Deep Dye Drop gating Documentation

Kartik Subramanian, Marc Hafner

Aug 17, 2023

Contents

The Deep Dye Drop assay provides detailed readouts on the cell cycle in addition to the live/dead counts. Cells are treated with Hoechst (to stain nuclei) and LIVE/DEAD Red (LDR, to stain dead cells) dyes in multi-well plates. In addition, cells can be treated with EdU (to stain S-phase cells) and an antibody for phospho-histone H3 (to stain M-phase cells). Live cells are assigned to a phase of the cell cycle based on their DNA content, and EdU and pH3 intensities. Subsequently, cells are imaged at the single-cell level. The *cell_cycle_gating* script described in this documentation takes the resultant single cell imaging data as input and automatically classifies cells into live or dead and further classifies live cells into G1, G2, S, or M phase of the cell cycle.

CHAPTER 1

License and funding

Deep Dye Drop's automated gating package is currenlty available under the MIT license .The package was developed with funding from U54 grant HL127365, "The Library of Integrated Network-Based Cellular Signatures" under the NIH Common Fund program, and NCI U54 grant CA225088 for the Harvard Medical School (HMS) Center for Cancer Systems Pharmacology (CCSP).

CHAPTER 2

Installation

Deep Dye Drop's automated gating script requires Python 3.

In order to install, *cd* into a local folder of your chocice using the command line. As shown below, clone the repository to the local folder and use pip to install the cell cycle gating scripts.

```
git clone https://github.com/datarail/DrugResponse.git
pip install -e DrugResponse/python
```

CHAPTER 3

Getting started

3.1 Working with data generated from Columbus

- Object level data for a given well is saved as a `.txt` file. The data for all wells in a given plate is saved in a folder. The name of the folder typically takes the form `abcdef[123]`. The `abcdef` prefix in the folder name should correspond to the barcode assigned to the plate.
- `cd` into the directory that contains the object level data folders.
- Start a Jupyter notebook or Ipython session and execute the lines of code below:

```
from cell_cycle_gating import run_cell_cycle_gating as rccg

obj_folder = 'abcdef[123]' # Name of object level folder

# Map user defined channel names to standarized names required by the script
ndict = {'Nuclei Selected - EdUINT': 'edu',
         'Nuclei Selected - DNAcontent': 'dna',
         'Nuclei Selected - LDRTXT SER Spot 8 px' : 'ldr',
         'Nuclei Selected - pH3INT': 'ph3'}

# Run gating script
dfs = rccg.run(obj_folder, ndict)
```

- The dataframe `dfs` returns well-level summary of number of live/dead cells and fraction of cells in each phase of the cell cycle.
- The script saves a pdf showing the gating on each DNA v EDU scatter plot for review. By default the pdf file uses the name of the folder as the file name *i.e* `summary_abcdef[123].pdf`
- The dataframe df is also saved as a .csv file with the same name as the object level folder *i.e* `summary_abdef[123].csv`

3.2 Working with data generated from IXM

- The standard column names generated in Metaexpress differ from the Operetta. Please update `ndict` as below and add an additional argument to `rccg.run()`. Further, the entire dataset for a given plate is saved as a single .txt file instead of seperate files per well in a folder. See below:

```
from cell_cycle_gating import run_cell_cycle_gating as rccg

obj_file = 'filename.txt' # Name of object level file


# Map user defined channel names to standarized names required by the script
ndict = {'Well Name' : 'well',
         'Cell: EdUrawINT (DDD-bckgrnd)' : 'edu',
         'Cell: LDRrawINT (DDD-bckgrnd)' : 'ldr',
         'Cell: DNAcontent (DDD-bckgrnd)' : 'dna'}

# Run gating script
dfs = rccg.run(obj_file, ndict, system='ixm', header=7)
```

3.3 Merging metadata information to output

If you have well level metadata that maps each well to sample conditions, then the above code block can be modified as follows:

```
# Load metadata file
import pandas as pd
dfm = pd.read_csv('metadata.csv')

# Run gating script, this time passing dfm as an additional argument.
dfs = rccg.run(obj, ndict, dfm)
```

Note that the metadata file should contain the following header columns:

- `barcode`, `well`, `cell_line`, `agent`, `concentration`.
- Fields in the `barcode` column should match the prefix in the folder name. *i.e* `abdcdef`

3.4 Not really Deep Dye Drop

By default, the gating code expects that you have all 4 channels *i.e* DNA, EdU, LDR, pH3. However, if you do not have LDR and/or pH3 channels, modify the main line of the code as shown below:

```
dfs = rccg.run(obj, ndict, dfm,
               ph3_channel=False, # If no pH3 channel
               ldr_channel=False # If no LDR channel
              )
```

3.5 Gating corrections using control wells

Automated gating does not always work well, in which case you can apply the automated gating from control wells for a given cell line across corresponding treatment wells. In the first line of code below, by setting `control_based_gating=True`, automated gating is run only on the control plates. The results are saved in .csv and .pdf files with the prefix `control_summary_`. The function also returns a second dataframe `dfg` that contains information on the gates in the control wells. In the second line, the script is run a second time with the arguement `control_gates=dfg` so that errors in automated gating are corrected based on control gating.

```
dfs, dfg = rccg.run(obj, ndict, dfm,
                    control_based_gating=True)
dfs2 = rccg.run(obj, ndict, dfm, control_gates=dfg)
```

If you want to manually adjust gates across all wells, you can provide a list of fudge factors *i.e.* by what magnitude and in which direction you want to change the DNA gates. There are 4 gates you can adjust; G1-left, G1-right, G2-left, and G2-right. For instance, if you want to move G2-left (3rd gate) furrther left by a magnitude of 0.05, set `fudge_gates=[0, 0, -0.05, 0]`. If you want to move G2-right (4th gate) by a magnitue of 0.2 to the right, set `fudge_gates=[0, 0, 0, 0.2]`. In the code example below, we have applied gates from the control but also decided to move the 3rd and 4th gates to the left by 0.05 and 0.1 units in log(DNA) scale.

```
dfs2 = rccg.run(obj, ndict, dfm, control_gates=dfg,
                fudge_gates=[0, 0, -0.05, -0.1])
```

3.6 Additional plotting functions

- To plot cell cycle fractions:

```
import pandas as pd
from cell_cycle_gating import plot_fractions

dfs = pd.read_csv('summary_abcdef[123].csv')
plot_fractions(dfs)
```

CHAPTER 4

Run cell cycle gating (automated)

CHAPTER 5

Manual gating

CHAPTER 6

Cell cycle phases (based on DNA and EdU staining)

CHAPTER 7

Dead cell filter (based on DNA and LDR staining)

CHAPTER 8

pH3 filter

cell_cycle_gating.ph3_filter.**compute_log_ph3**(*ph3*, *x_ph3=None*)

Compute log of pH3 intensities

Parameters

- **ph3** (*1d array*) – ph3 intensities across all cells in a well
- **x_ph3** (*1d array*) – uniformly distributed 1d grid based on expected range of pH3 intensities

Returns **log_ph3** – log pf pH3 intensities across all cells in a well

Return type 1d array

cell_cycle_gating.ph3_filter.**evaluate_Mphase**(*log_ph3*, *ph3_cutoff*, *cell_identity*, *ax=None*)

Reassigns membership of each cell based on M phase identified by pH3

Parameters

- **log_ph3** (*1d array*) – log of ph3 intensities across all cells in a well
- **ph3_cutoff** (*1d array*) – pH3 gating on kernel density minima
- **cell_identity** (*1d array*) – membership of each cell in cell cycle phase (1=G1, 2=S, 3=G2)
- **ax** (*subplot object*) – relative positional reference of subplot in master summary plot

Returns **fractions** – keys are cell cycle phases (G1, G2, S, M) and values are fractions of cells in each phase

Return type dict

cell_cycle_gating.ph3_filter.**get_ph3_gates**(*ph3*, *cell_identity*, *x_ph3=None*, *ph3_cutoff=None*)

Gating based on pH3 intensities

Parameters

- **ph3** (*1d array*) – ph3 intensities across all cells in a well

- **cell_identitity** (*1d array*) – membership of each cell in cell cycle phase (1=G1, 2=S, 3=G2)
- **x_ph3** (*1d array*) – uniformly distributed 1d grid based on expected range of pH3 intensities
- **ph3_cutoff** (*Optional[numpy float]*) – User defined pH3 gating

Returns

- **f_ph3** (*1d array*) – kernel density estimate of pH3 distribution
- **ph3_cutoff** (*numpy float*) – pH3 gating on kernel density minima
- **ph3_lims** (*list of floats*) – bounds on pH3 intensities used as x_lim for plots

cell_cycle_gating.ph3_filter.**plot_summary**(*ph3*, *cell_identity*, *x_ph3=None*, *ph3_cutoff=None*, *well=None*)

Summary plot of pH3 based gating

Parameters

- **ph3** (*1d array*) – ph3 intensities across all cells in a well
- **cell_identity** (*1d array*) – membership of each cell in cell cycle phase (1=G1, 2=S, 3=G2)
- **x_ph3** (*1d array*) – uniformly distributed 1d grid based on expected range of pH3 intensities
- **ph3_cutoff** (*1d array*) – (optional) USER defined pH3 gating
- **well** (*str*) – name of well on 96/384 well plate

Returns **fractions** – keys are cell cycle phases (G1, G2, S, M) and values are fractions of cells in each phase

Return type dict

CHAPTER 9

Peak identification

cell_cycle_gating.findpeaks.**findpeaks**(*signal*, *npeaks=None*, *thresh=0.25*)

Returns the amplitude , location and half-prominence width of peaks from the input signal

Parameters

- **signal** (*list*) –
- **npeaks** (*int*) – number of peaks, locations and width returned sorted from highest to lowes amplitude peaks
- **thresh** (*Optional[float]*) – threshold below which secondary peaks will not be reported. Default is 0.25

Returns

- **peak_amp** (*array of float*) – amplitude of peaks
- **peak_loc** (*array of float*) – location of peaks
- **width** (*array of float*) – list of widths at half-prominence

cell_cycle_gating.findpeaks.**get_prominence_reference_level**(*signal*, *peak*, *peak_loc*)

Returns the amplitude and location of the lower reference level of a peaks prominence. Note that prominence is the the length from the reference level upto the peak

Parameters

- **signal** (*1D-array*) –
- **peak** (*float*) – amplitude of peak whose prominece is to be computed
- **peak_loc** (*float*) – location on X-axis of peak whose prominence is to be computed

Returns

- **reference_loc** (*float*) – location of X-axis of peak whose prominence is to be computed. Should equal peak_loc
- **reference_level** (*float*) – lower reference level of peak prominence.

cell_cycle_gating.findpeaks.**get_width_half_prominence**(*signal*, *peak*, *peak_loc*)

CHAPTER 10

Plotting functions

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index