

1 **VESICAL: An open-source thermodynamic model engine**
2 **for mixed volatile (H₂O-CO₂) solubility in silicate melts**

3 **Kayla Iacovino¹, Simon Matthews^{2,3}, Penny Wieser⁴, Gordon M. Moore¹,**
4 **Florence Bégué⁵**

5 ¹Jacobs, NASA Johnson Space Center, Houston, TX 77058, USA

6 ²Johns Hopkins University, Department of Earth and Planetary Sciences, Baltimore, MD 21218, USA

7 ³University of Iceland, Institute of Earth Sciences, Askja, Sturlugata 7, 101 Reykjavik, Iceland

8 ⁴University of Cambridge, Department of Earth Sciences, Downing Street, Cambridge CB2 3EQ, UK

9 ⁵University of Geneva, Department of Earth Sciences, Geneva, Switzerland

10 **Key Points:**

- 11 • The first comprehensive volatile solubility tool capable of processing large datasets
12 automatically
13 • Seven built-in solubility models, with automatic calculation and plotting function-
14 ality
15 • Built in python and easily usable by scientists with any level of coding skill

Corresponding author: Kayla Iacovino, kayla.iacovino@nasa.gov

Abstract

Thermodynamics has been fundamental to the interpretation of geologic data and modeling of geologic systems for decades. However, more recent advancements in computational capabilities and a marked increase in researchers' accessibility to computing tools has outpaced the functionality and extensibility of currently available modeling tools. Here we present VESIcal (Volatile Equilibria and Saturation Identification calculator): the first comprehensive modeling tool for H₂O, CO₂, and mixed (H₂O-CO₂) solubility in silicate melts that: **a**) allows users access to seven of the most popular models, plus easy inter-comparison between models; **b**) provides universal functionality for all models (e.g., functions for calculating saturation pressures, degassing paths, etc.); **c**) can process large datasets (1,000's of samples) automatically; **d**) can output computed data into an Excel spreadsheet or CSV file for simple post-modeling analysis; **e**) integrates plotting capabilities directly within the tool; and **f**) provides all of these within the framework of a python library, making the tool extensible by the user and allowing any of the model functions to be incorporated into any other code capable of calling python. The tool is presented within this manuscript, which may be read as a static PDF but is better experienced via the Jupyter notebook version of this manuscript. We present here worked examples accessible to python users with a range of skill levels. The basic functions of VESIcal can also be accessed via a web app (<https://vesical.anvil.app>). The VESIcal python library is open-source and available for download at <https://github.com/kaylai/VESIcal> and can be installed via pip.

Plain Language Summary

Geologists use numerical models to understand and predict how volcanoes behave during storage (pre-eruption), eruption, and the composition and amount of volcanic gas released into the atmosphere of Earth and other planets. Most models are made by performing experiments on a limited dataset and creating a model that applies to that dataset. Some models combine lots of these individual models to make a generalized model that can apply to lots of different volcanoes. Many of these different models exist, and they all have specific uses, limitations, and pitfalls. Here we present the first tool, VESIcal, which acts as a simple interface to seven of the most commonly used models. VESIcal is written in python, so users can use VESIcal as an application or include it in their own models. VESIcal is the first tool that allows geologists to model thousands of data points automatically and provides a simple platform to compare results from different models in a way never before possible.

0.1 Introduction

Understanding the solubility and degassing of volatiles in silicate melts is a crucial component of modeling volcanic systems. As dissolved components, volatiles (primarily H₂O and CO₂) affect magma viscosity, rheology, and crystal growth. In addition, due to the strong dependence of volatile solubility on pressure, measured volatile concentrations in preserved high-pressure melts (i.e., melt inclusions: liquid magma trapped within crystals at high pressure, then brought to the surface during an eruption) can be used to determine pre-eruptive magmatic storage pressures, and thus depths. Importantly, volatile exsolution-driven overpressure of a magmatic system is likely the trigger of many explosive volcanic eruptions Tait et al. (1989); Blake (1984); Stock et al. (2016). Once triggered, further drops in magmatic pressure caused by ascent of magma within a volcanic conduit result in the continuous exsolution of volatiles from the melt. Volatile elements experience a large positive volume change when moving from a dissolved to exsolved free fluid state. This expansion fuels a dramatic increase in the magma's buoyancy, which can often lead to a runaway effect in which the ascent and degassing of volatile-bearing magma eventually erupts at the surface in an explosive fashion. Working in concert with seismic and gas monitoring data, pre-eruptive magmatic volatile concentrations

as well as solubility and degassing modelling can be used in forensic and sometimes in predictive scenarios, helping us to understand and potentially mitigate the effects of explosive eruptions.

All of these processes depend directly on the solubility – or the capacity of a magma to hold in solution – of volatile elements. Over the last several decades, a veritable explosion of new volatile solubility data has opened the door to a plethora of models describing the solubility of H_2O , CO_2 , or mixed H_2O - CO_2 fluid in magmas covering a wide compositional, pressure, and temperature range. Volatile solubility is highly dependent upon the composition of the host magma, making already challenging experiments more onerous to perform to encapsulate the range of magmas seen in nature. The most fundamental models Stolper (1982); Dixon et al. (1995); Moore et al. (1998) focus on a specific range of magma bulk compositions (e.g., basalt or rhyolite only). Later studies filled in compositional gaps, some with an increased focus on mixed-volatile (H_2O - CO_2) studies, increasing the natural applicability of our models to more systems Liu et al. (2005); Iacono-Marziano et al. (2012); Iacovino et al. (2013). To date, there have been only a few significant efforts to create a holistic thermodynamic model calibrated by a wide range of data in the literature. The most popular are MagmaSat (the mixed-volatile solubility model built into the software package MELTS v. 1.2.0; Ghiorso & Gualda (2015)) and the model of Papale et al. (2006). Both of these studies have made their source code available; the Papale et al. (2006) FORTRAN source code (titled Solwcad), web app, and a Linux program can be found at <http://www.pi.ingv.it/progetti/eurovolc/>, and very recently MagmaSat has been made accessible via the ENKI thermodynamic python framework (<http://enki-portal.org/>).

Despite this communal wealth of solubility models, quantitative calculations of volatile solubility, and by extension saturation pressures, equilibrium fluid compositions, and degassing paths, remains a time-consuming endeavor. Modeling tools that are available are typically unable to process more than one sample at a time, requiring manual entry of the concentrations of 8-10 major oxides, temperature, as well as CO_2 and H_2O concentrations to calculate saturation pressures, or $X_{\text{H}_2\text{O}}$ to calculate dissolved volatile contents. This is particularly problematic for melt inclusion studies, where saturation pressures are calculated for hundreds of inclusions, each with different entrapment temperatures, CO_2 , H_2O , and major element concentrations. For example, the saturation pressures from 105 Gakkell ridge melt inclusions calculated in MagmaSat by Bennett et al. (2019) required the manual entry of 1,365 values! The potential for user error in this data entry stage should not be overlooked.

In many cases, newly published solubility models do not include an accompanying tool, requiring users to correctly combine and interpret the relevant equations Dixon et al. (1995); Dixon (1997); Liu et al. (2005); Shishkina et al. (2014). This is problematic from a perspective of reproducibility of the multitude of studies utilizing these models, especially given that some of the equations in the original manuscripts contain typos or formatting errors. For some models, an Excel spreadsheet was provided, or available at request from the authors. For example, Newman & Lowenstern (2002) included a simplified version of the Dixon (1997) model as part of “VolatileCalc”, which was written in Visual Basic for Excel. Due to its simplicity, allowing users to calculate saturation pressures, degassing paths, isobars and isopleths with a few button clicks and pop-up boxes, this tool has proved extremely popular (with 836 citations at the time of writing). However, to calculate saturation pressures using VolatileCalc, the user must individually enter the SiO_2 , H_2O , CO_2 content and temperature of every single sample into pop-up boxes. Similarly, the Excel spreadsheet for the Moore et al. (1998) model calculates dissolved H_2O contents based on the concentration of 9 oxides, temperature, and the fraction of $X_{\text{H}_2\text{O}}$ in the vapor, which must be pasted in for every sample. Finally, Allison et al. (2019) provide an Excel spreadsheet that allows users to calculate fugacities, partial pressures, isobars, isopleths and saturation pressures. Again, parameters

120 for each sample must be entered individually, with no way to calculate large numbers
 121 of samples automatically.

122 Some of these published models and tools are at risk of being lost to time, since
 123 spreadsheet tools (particularly earlier studies published before journal-provided hosting
 124 of data and electronic supplements was commonplace) must be obtained by request to
 125 the author. Even if the files are readily available, programs used to open and operate
 126 them may not support depreciated file formats. More recently, authors have provided
 127 web-hosted interfaces to calculating saturation pressures and dissolved volatile contents
 128 (e.g., Iacono-Marziano et al. (2012); <http://calcul-isto.cnrs-orleans.fr/>, and Ghiorso &
 129 Gualda (2015); http://melts.ofm-research.org/CORBA_CTserver/GG-H2O-CO2.html).
 130 Ghiorso & Gualda (2015) also provide a Mac application. While more accessible in the
 131 present time, this does not negate the issue of the longevity of these models. The link
 132 provided in the Iacono-Marziano et al. (2012) manuscript returns an error “this site can-
 133 not be reached”, although email contact with the author directed us towards the newer
 134 link given above. Similarly, the link to the H₂O-CO₂ equation of state web calculator
 135 that Duan & Zhang (2006) provided in their manuscript returns a 404 error.

136 While we certainly advocate for the continued refinement of solubility models, in-
 137 cluding the completion of new experiments in poorly studied yet critical compositional
 138 spaces such as andesites Wieser et al. (2021), a perhaps more crucial step at this junc-
 139 ture is in the development of a tool that can apply modern computational solutions to
 140 making our current knowledge base of volatile solubility in magmas accessible and en-
 141 doring.

142 Here we present VESIcal (Volatile Equilibria and Saturation Identification calcu-
 143 lator): a python-based thermodynamic volatile solubility model engine that incorporates
 144 seven popular volatile solubility models under one proverbial roof. The models included
 145 in VESIcal are (also see Table 1):

- 146 1. MagmaSat: VESIcal’s default model. The mixed-volatile solubility model within
 147 MELTS v. 1.2.0 Ghiorso & Gualda (2015)
- 148 2. Dixon: The simplification of the Dixon (1997) model as implemented in Volatile-
 149 Calc Newman & Lowenstern (2002)
 - 150 • DixonWater and DixonCarbon available as pure-fluid models
- 151 3. MooreWater: (Moore et al. (1998); water only, but H₂O fluid concentration can
 152 be specified)
- 153 4. Liu: Liu et al. (2005)
 - 154 • LiuCarbon and LiuWater available as pure-fluid models
- 155 5. IaconoMarziano: Iacono-Marziano et al. (2012)
 - 156 • IaconoMarzianoWater and IaconoMarzianoCarbon available as pure-fluid mod-
 157 els
- 158 6. ShishkinaIdealMixing: Shishkina et al. (2014) using pure-H₂O and pure-CO₂ mod-
 159 els and assuming ideal mixing. In general, the pure-fluid versions of this model
 160 should be used
 - 161 • ShishkinaWater and ShishkinaCarbon available as pure-fluid models
- 162 7. AllisonCarbon: Allison et al. (2019)
 - 163 • AllisonCarbon_vesuvius (default; phonotephrite from Vesuvius, Italy)
 - 164 • AllisonCarbon_sunset (alkali basalt from Sunset Crater, AZ, USA)
 - 165 • AllisonCarbon_sfvt (basaltic andesite from San Francisco Volcanic Field, AZ,
 166 USA)
 - 167 • AllisonCarbon_erebus (phonotephrite from Erebus, Antarctica)

- 168 • AllisonCarbon_etna (trachybasalt from Etna, Italy)
- 169 • AllisonCarbon_stromboli (alkali basalt from Stromboli, Italy)

170 As any individual model is only valid within its calibrated range (see below), and
 171 each model is parameterized and expressed differently (e.g., empirical vs. thermodynamic
 172 models), it is impractical to simply combine them into one large model. Instead, VESI-
 173 cal is a single tool that can access and utilize all built-in models, giving VESIcal an ex-
 174 tensive pressure-temperature-composition calibration range ?. VESIcal is capable of per-
 175 forming a wide array of calculations on large datasets automatically, with built-in func-
 176 tionality for extracting data from an Excel or CSV file. In addition, the code is written
 177 such that it is flexible (sample, calculation type, and model type can be chosen discreetly)
 178 and extensible (VESIcal code can be imported for use in python scripts, and the code
 179 is formatted such that new volatile models can be added).

180 Importantly, VESIcal has been designed for practicality and ease of use. It is de-
 181 signed to be used by anyone, from someone who is completely unfamiliar with coding
 182 to an adept programmer. The non-coder user can interact with VESIcal through a we-
 183 bapp (<https://vesical.anvil.app>) or directly within this manuscript, which utilizes the user-
 184 friendly Jupyter Notebook format, allowing them to upload a file with data, execute the
 185 various example calculations provided below, and save the results to an Excel or CSV
 186 file to work with outside of VESIcal. This notebook also incorporates built-in plotting
 187 options for easy visualization of user data and calculated results. More experienced pro-
 188 grammers may wish to use the more advanced functionality provided by VESIcal, includ-
 189 ing the ability to hybridize models (e.g., use one model for H₂O and another for CO₂)
 190 or write their own routines and code calling VESIcal methods. VESIcal is an open source
 191 tool and as such is far less prone to the preservation issues discussed above. Because the
 192 VESIcal code is hosted on GitHub, every change to the code is tracked publicly Perkel
 193 (2016). VESIcal's current release (version 1.0.1) is also archived on Zenodo, which pro-
 194 vides a static citable DOI Iacovino, Matthews, Wieser, Moore, & Begue (2021) for the
 195 current version of the code, along with a snapshot of the GitHub repository at the time
 196 of release.

197 A detailed history of volatile solubility modeling and the implications of VESIcal
 198 are explored in detail in the companion manuscript to this work, Wieser et al. (2021).

199 0.2 Research Methodology

200 Navigating the array of models implemented in VESIcal can be challenging. How
 201 can a user determine which model best suits their needs? MagmaSat (the default model
 202 in VESIcal) is the most widely calibrated in P-T-X space, and so we recommend it for
 203 the majority of cases. Where a user wishes to use the other implemented models, we pro-
 204 vide some tools to help choose the most appropriate model (see Supplement). These tools
 205 are described in more detail in Section 0.3.4 on comparing user data to model calibra-
 206 tions.

207 A list of model names recognized by VESIcal can be retrieved by executing the com-
 208 mand `v.get_model_names()`, assuming VESIcal has been imported as `v` as is demon-
 209 strated in worked examples below. Note that the model names as listed in the previous
 210 section are given in terms of how to call them within VESIcal (e.g., `model='MooreWater'`).
 211 Allison et al. (2019) provides unique model equations for each of the six alkali-rich mafic
 212 magmas investigated in their study. The default model in VESIcal is that calibrated for
 213 Vesuvius magmas, whose calibration has the widest pressure range of the study Table 1.
 214 Setting a model name of `'AllisonCarbon'` within VESIcal will thus result in calcula-
 215 tions using the AllisonCarbon_vesuvius model equations.

Table 1. Calibration ranges of VESIcal models

Model/Reference	Species	P (bar)	T (°C)	Compositional range	Notes
MagmaSat Ghiorso & Gualda (2015)	H ₂ O	0–20,000 ¹	550–1420 ¹	Very broad compositional range of natural silicate melts: sub-alkaline microbasalts to rhyolites, including a variety of mafic and silicic alkaline compositions	¹ Ranges extracted from Fig. 2d of Ghiorso & Gualda (2015)
CO ₂	0–30,000 ¹	1139–1400 ¹			
H ₂ O - CO ₂	0–10,000 ¹	800–1400 ¹			
Dixon Simplification of Dixon (1997) used in VolatileCalc (Newman & Lowenstern (2002))	H ₂ O - CO ₂	0–5000 ¹ 0–2000 ² 0–1000 ³	600–1500 ¹ (1200) ⁴	Alkali basalts: 40–49 wt% SiO ₂	¹ Warnings implemented in VolatileCalc (Newman & Lowenstern (2002)). ² Calibration range suggested by Lesne et al. (2011) ³ Calibration range suggested by Iacono-Marziano et al. (2012) ⁴ Calibration temperature of Dixon (1997)
Moore Moore et al. (1998)	H ₂ O	0–3000 ¹	700–1200 ¹	Broad compositional range: sub-alkaline basalts to rhyolites, alkaline trachybasalts-andesites, foidites, phonolites	¹ Author-suggested calibration range. The calibration dataset spans 190 to 6067 bar, and 800–1200°C
Liu Liu et al. (2005)	H ₂ O - CO ₂	0–5000 ¹	700–1200 ¹	Haplogranites and rhyolites	¹ Author-suggested calibration range for the mixed fluid model. The calibration dataset

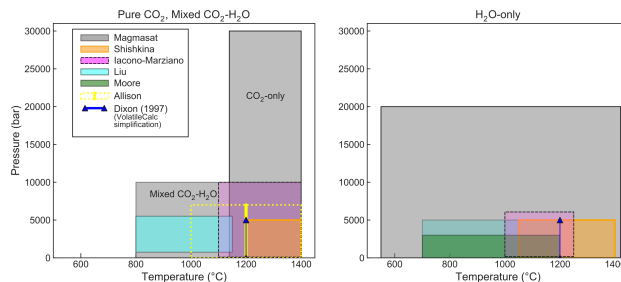


Figure 1. Illustrations showing the calibrated ranges of VESICAL models in pressure-temperature space. Due to difficulty in differentiating between pure- CO_2 and mixed fluid experiments in the literature, plots are subdivided into: experiments performed with pure- CO_2 or mixed ($\text{H}_2\text{O}-\text{CO}_2$) fluid; and pure- H_2O fluid.

216 All of the calculations implemented in VESICAL can be performed using any of the
 217 models included. The code is structured by calculation rather than by model, which pro-
 218 vides an intuitive way for users to interact with the code and compare outputs from mul-
 219 tiple models. Each calculation class is instantiated with the model name and any appli-
 220 cable data as arguments. It then performs five key functions: 1) creates the requested
 221 model object and performs any necessary pre-processing (e.g., ensuring relevant data are
 222 present; normalizing data); 2) takes user input and performs the mathematical calcula-
 223 tion; 3) does any necessary processing of the output (e.g., normalizing totals); 4) checks
 224 that the model is being used within its calibrated range; and 5) stores calculated out-
 225 puts in an intuitive and manipulatable format (e.g., a python dictionary, a figure, or a
 226 pandas DataFrame). Results of calculations can be saved to one or more Excel or CSV
 227 files. To demonstrate that VESICAL returns results which are comparable with pre-existing
 228 tools, we have performed a number of tests, which are described in the Supplementary
 229 Information (Text S2). For single-sample calculations, the calculation object has the fol-
 230 lowing attributes that can be called by the user: `model_name`, `sample` (both provided
 231 by the user), `model` (an instance of the Model class used to run the calculations of in-
 232 terest), `result` (the result of the calculations), and `calib_check` (the results of the cali-
 233 bration check).

234 *0.2.1 Model Calibrations and Benchmarking*

235 The pressure, temperature, and composition calibration ranges of the seven mod-
 236 els implemented in VESICAL are shown in Table 1 and Figure 1. VESICAL abides by state-
 237 ments of caution made by the authors of these models regarding their extrapolation by
 238 informing the user if a calculation is being performed outside of a model’s calibrated range.
 239 In this case, the code returns a warning message, which is as specific as possible, along
 240 with the requested output. We also provide several Jupyter notebooks in the supplement-
 241 ary material (Supplementary Text S3-S4 and Supplementary files S1-S7), allowing users
 242 to plot their data amongst the calibrations of the different models to assess their suit-
 243 ability for less objective measures. Detailed descriptions of the seven solubility models
 244 implemented in VESICAL, including information about their calibration range in terms
 245 of melt composition, pressure, and temperature, are given in Wieser et al. (2021)

246 Testing was undertaken to ensure that VESICAL faithfully reproduces the results
 247 of all incorporated models. When possible, all models were benchmarked by testing VESI-
 248 cal outputs against those of a relevant published calculator (e.g., web apps or Excel macros).
 249 The models of Shishkina et al. (2014) and Liu et al. (2005) were published with no such

250 tool and so testing instead compares VESICAL outputs to experimental conditions or anal-
 251 yses and, where possible, plots VESICAL results against published figures. All models un-
 252 derwent multiple tests, the results of which are shown in the supplement (Supplemen-
 253 tary Text S3-S4 and Supplemental Jupyter Notebooks S1-S7). For all models, VESICAL
 254 reproduced the results from previous tools (e.g., web apps, Excel spreadsheets) to within
 255 $\pm 1\%$ relative and often on the order of $\pm 0.1\%$ relative.

256 MagmaSat, VESICAL's default model, underwent three tests, the results of which
 257 are shown in Figure 2: 1. Comparison of saturation pressures from MORB melt inclu-
 258 sions in VESICAL to those published by Bennett et al. (2019), who used the MagmaSat
 259 Mac App ($R^2=0.99998$; Figure 2); 2. Comparison of fluid composition (X_{H_2O}) calculated
 260 with VESICAL and the web app ($R^2=0.999$, identical considering the web app returns 2dp;
 261 Figure 2); 3. Comparison of isobars for the Early Bishop Tuff calculated with VESICAL
 262 (star symbols) and isobars published in Figure 14 of Ghiorso & Gualda (2015) (Figure 2).
 263 VESICAL outputs using the model of Dixon (1997) were tested against outputs from the
 264 VolatileCalc Excel spreadsheet Newman & Lowenstern (2002) and a widely used Excel
 265 macro Tucker et al. (2019).

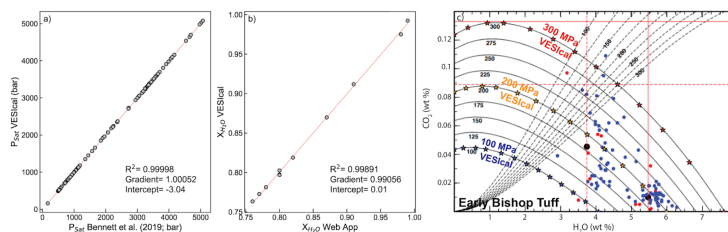


Figure 2. Benchmarking of VESICAL against MagmaSat. a. Comparison of saturation pressures calculated with VESICAL against those by Bennett et al. (2019) using the MagmaSat app for Mac. Samples are all MORB melt inclusions, and pressures were calculated at a temperature unique to each sample. b. Equilibrium fluid compositions calculated with VESICAL against those calculated with the MagmaSat web app. c. Individual points along the 1,000, 2,000, and 3,000 bar isobars for the Early Bishop Tuff rhyolite calculated with VESICAL (stars) and plotted atop isobars published in Figure 14 of Ghiorso & Gualda (2015).

266 0.2.2 Format of the python library

267 In this section, the basic organization and use cases of VESICAL are discussed. VESI-
 268 cal relies heavily on python pandas, a python package designed for working with tab-
 269 ulated data. Knowledge of pandas is not required to use VESICAL, and we refer the user
 270 to the pandas documentation for an overview of the package (<https://pandas.pydata.org/pandas-docs/stable/userguide/index.html>).
 271

272 Specific details on how to perform model calculations are discussed in Section 0.3
 273 and include worked examples. The VESICAL library is written so that users can inter-
 274 act first and foremost with the calculation they want to perform. Five standard calcu-
 275 lations can be performed with any model in the library:

- 276 1. `calculate_dissolved_volatiles()`
- 277 2. `calculate_equilibrium_fluid_composition()`
- 278 3. `calculate_saturation_pressure()`
- 279 4. `calculate_isobars_and_isopleths()` (plus functionality for plotting; only for
 280 mixed volatiles models)

281 5. `calculate_degassing_path()` (plus functionality for plotting; only for mixed volatiles
282 models).

283 Figure 3 illustrates the basic organization of the code. First, the user determines
284 which calculation they wish to perform by accessing one of the five core calculation classes
285 (listed above). In this step, the user specifies any input parameters needed for the cal-
286 culation (e.g., sample composition in wt% oxides, pressure in bars, temperature in °C,
287 and fluid composition “X_{fluid}” in terms of XH₂O^{fluid}) as well as the model they wish
288 to use. The default model is MagmaSat, but the user may specify any model in the li-
289 brary. As an example, the code to calculate the saturation pressure of some sample us-
290 ing the MagmaSat model would be written as:

```
saturation_pressure_calculation =
→ calculate_saturation_pressure(sample=mysample, temperature=850.0)
```

291 where `mysample` is a variable (VESIcal Sample object) containing the composition
292 of the sample, and the temperature is given in °C. Here, this line of code creates a Cal-
293 culate object, which is something that can be given a variable name and stored so that
294 the user can call upon this object for viewing or manipulation later. In this example, we
295 name the object “saturation_pressure_calculation”, but this can be any variable name
296 desired by the user. The Calculate object stores important information about the cal-
297 culation, including the result. The result of the calculation or calibration check can be
298 accessed as:

```
saturation_pressure_calculation.result
saturation_pressure_calculation.calib_check
```

299 In python, the object creation and attribute access can be combined into a single
300 line, with the understanding that the Calculate object will not be accessible to the user.
301 This usage is used in the remaining examples throughout the manuscript and would be
302 written as:

```
saturation_pressure = calculate_saturation_pressure(sample=mysample,
→ temperature=850.0).result
```

303 If a different model is desired, for example Dixon (1997), it can be passed as:

```
calculate_saturation_pressure(sample=mysample, temperature=850.0,
→ model='Dixon').result
```

304 The core calculation classes each perform two functions: 1) a check is performed
305 to ensure that the user input is within the model’s recommended calibration range; 2)
306 the `calculate()` method sends the user input to the appropriate model.

307 Users can process individual samples (single-sample calculations) or entire datasets
308 (batch calculations; Figure 4). If processing more than one sample, the “simplest” way
309 to interact with VESIcal is via batch calculations. Here, the user provides input data
310 in the form of a Microsoft Excel spreadsheet (.xlsx file) or CSV file and instructs the model
311 to perform whatever calculation is desired. The model is run on all samples and returns
312 data formatted like a spreadsheet (using the python pandas package), which contains
313 the user’s original input data plus whatever model outputs were calculated. The user
314 can continue to work with returned data by saving the result to a variable (as is shown
315 in all examples in this manuscript). Data can then be exported to an Excel or CSV file
316 with a simple command (see Section 0.3.12).

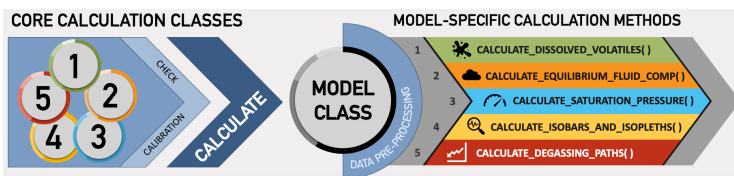


Figure 3. Flowchart illustrating the basic organization of the python library. First, a user chooses a calculation to perform and calls one of the five core calculation classes. Here, any necessary parameters are passed such as sample composition, pressure, and temperature. A check is run to ensure the calculation is being performed within model-specified limits. The `Calculate()` class then calls on one of the `Model()` classes. The default model is MagmaSat, but a user may specify a different model when defining the calculation parameters. Standard pre-processing is then performed on the input data, and this pre-processing step is unique to each model. The processed data are then fed into a model-specific method to perform the desired core calculation.

317 The syntax for processing a single sample is very similar to that for batch calculations
 318 but provides the user direct access to more advanced features that cannot be accessed
 319 via batch calculations (e.g., specifying fugacity or activity model, hybridizing models;
 320 see Section 0.3.11). This also gives the user more flexibility in integrating any VESI-
 321 cal model function into some other python code.

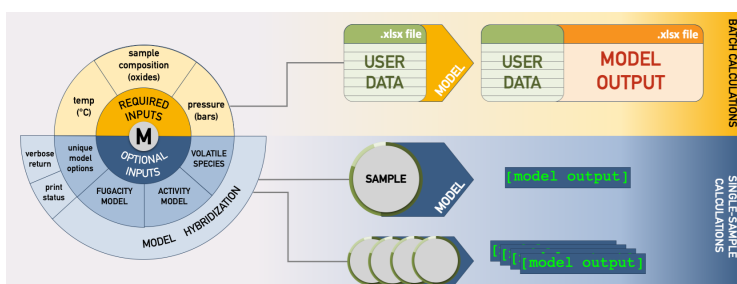


Figure 4. Flowchart illustrating the different operational paths. On top, batch calculation is shown, in which an Excel or CSV file with any amount of samples is fed into the model, calculations are performed, and the original user data plus newly calculated values are returned and can be saved as an Excel or CSV file. Below, single-sample calculation is shown. These methods can run calculations on one sample at a time, but multi-sample calculations can be performed iteratively with code written by the user. Calculated values are returned as a variable. For single-sample calculations, more advanced modeling options can be set, and hybridization of models can be performed.

322 0.2.3 Running the code

323 VESICAL can be used in a number of ways: via this Jupyter notebook, via the VESI-
 324 cal web app, or by directly importing VESICAL into any python script.

325 VESICAL was born from functionality provided by ENKI and so all the files neces-
 326 sary to use VESICAL are hosted on the ENKI server (<http://enki-portal.org/>). A unique
 327 personal coding environment can be initiated by logging into the ENKI production server
 328 using a GitLab username and password (which is free to obtain; see directions on the

ENKI website for specifics). The simplest way to use VESICAL while retaining all of its functionality is within this very manuscript, in the form of a Jupyter notebook. Because this manuscript and VESICAL python library files are hosted on the ENKI server, code can be manipulated and executed in the code cells below. Making changes won't affect the public version of this manuscript. Likewise, any user can write their own python code using VESICAL by creating a Jupyter notebook on the ENKI server and importing VESICAL as is demonstrated in the code below.

Computation time on the ENKI server is limited by the server itself. VESICAL may run faster if installed locally. Advanced instructions on installing VESICAL on your own computer are provided in the Supplement (Supplementary Text S1). Note that VESICAL requires installation of the ENKI thermoengine library to function properly. Thermoengine is written in python but is based on the original MELTS code Ghiorso & Sack (1995); Ghiorso & Gualda (2015), which contains MacOS-specific header files. The result is that thermoengine is most easily installed on MacOS but can be installed on Windows and Linux operating systems via Docker (see thermoengine documentation for installation instructions; <https://gitlab.com/ENKI-portal/ThermoEngine>).

The most limited but simplest method to interacting with VESICAL is through the web app (<https://vesical.anvil.app>). The web app can currently perform three of the five core calculations in batch process mode (via upload of an Excel or CSV file). Some, but not all, optional parameters can be set.

To run the code in this notebook, nothing needs to be installed. Simply execute the code cells below, changing parameters as desired. Custom data may be processed by uploading an Excel or CSV file into the same folder containing this notebook and then changing the filename in Section 0.3.3.

0.2.4 Documentation

This manuscript serves as an introduction to the VESICAL library aimed at python users of all levels. However, the code itself is documented with explanations of each method, its input parameters, and its returned values. This documentation can be accessed at our readthedocs website (<https://vesical.readthedocs.io/>). The documentation for any function can be viewed in a Jupyter Notebook by typing the function followed by a question mark and executing the cell (e.g., `v.calculate_saturation_pressure?`).

Video tutorials are also available on the VESICAL YouTube (<https://www.youtube.com/channel/UCpvCCs5K>). The first tutorial covers the basics of VESICAL, and subsequent videos cover the calculation of saturation pressures, dissolved volatile contents, and degassing paths. More videos for specific features and uses are planned.

0.2.5 Generic methods for calculating mixed-fluid properties

VESICAL provides a set of methods for calculating the properties of mixed CO₂-H₂O fluids, which can be used with any combination of H₂O and CO₂ solubility model. The use of generic methods allows additional models to be added to VESICAL by defining only the (simpler) expressions describing pure fluid solubility. Non-ideality of mixing in the fluid or magma phases can be incorporated by specifying activity and fugacity models. A complete description of these methods, including all relevant equations, can be found in the Supplement (Supplementary Text S2).

0.3 Workable example uses

In this section we detail how to use the various functions available in VESICAL through worked examples. The python code presented below may be copied and pasted into a script or can be edited and executed directly within the Jupyter notebook version of this

376 manuscript. For all examples, code in Sections Section 0.3.2 and Section 0.3.4 must be
 377 executed to initialize the model and import data from the provided companion Excel file.
 378 The following sections then may be executed on their own and do not need to be exe-
 379 cuted in order.

380 In each example below, a generic “method structure” is given along with definitions
 381 of unique, required, and optional user inputs. The method structure is simply for illus-
 382 trative purposes and gives default values for every argument (input). In some cases, ex-
 383 ecuting the method structure as shown will not produce a sensible result. For example,
 384 the default values for the `plot()` function ? contain no data, and so no plot would be
 385 produced. Users should replace the default values shown with values corresponding to
 386 the samples or conditions of interest.

387 All examples will use the following sample data by default (but this can be changed
 388 by the user):

- 389 • Dataset from `example_data.xlsx` loaded in Section 0.3.3.1 (variable name `myfile`)
- 390 • Single composition defined in Section 0.3.3.2 (variable name `mysample`)
- 391 • Sample 10* extracted from `example_data.xlsx` dataset in Section 0.3.3.3 (vari-
 392 able name `sample_10`)

393 Calculations performed on single samples or on a dataset imported from an Ex-
 394 cel or CSV file containing many samples are executed in two distinct ways. Note that
 395 single sample calculations require that the argument `sample` be defined. To return the
 396 numerical result of the calculation, the `.result` method must be called, as shown be-
 397 low. Batch calculations are performed on the dataset itself, after that dataset is imported
 398 into VESICAL. Thus, the `sample` argument does not need to be defined discretely, since
 399 sample compositional information is stored within the dataset object. The two basic for-
 400 mats for performing calculations are:

401 **Single sample calculations** `myvariable = v.name_of_the_core_calculation(sample=mysample,`
 402 `argument1=value1, argument2=value2).result`
 403 **Batch calculations** `myvariable = myfile.name_of_the_core_calculation(argument1=value1,`
 404 `argument2=value2)`

405 where VESICAL has been imported as `v`, `myvariable` is some arbitrary variable name
 406 to which the user wishes to save the calculated output, `name_of_the_core_calculation`
 407 is one of the five core calculations, `mysample` is a variable containing compositional in-
 408 formation in wt% oxides, `myfile` is a variable containing an `BatchFile` object created
 409 by importing an Excel or CSV file, and `argument1`, `argument2`, `value1`, and `argument2`
 410 are two required or optional arguments and their user-assigned values, respectively.

411 Workable examples detailed here are:

- 412 1. Section 0.3.3
 - 413 • Section 0.3.3.1
 - 414 • Section 0.3.3.2
 - 415 • Section 0.3.3.3
 - 416 • Section 0.3.3.4
- 417 2. Section 0.3.5
- 418 3. Section 0.3.6
- 419 4. Section 0.3.7
- 420 5. Section 0.3.8
- 421 6. Section 0.3.9

- 422 7. Section 0.3.10.3
 423 8. Section 0.3.10.3
 424 9. Section 0.3.11
 425 10. Section 0.3.12

426 *0.3.1 Calculation class arguments and their definitions*

427 Each section below details what arguments are required or optional inputs and gives
 428 examples of how to perform the calculations. Table 2 lists all arguments, both required
 429 and optional, used in the five core calculations. Many of the function arguments have
 430 identical form and use across all calculations, and so we list these here. Any special cases
 431 are noted in the section describing that calculation.

432 The most commonly used arguments are:

433 **sample** *Single sample calculations only* The composition of a sample. A VESICAL Sam-
 434 ple object is created to hold compositional information about sample. A Sample
 435 object can be created from a dictionary or pandas Series containing values, with
 436 compositions of oxides in wt%, oxides in mol fraction, or cations in mol fraction.
 437 This argument is not needed for batch calculations since they are performed on
 438 BatchFile objects, which already contain sample information. See examples for
 439 details.

440 **temperature, pressure, and X_fluid** the temperature in °C, the pressure in bars, and
 441 the mole fraction of H₂O in the H₂O-CO₂ fluid, XH₂O^{fluid}. In all cases, X_fluid
 442 is optional, with a default value of 1 (pure H₂O fluid). Note that that X_fluid
 443 argument is only used for calculation of dissolved volatile concentrations.

444 *For single sample calculations* Temperature, pressure, and X fluid should be
 445 specified as a numerical value.

446 *For batch calculations* Temperature, pressure, and X_fluid can either be spec-
 447 ified as a numerical value or as strings referring to the names of columns within
 448 the file containing temperature, pressure, or X_fluid values for each sample. If
 449 a numerical value is passed for either temperature, pressure, or X_fluid, that
 450 will be the value used for one or all samples. If, alternatively, the user wishes
 451 to use temperature, pressure, and/or X_fluid information in their BatchFile ob-
 452 ject, the title of the column containing temperature, pressure, or X_fluid data
 453 should be passed in quotes (as a string) to **temperature**, **pressure**, and/or **X_fluid**,
 454 respectively. Note for batch calculations that if temperature, pressure, or XH₂O^{fluid}
 455 information exists in the BatchFile but a single numerical value is defined for
 456 one or both of these variables, both the original information plus the values used
 457 for the calculations will be returned.

458 **verbose** *Only for single sample calculations* Always an optional argument with a de-
 459 fault value of False. If set to True, additional values of interest, which were cal-
 460 culated during the main calculation, are returned in addition to the results of the
 461 calculation.

462 **print_status** *Only for batch calculations* Always an optional argument, which some-
 463 times defaults to True and other times defaults to False (see specific calculation
 464 section for details). If set to True, the progress of the calculation will be printed
 465 to the terminal. The user may desire to see the status of the calculation, as some
 466 calculations using MagmaSat can be somewhat slow, particularly for large datasets.

467 **model** Always an optional argument referring to the name of the desired solubility model
 468 to use. The default is always “MagmaSat”.

Table 2. Matrix of all arguments used in the five core calculations, the nature of the argument (required or optional) and the input type or default value.

SS = Single-sample. Batch = batch processing. Color of cells corresponds to the type of argument: green=required; orange=optional; gray=argument not used. Values in cells indicate the unit or type of data to input for required arguments or the default value in the case of optional arguments.

	dissolved_volatiles		equilibrium_fluid_saturation		pressure_isobars		isopleth_path	
	SS	Batch	SS	Batch	SS	Batch	SS	SS
sample	wt% oxides		wt% oxides		wt% oxides		wt% oxides	wt% oxides
temperature	°C	°C	°C	°C	°C	°C	°C	°C
pressure	bars	bars	bars	None				'saturation'
pressure_list							bars	
X_fluid	1	1						
isopleth_list							None	
verbose	False		False		False			
model	'MagmaSat'	'MagmaSat'	'MagmaSat'	'MagmaSat'	'MagmaSat'	'MagmaSat'	'MagmaSat'	'MagmaSat'
print_status		True		False		True	True	
smooth_isobars							True	
smooth_isopleths							True	
fractionate_vapor								0.0
init_vapor								0.0

469 **0.3.2 Initialize packages**

470 For any code using the VESICAL library, the library must be imported for use. Here
 471 we import VESICAL as `v`. Any time we wish to initialize a VESICAL object, that class name
 472 must be preceded by ‘`v.`’ (e.g., `v.calculate_saturation_pressure`). Specific exam-
 473 ples of this usage follow. Here we also import some other python libraries that we will
 474 be using in the worked examples below.

```
import VESICAL as v
import pandas as pd

#The following are options for formatting this manuscript
pd.set_option('display.max_colwidth', 0)
from IPython.display import display, HTML
%matplotlib inline
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

475 **0.3.3 Loading, viewing, and preparing user data**

476 All of the following examples will use data loaded in the code cells in this section.
 477 Both batch processing of data loaded from a file and single-sample processing are shown.
 478 An example file called `example_data.xlsx` is included with this manuscript. You can
 479 load in your own data by first ensuring that your file is in the same folder as this note-
 480 book and then by replacing the filename in the code cell below with the name of your
 481 file. The code cell below must be executed for the examples in the rest of this section
 482 to function properly.

483 **0.3.3.1 Batch processing** Batch calculations are always facilitated via the `BatchFile()`
 484 class, which the user uses to specify the filename corresponding to sample data. Load-
 485 ing in data is as simple as calling `BatchFile(filename)`. Optionally, `units` can be used
 486 to specify whether the data are in wt% oxides, mol fraction oxides, or mol fraction cations.
 487 Calculations will always be performed and returned with melt composition in the default
 488 units (wt% oxides unless changed by the user) and fluid composition in mol fraction.

489 **Structure of the input file:** A file containing compositions (and optional pres-
 490 sure, temperature, or $\text{XH}_2\text{O}^{\text{fluid}}$ information) on one or multiple samples can be loaded
 491 into VESICAL. The loaded file must be a Microsoft Excel file with the extension `.xls` or
 492 `.xlsx` or CSV file with the extension `.csv`. The file must be laid out in the same manner
 493 as the example file `example_data.xlsx`. The basic structure is also shown in Table 0.3.3.1.

494 Any extraneous columns that are not labeled as oxides or input parameters will
 495 be ignored during calculations. The first column titled ‘Label’ contains sample names.
 496 Note that the default assumption on the part of VESICAL is that this column will be ti-
 497 tled ‘Label’. If no ‘Label’ column is found, the first non-oxide column name will be set
 498 as the index column, meaning this is how samples can be accessed by name (see Section 0.3.3.3).
 499 An index column can be specified by the user using the argument `label` (see documen-
 500 tation below). The following columns must contain compositional information as oxides.
 501 The only allowable oxides are: SiO_2 , TiO_2 , Al_2O_3 , Fe_2O_3 , FeO , Cr_2O_3 , MnO , MgO , CaO ,
 502 NiO , CoO , Na_2O , K_2O , P_2O_5 , H_2O , and CO_2 . Currently, VESICAL can only read these
 503 oxide names exactly as written (e.g., with no leading or trailing spaces and with correct
 504 capitalization), but functionality to interpret variations in how these oxides are entered
 505 is planned (e.g., such that “sio2. ” would be understood as “SiO2”). All of these oxides
 506 need not be included; if for example your samples contain no NiO concentration infor-
 507 mation, you can omit the NiO column. Omitted oxide data will be set to 0 wt% concen-
 508 tration. If other oxide columns not listed here are included in your file, they will be ig-

509 nored during calculations. Notably, the order of the columns does not matter, as they
 510 are indexed by name rather than by position. Compositions can be entered either in wt%
 511 (the default), mol%, or mole fraction. If mol% or mole fraction data are loaded, this must
 512 be specified when importing the file.

513 Because VESICAL assumes a particular formatting of column names, we highly recom-
 514 mend that users examine their data after loading into VESICAL and before perform-
 515 ing calculations. The user data, as it will be used by VESICAL, can be viewed at any time
 516 with `myfile.get_data()` (see generation of Table 0.3.3.1 below).

517 Pressure, temperature, or $\text{XH}_2\text{O}^{\text{fluid}}$ data may optionally be included, if they are
 518 known. Column names for these data do not matter, as they can be specified by the user
 519 as will be shown in following examples.

520 The standard units used by VESICAL are always pressure in bars, temperature in
 521 °C, melt composition as oxides in wt%, and fluid composition as mol fraction (typically
 522 specified as X_{fluid} , the mol fraction of H_2O in an $\text{H}_2\text{O}-\text{CO}_2$ fluid, ranging from 0-1).
 523 Sample compositions may be translated between wt%, mol fraction, and mol cations if
 524 necessary.

525 **Class structure** `BatchFile(filename, sheet_name=0, file_type='excel', units='wtpt_oxides',`
 526 `label='Label', default_normalization='none', default_units='wtpt_oxides',`
 527 `dataframe=None)`

528 **Required inputs** `filename`: A file name must be passed in quotes. This file must be
 529 in the same folder as the notebook or script that is calling it. This imports the
 530 data from the file name given and saves it to a variable of your choosing.

531 **Optional inputs** **By default, the BatchFile class assumes that loaded data is in units of wt%; altern**

532 `sheet_name`: If importing data from an Excel file, this argument is used to spec-
 533 ify which sheet to import. Only one sheet can be imported to a single BatchFile
 534 object. The default is '0', which imports the first sheet in the file, regardless of
 535 its name.

536 `file_type`: Specifies whether the file being imported is an Excel or CSV file. This
 537 argument is never strictly necessary, as `BatchFile()` will automatically detect whether
 538 an imported file is Excel or CSV if the file extension is one of .xls or .xlsx (Ex-
 539 cel) or .csv (CSV).

540 `units`: The units in which data are input. The default value is 'wtpt_oxides'
 541 for data as wt% oxides. The user can pass 'mol_oxides' for data in mol fraction
 542 oxides or 'mol_cations' for data in mol fraction cations.

543 `default_normalization`: The type of normalization to apply to the data by de-
 544 fault. One of: None, 'standard', 'fixedvolatiles', or 'additionalvolatiles'.
 545 These normalization types are described in the section on normalization below.

546 `default_units`: The type of composition to return by default, one of: 'wtpt_oxides'
 547 (wt% oxides, default), 'mol_oxides' (mol fraction oxides), or 'mol_cations' (mol
 548 fraction cations).

549 `label`: This is optional but can be specified if the column title referring to sam-
 550 ple names is anything other than "Label". The default value is "Label". The de-
 551 fault value is "Label". If no "Label" column is present and the label argument is
 552 not specified, the first column whose first row is not one of VESICAL's recognized
 553 oxides will be set as the index column. The index column will be used to select
 554 samples by name.

555 `dataframe`: This argument is used for transforming a pandas DataFrame object
 556 into a VESICAL BatchFile object. For convenience, this functionality is also defined
 557 as a separate function `BatchFile_from_DataFrame(dataframe, units='wtpt_oxides',`
 558 `label='Label')`.

559 **Outputs** A special type of python object defined in the VESICAL code known as an Batch-
 560 File object.

```
myfile = v.BatchFile('Supplement/Datasets/example_data.xlsx')
```

561 Once the BatchFile object is created and assigned to a variable, the user can then
 562 access the data loaded from their file as `variable.get_data()`. In this example, the vari-
 563 able corresponding to the BatchFile object is named `myfile` and so the data in that
 564 file can be accessed with `myfile.get_data()`. Below, `myfile.get_data()` is saved to
 565 a variable we name `data`. The variable `data` is a pandas DataFrame object, which makes
 566 displaying the data itself quite simple and aesthetically pleasing, since pandas DataFrames
 567 mimic spreadsheets.

568 Usage of `get_data()` allows the user to retrieve the data as originally entered or
 569 in any units and with any normalization supported by VESICAL.

570 **Method structure** `get_data(self, normalization=None, units=None, asBatchFile=False)`

571 **Optional inputs** `normalization` or `units` may be passed, with options as defined in
 572 the description of BatchFile above.

573 `asBatchFile` Default is False. If True, will return a VESICAL BatchFile object.

574 **Outputs** A pandas dataframe or BatchFile object with all user data.

575 [H] [User input data: Compositions, pressures, and temperatures for several silicate
 576 melts as supplied in the file `example_data.xlsx`

```
data = myfile.get_data()
data
```

SiO2	TiO2	Al2O3	Fe2O3	Cr2O3	\		
Kil3 -6_1a			48.249207	2.222114	11.692194	0.00	0.0
Kil3 -6_3a			48.295691	2.165357	11.755584	0.00	0.0
Kil3 -6_4a			49.124079	2.360984	12.172833	0.00	0.0
10*			47.960000	0.780000	18.770000	0.00	0.0
19*			49.640000	0.710000	18.050000	0.00	0.0
25			50.320000	0.720000	18.030000	0.00	0.0
SAT -M12 -1			62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -2			62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -4			62.600000	0.630000	17.300000	2.01	0.0
samp. P1968a			76.974880	0.085516	3.110636	0.00	0.0
samp. P1968b			76.943845	0.133125	3.169657	0.00	0.0
samp. P1968c			77.187205	0.119506	3.167827	0.00	0.0
samp. HPR3 -1_XL -3			75.413966	0.095164	14.077692	0.00	0.0
samp. HPR3 -1_XL -4_INCL -1			76.613586	0.095843	13.476762	0.00	0.0
AW -6			48.030000	2.840000	18.120000	0.00	0.0
AW -46			52.980000	2.180000	20.490000	0.00	0.0
KI -07			44.610000	4.370000	14.410000	0.00	0.0
			FeO	MnO	MgO	NiO	CoO
			↔ CaO	\			
Kil3 -6_1a			0.000000	0.079999	14.183817	0.0	0.0
↔ 9.892732							
Kil3 -6_3a			0.000000	0.084045	13.403980	0.0	0.0
↔ 10.052578							
Kil3 -6_4a			0.000000	0.098809	11.997699	0.0	0.0
↔ 10.308188							
10*			10.920000	0.150000	6.860000	0.0	0.0
↔ 12.230000							

Kil3 -6_3a	Tucker et al. (2019)	Basalt
Kil3 -6_4a	Tucker et al. (2019)	Basalt
10*	Roggensack (2001)	Basalt
19*	Roggensack (2001)	Basalt
25	Roggensack (2001)	Basalt
SAT -M12 -1	Moore et al. (1998)	Andesite
SAT -M12 -2	Moore et al. (1998)	Andesite
SAT -M12 -4	Moore et al. (1998)	Andesite
samp. P1968a	Myers et al. (2019)	Rhyolite
samp. P1968b	Myers et al. (2019)	Rhyolite
samp. P1968c	Myers et al. (2019)	Rhyolite
samp. HPR3 -1_XL -3	Mercer et al. (2015)	Rhyolite
samp. HPR3 -1_XL -4_INCL -1	Mercer et al. (2015)	Rhyolite
AW -6	Iacovino et al. (2016)	Phonotephrite
AW -46	Iacovino et al. (2016)	Basaltic -Trachyandesite
KI -07	Iacovino et al. (2016)	Basanite

	Press	Temp
Kil3 -6_1a	62.5	1299.094712
Kil3 -6_3a	128.0	1283.419991
Kil3 -6_4a	100.0	1255.153759
10*	2000.0	1200.000000
19*	2000.0	1200.000000
25	2000.0	1200.000000
SAT -M12 -1	703.0	1100.000000
SAT -M12 -2	1865.0	1100.000000
SAT -M12 -4	2985.0	1050.000000
samp. P1968a	300.0	900.000000
samp. P1968b	300.0	900.000000
samp. P1968c	300.0	900.000000
samp. HPR3 -1_XL -3	300.0	0.000000
samp. HPR3 -1_XL -4_INCL -1	0.0	900.000000
AW -6	1500.0	1050.000000
AW -46	4000.0	1000.000000
KI -07	1500.0	1100.000000

577 For the rest of this manuscript, data will be pulled from the `example_data.xlsx`
578 file (Supplemental Dataset S1), which contains compositional information for basalts Tucker
579 et al. (2019); Roggensack (2001), andesites Moore et al. (1998), rhyolites Mercer et al.
580 (2015); Myers et al. (2019), and alkaline melts (phonotephrite, basaltic-trachyandesite,
581 and basanite from Iacovino et al. (2016)). Several additional example datasets from the
582 literature are available in the Supplement (Supplementary Datasets S2-S5; Table 0.3.3.1).
583 These include experimentally produced alkaline magmas from Iacovino et al. (2016), basaltic
584 melt inclusions from Kilauea Tucker et al. (2019) and Gakkal Ridge Bennett et al. (2019),
585 basaltic melt inclusions from Cerro Negro volcano, Nicaragua Roggensack (2001), and
586 rhyolite melt inclusions from the Taupo Volcanic Center, New Zealand Myers et al. (2019)
587 and a topaz rhyolite from the Rio Grande Rift Mercer et al. (2015). Where available,
588 the calibration datasets for VESICAL models are also provided (Supplementary Datasets
589 S6-S7).

590 [H] [] Example datasets included with VESICAL

```
pd.read_excel("tables/Table_Example_Data.xlsx", index_col="Filename")
```

```
Explanation \
Filename
```

example_data.xlsx	Example data used in this manuscript
alkaline.xlsx	Experimental glasses
basalts.xlsx	Melt inclusion glasses
cerro_negro.xlsx	Melt inclusion glasses
rhyolites.xlsx	Melt inclusion glasses

Compositions \

Filename	
example_data.xlsx	Wide comp. range
alkaline.xlsx	Basanite to Tephriphonolite
basalts.xlsx	Basaltic
cerro_negro.xlsx	Basaltic
rhyolites.xlsx	Rhyolitic

Filename	
example_data.xlsx	Iacovino et al. (2016); Mercer et al. (2015); Myers et al. (2019); Roggensack (2001); Tucker et al. (2019)
alkaline.xlsx	Iacovino et al. (2016)
basalts.xlsx	Tucker et al. (2019); Bennett et al. (2019)
cerro_negro.xlsx	Roggensack (2001)
rhyolites.xlsx	Mercer et al. (2015); Myers et al. (2019)

591 *0.3.3.2 Defining a single sample* More advanced functionality of VESICAL is fa-
 592 cilitated directly through the five core calculation classes. Each calculation requires its
 593 own unique inputs, but all calculations require that a sample composition be passed. We
 594 can pass in a sample either as a python dictionary or pandas Series. Below, we define
 595 a sample and name it `mysample`. Oxides are given in wt%. Only the oxides shown here
 596 can be used, but not all oxides are required. Any extra oxides (or other information not
 597 in the oxide list) the user defines will be ignored during calculations.

598 Much like is done to create a BatchFile object, we can create a VESICAL Sample
 599 object to represent our sample composition.

600 **Class structure** `Sample(composition, units='wtpt_oxides', default_normalization='none',`
 601 `default_units='wtpt_oxides')`

602 **Required inputs** `composition`: The composition of the sample in the format spec-
 603 ified by the `units` parameter. The default is oxides in wt%.

604 **Optional inputs** `units`, `default_normalization`, and `default_units` have the same
 605 meaning here as in the BatchFile class described above.

606 **Outputs** A special type of python object defined in the VESICAL code known as a Sam-
 607 ple object.

608 To manually input a bulk composition, fill in the oxides in wt% below:

```
mysample = v.Sample({'SiO2': 77.3,
                    'TiO2': 0.08,
                    'Al2O3': 12.6,
                    'Fe2O3': 0.207,
                    'Cr2O3': 0.0,
                    'FeO': 0.473,
                    'MnO': 0.0,
                    'MgO': 0.03,
                    'NiO': 0.0,
```

```
'CoO' : 0.0,
'CaO' : 0.43,
'Na2O' : 3.98,
'K2O' : 4.88,
'P2O5' : 0.0,
'H2O' : 6.5,
'CO2' : 0.05})
```

609 To see the composition of `mysample`, use the `get_composition(species=None,`
610 `normalization=None, units=None, exclude_volatiles=False, asSampleClass=False)`
611 method. By default, the composition is returned exactly as input above. `species` can
612 be set as an element or oxide (e.g., “Si” or “SiO₂”) to return the float value for only that
613 species. The composition can automatically be normalized using any of the standard nor-
614 malization functions listed above and can be returned in any of the units discussed above.
615 As with the `BatchFile.get_data()` function, a sample composition can be returned as
616 a dictionary (default) or as a VESICAL Sample object (if `asSampleClass` is set to `True`).

```
mysample.get_composition()
```

```
SiO2      77.300
TiO2      0.080
Al2O3     12.600
Fe2O3     0.207
Cr2O3     0.000
FeO       0.473
MnO       0.000
MgO       0.030
NiO       0.000
CoO       0.000
CaO       0.430
Na2O      3.980
K2O       4.880
P2O5      0.000
H2O       6.500
CO2       0.050
dtype: float64
```

617 The oxides considered by VESICAL are:

```
print(v.oxides)
```

```
['SiO2', 'TiO2', 'Al2O3', 'Fe2O3', 'Cr2O3', 'FeO', 'MnO', 'MgO', 'NiO',
 → 'CoO', 'CaO', 'Na2O', 'K2O', 'P2O5', 'H2O', 'CO2']
```

618 *0.3.3.3 Extracting a single sample from a batch file* Defined within the `BatchFile()`
619 class, the method `get_sample_composition()` allows for the extraction of a melt com-
620 position from a loaded Excel or CSV file.

621 **Method structure** `myfile.get_sample_composition(samplename, species=None,`
622 `normalization=None, units=None, asSampleClass=False)`

623 **Required inputs** `samplename`: The name of the sample, as a string, as defined in the
624 ‘Label’ column of the input file.

625 **Optional inputs** `species`: This is used if only the concentration of a single species (ei-
626 ther oxide or element) is desired.

627 **normalization:** This is optional and determines the style of normalization per-
 628 formed on a sample. The default value is `None`, which returns the value-for-value
 629 un-normalized composition. Other normalization options are described in the Batch-
 630 File class description above.
 631 **units:** The default is `wt% oxides`. Other options are described in the BatchFile
 632 class description above.
 633 **asSampleClass:** Can be `True` or `False` (default). If set to `False`, this will return
 634 a dictionary with compositional values. If set to `True`, this will return a `Sample`
 635 object with compositional data stored within.
 636 **Outputs** The bulk composition stored in a dictionary or `Sample` object.

```
"""To get composition from a specific sample in the input data:"""
sample_10 = myfile.get_sample_composition('10*', asSampleClass=True)
```

```
"""To see the extracted sample composition, uncomment the line below by
→ removing the # and execute this code cell"""
#sample_10.get_composition()
```

```
'To see the extracted sample composition, uncomment the line below by
→ removing the # and execute this code cell'
```

637 *0.3.3.4 Normalizing and transforming data* Before performing model calcula-
 638 tions on your data, it may be desired to normalize the input composition to a total of
 639 100 wt%. For a user to decide whether normalization is prudent, is important to under-
 640 stand the influence any normalization, or lack thereof, to a composition will have on mod-
 641 eling results. Electron microprobe analyses of major elements in silicate glasses combined
 642 with volatile element analyses by SIMS and FTIR often sum to less than 100 wt%. This
 643 deficiency is normally attributed to subsurface charging, matrix corrections, and unknown
 644 redox states of Fe and S during analyses by electron microprobe Hughes et al. (2019).
 645 As an example, when normalized, a volatile-free basalt with a measured SiO_2 content
 646 of 46 wt% and an analytical total of 97 wt% actually contains 47.4 wt% SiO_2 (46/0.97;
 647 a 3% relative change in silica content). Many studies report major element data normal-
 648 ized to 100% with volatiles listed separately. The result is that, value for value, litera-
 649 ture datasets can have totals several wt% less than 100 (if raw data are reported) or sev-
 650 eral wt% higher than 100 (if major elements are normalized anhydrous).

651 To deal with this variation, VESICAL provides users with four options for normal-
 652 ization. Normalization types are: - `None` (no normalization) - `'standard'`: Normalizes an
 653 input composition to 100%. - `'fixedvolatiles'`: Normalizes major element oxides to 100
 654 wt%, including volatiles. The volatile wt% will remain fixed, whilst the other major el-
 655 ement oxides are reduced proportionally so that the total is 100 wt%. - `'additionalvolatiles'`:
 656 Normalizes major element oxide wt% to 100%, assuming it is volatile-free. If H_2O or CO_2
 657 are passed to the function, their un-normalized values will be retained in addition to the
 658 normalized non-volatile oxides, summing to >100%.

659 Normalization can be performed on a `Sample` object or on all samples within a Batch-
 660 File object using the `get_composition()` or `get_data()` methods (e.g., `myfile.get_composition(normalization=`
 661 `or mysample.get_composition(normalization='additionalvolatiles')). Note that,
 662 since a BatchFile object may have other data in addition to sample compositions (e.g.,
 663 information on pressure, temperature, other user notes), BatchFile.get_composition()
 664 returns only compositional data, where as BatchFile.get_data() returns all data stored
 665 in the BatchFile object. The normalization argument can be passed to either. In the
 666 example below, we obtain the standard normalization of mysample and myfile and save
 667 these to new Sample and BatchFile objects called mysample_normalized and myfile_normalized.
 668 Note that asSampleClass or asBatchFile must be set to True in order to return a Sam-
 669 ple or BatchFile object. Without this argument, a dictionary or pandas DataFrame will`

670 be returned and new Sample or BatchFile objects will need to be constructed from those
 671 in order to perform calculations on the normalized datasets.

```

"""Retrieve the standard normalization for one sample"""
mysample_normalized = mysample.get_composition(normalization="standard",
↪ asSampleClass=True)
#print(mysample_normalized.get_composition())

"""Retrieve the standard normalization for all samples in a BatchFile"""
myfile_normalized = myfile.get_data(normalization="standard",
↪ asBatchFile=True)
#print(myfile_normalized.get_data())

```

672 The Liu and all six AllisonCarbon models are not sensitive to normalization be-
 673 cause they contain no compositional terms. Similarly, the expressions for Shishkina and
 674 MooreWater contain compositional terms expressed solely in terms of anhydrous cation
 675 fractions; the `additionalvolatiles` and `fixedvolatiles` normalization routines do not
 676 affect the relative abundances of major elements (and therefor anhydrous cation frac-
 677 tions). Thus, Shishkina and MooreWater are only affected by the standard normaliza-
 678 tion routine. In contrast, the Dixon model is highly sensitive to the choice of normal-
 679 ization because its compositional term for both H₂O and CO₂ is expressed solely in terms
 680 of the absolute melt SiO₂ content.

681 The expressions of Iacono-Marziano are parameterized in terms of hydrous cation
 682 fractions and NBO/O, and so this model is sensitive to `additionalvolatiles` or `fixedvolatiles`
 683 normalization routines, which will change the relative proportions of volatiles to major
 684 elements. Even so, the effect of normalization on volatile solubility calculations is rel-
 685 atively small and of similar magnitude to the discrepancy between the hydrous total and
 686 100 for the hydrous model. Thus, the choice of normalization is only important when
 687 data has hydrous totals that differ significantly from 100%. The Iacono-Marziano web
 688 app normalizes input data a la VESICAL's `additionalvolatiles` normalization routine. For
 689 consistency with the web app, VESICAL automatically uses the `additionalvolatiles` nor-
 690 malization during calculations with this model.

691 The implementation of MagmaSat in VESICAL is sensitive to the relative propor-
 692 tion of major and volatile element components rather than the absolute concentrations
 693 entered (as with the whole MELTS family of models). Thus, calculations using raw, fixed-
 694 and `additionalvolatile` routines yield different results. If the hydrous total of an input
 695 composition is less than 100%, the `fixedvolatile` routine effectively reduces the relative
 696 proportion of volatiles to major elements, so calculated saturation pressures go down.
 697 Conversely, if inputs have high hydrous totals, the `fixedvolatile` routine increases the rel-
 698 ative proportion of volatiles in the system, so the saturation pressure goes up. As with
 699 Iacono-Marziano, the percent discrepancy between calculations for different normaliza-
 700 tion routines is similar to the difference between the total and 100%. For saturation pres-
 701 sure calculations, the MagmaSat app automatically normalizes input data in a manner
 702 identical to VESICAL's `fixedvolatiles` routine, and so this normalization is forced on all
 703 samples for such calculations with MagmaSat in VESICAL. Further discussion on the ef-
 704 fect of normalization in MagmaSat is provided in Supplementary Text S5 (and Supple-
 705 mentary Figs S22-S26).

706 For example, consider a basalt with a measured SiO₂ content of 47.4 wt%, 1000
 707 ppm dissolved CO₂, and an anhydrous (volatile-free) total of 96.77 wt%:

```

mybasalt = v.Sample({'SiO2': 47,
                    'TiO2': 1.01,
                    'Al2O3': 17.46,

```

```
'Fe2O3': 0.89,
'FeO': 7.18,
'MgO': 7.63,
'CaO': 12.44,
'Na2O': 2.65,
'K2O': 0.03,
'P2O5': 0.08,
'CO2': 0.1})
```

708 We can apply each normalization routine to this sample and examine how this will
709 affect the saturation pressure predicted by each model:

```
"""Normalize three ways"""
mybasalt_std = mybasalt.get_composition(normalization="standard",
→ asSampleClass=True)
mybasalt_add =
→ mybasalt.get_composition(normalization="additionalvolatiles",
→ asSampleClass=True)
mybasalt_fix = mybasalt.get_composition(normalization="fixedvolatiles",
→ asSampleClass=True)

"""Choose a model to test"""
mymodel = "IaconoMarziano"

for basalt, normtype in zip([mybasalt, mybasalt_std, mybasalt_add,
→ mybasalt_fix],
→ ["Raw", "standard", "additionalvolatiles",
→ "fixedvolatiles"]):
print(str(normtype) +
" Saturation Pressure = " +
str(v.calculate_saturation_pressure(sample=basalt,
→ temperature=1200, model=mymodel).result) +
" bars")
```

```
Raw Saturation Pressure = 1848.0344238069026 bars
standard Saturation Pressure = 1906.5480423917331 bars
additionalvolatiles Saturation Pressure = 1848.2699609182075 bars
fixedvolatiles Saturation Pressure = 1848.263700904032 bars
```

710 Because the compositional effect on H₂O solubility is smaller, so are the changes
711 in calculated saturation pressures for a pure-H₂O system, but they can still be signif-
712 icant for H₂O-rich liquids (where high H₂O contents can change totals and therefor SiO₂
713 contents more dramatically).

714 **0.3.4 Comparing User Data to Model Calibrations: Which Model Should** 715 **I Use?**

716 MagmaSat is the most thermodynamically robust model implemented in VESICAL,
717 and thus it is the most generally appropriate model to use (n.b. that it is also the most
718 computationally expensive). However, one of the strengths of VESICAL is its ability to
719 utilize up to seven different solubility models. Each of these models is based on its own
720 calibration dataset, meaning the pressure-temperature-composition space over which mod-
721 els are calibrated is quite variable from model to model. The individual model calibra-
722 tions are discussed in detail in this manuscript's companion paper Wieser et al. (2021).

723
724
725
726
727
728
729
730
731

For the remainder of this section, all example calculations are carried out with MagmaSat, the default model of VESICAL. To use any other VESICAL model, simply add `model=` and the name of the desired model in quotes to any calculation (e.g., `v.calculate_dissolved_volatiles(temperature=1000, model="Dixon")`). The model names recognized by VESICAL are: MagmaSat, ShishkinaIdealMixing, Dixon, IaconoMarziano, Liu, AllisonCarbon, MooreWater. For more advanced use cases such as hybridizing models (see Section 0.3.11), pure-H₂O and pure-CO₂ models from within a mixed-fluid model can be used by adding ‘Water’ or ‘Carbon’ to the model name (e.g., DixonCarbon; note that MagmaSat does not have this functionality).

732
733
734
735
736
737
738
739

Determination of the appropriate model to use with any sample is crucial to the correct application of these models, and so we stress the importance of understanding how a model’s calibration space relates to the sample at hand. VESICAL includes some built-in functionality for comparing melt compositions from user loaded data to those in the datasets upon which each of the VESICAL models is calibrated using the method `calib_plot`. This can be visualized as a total alkalis vs silica (TAS) diagram (with fields and labels via the python `tasplot` library by J. Stevenson; <https://bitbucket.org/jstevenson5/tasplot/src/master/>; Figure 0.3.4 a) or as any x-y plot in which x and y are oxides (Figure 0.3.4 b).

740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766

Method structure `calib_plot(user_data=None, model='all', plot_type='TAS', zoom=None, save_fig=False)`

Optional inputs `user_data`: The default value is None, in which case only the model calibration set is plotted. User provided sample data describing the oxide composition of one or more samples. Multiple samples can be passed as an `BatchFile` object or pandas `DataFrame`. A single sample can be passed as a pandas `Series`. `model`: The default value is ‘all’, in which case all model calibration datasets will be plotted. Otherwise, any model can be plotted by passing the name of the model desired (e.g., ‘Liu’). Multiple models can be plotted by passing them as strings within a list (e.g., [‘Liu’, ‘Dixon’])

`plot_type`: The default value is ‘TAS’, which returns a total alkalis vs silica (TAS) diagram. Any two oxides can be plotted as an x-y plot by setting `plot_type=‘xy’` and specifying x- and y-axis oxides, e.g., `x=‘SiO2’, y=‘Al2O3’`.

`zoom`: The default is None in which case axes will be set to the default of $35 \leq x \leq 100$ wt% and $0 \leq y \leq 25$ wt% for TAS type plots and the best values to show the data for xy type plots. The user can pass “`user_data`” to plot the figure where the x and y axes are scaled down to zoom in and only show the region surrounding the `user_data`. A list of tuples may be passed to manually specify x and y limits. Pass in data as [(`x_min`, `x_max`), (`y_min`, `y_max`)]. For example, the default limits here would be passed in as [(35,100), (0,25)].

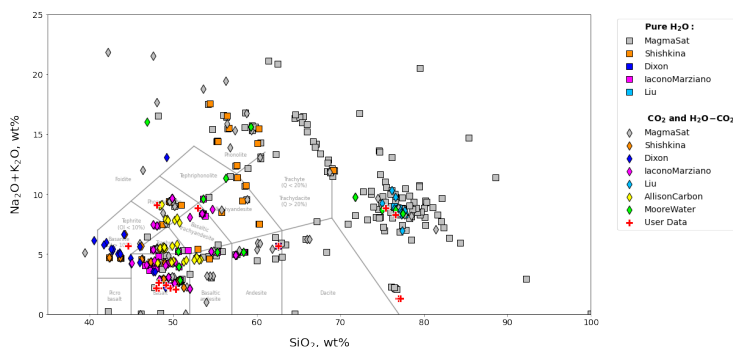
`save_fig`: The default value is False, in which case the plot will be generated and displayed but not saved. If the user wishes to save the figure, the desired filename (including the file extension, e.g., .png) can be passed here. Note that all plots in this Jupyter notebook can be saved by right clicking the plot and choosing “Save Image As...”.

Outputs The function returns fig and axes matplotlib objects defining a TAS or x-y plot of user data and model calibration data.

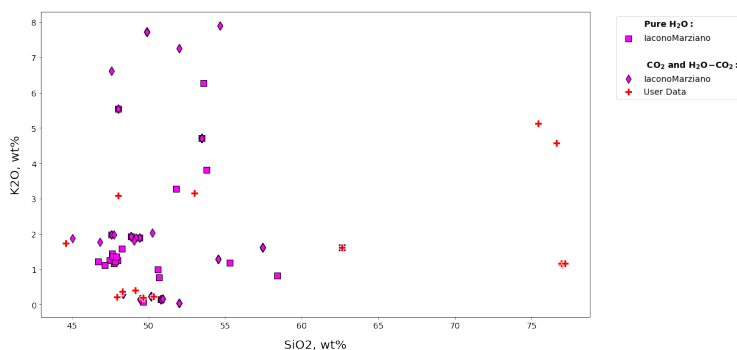
767

[H]

```
v.calib_plot(user_data=myfile)
v.calib_plot(user_data=myfile, model='IaconoMarziano', plot_type='xy',
  ↪ x='SiO2', y='K2O', save_fig=False)
v.show()
```



768



769

770 `[]`Example calibration
 771 plots. a. The default plot with `user_data` defined as `myfile` and no other options set. This
 772 produces a TAS diagram with the user data plotted atop data from calibration datasets
 773 for all models. b. A plot with all options specified. This example produces an x-y plot
 774 for `user_data` (`myfile`) and the Iacono-Marziano calibration dataset where x and y are
 775 SiO₂ and K₂O concentration in wt%. Symbol shapes correspond to the volatile compo-
 sition of experiments used to calibrate the model.

776 Using the functionality built into python and the matplotlib library, user data can
 777 be plotted on its own at any time, including before any calculations are performed. Al-
 778 most any plot type imaginable can be produced, and users should refer to the matplotlib
 779 documentation (<https://matplotlib.org/3.2.1/index.html>) if more complex plotting is de-
 780 sired.

781 0.3.5 Calculating dissolved volatile concentrations

782 The `calculate_dissolved_volatiles()` function calculates the concentration of
 783 dissolved H₂O and CO₂ in the melt at a given pressure-temperature condition and with
 784 a given H₂O-CO₂ fluid composition, defined as the mole fraction of H₂O in an H₂O-CO₂
 785 fluid ($X_{H_2O}^{fluid}$). The default MagmaSat model relies on the underlying functionality
 786 of MELTS, whose basic function is to calculate the equilibrium phase assemblage given
 787 the bulk composition of the system and pressure-temperature conditions. To calculate
 788 dissolved volatile concentrations thus requires computing the equilibrium state of a sys-
 789 tem at fixed pressure and temperature over a range of bulk volatile concentrations un-
 790 til a solution is found that satisfies the user defined fluid composition.

791 First, the function makes an initial guess at the appropriate bulk volatile concen-
 792 trations by finding the minimum dissolved volatile concentrations in the melt at satu-
 793 ration, while asserting that the weight fraction of H₂O/(H₂O+CO₂) in the system is equal
 794 to the user input mole fraction of H₂O/(H₂O+CO₂) in the fluid. This is done by increas-
 795 ing the H₂O and CO₂ concentrations appropriately until a fluid phase is stable. Once
 796 fluid saturation is determined, the code then performs directional, iterative, and progres-

797 sively more refined searches, increasing the proportion of H₂O or CO₂ in the system if
 798 the mole fraction of H₂O calculated in the fluid is greater than or less than that defined
 799 by the user, respectively. Four iterative searches are performed; the precision of the match
 800 between the calculated and defined XH₂O^{fluid} increases from 0.1 in the first iteration to
 801 0.01, 0.001, and finally to 0.0001. Thus, the calculated dissolved volatile concentrations
 802 correspond to a system with XH₂O^{fluid} within 0.0001 of the user defined value.

803 For non-MagmaSat models, dissolved volatile concentrations are calculated directly
 804 from model equations.

805 **Method structure** Single sample: `calculate_dissolved_volatiles(sample, temperature,`
 806 `pressure, X_fluid=1, verbose=False, model='MagmaSat').result`
 807 BatchFile batch process: `myfile.calculate_dissolved_volatiles(temperature,`
 808 `pressure, X_fluid=1, print_status=True, model='MagmaSat')`

809 **Standard inputs** `sample, temperature, pressure, X_fluid, model` (see Section 0.3.1).

810 **Unique optional inputs** `verbose`: *Only for single sample calculations.* Default value
 811 is False, in which case H₂O and CO₂ concentrations are returned. If set to True,
 812 additional parameters are returned in a dictionary: H₂O and CO₂ concentrations
 813 in the fluid in mole fraction, temperature, pressure, and proportion of the fluid
 814 in the system in wt%.

815 `print_status`: *Only for batch calculations.* The default value is True, in which
 816 case the progress of the calculation will be printed to the terminal. The user may
 817 desire to see the status of the calculation, as this particular function can be quite
 818 slow, averaging between 3-5 seconds per sample.

819 **Calculated outputs** If the single-sample method is used, a dictionary with keys 'H2O'
 820 and 'CO2' corresponding to the calculated dissolved H₂O and CO₂ concentrations
 821 in the melt is returned (plus additional variables 'temperature' in °C, 'pressure'
 822 in bars, 'XH2O_fl', 'XCO2_fl', and 'FluidProportion_wtper' (the proportion of
 823 the fluid in the system in wt%) if `verbose` is set to True).

824 If the BatchFile method is used, a pandas DataFrame is returned with sample in-
 825 formation plus calculated dissolved H₂O and CO₂ concentrations in the melt, the
 826 fluid composition in mole fraction, and the proportion of the fluid in the system
 827 in wt%. Pressure (in bars) and Temperature (in °C) columns are always returned.

828 *"""Calculate dissolved volatiles for sample 10*"""*

```
v.calculate_dissolved_volatiles(sample=sample_10, temperature=900.0,
↪ pressure=2000.0,
                                X_fluid=0.5, verbose=True).result
```

```
{'H2O_liq': 2.69352739399806,
 'CO2_liq': 0.0638439414375309,
 'XH2O_fl': 0.500092686493868,
 'XCO2_fl': 0.499907313506132,
 'FluidProportion_wt': 0.18407321260435108}
```

829 *"""Calculate dissolved for all samples in an BatchFile object"""*

```
dissolved = myfile.calculate_dissolved_volatiles(temperature=900.0,
↪ pressure=2000.0, X_fluid=1, print_status=True)
```

830 [H] [Modeled dissolved volatile concentrations

dissolved

```
SiO2      TiO2      Al2O3  Fe2O3  Cr2O3  \
Kil3 -6_1a      48.249207  2.222114  11.692194  0.00  0.0
```

Kil3 -6_3a	48.295691	2.165357	11.755584	0.00	0.0
Kil3 -6_4a	49.124079	2.360984	12.172833	0.00	0.0
10*	47.960000	0.780000	18.770000	0.00	0.0
19*	49.640000	0.710000	18.050000	0.00	0.0
25	50.320000	0.720000	18.030000	0.00	0.0
SAT -M12 -1	62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -2	62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -4	62.600000	0.630000	17.300000	2.01	0.0
samp. P1968a	76.974880	0.085516	3.110636	0.00	0.0
samp. P1968b	76.943845	0.133125	3.169657	0.00	0.0
samp. P1968c	77.187205	0.119506	3.167827	0.00	0.0
samp. HPR3 -1_XL -3	75.413966	0.095164	14.077692	0.00	0.0
samp. HPR3 -1_XL -4_INCL -1	76.613586	0.095843	13.476762	0.00	0.0
AW -6	48.030000	2.840000	18.120000	0.00	0.0
AW -46	52.980000	2.180000	20.490000	0.00	0.0
KI -07	44.610000	4.370000	14.410000	0.00	0.0

	FeO	MnO	MgO	NiO	CoO	...	\
Kil3 -6_1a	0.000000	0.079999	14.183817	0.0	0.0	...	
Kil3 -6_3a	0.000000	0.084045	13.403980	0.0	0.0	...	
Kil3 -6_4a	0.000000	0.098809	11.997699	0.0	0.0	...	
10*	10.920000	0.150000	6.860000	0.0	0.0	...	
19*	10.540000	0.190000	6.430000	0.0	0.0	...	
25	10.110000	0.140000	5.650000	0.0	0.0	...	
SAT -M12 -1	2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -2	2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -4	2.010000	0.060000	2.650000	0.0	0.0	...	
samp. P1968a	4.788883	0.000000	12.549439	0.0	0.0	...	
samp. P1968b	4.763435	0.000000	12.446403	0.0	0.0	...	
samp. P1968c	4.814076	0.000000	12.229534	0.0	0.0	...	
samp. HPR3 -1_XL -3	0.654992	0.125882	0.012003	0.0	0.0	...	
samp. HPR3 -1_XL -4_INCL -1	0.620769	0.113495	0.032069	0.0	0.0	...	
AW -6	9.600000	0.230000	3.080000	0.0	0.0	...	
AW -46	5.540000	0.200000	2.000000	0.0	0.0	...	
KI -07	10.600000	0.170000	7.690000	0.0	0.0	...	

	ROCK TYPE	Press	Temp	\
Kil3 -6_1a	Basalt	62.5	1299.094712	
Kil3 -6_3a	Basalt	128.0	1283.419991	
Kil3 -6_4a	Basalt	100.0	1255.153759	
10*	Basalt	2000.0	1200.000000	
19*	Basalt	2000.0	1200.000000	
25	Basalt	2000.0	1200.000000	
SAT -M12 -1	Andesite	703.0	1100.000000	
SAT -M12 -2	Andesite	1865.0	1100.000000	
SAT -M12 -4	Andesite	2985.0	1050.000000	
samp. P1968a	Rhyolite	300.0	900.000000	
samp. P1968b	Rhyolite	300.0	900.000000	
samp. P1968c	Rhyolite	300.0	900.000000	
samp. HPR3 -1_XL -3	Rhyolite	300.0	0.000000	
samp. HPR3 -1_XL -4_INCL -1	Rhyolite	0.0	900.000000	
AW -6	Phonotephrite	1500.0	1050.000000	
AW -46	Basaltic -Trachyandesite	4000.0	1000.000000	
KI -07	Basanite	1500.0	1100.000000	

	H2O_liq_VESICAL	CO2_liq_VESICAL \
Kil3 -6_1a	5.256561	0.0
Kil3 -6_3a	5.417720	0.0
Kil3 -6_4a	5.353421	0.0
10*	4.984021	0.0
19*	5.134419	0.0
25	5.189068	0.0
SAT -M12 -1	5.810439	0.0
SAT -M12 -2	5.810439	0.0
SAT -M12 -4	5.810439	0.0
samp. P1968a	6.484749	0.0
samp. P1968b	6.473813	0.0
samp. P1968c	6.482109	0.0
samp. HPR3 -1_XL -3	6.097630	0.0
samp. HPR3 -1_XL -4_INCL -1	6.138658	0.0
AW -6	5.856636	0.0
AW -46	5.879457	0.0
KI -07	4.918430	0.0

	Temperature_C_VESICAL	Pressure_bars_VESICAL \
Kil3 -6_1a	900.0	2000.0
Kil3 -6_3a	900.0	2000.0
Kil3 -6_4a	900.0	2000.0
10*	900.0	2000.0
19*	900.0	2000.0
25	900.0	2000.0
SAT -M12 -1	900.0	2000.0
SAT -M12 -2	900.0	2000.0
SAT -M12 -4	900.0	2000.0
samp. P1968a	900.0	2000.0
samp. P1968b	900.0	2000.0
samp. P1968c	900.0	2000.0
samp. HPR3 -1_XL -3	900.0	2000.0
samp. HPR3 -1_XL -4_INCL -1	900.0	2000.0
AW -6	900.0	2000.0
AW -46	900.0	2000.0
KI -07	900.0	2000.0

	X_fluid_input_VESICAL	Model	Warnings
Kil3 -6_1a	1	MagmaSat	
Kil3 -6_3a	1	MagmaSat	
Kil3 -6_4a	1	MagmaSat	
10*	1	MagmaSat	
19*	1	MagmaSat	
25	1	MagmaSat	
SAT -M12 -1	1	MagmaSat	
SAT -M12 -2	1	MagmaSat	
SAT -M12 -4	1	MagmaSat	
samp. P1968a	1	MagmaSat	
samp. P1968b	1	MagmaSat	
samp. P1968c	1	MagmaSat	
samp. HPR3 -1_XL -3	1	MagmaSat	
samp. HPR3 -1_XL -4_INCL -1	1	MagmaSat	
AW -6	1	MagmaSat	
AW -46	1	MagmaSat	

KI -07

1

MagmaSat

[17 rows x 27 columns]

829

0.3.6 Calculating equilibrium fluid compositions

830 The `calculate_equilibrium_fluid_comp()` function calculates the composition
 831 of a fluid phase in equilibrium with a given silicate melt with known pressure, temper-
 832 ature, and dissolved H₂O and CO₂ concentrations. The calculation is performed sim-
 833 ply by calculating the equilibrium state of the given sample at the given conditions and
 834 determining if that melt is fluid saturated. If the melt is saturated, fluid composition and
 835 mass are reported back. If the calculation finds that the melt is not saturated at the given
 836 pressure and temperature, values of 0.0 will be returned for the H₂O and CO₂ concen-
 837 trations in the fluid.

838 **Method structure** Single sample: `calculate_equilibrium_fluid_comp(sample, temperature,`
 839 `pressure, verbose=False, model='MagmaSat').result`
 840 BatchFile batch process: `myfile.calculate_equilibrium_fluid_comp(temperature,`
 841 `pressure=None, print_status=False, model='MagmaSat')`

842 **Standard inputs** `sample, temperature, pressure, model` (see Section 0.3.1).

843 **Unique optional inputs** `verbose`: *Only for single sample calculations.* Default value
 844 is False, in which case H₂O and CO₂ concentrations in the fluid in mol fraction
 845 are returned. If set to True, additional parameters are returned in a dictionary:
 846 H₂O and CO₂ concentrations in the fluid, mass of the fluid in grams, and propor-
 847 tion of the fluid in the system in wt%.

848 `print_status`: *Only for batch calculations.* The default value is False. If True is
 849 passed, the progress of the calculation will be printed to the terminal.

850 **Calculated outputs** If the single-sample method is used, a dictionary with keys 'H2O'
 851 and 'CO2' is returned (plus additional variables 'FluidMass_grams' and 'Fluid-
 852 Proportion_wtper' if `verbose` is set to True).
 853 If the BatchFile method is used, a pandas DataFrame is returned with sample in-
 854 formation plus calculated equilibrium fluid compositions, mass of the fluid in grams,
 855 and proportion of the fluid in the system in wt%. Pressure (in bars) and Temper-
 856 ature (in °C) columns are always returned.

```
857 """Calculate fluid composition for the extracted sample"""
```

```
v.calculate_equilibrium_fluid_comp(sample=sample_10, temperature=900.0,  
858 → pressure=100.0).result
```

```
{'CO2': 0.00528661429366132, 'H2O': 0.994713385706339}
```

857 Below we calculate equilibrium fluid compositions for all samples at a single tem-
 858 perature of 900 °C and a single pressure of 1,000 bars. Note that some samples in this
 859 dataset have quite low volatile concentrations (e.g., the Tucker et al. (2019) basalts from
 860 Kilauea), and so are below saturation at this P-T condition. The fluid composition for
 861 undersaturated samples is returned as values of 0 for both H₂O and CO₂.

862 [H] [] Isothermally modeled equilibrium fluid compositions

```
"""Calculate fluid composition for all samples in an BatchFile object"""
```

```
eqfluid = myfile.calculate_equilibrium_fluid_comp(temperature=900.0,  
863 → pressure=1000.0)  
eqfluid
```

SiO2	TiO2	Al2O3	Fe2O3	Cr2O3	\				
Kil3 -6_1a			48.249207	2.222114	11.692194	0.00	0.0		
Kil3 -6_3a			48.295691	2.165357	11.755584	0.00	0.0		
Kil3 -6_4a			49.124079	2.360984	12.172833	0.00	0.0		
10*			47.960000	0.780000	18.770000	0.00	0.0		
19*			49.640000	0.710000	18.050000	0.00	0.0		
25			50.320000	0.720000	18.030000	0.00	0.0		
SAT -M12 -1			62.600000	0.630000	17.300000	2.01	0.0		
SAT -M12 -2			62.600000	0.630000	17.300000	2.01	0.0		
SAT -M12 -4			62.600000	0.630000	17.300000	2.01	0.0		
samp. P1968a			76.974880	0.085516	3.110636	0.00	0.0		
samp. P1968b			76.943845	0.133125	3.169657	0.00	0.0		
samp. P1968c			77.187205	0.119506	3.167827	0.00	0.0		
samp. HPR3 -1_XL -3			75.413966	0.095164	14.077692	0.00	0.0		
samp. HPR3 -1_XL -4_INCL -1			76.613586	0.095843	13.476762	0.00	0.0		
AW -6			48.030000	2.840000	18.120000	0.00	0.0		
AW -46			52.980000	2.180000	20.490000	0.00	0.0		
KI -07			44.610000	4.370000	14.410000	0.00	0.0		
			FeO	MnO	MgO	NiO	CoO	...	\
Kil3 -6_1a			0.000000	0.079999	14.183817	0.0	0.0	...	
Kil3 -6_3a			0.000000	0.084045	13.403980	0.0	0.0	...	
Kil3 -6_4a			0.000000	0.098809	11.997699	0.0	0.0	...	
10*			10.920000	0.150000	6.860000	0.0	0.0	...	
19*			10.540000	0.190000	6.430000	0.0	0.0	...	
25			10.110000	0.140000	5.650000	0.0	0.0	...	
SAT -M12 -1			2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -2			2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -4			2.010000	0.060000	2.650000	0.0	0.0	...	
samp. P1968a			4.788883	0.000000	12.549439	0.0	0.0	...	
samp. P1968b			4.763435	0.000000	12.446403	0.0	0.0	...	
samp. P1968c			4.814076	0.000000	12.229534	0.0	0.0	...	
samp. HPR3 -1_XL -3			0.654992	0.125882	0.012003	0.0	0.0	...	
samp. HPR3 -1_XL -4_INCL -1			0.620769	0.113495	0.032069	0.0	0.0	...	
AW -6			9.600000	0.230000	3.080000	0.0	0.0	...	
AW -46			5.540000	0.200000	2.000000	0.0	0.0	...	
KI -07			10.600000	0.170000	7.690000	0.0	0.0	...	
			CITATION		ROCK TYPE				
			→ \						
Kil3 -6_1a			Tucker et al. (2019)		Basalt				
Kil3 -6_3a			Tucker et al. (2019)		Basalt				
Kil3 -6_4a			Tucker et al. (2019)		Basalt				
10*			Roggensack (2001)		Basalt				
19*			Roggensack (2001)		Basalt				
25			Roggensack (2001)		Basalt				
SAT -M12 -1			Moore et al. (1998)		Andesite				
SAT -M12 -2			Moore et al. (1998)		Andesite				
SAT -M12 -4			Moore et al. (1998)		Andesite				
samp. P1968a			Myers et al. (2019)		Rhyolite				
samp. P1968b			Myers et al. (2019)		Rhyolite				
samp. P1968c			Myers et al. (2019)		Rhyolite				
samp. HPR3 -1_XL -3			Mercer et al. (2015)		Rhyolite				
samp. HPR3 -1_XL -4_INCL -1			Mercer et al. (2015)		Rhyolite				
AW -6			Iacovino et al. (2016)		Phonotephrite				

AW -46 Iacovino et al. (2016) Basaltic -Trachyandesite
 KI -07 Iacovino et al. (2016) Basanite

	Press	Temp	XH2O_fl_VESICAL \
Kil3 -6_1a	62.5	1299.094712	0.000000
Kil3 -6_3a	128.0	1283.419991	0.000000
Kil3 -6_4a	100.0	1255.153759	0.000000
10*	2000.0	1200.000000	0.984531
19*	2000.0	1200.000000	0.974997
25	2000.0	1200.000000	0.990107
SAT -M12 -1	703.0	1100.000000	1.000000
SAT -M12 -2	1865.0	1100.000000	1.000000
SAT -M12 -4	2985.0	1050.000000	1.000000
samp. P1968a	300.0	900.000000	0.977773
samp. P1968b	300.0	900.000000	0.996799
samp. P1968c	300.0	900.000000	0.997028
samp. HPR3 -1_XL -3	300.0	0.000000	0.997770
samp. HPR3 -1_XL -4_INCL -1	0.0	900.000000	0.997273
AW -6	1500.0	1050.000000	0.261572
AW -46	4000.0	1000.000000	0.897441
KI -07	1500.0	1100.000000	0.826014

	XCO2_fl_VESICAL	Temperature_C_VESICAL \
Kil3 -6_1a	0.000000	900.0
Kil3 -6_3a	0.000000	900.0
Kil3 -6_4a	0.000000	900.0
10*	0.015469	900.0
19*	0.025003	900.0
25	0.009893	900.0
SAT -M12 -1	0.000000	900.0
SAT -M12 -2	0.000000	900.0
SAT -M12 -4	0.000000	900.0
samp. P1968a	0.022227	900.0
samp. P1968b	0.003201	900.0
samp. P1968c	0.002972	900.0
samp. HPR3 -1_XL -3	0.002230	900.0
samp. HPR3 -1_XL -4_INCL -1	0.002727	900.0
AW -6	0.738428	900.0
AW -46	0.102559	900.0
KI -07	0.173986	900.0

	Pressure_bars_VESICAL	Model \
Kil3 -6_1a	1000.0	MagmaSat
Kil3 -6_3a	1000.0	MagmaSat
Kil3 -6_4a	1000.0	MagmaSat
10*	1000.0	MagmaSat
19*	1000.0	MagmaSat
25	1000.0	MagmaSat
SAT -M12 -1	1000.0	MagmaSat
SAT -M12 -2	1000.0	MagmaSat
SAT -M12 -4	1000.0	MagmaSat
samp. P1968a	1000.0	MagmaSat
samp. P1968b	1000.0	MagmaSat
samp. P1968c	1000.0	MagmaSat
samp. HPR3 -1_XL -3	1000.0	MagmaSat

```
samp. HPR3 -1_XL -4_INCL -1 1000.0          MagmaSat
AW -6                                     1000.0          MagmaSat
AW -46                                    1000.0          MagmaSat
KI -07                                    1000.0          MagmaSat
```

```

Warnings
Kil3 -6_1a          Sample not saturated at these conditions
Kil3 -6_3a          Sample not saturated at these conditions
Kil3 -6_4a          Sample not saturated at these conditions
10*
19*
25
SAT -M12 -1
SAT -M12 -2
SAT -M12 -4
samp. P1968a
samp. P1968b
samp. P1968c
samp. HPR3 -1_XL -3
samp. HPR3 -1_XL -4_INCL -1
AW -6
AW -46
KI -07
```

```
[17 rows x 26 columns]
```

863 Below, we calculate equilibrium fluid compositions for the same dataset using tem-
864 peratures and pressures as defined in the input data (Table 0.3.3.1). Note that Samples
865 “samp. HPR3-1_XL-3” and “samp. HPR3-1_XL-4_INCL-1” have a user-defined value
866 of 0.0 for temperature and pressure, respectively. VESICAL automatically skips the cal-
867 culation of equilibrium fluids for these samples and returns a warning to the user, which
868 are both printed to the terminal below and appended to the “Warnings” column in the
869 returned data.

870 [H] **Modeled equilibrium fluid compositions with unique temperatures.**
871 Warnings “Bad temperature” and “Bad pressure” indicate that no data (or 0.0 value
872 data) was given for the temperature or pressure of that sample, in which case the
873 calculation of that sample is skipped.

```
"""Calculate fluid composition for all samples with unique pressure and
→ temperature values for each sample.
Pressure and temperature values are taken from columns named "Press" and
→ "Temp" in the example BatchFile"""
```

```
eqfluid_wtemps =
→ myfile.calculate_equilibrium_fluid_comp(temperature='Temp',
→ pressure='Press')
eqfluid_wtemps
```

```
SiO2      TiO2      Al2O3  Fe2O3  Cr2O3  \
Kil3 -6_1a  48.249207  2.222114  11.692194  0.00  0.0
Kil3 -6_3a  48.295691  2.165357  11.755584  0.00  0.0
Kil3 -6_4a  49.124079  2.360984  12.172833  0.00  0.0
10*        47.960000  0.780000  18.770000  0.00  0.0
19*        49.640000  0.710000  18.050000  0.00  0.0
25         50.320000  0.720000  18.030000  0.00  0.0
```

SAT -M12 -1	62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -2	62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -4	62.600000	0.630000	17.300000	2.01	0.0
samp. P1968a	76.974880	0.085516	3.110636	0.00	0.0
samp. P1968b	76.943845	0.133125	3.169657	0.00	0.0
samp. P1968c	77.187205	0.119506	3.167827	0.00	0.0
samp. HPR3 -1_XL -3	75.413966	0.095164	14.077692	0.00	0.0
samp. HPR3 -1_XL -4_INCL -1	76.613586	0.095843	13.476762	0.00	0.0
AW -6	48.030000	2.840000	18.120000	0.00	0.0
AW -46	52.980000	2.180000	20.490000	0.00	0.0
KI -07	44.610000	4.370000	14.410000	0.00	0.0

	FeO	MnO	MgO	NiO	CoO	...	\
Kil3 -6_1a	0.000000	0.079999	14.183817	0.0	0.0	...	
Kil3 -6_3a	0.000000	0.084045	13.403980	0.0	0.0	...	
Kil3 -6_4a	0.000000	0.098809	11.997699	0.0	0.0	...	
10*	10.920000	0.150000	6.860000	0.0	0.0	...	
19*	10.540000	0.190000	6.430000	0.0	0.0	...	
25	10.110000	0.140000	5.650000	0.0	0.0	...	
SAT -M12 -1	2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -2	2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -4	2.010000	0.060000	2.650000	0.0	0.0	...	
samp. P1968a	4.788883	0.000000	12.549439	0.0	0.0	...	
samp. P1968b	4.763435	0.000000	12.446403	0.0	0.0	...	
samp. P1968c	4.814076	0.000000	12.229534	0.0	0.0	...	
samp. HPR3 -1_XL -3	0.654992	0.125882	0.012003	0.0	0.0	...	
samp. HPR3 -1_XL -4_INCL -1	0.620769	0.113495	0.032069	0.0	0.0	...	
AW -6	9.600000	0.230000	3.080000	0.0	0.0	...	
AW -46	5.540000	0.200000	2.000000	0.0	0.0	...	
KI -07	10.600000	0.170000	7.690000	0.0	0.0	...	

	H2O	CO2	CITATION	\
Kil3 -6_1a	0.424695	0.002873	Tucker et al. (2019)	
Kil3 -6_3a	0.425984	0.006786	Tucker et al. (2019)	
Kil3 -6_4a	0.437758	0.004984	Tucker et al. (2019)	
10*	4.500000	0.047900	Roggensack (2001)	
19*	5.100000	0.111300	Roggensack (2001)	
25	5.200000	0.043700	Roggensack (2001)	
SAT -M12 -1	2.620000	0.000000	Moore et al. (1998)	
SAT -M12 -2	5.030000	0.000000	Moore et al. (1998)	
SAT -M12 -4	6.760000	0.000000	Moore et al. (1998)	
samp. P1968a	4.340000	0.007000	Myers et al. (2019)	
samp. P1968b	5.850000	0.012300	Myers et al. (2019)	
samp. P1968c	5.754571	0.010663	Myers et al. (2019)	
samp. HPR3 -1_XL -3	5.943750	0.010000	Mercer et al. (2015)	
samp. HPR3 -1_XL -4_INCL -1	5.340000	0.008000	Mercer et al. (2015)	
AW -6	1.420000	0.129800	Iacovino et al. (2016)	
AW -46	4.760000	0.343900	Iacovino et al. (2016)	
KI -07	2.900000	0.113100	Iacovino et al. (2016)	

	ROCK TYPE	Press	Temp	\
Kil3 -6_1a	Basalt	62.5	1299.094712	
Kil3 -6_3a	Basalt	128.0	1283.419991	
Kil3 -6_4a	Basalt	100.0	1255.153759	
10*	Basalt	2000.0	1200.000000	

19*	Basalt	2000.0	1200.000000
25	Basalt	2000.0	1200.000000
SAT -M12 -1	Andesite	703.0	1100.000000
SAT -M12 -2	Andesite	1865.0	1100.000000
SAT -M12 -4	Andesite	2985.0	1050.000000
samp. P1968a	Rhyolite	300.0	900.000000
samp. P1968b	Rhyolite	300.0	900.000000
samp. P1968c	Rhyolite	300.0	900.000000
samp. HPR3 -1_XL -3	Rhyolite	300.0	0.000000
samp. HPR3 -1_XL -4_INCL -1	Rhyolite	0.0	900.000000
AW -6	Phonotephrite	1500.0	1050.000000
AW -46	Basaltic -Trachyandesite	4000.0	1000.000000
KI -07	Basanite	1500.0	1100.000000

	XH2O_fl_VESIcal	XC02_fl_VESIcal	Model \
Kil3 -6_1a	0.586164	0.413836	MagmaSat
Kil3 -6_3a	0.286160	0.713840	MagmaSat
Kil3 -6_4a	0.377439	0.622561	MagmaSat
10*	0.892371	0.107629	MagmaSat
19*	0.918888	0.081112	MagmaSat
25	0.955803	0.044197	MagmaSat
SAT -M12 -1	1.000000	0.000000	MagmaSat
SAT -M12 -2	1.000000	0.000000	MagmaSat
SAT -M12 -4	1.000000	0.000000	MagmaSat
samp. P1968a	0.998764	0.001236	MagmaSat
samp. P1968b	0.998686	0.001314	MagmaSat
samp. P1968c	0.998831	0.001169	MagmaSat
samp. HPR3 -1_XL -3	NaN	NaN	MagmaSat
samp. HPR3 -1_XL -4_INCL -1	NaN	NaN	MagmaSat
AW -6	0.000000	0.000000	MagmaSat
AW -46	0.492213	0.507787	MagmaSat
KI -07	0.681758	0.318242	MagmaSat

Warnings

Kil3 -6_1a
 Kil3 -6_3a
 Kil3 -6_4a
 10*
 19*
 25
 SAT -M12 -1
 SAT -M12 -2
 SAT -M12 -4
 samp. P1968a
 samp. P1968b
 samp. P1968c
 samp. HPR3 -1_XL -3 Calculation skipped. Bad temperature.
 samp. HPR3 -1_XL -4_INCL -1 Calculation skipped. Bad pressure.
 AW -6 Sample not saturated at these conditions
 AW -46
 KI -07

[17 rows x 24 columns]

874 *0.3.6.1 Converting fluid composition units* The fluid composition is always re-
 875 turned in units of mol fraction. Two functions exist to transform only the H₂O-CO₂ fluid
 876 composition between mol fraction and wt% and can be applied to returned data sets from
 877 calculations. Both functions require that the user provide the dataframe containing fluid
 878 composition information plus the names of the columns corresponding to the H₂O and
 879 CO₂ concentrations in the fluid. The default values for column names are set to those
 880 that may be returned by VESICAL core calculations, such that they need not be speci-
 881 fied unless the user has changed them or is supplying their own data (e.g., imported data
 882 not processed through a core calculation).

883 **Method structure** Mol fraction to wt%: `fluid_molfrac_to_wt(data, H2O_colname='XH2O_fl_VESICAL',`
 884 `CO2_colname='XCO2_fl_VESICAL')`

885 Wt% to mol fraction: `fluid_wt_to_molfrac(data, H2O_colname='H2O_fl_wt',`
 886 `CO2_colname='CO2_fl_wt')`

887 **Required inputs** `data`: A pandas DataFrame containing columns for H₂O and CO₂
 888 concentrations in the fluid.

889 **Optional inputs** `H2O_colname` and `CO2_colname`: The default values are 'XH2O_fl'
 890 and 'XCO2_fl' if input data are in mol fraction or 'H2O_fl_wt' and 'CO2_fl_wt'
 891 if the data are in wt%. Strings containing the name of the columns correspond-
 892 ing to the H₂O and CO₂ concentrations in the fluid.

893 **Calculated outputs** The original data passed plus newly calculated values are returned
 894 in a DataFrame.

895 [H] []Equilibrium fluid compositions converted from mol fraction to wt%

"""Converting from mol fraction to wt%"""

eqfluid_wt = v.fluid_molfrac_to_wt(eqfluid)

eqfluid_wt

SiO2	TiO2	Al2O3	Fe2O3	Cr2O3	\		
Kil3 -6_1a			48.249207	2.222114	11.692194	0.00	0.0
Kil3 -6_3a			48.295691	2.165357	11.755584	0.00	0.0
Kil3 -6_4a			49.124079	2.360984	12.172833	0.00	0.0
10*			47.960000	0.780000	18.770000	0.00	0.0
19*			49.640000	0.710000	18.050000	0.00	0.0
25			50.320000	0.720000	18.030000	0.00	0.0
SAT -M12 -1			62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -2			62.600000	0.630000	17.300000	2.01	0.0
SAT -M12 -4			62.600000	0.630000	17.300000	2.01	0.0
samp. P1968a			76.974880	0.085516	3.110636	0.00	0.0
samp. P1968b			76.943845	0.133125	3.169657	0.00	0.0
samp. P1968c			77.187205	0.119506	3.167827	0.00	0.0
samp. HPR3 -1_XL -3			75.413966	0.095164	14.077692	0.00	0.0
samp. HPR3 -1_XL -4_INCL -1			76.613586	0.095843	13.476762	0.00	0.0
AW -6			48.030000	2.840000	18.120000	0.00	0.0
AW -46			52.980000	2.180000	20.490000	0.00	0.0
KI -07			44.610000	4.370000	14.410000	0.00	0.0

	FeO	MnO	MgO	NiO	CoO	...	\
Kil3 -6_1a	0.000000	0.079999	14.183817	0.0	0.0	...	
Kil3 -6_3a	0.000000	0.084045	13.403980	0.0	0.0	...	
Kil3 -6_4a	0.000000	0.098809	11.997699	0.0	0.0	...	
10*	10.920000	0.150000	6.860000	0.0	0.0	...	
19*	10.540000	0.190000	6.430000	0.0	0.0	...	
25	10.110000	0.140000	5.650000	0.0	0.0	...	

SAT -M12 -1	2.010000	0.060000	2.650000	0.0	0.0	...
SAT -M12 -2	2.010000	0.060000	2.650000	0.0	0.0	...
SAT -M12 -4	2.010000	0.060000	2.650000	0.0	0.0	...
samp. P1968a	4.788883	0.000000	12.549439	0.0	0.0	...
samp. P1968b	4.763435	0.000000	12.446403	0.0	0.0	...
samp. P1968c	4.814076	0.000000	12.229534	0.0	0.0	...
samp. HPR3 -1_XL -3	0.654992	0.125882	0.012003	0.0	0.0	...
samp. HPR3 -1_XL -4_INCL -1	0.620769	0.113495	0.032069	0.0	0.0	...
AW -6	9.600000	0.230000	3.080000	0.0	0.0	...
AW -46	5.540000	0.200000	2.000000	0.0	0.0	...
KI -07	10.600000	0.170000	7.690000	0.0	0.0	...

	Press	Temp	XH2O_fl_VESICAL	\
Kil3 -6_1a	62.5	1299.094712	0.000000	
Kil3 -6_3a	128.0	1283.419991	0.000000	
Kil3 -6_4a	100.0	1255.153759	0.000000	
10*	2000.0	1200.000000	0.984531	
19*	2000.0	1200.000000	0.974997	
25	2000.0	1200.000000	0.990107	
SAT -M12 -1	703.0	1100.000000	1.000000	
SAT -M12 -2	1865.0	1100.000000	1.000000	
SAT -M12 -4	2985.0	1050.000000	1.000000	
samp. P1968a	300.0	900.000000	0.977773	
samp. P1968b	300.0	900.000000	0.996799	
samp. P1968c	300.0	900.000000	0.997028	
samp. HPR3 -1_XL -3	300.0	0.000000	0.997770	
samp. HPR3 -1_XL -4_INCL -1	0.0	900.000000	0.997273	
AW -6	1500.0	1050.000000	0.261572	
AW -46	4000.0	1000.000000	0.897441	
KI -07	1500.0	1100.000000	0.826014	

	XC02_fl_VESICAL	Temperature_C_VESICAL	\
Kil3 -6_1a	0.000000	900.0	
Kil3 -6_3a	0.000000	900.0	
Kil3 -6_4a	0.000000	900.0	
10*	0.015469	900.0	
19*	0.025003	900.0	
25	0.009893	900.0	
SAT -M12 -1	0.000000	900.0	
SAT -M12 -2	0.000000	900.0	
SAT -M12 -4	0.000000	900.0	
samp. P1968a	0.022227	900.0	
samp. P1968b	0.003201	900.0	
samp. P1968c	0.002972	900.0	
samp. HPR3 -1_XL -3	0.002230	900.0	
samp. HPR3 -1_XL -4_INCL -1	0.002727	900.0	
AW -6	0.738428	900.0	
AW -46	0.102559	900.0	
KI -07	0.173986	900.0	

	Pressure_bars_VESICAL	Model	\
Kil3 -6_1a	1000.0	MagmaSat	
Kil3 -6_3a	1000.0	MagmaSat	
Kil3 -6_4a	1000.0	MagmaSat	
10*	1000.0	MagmaSat	

19*	1000.0	MagmaSat
25	1000.0	MagmaSat
SAT -M12 -1	1000.0	MagmaSat
SAT -M12 -2	1000.0	MagmaSat
SAT -M12 -4	1000.0	MagmaSat
samp. P1968a	1000.0	MagmaSat
samp. P1968b	1000.0	MagmaSat
samp. P1968c	1000.0	MagmaSat
samp. HPR3 -1_XL -3	1000.0	MagmaSat
samp. HPR3 -1_XL -4_INCL -1	1000.0	MagmaSat
AW -6	1000.0	MagmaSat
AW -46	1000.0	MagmaSat
KI -07	1000.0	MagmaSat

Warnings \

Kil3 -6_1a	Sample not saturated at these conditions
Kil3 -6_3a	Sample not saturated at these conditions
Kil3 -6_4a	Sample not saturated at these conditions

10*
 19*
 25
 SAT -M12 -1
 SAT -M12 -2
 SAT -M12 -4
 samp. P1968a
 samp. P1968b
 samp. P1968c
 samp. HPR3 -1_XL -3
 samp. HPR3 -1_XL -4_INCL -1
 AW -6
 AW -46
 KI -07

	H2O_fl_wt	CO2_fl_wt
Kil3 -6_1a	NaN	NaN
Kil3 -6_3a	NaN	NaN
Kil3 -6_4a	NaN	NaN
10*	96.304445	3.695555
19*	94.106168	5.893832
25	97.617908	2.382092
SAT -M12 -1	100.000000	0.000000
SAT -M12 -2	100.000000	0.000000
SAT -M12 -4	100.000000	0.000000
samp. P1968a	94.740209	5.259791
samp. P1968b	99.221744	0.778256
samp. P1968c	99.277291	0.722709
samp. HPR3 -1_XL -3	99.457027	0.542973
samp. HPR3 -1_XL -4_INCL -1	99.336700	0.663300
AW -6	12.666745	87.333255
AW -46	78.179789	21.820211
KI -07	66.031544	33.968456

[17 rows x 28 columns]

""Converting from wt% to mol fraction""

```
eqfluid_mol = v.fluid_wt_to_molfrac(eqfluid_wt)
```

```
eqfluid_mol
```

SiO2	TiO2	Al2O3	Fe2O3	Cr2O3	\				
Kil3 -6_1a			48.249207	2.222114	11.692194	0.00	0.0		
Kil3 -6_3a			48.295691	2.165357	11.755584	0.00	0.0		
Kil3 -6_4a			49.124079	2.360984	12.172833	0.00	0.0		
10*			47.960000	0.780000	18.770000	0.00	0.0		
19*			49.640000	0.710000	18.050000	0.00	0.0		
25			50.320000	0.720000	18.030000	0.00	0.0		
SAT -M12 -1			62.600000	0.630000	17.300000	2.01	0.0		
SAT -M12 -2			62.600000	0.630000	17.300000	2.01	0.0		
SAT -M12 -4			62.600000	0.630000	17.300000	2.01	0.0		
samp. P1968a			76.974880	0.085516	3.110636	0.00	0.0		
samp. P1968b			76.943845	0.133125	3.169657	0.00	0.0		
samp. P1968c			77.187205	0.119506	3.167827	0.00	0.0		
samp. HPR3 -1_XL -3			75.413966	0.095164	14.077692	0.00	0.0		
samp. HPR3 -1_XL -4_INCL -1			76.613586	0.095843	13.476762	0.00	0.0		
AW -6			48.030000	2.840000	18.120000	0.00	0.0		
AW -46			52.980000	2.180000	20.490000	0.00	0.0		
KI -07			44.610000	4.370000	14.410000	0.00	0.0		

	FeO	MnO	MgO	NiO	CoO	...	\
Kil3 -6_1a	0.000000	0.079999	14.183817	0.0	0.0	...	
Kil3 -6_3a	0.000000	0.084045	13.403980	0.0	0.0	...	
Kil3 -6_4a	0.000000	0.098809	11.997699	0.0	0.0	...	
10*	10.920000	0.150000	6.860000	0.0	0.0	...	
19*	10.540000	0.190000	6.430000	0.0	0.0	...	
25	10.110000	0.140000	5.650000	0.0	0.0	...	
SAT -M12 -1	2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -2	2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -4	2.010000	0.060000	2.650000	0.0	0.0	...	
samp. P1968a	4.788883	0.000000	12.549439	0.0	0.0	...	
samp. P1968b	4.763435	0.000000	12.446403	0.0	0.0	...	
samp. P1968c	4.814076	0.000000	12.229534	0.0	0.0	...	
samp. HPR3 -1_XL -3	0.654992	0.125882	0.012003	0.0	0.0	...	
samp. HPR3 -1_XL -4_INCL -1	0.620769	0.113495	0.032069	0.0	0.0	...	
AW -6	9.600000	0.230000	3.080000	0.0	0.0	...	
AW -46	5.540000	0.200000	2.000000	0.0	0.0	...	
KI -07	10.600000	0.170000	7.690000	0.0	0.0	...	

	XH2O_fl_VESIcal	XC02_fl_VESIcal	\
Kil3 -6_1a	0.000000	0.000000	
Kil3 -6_3a	0.000000	0.000000	
Kil3 -6_4a	0.000000	0.000000	
10*	0.984531	0.015469	
19*	0.974997	0.025003	
25	0.990107	0.009893	
SAT -M12 -1	1.000000	0.000000	
SAT -M12 -2	1.000000	0.000000	
SAT -M12 -4	1.000000	0.000000	
samp. P1968a	0.977773	0.022227	
samp. P1968b	0.996799	0.003201	
samp. P1968c	0.997028	0.002972	

samp. HPR3 -1_XL -3	0.997770	0.002230
samp. HPR3 -1_XL -4_INCL -1	0.997273	0.002727
AW -6	0.261572	0.738428
AW -46	0.897441	0.102559
KI -07	0.826014	0.173986

	Temperature_C_VESIcal	Pressure_bars_VESIcal	\
Kil3 -6_1a	900.0	1000.0	
Kil3 -6_3a	900.0	1000.0	
Kil3 -6_4a	900.0	1000.0	
10*	900.0	1000.0	
19*	900.0	1000.0	
25	900.0	1000.0	
SAT -M12 -1	900.0	1000.0	
SAT -M12 -2	900.0	1000.0	
SAT -M12 -4	900.0	1000.0	
samp. P1968a	900.0	1000.0	
samp. P1968b	900.0	1000.0	
samp. P1968c	900.0	1000.0	
samp. HPR3 -1_XL -3	900.0	1000.0	
samp. HPR3 -1_XL -4_INCL -1	900.0	1000.0	
AW -6	900.0	1000.0	
AW -46	900.0	1000.0	
KI -07	900.0	1000.0	

Model

↔ Warnings \

Kil3 -6_1a	MagmaSat	Sample not saturated at these
↔ conditions		
Kil3 -6_3a	MagmaSat	Sample not saturated at these
↔ conditions		
Kil3 -6_4a	MagmaSat	Sample not saturated at these
↔ conditions		
10*	MagmaSat	
19*	MagmaSat	
25	MagmaSat	
SAT -M12 -1	MagmaSat	
SAT -M12 -2	MagmaSat	
SAT -M12 -4	MagmaSat	
samp. P1968a	MagmaSat	
samp. P1968b	MagmaSat	
samp. P1968c	MagmaSat	
samp. HPR3 -1_XL -3	MagmaSat	
samp. HPR3 -1_XL -4_INCL -1	MagmaSat	
AW -6	MagmaSat	
AW -46	MagmaSat	
KI -07	MagmaSat	

	H2O_fl_wt	CO2_fl_wt	XH2O_fl	XC02_fl
Kil3 -6_1a	NaN	NaN	NaN	NaN
Kil3 -6_3a	NaN	NaN	NaN	NaN
Kil3 -6_4a	NaN	NaN	NaN	NaN
10*	96.304445	3.695555	0.984531	0.015469
19*	94.106168	5.893832	0.974997	0.025003
25	97.617908	2.382092	0.990107	0.009893

SAT -M12 -1	100.000000	0.000000	1.000000	0.000000
SAT -M12 -2	100.000000	0.000000	1.000000	0.000000
SAT -M12 -4	100.000000	0.000000	1.000000	0.000000
samp. P1968a	94.740209	5.259791	0.977773	0.022227
samp. P1968b	99.221744	0.778256	0.996799	0.003201
samp. P1968c	99.277291	0.722709	0.997028	0.002972
samp. HPR3 -1_XL -3	99.457027	0.542973	0.997770	0.002230
samp. HPR3 -1_XL -4_INCL -1	99.336700	0.663300	0.997273	0.002727
AW -6	12.666745	87.333255	0.261572	0.738428
AW -46	78.179789	21.820211	0.897441	0.102559
KI -07	66.031544	33.968456	0.826014	0.173986

[17 rows x 30 columns]

897

0.3.7 Calculating saturation pressures

898

899

900

901

902

The `calculate_saturation_pressure()` function calculates the minimum pressure at which a given silicate melt with known temperature and H₂O and CO₂ concentrations would be saturated with fluid. For MagmaSat, this is calculated by finding the pressure at which the smallest amount of vapor is present. This function also calculates the composition of the vapor in equilibrium with the melt at those conditions.

903

904

905

906

907

908

909

The function works by calculating the equilibrium state of the given melt at very high pressure (20,000 bars). If no fluid is present at this pressure, the melt is undersaturated, and pressure is decreased in steps of 1,000 bars until the mass of vapor is >0 grams. If fluid is present, the saturation limit is found by increasing the pressure iteratively until the point at which no fluid is present. At this point, the pressure space is narrowed and searched in steps of 100 bars and then in steps of 10 bars until the saturation pressure is found. Thus, these calculations are accurate to 10 bars.

910

911

912

913

914

915

For non-MagmaSat models, we use Brent's minimization method (via `scipy`'s `root_scalar` optimization function) to find the pressure that satisfies the computational constraints. This is achieved by iterative calculation of the dissolved volatile concentration over a range of pressures and minimizing the difference between computed and given concentrations. This is only practical for non-MagmaSat models, where the dissolved volatiles calculation is extremely fast.

916

917

918

919

Method structure Single sample: `calculate_saturation_pressure(sample, temperature, verbose=False, model='MagmaSat').result`
 BatchFile batch process: `myfile.calculate_saturation_pressure(temperature, print_status=True, model='MagmaSat')`

920

921

922

923

924

925

926

927

Standard inputs `sample`, `temperature`, `model` (see Section 0.3.1).

Unique optional inputs `verbose`: *Only for single sample calculations.* Default value is False, in which case the saturation pressure in bars is returned. If set to True, additional parameters are returned in a dictionary: saturation pressure in bars, H₂O and CO₂ concentrations in the fluid, mass of the fluid in grams, and proportion of the fluid in the system in wt%.

`print_status`: *Only for batch calculations.* The default value is True, in which case the progress of the calculation will be printed to the terminal.

928

929

930

931

Calculated outputs If the single-sample method is used, the saturation pressure in bars is returned as a numerical value (float) (plus additional variables 'XH₂O_fl', 'XCO₂_fl', 'FluidMass_grams', and 'FluidProportion_wtper' if `verbose` is set to True).

932 If the BatchFile method is used, a pandas DataFrame is returned with sample in-
 933 formation plus calculated saturation pressures, equilibrium fluid compositions, mass
 934 of the fluid in grams, and proportion of the fluid in the system in wt%. Temper-
 935 ature (in °C) is always returned.

```
"""Calculate the saturation pressure of the single sample we defined in
↳ Section 3.3.1 at 925 degrees C"""
```

```
v.calculate_saturation_pressure(sample=mysample, temperature=925.0,
↳ verbose=True).result
```

```
{'SaturationP_bars': 2960,
'FluidMass_grams': 0.0018160337487088,
'FluidProportion_wt': 0.0018160337487087978,
'XH2O_fl': 0.838064480487942,
'XC02_fl': 0.161935519512058}
```

936

[H] []Isothermally modeled saturation pressures

```
"""Calculate the saturation pressure for all samples in an BatchFile object
↳ at 925 degrees C"""
```

```
satPs = myfile.calculate_saturation_pressure(temperature=925.0)
satPs
```

SiO2	TiO2	Al2O3	Fe2O3	Cr2O3	\				
Kil3 -6_1a			48.249207	2.222114	11.692194	0.00	0.0		
Kil3 -6_3a			48.295691	2.165357	11.755584	0.00	0.0		
Kil3 -6_4a			49.124079	2.360984	12.172833	0.00	0.0		
10*			47.960000	0.780000	18.770000	0.00	0.0		
19*			49.640000	0.710000	18.050000	0.00	0.0		
25			50.320000	0.720000	18.030000	0.00	0.0		
SAT -M12 -1			62.600000	0.630000	17.300000	2.01	0.0		
SAT -M12 -2			62.600000	0.630000	17.300000	2.01	0.0		
SAT -M12 -4			62.600000	0.630000	17.300000	2.01	0.0		
samp. P1968a			76.974880	0.085516	3.110636	0.00	0.0		
samp. P1968b			76.943845	0.133125	3.169657	0.00	0.0		
samp. P1968c			77.187205	0.119506	3.167827	0.00	0.0		
samp. HPR3 -1_XL -3			75.413966	0.095164	14.077692	0.00	0.0		
samp. HPR3 -1_XL -4_INCL -1			76.613586	0.095843	13.476762	0.00	0.0		
AW -6			48.030000	2.840000	18.120000	0.00	0.0		
AW -46			52.980000	2.180000	20.490000	0.00	0.0		
KI -07			44.610000	4.370000	14.410000	0.00	0.0		
			FeO	MnO	MgO	NiO	CoO	...	\
Kil3 -6_1a			0.000000	0.079999	14.183817	0.0	0.0	...	
Kil3 -6_3a			0.000000	0.084045	13.403980	0.0	0.0	...	
Kil3 -6_4a			0.000000	0.098809	11.997699	0.0	0.0	...	
10*			10.920000	0.150000	6.860000	0.0	0.0	...	
19*			10.540000	0.190000	6.430000	0.0	0.0	...	
25			10.110000	0.140000	5.650000	0.0	0.0	...	
SAT -M12 -1			2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -2			2.010000	0.060000	2.650000	0.0	0.0	...	
SAT -M12 -4			2.010000	0.060000	2.650000	0.0	0.0	...	
samp. P1968a			4.788883	0.000000	12.549439	0.0	0.0	...	
samp. P1968b			4.763435	0.000000	12.446403	0.0	0.0	...	
samp. P1968c			4.814076	0.000000	12.229534	0.0	0.0	...	

samp. HPR3 -1_XL -3	0.654992	0.125882	0.012003	0.0	0.0	...
samp. HPR3 -1_XL -4_INCL -1	0.620769	0.113495	0.032069	0.0	0.0	...
AW -6	9.600000	0.230000	3.080000	0.0	0.0	...
AW -46	5.540000	0.200000	2.000000	0.0	0.0	...
KI -07	10.600000	0.170000	7.690000	0.0	0.0	...

	Press	Temp	SaturationP	bars_VESICAL	\
Kil3 -6_1a	62.5	1299.094712	60		
Kil3 -6_3a	128.0	1283.419991	130		
Kil3 -6_4a	100.0	1255.153759	100		
10*	2000.0	1200.000000	2500		
19*	2000.0	1200.000000	3600		
25	2000.0	1200.000000	2750		
SAT -M12 -1	703.0	1100.000000	550		
SAT -M12 -2	1865.0	1100.000000	1590		
SAT -M12 -4	2985.0	1050.000000	2540		
samp. P1968a	300.0	900.000000	1100		
samp. P1968b	300.0	900.000000	1790		
samp. P1968c	300.0	900.000000	1730		
samp. HPR3 -1_XL -3	300.0	0.000000	2090		
samp. HPR3 -1_XL -4_INCL -1	0.0	900.000000	1730		
AW -6	1500.0	1050.000000	1220		
AW -46	4000.0	1000.000000	4800		
KI -07	1500.0	1100.000000	1510		

	Temperature_C_VESICAL	XH2O_fl_VESICAL	\
Kil3 -6_1a	925.0	0.469913	
Kil3 -6_3a	925.0	0.215529	
Kil3 -6_4a	925.0	0.292354	
10*	925.0	0.796514	
19*	925.0	0.702654	
25	925.0	0.836895	
SAT -M12 -1	925.0	1.000000	
SAT -M12 -2	925.0	1.000000	
SAT -M12 -4	925.0	1.000000	
samp. P1968a	925.0	0.972472	
samp. P1968b	925.0	0.972875	
samp. P1968c	925.0	0.975614	
samp. HPR3 -1_XL -3	925.0	0.951891	
samp. HPR3 -1_XL -4_INCL -1	925.0	0.950741	
AW -6	925.0	0.231708	
AW -46	925.0	0.456750	
KI -07	925.0	0.684729	

	XC02_fl_VESICAL	FluidMass_grams_VESICAL	\
Kil3 -6_1a	0.530087	0.000836	
Kil3 -6_3a	0.784471	0.000038	
Kil3 -6_4a	0.707646	0.000635	
10*	0.203486	0.001232	
19*	0.297346	0.000797	
25	0.163105	0.000226	
SAT -M12 -1	0.000000	0.012903	
SAT -M12 -2	0.000000	0.001052	
SAT -M12 -4	0.000000	0.016093	
samp. P1968a	0.027528	0.007924	

samp. P1968b	0.027125	0.006671
samp. P1968c	0.024386	0.008637
samp. HPR3 -1_XL -3	0.048109	0.002941
samp. HPR3 -1_XL -4_INCL -1	0.049259	0.002864
AW -6	0.768292	0.000093
AW -46	0.543250	0.000938
KI -07	0.315271	0.000431

	FluidSystem_wt_VESICAL	Model	Warnings
Kil3 -6_1a	0.000836	MagmaSat	
Kil3 -6_3a	0.000038	MagmaSat	
Kil3 -6_4a	0.000635	MagmaSat	
10*	0.001232	MagmaSat	
19*	0.000797	MagmaSat	
25	0.000226	MagmaSat	
SAT -M12 -1	0.012903	MagmaSat	
SAT -M12 -2	0.001052	MagmaSat	
SAT -M12 -4	0.016093	MagmaSat	
samp. P1968a	0.007924	MagmaSat	
samp. P1968b	0.006671	MagmaSat	
samp. P1968c	0.008637	MagmaSat	
samp. HPR3 -1_XL -3	0.002941	MagmaSat	
samp. HPR3 -1_XL -4_INCL -1	0.002864	MagmaSat	
AW -6	0.000093	MagmaSat	
AW -46	0.000938	MagmaSat	
KI -07	0.000431	MagmaSat	

[17 rows x 28 columns]

937 [H] **Modeled saturation pressures with unique temperatures** The warning
 938 "Bad temperature" indicates that no data (or 0.0 value data) was given for the
 939 temperature of that sample, in which case the calculation of that sample is skipped.

```

"""Calculate the saturation pressure for all samples in an BatchFile
  ↪ object, taking temperature
  values from a column named "Temp" in the BatchFile"""
satPs_wtemps = myfile.calculate_saturation_pressure(temperature="Temp")
satPs_wtemps

```

SiO2	TiO2	Al2O3	Fe2O3	Cr2O3	\		
Kil3 -6_1a		48.249207	2.222114	11.692194	0.00	0.0	
Kil3 -6_3a		48.295691	2.165357	11.755584	0.00	0.0	
Kil3 -6_4a		49.124079	2.360984	12.172833	0.00	0.0	
10*		47.960000	0.780000	18.770000	0.00	0.0	
19*		49.640000	0.710000	18.050000	0.00	0.0	
25		50.320000	0.720000	18.030000	0.00	0.0	
SAT -M12 -1		62.600000	0.630000	17.300000	2.01	0.0	
SAT -M12 -2		62.600000	0.630000	17.300000	2.01	0.0	
SAT -M12 -4		62.600000	0.630000	17.300000	2.01	0.0	
samp. P1968a		76.974880	0.085516	3.110636	0.00	0.0	
samp. P1968b		76.943845	0.133125	3.169657	0.00	0.0	
samp. P1968c		77.187205	0.119506	3.167827	0.00	0.0	
samp. HPR3 -1_XL -3		75.413966	0.095164	14.077692	0.00	0.0	
samp. HPR3 -1_XL -4_INCL -1		76.613586	0.095843	13.476762	0.00	0.0	
AW -6		48.030000	2.840000	18.120000	0.00	0.0	

AW -46	52.980000	2.180000	20.490000	0.00	0.0
KI -07	44.610000	4.370000	14.410000	0.00	0.0
	FeO	MnO	MgO	NiO	CoO ... \
Kil3 -6_1a	0.000000	0.079999	14.183817	0.0	0.0 ...
Kil3 -6_3a	0.000000	0.084045	13.403980	0.0	0.0 ...
Kil3 -6_4a	0.000000	0.098809	11.997699	0.0	0.0 ...
10*	10.920000	0.150000	6.860000	0.0	0.0 ...
19*	10.540000	0.190000	6.430000	0.0	0.0 ...
25	10.110000	0.140000	5.650000	0.0	0.0 ...
SAT -M12 -1	2.010000	0.060000	2.650000	0.0	0.0 ...
SAT -M12 -2	2.010000	0.060000	2.650000	0.0	0.0 ...
SAT -M12 -4	2.010000	0.060000	2.650000	0.0	0.0 ...
samp. P1968a	4.788883	0.000000	12.549439	0.0	0.0 ...
samp. P1968b	4.763435	0.000000	12.446403	0.0	0.0 ...
samp. P1968c	4.814076	0.000000	12.229534	0.0	0.0 ...
samp. HPR3 -1_XL -3	0.654992	0.125882	0.012003	0.0	0.0 ...
samp. HPR3 -1_XL -4_INCL -1	0.620769	0.113495	0.032069	0.0	0.0 ...
AW -6	9.600000	0.230000	3.080000	0.0	0.0 ...
AW -46	5.540000	0.200000	2.000000	0.0	0.0 ...
KI -07	10.600000	0.170000	7.690000	0.0	0.0 ...
	ROCK TYPE	Press	Temp	\	
Kil3 -6_1a	Basalt	62.5	1299.094712		
Kil3 -6_3a	Basalt	128.0	1283.419991		
Kil3 -6_4a	Basalt	100.0	1255.153759		
10*	Basalt	2000.0	1200.000000		
19*	Basalt	2000.0	1200.000000		
25	Basalt	2000.0	1200.000000		
SAT -M12 -1	Andesite	703.0	1100.000000		
SAT -M12 -2	Andesite	1865.0	1100.000000		
SAT -M12 -4	Andesite	2985.0	1050.000000		
samp. P1968a	Rhyolite	300.0	900.000000		
samp. P1968b	Rhyolite	300.0	900.000000		
samp. P1968c	Rhyolite	300.0	900.000000		
samp. HPR3 -1_XL -3	Rhyolite	300.0	0.000000		
samp. HPR3 -1_XL -4_INCL -1	Rhyolite	0.0	900.000000		
AW -6	Phonotephrite	1500.0	1050.000000		
AW -46	Basaltic -Trachyandesite	4000.0	1000.000000		
KI -07	Basanite	1500.0	1100.000000		
	SaturationP_bars_VESIcal	XH20_fl_VESIcal	\		
Kil3 -6_1a	60.0	0.493184			
Kil3 -6_3a	110.0	0.266595			
Kil3 -6_4a	90.0	0.337738			
10*	2540.0	0.817548			
19*	3650.0	0.725724			
25	2850.0	0.855214			
SAT -M12 -1	580.0	1.000000			
SAT -M12 -2	1650.0	1.000000			
SAT -M12 -4	2610.0	1.000000			
samp. P1968a	1090.0	0.972916			
samp. P1968b	1780.0	0.973133			
samp. P1968c	1720.0	0.975860			
samp. HPR3 -1_XL -3	NaN	NaN			

samp. HPR3 -1_XL -4_INCL -1	1730.0	0.951017
AW -6	1280.0	0.228644
AW -46	4910.0	0.458904
KI -07	1590.0	0.679643

	XC02_fl_VESICAL	FluidMass_grams_VESICAL	\
Kil3 -6_1a	0.506816	0.000610	
Kil3 -6_3a	0.733405	0.000700	
Kil3 -6_4a	0.662262	0.000807	
10*	0.182452	0.001532	
19*	0.274276	0.000669	
25	0.144786	0.000849	
SAT -M12 -1	0.000000	0.003442	
SAT -M12 -2	0.000000	0.015280	
SAT -M12 -4	0.000000	0.008153	
samp. P1968a	0.027084	0.008855	
samp. P1968b	0.026867	0.005916	
samp. P1968c	0.024140	0.008088	
samp. HPR3 -1_XL -3	NaN	NaN	
samp. HPR3 -1_XL -4_INCL -1	0.048983	0.003350	
AW -6	0.771356	0.001475	
AW -46	0.541096	0.001767	
KI -07	0.320357	0.001914	

	FluidSystem_wt_VESICAL	Model	\
Kil3 -6_1a	0.000610	MagmaSat	
Kil3 -6_3a	0.000700	MagmaSat	
Kil3 -6_4a	0.000807	MagmaSat	
10*	0.001532	MagmaSat	
19*	0.000669	MagmaSat	
25	0.000849	MagmaSat	
SAT -M12 -1	0.003442	MagmaSat	
SAT -M12 -2	0.015280	MagmaSat	
SAT -M12 -4	0.008153	MagmaSat	
samp. P1968a	0.008855	MagmaSat	
samp. P1968b	0.005916	MagmaSat	
samp. P1968c	0.008088	MagmaSat	
samp. HPR3 -1_XL -3	NaN	MagmaSat	
samp. HPR3 -1_XL -4_INCL -1	0.003350	MagmaSat	
AW -6	0.001475	MagmaSat	
AW -46	0.001767	MagmaSat	
KI -07	0.001914	MagmaSat	

Warnings

Kil3 -6_1a
 Kil3 -6_3a
 Kil3 -6_4a
 10*
 19*
 25
 SAT -M12 -1
 SAT -M12 -2
 SAT -M12 -4
 samp. P1968a
 samp. P1968b

```

samp. P1968c
samp. HPR3 -1_XL -3          Calculation skipped. Bad temperature.
samp. HPR3 -1_XL -4_INCL -1
AW -6
AW -46
KI -07

```

```
[17 rows x 27 columns]
```

940 *0.3.8 Calculating isobars and isopleths*

941 In this example, we demonstrate how isobars (lines of constant pressure) and iso-
 942 pleths (lines of constant fluid composition) can be calculated for any one composition.
 943 A single melt composition can be extracted from a loaded batch file, or a composition
 944 can be entered by hand and stored within a dictionary. Due to computational intensity,
 945 isobars and isopleths can only be computed for one sample composition at a time.

946 Once a single composition is defined, conditions over which to calculate isobars and
 947 isopleths must be specified. The generated plot is isothermal, so only one temperature
 948 can be chosen. Isobars and isopleths can be calculated for any number of pressures or
 949 $X_{H_2O}^{fluid}$ values, respectively, passed as lists.

950 The calculation is performed by iterating through possible concentrations of H_2O
 951 and CO_2 and calculating the equilibrium state for the system. The iteration begins at
 952 a fixed H_2O concentration, increasing the CO_2 concentration in steps of 0.1 wt% until
 953 a fluid phase is stable. The H_2O concentration is then increased by 0.5 wt% and CO_2
 954 is again increased from 0 until a fluid phase is stable. This process is repeated for H_2O
 955 values ranging from 0–15 wt%. The H_2O and CO_2 concentrations from each system for
 956 which a fluid phase was found to be stable are saved and written to a pandas DataFrame,
 957 which is returned upon completion of the calculation.

958 Isobars and isopleths are computed at fixed H_2O - CO_2 points for any given pres-
 959 sure. To generate curves using the MagmaSat model, polynomials are fit to computed
 960 points using numpy's polyfit method. This can be optionally disabled by setting `smooth_isobars`
 961 or `smooth_isopleths` to False. The curvature of the isobars depends strongly on the num-
 962 ber of points used to fit a polynomial, deemed “control points”, with curve fits becom-
 963 ing more accurate to the model as the number of control points increases. We found that
 964 above five control points, changes to the shape of the curve fits becomes negligible. Thus,
 965 as a compromise between accuracy and computation time, and to maintain consistency,
 966 MagmaSat isobars are always computed with 5 control points at $X_{H_2O}^{fluid}$ values of 0,
 967 0.25, 0.5, 0.75, and 1. Because non-MagmaSat models compute extremely quickly, all
 968 non-MagmaSat models use 51 control points per isobar and do not utilize polynomial
 969 fits to the data by default.

970 **Method structure** *Only single sample calculations.* `calculate_isobars_and_isopleths(sample,`
 971 `temperature, pressure_list, isopleth_list=None, smooth_isobars=True,`
 972 `smooth_isopleths=True, print_status=True, model="MagmaSat").result`

973 **Standard inputs** `sample, temperature, model` (see Section 0.3.1).

974 **Unique required inputs** `pressure_list`: A list of all pressures in bars at which to
 975 calculate isobars. If only one value is passed it can be as float instead of list.

976 **Unique optional inputs** `isopleth_list`: The default value is None in which case only
 977 isobars will be calculated. A list of all fluid composition values, in mole fraction
 978 H_2O ($X_{H_2O}^{fluid}$), at which to calculate isopleths. Values can range from 0–1. If

979 only one value is passed it can be as float instead of list. N.b. that, due to the method
 980 of isobar smoothing using control points as outlined above, each isopleth value passed
 981 here not equal to one of the five standard control point values (0, 0.25, 0.5, 0.75,
 982 or 1) will result in an an additional control point being used to smooth the iso-
 983 bars. Thus, entering additional isopleth values results not only in more isopleth
 984 outputs but also in “smoother” (i.e., more well constrained) isobars.
 985 `smooth_isobars` and `smooth_isopleths`: The default value for both of these ar-
 986 guments is True, in which case polynomials will be fit to the computed data points.
 987 `print_status`: The default value is True. If True, the progress of the calculations
 988 will be printed to the terminal.

989 **Calculated outputs** The function returns two pandas DataFrames: the first has iso-
 990 bar data, and the second has isopleth data. Columns in the isobar dataframe are
 991 ‘Pressure’, ‘H2O_{melt}’, and ‘CO2_{melt}’, corresponding to pressure in bars and dis-
 992 solved H₂O and CO₂ in the melt in wt%. Columns in the isopleth dataframe are
 993 ‘XH₂O_{fl}’, ‘H₂O_{liq}’, and ‘CO₂_{liq}’, corresponding to XH₂O^{fluid} and dissolved
 994 H₂O and CO₂ in the melt in wt%.

```
"""Define all variables to be passed to the function for calculating
→ isobars and isopleths"""
```

```
"""Define the temperature in degrees C"""
temperature = 1200.0
```

```
"""Define a list of pressures in bars:"""
pressures = [1000.0, 2000.0, 3000.0]
```

995 Next, the H₂O and CO₂ dissolved in the melt at saturation is calculated at the spec-
 996 ified temperature and over the range of specified pressures. Note that, because this func-
 997 tion calculates two things (isobars and isopleths), two variable names must be given (be-
 998 low, “isobars, isopleths”). This calculation can be quite slow, and so it is recommended
 999 to set `print_status` to True.

```
isobars, isopleths = v.calculate_isobars_and_isopleths(sample=sample_10,
temperature=temperature,
pressure_list=pressures,
→ isopleth_list=[0.25,0.5,0.75]).result
```

```
Calculating isobar at 1000.0 bars
done.
```

```
Calculating isobar at 2000.0 bars
done.
```

```
Calculating isobar at 3000.0 bars
done.
```

```
Done!
```

1000 *0.3.9 Calculating degassing paths*

1001 A degassing path is a series of volatile concentrations both in the melt and fluid
 1002 that a magma will follow during decompression. In the calculation, the saturation pres-
 1003 sure is computed, and then the system is equilibrated along a trajectory of decreasing
 1004 pressure values at discrete steps. The default number of steps to calculate is 50, but this
 1005 can be defined by the user by setting the argument `steps` to any integer value. A de-
 1006 tailed explanation of how non-MagmaSat models handle the calculation of mixed-fluid

1007 composition can be found in the supplement (Supplementary Text S2). If so desired, this
 1008 calculation can be performed for any initial pressure, but the default is the saturation
 1009 pressure. If a pressure is specified that is above the saturation pressure, the calculation
 1010 will simply proceed from the saturation pressure, since the magma cannot degas until
 1011 it reaches saturation.

1012 Completely open-system, completely closed-system or partially open-system degassing
 1013 paths can be calculated by specifying what proportion of the fluid to fractionate. The
 1014 fluid fractionation value can range between 0 (closed-system: no fluid is removed, all is
 1015 retained at each pressure step) and 1 (open-system: all fluid is removed, none is retained
 1016 at each pressure step). Closed and partially open-system runs allow the user to specify
 1017 the initial presence of exsolved fluid that is in equilibrium with the melt at the starting
 1018 pressure.

1019 **Method structure** *Only single-sample calculations.* `calculate_degassing_path(sample,`
 1020 `temperature, pressure="saturation", fractionate_vapor=0.0, init_vapor=0.0,`
 1021 `steps=50, model="MagmaSat").result`

1022 **Standard inputs** `sample, temperature, model` (see Section 0.3.1).

1023 **Unique optional inputs** `pressure`: The pressure at which to begin the degassing cal-
 1024 culations, in bars. Default value is 'saturation', which runs the calculation with
 1025 the initial pressure at the saturation pressure. If a pressure greater than the sat-
 1026 uration pressure is input, the calculation will start at saturation, since this is the
 1027 first pressure at which any degassing will occur.

1028 `fractionate_vapor`: Proportion of vapor removed at each pressure step. Default
 1029 value is 0.0 (completely closed-system degassing). Specifies the type of calcula-
 1030 tion performed, either closed system (0.0) or open system (1.0) degassing. If any
 1031 value between <1.0 is chosen, user can also specify the 'init_vapor' argument (see
 1032 below). A value in between 0 and 1 will remove that proportion of vapor at each
 1033 step. For example, for a value of 0.2, the calculation will remove 20% of the vapor
 1034 and retain 80% of the vapor at each pressure step.

1035 `init_vapor`: Default value is 0.0. Specifies the amount of vapor (in wt%) coex-
 1036 isting with the melt before degassing.

1037 `steps`: Default value is 50. Specifies the number of steps in pressure space at which
 1038 to calculate dissolved volatile concentrations.

1039 **Calculated outputs** The function returns a pandas DataFrame with columns as: 'Pres-
 1040 sure_bars', 'H2O_liq' and 'CO2_liq' (the concentration of H₂O and CO₂ in the
 1041 melt, in wt%), 'XH2O_fl' and 'XCO2_fl' (the composition of the H₂O-CO₂ fluid,
 1042 in mol fraction), and 'FluidProportion_wt' (the proportion of fluid in the fluid-
 1043 melt system, in wt%).

1044 **Note:** The following two cells can take up to a minute to execute.

```
temp = 1200 #temperature in °C
```

```
"""Calculate open, closed, and closed + 2 wt% initial vapor"""
closed_df = v.calculate_degassing_path(sample=sample_10,
  ↪ temperature=temp).result
open_df = v.calculate_degassing_path(sample=sample_10, temperature=temp,
  ↪ fractionate_vapor=1.0).result
half_df = v.calculate_degassing_path(sample=sample_10, temperature=temp,
  ↪ fractionate_vapor=0.5).result
exsolved_df = v.calculate_degassing_path(sample=sample_10,
  ↪ temperature=temp, init_vapor=2.0).result
```

```

1045 """Calculate closed -system degassing starting from a pressure of 2000
1046 → bars"""
1047 start2000_df = v.calculate_degassing_path(sample=sample_10,
1048 → temperature=temp, pressure=2000.0).result

```

```

1049 [=====] 100% Calculating degassing path...
1050 [=====] 100% Calculating degassing path...
1051 [=====] 100% Calculating degassing path...
1052 [=====] 100% Calculating degassing path...
1053 [=====] 100% Calculating degassing path...

```

1045 0.3.10 Plotting

1046 After calculating isobars, isopleths, and degassing paths, any or all of these may
 1047 be plotted in an H₂O versus CO₂ plot with one simple function call. The plot will be
 1048 printed directly in the notebook or, if the code is run as script in a command line, the
 1049 plot will appear in its own window, at which point it can be saved as an image file. VESI-
 1050 cal's `plot` function takes in lists of pandas DataFrames with calculated isobar, isopleth,
 1051 and degassing path information (e.g., output from `calculate_isobars_and_isopleths()`
 1052 or `calculate_degassing_path()`) and plots data as isobars (lines of constant pressure),
 1053 isopleths (lines of constant fluid composition), and degassing paths (lines indicating the
 1054 concentrations of H₂O and CO₂ in a melt equilibrated along a path of decreasing pres-
 1055 sure).

1056 Labels can be assigned to isobars, isopleths, and/or degassing paths separately. Any
 1057 or all of these data can be passed to the `plot` function. Multiple sets of plottable data
 1058 can be passed. For example, isobars calculated with two different models can be passed
 1059 to the `isobars` argument as a list.

1060 VESICAL's plotting function is entirely based on python's matplotlib library, which
 1061 comes standard with many installations of python. With matplotlib, users can create
 1062 a large variety of plots (note that direct matplotlib functionality is used to create cus-
 1063 tom plots in several of this manuscript's supplementary Jupyter notebooks), and users
 1064 should refer to the matplotlib documentation (<https://matplotlib.org/3.2.1/index.html>)
 1065 if more complex plotting is desired. If preferred, VESICAL outputs can be saved to an Ex-
 1066 cel or CSV file (see Section 0.3.12), and plotting can be done in any plotting program
 1067 desired (e.g., MS Excel).

1068 The function returns both fig and axes matplotlib objects, which can be further edited
 1069 by the user or plotted directly. Following matplotlib convention, the results of `plot()`
 1070 should be saved to objects such as `fig`, `ax` as:

```
1071 fig, ax = v.plot([options])
```

1072 where [options] represents any optional inputs as defined here. Variables `fig` and
 1073 `ax` can then be edited further using matplotlib tools. For example, the user might wish
 1074 to set the minimum x-axis value to 0.5 as:

```
1075 ax.set_xlim(left=0.5)
```

1076 In Jupyter Notebook, a plot is automatically shown, but in the command line, the
 1077 plot will only display after executing `v.show()`.

```

1078 Method structure plot(isobars=None, isopleths=None, degassing_paths=None,
1079 custom_H2O=None, custom_CO2=None, isobar_labels=None, isopleth_labels=None,
1080 degassing_path_labels=None, custom_labels=None, custom_colors="VESICAL",
1081 custom_symbols=None, markersize=10, save_fig=False, extend_isobars_to_zero=True,
1082 smooth_isobars=False, smooth_isopleths=False)

```

1083 **Optional inputs** `isobars`: DataFrame object containing isobar information as calcu-
1084 lated by `calculate_isobars_and_isopleths()`. Or a list of DataFrame objects.
1085 `isopleths`: DataFrame object containing isopleth information as calculated by
1086 `calculate_isobars_and_isopleths()`. Or a list of DataFrame objects.
1087 `degassing_paths`: List of DataFrames with degassing information as generated
1088 by `calculate_degassing_path()`.
1089 `custom_H2O`: List of floats or array-like shapes of H₂O concentration values to plot
1090 as points. For example `myfile.data['H2O']` is one array-like shape (here, pan-
1091 das.Series) of H₂O values. Must be passed with `custom_CO2` and must be same
1092 length as `custom_CO2`.
1093 `custom_CO2`: List of floats or array-like shapes of CO₂ values to plot as points. For
1094 example `myfile.data['CO2']` is one array-like shape of CO₂ values. Must be passed
1095 with `custom_H2O` and must be same length as `custom_H2O`.
1096 `isobar_labels`: Labels for the plot legend. Default is None, in which case each
1097 plotted line will be given the generic legend name of “Isobars n”, with n referring
1098 to the nth isobars passed. Isobar pressure is given in parentheses. The user can
1099 pass their own labels as a list of strings. If more than one set of isobars is passed,
1100 the labels should refer to each set of isobars, not each pressure.
1101 `isopleth_labels`: Labels for the plot legend. Default is None, in which case each
1102 plotted isopleth will be given the generic legend name of “Isopleth n”, with n re-
1103 ferring to the nth isopleths passed. Isopleth XH₂O values are given in parenthe-
1104 ses. The user can pass their own labels as a list of strings. If more than one set
1105 of isopleths is passed, the labels should refer to each set of isopleths, not each XH₂O
1106 value.
1107 `degassing_path_labels`: Labels for the plot legend. Default is None, in which
1108 case each plotted line will be given the generic legend name of “Pathn”, with n
1109 referring to the nth degassing path passed. The user can pass their own labels as
1110 a list of strings.
1111 `custom_labels`: Labels for the plot legend. Default is None, in which case each
1112 group of custom points will be given the generic legend name of “Customn”, with
1113 n referring to the nth degassing path passed. The user can pass their own labels
1114 as a list of strings.
1115 `custom_colors` and `custom_symbols`: Custom colors and symbol shapes can be
1116 specified for (`custom_H2O`, `custom_CO2`) points. A list of color values or symbol
1117 types readable by Matplotlib (see Matplotlib documentation) can be entered. The
1118 length of this list must be equal to the lengths of `custom_H2O` and `custom_CO2`.
1119 If nothing is specified for `custom_colors`, VESICAL’s default colors will be used. If
1120 nothing is specified for `custom_symbols`, all points will be plotted as filled circles.
1121 `markersize`: The size of the symbols can be specified here. If not specified, the
1122 default value is marker size 10.
1123 `save_fig`: Default value is False, in which case the figure will not be saved. If a
1124 string is passed, the figure will be saved with the string as the filename. The string
1125 must include the file extension.

1126 **Advanced inputs: Most users will not need to use these inputs.** `extend_isobars_to_zero`:
1127 If set to True (the default), isobars will be extended to the plot axes, which are
1128 at x=0 and y=0, even if there is a finite solubility at zero partial pressure.
1129 `smooth_isobars` and `smooth_isopleths`: If set to True, isobar or isopleth data
1130 will be fit to a polynomial and plotted. If set to False (the default), the raw in-
1131 put data will be plotted. Note that MagmaSat `calculate_isobars_and_isopleths()`
1132 calculations return already “smoothed” data (that is, the raw data are fit to poly-
1133 nomials before being returned). Raw “unsmoothed” data can be returned by Mag-
1134 maSat `calculate_isobars_and_isopleths()` (see documentation on this method).

1135 **Calculated outputs** The function returns fig and axes matplotlib objects defining a
1136 plot with x-axis as H₂O wt% in the melt and y-axis as CO₂ wt% in the melt. Iso-
1137 bars, or lines of constant pressure at which the sample magma composition is sat-

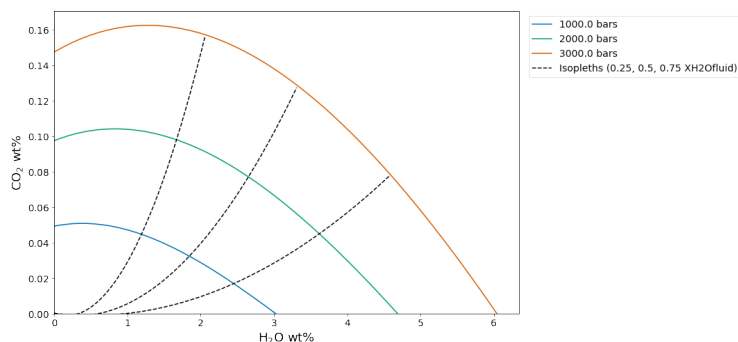
1138 urated, and isopleths, or lines of constant fluid composition at which the sample
 1139 magma composition is saturated, are plotted if passed. Degassing paths, or the
 1140 concentration of dissolved H_2O and CO_2 in a melt equilibrated along a path of
 1141 decreasing pressure, is plotted if passed.

1142 *0.3.10.1 A simple example: Isobars and isopleths* Here we plot the isobars at 1,000,
 1143 2,000, and 3,000 bars and isopleths at 0.25, 0.5, and 0.75 $\text{XH}_2\text{O}^{\text{fluid}}$ calculated for sam-
 1144 ple ‘10*’ at 1,200 °C in Section 0.3.8 onto one plot.

1145

[H]

```
fig, ax = v.plot(isobars=isobars, isopleths=isopleths)
v.show()
```



1146

1147 `fig`, `ax = v.plot(isobars=isobars, isopleths=isopleths)`
 1148 calculated for the sample, temperature, pressures, $\text{XH}_2\text{O}^{\text{fluid}}$ values, and with the model
 1149 as defined in Section 0.3.8. Manuscript default values are sample 10* at a 1,200 °C with
 1150 isobars at 1,000, 2,000, and 3,000 bars, isopleths at $\text{XH}_2\text{O}^{\text{fluid}} = 0, 0.25, 0.5, 0.75,$ and
 1151 1 calculated with MagmaSat

1152

1153 When plotting isobars and isopleths via MagmaSat, the values calculated by `calculate_isobars_and_isop`
 1154 are used to calculate polynomial fits using Numpy’s `polyfit`. These polynomial fits, not
 1155 the raw calculated data, are what have been plotted above. This method of fitting poly-
 1156 nomial curves to these data is common in the literature Newman & Lowenstern (2002);
 1157 Iacono-Marziano et al. (2012); Iacovino et al. (2013) and is likely a very close approx-
 1158 imation of the true saturation surface. Non-MagmaSat models do not calculate polynomi-
 al fits by default, but this can be done by passing `smooth_isobars=True` and `smooth_isopleths=True`
 to `plot()`.

1159

1160 A user may wish to apply custom formatting to the plot, in which case the poly-
 1161 nomial fits can be calculated and returned as a pandas DataFrame, which the user can
 1162 then plot up manually using Matplotlib, MS Excel, or some other preferred method. To
 1163 calculate polynomial fits to isobar and isopleth data, isobars and isopleths can be passed
 1164 to `smooth_isobars_and_isopleths()`. For this advanced case, we refer the reader to
 the documentation.

1165

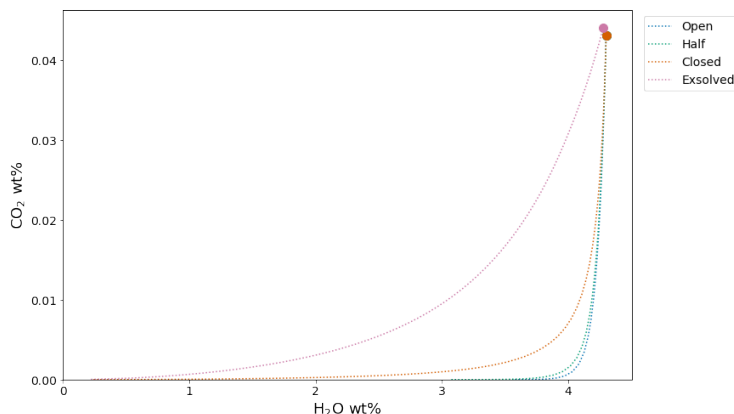
1166 *0.3.10.2 A simple example: Degassing paths* Here we plot all four degassing paths
 1167 calculated for sample ‘10*’ at 1,200 °C in Section 0.3.9 onto one plot. We designate la-
 bels of “Open”, “Half”, “Closed”, and “Exsolved” for the legend.

1168

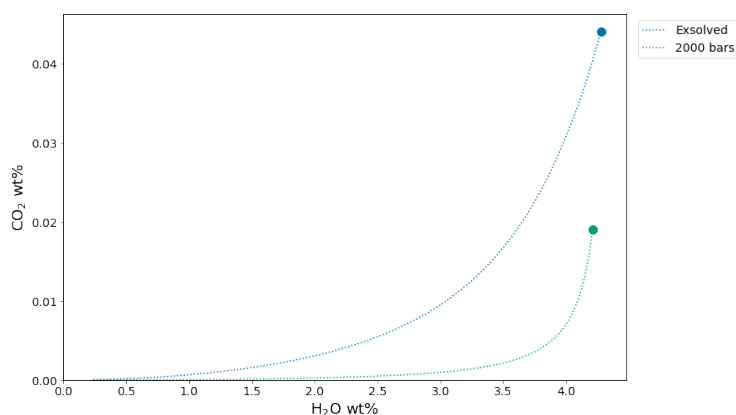
[H]

```
fig, ax = v.plot(degassing_paths=[open_df, half_df, closed_df,
↪  exsolved_df],
                degassing_path_labels=["Open", "Half", "Closed", "Exsolved"])
fig, ax = v.plot(degassing_paths=[exsolved_df, start2000_df],
```

```
degassing_path_labels=["Exsolved", "2000 bars"])
v.show()
```



1169



1170

1171 [] Degassing paths calculated for the sample, temperature, degassing style, initial exsolved fluid wt%, starting pressure, and model as designated in Section 0.3.9. Default manuscript values are sample 10* at 1,200 °C. “Open”, “Half”, and “Closed” curves in (a) represent open-system, partially open-system (50% fractionated fluid), and closed-system degassing paths, respectively, starting at the saturation pressure. The “Exsolved” curve in (b) represents closed-system degassing with an initial exsolved fluid wt% = 2.0. The “2000” curve in (b) represents closed-system degassing calculated starting at a pressure of 2,000 bars.

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

0.3.10.3 Plotting multiple calculations One of the major advantages to VESIcal over any other modeling tool is the ability to quickly calculate and plot multiple calculations. VESIcal’s `plot()` function is built on top of the popular Matplotlib python library and is designed to work with any VESIcal generated data. It can automatically plot and label one or multiple calculations. In addition, it can plot, as a scatter plot, any x-y points. The plot function always generates plots with H₂O on the x-axis and CO₂ on the y-axis. `scatterplot()` will take in and plot any x-y data with custom x- and y-axis labels. Generating other commonly used petrologic plots (e.g. Harker style diagrams) is already possible with Matplotlib, and so VESIcal does not duplicate this functionality, however this may be added in future updates.

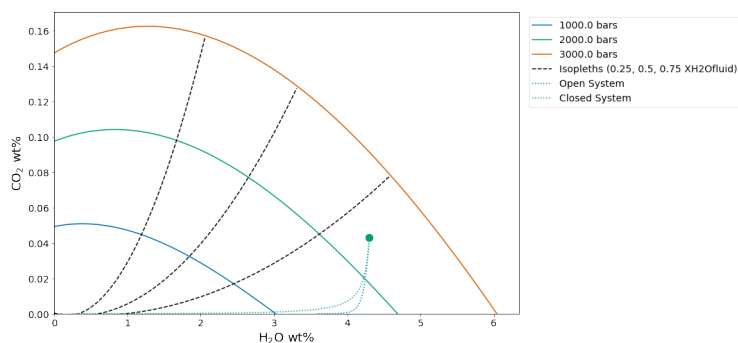
It may be tempting to plot multiple calculations on multiple samples and compare them, however we strongly caution against plotting data that do not correspond. For example, isobars and isopleths are calculated isothermally. If degassing paths are also plotted, the user should ensure that the degassing paths were calculated at the same temperature as the isobars and isopleths.

1194 *Isobars, isopleths, and degassing paths* In this example we will use data imported
 1195 in Section 0.3.4 and calculations performed in Sections Section 0.3.7 and Section 0.3.8.
 1196 Of course, all of the data calculated with VESICAL can be exported to an Excel or CSV
 1197 file for manipulation and plotting as desired. However, some examples of plotting that
 1198 can be done within this notebook or in a python script are shown below. Here we plot:

- 1199 • Isobars calculated at 1200 °C and pressures of 1,000, 2,000, and 3,000 bars for sam-
 1200 ple 10*
- 1201 • Isopleths calculated at 1200 °C and $\text{XH}_2\text{O}^{\text{fluid}}$ values of 0, 0.25, 0.5, 0.75, and 1
 1202 for sample 10*
- 1203 • An open-system degassing path for sample 10*
- 1204 • A closed-system degassing path for sample 10*

1205 [H]

```
fig, ax = v.plot(isobars=isobars,
                 isopleths=isopleths,
                 degassing_paths=[open_df, closed_df],
                 degassing_path_labels=["Open System", "Closed System"])
v.show()
```



1206 [Example of plotting
 1207 multiple calculations on one plot. Isobars and isopleths as defined in Section 0.3.8 and
 1208 shown in Section 0.3.10.1 and degassing curves as defined in Section 0.3.9 and shown in
 1209 Section 0.3.10.2. Default manuscript values are for sample 10* at 1,200 °C with isobars
 1210 at 1,000, 2,000, and 3,000 bars, isopleths at $\text{XH}_2\text{O}^{\text{fluid}}$ values of 0, 0.25, 0.5, 0.75, and
 1211 1 with an open-system and a closed-system degassing path.

1212 *Isobars, isopleths, and degassing paths for multiple samples* First, we will calcu-
 1213 late some new data for two different samples: a basanite Iacovino et al. (2016) and a rhy-
 1214 olite Myers et al. (2019). For both samples we will calculate and then plot:

- 1215 • Isobars and isopleths at 1100 °C, pressures of 1,000 and 2,000 bars and fluid com-
 1216 positions of $\text{XH}_2\text{O}^{\text{fluid}}$ of 0.25, 0.5, and 0.75
- 1217 • Closed-system degassing paths at 1100 °C

```
basanite_sample = myfile.get_sample_composition('KI -07',
↪ asSampleClass=True)
rhyolite_sample = myfile.get_sample_composition('samp. P1968a',
↪ asSampleClass=True)

basanite_isobars, basanite_isopleths =
↪ v.calculate_isobars_and_isopleths(sample=basanite_sample,
↪ temperature=1100,
```

```

→ pressure_list=[1000,
→ 2000],
→ isopleth_list=[0.2,

rhyolite_isobars, rhyolite_isopleths =
→ v.calculate_isobars_and_isopleths(sample=rhyolite_sample,

→ temperature=1100,
→ pressure_list=[1000,
→ 2000],
→ isopleth_list=[0.2,

basanite_degassing_path =
→ v.calculate_degassing_path(sample=basanite_sample,

→ temperature=1100).result

rhyolite_degassing_path =
→ v.calculate_degassing_path(sample=rhyolite_sample,

→ temperature=1100).result

Calculating isobar at 1000 bars
done.
Calculating isobar at 2000 bars
done.
Done!
Calculating isobar at 1000 bars
done.
Calculating isobar at 2000 bars
done.
Done!
[=====] 100% Calculating degassing path...
[=====] 100% Calculating degassing path...

```

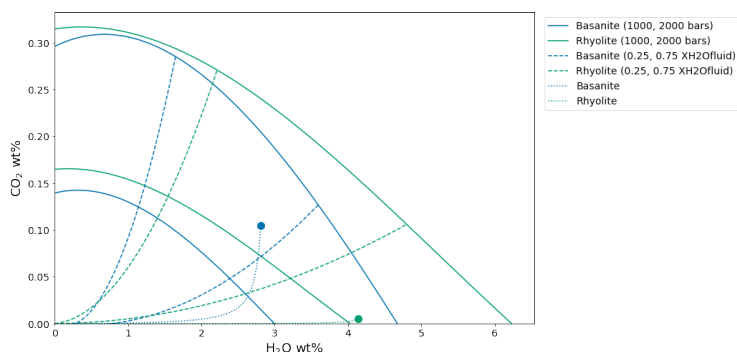
1218

[H]

```

fig, ax = v.plot(isobars=[basanite_isobars, rhyolite_isobars],
                 isopleths=[basanite_isopleths, rhyolite_isopleths],
                 degassing_paths=[basanite_degassing_path,
→ rhyolite_degassing_path],
                 isobar_labels=["Basanite", "Rhyolite"],
                 isopleth_labels=["Basanite", "Rhyolite"],
                 degassing_path_labels=["Basanite", "Rhyolite"])
v.show()

```



1219

1220

1221

1222

1223

1224

1225

[] Example of plotting

multiple calculations from multiple samples on the same plot. Note that the colors are automatically set to correspond to each sample for all plotted items (here, isobars, isopleths, and degassing paths). Samples, pressures, temperatures, $X_{H_2O}^{fluid}$ values, and degassing path styles are defined above in this section. Manuscript default values are for a basanite (sample KI-07) and a rhyolite (sample `samp. P1968a`) at 1,100 °C, 1,000 and 2,000 bars, and $X_{H_2O}^{fluid} = 0.25$ and 0.75 and closed-system degassing.

1226

0.3.11 Model hybridization (Advanced)

1227

1228

1229

1230

1231

1232

1233

1234

One of the advantages of implementing the solubility models in a generic python module is the flexibility this affords the user in changing the way solubility models are defined and used. In particular, the structure allows any combination of pure-fluid models to be used together in modeling mixed-fluids, and fugacity or activity models can be quickly changed without modifying code. This allows advanced users to see how changing a fugacity or activity model implemented in any particular solubility model would affect model results. Instructions for hybridizing models can be found in Supplemental Jupyter notebook S10.

1235

0.3.12 Exporting data

1236

1237

1238

1239

Once batch calculations have been performed, they can be exported to an Excel or CSV file with the `save_excel()` and `save_csv()` commands. These operations require that the user define a filename (what to name your new file) and a list of the calculation results to save to this file or files.

1240

1241

1242

1243

1244

1245

1246

Note that this requires that calculations have been assigned to variable names, which has been done in all of the given examples. For example, to calculate saturation pressures of an imported file saved to the variable 'myfile' and simply print the output, the user can type `myfile.calculate_saturation_pressures([options])`, where '[options]' are the required and optional inputs. However, to save this result to a variable (e.g., called 'my_satPs') so that it can be accessed later, the correct python syntax would be `my_satPs = myfile.calculate_saturation_pressures([options])`.

1247

1248

1249

1250

1251

1252

Multiple calculations can be saved at once. If saving to an Excel file, each calculation is saved as its own sheet within a single file. If desired, the user can define the names of each of these sheets. If not specified, the sheets will be named 'Original_User_Data', which contains the original input data, and then 'CalcN' where N is the nth calculation in a list of calculations. If saving multiple calculations to a CSV file, each calculation will be saved to its own CSV file, and a file name for each of these is required.

1253

1254

1255

1256

1257

Advanced users note that the `calculations` argument takes in any pandas DataFrame object, meaning this functionality is not limited to VESICAL's prescribed outputs. The `save_excel()` and `save_csv()` methods use the pandas `to_excel` and `to_csv` methods, however not all options are implemented here. If saving to a CSV file, any arguments that can be passed to pandas `to_csv` method may be passed to VESICAL's `save_csv()`.

1258 **Method structures** `save_excel(filename, calculations, sheet_name=None)``save_csv(filenamees,`
1259 `calculations)`
1260 **save_excel() Required inputs** `filename` (Excel): Name of the file to create. The
1261 extension (.xlsx) should be included along with the name itself, all in quotes (e.g.,
1262 `filename='myfile.xlsx')`.`calculations`: A list of variables containing calcu-
1263 lated outputs from any of the core BatchFile functions: `calculate_dissolved_volatiles()`,
1264 `calculate_equilibrium_fluid_comp()`, and `calculate_saturation_pressure()`.
1265 This must be passed as a list type variable, even if only one calculation is given.
1266 This is done by enclosing the variable in square brackets (e.g., `calculations=[my_calculation]`).
1267 **save_excel() Optional inputs** `sheet_name`: The default value is None, in which case
1268 sheets will be saved as 'Original_User_data' (the data input by the user) followed
1269 by 'CalcN' where N is the nth calculation in `calculations`. Otherwise, a list of
1270 names for the sheets can be passed, with the names in quotes (e.g. `sheet_name=['SaturationPressures']`)
1271 'Original_User_data' will always be saved as the first sheet.
1272 **save_csv() Required inputs** `filenames` (CSV): Name of the file or files to create.
1273 The extension (.csv) should be included. If more than one filename is passed, it
1274 should be passed as a list. This is done by enclosing the filenames in square brack-
1275 ets (e.g., `filenames=["file1.csv", "file2.csv"]`).`calculations`: same as for
1276 `save_excel()`. Must be same length as `filenames`.
1277 **Calculated outputs** An Excel or CSV file or files will be saved to the active directory
1278 (i.e., the same folder as this manuscript notebook or wherever the code is being
1279 used).

1280 Here we save five of the calculations performed on an imported data file earlier in
1281 this manuscript. The original user-input data are stored in the BatchFile object 'myfile'.
1282 In the following line we use the method `save_excel()` to save the original data and a
1283 list of calculations given by the `calculations` argument to an Excel file.

```
myfile.save_excel(filename='testsave.xlsx',
                  calculations=[dissolved, eqfluid, eqfluid_wtemps,
                               ↪ satPs, satPs_wtemps],
                  sheet_names=['dissolved', 'eqfluid',
                               ↪ 'eqfluid_wtemps', 'SaturationPs',
                               ↪ 'SatPs_wtemps'])
```

Saved testsave.xlsx

1284 *0.3.12.1 Saving data for re-import into VESICAL* In many cases, it may be prefer-
1285 able to compute large amounts of data using VESICAL and then reimport them, either
1286 to perform more analysis or to plot the data. Likewise, a user may wish to compute data
1287 in VESICAL and then send the results to a colleague, who can then re-import that data
1288 into VESICAL directly. For this case, we suggest using python's pickle package (<https://wiki.python.org/moin/UsingPickler>)
1289 Any python object, such as the results of a VESICAL calculation, can be "pickled" or saved
1290 as a python-readable file. To use pickle, users must first import the pickle module, then
1291 "dump" the desired contents to a pickle file. The pickled data can be accessed by "load-
1292 ing" the pickled file.

1293 Below we pickle our computed dissolved volatile concentrations by dumping our
1294 variable `dissolved` to a pickle file that we name "dissolved.p".

```
import pickle

pickle.dump(dissolved, open("dissolved.p", "wb"))
```

1295 In another python file or terminal session, `dissolved` can be loaded back in via:

```
import pickle
```

```
dissolved = pickle.load(open("dissolved.p", "rb"))
```

1296

0.4 Discussion and Applications

1297

0.4.1 Compositional Variation Within Datasets and Best Practices

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

It has been clearly shown that the composition of a melt plays a strong role in determining the solubility of H₂O and CO₂ in magmas Papale et al. (2006); Moore (2008); Ghiorso & Gualda (2015); Wieser et al. (2021). Thus, compositional variance must be accounted for in any study examining solubility in multiple samples. A key use case where VESICAL can facilitate the adoption of this practice is in melt inclusion (MI) studies; specifically, where a single suite of MI with multiple melt compositions is examined using solubility models to interrogate magmatic degassing processes. Prior to the availability of VESICAL, the difficulty associated with performing multiple model calculations on multiple samples resulted in very few studies accounting for any compositional variance within their datasets. Indeed, until now, it has been difficult to even assess whether the potentially minimal compositional variance within a suite of melt inclusions from a single volcanic eruption would have any measurable effect on solubilities calculated for different MI.

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

Using VESICAL, we can address the question: what is the quantitative effect of compositional variation within a single suite of melt inclusions upon calculated melt inclusion saturation pressures? And, how does this affect conclusions that might be drawn regarding volcanic degassing and eruptive processes? To investigate this, we use a dataset of basaltic melt inclusions from Cerro Negro volcano, Nicaragua Roggensack (2001). The compositional variation of these MI, while relatively restricted, results in quite variable mixed-fluid solubilities from sample to sample. To determine the end-member compositions within the dataset corresponding to the samples with the maximum and minimum combined H₂O-CO₂ solubilities, isobars were computed at 1200 °C and 3,000 bars for all samples using the MagmaSat model in VESICAL. Maximum and minimum samples were taken as the isobar curves with the smallest and largest integral (area under the curve). We refer to this value as the “integrated mixed-volatile solubility” value, IMS, in units of concentration squared. The samples that produced maximum and minimum integrated solubilities are shown in Figures Figure 5 and Figure 6 in blue and green, respectively (sample 41b*, IMS=0.81 and 36a*, IMS=0.66 wt%² at 3,000 bars). A composition representing the average of all MI in the dataset is shown in orange (“Average Sample”, IMS=0.70 wt%² at 3,000 bars). A Jupyter notebook to reproduce these calculations is provided in the supplement (Supplementary Jupyter Notebook S8).

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

At all pressures, the integrated mixed-volatile solubility across the Cerro Negro dataset varies as much as 10% relative. For these MI, this results in as much as 11.5% relative error in the calculation of saturation pressures (average error for the entire dataset of 6.8% relative). It is noteworthy that this error is not systematic either in terms of absolute value or sign. For example, when calculated using their own compositions, saturation pressures for maximum and minimum samples 41b* and 36a* are 3050 and 3090 bars, respectively. But, saturation pressures calculated for both of these MI using the dataset’s average composition are 3020 and 3250 bars, respectively. That is an error of -30 and +160 bars or -1% and +5% respectively. Errors in these calculations, thus, may be quite small. But, in any case, removing this error completely is a simple task using VESICAL, and so we recommend that studies adopt the practice of calculating volatile solubilities (and associated values) in melts using the composition unique to each melt investigated.

1342

1343

Even in cases where solubility values (e.g., saturation pressures) are not calculated, the error highlighted above plagues any isobar diagram over which multiple melt com-

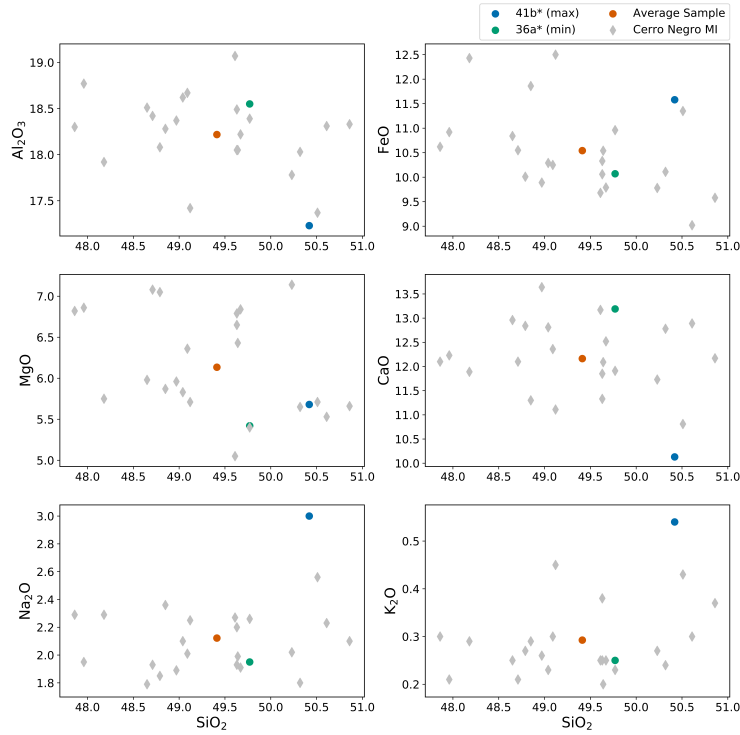


Figure 5. Harker style diagrams illustrating the compositional range of MIs from Cerro Negro volcano from Roggensack (2001). The “Average Sample” plotted as an orange dot represents a fictitious sample, calculated as the average of all MIs in the dataset. Sample 41b* and 36a* are the names of samples that produced isobars with maximum and minimum area under the curve, respectively (see text). Gray diamonds are all other data in the dataset.

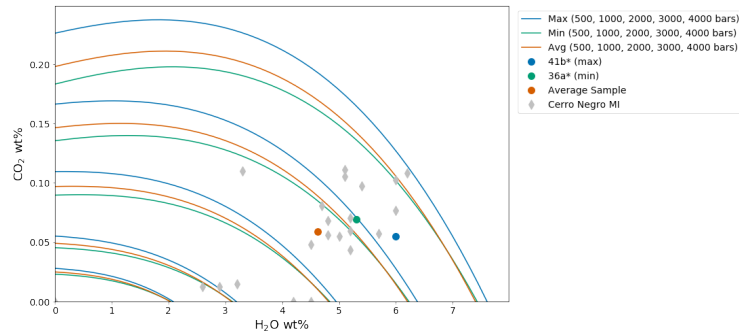


Figure 6. H₂O-CO₂ diagram with isobars for MI from Cerro Negro volcano Roggensack (2001) computed by VESICAL using MagmaSat at 1200 °C, pressures of 500, 1000, 2000, 3000, and 4,000 bars. Curves shown are polynomials fitted to data computed by VESICAL. Blue and green curves correspond to samples 41b* and 36a*, which produced isobars with maximum and minimum area under the curve, respectively. Orange isobars were those computed for a fictitious sample representing the average composition of the MI dataset. Gray diamonds are all other data in the dataset.

positions are plotted (e.g., Figure 6). Alternative plots to the commonly used H₂O-CO₂ diagram are shown in Figure 7, in which the same dataset is plotted in terms of computed saturation pressure (at 1200 °C calculated with VESICAL using MagmaSat) versus dissolved H₂O, dissolved CO₂, and fluid composition (as XH₂O^{fluid} calculated with VESICAL using MagmaSat). These plots avoid the issues discussed above as they are compositionally independent, since the saturation pressure is calculated individually for each sample composition. Degassing trends are more accurately represented; H₂O and CO₂ concentrations lie along expected degassing trends with much less scatter than the H₂O-CO₂ plot. We can also see from this figure that the fluid composition during this eruption at Cerro Negro remained relatively constant at XH₂O^{fluid} ~0.8 from reservoir to surface, suggesting a scenario approaching closed-system degassing (i.e., melt volatile concentrations are buffered by the co-existing fluid composition). This is discussed in more detail in the companion paper Wieser et al. (2021).

0.4.2 Model Comparisons

One of the possible workflows enabled through VESICAL is the ability to compute and compare (numerically and graphically) results from several models at once. To illustrate this point, we will take two single samples within the calibrated compositional range of several models, calculate isobars at multiple pressures, and plot the results. This is a common way to compare the solubility surface computed by different models for a single melt composition, and it is particularly useful since it quickly highlights the significant variation that exists between published models. The results of this exercise are shown here, and a Jupyter notebook to reproduce the code and calibration checks is available in the Supplement (Supplementary Jupyter Notebook S9).

We use a fictitious alkali basalt that we name “alkbasalt” and a fictitious rhyolite whose compositions are given in Table 0.4.2. The use of VESICAL’s `calib_plot()` function (see supplement) illustrates that the composition of the alkali basalt is within the compositional calibration ranges of four mixed-fluid solubility models: MagmaSat, Iacono-Marziano, Dixon, and ShishkinaIdealMixing. The rhyolite is within the ranges of MagmaSat and Liu. Isobars were calculated with these models at 1200 °C for alkbasalt and 800 °C for rhyolite and pressures of 500, 1,000, and 2,000 bars, using the below code:

```
[H] []Melt compositions used for modeling

model_comps = v.BatchFile("tables/Table_Model_Comps.xlsx")
model_comps.data

SiO2 TiO2 Al2O3 Fe2O3 FeO MnO MgO CaO Na2O K2O \
Alkali Basalt 49.00 1.27 19.7 3.74 5.33 0.17 4.82 8.85 4.23
↪ 1.00
Rhyolite 77.19 0.06 12.8 0.00 0.94 0.00 0.03 0.53 3.98
↪ 4.65

                P2O5 H2O CO2 Cr2O3 NiO CoO
Alkali Basalt 0.37 4.51 0.25 0.0 0.0 0.0
Rhyolite 0.00 0.26 0.05 0.0 0.0 0.0

alkbasalt = model_comps.get_sample_composition("Alkali Basalt",
↪ asSampleClass=True)
rhyolite = model_comps.get_sample_composition("Rhyolite",
↪ asSampleClass=True)

alkbasalt_isobars, alkbasalt_isopleths =
↪ v.calculate_isobars_and_isopleths(sample=alkbasalt,
```

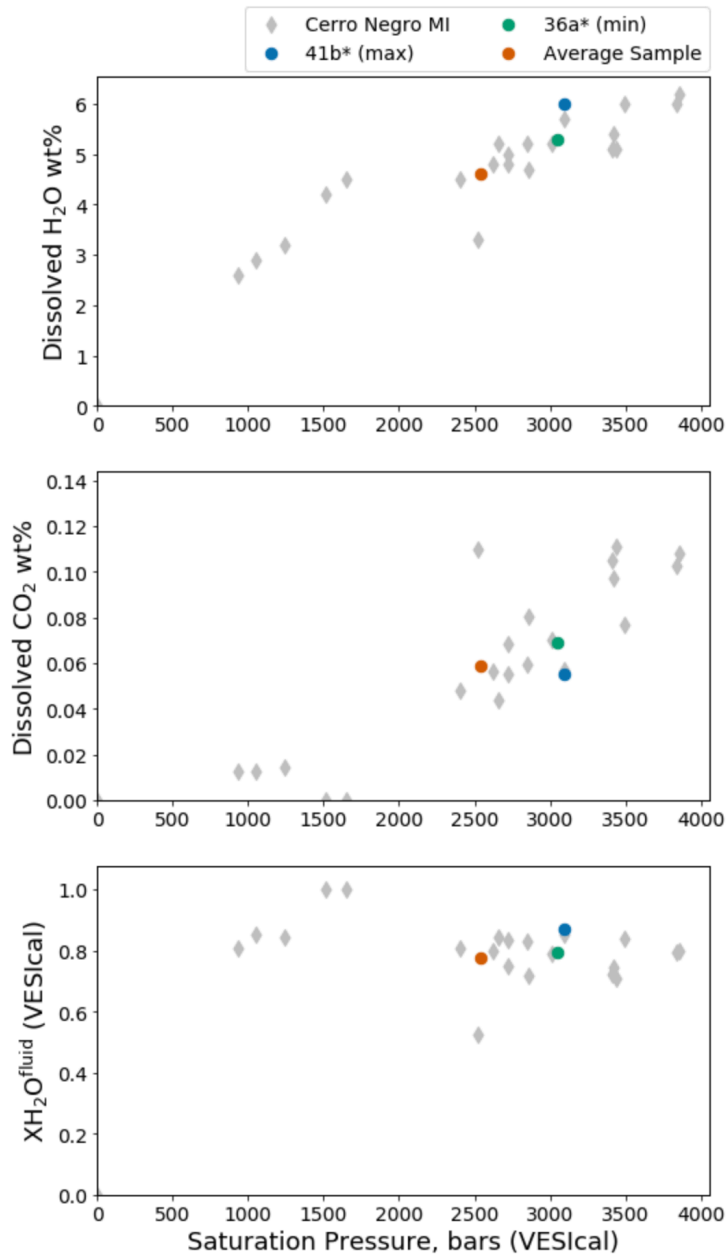


Figure 7. Saturation pressure at 1200 °C calculated using VESIcal with MagmaSat versus measured dissolved H₂O and CO₂ concentrations and calculated fluid composition in Cerro Negro melt inclusions. These plots meaningfully illustrate degassing processes while avoiding issues associated with commonly used H₂O-CO₂ diagrams, which occur with even minor compositional variation within a given dataset.

```
→ temperature=1200
→ pressure_list=[500,
→ 1000,
→ 2000],

→ isopleth_list=[0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]

→ print_status=True

rhyolite_isobars, rhyolite_isopleths =
→ v.calculate_isobars_and_isopleths(sample=rhyolite,

→ temperature=800,

→ pressure_list=[500,
→ 1000,
→ 2000],

→ isopleth_list=[0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]

Iac_alkbasalt_isobars, Iac_alkbasalt_isopleths =
→ v.calculate_isobars_and_isopleths(sample=alkbasalt,

→ temperature=1200,

→ pressure_list=[500,
→ 1000,
→ 2000],

→ isopleth_list=[0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]

→ model="IAC"

Dixon_alkbasalt_isobars, Dixon_alkbasalt_isopleths =
→ v.calculate_isobars_and_isopleths(sample=alkbasalt,

→ temperature=1200,

→ pressure_list=[500,
→ 1000,
→ 2000],

→ isopleth_list=[0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]

→ model="DIXON"

Shish_alkbasalt_isobars, Shish_alkbasalt_isopleths =
→ v.calculate_isobars_and_isopleths(sample=alkbasalt,

→ temperature=1200,

→ pressure_list=[500,
→ 1000,
→ 2000],

→ isopleth_list=[0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]

→ model="SHISH"
```

↔ isop

↔ mode

```
Liu_rhyolite_isobars, Liu_rhyolite_isopleths =
  ↪ v.calculate_isobars_and_isopleths(sample=rhyolite,
```

↪ temperatur

↪ pressure_l

↪ 1000,

↪ 2000],

↪ isopleth_l

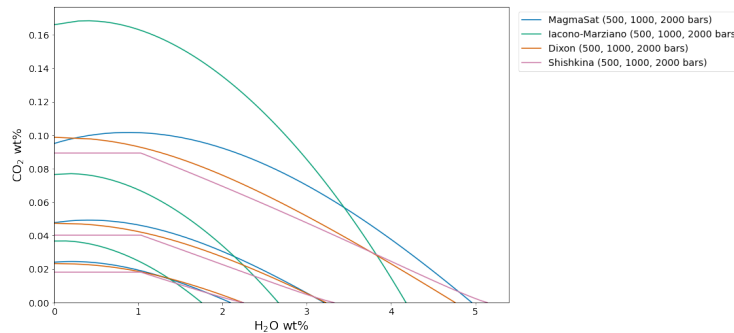
↪ model="Liu

```
Calculating isobar at 500 bars
done.
Calculating isobar at 1000 bars
done.
Calculating isobar at 2000 bars
done.
Done!
Calculating isobar at 500 bars
done.
Calculating isobar at 1000 bars
done.
Calculating isobar at 2000 bars
done.
Done!
```

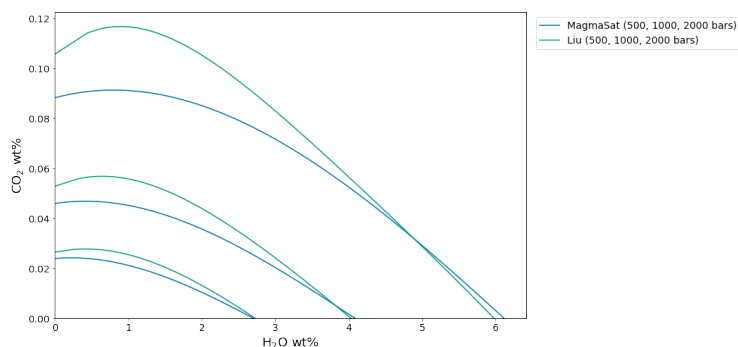
1375

[H]

```
fig, ax = v.plot(isobars=[alkbasalt_isobars, Iac_alkbasalt_isobars,
  ↪ Dixon_alkbasalt_isobars, Shish_alkbasalt_isobars],
  isobar_labels=["MagmaSat", "Iacono -Marziano", "Dixon",
  ↪ "Shishkina"])
fig, ax = v.plot(isobars=[rhyolite_isobars, Liu_rhyolite_isobars],
  ↪ isobar_labels=["MagmaSat", "Liu"])
v.show()
```



1376



1377

1378

1379

1380

1381

Isobars plotted for an alkali basalt (a) and rhyolite (b) with VESICAL for five mixed-fluid solubility models. For alkali basalt, MagmaSat, Iacono-Marziano, Dixon, and ShishkinaIdealMixing were used to create isobars at 1,200 °C. For rhyolite, MagmaSat and Liu were used to create isobars at 800 °C.

1382

1383

1384

1385

1386

1387

1388

It is immediately clear from Figure 5 that major disagreement exists between these models. For the alkali basalt, MagmaSat and Dixon show the best agreement, particularly at pressures <2000 bars. However, the mismatch between these models (and, indeed, between all models) increases with pressure. The Iacono-Marziano model is calibrated for highly depolymerized alkali basalts resulting in an increased capacity of the melt to dissolve CO_3^{2-} . That may explain why this model predicts significantly higher CO_2 solubilities at $\text{XH}_2\text{O}^{\text{fluid}}$ values approaching 0.

1389

1390

1391

1392

1393

1394

1395

1396

1397

The ShishkinaIdealMixing model displays nearly linear isobars, with finite solubility below ~1 wt% dissolved H_2O . This is a consequence of the model calibration; the pure- H_2O solubility expression of ShishkinaIdealMixing is not calibrated with any experiments at low PH_2O . This results in a finite solubility at low dissolved H_2O concentrations, such that the zero-pressure solubility is not zero. This produces significant model error at low but non-zero values of $\text{XH}_2\text{O}^{\text{fluid}}$. Thus, we caution the user against using the ShishkinaIdealMixing model at low but non-zero $\text{XH}_2\text{O}^{\text{fluid}}$ or when fluids deviate far from pure H_2O or pure CO_2 . In general, the Shishkina model should only be used for modeling pure- H_2O or pure- CO_2 fluids. This is discussed in more detail in Wieser et al. (2021).

1398

1399

1400

The models of MagmaSat and Liu show a similar level of disagreement for H_2O - CO_2 solubility in the rhyolite, with Liu predicting much higher dissolved CO_2 concentrations at low $\text{XH}_2\text{O}^{\text{fluid}}$ (<20 relative% or ~220 ppm at $\text{XH}_2\text{O}^{\text{fluid}}=0.1$).

1401

0.4.3 Sensitivity and error analysis

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

To date, very few studies have compared the sensitivity of their pressure estimates to the choice of solubility model, or propagated errors inherent to measurements of volatile concentrations in melts using SIMS, FTIR and Raman Spectroscopy into an error bar in terms of saturation pressure. In contrast, VESICAL allows users to import an Excel or CSV spreadsheet with each row containing the major element and volatile contents of each inclusion, as well as a temperature at which to evaluate solubility. Using the batch calculation functions, VESICAL will automatically calculate the saturation pressure for each row, using a user-specified model. Thus, users can more easily compare results from different solubility models, to robustly assess their applicability for the system of interest. Additionally, users could load a different spreadsheet, where the CO_2 and H_2O concentrations are adjusted to reflect the analytical uncertainty on the instrument used, allowing error bars on the saturation pressure to be calculated for every single inclusion. The modular and open-source nature of VESICAL also allows the user to combine the code with other Python3 modules. For example, users could utilize Markov chain Monte Carlo (MCMC; e.g., the python library emcee) methods to robustly calculate error distributions for each sample. In future releases, automatic sensitivity and error analysis on datasets

1418 and calculated results may be implemented directly within VESICAL, building on exist-
 1419 ing tools within the python community.

1420 **0.4.4 Future development**

1421 VESICAL represents the first comprehensive volatile solubility modeling tool of its
 1422 kind, including the feature that VESICAL is extensible. VESICAL is written so that imple-
 1423 menting new or yet-to-be-implemented solubility models is as simple as possible. To im-
 1424 plement a new model, python code describing the model equations needs to be written,
 1425 and this model name needs to be added to a list of model names within the code. To
 1426 make this as simple as possible such that the original authors of VESICAL are not the only
 1427 people who can develop the code, planned future work includes the creation of detailed
 1428 instructions (including instructional videos) illustrating this process.

1429 Likewise, new features can be added at any time, and enthusiastic members of the
 1430 community who wish to help bring such features to VESICAL are very welcome. Users can
 1431 contribute to VESICAL’s code, implementing new models and new features, via github
 1432 (<https://github.com/kaylai/VESICAL>). The repository is public, but we encourage users
 1433 who wish to contribute to the code to fork the repository into their private workspace
 1434 on github. Once edits to the code are complete, the new code can be added to VESI-
 1435 cal by creating a “Pull Request” inside of github. Changes and enhancements to VESI-
 1436 cal will correspond to a change in the code’s version number. The published version of
 1437 the code documented in this manuscript and archived on Zenodo is version 1.0.1 (DOI:
 1438 Iacovino, Matthews, Wieser, Moore, & Bégue (2021)). Planned features not implemented
 1439 in this release include: 1. Models to calculate sample oxygen fugacity from $\text{Fe}^{2+}/\Sigma\text{Fe}$
 1440 and vice versa; 2. Additional volatiles such as sulfur; 3. More thermodynamic solubil-
 1441 ity models such as that of Papale et al. (2006); 4. Sensitivity and error analysis func-
 1442 tions.

1443 **0.4.5 How to cite VESICAL and its models**

1444 To cite computations done using VESICAL, please cite this manuscript, the VESI-
 1445 cal version number, as well as the model(s) used. Note that if a model was not speci-
 1446 fied during calculations, the default model of MagmaSat was used and should be cited
 1447 as “MagmaSat Ghiorso & Gualda (2015)”. For example: “Calculations were performed
 1448 using VESICAL Iacovino, Matthews, Wieser, Moore, & Bégue (2021) with the models of
 1449 Shishkina et al. (2014) and Dixon (1997).” The web-app always runs on the most up-
 1450 to-date version of the VESICAL code, but it is best practice to note if the web-app was
 1451 used (“Calculations were performed using the VESICAL web-app Iacovino, Matthews, Wieser,
 1452 Moore, & Bégue (2021)...”). We also encourage users to be as explicit as possible as to
 1453 the conditions used for modelling. This includes stating the pressure, temperature, volatile
 1454 concentration, and bulk magma composition used in modelling. In the best case, VESI-
 1455 cal users will provide their code (e.g., as a Jupyter Notebook or .py file) along with their
 1456 publication such that it can be easily replicated.

1457 **0.5 Conclusions**

1458 VESICAL is a thermodynamic mixed-volatile solubility engine designed to meet the
 1459 growing computational needs of the igneous petrology community. Seven commonly used
 1460 volatile solubility models are built into VESICAL, which employs the most diversely cal-
 1461 ibrated (chemically and in P-T space) of the group, MagmaSat Ghiorso & Gualda (2015),
 1462 as the default model. VESICAL can perform five core calculations with any mixed-fluid
 1463 model and three core calculations with any model (mixed-fluid, CO_2 -only, H_2O -only).
 1464 VESICAL allows for automatic calculation of large datasets and robust built-in plotting
 1465 capability.

1466 Alongside model frameworks such as ENKI, VESICAL represents an early step forward
 1467 toward creating a generalized thermodynamic framework to model whole scale mag-
 1468 netic processes. Such a framework builds upon the key tenets of VESICAL; namely: fun-
 1469 damental thermodynamic underpinning; inclusion of existing modeling strategies; python
 1470 powered, open-source, and extensible code base; high usability at all levels; benchmark-
 1471 ing and testing; and power as a responsive and predictive tool.

1472 Open Research

1473 The VESICAL software is open source and is hosted on github (<https://github.com/kaylai/VESICAL>).
 1474 The version of VESICAL used in this manuscript is version 1.0.1 and is archived on Zen-
 1475 do (Iacovino, Matthews, Wieser, Moore, & Begue (2021)). VESICAL runs on top of ther-
 1476 moengine, a python package that is a part of the ENKI framework (<http://enki-portal.org/>).
 1477 The thermoengine library is open source and is available on GitLab ([https://gitlab.com/ENKI-](https://gitlab.com/ENKI-portal/ThermoEngine)
 1478 [portal/ThermoEngine](https://gitlab.com/ENKI-portal/ThermoEngine)). VESICAL was written in Python3 and should be stable up to at
 1479 least Python version 3.7.6. In addition to thermoengine, VESICAL requires the following
 1480 standard libraries (with versions used for testing indicated in brackets): pandas (1.0.1),
 1481 numpy (1.18.1), matplotlib (3.1.2), cycler (0.10.0), scipy (1.4.1), and sympy (1.5.1). The
 1482 VESICAL webapp interface runs through Anvil (anvil.works), which executes VESICAL code
 1483 on a cloud server. The code that facilitates the link between the anvil interface and the
 1484 VESICAL code is available on the VESICAL github. VESICAL can also be used within a Jupyter
 1485 Notebook and is hosted on the ENKI JupyterHub (<https://server.enki-portal.org/hub/login>)
 1486 such that the code can be accessed without installation on the user's local machine.

1487 All data sets used in this manuscript are available on the VESICAL github as well
 1488 as in the Supplementary Material of this manuscript. The example dataset used for worked
 1489 examples in Section 0.3 (`example_data.xlsx` file; Supplemental Dataset S1) contains
 1490 compositional information for basalts Tucker et al. (2019); Roggensack (2001), andesites
 1491 Moore et al. (1998), rhyolites Mercer et al. (2015); Myers et al. (2019), and alkaline melts
 1492 (photeophrite, basaltic-trachyandesite, and basanite from Iacovino et al. (2016)). Sev-
 1493 eral additional example datasets from the literature are available in the Supplement (Sup-
 1494 plementary Datasets S2-S5; Table 0.3.3.1). These include experimentally produced al-
 1495 kaline magmas from Iacovino et al. (2016) (`alkaline.xlsx`), basaltic melt inclusions from
 1496 Kilauea Tucker et al. (2019) and Gakkel Ridge Bennett et al. (2019) (`basalts.xlsx`),
 1497 basaltic melt inclusions from Cerro Negro volcano, Nicaragua Roggensack (2001) (`cerro_negro.xlsx`),
 1498 and rhyolite melt inclusions from the Taupo Volcanic Center, New Zealand Myers et al.
 1499 (2019) and a topaz rhyolite from the Rio Grande Rift Mercer et al. (2015) (`rhyolites.xlsx`).
 1500 Where available, the calibration datasets for VESICAL models are also provided (Supple-
 1501 mentary Datasets S6-S7).

1502 Acknowledgments

1503 This manuscript is dedicated to the memory of Dr. Peter Fox without whom none of this
 1504 work would have been possible. We thank Peter for his encouragement of this work, his
 1505 editorial handling of the manuscript, and for blazing a path for bringing executable manuscripts
 1506 to AGU journals. KI and GMM were supported by the NASA Jacobs JETS Contract
 1507 #NNJ13HA01C. PEW acknowledges support from a NERC DTP studentship (NE/L002507/1).
 1508 The authors thank Jackie Dixon and Bob Myhill for reviews, which greatly helped strengthen
 1509 this manuscript and the VESICAL code. The authors would also like to thank Mark Ghiorso,
 1510 Aaron Wolf, and the ENKI team for pushing thermodynamic modeling into the future
 1511 and for making this publication possible; Chelsea Allison and Giada Iacono-Marziano
 1512 for discussions on their published models and how to properly implement them in VESI-
 1513 cal; Christy B. Till for support of KI during early coding work with MagmaSat; and pre-
 1514 sentationgo for style elements used in flowchart figures. Permission for the use of the VESI-
 1515 cal fox logo was graciously provided by DeviantArt user Twai.

1516

References

- 1517 Allison, C., Roggensack, K., & Clarke, A. (2019). H₂O-CO₂ solubility in alkali-
 1518 rich mafic magmas: New experiments at mid-crustal pressures. *Contributions*
 1519 *to Mineralogy and Petrology*, 174. Retrieved from [https://doi.org/10.1007/](https://doi.org/10.1007/s00410-019-1592-4)
 1520 [s00410-019-1592-4](https://doi.org/10.1007/s00410-019-1592-4) doi: 10.1007/s00410-019-1592-4
- 1521 Bennett, E., Jenner, F., Millet, M.-A., Cashman, K., & Lissenberg, J. (2019).
 1522 Deep roots for mid-ocean-ridge volcanoes revealed by plagioclase-hosted melt
 1523 inclusions. *Nature*, 572(235). Retrieved from [https://doi.org/10.1038/](https://doi.org/10.1038/s41586-019-1448-0)
 1524 [s41586-019-1448-0](https://doi.org/10.1038/s41586-019-1448-0) doi: 10.1038/s41586-019-1448-0
- 1525 Blake, S. (1984). Volatile oversaturation during the evolution of silicic magma
 1526 chambers as an eruption trigger. *Journal of Geophysical Research*, 89, 8237–
 1527 8244. Retrieved from <https://doi.org/10.1029/jb089ib10p08237> doi:
 1528 [10.1029/jb089ib10p08237](https://doi.org/10.1029/jb089ib10p08237)
- 1529 Dixon, J. (1997). Degassing of alkalic basalts. *American Mineralogist*, 82, 368–
 1530 378. Retrieved from [https://doi.org/10.2138/am-](https://doi.org/10.2138/am-1997-3-415)
 1531 [1997-3-415](https://doi.org/10.2138/am-1997-3-415) doi: 10.2138/am-1997-3-415
- 1532 Dixon, J., Stolper, E., & Holloway, J. (1995). An experimental study of water
 1533 and carbon dioxide solubilities in mid-ocean ridge basaltic liquids. part i: Cali-
 1534 bration and solubility models. *Journal of Petrology*, 36, 1633–1646. Retrieved
 1535 from <https://doi.org/10.1093/oxfordjournals.petrology.a037267> doi:
 1536 [10.1093/oxfordjournals.petrology.a037267](https://doi.org/10.1093/oxfordjournals.petrology.a037267)
- 1537 Duan, Z., & Zhang, Z. (2006). Equation of state of the H₂O, CO₂, and H₂O-CO₂
 1538 systems up to 10 gpa and 2573.15 k: Molecular dynamics simulations with
 1539 ab initio potential surface. *Geochimica et Cosmochimica Acta*, 70, 2311–
 1540 2324. Retrieved from <https://doi.org/10.1016/j.gca.2006.02.009> doi:
 1541 [10.1016/j.gca.2006.02.009](https://doi.org/10.1016/j.gca.2006.02.009)
- 1542 Ghiorso, M., & Gualda, G. (2015). An H₂O–CO₂ mixed fluid saturation model com-
 1543 patible with rhyolite-melts. *Contributions to Mineralogy and Petrology*, 169, 1–30.
 1544 Retrieved from <https://doi.org/10.1007/s00410-015-1141-8> doi: 10.1007/
 1545 [s00410-015-1141-8](https://doi.org/10.1007/s00410-015-1141-8)
- 1546 Ghiorso, M., & Sack, R. (1995). Chemical mass transfer in magmatic processes. iv.
 1547 a revised and internally consistent thermodynamic model for the interpolation and
 1548 extrapolation of liquid-solid equilibria in magmatic systems at elevated tempera-
 1549 tures and pressures. *Contributions to Mineralogy and Petrology*, 119, 197–212. Re-
 1550 trieved from <https://doi.org/10.1007/bf00307281> doi: 10.1007/bf00307281
- 1551 Hughes, E., Buse, B., Kearns, S., Blundy, J., Kilgour, G., & Mader, H. (2019).
 1552 Low analytical totals in epma of hydrous silicate glass due to subsurface charg-
 1553 ing: Obtaining accurate volatiles by difference. *Chemical Geology*, 505, 48–
 1554 56. Retrieved from <https://doi.org/10.1016/j.chemgeo.2018.11.015> doi:
 1555 [10.1016/j.chemgeo.2018.11.015](https://doi.org/10.1016/j.chemgeo.2018.11.015)
- 1556 Iacono-Marziano, G., Morizet, Y., Trong, E., & Gaillard, F. (2012). New ex-
 1557 perimental data and semi-empirical parameterization of H₂O-CO₂ solubility in
 1558 mafic melts. *Geochimica et Cosmochimica Acta*, 97, 1–23. Retrieved from
 1559 <https://doi.org/10.1016/j.gca.2012.08.035> doi: 10.1016/j.gca.2012.08.035
- 1560 Iacovino, K., Matthews, S., Wieser, P., Moore, G., & Bégué, F. (2021). *Jupyter*
 1561 *notebook vesical: An open-source thermodynamic model engine for mixed volatile*
 1562 *(H₂O-CO₂) solubility in silicate melts*. Zenodo. Retrieved from [https://doi.org/](https://doi.org/10.5281/zenodo.5095409)
 1563 [10.5281/zenodo.5095409](https://doi.org/10.5281/zenodo.5095409) doi: 10.5281/zenodo.5095409
- 1564 Iacovino, K., Matthews, S., Wieser, P. E., Moore, G. M., & Begue, F. (2021). *Vesi-*
 1565 *cal v. 1.0.1*. Zenodo. Retrieved from <https://zenodo.org/record/5095382> doi:
 1566 [10.5281/ZENODO.5095382](https://zenodo.org/record/5095382)
- 1567 Iacovino, K., Moore, G., Roggensack, K., Oppenheimer, C., & Kyle, P. (2013).
 1568 H₂O-CO₂ solubility in mafic alkaline magma: Applications to volatile sources and
 1569 degassing behavior at Erebus volcano, Antarctica. *Contributions to Mineral-*

- 1570 *ogy and Petrology*, 166, 845–860. Retrieved from <https://doi.org/10.1007/s00410-013-0877-2> doi: 10.1007/s00410-013-0877-2
- 1571
- 1572 Iacovino, K., Oppenheimer, C., Scaillet, B., & Kyle, P. (2016). Storage and evolu-
1573 tion of mafic and intermediate alkaline magmas beneath ross island, antarctica.
1574 *Journal of Petrology*, 57, 93–118. Retrieved from <https://doi.org/10.1093/petrology/egv083> doi: 10.1093/petrology/egv083
- 1575
- 1576 Lesne, P., Scaillet, B., Pichavant, M., Iacono-Marziano, G., & Jean-Michel, B.
1577 (2011). The h₂O solubility of alkali basaltic melts: An experimental study. *Con-*
1578 *tributions to Mineralogy and Petrology*, 162, 133–151. Retrieved from <https://doi.org/10.1007/s00410-010-0588-x> doi: 10.1007/s00410-010-0588-x
- 1579
- 1580 Liu, Y., Zhang, Y., & Behrens, H. (2005). Solubility of h₂O in rhyolitic melts at
1581 low pressures and a new empirical model for mixed h₂O-co₂ solubility in rhy-
1582 olitic melts. *Journal of Volcanology and Geothermal Research*, 143, 219–235.
1583 Retrieved from <https://doi.org/10.1016/j.jvolgeores.2004.09.019> doi:
1584 10.1016/j.jvolgeores.2004.09.019
- 1585 Mercer, C., Hofstra, A., Todorov, T., Roberge, J., Burgisser, A., Adams, D., &
1586 Cosca, M. (2015). Pre-eruptive conditions of the hideaway park topaz rhy-
1587 olite: Insights into metal source and evolution of magma parental to the hen-
1588 derson porphyry molybdenum deposit, colorado. *Journal of Petrology*, 56,
1589 645–679. Retrieved from <https://doi.org/10.1093/petrology/egv010> doi:
1590 10.1093/petrology/egv010
- 1591 Moore, G. (2008). Interpreting h₂O and co₂ contents in melt inclusions: Constraints
1592 from solubility experiments and modeling. *Reviews in Mineralogy and Geochem-*
1593 *istry*, 69(1), 333–362. Retrieved from <https://doi.org/10.2138/rmg.2008.69.9>
1594 doi: 10.2138/rmg.2008.69.9
- 1595 Moore, G., Vennemann, T., & Carmichael, I. (1998). An empirical model for the
1596 solubility of h₂O in magmas to 3 kilobars. *American Mineralogist*, 83, 36–42. Re-
1597 trieved from <https://doi.org/10.2138/am-1998-1-203> doi: 10.2138/am-1998-1
1598 -203
- 1599 Myers, M., Wallace, P., & Wilson, C. (2019). Inferring magma ascent timescales
1600 and reconstructing conduit processes in explosive rhyolitic eruptions using dif-
1601 fusive losses of hydrogen from melt inclusions. *Journal of Volcanology and*
1602 *Geothermal Research*, 369, 95–112. Retrieved from <https://doi.org/10.1016/j.jvolgeores.2018.11.009> doi: 10.1016/j.jvolgeores.2018.11.009
- 1603
- 1604 Newman, S., & Lowenstern, J. (2002). Volatilecalc: A silicate melt-h₂O-co₂ solu-
1605 tion model written in visual basic for excel. *Computers & Geosciences*, 28, 597–
1606 604. Retrieved from [https://doi.org/10.1016/s0098-3004\(01\)00081-4](https://doi.org/10.1016/s0098-3004(01)00081-4) doi:
1607 10.1016/s0098-3004(01)00081-4
- 1608 Papale, P., Morretti, R., & Barbato, D. (2006). The compositional dependence of
1609 the saturation surface of h₂O + co₂ fluids in silicate melts. *Chemical Geology*, 229,
1610 78–95. Retrieved from <https://doi.org/10.1016/j.chemgeo.2006.01.013> doi:
1611 10.1016/j.chemgeo.2006.01.013
- 1612 Perkel, J. (2016). Democratic databases: Science on github. *Nature*, 538. Retrieved
1613 from <https://doi.org/10.1038/538127a> doi: 10.1038/538127a
- 1614 Roggensack, K. (2001). Unraveling the 1974 eruption of fuego volcano (guatemala)
1615 with small crystals and their young melt inclusions. *Geology*, 29, 911–914.
1616 Retrieved from [https://doi.org/10.1130/0091-7613\(2001\)029<0911:
1617 uteofv>2.0.co;2](https://doi.org/10.1130/0091-7613(2001)029<0911:uteofv>2.0.co;2) doi: <https://doi.org/c57htn>
- 1618 Shishkina, T., Botcharnikov, R., Holtz, F., Almeev, R., Jazwa, A., & Jakubiak, A.
1619 (2014). Compositional and pressure effects on the solubility of h₂O and co₂ in
1620 mafic melts. *Chemical Geology*, 388, 112–129. Retrieved from [https://doi.org/
1621 10.1016/j.chemgeo.2014.09.001](https://doi.org/10.1016/j.chemgeo.2014.09.001) doi: 10.1016/j.chemgeo.2014.09.001
- 1622 Stock, M., Humphreys, M., Smith, V., Isaia, R., & Pyle, D. (2016). Late-stage
1623 volatile saturation as a potential trigger for explosive volcanic eruptions. *Nature*

- 1624 *Geoscience*, 9(3), 249–254. Retrieved from <https://doi.org/10.1038/ngeo2639>
1625 doi: 10.1038/ngeo2639
- 1626 Stolper, E. (1982). The speciation of water in silicate melts. *Geochimica et Cos-*
1627 *mochimica Acta*, 46(12), 2609–2620. Retrieved from [https://doi.org/10.1016/](https://doi.org/10.1016/0016-7037(82)90381-7)
1628 [0016-7037\(82\)90381-7](https://doi.org/10.1016/0016-7037(82)90381-7) doi: 10.1016/0016-7037(82)90381-7
- 1629 Tait, S., Jaupart, C., & Vergnolle, S. (1989). Pressure, gas content and erup-
1630 tion periodicity of a shallow, crystallising magma chamber. *Earth and Planetary*
1631 *Science Letters*, 92(1), 107–123. Retrieved from [https://doi.org/10.1016/](https://doi.org/10.1016/0012-821x(89)90025-3)
1632 [0012-821x\(89\)90025-3](https://doi.org/10.1016/0012-821x(89)90025-3) doi: 10.1016/0012-821x(89)90025-3
- 1633 Tucker, J., Hauri, E., Pietruszka, A., Garcia, M., Marske, J., & Trusdell, F. (2019).
1634 A high carbon content of the hawaiian mantle from olivine-hosted melt in-
1635 clusions. *Geochimica et Cosmochimica Acta*, 254, 156–172. Retrieved from
1636 <https://doi.org/10.1016/j.gca.2019.04.001> doi: 10.1016/j.gca.2019.04.001
- 1637 Wieser, P. E., Iacovino, K., Matthews, S., Moore, G., & Allison, C. M. (2021).
1638 Vesical part ii: A critical approach to volatile solubility modelling using an
1639 open-source python3 engine. *Earth ArXiv*. Retrieved from [https://doi.org/](https://doi.org/10.31223/X5K03T)
1640 [10.31223/X5K03T](https://doi.org/10.31223/X5K03T) doi: 10.31223/X5K03T