my-vignette

{r, include = FALSE, message=FALSE} knitr::opts\_chunk$set( collapse = TRUE, comment = "#>" ) devtools::load\_all() x <- system.file("Pictures/const\_batch\_effect1.jpg", package="simulatr") cat(x, file = "C:/Users/ASUS/simulatr/tmp.txt")

## Introduction

We need data; be it data analysis, training, or evaluation of a model. In this package, we provide simulated data according to the need of the users. Our package is easy-to-use and cater to the need of the users. Users can control the number of features/samples, the correlation between features/samples and the amount of noise/bias within the data. We have designed this package keeping biological data in mind. So, this package also provides functions to retrieve data from NCBI.

* [Introduction](#introduction)
* [Installation](#installation)
* [Usage](#usage)
* [Use Cases](#use-cases)
  + [Simulate a dataset](#simulate-a-dataset)
  + [List of the platforms](#list-of-the-platforms)
  + [Retrieve data for a given platform](#retrieve-data-for-a-given-platform)
  + [List of the datasets](#list-of-the-datasets)
* [Importing the package](#importing-the-package)

## Installation

# From Github (development version)  
devtools::install\_github("toscm/simulatr")  
# From CRAN (stable version)  
install.packages("simulatr") # not yet available

## Usage

1. Call list\_platforms() to get the list of all platforms.
2. Choose a platform from the list and get information about the platform using get\_gpl\_data(“GPL95”).
3. Get information about the gse numbers related to the given platform with list\_datasets(platform = “GPL95”).
4. Use the gse numbers to get dataset using get\_dataset(“GSE803”) or simulate\_gse(15,15,“GSE803”). You can use the dataframe returned by these functions as a base\_dataset.
5. Simulate dataset with simulate\_dataset().
6. Control the noise, bias or correlation of the simulated dataset with the functions provided by the package.

## Use Cases

### Simulate a dataset

#### The simplest one

Generates a (5,5) dataframe with random data.

simulatr::simulate\_dataset()  
  
#result:  
  
# X1 X2 X3 X4 X5  
# 1 0.07954922 1.0797440 0.6428168 0.6676199 0.7272287  
# 2 0.15126274 -0.2704434 -0.1258170 -1.2300756 -0.2543250  
# 3 -0.34563341 -0.2696411 0.7153013 0.9804060 1.3672825  
# 4 0.03857418 -0.3278670 0.7207698 -1.3004286 -0.4472933  
# 5 -0.29393465 0.1100270 -1.5629572 -1.7656558 0.9456364

#### With given dimension

The users can define the dimension of the simulated data. n is the number of samples and p is the number of features (e.g. genes).

simulatr::simulate\_dataset(n = 2, p = 3)  
  
# result :  
  
# X1 X2 X3  
# 1 2.5037109 0.7780662 -0.8493910  
# 2 -0.4439368 -0.5588823 0.4590951

#### With given data

The users can provide a dataframe from which they want to derive the simulated data.

simulatr::simulate\_dataset(n = 5,   
 p = 5,   
 base = data.frame(matrix(stats::rnorm(6 \* 6), 6, 6)))  
  
# result :  
  
# X4 X5 X1 X2 X6  
# 5 0.1256898 -0.3160537 0.04232887 1.18364610 0.02548396  
# 3 -0.6574073 0.2382905 1.80930896 -0.80482068 1.40157078  
# 1 1.9728739 -0.5976392 0.14674887 -1.29226331 0.48473688  
# 6 -2.1417302 0.5517731 1.76103325 -0.22870073 0.66565249  
# 4 0.2322724 -2.5704606 0.96798778 0.03665393 2.24897548

#### With given gse

The users can use simulate\_gse function as base. In that case, the users will get simulated data related to that specific gse number.

simulatr::simulate\_dataset(n = 5,   
 p = 5,   
 base = simulatr::simulate\_gse(n = 10, p = 10, "GSE3821"))  
  
# result :  
  
# GSM87671 GSM87665 GSM87669 GSM87674 GSM87667  
# 10484\_at 51.5 93.5 13.8 90.3 125.1  
# 2753\_at 205.3 565.3 31.8 7.4 166.9  
# 3632\_at 32.6 52.5 19.1 44.3 129.5  
# 9506\_at 922.2 289.4 118.0 236.0 0.6  
# 7886\_at 1.0 8.0 24.7 33.3 1.7

#### With given noise

The users can choose the type of noise they want to add to their simulated data. They can either provide their own noise (a n\*p dimensional matrix) or choose from the given noise functions. If they choose the given noise function, they have to provide arguments for the chosen one. We provide noise function with gaussian, poisson, exponential, binomial and uniform distribution.

##### Gaussian/normal noise function

Here argument sd is the standard deviation of the noise.

simulatr::simulate\_dataset(n = 5,   
 p = 5,   
 noise\_func = random\_noise,   
 noise\_func\_args = list(sd = 1))  
  
# result :  
  
# X1 X2 X3 X4 X5  
# 1 -0.69791555 0.65353128 0.1576375 0.8861998 0.08378272  
# 2 0.07894113 4.62759419 -0.5398236 2.5323452 1.95286403  
# 3 0.36148932 -0.03017104 2.6283972 2.4019776 1.61741591  
# 4 1.52561571 1.38624166 1.4993830 0.3052081 1.10903123  
# 5 1.93360652 -2.11722575 3.3801689 0.2635528 3.13714589

##### Poisson noise

Here lambda is the argument of poisson noise.

simulatr::simulate\_dataset(n = 5,   
 p = 5,   
 noise\_func = poisson\_noise,   
 noise\_func\_args = list(lambda = 1))  
  
# result :  
  
# X1 X2 X3 X4 X5  
# 1 2.5808568 0.4168481 1.7403153 -0.3837684 1.3131605  
# 2 0.7283494 2.2950450 0.6518982 2.4346692 -3.1814076  
# 3 0.9899551 1.0138370 -0.3008775 1.2168060 0.8645686  
# 4 -0.1871961 1.8423962 0.1944272 0.8158043 2.1499120  
# 5 -1.0280963 2.6924890 0.3397274 0.9826114 -0.2875561

##### Uniform noise

Here min and max are the function arguments of uniform noise. min and max define the range of the noise.

simulatr::simulate\_dataset(n = 5,   
 p = 5,   
 noise\_func = uniform\_noise,   
 noise\_func\_args = list(min = 1, max = 2))  
  
# result :  
  
# X1 X2 X3 X4 X5  
# 1 1.928619 2.4364068 -0.5226305 1.588079 -0.4676674  
# 2 3.577087 2.4022115 0.4317342 1.522071 1.6404145  
# 3 2.677303 0.6640803 1.4868414 2.392668 1.6421116  
# 4 0.843235 0.3078904 3.4493710 1.172779 1.5029608  
# 5 1.722469 -0.1935247 2.0272924 1.934207 1.8488969

##### Binomial noise

Here size and prob are the function arguments. size defines the number of trials and prob defines the probability of success on each trial.

simulatr::simulate\_dataset(n = 5,   
 p = 5,   
 noise\_func = binomial\_noise,   
 noise\_func\_args = list(size = 10, prob = 0.5))  
  
# result :  
  
# X1 X2 X3 X4 X5  
# 1 6.369510 3.070095 5.110972 2.846576 7.401457  
# 2 7.508578 4.070030 6.120108 7.377817 4.725431  
# 3 1.207935 4.728687 3.429556 3.599847 4.440382  
# 4 5.121097 5.172166 5.484836 5.188745 6.384407  
# 5 5.342285 6.711157 5.616503 3.893173 3.452521

##### Exponential noise

Here rate is the function argument.

simulatr::simulate\_dataset(n = 5,   
 p = 5,   
 noise\_func = exponential\_noise,   
 noise\_func\_args = list(rate = 1))  
  
result :  
  
 X1 X2 X3 X4 X5  
1 -0.2035384 0.01890351 3.8764838 1.3464868 -0.01945084  
2 -1.1027431 1.23864130 -1.4897532 0.5608686 0.55288751  
3 0.5977199 1.26828733 -0.4237199 0.1515011 2.28350774  
4 -0.1596974 2.01130450 1.4105792 3.1737743 0.99300449  
5 0.1805583 -0.46096453 0.9261371 0.3952754 0.85035973

#### With given bias

The users can choose the type of bias they want to add to their simulated data. They can either provide their own bias (a (n, p) dimensional matrix) or choose from the given bias functions. If they choose the given bias function, they have to provide arguments for the chosen one. We provide bias function named constant\_batch\_effect.

##### Constant batch effect

n and p denotes the number of samples and features respectively.

b denotes the batch each sample belongs to. Suppose, the samples come from 3 different places. The users can define which sample belongs to which place. If *b = c(1,2,1,2,3,3,2)*, that means sample1 belongs to batch 1, sample 2 belongs to batch 2, sample 3 belongs to batch 1, sample 5 belongs to batch 3 and so on.

f denotes the number of features to be affected. If the user chooses *f = 4*, then 4 features will be randomly selected.

*s = c(1,2,1)* means batch 1, 2 and 3 will be affected by 1, 2 and 1 respectively. s denotes the strength of the batch effect, i.e., how much the feature values within a batch are changed through the batch effect. Here, sample 2 and 5 belong to batch 1 and other samples belong to batch 2. Samples in batch 1 have their feature values increased by s = 1 whereas samples in batch 2 have their feature values increased by s = 2. Features 1,3,4 and 5 are affected by the batch effects.

Here is an example :

[const\_batch\_effect.jpg]{#id .class width=30 height=20px}

simulatr::simulate\_dataset( n = 7,  
 p = 8,  
 base = data.frame(matrix(0, 7, 8)),  
 bias\_func = constant\_batch\_effect,  
 bias\_func\_args = (list(b = c(1,2,1,2,3,3,2),  
 f = 4,  
 s = c(1,2,1))))  
  
# result :  
  
# X1 X2 X3 X4 X5 X6 X7 X8  
# 1 1 0 1 1 0 1 0 0  
# 2 2 0 2 2 0 2 0 0  
# 3 1 0 1 1 0 1 0 0  
# 4 2 0 2 2 0 2 0 0  
# 5 1 0 1 1 0 1 0 0  
# 6 1 0 1 1 0 1 0 0  
# 7 2 0 2 2 0 2 0 0

The users can call a simplified version of constant\_batch\_effect. If users define b = 2, then samples will be randomly assigned to 2 different batches.

simulatr::simulate\_dataset( n = 5,  
 p = 5,  
 base = data.frame(matrix(0, 5, 5)),  
 bias\_func = constant\_batch\_effect,  
 bias\_func\_args = (list(b = 2, f = 4, s = c(1, 2))))  
  
# result :  
  
# X1 X2 X3 X4 X5  
# 1 2 0 2 2 2  
# 2 1 0 1 1 1  
# 3 2 0 2 2 2  
# 4 2 0 2 2 2  
# 5 1 0 1 1 1

Here, sample 2 and sample 5 belongs to batch 1 and other samples belong to batch 2. Batch 1 is increased by 1 wheras batch 2 is increased by 2.Four featues, feature 1, feature 3, feature 4 and feature 5, are randomly selected for the effect.

In a more simplified version, if the first batch is affected by 0, then user can skip that.

simulatr::simulate\_dataset(n = 5,  
 p = 5,  
 base = data.frame(matrix(0, 5, 5)),  
 bias\_func = constant\_batch\_effect,  
 bias\_func\_args = (list(b = 2, f = 2, s = 1)))  
  
# result :  
  
# X1 X2 X3 X4 X5  
# 1 0 0 0 0 0  
# 2 1 1 0 0 0  
# 3 1 1 0 0 0  
# 4 0 0 0 0 0  
# 5 0 0 0 0 0

Here, feature 1 and 2 are affected. Sample 1, 3 and 4 belong to batch 1. Other samples belong to batch 2. Batch 1 is increased by 0 and batch 2 is increased by 1.

#### Some more examples

simulatr::simulate\_dataset( n = 7,  
 p = 8,  
 base = simulate\_gse(10, 10, gse = "GSE461"),  
 bias\_func = constant\_batch\_effect,  
 bias\_func\_args = (list(b = 2, f = 4, s = c(1, 2))))  
  
# Result:  
  
# GSM7490 GSM7492 GSM7490.1 GSM7493 GSM7490.2 GSM7496 GSM7490.3 GSM7496.1  
# 2733\_g\_at 10.3 354.7 10.6 117.4 90.2 406.0 92.6 90.2  
# 10444\_at 1021.6 589.6 654.3 401.7 51.5 447.1 42.4 5.7  
# 11127\_at 20.0 561.9 512.3 655.7 40.6 1049.8 1671.4 1582.4  
# 4933\_at 4875.3 55.6 447.1 551.4 1216.3 20.1 8.2 384.2  
# 11275\_at 67.2 580.1 4.5 95.2 77.1 85.8 57.6 233.4  
# 6862\_at 197.2 136.2 42.8 202.4 0.8 22.1 114.1 174.8  
# 5644\_at 699.3 72.2 2345.4 18.9 0.8 139.3 44.1 723.0

simulatr::simulate\_dataset( n = 20,  
 p = 4,  
 base = get\_dataset(gse = "GSE461"),  
 bias\_func = constant\_batch\_effect,  
 bias\_func\_args = (list(b = 2, f = 4, s = c(1,2))))  
  
# Result:  
  
# GSM7492 GSM7495 GSM7496 GSM7491  
# 8503\_at 4.5 19.1 25.7 19.5  
# 5374\_at 28.9 9.0 3.5 2.9  
# 5238\_at 839.4 1309.4 1230.5 920.0  
# 5386\_at 373.2 332.9 483.9 272.7  
# 5194\_at 632.5 597.5 499.2 775.1  
# 7123\_at 64.1 85.4 104.2 117.2  
# 5333\_at 370.6 76.4 187.0 323.6  
# 7958\_at 4.3 6.2 4.5 3.2  
# 6856\_at 404.5 614.0 618.9 530.4  
# 10770\_at 2412.1 2193.4 1652.2 2670.6  
# 10993\_at 3354.7 1345.3 1857.6 3500.1  
# 8252\_at 2678.9 1661.9 1087.9 2632.1  
# 6981\_at 561.3 268.1 180.1 438.2  
# 10827\_s\_at 115.3 108.6 76.7 121.6  
# 11136\_at 508.4 316.6 214.7 471.8  
# 8175\_at 21.9 5.1 15.0 14.9  
# 10190\_at 133.7 166.6 114.1 41.6  
# 7720\_at 4.1 2.8 1.5 3.1  
# 10242\_at 658.5 1197.6 1259.5 616.7  
# 2314\_at 2.7 3.5 2.5 4.0

simulatr::simulate\_dataset( n = 10,  
 p = 4,  
 base = get\_dataset(gse = "GSE461"),  
 noise\_func = uniform\_noise,  
 noise\_func\_args = list(min = 1, max = 2),  
 bias\_func = constant\_batch\_effect,  
 bias\_func\_args = list(b = 2, f = 4, s = c(1, 2)))  
  
# Result :  
# GSM7492 GSM7495 GSM7493 GSM7496  
# 5706\_at 138.765988 140.009436 122.18638 142.362293  
# 5055\_at 46.729894 32.160862 43.61547 41.714460  
# 5662\_at 636.360690 518.906011 709.55725 330.161930  
# 3580\_f\_at 1319.538146 2992.370463 1400.84572 2042.791904  
# 6477\_at 619.231089 632.390663 627.33365 427.670557  
# 11170\_at 4.974592 21.439322 22.32108 21.185163  
# 4971\_at 51.205235 84.743411 41.68645 59.699879  
# 6589\_at 1090.293347 1199.824401 1041.52099 1420.509870  
# 2647\_at 11.021299 4.667569 11.37919 6.209028  
# 5387\_at 245.777407 211.993438 124.88081 335.907862

simulatr::simulate\_dataset( n = 8,  
 p = 4,  
 base = simulate\_gse(15, 15, "GSE803"),  
 noise\_func = uniform\_noise,  
 noise\_func\_args = list(min = 1, max = 2),  
 bias\_func = constant\_batch\_effect,  
 bias\_func\_args = list(b = 2, f = 4, s = c(1, 2)))  
  
# Result:  
  
# GSM12738 GSM12723 GSM12649 GSM12698  
# 31366\_at 200.53742 12.80908 647.98507 9.251130  
# 38901\_at 12.44034 296.25831 27.65901 3.721115  
# 41774\_at 16.11559 337.79170 759.81003 16.470416  
# 310\_s\_at 459.91977 36.69029 161.48037 647.954018  
# 31608\_g\_at 1511.23696 219.72824 17.24672 9.169994  
# 35327\_at 180.59832 17.05115 216.95698 97.889850  
# 38910\_at 1174.29895 60.09656 15.56079 4.801443  
# 35920\_at 17.45200 158.22371 41.74438 11.826267

The users can provide their own noise or bias matrix if they want.

simulatr::simulate\_dataset( n = 4,  
 p = 4,  
 base = simulate\_gse(15, 15, "GSE803"),  
 noise\_func = matrix(1, 4, 4),  
 bias\_func = matrix(2, 4, 4))  
  
# Result :   
  
# GSM12718 GSM12723 GSM12674 GSM12723.1  
# 36308\_at 77.2 782.3 61.8 782.3  
# 39768\_at 78.1 22.6 126.7 22.6  
# 40584\_at 14.0 1722.5 6.4 1722.5  
# 37259\_at 10.4 964.2 22.5 964.2

### List of the platforms

Provides list of all the platforms(e.g. GPL97) available in NCBI.

utils::head(simulatr::list\_platforms(),3)  
  
# A part of the result that can be retrieved by this function :  
  
# Accession Title   
# 1 GPL25897 Illumina HiSeq 4000 (Fagopyrum hailuogouense)   
# 2 GPL25881 Qiagen Mouse Inflammatory Response and Autoimmunity PCR Array (PAMM-077Z)   
# 3 GPL25892 HiSeq X Ten (Rattus rattus)   
# Technology Taxonomy Data.Rows Samples.Count Series.Count  
# 1 high-throughput sequencing Fagopyrum hailuogouense 0 6 1  
# 2 RT-PCR Mus musculus 96 34 1  
# 3 high-throughput sequencing Rattus rattus 0 8 1  
# Contact Release.Date  
# 1 GEO Dec 05, 2018  
# 2 Patricia Silveyra Dec 04, 2018  
# 3 GEO Dec 04, 2018

### Retrieve data for a given platform

The users can retrieve data for a specific platform (e.g. GPL97)

simulatr::get\_gpl\_data("GPL96")  
  
# # result :  
# GEO Accession Title Technology Organism  
# a "GPL96" "[HG-U133A] Affymetrix Human Genome U133A Array" "in situ oligonucleotide" "Homo sapiens"   
# Status  
# a "Public on Mar 11 2002"

### List of the datasets

Provides information about all series (e.g. GSE465) related to a specific platform (e.g. GPL95)

simulatr::list\_datasets(platform = "GPL95")  
  
# # result :  
# Accession Title  
# 20365 GSE465 Expression profiling in the muscular dystrophies  
# 151738 GSE762 CCFAlmasan\_CaP1  
# 32434 GSE803 GeneNote-Gene Normal tissue Expression  
# 74830 GSE1007 Molecular profiles(HG-U95B,C,D,E) of dystrophin-deficient and normal human skeletal muscle  
# 147372 GSE1302 primary trophoblast study  
# 20322 GSE2508 Expression profiling in adipocytes of obese humans  
# 164705 GSE5949 Comparison between cell lines from 9 different cancer tissue (NCI-60) (U95 platform)  
# 16539 GSE51625 Expression data from human abdominal, subcutaneous adipose tissue  
# Series.Type Taxonomy Sample.Count  
# 20365 Expression profiling by array Homo sapiens 57  
# 151738 Expression profiling by array Homo sapiens 25  
# 32434 Expression profiling by array Homo sapiens 120  
# 74830 Expression profiling by array Homo sapiens 86  
# 147372 Expression profiling by array Homo sapiens 75  
# 20322 Expression profiling by array Homo sapiens 195  
# 164705 Expression profiling by array Homo sapiens 300  
# 16539 Expression profiling by array Homo sapiens 120  
# Datasets Supplementary.Types  
# 20365 GDS214;GDS262;GDS264;GDS265;GDS270 CEL  
# 151738 GDS719;GDS720;GDS721;GDS722;GDS723  
# 32434 GDS422;GDS423;GDS424;GDS425;GDS426  
# 74830 GDS609;GDS610;GDS611;GDS612 CEL;EXP;RPT  
# 147372  
# 20322 GDS1495;GDS1496;GDS1497;GDS1498;GDS3601;GDS3602  
# 164705 CEL;EXP  
# 16539 CEL  
# Supplementary.Links PubMed.ID SRA.Accession  
# 20365 ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSEnnn/GSE465/suppl 11121445  
  
# 151738  
  
# 32434 15388519  
  
# 74830 ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE1nnn/GSE1007/suppl 12698323  
  
# 147372 15161964  
  
# 20322 16059715;18424589  
  
# 164705 ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE5nnn/GSE5949/suppl 20053763;26048278;25003721  
  
# 16539 ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE51nnn/GSE51625/suppl 24103848  
  
# Contact Release.Date  
# 20365 Eric P Hoffman Jul 16, 2003  
# 151738 John Gilmary Hissong Jun 01, 2004  
# 32434 Orit Shmueli Nov 19, 2003  
# 74830 Judith Haslett Feb 04, 2004  
# 147372 Brig Mecham Apr 13, 2004  
# 20322 Yong-Ho Lee Jan 01, 2006  
# 164705 Uma T Shankavaram Aug 06, 2007  
# 16539 Carol Shoulders Oct 29, 2013

## Importing the package

{r setup} library(simulatr)