betabinomial

M. Hasan Celik

Jan 11, 2023

FEATURE OVERVIEW

CHAPTER

ONE

QUICK START

CHAPTER

TWO

BETABINOMIAL

Implementation of Beta-Binomial (https://en.wikipedia.org/wiki/Beta-binomial_distribution) in python for parameters inference with moment method estimation and statistical testing on count data.

Documentation

2.1 Installation

```
pip install betabinomial
```

2.2 Example

```
import numpy as np
from betabinomial import BetaBinomial, pval_adj


bb = BetaBinomial()

# total counts
n = np.array([
  [5, 2, 5, 6, 6],
  [8, 8, 0, 9, 1],
  [8, 2, 6, 1, 7]
])
# event count
k = np.array([
  [3, 1, 4, 1, 2],
  [8, 7, 0, 9, 1],
  [0, 0, 0, 0, 2]
])

# Infer `alpha` and `beta` parameters from counts
bb.infer(k, n)
```

(continues on next page)

(continued from previous page)

```
bb.alpha
# [[ 11.45811965]
#  [121.01628682]
#  [0.43620744]]

bb.beta
# [[13.332114  ]
#  [ 4.97492014]
#  [ 5.41047636]]

# Statistical testing with inferred `alpha` and `beta`
pval = bb.pval(k, n, alternative='two-sided')
# array([[0.33287737, 0.44653957, 0.06266123, 0.35378069, 0.85568061],
#        [0.        , 0.53825136, 0.        , 0.        , 0.        ],
#       [0.67209923, 0.26713023, 0.57287758, 0.14921533, 0.10535054]])

# Adjust p-value with multiple testing correction
padj = pval_adj(pval)
# array([[0.53067103, 0.60891759, 0.18798369, 0.53067103, 0.85568061],
#        [0.        , 0.6610126 , 0.        , 0.        , 0.        ],
#        [0.72010631, 0.50086919, 0.6610126 , 0.31974714, 0.26337634]])
```

2.3 Citation

If you use this package in academic publication, please cite:

```
@article{celik2022analysis,
  title={Analysis of alternative polyadenylation from long-read or short-read RNA-seq␣
↪with LAPA},
  author={Celik, Muhammed Hasan and Mortazavi, Ali},
  journal={bioRxiv},
  year={2022},
  publisher={Cold Spring Harbor Laboratory}
}
```

CHAPTER

THREE

EXAMPLES

[1]:
```
import numpy as np
from scipy.stats import betabinom, beta as beta_dist
```

Let's create example count data with following shape

[2]:
```
size = (2000, 500)
```

Ground truth `alpha` and `beta` parameters to sample count data

[3]:
```
alpha_true = np.random.random((size[0], 1)) * 10
beta_true = 10 - alpha_true
```

Sampling counts (`k`) and total counts (`n`) with ground truth

[4]:
```
n = (np.random.random(size) * 2000).astype('int')
k = betabinom.rvs(n, alpha_true * np.ones(size),  beta_true * np.ones(size))
```

[5]:
```
n, k
```

[5]:
```
(array([[ 643,  402,  736, ..., 1730, 1362,  236],
        [1261, 1908, 1173, ...,  859,  909,  475],
        [ 966, 1060, 1435, ...,   10,  582, 1693],
        ...,
        [ 978, 1388,  261, ...,    9, 1045,  214],
        [ 715, 1987, 1379, ..., 1367,   59, 1000],
        [ 946,  287, 1368, ...,  492,  451, 1277]]),
 array([[ 249,  197,  276, ..., 1553,  503,  118],
        [ 196,  527,  130, ...,    9,  103,  179],
        [ 539,  472,  816, ...,    6,  347,  735],
        ...,
        [ 306,  964,  128, ...,    5,  601,  150],
        [ 655, 1922, 1171, ..., 1293,   45,  810],
        [ 583,  178,  248, ...,   64,  195,  382]]))
```

[6]:
```
from betabinomial import BetaBinomial
```

[7]:
```
bb = BetaBinomial()
```

Inference of `alpha` and `beta` parameters from count data

[8]:
```
bb.infer(k, n)
```

```
 42%|     | 423/1000 [00:30<00:41, 13.79it/s]
```

[8]:
```
BetaBinomial[2000]
```

Infered `alpha` and `beta`

[9]:
```
bb.alpha, bb.beta
```

[9]:
```
(array([[5.90111233],
        [1.17191851],
        [6.27580867],
        ...,
        [6.87969866],
        [8.99132085],
        [3.20719648]]),
 array([[4.92094763],
        [8.41647732],
        [3.85951562],
        ...,
        [3.69910629],
        [0.87364963],
        [7.37440814]]))
```

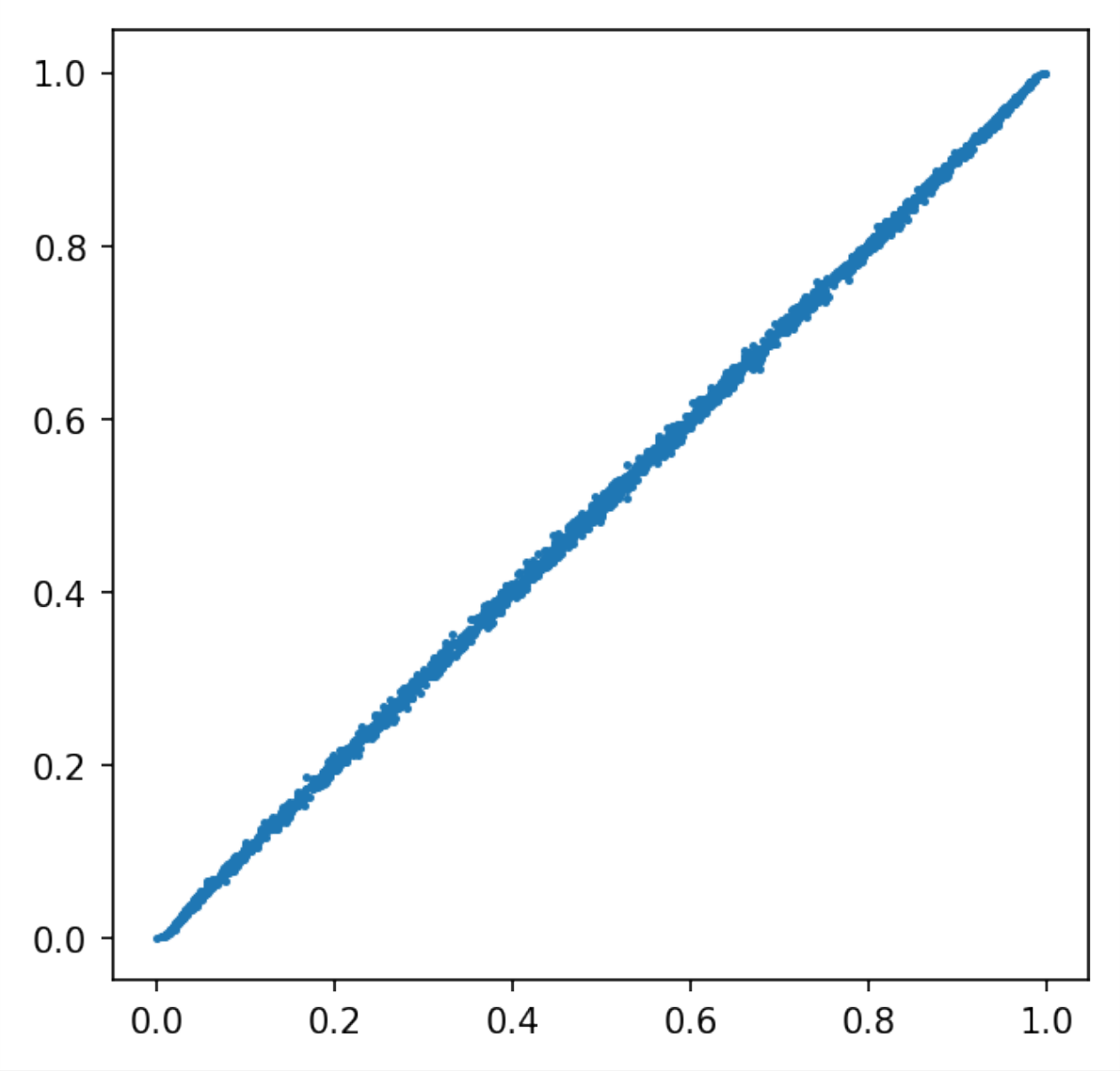
[10]:
```
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (5, 5)
plt.rcParams["figure.dpi"] = 150
```

Plot true beta distribution mean `(alpha / (alpha + beta))` against inferred

[11]:
```
true_beta_mean = alpha_true / (alpha_true + beta_true)
plt.scatter(bb.beta_mean(), true_beta_mean, s=2)
```

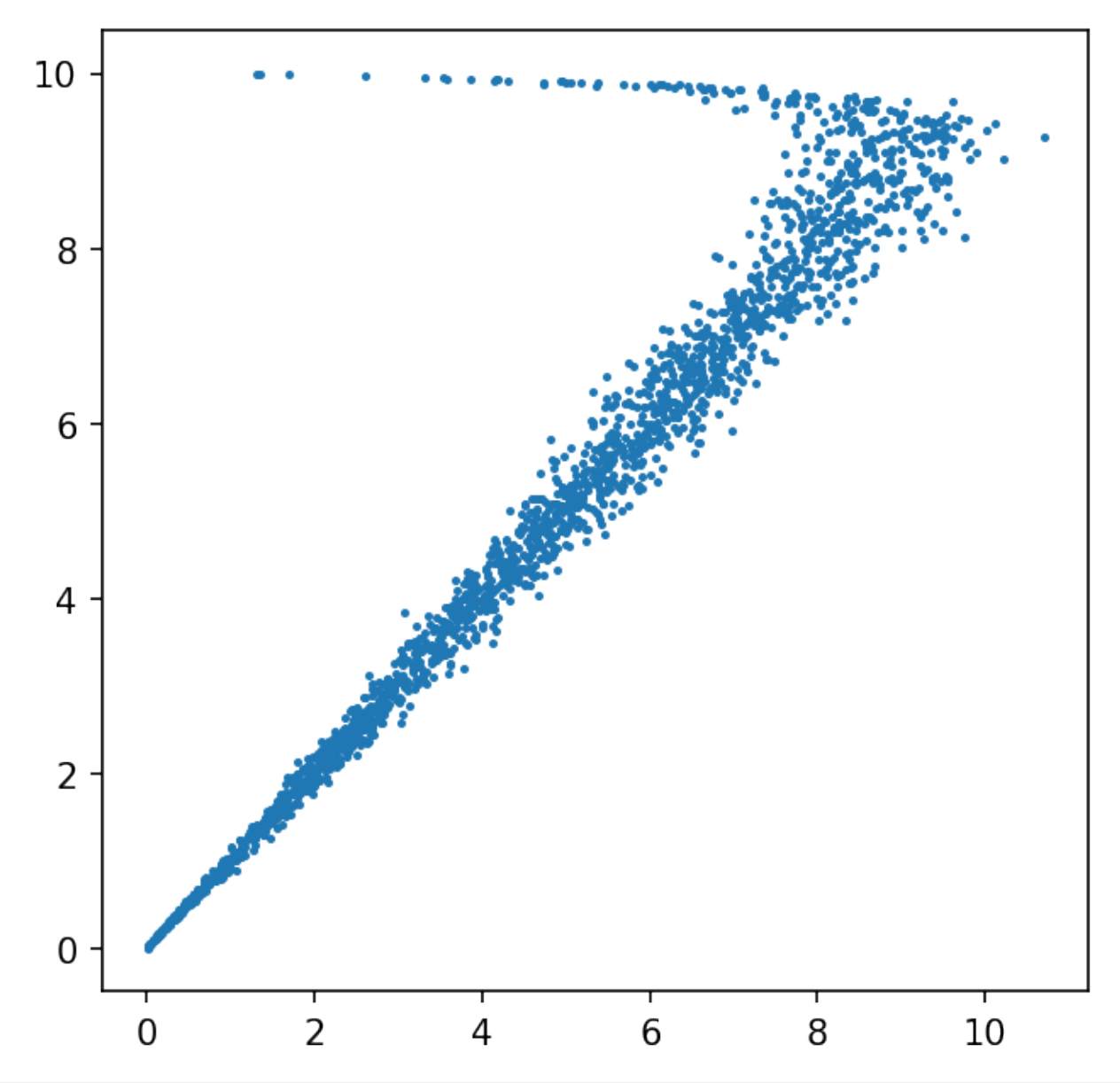
[11]:
```
<matplotlib.collections.PathCollection at 0x7f6e844737f0>
```



Plot true and predicted `alpha` and `beta` values

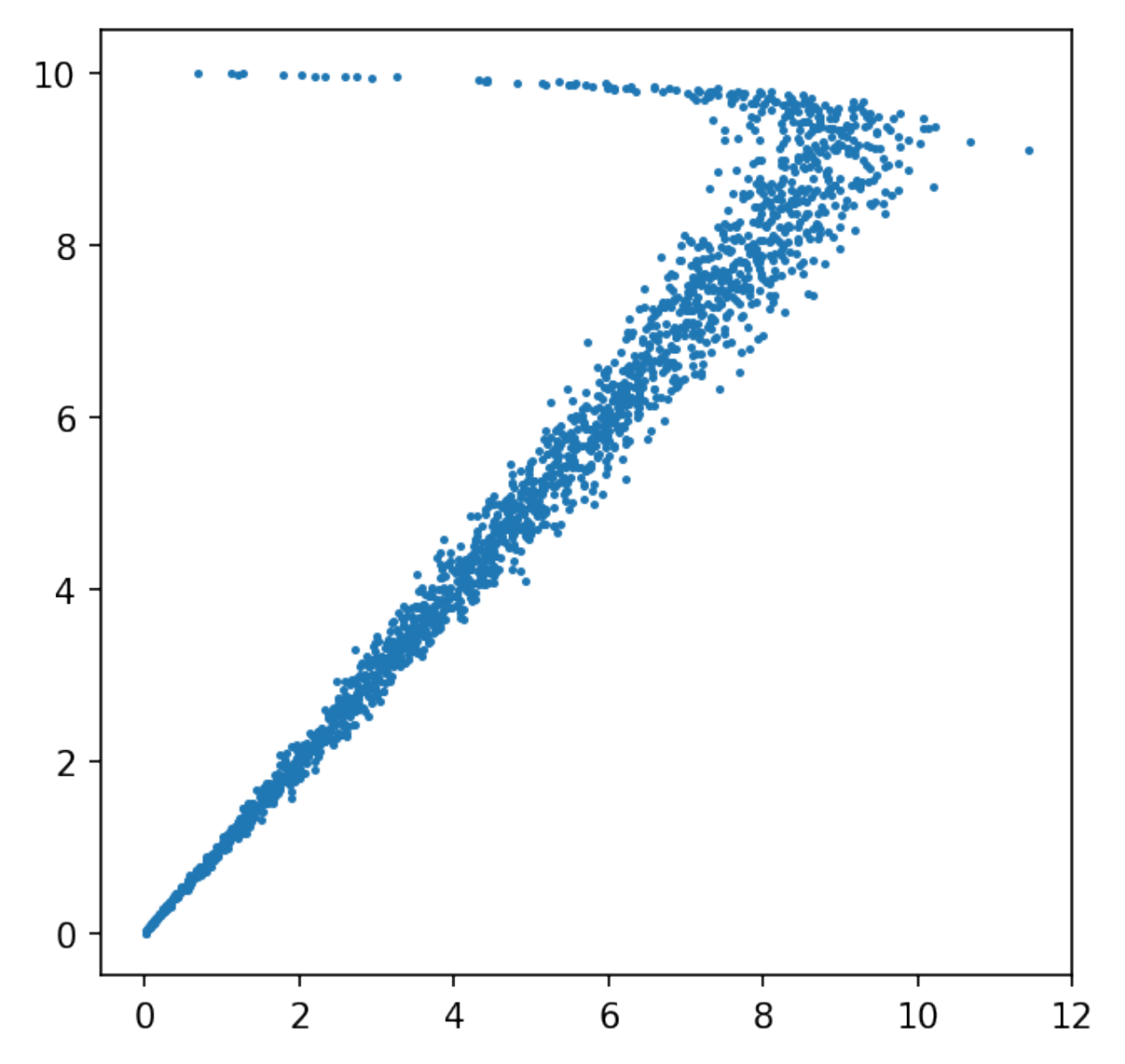
[12]:
```
plt.scatter(bb.alpha, alpha_true, s=2)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x7f6e84372eb8>
```



[13]:
```
plt.scatter(bb.beta, beta_true, s=2)
```

[13]:
```
<matplotlib.collections.PathCollection at 0x7f6e8435c320>
```



Plot a example distribution and true and predicted beta distribution

[14]:
```
row = np.argwhere((true_beta_mean < 0.8) & (true_beta_mean > 0.2))[0][0]

x = np.linspace(
    beta_dist.ppf(0.001, alpha_true[row], beta_true[row]),
    beta_dist.ppf(0.999, alpha_true[row], beta_true[row])
)
plt.plot(x, beta_dist.pdf(x, alpha_true[row], beta_true[row]), color='red', linewidth=3)

x = np.linspace(
    beta_dist.ppf(0.001, bb.alpha[row], bb.beta[row]),
    beta_dist.ppf(0.999, bb.alpha[row], bb.beta[row])
)
plt.hist((k / n)[row, :], bins=500)
```

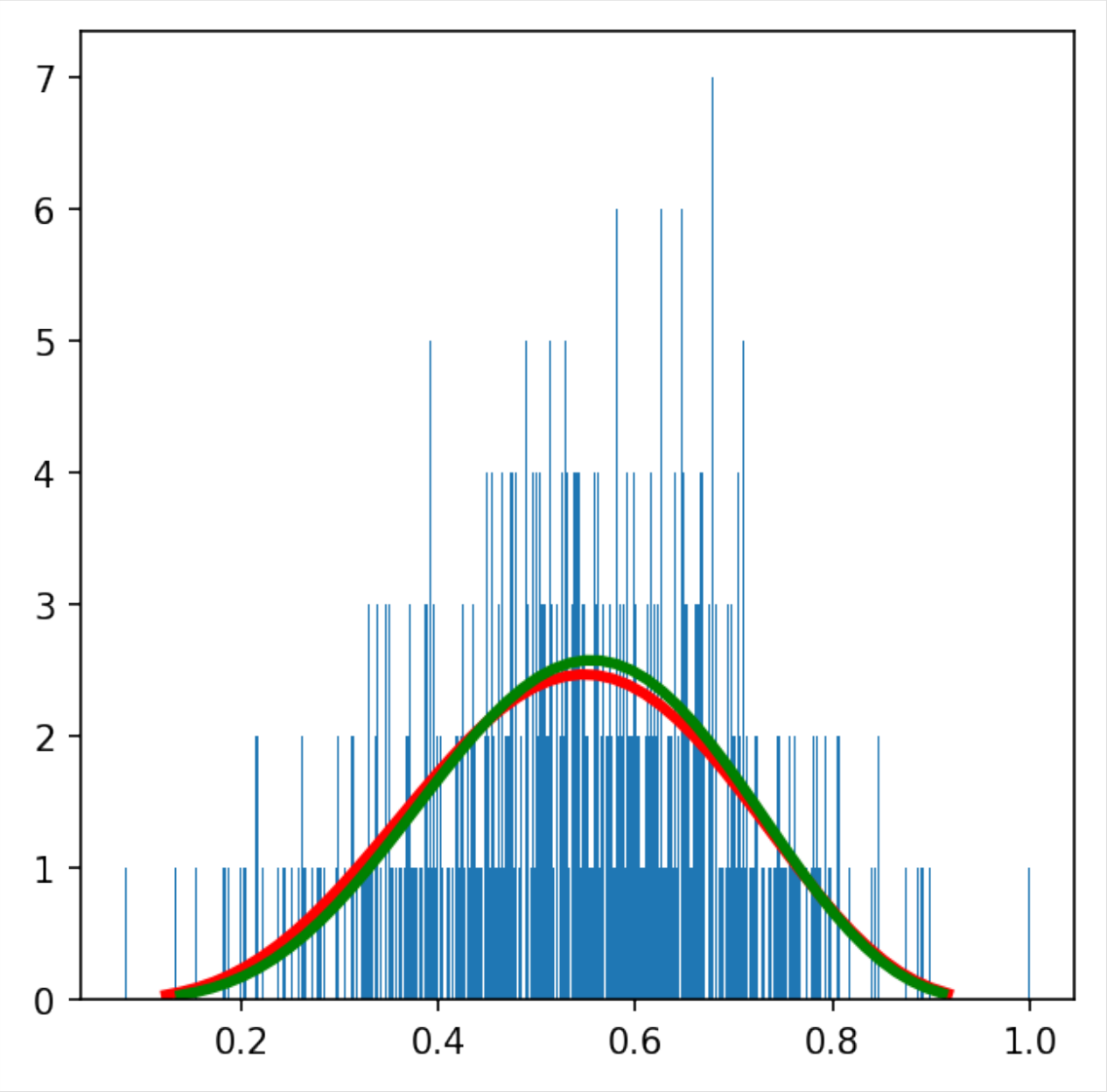
(continues on next page)

(continued from previous page)

```
plt.plot(x, beta_dist.pdf(x, bb.alpha[row], bb.beta[row]), color='green', linewidth=3)
```

```
/home/cs/anaconda3/envs/betabinomial/lib/python3.6/site-packages/ipykernel_launcher.py:
↪13: RuntimeWarning: invalid value encountered in true_divide
  del sys.path[0]
```

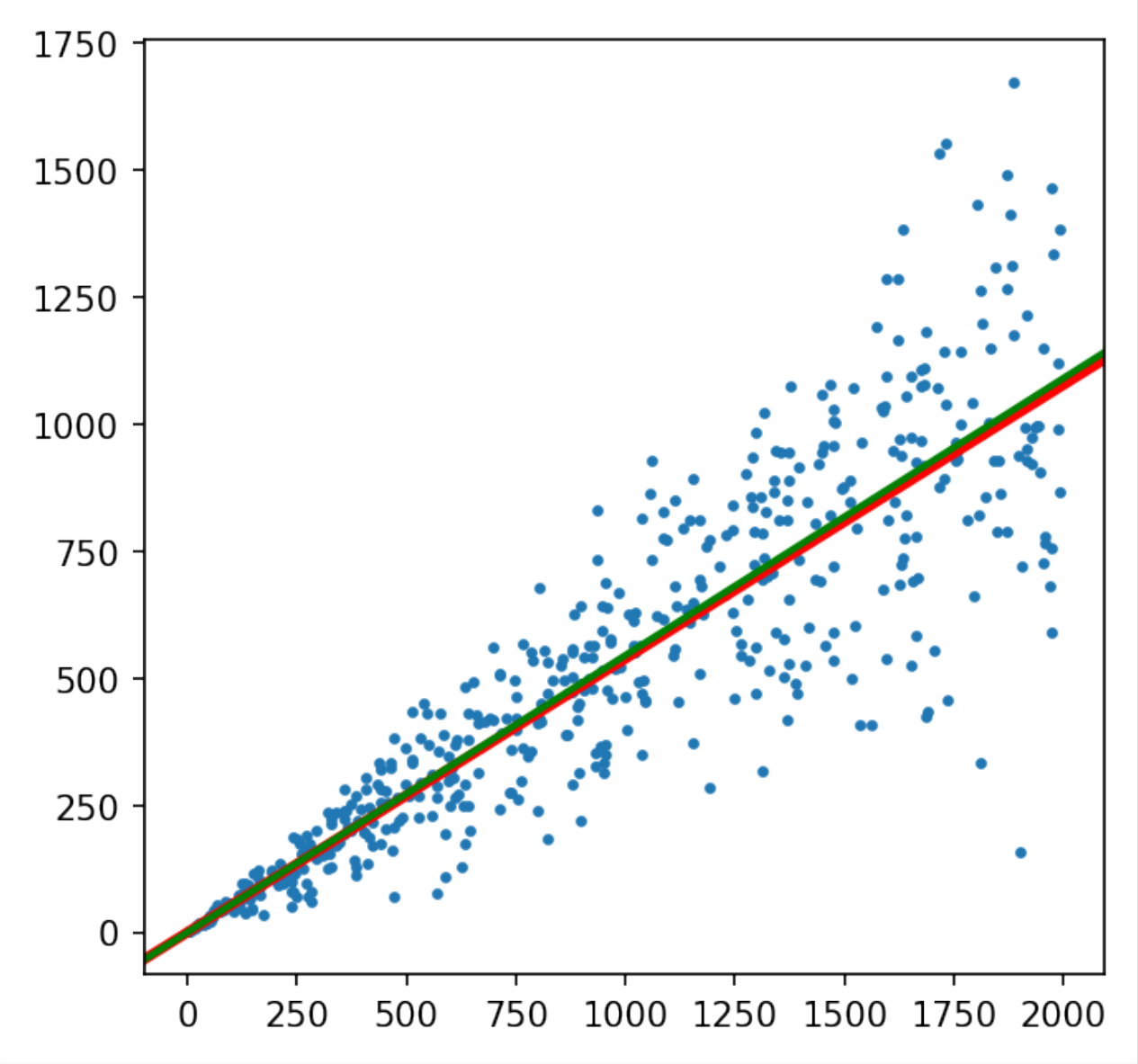
[14]:
```
[<matplotlib.lines.Line2D at 0x7f6e84320a20>]
```



Plot an example count data and predicted and true beta mean.

[15]:
```
plt.scatter(n[row, :], k[row, :], s=5)
plt.axline((0, 0), slope=alpha_true[row] / (alpha_true[row] + beta_true[row]), color='red
↪', linewidth=3)
plt.axline((0, 0), slope=bb.beta_mean()[row], color='green', linewidth=2)
```

[15]:
```
<matplotlib.lines._AxLine at 0x7f6e77c4a940>
```



Perform beta binomial test and return p-values

[16]:
```
%%time
pval = bb.pval(k, n, alternative='two-sided') # 'less', 'greater'
```

```
CPU times: user 1min 13s, sys: 132 ms, total: 1min 13s
Wall time: 1min 13s
```

[17]:
```
pval
```

[17]:
```
array([[0.3024323 , 0.71783721, 0.26493792, ..., 0.0040517 , 0.24639214,
        0.77164253],
       [0.5916317 , 0.17327847, 0.88024622, ..., 0.117335  , 0.86065672,
        0.05210928],
```

(continues on next page)

(continued from previous page)

```
       [0.66576406, 0.25957367, 0.71324979, ..., 0.93432651, 0.84825939,
        0.23008366],
       ...,
       [0.02469224, 0.81291417, 0.29115387, ..., 0.80582631, 0.58397361,
        0.77351307],
       [0.79828567, 0.63339124, 0.38594751, ..., 0.9104928 , 0.17621192,
        0.24878696],
       [0.03657361, 0.0358853 , 0.40090043, ..., 0.18886599, 0.35276397,
        0.94860864]])
```

Multiple testing correction for p-values

[18]:
```
from betabinomial import pval_adj

padj = pval_adj(pval)
```

[19]:
```
padj
```

[19]:
```
array([[0.98238656, 0.99985032, 0.97738457, ..., 0.14635016, 0.97299313,
        0.99988192],
       [0.9991514 , 0.94910965, 0.99988192, ..., 0.90040026, 0.99988192,
        0.75038205],
       [0.99983112, 0.97607204, 0.99985032, ..., 0.99988192, 0.99988192,
        0.96874961],
       ...,
       [0.54627678, 0.99988192, 0.98155834, ..., 0.99988192, 0.99883718,
        0.99988192],
       [0.99988192, 0.99968944, 0.99022587, ..., 0.99988192, 0.95044065,
        0.97365185],
       [0.65882102, 0.65395819, 0.9915473 , ..., 0.95580413, 0.98791013,
        0.99988192]])
```

log-fold change based on the beta-binomial expectation and measured values:

$$\mu = \frac{n * \alpha}{\alpha + \beta}$$

$$logFC = \log(\frac{k}{\mu})$$

[20]:
```
logfc = bb.logFC(k, n)
```

```
/home/cs/Projects/betabinomial/betabinomial/betabinomial.py:128: RuntimeWarning: invalid␣
↪value encountered in true_divide
  return k / self.mean(n)
/home/cs/Projects/betabinomial/betabinomial/betabinomial.py:131: RuntimeWarning: divide␣
↪by zero encountered in log
  return np.log(self.FC(k, n))
```

[21]:
```
logfc
```

[21]:
```
array([[-0.34224605, -0.10680258, -0.37438348, ...,  0.49851291,
        -0.38967354, -0.0867014 ],
```

(continues on next page)

(continued from previous page)

```
       [ 0.24036577,  0.81530114, -0.09787395, ..., -2.4566329 ,
        -0.07570466,  1.12598245],
       [-0.10412384, -0.32972077, -0.08518135, ..., -0.0315012 ,
        -0.03782124, -0.35506246],
       ...,
       [-0.73164695,  0.06574977, -0.28221253, ..., -0.15750905,
        -0.12289961,  0.07493689],
       [ 0.00508308,  0.05947073, -0.07077013, ...,  0.03707693,
        -0.17814457, -0.11799065],
       [ 0.70966451,  0.71602123, -0.51395646, ..., -0.84587574,
         0.35525211, -0.01312835]])
```

z-score based on the beta-binomial mean and variance and measured values:

$$\mu = \frac{n * \alpha}{\alpha + \beta}$$

$$\rho = \frac{1}{1 + \alpha + \beta}$$

$$\sigma^2 = n * \mu * (1 - \mu) * (1 + (n - 1) * \rho)$$

$$z_{score} = \frac{k - \mu}{\sigma}$$

[22]:
```
zscore = bb.z_score(k, n)
```

```
/home/cs/Projects/betabinomial/betabinomial/betabinomial.py:155: RuntimeWarning: invalid␣
↪value encountered in true_divide
  return (k - self.mean(n)) / np.sqrt(self.variance(n))
```

[23]:
```
zscore
```

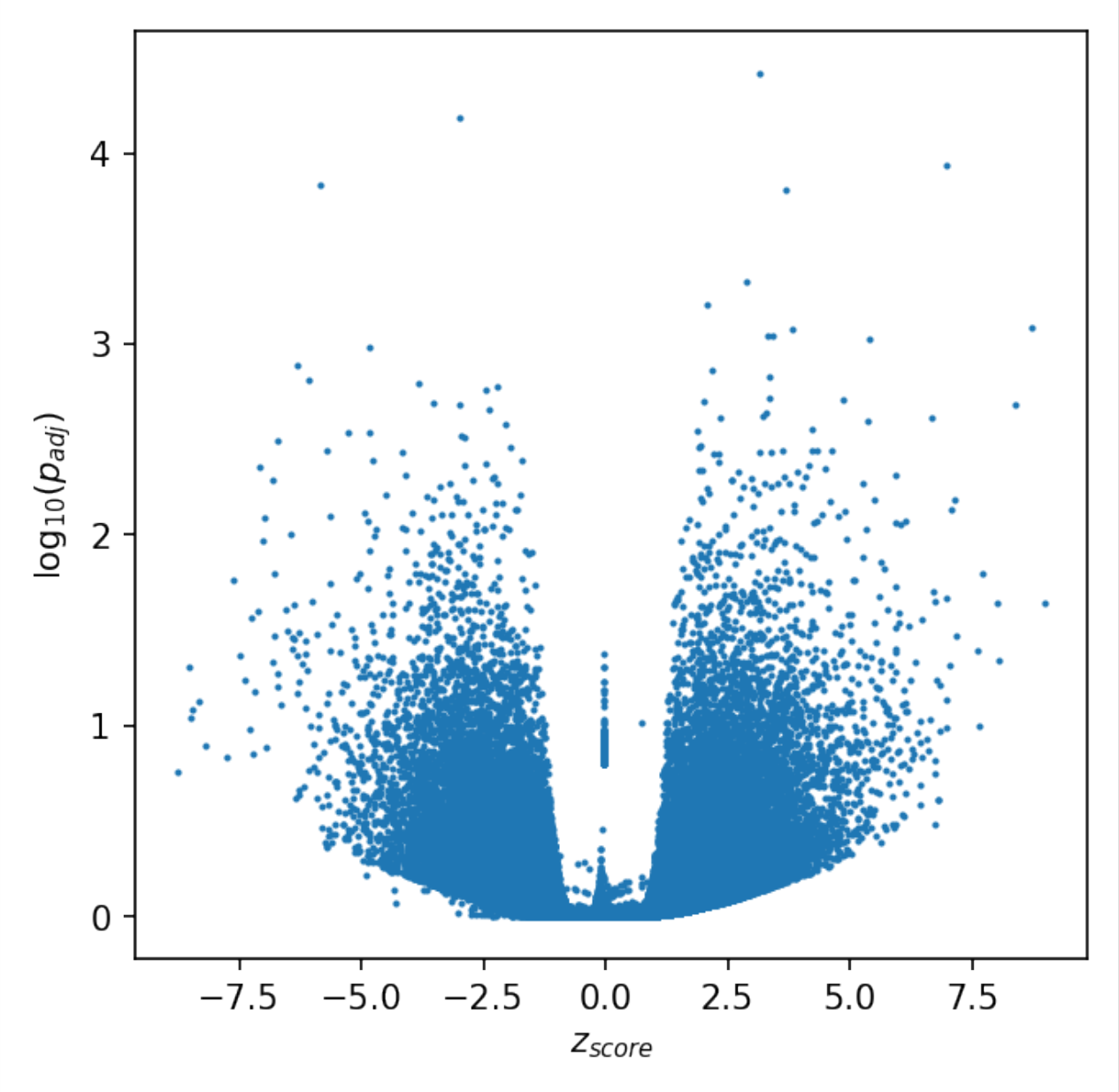
[23]:
```
array([[-1.08218878, -0.37637173, -1.16727504, ...,  2.42577144,
        -1.21031726, -0.3057654 ],
       [ 0.32867455,  1.52591747, -0.11275043, ..., -1.103992  ,
        -0.08806581,  2.5043738 ],
       [-0.41859068, -1.18950786, -0.3462346 , ..., -0.09299191,
        -0.15657404, -1.26796308],
       ...,
       [-2.39498205,  0.3141733 , -1.11860139, ..., -0.45850918,
        -0.53397077,  0.35250097],
       [ 0.0535196 ,  0.64634314, -0.71991724, ...,  0.39799326,
        -1.59715588, -1.17112662],
       [ 2.30620724,  2.30604331, -0.89846667, ..., -1.26753798,
         0.94625376, -0.02915105]])
```

Plot volcona plot with `np.log10(padj)` and `zscore`

[30]:
```
plt.scatter(zscore.ravel(), -np.log10(padj.ravel()), s=1)
plt.xlabel('$z_{score}$')
plt.ylabel('$\log_{10}(p_{adj})$')
```

```
/home/cs/anaconda3/envs/betabinomial/lib/python3.6/site-packages/ipykernel_launcher.py:1:
↪ RuntimeWarning: divide by zero encountered in log10
  """Entry point for launching an IPython kernel.
```

[30]:
```
Text(0, 0.5, '$\\log_{10}(p_{adj})$')
```

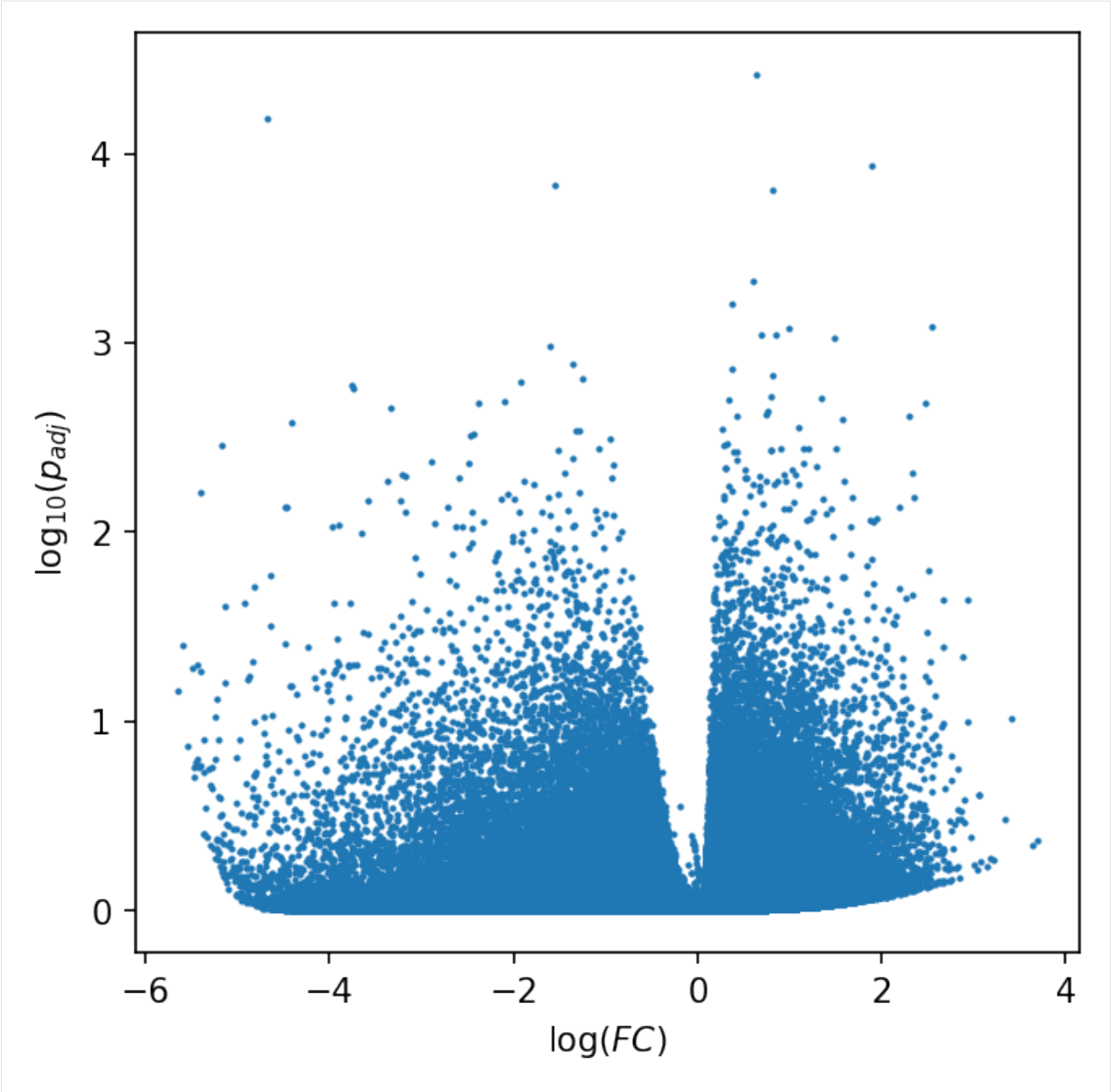


Plot volcona plot with `np.log10(padj)` and `logFC`

[29]:
```
plt.scatter(logfc.ravel(), -np.log10(padj.ravel()), s=1)
plt.xlabel('$\log(FC)$')
plt.ylabel('$\log_{10}(p_{adj})$')
```

```
/home/cs/anaconda3/envs/betabinomial/lib/python3.6/site-packages/ipykernel_launcher.py:1:
↪ RuntimeWarning: divide by zero encountered in log10
  """Entry point for launching an IPython kernel.
```

[29]:
```
Text(0, 0.5, '$\\log_{10}(p_{adj})$')
```


4
3
2
1
0
$\log_{10}(p_{adj})$
−6
−4
−2
0
2
4
log(FC)

CHAPTER

FOUR

API REFERENCE

This page contains auto-generated API reference documentation[1].

4.1 betabinomial

4.1.1 Submodules

betabinomial.betabinomial

Module Contents

Classes

BetaBinomial	Beta-binomial distribution to perform statistical testing on count data.

Functions

pval_adj(pval[, method, alpha])	Multiple testing correction for p-value matrix obtained

class betabinomial.betabinomial.**BetaBinomial**(*alpha=None*, *beta=None*)

Beta-binomial distribution to perform statistical testing on count data.

Parameters

- **alpha** (np.ndarray, optional) – *alpha* parameter as column vector of beta-binomial. *alpha* parameter can be learned with *infer* function. Defaults to None.
- **beta** (np.ndarray, optional) – *beta* parameter as column vector of beta-binomial. *beta* parameter can be learned with *infer* function. Defaults to None.

alpha

alpha parameter as column vector of beta-binomial. *alpha* parameter can be learned with *infer* function. Defaults to None.

Type
np.ndarray

[1] Created with sphinx-autoapi

beta

beta parameter as column vector of beta-binomial. *beta* parameter can be learned with *infer* function. Defaults to None.

Type
np.ndarray

Examples

Initilize with alpha and beta vector

```
>>> BetaBinomial(
>>>     alpha=np.array([[1.], [2.], [3.]])
>>>     beta=np.array([[0.5], [0.1], [2]])
>>> )
BetaBinomial[3]
```

Examples

Initilize with single alpha and beta values

```
>>> BetaBinomial(
>>>     alpha=np.array([[1.]])
>>>     beta=np.array([[1]])
>>> )
BetaBinomial[1]
```

Examples

Initilize without alpha and beta

```
>>> BetaBinomial()
BetaBinomial[]
```

infer(*k*, *n*, *theta=0.001*, *max_iter=1000*)

Infer alpha and beta parameters of beta-binomial from k and n counts.

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.
- **theta** (float, optional) – Error between iterations to stop inference.
- **max_iter** – Maximum number of iterations.

_update(*k*, *n*, *alpha_old*, *beta_old*)

_convergence(*alpha_old*, *alpha*, *beta_old*, *beta*, *theta*)

beta_mean()

The mean of beta distrubution = *alpha / (alpha+beta)*

mean(*n*)

The expected number of k *E[k] = n * alpha / (alpha+beta)*

Parameters
n (*np.ndarray*) – total number of counts events.

fold_change(*k*, *n*)

Fold change between observed k and E[k]

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.

log_fc(*k*, *n*)

Log-fold change between observed k and E[k]

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.

cdf(*k*, *n*)

CDF of beta-binomial distribution with given *k* and *n* and inferred *alpha* and *beta* parameters.

pval(*k*, *n*, *alternative='two-sided'*)

Statistical testing with beta-binomial based on given

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.
- **alternative** – {'two-sided', 'less', 'greater'}

z_score(*k*, *n*)

z-score based on the *k* and *n* and inferred *alpha* and *beta* parameters.

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.

intra_class_corr()

Intra or inter class corrections.

variance(*n*)

Variance of beta-binomial distribution.

Parameters
n (*np.ndarray*) – total number of counts events.

__repr__()

Return repr(self).

betabinomial.betabinomial.**pval_adj**(*pval*, *method='fdr_bh'*, *alpha=0.05*)

Multiple testing correction for p-value matrix obtained from *BetaBinomial.pval*

Parameters

- **pval** (*np.ndarray*) – matrix of p-values.

- **method** (*str*) – Multiple correction method defined based on *statsmodels.stats.multitest.multipletests*.

4.1.2 Package Contents

Classes

BetaBinomial	Beta-binomial distribution to perform statistical testing on count data.

Functions

pval_adj(pval[, method, alpha])	Multiple testing correction for p-value matrix obtained

class betabinomial.**BetaBinomial**(*alpha=None*, *beta=None*)

Beta-binomial distribution to perform statistical testing on count data.

Parameters

- **alpha** (np.ndarray, optional) – *alpha* parameter as column vector of beta-binomial. *alpha* parameter can be learned with *infer* function. Defaults to None.
- **beta** (np.ndarray, optional) – *beta* parameter as column vector of beta-binomial. *beta* parameter can be learned with *infer* function. Defaults to None.

alpha

alpha parameter as column vector of beta-binomial. *alpha* parameter can be learned with *infer* function. Defaults to None.

Type
np.ndarray

beta

beta parameter as column vector of beta-binomial. *beta* parameter can be learned with *infer* function. Defaults to None.

Type
np.ndarray

Examples

Initilize with alpha and beta vector

```
>>> BetaBinomial(
>>>     alpha=np.array([[1.], [2.], [3.]])
>>>     beta=np.array([[0.5], [0.1], [2]])
>>> )
BetaBinomial[3]
```

Examples

Initilize with single alpha and beta values

```
>>> BetaBinomial(
>>>     alpha=np.array([[1.]])
>>>     beta=np.array([[1]])
>>> )
BetaBinomial[1]
```

Examples

Initilize without alpha and beta

```
>>> BetaBinomial()
BetaBinomial[]
```

infer(*k*, *n*, *theta=0.001*, *max_iter=1000*)

Infer alpha and beta parameters of beta-binomial from k and n counts.

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.
- **theta** (float, optional) – Error between iterations to stop inference.
- **max_iter** – Maximum number of iterations.

_update(*k*, *n*, *alpha_old*, *beta_old*)

_convergence(*alpha_old*, *alpha*, *beta_old*, *beta*, *theta*)

beta_mean()

The mean of beta distrubution = *alpha / (alpha+beta)*

mean(*n*)

The expected number of k *E[k] = n * alpha / (alpha+beta)*

Parameters

n (*np.ndarray*) – total number of counts events.

fold_change(*k*, *n*)

Fold change between observed k and E[k]

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.

log_fc(*k*, *n*)

Log-fold change between observed k and E[k]

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.

cdf(*k*, *n*)

CDF of beta-binomial distribution with given *k* and *n* and inferred *alpha* and *beta* parameters.

pval(*k*, *n*, *alternative='two-sided'*)

Statistical testing with beta-binomial based on given

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.
- **alternative** – {'two-sided', 'less', 'greater'}

z_score(*k*, *n*)

z-score based on the *k* and *n* and inferred *alpha* and *beta* parameters.

Parameters

- **k** (*np.ndarray*) – count matrix of observations.
- **n** (*np.ndarray*) – total number of counts events.

intra_class_corr()

Intra or inter class corrections.

variance(*n*)

Variance of beta-binomial distribution.

Parameters

n (*np.ndarray*) – total number of counts events.

__repr__()

Return repr(self).

betabinomial.**pval_adj**(*pval*, *method='fdr_bh'*, *alpha=0.05*)

Multiple testing correction for p-value matrix obtained from *BetaBinomial.pval*

Parameters

- **pval** (*np.ndarray*) – matrix of p-values.
- **method** (*str*) – Multiple correction method defined based on *statsmodels.stats.multitest.multipletests*.

CHAPTER

FIVE

BETABINOMIAL

Implementation of Beta-Binomial (https://en.wikipedia.org/wiki/Beta-binomial_distribution) in python for parameters inference with moment method estimation and statistical testing on count data.

Documentation

5.1 Installation

```
pip install betabinomial
```

5.2 Example

```
import numpy as np
from betabinomial import BetaBinomial, pval_adj


bb = BetaBinomial()

# total counts
n = np.array([
  [5, 2, 5, 6, 6],
  [8, 8, 0, 9, 1],
  [8, 2, 6, 1, 7]
])
# event count
k = np.array([
  [3, 1, 4, 1, 2],
  [8, 7, 0, 9, 1],
  [0, 0, 0, 0, 2]
])

# Infer `alpha` and `beta` parameters from counts
bb.infer(k, n)
```

(continues on next page)

(continued from previous page)

```
bb.alpha
# [[ 11.45811965]
#  [121.01628682]
#  [0.43620744]]

bb.beta
# [[13.332114  ]
#  [ 4.97492014]
#  [ 5.41047636]]

# Statistical testing with inferred `alpha` and `beta`
pval = bb.pval(k, n, alternative='two-sided')
# array([[0.33287737, 0.44653957, 0.06266123, 0.35378069, 0.85568061],
#        [0.        , 0.53825136, 0.        , 0.        , 0.        ],
#       [0.67209923, 0.26713023, 0.57287758, 0.14921533, 0.10535054]])

# Adjust p-value with multiple testing correction
padj = pval_adj(pval)
# array([[0.53067103, 0.60891759, 0.18798369, 0.53067103, 0.85568061],
#        [0.        , 0.6610126 , 0.        , 0.        , 0.        ],
#        [0.72010631, 0.50086919, 0.6610126 , 0.31974714, 0.26337634]])
```

5.3 Citation

If you use this package in academic publication, please cite:

```
@article{celik2022analysis,
  title={Analysis of alternative polyadenylation from long-read or short-read RNA-seq␣
↪with LAPA},
  author={Celik, Muhammed Hasan and Mortazavi, Ali},
  journal={bioRxiv},
  year={2022},
  publisher={Cold Spring Harbor Laboratory}
}
```

CHAPTER

SIX

INDICES AND TABLES

- genindex

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

F

I

L

M

P

V

Z