PyDSC Handbook

Aline Cisse[*], Judith Peters[†], Leonardo Chiappisi[‡]

4 March 2020, pyDSC version 0.5.1

Contents

[*]Laboratoire Interdisciplinaire de Physique, Institut Laue-Langevin, Univ. Grenoble-Alpes, France, cissea@ill.fr

[†]Laboratoire Interdisciplinaire de Physique, Institut Laue-Langevin, Univ. Grenoble-Alpes, France, jpeters@ill.fr

[‡]Institut Laue-Langevin, Stranski Laboratorium für Physikalische und Theoretische Chemie, Institut für Chemie, Technische Universität Berlin, Germany, leonardo.chiappisi@tu-berlin.de

1 Comments on the DSC Experiment

To ensure an optimal data correction and use of the script, the following procedure was applied during the experiment (see a summary Fig. 1).

- First of all, for the standard sample/buffer runs, the mass of solution in each DSC cell (sample and reference) was measured. From this information and the known concentration of our sample, the following masses should be known: in the sample cell; sample and buffer mass, in the reference cell; buffer mass. Attention must be paid to minimize the difference of mass between the buffer of each cell, $\Delta m_{BS} \sim 0$.
- Then, buffer/buffer runs were conducted. Same mass measurements as previously were applied, so the buffer mass is known in each cell. However, in that case, a mass difference of around 10% of the total volume must be ensured to have a clear signal, $\Delta m_{BB} \sim 0.1 m_S$.
- Finally empty cell/empty cell runs were performed. To have the most rigorous corrections possible, the empty cells should be the same all along these three runs, and each of them kept as sample or reference definition during all experiments.

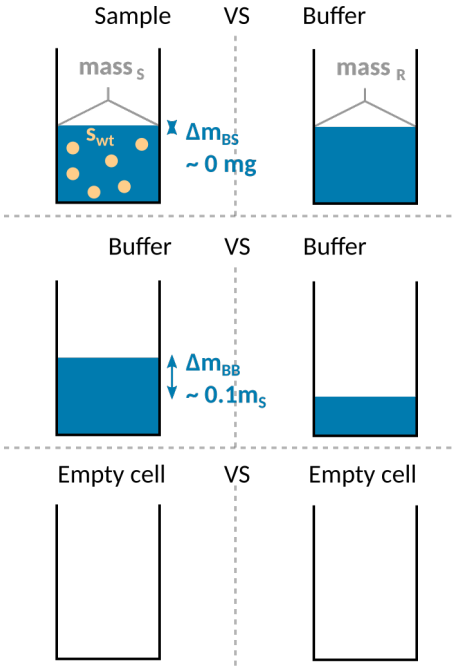

Figure 1: Scheme of the different runs in a DSC experiment. mass_S refers to the mass of the solution in the sample cell, mass_R to the mass of the buffer in the reference cell. s_{wt} is the concentration of the sample in weight percent. Δm_{BS} and Δm_{BB} are the difference of buffer mass between the sample and the reference cell, respectively in sample/buffer runs and buffer/buffer runs.

2 PyDSC

2.1 Theory

The script is written in Python 3, it makes use of the SciPy Package[1] and is platform independent. Several operations on the data are performed, provided the required information are available. The following operations are executed on the raw data files:

1. **Data averaging and resizing:** The DSC raw thermograms of the sample and, if provided, of the empty cell and buffer-buffer runs are read by the program and the points which fall outside the temperature region of interest are discarded. The reference measurements are averaged. If requested by the user, in order to facilitate any further data treatment, the number of points is reduced by a factor N, defined by the user.

2. **Heat flow conversion in heat capacity:** The heat flow is converted into a heat capacity upon division by the heat rate. The heat rate is either provided by the user, or it is determined from the raw data file if also the time information is contained. The raw data are finally plotted.

3. **Empty cell and buffer correction:** If the data file corresponding to the empty cell/empty cell and/or buffer/buffer runs are provided, the raw data are corrected for both contributions. In particular, the empty cell/empty cell run is used to take into account any non-constant contribution arising from the empty cells or the instrument. The buffer/buffer run is used to take into account any contribution arising from a mismatch in amount of buffer contained in the reference and in the sample cell. In detail, the apparent heat capacity of the sample run Cp_s, is corrected for the apparent heat capacity of the empty cell Cp_{EC} and for the apparent heat capacity of the buffer/buffer run Cp_{bb} as:

$$\overline{Cp_s} = Cp_s - Cp_{EC} - \frac{\Delta m_{BS}(Cp_{bb} - Cp_{EC})}{\Delta m_{BB}} \tag{1}$$

   with Δm_{BS} and Δm_{BB} being the mass difference between the buffer contained in the sample and in the reference in the sample run, and between the buffers in the buffer/buffer run. If no empty cell/empty cell or buffer/buffer runs are provided, this correction will not be performed and the non-corrected sample run will be further analysed. In most of the cases, Δm_{BS} should be so small that the last term might be negligible.

4. **Normalisation of apparent heat flow:** The apparent heat flow of the sample run is finally normalized by the amount of solute. In particular, the apparent heat capacity is normalized either by the mass of solute, resulting in a Cp_s given in J K^{-1} g^{-1}, or by the moles of solute, if the molecular weight is known, thus resulting in a Cp_s given in J K^{-1} mol^{-1}.

5. **Baseline correction:** The core task carried out by the program is to compute a physically meaningful baseline. The baseline is constructed following an iterative procedure proposed elsewhere[2, 3]. In detail, the baseline $CP_{bl}(T)$ is defined as follows:

$$CP_{bl}(T) = (1-\alpha)CP_{bl}^{pre}(T) + \alpha \cdot CP_{bl}^{post}(T) \tag{2}$$

   with $CP_{bl}^{pre}(T)$ and $CP_{bl}^{post}(T)$ are the baselines determined in the regions at lower and higher temperatures than the peak, respectively. α is the degree of convertion, defined as:

$$\alpha(T) = \frac{\int_0^T CP_s(T) - CP_{bl}(T)\mathrm{d}T}{\int_0^\infty CP_s(T) - CP_{bl}(T)\mathrm{d}T} \tag{3}$$

where $CP_s(T)$ is the apparent heat capacity difference between sample and reference, after normalization and eventual correction. $CP_{bl}^{pre}(T)$ and $CP_{bl}^{post}(T)$ represent the temperature dependent difference of heat capacity of sample and reference, eventually corrected by the contribution from the empty cells and solvent mismatch. They are assumed to exhibit a linear dependence from temperature.

At this stage of the program, the enthalpy change of the process, computed as $\Delta H = \int_0^\infty CP_s(T) - CP_{bl}(T)\,dT$, is evaluated and its uncertainty is estimated. The uncertainty in ΔH is estimated from the stochastic fluctuations of the signal in the regions where $CP_{bl}^{pre}(T)$ and $CP_{bl}^{post}(T)$ are determined. By definition, in these regions no processes take place, and the signal fluctuates around an apparent heat capacity of zero J K^{-1}, which is used to determine the standard deviation of the heat capacity. The heat exchanged during the examined process is computed by integration using the trapezoidal rule, and its uncertainty by standard error propagation.

2.2 General use of the script

1. Save your thermograms in the associated folder `rawdata`.
2. Open `dsc_input.py` (Spyder editor or other any code editor you might use) and fill it with the names of the raw datafiles to analyze, as well as with parameters according to your experiment (see the part 2.3 for more details).
3. Check that an `Output` folder is already created (it will be necessary to have this folder, so the script can save the results in it, or else it will return an error).
4. Run the script following one of the following procedures (with Anaconda - Spyder or from a terminal).

2.2.1 From Anaconda - Spyder

5. Install Anaconda or Miniconda (see webpage[1] for the installation). The different libraries (numpy, scipy) and the editor Spyder will be automatically installed.
6. Run Spyder and open `pyDSC.py`
7. Run the file (play button in the top or press F5)

2.2.2 From a terminal

5. Install Python 3 and the associated libraries in your computer (or check if already done)
6. Open a terminal and go to the folder of interest with `cd` command
7. Write : `python3 pyDSC.py`
8. The script will be run, messages concerning the advancement will be displayed in the terminal

The output files will be saved in the `Output` folder and pdf files will be created for the graphs (see the part 2.4 for more details).

[1] https://www.anaconda.com/distribution/

2.3 Input file : `dsc_input.py`

A *python* dictionary to fill with the names of the rawdata files, for heating and cooling cycles, for each type of run, and with the parameters of the experiment. Details written here are also given within the file in the comments, and examples are given as a model.

`Folder` : relative path to pyDSC.py where datafiles are stored.

- `Heating_runs` : *python* list containing the filenames of the ASCII files with raw heating data.
- `Cooling_runs` : *python* list containing the filenames of the ASCII files with raw cooling data.
- `Empty_cell_heat_runs` : *python* list containing the filenames of the ASCII files with raw heating data of the empty cell measurement.
- `Empty_cell_cool_runs` : *python* list containing the filenames of the ASCII files with raw cooling data of the empty cell measurement.
- `Buffer_heat_runs` : *python* list containing the filenames of the ASCII files with raw heating data of the buffer measurement.
- `Buffer_cool_runs` : *python* list containing the filenames of the ASCII files with raw cooling data of the buffer measurement.
  Leave empty the latter lists if no corresponding measurement was performed.

`Dataformat` : refers to the format in which the dara are saved. It depends on which type of instrument is used. The options are:

`Setaram3`: raw ascii data which, in addition to the header of arbitrary length, contain three columns: Time(s), Furnace Temperature (°C or K), and HeatFlow (mW). The script will detect the «Furnace» word in the header to determine the first line of data. Example data are:

```
Sample_name
Creation Date : 10/16/2018 4:33:29 PM
User : PSCM

HeatFlow :
Initial Mass : 456 mg
Molar Mass : N/A

Time (s)   Furnace Temperature (°C) HeatFlow (mW)
0          4.843128                 -0.006023
4          4.82847                  -0.005714
8          4.691505                 -0.004915
...
4792       85.045596                -0.522189
4796       85.114593                -0.520797
4800       85.183724                -0.521634
```

`Setaram3temptime`: raw ascii data, which, in addition to the header of arbitrary length, contain three columns: Furnace Temperature (°C or K), Time(s), and HeatFlow (mW).

The script will detect the «Furnace» word in the header to determine the first line of data. Example data are:

```
Sample_name
Creation Date : 10/16/2018 4:33:29 PM
User : PSCM

HeatFlow :
Initial Mass : 456 mg
Molar Mass : N/A

Furnace Temperature (°C)    Time (s)    HeatFlow (mW)
4.843128                    0           -0.006023
4.82847                     4           -0.005714
4.691505                    8           -0.004915
...
85.045596                   4792        -0.522189
85.114593                   4796        -0.520797
85.183724                   4800        -0.521634
```

Setaram4: raw ascii data, which in addition to the header of arbitrary length, contain four columns: Index, Time(s), Furnace Temperature (°C or K), and HeatFlow (mW). The script will detect the «Furnace» word in the header to determine the first line of data. Example data are:

```
Sample_name
Creation Date : 10/16/2018 4:33:29 PM
User : PSCM

HeatFlow :
Initial Mass : 456 mg
Molar Mass : N/A

Index Time (s)   Furnace Temperature (°C) HeatFlow (mW)
1     0          4.843128                 -0.006023
2     4          4.82847                  -0.005714
3     8          4.691505                 -0.004915
...
1199  4792       85.045596                -0.522189
1200  4796       85.114593                -0.520797
1201  4800       85.183724                -0.521634
```

3cols: raw ascii data with a **one-line header** which contain the data in a three column structure: Time(s), Furnace Temperature (°C or K), and HeatFlow (mW). In that case the script will consider by default a fixed header of one line, and will not look for «Furnace» word. It avoids error messages in the import process if you do not have a «Furnace» word in your header. Example data are:

```
Time (s)    Temperature (°C)    HeatFlow (mW)
0           4.843128          -0.006023
4           4.82847           -0.005714
8           4.691505          -0.004915
...
4792        85.045596         -0.522189
4796        85.114593         -0.520797
4800        85.183724         -0.521634
```

3cols_variable_header: raw ascii data with a header of variable length which contain the data in a three column structure: Time(s), Temperature (°C or K), and HeatFlow (mW). The length of the header has to be specified in the «Header_length» parameter. Example data are:

```
Run 626
Instrument 2920 DSC       V2.6A
Module DSC Dual Sample Cell - Sample A
Sample wax
Size 2.85000 mg
    .
    .
    .
Cell# 116
Nsig 3
Sig1 Time (s)
Sig2 Temperature (øC)
Sig3 Heat Flow (mW)

0.0000000 -31.16051 -9.3378E-3
0.02833333 -31.15951 -.01090668
0.06166668 -31.16015 -.01214525
...
113.9295 24.13363 -.09980994
114.2628 24.13402 -.09811464
```

3cols_variable_header_temp_power_time: raw ascii data with a header of variable length which contain the data in a three column structure: Temperature (°C or K), HeatFlow (mW), and Time(s). The length of the header has to be specified in the «Header_length» parameter. Example data are:

```
Run 626
Instrument 2920 DSC       V2.6A
Module DSC Dual Sample Cell - Sample A
Sample wax
Size 2.85000 mg
    .
    .
    .
Cell# 116
Nsig 3
```

```
Sig1 Temperature (øC)
Sig2 Heat Flow (mW)
Sig3 Time (s)

-31.16051 -9.3378E-3 0.0000000
-31.15951 -.01090668 0.02833333
-31.16015 -.01214525 0.06166668
...
24.13363 -.09980994 113.9295
24.13402 -.09811464 114.2628
```

4cols_variable_header: raw ascii data with a header of variable length which contain the data in a four column structure: Index, Time(s), Temperature (°C or K), and HeatFlow (mW). The length of the header has to be specified in the «Header_length» parameter. Example data are:

```
Run 122
Instrument 2920 DSC       V2.6A
Module DSC Dual Sample Cell - Sample A
Sample PEG
Size 12.2340 mg
    .
    .
    .
Cell# 10
Nsig 4
Sig1    Index
Sig2 Time (min)
Sig3 Temperature (øC)
Sig4 Heat Flow (mW)

1   0.0000000 -31.16051 -9.3378E-3
2   0.02833333 -31.15951 -.01090668
3   0.06166668 -31.16015 -.01214525
...
3452    113.9295 24.13363 -.09980994
3453    114.2628 24.13402 -.09811464
```

TA_temp_power_time: raw ascii data with a header of length 1 which contain the data in a seven column structure: Temperature (°C), HeatFlow (mW), Time (s), Pressure (A), Scan Rate (°C/min), AnalysisData and CorrectedData. Example data are:

```
    scan 110/28/20203:56:02 PM 2
temp(øC) power time(seconds) pressure(A) scan rate(øC/min) AnalysisData CorrectedDa
4.02366864421327 -28.5071680896326 95114 0.0103828310966492 0.00105563418502541
526.970414686355 526.970414686355
4.02366390568482 -28.3597811366128 95115 0.0103843212127686 0.00105563418502541
527.072004812195 527.072004812195
4.02366901295593 -28.1632642654223 95116 0.0103843212127686 0.00105563418502541
527.317882196839 527.317882196839
    .
```

.
.

In all these cases, we assume that **the columns are separated by space or tab space**. Either your instrument is directly giving you .txt or .dat files (among others) in one of the Setaram format, either you need to format them by hand (for example, from an excel file, copy and paste the three columns of interest in the right order with a one-line header in an empty .txt or .dat file, and use `3cols` option).

- `Header_length` : length of the file header, needed only for the dataformat which need to have the header length to be specified.

`mass_s` : refers to the mass of the solution in the sample cell (see $mass_S$ in Fig. 1), **in mg**.

`mass_r` : refers to the mass of the buffer in the reference cell (see $mass_R$ in Fig. 1), **in mg**.

`mass_bb` : refers to the difference of buffer mass Δm_{BB} in the buffer/buffer runs between the sample and reference cell (see Fig. 1), **in mg**.

`s_wt` : is the concentration of the sample in the sample/buffer run (see s_{wt} in Fig. 1). It is expressed **in weight percent**, so for example 1% wt will be 0.01, 100% wt will be 1.0.

`Mw` : is the molecular weight of the sample **in g/mol**. It is optional.

`ROI_h` : is the temperature region of interest for the heating runs. Fill with : Initial temperature, Final temperature, **in °C**. The data points from heating scans falling outside this area will be discarded.

`ROI_c` : is the temperature region of interest for the cooling runs. Fill with : Initial temperature, Final temperature, **in °C**. Can be different from the heating runs. The data points from cooling scans falling outside this area will be discarded.

`ROP_h` : is the temperature region where the peak is found in the heating scans,. Outside this region, linear fits will be performed in order to evaluate the theoretical baseline. Before this region, it will be $CP_{bl}^{pre}(T)$, and after this region, $CP_{bl}^{post}(T)$. Fill with `Initial temperature, Final temperature` **in °C**.

`ROP_c` : is the temperature region where the peak is found in the cooling scans. Outside this region, linear fits will be performed in order to evaluate the theoretical baseline. Before this region, it will be $CP_{bl}^{pre}(T)$, and after this region, $CP_{bl}^{post}(T)$. Fill with `Initial temperature, Final temperature` **in °C**.

`Scanrate_h` : is the scan rate of the heating runs **in K/min**. If the data contain the time and temperature information, this line will be ignored and an average scan rate will be calculated from the time and temperature data.

`Scanrate_c` : is the scan rate of the cooling runs **in K/min**. If the data contain the time and temperature information, this line will be ignored and an average scan rate will be calculated from the time and temperature data.

`bins` : represents the size of bins used to reduce the file size, i.e., a file of length N is reduced to N/bins, whereby bins number of points are averaged.

`Input` : Convention used for input files, can be exo-up or exo-down.

`Output` : Convention used for input files, can be exo-up or exo-down.

`Exo_in_plot` : True or False, defines weather an arraw, indicating convention exo-up or exo-down are drawn on the graphs.

`unit_time` : unit in which the time is given, can be either min or s.

`unit_temp` : unit in which the temperature is given, can be K or degC

`unit_power` : unit in which the heatflow is given, can be uW (microWatt), mW (milliWatt), or W (Watt)

2.4 Output files

2.4.1 pdf files

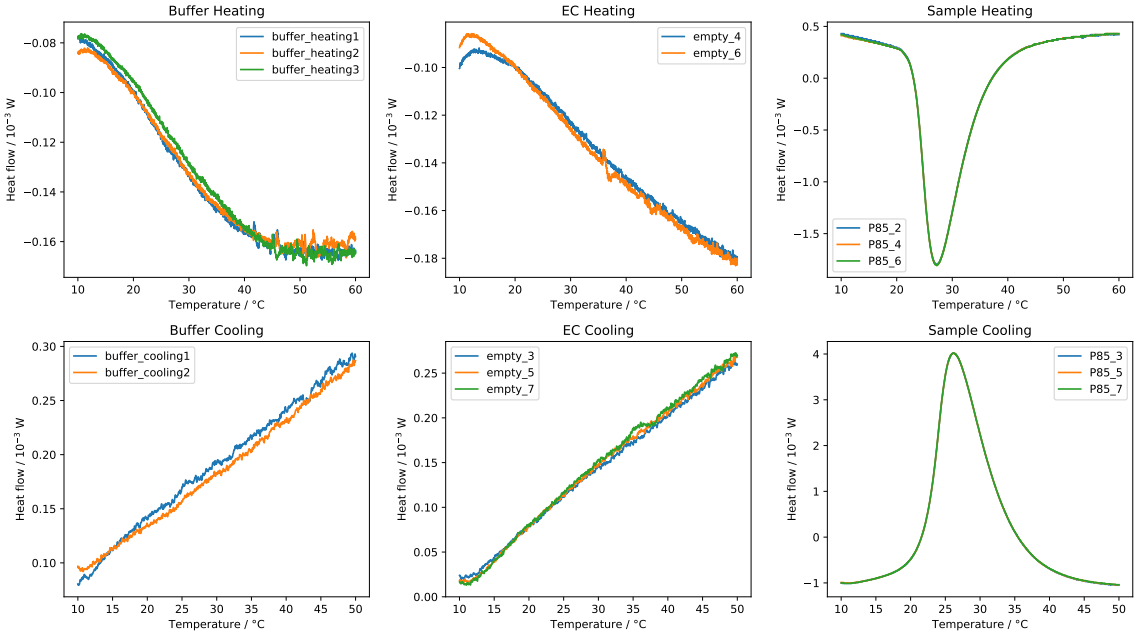

Figure 2: `Rawdata.pdf` shows the rawdata given by the instrument.

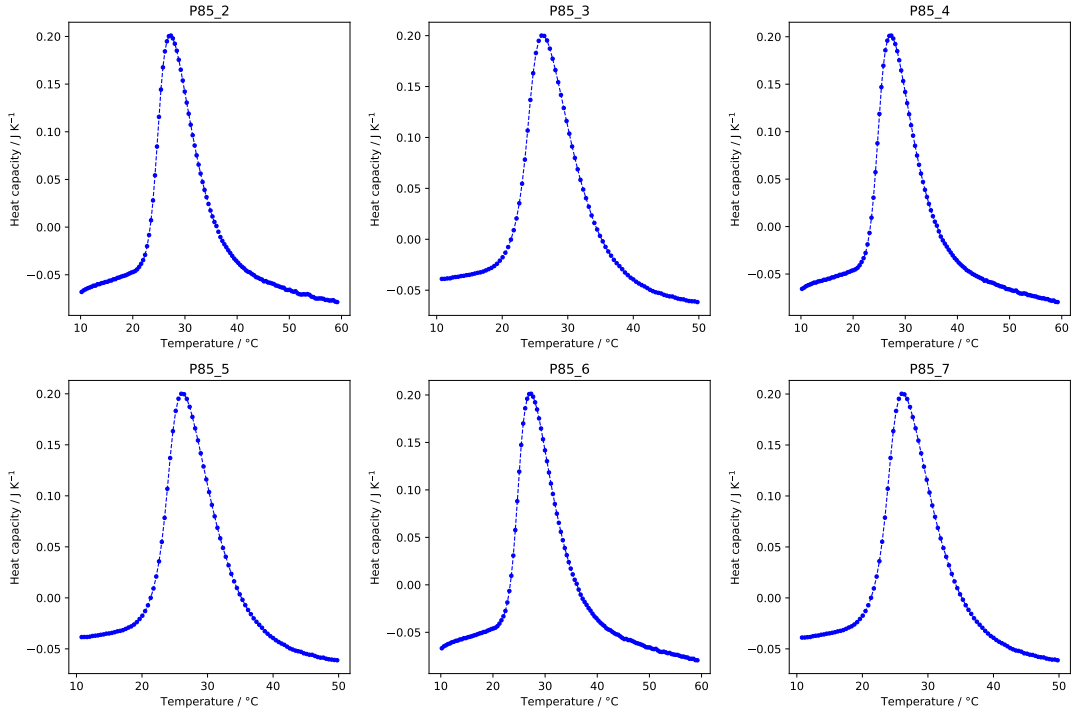

Figure 3: `Corrected_data.pdf` shows the data corrected by buffer and empty cell, and normalized by the scan rate (but not normalized yet by the sample mass). These curves help to visualizing the data points, and checking the binning.

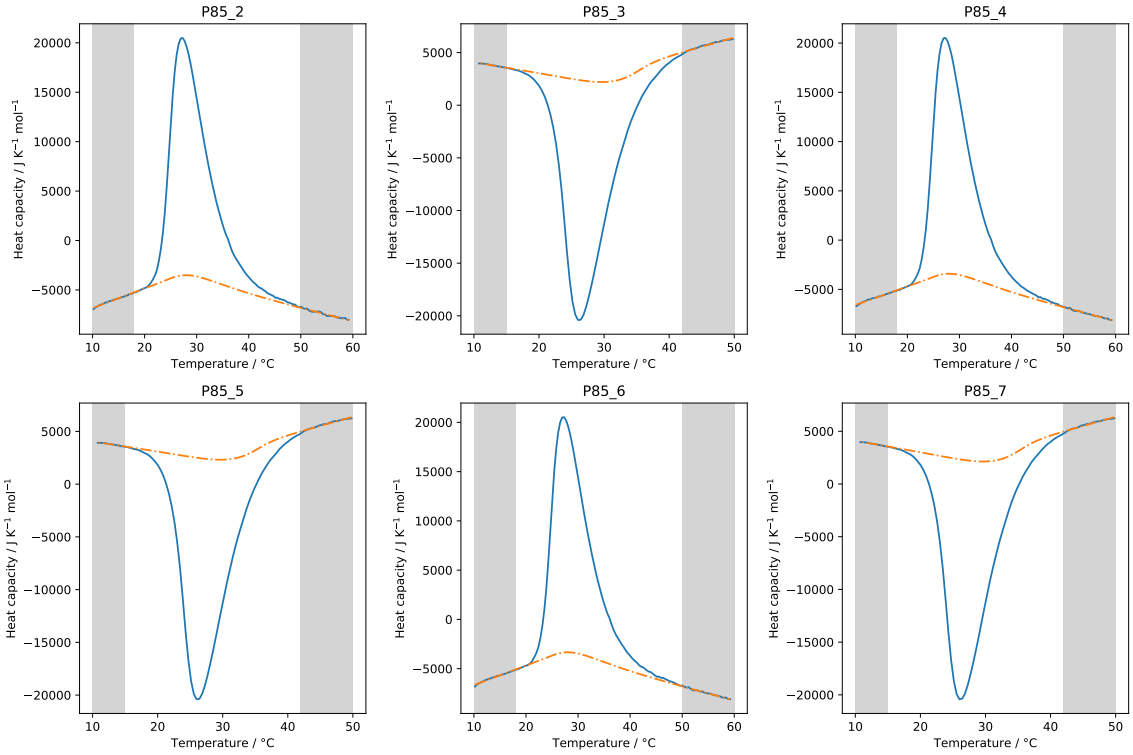

Figure 4: `Baseline_data.pdf` shows the corrected data, along with the baseline in orange dotted line, constructed following Eq. (2). The grey areas correspond to the respective range of pre-peak and post-peak linear fits used in Eq. (2). If there is no clear peak to create a baseline, like in the figure in the right, put a very small `ROP` (in `Input_params.txt`, for example here we chose «`69.9, 70.0`» for `ROP_c`), and the script will not try to create a baseline.

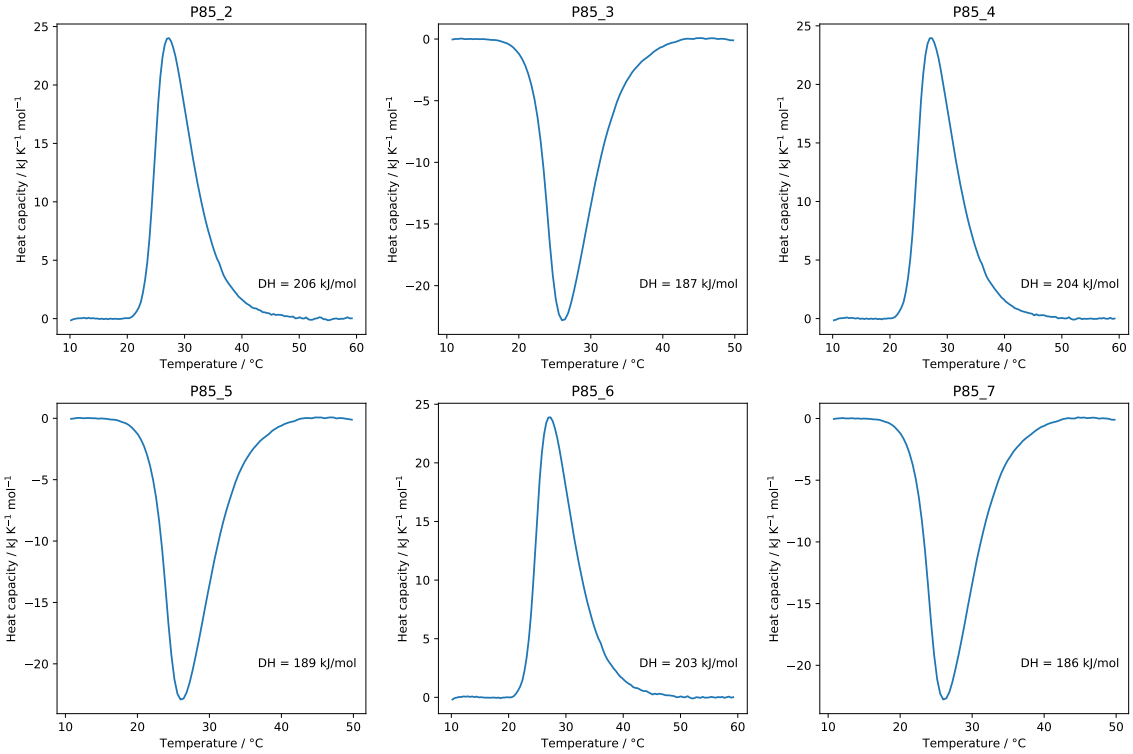

Figure 5: `Final_data.pdf` shows the thermograms after correction and after subtraction of the baseline. It is the data used for calculating the excess enthalpy ΔH, displayed in the figure.

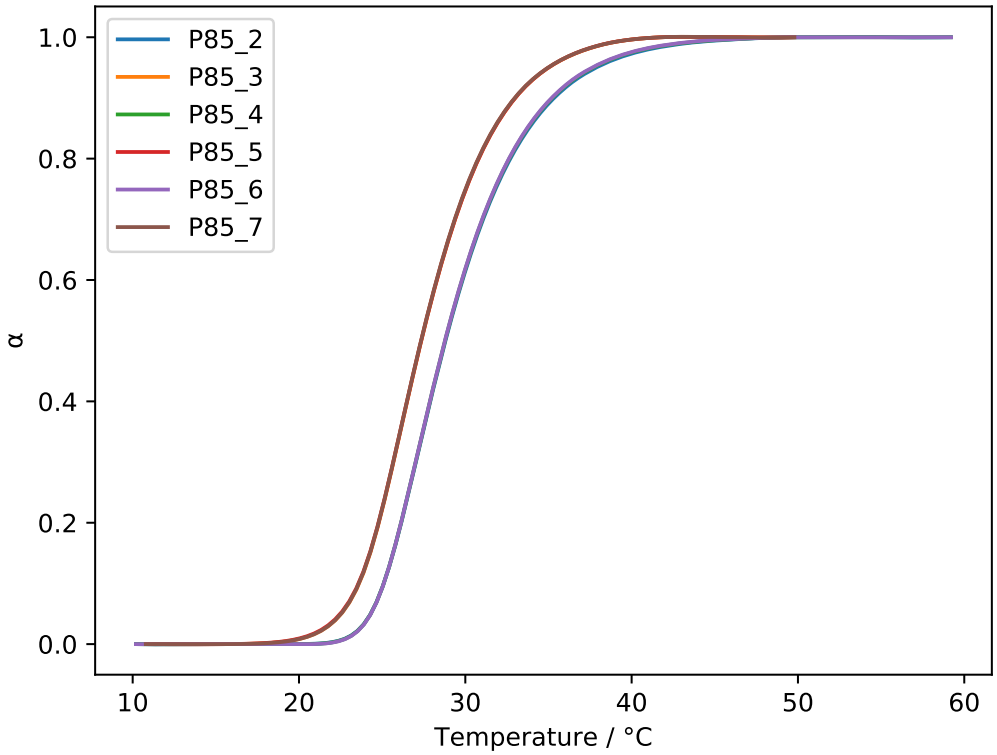

Figure 6: `alpha.pdf` shows the degree of conversion determined for each sample run, as a function of temperature.

2.4.2 `txt` files in the `Output` folder

For each experiment, the script will create a `txt` file displaying a header, explaining how the data were corrected and displaying thermodynamic parameters ΔH, T_m and ΔC_p, as well as five columns :

1. Temperature column `Temp/ [degC]`, in °C.
2. The final heat capacity `CP-baseline / [J/K/g]` or `CP-baseline / [J/K/mol]`, corresponding to the corrected data subtracted by the baseline (the ones displayed in Fig. 6), in J/K/g or in J/K/mol if the molecular weight is given in the input files.
3. The corrected heat capacity `CP [J/K/g]` or `CP [J/K/mol]`, in J/K/g or J/K/mol (these data are shown in `Baseline_data.pdf` displayed in Fig. 4).
4. The final baseline `baseline [J/K/g]` or `baseline [J/K/mol]`, in J/K/g or J/K/mol, constructed in an iterative way, following Eq. (2), and displayed in `Final_data.pdf`, shown in Fig. 4.
5. The enthalpy `H [J/g]` or `H [J/mol]`, in J/g or J/mol, calculated as the integral over temperature of the final heat capacity (column 2): $\mathrm{H} = \int_0^T [CP_s(T) - CP_{bl}(T)]\mathrm{d}T$.

If a molecular weight is added in the input file, it will display these data in J/mol.

3 Some functions of PyDSC (defined in `DSC1.py`)

3.1 `correction`

Corrects for empty cell VS empty cell, and buffer VS buffer runs following Eq. (1). More specifically :

1. It takes the average of all the scans performed for empty cell VS empty cell, and buffer VS buffer ;
2. Then it does an interpolation of this average ;
3. The correction as written in Eq. (1) is applied, taking the interpolations for empty cell and buffer contribution.

3.2 `normalize_sampleruns`

Normalizes the corrected heat flux by the sample mass (or eventually the molar mass if given in the input files), and the scan rate.

3.3 `baseline`

Creates the baseline in an iterative way as written in Eq. (2), calculates the enthalpy H, excess enthalpy ΔH and heat capacity difference ΔC_p (defined as the difference at transition temperature).

4 Changelog

2019.02.05 corrected bug in dsc.plot (wrong value for enthalpy displayed) and reads Mw from input file.

2019.02.19 dsc.correction function was added.

2019.05.02 correction of small bugs in the dsc.correction function and addition of the plot_baseline_data function.

2019.05.29 Included input dataformat 3cols and evaluated error in DH from data noise. Correction small bugs.

2019.05.30 Small bug correction.

2019.06.05 Added latin1 encoding in the setaram data formats

2019.06.07 Small bug corrections.

2019.06.25 Several information are now included in the output files. Program takes care of exo-up or exo-down convention. Version number and date will be added to the exported files.

2019.10.31 In plot.plot_raw_data the raw heatflow is not plotted.

2019.11.06 Small bugs corrected.

2019.11.13 Added the function plot_alpha, which plots the degree of conversion of all sample runs. Changed degC to °C in plots.

2020.01.08 Corrected bug in DSCplot and added encoding option to the input_params file.

2020.02.27 Added Exo-up or Exo-down arrow in the DSC Plots

2020.03.03 Included TA_temp_power_time data format

2020.03.04 Script automatically tries to read the datafiles using different encodings. The encoding parameter is not required anymore in the input_params file. Small bugs corrected.

2021.01.20 Fix of small bugs in the correction module of DSC1.py (removal of [0] after the params["]). Update of the handbook with other dataformat, and how to fill `dsc_input.py` with the parameters of its own experiment.

References

[1] Eric Jones, Travis Oliphant, and Pearu Paterson. SciPy: Open source scientific tools for Python.

[2] G Van der Plaats. A theoretical evaluation of a heat-flow differential scanning calorimeter. Thermochim. Acta, 12:77–82, 1984.

[3] D V Malakhov and M. K. Abou Khatwa. Constructing a self-consistent integral baseline by using cubic splines. J. Therm. Anal. Calorim., 87(2):595–599, feb 2007.