VICTOR

Release 0.1

Lev & Krasnoff

Mar 08, 2023

CONTENTS

VICTOR is a JupyterHub based platform for volcano science modeling and data analysis. All of our infrastructure and modeling code is open-source and/or publicly available. We hope for VICTOR to function as a continously evolving service, fostering novel mathematical models and encouraging further data sharing among the volcanology community and beyond.

For more detailed information on the project overall, check out the *Usage* section.

For details on running each individual model, please go to the *Models* section.

Note: This project is under active development and is subject to changes.

CHAPTER

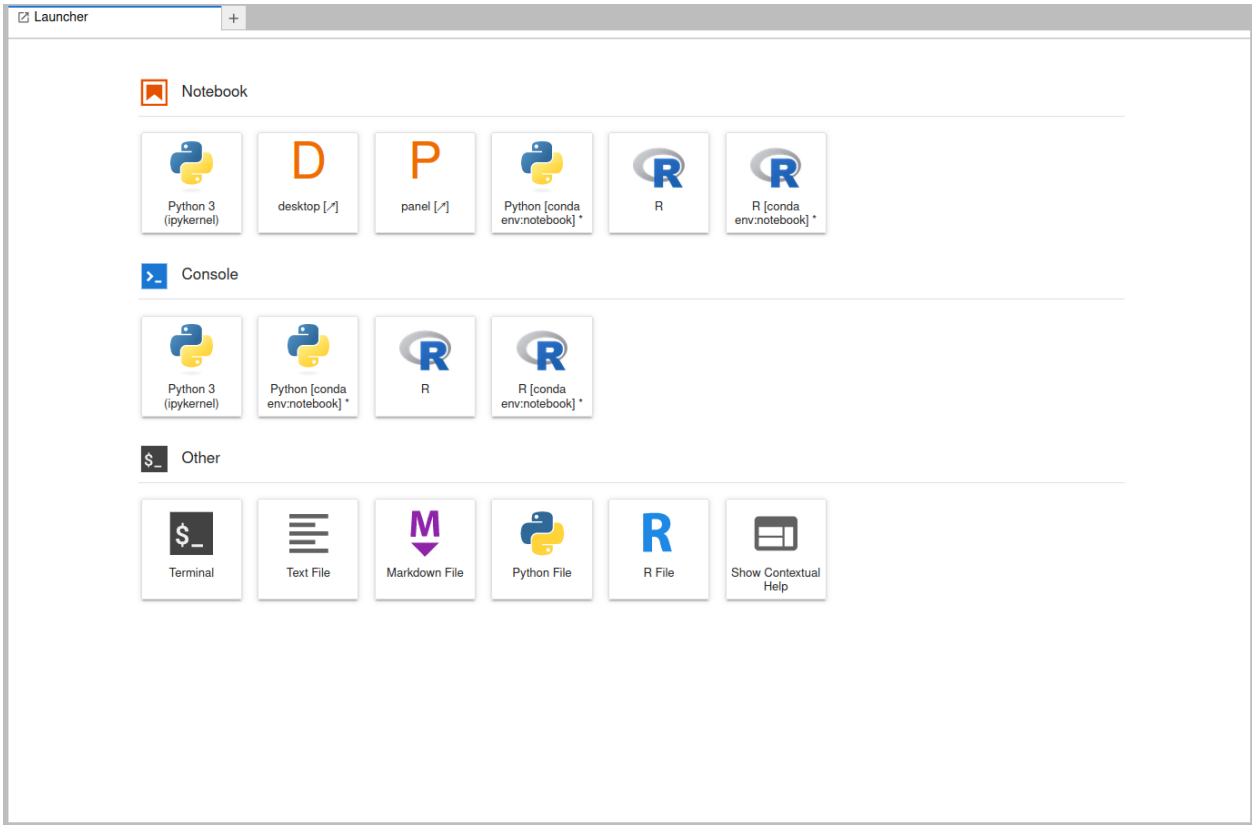
ONE

CONTENTS

1.1 Usage

1.1.1 Registration

To gain access to VICTOR, you must first register. Currently, we are authenticating users through Github, and thus all users are required to possess an account. To register, please visit the VICTOR website and click on the *Sign Up* button on the right of the menu. This will direct you to a form where you can provide the neccesary information. Once you are able to access the hub, you will see a screen very similar to this:



1.1.2 The Launcher

The first screen you will be met with is the launcher, providing an variety of environments to utilize. The first row of buttons is comprised of highly interactive Jupyter notebooks, as well as the desktop application. The VICTOR team advises using only the `[conda env: notebook]` tabs, as this is the environment that includes all the neccesary scientific packages for utilizing and visualizing models.

The second row of buttons opens consoles, a simplified interactive environment where you can enter lines of code and run them, though not able to embed images or renderings such as matplotlib.

The final row of buttons provides quick access to static files in various languages, as well as for a simple text file. Additionally, access to a bash terminal is accessible here, allowing for advanced users to maximize productivity in this environment.

1.1.3 Run Models

To run models, there are a few simple steps. First, navigate to the `shared` folder. You can either run the the `*****_setup.sh` files directly, or copy one, either through manual selection (right click + copy) with the mouse, or through the terminal with `cp shared/*****_setup.sh .` when you are in the home directory.

Note: All files in the `shared` folder are read/execute only. If you would like to contribute models, data, or ideas for improvement, please contact victor@ldeo.columbia.edu.

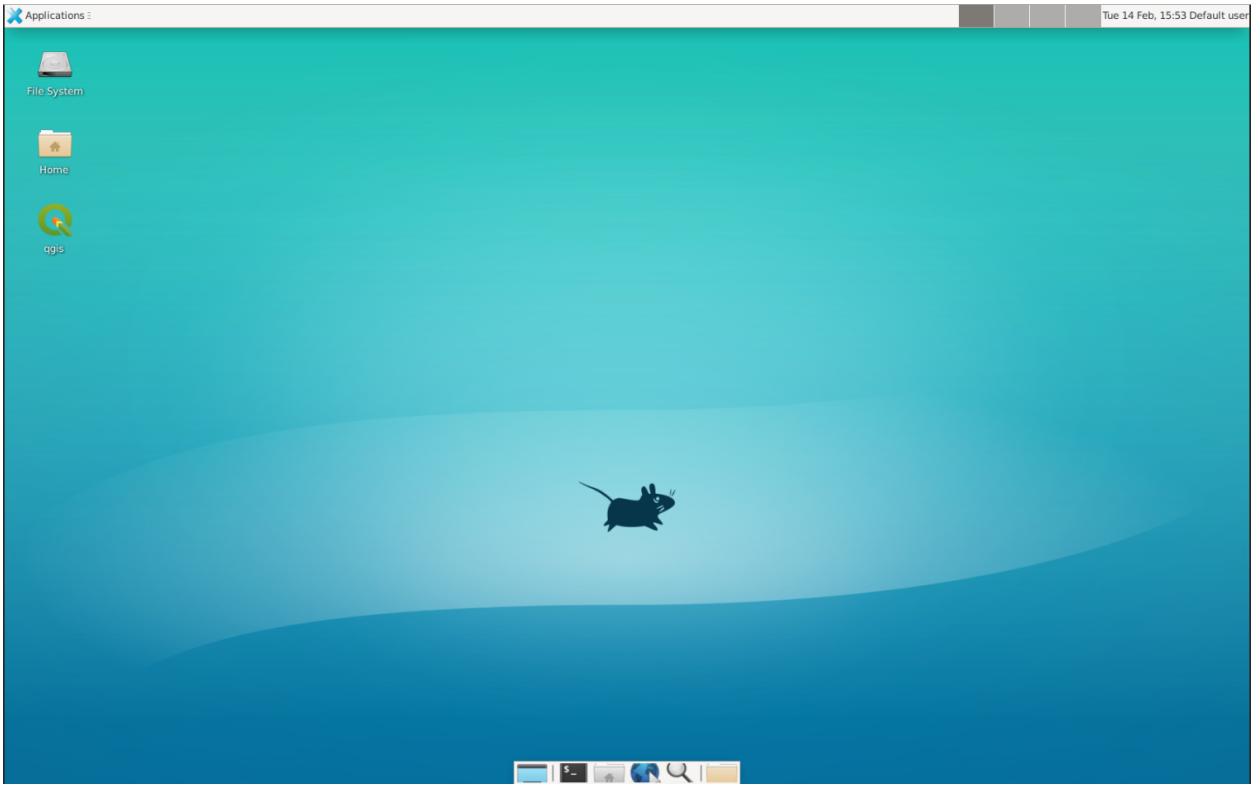
Once this runs, you will have all necessary files contained in a new folder in your home directory. Most folders will simply contain the executable and the example notebook. All Juptyter notebook workflows will generate most of the supplemental files necessary for the model to run.

However, DEMs are not automatically included. Users then have 4 primary options: 1. Navigate to the DEMs folder in `shared` and copy the relevant file to your home directory, if the file needs to be altered. 2. Read the DEM from `$HOME/shared/DEMs` directly in your workflow, if no changes are required. 3. Import your own DEM from a local machine, dragging and dropping into the file tree. 4. Utilize S3 buckets or an SQL connection (using boto3 or mysql python packages) to load files remotely.

At this point, users can go through the first few cells immediately succeeding the import statement, inputting parameters as needed. Thorough descriptions of each parameter are included. Once finished, the user can simply press the fast forward symbol to run all cells, or `shift + enter/return` to run each cell individually.

1.1.4 Accessing The Virtual Desktop

Upon clicking the desktop button on the launcher, another tab will open, displaying a screen as seen below. From here, you can access a fully featured QGIS installation, with many other programs scheduled to be added as well. The file system is connected to your personal files, and a web browser is also available for your convenience.



1.1.5 Citations and References

Below are citations and related works used to create this project. To add additional citations or for clarification, contact victor@ldeo.columbia.edu

Conflow

See *Conflow/Confort* here.

Hazmap

See *Hazmap* here.

HYSPLIT

See *Hysplit* here.

IMEX-Lava

See *IMEX* here.

Molasses

See *MOLASSES* here.

MrLavaLoba

See *MrLavaLoba* here.

pyFLOWGO

See *pyFLOWGO* here.

Tephra2

See *Tephra2* here.

Titan2D

See *Titan2D* here.

1.1.6 Contributing Models and Hub Additions

If you believe your model would be a good fit for our platform, please email victor@ldeo.columbia.edu with a link to the code on a version-control platform as well as a brief explanation. For additions to the hub itself, please refer to our Github repository. Create an issue for general advice, or create a pull request for specific changes/updates.

1.2 Models

1.2.1 Models Overview

VICTOR supports a wide array of both numerical and probabalistic models on its systems. Execution of the models can be done directly from the command line through the command line, though we recommend running everything through Jupyter Notebooks.

Each notebook is built uniquely for the model, though as many aspects as possible are generalized. Most often, this results in significantly diverging methods of parameter input, but a very similar output and visualization. For example, current models are built in Fortran, C, C++, and Python, and thus require different libraries and input formats. Visualizations and data parsing, however, are all done primarily using the *rasterio*, *matplotlib*, and *xarray* libraries, though others are available for your convenience. Additionally, common actions for the hub, such as file I/O and cross-library visualization are referenced in the *VICTOR* module.

Note: Due to the varied languages and file requirements, additional files neccesary for the models may be included in the generated folders. Intended output files will be clearly specified by both the workflow and documentation.

1.2.2 Utilizing Workflows

Below are brief descriptions of each model currently available, and their associated example workflows. The VICTOR team recommends use of the workflows at least through the output standardization process for maximum user experience. All outputs will be in either ascii shapefile, csv, or netCDF formats depending on the model, for compatibility's sake.

VICTOR offers three sets of options for visualization. The first is through any of the dozen libraries included in the built-in conda environment. Second is the custom library `victor.py` for sharp and accurate plotting through a handful of reuable functions. For the most versastile work, a virtual desktop with QGIS resides for a fully featured experience.

1.2.3 Conflow/Confort

Confort is an updated version of Conflow, an open-source numerical model for flow in eruptive conduits during steady-state pyroclastic eruptions. Confort's improvements include more accurate rheological parameters and equations, evaluations of crystal-bearing rheology, additions of crystal and vesicle size distribution, and integration of degassing in both equilibrium and disequilibrium conditions.

The example workflow sparse and should be fairly easy to follow. After importing the necessary libraries, The first cell contains all input parameters, ranging from the pressure at the beginning and ends of the model to the weight percent of various chemical compounds and particles. The following two cells can be run without input. Following these, please thoroughly read the markdown cell, and select 7 outputs from the list of 27. Input those numbers into the next cell in a list. Every subsequent cell can be run without user interaction. There is an intermediate output specified by the `name` variable, but the most gracefully formatted file will always be `Conflow.csv`, output to the current directory.

References:

Silvia Campagnola; Claudia Romano; Larry G Mastin; Alessandro Vona (2016), "Confort 15 (Conflow improvement)," https://theghub.org/resources/3743.

1.2.4 Disgas

DISGAS (passive DISpersion of GASes and particles) is a Eulerian model for passive dispersion of diluted gas and fine dust particles. Turbulent diffusion is based on the K-theory, and the wind field can be evaluated assuming either a wind profile based on the similarity theory or using a terrain-following mass consistent wind model. The model can be used to forecast concentration of gas (or dust) over complex terrains.

The DISGAS workflow begins with parameters relating to the date of the simulation, as well as its duration, and options concerning if it was continuing from an existing run. Next, one must enter data about the grid and general topography. The grid information is required, though the exact elevation can either be sourced from a file or simplified into a slope. The third cell asks the user how they want to treat the model. When treated as a gas with no settling velocity, extra parameters are not needed. However, when treated as a set of particles, the physical properties and mathematical methods to calculate the settling velocity. Then, the user must specify the vertical and horizontal wind turbulence models as well as the soil roughness model and diffusion coefficients. The final input cell requests the user to input file paths for supplemental input data in addition to output intervals and the option to output directional velocities and concentration.

The next two cells format the input and run the model. Depending on the number of wind data points provided, multiple layers will be output. The user must then specify a layer, and can then run the following cell to output a set of plots over the timespan.

References

A. Costa, G. Macedonio, Chiodini G., 2005. Numerical model of gas dispersion emitted from volcanic sources. Annals of Geophysics, Vol. 48: 805-815.

Granieri D., Costa A., Macedonio G., Chiodini G., Bisson M. (2013) Carbon dioxide in the city of Naples: contribution and effects of the volcanic source, J. Volcanol. Geotherm. Res., Vol. 260: 52-61, doi: 10.1016/j.jvolgeores.2013.05.003

Costa A., Macedonio G. (2016) DISGAS: A model for passive DISpersion of GAS, Rapporti tecnici INGV, N. 332, Istituto Nazionale Di Geofisica e Vulcanologia, Italy

1.2.5 Hazmap

Hazmap is a computer program for simulating sedimentation of volcanic particles from discrete point sources and which outputs the corresponding ground deposit in its aptly named deposit mode. Additionally, Hazmap is able to evaluate the probability of overcoming a given loading threshold in the ground deposit by using a set of different wind profiles recorded in different days in its probability mode.

The example Hazmap workflow begins with a variety of flags and specifications for the Hazmap grid and output structure. Comments should give some context for the inputs, though a manual is hyperlinked for the user's convenience. The next cell is the last that requires user input. Take note that all four of `diameters, densities, shapes, weight_percent` should be equal lengths, and equal to `num_particle_types`. The weights should also add up to 100, as they are percentages.

Subsequent cells can be run without additional interactions, resulting in a netCDF file named `hazmap.nc` and a contour graph. We are currently working on adding a basemap background to this graph.

References:

Macedonio et al., 2005 G. Macedonio, A. Costa and A. Longo, A computer model for volcanic ash fallout and assessment of subsequent hazard, Comput. Geosci. 31 (7) (2005), pp. 837–845.

Antonio Costa (2013), "Hazmap," https://theghub.org/resources/hazmap.

1.2.6 Hysplit

The Hybrid Single-Particle Lagrangian Integrated Trajectory model (HYSPLIT)[1] is a computer model created by NOAA that is used to compute air parcel trajectories to determine how far and in what direction a parcel of air, and subsequently air pollutants, will travel.

VICTOR contains the entirety of Hysplit, though our workflow focuses on modeling ash deposition and concentration. First, the user is asked to specify the particle distribution configuration, vertical and horizontal turbulence models, as well as the output file name. Equally as important in the first cell is the number of particles per cycle, as well as the maximum particles released.

The second input cell requires the user to enter the start date, latitude/longitude of the volcano and the ash column, and the maximum runtime of the model. It also requires an input data grid. For each particle, an identifier, along with emission rate, hours of emission, and start time are necessary.

The final input cell has the user concentration grid information, along with sampling interval timing, and then a swath of particle information including, but not limited to, the density, diameter, deposition velocity and decay rate if it is an unstable molecule.

Upon completing the inputs, the user will run the model and be given a choice of timesteps to pick from. After this choice, every other cell can be run. Three images will be the result. First, the workflow uses a built-in visualizer from Hysplit. Next, it uses the matplotlib library. Finally, we use Bokeh for and interactivate and more data-rich experience.

References:

Stein, A.F., Draxler, R.R, Rolph, G.D., Stunder, B.J.B., Cohen, M.D., and Ngan, F., (2015). NOAA's HYSPLIT atmospheric transport and dispersion modeling system, Bull. Amer. Meteor. Soc., 96, 2059-2077, http://dx.doi.org/10.1175/BAMS-D-14-00110.

Rolph, G., Stein, A., and Stunder, B., (2017). Real-time Environmental Applications and Display sYstem: READY. Environmental Modelling & Software, 95, 210-228, https://doi.org/10.1016/j.envsoft.2017.06.025this link opens in a new window. (http://www.sciencedirect.com/science/article/pii/S1364815217302360)

1.2.7 IMEX

IMEX-SfloW2d is a depth-averaged numerical flow model for pyroclastic avalanches. The configuration file is extremely in depth, so the workflow splits it into more manageable pieces. We begin with simple parameters to set a run name, simulation time constraints, and output files. Next are radial source parameters, described as where `The source of mass is initialized. The cells belonging to the source are are identified ( source_cell(j,k) = 2 )`. The next cell sets bounds for the DEM we use, and some flags that allow for more granular setting of constants. The next cell functions as a sanity check for the DEM.

After the DEM, we set temperature parameters of the environment and related material thermal constants, followed by the algorithms selected for the numerical slope calculations for each cell. Gravity is a configurable option for future flexibility. Rheological parameters and constants are then assigned, followed by gas transport parameters, which constitute gas attributes and pressure specification.

The given parameters are a condensed version of the overall choices. Additional scenarios can be added, such as the pyroclastic source generating from a collapsing volume. Further documentation will be provided in the future, though the souce code is the only reference for now. All values after the DEM check can be kept as is for a reasonable estimate. The three cells before are the only places that must be changed in reference to the DEM to function properly.

Subsequent cells write out the config files and run the model. The only other place input is neccesary is a one line cell with the `step` variable. IMEX outputs data at every dt chosen by the user, so in order to view data at a given timestamp, you **must** choose a step. All subsequent cells can ran without input to give a detailed output of both temperature and thickness of the flow at a given time. Additionally, seperate netCDF files containing time series data for the temperature and depth are both supplied as output, along with a JPG of the figure.

References:

Elisa Biaioli's thesis: https://dx.doi.org/10.15167/biagioli-elisa_phd2021-10-27

E. Biagioli, M. de' Michieli Vitturi, and F. Di Benedetto. Modified shallow water model for viscous fluids and positivity preserving numerical approximation. Applied Mathematical Modeling, 94:482–505, 2021. doi: 10.1016/j.apm.2020.12.036.

M. de' Michieli Vitturi, T. Esposti Ongaro, G. Lari, and A. Aravena. IMEX_SfloW2D 1.0. a depth-averaged numerical flow model for pyroclastic avalanches. Geosci. Model Dev., 12: 581–595, 2019. doi: 10.5194/gmd-12-581-2019.

1.2.8 MOLASSES

MOdular LAva Simulation Software for Earth Science, or MOLASSES for short, is a probabalistic lava flow simulation tool. The required inputs are very straightforward. In the first cell after the imports, all the user mnust enter is the residual thickness, the total volume of lava erupted, the pulse volume per simulation tick, and the DEM filename, along with the origin points in UTM of the eruption. The user may optionally repeat runs due to the probabalisticnature of the model. After this cell, the rest of the model can run without input. If desired, the zoom level can be selected between a snapshot of the flow area and the overall DEM with the flow overlayed. The workflow will output a well formatted CSV named `flow.csv` for the user, as well as a JPG of the final figure.

References:

Connor, L. J., Connor, C. B., Meliksetian, K., & Savov, I. (2012) Probabilistic approach to modeling lava flow inundation: a lava flow hazard assessment for a nuclear facility in Armenia. Journal of Applied Volcanology (1):3. DOI 10.1186/2191-5040-1-3

Kubanek, J., Richardson, J. A., Charbonnier, S. J., & Connor, L. J. (2015) Lava flow mapping and volume calculations for the 2012–2013 Tolbachik, Kamchatka, fissure eruption using bistatic TanDEM-X InSAR. Bulletin of Volcanology 77(12):106. DOI 10.1007/s00445-015-0989-9

1.2.9 MrLavaLoba

MrLavaLoba is a stochastic model for simulating lava flows, written in Python. The workflow for this model begins with a large amount of text, explaining input parameters in detail. After neccesary libraries are imported, all parameters are in the next cell. A DEM sanity check follows, continuing on to write out the input files and run the model. A convenient progress bar will show the remaining time for model calculations. MrLavaLoba outputs snapshows at a given *dt* interval, so the user must pick a step to visualize. The rest of the workflow configures and displays the flow based on the output shapefiles given, saving a JPG of the final figure.

References:

M. de' Michieli Vitturi and S. Tarquini. MrLavaLoba: A new probabilistic model for the simulation of lava flows as a settling process, Journal of Volcanology and Geothermal Research, Volume 349, 2018, Pages 323-334, ISSN 0377-0273, https://doi.org/10.1016/j.jvolgeores.2017.11.016.

1.2.10 pyFLOWGO

Lava flow advance may be modeled through tracking the evolution of the lava's thermo-rheological properties, which are defined by viscosity and yield strength. These rheological properties evolve, in turn, with cooling and crystallization. Such model was conceived by Harris and Rowland (2001) who developed a 1-D model, FLOWGO, in which velocity of a control volume flowing down a channel depends on rheological properties computed following the lava cooling and crystallization path estimated via a heat balance box model. pyFLOWGO is an updated version written completely in Python for increased flexibility and modernity.

The first input cell directly follows the imports, simply asking for the name of the flow, the slope file, which is *not* a DEM, and the step size. The next cell requests flags to calculate a specific type of flux. Following this, the user must pick the method used for calculating various aspects of the lava's physical properties. Next, the physical dimensions of the channel should be entered. The final two cells specify eruption event parameters and thermal parameters. All subsequent cells can be run without further alteration. In this case, the visualizations are done through a Python script included in the pyFLOWGO library.

References:

Chevrel, M., Labroquere, J., Harris, A., and Rowland, S. (2017). Pyflowgo: an open-source platform for simulation of channelized lava thermo-rheological properties. Computational Geosciences.

1.2.11 Tephra2

Tephra2 is a tephra dispersion model, that estimates the mass of tephra that would accumulate at a site or over a region, given explosive eruption conditions. There are a variety of inputs required here for an accurate representation.

The user must first input coordinate and date information to grab reanalysis data. In order to make the experience as simple as possible, we use the Copernicus API. However, as long as the user follows the provided format in the Github. The user can then run the next handful of cells until they see the heading for the configuration file. Here, the user must input quantitative data about the tephra expulsion itself, though the vent UTM coordinates are assumed to be at the same position as the wind file by default. Following the first 7 main inputs, another 12 optional inputs are included for more granular modeling, though defaults will be used if not set. The user can then continue again until they reach the grid file header. The grid radius, spacing, and elevation must be input, where the the volcano's UTM coordinates again are assumed to be the same. From here, every cell through the end can be run resulting in an isomass tricontour of the tephra dispersion. The VICTOR team is working on adding a basemap and additional data to the visualization at the moment.

References:

Bonadonna, C., Connor, C. B., Houghton, B. F., Connor, L., Byrne, M., Laing, A., and Hincks, T. K. (2005) Probabilistic modeling of tephra dispersal: Hazard assessment of a multiphase rhyolitic eruption at Tarawera, New Zealand, Journal of Geophysical Research: Solid Earth 110(B3). DOI 10.1029/2003JB002896

Connor, Laura J., and Charles B. Connor (2006) Inversion is the key to dispersion: understanding eruption dynamics by inverting tephra fallout In H. M. Mader, S. G. Coles, C. B. Connor & L. J. Connor (Eds.), Statistics in Volcanology, Geological Society of London Special Publications 231. DOI 10.1144/IAVCEI001.18

Biass, Sebastien, Bagheri, Gholamhossein, Aeberhard, William H., and Bonadonna, Costanza (2014) TError: towards a better quantification of the uncertainty propagated during the characterization of tephra deposits, Statistics in Volcanology 1(2):1-27. DOI 10.5038/2163-338X.1.2

Biass, S., Bonadonna, C., Connor, L., and Connor, C. (2016) TephraProb: a Matlab package for probabilistic hazard assessments of tephra fallout, Journal of Applied Volcanology 5(1):10. DOI 10.1186/s13617-016-0050-5

1.2.12 Titan2D

TITAN2D is a geoflow simulation software application, specifically used for granular flows. As a deterministic model, it requires a large array of parameters to be properly configured.

To begin, the user enters information for DEM format, the DEM itself, as well as some fundamental constants. This first section also includes iteration limits, and output intervals. Next, numeric parameters are required. The user can choose to toggle adaptive mesh refinements for more accurate calculations at each timestep, along with the size of the initial pile and the order of PDE to solve. Finally, the user must specify the material model and associated constants. We select the Coloumb model by default, though there are a total of four options.

Numerous optional additions can be made, including extra points of origin for lava, flux locations, and discharge planes for measuring flow over an are are all toggleable options for the user. After this, the user can run another 4 cells and choose a timestamp once the model finishes running. All following cells can then be run and result in a very detailed snapshot of the lava depth at the moment specified.

References: Patra, A., Bevilacqua, A., Akhavan-Safaei, A., Pitman, E. B., Bursik, M., & Hyman, D. (2020). Comparative analysis of the structures and outcomes of geophysical flow models and modeling assumptions using uncertainty quantification. Frontiers in Earth Science, 8. https://doi.org/10.3389/feart.2020.00275

1.2.13 TWODEE-2

TWODEE is a code for dispersion of heavy gases based on the solution of a shallow water equations system for fluid depth, depth-averaged horizontal velocities and depth-averaged fluid density. The workflow begins with a cell for the user to set parameters related to the date, runtime, and name of the current simulation. Next, the user must input spacing values and UTM values for the topography. If a file is provided, elevation is sourced from it though if not, a generalized slope is required from user entered values. The following two cells require numerical terms, including the densities of the two gasses being compared and many environmental and entrainment coefficients as well as physical constants. Subsequently, the user is asked to enter some location data for the meteorology, or more aptly the wind. The second to last configuration cell simply asks the user to enter paths to various files, depending on the mode the user chose. If not required, the cell can be left blank or as-is from the template. Finally, output parameters can be withheld or added as needed, allowing for highly flexible output files.

The next two cells can be run without any change, as they are creating a formatted input file and running the model. The following two cells open the result file and give a brief description of the possible values to display. These values range from wind velocity and cloud thickness to gas concentration and altitude of critical concentration. Currently, the user must then enter the set of values they want to display, and a lower bound. The bound allows for more accurate visualizations due to negligable low value data points. The final cell can be run as is, and will result in a sharp, detailed plot of the chosen data over the topography.

References Hankin, R., Britter, R. (1999a). TWODEE: the Health and Safety Laboratory's shallow layer model for heavy gas dispersion. Part 1. Mathematical basis and physical assumptions. J. Hazard. Mater. A66, 211-226.

Hankin, R., Britter, R. (1999b). TWODEE: the Health and Safety Laboratory's shallow layer model for heavy gas dispersion. Part 2. Outline and validation of the computational scheme. J. Hazard. Mater. A66, 227-237.

Hankin, R., Britter, R. (1999c). TWODEE: the Health and Safety Laboratory's shallow layer model for heavy gas dispersion. Part 3. Experimental validation (Thorney island). J. Hazard. Mater. A66, 237-261.

Costa A., Chiodini G., Granieri D., Folch A., Hankin R.K.S., Caliro S., Cardellini C., Avino R. (2008). A shallow layer model for heavy gas dispersion from natural sources: application and hazard assessment at Caldara di Manziana, Italy., Geochem. Geophys. Geosyst., 9, Q03002, doi: 10.1029/2007GC001762.

Folch A., Costa A., Hankin R.K.S., 2009. TWODEE-2: A shallow layer model for dense gas dispersion on complex topography, Comput. Geosci., doi:10.1016/j.cageo.2007.12.017

Chiodini G., Granieri D., Avino R., Caliro S., Costa A., Minopoli C., Vilardo G., (2010) Non-volcanic CO2 Earth degassing: The case of Mefite di Ansanto (Southern Apennines), Italy, Geophys. Res. Lett., Vol. 37, L11303, doi: 10.1029/2010GL042858

1.3 Victor Library

1.3.1 Victor Library Overview

The VICTOR library is, at its core, a collection of convenient custom functions for downloading, accessing, manipulating, and visualizing datum that are vital to a volcano scientist's research. As more models and wider functionality are added, the VICTOR library will grow.

1.3.2 Plotting Model Outputs

```
def plot_dem(dem, fig, ax, coords=None)
```

`dem`: The relative or absolute filepath to DEM. Must be an ascii shapefile or GeoTIFF format, as georeferencing is necessary for proper formatting.

`fig`: A `matplotlib.figure` object. This is the complete plot, both within the axes and outside. Example use case: `fig = plt.figure(figsize=(12, 8))`

`ax`: A `matplotlib.axes` object. This is the area concerning the data itself, as well as labels, titles, and the like. `ax = plt.axes(projection=ccrs.epsg(32628))`

`coords`: Optional parameter to add coordinates. If adding more than one set, provide in [[x_1,y_1],[x_2,y_2],...] format in multidimensional array.

```
def plot_flow(dem, flow, fig, ax, coords,zoom=True,model=None, label="Thickness of␣
↪residual (m)"):
```

`dem`: The relative or absolute filepath to DEM. Must be an ASCII shapefile or GeoTIFF format, as georeferencing is necessary for proper formatting.

`flow`: The relative or absolute file path to the flow data. Can be ASCII, geoTIFF, or csv formatted (without headers).

`fig`: A `matplotlib.figure` object. This is the complete plot, both within the axes and outside. Example use case: `fig = plt.figure(figsize=(12, 8))`

`ax`: A `matplotlib.axes` object. This is the area concerning the data itself, as well as labels, titles, and the like. `ax = plt.axes(projection=ccrs.epsg(32628))`

`coords`: Optional parameter to add coordinates. If adding more than one set, provide in [[x_1,y_1],[x_2,y_2],...] format in multidimensional array.

`zoom`: Optional parameter to toggle between close-up view around flow only, and view at the scale of the overall raster.

`model`: Optional parameter, can be ignored unless using MrLavaLoba model, where this should be set to "mrlavaloba" to account for a slight variation in formatting.

`label`: Optional parameter to set the label for the color map in the plot, defaults to description of flow thickness.

```
def plot_titan(dem, flow, fig, ax, coords,zoom=True):
```

`dem`: The relative or absolute filepath to DEM. Must be in geoTIFF format, as georeferencing is necessary for proper formatting.

`flow`: Must be a Pandas dataframe.

`fig`: A `matplotlib.figure` object. This is the complete plot, both within the axes and outside. Example use case: `fig = plt.figure(figsize=(12, 8))`

`ax`: A `matplotlib.axes` object. This is the area concerning the data itself, as well as labels, titles, and the like. `ax = plt.axes(projection=ccrs.epsg(32628))`

`coords`: Optional parameter to add coordinates. If adding more than one set, provide in [[x_1,y_1],[x_2,y_2],...] format in multidimensional array.

`zoom`: Optional parameter to toggle between close-up view around flow only, and view at the scale of the overall raster.

```
def make_titan_gif(dem, fig, ax, coords, max_iter, diter, gif_name):
```

`dem`: The relative or absolute filepath to DEM. Must be in geoTIFF format, as georeferencing is necessary for proper formatting.

`fig`: A `matplotlib.figure` object. This is the complete plot, both within the axes and outside. Example use case: `fig = plt.figure(figsize=(12, 8))`

`ax`: A `matplotlib.axes` object. This is the area concerning the data itself, as well as labels, titles, and the like. `ax = plt.axes(projection=ccrs.epsg(32628))`

`coords`: Optional parameter to add coordinates. If adding more than one set, provide in [[x_1,y_1],[x_2,y_2],...] format in multidimensional array.

`max_iter`: The maximium number of iterations provided by the user in Titan2D.

`diter`: The number of iterations between outputs, set by the user in the Titan2D input file.

`gif_name`: The name used to save the gif to your current directory.

1.3.3 Upload/Download from the Cloud

```
def download_file_gcp(bucket_name, source_blob_name, destination_file_name, api_creds_
↪json):
```

`bucket_name`: The name of the Google Cloud bucket that you would like to access.

`source_blob_name`: The name of the file you would like to download from the bucket.

`destination_file_name`: The name you would like the file to have when it is downloaded locally.

`api_creds_json`: The relative or absolute path to the JSON file with the user's credentials. To obtain this file , refer to this documentation from Google. . Make sure the IAM API is active, and follow the directions linked.

```
def upload_file_gcp(bucket_name, source_file_name, destination_blob_name,api_cred_json):
```

bucket_name: The name of the Google Cloud bucket that you would like to access.

source_file_name: The name of the local file that you would like to upload to the specified bucket.

destination_blob_name: The name you would like the file to have when it is uploaded.

api_creds_json: The relative or absolute path to the JSON file with the user's credentials. To obtain this file , refer to this documentation from Google. . Make sure the IAM API is active, and follow the directions linked.

```
def download_file_aws(access_key, secret_access_key, bucket_name, blob_name, file_name,␣
↪session_token=None):
```

access_key and secret_access_key: These are unique identifiers for your AWS root or IAM account. Refer to instructions here to learn how to get these identifiers.

bucket_name: The name of the S3 bucket that you would like to access.

blob_name: The name of the file you would like to download from the bucket.

file_name: The name you would like the file to have when it is downloaded locally.

session_token: An optional field, used for time-limited access to a bucket. Recommended only for advanced users, or those familar with AWS.

```
def upload_file_aws(access_key, secret_access_key, bucket_name, blob_name, file_name,␣
↪session_token=None):
```

access_key and secret_access_key: These are unique identifiers for your AWS root or IAM account. Refer to instructions here to learn how to get these identifiers.

bucket_name: The name of the S3 bucket that you would like to access.

blob_name: The name the file should have when uploaded to the S3 bucket.

file_name: The name (and path if not in the current directory) of the local file to upload.

session_token: An optional field, used for time-limited access to a bucket. Recommended only for advanced users, or those familar with AWS.

```
def download_from_azure(conn_string, container_name, blob_name, local_file_name)
```

conn_string: A unique identifer to connect to an Azure storage. Refer to Microsoft's documentation for information on how to get your connection string.

container_name: The name of the Azure storage module that you would like to access.

blob_name: The name of the file you would like to download from Azure storage.

local_file_name: The name you would like the file to have when downloaded locally.

```
def upload_to_azure(conn_string, container_name, blob_name, local_file_name)
```

conn_string: A unique identifer to connect to an Azure storage. Refer to Microsoft's documentation for information on how to get your connection string.

container_name: The name of the Azure storage module that you would like to access.

blob_name: The name you would like the uploaded file to have in Azure storage.

local_file_name: The name (and path if not in the current directory) of the local file that should be uploaded.

1.3.4 Download DEMs

```
def download_dem(north, south, east, west, outputFormat, dataset)
```

`north`: Upper bound for latitude.

`south`: Lower bound for latitude.

`east`: Right bound for longitude.

`west`: Left bound for longitude.

`outputFormat`: Allows you to request either an ASCII shapefile (AAIGrid) or GeoTIFF (GTiff) formatted DEM.

`dataset`: Allows you to choose from different DEM providers. This allows for for varying resolutions, as well as alternatives in case of missing data. Additionally, some of the datasets include bathymetry, or unique coverage. The sets are defined as: SRTMGL3, SRTMGL1, SRTMGL1_E, AW3D30, AW3D30_E, SRTM15Plus, NASADEM, COP30, COP90, EU_DTM.

```
def download_dem_utm(north, zone_north, south, zone_south, east, zone_east, west, zone_
↪west, outputFormat, dataset):
```

`north`: Upper bound for northing in UTM.

`zone_north`: UTM zone for the upper bound northing coordinate, format as a string with the number, then the letter, i.e. “10S”

`south`: Lower bound for northing in UTM.

`zone_south`: UTM zone for the lower bound northing coordinate.

`east`: Right bound for easting in UTM.

`zone_west`: UTM zone for the right bound easting coordinate.

`west`: Left bound for easting in UTM.

`zone_west`: UTM zone for the left bound easting coordinate.

`outputFormat`: Allows you to request either an ASCII shapefile (AAIGrid) or GeoTIFF (GTiff) formatted DEM.

`dataset`: Allows you to choose from different DEM providers. This allows for for varying resolutions, as well as alternatives in case of missing data. Additionally, some of the datasets include bathymetry, or unique coverage. The sets are defined as: SRTMGL3, SRTMGL1, SRTMGL1_E, AW3D30, AW3D30_E, SRTM15Plus, NASADEM, COP30, COP90, EU_DTM.