mineral_diffusion_timescales

December 18, 2020

1 Mineral diffusion timescales

1.1 1. Determination of diffusion timescales

SEM has been used to determine the concentration of certain elements (Fe, Mg). These elements will diffuse away from crystals according to

$$\frac{\partial C}{\partial t} - D\frac{\partial^2 C}{\partial x^2} = 0. \tag{1}$$

An analytical solution to (1) is

$$C(x,t) = C_2 + \frac{C_1 - C_2}{2}\mathrm{erfc}\left(\frac{x}{2\sqrt{Dt}}\right) \tag{2}$$

As to the diffusivity, we have from wiki

$$D(T) = D_0 \exp\left(-\frac{E_A}{RT}\right), \tag{3}$$

where E_A, R and D_0 are constants.

By repeating this process over many samples from the same eruptive sequence, we may look to see how the results are distributed.

TO DO: how to take account of uncertainty on individual timescale measurements for the population?

UPDATE (2020-05-02) it appears impossible to propagate errors within a Gamma distribution. Maybe the data could be transformed, e.g. by $(x_i - \mathrm{loc})/\sigma_i$, but would these themselves follow a Gamma distribution and how would the parameters obtained from fitting the transformed population relate to those from the fit of the untransformed population?

If all the errors are as small as in the example below and homogeneous, then propagation of timescale errors is not really an issue.

[3]:
```
from functools import partial

from scipy.special import erfc, gammaln, polygamma, gamma
from scipy.stats import expon, kstest, norm, chi2
```

```python
from scipy.stats import gamma as gamma_distn
from scipy.optimize import curve_fit, newton, fmin
from scipy.constants import R  # Ideal-gas constant
from matplotlib.ticker import MultipleLocator

import pandas as pd # Use pandas' inherrent excel support, rather than numpy
import numpy as np
import matplotlib.pyplot as plt

pd.options.display.float_format = '{:,.2f}'.format
```

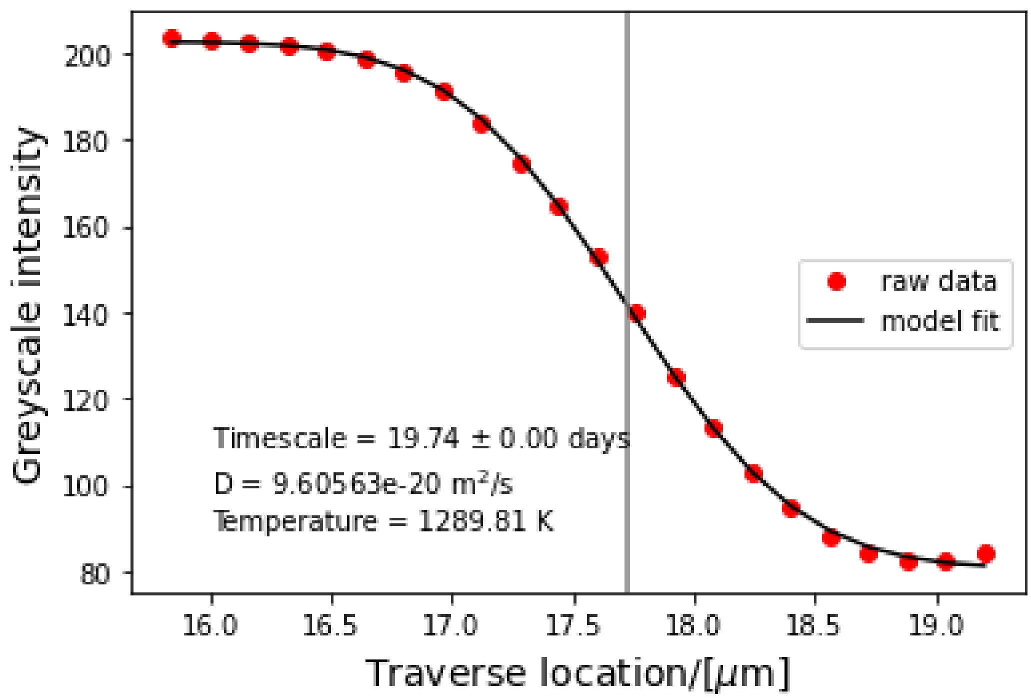
[4]:
```python
E  = -308000    # Activation energy
D0 =  2.8551e-7 # Base diffusion coefficient
T  =  1281.75   # Melt matrix temperature, defined by Dohmen et al., 2016

from diffusion_timescale_modelling.diffusion_timescale_modelling import (
    diffusion_coeff, temp_from_diff_coeff, analytical_soln, fit_wrapper)

D    = diffusion_coeff(T)
Dlow = diffusion_coeff(T - 30)
Dupp = diffusion_coeff(T + 30)

print([Dlow, D, Dupp])
```

```
[4.0109813722139485e-20, 8.018014009263838e-20, 1.5528285872285503e-19]
```

[3]:
```python
data = pd.read_excel('1904A-69-1.xls', sheet_name='raw')

# # Load approximate limits for the transects.  These are
# # approximately the same as in the excel spreadsheet.
# xmin = np.load('xmin.npy')
# xmax = np.load('xmax.npy')

# x = data['distance'][data['distance'] >= xmin][data['distance'] <= xmax]
# y = data['greyscale'][data['distance'] >= xmin][data['distance'] <= xmax]
x = data['distance']
y = data['greyscale']

# p0 = [C1, C2, loc, tau, D]
p0 = [y.max(), y.min(), x.mean(), 2e6, D]

lower_bounds = [-np.inf, -np.inf, -np.inf, 0, Dlow]
upper_bounds = [ np.inf,  np.inf,  np.inf, np.inf, Dupp]
bounds = [lower_bounds, upper_bounds]
```

[4]:
```python
popt, pcov, (Test, timescale, ts_sigma, diffusion) = fit_wrapper(
    x, y, p0, bounds=bounds)
```

```
_ = plt.plot(x * 1e6, y, 'or', label='raw data')

_ = plt.xlabel(r'Traverse location/[$\mu$m]', fontsize=14)
_ = plt.ylabel(r'Greyscale intensity', fontsize=14)

_ = plt.plot(x * 1e6, analytical_soln(x, *popt), '-k', label='model fit')
_ = plt.legend(loc='right')

# print(np.sum((y - analytical_soln(x, *popt))**2))

# The midpoint of the curve
_ = plt.axvline(popt[2] * 1e6, c='gray')

timescale_text = (
    'Timescale = %.2f $\pm$ %.2f days\nD = %g m$^2$/s\nTemperature = %.2f K' %
    (timescale, ts_sigma, diffusion, Test))
_ = plt.text(16., 90., timescale_text)
```



The fit of the curve to the data is visually good and, as shown below, the standard errors on the variables are also good (typically many orders of magnitude smaller than the predicted value). The goodness-of-fit, is determined by a chi-squared statistic which should be close to 1 for a good fit. In this case we find $\chi^2_{\text{est}} = 1.249$.

[6]:
```
popt

np.sqrt(np.diag(pcov))
```

[6]:
```
array([2.02537043e+02, 8.06240455e+01, 1.77224527e-05, 1.70536584e+06,
       9.60563127e-20])
```

[6]:
```
array([4.34643366e-17, 3.35636315e-21, 5.22336238e-09, 2.75797956e-25,
       3.01339060e-21])
```

1.2 2. Timescale modelling via a Gamma distribution

Eruptive processes and magma upwelling are typically Poisson processes, i.e. they occur at random times though the time between events follows a "Poisson distribution" or more correctly, as the time is a continuous variable, a Gamma distribution

$$\text{Gamma}(x;\alpha,\beta) = \frac{\beta^{\alpha} x^{\alpha-1}\exp(-\beta x)}{\Gamma(\alpha)}, \tag{4}$$

where $\Gamma(x)$ is the gamma function. Do then the properties, that is the moments such as mean and variance, of this distribution vary from process to process?

To obtain these moments, maximum likelihood estimators can be evaluated for the set of data $\{x_i\}$ from the likelihood function as

$$\mathcal{L} = \prod_{i=1}^{n} \frac{\beta^{\alpha} x_i^{\alpha-1}\exp(-\beta x_i)}{\Gamma(\alpha)}. \tag{5}$$

However a more useful metric is the *negative log likelihood*, which can be minimised using standard numerical routines:

$$\begin{aligned}\log\mathcal{L}^{-}(x;\alpha,\beta) &= \sum_{i=1}^{n}\left(\beta x_i + \log\Gamma(\alpha) - \alpha\log\beta - (\alpha-1)\log(x_i)\right)\\ &= n\beta\bar{x} + n\log\Gamma(\alpha) - n\alpha\log\beta - (\alpha-1)\sum_{i=1}^{n}\log(x_i)\end{aligned} \tag{6}$$

Thus, the optimal values of α and β can be found by minimising $\log\mathcal{L}^{-}$, i.e. maximising the likelihood function.

1.2.1 Derivatives

First derivatives and jacobian

$$\frac{\partial \log \mathcal{L}^-}{\partial \alpha} = n\Psi^{(0)}(\alpha) - n \log \beta - \sum_i \log(x_i), \tag{7}$$

$$\frac{\partial \log \mathcal{L}^-}{\partial \beta} = n\bar{x} - \frac{n\alpha}{\beta}, \tag{8}$$

where $\Psi^{(n)}(\alpha)$ is the nth derivative of the digamma function (`polygamma(0, alpha)`), $\bar{x} = 1/n \sum_{i=1}^{n} x_i$. Thus we write the jacobian as

$$\mathcal{J}(\theta) = \begin{bmatrix} n\Psi^{(0)}(\alpha) - n \log \beta - \sum_i \log(x_i) \\ n\bar{x} - \dfrac{n\alpha}{\beta} \end{bmatrix}. \tag{9}$$

We note that the maximum likelihood estimators are found when $\mathcal{J}(\theta) = 0$, hence

$$f(\alpha) = n(\log \alpha - \log \bar{x}) + \sum_i \log x_i - n\Psi^{(0)}(\alpha) = 0, \tag{10}$$

$$\text{and } \alpha/\beta = \bar{x}. \tag{11}$$

We solve these two equations to find α and β.

Second derivatives and hessian The parameter variance depends on the matrix of second derivatives (the hessian), so we calculate these.

$$\frac{\partial^2 \log \mathcal{L}^-}{\partial \alpha^2} = n\Psi^{(1)}(\alpha), \tag{12}$$

$$\frac{\partial^2 \log \mathcal{L}^-}{\partial \alpha \partial \beta} = -\frac{n}{\beta}, \tag{13}$$

$$\frac{\partial^2 \log \mathcal{L}^-}{\partial \beta^2} = \frac{n\alpha}{\beta^2}, \tag{14}$$

$$\therefore \mathcal{H}(\theta) = \begin{bmatrix} n\Psi^{(1)}(\alpha) & -\dfrac{n}{\beta} \\ -\dfrac{n}{\beta} & \dfrac{n\alpha}{\beta^2} \end{bmatrix} \tag{15}$$

Parameter varance and Fisher's Information Matrix Parameter variance can be calculated from

$$\mathrm{var}(\theta) = (-E[\mathcal{H}(\theta)])^{-1}. \tag{16}$$

Noting that

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}, \tag{17}$$

$$\therefore \mathrm{var}(\theta) = \frac{1}{n(\hat{\alpha}\Psi^{(1)}(\hat{\alpha}) - 1)} \begin{bmatrix} \hat{\alpha} & \hat{\beta} \\ \hat{\beta} & \hat{\beta}^2 \Psi^{(1)}(\hat{\alpha}) \end{bmatrix} \tag{18}$$

[1]:
```python
from my_stats.mle_gamma import se_estimates
from diffusion_timescale_modelling.diffusion_timescale_modelling import (
    calc_fit_plot, sorted_data_to_df)

eruption_dict = {}
```

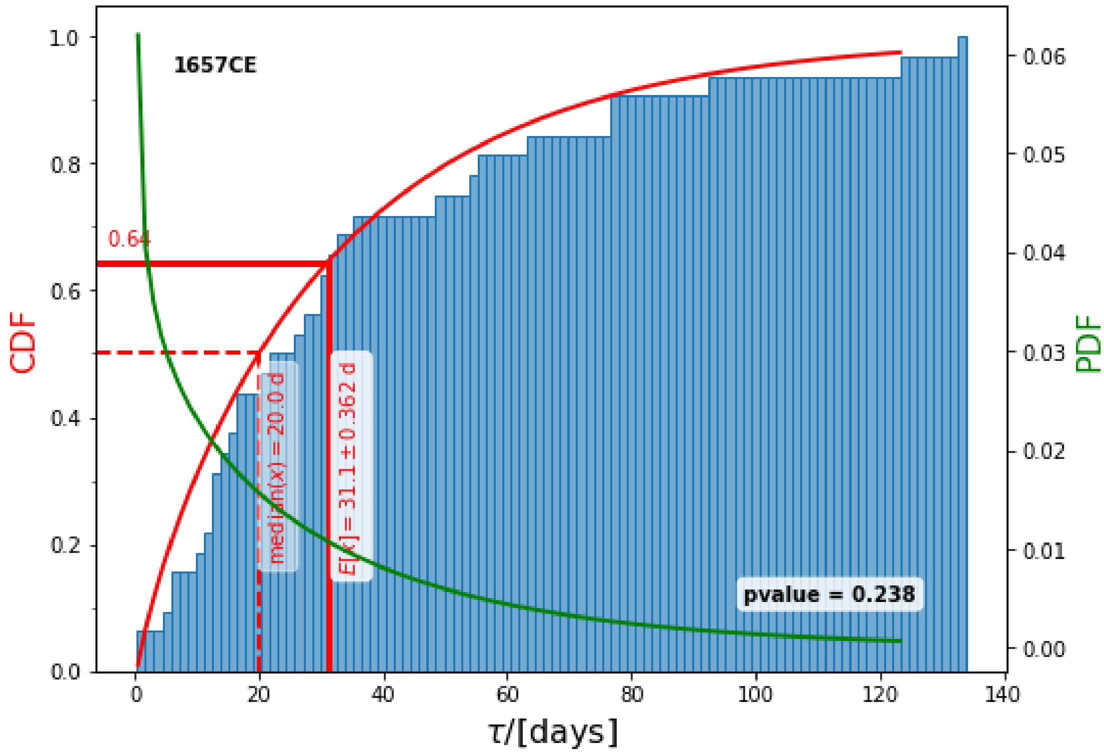
1.3 Calculate and plot the distribution of timescales

The timescales, expressed in days, are contained in a spreadsheet with multiple sheets. Each sheet is for a different eruptive event.

First, for the 1657 AD event, the data are well represented by the distribution as attested to by the p-value: greater than 0.05 is generally good. The expected value, $E[x]$ is 14.6 days, which can be interpreted as the "mean" residence time of the minerals in melting conditions being around two weeks. The median value is similar at 8.3 days, i.e. half of your minerals were in contact with new magma only 8 days before eruption.

I've noticed quite a significant difference between the predicted timescales in the current set of data and the previous spreadsheet which, in all cases have shifted towards much shorter times

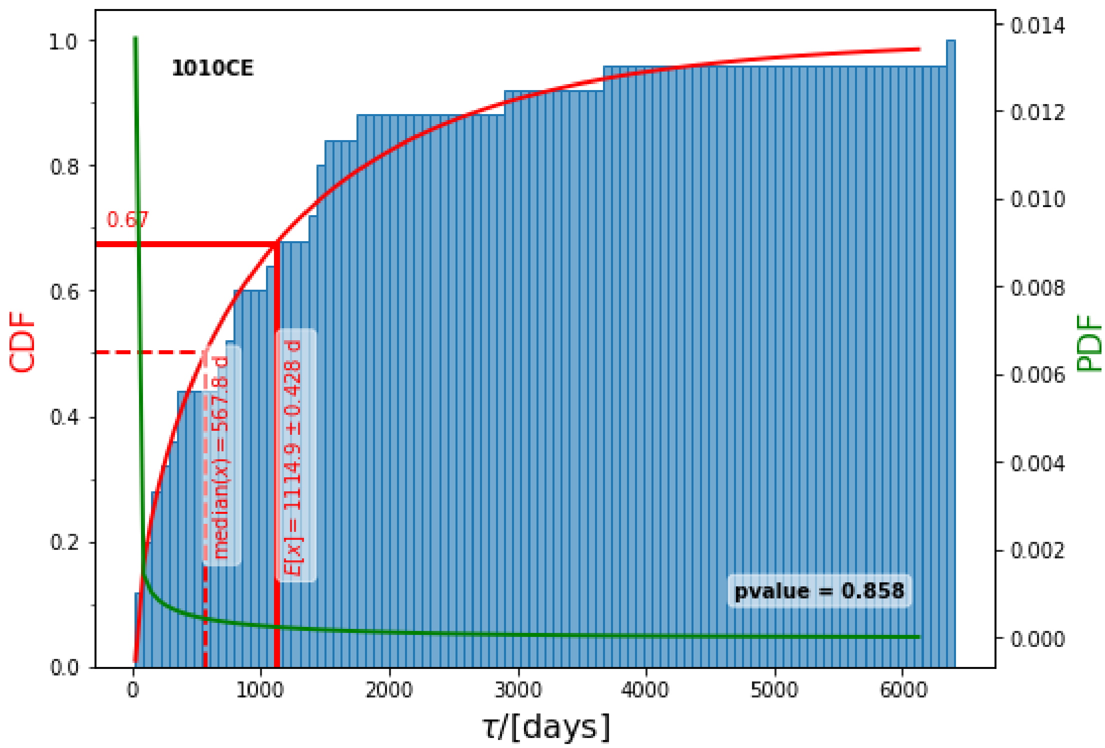
[23]:
```python
# '03-06-timescales.xlsx'
eruption = '1657CE'
popt, data, se, _ = calc_fit_plot(filename='01-07-sorted-timescales.xlsx',
                                  sheet_name=eruption, cdf=True, nbins=100,
                                  ppf_tail=.975)
eruption_dict[eruption] = dict(mean=gamma_distn.mean(*popt),
                               med=gamma_distn.median(*popt),
                               se=se[-1])
```



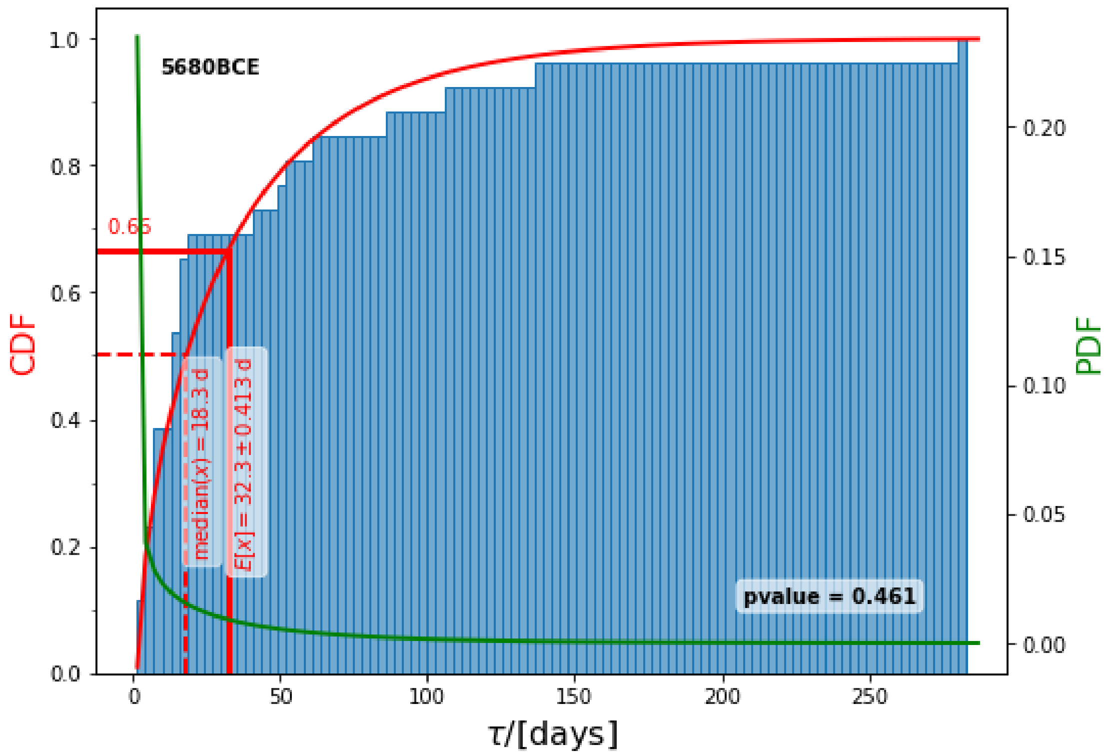
1657CE
pvalue = 0.238
0.64
median(x) = 20.0 d
E[x] = 31.1 ± 0.362 d
CDF
PDF
τ/[days]

The 1010 AD eruption is odd in that there are a few data points at very long times which completely skews the distribution. Here, times beyond 4000 days (10.9 years) have been removed.

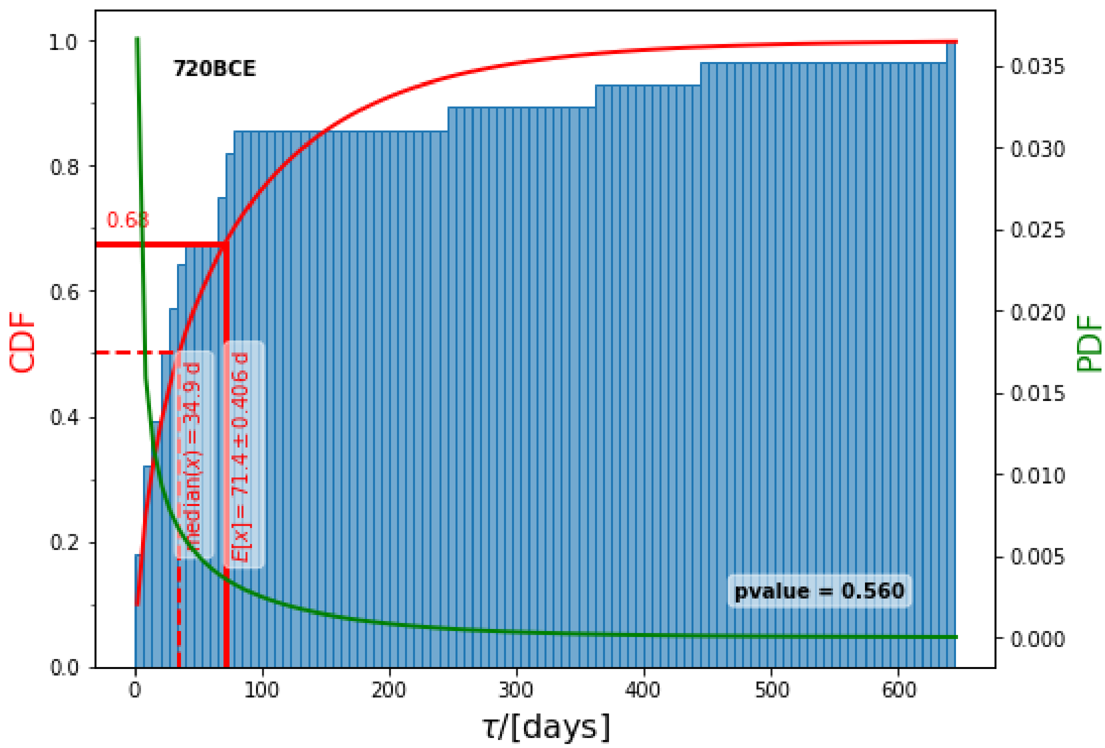
```
[13]: eruption  = '1010CE'
popt, data, se, _ = calc_fit_plot(filename='01-07-sorted-timescales.xlsx',
                                  sheet_name=eruption, cdf=True, nbins=100,
                                  ppf_head=.01, ppf_tail=.985)
eruption_dict[eruption] = dict(mean=gamma_distn.mean(*popt),
                               med= gamma_distn.median(*popt),
                               se=se[-1])
```



```
[17]: eruption = '5680BCE'
popt, data, se, _ = calc_fit_plot(filename='01-07-sorted-timescales.xlsx',
                                  sheet_name=eruption, cdf=True, nbins=100,
                                  ppf_head=0.01, ppf_tail=.999)
eruption_dict[eruption] = dict(mean=gamma_distn.mean(*popt),
                               med= gamma_distn.median(*popt),
                               se=se[-1])
```




```
[21]: eruption  = '720BCE'
ts_cutoff = None   # 1000
popt, data, se, _ = calc_fit_plot(filename='01-07-sorted-timescales.xlsx',
                                  sheet_name=eruption, cdf=True, nbins=100,
                                  ppf_head=0.10, ppf_tail=.998,␣
 ↪ts_cutoff=ts_cutoff)
eruption_dict[eruption] = dict(mean=gamma_distn.mean(*popt),
                               med= gamma_distn.median(*popt),
                               se=se[-1])
```



[24]:
```
# 'se' is a tuple containing the standard errors for the parameter estimates
# from the fitting of the distribution to the data.  The last entry is the
# standard error on the timescale.
se

# Table of timescales for all eruptions
timescales_df = pd.DataFrame.from_dict(eruption_dict)
timescales_df.rename({'mean' : 'Expected timescale/[days]',
                      'med'  : 'Median timescale/[days]',
                      'se'   : 'Timescale std. error/[days]'}, inplace=True)
timescales_df
```

[24]: (0.17959663017847713, 0.007862925946281029, 0.36241589048348166)

[24]:

	1657CE	1010CE	5680BCE	720BCE
Expected timescale/[days]	31.13	1,114.88	32.27	71.37
Median timescale/[days]	20.03	567.78	18.25	34.86
Timescale std. error/[days]	0.36	0.43	0.41	0.41

[94]:
```
df = pd.read_csv('timescale_fitting_df.csv')
sorted_df = sorted_data_to_df(df)
```

[104]:
```python
df_720BCE = sorted_df[sorted_df['eruption'] == '720BCE'].reset_index(drop=True)
print(df_720BCE.to_markdown())
```

```
|    | eruption    | sample    |   Tmeas |    Test |   timescale |   n_good |
ts_std |
|---:|:------------|:----------|--------:|--------:|------------:|---------:|-----
---------:|
|  0 | 720BCE      | 1904A-02 | 1281.75 | 1264.8  |     0.05615 |        2 |
0.00601041 |
|  1 | 720BCE      | 1904A-10 | 1281.75 | 1251.86 |     0.747   |        3 |
0.0621134  |
|  2 | 720BCE      | 1904A-12 | 1286.85 | 1256.93 |     0.789733 |        3 |
0.517999   |
|  3 | 720BCE      | 1904A-08 | 1289.4  | 1259.78 |     0.85245 |        2 |
0.619213   |
|  4 | 720BCE      | 1904A-06 | 1281.75 | 1251.75 |     1.61533 |        3 |
0.420397   |
|  5 | 720BCE      | 1904A-82 | 1281.75 | 1251.75 |     3.0551  |        3 |
1.32338    |
|  6 | 720BCE      | 1904A-79 | 1281.75 | 1273.62 |     8.59597 |        3 |
2.71738    |
|  7 | 720BCE      | 1904A-62 | 1281.75 | 1268.31 |     8.8243  |        3 |
0.293761   |
|  8 | 720BCE      | 1904A-33 | 1281.75 | 1257.59 |     9.7507  |        4 |
1.7323     |
|  9 | 720BCE      | 1904A-71 | 1278.88 | 1266.73 |    11.4816  |        3 |
1.36801    |
| 10 | 720BCE      | 1904A-14 | 1281.75 | 1267.17 |    15.1669  |        3 |
0.82353    |
| 11 | 720BCE      | 1904A-83 | 1281.75 | 1272.72 |    15.9109  |        3 |
16.7554     |
| 12 | 720BCE      | 1904A-74 | 1281.75 | 1268.44 |    20.1461  |        5 |
7.5877     |
| 13 | 720BCE      | 1904A-04 | 1289.4  | 1271.75 |    23.47    |        6 |
5.41467    |
| 14 | 720BCE      | 1904A-69 | 1286.85 | 1274.52 |    24.4142  |        3 |
6.96205    |
| 15 | 720BCE      | 1904A-32 | 1281.75 | 1277.1  |    28.7815  |        2 |
1.0485     |
| 16 | 720BCE      | 1904A-26 | 1281.75 | 1281.6  |    29.7235  |        3 |
8.95008    |
| 17 | 720BCE      | 1904A-61 | 1281.75 | 1276.34 |    34.7055  |        3 |
40.611      |
| 18 | 720BCE      | 1904A-22 | 1281.75 | 1282.35 |    35.1067  |        4 |
14.265      |
| 19 | 720BCE      | 1904A-46 | 1281.75 | 1295.13 |    44.1859  |        3 |
50.6408     |
```

```
| 20 | 720BCE     | 1904A-30 | 1281.75 | 1291.2  |   65.3741   |         6 |
34.3131     |
| 21 | 720BCE     | 1904A-47 | 1281.75 | 1286.18 |   65.7248   |         5 |
57.1405     |
| 22 | 720BCE     | 1904A-77 | 1281.75 | 1301.27 |   74.3681   |         3 |
32.3922     |
| 23 | 720BCE     | 1904A-37 | 1281.75 | 1273.11 |   76.0818   |         3 |
12.1657     |
| 24 | 720BCE     | 1904A-01 | 1281.75 | 1288    |   81.867    |         3 |
38.2782     |
| 25 | 720BCE     | 1904A-48 | 1281.75 | 1289.69 |  247.579    |         3 |
16.6273     |
| 26 | 720BCE     | 1904A-76 | 1281.75 | 1302.76 |  362.371    |         4 |
188.399      |
| 27 | 720BCE     | 1904A-40 | 1281.75 | 1308.53 |  444.898    |         5 |
384.026      |
| 28 | 720BCE     | 1904A-28 | 1281.75 | 1309.22 |  644.106    |         3 |
84.1751     |
| 29 | 720BCE     | 1904A-54 | 1281.75 | 1310.16 | 5339.51     |         6 |
2676.98       |
```

[41]:

```
%%bash
jupyter nbconvert --to pdf mineral_diffusion_timescales.ipynb
```

```
[NbConvertApp] Converting notebook AMetcalf_Soufriere_timescales.ipynb to pdf
[NbConvertApp] Support files will be in AMetcalf_Soufriere_timescales_files/
[NbConvertApp] Making directory ./AMetcalf_Soufriere_timescales_files
[NbConvertApp] Making directory ./AMetcalf_Soufriere_timescales_files
[NbConvertApp] Making directory ./AMetcalf_Soufriere_timescales_files
[NbConvertApp] Making directory ./AMetcalf_Soufriere_timescales_files
[NbConvertApp] Making directory ./AMetcalf_Soufriere_timescales_files
[NbConvertApp] Writing 47263 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 188744 bytes to AMetcalf_Soufriere_timescales.pdf
```

[ ]:

```
```