Beo

Elco Luijendijk, elco.luijendijk@geo.uni-goettingen.de

Table of contents

Introduction

Beo is a model of heat flow in hot springs and hydrothermal systems. The model code uses the generic finite element code escript (https://launchpad.net/escript-finley) to solve the advective and conductive heat flow equations in a 2D cross-section of the subsurface. The modeled temperatures can be compared to temperature daat from thermal springs or subsurface temperatures records from nearby boreholes. The resulting temperature history can also be used to calculate two low-temperature thermochronometers, apatite fission track data and apatite (U-Th)/He data or vitrinite reflectance data. The modeled values of these thermochronometers can then be compared to measured values. Beo also support automated model runs to explore which parameter values like fluid fluxes, fault geometry and age of the hydrothermal system best match the thermochronometer data, as well as present-day spring temperature data or temperature records in nearby boreholes.

Installing & running

- Install Escript
  - get the code here: https://launchpad.net/escript-finley
  - an installation guide can be found here: http://esys.geocomp.uq.edu.au/docs

- Unzip the beo source code

- Navigate to the directory where you have installed escript. Go to the subdirectory bin (`python-escript/bin/`) and run Beo by executing the following command from the command line:

```
./run-escript beo_dir/beo.py
```

Where `beo_dir` is the directory where you have saved Beo.

Alternatively use the command

```
./run-escript -e
```

This will show you three lines that define environment variables that your system needs to be able to find the location of escript. Add these lines to your .bashrc (Ubuntu linux) or profile file in your home directory. After adding these lines and logging out and in again, you can start beo by going to the directory where the beo code is located (so not to the escript/bin directory) and start beo.py like any regular python code:

```
python beo.py model_parameters/model_parameters.py
```

where `model_parameters/model_parameters.py` is a file containing all model parameters. An example file called `model_parameters.py` is located in the directory `model_parameters`.

Required modules

- numpy: http://www.numpy.org/
- For creating figures (optional):
  - scipy: http://scipy.org/scipylib/index.html
  - matplotlib: http://matplotlib.org/downloads.html

These modules are available as standalone packages. For mac and windows an easy way to get a working version of python and these modules is to install a full Python environemnt like Anaconda (https://www.anaconda.com/), Enthought Canopy (https://www.enthought.com/products/canopy) or pythonxy (https://code.google.com/p/pythonxy/wiki/Welcome).

Beo was tested on Ubuntu 14.04 and 16.04

Model input & output

Input:

All model input parameters are contained in the file `model_parameters.py` located in the directory model_parameters. The class `ModelParameters` contains all parameters needed for a single model run. See the comments in this part of the file for a brief explanation of what each parameter does.

Multiple model runs

Optionally you can start automated runs to test a range of parameter combinations. This is useful for automated sensitivity or uncertainty analysis.

The model input file contains a class calle `ParameterRanges`. Any parameter value that is included in this class will be used as input for a single model run. All results will be stored an written to a comma separated (.csv) file names `model_output/model_params_and_results_x_runs.csv`.

You can include any model parameter in the automated runs. Simply copy a parameter from the `ModelParameters` class into the `ParameterRanges` class and add _s to the parameter name. For instance to test multiple values of the thermal gradient, add `thermal_gradient_s = [0.03, 0.04, 0.05]` to test the effect of geothermal gradients of 0.03, 0.04 and 0.05 C/m, respectively.

There are two options for running multiple model runs. The default is a sensitivity run. In each model run a single parameter will be changed, while all other parameters are kept constant at the default value specified in the `ModelParameters` class. Alternatively you can test all parameter combinations by changing `parameter_combinations = False` to `parameter_combinations = True`. Note that this will generate a lot of model runs, testing ten parameter values for two parameters each will generate 10*10 = 100 model runs, for three parameters this increase to a 1000 model runs, etc…

Output

- After each model run, the modeled temperature field and (U-Th)/He data are stored in the directory `model_output` as a .pck file, which can be read using Python's pickle module.
- In addition, Beo saves a comma separated file containing the model parameters and a summary of the results for each model run and each timestep in the same directory.

Making figures

Currently there are two Python scripts that will generate figures of the model output:

- The script `make_figures.py` will make a single figure of the final temperature field for output files (with extension .pck) found in the directory `model_output`. After running this script you will be prompted to select the output file that you would like a figure of. The file `model_parameters/figure_params.py` contains a number of parameters that control the figure, such as which timeslices to show, the min. and max. coordinates of the area to show, etc.. THe resulting figure is saved as a .png file in the same directory as the model output file.
- `make_figure_2models.py` will make a figure with two panels containing the modeled temperatures and (U-Th)/He data for two model runs. THe model runs are specified in the script itself, for example like this: `files = ['model_output/T_field_model_run_14_(-3000.0, 0.03).pck', 'model_output/T_field_model_run_32_(-6000.0, 0.03).pck']`. There are a number of parameters in line 62 to 103 in this script itself that you can adjust how the resulting figure looks. The figure is saved to the `model_output` directory.

Model Background

Heat transport

Beo uses the generic finite element code escript to solve the heat flow equation and model conductive and advective heat transport in and around hydrothermal systems. Escript is designed to solve a single partial differential equation (PDE):

$$-\nabla(A\nabla u + Bu) + C\nabla u + Du = -\nabla X + Y$$

Where u is the variable to be solved and A< B, C, D, X and Y are constants.

The heat flow equation used by Beo to model conductive and advective heat flow is given by:

$$\rho_b c_b \frac{\partial T}{\partial t} = \nabla \kappa \nabla T - \rho_f c_{p,f} \vec{q} \nabla T$$

In which T is temperature (K), t is time (sec), c is heat capacity, ρ is bulk density (kgm^{-3}), κ is thermal conductivity ($Wm^{-1}s^{-1}$), ϕ is porosity, ρ is density (kg m^{-3}), c the heat capacity ($Jkg^{-1}K^{-1}$) and $\vec{v}$ is the flow velocity vector (m s^{-1}). Subscripts $_b$, $_f$ and $_s$ denote properties of the bulk material, the fluid and the solid phase, respectively.

Implicit form

Beo solves the implicit form of the heat flow equation by discretization of the derivative of T over time:

$$\frac{\partial T}{\partial t} = \frac{T^{t+1} - T^t}{\delta t}$$

Which when inserted into heat transport equation yields:

$$\rho_b c_b \frac{T^{t+1} - T^t}{\partial t} = \nabla \kappa \nabla T^{t+1} - \rho_f c_{p,f} \vec{q} \nabla T^{t+1}$$

which can be rearranged to:

$$-\nabla dt\, \kappa \nabla T^{t+1} + dt\, \phi \rho_f c_{p,f} \vec{q} \nabla T^{t+1} + \rho_b c_b T^{t+1} = \rho_b c_b T^t$$

Casting the implicit heat transport equation into the escript PDE yields:

$$-\nabla \underbrace{dt\, \kappa}_{A} \nabla T^{t+1} + (\underbrace{dt \rho_f c_f \vec{q}}_{C}) \nabla T^{t+1} + \underbrace{\rho_b c_b}_{D} T^{t+1} = \underbrace{\rho_b c_b T^t}_{Y}$$

with the following values for the escript constants:

$u = T^{t+1}$

$A = dt\, \kappa$

$C = dt \rho_f c_f \vec{q}$

$D = \rho_b c_b$

$Y = \rho_b c_b T^t$

Steady-state

The initial (undisturbed) background temperature is calculated by solving the steady-state heat flow equation, i.e. the heat flow equation with the term $\frac{\partial T}{\partial t}$ set to zero. For the initial temperatures the heat advection term is set to zero. In addition, there is also an option to subsequently model steady-state temperaratures with advective heat flow included. The steady-state heat flow equation solved in Beo is:

$$0 = \nabla \kappa \nabla T - \rho_f c_f \vec{q} \nabla T$$

$$-\nabla \kappa \nabla T + \rho_f c_f \vec{q} \nabla T = 0$$

THe equation can be cast into the partial differential equation solvec by escript:

$$-\nabla (A \nabla u B u) + C \nabla u + D u = -\nabla X + Y$$

Which can be cast into the same form as the escript PDE:

$$-\nabla \underbrace{\kappa}_{A} \nabla T + \underbrace{(\rho_f c_f \vec{q})}_{C} \nabla T = 0$$

This yields the following parameters that Beo passes on to escript:

$u = T$

$A = \kappa$

$C = \rho_f c_f \vec{q}$

Explanation of model parameters

The parameters that control Beo are stored in a python file called `model_parameters.py` in the class ModelParams. The following section describes each of the model parameters.

Data types

The parameters can be several python data types:

- boolean: this is a variable that can either be True or False. Used to control model options, such as `create_mesh_fig = True`
- numbers: numbers can be either float (2.3) or integers (2).
- strings: text that is bracketed by ” or “”, like this: “this is a string”.
- lists: a list of numbers, strings (text) or a mixture of these. Example: `phi0 = [0.45, 0.65, 0.45]`
- numpy arrays: arrays containing numbers. Similar to lists, but can only contain numbers. Arrays can either look like this: `example_array = np.array([1, 2, 3])`, which means an array of three numbers 1, 2 and 3. Other options are creating a range of numbers: `Ls = np.arange(2500, 52500, 2500)`, which creates an array called `Ls` that contains a range of numbers from 2500 to 52500, with steps of 2500. See the numpy documentation (https://docs.scipy.org/doc/numpy/user/basics.creation.html#arrays-creation and https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html#routines-array-creation) for more options to create arrays.

General parameters

- `output_folder`: string. directory to store model output
- `output_fn_adj`: string, name to add to output files generated by Beo = 'beowawe'
- `steady_state`: boolean, option to choose a steady state (no changes over time) or transient model. note that regardless of this setting, the initial condition of transient model is the steady-state solution without any advection
- `vapour_correction`: boolean, use a function to keep the temperature below the maximum temperature in the vapour pressure curve. Note that this simply limits temperature and calculate where phase changes to vapour would happen, there is no vapour transport and the effect of phase changes on the energy/enthalpy budget are also not taken into account.

Model dimensions and mesh parameters

- `width`: number, width of the model domain on either side of the first fault (m)
- `total_depth`: number, depth of the model domain (m)
- `air_height`: number, height of the layer of air that is included in the model domain above the land surface (m). This layer is included for a more accurate simulation of land surface temperature as opposed to the traditional specified temperature or specified heat fluxes at the land boundary.
- `z_fine`: number, depth to fine discretization near surface (m)
- `cellsize`: number, default cellsize (m)
- `cellsize_air`: number, cellsize in the air layer (m)
- `cellsize_surface`: number, cellsize at surface layers (m)
- `cellsize_fine`: number, fine cellsize near surface (up to depth = `z_fine`) (m)
- `cellsize_fault`: number, cellsize in fault zone (m)
- `cellsize_base`: number, cellsize at the lower left and right corners of the model domain (m)
- `use_mesh_with_buffer`: boolean, Use a buffer zone around fault with the same cell size as the fault. this is to reduce model instability.
- `fault_buffer_zone`: number, size of the buffer zone around the fault (m)

Exhumation parameters

- `add_exhumation`: boolean, add exhumation or not. If set to True, the land surface is lowered over time to model erosion or exhumation.
- `exhumation_rate`: number, exhumation rate in m/yr

- `exhumation_steps`: integer number, number of grid layers between initial and final surface level, the more layers, the more smooth and accurate the exhumation history, but this also slows the model down somewhat
- `min_layer_thickness`: number, minimum layer thickness, if the exhumation steps result in surfaces that are less than the min thickness apart, the number of steps is reduced default value is 1.0 m, reduce this value if gmsh returns an error while creating the mesh

Temperature boundary conditions

- `air_temperature`: number, specified temperature at the top of the air layer (degrees C)
- `thermal_gradient`: number, goethermal gradient (degrees C / m). new version: calculate bottom T using a fixed geothermal gradient./r
- `basal_heat_flux`: number, specified heat flux at the bottom of the model domain (W m-2). Set to None if T bnd is used

Model layers

- `layer_bottom`: list of numbers, elevation of layers either side of the fault. structured like this: [[depth layer 1 left, depth layer 1 right], [depth layer 2 left, depth layer 2 right], [depth layer 3 left, depth layer 3 right], etc…]. Layers are counted from bottom to top. leave depth of first layer at arbitrarily high value to make sure the entire model domain is covered. note that currently only 1 fault is taken into account…. example: `layer_bottom = [[-20000.0, -20250.0], [-810.0, -1060.0],[-530.0, -780.0], [-440.0, -690.0], [-210.0, -460.0]]`

Thermal parameters

Note that only thermal conductivity and porosity are varied between layers, the other parameters are constant. note K_air is not used if variable_K_air is set to True

- `porosities`: list of numbers, porosity for each layer
- `K_solids`: list of numbers, thermal conductivity of the solid matrix for each layer (W m-1 K-1)
- `K_air`: number, thermal conductivity of air (W m-1 K-1)
- `K_water`: number, thermal conductivity of pore water (W m-1 K-1)
- `rho_air` : number, density of air (kg m-3)
- `rho_f`: number, density of pore water (kg m-3)
- `rho_s` : number, density of the solid matrix (kg m-3)
- `c_air`: number, heat capacity of air (J kg K-1)

- `c_f`: number, heat capacity of pore water (J kg K-1)
- `c_s`: number, heat capacity of the solid matrix (J kg K-1)
- `variable_K_air`: boolean, option to use a fixed thermal conductivity for air or to use a variable thermal conductivity that depends on the surface temperature and that is used to approximate the actual sensible and latent heat flux at the land surface.
- `ra`: number, aerodynamic resistance, see Liu et al (2007), Hydrology and Earth System Sciences 11 (2)
- `dz`: number, measurement height for aerodynamic resistance (m)

Timesteps and output

- `N_outputs`: list of numbers, number of steps at which output files are generated. this is not used when exhumation > 0, in this case output is generated once each new surface level is reached. the number of surfaces is controlled by the `exhumation_steps` parameter
- `dt`: number, size of a single timestep (sec)
- `durations`: list of numbers, duration of each timestep slice (sec). Note that you can define one or more different timeslices, for instance to first model 500 years of heating and then 1000 years of recovery by setting `durations = [500.0, 1000.0]`. You can define different fluid fluxes in faults or horizontal aquifers in the parameters `fault_fluxes` or `aquifer_fluxes` below.
- `target_zs`: list of numbers, target depth slices for calculating temperature and U-Th/He. In case of exhumation, this values is overridden and set equal to each exhumation step layer. In this way one can track the AHe response to each layer that comes to the surface in a longer time period

Fault parameters

Beo simulates advective heat transport in faults. Note that beo does not model actual fluid flow, ie it does not solve Darcy's law or include permeability or such. Fluid flow is simply a fixed advective flux that is prescribed by the user.

- `fault_xs`: list of numbers, x location of fault (m):
- `fault_widths` : list of numbers, width of the fault zone (m)
- `fault_angles` : list of numbers, angle of the fault zone (degrees).
- `fault_bottoms`: list of numbers, elevation of bottom of fault (m)
- `fault_segments`: list of numbers, different segments of the fault, list of the top bnd of each segments starting from the bottom, nested list: [[segment_top1_fault1, segment_top2_fault1], [segment_top1_fault2, segment_top2_fault2], etc…]. This is used to be able to assign a different fluid flux to different parts of a single fault (for instance below and above a connected aquifer)

- `fault_fluxes`, list of numbers, fluid advection rates in faults (m2/sec). This is a nested list that is structured as follows: [[[fault1_segment1_t1, fault1_segment2_t1], [fault2_segment1_t1, fault2_segment2_t1], etc…]. Note that the flux in specified in m2/sec, i.e., the integrated flux over the entire width of the fault zone. t1 and t2 refer to different time periods defined in the parameter `durations`.

Aquifer parameters

In Beo aquifers are used for modeling horizontal advective flow, for instance horizontal flow in a permeable layer connected to a fault. Use `aquifer_top = [None]` to not use this. For multiple aquifers start at the lowest aquifer

- `aquifer_tops`: list of numbers, elevation of the top of one or more aquifers (m). Use `aquifer_top = [None]` to not use aquifers in your model run.
- `aquifer_bottoms`: list of numbers, elevation of the bottom of one or more aquifers (m).
- `aquifer_fluxes`: list of numbers, nested list of the fluid fluxes each aquifer (m s-1). The list is structured like this: [[flux_aquifer1_t1, flux_aquifer1_t1], [flux_aquifer2_t2, flux_aquifer2_t2]]. t1 and t2 refer to different time periods defined in the parameter `durations`.
- `aquifer_left_bnds`: list of numbers, left hand side of aquifer. right hand bnd is assumed to be the fault zone

Borehole temperature data

- `analyse_borehole_temp`: boolean, option to calculate temperature data for one or several boreholes. note that there seems to be a bug in the output timesteps for the temperature calculation, avoid using this for now…
- `temperature_file`: string, name of the file that contains temperature data.
- `borehole_names`: list of strings. Names of the boreholes to include in your analysis.
- `report_borehole_xcoords`, boolean. Report the coordinates of the top of the borehole over time. For debugging purposes.
- `borehole_xs`: list of numbers, x-coordinates of boreholes with temperature data (m), note location is now relative to the location of the first fault ie, -100 m means 100 m to the left of the fault. the model code automatically calculates the correct position to take into account the changing position of the fault surface over time due to exhumation

Apatite U-Th/He params

- `calculate_he_ages`: boolean, option to model AHe ages or not. Ages are modeled for the depths specified in the parameter `target_zs` or at the land surface if exhumation is turned on.
- `model_AHe_samples`: boolean, model the AHe ages of specified samples, each with a different apatite radius, U238 and Th232 concentration.
- `AHe_data_file`: string. Filename of the file containing the AHe data.
- `profile_number`: integer number, number of the profile to include in modeling AHe ages. This can be used to not model all the AHe samples in a single datafile, but only those for which the entry in the profile column in the datafile equals `profile_number`.
- `save_AHe_ages`: boolean, save the AHe ages at the surface to a separate file
- `AHe_method`: string, method to calculate helium diffusivity, use Wolf1996, Farley2000 or RDAAM
- `AHe_timestep_reduction`, integer number. Use temperature at each x timestep for the calculation of AHe ages. This should be an integer. Ideally this should be 1, but higher numbers significantly speed up the model code
- `t0`: number, crystallization age (sec), the age of the apatite crystal (which may be older than the age of the hydrothermal system)
- `T0`: number, temperature of apatites after crystallization and before hydrothermal heating (degrees C)
- `T_surface` : number, temperature at the surface (degrees C)
- `radius`: number, radius of the default apatite (m)
- `U238`: number, concentration uranium-238 for the default apatite (ppm?)
- `Th232`: number, concentration thorium-232 for the default apatite (ppm?)
- `alpha_ejection`: boolean, model alpha ejection or not.
- `stopping_distance`: number, alpha ejection stopping distance (um), see Ketcham (2011) for estimates
- `partial_reset_limit`: number, relative limit to consider a sample partial reset or not, ie if 0.95 a sample will be considered partially reset if the modeled uncorrected AHe age is less than 0.95 x the maximum age in the system.
- `reset_limit`: number, absolute age limit (My) below which samples are considered reset (ie. AHe age ~0 My)