pulseTD

XinWang

2019-05-05

Contents

Information

Package: pulseTD

Type: Package

Author: XinWang

Maintainer: The package maintainer xindd_2014@163.com

Title: Identification of Transcriptional Dynamics using Pulse Models via 4su-Seq Data and RNA-Seq Data

Version: 0.1.0

Date: 2019-5-5

Description: This package, based on 4sU-seq data and RNA-seq data, analyzes the transcription, processing and degradation rates of genes. pulseTD can not only recognize the transcriptional dynamic rate of the measurement time points, but also obtain continuous changes in transcriptional dynamics. More importantly, it is able to predict the trend of mRNA transcription and expression changes in the future. In terms of performance, pulseTD has better robustness and accuracy than other methods.

License: GPL-2

Encoding: UTF-8

Depends: R (>= 3.4.0)

Imports: AnnotationDbi,SummarizedExperiment,Rsamtools,Biobase,S4Vectors,methods,parallel,GenomicFeatures, ggplot2, grid, GenomicAlignments

RoxygenNote: 6.1.1

Suggests: knitr, rmarkdown, TxDb.Hsapiens.UCSC.hg19.knownGene

VignetteBuilder: knitr

Abstract

The life cycle of intracellular mRNA mainly undergoes transcriptional production, splicing maturation and degradation processes. We refer to the dynamic changes of these processes over time as transcriptional dynamics. Under the influence of external disturbances and other factors, the common regulation of transcriptional dynamic processes leads to different levels of RNA expression. The emergence of labeled RNA (4sU-RNA) has made it possible for us to analyze this transcriptional dynamics. The pulseTD package is analyzed with 4sU-seq data to resolve the transcriptional dynamics of the gene. PulseTD constructed based on pulse model can identify the transcriptional dynamic rate of the measurement time node, and can also recognize the continuous change of mRNA transcription dynamics during the monitoring time, and the model can also predict the trend of mRNA transcription and expression changes in the future.

The workflow of the pulseTD packages is:

1. The input data is the alignment file of 4sU-seq labeled data and unlabeled data (Labeled bam and UnLabeled bam respectively).The expression values of pre-mRNA, mRNA and label-mRNA were calculated separately.
2. The fitted pulse model is an optimization problem. There are six parameters $\Theta = (h_0, h_1, h_2, t_1, t_2, \beta)$, which are determined by minimizing the stationary error.
3. Due to the influence of random initial values, the parameters of the fitting failure will occur, and the parameters will be re-estimated using correctionParams.
4. Solving the dynamic rate of transcription, including transcription, processing and degradation rates, or predicting steady state rates.
5. Finally, the expression value of the gene is predicted based on the transcription rate of the three stages.

Evaluation of expression values of total RNA and labeled RNA

In the process of calculating the expression value, we use the RPKM calculation method. The user only needs to provide a list of the bam files of the reads alignment result, and the annotation file in the txdb format.The expression value can be obtained by using the estimateExpression() function. Test files are provided in the pulseTD package.

```
library(pulseTD)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
```

```
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
test_path <- file.path(system.file(package="pulseTD"),'extdata/test1.sorted.bam')
test_path2 <- file.path(system.file(package="pulseTD"),'extdata/test2.sorted.bam')
```

```
rpkmres <- estimateExpression(txdb,c(test_path,test_path2), by='gene')
```

```
data('rpkmres', package='pulseTD')
head(rpkmres$total_exp)
```

```
##           test1.sorted.bam test2.sorted.bam
## 1                   0.0000           0.0000
## 10                  0.0000           0.0000
## 100                 0.0000           0.0000
## 1000              211.6078           0.0000
## 10000               0.0000         126.8966
```

```
## 100008586           0.0000           0.0000
```

```
head(rpkmres$pre_exp)
```

```
##           test1.sorted.bam test2.sorted.bam
## 1                   0.0000           0.0000
## 10                  0.0000           0.0000
## 100                 0.0000           0.0000
## 1000              211.6078           0.0000
## 10000               0.0000         126.8966
## 100008586           0.0000           0.0000
```

Fitting pulse model parameters

pulseTD uses a pulse model, which is multiplied by two sigmoid functions, the parameter vector $(h_0, h_1, h_2, t_1, t_2, \beta)$, where h_0, h_1, h_2 represent the initial state rate, the value, the peak rate value and the steady state rate value again, t_1, t_2 are the maximum times of the first and second rise or fall changes, respectively, and β is the slope of the two changes. The unknown vector $X = (\Theta_\alpha, \Theta_\gamma, \Theta_\beta)$ has 15 parameters. We use the R stats function nlminb to optimize.

```
data('rpkmSim', package='pulseTD')
rpkm_TL <- rpkmSim$labexon[1:2,]
rpkm_PT <- rpkmSim$totintr[1:2,]
rpkm_TT <- rpkmSim$totexon[1:2,]
TimeGrid <- c(0, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180)
tL <- 10
```

```
pulseRates<- estimateParams(rpkm_TL, rpkm_TT, rpkm_PT,TimeGrid, tL, clusterNumber=1,loopnumber=10)
```

Correction pulse model parameters

In the process of parameter fitting, we adopt the method of random initial value. Due to the size of the random initial value or the overflow of the number of iterations, the parameters of the fitting failure may occur, which may cause the data of some genes lose meaning. Not conducive to our subsequent analysis. To this end, we re-estimate the genes that failed to fit, and if they fail again, these genes will be filtered out. Only need to use the correctionParams function to complete the re-evaluation of the parameters.

```
data('pulseRates', package='pulseTD')
pulseRates_correct = correctionParams(pulseRates)
```

```
## There are no parameters that need to be corrected
```

```
pulseRates_correct@fitfailure
```

```
## NULL
```

Get steady state parameters

If we need to analyze the steady state characteristics of the transitional dynamics, we can obtain the pulse function parameters corresponding to the transcription, processing and degradation of each gene, and also use the pulseModel to obtain the corresponding rate curve.

```
data('pulseRates', package='pulseTD')
TimeGrid <- c(0, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180)
pulseRates_correct <- correctionParams(pulseRates)
```

```
## There are no parameters that need to be corrected
```

```
transcription_params = getParams(pulseRates,'transcription')
degradation_params = getParams(pulseRates, 'degradation')
processing_params = getParams(pulseRates, 'processing')
head(transcription_params)
```

```
##                    h_0       h_1       h_2       t_1       t_2       beta
## NM_001001144 0.1561820 0.1776833 0.1450480 64.821521 40.470711 49.8882040
## NM_001001181 0.3068200 1.1056015 0.3511529  8.286115  9.791383  5.1779610
## NM_001001182 0.3785050 0.5523380 0.4624175 17.398237 90.318868  4.9183223
## NM_001001321 0.0792062 0.1026056 0.0693335 62.871792 74.762393  5.4155110
## NM_001001327 0.1265100 0.2469133 0.1433675  2.855637 44.712863  6.0563221
## NM_001001491 2.5050253 0.1000000 3.5309954  9.097042 17.498354  0.1638573
```

```
head(degradation_params)
```

```
##                    h_0       h_1       h_2       t_1      t_2       beta
## NM_001001144 0.1561820 0.1139163 0.1450480  0.324933  5.12593 8.70622763
## NM_001001181 0.3068200 1.2281898 0.3511529  0.000000  0.00000 0.06411246
## NM_001001182 0.3785050 0.6257900 0.4624175 48.157458 66.69268 0.07761252
## NM_001001321 0.0792062 0.1000000 0.0693335 38.126117 84.86571 4.03770095
## NM_001001327 0.1265100 0.7919398 0.1433675  0.000000  0.00000 0.04667249
## NM_001001491 2.5050253 0.3040949 3.5309954  0.000000 12.74953 0.12427920
```

```
head(processing_params)
```

```
##                    h_0       h_1       h_2       t_1       t_2       beta
## NM_001001144 0.1561820 0.1782793 0.1445186 68.816474 37.993956 11.3717946
## NM_001001181 0.3068200 0.8898962 0.3504246  5.664148 13.567671  0.7093536
## NM_001001182 0.3785050 0.5649453 0.4624175 23.921890 84.723696  0.3126343
## NM_001001321 0.0792062 0.1766850 0.0693335  1.983009  1.508767  0.0000000
## NM_001001327 0.1265100 0.2528995 0.1433675  1.915350 44.974753  0.3302034
## NM_001001491 2.5050253 4.5369477 3.5309954  7.151024 23.143396  0.1736154
```

```
transcription_params = getParams(pulseRates, 'transcription', genename=c(1,2,3))
head(transcription_params)
```

```
##                   h_0       h_1       h_2       t_1       t_2      beta
## NM_001001144 0.156182 0.1776833 0.1450480 64.821521 40.470711 49.888204
## NM_001001181 0.306820 1.1056015 0.3511529  8.286115  9.791383  5.177961
## NM_001001182 0.378505 0.5523380 0.4624175 17.398237 90.318868  4.918322
```

```
# get pulse Model value
transcription_pulse = pulseModel(as.matrix(transcription_params[1,]), TimeGrid)
degradation_pulse = pulseModel(as.matrix(degradation_params[1,]), TimeGrid)
processing_pulse = pulseModel(as.matrix(processing_params[1,]), TimeGrid)
```

Transcriptional dynamic rates

The pulseTD method can recognize the transcription, splicing maturation, degradation rate and predict the future trend of transcriptional dynamic rate at any time during the detection period.

Get Transcriptional dynamic rate

This function is used to calculate the transcriptional dynamic rate of a gene. You can get the discrete or continuous rate values of the measurement time node. At the same time, it has a predictive function that

provides rate values for any future time node or any range of time.

```
data('pulseRates', package='pulseTD')
pulseRates_correct <- correctionParams(pulseRates)
```

```
## There are no parameters that need to be corrected
```

```
transcription = getRates(pulseRates_correct,'transcription')
degradation = getRates(pulseRates_correct, 'degradation')
processing = getRates(pulseRates_correct, 'processing')
head(transcription)
```

```
##                        0        15        30        45        60        75
## NM_001001144 0.1561820 0.1561820 0.1561820 0.1561820 0.1561820 0.1776833
## NM_001001181 0.3068200 1.1056015 0.3511529 0.3511529 0.3511529 0.3511529
## NM_001001182 0.3785050 0.3785063 0.5523380 0.5523380 0.5523380 0.5523380
## NM_001001321 0.0792062 0.0792062 0.0792062 0.0792062 0.0792062 0.1026056
## NM_001001327 0.1265100 0.2469133 0.2469133 0.2469133 0.1433675 0.1433675
## NM_001001491 2.9578325 4.1658896 4.0118622 3.5959630 3.5368318 3.5314972
##                       90       105       120       135       150       165
## NM_001001144 0.1776833 0.1776833 0.1450480 0.1450480 0.1450480 0.1450480
## NM_001001181 0.3511529 0.3511529 0.3511529 0.3511529 0.3511529 0.3511529
## NM_001001182 0.5523380 0.5523379 0.4624175 0.4624175 0.4624175 0.4624175
## NM_001001321 0.1026056 0.1026056 0.1026056 0.1026056 0.0693335 0.0693335
## NM_001001327 0.1433675 0.1433675 0.1433675 0.1433675 0.1433675 0.1433675
## NM_001001491 3.5310384 3.5309991 3.5309957 3.5309955 3.5309954 3.5309954
##                      180
## NM_001001144 0.1450480
## NM_001001181 0.3511529
## NM_001001182 0.4624175
## NM_001001321 0.0693335
## NM_001001327 0.1433675
## NM_001001491 3.5309954
```

If you want to get the coefficients of the parameters, you only need to divide the expression on the basis of the rate:

```
if(length(pulseRates_correct@fitfailure)==0){
  genename=pulseRates_correct@genenames
}else{
  genename=pulseRates_correct@genenames[-pulseRates_correct@fitfailure]
}
data('rpkmSim', package='pulseTD')
simTL <- rpkmSim$labexon[c(1,2,3),]
simPT <- rpkmSim$totintr[c(1,2,3),]
simTT <- rpkmSim$totexon[c(1,2,3),]
trans_factor <- getRates(pulseRates_correct,'transcription', genename=c(1,2,3))
degr_factor <- getRates(pulseRates_correct, 'degradation', genename=c(1,2,3)) /(simTT-simPT)
proc_factor <- getRates(pulseRates_correct,'processing',genename=c(1,2,3))/simPT
head(degr_factor)
```

```
##                         0         15         30         45         60
## NM_001001144 0.03134559 0.02788742 0.02719004 0.02680691 0.02519974
## NM_001001181 0.04602746 0.03586912 0.02862095 0.02705451 0.02781393
## NM_001001182 0.03589241 0.03776641 0.03731866 0.03818952 0.04097445
##                        75         90        105        120        135
## NM_001001144 0.02246530 0.02116933 0.02041119 0.02034475 0.02093304
```

```
## NM_001001181 0.02854330 0.02824130 0.02910425 0.02720460 0.02566296
## NM_001001182 0.04362242 0.04662836 0.04634300 0.04511589 0.04436894
##                     150        165        180
## NM_001001144 0.02157378 0.02146326 0.02177970
## NM_001001181 0.02605172 0.02636889 0.02597678
## NM_001001182 0.04402819 0.04346080 0.04398772
```

Predicting transcriptional dynamic rates

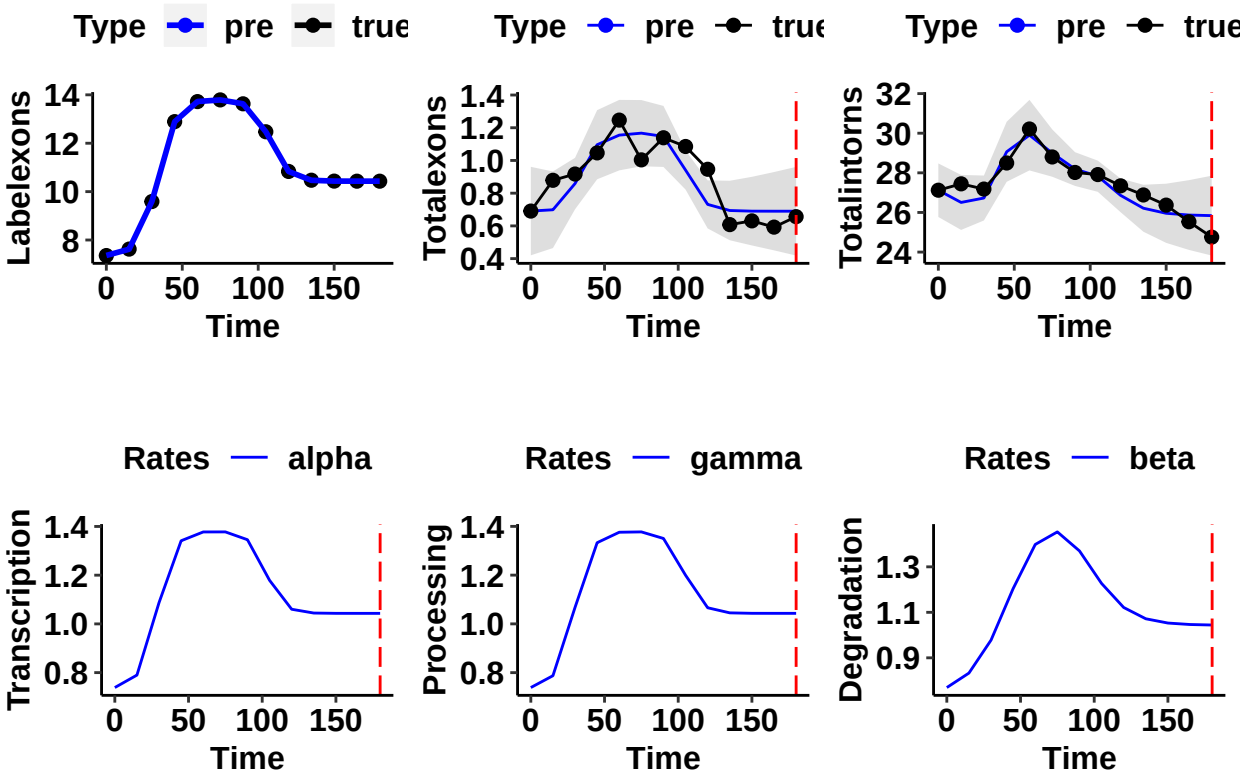
Predicting the transcription dynamic rate requires only adding a specific time series to the getRates function, for example:

```
transcription_pre <- getRates(pulseRates_correct, 'transcription', timevector=seq(0,360, 15))
degradation_pre <- getRates(pulseRates_correct, 'degradation', timevector=seq(0,360, 15))
processing_pre <- getRates(pulseRates_correct, 'processing', timevector <- seq(0,360, 15))
head(degradation_pre)
```

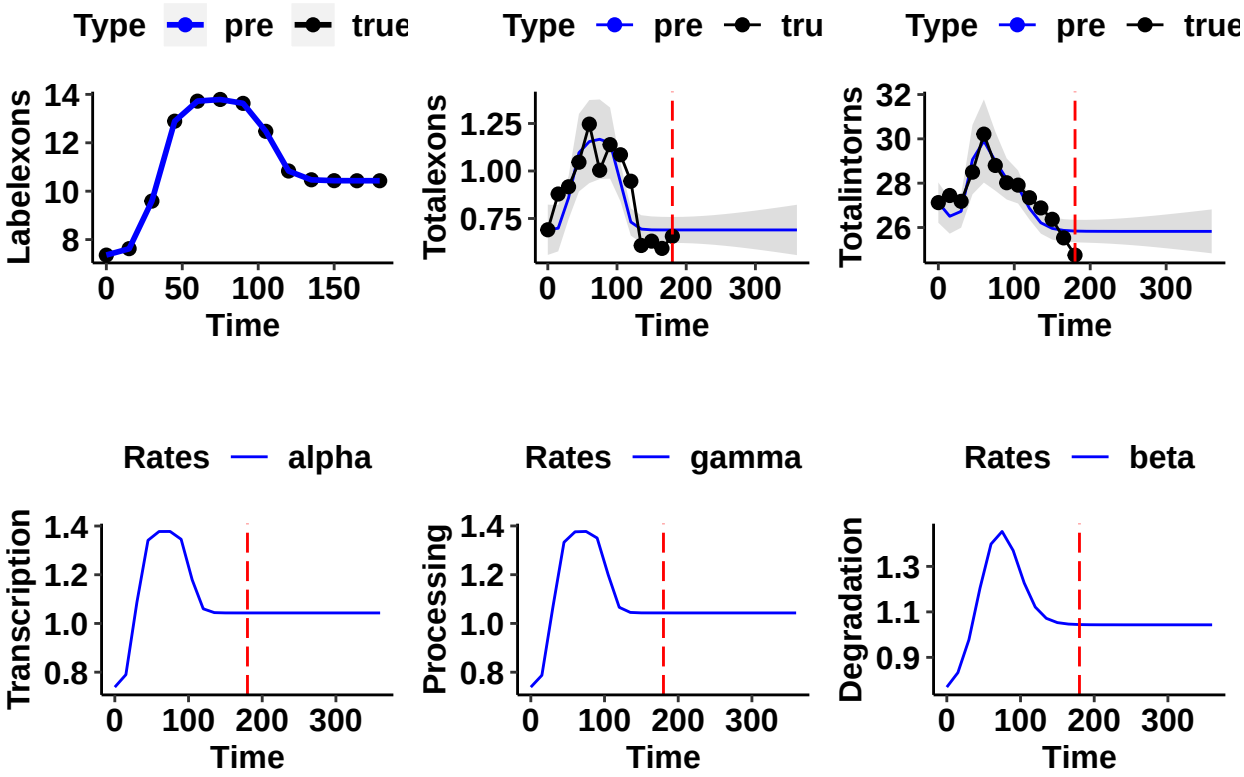
```
##                       0        15        30        45        60        75
## NM_001001144 0.1538244 0.1450480 0.1450480 0.1450480 0.1450480 0.1450480
## NM_001001181 0.4934715 0.4705244 0.4186859 0.3817699 0.3636916 0.3560767
## NM_001001182 0.3842426 0.3959850 0.4269065 0.4865133 0.5532558 0.5916279
## NM_001001321 0.0792062 0.0792062 0.0792062 0.1000000 0.1000000 0.1000000
## NM_001001327 0.2711799 0.2585965 0.2265018 0.1944857 0.1718412 0.1583731
## NM_001001491 3.9406900 4.2221897 3.7347260 3.5667215 3.5366453 3.5318740
##                      90       105        120       135       150       165
## NM_001001144 0.1450480 0.1450480 0.14504800 0.1450480 0.1450480 0.1450480
## NM_001001181 0.3530547 0.3518828 0.35143234 0.3512598 0.3511938 0.3511686
## NM_001001182 0.5961126 0.5711724 0.52720557 0.4904667 0.4723694 0.4656624
## NM_001001321 0.1000000 0.1000000 0.09999983 0.0693335 0.0693335 0.0693335
## NM_001001327 0.1510469 0.1472389 0.14530443 0.1443329 0.1438478 0.1436062
## NM_001001491 3.5311317 3.5310166 3.53099871 3.5309959 3.5309955 3.5309954
##                     180       195       210       225       240       255
## NM_001001144 0.1450480 0.1450480 0.1450480 0.1450480 0.1450480 0.1450480
## NM_001001181 0.3511589 0.3511552 0.3511538 0.3511533 0.3511531 0.3511530
## NM_001001182 0.4634447 0.4627396 0.4625182 0.4624490 0.4624273 0.4624206
## NM_001001321 0.0693335 0.0693335 0.0693335 0.0693335 0.0693335 0.0693335
## NM_001001327 0.1434861 0.1434264 0.1433967 0.1433820 0.1433747 0.1433711
## NM_001001491 3.5309954 3.5309954 3.5309954 3.5309954 3.5309954 3.5309954
##                     270       285       300       315       330       345
## NM_001001144 0.1450480 0.1450480 0.1450480 0.1450480 0.1450480 0.1450480
## NM_001001181 0.3511530 0.3511529 0.3511529 0.3511529 0.3511529 0.3511529
## NM_001001182 0.4624185 0.4624178 0.4624176 0.4624175 0.4624175 0.4624175
## NM_001001321 0.0693335 0.0693335 0.0693335 0.0693335 0.0693335 0.0693335
## NM_001001327 0.1433693 0.1433684 0.1433679 0.1433677 0.1433676 0.1433676
## NM_001001491 3.5309954 3.5309954 3.5309954 3.5309954 3.5309954 3.5309954
##                     360
## NM_001001144 0.1450480
## NM_001001181 0.3511529
## NM_001001182 0.4624175
## NM_001001321 0.0693335
## NM_001001327 0.1433675
## NM_001001491 3.5309954
```

Draw transcriptional dynamics

```
data('pulseRates', package='pulseTD')
plotRates(pulseRates, 15)
```




```
plotRates(pulseRates, 15, predict=c(0,360,15))
```



Predicted gene expression value

This function is used to predict the expression of all gene at a given time, including the expression of pre-mRNA and the expression of total mRNA. End time and time interval can be arbitrarily defined

```
data('pulseRates', package='pulseTD')
pulseRates_correct = correctionParams(pulseRates)
```

```
## There are no parameters that need to be corrected
```

```
TimeGrid = seq(0,180,15)
```

```
preExp = predictExpression(pulseRates_correct, tg=TimeGrid)
```

```
data('preExp', package='pulseTD')
df = data.frame(preExp[['NM_001002011']])
head(df)
```

```
##          PT       TT       upPT     downPT     upTT   downTT
## 1 0.2398300 34.47230 0.2816086 0.1980514 39.13227 29.81233
## 2 0.3456477 30.53136 0.4515708 0.2397246 31.88580 29.17692
## 3 0.2210542 31.21627 0.2552815 0.1868269 32.64075 29.79178
## 4 0.1803224 31.28001 0.2279913 0.1326536 32.55389 30.00613
## 5 0.1911017 30.22977 0.2281425 0.1540609 31.12172 29.33782
## 6 0.2012920 29.62785 0.2311313 0.1714527 30.49318 28.76252
```

SessionInfo

```
sessionInfo()
```

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17134)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=Chinese (Simplified)_China.936
## [3] LC_MONETARY=Chinese (Simplified)_China.936
## [4] LC_NUMERIC=C
## [5] LC_TIME=Chinese (Simplified)_China.936
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [2] GenomicFeatures_1.30.0
##  [3] AnnotationDbi_1.40.0
##  [4] Biobase_2.38.0
##  [5] GenomicRanges_1.30.0
##  [6] GenomeInfoDb_1.14.0
##  [7] IRanges_2.12.0
##  [8] S4Vectors_0.16.0
##  [9] BiocGenerics_0.24.0
## [10] pulseTD_0.1.0
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-6               matrixStats_0.52.2
##  [3] fs_1.2.6                   usethis_1.4.0
##  [5] devtools_2.0.2             bit64_0.9-7
##  [7] progress_1.1.2             httr_1.3.1
##  [9] rprojroot_1.3-2            tools_3.4.4
## [11] backports_1.1.2            R6_2.2.2
## [13] DBI_0.7                    lazyeval_0.2.1
## [15] colorspace_1.3-2           withr_2.1.2
## [17] prettyunits_1.0.2          processx_3.2.0
## [19] RMySQL_0.10.13             bit_1.1-12
## [21] compiler_3.4.4             cli_1.0.1
## [23] xml2_1.2.0                 desc_1.2.0
## [25] DelayedArray_0.4.1         labeling_0.3
## [27] rtracklayer_1.38.2         scales_0.5.0
## [29] callr_3.0.0                commonmark_1.7
## [31] stringr_1.2.0              digest_0.6.18
## [33] Rsamtools_1.30.0           rmarkdown_1.11
## [35] XVector_0.18.0             base64enc_0.1-3
## [37] pkgconfig_2.0.1            htmltools_0.3.6
## [39] sessioninfo_1.1.1          rlang_0.3.1
```

```
## [41] rstudioapi_0.8            RSQLite_2.0
## [43] bindr_0.1.1               BiocParallel_1.12.0
## [45] dplyr_0.7.4               RCurl_1.95-4.9
## [47] magrittr_1.5              GenomeInfoDbData_1.0.0
## [49] Matrix_1.2-12             Rcpp_0.12.16
## [51] munsell_0.4.3             stringi_1.1.6
## [53] yaml_2.2.0                SummarizedExperiment_1.8.1
## [55] zlibbioc_1.24.0           pkgbuild_1.0.2
## [57] plyr_1.8.4                grid_3.4.4
## [59] blob_1.1.0                crayon_1.3.4
## [61] lattice_0.20-35           Biostrings_2.46.0
## [63] knitr_1.21                ps_1.2.1
## [65] pillar_1.0.1              biomaRt_2.34.1
## [67] pkgload_1.0.2             XML_3.98-1.9
## [69] glue_1.3.0                evaluate_0.13
## [71] remotes_2.0.2             gtable_0.2.0
## [73] purrr_0.2.5               assertthat_0.2.0
## [75] ggplot2_3.1.0             xfun_0.4
## [77] roxygen2_6.1.1            tibble_1.4.1
## [79] GenomicAlignments_1.14.1  memoise_1.1.0
## [81] bindrcpp_0.2.2
```