

pyUserCalc: A revised Jupyter notebook calculator for uranium-series disequilibria in basalts

Lynne J Elkins^{1,1} and Marc Spiegelman^{2,2}

¹University of Nebraska-Lincoln

²Columbia University

December 25, 2020

Abstract

Meaningful analysis of uranium-series isotopic disequilibria in basaltic lavas relies on the use of complex forward numerical models like dynamic melting (McKenzie, 1985) and equilibrium porous flow (Spiegelman and Elliott, 1993). Historically, such models have either been solved analytically for simplified scenarios, such as constant melting rate or constant solid/melt trace element partitioning throughout the melting process, or have relied on incremental or numerical calculators with limited power to solve problems and/or restricted availability. The most public numerical solution to reactive porous flow, UserCalc (Spiegelman, 2000) was maintained on a private institutional server for nearly two decades, but that approach has been unsustainable in light of modern security concerns. Here we present a more long-lasting solution to the problems of availability, model sophistication and flexibility, and long-term access in the form of a cloud-hosted, publicly available Jupyter notebook. Similar to UserCalc, the new notebook calculates U-series disequilibria during time-dependent, equilibrium partial melting in a one-dimensional porous flow regime where mass is conserved. In addition, we also provide a new disequilibrium transport model which has the same melt transport model as UserCalc, but approximates rate-limited diffusive exchange of nuclides between solid and melt using linear kinetics. The degree of disequilibrium during transport is controlled by a Damköhler number, allowing the full spectrum of equilibration models from complete fractional melting ($Da = 0$) to equilibrium transport ($Da = [?]$).

1 **pyUserCalc: A revised Jupyter notebook calculator for**
2 **uranium-series disequilibria in basalts**

3 **Lynne J. Elkins¹, Marc Spiegelman²**

4 ¹University of Nebraska-Lincoln, Lincoln, NE, USA, lelkins@unl.edu

5 ²Lamont-Doherty Earth Observatory of Columbia University, Palisades, NY, USA, mspieg@ldeo.columbia.edu

6 **Key Points:**

- 7 • Cloud-based Jupyter notebook presents an open source, reproducible tool
8 for modeling U-series in basalts
- 9 • Equilibrium and pure disequilibrium porous flow U-series models with 1D
10 conservation of mass
- 11 • Scaled porous flow model introduces incomplete equilibrium scenario with
12 reaction rate limitations

Corresponding author: Lynne J. Elkins, lelkins@unl.edu

Abstract

Meaningful analysis of uranium-series isotopic disequilibria in basaltic lavas relies on the use of complex forward numerical models like dynamic melting (McKenzie, 1985) and equilibrium porous flow (Spiegelman and Elliott, 1993). Historically, such models have either been solved analytically for simplified scenarios, such as constant melting rate or constant solid/melt trace element partitioning throughout the melting process, or have relied on incremental or numerical calculators with limited power to solve problems and/or restricted availability. The most public numerical solution to reactive porous flow, UserCalc (Spiegelman, 2000) was maintained on a private institutional server for nearly two decades, but that approach has been unsustainable in light of modern security concerns. Here we present a more long-lasting solution to the problems of availability, model sophistication and flexibility, and long-term access in the form of a cloud-hosted, publicly available Jupyter notebook. Similar to UserCalc, the new notebook calculates U-series disequilibria during time-dependent, equilibrium partial melting in a one-dimensional porous flow regime where mass is conserved. In addition, we also provide a new disequilibrium transport model which has the same melt transport model as UserCalc, but approximates rate-limited diffusive exchange of nuclides between solid and melt using linear kinetics. The degree of disequilibrium during transport is controlled by a Damköhler number, allowing the full spectrum of equilibration models from complete fractional melting ($Da = 0$) to equilibrium transport ($Da = \infty$).

1 Introduction

Continuous forward melting models are necessary to interpret the origins of empirically-measured U-series isotopic disequilibria in basaltic lavas, but the limited and unreliable availability of reproducible tools for making such calculations remains a persistent problem for geochemists. To date, a number of models have been developed for this task, including classical dynamic melting after McKenzie (1985) and the reactive porous flow model of Spiegelman and Elliott (1993). There have since been numerous approaches to using both the dynamic and porous flow models that range from simplified analytical solutions (e.g., Sims et al., 1999; Zou, 1998; Zou and Zindler, 2000) to incremental dynamic melting calculators (Stracke et al., 2003), two-porosity calculators (Jull et al., 2002; Lundstrom et al., 2000; Sims et al., 2002), and one-dimensional numerical solutions to reactive porous flow (Spiegelman, 2000) and dynamic melting (Bourdon et al., 2005; Elkins et al., 2019). Unfortunately, some of the approaches published since 1990 lacked publicly available tools that would permit others to directly apply the authors' methods, and while the more simplified and incremental approaches remain appropriate for asking and approaching some questions, they are insufficient for other applications that require more complex approaches (e.g., two-lithology melting; Elkins et al., 2019). Other tools like UserCalc that were available to public users (Spiegelman, 2000) were limited in application and have since become unavailable.

In light of the need for more broadly accessible and flexible solutions to U-series disequilibrium problems in partial melting, here we present a cloud-server hosted, publicly available numerical calculator for one-dimensional, decompression partial melting. The tool is provided in a Jupyter notebook with importable Python code and can be accessed from a web browser. Users will be able to access and use the tool using a free cloud server account, or on their own computer given any standard Python distribution. As shown below, the notebook is structured to permit the user to select one of two primary model versions, either classical reactive porous flow after Spiegelman and Elliott (1993) and Spiegelman (2000),

65 or a new disequilibrium transport model, developed after the appendix formulas
 66 of Spiegelman and Elliott (1993). The new model ranges from pure disequilibrium
 67 porous flow transport (i.e., the mass-conserved equivalent of true fractional melt-
 68 ing over time) to a "scaled" disequilibrium scenario, where the degree of chemical
 69 equilibrium that is reached is determined by the relationship between the rate of
 70 chemical reaction and the solid decompression rate (which is, in turn, related to
 71 the overall melting rate), in the form of a Damköhler number.

72 This scaled disequilibrium model resembles the classic dynamic melting model of
 73 McKenzie (1985), with the caveat that ours is the first U-series melting model de-
 74 veloped for near-fractional, disequilibrium transport where mass is also conserved
 75 within a one-dimensional melting regime. That is, rather than controlling the
 76 quantity of melt that remains in equilibrium with the solid using a fixed residual
 77 porosity, the melt porosity is controlled by Darcy's Law and mass conservation
 78 constraints after Spiegelman and Elliott (1993), and the "near-fractional" scenario
 79 is simulated using the reaction rate of the migrating liquid with the upwelling
 80 solid matrix.

81 **2 Calculating U-series in basalts during mass-conserved, one-dimensional** 82 **porous flow**

83 **2.1 Solving for equilibrium transport**

84 Here we consider several forward melting models that calculate the concen-
 85 trations and activities of U-series isotopes (^{238}U , ^{230}Th , ^{226}Ra , ^{235}U , and ^{231}Pa)
 86 during partial melting and melt transport due to adiabatic mantle decompression.
 87 Following Spiegelman and Elliott (1993), we start with conservation of mass equa-
 88 tions for the concentration of a nuclide i , assuming chemical equilibrium between
 89 melt and solid:

$$90 \quad \frac{\partial}{\partial t} [\rho_f \phi + \rho_s (1 - \phi) D_i] c_i^f + \nabla \cdot [\rho_f \phi v + \rho_s (1 - \phi) D_i V] c_i^f = \lambda_{i-1} [\rho_f \phi + \rho_s (1 - \phi) D_{i-1}] c_{i-1}^f - \lambda_i [\rho_f \phi + \rho_s (1 - \phi) D_i] c_i^f \quad (1)$$

91 where t is time, c_i^f is the concentration of nuclide i in the melt, D_i is the bulk
 92 solid/liquid partition coefficient for nuclide i , ρ_f is the density of the fluid and
 93 ρ_s is the density of the solid, ϕ is the porosity (local volume fraction of melt), v is
 94 the velocity of the melt and V the velocity of the solid in three dimensions, λ_i is
 95 the decay constant of nuclide i , and $(i - 1)$ indicates the radioactive parent of nu-
 96 clide i see Table 1. Equation (1) states that the change in total mass of nuclide i in
 97 both the melt and the solid is controlled by the divergence of the mass flux trans-
 98 ported by both phases and by the radioactive decay of both parent and daughter
 99 nuclides (i.e., the right hand side of the equation above).

Table 1: List of Variables Used in This Study

Variable	Definition
c_i^f	Concentration of nuclide i in the liquid
c_i^s	Concentration of nuclide i in the solid
U_i^f	Natural log of the concentration of nuclide i in the liquid relative to its initial concentration
U_i^s	Natural log of the concentration of nuclide i in the solid relative to its initial concentration
U_i^{stable}	Stable element component of U_i^f
U_i^{rad}	Radiogenic component of U_i^f
a_i	Activity of nuclide i
a_i^0	Initial activity of nuclide i
z	Height in a one-dimensional melting column
h	Total height of the melting column
ζ	$= z/h$, Dimensionless fractional height in scaled one-dimensional melting column
D_i	Bulk solid/liquid partition coefficient for nuclide i
D_i^0	Initial bulk solid/liquid partition coefficient for nuclide i
ρ_f	Density of the liquid
ρ_s	Density of the solid
ϕ	Porosity (volume fraction of liquid present)
ϕ_0	Maximum or reference porosity
V	Solid velocity
v	Liquid velocity
W	One-dimensional solid velocity
w	One-dimensional liquid velocity
W_0	Solid mantle upwelling velocity
λ_i	Decay constant for nuclide i
λ_i'	$= \lambda_i h / W_0$, Decay constant for nuclide i scaled by solid transport time
Γ	Melting rate
Γ_0	Constant melting rate
F_{max}	Maximum degree of melting
w_{eff}^i	Effective liquid velocity of nuclide i
R_i^{i-1}	Ingrowth factor
α_i	Initial degree of secular disequilibrium in the unmelted solid
k	Permeability
K_r	Relative permeability factor
n	Permeability exponent
A_d	Permeability calibration function
\Re	Reactivity rate factor
d	Diffusion/Reaction length scale (e.g., grain-size)
Da	Dahmköhler number

100 The equilibrium model of Spiegelman and Elliott (1993) assumes that complete
101 chemical equilibrium is maintained between the migrating partial melt and the
102 solid rock matrix along a decompressing one-dimensional column. To close the
103 equations, they assume that melt transport is described by a simplified form of
104 Darcy's Law for permeable flow through the solid matrix. In one dimension, for
105 a steady-state upwelling column of melting mantle rocks, they defined the one-
106 dimensional melt and solid velocities (w and W , respectively), and expressed the
107 melt and solid fluxes as functions of height (z) in terms of a constant melting rate
108 Γ_0 :

$$\rho_f \phi w = \Gamma_0 z \quad (2)$$

$$\rho_s(1 - \phi)W = \rho_s W_0 - \Gamma_0 z \quad (3)$$

where W_0 is the solid mantle upwelling rate, and Γ_0 is equivalent to $\rho_s W_0 F_{max}$ divided by the depth h for a maximum degree of melting F_{max} .

Assuming an initial condition of secular equilibrium, where the initial activities $\lambda_i c_{i,0}^f D_i$ are equivalent for parent and daughter nuclides, they derived a system of differential equations for the concentration c_i^f in any decay chain, which can be solved numerically using equation (10) from that paper:

$$\frac{dc_i^f}{d\zeta} = c_i^f \frac{(D_i - 1)F_{max}}{D_i + (1 - D_i)F_{max}\zeta} + \lambda_i h \left[\frac{D_i[D_{i-1} + (1 - D_{i-1})F_{max}\zeta]}{D_{i-1}[D_i + (1 - D_i)F_{max}\zeta]} \frac{c_{i-1}^f}{w_{eff}^{i-1}} - \frac{c_i^f}{w_{eff}^i} \right] \quad (4)$$

where c_i^f is the scaled melt concentration ($= c_i^f / c_{i,0}^f$), ζ is the dimensionless fractional height in the scaled column, equal to 0 at the base and 1 at the top, and

$$w_{eff}^i = \frac{\rho_f \phi w + \rho_s(1 - \phi)D_i W}{\rho_f \phi + \rho_s(1 - \phi)D_i} \quad (5)$$

is the effective velocity for element i .

In their appendix, Spiegelman and Elliott (1993) developed the more general (and, arguably, realistic) form where Γ and D_i are functions of height z . The UserCalc model of Spiegelman (2000) then formulated a one-dimensional numerical integration for the concentrations of selected U-series isotopes in continuously produced partial melts with height z , after the equilibrium formulas above. The concentration expression derived by Spiegelman (2000) for the equilibrium scenario (formula 6 in that reference) is:

$$\frac{dc_i^f}{dz} = \frac{-c_i^f(z)}{F(z) + (1 - F(z))D_i(z)} \frac{d}{dz} [F(z) + (1 - F(z))D_i(z)] + \frac{\lambda_{i-1} \overline{\rho D_{i-1}} c_{i-1}^f(z) - \lambda_i \overline{\rho D_i} c_i^f(z)}{\rho_s W_0 [F(z) + (1 - F(z))D_i(z)]} \quad (6)$$

where F is the degree of melting. Spiegelman (2000) further observed that solving for the natural log of the concentrations normalized to the initial concentration of i , U_i , rather than the concentrations themselves, is more accurate, particularly for highly incompatible elements (formulas 7-9 in that reference). This is because log concentrations change linearly during melting, rather than exponentially, and are more numerically stable to calculate.

$$U_i^f = \ln \left(\frac{c_i^f}{c_{i,0}^f} \right) \quad (7)$$

$$\frac{dU_i^f}{dz} = \frac{1}{c_i^f(z)} \frac{dc_i^f}{dz} \quad (8)$$

$$\frac{dU_i^f}{dz} = \frac{-1}{F(z) + (1 - F(z))D_i(z)} \frac{d}{dz} [F(z) + (1 - F(z))D_i(z)] + \frac{\lambda_i}{w_{eff}^i} [R_i^{i-1} \exp[U_{i-1}^f(z) - U_i^f(z)] - 1] \quad (9)$$

138

139 For the formulas above, Spiegelman (2000) defined a series of variables that allow
140 for simpler integration formulas and aid in efficient solution of the model, namely

$$\overline{\rho D_i} = \rho_f \phi + \rho_s (1 - \phi) D_i(z), \quad (10)$$

141

$$\overline{F} = F(z) + (1 - F(z))D_i(z), \quad (11)$$

142

$$R_i^{i-1} = \alpha_i \frac{D_i^0}{D_{i-1}^0} \frac{\overline{\rho D_{i-1}}}{\overline{\rho D_i}}, \quad (12)$$

143

$$\alpha_i = \frac{\lambda_{i-1} c_{(i-1),0}^s}{\lambda_i c_{i,0}^s}, \quad (13)$$

144

145 and substituting from the formulas above

$$w_{eff}^i = \frac{\rho_s W_0 \overline{F}}{\overline{\rho D_i}}. \quad (14)$$

146

147 where D_i^0 is the initial bulk solid/melt partition coefficient for element i , R_i^{i-1} is
148 the ingrowth factor, and α is the initial degree of secular disequilibrium in the
149 unmelted solid.

150 $U_i(z) = \ln(c_f(z)/c_f^0)$, the log of the total concentration of nuclide i in the melt,
151 can then be decomposed into

$$U_i(z) = U_i^{stable}(z) + U_i^{rad}(z) \quad (15)$$

152

153 where

$$U_i^{stable}(z) = \ln \left[\frac{D_i^0}{\overline{F} D_i(z)} \right] \quad (16)$$

154

155 is the log concentration of a stable nuclide with the same partition coefficients,
156 and $U_i^{rad}(z)$ is the radiogenic ingrowth component. An alternate way of writing
157 the radiogenic ingrowth component of equation (9) of Spiegelman (2000) is:

$$\frac{dU_i^{rad}}{d\zeta} = \lambda_i' \frac{\overline{\rho D_i}}{\overline{F} D_i} \left[R_i^{i-1} \exp[U_{i-1}(\zeta) - U_i(\zeta)] - 1 \right] \quad (17)$$

158

159 where

$$\lambda'_i = \frac{h\lambda_i}{W_0} \quad (18)$$

is the decay constant of nuclide i , scaled by the solid transport time (h/W_0) across a layer of total height h . Note Eq. (17) is solved over a column of dimensionless height 1 where $\zeta \in [0, 1]$.

Using these equations, the UserCalc reactive porous flow calculator accepted user inputs for both $F(z)$ and $D_i(z)$. The method uses a formula for the melt porosity ($\phi(z)$) based on a Darcy's Law expression with a scaled permeability factor (formula 20 from Spiegelman (2000)):

$$K_r(z)A_d\phi^n(1-\phi)^2 + \phi[1 + F(z)(\frac{\rho_s}{\rho_f} - 1)] - \frac{\rho_s}{\rho_f}F(z) = 0 \quad (19)$$

where $K_r(z)$ is the scaled permeability with height z , A_d is a permeability calibration function, and n is the permeability exponent. The permeability exponent for a tube-shaped fluid network is expected to be $n = 2$, while for a sheet-shaped network it is 3; recent measurements of the permeabilities of experimental magmatic melt networks suggest realistic magma migration occurs in a manner intermediate between these two scenarios, with $n = 2.6$ (Miller et al., 2014). The relative permeability K_r is calculated with respect to the permeability at the top of the column, i.e. depth $z = z_{final}$:

$$K_r(z) = \frac{k(z)}{k(z_{final})} \quad (20)$$

and allows for locally enhanced flow (e.g., mimicking the effects of a relatively low viscosity fluid).

Our model implementation reproduces and builds on the prior efforts summarized above, using a readily accessible computer language (Python) and web application (Jupyter notebooks).

2.2 Solving for complete disequilibrium transport

We further present a calculation tool that solves a similar set of equations for pure chemical disequilibrium transport during one-dimensional decompression melting. This model assumes that the solid produces an instantaneous fractional melt in local equilibrium with the solid; however, the melt is not allowed to back-react with the solid during transport, as it would in the equilibrium model above. In the limiting condition defined by stable trace elements (i.e., without radioactive decay), the model reduces to the calculation for an accumulated fractional melt. The model solves for the concentration of each nuclide i in the solid (s) and liquid (f) using equations (26) and (27) of Spiegelman and Elliott (1993):

$$\frac{dc_i^s}{dz} = \frac{c_i^s(z)(1 - \frac{1}{D_i(z)})}{1 - F(z)} \frac{dF}{dz} + \frac{1 - \phi}{W_0(1 - F(z))} [\lambda_{i-1}c_{i-1}^s(z) - \lambda_i c_i^s(z)] \quad (21)$$

$$\frac{dc_i^f}{dz} = \frac{\frac{c_i^s(z)}{D_i(z)} - c_i^f(z)}{F(z)} \frac{dF}{dz} + \frac{\rho_f \phi}{\rho_s W_0 F(z)} [\lambda_{i-1}c_{i-1}^f(z) - \lambda_i c_i^f(z)] \quad (22)$$

195 which maintain conservation of mass for both fluid and solid individually, and
 196 do not assume chemical equilibration between the two phases. As above, the
 197 equations depend on $F(z)$ and $D_i(z)$, i.e. melt fractions and bulk rock partition
 198 coefficients that can vary with depth.

199 As above, the solid and fluid concentration equations are rewritten in terms of the
 200 logs of the concentrations:

$$201 \quad U_i^s(z) = \ln \left(\frac{c_i^s(z)}{c_{i,0}^s} \right), \quad U_i^f(z) = \ln \left(\frac{c_i^f(z)}{c_{i,0}^f} \right) \quad (23)$$

202 and thus

$$203 \quad \frac{dU_i}{dz} = \frac{1}{c_i(z)} \frac{dc_i}{dz} \quad (24)$$

204 We assume that initial $c_{i,0}^s = D_{i,0} c_{i,0}^f$. Also as above, the log concentration equa-
 205 tions can be broken into stable and radiogenic components, where the stable log
 206 concentration equations are:

$$207 \quad \frac{dU_i^{s,stable}}{dz} = \frac{1 - \frac{1}{D_i(z)}}{1 - F(z)} \frac{dF}{dz} \quad (25)$$

$$208 \quad \frac{dU_i^{f,stable}}{dz} = \frac{\frac{D_i^0}{D_i(z)} \exp(U_i^s(z) - U_i^f(z))}{F(z)} \frac{dF}{dz} \quad (26)$$

209 which are equivalent to a model for a fractionally melted residual solid and an
 210 accumulated fractional melt for the liquid.

211 Reincorporating this with the radiogenic component and scaling all distances by h
 212 gives the dimensionless equations:

$$213 \quad \frac{dU_i^s}{d\zeta} = \frac{1 - \frac{1}{D_i(\zeta)}}{1 - F(\zeta)} \frac{dF}{d\zeta} + \frac{1 - \phi}{1 - F(\zeta)} \lambda'_i \left[\frac{\alpha_{i-1}}{\alpha_i} \exp[U_{i-1}^s(\zeta) - U_i^s(\zeta)] - 1 \right] \quad (27)$$

$$214 \quad \frac{dU_i^f}{d\zeta} = \frac{\frac{D_i^0}{D_i(\zeta)} \exp(U_i^s(\zeta) - U_i^f(\zeta))}{F(\zeta)} + \frac{\rho_f \phi}{\rho_s F(\zeta)} \lambda'_i \left[\frac{D_i^0 \alpha_{i-1}}{D_{i-1}^0 \alpha_i} \exp[U_{i-1}^f(\zeta) - U_i^f(\zeta)] - 1 \right] \quad (28)$$

215 **2.3 Solving for transport with chemical reactivity rates**

216 The two models described above are end members for complete equilibrium and
 217 complete disequilibrium transport. For stable trace elements, these models pro-
 218 duce melt compositions that are equivalent to batch melting and accumulated
 219 fractional melting (e.g., Spiegelman and Elliott, 1993). However, the actual trans-
 220 port of a reactive fluid (like a melt) through a solid matrix can fall anywhere be-
 221 tween these end members depending on the rate of transport and re-equilibration
 222 between melt and solid, which can be sensitive to the mesoscopic geometry of
 223 melt and solid (e.g., Spiegelman and Kenyon, 1992). In an intermediate scenario,

224 we envision that some reaction occurs, but chemical equilibration is incomplete
 225 due to slow reaction rates relative to the differential transport rates for the fluid
 226 and solid. If reaction times are sufficiently rapid to achieve chemical exchange
 227 over the lengthscale of interest before the liquid segregates, chemical equilibrium
 228 can be achieved; but for reactions that occur more slowly than effective trans-
 229 port rates, only partial chemical equilibrium can occur (e.g., Grose and Afonso,
 230 2019; Iwamori, 1993, 1994; Kogiso et al., 2004; Liang and Liu, 2016; Peate and
 231 Hawkesworth, 2005; Qin et al., 1992; Yang et al., 2000). Such reaction rates can in-
 232 clude, for example, the rate of chemical migration over the distance between high
 233 porosity veins or channels (e.g., Aharonov et al., 1995; Jull et al., 2002; Spiegelman
 234 et al., 2001; Stracke and Bourdon, 2009); or, at the grain scale, the solid chemical
 235 diffusivity of elements over the diameter of individual mineral grains (e.g., Qin
 236 et al., 1992; Feineman and DePaolo, 2003; Grose and Afonso, 2019; Oliveira et al.,
 237 2020; Van Orman et al., 2002a, 2006).

238 To model this scaled reactivity scenario, we start with our equations for disequi-
 239 librium transport in a steady-state, one-dimensional conservative system, and
 240 add a chemical back-reaction term that permits exchange of elements between the
 241 fluid and the solid. The reaction term is scaled by a reactivity rate factor, \mathfrak{R} and
 242 expressed in $\text{kg}/\text{m}^3/\text{yr}$. (i.e., the same units as the melting rate). The reactivity
 243 rate thus behaves much like the melting rate by governing the rate of exchange
 244 between the solid and liquid phases, effectively scaling the degree to which chem-
 245 ical exchange can occur. This new term could simulate a number of plausible
 246 scenarios that would physically limit the rate of chemical exchange by transport
 247 along a given distance in a linear manner, such as the movement or diffusion of
 248 nuclides through the porous solid matrix between melt channels a given distance
 249 apart.

250 First, returning to the conservation of mass equations for a steady-state, one-
 251 dimensional, reactive system of stable trace elements, and using $\Gamma(z)$ to represent
 252 the melting rate:

$$253 \quad \frac{d}{dz} \rho_f \phi w = \Gamma(z) \quad (29)$$

$$254 \quad \frac{d}{dz} \rho_s (1 - \phi) W = -\Gamma(z) \quad (30)$$

$$255 \quad \frac{d}{dz} \rho_f \phi w c_i^f(z) = \frac{c_i^s(z)}{D_i(z)} \Gamma(z) - \mathfrak{R} \left(c_i^f(z) - \frac{c_i^s(z)}{D_i(z)} \right) \quad (31)$$

$$256 \quad \frac{d}{dz} \rho_s (1 - \phi) W c_i^s(z) = -\frac{c_i^s(z)}{D_i(z)} \Gamma(z) + \mathfrak{R} \left(c_i^f(z) - \frac{c_i^s(z)}{D_i(z)} \right) \quad (32)$$

257 where, for an adiabatic upwelling column,

$$258 \quad \Gamma(z) = \rho_s W_0 \frac{dF}{dz} \quad (33)$$

259 From this, the equations (29) and (30) can be integrated (with appropriate bound-
 260 ary conditions at $z = 0$) to give

$$261 \quad \rho_f \phi w = \rho_s W_0 F(z) \quad (34)$$

$$262 \quad \rho_s(1 - \phi)W = \rho_s W_0(1 - F(z)) \quad (35)$$

263 Next, we expand the concentration equations to include the reactivity factor, and
 264 substitute the conservation of total mass determined above:

$$265 \quad \rho_s W_0 F(z) \frac{d}{dz} c_i^f(z) + c_i^f(z) \Gamma(z) = \frac{c_i^s(z)}{D_i(z)} \Gamma(z) - \Re \left(c_i^f(z) - \frac{c_i^s(z)}{D_i(z)} \right) \quad (36)$$

$$266 \quad \rho_s W_0(1 - F(z)) \frac{d}{dz} c_i^s(z) - c_i^s(z) \Gamma(z) = -\frac{c_i^s(z)}{D_i(z)} \Gamma(z) + \Re \left(c_i^f(z) - \frac{c_i^s(z)}{D_i(z)} \right) \quad (37)$$

267 If we then combine the $\Gamma(z)$ terms and rearrange:

$$268 \quad \rho_s W_0 F(z) \frac{d}{dz} c_i^f(z) = \Gamma(z) \left(\frac{c_i^s(z)}{D_i(z)} - c_i^f(z) \right) - \Re \left(c_i^f(z) - \frac{c_i^s(z)}{D_i(z)} \right) \quad (38)$$

$$269 \quad \rho_s W_0(1 - F(z)) \frac{d}{dz} c_i^s(z) = \Gamma(z) c_i^s(z) \left(1 - \frac{1}{D_i(z)} \right) + \Re \left(c_i^f(z) - \frac{c_i^s(z)}{D_i(z)} \right) \quad (39)$$

270 We can now divide the fluid and solid equations by c_i^f and c_i^s , respectively, and
 271 rearrange the W_0 terms:

$$272 \quad \frac{1}{c_i^f(z)} \frac{dc_i^f}{dz} = \frac{1}{\rho_s W_0 F(z)} \left[\Gamma(z) \left(\frac{c_i^s(z)}{D_i(z) c_i^f(z)} - 1 \right) - \Re \left(1 - \frac{c_i^s(z)}{D_i(z) c_i^f(z)} \right) \right] \quad (40)$$

$$273 \quad \frac{1}{c_i^s(z)} \frac{dc_i^s}{dz} = \frac{1}{\rho_s W_0(1 - F(z))} \left[\Gamma(z) \left(1 - \frac{1}{D_i(z)} \right) + \frac{\Re}{D_i(z)} \left(\frac{D_i(z) c_i^f(z)}{c_i^s(z)} - 1 \right) \right] \quad (41)$$

274 The first terms on the right-hand side of each of these equations are identical to
 275 pure disequilibrium melting, such that if \Re is zero, the equations reduce to the
 276 disequilibrium transport case of Spiegelman and Elliott (1993).

277 To solve, the final terms that involve the reactivity factor can be further rewritten
 278 using the definitions for U_i^f and U_i^s :

$$279 \quad c_i^f(z) = c_{i,0}^f \exp[U_i^f(z)] = \frac{c_{i,0}^s}{D_i^0} \exp[U_i^f(z)] \quad (42)$$

$$280 \quad c_i^s(z) = c_{i,0}^s \exp[U_i^s(z)] \quad (43)$$

281 Thus:

$$282 \quad \frac{D_i(z) c_i^f(z)}{c_i^s(z)} = \frac{D_i(z)}{D_i^0} \exp[U_i^f(z) - U_i^s(z)] \quad (44)$$

$$\frac{c_i^s(z)}{D_i(z)c_i^f(z)} = \frac{D_i^0}{D_i(z)} \exp[U_i^s(z) - U_i^f(z)] \quad (45)$$

and:

$$\frac{dU_i^f}{dz} = \frac{1}{\rho_s W_0 F(z)} \left[\Gamma(z) \left(\frac{D_i^0}{D_i(z)} \exp[U_i^s(z) - U_i^f(z)] - 1 \right) - \Re \left(1 - \frac{D_i^0}{D_i(z)} \exp[U_i^s(z) - U_i^f(z)] \right) \right] \quad (46)$$

$$\frac{dU_i^s}{dz} = \frac{1}{\rho_s W_0 (1 - F(z))} \left[\Gamma(z) \left(1 - \frac{1}{D_i(z)} \right) + \frac{\Re}{D_i(z)} \left(\frac{D_i(z)}{D_i^0} \exp[U_i^f(z) - U_i^s(z)] - 1 \right) \right] \quad (47)$$

Finally, substituting adiabatic upwelling and scaling with depth by h , and adding radioactive terms gives the full solutions for the dimensionless equations $dU_i/d\zeta$:

$$\frac{dU_i^f}{d\zeta} = \frac{1}{F(\zeta)} \left[\frac{dF}{d\zeta} \left(\frac{D_i^0}{D_i(\zeta)} \exp[U_i^s(\zeta) - U_i^f(\zeta)] - 1 \right) \right] - \frac{\Re h}{\rho_s W_0 F(\zeta)} \left[1 - \frac{D_i^0}{D_i(\zeta)} \exp[U_i^s(\zeta) - U_i^f(\zeta)] \right] + \frac{\rho_f \phi}{\rho_s F} \lambda_i' \left[\frac{D_i^0 \alpha_{i-1}}{D_{i-1}^0 \alpha_i} \exp[U_{i-1}^f(\zeta) - U_i^f(\zeta)] - 1 \right] \quad (48)$$

$$\frac{dU_i^s}{d\zeta} = \frac{1}{(1 - F(\zeta))} \left[\frac{dF}{d\zeta} \left(1 - \frac{1}{D_i(\zeta)} \right) \right] + \frac{\Re h}{\rho_s W_0 D_i(\zeta) (1 - F(\zeta))} \left[\frac{D_i(\zeta)}{D_i^0} \exp[U_i^f(\zeta) - U_i^s(\zeta)] - 1 \right] + \frac{1 - \phi}{1 - F(\zeta)} \lambda_i' \left[\frac{\alpha_{i-1}}{\alpha_i} \exp[U_{i-1}^s(\zeta) - U_i^s(\zeta)] - 1 \right] \quad (49)$$

where h is the total height of the melting column.

2.3.1 The Dahmköhler number

The dimensionless combination

$$Da = \frac{\Re h}{\rho_s W_0} \quad (50)$$

is the Dahmköhler number, which governs the reaction rate relative to the solid transport time. If re-equilibration is limited by solid state diffusion, \Re can be estimated using:

$$\Re \approx \frac{\rho_s \mathcal{D}_i}{d^2} \quad (51)$$

where \mathcal{D}_i is the *solid state* diffusivity of element i , and d is a nominal spacing between melt-channels (this spacing could, for example, be the average grain diameter for grain-scale channels, or 10 cm for closely spaced veins).

302 In this case (which we will assume for this paper), the Dahmköhler number can
 303 be written

$$304 \quad Da = \frac{D_i h}{W_0 d^2} \quad (52)$$

305 Substituting the definition of Da above yields the final dimensionless ODEs for
 306 the disequilibrium transport model:

$$307 \quad \frac{dU_i^f}{d\zeta} = \frac{1}{F(\zeta)} \left(\frac{dF}{d\zeta} + Da \right) \left(\frac{D_i^0}{D_i(\zeta)} \exp[U_i^s(\zeta) - U_i^f(\zeta)] - 1 \right) + \frac{\rho_f \phi}{\rho_s F(\zeta)} \lambda'_i \left[\frac{D_i^0 \alpha_{i-1}}{D_{i-1}^0 \alpha_i} \exp[U_{i-1}^f(\zeta) - U_i^f(\zeta)] - 1 \right] \quad (53)$$

$$308 \quad \frac{dU_i^s}{d\zeta} = \frac{1}{(1 - F(\zeta))} \left[\frac{dF}{d\zeta} \left(1 - \frac{1}{D_i(\zeta)} \right) + \frac{Da}{D_i(\zeta)} \left(\frac{D_i(\zeta)}{D_i^0} \exp[U_i^f(\zeta) - U_i^s(\zeta)] - 1 \right) \right] + \quad (54)$$

$$\frac{1 - \phi}{1 - F(\zeta)} \lambda'_i \left[\frac{\alpha_{i-1}}{\alpha_i} \exp[U_{i-1}^s(\zeta) - U_i^s(\zeta)] - 1 \right]$$

309 with initial conditions $U_i^s = U_i^f = 0$.

310 In the limit where the Dahmköhler number approaches zero, the above formulas
 311 reduce to pure disequilibrium transport, whereas if Da approaches infinity (i.e.,
 312 infinitely fast reactivity compared to physical transport), the system approaches
 313 equilibrium conditions ($c_i^s \rightarrow D_i c_i^f$).

314 2.3.2 Initial conditions

315 Inspection of equation (53) shows that for the initial conditions described above
 316 and $F(0) = 0$, $\frac{dU_i^f}{d\zeta}$ is ill-defined (at least numerically in a floating-point system).
 317 However, taking the limit $\zeta \rightarrow 0$ and applying L'Hôpital's rule yields

$$318 \quad \lim_{\zeta \rightarrow 0} \frac{dU_i^f}{d\zeta} = \frac{U_i^s(0) - U_i^f(0)}{F'(0)} \left(\frac{dF}{d\zeta} + Da \right) + \lambda'_i \left[\frac{D_i^0 \alpha_{i-1}}{D_{i-1}^0 \alpha_i} - 1 \right] \quad (55)$$

319 where

$$320 \quad U_i^s(0) = \left. \frac{dU_i^s}{d\zeta} \right|_{\zeta=0} \quad (56)$$

$$321 \quad U_i^f(0) = \left. \frac{dU_i^f}{d\zeta} \right|_{\zeta=0} \quad (57)$$

$$322 \quad F'(0) = \left. \frac{dF}{d\zeta} \right|_{\zeta=0} \quad (58)$$

323 The initial radiogenic term also uses the limit from equation (34):

$$\lim_{\zeta \rightarrow 0} \frac{\rho_f \phi}{\rho_s F} = \frac{W_0}{w(0)} = 1 \quad (59)$$

Rearranging equation (55) gives the value for $U_i^f(0)$ for $F = 0$ as

$$\lim_{\zeta \rightarrow 0} \frac{dU_i^f}{d\zeta} = \frac{1}{2 + \frac{Da}{F'(0)}} \left[U_i^s(0) \left(1 + \frac{Da}{F'(0)} \right) + \lambda_i' \left[\frac{D_i^0 \alpha_{i-1}}{D_{i-1}^0 \alpha_i} - 1 \right] \right] \quad (60)$$

3 A pyUserCalc Jupyter notebook

3.1 Code design

The `UserCalc` Python package implements both equilibrium and disequilibrium transport models and provides a set of code classes and utility functions for calculating and visualizing the results of one-dimensional, steady-state, partial melting forward models for both the ^{238}U and ^{235}U decay chains. The code package is organized into a set of Python classes and plotting routines, which are documented in the docstrings of the classes and also demonstrated in detail below. Here we briefly describe the overall functionality and design of the code, which is open-source and can be modified to suit an individual researcher's needs. The code is currently available in a Git repository (<https://gitlab.com/ENKI-portal/pyUsercalc>), and any future edits or merge requests will be managed through GitLab.

The equilibrium and disequilibrium transport models described above have each been implemented as Python classes with a generic code interface:

```

342  ...
343  Interface:
344  -----
345      model(alpha0, lambdas, D, W0, F, dFdz, phi, rho_f=2800., rho_s=3300.,
346            method=method, Da=inf)
347
348  Parameters:
349  -----
350      alpha0 : numpy array of initial activities
351      lambdas : numpy array of decay constants scaled by solid transport
352               time
353      D      : Function D(z) -- returns an array of partition coefficients
354               at scaled height z
355      W0     : float -- Solid mantle upwelling rate
356      F      : Function F(z) -- returns the degree of melting F
357      dFdz   : Function dFdz(z) -- returns the derivative of F
358      phi    : Function phi(z) -- returns the porosity
359      rho_f  : float -- melt density
360      rho_s  : float -- solid density
361      method : string -- ODE time-stepping scheme to be passed to
362               solve_ivp (one of 'RK45', 'Radau', 'BDF')
363      Da     : float -- Dahmkohler Number (defaults to \inf, unused in
364               equilibrium model)
365
366  Required Method:
367  -----
368      model.solve(): returns depth and log concentration numpy arrays z,
369      Us, Uf

```

370

371 which solves the scaled equations (i.e., equation (9) or equations (53) and (54) for
 372 the log concentrations of nuclides U_i^f and U_i^s in a decay chain of arbitrary length,
 373 with scaled decay constants λ_i' and initial activity ratios α_0 . In the code, we use the
 374 variable z for the scaled height in the column (i.e. $z \equiv \zeta$), and the model equations
 375 assume a one-dimensional column with scaled height $0 \leq z \leq 1$. The bulk partition
 376 coefficients $D_i(z)$, degree of melting $F(z)$, melting rate $dF/dz(z)$, and porosity $\phi(z)$
 377 are provided as functions of height in the column. Optional arguments include the
 378 melt and solid densities ρ_f and ρ_s , the Dahmköhler number Da , and the preferred
 379 numerical integration method (see `scipy.integrate.solve_ivp`). Some of these
 380 variables, such as $D_i(z)$ and $F(z)$, are provided by the user as described further
 381 below, and are then interpolated using model functions.

382

`UserCalc` provides two separate model classes, `EquilTransport` and
 383 `DisequilTransport`, for the different transport models; the user could add any
 384 other model that uses the same interface, if desired. Most users, however, will not
 385 access the models directly but rather through the driver class `UserCalc`,
 386 which provides support for solving and visualizing column models for the relevant
 387 ^{238}U and ^{235}U decay chains. The general interface for the `UserCalc` class is:

388

```
389 A class for constructing solutions for 1-D, steady-state, open-system
390 U-series transport calculations as in Spiegelman (2000) and
391 Elkins and Spiegelman (2021).
```

392

```
393 Usage:
```

394

```
394 -----
```

395

```
396     us = UserCalc(df,dPdZ = 0.32373,n = 2.,tol=1.e-6,phi0 = 0.008,
397                 W0 =3.,model=EquilTransport,Da=None,stable=False,
398                 method='Radau')
```

399

```
400 Parameters:
```

401

```
401 -----
```

402

```
402 df : A pandas dataframe with columns ['P','F', Kr','DU','DTh',
403 'DRa','DPa']
```

404

```
404 dPdZ : float -- Pressure gradient, to convert pressure P to
405 depth z
```

406

```
406 n : float -- Permeability exponent
```

407

```
407 tol : float -- Tolerance for the ODE solver
```

408

```
408 phi0 : float -- Reference melt porosity
```

409

```
409 W0 : float -- Upwelling velocity (cm/yr)
```

410

```
410 model : class -- A U-series transport model class (one of
411 EquilTransport or DisequilTransport)
```

412

```
412 Da : float -- Optional Da number for disequilibrium transport model
```

413

```
413 stable : bool
```

414

```
414     True: calculates concentrations for non-radiogenic nuclides with same
415     chemical properties (i.e. sets lambda=0)
```

416

```
416     False: calculates the full radiogenic problem
```

417

```
417 method : string
```

418

```
418     ODE time-stepping method to pass to solve_ivp (usually one of 'Radau',
419     'BDF', or 'RK45')
```

420

```
420 ----
```

421 The principal required input is a spreadsheet containing the degree of melting $F(P)$,
 422 relative permeability $K_r(P)$, and bulk partition coefficients for the elements D_U ,
 423 D_{Th} , D_{Ra} and D_{Pa} as functions of pressure P . The structure of the input data spread-
 424 sheet is the same as that described in Spiegelman (2000), and is illustrated in Table
 425 2 below. Because the user provides $F(z)$, $K_r(z)$, and bulk solid $D_i(z)$ input informa-
 426 tion to the model directly, any considerations such as mineral modes, mineral/melt
 427 D_i values, and productivity variations are external to this model and must be de-
 428 veloped by the user separately. Once given this spreadsheet by the user, the code
 429 routine initializes the decay constants for the isotopic decay chains and provides
 430 functions to interpolate $F(z)$ and $D_i(z)$ and calculate the porosity $\phi(z)$. Once thus
 431 initialized, the `UserCalc` class further provides the following methods:

```

432     ...
433     Principal Methods:
434     -----
435         phi                : returns porosity as a function of column
436                             height
437         set_column_parameters : resets principal column parameters
438                             phi0, n, W0
439         solve_1D            : 1D column solution for a single Decay
440                             chain with arbitrary D, lambda, alpha_0
441         solve_all_1D        : Solves a single column model for both 238U
442                             and 235U chains.
443                             returns a pandas dataframe
444         solve_grid          : Solves multiple column models for a grid
445                             of porosities and upwelling rates
446                             returns a 3-D array of activity ratios
447     ...

```

448 Of these, the principal user-facing methods are:

- 449 1. `UserCalc.solve_all_1D`, which returns a `pandas.DataFrame` table that
 450 contains, at each depth, solutions for the porosity (ϕ), the log concentrations
 451 of the specified nuclides in the ^{238}U and ^{235}U decay chains in both the melt
 452 and the solid, and the U-series activity ratios.
- 453 2. `UserCalc.solve_grid`, which solves for a grid of one-dimensional solutions
 454 for different reference porosities (ϕ_0) and solid upwelling rates (W_0) and
 455 returns arrays of U-series activity ratios at a specified depth (usually the top
 456 of the column), as described in Spiegelman and Elliott (1993).

457 3.1.1 Visualization Functions

458 In addition to the principal classes for calculating U-series activity ratios in partial
 459 melts, the `UserCalc` package also provides functions for visualizing model inputs
 460 and outputs. The primary plotting functions include:

- 461 1. `UserCalc.plot_inputs(df)` : Visualizes the input dataframe to show $F(P)$,
 462 $K_r(P)$ and $D_i(P)$.
- 463 2. `UserCalc.plot_1Dcolumn(df)` : Visualizes the output dataframe for a single
 464 one-dimensional melting column.
- 465 3. `UserCalc.plot_contours(phi0, W0, act)` : Visualizes the output of
 466 `UserCalc.solve_grid` by generating contour plots of activity ratios at a
 467 specific depth as functions of the porosity (ϕ_0) and solid upwelling rate (W_0).

468 4. `UserCalc.plot_mesh_Ra(Th,Ra,W0,phi0)` and `UserCalc.plot_mesh_Pa(Th,Pa,W0,phi0)` :
 469 Generates 'mesh' plots showing results for different ϕ_0 and W_0 values on
 470 ($^{226}\text{Ra}/^{230}\text{Th}$) vs. ($^{230}\text{Th}/^{238}\text{U}$) and ($^{231}\text{Pa}/^{235}\text{U}$) vs. ($^{230}\text{Th}/^{238}\text{U}$) activity
 471 diagrams.

472 Both the primary solver routines and visualization routines will be demonstrated in
 473 detail below.

474 3.1.2 Miscellaneous Convenience Functions

475 Finally, the UserCalc module also provides a simple input spreadsheet generator
 476 similar to the one provided with the original UserCalc program of Spiegelman
 477 (2000). This tool is described more fully in the accompanying Jupyter notebook
 478 `twolayermodel.ipynb` in the Supplemental Materials, and has the interface:

```
479 df = UserCalc.twolayermodel(P, F, Kr, D_lower, D_upper, N=100, P_lambda=1)
```

480 3.2 An example demonstrating pyUserCalc functionality for a single melting 481 column

482 The Python code cells embedded below provide an example problem that demon-
 483 strates the use and behavior of the model for a simple, two-layer upwelling mantle
 484 column, with a constant melting rate within each layer and constant $K_r = 1$. This
 485 example is used to compare the outcomes from the original UserCalc equilibrium
 486 model (Spiegelman, 2000) to various other implementations of the code, such as
 487 pure disequilibrium transport and scaled reactivity rates, as described above.

488 To run the example code and use this article as a functioning Jupyter notebook,
 489 while in a web-enabled browser the user should select an embedded code cell by
 490 mouse-click and then simultaneously type the 'Shift' and 'Enter' keys to run the cell,
 491 after which selection will automatically advance to the following cell. The first cell
 492 below imports necessary code libraries to access the Python toolboxes and functions
 493 that will be used in the rest of the program:

```
[1]: # Select this cell with by mouseclick, and run the code by simultaneously
      ↪typing the 'Shift' + 'Enter' keys.
      # If the browser is able to run the Jupyter notebook, a number [1] will
      ↪appear to the left of the cell.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Import UserCalc:
import UserCalc
```

494 3.2.1 Entering initial input information and viewing input data

495 In the full Jupyter notebook code available in the Git repository and provided here
 496 as Supplementary Materials, the user can edit a notebook copy and indicate their
 497 initial input data, as has been done for the sample data set below. The name for the
 498 user's input data file should be set in quotes (i.e., replacing the word 'sample' in
 499 the cell below with the appropriate filename, minus the file extension). This name
 500 will be used both to find the input file and to label any output files produced. Our
 501 sample file can likewise be downloaded and used as a formatting template for other
 502 input files (see Supplementary Materials), and is presented as a useful example
 503 below. The desired input file should be saved to a 'data' folder in the notebook di-
 504

505 rectory prior to running the code. If desired, a similarly simple two-layer input file
506 can also be generated using the calculator tool provided in the supplementary code.

507 Once the cell has been edited to contain the correct input file name, the user should
508 run the cell using the technique described above:

```
509 [2]: runname='sample'
```

510 The next cell below will read in the input data using the user filename specified
511 above:

```
512 [3]: input_file = 'data/{}.csv'.format(runname)  
df = pd.read_csv(input_file, skiprows=1, dtype=float)  
df
```

Table 2: Input data table for example tested here, showing pressures in kbar (P), degree of melting (F), permeability coefficient (K_r), and bulk solid/melt partition coefficients (D_i) for the elements of interest, U, Th, Ra, and Pa. This table illustrates the format required for input files for this model.

	P	F	Kr	DU	DTh	DRa	DPa
0	40.0	0.00000	1.0	0.00900	0.00500	0.00002	0.00001
1	39.0	0.00241	1.0	0.00900	0.00500	0.00002	0.00001
2	38.0	0.00482	1.0	0.00900	0.00500	0.00002	0.00001
3	37.0	0.00723	1.0	0.00900	0.00500	0.00002	0.00001
4	36.0	0.00964	1.0	0.00900	0.00500	0.00002	0.00001
5	35.0	0.01210	1.0	0.00900	0.00500	0.00002	0.00001
6	34.0	0.01450	1.0	0.00900	0.00500	0.00002	0.00001
7	33.0	0.01690	1.0	0.00900	0.00500	0.00002	0.00001
8	32.0	0.01930	1.0	0.00900	0.00500	0.00002	0.00001
9	31.0	0.02170	1.0	0.00900	0.00500	0.00002	0.00001
10	30.0	0.02410	1.0	0.00900	0.00500	0.00002	0.00001
11	29.0	0.02650	1.0	0.00900	0.00500	0.00002	0.00001
12	28.0	0.02890	1.0	0.00900	0.00500	0.00002	0.00001
13	27.0	0.03130	1.0	0.00900	0.00500	0.00002	0.00001
14	26.0	0.03370	1.0	0.00900	0.00500	0.00002	0.00001
15	25.0	0.03620	1.0	0.00900	0.00500	0.00002	0.00001
16	24.0	0.03860	1.0	0.00900	0.00500	0.00002	0.00001
17	23.0	0.04100	1.0	0.00899	0.00500	0.00002	0.00001
18	22.0	0.04340	1.0	0.00893	0.00498	0.00002	0.00001
19	21.0	0.04610	1.0	0.00852	0.00488	0.00002	0.00001
20	20.0	0.05000	1.0	0.00700	0.00450	0.00002	0.00001
21	19.0	0.05610	1.0	0.00548	0.00412	0.00002	0.00001
22	18.0	0.06340	1.0	0.00507	0.00402	0.00002	0.00001
23	17.0	0.07100	1.0	0.00501	0.00400	0.00002	0.00001
24	16.0	0.07860	1.0	0.00500	0.00400	0.00002	0.00001
25	15.0	0.08620	1.0	0.00500	0.00400	0.00002	0.00001
26	14.0	0.09370	1.0	0.00500	0.00400	0.00002	0.00001
27	13.0	0.10133	1.0	0.00500	0.00400	0.00002	0.00001
28	12.0	0.10892	1.0	0.00500	0.00400	0.00002	0.00001
29	11.0	0.11651	1.0	0.00500	0.00400	0.00002	0.00001
30	10.0	0.12410	1.0	0.00500	0.00400	0.00002	0.00001
31	9.0	0.13169	1.0	0.00500	0.00400	0.00002	0.00001
32	8.0	0.13928	1.0	0.00500	0.00400	0.00002	0.00001
33	7.0	0.14687	1.0	0.00500	0.00400	0.00002	0.00001
34	6.0	0.15446	1.0	0.00500	0.00400	0.00002	0.00001
35	5.0	0.16205	1.0	0.00500	0.00400	0.00002	0.00001
36	4.0	0.16964	1.0	0.00500	0.00400	0.00002	0.00001
37	3.0	0.17723	1.0	0.00500	0.00400	0.00002	0.00001
38	2.0	0.18482	1.0	0.00500	0.00400	0.00002	0.00001
39	1.0	0.19241	1.0	0.00500	0.00400	0.00002	0.00001
40	0.0	0.20000	1.0	0.00500	0.00400	0.00002	0.00001

513 The next cell will visualize the input dataframe in Figure 1, using the utility function
514 `plot_inputs` :

```
[4]: fig = UserCalc.plot_inputs(df)
```

515

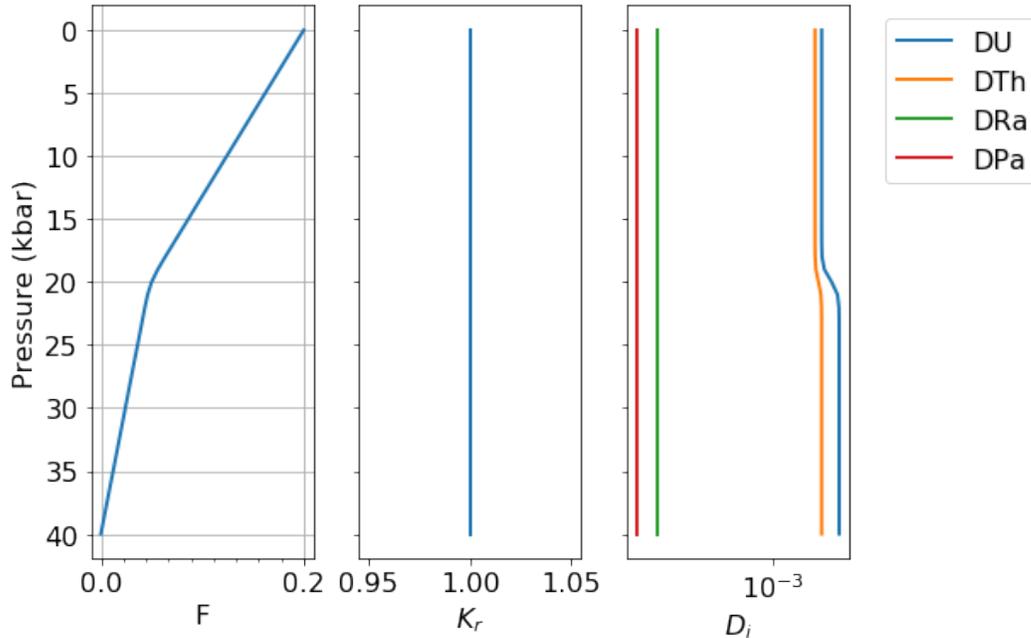


Figure 1: Diagrams showing example input parameters F , K_r , and D_i as a function of pressure, for the sample input file tested here.

516 3.2.2 Single column equilibrium transport model

517 In its default mode, `UserCalc` solves the one-dimensional steady-state equilibrium
 518 transport model described in Spiegelman (2000). Below we will initialize the model,
 519 solve for a single column and plot the results.

520 First we set the physical parameters for the upwelling column and initial conditions:

```
[5]: # Maximum melt porosity:
phi0 = 0.008

# Solid upwelling rate in cm/yr. (to be converted to km/yr. in the driver_
  ↪function):
W0 = 3.

# Permeability exponent:
n = 2.

# Solid and liquid densities in kg/m3:
rho_s = 3300.
rho_f = 2800.

# Initial activity values (default is 1.0):
alpha0_238U = 1.
alpha0_235U = 1.
```

521

```

alpha0_230Th = 1.
alpha0_226Ra = 1.
alpha0_231Pa = 1.
alpha0_all = np.array([alpha0_238U, alpha0_230Th, alpha0_226Ra,
→alpha0_235U, alpha0_231Pa])

```

522

523 Next, we initialize the default equilibrium model:

```
[6]: us_eq = UserCalc.UserCalc(df)
```

524

525 and run the model for the input code and display the results for the final predicted
526 melt composition in List 1:

```
[7]: df_out_eq = us_eq.solve_all_1D(phi0,n,W0,alpha0_all)
df_out_eq.tail(n=1)
```

527

```
[7]:
      P      z      F      phi (230Th/238U) (226Ra/230Th) (231Pa/235U) Uf_238U
40  0.0  0.0  0.2  0.008  1.164941    1.590091    2.10557   -3.121055

      Uf_230Th Uf_226Ra  Us_238U  Us_230Th  Us_226Ra  Uf_235U  Uf_231Pa
40 -3.556171 -8.613841 -3.121055 -3.556171 -8.613841 -3.121909 -9.179718

      Us_235U  Us_231Pa
40 -3.121909 -9.179718

```

528

529 **List 1.** Model output results for the equilibrium melting scenario tested above.

530 The cell below produces Figure 2, which shows the model results with depth:

```
[8]: fig = UserCalc.plot_1Dcolumn(df_out_eq)
```

531


```
[11]: fig, axes = UserCalc.plot_1Dcolumn(df_out)
      for s in ['(230Th/238U)', '(226Ra/230Th)', '(231Pa/235U)']:
          axes[2].plot(df_out_eq[s], df_out['P'], '--', color='grey')
      axes[2].set_title('Da = {}'.format(us_diseq.Da))
      plt.show()
```

544

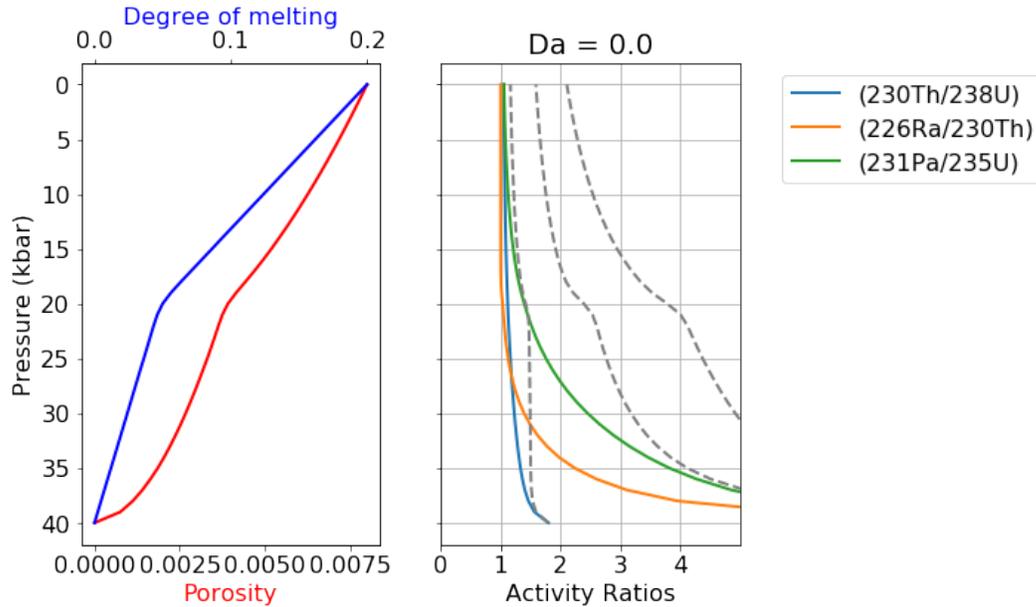


Figure 3: Disequilibrium model output results for the degree of melting, residual melt porosity, and activity ratios ($^{230}\text{Th}/^{238}\text{U}$), ($^{226}\text{Ra}/^{230}\text{Th}$), and ($^{231}\text{Pa}/^{235}\text{U}$) as a function of pressure, for the Damköhler number shown ($Da = 0$). For comparison, the dashed grey curves show solutions for the equilibrium transport model.

545 The dashed grey curves in Figure 3 illustrate the equilibrium transport solution,
 546 which is significantly different from the disequilibrium solution. If we increase the
 547 value of Da , however, the disequilibrium transport solution should converge to-
 548 wards the equilibrium scenario. To illustrate this, below we calculate the result for
 549 $Da = 1$:

```
[12]: # Reset the Da number in the reactive transport model to 1:
      us_diseq.Da=1.

      # Recalculate the model:
      df_out = us_diseq.solve_all_1D(phi0,n,W0,alpha0_all)
      df_out_eq.tail(n=1)
```

550

```
[12]:      P      z      F      phi (230Th/238U) (226Ra/230Th) (231Pa/235U) Uf_238U
      40 0.0  0.0  0.2  0.008  1.164941      1.590091      2.10557 -3.121055

      Uf_230Th Uf_226Ra Us_238U Us_230Th Us_226Ra Uf_235U Uf_231Pa
      40 -3.556171 -8.613841 -3.121055 -3.556171 -8.613841 -3.121909 -9.179718
```

551

```
Us_235U Us_231Pa
40 -3.121909 -9.179718
```

552

553

554

List 3. Model output results for the disequilibrium melting scenario tested above, where $Da = 1$.

```
[13]: fig, axes = UserCalc.plot_1Dcolumn(df_out)
for s in ['(230Th/238U)', '(226Ra/230Th)', '(231Pa/235U)']:
    axes[2].plot(df_out_eq[s], df_out['P'], '--', color='grey')
axes[2].set_title('Da = {}'.format(us_diseq.Da))
plt.show()
```

555

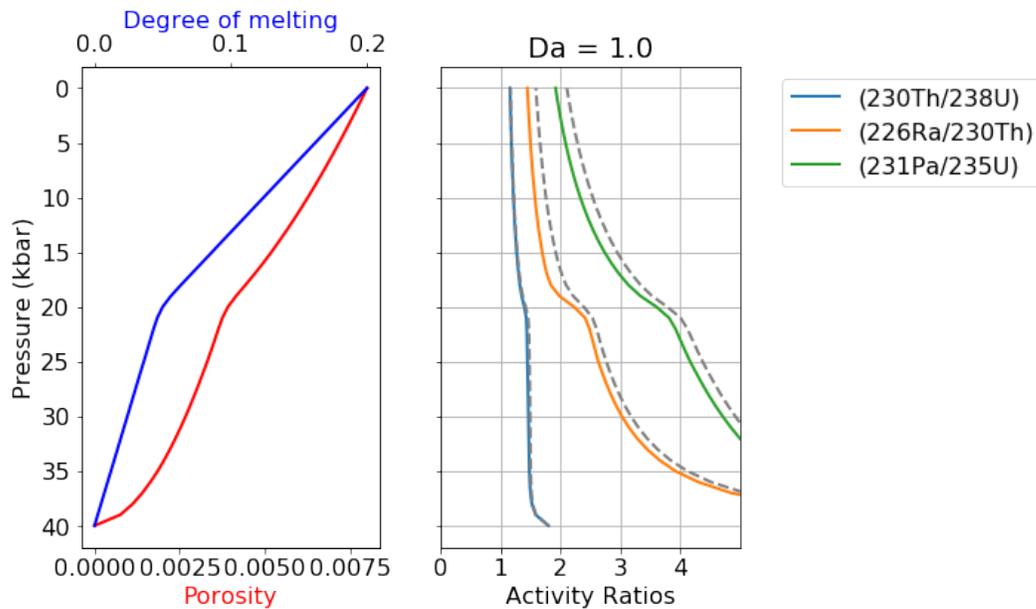


Figure 4: Disequilibrium model output as in Figure 3, but for $Da = 1$.

556

557

The outcome of the above calculation (Figure 4, List 3) approaches the equilibrium scenario more closely, as predicted. Below is an additional comparison for $Da = 10$:

```
[14]: # Reset the Da number in the reactive transport model to 10:
us_diseq.Da=10.

# Recalculate and plot the model:
df_out = us_diseq.solve_all_1D(phi0,n,W0,alpha0_all)
fig, axes = UserCalc.plot_1Dcolumn(df_out)
for s in ['(230Th/238U)', '(226Ra/230Th)', '(231Pa/235U)']:
    axes[2].plot(df_out_eq[s], df_out['P'], '--', color='grey')
axes[2].set_title('Da = {}'.format(us_diseq.Da))
plt.show()
```

558

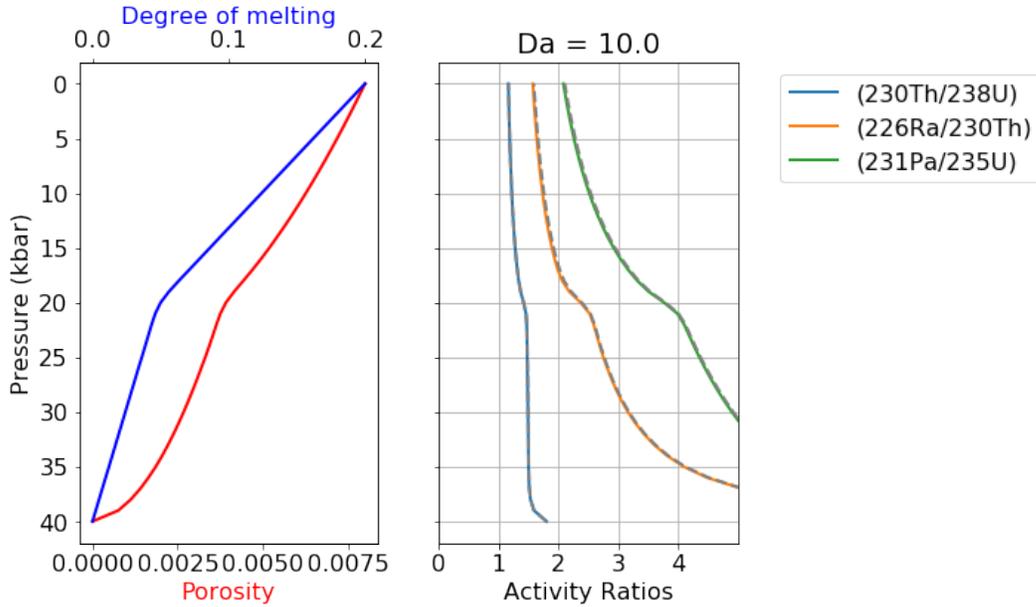


Figure 5: Disequilibrium model output as in Figure 3, but for $Da = 10$.

559 For $Da = 10$ (Figure 5), the activity ratios in the melt are indistinguishable from the
 560 equilibrium calculation, suggesting that a Damköhler number of 10 is sufficiently
 561 high for a melting system to approach chemical equilibrium, and illustrating that
 562 the equilibrium model of Spiegelman and Elliott (1993) and Spiegelman (2000) is
 563 the limiting case for the more general disequilibrium model presented here. For this
 564 problem, equilibrium transport always provides an upper bound on activity ratios.

565 3.2.4 Stable element concentrations

566 For a stable element, i.e., $\lambda_i = 0$, Spiegelman and Elliott (1993) showed that the
 567 equilibrium melting model reduces identically to simple batch melting (Shaw, 1970),
 568 while the disequilibrium model with $Da = 0$ is equivalent to true fractional melting.
 569 This presents a useful test of the calculator that verifies the program is correctly calcu-
 570 lating stable concentrations. To simulate stable element concentrations for U, Th,
 571 Ra, and Pa during equilibrium melting, we can use the same input file example as
 572 above and simply test the scenario where λ_i values are equal to zero.

573 First, we impose a "stable" condition that changes all decay constants $\lambda_i = 0$:

```
574 [15]: us_eq = UserCalc.UserCalc(df, stable=True)
df_out_eq = us_eq.solve_all_1D(phi0, n, W0, alpha0_all)
df_out_eq.tail(n=1)
```

```
[15]:      P      z      F      phi (230Th/238U) (226Ra/230Th) (231Pa/235U) Uf_238U
40  0.0  0.0  0.2  0.008  1.003937  1.015919  1.019959  -3.120895

      Uf_230Th Uf_226Ra  Us_238U  Us_230Th  Us_226Ra  Uf_235U  Uf_231Pa
40  -3.704753 -9.21042 -3.120895 -3.704753 -9.21042 -3.120895 -9.903528

      Us_235U  Us_231Pa
40  -3.120895 -9.903528
```

575

576 **List 4.** Model output results for equilibrium porous flow melting where $\lambda_i = 0$,
 577 simulating stable element behavior for U, Th, Ra, and Pa and thus true (instan-
 578 taneous) batch melting.

579 For comparison with the results in List 4, we can use the batch melting equation
 580 (Shaw, 1970) to calculate the concentrations of U, Th, Ra, and Pa using the input
 581 values in Table 2 for $F(z)$ and D_i , where:

$$582 \quad \frac{c_i^f}{c_i^0} = \frac{1}{F + D_i(1 - F)} \quad (61)$$

583 and determine radionuclide activities for the batch melt using the definition of the
 584 activity a for a nuclide i :

$$585 \quad a_i = \lambda_i c_i^f \quad (62)$$

586 and the initial nuclide activities a_i^0 , such that:

$$587 \quad a_i = \frac{a_i^0}{F + D_i(1 - F)} \quad (63)$$

588 As the activity ratios in List 5 illustrate, the outcomes of this simple batch melting
 589 equation are identical to those produced by the model for equilibrium transport and
 590 $\lambda = 0$.

```

16]: df_batch=df[['P', 'F', 'DU', 'DTh', 'DRa', 'DPa']]
df_batch['(230Th/238U)'] = (alpha0_all[1]/(df_batch.F-df_batch.F*df_batch.
    ↳DTh+df_batch.DTh))/(alpha0_all[0]/(df_batch.F-df_batch.F*df_batch.
    ↳DU+df_batch.DU))
df_batch['(226Ra/230Th)'] = (alpha0_all[2]/(df_batch.F-df_batch.
    ↳F*df_batch.DRa+df_batch.DRa))/(alpha0_all[1]/(df_batch.F-df_batch.
    ↳F*df_batch.DTh+df_batch.DTh))
df_batch['(231Pa/235U)'] = (alpha0_all[4]/(df_batch.F-df_batch.F*df_batch.
    ↳DPa+df_batch.DPa))/(alpha0_all[3]/(df_batch.F-df_batch.F*df_batch.
    ↳DU+df_batch.DU))

# Extract columns and concatenate dataframes
cols = ['P', 'F', '(230Th/238U)', '(226Ra/230Th)', '(231Pa/235U)']
df_compare = pd.concat([ df_batch[cols].tail(1), df_out_eq[cols].tail(1)])
df_compare['model'] = ['Batch Melting', 'Equilibrium Transport: stable_
    ↳elements']
df_compare.set_index('model')
591

```

```

16]:
model
Batch Melting          0.0 0.2   1.003937   1.015919
Equilibrium Transport: stable elements 0.0 0.2   1.003937   1.015919

(231Pa/235U)
model
592

```

Batch Melting 1.019959
 Equilibrium Transport: stable elements 1.019959

593

594 **List 5.** Simple batch melting calculation results using the methods of Shaw
 595 (1970), demonstrating identical activity ratio results to those calculated using
 596 the equilibrium transport model with $\lambda_i = 0$.

597 Similarly, we can also determine pure disequilibrium melting using the disequilib-
 598 rium transport model with $\lambda_i = 0$. A simple fractional melting problem is easiest
 599 to test using constant melt productivity and partitioning behavior, so here we test a
 600 simplified, one-layer scenario with constant dF/dz and D_i values:

```
[17]: input_file_2 = 'data/simple_sample.csv'
df_test = pd.read_csv(input_file_2, skiprows=1, dtype=float)
UserCalc.plot_inputs(df_test)
df_test.tail(n=1)
```

601

[17]: P F Kr DU DTh DRa DPa
 40 0.0 0.0964 1.0 0.009 0.005 0.00002 0.00001

602

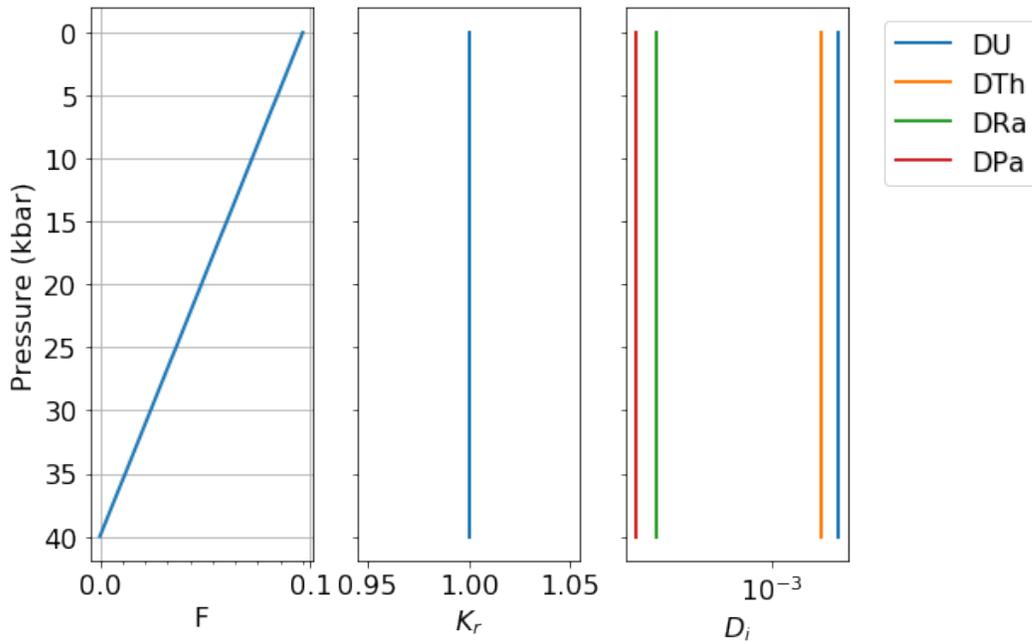


Figure 6: Simple alternative input file with constant melt productivity and constant solid/melt partitioning, used here to test pure fractional melting outputs.

603 We note that numerical ODE solvers may not successfully solve for pure fractional
 604 melting with $Da = 0$ and stable elements, because the resulting extreme changes in
 605 solid concentrations for highly incompatible elements are difficult to resolve using
 606 numerical methods. Stable solutions can nonetheless be obtained for very small
 607 values of Da that approach $Da = 0$, and such solutions still provide a useful test

608 of the disequilibrium transport model. Here we use $Da = 10^{-10}$; for such low Da
 609 values, the liquid closely approaches the composition of an accumulated fractional
 610 melt, and although the liquid and solid outcomes are slightly different from pure
 611 fractional melting, the solid is still essentially depleted of all incompatible nuclides.

```
[18]: us_diseq_test = UserCalc.UserCalc(df_test, model=UserCalc.  
      ↪DisequilTransport, stable=True, Da=1.e-10)
```

612

```
[19]: df_diseq_test = us_diseq_test.solve_all_1D(phi0,n,W0,alpha0_all)
```

613

614 Similar to our approach for equilibrium and batch melting, we can compare the
 615 results of disequilibrium transport for stable elements with pure fractional melting
 616 for constant partition coefficients using the definition of aggregated fractional melt
 617 concentrations (Figure 7):

$$618 \quad \frac{c_i^s}{c_i^{s,0}} = (1-F)^{1/D_i-1} \quad (64)$$

$$619 \quad \frac{c_i^f}{c_i^{f,0}} = \frac{D_i}{F} \left(1 - (1-F)^{1/D_i} \right) \quad (65)$$

620 or in log units:

$$621 \quad U_i^s = (1/D_i - 1) \log(1-F) \quad (66)$$

$$622 \quad U_i^f = \log \left(1 - (1-F)^{1/D_i} \right) + \log \left(\frac{D_i}{F} \right) \quad (67)$$

```
[20]: df_frac=df_test[['P','F','DU','DTh','DRa','DPa']]  
df_frac['(230Th/238U)'] = ((alpha0_all[1]/df_frac.F)*(1.-(1.-df_frac.  
  ↪F)**(1./df_frac.DTh))) / ((alpha0_all[0]/df_frac.F)*(1.-(1.-df_frac.  
  ↪F)**(1./df_frac.DU)))  
df_frac['(226Ra/230Th)'] = ((alpha0_all[2]/df_frac.F)*(1.-(1.-df_frac.  
  ↪F)**(1./df_frac.DRa))) / ((alpha0_all[1]/df_frac.F)*(1.-(1.-df_frac.  
  ↪F)**(1./df_frac.DTh)))  
df_frac['(231Pa/235U)'] = ((alpha0_all[4]/df_frac.F)*(1.-(1.-df_frac.  
  ↪F)**(1./df_frac.DPa))) / ((alpha0_all[3]/df_frac.F)*(1.-(1.-df_frac.  
  ↪F)**(1./df_frac.DU)))
```

623

```
[21]: fig, axes = UserCalc.plot_1Dcolumn(df_diseq_test)  
for s in ['(230Th/238U)', '(226Ra/230Th)', '(231Pa/235U)']:  
    axes[2].plot(df_frac[s], df_diseq_test['P'], '--', color='black')  
plt.show()
```

624

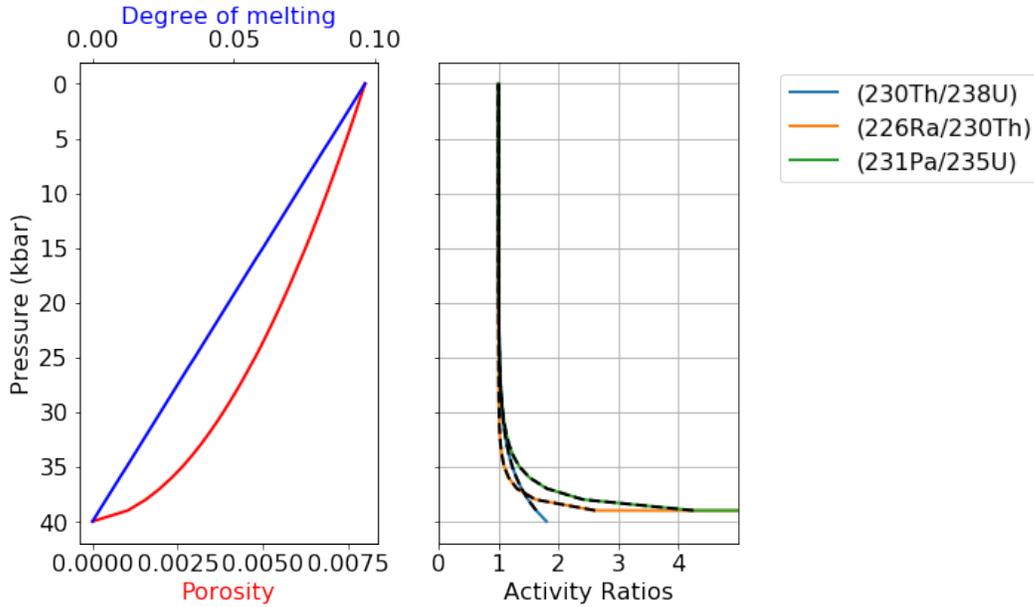


Figure 7: Model output results for the degree of melting, residual melt porosity, and activity ratios ($^{230}\text{Th}/^{238}\text{U}$), ($^{226}\text{Ra}/^{230}\text{Th}$), and ($^{231}\text{Pa}/^{235}\text{U}$) as a function of pressure. The solid curves plot the results of pure fractional melting for stable elements, while the dashed black curves illustrate the outcomes of the disequilibrium transport model with $Da = 10^{-10}$ and $\lambda_i = 0$. The outcomes of the two methods are indistinguishable.

625 3.2.5 Considering lithospheric transport scenarios

626 In mantle decompression melting scenarios, melting is expected to cease in the
 627 shallow, colder part of the regime where a lithospheric layer is present. The effects
 628 of cessation of melting prior to reaching the surface can be envisioned as affecting
 629 magma compositions in a number of ways, some of which could be calculated using
 630 the models presented here by setting $dF = 0$.

631 There are, however, several limitations when using our transport models to simulate
 632 lithospheric melt transport in this way, as the model equations are written to track
 633 steady-state decompression and melting. The first limitation is thus the underlying
 634 assumption that the solid is migrating and experiencing progressive melt depletion
 635 in the model, while the solid lithosphere should in fact behave as a rigid matrix
 636 that does not experiencing upwelling. For the disequilibrium transport model with
 637 $Da = 0$, no chemical reequilibration occurs while $dF = 0$, so the lack of solid mi-
 638 gration after the cessation of melting does not pose a problem; instead, in the pure
 639 disequilibrium transport case, imposing $dF = 0$ simply allows for radioactive decay
 640 and ingrowth during transport through the lithospheric layer.

641 The equilibrium transport model, on the other hand, permits full equilibration
 642 even if $dF = 0$, but the liquid composition does not directly depend on the solid
 643 concentration, $c_i^s(z)$, so ongoing chemical reequilibration between the liquid and a
 644 modified lithospheric solid could be simulated by modifying the bulk solid/liquid
 645 partition coefficients D_i . However, the underlying model assumes that the liquid
 646 with mass proportion F_{max} reequilibrates with the solid matrix in a steady-state
 647 transport regime, at the maximum reference porosity, which may not accurately
 648 simulate the transport regime through the fixed lithosphere with no melting. Be-
 649 cause it does not directly consider mineral abundances or compositions, the model

650 also does not account for complexities such as low temperature melt-rock reaction
651 or mineral growth.

652 The case of the scaled disequilibrium transport model with $Da > 0$ is the most
653 complex, since the model directly calculates reequilibration of the liquid with a
654 progressively melting solid layer, and thus may not accurately simulate transport
655 through the fixed solid lithosphere. We advise that if the model is used in this way,
656 the results must be interpreted with additional caution.

657 Finally, calculating a given transport model through the upwelling asthenosphere
658 and into a fixed overlying lithospheric layer neglects an additional, significant limi-
659 tation: namely that melt-rock interactions, and thus the magma transport style, may
660 be different in the lithosphere than in the melting asthenosphere. As noted above,
661 this could also include low-temperature reactions and the growth of new mineral
662 phases. While it is not possible to change transport models during a single 1D run
663 in the current implementation, one alternative approach is to change the relative
664 permeability, K_r , in the lithosphere, in addition to modifying the bulk partition co-
665 efficients to reflect lithospheric values. It may also be possible to run a separate,
666 second-stage lithospheric calculation with modified input parameters and revised
667 liquid porosity constraints, but this option is not currently implemented and would
668 require an expansion of the current model.

669 Despite these caveats, there are some limited scenarios where users may wish to
670 simulate equilibrium or disequilibrium magma transport through a capping layer
671 with constant $dF = 0$, constant $\phi = \phi_0$, and revised D_i values for a modified layer
672 mineralogy. The cells below provide options for modifying the existing input data
673 table to impose such a layer. The first cell identifies a final melting pressure P_{Lithos} ,
674 which is defined by the user in kbar. This value can be set to 0.0 if no lithospheric
675 cap is desired; in the example below, it has been set at 5.0 kbar. There are two overall
676 options for how this final melting pressure could be used to modify the input data.
677 The first option (implemented in the Supplementary Materials but not tested here)
678 simply deletes all lines in the input dataframe for depths shallower than P_{Lithos} . This
679 is a straightforward option for a one-dimensional column scenario, where melting
680 simply stops at the base of the lithosphere and the composition of the melt product
681 is observed in that position. This is an effective way to limit further chemical inter-
682 actions after melting has ceased; it fails to account for additional radioactive decay
683 during lithospheric melt transport, but subsequent isotopic decay over a fixed trans-
684 port time interval could then be calculated using the radioactive decay equations for
685 U-series nuclides.

686 A second option, shown here to demonstrate outcomes, changes the degree of melt-
687 ing increments (dF) to a value of 0 for all depths shallower than P_{Lithos} , but allows
688 model calculations to continue at shallower depths. This is preferable if the user
689 aims to track additional radioactive decay and/or chemical exchange after melting
690 has ceased and during subsequent transport through the lithospheric layer, and
691 shall be explored further below.

```
692 [22]: Plithos = 5.0

        Pfinal = df.iloc[(df['P']-Plithos).abs().idxmin()]
        F_max = Pfinal[1].tolist()
        df.loc[(df['P'] < Plithos),['F']] = F_max
```

693 For equilibrium transport scenarios, the cell below offers one possible option for
694 modifying lithospheric solid/melt bulk partition coefficients. We note that if the

695 disequilibrium transport model is used with $Da = 0$ (i.e., pure chemical disequilibrium)
 696), this cell is not necessary.

697 The option demonstrated below imposes new, constant melt-rock partition coef-
 698 ficients during lithospheric transport. These values are assumed to be fixed. An
 699 alternative choice, included in the Supplementary Materials, instead fixes the shal-
 700 lower lithospheric solid/melt bulk partition coefficients such that they are equal to
 701 D_i values at the depth where melting ceased (i.e., P_{Lithos}).

```
[23]: # Define new bulk solid/liquid partition coefficients for the
      ↪ lithospheric layer:
      D_U_lith = 0.002
      D_Th_lith = 0.006
      D_Ra_lith = 0.00002
      D_Pa_lith = 0.00001

      # Implement the changed values defined above:
      df.loc[(df['P'] < Plithos),['DU']] = D_U_lith
      df.loc[(df['P'] < Plithos),['DTh']] = D_Th_lith
      df.loc[(df['P'] < Plithos),['DRa']] = D_Ra_lith
      df.loc[(df['P'] < Plithos),['DPa']] = D_Pa_lith
```

702
 703 Following any changes implemented above, the cells below will process and display
 704 the refined input data (Figure 8, Table 3).

```
[24]: UserCalc.plot_inputs(df)
```

705

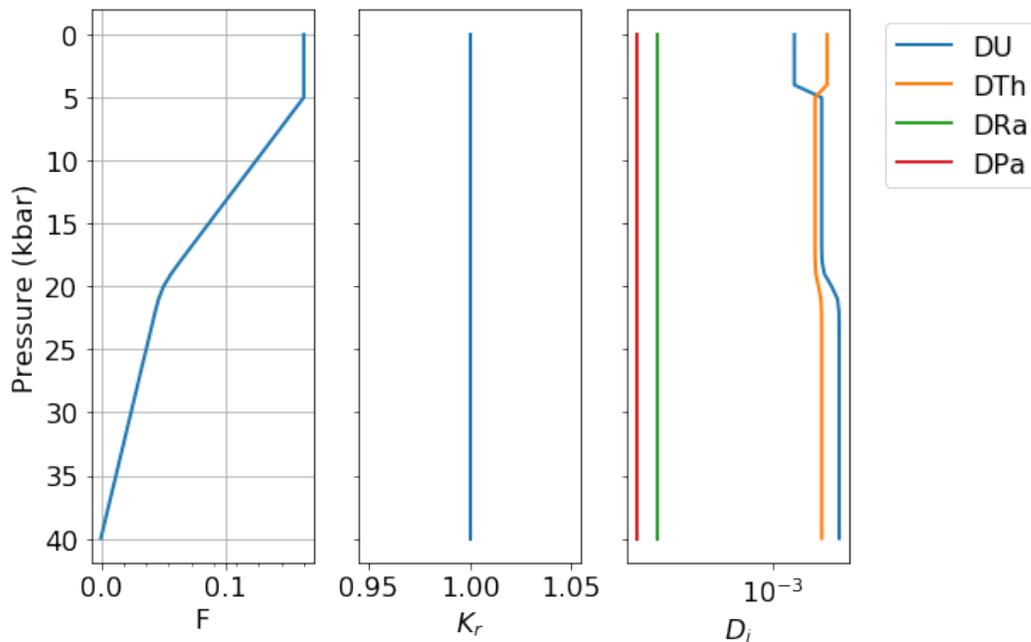


Figure 8: Diagrams showing input parameters F , K_r , and D_i as a function of pressure, for the example input file and modified lithospheric conditions.

[25] : df

706

Table 3: Input data table for an example scenario with modified lithospheric transport conditions, showing pressures in kbar (P), degree of melting (F), permeability coefficient (K_r), and bulk solid/melt partition coefficients (D_i) for the elements of interest, U, Th, Ra, and Pa.

	P	F	Kr	DU	DTh	DRa	DPa
0	40.0	0.00000	1.0	0.00900	0.00500	0.00002	0.00001
1	39.0	0.00241	1.0	0.00900	0.00500	0.00002	0.00001
2	38.0	0.00482	1.0	0.00900	0.00500	0.00002	0.00001
3	37.0	0.00723	1.0	0.00900	0.00500	0.00002	0.00001
4	36.0	0.00964	1.0	0.00900	0.00500	0.00002	0.00001
5	35.0	0.01210	1.0	0.00900	0.00500	0.00002	0.00001
6	34.0	0.01450	1.0	0.00900	0.00500	0.00002	0.00001
7	33.0	0.01690	1.0	0.00900	0.00500	0.00002	0.00001
8	32.0	0.01930	1.0	0.00900	0.00500	0.00002	0.00001
9	31.0	0.02170	1.0	0.00900	0.00500	0.00002	0.00001
10	30.0	0.02410	1.0	0.00900	0.00500	0.00002	0.00001
11	29.0	0.02650	1.0	0.00900	0.00500	0.00002	0.00001
12	28.0	0.02890	1.0	0.00900	0.00500	0.00002	0.00001
13	27.0	0.03130	1.0	0.00900	0.00500	0.00002	0.00001
14	26.0	0.03370	1.0	0.00900	0.00500	0.00002	0.00001
15	25.0	0.03620	1.0	0.00900	0.00500	0.00002	0.00001
16	24.0	0.03860	1.0	0.00900	0.00500	0.00002	0.00001
17	23.0	0.04100	1.0	0.00899	0.00500	0.00002	0.00001
18	22.0	0.04340	1.0	0.00893	0.00498	0.00002	0.00001
19	21.0	0.04610	1.0	0.00852	0.00488	0.00002	0.00001
20	20.0	0.05000	1.0	0.00700	0.00450	0.00002	0.00001
21	19.0	0.05610	1.0	0.00548	0.00412	0.00002	0.00001
22	18.0	0.06340	1.0	0.00507	0.00402	0.00002	0.00001
23	17.0	0.07100	1.0	0.00501	0.00400	0.00002	0.00001
24	16.0	0.07860	1.0	0.00500	0.00400	0.00002	0.00001
25	15.0	0.08620	1.0	0.00500	0.00400	0.00002	0.00001
26	14.0	0.09370	1.0	0.00500	0.00400	0.00002	0.00001
27	13.0	0.10133	1.0	0.00500	0.00400	0.00002	0.00001
28	12.0	0.10892	1.0	0.00500	0.00400	0.00002	0.00001
29	11.0	0.11651	1.0	0.00500	0.00400	0.00002	0.00001
30	10.0	0.12410	1.0	0.00500	0.00400	0.00002	0.00001
31	9.0	0.13169	1.0	0.00500	0.00400	0.00002	0.00001
32	8.0	0.13928	1.0	0.00500	0.00400	0.00002	0.00001
33	7.0	0.14687	1.0	0.00500	0.00400	0.00002	0.00001
34	6.0	0.15446	1.0	0.00500	0.00400	0.00002	0.00001
35	5.0	0.16205	1.0	0.00500	0.00400	0.00002	0.00001
36	4.0	0.16205	1.0	0.00500	0.00400	0.00002	0.00001
37	3.0	0.16205	1.0	0.00500	0.00400	0.00002	0.00001
38	2.0	0.16205	1.0	0.00500	0.00400	0.00002	0.00001
39	1.0	0.16205	1.0	0.00500	0.00400	0.00002	0.00001
40	0.0	0.16205	1.0	0.00500	0.00400	0.00002	0.00001

707 The cells below will rerun the end member models for the modified lithospheric
708 input file. First, equilibrium transport:

```
709 [26]: us_eq = UserCalc.UserCalc(df,stable=False)
df_out_eq = us_eq.solve_all_1D(phi0,n,W0,alpha0_all)
```

710 And second, for disequilibrium transport with $Da = 0$:

```
711 [27]: us_diseq = UserCalc.UserCalc(df,model=UserCalc.
↳DisequilTransport, Da=0,stable=False)
df_out_diseq = us_diseq.solve_all_1D(phi0,n,W0,alpha0_all)
```

712 List 6 below displays the activity ratios determined for the final melt compositions
713 at the end of the two simulations (i.e., the tops of the one-dimensional melting
714 columns).

```
715 [28]: df_compare = pd.concat([df_out_eq.tail(n=1), df_out_diseq.tail(n=1)])
df_compare['model'] = ['Equilibrium Transport', 'Disequilibrium Transport']
df_compare.set_index('model')
```

```
[28]:
```

	P	z	F	phi	(230Th/238U)	
model						
Equilibrium Transport	0.0	0.0	0.16205	0.008	1.015792	
Disequilibrium Transport	0.0	0.0	0.16205	0.008	1.039704	
		(226Ra/230Th)	(231Pa/235U)	Uf_238U	Uf_230Th	
model						
Equilibrium Transport		1.894057	1.792975	-2.901132	-3.473250	
Disequilibrium Transport		1.000828	1.034719	-2.891833	-3.440684	
		Uf_226Ra	Us_238U	Us_230Th	Us_226Ra	Uf_235U
model						
Equilibrium Transport	-8.355990	-2.901132	-3.473250	-8.355990	-2.902001	
Disequilibrium Transport	-8.96132	-30.351986	-30.353121	-30.353146	-2.88492	
		Uf_231Pa	Us_235U	Us_231Pa		
model						
Equilibrium Transport	-9.120520	-2.902001	-9.120520			
Disequilibrium Transport	-9.653185	-30.272812	-30.272749			

717 **List 6.** Model output results for the disequilibrium ($Da = 0$) melting scenarios
718 tested here, with modified lithospheric input conditions.

719 The following cell generates Figure 9, which illustrates outcomes with depth for the
720 equilibrium and disequilibrium transport models. The model outcomes for the two
721 transport scenarios are notably different, particularly for the shorter-lived isotopic
722 pairs.

```
723 [29]: fig, axes = UserCalc.plot_1Dcolumn(df_out_diseq)
axes[2].set_prop_cycle(None)
for s in ['(230Th/238U)', '(226Ra/230Th)', '(231Pa/235U)']:
axes[2].plot(df_out_eq[s],df_out['P'],'--')
```

```
axes[2].set_title('Da = {}'.format(us_diseq.Da))
plt.show()
```

724

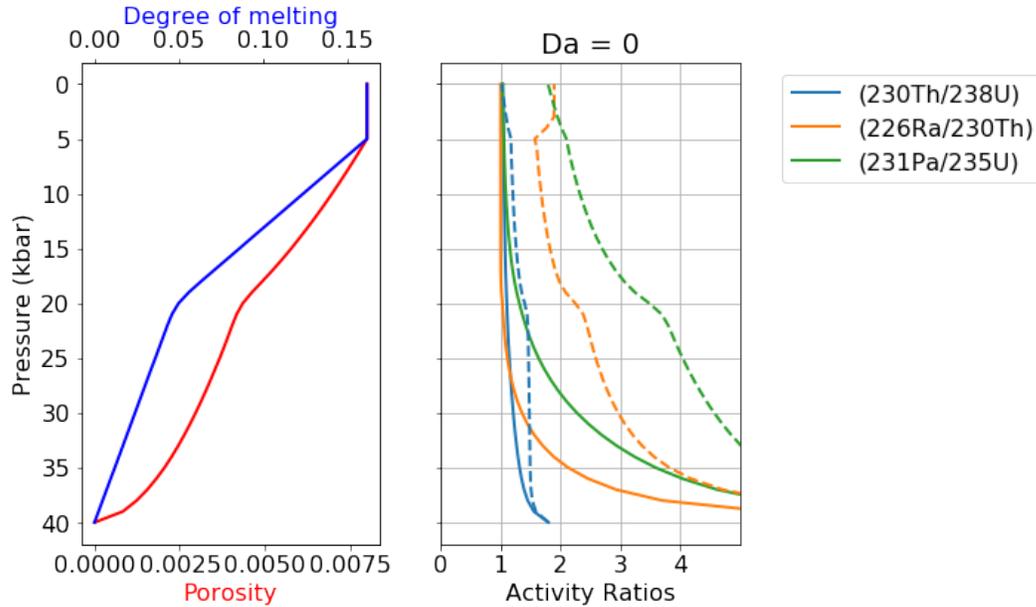


Figure 9: Comparison of equilibrium (dashed) and disequilibrium ($Da = 0$; solid) transport model output results for the degree of melting, residual melt porosity, and activity ratios ($^{230}\text{Th}/^{238}\text{U}$), ($^{226}\text{Ra}/^{230}\text{Th}$), and ($^{231}\text{Pa}/^{235}\text{U}$) as a function of pressure, for the modified lithospheric transport scenario explored above. Symbols and lines as in Figure 3.

725

3.3 Batch operations

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

For many applications, it is preferable to calculate an ensemble of model scenarios over a range of input parameters directly related to questions about the physical constraints on melt generation, such as the maximum residual or reference melt porosity (ϕ_0) and the solid mantle upwelling rate (W_0). The cells below determine a series of one-dimensional column results for the the equilibrium transport model and the parameters defined above (that is, the input conditions shown in Table 3 with $n = 2$, $\rho_s = 3300 \text{ kg/m}^3$, and $\rho_f = 2800 \text{ kg/m}^3$), but over a range of values for ϕ_0 and W_0 ; these results are then shown in a series of figures. The user can select whether to define the specific ϕ_0 and W_0 values as evenly spaced log grid intervals (option 1) or with manually specified values (option 2). As above, all upwelling rates are entered in units of cm/yr. We note that because some of these models tend to be stiff and the Radau solver is relatively computationally expensive, the batch operations below may require a few minutes of computation time for certain scenarios. Here we show the results for the default equilibrium model over a range of selected ϕ_0 and W_0 values:

[30]:

```
# Option 1 (evenly spaced log grid intervals):
# phi0 = np.logspace(-3, -2, 11)
# W0 = np.logspace(-1, 1, 11)

# Option 2 (manual selection of values):
```

741

```

phi0 = np.array([0.001, 0.002, 0.005, 0.01])
W0 = np.array([0.5, 1., 2., 5., 10., 20., 50.])

import time
tic = time.perf_counter()
toc = time.perf_counter()

# Calculate the U-238 decay chain grid values:
act = us_eq.solve_grid(phi0, n, W0, us_eq.D_238, us_eq.lambdas_238, us_eq.
    ↪alphas_238)
Th = act[0]
Ra = act[1]
df = pd.DataFrame(Th)
df = pd.DataFrame(Ra)

```

742

```

W = 0.5 . . . . .
W = 1.0 . . . . .
W = 2.0 . . . . .
W = 5.0 . . . . .
W = 10.0 . . . . .
W = 20.0 . . . . .
W = 50.0 . . . . .

```

```

[31]: # Calculate the U-235 decay chain grid values:
act_235 = us_eq.solve_grid(phi0, n, W0, us_eq.D_235, us_eq.lambdas_235, ↪
    ↪us_eq.alphas_235)
Pa = act_235[0]
df = pd.DataFrame(Pa)

```

743

```

W = 0.5 . . . . .
W = 1.0 . . . . .
W = 2.0 . . . . .
W = 5.0 . . . . .
W = 10.0 . . . . .
W = 20.0 . . . . .
W = 50.0 . . . . .

```

744 The figures below illustrate the batch model results in a variety of ways. First, each
 745 isotopic activity ratio is contoured in ϕ_0 vs. W_0 space (Figure 10), using figures sim-
 746 ilar to the contour plots of Spiegelman (2000). The model outcomes for W_0 and ϕ_0
 747 values are also contoured as mesh "grids" in activity ratio-activity ratio plots (Fig-
 748 ure 11). These diagrams show the outcomes for model runs with a given W_0 and ϕ_0
 749 value at each grid intersection point, and each curve shows outcomes for a constant
 750 W_0 value with variable ϕ_0 or vice versa, as indicated in the figure legend. Because
 751 this particular example shows results for the equilibrium transport model, and the
 752 input values for the shallow, spinel peridotite layer of the sample input file define
 753 $D_U < D_{Th}$, we note that some of the results exhibit $(^{230}\text{Th}/^{238}\text{U}) < 1.0$ in Figure 11.

```

[32]: UserCalc.plot_contours(phi0,W0,act, figsize=(12,12))

```

754

```

[33]: UserCalc.plot_contours(phi0,W0,act_235)

```

755

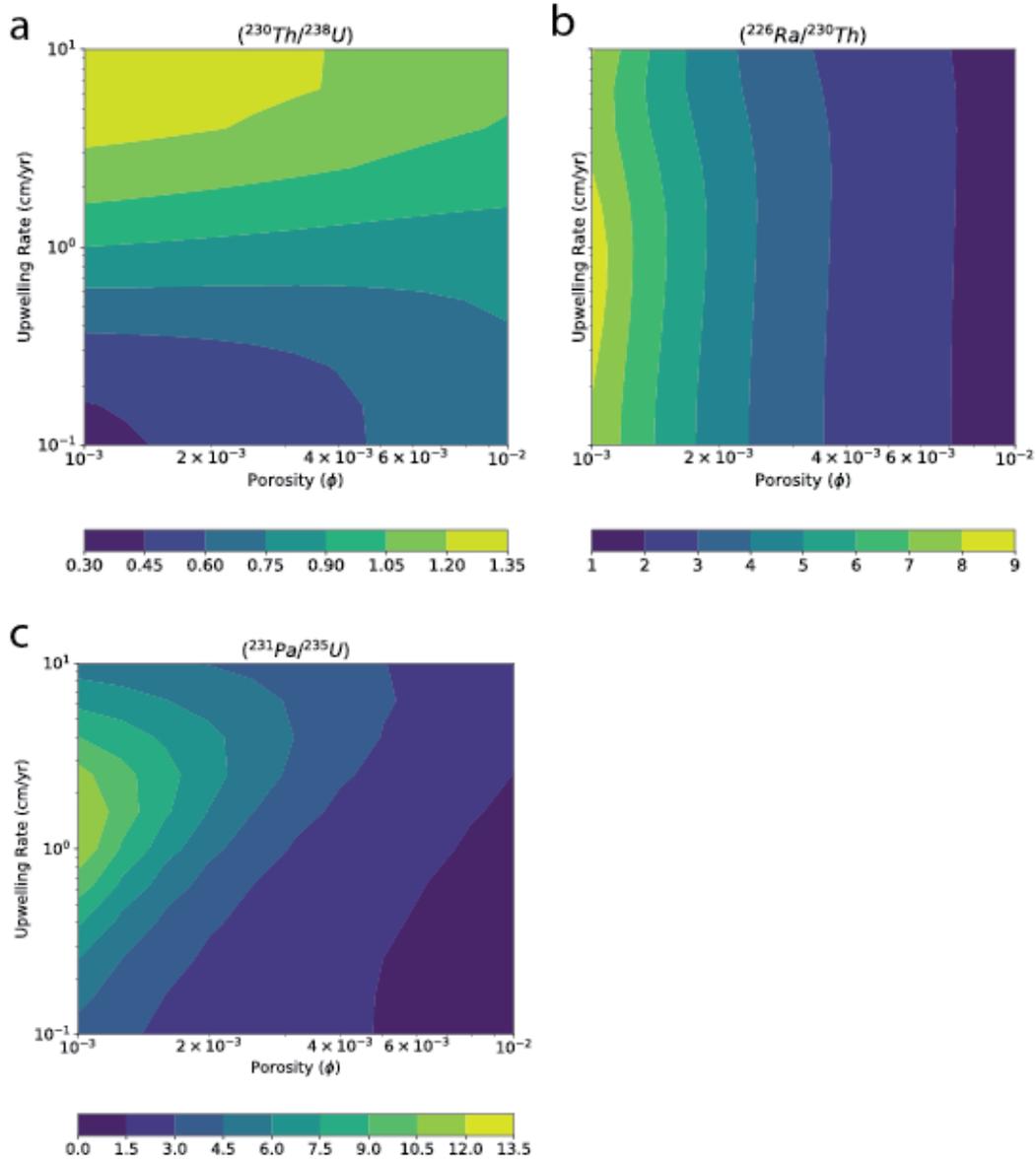


Figure 10: Diagrams of upwelling rate (W_0) vs. maximum residual melt porosity (ϕ) showing contoured activity ratios for (a) $(^{230}\text{Th}/^{238}\text{U})$, (b) $(^{226}\text{Ra}/^{230}\text{Th})$, and (c) $(^{231}\text{Pa}/^{235}\text{U})$.

[34]: `UserCalc.plot_mesh_Ra(Th,Ra,W0,phi0)`

756

[35]: `UserCalc.plot_mesh_Pa(Th,Pa,W0,phi0)`

757

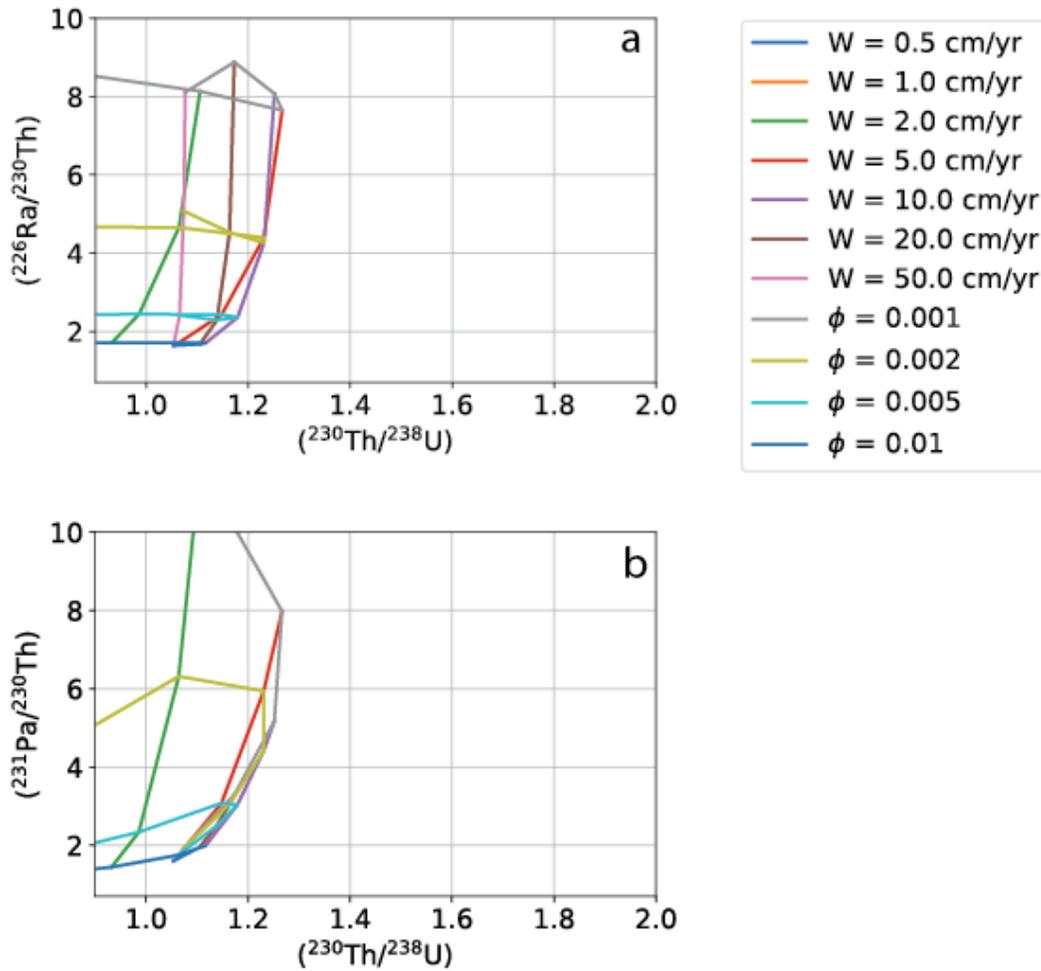


Figure 11: Diagrams showing (a) $(^{226}\text{Ra}/^{230}\text{Th})$ vs. $(^{230}\text{Th}/^{238}\text{U})$ and (b) $(^{231}\text{Pa}/^{230}\text{Th})$ vs. $(^{230}\text{Th}/^{238}\text{U})$ for the gridded upwelling rate (W_0) and maximum residual porosity (ϕ) values defined above.

758 4 Summary

759 We present pyUserCalc, an expanded, publicly available, open-source version of
 760 the UserCalc code for determining U-series disequilibria generated in basalts by
 761 one-dimensional, decompression partial melting. The model has been developed
 762 from conservation of mass equations with two-phase (solid and liquid) porous flow
 763 and permeability governed by Darcy's Law. The model reproduces the functionali-
 764 ty of the original UserCalc equilibrium porous flow calculator (Spiegelman, 2000)
 765 in pure Python code, and implements a new disequilibrium transport model. The
 766 disequilibrium transport code includes reactivity rate-limited chemical equilibration
 767 calculations controlled by a Damköhler number, Da . For stable elements with decay
 768 constants equal to zero, the equilibrium model reduces to batch melting and the
 769 disequilibrium transport model with $Da = 0$ to pure fractional melting. The method
 770 presented here can be extended to other applications in geochemical porous flow
 771 calculations in future work.

772 Acknowledgments

773 We thank two anonymous reviewers for thoughtful feedback that strengthened this
 774 manuscript. We thank K.W.W. Sims and P. Kelemen for initiating early discussions
 775 about creating a new porous flow disequilibrium transport calculator back in 2008.
 776 We also thank M. Ghiorso for inviting L. Elkins to join the ENKI working group
 777 and thereby catalyzing this fresh effort, and we further thank the working group
 778 for their helpful suggestions and feedback. L. Elkins received ENKI working group
 779 travel assistance that contributed to this research effort, and was supported by NSF
 780 award OCE-1658011. M. Spiegelman was supported by the ENKI NSF SI² award
 781 NSF-ACI1550337.

782 Data Availability Statement

783 The data set for this research consists of a code package, which
 784 is available in several ways: 1) in the supporting information, 2)
 785 through a binder container (at https://mybinder.org/v2/gl/ENKI-portal%2FpyUsercalc/master?filepath=pyUserCalc_manuscript.ipynb),
 786 and 3) in the ENKI GitLab data repository (<https://gitlab.com/ENKI-portal/pyUsercalc>), which can also be accessed at the ENKI cloud server
 787 (<https://server.enki-portal.org/hub/login>) with a free GitLab account (register
 788 at <https://gitlab.com/ENKI-portal>).

791 References

- 792 Aharonov, E., J. A. Whitehead, P. Kelemen, and M. Spiegelman (1995), Channel-
 793 ing instability of upwelling melt in the mantle, *Journal of Geophysical Research: Solid*
 794 *Earth*, **100**(B10), 20433-20450.
- 795 Bourdon, B., S. P. Turner, and N. M. Ribe (2005), Partial melting and upwelling rates
 796 beneath the Azores from a U-series isotope perspective, *Earth and Planetary Science*
 797 *Letters*, **239**, 42-56.
- 798 Elkins, L. J., B. Bourdon, and S. Lambart (2019), Testing pyroxenite versus peridot-
 799 tie sources for marine basalts using U-series isotopes, *Lithos*, **332-333**, 226-244, doi:
 800 210.1016/j.lithos.2019.1002.1011.
- 801 Feineman, M. D., and D. J. DePaolo (2003), Steady-state ²²⁶Ra/²³⁰Th disequilib-
 802 rium in mantle minerals: implications for melt transport rates in island arcs, *Earth*
 803 *and Planetary Science Letters*, **215**(3-4), 339-355.
- 804 Grose, C. J., and J. C. Afonso (2019), Chemical disequilibria, lithospheric thickness,
 805 and the source of ocean island basalts, *Journal of Petrology*, **60**(4), 755-790.
- 806 Iwamori, H. (1993), Dynamic disequilibrium melting model with porous flow and
 807 diffusion-controlled chemical equilibration, *Earth and Planetary Science Letters*, **114**(2-
 808 3), 301-313.
- 809 Iwamori, H. (1994), ²³⁸U-²³⁰Th-²²⁶Ra and ²³⁵U-²³¹Pa disequilibria produced by
 810 mantle melting with porous and channel flows, *Earth and Planetary Science Letters*,
 811 **125**, 1-16.
- 812 Jull, M., P. Kelemen, and K. Sims (2002), Consequences of diffuse and channelled
 813 porous melt migration on uranium series disequilibria., *Geochimica Et Cosmochimica*
 814 *Acta*, **66**, 4133-4148.
- 815 Kogiso, T., M. M. Hirschmann, and P. W. Reiners (2004), Length scales of mantle het-
 816 erogeneities and their relationship to ocean island basalt geochemistry, *Geochimica et*
 817 *Cosmochimica Acta*, **68**(2), 345-360.

- 818 Liang, Y., and B. Liu (2016), Simple models for disequilibrium fractional melting and
819 batch melting with application to REE fractionation in abyssal peridotites, *Geochimica et Cosmochimica Acta*, **173**, 181-197.
820
- 821 Lundstrom, C., J. Gill, and Q. Williams (2000), A geochemically consistent hypothesis
822 for MORB generation, *Chemical Geology*, **162**(2), 105-126.
- 823 McKenzie, D. (1985), Th-230-U-238 Disequilibrium and the Melting Processes beneath
824 Ridge Axes, *Earth and Planetary Science Letters*, **72**(2-3), 149-157.
- 825 Oliveira, B., J. C. Afonso, and R. Tilhac (2020), A disequilibrium reactive transport model for mantle magmatism, *Journal of Petrology*,
826 <https://doi.org/10.1093/petrology/egaa1067>.
827
- 828 Peate, D. W., and C. J. Hawkesworth (2005), U series disequilibria: insights into
829 mantle melting and the timescales of magma differentiation, *Reviews of Geophysics*,
830 **43**(1).
- 831 Qin, Z., F. Lu, and A. T. Anderson (1992), Diffusive reequilibration of melt and fluid
832 inclusions, *American Mineralogist*, **77**(5-6), 565-576.
- 833 Shaw, D. M. (1970), Trace element fractionation during anatexis, *Geochimica et
834 Cosmochimica Acta*, **34**(2), 237-243.
- 835 Sims, K. W. W., D. J. DePaolo, M. T. Murrell, W. S. Baldrige, S. Goldstein, D.
836 Clague, and M. Jull (1999), Porosity of the melting zone and variations in the solid
837 mantle upwelling rate beneath Hawaii: Inferences from U-238-Th-230-Ra-226 and
838 U-235-Pa-231 disequilibria, *Geochimica Et Cosmochimica Acta*, **63**(23-24), 4119-4138.
- 839 Sims, K. W. W., et al. (2002), Chemical and isotopic constraints on the generation
840 and transport of magma beneath the East Pacific Rise, *Geochimica Et Cosmochimica
841 Acta*, **66**(19), 3481-3504.
- 842 Spiegelman, M. (2000), UserCalc: a web-based uranium series calculator for magma
843 migration problems, *Geochemistry Geophysics Geosystems*, **1**(8), 1016.
- 844 Spiegelman, M., and P. Kenyon (1992), The requirements for chemical disequilibrium
845 during magma migration, *Earth and Planetary Science Letters*, **109**(3-4), 611-620.
- 846 Spiegelman, M., and T. Elliott (1993), Consequences of Melt Transport for Uranium
847 Series Disequilibrium in Young Lavas, *Earth and Planetary Science Letters*, **118**(1-4),
848 1-20.
- 849 Stracke, A., and B. Bourdon (2009), The importance of melt extraction for tracing
850 mantle heterogeneities, *Geochimica Et Cosmochimica Acta*, **73**, 218-238.
- 851 Stracke, A., A. Zindler, V. J. M. Salters, D. McKenzie, and K. Gronvold (2003), The
852 dynamics of melting beneath Theistareykir, northern Iceland, *Geochemistry Geophysics Geosystems*, **4**, 8513.
853
- 854 Van Orman, J. A., T. L. Grove, and N. Shimizu (2002a), Diffusive fractionation of
855 trace elements during production and transport of melting in the earth's upper
856 mantle, *Earth and Planetary Science Letters*, **198**, 93-112.
- 857 Van Orman, J. A., A. E. Saal, B. Bourdon, and E. H. Hauri (2006), Diffusive fractionation of U-series radionuclides during mantle melting and shallow-level melt-cumulate interaction, *Geochimica et Cosmochimica Acta*, **70**(18), 4797-4812.
858
859

- 860 Yang, Z., Sista, S., Elmer, J. W., and DebRoy, T. (2000), Three dimensional Monte
861 Carlo simulation of grain growth during GTA welding of titanium. *Acta Materialia*,
862 **48**(20), 4813-4825.
- 863 Zou, H., and A. Zindler (2000), Theoretical studies of ^{238}U - ^{230}Th - ^{226}Ra and ^{235}U -
864 ^{231}Pa disequilibria in young lavas produced by mantle melting, *Geochimica et Cos-*
865 *mochimica Acta*, **64**(10), 1809-1817.

pyUserCalc: A revised Jupyter notebook calculator for uranium-series disequilibria in basalts

Elkins, Lynne J.¹ and Spiegelman, Marc²

¹ University of Nebraska-Lincoln, Lincoln, NE, USA, lelkins@unl.edu ² Lamont-Doherty Earth Observatory of Columbia University, Palisades, NY, USA, mspieg@ldeo.columbia.edu

Additional Supporting Information (Files uploaded separately)

Caption for Data Set S1

Introduction

The provided file is a compressed directory containing the following files:

- README.md
- pyUserCalc_manuscript.ipynb
- pyUserCalc-v3.1.ipynb
- twolayermodel.ipynb
- UserCalc.py

as well as a "data" directory containing three sample data files:

- sample.csv
- simple_sample.csv
- sample_twolayer_model.csv

This file directory can be used to access and run the pyUserCalc code described in the manuscript using a standard Python distribution and Jupyter browser environment.

Data Set S1. The provided file is a compressed directory that contains Python and Jupyter notebook code files. The directory, once uncompressed, can be used to access and run the model code described in the manuscript using a standard Python distribution (such as Anaconda) and Jupyter browser environment, like JupyterLab or Jupyter Notebook. The *README.md* text file explains what code is in the directory. The directory file *pyUserCalc_manuscript.ipynb* is a Jupyter notebook version of the full manuscript, where the reader can actively run and test the embedded Python code cells. The file *pyUserCalc-v3.1.ipynb* is a Jupyter notebook containing the fully functional code for production work with the model. *twolayermodel.ipynb* is a Jupyter notebook tool for producing two-layer mantle input data files. *UserCalc.py* is a Python file containing the UserCalc driver and model classes, as well as some convenient visualization methods that can be imported using either of the notebook files. The “data” directory contains the sample data files, *sample.csv* and *simple_sample.csv*, which are used for example calculations in the two notebook files and also provide a useful template for other data input files, as well as the output from the two layer calculator, *sample_twolayer_model.csv*.

The supporting directory is also provided in a public Git repository (<https://gitlab.com/ENKI-portal/pyUsercalc>); if preferred, readers can request a free ENKI GitLab account and access and run these code files online using the ENKI JupyterLab browser environment. To do this, the reader should 1) register for a GitLab account (<https://gitlab.com/ENKI-portal>), log into the ENKI cloud server (<https://server.enki-portal.org/hub/login>), close the welcome screen, open a Terminal window and type “git clone <https://gitlab.com/ENKI-portal/pyUsercalc.git>” to clone the repository, and then open the desired notebook files in the pyUserCalc directory.

Alternately, the manuscript notebook is provided for direct access using a binder container (https://mybinder.org/v2/gl/ENKI-portal%2FpyUsercalc/master?filepath=pyUserCalc_manuscript.ipynb).