"/>
        <snes_type name="ksponly"/>
        <relative_error>
          <real_value rank="0">1e-8</real_value>
        </relative_error>
        <absolute_error>
          <real_value rank="0">1e-11</real_value>
        </absolute_error>
        <max_iterations>
          <integer_value rank="0">20</integer_value>
        </max_iterations>
        <monitors>
          <residual/>
          <convergence_file/>
        </monitors>
        <linear_solver>
          <iterative_method name="fgmres">
            <restart>
              <integer_value rank="0">30</integer_value>
            </restart>
            <relative_error>
              <real_value rank="0">1e-6</real_value>
            </relative_error>
            <absolute_error>
              <real_value rank="0">1e-11</real_value>
            </absolute_error>
            <max_iterations>
              <integer_value rank="0">100</integer_value>
            </max_iterations>
            <zero_initial_guess/>
            <monitors>
              <preconditioned_residual/>
            </monitors>
          </iterative_method>
          <preconditioner name="fieldsplit">
            <composite_type name="multiplicative"/>
            <fieldsplit name="Temperature">
              <field name="Temperature"/>
              <monitors/>
              <linear_solver>
                <iterative_method name="preonly"/>
                <preconditioner name="lu">
                  <factorization_package name="mumps"/>
                </preconditioner>
              </linear_solver>
            </fieldsplit>
            <fieldsplit name="Eclogite">
              <field name="X"/>
              <monitors/>
              <linear_solver>
                <iterative_method name="preonly"/>
                <preconditioner name="lu">
                  <factorization_package name="mumps"/>
                </preconditioner>
              </linear_solver>
            </fieldsplit>
          </preconditioner>
        </linear_solver>
        <never_ignore_solver_failures/>
      </type>
      <solve name="in_timeloop"/>
    </nonlinear_solver>
  </system>
</terraferma_options>
```

```
##
## End file: latent_heat.tfml
##
```