python **I**soconversional **C**omputations for **N**on-**I**sothermal **K**inetics.
pICNIK

Erick Ramírez Orozco

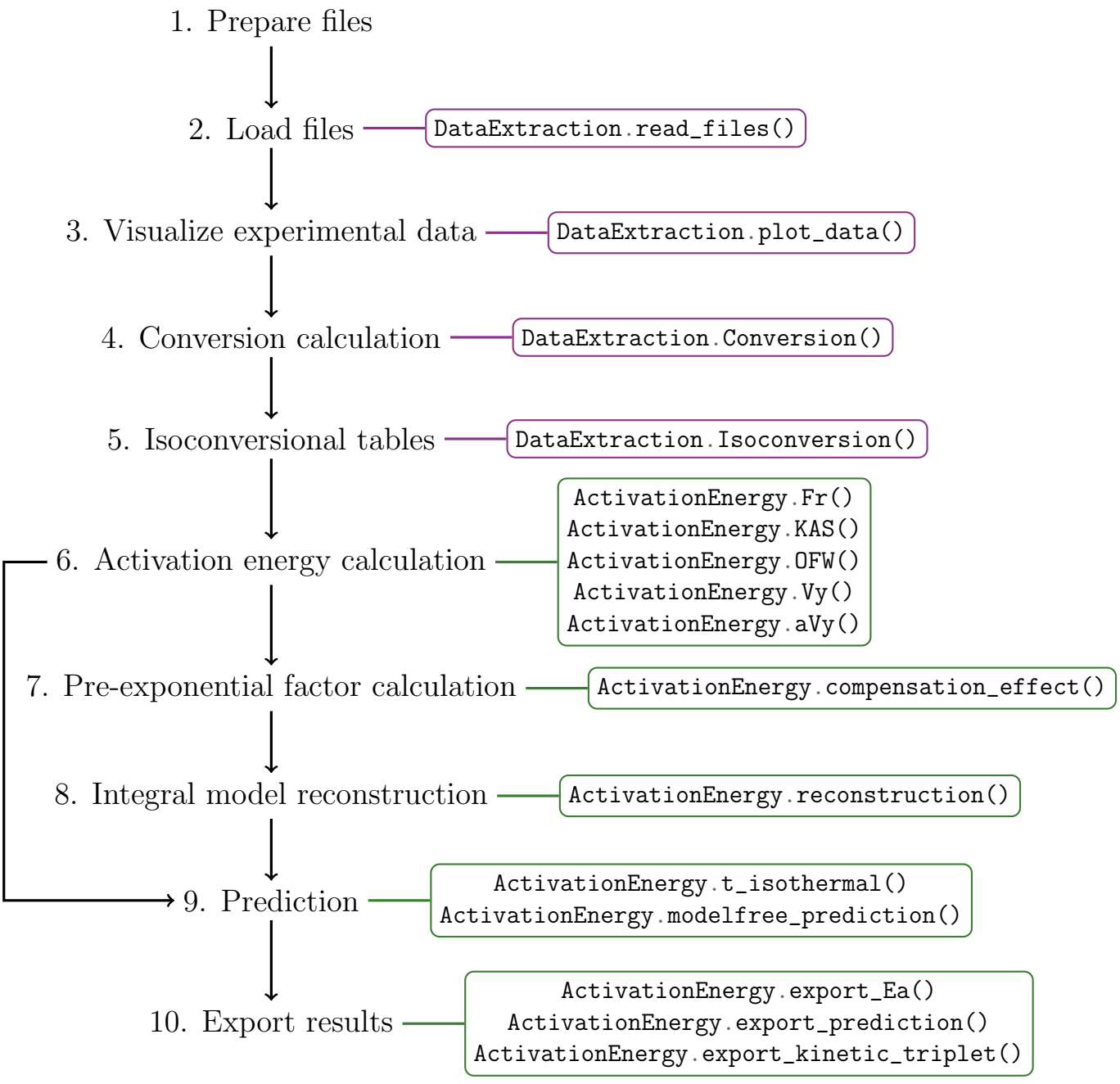
October 2024

1 The Work Flow

The pICNIK package is thought to be used as an interactive python package to analyze thermally controlled data. As an interactive package it is recommended to work with an ipython platform such as jupyter[ref] notebooks.

The package consists on two objects: `DataExtraction` and `ActivationEnergy`.

The workflow to analyze data with pICNIK consists in the following steps:



1.1 Tutorial. Single step process.

To illustrate the use of pICNIK lets work with an example. To follow this tutorial, data files can be downloaded from:
https://github.com/ErickErock/pICNIK/tree/main/examples/Constant_E

The data used in this example is a simulated process that follows an F1 reaction model: $f(\alpha) = 1 - \alpha$, with the Arrhenius parameters: $E = 75$ kJ/mol and $\ln(A/min) = 12$

The data represents four thermogravitric experiments of linear temperature programs with heating rates of $(2.5, 5, 10$ and $20)$ K/min

1.1.1 Prepare files

pICNIK works with files with extension .csv or .dat. The files must be made of three columns: "time" in minutes [min], "temperature" in Kelvin [K]. The third column was originally thought as the mass in milligrams [mg], but it can be any other temperature-controlled physical property of the sample or even the conversion values computed by other means. Once you've prepared all your data files, create a list containing the paths to the files[1]:

```
one_step = ['HOME/pICNIK/examples/Constant_E/E_cnt_2.5.csv',
            'HOME/pICNIK/examples/Constant_E/E_cnt_5.csv',
            'HOME/pICNIK/examples/Constant_E/E_cnt_10.csv',
            'HOME/pICNIK/examples/Constant_E/E_cnt_20.csv']
```

1.1.2 Load files

The `DataExtraction` object is equipped with functions to read and manipulate data from thermally controlled experiments. An exahustive list of all available functions, along with a description, can be found in the ——— section of this manual.

To load files you have to create an instance of the `DataExtraction` object. Then, simply use the `read_files` function.

```
import picnik as pnk

xtr = pnk.DataExtraction
B, T0 = xtr.read_files(data)
```

The `read_files` function takes as obligatory parameter the list of paths to the data files and returns a `numpy`[ref] array of computed heating rates (`B`), another array containing the initial experimental temperature in K (`T0`) and a graphical summary of the expirmental data (Fig. 1.1).

1.1.3 Visualize experimental data and conversion calculation

To calculate conversion values one must first identify the temperature range of interest. This can me done with the `plot_data` function to visualize the change in mass as a function of temperature.

[1] This information can be found in the properties of the file.

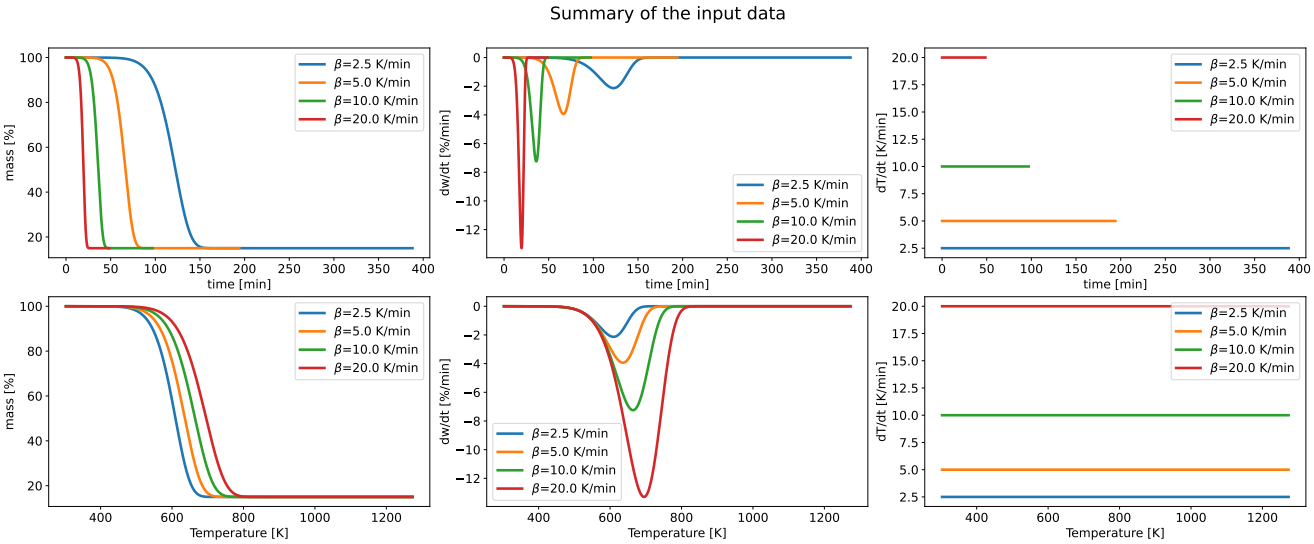


Figure 1.1: Results obtained from the `read_files` function. Top: time dependent data. Bottom: temperature dependent data.

```
xtr.plot_data(x_data='temperature',
              y_data='TG',
              x_units='K',
              y_units='%')
```

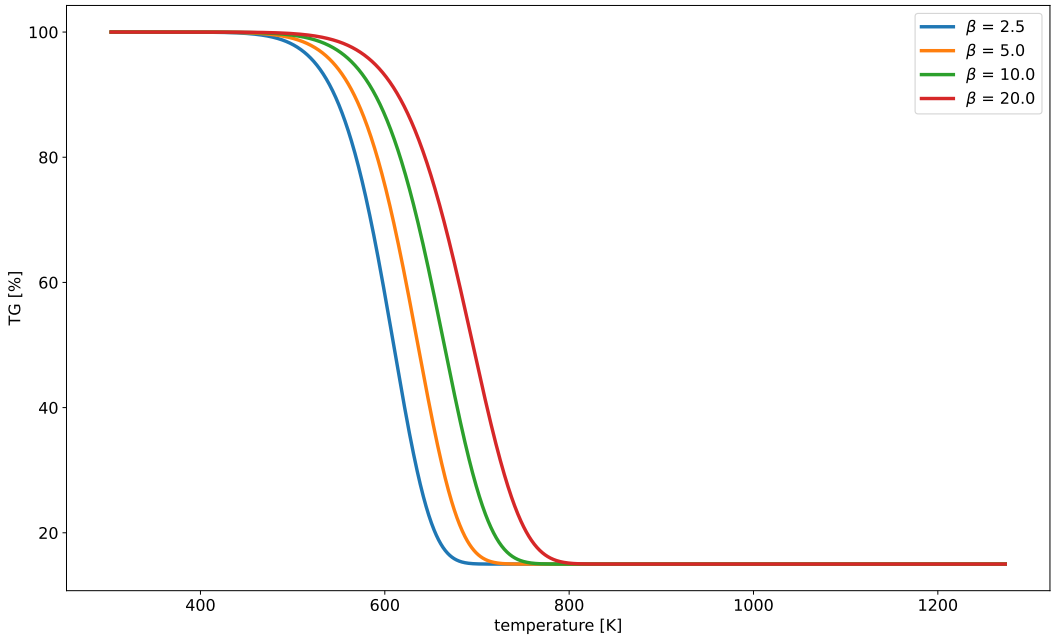


Figure 1.2: Temperature dependent weight loss.

Then, the conversion is computed by feeding the temperature range to the `Conversion` function. This function returns a plot of conversion against temperature along with the temperature range (Fig. 1.3).

```
xtr.Conversion(420,820)
```

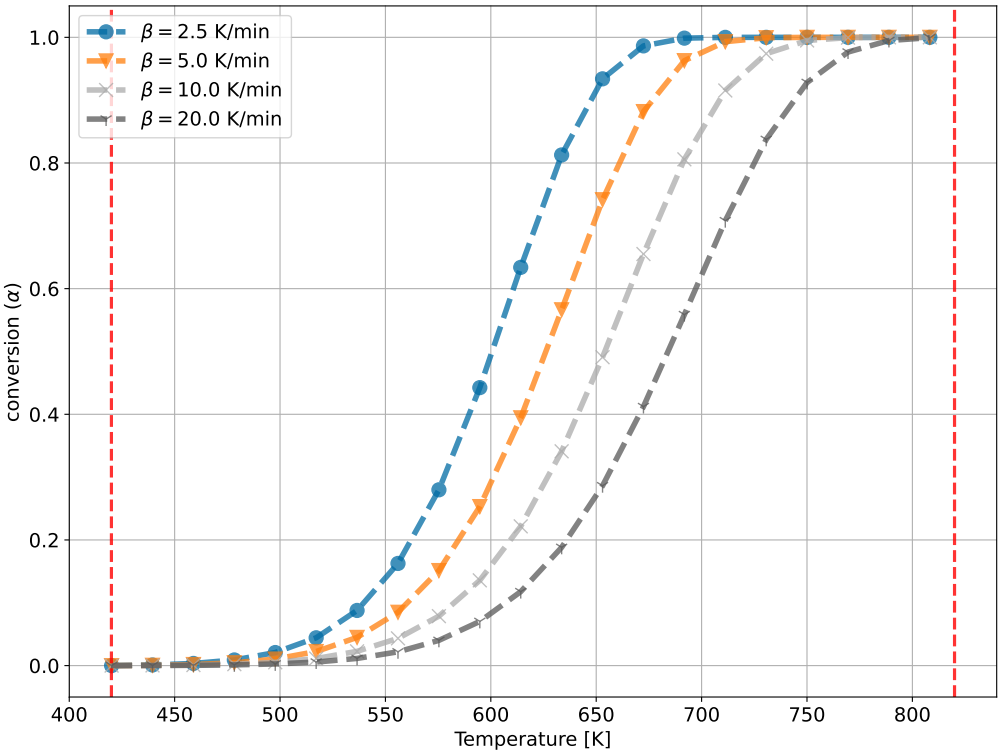


Figure 1.3: Conversion plot in the temperature range suited for analisys.

1.1.4 Isoconversional tables

Now that you have your conversion values computed, the next step is to build the Isoconversional tables with the `Isoconversion` function. This tables arrange the experimental information according to conversion values.

```
Iso_Tables  = xtr.Isoconversion(d_a = 0.005)
```

The parameter `d_a` assigns the difference between conversion values, `d_a` $\equiv \Delta\alpha = \alpha(t+\Delta t) - \alpha(t)$.

`Iso_Tables` is a list containg three isoconversional tables containing temperatures, times and conversion rates, respectively. For n heating programs, the tables contain $n+1$ columns. The first column correspond to the conversion values and is treated as an index for the rows, *i.e.* it has no title. The following n columns correspond to the heating rates of the temperature programs in increasing heating-rate order (see Table 1.1).

Table 1.1: Table of Isoconversional temperatures. The first column with no title corresponds to conversion values which, according to `d_a`, equals 0.005. Only a fraction of the table is presented for clarity.

	HR 2.5 K/min	HR 5.0 K/min	HR 10.0 K/min	HR 20.0 K/min
0.000	420.093500	420.093500	420.093500	420.093500
0.065	527.797200	547.611600	568.954600	591.963400
0.130	548.654500	570.079000	593.183300	618.140700
0.195	562.156200	584.635700	608.905800	635.161700
0.260	572.569800	595.873600	621.059100	648.338000
0.325	581.306200	605.309700	631.274900	659.427200
0.390	589.026700	613.655400	640.318700	669.254400
0.455	596.109300	621.317400	648.629000	678.293000
0.520	602.806400	628.567600	656.499000	686.860500
0.585	609.313700	635.617400	664.157800	695.205000
0.650	615.811700	642.662300	671.817300	703.557300
0.715	622.502700	649.921900	679.716700	712.178300
0.780	629.667200	657.701500	688.189400	721.432700
0.845	637.796400	666.536500	697.820800	731.962200
0.910	648.058200	677.701600	710.006900	745.296800
0.975	665.836700	697.078800	731.195600	768.491000

1.1.5 Activation Energy calculation

The activation energy can be calculated with the functions implemented in the `ActivationEnergy` object. The first thing to do is to create an instance of the object:

```
ace = pnk.ActivationEnergy(B,
                           T0,
                           Iso_Tables)
```

The object must be feeded with results obtained from `DataExtraction`, namely: the heating ramps list (`B`), the initial temperatures list (`T0`) and the isoconversional tables list (`Iso_Tables`).

The activation energy can then be calculated by means of any of the five implemented methods in pICNIK: the Friedman method (`ace.Fr()`), the Kissinger-Akahira-Sunose method (`ace.KAS()`), the Osawa-Flynn-Wall method (`ace.OFW()`), the first Vyzovkin method (`ace.Vy()`) and the advanced Vyazovkin method (`ace.aVy()`). In this example we will work with the advanced Vyazovkin method but every step is equivalent for each isoconversional method. To use other methods just replace `aVy` for the corresponding keyword as mentioned above but without the argument `(5,180)`[2]. For more

```
aVy = ace.aVy((5,180))
ace.Ea_plot(ylim=(50,95))
ace.export_Ea()
```

The results can be visualized with the `Ea_plot` function and exported with the `export_Ea` function.

[2]This argument is necessary for the Vyazovkin methods because they are based on the minimization of a function.

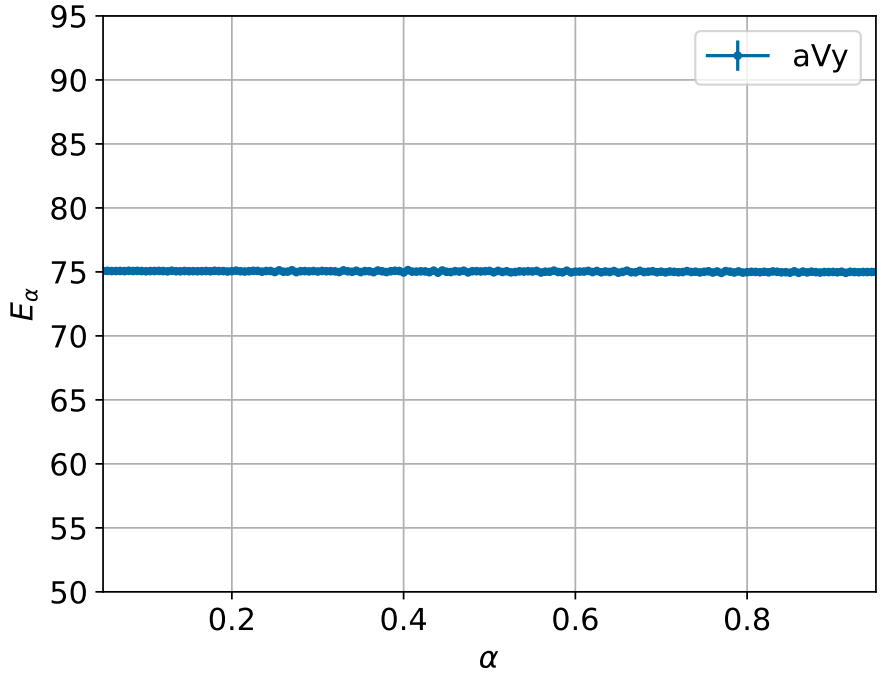


Figure 1.4: Activation energy.

All five methods return a list containing: *i*) conversion values, *ii*) averaged temperature values associated to the conversion values, *iii*) activation energies in [kJ mol^{-1}] and, *iv*) associated error of the activation energies in [kJ mol^{-1}].

1.1.6 Model-free prediction

Once the aactivation energy has been computed it can be used to perform model-free predictions. The next function is based on the integral isoconversional principle: $g(\alpha) = \text{constant}$, which implies an equality between temperature integrals to reach a given conversion[ref]:

$$J[E_\alpha, T(t)_i] = J[E_\alpha, T(t)_j] \tag{1.1}$$

where $T(t)_i$ and $T(t)_j$ are two different temperature programs. This equation is so general that it can be used to make predictions under arbitrary temperature programs, $T(t)$. The more common are: *i*) isothermal, *ii*) linear heating, and *iii*) a mix of the latter two. With pICNIK these cases are treated by the same function: `modelfree_prediction`.

```
# Isothermal with T=575 K
ap,Tp,tp  = ace.modelfree_prediction(aVy[2],
                                     B=0,
                                     isoT =575,
                                     alpha=0.999,
                                     bounds = (10,10))

# Linear heating with ramp of 10 K/min
```

```
ap2,Tp2,tp2 = ace.prediction(aVy[2],
                             B=10,
                             alpha=0.999,
                             bounds = (10,10))

# Linear heating rate of 5 K/min until a temperature of 575 K, then isothermal
def Temp_program(t):
    T = [(25+273 + 5*k) if k <= 53 else 575 for k in t]
    return np.array(T)

ap3,Tp3,tp3 = ace.prediction(aVy[2],
                             B=10,
                             T_func = Temp_program,
                             alpha=0.999,
                             bounds = (10,10))
```

The first parameter must be an array of activation energy values (`aVy[2]`). The parameter `B` assigns a linear heating ramp, `alpha` determines the final conversion value to be reached in the prediction. Finally, the `bounds` parameter is key to make a good prediction. It represents a time interval (in [min]) where to search the time required to reach a given conversion. Thus, the elements of this range should be around the order of magnitude expected for the process. Note that, for the prediction with a mixed temperature program, a function must be defined in a way that it loops over the elements of a time array and evaluate a conditional function depending on the desired temperature program. Also, when `T_func` is assigned, the parameter `B` is ignored but must be written.

The `modelfree_prediction` function returns: the predicted conversion, the predicted temperature and the predicted time. The results of the three predictions are plotted in Figure 1.5.

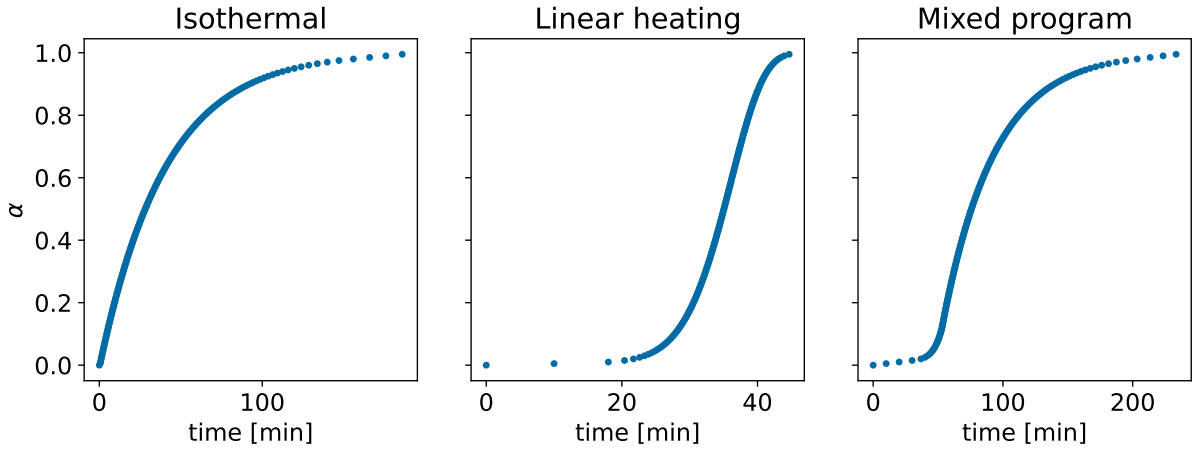


Figure 1.5: Predicted conversion as a function of time for three different temperature programs. Left: Isothermal with $T = 575\,\mathrm{K}$. Center: Linear heating program with $\beta = 5\,\mathrm{K\,min^{-1}}$. Right: Lienar heating with $\beta = 5\,\mathrm{K\,min^{-1}}$ until $T = 575\,\mathrm{K}$, then, isothermal.

To export the predictiions, pICNIK has the `export_prediction` function which takes the predicted time, temperature and conversion along with a file name. Note that the file extension

must be ".csv".

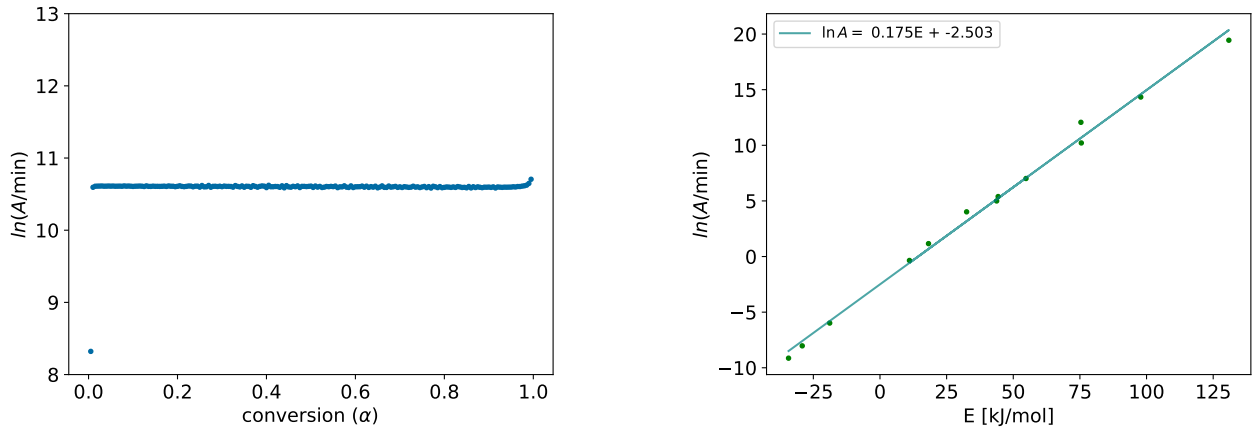
```
export_prediction(tp, Tp, ap, name="isoT_prediction.csv")
export_prediction(tp2, Tp2, ap2, name="linear_prediction.csv")
export_prediction(tp3, Tp3, ap3, name="mixed_prediction.csv")
```

Also, the kinetic triplet, (E_α, A_α, $g(\alpha)$), can be used to get isothermal predictions as will be shown below.

1.1.7 Pre-exponential factor

The pre-exponential factor is computed by means of the so-called compensation effect, which implies a linear relation between the pre-exponential factor and the activation energy: $\ln A = a + bE$

```
ln_A,a, b, Avals, Evals = ace.compensation_effect(aVy[2],B=B[-1])
```



(a) (b)

Figure 1.6: Pre-exponential factor and compensation effect.

A linear regression is computed over a set of E_i, $\ln A_i$ to obtain the parameters a and b. The set of E_i, $\ln A_i$ pairs is obtained from fitting different models $f(\alpha)_i$ to the experimental data.

The `compensation_effect` function takes as parameters the activation energy, (`aVy[2]`) and an experimental heating rate, (`B[-1]`). The function returns five results: *i*) $\ln A_\alpha$ (`ln_A`) corresponding to the conversion values associated to the activation energies (`aVy[0]`); *ii, iii*) The linear parameters a and b (`a, b`); *iv, v*) The set of E_i, $\ln A_i$ pairs (`Avals, Evals`).

Careful attention should be given to the results obtained with this method as it strongly depends on the results of the fitting procedure. In this example, the programmed value for $\ln A_\alpha$ is $12\,\mathrm{min}^{-1}$, nevertheless the function returned a value of $\ln A_\alpha 10.6\,\mathrm{min}^{-1}$ which is a good approximation but may lead to incorrect models as will be shown below. More reliable results can be obtained by fitting over more kinetic models.

1.1.8 Model reconstruction

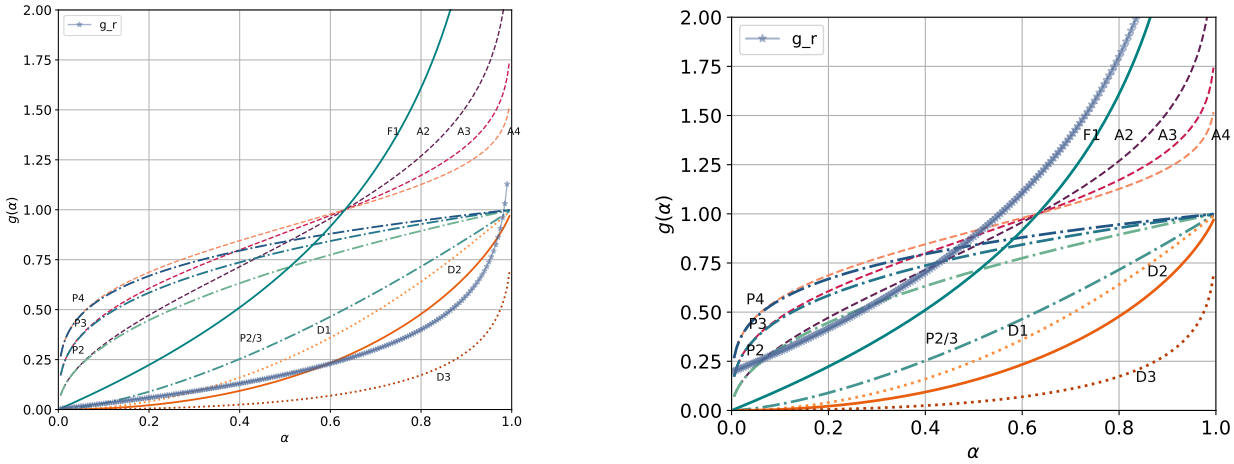
The numerical reconstruction of the reaction model is carried on in its integral form, $g(\alpha)$ Given an array of activation energy, E, and an array of pre-exponential factor, the integral reaction model can be computed as:

$$g(\alpha) = \sum_i g(\alpha_i) = \sum_i A_{\alpha_i} \int_{t_{\alpha_{i-1}}}^{t_{\alpha_i}} \exp(-\frac{E_{\alpha_i}}{RT(t_{\alpha_i})})dt \tag{1.2}$$

```
g_r  = ace.reconstruction(aVy[2],
                          np.exp(ln_A),
                          B[0])
```

The function `reconstruction` requires three paremeters, an activation energy array (`aVy[2]`), a pre-exponential factor array (`np.exp(ln_A)`)[3]

The result is an array containing the values of $g(\alpha)$ [(]also corresponding to the conversion values associated to the activation energies (`aVy[0]`)] and a graphical comparison of the computed $g(\alpha)$ against a series of models implemented in pICNIK (see Fig. 1.7(a))). Note the mentioned result missleading to a diffusion model when the simulated data correspond toa F1 model. When the value for the pre-exponential factor is replaced by the programmed one (exp(12)) the computed model agrees better with the programmed one, Fig. 1.7(b).



(a) (b)

Figure 1.7: Numerically reconstructed integral model $g(\alpha)$. Left, computed with $\ln(A_\alpha) = 10.6$. Right, computed with $\ln(A_\alpha) = 12$. Both were computed with `aVy[2]`.

Once you have the kinetic triplet it can be exported with the `export_kinetic_triplet` which takes the triplet as parameters:

```
ace.export_kinetic_triplet(aVy[2],
                           ln_A,
                           g_a=g_r)
```

[3]This is equivalent to $\exp[\ln(A_\alpha)] = A_\alpha$.

1.1.9 Model based prediction

To compute isothermal predictioins based on the kinetic triplet use the `t_isothermal` function. This function is based on the relation:

$$t_{\alpha_i} = \frac{\sum_i g(\alpha_i)}{A \exp\left(-\frac{E}{RT_0}\right)} \tag{1.3}$$

The function takes as parameters: an activation energy array, the logarithmic pre-exponential factor array, the temperature, an index to pick a heating rate, the integral model array and a conversion array.

```
tim_pred  = ace.t_isothermal(aVy[2],
                             ln_A,
                             575,
                             col=0,
                             g_a=g_r,
                             alpha=aVy[0])
```

As results, the function returns an array of times required to reach the conversion values given in `alpha`, Fig. 1.8.

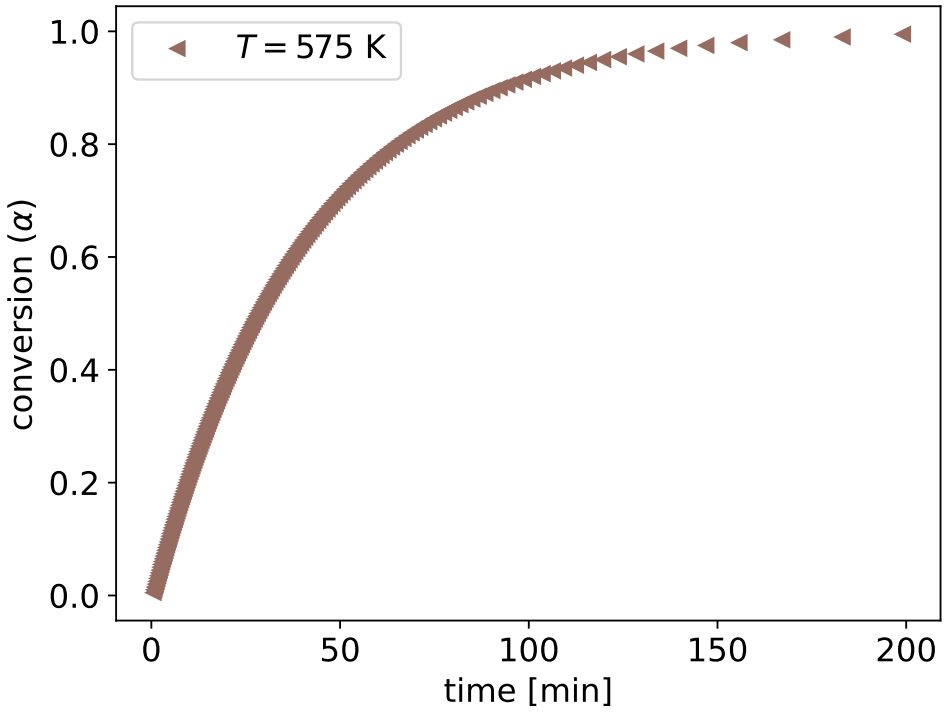


Figure 1.8: Isothermal model-based prediction. $T = 575\,\mathrm{K}$

Finally, this prediction can be exported with the `export_prediction` function mentioned above.

```
ace.export_prediction(tim_pred,
                      575,
                      aVy[0],
                      isothermal=True,
                      name='model_based_prediction.csv)
```

Note that in this case the temperature parameter is the isothermal value and the keyword `isothermal` has to be added with the Boolean value `True`.