Supplement

Parameterization of 1D diffusion model results

We use a generalized logistic function to fit model results from the 1D diffusion model presented in (Barth et al., 2019):

$$y = \frac{100}{(1 + Ae^{Bx})^C}$$

Where A, B, C are free parameters and $x = log_{10}\left(a\sqrt{\frac{b\ dP/dt}{a\ K_d\ D}}\right)$.

We find the best-fit values for A, B, C using Matlab's fminsearch or python's scipy.optimize.fmin: A = 0.011; B = 4.05; C = 1.36.

See equivalent matlab and python scripts below:

Python script

```
import numpy as np
from scipy.optimize import fmin
import matplotlib.pyplot as plt

##p5 = results from 1D diffusion model in Barth et al., 2019. Percent reequilibration f
or dP/dt 0.5 MPa/s, a/b = 1, Kd = 0.001, Dol = 1.7e-
10 m2/s, a (size) ranging from 1 - 100 um in logspace.

p5 = np.array([96.1763, 90.4939, 80.2819, 62.2821, 37.3285, 17.2592,  6.8656,  2.5530,
 0.9184,  0.3199]);

##p05 = same as above but for dP/dt = 0.05 MPa/s

p05 = np.array([99.9889, 99.5530, 97.0826, 92.2372, 83.4167, 67.6800, 43.6250, 21.3260,
  8.7188,  3.2817]);

size = np.logspace(-6,-4,10); #MI size in m

Dol = 9.6e-6*np.exp(-
125000/(8.314*(1100+273.15))); #diffusivity of H in Olivine (m2/s) from Barth et al., 2
019


xdata = np.concatenate((np.log10(size*np.sqrt(0.5/0.001/Dol)),np.log10(size*np.sqrt(0.0
5/0.001/Dol))));
ydata = np.concatenate((p5,p05));

x0 = np.random.rand(3)

def fun(x):
    A = x[0];
    B = x[1];
    C = x[2];

    min_ydata = 100./((1+A*np.exp(B*(xdata)))**C);
```

```
29.
30.     sse = np.sum((ydata - min_ydata)**2);
31.
32.     return sse
33.
34. bestx = fmin(fun, x0)
35.
36. x = np.linspace(-.5,2.5,50);
37. y = 100./((1+bestx[0]*np.exp(bestx[1]*(x)))**bestx[2]);
38.
39. plt.plot(xdata,ydata,'rx', label = 'data');  #Plotting model results ('data')
40. plt.plot(x,y,'b-
    ', label = 'best fit logistic function') #Plotting best fit logistic function
41.
42. plt.legend()
43. plt.xlabel('$log_{10}(a \sqrt{\dfrac{b \cdot dP/dt}{a \cdot K_d \cdot D}})$')
44. plt.ylabel('% Reequilibration')
```

Matlab script

```
%p5 = results from 1D diffusion model in Barth et al., 2019. Percent
reequilibration for dP/dt 0.5 MPa/s, a/b = 1, Kd = 0.001, Dol = 1.7e-10 m2/s,
a (size) ranging from 1 — 100 um in logspace.

p5 = [96.1763   90.4939   80.2819   62.2821   37.3285   17.2592    6.8656
2.5530    0.9184    0.3199];

%p05 = same as above but for dP/dt = 0.05 MPa/s

p05 = [99.9889   99.5530   97.0826   92.2372   83.4167   67.6800   43.6250
21.3260    8.7188    3.2817];

size = logspace(-6,-4,10); %MI size in m

Dol = 9.6*10^-6*exp(-125000/(8.314*(1100+273.15))); %diffusivity of H in
Olivine (m2/s) from Barth et al., 2019

xdata = [log10(size*sqrt(0.5/0.001/Dol)),log10(size*sqrt(0.05/0.001/Dol))];
ydata = [p5,p05];

fun = @(x)sseval(x,xdata,ydata);

x0 = rand(3,1);

options = optimset('MaxFunEvals',1e5,'MaxIter',1e5, 'TolFun',1e-10);
bestx = fminsearch(fun,x0,options)

x = linspace(-.5,2.5,50);
y = 100./((1+bestx(1)*exp(bestx(2)*(x))).^bestx(3));

figure, plot(xdata,ydata,'rx'); hold on; %Plotting model results ('data')
plot(x,y,'b-') %Plotting best fit logistic function
```

```
legend('data', 'best fit logistic function')
xlabel('$log_{10}(a \sqrt{\frac{b \cdot dP/dt}{a \cdot K_d \cdot
D}})$','Interpreter','latex','FontSize',20)
ylabel('% Reequilibration')

%-------------------------------------------------------------

function sse = sseval(x,xdata,ydata)
A = x(1);
B = x(2);
C = x(3);

min_ydata = 100./((1+A*exp(B*(xdata))).^C);

sse = sum((ydata - min_ydata).^2);
```

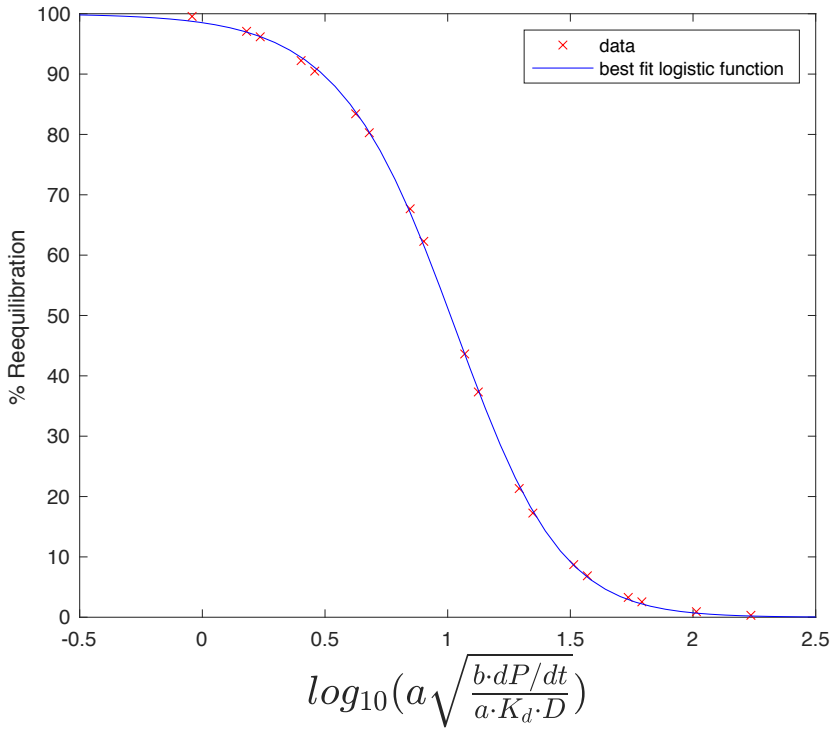

Figure S1. % Reequilibration (water loss) versus x (defined above) for best fit logistic function and data (from 1D diffusion model results).

Python script for Monte Carlo simulations of melt inclusion water loss

Using the parameterization in equations 1 and 2, we can use Monte Carlo simulations to explore propagation of uncertainties from different parameters to output of water loss or decompression rate.

```
import numpy as np
import matplotlib.pyplot as plt
from decimal import Decimal
from scipy.optimize import fsolve
#### Define mean and standard deviation of different parameters here
# Leave either dPdt_mean_sig OR WL_mean_sig empty, depending on which one you are solving for.

kd_mean_sig = [0.001, 0.0002]     #ol-melt H partition coefficient
D0_mean_sig = [9.6e-6, 5e-7]      #preexponential in arrhenius relationship for diffusivity of H in ol (m2/s)
Q_mean_sig = [125000, 1500]       #activation energy for diffusivity of H in ol (J/mol)

T_mean_sig = [1100, 20]           #temperature (oC)
a_mean_sig = [20e-6, 2e-6]        #MI radius (m)
b_mean_sig = [40e-6, 2e-6]        #distance between MI and olivine edge along a (m)

##Choose one of these to solve for (leave empty):
dPdt_mean_sig = [-1.4,0.1]#[]#    #magma decompression rate (MPa/s)
WL_mean_sig = []#[40,10]#         #water loss %
params = np.array([])

for i in range(10000):

    kd = np.random.normal(kd_mean_sig[0],kd_mean_sig[1])
    D0 = np.random.normal(D0_mean_sig[0],D0_mean_sig[1])
    Q = np.random.normal(Q_mean_sig[0],Q_mean_sig[1])
    T = np.random.normal(T_mean_sig[0],T_mean_sig[1])
    D = D0*np.exp(-Q/(8.314*(T+273.15)))
    a = np.random.normal(a_mean_sig[0],a_mean_sig[1])
    b = np.random.normal(b_mean_sig[0],b_mean_sig[1])


    if not WL_mean_sig:
        dPdt = 10**np.log(np.random.lognormal(dPdt_mean_sig[0],dPdt_mean_sig[1]))

        x = np.log10(a*np.sqrt(b*dPdt/(a*kd*D)))

        WL = 100/((1+.011*np.exp(4.05*x))**1.36)

    elif not dPdt_mean_sig:
        WL = np.random.normal(WL_mean_sig[0],WL_mean_sig[1])

        x = np.log(((100/WL)**(1/1.36)-1)/0.011)/4.05

        dPdt = (10**x/a)**2*a/b*kd*D


    param = [kd,D,a,b,dPdt,WL]

```

```
    if np.size(params) < 1:
        params=np.copy(param)
    else:
        params = np.vstack((params, param))

fig, axs = plt.subplots(3,2)
h0 = axs[0,0].hist(params[:,0],bins=50)
axs[0,0].vlines([np.mean(params[:,0]),np.percentile(params[:,0],2.5),np.percentile(params[:,0],97.5)],0,axs[0,0].get_ylim()[1])
axs[0,0].set_xlabel('Kd')
axs[0,0].set_title('Kd = '+str(np.mean(params[:,0]))+'+'+str(np.mean(params[:,0])-np.percentile(params[:,0],2.5))+'-'+str(np.percentile(params[:,0],97.5)-np.mean(params[:,0])))

h1 = axs[1,0].hist(np.log10(params[:,1]),bins=50)
axs[1,0].vlines([np.log10(np.mean(params[:,1])),np.log10(np.percentile(params[:,1],2.5)),np.log10(np.percentile(params[:,1],97.5))],0,axs[1,0].get_ylim()[1])
axs[1,0].set_xlabel('log10(D) (m2/s)')

h2 = axs[2,0].hist(1e6*params[:,2],bins=50)
axs[2,0].vlines([1e6*np.mean(params[:,2]),1e6*np.percentile(params[:,2],2.5),1e6*np.percentile(params[:,2],97.5)],0,axs[2,0].get_ylim()[1])
axs[2,0].set_xlabel('MI size (um)')

h3 = axs[0,1].hist(1e6*params[:,3],bins=50)
axs[0,1].vlines([1e6*np.mean(params[:,3]),1e6*np.percentile(params[:,3],2.5),1e6*np.percentile(params[:,3],97.5)],0,axs[0,1].get_ylim()[1])
axs[0,1].set_xlabel('MI-olivine edge dist along a (um)')

h4 = axs[1,1].hist(np.log10(params[:,4]),bins=50)
axs[1,1].vlines([np.log10(np.mean(params[:,4])),np.log10(np.percentile(params[:,4],2.5)),np.log10(np.percentile(params[:,4],97.5))],0,axs[1,1].get_ylim()[1])
axs[1,1].set_xlabel('log10(dP/dt)(MPa/s)')

h5 = axs[2,1].hist(params[:,5],bins=50)
axs[2,1].vlines([np.mean(params[:,5]),np.percentile(params[:,5],2.5),np.percentile(params[:,5],97.5)],0,axs[2,1].get_ylim()[1])
axs[2,1].set_xlabel('Water loss %')


fig.set_figheight(15)
fig.set_figwidth(15)

```

Insensitivity of water loss to initial pressure deeper than water saturation

Melt inclusion water loss does not occur at depths greater than water saturation, and so diffusion modeling is insensitive to ascent beneath this depth. Consider the case when a MI ascends from 400 MPa versus 240 MPa (figure S2). Because water degassing only begins at ~240 MPa, if the melt is water-saturated, the two MIs will contain roughly the same water concentration. If the decompression rate is the same, MIs will lose the same amount of water since they spend the same amount of time at depths shallower than water saturation. However, the total ascent time will be longer for the melt inclusion that ascended from deeper.

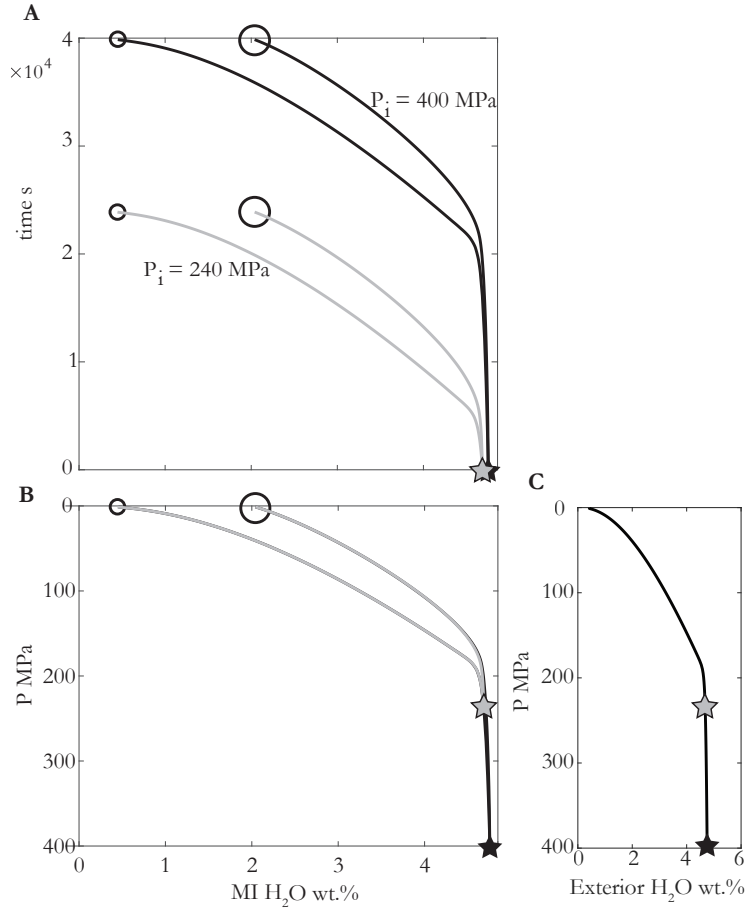

Figure S2. Melt inclusion H_2O concentration versus time (A) and pressure (B). Black curves show model starting at 400 MPa (black star, H_2O = 4.75 wt.%), grey curves start at 240 MPa (grey star, H_2O = 4.68 wt.%). Both models are for the same decompression rate of 0.01 MPa/s, so the model with deeper Pi decompresses for longer duration. Final MI H_2O shown by large circle and small circle (for 10 um and 1 um size respectively). (C) shows H_2O solubility versus pressure (boundary condition) modeled by Solex for open system degassing. Stars show two different initial pressures.

Fixed versus degassing boundary condition

The choice of boundary condition fixed versus degassing boundary condition strongly affects the relationship between MI water loss and MI size (figure S2). If there is no relationship between MI size and water loss, but the MI water concentration is not equal to either the parental magma or the degassed groundmass, it suggests that a stalling event occurred (pale curves in figure S2).

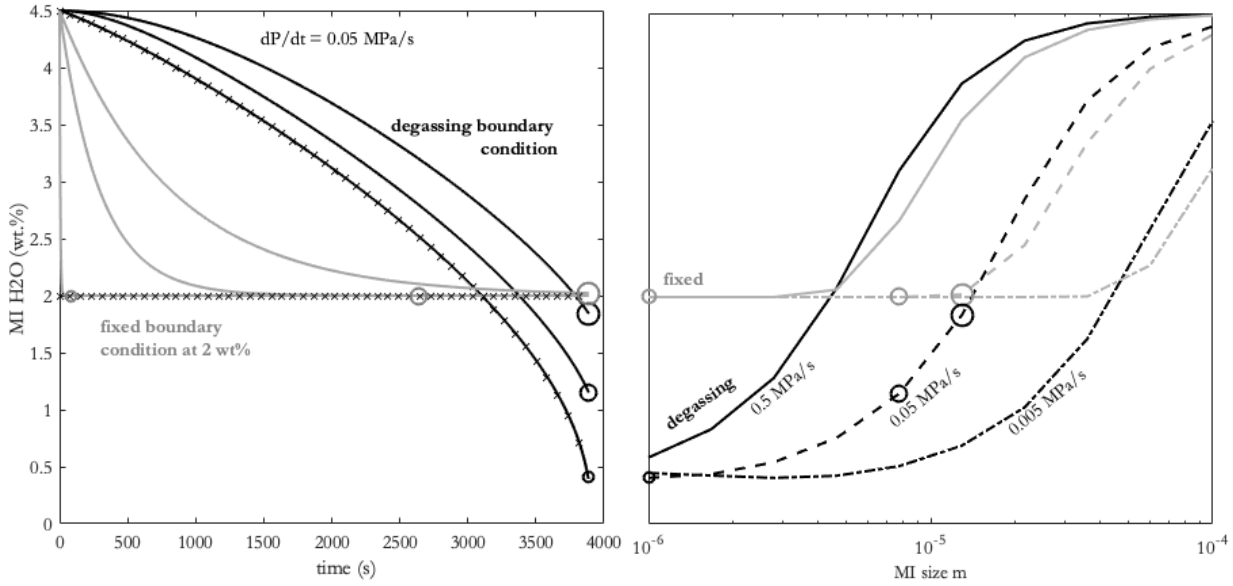

Figure S3. Left: melt inclusion H_2O concentration over time during ascent for different boundary conditions (shown by x marker s) and three different MI sizes (shown on right panel). b/a = 1, D = 1.7e-10 m^2/s, K_d = 0.001. Note that the smallest MI path tracks the boundary condition as it remains in equilibrium with the host magma. A fixed boundary condition at 2 wt.% H_2O (grey) gives the potential for melt inclusions to all reequilibrate to this H_2O concentration, regardless of their size, if stalling time is long enough. By contrast, a degassing boundary condition (black) will never allow melt inclusions of different sizes to reequilibrate to the same H_2O concentration (other than the final, groundmass H_2O concentration). Right: MI H_2O concentration versus size for different decompression rates. Note that decompression rate for Left corresponds to middle curve on right (0.05 MPa/s), with corresponding MI sizes shown by circles.

Insensitivity of model results to different degassing model boundary conditions

In theory, observations of water loss could constrain the type of degassing in the magma (open, closed, excess vapor). However, in practice we find that it makes very little difference to the resulting curves of water loss versus MI size.

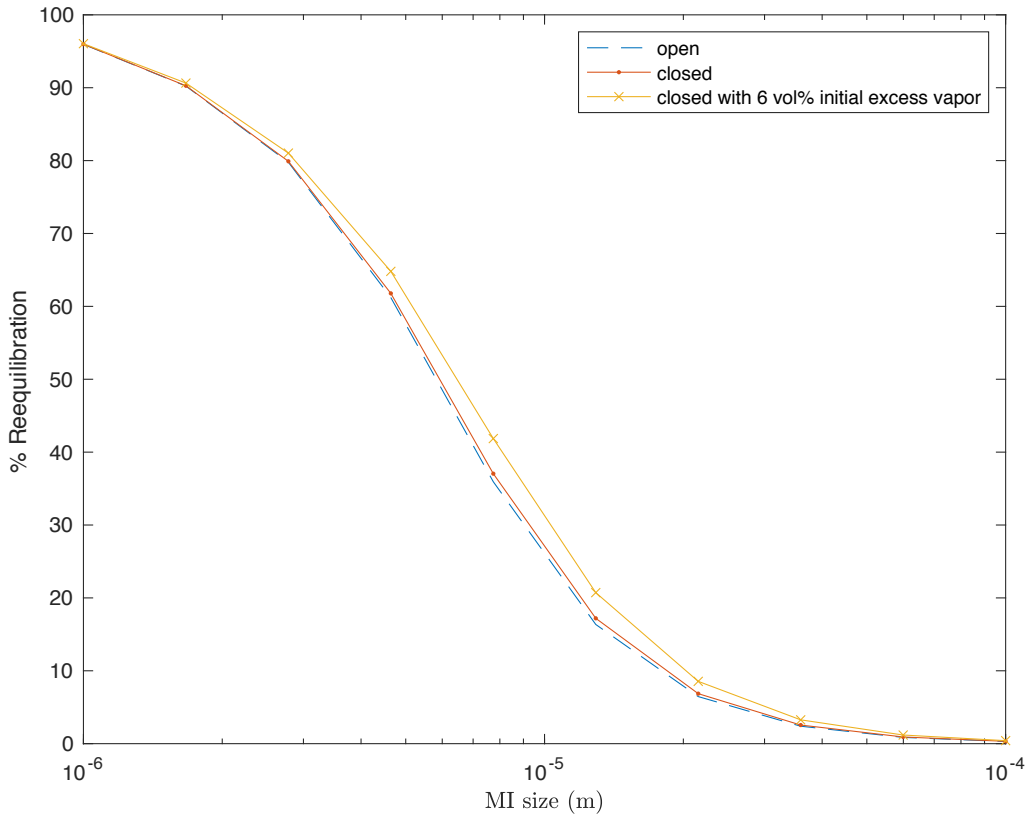

Figure S4. % reequilibration versus MI size for different degassing paths. dP/dt is 0.5 MPa/s, a/b = 1, Kd = 0.001, D = 1.7e-10 m2/s.

Uncertainty in diffusivity

Ferriss et al., 2018 does not report uncertainty in activation energy, E_A, or pre-exponential term, D_0. We assume the same uncertainty in E_A as Mackwell and Kohlstedt (1990), that is, ± 30 kJ/mol. We propagate this to an uncertainty in D_0 by using a chi-squared regression. Uncertainty in D_0 is found where χ^2 is twice that at its minimum.

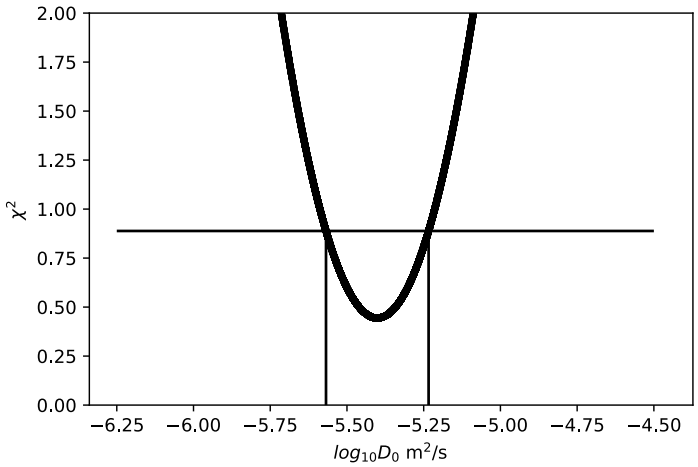

*Figure S5. Chi squared for D_0 for Ferriss et al., 2018 Arrhenius law along **a**.*

Barth et al., 2019 only have two data points to define their Arrhenius relationship and as such, it is not appropriate to perform a chi-squared regression to determine uncertainty. To estimate uncertainty in E_A and D_0 we calculate two Arrhenius relations that go through the minimum and maximum extremes in their reported uncertainties on these two data points, as shown in their figure 6.