rampedpyrox Documentation

Release 0.0.2

Jordon D. Hemingway

January 26, 2017

Contents

`rampedpyrox` is a Python package for analyzing experimental kinetic data and accompanying chemical/isotope compositional information. `rampedpyrox` is especially suited for comparing kinetic and isotope results from ramped-temperature instruments such as Ramped PyrOx, RockEval, pyrolysis gc (pyGC), thermogravimitry (TGA), etc. This package converts measured time-series data into rate/activation energy distributions using a selection of reactive continuum models, including the Distributed Activation Energy Model (DAEM) for non-isothermal data. Additionally, this package calculates the range of rate/activation energy values associated with each isotope "fraction" and performs necessary isotope corrections (blank, mass balance, kinetic fractionation).

CHAPTER 1

Package Information

Authors Jordon D. Hemingway (jordon_hemingway@fas.harvard.edu)

Version 0.1.2

Release 26 January 2017

License GNU GPL v3 (or greater)

url http://github.com/FluvialSeds/rampedpyrox

CHAPTER 2

Bug Reports

This software is still in active deveopment. Please report any bugs directly to me.

CHAPTER 3

How to Cite

When analyzing data with `rampedpyrox` to be used in a peer-reviewed journal, please cite this package as:

- J.D. Hemingway. *rampedpyrox*: open-source tools for thermoanalytical data analysis, 2016-, http://github.com/FluvialSeds/rampedpyrox [online; accessed 2017-01-26]

Additionally, please cite the following peer-reviewed manuscript describing the deveopment of the package and Ramped PyrOx data treatment:

- J.D. Hemingway et al. **(in prep)** An inverse model for relating organic carbon thermal reactivity and isotope composition using Ramped PyrOx.

If using Ramped PyrOx data generated by the NOSAMS instrument, the following manuscript contains relevant information regarding blank carbon composition, isotope mass balance, and the magnitude of the kinetic isotope effect:

- J.D. Hemingway et al. (2017) Assessing the blank carbon contribution, isotope mass balance, and kinetic isotope fractionation of the ramped pyrolysis/oxidation instrument at NOSAMS. *Radiocarbon*, **in press**.

CHAPTER 4

Package features

`rampedpyrox` currently contains the following features relevant to non-isothermal kinetic analysis:

- Stores and plots thermogram data
- Performs first-order DAEM inverse model to estimate activation energy distributions, $p_0(E)$
  - Regularizes ("smoothes") $p_0(E)$ using Tikhonov Regularization
    - Automated or user-defined regularization value
- Calculates subset of $p_0(E)$ contained in each RPO collection fraction
  - Automatically blank-corrects inputted isotope values using any known blank carbon composition
  - Corrects measured $^{13}C/^{12}C$ ratios for the kinetic isotope effect (KIE) during heating
- Calculates and stores model performance metrics and goodness of fit statistics
- Generates plots of thermograms, $p_0(E)$, and E vs. isotope values for each RPO fraction
- Allows for forward-modeling of any arbitrary time-temperature history, *e.g.* to determine the decomposition rates and isotope fractionation during geologic organic carbon matruation

4.1 Future Additions

Future versions of `rampedpyrox` will aim to include:

- Better support for isothermal experimental conditions
- Non-first-order kinetic models

CHAPTER 5

License

This product is licensed under the GNU GPL license, version 3 or greater.

CHAPTER 6

Table of contents

6.1 Comprehensive Walkthrough

The following examples should form a comprehensive walkthough of downloading the package, getting thermogram data into the right form for importing, running the DAEM inverse model to generate an activation energy (E) probability density function [p_0(E)], determining the E range contained in each RPO fraction, correcting isotope values for blank and kinetic fractionation, and generating all necessary plots and tables for data analysis.

For detailed information on class attributes, methods, and parameters, consult the *Package Reference Documentation* or use the `help()` command from within Python.

6.1.1 Quick guide

Basic runthrough:

```
#import modules
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import rampedpyrox as rp

#generate string to data
tg_data = '/folder_containing_data/tg_data.csv'
iso_data = '/folder_containing_data/iso_data.csv'

#make the thermogram instance
tg = rp.RpoThermogram.from_csv(
        tg_data,
        bl_subtract = True,
        nt = 250)

#generate the DAEM
daem = rp.Daem.from_timedata(
        tg,
        log10k0 = 10, #assume a constant value of 10
        E_max = 350,
        E_min = 50,
        nE = 400)

#run the inverse model to generate an energy complex
ec = rp.EnergyComplex.inverse_model(
```

```
        daem,
        tg,
        omega = 'auto') #calculates best-fit omega

#forward-model back onto the thermogram
tg.forward_model(daem, ec)

#calculate isotope results
ri = rp.RpoIsotopes.from_csv(
        iso_data,
        daem,
        ec,
        blk_corr = True, #uses values for NOSAMS instrument
        bulk_d13C_true = [-24.9, 0.1], #true d13C value
        mass_err = 0.01,
        DE = 0.0018) #value from Hemingway et al., 2017

#compare corrected isotopes and E values
print(ri.ri_corr_info)
```

6.1.2 Downloading the package

Using the `pip` package manager

`rampedpyrox` and the associated dependencies can be downloaded directly from the command line using `pip`:

```
$ pip install rampedpyrox
```

You can check that your installed version is up to date with the latest release by doing:

```
$ pip freeze
```

Downloading from source

Alternatively, `rampedpyrox` source code can be downloaded directly from my github repo. Or, if you have git installed:

```
$ git clone git://github.com/FluvialSeds/rampedpyrox.git
```

And keep up-to-date with the latest version by doing:

```
$ git pull
```

from within the rampedpyrox directory.

Dependencies

The following packages are required to run `rampedpyrox`:

- python >= 2.7, including Python 3.x
- matplotlib >= 1.5.2
- numpy >= 1.11.1
- pandas >= 0.18.1

- scipy >= 0.18.0

If downloading using `pip`, these dependencies (except python) are installed automatically.

Optional Dependencies

The following packages are not required but are highly recommended:

- ipython >= 4.1.1

Additionally, if you are new to the Python environment or programming using the command line, consider using a Python integrated development environment (IDE) such as:

- wingware
- Enthought Canopy
- Anaconda
- Spyder

Python IDEs provide a "MATLAB-like" environment as well as package management. This option should look familiar for users coming from a MATLAB or RStudio background.

6.1.3 Getting data in the right format

Importing thermogram data

For thermogram data, this package requires that the file is in *.csv* format, that the first column is *date_time* index in an **hh:mm:ss AM/PM** format, and that the file contains 'CO2_scaled' and 'temp' columns [1]. For example:

date_time	temp	CO2_scaled
10:24:20 AM	100.05025	4.6
10:24:21 AM	100.09912	5.3
10:24:22 AM	100.11413	5.1
10:24:23 AM	100.22759	4.9

Once the file is in this format, generate a string pointing to it in python like this:

```
#create string of path to data
tg_data = '/path_to_folder_containing_data/tg_data.csv'
```

Importing isotope data

If you are importing isotope data, this package requires that the file is in *.csv* format and that the first two rows correspond to the starting time of the experiment and the initial trapping time of fraction 1, respectively. Additionally, the file must contain a 'fraction' column and isotope/mass columns must have *ug_frac*, *d13C*, *d13C_std*, *Fm*, and *Fm_std* headers. For example:

date_time	fraction	ug_frac	d13C	d13C_std	Fm	Fm_std
10:24:20 AM	-1	0	0	0	0	0
10:45:10 AM	0	0	0	0	0	0
11:32:55 AM	1	69.05	-30.5	0.1	0.8874	0.0034
11:58:23 AM	2	105.81	-29.0	0.1	0.7945	0.0022

[1] Note: If analyzing samples run at NOSAMS, all other columns in the *tg_data* file generated by LabView are not used and can be deleted or given an arbitrary name.

Here, the *ug_frac* column is composed of manometrically determined masses rather than those determined by the infrared gas analyzer (IRGA, *i.e.* photometric). **Important:** The *date_time* value for fraction '-1' must be the same as the *date_time* value for the first row in the *tg_data* thermogram file **and** the value for fraction '0' must the initial time when trapping for fraction 1 began.

Once the file is in this format, generate a string pointing to it in python like this:

```
#create string of path to data
iso_data = '/path_to_folder_containing_data/iso_data.csv'
```

6.1.4 Making a TimeData instance (the Thermogram)

Once the *tg_data* string been defined, you are ready to import the package and generate an `rp.RpoThermogram` instance containing the thermogram data. `rp.RpoThermogram` is a subclass of `rp.TimeData` – broadly speaking, this handles any object that contains measured time-series data. It is important to keep in mind that your thermogram will be down-sampled to *nt* points in order to smooth out high-frequency noise and to keep Laplace transform matrices to a manageable size for inversion (see *Setting-up the model* below). Additionally, because the inversion model is sensitive to boundary conditions at the beginning and end of the run, there is an option when generating the thermogram instance to ensure that the baseline has been subtracted. Note that temperature and ppm CO2 uncertainty is not inputted – any noise is dealt with during regularization (see *Regularizing the inversion* below):

```
#load modules
import rampedpyrox as rp

#number of timepoints to be used in down-sampled thermogram
nt = 250

tg = rp.RpoThermogram.from_csv(
        data,
        bl_subtract = True, #subtract baseline
        nt = nt)
```

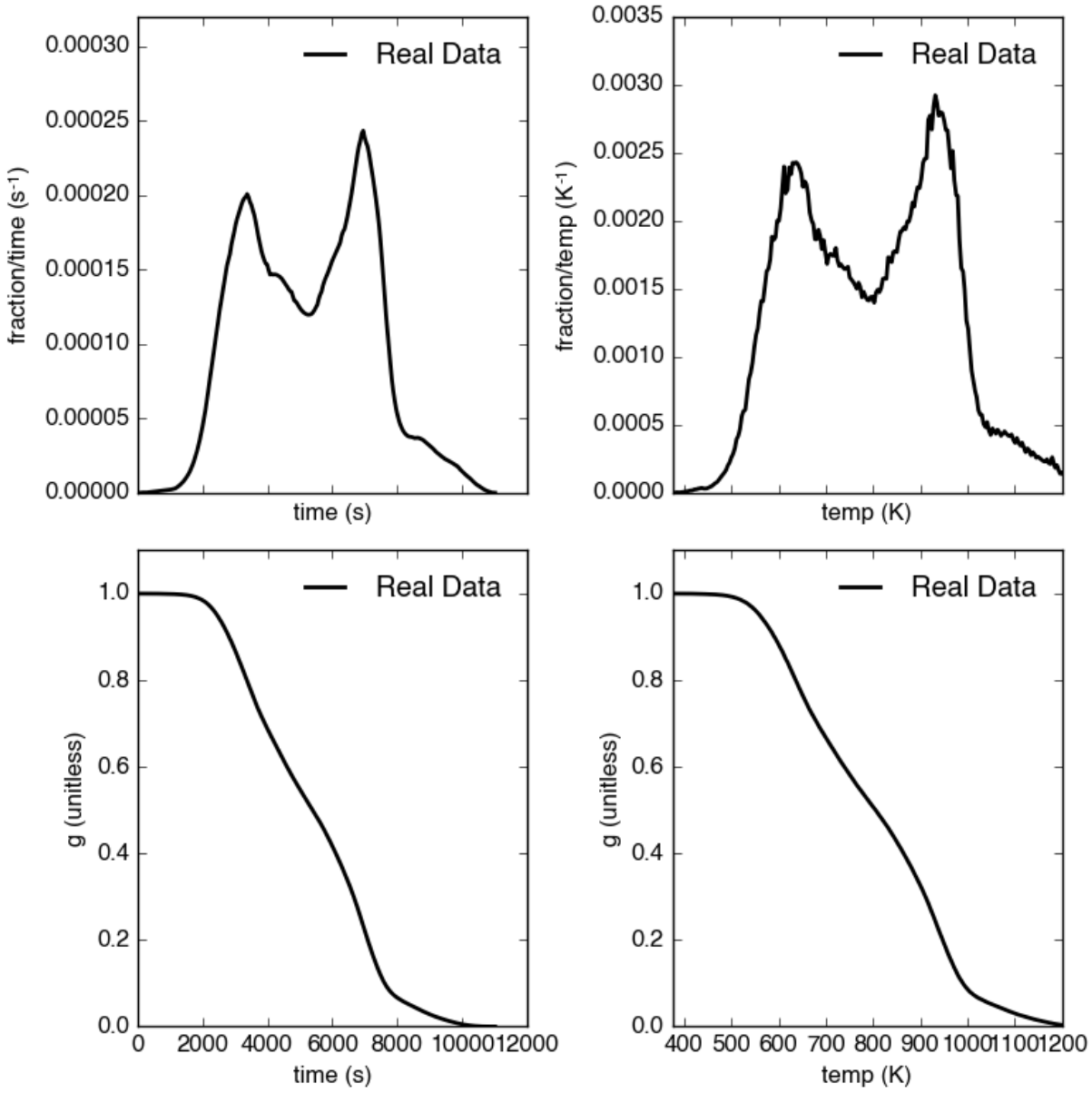
Plot the thermogram and the fraction of carbon remaining against temperature [2] or time:

```
#load modules
import matplotlib.pyplot as plt

#make a figure
fig, ax = plt.subplots(2, 2,
        figsize = (8,8),
        sharex = 'col')

#plot results
ax[0, 0] = tg.plot(
        ax = ax[0, 0],
        xaxis = 'time',
        yaxis = 'rate')

ax[0, 1] = tg.plot(
        ax = ax[0, 1],
        xaxis = 'temp',
        yaxis = 'rate')

ax[1, 0] = tg.plot(
        ax = ax[1, 0],
```

[2] Note: For the NOSAMS Ramped PyrOx instrument, plotting against temperature results in a noisy thermogram due to the variability in the ramp rate, dT/dt.

```
        xaxis = 'time',
        yaxis = 'fraction')

ax[1, 1] = tg.plot(
        ax = ax[1, 1],
        xaxis = 'temp',
        yaxis = 'fraction')

#adjust the axes
ax[0, 0].set_ylim([0, 0.00032])
ax[0, 1].set_ylim([0, 0.0035])
ax[1, 1].set_xlim([375, 1200])

plt.tight_layout()
```

Resulting plots look like this:



Additionally, thermogram summary info are stored in the *tg_info* attribute, which can be printed or saved to a .csv file:

```
#print in the terminal
print(tg.tg_info)

#save to csv
tg.tg_info.to_csv('file_name.csv')
```

This will create a table similar to:

t_max (s)	6.95e+03
t_mean (s)	5.33e+03
t_std (s)	1.93e+03
T_max (K)	9.36e+02
T_mean (K)	8.00e+02
T_std (K)	1.61e+02
max_rate (frac/s)	2.43e-04
max_rate (frac/K)	2.87e-04

6.1.5 Setting-up the model

The inversion transform

Once the `rp.RpoThermogram` instance has been created, you are ready to run the inversion model and generate a regularized and discretized probability density function (pdf) of the rate/activation energy distribution, p. For non-isothermal thermogram data, this is done using a first-order Distributed Activation Energy Model (DAEM) [3] by generating an `rp.Daem` instance containing the proper transform matrix, A, to translate between time and activation energy space [4]. This matrix contains all the assumptions that go into building the DAEM inverse model as well as all of the information pertaining to experimental conditions (*e.g.* ramp rate) [5]. Importantly, the transform matrix does not contain any information about the sample itself – it is simply the model "design" – and a single `rp.Daem` instance can be used for multiple samples provided they were analyzed under identical experimental conditions (however, this is not recommended, as subtle differences in experimental conditions such as ramp rate could exist).

One critical user input for the DAEM is the Arrhenius pre-exponential factor, *k:sub:'0'* (inputted here in $\log_{10}$ form). Because there is much discussion in the literature over the constancy and best choice of this parameter (the so-called 'kinetic compensation effect' or KCE [6]), this package allows *log:sub:'10'k:sub:'0'* to be inputted as a constant, an array, or a function of E.

For convenience, you can create any model directly from either time data or rate data, rather than manually inputting time, temperature, and rate vectors. Here, I create a DAEM using the thermogram defined above and allow E to range from 50 to 400 kJ/mol:

```
#define log10k0, assume constant value of 10
log10k0 = 10 #value advocated in Hemingway et al. (in prep)

#define E range (in kJ/mol)
E_min = 50
E_max = 400
nE = 400 #number of points in the vector

#create the DAEM instance
daem = rp.Daem.from_timedata(
        tg,
        log10k0 = log10k0,
        E_max = E_max,
        E_min = E_min,
        nE = nE)
```

[3] Braun and Burnham (1999), *Energy & Fuels*, **13(1)**, 1-22 provides a comprehensive review of the kinetic theory, mathematical derivation, and forward-model implementation of the DAEM.

[4] See Forney and Rothman (2012), *Biogeosciences*, **9**, 3601-3612 for information on building and regularizing a Laplace transform matrix to be used to solve the inverse model using the L-curve method.

[5] See Hemingway et al. **(in prep)** for a step-by-step mathematical derivation of the DAEM and the inverse solution applied here.

[6] See White et al. (2011), *J. Anal. Appl. Pyrolysis*, **91**, 1-33 for a review on the KCE and choice of *log:sub:'10'k:sub:'0'*.

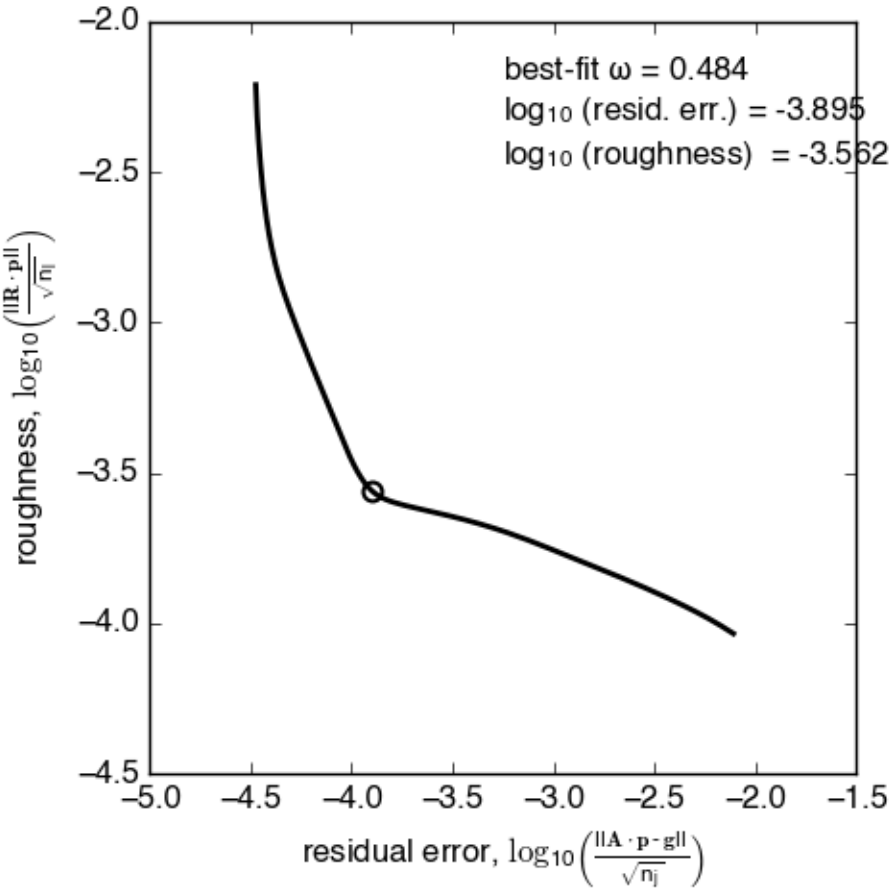
Regularizing the inversion

Once the model has been created, you must tell the package how much to 'smooth' the resulting $p_0(E)$ distribution. This is done by choosing an *omega* value to be used as a smoothness weighting factor for Tikhonov regularization [7]. Higher values of *omega* increase how much emphasis is placed on minimizing changes in the first derivative at the expense of a better fit to the measured data, which includes analytical uncertainty. Rractically speaking, regularization aims to "fit the data while ignoring the noise." This package can calculate a best-fit *omega* value using the L-curve method [5].

Here, I calculate and plot L curve for the thermogram and model defined above:

```
#make a figure
fig,ax = plt.subplots(1, 1,
        figsize = (5, 5))

om_best, ax = daem.calc_L_curve(
        tg,
        ax = ax,
        plot = True)

plt.tight_layout()
```

Resulting L-curve plot looks like this, here with a calculated best-fit omega value of 0.484:



[7] See Hansen (1994), *Numerical Algorithms*, **6**, 1-35 for a discussion on Tikhonov regularization.

6.1.6 Making a RateData instance (the inversion results)

After creating the `rp.Daem` instance and deciding on a value for *omega*, you are ready to invert the thermogram and generate an Activation Energy Complex (EC). An EC is a subclass of the more general `rp.RateData` instance which, broadly speaking, contains all rate and/or activation energy information. That is, the EC contains an estimate of the underlying E distribution, $p_0(E)$, that is intrinsic to a particular sample for a particular degradation experiment type (*e.g.* combustion, *uv* oxidation, enzymatic degradation, etc.). A fundamental facet of this model is the realization that degradation of any given sample can be described by a distribution of reactivities as described by activation energy.

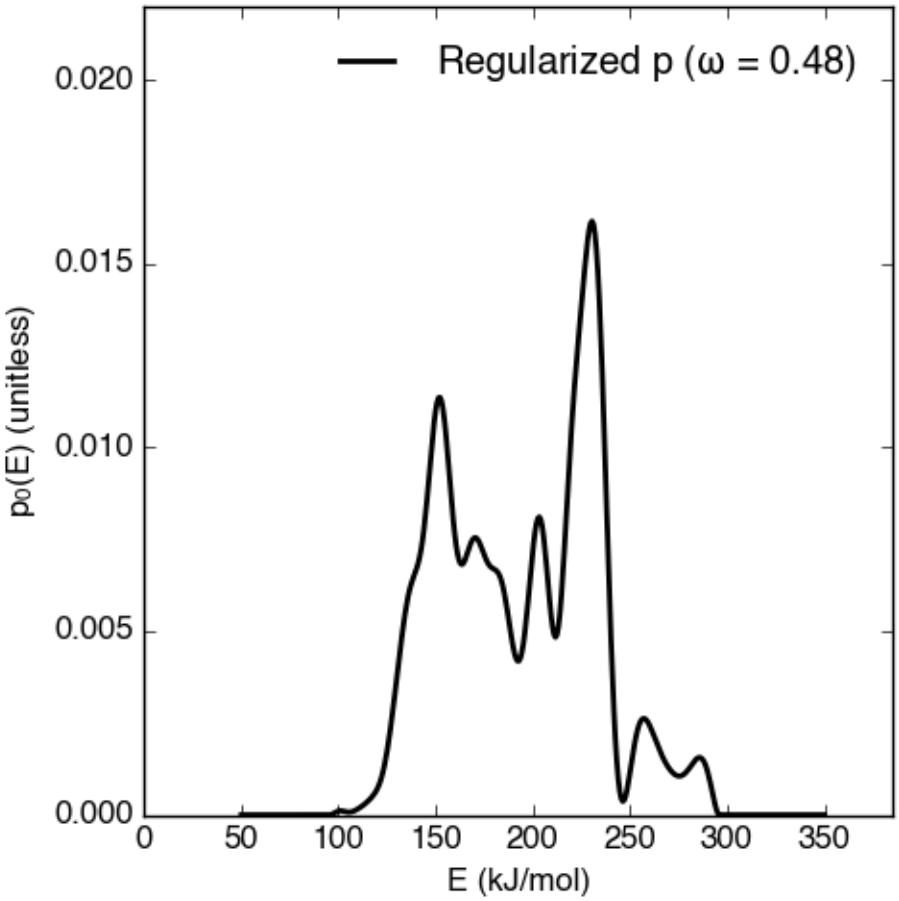
Here I create an energy complex with *omega* set to 'auto':

```
ec = rp.EnergyComplex.inverse_model(
        daem,
        tg,
        omega = 'auto')
```

I then plot the resulting deconvolved energy complex:

```
#make a figure
fig,ax = plt.subplots(1, 1,
        figsize = (5,5))

#plot results
ax = ec.plot(ax = ax)

ax.set_ylim([0, 0.022])
plt.tight_layout()
```

Resulting p:sub: *0*(E) looks like this:



EnergyComplex summary info are stored in the *ec_info* attribute, which can be printed or saved to a .csv file:

```
#print in the terminal
print(ec.ec_info)

#save to csv
ec.ec_info.to_csv('file_name.csv')
```

This will create a table similar to:

E_max (kJ/mol)	230.45
E_mean (kJ/mol)	194.40
E_std (kJ/mol)	39.58
p0(E)_max	0.02

Additionally, goodness of fit residual RMSE and roughness values can be viewed:

```
#residual rmse for the model fit
ec.resid

#regularization roughness norm
ec.rgh
```

Forward modeling the estimated thermogram

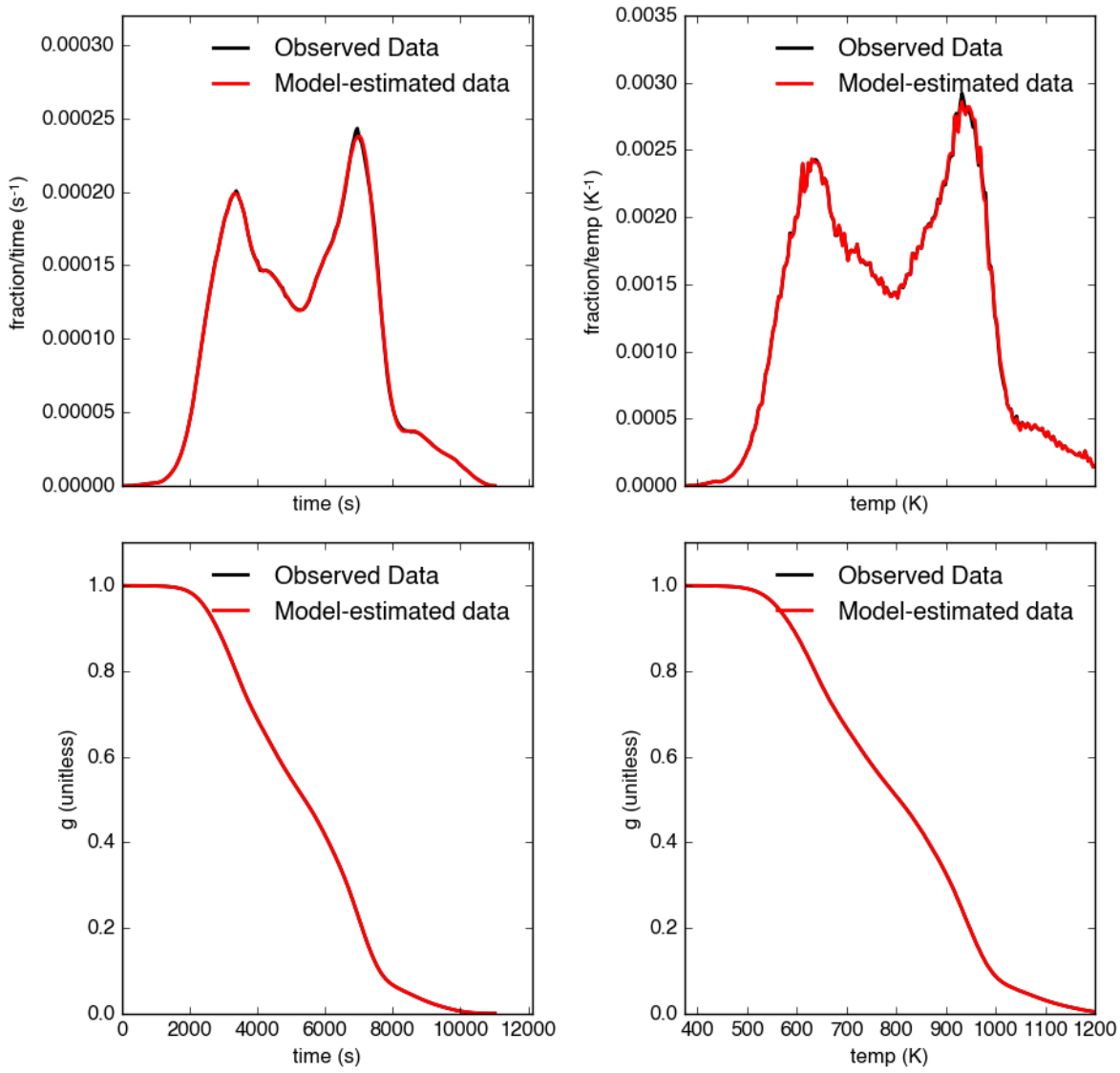
Once the `rp.EnergyComplex` instance has been created, you can forward-model the predicted thermogram and compare with measured data using the `forward_model` method of any `rp.TimeData` instance. For example:

```
tg.forward_model(daem, ec)
```

The thermogram is now updated with modeled data and can be plotted:

```
#make a figure
fig, ax = plt.subplots(2, 2,
        figsize = (8,8),
        sharex = 'col')

#plot results
ax[0, 0] = tg.plot(
        ax = ax[0, 0],
        xaxis = 'time',
        yaxis = 'rate')

ax[0, 1] = tg.plot(
        ax = ax[0, 1],
        xaxis = 'temp',
        yaxis = 'rate')

ax[1, 0] = tg.plot(
        ax = ax[1, 0],
        xaxis = 'time',
        yaxis = 'fraction')

ax[1, 1] = tg.plot(
        ax = ax[1, 1],
        xaxis = 'temp',
        yaxis = 'fraction')

#adjust the axes
ax[0, 0].set_ylim([0, 0.00032])
ax[0, 1].set_ylim([0, 0.0035])
ax[1, 1].set_xlim([375, 1200])

plt.tight_layout()
```

Resulting plot looks like this:



Predicting thermograms for other time-temperature histories

One feature of the `rampedpyrox` package is the ability to forward-model degradation rates for any arbitrary time-temperature history once the estimated p:sub: *0*(E) distribution has been determined. This allows users the ability to:

- Quickly analyze a small amount of sample with a fast ramp rate in order to estimate p:sub: *0*(E), then forward-model the thermogram for a typical ramp rate of 5K/min in order to determine the best times to toggle gas collection fractions.
  - This feature could allow for future development of an automated Ramped PyrOx system.
- Manipulate oven ramp rates and temperature programs in an similar way to a gas chromatograph (GC) in order to separate co-eluting components, mimic real-world environmental heating rates, etc.
- Predict petroleum maturation and evolved gas isotope composition over geologic timescales [8].

[8] See Dieckmann (2005) *Marine and Petroleum Geology*, **22**, 375-390 and Dieckmann et al. (2006) *Marine and Petroleum Gelogoy*, **23**, 183-199

Here, I will use the above-created p:sub: *0*(E) energy complex to generate a new DAEM with a ramp rate of 15K/min up to 950K, then hold at 950K:

```
#import modules
import numpy as np

#extract the Ee array from the energy complex
E = ec.E

#make an array of 350 points going from 0 to 5000 seconds
t = np.linspace(0, 5000, 350)

#calculate the temperature at each timepoint, starting at 373K
T = 373 + (15./60)*t

ind = np.where(T > 950)
T[ind] = 950

#use the same log10k0 value as before
log10k0 = 10

#make the new model
daem_fast = rp.Daem(
        E,
        log10k0,
        t,
        T)

#make a new thermogram instance by inputting the time
# and temperature arrays. This "sets up" the thermogram
# for forward modeling
tg_fast = rp.RpoThermogram(t, T)

#forward-model the energy complex onto the new thermogram
tg_fast.forward_model(daem_fast, ec)
```

Note: Because a portion of this time-temperature history is isothermal, this calculation will inevitably divide by *0* while calculating some metrics. As a result, it will generate some warnings and will fail to calculate an average decay temperature. Results plotted against time are still valid and robust.

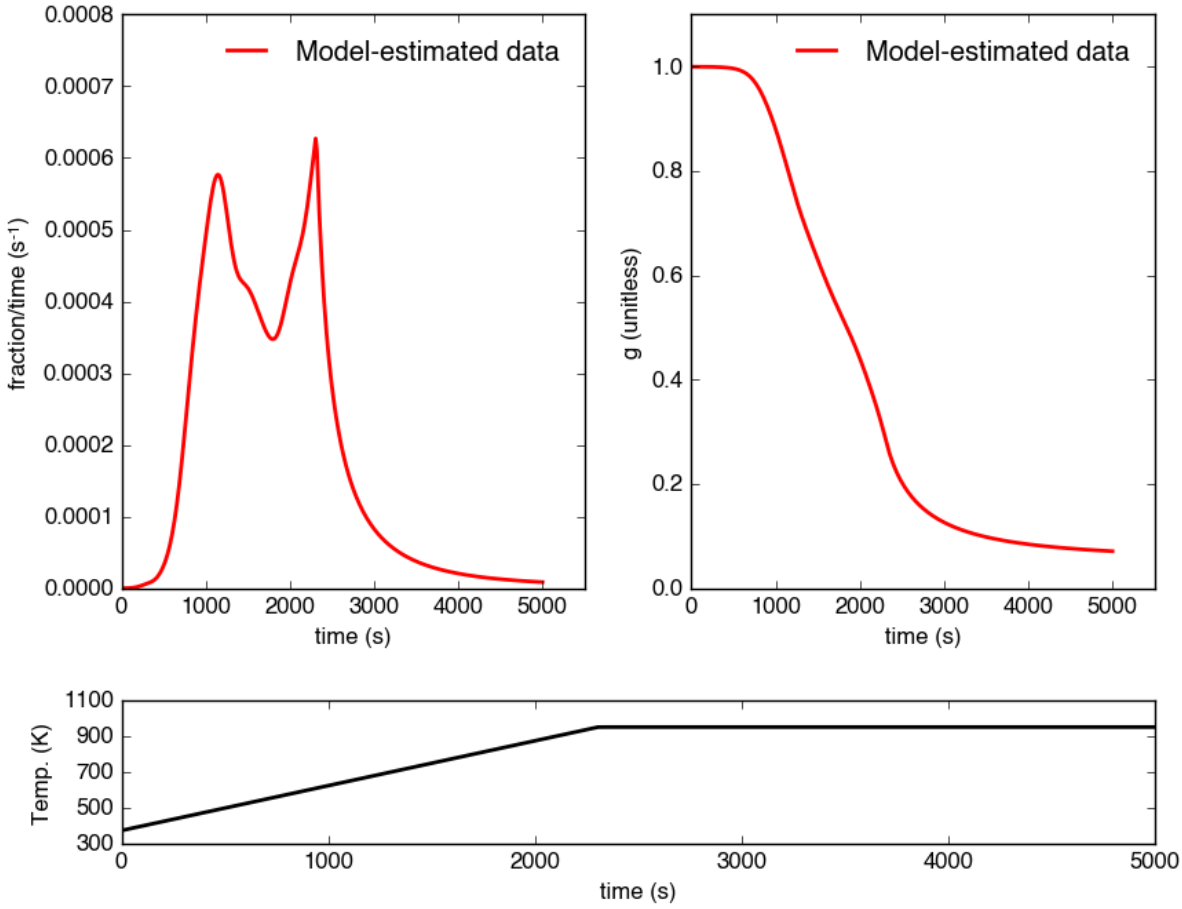
The *tg_fast* thermogram now contains modeled data and can be plotted:

```
#import additional modules
import matplotlib.gridspec as gridspec

#make a figure
gs = gridspec.GridSpec(2, 2, height_ratios=[4,1])

ax1 = plt.subplot(gs[0,0])

ax2 = plt.subplot(gs[0,1])

ax3 = plt.subplot(gs[1,:])


#plot results
ax1 = tg_fast.plot(
        ax = ax1,
```

for a discussion on the limitations of predicting organic carbon maturation over geologic timescales using laboratory experiments.

```
        xaxis = 'time',
        yaxis = 'rate')

ax2 = tg_fast.plot(
        ax = ax2,
        xaxis = 'time',
        yaxis = 'fraction')

#plot time-temperature history
ax3.plot(
        tg_fast.t,
        tg_fast.T,
        linewidth = 2,
        color = 'k')

#set labels
ax3.set_xlabel('time (s)')
ax3.set_ylabel('Temp. (K)')

#adjust the axes
ax1.set_ylim([0, 0.0008])
ax3.set_yticks([300, 500, 700, 900, 1100])

plt.tight_layout()
```

Which generates a plot like this:



6.1.7 Importing and correcting isotope values

At this point, the thermogram, DAEM model, and p:sub: *0*(E) distribution have all been created. Now, the next step is to import the RPO isotope values and to calculate the distribution of E values corresponding to each RPO fraction. This is This is done by creating an `rp.RpoIsotopes` instance using the `from_csv` method. If the sample was run on the NOSAMS Ramped PyrOx instrument, setting `blank_corr = True` and an appropriate value for `mass_rerr` will automatically blank-correct values according to the blank carbon estimation of Hemingway et al. (2017) [9] [10]. Additionally, if ^{13}C isotope composition was measured, these can be further corrected for any mass-balance discrepancies and for kinetic isotope fractionation within the RPO instrument [5] [9].

Here I create an `rp.RpoIsotopes` instance and input the measured data:

```
ri = rp.RpoIsotopes.from_csv(
        iso_data,
        daem,
        ec,
        blk_corr = True,
        bulk_d13C_true = [-25.0, 0.1], #independently measured true mean, std.
```

[9] Hemingway et al., (2017), *Radiocarbon*, determine the blank carbon flux and isotope composition for the NOSAMS instrument. Additionaly, this manuscript estimates that a DE value of 0.3 - 1.8 J/mol best explains the NOSAMS Ramped PyrOx stable-carbon isotope KIE.

[10] Blank composition calculated for other Ramped PyrOx instuments can be inputted by changing the default `blk_d13C`, `blk_flux`, and `blk_Fm` parameters.

```
        mass_err = 0.01, #1 percent uncertainty in mass
        DE = 0.0018) #1.8 J/mol for KIE
```

While creating the *RpoIsotopes* instance and correcting isotope composition, this additionally calculated the distribution of E values contained within each RPO fraction. That is, carbon described by this distribution will decompose over the inputted temperature ranges and will result in the trapped $CO_2$for each fraction [5]. These distributions can now be compared with measured isotopes in order to determine the relationship between isotope composition and reaction energetics.

A summary table can be printed or saved to .csv according to:

```
#print to terminal
print(ri.ri_corr_info)

#save to .csv file
ri.ri_corr_info.to_csv('file_to_save.csv')
```

Note: This displays the fractionation, mass-balance, and KIE corrected isotope values. To view raw (inputted) values, use *ri_raw_info* instead.

This will result in a table similar to:

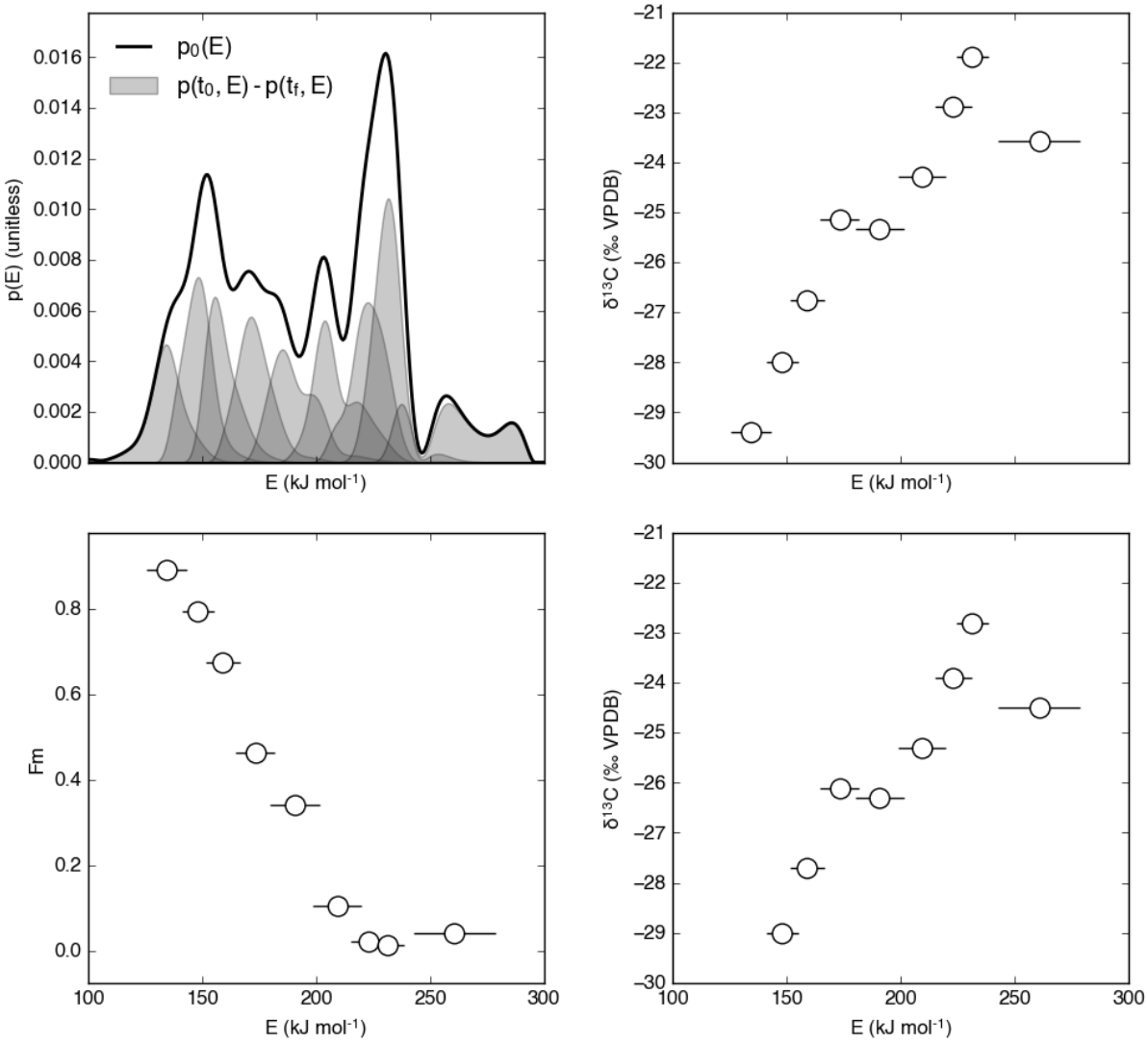
	t0 (s)	tf (s)	E (kJ/mol)	E_std	mass (ugC)	mass_std	d13C (VPDB)	d13C_std	Fm	Fm_std
1	754	2724	134.12	8.83	68.32	0.70	-29.40	0.15	0.89	3.55e-3
2	2724	3420	148.01	6.96	105.55	1.06	-27.99	0.15	0.80	2.21e-3
3	3420	3966	158.84	7.47	82.42	0.83	-26.76	0.15	0.68	2.81e-3
4	3966	4718	173.13	8.55	92.56	0.93	-25.14	0.15	0.46	3.21e-3
5	4718	5553	190.67	10.82	85.56	0.86	-25.33	0.15	0.34	2.82e-3
6	5553	6328	209.20	10.59	98.43	0.98	-24.29	0.15	0.11	2.22e-3
7	6328	6940	222.90	8.12	101.50	1.01	-22.87	0.15	0.02	1.91e-3
8	6940	7714	231.30	7.13	125.57	1.26	-21.88	0.15	0.01	1.81e-3
9	7714	11028	260.63	17.77	86.55	0.90	-23.57	0.16	0.04	2.42e-3

Additionally, the E distributions contained within each RPO fraction can be plotted along with isotope vs. E cross plots. Here, I'll plot the distributions and cross plots for both ^{13}C and ^{14}C (corrected). Lastly, I'll plot using the raw (uncorrected) ^{13}C values as a comparison:

```
#make a figure
fig, ax = plt.subplots(2, 2,
        figsize = (8,8),
        sharex = True)

#plot results
ax[0, 0] = ri.plot(
        ax = ax[0, 0],
        plt_var = 'p0E')

ax[0, 1] = ri.plot(
```

```
        ax = ax[0, 1],
        plt_var = 'd13C',
        plt_corr = True)

ax[1, 0] = ri.plot(
        ax = ax[1, 0],
        plt_var = 'Fm',
        plt_corr = True)

ax[1, 1] = ri.plot(
        ax = ax[1, 1],
        plt_var = 'd13C',
        plt_corr = False) #plotting raw values

#adjust the axes
ax[0,0].set_xlim([100,300])
ax[0,1].set_ylim([-30,-21])
ax[1,1].set_ylim([-30,-21])

plt.tight_layout()
```

Which generates a plot like this:



Additional Notes on the Kinetic Isotope Effect (KIE)

While the KIE has no effect on Fm values since they are fractionation-corrected by definition [11], mass-dependent kinetic fractionation effects must be explicitly accounted for when estimating the source carbon stable isotope composition during any kinetic experiment. For example, the KIE can lead to large isotope fractionation during thermal generation of methane and natural gas over geologic timescales [8] or during photodegradation of organic carbon by *uv* light [15].

As such, the `rampedpyrox` package allows for direct input of *DE* values [DE = E(^{13}C) - E(^{12}C), in kJ/mol] when correcting Ramped PyrOx isotopes. However, the magnitude of this effect is likely minimal within the NOSAMS Ramped PyrOx instrument – Hemingway et al. (2017) determined a best-fit value of 0.3e-3 - 1.8e-3 kJ/mol for a suite of standard reference materials [9] – and will therefore lead to small isotope corrections for samples analyzed on this instrument (*i.e.* << 1 per mille)

[11] See Stuiver and Polach (1977), *Radiocarbon*, **19(3)**, 355-363 for radiocarbon notation and data treatment.

6.1.8 Notes and References

6.2 Package Reference Documentation

The following classes and methods form the *rampedpyrox* package:

6.2.1 Ramped PyrOx classes

rampedpyrox.RpoThermogram(t, T[, g])	Class for inputting and storing Ramped PyrOx true (observed) and estim
rampedpyrox.Daem(E, log10k0, t, T)	Class to calculate the *DAEM* model transform.
rampedpyrox.EnergyComplex(E[, p])	Class for inputting and storing Ramped PryOx activation energy distribu
rampedpyrox.RpoIsotopes(model, ratedata, t_frac)	Class for inputting Ramped PyrOx isotopes, calculating p0(E) contained

rampedpyrox.RpoThermogram

class rampedpyrox.**RpoThermogram**(*t*, *T*, *g=None*)

Class for inputting and storing Ramped PyrOx true (observed) and estimated (forward-modelled) thermograms, calculating goodness of fit statistics, and reporting summary tables.

Parameters

- **t** (*array-like*) – Array of time, in seconds. Length *nt*.
- **T** (*array-like*) – Array of temperature, in Kelvin. Length *nt*.
- **g** (*None or array-like*) – Array of the true fraction of carbon remaining at each timepoint, with length *nt*. Defaults to *None*.

Warning:
UserWarning If attempting to use isothermal data to create an rp.RpoThermogram instance. Consider using an alternate rp.TimeData subclass (to be added in future versions).

Notes

Important: The inverse model used herein is highly sensitive to boundary effects. To avoid unnecessarily large regularizations ensure that inputted data are completely at baseline (ppm CO2 = 0) at the beginning and the end of the experiment (can use the *bl_subtract* flag to enforce that this is true.)

See also:

Daem rp.Model subclass used to generate the distributed activation energy model (DAEM)transform matrix for RPO data and translate between time- and E-space.

EnergyComplex rp.RateData subclass for storing and analyzing RPO energy (rate) data.

Examples

Generating an arbitrary bare-bones thermogram containing only *t* and *T*:

```
#import modules
import numpy as np
import rampedpyrox as rp

#generate arbitrary data
t = np.arange(1,100) #100 second experiment
beta = 0.5 #K/second
T = beta*t + 273.15 #K

#create instance
tg = rp.RpoThermogram(t,T)
```

Generating a real thermogram using an RPO output .csv file and the `rp.RpoThermogram.from_csv` class method, and subtracting the baseline:

```
#import modules
import rampedpyrox as rp

#create path to data file
file = 'path_to_folder_containing_data/thermogram_data.csv'

#create instance using baseline-subtracted CO2 data
tg = rp.RpoThermogram.from_csv(
        file,
        bl_subtract = True,
        nt = 250) #number of down-sampled time points
```

Manually adding some model-estimated fraction remaining data as *ghat*:

```
#assuming ghat has been generating using a ``rp.Daem`` model
tg.input_estimated(ghat)
```

Or, instead, you can input model-estimated g data directly from a given `rp.Daem` and `rp.EnergyComplex` instance (*i.e.* run the forward model):

```
#assuming ``rp.Daem`` named daem and ``rp.EnergyComplex`` named ec
tg.forward_model(daem, ec)
```

Plotting the resulting observed and modelled thermograms (note scatter when plotted against temp due to short fluctuations in measured ramp rate. For a "smooth" plot, always plot against time, as this is the master variable.):

```
#import additional modules
import matplotlib.pyplot as plt

#create figure
fig, ax = plt.subplots(1,2)

#plot resulting rates against time and temp
ax[0] = tg.plot(
        ax = ax[0],
        xaxis = 'time',
        yaxis = 'rate')

ax[1] = tg.plot(
        ax = ax[1],
        xaxis = 'temp',
        yaxis = 'rate')
```

Printing a summary of the observed and modelled thermograms:

```
print(tg.tg_info)
print(tg.tghat_info)
```

Attributes

dghatdt [numpy.ndarray] Array of the derivative of the estimated fraction of carbon remaining with respect to time at each timepoint, in fraction/second. Length *nt*.

dghatdT [numpy.ndarray] Array of the derivative of the estimated fraction of carbon remaining with respect to temperature at each timepoint, in fraction/Kelvin. Length *nt*.

dgdt [numpy.ndarray] Array of the derivative of the true fraction of carbon remaining with respect to time at each timepoint, in fraction/second. Length *nt*.

dgdT [numpy.ndarray] Array of the derivative of the true fraction of carbon remaining with respect to temperature at each timepoint, in fraction/Kelvin. Length *nt*.

dTdt [numpy.ndarray] Array of the derivative of temperature with respect to time (*i.e.* the instantaneous ramp rate) at each timepoint, in Kelvin/second. Length *nt*.

g [numpy.ndarray] Array of the true fraction of carbon remaining at each timepoint. Length *nt*.

ghat [numpy.ndarray] Array of the estimated fraction of carbon remaining at each timepoint. Length *nt*.

nt [int] Number of timepoints.

resid [float] The residual root mean square error (RMSE) between observed and modelled thermograms, g and ghat.

t [numpy.ndarray] Array of timepoints, in seconds. Length *nt*.

T [numpy.ndarray] Array of temperature, in Kelvin. Length *nt*.

tg_info [pd.Series] Series containing the observed thermogram summary info:

t_max (s),

t_mean (s),

t_std (s),

T_max (K),

T_mean (K),

T_std (K),

max_rate (frac/s),

max_rate (frac/K),

tghat_info [pd.Series] Series containing the modelled thermogram summary info:

t_max (s),

t_mean (s),

t_std (s),

T_max (K),

T_mean (K),

T_std (K),

max_rate (frac/s),

max_rate (frac/K),

Methods

forward_model(model, ratedata)	Forward-models rate data for a given model and populates the thermogram with model-estimat
from_csv(file[, bl_subtract, nt])	Class method to directly import RPO data from a .csv file and create an rp.RpoThermogra
input_estimated(ghat)	Inputs estimated thermogram into the rp.RpoThermogram instance and calculates statistics
plot([ax, xaxis, yaxis])	Plots the true and model-estimated thermograms against time or temp.

rampedpyrox.RpoThermogram.forward_model

RpoThermogram.**forward_model**(*model*, *ratedata*)

Forward-models rate data for a given model and populates the thermogram with model-estimated data.

Parameters

- **model** (*rp.Model*) – The rp.Daem instance used to calculate the forward model.
- **ratedata** (*rp.RateData*) – The rp.EnergyComplex instance containing the reactive continuum data.

Warning:
UserWarning If using an an isothermal model type for an RPO run.
UserWarning If using a non-energy complex ratedata type for an RPO run.

Raises

- ArrayError – If *nE* is not the same in the rp.Model instance and the rp.RateData instance.
- ArrayError – If *nt* is not the same in the rp.Model instance and the rp.TimeData instance.
- ArrayError – If the rp.RateData instance has no attribute *p*.

See also:

input_estimated() Method used for inputting model-estimated data directly.

EnergyComplex.inverse_model Class for creating an rp.EnergyComplex instance and calculating the inverse model.

rampedpyrox.RpoThermogram.from_csv

classmethod RpoThermogram.**from_csv**(*file*, *bl_subtract=True*, *nt=250*)

Class method to directly import RPO data from a .csv file and create an rp.RpoThermogram class instance.

Parameters

- **file** (*str or pd.DataFrame*) – File containing isotope data, either as a path string or a dataframe.
- **bl_subtract** (*Boolean*) – Tells the program whether or not to linearly subtract the baseline such that ppmCO2 returns to 0 at the beginning and end of the run. Defaults to *True*. **To minimize boundary effects, this should typically be set to 'True' regardless of previous data treatment.**
- **nt** (*int*) – The number of time points to use. Defaults to 250.

Notes

If using the *all_data* file generated by the NOSAMS RPO LabView program, the date_time column must be converted to **hh:mm:ss AM/PM** format and a header row should be added with the following columns:

date_time,

T_room,

P_room,

CO2_raw,

corr_int,

corr_slope,

temp,

CO2_scaled,

flow_rate,

dTdt,

fraction,

ug_frac,

ug_sum

(Note that all columns besides *date_time*, *temp*, and *CO2_scaled* are unused.) Ensure that all rows before the start of temperature ramping and after the ovens have been turned off have been removed.

When down-sampling, t contains the midpoints of each time bin and g and T contain the corresponding temp. and fraction remaining at each midpoint.

See also:

RpoIsotopes.from_csv() Classmethod for creating `rp.RpoIsotopes` instance directly from a .csv file.

rampedpyrox.RpoThermogram.input_estimated

`RpoThermogram.`**`input_estimated`**(*ghat*)

Inputs estimated thermogram into the `rp.RpoThermogram` instance and calculates statistics.

Parameters **`ghat`** (*array-like*) – Array of estimated fraction of total carbon remaining at each timestep. Length *nt*.

See also:

forward_model() Method for directly inputting estimated data from a given model and ratedata.

rampedpyrox.RpoThermogram.plot

`RpoThermogram.`**`plot`**(*ax=None*, *xaxis='time'*, *yaxis='rate'*)

Plots the true and model-estimated thermograms against time or temp.

Parameters

- **ax** (*None or matplotlib.axis*) – Axis to plot on. If *None*, automatically creates a `matplotlip.axis` instance to return. Defaults to *None*.
- **xaxis** (*str*) – Sets the x axis unit, either 'time' or 'temp'. Defaults to 'time'.
- **yaxis** (*str*) – Sets the y axis unit, either 'fraction' or 'rate'. Defaults to 'rate'.

Returns **ax** – Updated axis instance with plotted data.

Return type matplotlib.axis

Raises

- `StringError` – If *xaxis* is not 'time' or 'temp'.
- `StringError` – if *yaxis* is not 'fraction' or 'rate'.

rampedpyrox.Daem

class `rampedpyrox.`**Daem**(*E*, *log10k0*, *t*, *T*)

Class to calculate the *DAEM* model transform. Used for ramped-temperature kinetic problems such as Ramped PyrOx, pyGC, TGA, etc.

Parameters

- **E** (*array-like*) – Array of E values, in kJ/mol. Length *nE*.
- **log10k0** (*scalar, array-like, or lambda function*) – Arrhenius pre-exponential factor, either a constant value, array-like with length *nE*, or a lambda function of E (in kJ).
- **t** (*array-like*) – Array of time, in seconds. Length *nt*.
- **T** (*array-like*) – Array of temperature, in Kelvin. Length *nt*.

Warning:
UserWarning If attempting to use isothermal data to create a `Daem` model instance.

See also:

RpoThermogram `rp.TimeData` subclass for storing and analyzing RPO time/temperature data.

EnergyComplex `rp.RateData` subclass for storing and analyzing RPO rate data.

Examples

Creating a DAEM using manually-inputted *E*, *k0*, *t*, and *T*:

```
#import modules
import numpy as np
import rampedpyrox as rp

#generate arbitrary data
t = np.arange(1,100) #100 second experiment
beta = 0.5 #K/second
T = beta*t + 273.15 #K

E = np.arange(50, 350) #kJ/mol
log10k0 = 10 #s-1
```

```
#create instance
daem = rp.Daem(E, log10k0, t, T)
```

Creating a DAEM from real thermogram data using the `rp.Daem.from_timedata` class method:

```
#import modules
import rampedpyrox as rp

#create thermogram instance
tg = rp.RpoThermogram.from_csv('some_data_file.csv')

#create Daem instance
daem = rp.Daem.from_timedata(
        tg,
        E_max = 350,
        E_min = 50,
        nE = 250,
        log10k0 = 10)
```

Creating a DAEM from an energy complex using the `rp.Daem.from_ratedata` class method:

```
#import modules
import rampedpyrox as rp

#create energycomplex instance
ec = rp.EnergyComplex(E, p0E)

#create Daem instance
daem = rp.Daem.from_ratedata(
        ec,
        beta = 0.08,
        log10k0 = 10,
        nt = 250,
        t0 = 0,
        T0 = 373,
        tf = 1e4)
```

Plotting the L-curve of a Daem to find the best-fit omega value:

```
#import modules
import matplotlib.pyplot as plt

#create figure
fig, ax = plt.subplots(1,1)

#plot L curve
om_best, ax = daem.calc_L_curve(
        tg,
        ax = None,
        plot = True,
        om_min = 1e-3,
        om_max = 1e2,
        nOm = 150)
```

Attributes

A : np.ndarray

E [np.ndarray] Array of E values, in kJ/mol. Length *nE*.

nE [int] Number of activation energy points.

nt [int] Number of timepoints.

t [np.ndarray] Array of timepoints, in seconds. Length *nt*.

T [np.ndarray] Array of temperature, in Kelvin. Length *nt*.

References

[1] R.L Braun and A.K. Burnham (1987) Analysis of chemical reaction kinetics using a distribution of activation energies and simpler models. *Energy & Fuels*, **1**, 153-161.

[2] B. Cramer et al. (1998) Modeling isotope fractionation during primary cracking of natural gas: A reaction kinetic approach. *Chemical Geology*, **149**, 235-250.

[3] V. Dieckmann (2005) Modeling petroleum formation from heterogeneous source rocks: The influence of frequency factors on activation energy distribution and geological prediction. *Marine and Petroleum Geology*, **22**, 375-390.

[4] D.C. Forney and D.H. Rothman (2012) Common structure in the heterogeneity of plant-matter decay. *Journal of the Royal Society Interface*, rsif.2012.0122.

[5] D.C. Forney and D.H. Rothman (2012) Inverse method for calculating respiration rates from decay time series. *Biogeosciences*, **9**, 3601-3612.

[6] P.C. Hansen (1987) Rank-deficient and discrete ill-posed problems: Numerical aspects of linear inversion (monographs on mathematical modeling and computation). *Society for Industrial and Applied Mathematics*.

[7] P.C. Hansen (1994) Regularization tools: A Matlab package for analysis and solution of discrete ill-posed problems. *Numerical Algorithms*, **6**, 1-35.

[8] C.C. Lakshmananan et al. (1991) Implications of multiplicity in kinetic parameters to petroleum exploration: Distributed activation energy models. *Energy & Fuels*, **5**, 110-117.

[9] J.E. White et al. (2011) Biomass pyrolysis kinetics: A comparative critical review with relevant agricultural residue case studies. *Journal of Analytical and Applied Pyrolysis*, **91**, 1-33.

Methods

calc_L_curve(timedata[, ax, nOm, om_max, ...])	Function to calculate the L-curve for a given model and timedata instance in o
from_ratedata(ratedata[, beta, log10k0, nt, ...])	Class method to directly generate an `rp.Daem` instance using data stored in a
from_timedata(timedata[, E_max, E_min, ...])	Class method to directly generate an `rp.Daem` instance using data stored in a

rampedpyrox.Daem.calc_L_curve

`Daem.`**`calc_L_curve`**(*timedata*, *ax=None*, *nOm=150*, *om_max=100.0*, *om_min=0.001*, *plot=False*)

Function to calculate the L-curve for a given model and timedata instance in order to choose the best-fit smoothing parameter, omega.

Parameters

- **timedata** (*rp.TimeData*) – `rp.TimeData` instance containing the time and fraction remaining arrays to use in L curve calculation.
- **ax** (*None or matplotlib.axis*) – Axis to plot on. If *None* and `plot = True`, automatically creates a `matplotlip.axis` instance to return. Defaults to *None*.
- **nOm** (*int*) – Number of omega values to consider. Defaults to 150.

- **om_max** (*float or int*) – Maximum omega value to search. Defaults to 1e2.
- **om_min** (*float or int*) – Minimum omega value to search. Defaults to 1e-3.
- **plot** (*Boolean*) – Tells the method to plot the resulting L curve or not. Defaults to *False*.

Returns

- **om_best** (*float*) – The calculated best-fit omega value.
- **axis** (*None or matplotlib.axis*) – If `plot = True`, returns an updated axis handle with plot.

Raises

- `ScalarError` – If *om_max* or *om_min* are not scalar.
- `ScalarError` – If *nOm* is not int.

See also:

calc_L_curve() Package-level method for `calc_L_curve`.

References

[1] D.C. Forney and D.H. Rothman (2012) Inverse method for calculating respiration rates from decay time series. *Biogeosciences*, **9**, 3601-3612.

[2] P.C. Hansen (1987) Rank-deficient and discrete ill-posed problems: Numerical aspects of linear inversion (monographs on mathematical modeling and computation). *Society for Industrial and Applied Mathematics*.

[3] P.C. Hansen (1994) Regularization tools: A Matlab package for analysis and solution of discrete ill-posed problems. *Numerical** **Algorithms*, **6**, 1-35.

rampedpyrox.Daem.from_ratedata

classmethod `Daem.`**`from_ratedata`**(*ratedata*, *beta=0.08*, *log10k0=10*, *nt=250*, *t0=0*, *T0=373*, *tf=10000.0*)

Class method to directly generate an `rp.Daem` instance using data stored in an `rp.RateData` instance.

Parameters

- **ratedata** (*rp.RateData*) – `rp.RateData` instance containing the E array to use for creating the DAEM.
- **beta** (*int or float*) – Temperature ramp rate to use in model, in Kelvin/second. Defaults to 0.08 (*i.e.* 5K/min)
- **log10k0** (*scalar, array-like, or lambda function*) – Arrhenius pre-exponential factor, either a constant value, array- likewith length *nE*, or a lambda function of E. Defaults to 10.
- **nt** (*int*) – The number of time points to use. Defaults to 250.
- **t0** (*int or float*) – The initial time to be used in the model, in seconds. Defaults to 0.
- **T0** (*int or float*) – The initial temperature to be used in the model, in Kelvin. Defaults to 373.

- **tf** (*int or float*) – The final time to be used in the model, in seconds. Defaults to 10,000.

Warning:
UserWarning If attempting to create a DAEM with a non-EnergyComplex ratedata instance.

See also:

from_timedata() Class method to directly generate an `rp.Daem` instance using data stored in an `rp.TimeData` instance.

rampedpyrox.Daem.from_timedata

classmethod `Daem.`**from_timedata**(*timedata*, *E_max=350*, *E_min=50*, *log10k0=10*, *nE=250*)

Class method to directly generate an `rp.Daem` instance using data stored in an `rp.TimeData` instance.

Parameters

- **timedata** (*rp.TimeData*) – `rp.TimeData` instance containing the time array to use for creating the DAEM.
- **E_max** (*int*) – The maximum activation energy value to consider, in kJ/mol. Defaults to 350.
- **E_min** (*int*) – The minimum activation energy value to consider, in kJ/mol. Defaults to 50.
- **log10k0** (*scalar, array-like, or lambda function*) – Arrhenius pre-exponential factor, either a constant value, array- likewith length *nE*, or a lambda function of E. Defaults to 10.
- **nE** (*int*) – The number of activation energy points. Defaults to 250.

Warning:
UserWarning If attempting to create a DAEM with an isothermal timedata instance.

See also:

from_ratedata() Class method to directly generate an `rp.Daem` instance using data stored in an `rp.RateData` instance.

rampedpyrox.EnergyComplex

class `rampedpyrox.`**EnergyComplex**(*E*, *p=None*)

Class for inputting and storing Ramped PryOx activation energy distributions.

Parameters

- **E** (*array-like*) – Array of activation energy, in kJ/mol. Length *nE*.
- **p** (*None or array-like*) – Array of the regularized pdf of the E distribution, p0E. Length *nE*. Defaults to *None*.

Raises `ArrayError` – If the any value in *E* is negative.

See also:

Daem `rp.Model` subclass used to generate the Laplace transform for RPO data and translate between time- and E-space.

RpoThermogram `rp.TimeData` subclass containing the time and fraction remaining data used for the inversion.

Examples

Generating a bare-bones energy complex containing only E and p:

```
#import modules
import rampedpyrox as rp
import numpy as np

#generate arbitrary Gaussian data
E = np.arange(50, 350)

def Gaussian(x, mu, sig):
        scalar = (1/np.sqrt(2*np.pi*sig**2))*
        y = scalar*np.exp(-(x-mu)**2/(2*sig**2))
        return y

p = Gaussian(E, 150, 10)

#create the instance
ec = rp.EnergyComplex(E, p = p)
```

Or, insteand run the inversion to generate an energy complex using an `rp.RpoThermogram` instance, tg, and an `rp.Daem` instance, daem:

```
#keeping defaults, not combining any peaks
ec = rp.EnergyComplex(
        daem,
        tg,
        omega = 'auto')
```

Plotting the resulting regularized energy complex:

```
#import additional modules
import matplotlib.pyplot as plt

#create figure
fig, ax = plt.subplots(1,1)

#plot resulting E pdf, p0E
ax = ec.plot(ax = ax)
```

Attributes

E [np.ndarray] Array of activation energy, in kJ/mol. Length nE.

nE [int] Number of E points.

ec_info [pd.Series] Series containing the observed EnergyComplex summary info:

E_max (kJ/mol),

E_mean (kJ/mol),

E_std (kJ/mol),

p0(E)_max (unitless)

omega [float] Tikhonov regularization weighting factor.

p [np.ndarray] Array of the pdf of the E distribution, p0E. Length *nEa*.

resid [float] The RMSE between the measured thermogram data and the estimated thermogram using the p (ghat). Used for determining the best-fit omega value.

rgh : The roughness RMSE. Used for determining best-fit omega value.

References

[1] B. Cramer (2004) Methane generation from coal during open system pyrolysis investigated by isotope specific, Gaussian distributed reaction kinetics. *Organic Geochemistry*, **35**, 379-392.

[2] D.C. Forney and D.H. Rothman (2012) Common structure in the heterogeneity of plant-matter decay. *Journal of the Royal Society Interface*, rsif.2012.0122.

[3] D.C. Forney and D.H. Rothman (2012) Inverse method for calculating respiration rates from decay time series. *Biogeosciences*, **9**, 3601-3612.

Methods

input_estimated([omega, resid, rgh])	Inputs estimated rate data into the `rp.EnergyComplex` instance and calculates st
inverse_model(model, timedata[, omega])	Generates an energy complex by inverting an `rp.TimeData` instance using a given
plot([ax])	Plots the pdf of E, p0E, against E.

rampedpyrox.EnergyComplex.input_estimated

`EnergyComplex.`**`input_estimated`**(*omega=0*, *resid=0*, *rgh=0*)

Inputs estimated rate data into the `rp.EnergyComplex` instance and calculates statistics.

Parameters

- **omega** (*scalar*) – Tikhonov regularization weighting factor used to generate estimated data. Defaults to 0.
- **resid** (*float*) – Residual RMSE for the inputted estimated data. Defaults to 0.
- **rgh** (*float*) – Roughness RMSE for the inputted estimated data. Defaults to 0.

rampedpyrox.EnergyComplex.inverse_model

classmethod `EnergyComplex.`**`inverse_model`**(*model*, *timedata*, *omega='auto'*)

Generates an energy complex by inverting an `rp.TimeData` instance using a given `rp.Model` instance.

Parameters

- **model** (*rp.Model*) – `rp.Model` instance containing the A matrix to use for inversion.
- **timedata** (*rp.TimeData*) – `rp.TimeData` instance containing the timeseries data to invert.
- **omega** (*scalar or 'auto'*) – Smoothing weighting factor for Tikhonov regularization. Defaults to 'auto'.

Warning:
UserWarning If `scipy.optimize.least_squares` cannot converge on a solution.
UserWarning If attempting to use timedata that is not a `rp.RpoThermogram` instance.
UserWarning If attempting to use a model that is not a `rp.Daem` instance.

See also:

RpoThermogram.forward_model() `rp.TimeData` method for forward-modeling an `rp.RateData` instance using a particular model.

rampedpyrox.EnergyComplex.plot

`EnergyComplex.`**`plot`**(*ax=None*)
Plots the pdf of E, p0E, against E.

Keyword Arguments **ax** (*None or matplotlib.axis*) – Axis to plot on. If *None*, automatically creates a `matplotlip.axis` instance to return. Defaults to *None*.

Returns **ax** – Updated axis instance with plotted data.

Return type matplotlib.axis

rampedpyrox.RpoIsotopes

class `rampedpyrox.`**`RpoIsotopes`**(*model*, *ratedata*, *t_frac*, *d13C_raw=None*, *d13C_raw_std=None*, *Fm_raw=None*, *Fm_raw_std=None*, *m_raw=None*, *m_raw_std=None*, *blk_corr=False*, *mb_corr=False*, *kie_corr=False*)
Class for inputting Ramped PyrOx isotopes, calculating p0(E) contained in each RPO fraction, correcting isotope values for blank contribution, mass balance, and kinetic fractionation (d13C only), and storing resulting data and statistics.

Parameters

- **`blk_corr`** (*`boolean`*) – Boolean to determine if inputted isotope data have been blank corrected, defaults to *False*.
- **`d13C_raw`** (*`None or array-like`*) – Array of the raw d13C values (VPDB) of each measured fraction, length *nFrac*. Defaults to *None*.
- **`d13C_raw_std`** (*`None or array-like`*) – The standard deviation of *d13C_raw* with length *nFrac*. Defaults to zeros or *None* if *d13C_raw* is *None*.
- **`Fm_raw`** (*`None or array-like`*) – Array of the raw Fm values of each measured fraction, length *nFrac*. Defaults to *None*.
- **`Fm_raw_std`** (*`None or array-like`*) – The standard deviation of *Fm_raw* with length *nFrac*. Defaults to zeros or *None* if *Fm_raw* is *None*.
- **`kie_corr`** (*`boolean`*) – Boolean to determine if inputted d13C data have been fractionation corrected, defaults to *False*.
- **`m_raw`** (*`None or array-like`*) – Array of the raw masses (ugC) of each measured fraction, length *nFrac*. Defaults to *None*.
- **`m_raw_std`** (*`None or array-like`*) – The standard deviation of *d13C_raw* with length *nFrac*. Defaults to zero or *None* if *m_raw* is *None*.

- **mb_corr** (*boolean*) – Boolean to determine if inputted d13C data have been mass-balance corrected, defaults to *False*.
- **model** (*rp.Daem*) – `rp.Daem` instance associated with the inputted energy complex, used for calculating the fractional E distributions and for KIE d13C correction.
- **ratedata** (*rp.EnergyComplex*) – `rp.EnergyComplex` instance containing p0(E) distribution for the thermogram associated with inputted isotopes. Used for calculating the fractional E distributions and for KIE d13C correction.
- **t_frac** (*None or array-like*) – 2d array of the initial and final times of each fraction, in seconds. Shape [*nFrac* x 2]. Defaults to *None*.

Warning:
UserWarning If using an an isothermal model type for an RPO run.
UserWarning If using a non-energy complex ratedata type for an RPO run.

Raises

- `ArrayError` – If *t_frac* is not array-like.
- `ArrayError` – If *nE* is not the same in the `rp.Model` instance and the `rp.RateData` instance.

Notes

When inputting *t_frac*, a time of 0 (i.e. *t0*, the initial time) is defined as the first timepoint in the `RpoThermogram` instance. If time passed between the thermogram *t0* and the beginning of fraction 1 trapping (as is almost always the case), *t_frac* must be adjusted accordingly. This is done automatically when importing from .csv (see `RpoIsotopes.from_csv`) documenatation for info.

See also:

Daem `Model` subclass used to generate the transform for RPO data and translate between time- and E-space.

EnergyComplex `RateData` subclass for storing and analyzing RPO rate data.

RpoThermogram `TimeData` subclass containing the time and fraction remaining data used for the inversion.

Examples

Generating a bare-bones isotope result instance containing only arbitrary time and Fm data for a given energy complex instance, ec, and a given model instance, Daem:

```
#import modules
import rampedpyrox as rp
import numpy as np

#generate arbitrary data for 3 fractions
t_frac = [[100, 200], [200, 300], [300, 1000]]
t_frac = np.array(t_frac)

Fm_raw = [1.0, 0.5, 0.0]

#create instance
ri = rp.RpoIsotopes(
        daem,
        ec,
```

```
        t_frac,
        Fm_raw = Fm_raw)
```

Generating a isotope result instance using an RPO output .csv file and the `RpoIsotopes.from_csv` class method:

```
#import modules
import rampedpyrox as rp

#create path to data file
file = 'path_to_folder_containing_data/isotope_data.csv'

#create instance
ri = rp.RpoThermogram.from_csv(
        file,
        model,
        ratedata,
        blk_corr = True,
        mass_err = 0.01,
        DE = 0.0018)
```

This will automatically correct inputted isotopes for the inputted instrument blank carbon contribution using the *blk_corr* flag and will assumed a 1 percent uncertainty in mass measurements. Additionally, this will fractionation-correct d13C data (if they exist) using a KIE DE of 1.8 J/mol. **NOTE:** See `RpoIsotopes.from_csv` documentation for instructions on getting the .csv file in the right format.

Plotting resulting p0(E) contained in each RPO fraction:

```
#import additional modules
import matplotlib.pyplot as plt

#create figure
fig, ax = plt.subplots(1,3)

#plot p0(E) distributions
ax[0] = ri.plot(
        ax = ax[0],
        plt_var = 'p0E')
```

Plotting resulting isotope vs. E scatter plots:

```
#plot d13C data
ax[1] = ri.plot(
        ax = ax[1],
        plt_var = 'd13C',
        plt_corr = True) #plotting corrected values

#plot Fm data
ax[2] = ri.plot(
        ax = ax[2],
        plt_var = 'Fm',
        plt_corr = True) #plotting corrected values
```

Printing a summary of the raw and corrected isotope values:

```
#raw fraction information
print(ri.ri_raw_info)

#corrected fraction information
print(ri.ri_corr_info)
```

Attributes

d13C_corr [np.ndarray] Array of the d13C values (VPDB) of each measured fraction, corrected for any of: blank, mass-balance, KIE. Length *nFrac*.

d13C_corr_std [np.ndarray] The standard deviation of the d13C values (VPDB) of each measured fraction, corrected for any of: blank, mass-balance, KIE. Length *nFrac*.

d13C_raw [np.ndarray] Array of the raw d13C values (VPDB) of each measured fraction, length *nFrac*.

d13C_raw_std [np.ndarray] The standard deviation of *d13C_raw* with length *nFrac*.

E_frac [np.ndarray] Array of the mean E value (kJ) contained in each measured fraction as calculated by the inverse model, length *nFrac*.

E_frac_std [np.ndarray] The standard deviation of E (kJ) contained in each measured fraction as calculated by the inverse model, length *nFrac*.

Fm_corr [np.ndarray] Array of the blank-corrected Fm values of each measured fraction, length *nFrac*.

Fm_corr_std [np.ndarray] The standard deviation of *Fm_corr* with length *nFrac*.

Fm_raw [np.ndarray] Array of the raw Fm values of each measured fraction, length *nFrac*.

Fm_raw_std [np.ndarray] The standard deviation of *Fm_raw* with length *nFrac*.

m_corr [np.ndarray] Array of the blank-corrected masses (ugC) of each measured fraction, length *nFrac*.

m_corr_std [np.ndarray] The standard deviation of *m_corr* with length *nFrac*.

m_raw [np.ndarray] Array of the raw masses (ugC) of each measured fraction, length *nFrac*.

m_raw_std [np.ndarray] The standard deviation of *m_raw* with length *nFrac*.

nFrac [int] The number of measured fractions.

ri_corr_info [pd.DataFrame] Dataframe containing the inputted summary info, using corrected isotopes:

time (init. and final),

E (mean and std.),

mass (mean and std.),

d13C (mean and std.),

Fm (mean and std.)

ri_raw_info [pd.DataFrame] Dataframe containing the inputted summary info, using raw isotopes:

time (init. and final),

E (mean and std.),

mass (mean and std.),

d13C (mean and std.),

Fm (mean and std.)

t_frac [np.ndarray] 2d array of the initial and final times of each fraction, in seconds. Shape [*nFrac* x 2].

Methods

blank_correct([blk_d13C, blk_flux, blk_Fm, ...])	Method to blank- and mass-balance correct raw isotope values.
from_csv(file, model, ratedata[, blk_corr, ...])	Class method to directly import RPO fraction data from a .csv file and create
kie_correct(model, ratedata[, DE])	Method for further correcting d13C values to account for kinetic isotope frac
plot([ax, plt_var, plt_corr])	Method for plotting results, either p0(E) distributions contained within each

rampedpyrox.RpoIsotopes.blank_correct

RpoIsotopes.**blank_correct**(*blk_d13C=(-29.0, 0.1), blk_flux=(0.375, 0.0583), blk_Fm=(0.555, 0.042), bulk_d13C_true=None*)

Method to blank- and mass-balance correct raw isotope values.

Parameters

- **blk_d13C** (*tuple*) – Tuple of the blank d13C composition (VPDB), in the form (mean, stdev.) to be used of `blk_corr = True`. Defaults to the NOSAMS RPO blank as calculated by Hemingway et al., Radiocarbon **2017**.
- **blk_flux** (*tuple*) – Tuple of the blank flux (ng/s), in the form (mean, stdev.) to be used of `blk_corr = True`. Defaults to the NOSAMS RPO blank as calculated by Hemingway et al., Radiocarbon **2017**.
- **blk_Fm** (*tuple*) – Tuple of the blank Fm value, in the form (mean, stdev.) to be used of `blk_corr = True`. Defaults to the NOSAMS RPO blank as calculated by Hemingway et al., Radiocarbon **2017**.
- **bulk_d13C_true** (*None or array*) – True measured d13C value (VPDB) for bulk material as measured independently (e.g. on a EA-IRMS). If not *None*, this value is used to mass-balance-correct d13C values as described in Hemingway et al., Radiocarbon **2017**. If not *none*, must be inputted in the form [mean, stdev.]

Warning:
UserWarning If already corrected for blank contribution
UserWarning If already corrected for 13C mass balance

References

[1] J.D. Hemingway et al. (2017) Assessing the blank carbon contribution, isotope mass balance, and kinetic isotope fractionation of the ramped pyrolysis/oxidation instrument at NOSAMS. *Radiocarbon*

rampedpyrox.RpoIsotopes.from_csv

classmethod RpoIsotopes.**from_csv**(*file, model, ratedata, blk_corr=False, blk_d13C=(-29.0, 0.1), blk_flux=(0.375, 0.0583), blk_Fm=(0.555, 0.042), bulk_d13C_true=None, DE=0.0018, mass_err=0.01*)

Class method to directly import RPO fraction data from a .csv file and create an `RpoIsotopes` class instance.

Parameters

- **blk_corr** (*Boolean*) – Tells the method whether or not to blank-correct isotope data. If *True*, blank-corrects according to inputted blank composition values. If *bulk_d13C_true* is not *None*, further corrects d13C values to ensure isotope mass balance (see Hemingway et al., Radiocarbon **2017** for details).

- **blk_d13C** (*tuple*) – Tuple of the blank d13C composition (VPDB), in the form (mean, stdev.) to be used of `blk_corr = True`. Defaults to the NOSAMS RPO blank as calculated by Hemingway et al., Radiocarbon **2017**.
- **blk_flux** (*tuple*) – Tuple of the blank flux (ng/s), in the form (mean, stdev.) to be used of `blk_corr = True`. Defaults to the NOSAMS RPO blank as calculated by Hemingway et al., Radiocarbon **2017**.
- **blk_Fm** (*tuple*) – Tuple of the blank Fm value, in the form (mean, stdev.) to be used of `blk_corr = True`. Defaults to the NOSAMS RPO blank as calculated by Hemingway et al., Radiocarbon **2017**.
- **bulk_d13C_true** (*None or array*) – True measured d13C value (VPDB) for bulk material as measured independently (e.g. on a EA-IRMS). If not *None*, this value is used to mass-balance-correct d13C values as described in Hemingway et al., Radiocarbon **2017**. If not *none*, must be inputted in the form [mean, stdev.]
- **DE** (*scalar*) – Value for the difference in E between 12C- and 13C-containing atoms, in kJ. Defaults to 0.0018 (the best-fit value calculated in Hemingway et al., **2017**).
- **file** (*str or pd.DataFrame*) – File containing RPO isotope data, either as a string pointing to a .csv file or as a `pd.DataFrame` instance.
- **mass_err** (*float*) – Relative uncertainty in mass measurements, typically as a sum of manometric uncertainty in pressure measurements and uncertainty in vacuum line volumes. Defaults to 0.01 (i.e. 1% relative uncertainty).
- **model** (*rp.Model*) – `rp.Model` instance containing the A matrix to use for inversion.
- **ratedata** (*rp.RateData*) – `rp.Ratedata` instance containing the reactive continuum data.

Notes

For bookkeeping purposes, the first 2 rows must be fractions "-1" and "0", where the timestamp for fraction "-1" is the first point in the *all_data* file used to create the `rp.RpoThermogram` instance, and the timestamp for fraction "0" is the t0 for the first fraction.

If mass, d13C, and Fm data exist, column names must be the following:

'ug_frac' and 'ug_frac_std'

'd13C' and 'd13C_std'

'Fm' and 'Fm_std'

See also:

RpoThermogram.from_csv() Classmethod for creating `rp.RpoThermogram` instance directly from a .csv file.

References

[1] J.D. Hemingway et al. (2017) Assessing the blank carbon contribution, isotope mass balance, and kinetic isotope fractionation of the ramped pyrolysis/oxidation instrument at NOSAMS. *Radiocarbon*

rampedpyrox.RpoIsotopes.kie_correct

RpoIsotopes.**kie_correct**(*model*, *ratedata*, *DE=0.0018*)

Method for further correcting d13C values to account for kinetic isotope fractionation occurring within the instrument.

Parameters

- **model** (*rp.Model*) – rp.Model instance containing the A matrix to use for inversion.
- **ratedata** (*rp.RateData*) – rp.Ratedata instance containing the reactive continuum data.
- **DE** (*scalar*) – Value for the difference in E between 12C- and 13C-containing atoms, in kJ. Defaults to 0.0018 (the best-fit value calculated in Hemingway et al., **2017**).

Warning:
UserWarning If already corrected for kinetic fractionation

References

[1] J.D. Hemingway et al. (2017) Assessing the blank carbon contribution, isotope mass balance, and kinetic isotope fractionation of the ramped pyrolysis/oxidation instrument at NOSAMS. *Radiocarbon*

rampedpyrox.RpoIsotopes.plot

RpoIsotopes.**plot**(*ax=None*, *plt_var='p0E'*, *plt_corr=True*)

Method for plotting results, either p0(E) distributions contained within each RPO fraction or isotopes vs. mean E for each RPO fraction.

Parameters

- **ax** (*None or matplotlib.axis*) – Axis to plot on. If *None*, automatically creates a matplotlip.axis instance to return. Defaults to *None*.
- **plt_var** (*str*) – Tells the method which variable to plot, available options are: 'p0E' (for fraction-specific p0(E) distributions), 'Fm', and d13C (isotope vs. fraction E scatter plots).
- **plt_corr** (*str*) – If *plt_var* is 'Fm' or 'd13C', *plt_corr* tells the method whether to plot raw or corrected values (if corrected values exist).

Returns **ax** – Updated axis instance with plotted data.

Return type matplotlib.axis

Raises

- ArrayError – if *plt_corr* is *True* but no corrected data exist.
- StringError – If *plt_var* is not 'p0E', 'Fm', or 'd13C'.

6.2.2 Ramped PyrOx methods

rampedpyrox.assert_len(data, n)	Asserts that an array has length *n* and *float* datatypes.

Table 6.6 – continued from previous page

rampedpyrox.calc_L_curve(model, timedata[, ...])	Function to calculate the L-curve for a given model and timedata instanc
rampedpyrox.derivatize(num, denom)	Method for derivatizing numerator, *num*, with respect to denominator, *d*
rampedpyrox.extract_moments(x, y)	Extracts 1st (mean) and 2nd (stdev) moments from a distribution.
rampedpyrox.plot_tg_isotopes(timedata, result)	Function to plot raw timedata (e.g.

rampedpyrox.assert_len

rampedpyrox.**assert_len**(*data*, *n*)

Asserts that an array has length *n* and *float* datatypes.

Parameters

- **data** (*scalar or array-like*) – Array to assert has length n. If scalar, generates an np.ndarray with length n.
- **n** (*int*) – Length to assert

Returns **array** – Updated array, now of class np.ndarray and with length n.

Return type np.ndarray

Raises

- ArrayError – If inputted data not int or array-like (excluding string).
- LengthError – If length of the array is not n.

rampedpyrox.calc_L_curve

rampedpyrox.**calc_L_curve**(*model*, *timedata*, *ax=None*, *plot=False*, *nOm=150*, *om_max=100.0*, *om_min=0.001*)

Function to calculate the L-curve for a given model and timedata instance in order to choose the best-fit smoothing parameter, *omega*.

Parameters

- **model** (*rp.Model*) – rp.Model instance containing the A matrix to use for L curve calculation.
- **timedata** (*rp.TimeData*) – rp.TimeData instance containing the time and fraction remaining arrays to use in L curve calculation.

Keyword Arguments

- **ax** (*None or matplotlib.axis*) – Axis to plot on. If *None* and plot = True, automatically creates a matplotlip.axis instance to return. Defaults to *None*.
- **plot** (*Boolean*) – Tells the method to plot the resulting L curve or not.
- **om_min** (*int*) – Minimum omega value to search. Defaults to 1e-3.
- **om_max** (*int*) – Maximum omega value to search. Defaults to 1e2.
- **nOm** (*int*) – Number of omega values to consider. Defaults to 150.

Returns

- **om_best** (*float*) – The calculated best-fit omega value.
- **axis** (*None or matplotlib.axis*) – If plot = True, returns an updated axis handle with plot.

Raises

- `ScalarError` – If *om_max* or *om_min* are not int or float.
- `ScalarError` – If *nOm* is not int.

See also:

Daem.calc_L_curve() Instance method for `calc_L_curve`.

References

[1] D.C. Forney and D.H. Rothman (2012) Inverse method for calculating respiration rates from decay time series. *Biogeosciences*, **9**, 3601-3612.

[2] P.C. Hansen (1987) Rank-deficient and discrete ill-posed problems: Numerical aspects of linear inversion (monographs on mathematical modeling and computation). *Society for Industrial and Applied Mathematics*.

[3] P.C. Hansen (1994) Regularization tools: A Matlab package for analysis and solution of discrete ill-posed problems. *Numerical Algorithms*, **6**, 1-35.

rampedpyrox.derivatize

`rampedpyrox.`**`derivatize`**(*num*, *denom*)

Method for derivatizing numerator, *num*, with respect to denominator, *denom*.

Parameters

- **num** (*int or array-like*) – The numerator of the numerical derivative function.
- **denom** (*array-like*) – The denominator of the numerical derivative function. Length *n*.

Returns **derivative** – An `np.ndarray` instance of the derivative. Length *n*.

Return type rparray

Raises `ArrayError` – If *denom* is not array-like.

See also:

`numpy.gradient()` The method used to calculate derivatives

Notes

This method uses the `np.gradient` method to calculate derivatives. If *denom* is a scalar, resulting array will be all `np.inf`. If both *num* and *denom* are scalars, resulting array will be all `np.nan`. If either *num* or *denom* are 1d and the other is 2d, derivative will be calculated column-wise. If both are 2d, each column will be derivatized separately.

rampedpyrox.extract_moments

`rampedpyrox.`**`extract_moments`**(*x*, *y*)

Extracts 1st (mean) and 2nd (stdev) moments from a distribution.

Parameters

- **x** (*np.ndarray*) – Array of x values, length *n*.

- **y** (*np.ndarray*) – Array of y values, length *n*.

Returns

- **mu** (*float*) – First moment of distribution.
- **sigma** (*float*) – Second moment of distribution.

rampedpyrox.plot_tg_isotopes

rampedpyrox.**plot_tg_isotopes**(*timedata*, *result*, *ax=None*, *plt_corr=True*)

Function to plot raw timedata (e.g. RPO thermogram) and isotope values.

Parameters

- **ax** (*None or matplotlib.axis*) – Axis to plot on. If *None*, automatically creates a matplotlip.axis instance to return. Defaults to *None*.
- **plt_corr** (*str*) – If *plt_var* is 'Fm' or 'd13C', *plt_corr* tells the method whether to plot raw or corrected values (if corrected values exist).
- **result** (*rp.Results*) – rp.Results instance containing the isotope results to plot.
- **timedata** (*rp.TimeData*) – rp.TimeData instance containing the derivative time-data (e.g. rpo thermogram) to plot.

Returns **ax** – Updated axis instance with plotted data.

Return type matplotlib.axis

Warning:
UserWarning If *timedata* does not contain derivative timedata, dgdt.
UserWarning If *result* does not contain any of the necessary isotope attributes.

ArrayError if *plt_corr* is *True* but no corrected data exist.

ArrayError If *result* does not contain any of: d13C, Fm.

6.2.3 References

The following references were used during creation of the core rampedpyrox pacakge or provide information regarding the choice of user-inputted parameters (*i.e. logk0*, *omega*, and DE).

[1] R.L Braun and A.K. Burnham (1987) Analysis of chemical reaction kinetics using a distribution of activation energies and simpler models. *Energy & Fuels*, **1**, 153-161.

[2] B. Cramer (2004) Methane generation from coal during open system pyrolysis investigated by isotope specific, Gaussian distributed reaction kinetics. *Organic Geochemistry*, **35**, 379-392.

[3] B. Cramer et al. (1998) Modeling isotope fractionation during primary cracking of natural gas: A reaction kinetic approach. *Chemical Geology*, **149**, 235-250.

[4] B. Cramer et al. (2001) Reaction kinetics of stable carbon isotopes in natural gas – Insights from dry, open system pyrolysis experiments. *Energy & Fuels*, **15**, 517-532.

[5] V. Dieckmann (2005) Modeling petroleum formation from heterogeneous source rocks: The influence of frequency factors on activation energy distribution and geological prediction. *Marine and Petroleum Geology*, **22**, 375-390.

[6] D.C. Forney and D.H. Rothman (2012) Common structure in the heterogeneity of plant-matter decay. *Journal of the Royal Society Interface*, rsif.2012.0122.

[7] D.C. Forney and D.H. Rothman (2012) Inverse method for calculating respiration rates from decay time series. *Biogeosciences*, **9**, 3601-3612.

[8] P.C. Hansen (1987) Rank-deficient and discrete ill-posed problems: Numerical aspects of linear inversion (monographs on mathematical modeling and computation). *Society for Industrial and Applied Mathematics*.

[9] P.C. Hansen (1994) Regularization tools: A Matlab package for analysis and solution of discrete ill-posed problems. *Numerical Algorithms*, **6**, 1-35.

[10] J.D. Hemingway et al. (2017) Assessing the blank carbon contribution, isotope mass balance, and kinetic isotope fractionation of the ramped pyrolysis/oxidation instrument at NOSAMS. *Radiocarbon*, **in press**.

[11] C.C. Lakshmananan et al. (1991) Implications of multiplicity in kinetic parameters to petroleum exploration: Distributed activation energy models. *Energy & Fuels*, **5**, 110-117.

[12] Rosenheim et al. (2008) Antarctic sediment chronology by programmed-temperature pyrolysis: Methodology and data treatment. *Geochemistry, Geophysics, Geosystems*, **9(4)**, GC001816.

[13] J.E. White et al. (2011) Biomass pyrolysis kinetics: A comparative critical review with relevant agricultural residue case studies. *Journal of Analytical and Applied Pyrolysis*, **91**, 1-33.

CHAPTER 7

Indices and tables

- genindex
- search