VENUSS User Manual

Janine Birnbaum[a,b], Einat Lev[a], Marc W. Spiegelman[a,c], and Yan Lavalleé[b]

[a] Lamont-Doherty Earth Observatory, Columbia University, NY, USA
[b] Department of Earth and Environmental Sciences, Ludwig-Maximilians-Universität München, Germany
[a] Applied Physics and Applied Mathematics, Columbia University, NY, USA

January 18, 2024

Abstract

Lavas and magmas are suspensions of partially molten rock with complex rheological properties. The viscosity of silicate melts varies over orders of magnitude in response to changes in temperature and composition, and melts solidify to amorphous glass or crystals at low temperatures. The ability to model the development of rigid solidified margins on lava flows and domes remains an important challenge in answering fundamental science questions and in hazard assessment of volcanic activities. We present a physics-based numerical model, Viscous-Elastic Numerically Unified Solver for Solidifying flows (VENUSS) introduced in Birnbaum (2023). This document provides the information required to run models using VENUSS.

1 Installation and requirements

VENUSS is built on the GetFEM finite element modeling library which requires a standard GNU compiler. See the GetFEM installation information at `https://getfem.org/download.html`. Additionally, a Python environment with the packages alphashape, math, matplotlib, numpy, scipy, shapely, and skfmm. For up-to-date information about the Python environment, refer to the VENUSS Github: `https://github.com/JanineBirnbaum18/VENUS`. For help building a Python environment from a specification file see the conda documentation.

For visualization of results, we recommend the use of Paraview, which is available for free at: `https://www.paraview.org/`. VENUSS contains Python scripts for use with Paraview for some default visualization options.

2 Contents of the software

VENUSS contains:

- `main.py`: the main script that builds and solves the model.
- `curvature.py`: script that defines supporting functions for use in `main.py`.
- `make_inputs.py`: script used to create json input files.
- `T_u_and_d.py`: visualization script for use with Paraview.
- `basic_viz.py`: visualization script for use with Paraview.
- `tests`: folder of example json input files.

3 Input parameters

Inputs to the model are specified in a json file that can be written manually, or by running `make_inputs.py`.

3.1 Mesh geometry

The code generates a structured mesh to divide the domain according to the following parameters:

- `ndim`: Integer number of dimensions, currently only `ndim=2` is supported.
- `symmetry`: String either "planar" or "axial" for 2D planar or cylindrical axisymmetric respectively.
- `L_x`: Length of the domain in the x-direction in meters.
- `L_y`: Length of the domain in the y-direction in meters.
- `nx`: Integer number of elements in the x-direction.
- `ny`: Integer number of elements in the y-direction.
- `etype`: Element type, can either be "cartesian" for rectangular elements or "regular simplices" for triangular elements. Free surfaces are currently supported only on Cartesian elements. For more information on meshing options, see the GetFEM documentation: `https://getfem.org/python/cmdref_Mesh.html`.

3.2 Basis functions

We provide the option to modify the basis functions, but suggest the following options which satisfy the inf-sup condition and result in stability:

- `u_k`: Integer polynomial order for velocity function space, 2 is recommended.
- `p_k`: Integer polynomial order for pressure function space, 1 is recommended.
- `t_k`: Integer polynomial order for temperature function space, 2 is recommended.
- `t_enrich`: Boolean, if `True` enrich the temperature function space with a ramp function centered about the free surface interface, recommended in cases with a strong temperature contrast across the interface and high Péclet number flows.
- `ls_k`: Integer polynomial order for level set function space, 1 is recommended.
- `mat_k`: Integer polynomial order for material properties interpolation, 1 is recommended.

3.3 Initial level sets

We provide generic options for the initial interface conditions. For most users, it is sufficient to toggle the flags `free_surface` and `solve_air` and set the free surface interface with `ls1p` for free-surface flows over a flat domain boundary. If more complex topography or resolution of thermal interaction with the flow base, toggle the flags `topography` and `solve_topography` and set the topography with `ls3p`. In most cases `ls1s`, `ls2p`, `ls2s`, and `ls3s` can be left at the default options.

- `free_surface`: Boolean, if `True` generate a model domain cut by an interface between two fluids.
- `solve_air`: Boolean, if `True` solve the entire model domain, else drop degrees of freedom in fluid 2.
- `ls1p`: String describing a function of `X` and `Y` whose zeroth-contour defines the initial interface between fluids 1 and 2.
- `ls1s`: `None` or string describing a function of `X` and `Y` for which the region less than zero defines the truncation of the initial interface between fluids 1 and 2. See `https://getfem.org/python/cmdref_`

LevelSet.html for more information. item ls2p: String describing a function of X and Y whose zeroth-contour defines the initial interface between fluid 1 and the solidified region. This function is reset within the code to conform to location of glass transition temperature contour.

- ls2s: None or string describing a function of X and Y for which the region less than zero defines the truncation of the initial interface between fluid 1 and the solidified region. This function is reset within the code to conform to location of glass transition temperature contour.
- topography: Boolean, if True generate a model domain cut by an interface with a third fluid.
- solve_topography; Boolean, if True solve for deformation in fluid 3.
- ls3p: String describing a function of X and Y whose zeroth-contour defines the interface between fluids 1 and 2 and fluid 3.
- ls3s: None or string describing a function of X and Y for which the region less than zero defines the truncation of the initial interface between fluids 1 and 2 and fluid 3.

3.4 Pressure

We provide options for the pressure solution including a switch between compressible and nearly incompressible flows and the use of the steady (Stokes) or time-dependent pressure conditions. Pressure must be fixed in at least one point on the domain, which can be selected using p_bound.

- compressible: Boolean, if True treat fluids as compressible, else use approximation for nearly incompressible.
- steady: Boolean, if True solve for instantaneous velocity and pressure solution. If compressible=True, set to False.
- steady_init: Boolean, if True the initial pressure and velocity conditions are set to the instantaneous steady solution, else the pressure field is initially p_amb and velocity is zero everywhere.
- p_amb: Ambient pressure, can be set to zero in most cases unless constitutive relationships are modified to require absolute pressure.
- p_bound: String with options "top_left", "top_right", "bottom_left", or "bottom_right" to specify one corner of the domain for the fixed pressure condition.
- rho1: Density of fluid 1 in kg/m^3.
- rho2: Density of fluid 2 in kg/m^3, only used if free_surface=True.
- rho3: Density of fluid 3 in kg/m^3, only used if topography=True.
- surface_tension: Surface tension of fluid 1 in fluid 2 (N/m).
- beta1: Compressibility of fluid 1 in 1/Pa, only used if compressible=True.
- beta2: Compressibility of fluid 2 in 1/Pa, only used if compressible=True and free_surface=True.
- beta3: Compressibility of fluid 3 in 1/Pa, only used if compressible=True and topography=True.

3.5 Temperature

To solve the evolution of the temperature field use the following set of parameters. If temperature=False, the remaining parameters in this block are not used:

- temperature: Boolean, if True solve for temperature evolution.
- solidification: Boolean, if True solve for solidification in regions below Tg. Requires temperature=True.
- T0: Initial lava temperature (°C).
- T_amb: Initial ambient temperature in fluids 2 or 3 as applicable (°C).

- `T_init`: Number-like, string, or `None`, definition of initial temperature field. If number-like, the initial temperature everywhere (°C). If string, a function of `X` and `Y` that describes the initial temperature field. If `None`, initial temperature field is set to `T0` within fluid 1 and `T_amb` in fluids 2 and 3 as applicable.
- `basal_temp_i`: Initial basal temperature, used only for conduction into fluid 3 when `topography=True` and `solve_topography=False`.
- `kappa1`: Thermal diffusivity in fluid 1 (m^2/s).
- `kappa2`: Thermal diffusivity in fluid 2 (m^2/s), only used if `free_surface=True`.
- `kappa3`: Thermal diffusivity in fluid 3 (m^2/s), only used if `topography=True`.
- `cp1`: Specific heat capacity in fluid 1 (J/KgK).
- `cp2`: Specific heat capacity in fluid 2 (J/KgK), only used if `free_surface=True`.
- `cp3`: Specific heat capacity in fluid 3 (J/KgK), only used if `topography=True`.
- `emissivity`: Emissivity in fluid 1 ($\frac{W/m^2}{W/m^2}$).
- `stefan_boltzmann`: Stefan-Boltzmann constant ($W/m^2\ K^4$).
- `crust_cover`: Fraction of surface area covered by crust for calculation of heat lost to radiation.
- `heat_transfer_coeff`: Heat transfer coefficient for forced convection within fluid 2 to transfer heat with fluid 1 (W/m^2K).

3.6 Viscosity

We provide a default option using the Vogel-Fulcher-Tammann (VFT) equation to describe the temperature dependence of melt velocity in fluid 1. We also provide a parameter option for a relative suspension viscosity to account for the presence of bubbles or crystals in the suspension. If a different description of the fluid viscosity is desired, modify `eta_exp` and use the dependent parameters of the following names:

- `eta_exp`: String description of viscosity (Pas), can be a function of `T` (temperature) and four tuneable parameters `vfta`, `vftb`, `vftc`, and `etar`. Default uses the Vogel–Fulcher–Tammann (VFT) equation.
- `vfta`: First viscosity parameter, default is VFT equation parameter A.
- `vftb`: Second viscosity parameter, default is VFT equation parameter B.
- `vftc`: Third viscosity parameter, default is VFT equation parameter C.
- `etar`: Fourth viscosity parameter, default is suspension relative viscosity.
- `Tg`: Number-like, string, or `None`, glass transition temperature (°C). If string, can be a function of `vfta`, `vftb`, `vftc`, and `etar`. If `None`, when melt viscosity according to VFT equation reaches 10^12 Pas.
- `max_eta`: Maximum viscosity (Pas).
- `eta2`: Viscosity of fluid 2 (Pas), only used if `free_surface=True`.
- `eta3`: Viscosity of fluid 3 (Pas), only used if `topography=True`.

3.7 Elastic properties

The elastic moduli are defined by:

- `E`: Young's modulus (Pa).
- `nu`: Poisson's ratio.

3.8 Body force

A body force can be implemented according to the following:

- `f_x`: Number-like, string, or `None`, body force per unit volume in the x-direction (N/m^3). If number-like, body force is applied everywhere in the domain. For string-like can depend on `X`, `Y`, `P_mat` (pressure), `T_mat` (temperature), and the material properties `rho` (density), `beta` (compressibility), `lambda` (first lamé parameter), `mu` (second lamé parameter), `kappa` (thermal diffusivity), `cp` (specific heat capacity), and `solid` (boolean that is true in solidified regions). If `None`, no body force is applied.
- `f_y`: Number-like, string, or `None`, body force per unit volume in the y-direction (N/m^3).

A common application is to include a body force due to gravity acting in either the vertical direction or at an angle to simulate a flow over an inclined surface. In these cases, the body force is:

$$f_x = \rho g \sin\theta \,, \tag{1a}$$

$$f_y = \rho g \cos\theta \,. \tag{1b}$$

The string inputs for a nearly incompressible material are "rho*9.81*0" and "rho*9.81*1" for `f_x` and `f_y`, respectively where 0 and 1 should be replaced according to the angle of the slope. In the case of linearized compressibility, the density dependence on pressure should be substituted directly into the string to yield "rho*(beta*(1-P_mat))*9.81*0" and "rho*(beta*(1-P_mat))*9.81*1", respectively.

3.9 Velocity boundary conditions

Velocity conditions on the domain boundaries are flexibly defined by the user:

- `left_ux`: Number-like, string, or `None`, Dirichlet condition on the x-component of velocity applied on the left boundary (m/s). If number-like, velocity is constant along the boundary. For string-like can depend on `X` and `Y`. If `None`, no Dirichlet condition is applied.
- `left_uy`: Number-like, string, or `None`, Dirichlet condition on the y-component of velocity applied on the left boundary (m/s).
- `left_dux`: Number-like, string, or `None`, Neumann (stress) condition on the x-component of velocity applied on the left boundary (Pa).
- `left_duy`: Number-like, string, or `None`, Neumann (stress) condition on the y-component of velocity applied on the left boundary (Pa).
- `right_ux`: Number-like, string, or `None`, Dirichlet condition on the x-component of velocity applied on the right boundary (m/s).
- `right_uy`: Number-like, string, or `None`, Dirichlet condition on the y-component of velocity applied on the right boundary (m/s).
- `right_dux`: Number-like, string, or `None`, Neumann (stress) condition on the x-component of velocity applied on the right boundary (Pa).
- `right_duy`: Number-like, string, or `None`, Neumann (stress) condition on the y-component of velocity applied on the right boundary (Pa).
- `top_ux`: Number-like, string, or `None`, Dirichlet condition on the x-component of velocity applied on the top boundary (m/s).
- `top_uy`: Number-like, string, or `None`, Dirichlet condition on the y-component of velocity applied on the top boundary (m/s).
- `top_dux`: Number-like, string, or `None`, Neumann (stress) condition on the x-component of velocity applied on the top boundary (Pa).

- `top_duy`: Number-like, string, or `None`, Neumann (stress) condition on the y-component of velocity applied on the top boundary (Pa).
- `right_ux`: Number-like, string, or `None`, Dirichlet condition on the x-component of velocity applied on the right boundary (m/s).
- `right_uy`: Number-like, string, or `None`, Dirichlet condition on the y-component of velocity applied on the right boundary (m/s).
- `right_dux`: Number-like, string, or `None`, Neumann (stress) condition on the x-component of velocity applied on the right boundary (Pa).
- `right_duy`: Number-like, string, or `None`, Neumann (stress) condition on the y-component of velocity applied on the right boundary (Pa).

While the above arguments can handle piece-wise velocity conditions, for the convenience of the user we provide an easy way to add a velocity inflow condition through boundaries defined within a box.

- `influx`: Boolean, if `True` allows for a prescribed velocity on a segment of the external boundary defined by a rectangular box using the following parameters.
- `fix_ls`: Boolean, if `True` pin the level-set on the boundary defined by
- `fix_ls_bound`. Useful for stabilizing the location of the free surface when velocity at the free surface is non-zero.
- `fix_ls_bound`: 'left', 'right', 'top', or 'bottom' to choose which boundary has a fixed free surface location. Only used if `fix_ls=True`.
- `influx_ux`: Number-like or string describing the velocity in the x-direction over the influx region. If string, can be a function of `X` and `Y`.
- `influx_uy`: Number-like or string describing the velocity in the y-direction over the influx region.
- `influx_left`: Position of the left boundary of the influx region (m).
- `influx_right`: Position of the right boundary of the influx region (m).
- `influx_top`: Position of the top boundary of the influx region (m).
- `influx_bottom`: Position of the bottom boundary of the influx region (m).

These conditions will overwrite the conditions on the domain boundaries above within the specified region.

Finally, if the degrees of freedom are dropped in fluid 3, velocity conditions can be applied by:

- `basal_velocity`: "no_slip" or "no_normal" velocity condition applied at the interface between fluids 1 and 2, and fluid 3 if `topography=True` and `solve_topography=False`.

3.10 Temperature boundary conditions

If thermal evolution is desired, the boundary conditions are implemented using:

- `left_t`: Number-like, string, or `None`. Dirichlet condition on temperature applied on the left boundary (°C).
- `left_dt`: Number-like, string, or `None`. Neumann condition on temperature applied on the left boundary (°C/m).
- `right_t`: Number-like, string, or `None`. Dirichlet condition on temperature applied on the right boundary (°C).
- `right_dt`: Number-like, string, or `None`. Neumann condition on temperature applied on the right boundary (°C/m).

- `top_t`: Number-like, string, or `None`. Dirichlet condition on temperature applied on the top boundary (°C).
- `top_dt`: Number-like, string, or `None`. Neumann condition on temperature applied on the top boundary (°C/m).
- `bottom_t`: Number-like, string, or `None`. Dirichlet condition on temperature applied on the bottom boundary (°C).
- `bottom_dt`: Number-like, string, or `None`. Neumann condition on temperature applied on the bottom boundary (°C/m).
- `influx_t`: Number-like, string, or `None`. Dirichlet condition on temperature applied on the boundary of the influx region (°C). Used only if `influx=True`.
- `influx_dt`: Number-like, string, or `None`. Neumann condition on temperature applied on the boundary of the influx region (°C/m).
- `surface_temp`: Number-like, string, or `None`. Dirichlet condition on temperature applied on the free surface (°C). Used only if `free_surface=True` and `solve_air=False`.
- `surface_flux`: Number-like, string, or `None`. Sink-term on temperature applied on the free surface (°C/m^3). Used only if `free_surface=True`. If number-like, sink term is constant in space and time. If the string argument includes "radiation", the sink term removes heat due to radiation according to:

$$q = \frac{\text{surface area}}{\rho_1 c p_1} \varepsilon\sigma(1 - \text{crust cover})((T_0 + 273)^4 - (T_{\text{amb}} + 273)^4) \,. \tag{2}$$

  If the string argument includes "forced convection", the sink term removes heat due to forced convection according to:

$$q = h_2 * (T_0 - T_{\text{amb}}) \,. \tag{3}$$

  The two strings arguments can be used together.
- `basal_temp`: Number-like or `None`. Dirichlet condition on temperature applied on the interface between fluids 1 and 2, and fluid 3. Used only if `topography=True` and `solve_topography=False`.
- `basal_flux`: Number-like, string, or `None`. Sink-term on temperature applied on the interface between fluids 1 and 2, and fluid 3 (°C/m^3). If number-like, the sink term is constant in space and time. If the string argument includes "conduction", the sink term removes heat due to conduction according to:

$$q = \kappa_3(T_0 - basal_t emp_i) \tag{4}$$

3.11 Time discretization

The model uses a constant time step, except for first time step, which is divided into sub-steps. The time discretization is defined by:

- `tf`: Total simulation time (s).
- `dt`: Time step (s).
- `ndt0`: Integer number of sub-steps for bootstrap into time-step. Recommended 10.

It is important to choose a stable time step such that

$$\Delta t \leq \frac{h}{2u} \,, \tag{5}$$

on each element and at each time in the simulation.

- `n_outer_iter`: Integer, if greater than one, iterate solution with an updated location of free surface for current time step. Recommended 2 when `free_surface=True` and 1 otherwise.

Additionally, it is possible to restart from a terminated simulation by:

- `restart`: Boolean, it `True` attempt to restart simulation from last computed time step stored in hdf5 file and input file.

If the hdf5 file does not exist or is missing a required field (in the case in which the input file has been modified to include a new field such as temperature), then the simulation will start from the beginning. This setting can also be used to extend a previous simulation by modifying only the final time, `tf`, and leaving all other parameters the same.

3.12 Solve options

We include the ability for the user to modify details of the solution, but recommend leaving these options to the default values:

- `stab_p`: Stabilization option for the pressure field specified using a string argument. Can be "None", "SUPG" for streamline upwind Petrov-Galrkin, or "GLS" for Galerkin least-squares. Recommended "SUPG" or "GLS", only used when `steady=False`.
- `stab_psi`: Stabilization option for the level set field specified using a string argument. Can be "None", "SUPG", or "GLS". Recommended "SUPG" or "GLS", only used when `free_surface=True`.
- `stab_d`: Stabilization option for the displacement field specified using a string argument. Can be "None", "SUPG", or "GLS". Recommended "SUPG" or "GLS", only used when `solidification=True`.
- `stab_t`: Stabilization option for the temperature field specified using a string argument. Can be "None", "SUPG", or "GLS". Recommended "GLS", only used when `temperature=True`.
- `epsilon_psi`: Stabilization coefficient for pseudo-viscous relaxation of the free surface where the stabilized interface velocity F_{stab} is:

$$\mathbf{F}_{\text{stabilized}} = \mathbf{F} - \frac{\epsilon}{\eta}(K - K_{\text{mean}})\frac{\boldsymbol{\nabla}\Psi}{|\boldsymbol{\nabla}\Psi|} \,. \tag{6}$$

  Recommended value is 0.001, where a lower value will result in a rougher surface and less over-relaxation.

- `kappa_psi`: Number-like stabilization coefficient for pseudo-diffusion of the level set field for curvature calculation where:

$$\Psi_{\text{smooth}} + \kappa_\Psi \boldsymbol{\nabla}^2 \Psi_{\text{smooth}} = \Psi \,. \tag{7}$$

  Recommended value is 10^{-6} where a lower value will result in a rougher surface.

- `max_residual`: Number-like maximum residual in the classical 1-norm for solution convergence.
- `max_iter`: Integer maximum number of Newton-Raphson iterations. If solutions converge, they typically do so within 4-5 iterations, for poor solutions, typically no improvement is seen with further iterations. A note of caution, the simulation will continue to run even if this convergence criteria is not met. Check the standard error output for details.

3.13 Output options

Model outputs are written to vtk files with the naming convention `[runid]_[fieldname]_[timestep].vtk`. The field names include the initial mesh, `mesh`, pressure, `P`, velocity, `u`, reference density, `rho`, level set for the free surface location, `Ls1`, computed curvature, `curvature` (useful for debugging), extension velocity `Fext`, the smoothed level set field, `psis`, temperature, `T`, and displacement, `d`. For example in a simulation with `runid=test`, the pressure field at time `t=2` with a time step `dt=0.5`, may be `test_P_020.vtk`.

- `outfile`: String location for output files. Recommended "./Results/runid" where each simulation is given a unique identifier. If this is changed, also provide the updated path when running main.py.

- `noutput`: Integer number to write every nth time step to an output file. By default the intermediate solutions to substeps in the first time step are nto written, even when `noutput=1`.
- `ndigits`: `None` or integer number of digits for file names. If `None`, the required number of digits is computed using the time step and final time. Explicitly setting the integer may be helpful in the case of running multiple simulations with the same final time, but different time steps, or if the final simulation time is likely to be extended using `restart=True`. The output times are in simulation time (not integer number of time steps), but the number of leading and trailing zeros do not indicate the location of a decimal place in seconds.
- `vtk`: Boolean, if `True` write output to vtk. Else, only the final time step is saved using `plots=True`.
- `plots`: Boolean, if `True` save images of final output to pdf.
- `plots_mesh`: Boolean, if `True` save image of initial mesh to pdf.

3.14 Analytical solutions

If desired, the solution can be compared to an analytical or different solution through the use of string arguments:

- `true_p`: String describing the pressure solution as a function of "X" and "Y" for position, and "ti" for the current time.
- `true_ux`: String describing the x-component of the velocity solution as a function of "X" and "Y" for position, and "ti" for the current time.
- `true_uy`: String describing the y-component of the velocity solution as a function of "X" and "Y" for position, and "ti" for the current time.
- `true_ls1`: String describing the level-set solution as a function of "X" and "Y" for position, and "ti" for the current time.
- `true_t`: String describing the temperature solution as a function of "X" and "Y" for position, and "ti" for the current time.
- `true_dx`: String describing the x-component of the displacement solution as a function of "X" and "Y" for position, and "ti" for the current time.
- `true_dy`: String describing the y-component of the displacement solution as a function of "X" and "Y" for position, and "ti" for the current time.

For each case, the provided field interpolated onto the mesh will be provided as a vtk for each time step, and the L2 distance between the calculated and provided fields will be written to the hdf5 output.

4 Outputs

Running `make_inputs.py` generates a json input file that is read by `main.py`. If the argument `viz` is specified, it also copies the chosen visualization script, edited to update the path to match the given file location and simulation name.

`main.py` generates several outputs depending on the specified options. An hdf5 file of the name `[outfile].h5` contains 1) for each variable field (velocity, pressure, free surface level set, temperature, and elastic displacement as applicable) the most recent two solutions at each degree of freedom which are progressively overwritten as the code progresses. The data set name for each field is "last_[field]" and "last2_[field]" for the most recent and immediately preceding solutions respectively, where [field] is the lowercase variable abbreviations "u" for velocity, "p" for pressure, "t" for temperature, "d" for elastic displacement, and "ls1" for the free surface level set. 2) In the case where `free_surface=True`, a scalar value of the area enclosed by the free surface in the most recent solution. And 3) the most recent simulation time in seconds. This information is used as an input if `restart=True`. Additionally, this file contains 4) a vector of all timesteps with the

L2-distance between the computed solution and specified analytical solution if `[field]_true` is given, or the L2-norm otherwise. The data set name for each is "err_[field]" with the field name as described above.

For full details of the simulation, the output at every nth time step can be output to vtk using `vtk=True`. The vtk file preserves the geometry of the solution and is possible to visualize using a vtk reader such as Paraview. The files are named according to `[runid]_[fieldname]_[timestep].vtk`. The field names include the initial mesh, `mesh`, pressure, `P`, velocity, `u`, reference density, `rho`, level set for the free surface location, `Ls1`, computed curvature, `curvature` (useful for debugging), extension velocity `Fext`, the smoothed level set field, `psis`, temperature, `T`, and displacement, `d`. For example in a simulation with `runid=test`, the pressure field at time `t=2` with a time step `dt=0.5`, may be `test_P_020.vtk`.

For ease of visualization, we provide scripts `basic_viz.py` and `T_u_and_d.py` for use with Paraview. If specified in `viz`, these scripts will be copied into the output directory with an updated path to the outputs for fast, basic visualization. `basic_viz.py` is a general script that produces render outputs of the pressure and velocity fields, and when applicable the temperature field and either the displacement or viscosity fields. If there is a free surface, each render view will have a contour matching the location of the free surface. Similarly, when the temperature field is provided, there will be a contour at the glass transition temperature. `T_u_and_d.py` provides render views of the temperature and velocity fields with line plots of horizontal cross-sections through the domain. To run, in Paraview, select "load state" and navigate to the desired py file.

Additionally, pdf images of the initial mesh and boundary conditions, and the final output of pressure, velocity, temperature, and free surface location can be created using `plots=True`.

5 Example cases

We provide input json files for the example cases provided in Birnbaum (2023):

- `cavity_30x.json`: Inputs for the lid-driven cavity problem for pressure and velocity fields.
- `relax_29x.json`: Inputs for the time-dependent relaxation of a sinusoidal free surface.
- `solidification_29x.json`: Inputs for temperature diffusion from one side of the domain with a temperature-dependent viscosity and an elastic solid.
- `spreading_drop_planar.json`: Inputs for isoviscous spreading drop in 2D planar geometry.
- `spread_drop_axial.json`: Inputs for isovisous spreading drop in axisymmetric geometry.

References

Birnbaum, J. (2023). *Multi-phase controls on lava dynamics determined through analog experiments, observations, and numerical modeling.* PhD thesis, Columbia University in the City of New York.

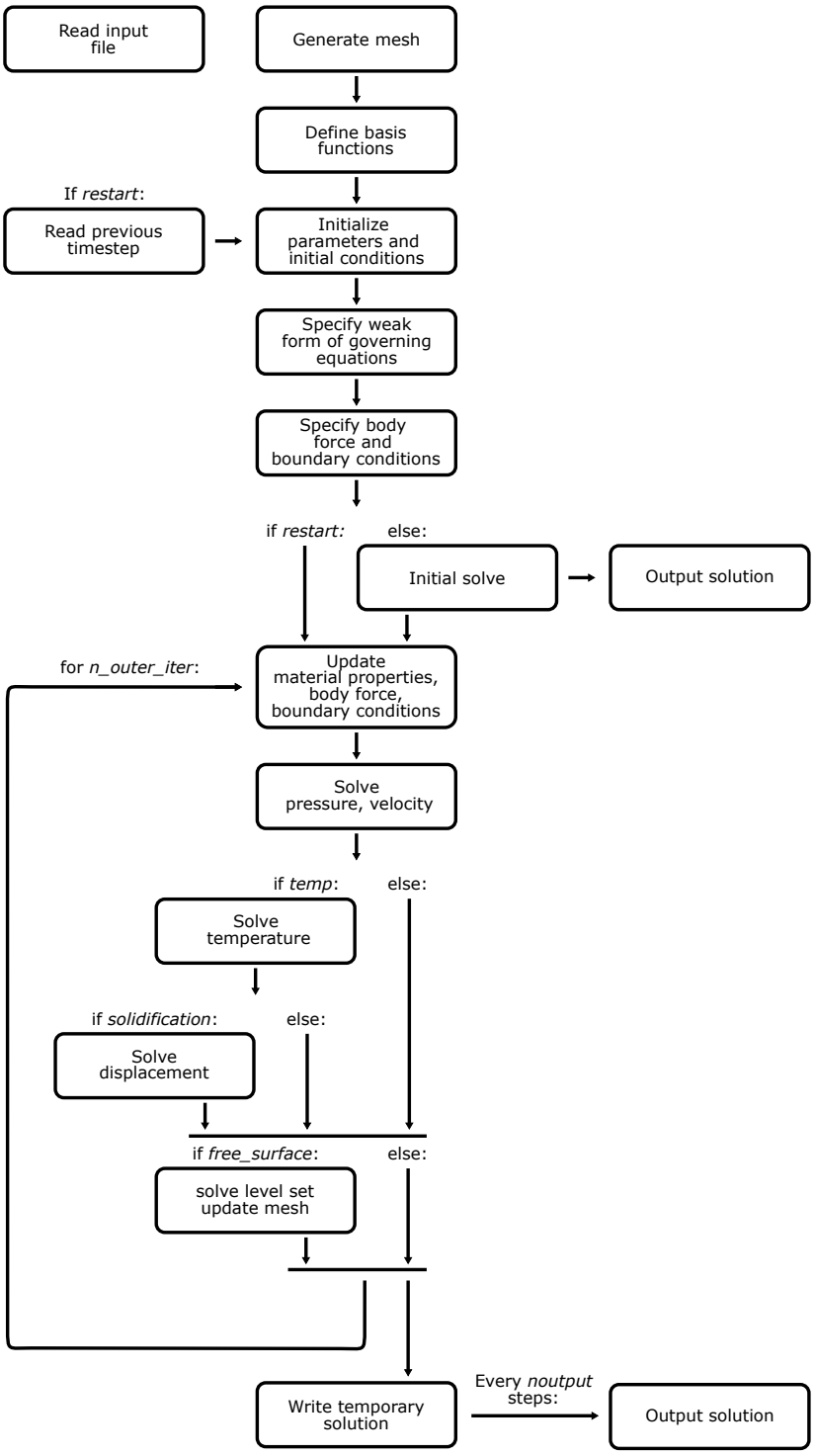

Figure 1: Overview of code structure.