Bias-UQ-PCR

Release v0.0.1

Jan 12, 2023

CONTENTS:

CHAPTER

ONE

GET DATA

src.get_data.**cycle_to_index**(*cycle: int*) → int

Parameters **cycle** (*int*) – cycle number i, 0 to n

Returns index in matrix or array

Return type int

src.get_data.**file_to_numpy**(*file_abs_path: str*) → numpy.ndarray

Parameters **file_abs_path** (*str*) – path to file name

Notes

All fluorescence values are divided by 10^6 before any analysis

Returns n by m matrix of fluorescence, **F**

Return type np.ndarray

src.get_data.**index_to_cycle**(*index: int*) → int

Parameters **index** (*int*) – index of cycle (0 to n-1)

Returns cycle number (1 to n)

Return type int

CHAPTER

TWO

WELLS

src.wells.**column_row_to_well**(*ix: int*, *iy: int*) → str
Convert indices to well

Parameters

- **ix** (*int*) – x (column) index
- **iy** (*int*) – y (row) index

Returns name of well

Return type str

src.wells.**number_to_column**(*number: int*) → int
Get column index associated with well number

Parameters **number** (*int*) – well number (0 to 95)

Returns column (0 to 11)

Return type int

src.wells.**number_to_row**(*number: int*) → int
Get the row number associated with a well number

Parameters **number** (*int*) – well number

Returns row number (0 to 7)

Return type int

src.wells.**number_to_well**(*number: int*) → str
Get well name from number

Parameters **number** (*int*) – well number (0 to 95)

Returns well name (A1 to H12)

Return type str

src.wells.**well_to_column**(*well: str*) → int
Convert well name to column

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

Returns **ix** – x-index for well (row)

Return type int

src.wells.**well_to_column_row**(*well: str*) → Tuple[int]
Convert well name to column, row

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

Returns column index, row index

Return type tuple(int, int)

src.wells.**well_to_number**(*well: str*) → int
Returns number of well

Parameters **well** (*str*) – name of well (e.g., "A1")

Returns well number (0 to 95)

Return type int

src.wells.**well_to_row**(*well: str*) → int
Well to y index

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

Returns **iy** – y-index of well (row)

Return type int

CHAPTER

THREE

AMPLIFICATION

class src.amplification.**Amplification**(*R: float*, *pbar: float*, *E_U0: numpy.ndarray*, *V_U0: numpy.ndarray*, *farray=<built-in function array>*, *sqrt2=1.4142135623730951*)

Parameters

- **l1** (*float*) – Largest eigenvalue of $\mathbf{A}$, λ_1
- **l2** (*float*) – Smallest eigenvalue of $\mathbf{A}$, λ_2
- **x1** (*np.ndarray*) – Right eigenvector of $\mathbf{A}$ corresponding to λ_1, $\mathbf{x}_1$
- **x2** (*np.ndarray*) – Right eigenvector of $\mathbf{A}$ corresponding to λ_2, $\mathbf{x}_2$
- **z1** (*np.ndarray*) – Left eigenvector of $\mathbf{A}$ corresponding to λ_1, $\mathbf{z}_1$
- **z2** (*np.ndarray*) – Left eigenvector of $\mathbf{A}$ corresponding to λ_2, $\mathbf{z}_2$
- **K1** (*np.ndarray*) – Matrix $\mathbf{K}_1$ defined in text to calculate variance
- **K2** (*np.ndarray*) – Matrix $\mathbf{K}_2$ defined in text to calculate variance

__init__(*R: float*, *pbar: float*, *E_U0: numpy.ndarray*, *V_U0: numpy.ndarray*, *farray=<built-in function array>*, *sqrt2=1.4142135623730951*) → None

Parameters

- **R** (*float*) – ratio of amplification probabilities, R
- **pbar** (*float*) – geometric mean of amplification probabilities, $\bar{p}$
- **E_U0** (*np.ndarray*) – initial expected values of both strands $\mathbb{E}[\mathbf{U}_0]$, 2 by 1 matrix
- **V_U0** (*np.ndarray*) – initial variances of both strands $\mathsf{Var}[\mathbf{U}_0]$, 2 by 2 matrix
- **farray** (*callable, optional*) – function to convert to array type, defaults to np.array
- **sqrt2** – square root of 2, defaults to float calculated by np.sqrt(2)

get_A() → numpy.ndarray

Returns matrix $\mathbf{A}$ calculated by decomposition (10)

Return type np.ndarray

get_Atoi(*i: int*) → numpy.ndarray

Parameters **i** (*int*) – cycle number

Returns matrix $\mathbf{A}^i$ calculated by decomposition (10)

Return type np.ndarray

get_EXi_over_EYi (*cycles: numpy.array*) → numpy.array
Calculate $\mathbb{E}[X_i]/\mathbb{E}[Y_i]$ for a variety of cycles i

Parameters **cycles** (*np.array*) – cycles (integers) to calculate for each i

Returns $\mathbb{E}[X_i]/\mathbb{E}[Y_i]$

Return type np.array

get_E_Ui (*i: int*) → numpy.ndarray

Parameters **i** (*int*) – cycle number

Returns matrix $\mathbb{E}[\mathbf{U}_i] = \mathbf{A}^i \mathbb{E}[\mathbf{U}_0]$

Return type np.ndarray

get_V_Ui (*i: int*) → numpy.ndarray

Parameters **i** (*int*) – cycle number

Returns complicated expression to calculate $\mathsf{Var}[\mathbf{U}_i]$

Return type np.ndarray

get_nu ()

Returns Computes ν, see Equation (22).

Return type float

get_x1z1TK1z1x1T () → numpy.ndarray

Returns computes $\mathbf{x}_1\mathbf{z}_1^\top \mathbf{K}_1 \mathbf{z}_1 \mathbf{x}_1^\top$

Return type np.ndarray

get_x1z1TK2z1x1T ()

Returns computes $\mathbf{x}_1\mathbf{z}_1^\top \mathbf{K}_2 \mathbf{z}_1 \mathbf{x}_1^\top$

Return type np.ndarray

src.amplification.**eval_E_D0** (*E_U0: numpy.ndarray*, *R: float*)
Calculates $\mathbb{E}[D_0]$ from $\mathbb{E}[\mathbf{U}_0]$ and R

Parameters

- **E_U0** (*np.ndarray*) – Initial expected value vector
- **R** (*float*) – square root of ratio of amplification probabilities

Returns $\mathbb{E}[D_0] := \mathbb{E}[X_i - RY_i]$

Return type float

src.amplification.**eval_E_S0** (*E_U0: numpy.ndarray*, *R: float*)
Calculates $\mathbb{E}[S_0]$ from $\mathbb{E}[\mathbf{U}_0]$ and R

Parameters

- **E_U0** (*np.ndarray*) – Initial expected value vector
- **R** (*float*) – square root of ratio of amplification probabilities

Returns $\mathbb{E}[S_0] := \mathbb{E}[X_i + RY_i]$

Return type float

src.amplification.**eval_R**(*p_fr: float*, *p_rf: float*)
Calculate R from p_{rf} and p_{fr}

Parameters

- **p_fr** (*float*) – probability of forward to reverse amplification, p_{fr}
- **p_rf** (*float*) – probability of reverse to forward amplification, p_{rf}

Returns $R = \sqrt{\frac{p_{\mathrm{rf}}}{p_{\mathrm{fr}}}}$

Return type float

src.amplification.**get_pfr_prf**(*R: float*, *pbar: float*)
Get p_{fr} and p_{rf} from R and $\bar{p}$

Parameters

- **R** (*float*) – square root of ratio of probabilities, R
- **pbar** (*float*) – geometric mean of probabilities, $\bar{p}$

Returns

Return type tuple(p_fr, p_rf)

CHAPTER

FOUR

MOLAR FLUORESCENCE

class src.molar_fluorescence.**MolarFluorescence**(*C: numpy.array, files: List[str], name: str*)

Parameters

- **f** (*np.ndarray*) – molar fluorescences for each cycle/well **f**, n by number of wells matrix Determined in units of fluorescence divided by (pmol/L)
- **df** (*np.ndarray*) – standard deviation in molar fluorescences σ, n by number of wells matrix Determined in units of fluorescence divided by (pmol/L)
- **q** (*int*) – number of plates (different concentrations)
- **n** (*int*) – number of cycles
- **m** (*int*) – number of wells
- **F** (*np.ndarray*) – raw fluorescence data (scaled by 10^6 as described in get_data.py). Tensor of dimensions $n \times m \times q$

__init__(*C: numpy.array, files: List[str], name: str*)

Parameters

- **C** (*np.array*) – concentrations of reporter in pmol/L
- **files** (*typing.List[str]*) – list of file names (absolute paths)
- **name** (*str*) – name of reporter (e.g., FAM, Probe)

calculate()

Perform calculations of **f** and standard deviation σ

CHAPTER

FIVE

KINETIC PCR

class src.kinetic_PCR.**HydrolysisProbes**(*C: float*, *Vol: float*, *f_plus: numpy.ndarray*, *f_minus: numpy.ndarray*, *R: float*, *pbar: float*, *E_U0: numpy.ndarray*, *V_U0: numpy.ndarray*, ***kwargs*)

Specific subclass for hydrolysis probes

Parameters

- **n** (*int*) – number of cycles
- **m** (*int*) – number of wells
- **d** (*np.ndarray*) – array of incremental increases in fluorescences, n by m
- **b** (*np.ndarray*) – array of background signals, n by m
- **E_F** (*np.ndarray*) – expected value of fluorescence, n by m, $\mathbb{E}[\mathbf{F}]$
- **V_F** (*np.ndarray*) – variance of fluorescence, n by m, $\mathsf{Var}[\mathbf{F}]$
- **E_DX** (*np.array*) – expected value of change in X, length n, $\mathbf{E}[\Delta X_i]$
- **V_DX** (*np.array*) – variance value of change in X, length n, $\mathsf{Var}[\Delta X_i]$
- **cycles** (*np.array*) – cycle numbers, 1 to n

__init__(*C: float*, *Vol: float*, *f_plus: numpy.ndarray*, *f_minus: numpy.ndarray*, *R: float*, *pbar: float*, *E_U0: numpy.ndarray*, *V_U0: numpy.ndarray*, ***kwargs*) → None

Parameters

- **R** (*float*) – ratio of amplification probabilities, R
- **pbar** (*float*) – geometric mean of amplification probabilities, $\bar{p}$
- **E_U0** (*np.ndarray*) – initial expected values of both strands $\mathbb{E}[\mathbf{U}_0]$, 2 by 1 matrix
- **V_U0** (*np.ndarray*) – initial variances of both strands $\mathsf{Var}[\mathbf{U}_0]$, 2 by 2 matrix
- **f_plus** (*np.ndarray*) – molar fluorescences for each cycle/well of active reporter $\mathbf{f}^+$, n by number of wells matrix Determined in units of fluorescence divided by (pmol/L)
- **f_minus** (*np.ndarray*) – molar fluorescences for each cycle/well of inactive reporter $\mathbf{f}^-$, n by number of wells matrix Determined in units of fluorescence divided by (pmol/L)
- **C** (*float*) – Total concentration of reporters in pmol/L, C
- **Vol** (*float*) – Total volume of solution in L, $\mathcal{V}$
- **kwargs** (*dict*) – extra kwargs for Amplification class

calculate()

Performs calculations, filling in E_F, V_F, E_DX, and V_DX.

get_Cov_XiX0(*i*)

Parameters **i** (*int*) – cycle number

Returns $\mathrm{Cov}[X_i, X_0]$

Return type float

get_E_DX(*i*) → float

Parameters **i** (*int*) – cycle number

Returns $\mathbb{E}[\Delta X_i]$

Return type float

get_V_DX(*i*)

Parameters **i** (*int*) – cycle number

Returns $\mathrm{Var}[\Delta X_i]$

Return type float

src.kinetic_PCR.**plot_fluorescence_curve**(*kls: src.kinetic_PCR.HydrolysisProbes*, *ax*, *well='A1'*)

Parameters

- **kls** (HydrolysisProbes) – Contains all amplification data. Has already computed values
- **ax** (*matplotlib.axes*) – axes to plot on
- **well** (*str*) – name of well to plot

Notes

- Plots expected value of flourescence and shades above and below with 3 standard deviations
- Also plots maximum expected value and mininum expected value of all wells in blue dashed lines

CHAPTER

SIX

PLOTTING FIGURES

plot_figures.**plot_figure2**(*R=0.9*, *pbar=0.85*, *max_cycle=4*)

Parameters

- **R** (*float, optional*) – choice for value of R, defaults to 0.9
- **pbar** (*float, optional*) – choice for value of $\bar{p}$, defaults to 0.85
- **max_cycle** (*int, optional*) – choice for maximum cycle to plot, defaults to 4

plot_figures.**plot_figure4**(*plus: src.molar_fluorescence.MolarFluorescence*, *minus: src.molar_fluorescence.MolarFluorescence*)

Plot figure 4

Parameters

- **plus** (MolarFluorescence) – fluorescence data associated with active probe
- **minus** (MolarFluorescence) – fluorescence data associated with inactive probe

plot_figures.**plot_figure5**(*f_plus: numpy.ndarray*, *f_minus: numpy.ndarray*, *C=0.125*, *Vol=2e-05*, *R=1.0*, *pbar=0.9*)

Parameters

- **f_plus** (*np.ndarray*) – molar fluorescences for each cycle/well of active reporter $\mathbf{f}^{+}$, n by number of wells matrix Determined in units of fluorescence divided by (pmol/L)
- **f_minus** (*np.ndarray*) – molar fluorescences for each cycle/well of inactive reporter $\mathbf{f}^{-}$, n by number of wells matrix Determined in units of fluorescence divided by (pmol/L)
- **C** (*float, optional*) – concentration in pmol/L, defaults to 0.125
- **Vol** (*float, optional*) – volume in L, $\mathcal{V}$, defaults to 20e-6
- **R** (*float, optional*) – square root of ratio of probabilities, R, defaults to 1.0
- **pbar** (*float, optional*) – geometric mean of probabilities, $\bar{p}$, defaults to 0.9

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

s