Dye/DNA Plates Documentation

Release v0.7

Robert F. DeJaco

Aug 28, 2022

CONTENTS:

CHAPTER

ONE

DYE / DNA PLATES

The purpose of this code is to enable reproduction and facilitate extension of the computational results associated with following work

> DeJaco, R. F.; Majikes, J. M.; Liddle, J. A.; Kearsley, A. J. Temperature-dependent Thermodynamic and Photophysical Properties of SYTO-13 Dye Bound to DNA.

The dependencies required can be installed via

```
>>> pip install -r requirements.txt
```

The paper can be reproduced with

```
>>> pytest .
```

in this directory. Alternatively, input the commands used in *Reproduction of Manuscript* in a file.

The raw data can be found in the directory *data/*

The documentation can be found in the doc/ directory. A pdf version of the documentation is available at doc/manual.pdf

CHAPTER

TWO

REPRODUCTION OF MANUSCRIPT

2.1 Raw Data and Scaling

First, we read-in the data associated with each data set (see Table 1 in paper) and store as a *src.get_data.RawData* class.

```
>>> import sys, os; sys.path.append(os.getcwd())
>>> from src.get_data import RawData
```

The data for each replicate plate possessing single-stranded DNA with $D_k = 1\times10^{-6}$ mol/L is input into the following structures

```
>>> SS_A_1 = RawData(fluorescence_file_name="ssDNA_2-23-2021.xls", D_k=1e-6, t="SS",␣
↪l="A")
>>> SS_B_1 = RawData(fluorescence_file_name="8-2-2021_GC_ssDNA.xls", D_k=1e-6, t="SS",
↪ l="B")
>>> SS_C_1 = RawData(fluorescence_file_name="1xSS_GC_11-3-2021_data.xls", D_k=1e-6, t=
↪"SS", l="C")
```

The data for each replicate plate possessing single-stranded DNA with $D_k = 2\times10^{-6}$ mol/L is input into the following structures

```
>>> SS_A_2 = RawData(fluorescence_file_name="2x_ssDNA_2-24-2021.xls", D_k=2e-6, t="SS
↪", l="A")
>>> SS_B_2 = RawData(fluorescence_file_name="gc_2xssDNA_11-2-2021_data.xls", D_k=2e-6,
↪ t="SS", l="B")
```

The data for each replicate plate possessing double-stranded DNA with $D_k = 1 \times 10^{-6}$ mol/L is input into the following structures

```
>>> DS_A_1 = RawData(fluorescence_file_name="GC_0p5_dsDNA_11-2-2021.xls", D_k=1e-6, t=
↪"DS", l="A")
>>> DS_B_1 = RawData(fluorescence_file_name="gc_dsDNA_1uM_12-10-2021_data.xls", D_
↪k=1e-6, t="DS", l="B")
```

The data for each replicate plate possessing double-stranded DNA with $D_k = 2 \times 10^{-6}$ mol/L is input into the following structures

```
>>> DS_A_2 = RawData(fluorescence_file_name="8-2-2021_GCdsDNA.xls", D_k=2e-6, t="DS",␣
↪l="A")
>>> DS_B_2 = RawData(fluorescence_file_name="gc_dsDNA_2uM_12-9-2021_data.xls", D_k=2e-
↪6, t="DS", l="B")
```

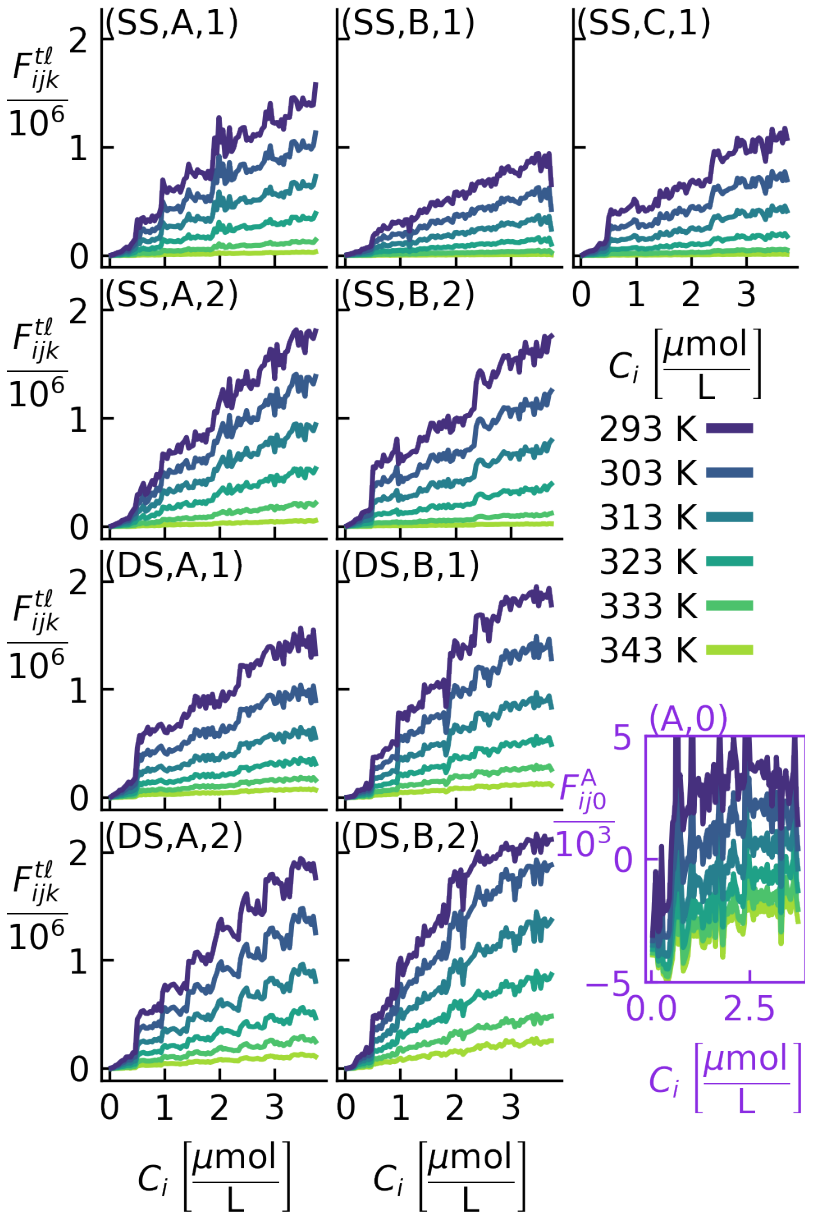
The data for the plate without DNA is input into the following structure

```
>>> A_1 = RawData(fluorescence_file_name="dyeOnly_11-6-2021_data.xls", D_k=0., t="None
↪", l="A")
```

Having read-in the raw data, we plot it via

```
>>> from src.plot_raw_data import make_figure_2
>>> make_figure_2(SS_A_1, SS_B_1, SS_C_1, SS_A_2, SS_B_2, DS_A_1, DS_B_1, DS_A_2, DS_
↪B_2, A_1)
```

which looks like



We combine the replicate plates by storing them as a *src.get_data.CombinedData* class.

```
>>> from src.get_data import CombinedData
```

The data for single-stranded DNA at $\mathbf{D} = 1$ is

```
>>> SS_1 = CombinedData(SS_A_1, SS_B_1, SS_C_1)
```

The data for single-stranded DNA at $\mathbf{D} = 2$ is

```
>>> SS_2 = CombinedData(SS_A_2, SS_B_2)
```

The data for double-stranded DNA at $\mathbf{D} = 1$ is

```
>>> DS_1 = CombinedData(DS_A_1, DS_B_1)
```

The data for double-stranded DNA at $\mathbf{D} = 2$ is

```
>>> DS_2 = CombinedData(DS_A_2, DS_B_2)
```

Having combined the data, we calculate F_{min} via

```
>>> F_min = 0
>>> from src.get_data import C_REF, F_REF
>>> for dataset in (SS_1, SS_2, DS_1, DS_2):
...     wells = dataset.C*C_REF <= 0.5e-6
...     "Num wells <= 0.5e-6 mol/L for %s, %i is %d" % (dataset.t, int(dataset.D),␣
↪len(dataset.C[wells]))
...     max_t_D = dataset.F[-1, wells].max()*F_REF
...     if max_t_D > F_min:
...         F_min = max_t_D
...
'Num wells <= 0.5e-6 mol/L for SS, 1 is 36'
'Num wells <= 0.5e-6 mol/L for SS, 2 is 24'
'Num wells <= 0.5e-6 mol/L for DS, 1 is 24'
'Num wells <= 0.5e-6 mol/L for DS, 2 is 24'
```

The value for F_{min} is

```
>>> F_min
231432.0
```

The subsets of the data are made via

```
>>> for dataset in (SS_1, SS_2, DS_1, DS_2):
...     dataset.make_subset(F_min/F_REF)
...     "Max temperature for %s, %i is %g K" % (dataset.t, int(dataset.D), dataset.T.
↪max())
...
'Max temperature for SS, 1 is 316.5 K'
'Max temperature for SS, 2 is 324.5 K'
'Max temperature for DS, 1 is 322 K'
'Max temperature for DS, 2 is 329 K'
```

2.2 Noise Removal

```
>>> from src.noise_removal import compute_M_LS
>>> import numpy as np
```

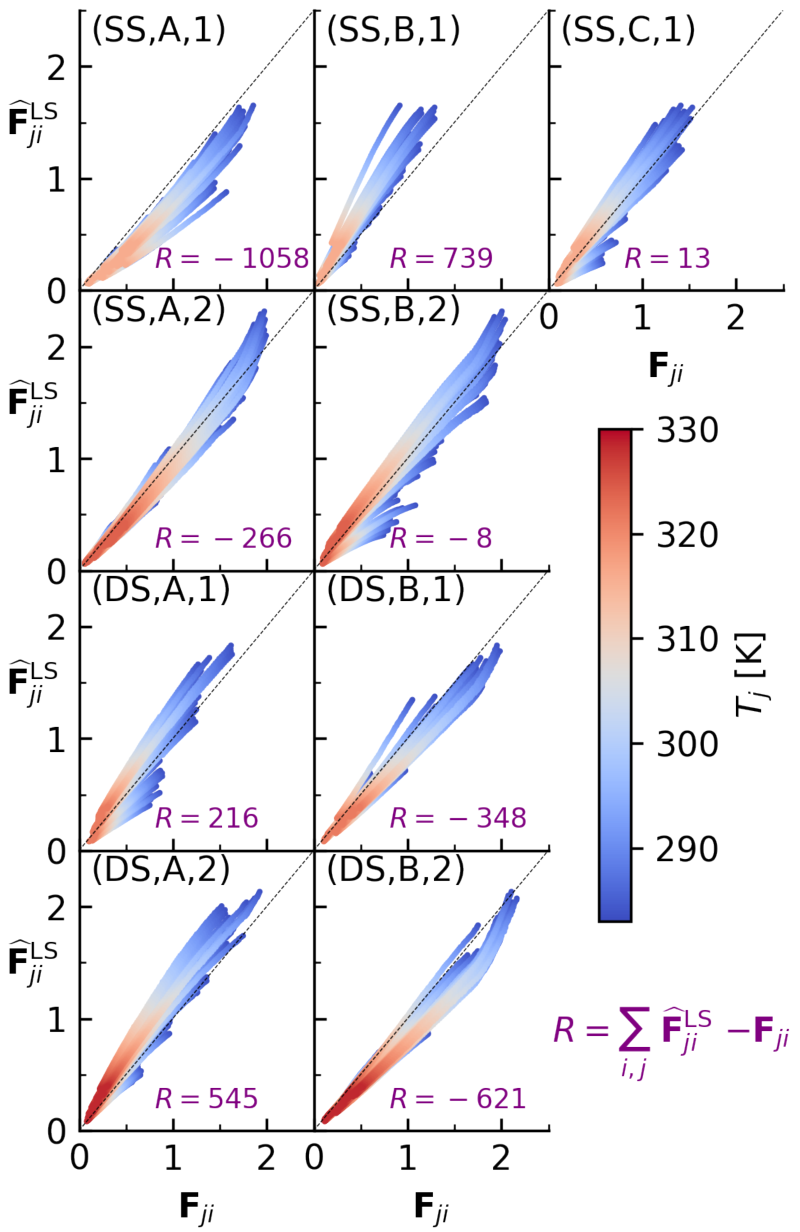
For each dataset, compute M^{LS} via Equation (21) and store the results,

```
>>> F_hats = []
>>> for dataset in (SS_1, SS_2, DS_1, DS_2):
...     M_LS = compute_M_LS(dataset.F, dataset.C)
...     F_hats.append(np.outer(M_LS, dataset.C))
...
```

and then plot $\mathbf{F}$ vs $\widehat{\mathbf{F}}$ for each combination via

```
>>> from src.plot_noise_removal import plot_Fhat_vs_F
>>> plot_Fhat_vs_F(
...     (SS_1.F, SS_2.F, DS_1.F, DS_2.F),
...     tuple(F_hats),
...     (SS_1.T, SS_2.T, DS_1.T, DS_2.T),
...     "figure3.png",
...     sname=r"$\widehat{\mathbf{F}}_{ji}^\mathrm{LS}$")
...
```

This is Figure 3 in the main text, which looks like



Subsequently, Equation (22) is solved using *src.noise_removal.predictor_corrector()* and $V(\mathbf{M})$ and $V(\mathbf{C})$ are calulated

```
>>> from src.noise_removal import predictor_corrector
>>> RHO_SQUARED = 0.1
>>> for dataset in (SS_1, SS_2, DS_1, DS_2):
...     dataset.M_tls, dataset.C_hat = predictor_corrector(
...         dataset.F, dataset.C, RHO_SQUARED
...     )
...     dataset.Fhat_tls = np.outer(dataset.M_tls, dataset.C_hat.T)
...
...     m, n = dataset.F.shape
...     H = np.vstack([
...             np.hstack([np.eye(n)*(RHO_SQUARED+np.inner(dataset.M_tls, dataset.M_
↪tls)), np.zeros((n, m))]),
```

(continues on next page)

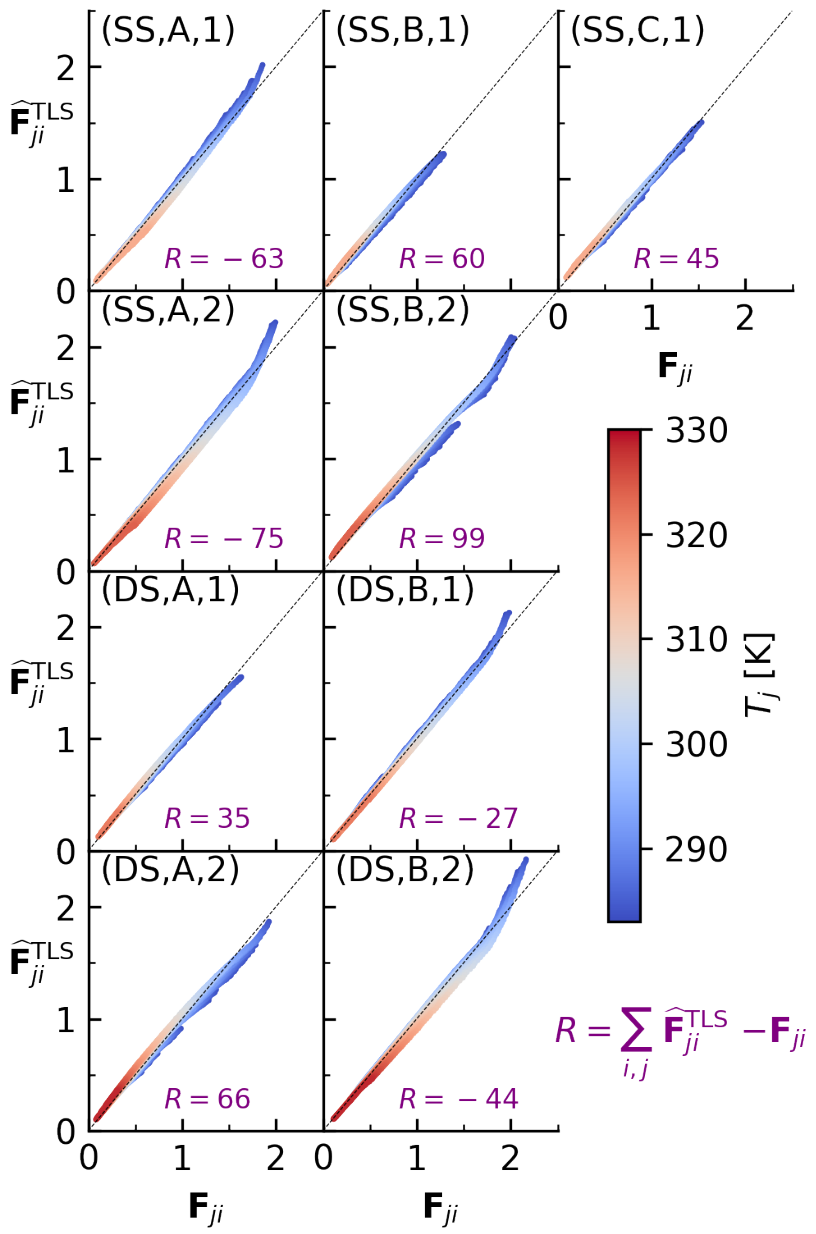
(continued from previous page)

```
...             np.hstack([np.zeros((m, n)), np.eye(m)*np.inner(dataset.C_hat,␣
↪dataset.C_hat)])
...         ])
...     dF = dataset.Fhat_tls - dataset.F
...     dC = dataset.C_hat - dataset.C
...     f_star = (dF*dF).sum() + RHO_SQUARED*(dC*dC).sum()
...     bbV = f_star / (m*(n-1))*np.linalg.inv(H)
...     dataset.V_C = np.array([bbV[i, i] for i in range(n)])
...     dataset.V_M = np.array([bbV[j, j] for j in range(n, n + m)])
...
...     dataset.M_std = np.sqrt(dataset.V_M)
...     dataset.C_std = np.sqrt(dataset.V_C)
...
Total number of iterations was 756
Total number of iterations was 1347
Total number of iterations was 1928
Total number of iterations was 3073
```

The results are plotted via Figure 4

```
>>> plot_Fhat_vs_F(
...     (SS_1.F, SS_2.F, DS_1.F, DS_2.F),
...     (SS_1.Fhat_tls, SS_2.Fhat_tls, DS_1.Fhat_tls, DS_2.Fhat_tls),
...     (SS_1.T, SS_2.T, DS_1.T, DS_2.T),
...     "figure4.png",
...     sname=r"$\widehat{\mathbf{F}}_{ji}^\mathrm{TLS}$")
...
```

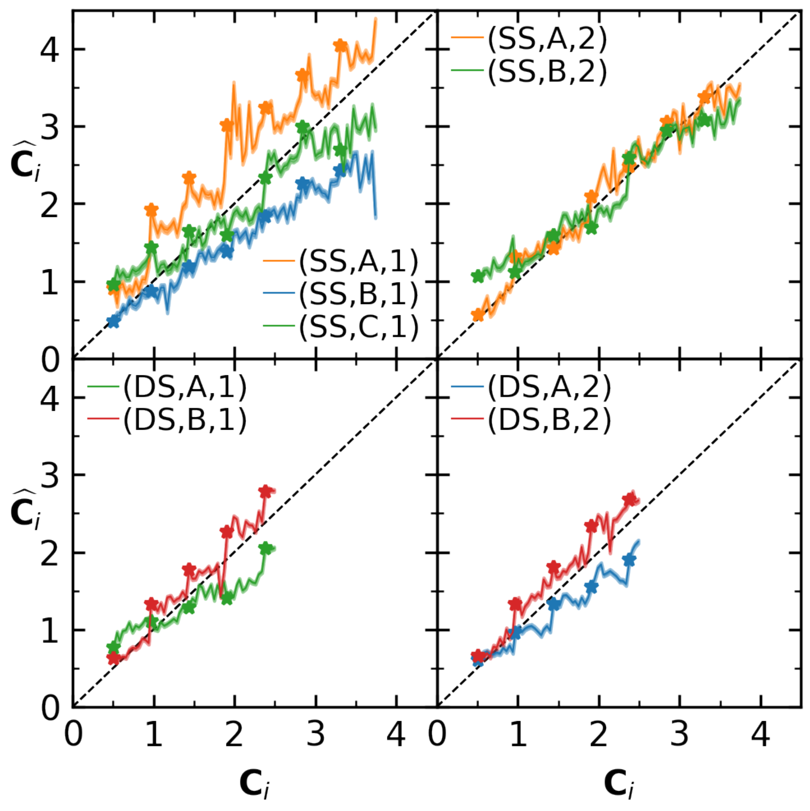
which looks like



and Figure 5,

```
>>> from src.plot_noise_removal import plot_Chat_vs_C
>>> plot_Chat_vs_C(
...     (SS_1.C, SS_2.C, DS_1.C, DS_2.C),
...     (SS_1.C_hat, SS_2.C_hat, DS_1.C_hat, DS_2.C_hat),
...     (SS_1.C_std, SS_2.C_std, DS_1.C_std, DS_2.C_std),
...     "figure5.png"
... )
...
```

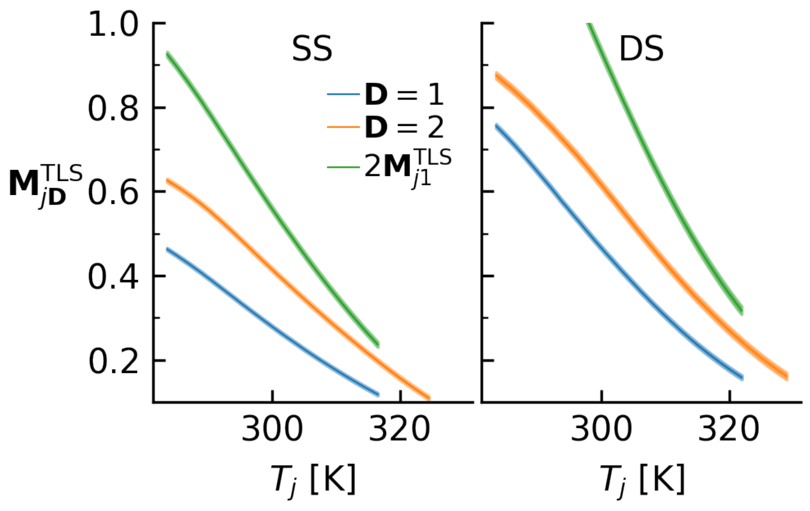
which looks like



and Figure 6,

```
>>> from src.plot_noise_removal import plot_figure6
>>> plot_figure6(
...     (SS_1.M_tls, SS_2.M_tls, DS_1.M_tls, DS_2.M_tls),
...     (SS_1.M_std, SS_2.M_std, DS_1.M_std, DS_2.M_std),
...     (SS_1.T, SS_2.T, DS_1.T, DS_2.T)
... )
...
```

which looks like



2.3 Parameter Extraction

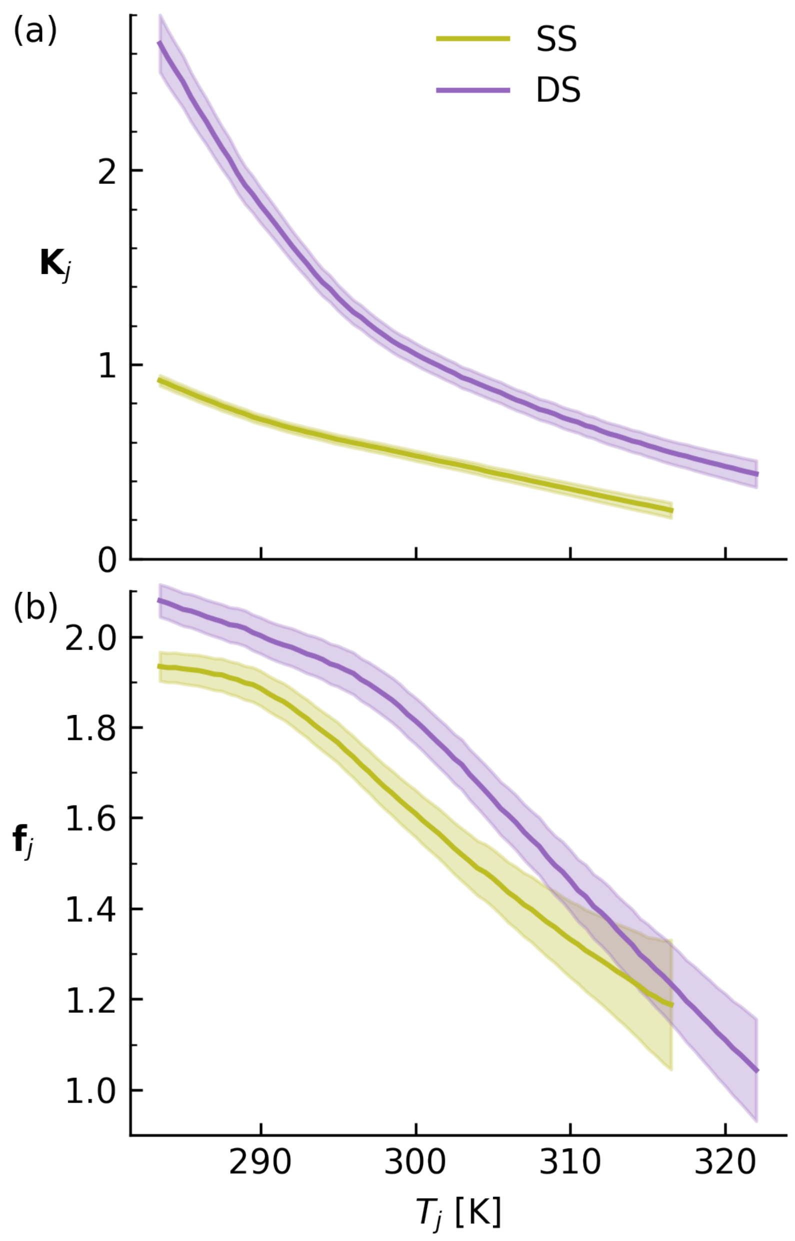
First, we combine the DNA concentrations associated with each DNA type into an instance of *src.parameter_extraction.Parameters*

```
>>> from src.parameter_extraction import Parameters
>>> SS = Parameters(SS_1, SS_2)
>>> DS = Parameters(DS_1, DS_2)
```

These instances now perform all parameter calculations; we can readily plot the Figure 7 as

```
>>> from src.plot_params import plot_figure7
>>> plot_figure7(
...     SS.T, SS.get_f(), SS.get_K(), SS.get_f_std(), SS.get_K_std(),
...     DS.T, DS.get_f(), DS.get_K(), DS.get_f_std(), DS.get_K_std(),
... )
...
```

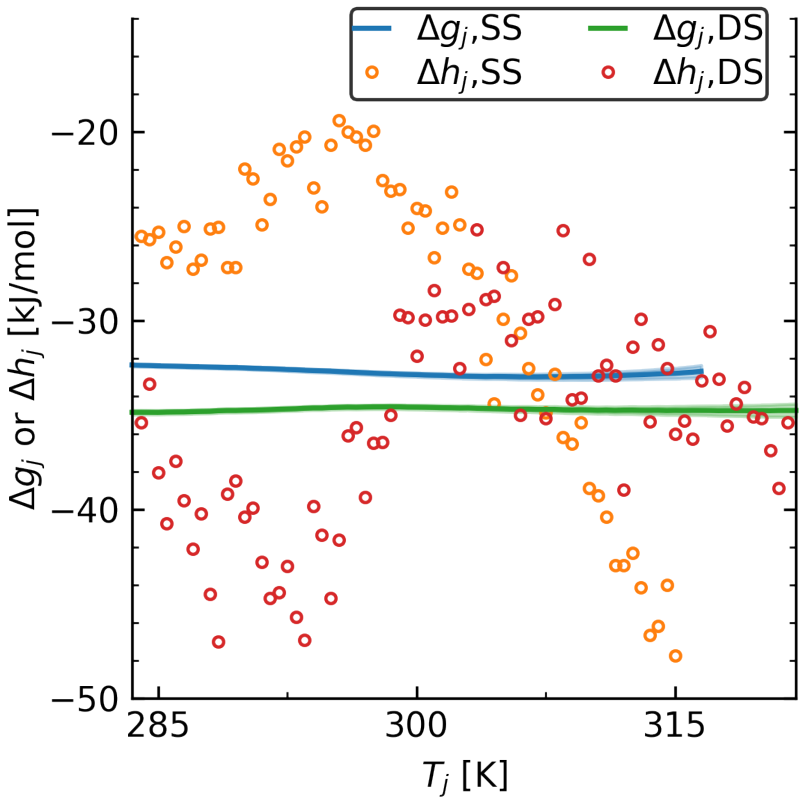
which looks like



and Figure 8,

```
>>> from src.plot_params import plot_figure8
>>> plot_figure8(SS, DS)
dg_SS at 295.00 K is -32.682584 +/- 0.088140
dg_DS at 295.00 K is -34.608172 +/- 0.108761
```

which looks like



2.4 Supplementary Figures

Figure S1 is made via

```
>>> from src.plot_raw_data import make_figure_S1
>>> make_figure_S1()
```

Figures S2, S3, S4 are made via

```
>>> from src.plot_params import plot_figure7, plot_figure8, plot_figure_S2, plot_
↪figure_S3, plot_figure_S4, plot_figure_S5
>>> plot_figure_S2(SS_1, SS_2, DS_1, DS_2)
>>> plot_figure_S3(SS, DS)
>>> plot_figure_S4(SS, DS)
```

Figure S5 is made via

```
>>> from src.parameter_extraction import calculate_relative_brightness, calculate_
↪relative_brightness_err
>>> rb = calculate_relative_brightness(SS.get_f(), DS.get_f())
>>> d_rb = calculate_relative_brightness_err(SS.M1, SS.M2, DS.M1, DS.M2,
...     SS.V_M1, SS.V_M2, DS.V_M1, DS.V_M2)
>>> plot_figure_S5(SS.T, rb, d_rb)
```

CHAPTER

THREE

MODULES

3.1 Wells

`src.wells.`**`column_row_to_well`**(*ix: int*, *iy: int*) → str

Convert indices to well

```
>>> column_row_to_well(0, 0)
'A1'
>>> column_row_to_well(0, 7)
'H1'
>>> column_row_to_well(11, 0)
'A12'
>>> column_row_to_well(11, 7)
'H12'
>>> well_to_row(column_row_to_well(11, 7))
7
>>> well_to_column(column_row_to_well(11, 7))
11
```

Parameters

- **`ix`** (*`int`*) – a index
- **`iy`** (*`int`*) – y index

Returns well

Return type str

`src.wells.`**`number_to_column`**(*number: int*) → int

```
>>> number_to_column(1)
1
>>> number_to_column(0)
0
>>> number_to_column(11)
11
>>> number_to_column(95)
11
>>> number_to_column(84)
0
```

`src.wells.`**`number_to_row`**(*number: int*) → int

```
>>> number_to_row(0)
0
>>> number_to_row(1)
0
>>> number_to_row(11)
0
>>> number_to_row(12)
1
>>> number_to_row(95)
7
>>> number_to_row(84)
7
```

`src.wells.`**`number_to_well`**(*number: int*) → str

Return number of well

```
>>> number_to_well(0)
'A1'
>>> number_to_well(11)
'A12'
>>> number_to_well(95)
'H12'
>>> number_to_well(84)
'H1'
```

`src.wells.`**`well_to_column`**(*well: str*) → int

Convert well name to column

```
>>> well_to_column('H100')
Traceback (most recent call last):
    ...
ValueError: '100' is not in list
>>> well_to_column('H1')
0
>>> well_to_column('A1')
0
>>> well_to_column('B12')
11
>>> well_to_column('B13')
Traceback (most recent call last):
    ...
ValueError: '13' is not in list
>>> well_to_column('Z1')
0
```

Parameters **`well`** (*`str`*) – name of well

Returns **ix** – a-index for well

Return type int

`src.wells.`**`well_to_number`**(*well: str*) → int

Return number of well

```
>>> well_to_number("A1")
0
>>> well_to_number("A12")
```

(continues on next page)

(continued from previous page)

```
11
>>> well_to_number("H12")
95
>>> well_to_number("H1")
84
```

src.wells.**well_to_row**(*well: str*) → int

Well to y index

```
>>> well_to_row('H100')
7
>>> well_to_row('H1')
7
>>> well_to_row('A1')
0
>>> well_to_row('B1')
1
>>> well_to_row('Z1')
Traceback (most recent call last):
    ...
ValueError: 'Z' is not in list
```

Parameters **well** (*str*) – well name

Returns **iy** – y-index of well

Return type int

3.2 Get Data

class src.get_data.**CombinedData**(**replicates*)

Combined data from several replicate plates. See changes to F and C in *Raw Data and Scaling* portion of *Results and Discussion*.

Variables

- **F** (*np.ndarray*) – Fluorescence data, $\mathbf{F}_k^t$ in Equation (16a)
- **C** (*np.ndarray*) – Dye concentrations, $\mathbf{C}_k^t$ in Equation (16a)
- **D** (*float*) – DNA concentration, $\mathbf{D}_k$ in Equation (16a)
- **t** (*str*) – Type of DNA, t, "SS" or "DS"
- **M_tls** (*np.array*) – $\mathbf{M}^{\mathrm{TLS}}$, set externally, defaults to np.array([])
- **C_hat** (*np.array*) – $\widehat{\mathbf{C}}$, set externally, defaults to np.array([])
- **V_M** (*np.array*) – $V(\mathbf{M})$ (see Section S1.2), defaults to np.array([])
- **V_C** (*np.array*) – $V(\mathbf{C})$ (see Section S1.2), defaults to np.array([])
- **M_std** (*np.array*) – $\sqrt{V(\mathbf{M})}$ (see Section S1.2), defaults to np.array([])
- **C_std** (*np.array*) – $\sqrt{V(\mathbf{C})}$ (see Section S1.2), defaults to np.array([])

__init__(**replicates*) → None

Combine replicate F and C

Parameters **`replicates`** (*`typing.List[Data]`*) – list of plates to gather together as replicates

`make_subset` (*F_min*)
Make subset of data as described in *Raw Data and Scaling* portion of manuscript

Note: **F** and **C** are overwritten.

`class` `src.get_data.`**`RawData`** (*fluorescence_file_name*, *D_k*, *t*, *l*, *dye_conc_file_name='dye_conc_uM.csv'*)
Stores raw data.

Variables

- **`F`** (*`np.ndarray`*) – Fluorescence data, $F_k^{t\ell}$ (see Equation 14 of main text)
- **`C`** (*`np.ndarray`*) – Dye concentrations, C (see Equation 13 of main text)
- **`D`** (*`float`*) – DNA concentration, D_k in mol/L
- **`t`** (*`str`*) – Type of DNA, t, `"SS"` or `"DS"` or `"None"`.
- **`l`** (*`str`*) – Replicate name, ℓ is A, B, or C

`__init__` (*fluorescence_file_name*, *D_k*, *t*, *l*, *dye_conc_file_name='dye_conc_uM.csv'*)
Scale the data before interpolating/solving optimization problem.

Parameters

- **`fluorescence_file_name`** (*`str`*) – name of fluorescence file within data folder
- **`D_k`** (*`float`*) – Total concentration of DNA in mol/L D_k
- **`dye_conc_file_name`** (*`str, optional`*) – name of dye concentration file name within data folder, defaults to `"dye_conc_uM.csv"`
- **`t`** (*`str`*) – Type of DNA, `"SS"`, `"DS"`, or `"None"`.
- **`l`** (*`str`*) – Replicate name, ℓ is A, B, or C

`src.get_data.`**`excel_to_data`** (*f_name: str*, *channel='GREEN'*)
Convert "Raw Data" sheet of excel file to pandas dataframe. Uses Equation (12) of manuscript to calculate temperature associated with each cycle.

Parameters

- **`f_name`** (*`str`*) – name of excel file
- **`channel`** (*`str, optional`*) – name of channel to investigate, defaults to `"GREEN"`

Returns Formatted data frame, with wells sorted from A1, A2 … H11, H12

Return type pd.DataFrame

`src.get_data.`**`get_C`** (*file_name*)
Get total dye concentration associated with each well.

Parameters **`file_name`** (*`str`*) – CSV file formatted like a 96-well plate. The top left corner looks like

Row	1	2
A	.	.
B	.	.
C	.	.

The values are concentrations of dye in units of mol/L

Returns Mapping of well name (“A1”,. . .) to dye concentration [units of mol/L]

Return type Dictionary

3.3 Noise Removal

`src.noise_removal.`**`compute_M_LS`**(*F*, *C*)

Calculate $\mathbf{M}$ by least-squares approximation

Returns M^{LS}, see Equation (21)

Return type np.array

`src.noise_removal.`**`compute_M_plus`**(*F*, *c_plus*)

Get updated guess for M

Parameters

- **`F`** (*`np.ndarray`*) – Fluorescence matrix $\mathbf{F}$
- **`c_plus`** (*`np.array`*) – Concentration matrix updated $\mathbf{c}_+$

Returns M_+ by Equation (S3b)

Return type np.array

`src.noise_removal.`**`compute_c_plus`**(*F*, *C*, *M_minus*, *rho_squared*)

Compute updated guess of concentrations, $\mathbf{c}_+$

Parameters

- **`F`** (*`np.ndarray`*) – Fluorescence $\mathbf{F}$
- **`C`** (*`np.array`*) – Dye Concentration $\mathbf{C}$
- **`M_minus`** (*`np.array`*) – Guess for M, M_-
- **`rho_squared`** (*`float`*) – Weight, ρ^2

Returns $\mathbf{c}_+$ by Equation (S3a)

Return type np.ndarray

`src.noise_removal.`**`predictor_corrector`**(*F*, *C*, *rho_squared*, *maxiter=100000*, *print_iter=True*)

Solve Equation (22) with predictor–corrector algorithm

Parameters

- **`F`** (*`np.ndarray`*) – Fluorescence data $\mathbf{F}$
- **`C`** (*`np.ndarray`*) – Dye concentration data $\mathbf{C}$
- **`rho_squared`** (*`float`*) – Weighting factor for concentrations, ρ^2 in Equation (22a)
- **`maxiter`** (*`int, optional`*) – maximum iterations allowed, by default 100000
- **`print_iter`** (*`bool, optional`*) – whether or not to print the total number of iterations performed, by default True

Returns **(M, c)** – solution, $(\mathrm{M}^{\mathrm{TLS}}, \widehat{\mathbf{C}})$

Return type tuple(np.array, np.array)

3.4 Parameter Extraction

class `src.parameter_extraction.`**`Parameters`**(*cls1: src.get_data.CombinedData, cls2: src.get_data.CombinedData*)

Stores multiple instances of CombinedData for one DNA type

Variables

- **`M1`** (`np.array`) – $\mathbf{M}^{\mathrm{TLS}}$ associated with $\mathbf{D} = 1$
- **`M2`** (`np.array`) – $\mathbf{M}^{\mathrm{TLS}}$ associated with $\mathbf{D} = 2$. Several high temperatures are removed to reflect smaller temperature range associated with $\mathbf{D} = 1$
- **`C1`** (`np.array`) – $\widehat{\mathbf{C}}$ associated with $\mathbf{D} = 1$
- **`C2`** (`np.array`) – $\widehat{\mathbf{C}}$ associated with $\mathbf{D} = 2$
- **`r`** (`np.array`) – r, as defined in Equation (S7).
- **`V_C1`** (`np.array`) – $V(\mathbf{C})$ associated with $\mathbf{D} = 1$
- **`V_C2`** (`np.array`) – $V(\mathbf{C})$ associated with $\mathbf{D} = 2$
- **`V_M1`** (`np.array`) – $V(\mathbf{M})$ associated with $\mathbf{D} = 1$
- **`V_M2`** (`np.array`) – $V(mathbf M)$ associated with $\mathbf{D} = 2$. Several high temperatures are removed to reflect smaller temperature range associated with $\mathbf{D} = 1$
- **`dT`** (`float`) – Change in temperature from one cycle to next, ΔT

`__init__`(*cls1: src.get_data.CombinedData, cls2: src.get_data.CombinedData*)

Initialize data

Note: Since different temperature ranges for each, need to make subset of dataset that has more temperatures. Dataset with lower DNA concentration `cls1` always has less temperatures.

Parameters

- **`cls1`** (`CombinedData`) – Data of DNA type at $\mathbf{D} = 1$
- **`cls2`** (`CombinedData`) – Data of DNA type at $\mathbf{D} = 2$

`get_K`() $\rightarrow$ numpy.array

Get $\mathbf{K}$ from vectorized version of Equation (24)

Returns $\mathbf{K}$

Return type np.array

`get_K_std`() $\rightarrow$ numpy.array

Get standard deviation estimate of $\mathbf{K}$

Returns

$$\sqrt{\Delta \mathbf{M}^2 \left(4V(\mathbf{M}_1) + 2V(\mathbf{M}_2)\right) + \frac{\Delta H^2}{8\Delta \mathbf{M}^4}\left(V(\mathbf{M}_1) + V(\mathbf{M}_2)\right)}$$

where $\Delta H := 2\mathbf{M}_1 - \mathbf{M}_2$, $\Delta \mathbf{M} := \mathbf{M}_2 - \mathbf{M}_1$

Return type np.array

`get_dg`() $\rightarrow$ numpy.array

Get free energy of dye binding, Δg, vectorized version of Equation (29).

Returns

Return type np.array

get_dg_std() → numpy.array

Get estimate of standard deviation in Δg.

Returns

$$\frac{RT_j}{2\Delta\mathbf{M}_j\left(2\mathbf{M}_{j1}-\mathbf{M}_{j2}\right)}\sqrt{\mathbf{M}_{j2}^2 V(\mathbf{M}_{j1})+\mathbf{M}_{j1}^2 V(\mathbf{M}_{j2})}$$

where $\Delta\mathbf{M}_j=\mathbf{M}_{j2}-\mathbf{M}_{j1}$.

Return type np.array

get_dh() → numpy.array

Get differential enthalpy of binding, Δh as the vectorized version of Equation (30).

Returns

Return type np.array

get_dh_std() → numpy.array

Get estimate of error in Δh_j

Returns

Return type np.array

get_f() → numpy.array

Get $\mathbf{f}$ from vectorized version of Equation (27)

Returns $\mathbf{f}$

Return type np.array

get_f_std() → numpy.array

Get standard deviation estimate of $\mathbf{f}$

Returns

$$\sqrt{V(\mathbf{M}_1)+V(\mathbf{M}_2)+\frac{E_B^2 V_A+E_A^2 V_B}{E_B^4}}$$

where

$$\begin{aligned}
V_A &:= (V_B+H_+^2)(V(\mathbf{M}_1)+V(\mathbf{M}_2)+\Delta\mathbf{M}^2)-E_A^2\\
V_B &:= 2V(\mathbf{M}_1)+V(\mathbf{M}_2)\\
E_A &:= \Delta\mathbf{M}H_+\\
E_B &:= 2\mathbf{M}_1-\mathbf{M}_2\\
\Delta\mathbf{M} &:= \mathbf{M}_2-\mathbf{M}_1\\
H_+ &:= 2\mathbf{M}_1+\mathbf{M}_2
\end{aligned}$$

Return type np.array

get_phi_1() → numpy.array

Get φ_1, vectorized version of Equation (S6a)

Returns

Return type np.array

get_phi_2()
Get φ_2, vectorized version of Equation (S6b)

Returns

Return type np.array

get_std_phi_1() $\rightarrow$ numpy.array
Get estimate of standard deviation in φ_1

Returns

$$\frac{2}{\mathbf{M}_{2j}}\sqrt{V(\mathbf{M}_{1j}) + r_j^2 V(\mathbf{M}_{2j})}$$

Return type np.array

get_std_phi_2()
Get estimate of standard deviation in φ_2

Returns

$$\frac{1}{\mathbf{M}_{1j}}\sqrt{V(\mathbf{M}_{2j}) + r_j^{-2} V(\mathbf{M}_{1j})}$$

Return type np.array

src.parameter_extraction.**calculate_relative_brightness**(*f_SS*, *f_DS*)
Calculate relative brightness, Equation (28).

Parameters

- **f_SS** (*np.array*) – Molar fluorescence of single-stranded DNA, $\mathbf{f}^{\mathrm{SS}}$.
- **f_DS** (*np.array*) – Molar fluorescence of double-stranded DNA, $\mathbf{f}^{\mathrm{DS}}$.

Returns Relative brightness, $\mathbf{f}_j^{\mathrm{DS}}/\mathbf{f}_j^{\mathrm{SS}}$ for each j associated with SS.

Return type np.array

src.parameter_extraction.**calculate_relative_brightness_err**(*SS_M1*, *SS_M2*, *DS_M1*, *DS_M2*, *SS_V_M1*, *SS_V_M2*, *DS_V_M1*, *DS_V_M2*) $\rightarrow$ numpy.array
Estimate error in relative brightness, Equation (28).

Parameters

- **SS_M1** (*np.array*) – $\mathrm{M}_1^{\mathrm{SS}}$
- **SS_M2** (*np.array*) – $\mathrm{M}_2^{\mathrm{SS}}$
- **DS_M1** (*np.array*) – $\mathrm{M}_1^{\mathrm{DS}}$
- **DS_M2** (*np.array*) – $\mathrm{M}_2^{\mathrm{DS}}$
- **SS_V_M1** (*np.array*) – $V(\mathrm{M}_1^{\mathrm{SS}})$ _description_
- **SS_V_M2** (*np.array*) – $V(\mathrm{M}_2^{\mathrm{SS}})$
- **DS_V_M1** (*np.array*) – $V(\mathrm{M}_1^{\mathrm{DS}})$
- **DS_V_M2** (*np.array*) – $V(\mathrm{M}_2^{\mathrm{DS}})$

Returns Estimate of error in relative brightness

Return type np.array

3.5 Plotting

3.5.1 Raw Data

`src.plot_raw_data.`**`make_figure_2`**(*SS_A_1: src.get_data.RawData*, *SS_B_1: src.get_data.RawData*, *SS_C_1: src.get_data.RawData*, *SS_A_2: src.get_data.RawData*, *SS_B_2: src.get_data.RawData*, *DS_A_1: src.get_data.RawData*, *DS_B_1: src.get_data.RawData*, *DS_A_2: src.get_data.RawData*, *DS_B_2: src.get_data.RawData*, *A_1: src.get_data.RawData*)

Makes Figure 2

`src.plot_raw_data.`**`make_figure_S1`**()

Makes Figure S1.

`src.plot_raw_data.`**`plot_linemap`**(*cls: src.get_data.RawData*, *ax*, *colorbar=False*, *get_ticks=False*, *ordered_by_row=True*)

Plot F vs C for various temperatures (colors)

Parameters

- **`cls`** (*`Data`*) – Instance of data (i.e., a dataset)
- **`ax`** (*`axis`*) – Matplotlib axis to plot on
- **`colorbar`** (*`bool, optional`*) – whether or not to plot colorbar, in which case the axis is colorbar axis, by default False
- **`get_ticks`** (*`bool, optional`*) – whether or not to return list of ticks, by default False
- **`ordered_by_row`** (*`bool, optional`*) – whether or not well concentrations are ordered by row, by default True

Returns Only returns list of `get_ticks=True`.

Return type None or list

3.5.2 Noise Removal

3.5.3 Parameters

`src.plot_params.`**`plot_figure_S5`**(*T*, *rb*, *d_rb*)

Plot figure S5, relative brightness

Parameters

- **`T`** (*`np.array`*) – array of temperatures
- **`rb`** (*`np.array`*) – Relative brightness, $\mathbf{f}_j^{\mathrm{DS}}/\mathbf{f}_j^{\mathrm{SS}}$
- **`d_rb`** (*`np.array`*) – standard deviation in relative brightness

3.6 Util

`src.util.`**`figure_name_to_abspath`**(*fname: str*) → str

Figure name to absolute path

Parameters **`fname`** (*`str`*) – name of figure

Returns absolute path to name of figure

Return type str

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX