

---

# **Dye/DNA Plates Documentation**

***Release v0.8***

**Robert F. DeJaco**

**Aug 30, 2022**



**CONTENTS:**

<b>1</b>	<b>Modules</b>	<b>1</b>
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
<b>2</b>	<b>Indices and tables</b>	<b>11</b>
	<b>Python Module Index</b>	<b>13</b>



## 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,  $\mathbf{F}_k^t$  in Equation (16a)
- $\mathbf{C}$  (`np.ndarray`) – Dye concentrations,  $\mathbf{C}_k^t$  in Equation (16a)
- $\mathbf{D}$  (`float`) – DNA concentration,  $\mathbf{D}_k$  in Equation (16a)
- $\mathbf{t}$  (*str*) – Type of DNA,  $t$ , "SS" or "DS"
- $\mathbf{M\_tls}$  (`np.array`) –  $\mathbf{M}^{\text{TLS}}$ , 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*, *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,  $\ell$  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,  $\ell$  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

## 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,  $\rho^2$

**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,  $\rho^2$  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

- **M1** (`np.array`) –  $M^{\text{TLS}}$  associated with  $D = 1$
- **M2** (`np.array`) –  $M^{\text{TLS}}$  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,  $\Delta T$

**\_\_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

$$\sqrt{\Delta M^2 (4V(M_1) + 2V(M_2)) + \frac{\Delta H^2}{8\Delta M^4} (V(M_1) + V(M_2))}$$

where  $\Delta H := 2M_1 - M_2$ ,  $\Delta M := M_2 - M_1$

**Return type** `np.array`

**get\_dg** () → `numpy.array`  
Get free energy of dye binding,  $\Delta g$ , vectorized version of Equation (29).

**Returns****Return type** np.array**get\_dg\_std**() → numpy.arrayGet estimate of standard deviation in  $\Delta g$ .**Returns**

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

where  $\Delta M_j = M_{j2} - M_{j1}$ .**Return type** np.array**get\_dh**() → numpy.arrayGet differential enthalpy of binding,  $\Delta h$  as the vectorized version of Equation (30).**Returns****Return type** np.array**get\_dh\_std**() → numpy.arrayGet estimate of error in  $\Delta h_j$ **Returns****Return type** np.array**get\_f**() → numpy.array

Get f from vectorized version of Equation (27)

**Returns f****Return type** np.array**get\_f\_std**() → numpy.array

Get standard deviation estimate of f

**Returns**

$$\sqrt{V(M_1) + V(M_2) + \frac{E_B^2 V_A + E_A^2 V_B}{E_B^4}}$$

where

$$V_A := (V_B + H_+^2)(V(M_1) + V(M_2) + \Delta M^2) - E_A^2$$

$$V_B := 2V(M_1) + V(M_2)$$

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

$$E_B := 2M_1 - M_2$$

$$\Delta M := M_2 - M_1$$

$$H_+ := 2M_1 + M_2$$

**Return type** np.array**get\_phi\_1**() → numpy.arrayGet  $\varphi_1$ , vectorized version of Equation (S6a)**Returns****Return type** np.array

**get\_phi\_2()**

Get  $\varphi_2$ , vectorized version of Equation (S6b)

**Returns**

**Return type** np.array

**get\_std\_phi\_1()** → numpy.array

Get estimate of standard deviation in  $\varphi_1$

**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  $\varphi_2$

**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.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  $\hat{\mathbf{F}}_{ijk}^{t\ell}$  against  $\mathbf{F}_{ijk}^{t\ell}$  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,  $f_j^{\text{DS}}/f_j^{\text{SS}}$
- **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