
Dye/DNA Plates Documentation

Release v2.0.0

Robert F. DeJaco

Jan 27, 2023

CONTENTS:

1	Modules	1
1.1	Wells	1
1.2	Get Data	3
1.3	Noise Removal	5
1.4	Parameter Extraction	6
1.5	Plotting	9
1.5.1	Raw Data	9
1.5.2	Noise Removal	9
1.5.3	Parameters	10
1.6	Util	10
2	Indices and tables	11
	Python Module Index	13

MODULES

1.1 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`

1.2 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

- \mathbf{F} (*np.ndarray*) – Fluorescence data, \mathbf{F}_D^t in Equation (16a)
- \mathbf{C} (*np.ndarray*) – Dye concentrations, \mathbf{C}_D^t in Equation (16a)
- \mathbf{D} (*float*) – DNA concentration, \mathbf{D} in Equation (16a)
- \mathbf{t} (*str*) – Type of DNA, t , "SS" or "DS"
- \mathbf{N} (*int*) – number of nucleobases per single strand, N
- $\mathbf{M_tls}$ (*np.array*) – \mathbf{M}^{TLS} , set externally, defaults to `np.array([])`
- $\mathbf{C_hat}$ (*np.array*) – $\hat{\mathbf{C}}$, set externally, defaults to `np.array([])`
- $\mathbf{V_M}$ (*np.array*) – $V(\mathbf{M})$ (see Section S1.2), defaults to `np.array([])`
- $\mathbf{V_C}$ (*np.array*) – $V(\mathbf{C})$ (see Section S1.2), defaults to `np.array([])`
- $\mathbf{M_std}$ (*np.array*) – $\sqrt{V(\mathbf{M})}$ (see Section S1.2), defaults to `np.array([])`
- $\mathbf{C_std}$ (*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*, *B_d*, *t*, *l*, *N*,
dye_conc_file_name='dye_conc_uM.csv')

Stores raw data.

Variables

- **F** (*np.ndarray*) – Fluorescence data, $F_d^{t\ell}$ (see Equation 14 of main text)
- **C** (*np.ndarray*) – Dye concentrations, C (see Equation 13 of main text)
- **B** (*float*) – DNA concentration, B_d in mol/L
- **t** (*str*) – Type of DNA, t , "SS" or "DS" or "None".
- **l** (*str*) – Replicate name, ℓ is A, B, or C
- **N** (*int*) – number of nucleobases per single strand

__init__ (*fluorescence_file_name*, *B_d*, *t*, *l*, *N*, *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
- **B_d** (*float*) – Total concentration of nucleobases in mol/L B_d
- **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
- **N** (*int*) – number of nucleobases per single strand

`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

1.3 Noise Removal

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

Calculate 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 F
- **c_plus** (`np.array`) – Concentration matrix updated 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, c_+

Parameters

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

Returns 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 F
- **C** (`np.ndarray`) – Dye concentration data 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, (M^{TLS} , \hat{C})

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

1.4 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

- **N**(*int*) – number of nucleobases, N
- **M1**(*np.array*) – M^{TLS} associated with $D = 1$
- **M2**(*np.array*) – M^{TLS} associated with $D = 2$. Several high temperatures are removed to reflect smaller temperature range associated with $D = 1$
- **C1**(*np.array*) – \hat{C} associated with $D = 1$
- **C2**(*np.array*) – \hat{C} associated with $D = 2$
- **r**(*np.array*) – r , as defined in Equation (S7).
- **v_C1**(*np.array*) – $V(C)$ associated with $D = 1$
- **v_C2**(*np.array*) – $V(C)$ associated with $D = 2$
- **v_M1**(*np.array*) – $V(M)$ associated with $D = 1$
- **v_M2**(*np.array*) – $V(\textit{mathbf{M}})$ associated with $D = 2$. Several high temperatures are removed to reflect smaller temperature range associated with $D = 1$
- **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 $D = 1$
- **cls2**(`CombinedData`) – Data of DNA type at $D = 2$

get_K() → numpy.array

Get **K** from vectorized version of Equation (24)

Returns **K**

Return type np.array

get_K_std() → numpy.array

Get standard deviation estimate of **K**

Returns

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

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

Return type np.array

get_dg () → 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

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

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

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 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

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

$$V_B := 2V(\mathbf{M}_1) + V(\mathbf{M}_2)$$

$$E_A := \Delta M H_+$$

$$E_B := 2\mathbf{M}_1 - \mathbf{M}_2$$

$$\Delta M := \mathbf{M}_2 - \mathbf{M}_1$$

$$H_+ := 2\mathbf{M}_1 + \mathbf{M}_2$$

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() → numpy.array

Get estimate of standard deviation in φ_1

Returns

$$\frac{2}{M_{2j}} \sqrt{V(M_{1j}) + r_j^2 V(M_{2j})}$$

Return type np.array

get_std_phi_2()

Get estimate of standard deviation in φ_2

Returns

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

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, f^{SS} .
- **f_DS** (*np.array*) – Molar fluorescence of double-stranded DNA, f^{DS} .

Returns Relative brightness, f_j^{DS}/f_j^{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)` →
numpy.array

Estimate error in relative brightness, Equation (28).

Parameters

- **SS_M1** (*np.array*) – M_1^{SS}
- **SS_M2** (*np.array*) – M_2^{SS}
- **DS_M1** (*np.array*) – M_1^{DS}
- **DS_M2** (*np.array*) – M_2^{DS}

- **SS_V_M1** (*np.array*) – $V(M_1^{SS})$ _description_
- **SS_V_M2** (*np.array*) – $V(M_2^{SS})$
- **DS_V_M1** (*np.array*) – $V(M_1^{DS})$
- **DS_V_M2** (*np.array*) – $V(M_2^{DS})$

Returns Estimate of error in relative brightness

Return type *np.array*

1.5 Plotting

1.5.1 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.make_figure_S3()
```

Makes Figure SX.

```
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

1.5.2 Noise Removal

```
src.plot_noise_removal.plot_error_F(ax, F_k_tl, F_hat_k_tl, T)
```

Plot the predicted fluorescence $\hat{F}_{ijk}^{t\ell}$ against $F_{ijk}^{t\ell}$ for $i = 1, \dots, n$ and $j = 1, \dots, n$.

Parameters

- **ax** (*matplotlib.axes*) – axes to plot on
- **F_k1** (*np.ndarray*) – Experimentally measured fluorescence $\mathbf{F}_k^{t\ell}$.
- **F_hat_k1** (*np.ndarray*) – Predicted fluorescence $\hat{\mathbf{F}}_k^{t\ell}$.
- **T** (*np.array*) – Temperatures in K

Returns image for colorbar

Return type img

1.5.3 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, $\mathbf{f}_j^{\text{DS}}/\mathbf{f}_j^{\text{SS}}$
- **d_rb** (*np.array*) – standard deviation in relative brightness

1.6 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

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

S

- `src.get_data`, 3
- `src.noise_removal`, 5
- `src.parameter_extraction`, 6
- `src.plot_noise_removal`, 9
- `src.plot_params`, 10
- `src.plot_raw_data`, 9
- `src.util`, 10
- `src.wells`, 1