0.0.1 Inner core convection

This section was contributed by Juliane Dannberg, and the model setup was inspired by discussions with John Rudge.

This is an example of convection in the Earth's inner core. The model is based on a spherical geometry. Three main particularities are constitutive of this inner core dynamics modelling: it consists of a self-gravitating sphere where the gravity increases linearly from zero at the center; the boundary conditions are permeable (material can cross the inner core boundary. See [**?**]); it's characterized by a single material with temperature dependent density that makes it unstably stratified as temperature increases towards the center of the core.

The setup is analogous to the models described in [**?**], and all material properties are chosen in a way so that the equations are non-dimensional.

The required heating model and changes to the material model are implemented in a shared library (`cookbooks/inner_core_convection/inner_core_convection.cc`).

In their non-dimensional form, the equations are

$$\nabla \cdot \sigma = -Ra T \mathbf{g}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

$$\left(\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right) - \nabla^2 T = H, \tag{3}$$

where Ra is the Rayleigh number and H is the 'source term'. The term H is an artificial one, that arises from the time-variation of the adiabatic temperature in the outer core. Following Deguen2013, with the proposed adimensionalization, $H = 6$.

Thermal evolution of the inner core. During the Earth evolution, the temperature in the core decreases due to secular cooling. The isentropic temperature profile is used as a reference temperature, and when making the equations non-dimensional, a term emerges in the energy conservation equation, mathematically similar to a source term. The dimensional value of this source term is used in the Rayleigh number, as it is in the temperature scale. A slow time-variation of the adiabatic temperature will be associated to a negative source term, and a strongly stratified inner core ($Ra < 0$). A positive value will be associated to a positive Ra number, which indicates a convectively unstable inner core if the Rayleigh number is large enough.

Mechanical boundary. The mechanical boundary conditions for the inner core are tangential stress-free and continuity of the normal stress at the inner-outer core boundary. The inner core boundary is a freezing/melting boundary, and thus can't be considered completely impermeable. For the non-dimensional equations, that means that we can define a "phase change number" $\mathcal{P}$ so that the normal stress at the boundary is $-\mathcal{P} u_r$ with the radial velocity u_r. This number characterizes the resistance to phase change at the boundary, with $\mathcal{P} \to \infty$ corresponding to infinitely slow melting/freezing (or a free slip boundary), and $\mathcal{P} \to 0$ corresponding to instantaneous melting/freezing (or a zero normal stress, corresponding to an open boundary).

In the weak form, this results in boundary conditions of the form of a surface integral:

$$\int_S \mathcal{P} (\mathbf{u} \cdot \mathbf{n})(\mathbf{v} \cdot \mathbf{n}) dS,$$

with the normal vector $\mathbf{n}$.

This phase change term is added to the matrix in the `cookbooks/inner_core_convection/inner_core_assembly.cc` plugin by using a signal (as described in Section **??**). The signal connects the function `set_assemblers_phase_boundary`, which is only called once at the beginning of the model run. It creates the new assembler `PhaseBoundaryAssembler` for the boundary faces of the Stokes system and adds it to the list of assemblers executed in every time step. The assembler contains the function `phase_change_boundary_conditions` that loops over all faces at the model boundary, queries the value of $\mathcal{P}$ from the material model, and adds the surface integral given above to the matrix:

```
#include <aspect/simulator_access.h>
#include <aspect/global.h>
#include <aspect/simulator.h>
#include <aspect/assembly.h>

#include <deal.II/base/quadrature_lib.h>
#include <deal.II/fe/fe_values.h>


#include "inner_core_convection.cc"

namespace aspect
{
  /**
   * A new assembler class that implements boundary conditions for the
   * normal stress and the normal velocity that take into account the
   * rate of phase change (melting/freezing) at the inner-outer core
   * boundary. The model is based on Deguen, Alboussiere, and Cardin
   * (2013), Thermal convection in EarthâĂŹs inner core with phase change
   * at its boundary. GJI, 194, 1310-133.
   *
   * The mechanical boundary conditions for the inner core are
   * tangential stress-free and continuity of the normal stress at the
   * inner-outer core boundary. For the non-dimensional equations, that
   * means that we can define a 'phase change number' $\mathcal{P}$ so
   * that the normal stress at the boundary is $-\mathcal{P} u_r$ with
   * the radial velocity $u_r$. This number characterizes the resistance
   * to phase change at the boundary, with $\mathcal{P}\rightarrow\infty$
   * corresponding to infinitely slow melting/freezing (free slip
   * boundary), and $\mathcal{P}\rightarrow0$ corresponding to
   * instantaneous melting/freezing (zero normal stress, open boundary).
   *
   * In the weak form, this results in boundary conditions of the form
   * of a surface integral:
   * $$\int_S \mathcal{P} (\mathbf u \cdot \mathbf n) (\mathbf v \cdot \mathbf n) dS,$$,
   * with the normal vector $\mathbf n$.
   *
   * The function value of $\mathcal{P}$ is taken from the inner core
   * material model.
   */
  template <int dim>
  class PhaseBoundaryAssembler :
    public aspect::internal::Assembly::Assemblers::AssemblerBase<dim>,
    public SimulatorAccess<dim>
  {

    public:

      virtual
      void
      phase_change_boundary_conditions (const typename DoFHandler<dim>::active_cell_iterator &cell,
                                        const unsigned int                                    face_no,
                                        internal::Assembly::Scratch::StokesSystem<dim>       &scratch,
                                        internal::Assembly::CopyData::StokesSystem<dim>      &data) const
      {
        const Introspection<dim> &introspection = this->introspection();
        const FiniteElement<dim> &fe            = this->get_fe();
        const unsigned int stokes_dofs_per_cell = data.local_dof_indices.size();
        const unsigned int n_q_points           = scratch.face_finite_element_values.n_quadrature_points;

        //assemble force terms for the matrix for all boundary faces
        if (cell->face(face_no)->at_boundary())
          {
            scratch.face_finite_element_values.reinit (cell, face_no);

            for (unsigned int q=0; q<n_q_points; ++q)
              {
                const double P = dynamic_cast<const MaterialModel::InnerCore<dim>&>
                                 (this->get_material_model()).resistance_to_phase_change
                                 .value(scratch.material_model_inputs.position[q]);

                for (unsigned int i = 0, i_stokes = 0; i_stokes < stokes_dofs_per_cell; /*increment at end of
              ↪ loop*/)
                  {
                    if (introspection.is_stokes_component(fe.system_to_component_index(i).first))
                      {
                        scratch.phi_u[i_stokes] = scratch.face_finite_element_values[introspection
                                                                                     .extractors.velocities].
              ↪ value(i, q);
                        ++i_stokes;
                      }
                    ++i;
                  }

                const Tensor<1,dim> normal_vector = scratch.face_finite_element_values.normal_vector(q);
                const double JxW = scratch.face_finite_element_values.JxW(q);

                // boundary term: P*u*n*v*n*JxW(q)
                for (unsigned int i=0; i<stokes_dofs_per_cell; ++i)
                  for (unsigned int j=0; j<stokes_dofs_per_cell; ++j)
                    data.local_matrix(i,j) += P *
                                              scratch.phi_u[i] *
                                              normal_vector *
                                              scratch.phi_u[j] *
                                              normal_vector *
                                              JxW;
              }
          }
      }
  };

  template <int dim>
  void set_assemblers_phase_boundary(const SimulatorAccess<dim> &simulator_access,
                                     internal::Assembly::AssemblerLists<dim> &assemblers,
                                     std::vector<dealii::std_cxx11::shared_ptr<
                                     internal::Assembly::Assemblers::AssemblerBase<dim> > > &assembler_objects
              ↪ )
  {
    AssertThrow (dynamic_cast<const MaterialModel::InnerCore<dim>*>
                 (&simulator_access.get_material_model()) != 0,
                 ExcMessage ("The phase boundary assembler can only be used with the "
                             "material model 'inner core material'!"));

    std_cxx11::shared_ptr<PhaseBoundaryAssembler<dim> > phase_boundary_assembler
    (new PhaseBoundaryAssembler<dim>());
    assembler_objects.push_back (phase_boundary_assembler);

    // add the terms for phase change boundary conditions
    assemblers.local_assemble_stokes_system_on_boundary_face
    .connect (std_cxx11::bind(&PhaseBoundaryAssembler<dim>::phase_change_boundary_conditions,
                              std_cxx11::cref (*phase_boundary_assembler),
                              std_cxx11::_1,
                              std_cxx11::_2,
                              // discard pressure_scaling,
                              // discard rebuild_stokes_matrix,
                              std_cxx11::_5,
                              std_cxx11::_6));


  }
}

template <int dim>
void signal_connector (aspect::SimulatorSignals<dim> &signals)
{
  signals.set_assemblers.connect (&aspect::set_assemblers_phase_boundary<dim>);
}

ASPECT_REGISTER_SIGNALS_CONNECTOR(signal_connector<2>,
                                  signal_connector<3>)
```

Instructions for how to compile and run models with a shared library are given in Section **??**.

Governing parameters. Analyzing Equations (1)–(3) reveals that two parameters determine the dynamics of convection in the inner core: the Rayleigh number Ra and the phase change number $\mathcal{P}$. Three main areas can be distinguished: the stable area, the plume convection area and the translation mode of convection area (Figure 1). For low Rayleigh numbers (below the critical value Ra_c), there is no convection and thermal diffusion dominates the heat transport. However, if the inner core is convectively unstable ($Ra>Ra_c$), the convection regime depends mostly on $\mathcal{P}$. For low $\mathcal{P}$ (<29), the convective translation mode dominates, where material freezes at one side and melts at the other side, so that the velocity field is uniform, pointing from the freezing to the melting side. Otherwise, at high $\mathcal{P}$ (>29), convection takes the usual form of plume convection. In this case, there is almost no melting or solidification associated with this mode. At intermediate values of P, the first unstable mode is a linear combinaison of the high-$\mathcal{P}$ convection mode and of the small-$\mathcal{P}$ translation mode.

Changing the values of Ra and $\mathcal{P}$ in the input file allows switching between the different regimes. The Rayleigh number can be changed by adjusting the magnitude of the gravity:

```
# The gravity has its maximum value at the boundary of inner and
# outer core, and decreases approximately linearly to zero towards
# the center of the core.
# The Rayleigh number used in the model is given by the magnitude
# of the gravity at the inner core/outer core boundary.
subsection Gravity model
  set Model name = radial linear

  subsection Radial linear
    set Magnitude at surface = 2     # <-- Ra
  end
end
```

The phase change number is implemented as part of the material model, and as a function that can depend on the spatial coordinates and/or on time:

```
subsection Material model
  set Model name = inner core material

# The 'inner core material' model also contains a function that
# represents the resistance to melting/freezing at the inner core
# boundary.
# For P-->inifinity, the boundary is a free slip boundary, and for
# P-->0, the boundary is an open boundary (with zero normal stress).
  subsection Inner core
    subsection Phase change resistance function
      set Variable names      = x,y,z
```

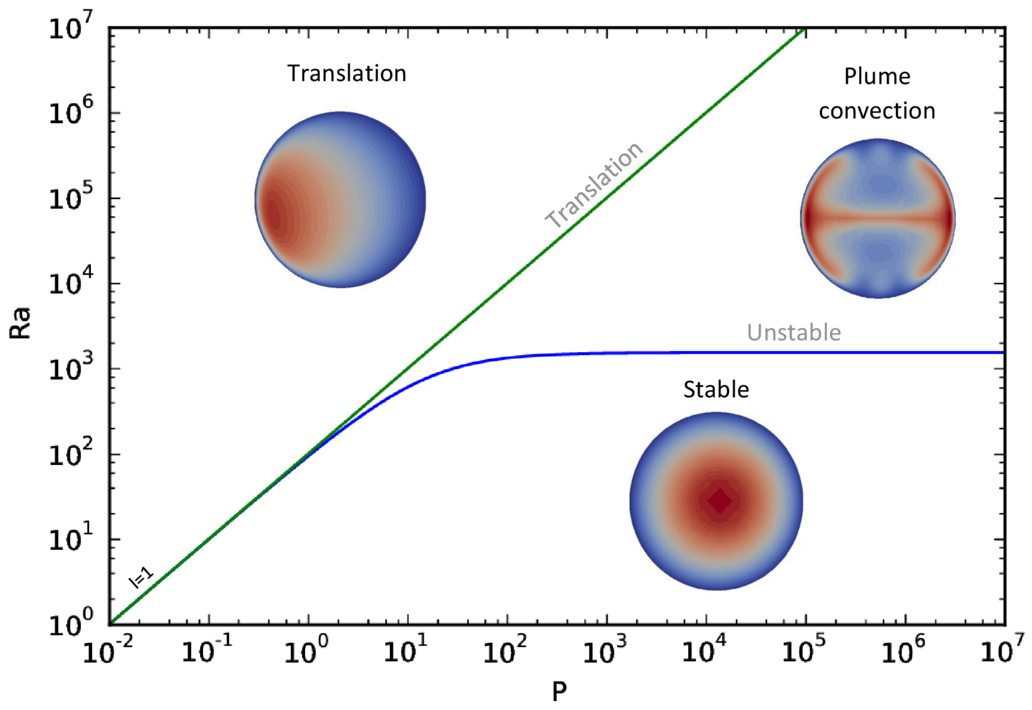

Figure 1: Stability diagram for convection in a sphere with phase change at its outer boundary. The stability curves for the first unstable mode (l=1) and the translation are obtained from [**?**].

```
      set Function expression = 1e-2     # <-- P
    end
  end
end
```

Figure 2 shows examples of the three regimes with $Ra = 10^5, \mathcal{P} = 10^4$ (plume convection), $Ra = 10^2, \mathcal{P} = 10^-1$ (translation), $Ra = 1, \mathcal{P} = 1$ (no convection).

Mesh refinement. In the particular case of the inner core dynamics, two main type of mesh refinement could be used. For simulations in the translative mode (Figure 1), the particular boundary implies that a finest refinement at the outer boundary is more appropriate. The boundary is permeable, allowing melting or freezing and transfer of material through the boundary. The temperature is set at 1 et the boundary, and for the translation case, a large temperature gradient is imposed at the boundary layer. For this, a specific refinement has to be set for defining correctly the boundary layer at the inner core boundary. For simulations in plume convection, an adaptative mesh-refinement (with the temperature field for exemple) is more adapted, otherwise, the plumes are not enough defined.

In order to have a mesh that is much finer at the outer boundary than in the center of the domain, this expression for the mesh refinement subsection can be used in the input file :

```
subsection Mesh refinement
  set Initial global refinement          = 1
  set Initial adaptive refinement        = 5
  set Strategy                           = minimum refinement function
  set Time steps between mesh refinement = 0

  subsection Minimum refinement function
```

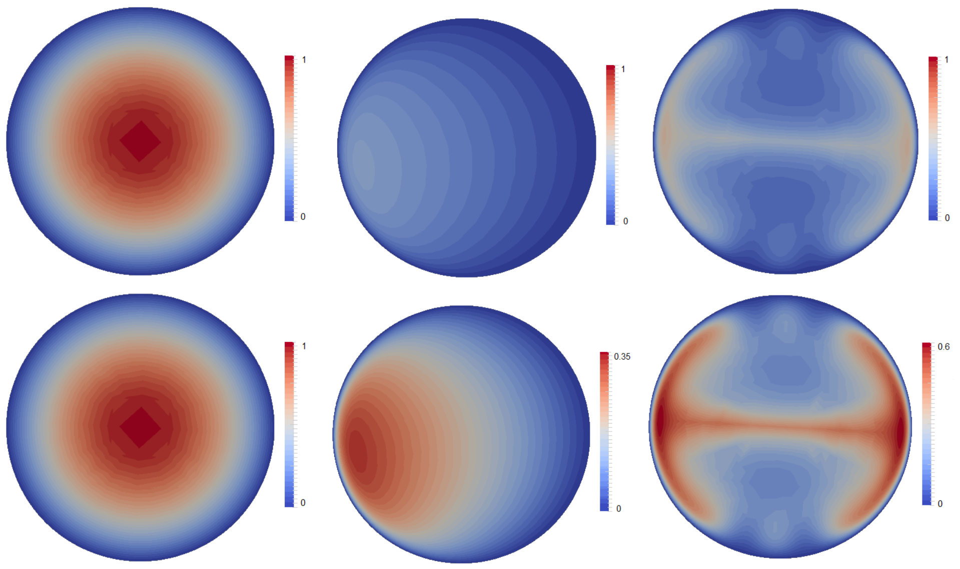

Figure 2: Convection regimes in the inner core for different values of Ra and $\mathcal{P}$. From left to right: no convection, translation, plume convection; the 2D slices of the top are with the default temperature scale while at the bottom an adaptative scale is used.

```
    set Variable names = depth, phi, theta
    set Function expression = if(depth>0.1,if(depth>0.2,1,5),6)
  end
end
```

The sum of the "Initial global refinement" and the "Initial adaptative refinement" is always the maximum refinement of the model. In order to have a finest mesh with a refinement of 6 (like for the exemple above), the sum of those two parameters should be 6. The "Initial global refinement" is used to start with an uniform mesh, and then it does a number of mesh refinement step equal to the "Initial adaptive refinement". For higher Rayleigh numbers, the mesh has to be better refine, especially in the thermal boundary layers. This can be achieve by changing the first term in the parenthesis, which corresponds to the pourcentage of the sphere you want to have a better refinment.

Scaling laws. In addition, [**?**] give scaling laws for the velocities in each regime derived from linear stability analysis, and show how numerical results compare to them. In the regimes of low $\mathcal{P}$, translation will start at a critical ratio of Rayleigh number and phase change number $\frac{Ra}{\mathcal{P}} = \frac{175}{2}$ with steady-state translation velocities being zero below this threshold and tending to $v_0 = \frac{175}{2}\sqrt{\frac{6}{5}\frac{Ra}{\mathcal{P}}}$ going towards higher values of $\frac{Ra}{\mathcal{P}}$. In the same way, translation velocities will decrease from v_0 with increasing $\mathcal{P}$, with translation transitioning to plume convection at $\mathcal{P} \sim 29$. Both trends are shown in Figure 3 and can be compared to Figure 8 and 9 in [**?**].

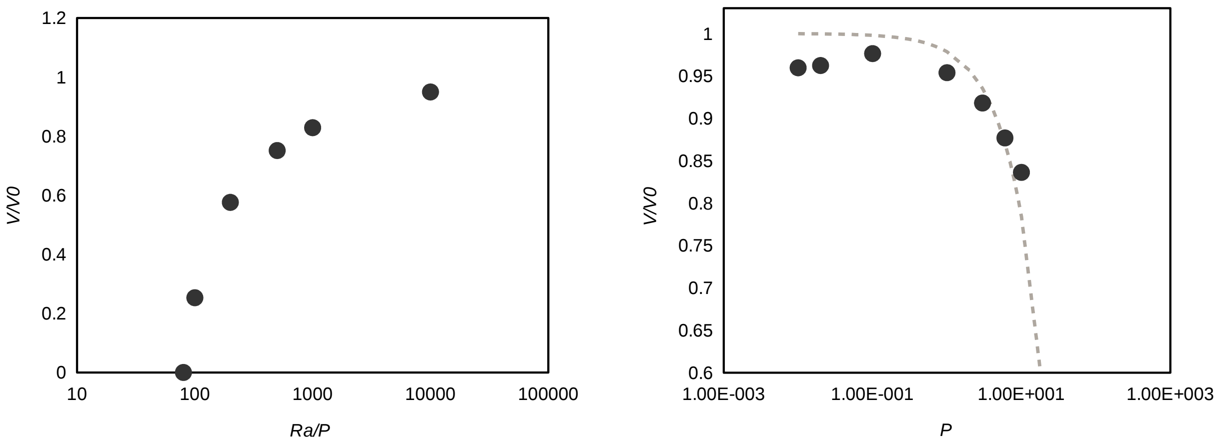

Figure 3: Translation rate (approximated by the average of the velocity component in the direction of translation), normalized to the low $\mathcal{P}$ limit estimate given in [?], as a function of $\frac{Ra}{\mathcal{P}}$ for $\mathcal{P} = 10^{-2}$ (left) and as a function of $\mathcal{P}$ for $Ra = 10^5$ (right). The dashed gray line gives the translation velocity predicted in the limit of low $\mathcal{P}$. Disagreement for larger values of $\mathcal{P}$ indicates that higher order terms (not included in the low $\mathcal{P}$ approximation) become important. Additionally, differences between the analytical and numerical model might be the result of limited resolution (only 12 elements in radial direction).