APPENDIX C

SUPPLEMENTARY INFORMATION TO CHAPTER 4

C.1 Introduction

The supplementary information we include here includes of a statement of the problem to be solved numerically, the finite element method formulation of viscoelasticity, the application of boundary conditions, a description of the mechanical solver, the general code structure, and a section of the Matlab code. Also included is an analysis of a sinuous topographic feature in the eastern section of the model domain.

C.2 Statement of problem to be solved numerically

In this study, a viscoelastic plate interface channel with temporal variations in its rheological properties is modeled to test the effect on western forearc topography of climate-controlled variations in sediment delivery to the trench and the subduction channel, and their influence on flow within the subduction channel. Mass-transfer processes such as underplating are not considered.

A Lagrangian description is used to follow deformation of the forearc as a result of stresses imposed on the subduction channel top by the subducting slab and by flow on the subduction channel, as well as in response to buoyancy forces originating from differences in density between subduction channel and forearc materials. The

subaerial and submarine surfaces are treated as free surfaces and their deformation is followed through time.

Flow in a subduction channel is simplified as a two-dimensional incompressible, Newtonian, isotropic, viscoelastic Stokes fluid flow. Subduction channel flow along a direction parallel to the trench axis is assumed small compared to flow along a plane normal to the trench axis, given the predominant forces acting on the channel (shear force applied by the subducting plate and gravity-derived force). Considering we have no constraints on the shear stress-shear rate law acting along the subduction channel, we assume a linear, Newtonian behavior. Even if subduction channel materials may present short-scale (i.e. grain size) anisotropy in viscosity, we assume no such anisotropy at longer scales. Since mantle and crustal motions can be considered highly viscous flows in geologic timescales, flow is very slow and the inertial forces are negligible. We thus use the Stokes approximation to fluid flow. Rocks can recover part of the deformation once stress disappears, so we use a viscoelastic rather than a purely viscous rheology. Finally, incompressibility of subduction channel materials is assumed.

C.3 Finite element method formulation of viscoelasticity

Applying Newton's second law to an arbitrary volume of fluid under the action of body and viscous forces leads to the equation of motion:

$$\rho \frac{Dv_i}{Dt} = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho f_i, \quad (1)$$

where the scalar ρ is density, v_i is the velocity field, σ_{ij} the stress tensor and ρf_i the body force. Under the assumption of incompressibility, the mass-conservation equation may be written:

$$\frac{\partial v_i}{\partial x_i}. \tag{2}$$

The stress-strain relationship for Newtonian, isotropic fluids is:

$$\sigma_{ij} = -p\delta_{ij} + \tau_{ij} = -p\delta_{ij} + \mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) = -p\delta_{ij} + 2\mu\dot{\varepsilon}_{ij}, \tag{3}$$

where $\dot{\varepsilon}_{ij}$ is the strain rate tensor, τ_{ij} is the deviatoric stress tensor, the scalars μ and p are the viscosity and pressure, respectively, and δ_{ij} is the Kronecker delta. Note that $\dot{\varepsilon}_{ij}$, σ_{ij} and τ_{ij} are symmetric tensors.

Replacing the stress-strain relationship of eq. (3) into the equation of motion eq. (1) results in the Navier-Stokes equations for Newtonian, isotropic viscous flow:

$$\rho\frac{Dv_i}{Dt} = \rho f_i - \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)\right]. \tag{4}$$

Under the assumption that inertial forces are much smaller that viscous forces, we may drop the Dv_i/Dt term entirely from eq. (4) and arrive at the Stokes approximation of Newtonian, isotropic viscous fluid flow:

$$\rho f_i - \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)\right] = 0. \tag{5}$$

It can be shown that 2D plane strain, incompressible, Newtonian, isotropic, Stokes viscous fluid flow under the influence of gravity can be described in a Cartesian coordinate system *(x,y)* as:

$$-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}\left[\mu\left(\frac{4}{3}\frac{\partial v_x}{\partial x}-\frac{2}{3}\frac{\partial v_y}{\partial y}\right)\right]+\frac{\partial}{\partial y}\left[\mu\left(\frac{\partial v_x}{\partial y}+\frac{\partial v_y}{\partial x}\right)\right]=0, \quad \text{in the x-axis} \tag{6}$$

$$\rho g-\frac{\partial p}{\partial y}+\frac{\partial}{\partial y}\left[\mu\left(\frac{4}{3}\frac{\partial v_y}{\partial y}-\frac{2}{3}\frac{\partial v_x}{\partial x}\right)\right]+\frac{\partial}{\partial x}\left[\mu\left(\frac{\partial v_x}{\partial y}+\frac{\partial v_y}{\partial x}\right)\right]=0, \quad \text{in the y-axis} \tag{7}$$

$$\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}=0, \tag{8}$$

where $f_i = (0,g)$.

A general stress-strain relationship for viscoelasticity is obtained by adding an elastic term to eq. (3):

$$\tau_{ij}+\frac{\mu}{G}\frac{D\tau_{ij}}{Dt}=2\mu\dot{\varepsilon}_{ij}, \tag{9}$$

where G is the elastic shear modulus, and the Jaumann objective deviatoric stress rate within a lagrangian grid is given by:

$$\tau_{ij}^{J}=\frac{D\tau_{ij}}{Dt}=\frac{\partial\tau_{ij}}{\partial t}-\omega_{ik}\tau_{kj}+\tau_{ik}\omega_{kj}, \tag{10}$$

where ω_{ij} is the rigid-body rotation tensor defined by:

$$\omega_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}-\frac{\partial u_j}{\partial x_i}\right), \tag{11}$$

where u_i is the displacement field. We can discretize eq. (9) in time to get:

$$\tau_{ij}^{new} = 2\mu_{ve}\dot{\varepsilon}_{ij} + \chi\tau_{ij}^{old\ J}, \tag{12}$$

where

$$\mu_{ve} = \frac{1}{1/\mu + 1/G\Delta t}, \tag{13}$$

$$\chi = (1 + \frac{G\Delta t}{\mu})^{-1}, \tag{14}$$

$$\tau_{ij}^{old\ J} = \tau_{ij}^{old} - \omega_{ik}^{old}\tau_{kj}^{old} + \tau_{ik}^{old}\omega_{kj}^{old}. \tag{15}$$

From eq. (12), the viscoelastic viscosity μ_{ve} can be expressed in terms of viscosity μ as:

$$\mu_{ve} = {\tau_{ij} - \chi\tau_{ij}^{old\ J}}/{2\dot{\varepsilon}_{ij}} = \mu - {\chi\tau_{ij}^{old\ J}}/{2\dot{\varepsilon}_{ij}}. \tag{16}$$

Substituting eq. (16) into eqs. (6) and (7) and expanding the equations, we obtain the expressions for 2D incompressible, Newtonian, isotropic, Stokes viscoelastic fluid flow under the influence of gravity in a Cartesian coordinate system *(x,y)*:

$$-\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left[\mu_{ve}\left(\frac{4}{3}\frac{\partial v_x}{\partial x} - \frac{2}{3}\frac{\partial v_y}{\partial y}\right)\right] + \frac{\partial}{\partial y}\left[\mu_{ve}\left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}\right)\right] = -\chi\frac{\partial \tau_{xx}^{old\ J}}{\partial x} - \chi\frac{\partial \tau_{xy}^{old\ J}}{\partial y}, \tag{17}$$

$$\rho g - \frac{\partial p}{\partial y} + \frac{\partial}{\partial y}\left[\mu_{ve}\left(\frac{4}{3}\frac{\partial v_y}{\partial y} - \frac{2}{3}\frac{\partial v_x}{\partial x}\right)\right] + \frac{\partial}{\partial x}\left[\mu_{ve}\left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}\right)\right] = -\chi\frac{\partial \tau_{yx}^{old\ J}}{\partial x} - \chi\frac{\partial \tau_{yy}^{old\ J}}{\partial y}, \tag{18}$$

in the x-axis and y-axis, respectively, with $f_i = (0,g)$. Finally, the incompressibility constraint completes the description:

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} = 0.$$

The strong form for the viscoelastic Stokes flow problem is given by eqs. (17), (18) and (8) for the 2D domain Ω, where the incompressibility constraint is imposed by a penalty method:

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{p}{\kappa} = 0, \tag{19}$$

where κ is large. The boundary Γ of domain Ω is divided into Γ_e, where essential (i.e. fixed velocity) boundary conditions are imposed, and Γ_n, where natural (i.e. traction) boundary conditions are imposed, such that:

$$\Gamma_e \cup \Gamma_n = \Gamma \quad \text{and} \quad \Gamma_e \cap \Gamma_n. \tag{20}$$

The velocity components and pressure field to be solved are approximated in the domain Ω by:

$$v_x(x,y) \approx \tilde{v}_x(x,y) = \sum^{nnod} N_i(x,y) v_x^i, \tag{21}$$

$$v_y(x,y) \approx \tilde{v}_y(x,y) = \sum^{nnod} N_i(x,y) v_y^i, \tag{22}$$

$$p(x,y) \approx \tilde{p}(x,y) = \sum^{np} \Pi_i(x,y) p^i, \tag{23}$$

where N_i are velocity shape functions, Π_i are pressure shape functions, *nnod* is the number of nodes in the discretized domain, and *np* is the number of pressure degrees of freedom.

The domain Ω is subdivided into a number *nel* of subdomains Ω_e, such that:

$$\Omega = \bigcap^{nel} \Omega_e, \tag{24}$$

where each subdomain Ω_e is a triangular element with seven nodes and quadratic shape functions for velocity, and three nodes and linear shape functions for pressure.

The stiffness matrix for the viscoelastic Stokes problem is written:

$$K^e = \begin{pmatrix} A & Q^T \\ Q & -M/\kappa \end{pmatrix} = \iint_{\Omega'} \begin{pmatrix} \mu_{ve}^e B^T D B & -B_{vol}^T \Pi^T \\ -\Pi B_{vol} & -\Pi\Pi^T/\kappa \end{pmatrix} dxdy, \tag{25}$$

where

$$B(x,y)v^e = \begin{pmatrix} \frac{\partial N_1}{\partial x}(x,y) & 0 & \dots \\ 0 & \frac{\partial N_1}{\partial y}(x,y) & \dots \\ \frac{\partial N_1}{\partial y}(x,y) & \frac{\partial N_1}{\partial x}(x,y) & \dots \end{pmatrix} \begin{pmatrix} v_x^1 \\ v_y^1 \\ v_x^2 \\ v_y^2 \\ \dots \\ v_x^7 \\ v_y^7 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}. \tag{26}$$

The linear system of equations to be solved is then:

$$\begin{pmatrix} A & Q^T \\ Q & -M/\kappa \end{pmatrix} v^e = (0\ \rho g\ 0\ \rho g \dots 0\ \rho g\ 0\ 0\ 0)^{-1}. \tag{27}$$

C.4 Boundary conditions

As with any boundary value problem for partial differential equations, boundary conditions need to be imposed to the boundary Γ of the problem domain Ω. Natural (i.e. force) boundary conditions are applied at the continental surface, where a null force is prescribed to nodes in the continental boundary, and a marine surface, where a force corresponding to the water column weight:

$$F = \rho_w g h(x), \tag{28}$$

is prescribed to nodes in the marine boundary.

Essential (i.e. fixed velocity) boundary conditions are applied at the rest of the boundaries of the problem domain. In particular, a fixed velocity corresponding to the relative convergence rate, v_s, is applied at the base of the subduction channel.

C.5 MILAMIN-based mechanical solver

The mechanical solver in the code is based on a modified version of MILAMIN: a MATLAB-based finite element method implementation [Dabrowski et al., 2008]. The equations to discretize and solve are eq. (17), (18) and (19). We modified the code to include the viscoelastic terms in eq. (17) and (18).

The incompressibility constraint given by eq. (19) is achieved through an iterative penalty method. A relatively small penalty factor κ is used, guaranteeing a

good condition number, and incompressibility of flow is achieved by Powell and Hestenes iterations [Dabrowski et al., 2008].

MILAMIN incorporates fast matrix assembly and storage, direct solver and reordering implementations (see Dabrowski et al. [2008] for a full discussion on the MILAMIN code architecture and performance). The main reasons for this being interchanging loops and operating on large matrices, reducing the number of operations by exploiting the system symmetry, and the optimization of cache reuse through element blocking [Dabrowski et al., 2008].

C.6 Code flow and general code structure

The general code structure is as follows:

A) PRE-PROCESSING

A.1) Clearing the environment

A.2) User inputs and constants

A.3) Mesh generation

A.4) Initialization and global storage

A.5) Boundary conditions

B) PROCESSING

B.1) Time stepping, move mesh, remesh, and interpolate stress field

B.2) Mechanical solver

C) POST-PROCESSING

C.1) Stress rotation

C.2) Time-dependent plots

The following script corresponds to the main Matlab file, from which all functions are called:

```
% CODE FOR 2D SUBDUCTION CHANNEL PROCESSES
% Based on MILAMIN: MATLAB-based FEM solver for large problems, Version 1.0
% Copyright (C) 2007, M. Dabrowski, M. Krotkiewski, D.W. Schmid
% University of Oslo, Physics of Geological Processes
% http://milamin.org
% See License file for terms of use.

% Modelling of viscoelastic flow within a subduction channel and response
% of free marine and continental surfaces to changes in viscosity of SC.
% Adds viscoelasticity to mechanical2d solver
%
% authors: Lars Rupke, Jason Phipps-Morgan, Nicolas J Cosentino, Miguel A.
% Martinez

%=========================================================================
%=========================================================================
% A. PREPROCESSING
%=========================================================================
%=========================================================================

%=========================================================================
% A.1. CLEARING and buffering
%=========================================================================

% clear environment
clc
close all
clear variables;
set(0, 'DefaultFigureRenderer', 'zbuffer');
```

```
%==========================================================================
% A.2. USER INPUT & CONSTANTS
%==========================================================================

% ADD MUTILS FOLDERS TO SEARCH PATH, modify according to local setting
addpath...
 (genpath('C:\Users\njc58\Desktop\Nico_nov14\SC_matlab\code\mutils-0.4-2')) %
change according to infrastructure used

% FIXED CONSTANTS
km           = 1000;                                  % how many meters in a km
ma           = 1e6*365*24*60*60;                      % how many seconds in a million
years
G            = [0;-9.81];                             % gravity, m/s2

% USER INPUT - material properties
Visco        = [7e18; 5e22; 7e18; 7e18; 7e18; 8.2e22; 2.6e22; 1.7e21];
% rock viscosities, Pa*s. Domains (unified subduction channel (SC); unified upper
plate; shallow % SC (SSC); fully-locked seismogenic zone (FLSZ); deep SC (DSC);
upper continental crust (UC);
% lower continental crust (LC); western continental lithospheric mantle (WM)).
% Roy & Roydan (2000): 8.2e22; 2.6e22; 1.7e21

Rho          = [2200; 2916; 2200; 2200; 2200; 2700; 3100; 3240];
% rock densities, kg/m3. Domains (unified SC; unified upper plate; SSC; FLSZ; DSC;
UC; LC; WM)

rhow         = 1020;                                  % water density,
kg/m3
Shearm       = [10e9; 45e9; 10e9; 10e9; 10e9; 25e9; 25e9; 67e9];
% elastic shear modulus, Pa. Domains (unified SC; unified upper plate; SSC; FLSZ;
DSC; UC; LC;
% UM)

% USER INPUT - material properties switches
%Visco_new     = [1e19 5e18 1e19 5e18 1e19 5e18 1e19 5e18 1e19 5e18 1e19 5e18
1e19];                                % new viscosity of SC due to climate change (or
only of SSC if DSC is switched on). Each entry
% represents a pulse of viscosity in the SC (or SSC) that travels downdip with velocity
of plate
% subduction. Fill in in chronologic order. = 0 if not applicable
```

```
%Visco_new_step = [200 750 800 850 890 990 1030 1200 1220 1240 1260 1280
1297];
% timestep at which new viscosity (due to climate change) starts to go downdip
(element=0 if
% no change). Fill in in order, related to Visco_new vector

Visco_new = 1e19;
G_new = [10e9 10e9];
Rho_new = [2200 2200];


% USER INPUT - domain geometry
vel_sub_all   = [76; 75; 107; 125; 115; 151]*km/ma;
% orthogonal component of relative convergence rate at 22ºS according to Somoza,
1998 (km/ma equals mm/yr): [current; 0-4.9 Ma; 4.9-10.8 Ma; 10.8-16 Ma; 16-20.2
Ma; 20.2-25.8 Ma]. This is velocity on map view, not along the boundary

vel_sub       = vel_sub_all(4);                    % initial subduction velocity
(timestep 0)
vel_sub = 71*km/ma;
angle_sub     = 23;                                % angle of subduction in
degrees
SC_thick      = 500 / sin(angle_sub/57.2957795);
% this is the apparent SC thickness in the horizontal direction. The numerator is the
real
% thickness, m

lower_limit   = 100*km;
 % depth of lower limit of domain, magnitude without minus sign (m)

farc_xlimit   = 400*km;
% forearc eastern limit on x (m). Maximum allowed by model is 1130km

farc_xlimit_alt = 0;
% altitude above sea level of eastern end of domain

coast_ini_x   = 115*km;
% initial position of coast in x (m). This position may be fixed throughout the run or
not.

inlet_depth   = 7450;
% initial inlet depth in absolute numbers, without signs (m). This position may be
fixed
% throughout the run or not.
```

```
FLSZ_downdip  = -37000;                                % downdip end of FLSZ
(m)
FLSZ_updip    = -25000;                                % updip end of FLSZ (m)
UC_LC_inter   = [8.08; 26.14; 44.28; 59.31; 81.08; 95.71; 114.06;...
            127.56; 135.92; 171.05; 179.59; 189.24; 208.14; 219.56;...
            240.59; 261.38; 279.09; 302.11; 321.54; 339.81;...
            358.7; 376.21; 422.98; 441.56; 528.72; 580.75;...
            612.20; 619.67; 659.63; 676.41; 695.96; 705.26;...
            736.84; 751.61; 765.97; 781.03; 803.88; 846.11;...
            859.41; 885.70; 923.71; 947.59; 981.09; 996.57;...
            1055.39; 1128.81; 1150.98]*1000;
% these are distance from trench (m) to vertices of upper crust-lower crust
discontinuity at 20
% 21ºS (Tassara et al., 2006)

erase_UC_LC=find(UC_LC_inter>farc_xlimit);
UC_LC_inter(erase_UC_LC(2:end))=[];

moho_inter    = [92.53; 95.57; 107.88; 128.40; 195.98; 246.34; 284.41;...
            320.17; 423.01; 501.50; 600.70; 696.89; 820.19; 968.57;...
            1069.20; 1130.28]*1000;
% these are distance from trench (m) to vertices of moho at 20-21ºS (Tassara et al.,
2006)

erase_moho=find(moho_inter>farc_xlimit);
moho_inter(erase_moho(2:end))=[];

UC_LC_depths  = [-7.43905; -6.68083; -5.66846; -5.66846; -4.39952;...
            -0.909959; -0.32032; -1.86166; -0.96326; -5.65846;...
            -4.2499; -5.89322; -8.28509; -8.80277; -9.97495;...
            -9.45205; -7.89728; -8.28509; -13.0953;...
            -6.6149; -5.80428; -9.45205; -8.6184; -8.50053;...
            -13.4508; -11.3047; -5.72878; -10.477; -11.7255;...
            -11.7255; -12.9249; -12.5251; -8.12748; -2.38048;...
            -3.15052; -8.1558; -14.578; -9.62775; -15.2852;...
            -11.3326; -9.08332; -6.57567; -12.5631; -5.08298;...
            -11.3303; -5.99464; -11.3569]*1000;
% these are depths from sea level (m) to vertices of upper crust-lower crust
discontinuity at 20
% 21ºS (Tassara et al., 2006)

UC_LC_depths(erase_UC_LC(2:end))=[];

moho_depths   = [-34.1277; -34.1277; -34.1277; -34.1277; -34.1277;...
            -34.1277; -34.1277; -34.1277; -34.1277; -34.1277;...
```

```
            -34.1277; -34.1277; -34.1277; -34.1277; -34.1277;...
            -34.1277]*1000;

moho_depths(erase_moho(2:end))=[];

bdy3_test  = 0;                                          % horizontal vel of bdy3 fixed =
sub vel

% USER INPUT - domain geometry switches
vel_sub_order = [vel_sub_all(3); vel_sub_all(2); vel_sub_all(1)];
% actual order of vel_sub in time, corresponding to vel_step(i)
%vel_step      = [480 900 1420 2010];
% 0 means no change in vel_sub through time. Otherwise, insert timestep at which
vel_sub
% changes, as many times as changes there are, in order related to vel_sub_all vector

vel_step      = [0 0 0];
% 0 means no change in vel_sub through time. Otherwise, insert timestep at which
vel_sub
% changes, as many times as changes there are, in order related to vel_sub_all vector

move_SC_x     = 2;
% when moving mesh, SC is free horizontally (= 1), or SC is horizontally fixed (= 2)

move_SC_y     = 1;
% if set to 1 it allows the inlet to move vertially, if set to 0 it stays fixed vertically

bdy3_def      = 0;
% in resample_interfaces2, do you want boundary 3 to stay linear (= 0) or to deform
(= 1)?

FLSZ          = 0;                          % 1 activates fully-locked seismogenic zone
(FLSZ)

TransVisco = [1e19 5e19 1e20 5e20 1e21 5e21 1e22];
trans_to_FLSZ = 0;
DSC_indep     = 0;
% 1 activates: climate-derived pulses of viscosity and G do not affect DSC

NSCrun        = 0;        % =1 implies no subduction channel run (otherwise, =0)
NSCrun2=0;
bdy9monot=0;

% USER INPUT - model parameters
dt            = 0.01*ma;                                 % time step duration
```

```
steps         = 1200;                                    % amount of time steps
cpoints_long  = 800;
% number of nodes on long geometry boundaries fed to mesh generator

cpoints_short = 10;
% number of nodes on short geometry boundaries fed to mesh generator

%Elsizes      = [1.02035e3 2*1.02035e5];      % 450,000 elements
%Elsizes      = [2.63e3 5.26e5];              % 200k
%Elsizes      = [18e3 36e5];                  % 55k
Elsizes       = [6.65e3 13.3e5];              % 100k
%Elsizes      = [3.747e3 7.494e5];                % 150k

siplot        = 0;                                    % 1 plots velocity field, 0 doesn't
sisiplot      = 0;                                    % 1 plots meshes, 0 doesn't
sisisiplot    = 0;                                % 1 plots deformed boundaries through
time, 0 does not
plot_test     = 0;                                % 1 plots test plots, 0 does not
remesh        = 0;
% I don't need to 'remesh' before the first time step (changes after 1st tstep)

nip           = 6;                                    % nip - Number of integration
points (6 or higher)
reorder       = 'amd';
% (reorder = 'amd') - AMD reordering, (reorder = 'metis') - METIS reordering

mode          = 'ascii';
% (mode = 'binary') - binary io in mesh generation, (mode = 'ascii')  - ascii  io in mesh
generation

box2          = 1;
% either 1 or 0. Different ways to define the boundaries as an input for Triangle. See
% explanation in generate_mesh_2.m

search_mode   = 'tsearch2-mutils';
% this is the method for finding the old mesh triangle element where a new node falls:
% tsearchn, only for Delaunay triangulations or tsearch2-mutils, good for non-
delaunay
% triangulations

tests         = 0; % inserting 1 activates some tests I've been doing on the interpolation
scheme
more_tests    = 0;% inserting 1 activates some tests I've been doing on the
interpolation scheme
```

```
%==========================================================================
% A.3. MESH GENERATION
%==========================================================================

fprintf(1, 'MESH GENERATION:      ');
tic

% geometry
[GEOMETRY, Geo_id, POINTS_1] = make_geometry(inlet_depth,farc_xlimit,...
                        coast_ini_x,lower_limit,angle_sub,...
                        SC_thick,cpoints_long,cpoints_short,...
                        farc_xlimit_alt);                    % this functions defines the mesh
boundaries

if (sisiplot)
   figure(1)
   plot(GEOMETRY(1,:), GEOMETRY(2,:),'-r.')
   axis([-10000 farc_xlimit+10000 -lower_limit-10000 10000])
   set(gca,'FontSize',18)
   title('initial boundaries','FontSize',20)
   xlabel('meters','FontSize',18)
   ylabel('meters','FontSize',18)
end                                                          % this plots the initial mesh
boundaries

% meshing
[GCOORD, ELEM2NODE, Point_id, Phase_id,Geo_mark, nnod, ndof, nel,...
   Corner_id]  = generate_mesh_2(GEOMETRY,Geo_id,Elsizes,mode,...
                          cpoints_long,cpoints_short, box2);

if (sisiplot)
    figure(2)
    trimesh(ELEM2NODE', GCOORD(1,:), GCOORD(2,:))
    axis([-10000 farc_xlimit+10000 -lower_limit-10000 10000])
    title('initial mesh')
end                                                          % this plots the initial mesh (all the
elements)


fprintf(1, num2str(toc,'%8.6f'));
fprintf(1, ['\n Number of nodes:   ', num2str(nnod)]);
fprintf(1, ['\n Number of elems:   ', num2str(nel),'\n']);
```

```
%==========================================================================
% A.4. INITIALIZATION & GLOBAL STORAGE
%==========================================================================

fprintf(1, 'INITIALIZATION & GLOBAL STORAGE:      ');
tic

Vel        = zeros(ndof*nnod  , 1);
d_istep    = zeros(size(Visco_new_step));
change     = zeros(size(Visco_new_step));

TAU_xx_old      = zeros(nel,nip);
TAU_yy_old      = zeros(nel,nip);
TAU_xy_old      = zeros(nel,nip);
THETA_all       = zeros(nel,nip);

fprintf(1, num2str(toc,'%8.6f'));

%==========================================================================
% A.5. BOUNDARY CONDITIONS
%==========================================================================

fprintf(1, '\nBOUNDARY CONDITIONS:      ');
tic

[Bc_ind, Bc_val, Point_id,TOP_F_BC, Top_force] = set_bcs_flow(GCOORD,...
            ELEM2NODE, Corner_id, Point_id, vel_sub, G,rhow,...
            angle_sub, SC_thick, bdy3_test, NSCrun);
% this sets the essential (velocity) and natural boundaries (force)

fprintf(1, num2str(toc,'%8.6f'));

%==========================================================================
%==========================================================================
% B. PROCESSING
%==========================================================================
```

```
%==========================================================================

for istep=1:steps                                        % main time loop

%    if istep > 11
%       siplot       = 1;
%       sisiplot     = 1;
%       sisisiplot   = 1;
%    end

   fprintf(1, '\nTIME STEP:      ');
   fprintf(1, num2str(istep,'%8.6f'));

   for i=1:size(vel_step,2)
      if vel_step(i) > 0
         if istep >= vel_step(i)
% this loop sets the convergence rate according to parameters above
            vel_sub = vel_sub_order(i);
         end
      end
   end

   for i=1:size(Visco_new_step,2)
      if Visco_new_step(i) > 0
% this loop defines some parameters for later adjustment of viscosity
         if istep >= Visco_new_step(i)
           change(i) = 1;
           d_istep(i) = (istep-Visco_new_step(i)+...
                      1)*dt*vel_sub*tan(angle_sub/57.2957795);
         end
      end
   end



%==========================================================================
   % B.1. MOVE MESH, REMESH AND INTERPOLATE STRESS FIELD

%==========================================================================
```

```
   if(remesh)                    % remeshing takes place at every timestep except
timestep=0
      fprintf(1, '\nREMESHING:      ');

      GCO = GCOORD;                                  % GCO are the old coordinates
of the old mesh
      E2N = ELEM2NODE;
% E2N is the connectivity matrix of the old mesh (previous to re-meshing)
      P_id = Point_id;
      C_id = Corner_id;                               % old flags

      fprintf(1, '\n move old mesh:      ');
      tic

      GCO2 = GCO + DISPL*dt;
% deform the old mesh: GCO2 are the new coordinates (of all nodes) of the old mesh
after they
% are moved

      fprintf(1, num2str(toc,'%8.6f'));

      fprintf(1, '\n resample interfaces:      ');
      tic

      [GEOMETRY, Geo_id] = resample_interfaces2(GCO2,GCO,P_id,C_id,...
                        cpoints_long,cpoints_short,POINTS_1,...
                        istep, bdy3_def,DISPL,dt,move_SC_x,...
                        move_SC_y, angle_sub, bdy9monot);
% resample interfaces (boundaries) to feed into Triangle for remeshing

      fprintf(1, num2str(toc,'%8.6f'));


      % the following smoothes out the corner between marine and
      % continental free surfaces, so that mesh generator does not
      % produce triangles there that are too small

      tr8 = GEOMETRY(:,Geo_id==8);
      tr9 = GEOMETRY(:,Geo_id==9);
      GEOMETRY_s = [tr8(:,end-5:end-3),tr9(:,3:5)];
      fitobj = fit(GEOMETRY_s(1,:)',GEOMETRY_s(2,:)','cubicinterp');

      if (plot_test)
         figure(44)
         plot(fitobj,GEOMETRY_s(1,:)',GEOMETRY_s(2,:)')
```

```
        legend('best')
        hold on
        plot(tr8(1,end-2:end),tr8(2,end-2:end), '*b')
        hold on
        plot(tr9(1,1:2),tr9(2,1:2), '*b')
    end

    y_int_8 = feval(fitobj,tr8(1,end-2:end)');
    y_int_9 = feval(fitobj,tr9(1,1:2)');

    find8 = find(Geo_id==8);
    find9 = find(Geo_id==9);

    GEOMETRY(2,find8(end-2:end)) = y_int_8';
    GEOMETRY(2,find9(1:2)) = y_int_9';

    for i=1:(size(GEOMETRY,2)-1)
        sizeel(i) = sqrt((GEOMETRY(1,i+1)-GEOMETRY(1,i))^2 +...
                    (GEOMETRY(2,i+1)-GEOMETRY(2,i))^2);
    end

    findsmall = find(sizeel<5);
    fgh(istep) = size(findsmall,2);
    GEOMETRY(:,findsmall) = [];
    Geo_id(findsmall) = [];


    % the following arrays store each timestep's domain boundaries

    coordsx_marine{istep} = GEOMETRY(1,Geo_id==9);
    coordsy_marine{istep} = GEOMETRY(2,Geo_id==9);
    coordsx_nonmarine{istep} = GEOMETRY(1,Geo_id==8);
    coordsy_nonmarine{istep} = GEOMETRY(2,Geo_id==8);

    if (sisisiplot)
        figure(4)
        plot(GEOMETRY(1,:), GEOMETRY(2,:),'-b.')
        axis([-10000 farc_xlimit+10000 -lower_limit-10000
max(GEOMETRY(2,:))+10000])
        set(gca,'FontSize',18)
        title('initial boundaries','FontSize',20)
        xlabel('meters','FontSize',18)
        ylabel('meters','FontSize',18)
        title('deformed boundaries')
        hold on
```

```
        plot([0:10000:400000], zeros(1,length([0:10000:400000])),'-r.')
% this plots the deformed boundaries, one after the other as dts pass
    end


    % remeshing

    fprintf(1, '\n remesh:      ');
    tic

    [GCOORD,ELEM2NODE,Point_id,Phase_id,Geo_mark,nnod,ndof,nel,...
       Corner_id] = generate_mesh_2(GEOMETRY,Geo_id,Elsizes,mode,...
                           cpoints_long,cpoints_short, box2);

    % GCOORD is now the new coordinates (after moved) of the new mesh,
    % meaning: old mesh is moved, then remeshed, then GCOORD is the
    % coordinates of these new nodes
    % ELEM2NODE is the connectivity matrix of this new mesh

    fprintf(1, num2str(toc,'%8.6f'));


    % ------------------------------------------------------------- %
    % ----------- tests to check quality of mesh ------------------- %

    if (more_tests)

       gest12         = sqrt((GCOORD(1,ELEM2NODE(1,:)) -...
                        GCOORD(1,ELEM2NODE(2,:))).^2 +...
                        (GCOORD(2,ELEM2NODE(1,:)) -...
                        GCOORD(2,ELEM2NODE(2,:))).^2);
       gest12_min      = min(gest12);
       gest23         = sqrt((GCOORD(1,ELEM2NODE(2,:)) -...
                        GCOORD(1,ELEM2NODE(3,:))).^2 +...
                        (GCOORD(2,ELEM2NODE(2,:)) -...
                        GCOORD(2,ELEM2NODE(3,:))).^2);
       gest23_min      = min(gest23);
       gest31         = sqrt((GCOORD(1,ELEM2NODE(3,:)) -...
                        GCOORD(1,ELEM2NODE(1,:))).^2 +...
                        (GCOORD(2,ELEM2NODE(3,:)) -...
                        GCOORD(2,ELEM2NODE(1,:))).^2);
       gest31_min      = min(gest31);
       gest_min(istep) = min([gest12_min, gest23_min, gest31_min]);

       if gest_min == gest12_min
```

```
   ind = find(gest12 == gest12_min);
   dfx = GCOORD(1,ELEM2NODE(1,ind));
   dfy = GCOORD(2,ELEM2NODE(1,ind));
elseif gest_min == gest23_min
   ind = find(gest23 == gest23_min);
   dfx = GCOORD(1,ELEM2NODE(1,ind));
   dfy = GCOORD(2,ELEM2NODE(1,ind));
else
   ind = find(gest31 == gest31_min);
   dfx = GCOORD(1,ELEM2NODE(1,ind));
   dfy = GCOORD(2,ELEM2NODE(1,ind));
end

h1(istep)    = size(find(GEOMETRY(1,Geo_id==8)<140000),2);
h2(istep)    = size(find(GEOMETRY(1,Geo_id==9)>120000),2);
h3(istep)    = size(find(GCOORD(1,Point_id==8)<140000),2);
h4(istep)    = size(find(GCOORD(1,Point_id==9)>120000),2);
rate8(istep) = h3(istep)/h1(istep);
rate9(istep) = h4(istep)/h2(istep);
abs8(istep)  = h3(istep) - h1(istep);
abs9(istep)  = h4(istep) - h2(istep);

q = MeshQuality(GCOORD',ELEM2NODE');

q_min(istep-1)    = min(q);
q_max(istep-1)    = max(q);
qfind             = find(q <= 0.5);
qbad_num(istep-1) = size(qfind,1);
x_coords_q_min    = GCOORD(1,ELEM2NODE(1,qfind));
y_coords_q_min    = GCOORD(2,ELEM2NODE(1,qfind));

fprintf(1, '\n min q:      ');
fprintf(1, num2str(q_min(istep-1),'%8.6f'));

fprintf(1, '\n # bad q:      ');
fprintf(1, num2str(qbad_num(istep-1),'%8.6f'));

for i=1:qbad_num(istep-1)
   fprintf(1, '\n x-coord of node 1 of element i with min q:      ');
   fprintf(1, num2str(x_coords_q_min(i),'%8.6f'));
   fprintf(1, '\n y-coords of node 1 of element i with min q:      ');
   fprintf(1, num2str(y_coords_q_min(i),'%8.6f'));
end

fprintf(1, '\n smallest element side in meters:      ');
```

```
        fprintf(1, num2str(gest_min(istep),'%8.6f'));

        fprintf(1, '\n x-coord of node 1 of triangle(s) with smallest side in meters:     ');
        fprintf(1, num2str(dfx,'%8.6f'));

        fprintf(1, '\n y-coord of node 1 of triangle(s) with smallest side in meters:     ');
        fprintf(1, num2str(dfy,'%8.6f'));

        fprintf(1, '\n      ');
        fprintf(1, num2str(toc,'%8.6f'));

        fprintf(1, '\n x rise in bdy8 nodes close to corner:      ');
        fprintf(1, num2str(rate8(istep),'%8.6f'));

        fprintf(1, '\n absolute rise in bdy8 nodes close to corner:      ');
        fprintf(1, num2str(abs8(istep),'%8.6f'));

        fprintf(1, '\n x rise in bdy9 nodes close to corner:      ');
        fprintf(1, num2str(rate9(istep),'%8.6f'));

        fprintf(1, '\n absolute rise in bdy9 nodes close to corner:      ');
        fprintf(1, num2str(abs9(istep),'%8.6f'));

        fprintf(1, '\n # of post-remesh bdy8 nodes close to corner:      ');
        fprintf(1, num2str(h3(istep),'%8.6f'));

        fprintf(1, '\n # of post-remesh bdy9 nodes close to corner:      ');
        fprintf(1, num2str(h4(istep),'%8.6f'));

    end

    if (sisiplot)
        figure(5)
        trimesh(ELEM2NODE', GCOORD(1,:), GCOORD(2,:))
        axis([-10000 farc_xlimit+10000 -lower_limit-10000
max(GEOMETRY(2,:))+10000])
        set(gca,'FontSize',18)
        title('deformed mesh','FontSize',20)
        xlabel('meters','FontSize',18)                    % this plots the deformed
mesh
        ylabel('meters','FontSize',18)
    end


    % ------------------------------------------------------------ %
```

```
% ---------- tests to check quality of mesh ---------- %

% in the next lines I want to see how the convex hull looks for GCO
% and GCO2

if (tests)

   GCO_temp_x  = (GCO(1,:))';
   GCO_temp_y  = (GCO(2,:))';
   GCO2_temp_x = (GCO2(1,:))';
   GCO2_temp_y = (GCO2(2,:))';

   CH_GCO  = convhull(GCO_temp_x, GCO_temp_y);
   CH_GCO2 = convhull(GCO2_temp_x, GCO2_temp_y);

   figure(20)
   plot(GCO_temp_x(CH_GCO),GCO_temp_y(CH_GCO),'r-')
   title('GCO convex hull','FontSize',20)
   axis([-2 42 -3.5 0.5] * 10000)

   figure(21)
   plot(GCO2_temp_x(CH_GCO2),GCO2_temp_y(CH_GCO2),'r-')
   title('GCO2 convex hull','FontSize',20)
   axis([-2 42 -3.5 0.5] * 10000)


% in the next lines I try tsearchn on GCO2 just for a group of NaN
% GCOORD nodes located inside the "square"

   GCOORD_x     = GCOORD(1,:);
   GCOORD_y     = GCOORD(2,:);
   kx           = find(2e5<GCOORD_x & GCOORD_x<3e5);
   GCOORD_trial = [GCOORD_x(kx); GCOORD_y(kx)];
   GCOORD_x     = GCOORD_trial(1,:);
   GCOORD_y     = GCOORD_trial(2,:);
   ky           = find(-1.5e4<GCOORD_y & GCOORD_y<-1e4);
   GCOORD_trial = [GCOORD_x(ky); GCOORD_y(ky)];

   nnodel_r       = nip;
   [~,~,~,~,~,Ind] = remesh_F_TAU(GCOORD, ELEM2NODE, GCO2, E2N,...
                    nnodel_r, TAU_xx_old, TAU_xy_old,...
                    TAU_yy_old, nip, nel, search_mode);
end
```

```
      % interpolate stresses

      fprintf(1, '\n INTERP STRESSES:      ');
      fprintf(1, '\n pre-searching:      ');
      tic

      nnodel_r = nip;
      [GIP_xF,GIP_yF, TAU_xx_old, TAU_xy_old, TAU_yy_old, Ind] = ...
         remesh_F_TAU(GCOORD, ELEM2NODE, GCO2, E2N, nnodel_r,...
                  TAU_xx_old, TAU_xy_old, TAU_yy_old, nip, nel,...
                  search_mode);


      % reset boundary conditions

      fprintf(1, '\n boundary conditions:      ');
      tic

      [Bc_ind, Bc_val, Point_id, TOP_F_BC, Top_force] =...
         set_bcs_flow(GCOORD, ELEM2NODE, Corner_id, Point_id,...
                   vel_sub, G,rhow,angle_sub, SC_thick, bdy3_test, NSCrun);

      fprintf(1, num2str(toc,'%8.6f'));


      clear 'E2N' 'GCO' 'P_id' 'C_id' 'GCO2'

   end

   remesh = 1;                                    % start remeshing after the first
time step




%=================================================================
==========
   % B.2. MECHANICAL SOLVER

%=================================================================
==========

   % mechanical2d.m has been modified from Lars' version to include
   % viscoelasticity.
```

```
   fprintf(1, '\nFLOW SOLUTION:      ');

   [Vel, PRES_IP, TAU_xx, TAU_yy, TAU_xy, STRAIN_xx, STRAIN_yy,...
    STRAIN_xy, E2all, Mu_all, W_xy, THETA_all] =...
         mechanical2d(moho_depths, moho_inter, UC_LC_inter,...
                  UC_LC_depths, DSC_indep, FLSZ_updip,...
                  FLSZ_downdip, FLSZ, change,inlet_depth,...
                  Visco_new, G_new, Rho_new, d_istep, ELEM2NODE, Phase_id,...
                  GCOORD, Rho, Visco,G, Bc_ind, Bc_val, nip,...
                  reorder, TOP_F_BC, Top_force, Shearm, dt,...
                  TAU_xx_old, TAU_xy_old, TAU_yy_old,...
                  THETA_all, more_tests, trans_to_FLSZ,...
                  TransVisco, NSCrun, NSCrun2);

   DISPL       = reshape(Vel,[ndof,nnod]);



%=========================================================================
%=========================================================================
   % C. POST-PROCESSING
%=========================================================================
%=========================================================================

   fprintf(1, '\nPOST-PROCESSING:      ');
   tic


%=========================================================================
   % C.1. STRESS ROTATION
%=========================================================================


   % STRESS ROTATION

   TAU_xx = TAU_xx + 2*W_xy.*TAU_xy*dt;
   TAU_yy = TAU_yy - 2*W_xy.*TAU_xy*dt;
   TAU_xy = TAU_xy - (TAU_xx - TAU_yy).*W_xy*dt;

   TAU_xx_old     = TAU_xx;
   TAU_yy_old     = TAU_yy;
   TAU_xy_old     = TAU_xy;
```

```
%=================================================================
   % C.2. TIME-DEPENDENT PLOTS
%=================================================================

   if (siplot)

   %PLOT PRESSURE
     %plotting
   %MAPPING IPS TO NODES
   nel = size(ELEM2NODE,2);
   EL2N      = zeros(nel,3);
   GCOORD_N  = zeros(2,nel*3);
%     TAU_xxn   = zeros(3,nel);
   Pres_n    = zeros(3,nel);
%     Mu_n      = zeros(3,nel);
%     Grav      = zeros(3,nel);
%     Diss      = zeros(3,nel);
%     E2_n      = zeros(3,nel);
%     Phase_n   = zeros(3,nel);
    Vyy_n       = zeros(1,nel*3);
%     Rho_n     = zeros(3,nel);

%     PHASE_IP = repmat(Phases', [1,6]);
%     DENS_IP = repmat(Dens, [1,6]);
   EL2N(1,:) = 1:3;
   nip = 6;
   nnodel = 6;
   [IP_X, IP_w]    = ip_triangle(nip);
   [   Nbig]    = shp_triangle(IP_X, nnodel);

         Vyy = DISPL(2,:);
   for i=1:nel
      is        = (i-1)*3+1; ie = (i-1)*3+3;
      GCOORD_N(:,is:ie) = GCOORD(:,ELEM2NODE([1 2 3],i));
      EL2N(i,:) = is:ie;

      Vyy_n(is:ie) =Vyy(ELEM2NODE([1 2 3],i));
%        Dummy      = Nbig'\TAU_xx(i,:)';
%        TAU_xxn(:,i)= Dummy(1:3);
      Dummy      = Nbig'\PRES_IP(i,:)';
      Pres_n(:,i)= Dummy(1:3);
%        Dummy      = Nbig'\WGRAV(i,:)';
%        Grav(:,i)= Dummy(1:3);
```

```
%        Dummy      = Nbig'\WDISS(i,:)';
%        Diss(:,i)= Dummy(1:3);
%        Dummy      = Nbig'\E2all(i,:)';
%        E2_n(:,i)= Dummy(1:3);
%        Dummy      = Nbig'\PHASE_IP(i,:)';
%        Phase_n(:,i)= Dummy(1:3);
%        Dummy      = Nbig'\DENS_IP(i,:)';
%        Rho_n(:,i)= Dummy(1:3);
%
   end


      figure(6), clf
   %
patch('faces',EL2N,'vertices',GCOORD_N'/km,'facevertexcdata',Pres_n(:),'FaceColor','
flat')

patch('faces',EL2N,'vertices',GCOORD_N'/km,'facevertexcdata',Vyy_n(:)*ma/km,'Fac
eColor','flat')
      shading interp
      axis tight
       title('vertical velocity (km/ma)')
   %    title('Pressure (Pa) - channel 1e17 Pa-s, matrix 1e21Pa-s')
      xlabel('Distance (km)')
      ylabel('Depth (km)')
      colorbar
     % caxis([0 1e9])
      caxis([min(Vyy_n(:)*ma/km) max(Vyy_n(:)*ma/km)])

        np = 50;
      xs = linspace(min(GCOORD(1,:)),max(GCOORD(1,:)),np);
      ys = linspace(min(GCOORD(2,:)),max(GCOORD(2,:)),np);



      [XX,YY] = meshgrid(xs,ys); % mesh to resample velocity for a uniform spaced
quiver plot
      GCOORD_O = [XX(:)';YY(:)']; % uniformly spaced points for quiver plot



      size_GCOORD_O = size(GCOORD_O(1,:));
      ELEM2NODE = uint32(ELEM2NODE);
      [tris,~,~] = tsearch2(GCOORD,ELEM2NODE(1:3,:),GCOORD_O);
      tris = reshape(tris,size_GCOORD_O);
```

```
      isnan_vec = find(tris==0);
      tris(isnan_vec)=1;
   %     tris_all          =
tsearch(GCOORD_N(1,:),GCOORD_N(2,:),double(EL2N),GCOORD_O(1,:),GCOO
RD_O(2,:));

      [Vxx_int]                    =
remesh_val(tris,GCOORD,GCOORD_O,DISPL(1,:),ELEM2NODE);
% (tris,GCOORD,gcoordO,Phi,double(ELEM2NODE)');
      [Vyy_int]                    =
remesh_val(tris,GCOORD,GCOORD_O,DISPL(2,:),ELEM2NODE);
% (tris,GCOORD,gcoordO,Phi,double(ELEM2NODE)');
      Vxx_int(isnan_vec) = 0;
      Vyy_int(isnan_vec) = 0;
      figure(6), hold on, quiver(XX(:)'/km, YY(:)'/km, Vxx_int, Vyy_int, 'k')


      axis tight
   end

   fprintf(1, num2str(toc,'%8.6f'));

end
```

C.7 Free surface elevation results at the eastern model boundary

Our subduction channel flow model results show a sinuous topographic feature, with a wavelength of 80 km and a magnitude of 500 m from the regional elevation level, at about 270 to 350 km from the trench (see Fig. 4-3a in Chapter 4). This feature is persistent throughout the full range of modeled subduction channel parameters (see Fig. 4-3a in Chapter 4). In order to assert that this is a robust feature associated with subduction channel flow, it is necessary to discard the possibility that it is associated to any geometrical element of the model domain. In the following paragraphs we look at the Moho topography, the depth extent of the subduction

channel, and the eastern model boundary, the three a priori main candidates for model parameters that may influence its formation.

Two simulations were performed keeping all physical and geometrical parameters constant except the topography of the Moho (Fig. C1a). One of these two simulations had a Moho topography as defined in Table 4-1 of Chapter 4, which is the configuration used in all the simulations in Chapter 4, while the other one was run with a flat topography. In both cases the sinuous topographic feature described in the preceding paragraph remained unchanged both in wavelength and magnitude (Fig. C1a).

Three simulations were performed keeping all physical and geometrical parameters constant except the subduction channel depth (Fig. C1b). Irrespective of how deep between 80 and 100 km below sea level the subduction channel reached, the sinuous topographic feature remained unchanged both in wavelength and magnitude (Fig. C1b).

Finally, five simulations were performed keeping all physical and geometrical parameters constant except the eastern limit of the model domain, which varied between 315 and 715 km from the initial trench position (Fig. C1c). No change in magnitude, wavelength or position with respect to the trench is observed (Fig. C1c). Note that in all cases the elevation of this feature's positive bulge is the maximum elevation attained in the forearc, and that east of that the elevations go down, albeit not monotonically, especially for the 715 km eastern boundary run (Fig. C1c).

Thus, it seems that this topographic feature is indeed robust and arises from the dynamics of subduction channel flow.

C.8 Supplementary figures

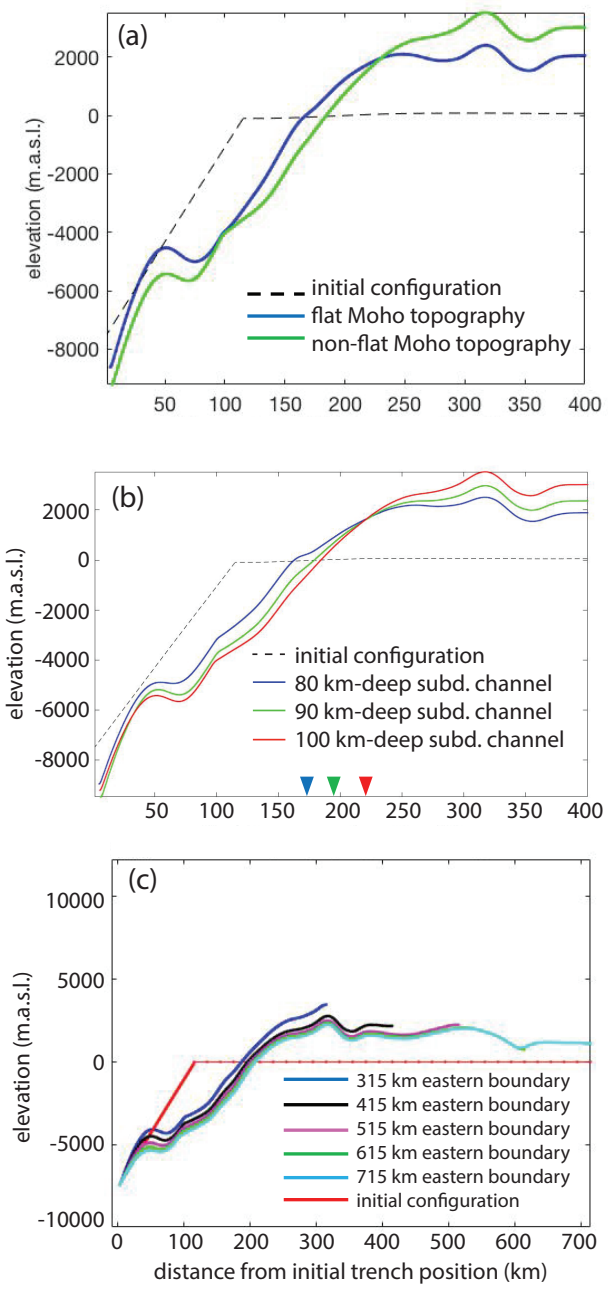

Figure C1. (a) Subduction channel flow under different Moho configurations. Subduction channel is 500 m thick and has a viscosity of 7 x10^{18} Pa s. Total run time is 12 myr. Please see Table 4-1 for non-flat Moho configuration. (b) Subduction channel flow under different depths of the subduction channel, which is 500 m thick and has a viscosity of 7 x10^{18} Pa s. Color arrows indicate the horizontal reach of the subduction channel. (c) Subduction channel flow within a model domain with differing eastern boundary extent. The subduction channel is 500 m thick and has a viscosity of 5 x10^{18} Pa s. In all cases see Table 4-1 for all other model parameters.