vignette

Xinyu Zou

May 25, 2019

This document introduces how to use the three functions in package "zouhw2". Please make sure to install packages "askpass" and "sys" in advance and enter "3" when installing this package. For the definition of each parameter in the functions, please refer to the documentations via help() function.

#Install and load package

```
library(devtools)
```

```
## Warning: package 'devtools' was built under R version 3.5.3
```

```
## Warning: package 'usethis' was built under R version 3.5.3
```

```
#install.packages("askpass")
#install.packages("sys")
install_github("xyzou685/STSCI6520_HW2")
```

```
## Skipping install of 'zouhw2' from a github remote, the SHA1 (b162868f) has not changed since last ins
##   Use `force = TRUE` to force installation
```

```
library(zouhw2)
```

#1. solve_ols(A,b,cores=1,method,iteration) set A to be a 10*10 square matrix, entries drawn IID from standard normal distribution,* $v = 1_{10}$*. b is a 10*1 vector and $Av = b$. When using this function with other input, need to make sure that column number of A=row number of b. Method must be either "GS" or "Jacobi". Now we use different approach to approximate v. You can change number of cores or number of iteration as you want.

```
n=10
L <- diag(0, n)
L[(row(L) - col(L)) == 1] <- -1
U <- diag(0, n)
U[(row(U) - col(U)) == -1] <- -1
D <- diag(2, n)
a <- L+D+U
v <- as.vector(rep(1,10))
b=a%*%v
#Gauss-Seidel
solve_ols(a,b,method = "GS",iteration=100)
```

```
##            [,1]
##  [1,] 0.9998859
##  [2,] 0.9997899
##  [3,] 0.9997182
##  [4,] 0.9996746
##  [5,] 0.9996602
##  [6,] 0.9996740
##  [7,] 0.9997125
##  [8,] 0.9997708
##  [9,] 0.9998427
## [10,] 0.9999214
```

```
#Sequential Jacobi
solve_ols(a,b,cores=1,method = "Jacobi",iteration=100)
```

```
##            [,1]
##  [1,] 0.9942989
##  [2,] 0.9890596
##  [3,] 0.9847067
##  [4,] 0.9815927
##  [5,] 0.9799700
##  [6,] 0.9799700
##  [7,] 0.9815927
##  [8,] 0.9847067
##  [9,] 0.9890596
## [10,] 0.9942989
```

```
#Parallel Jacobi
solve_ols(a,b,cores=2,method = "Jacobi",iteration=100)
```

```
## Warning: package 'doParallel' was built under R version 3.5.3
## Loading required package: foreach
## Warning: package 'foreach' was built under R version 3.5.3
## Loading required package: iterators
## Warning: package 'iterators' was built under R version 3.5.3
## Loading required package: parallel
## [[1]]
##            [,1]
##  [1,] 0.9549266
##  [2,] 0.9135232
##  [3,] 0.8790947
##  [4,] 0.8544948
##  [5,] 0.8416573
##  [6,] 0.8416573
##  [7,] 0.8544948
##  [8,] 0.8790947
##  [9,] 0.9135232
## [10,] 0.9549266
```

#2. algo_leverage(xi,yi,r=floor(0.2*length(yi))) *xi can be a* $n1$ vector or a np *matrix, yi is a* $n1$ vector. xi's row number(data size) must match yi's row number. r can be set as any integer between 1 and yi's length. In the tet code, we set xi to be a 1005 *matrix, yi to be a* 1001 vector, and $y_i = -x_i + \epsilon_i$.

```
x <- matrix(rnorm(500),100,5)
y <- x%*%rep(-1,5)+rnorm(100)
algo_leverage(x,y)
```

```
## [[1]]
##            [,1]
## [1,] -1.2241170
## [2,] -0.9230084
## [3,] -1.2223631
## [4,] -1.0311754
## [5,] -1.6199841
##
```

```
## [[2]]
##             [,1]
## [1,] -0.8184723
## [2,] -1.4469086
## [3,] -1.0055214
## [4,] -1.0466605
## [5,] -0.6286525
```

#3. elnet_coord(x,y,a,lambda,betahat=rep(0,NCOL(x)),maxiteration=100) same test setting as algo_leverage() function, but you can determine more parameters: $\alpha \in [0,1], \lambda$ can be any real number, maxiteration can be any positive integer. The returned β value is expected to be smaller as λ increases.

```
x <- matrix(rnorm(500),100,5)
y <- x%*%rep(-1,5)+rnorm(100)
elnet_coord(x,y,a=0.5,lambda=0.7)
```

```
## [1] -0.5202250 -0.3989350 -0.7507777 -0.5275186 -0.5222626
```