Package demo

William Shih, Ricardo Simpao, Nilay Varshney, Luke Yee

3/20/2020

Here is an example of how our package functions run. For our data set, we are using a “SGEMM GPU kernel performance Data Set,” which measures the running times of a matrix-matrix product, given different parameter combinations.

```r
library(devtools)
library(tidyverse)
library(STA141CFinal)
library(furrr)

set.seed(141)
dat = read_csv("sgemm_product.csv")
dat = dat[sample(241000, 1000),]
dat2 = dat[1:100,]
```

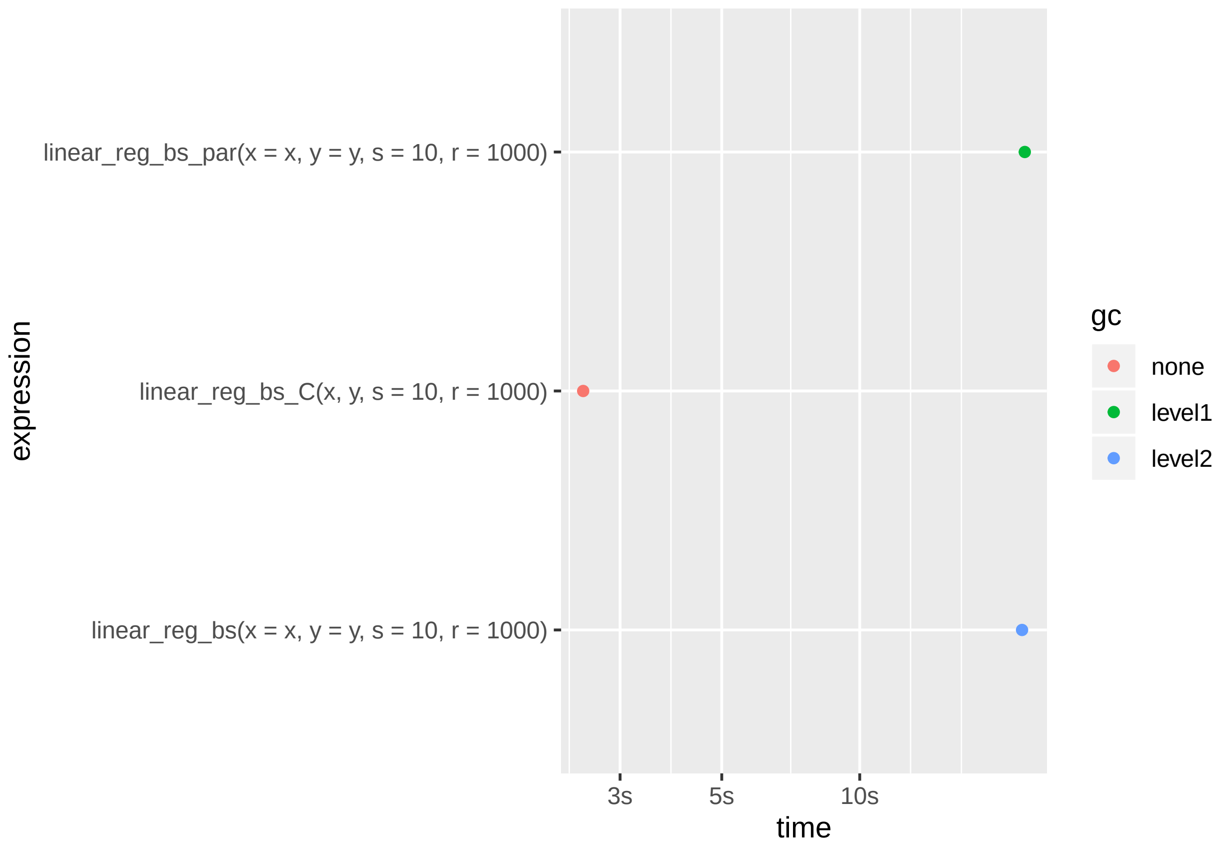
```r
#We specifiy a specific column set
y = dat$`Run1 (ms)`
x = dat[,1:(ncol(dat)-4)]

#linear model objects
fit = linear_reg_bs_C(x, y, s = 10, r = 1000)
```

Linear Regression with blb

```
## Warning: Some expressions had a GC in every iteration; so filtering is disabled.
```

```
## # A tibble: 3 x 6
##   expression                                           min median `itr/sec`
##   <bch:expr>                                        <bch:> <bch:>     <dbl>
## 1 linear_reg_bs(x = x, y = y, s = 10, r = 1000)     22.61s 22.61s    0.0442
## 2 linear_reg_bs_par(x = x, y = y, s = 10, r = 1000) 22.92s 22.92s    0.0436
## 3 linear_reg_bs_C(x, y, s = 10, r = 1000)            2.49s  2.49s    0.401
## # ... with 2 more variables: mem_alloc <bch:byt>, `gc/sec` <dbl>
```

```
## Warning in f(...): The default behavior of beeswarm has changed in version
## 0.6.0. In versions <0.6.0, this plot would have been dodged on the y-axis. In
## versions >=0.6.0, grouponX=FALSE must be explicitly set to group on y-axis.
## Please set grouponX=TRUE/FALSE to avoid this warning and ensure proper axis
## choice.
```



95 % Confidence Interval for Variable Coefficients

```
coef_CI(fit, alpha = 0.05)
```

```
##            Lower_Bounds   Estimates Upper_Bounds
## Intercept -250.9593638 -161.822787   -77.756876
## MWG          2.4803281    2.909166     3.349526
## NWG          2.3378785    2.785394     3.245074
## KWG          4.6214054    6.870881     9.220991
## MDIMC      -18.6137757  -15.942214   -13.365062
## NDIMC      -18.0645940  -15.529066   -13.084430
## MDIMA        0.8616486    2.773963     4.808043
## NDIMB       -0.1020447    1.983522     4.069686
## KWI          0.8409772    5.744059    10.709685
## VWM         -1.5499689    9.779875    20.961781
## VWN         -4.8347615    4.594360    14.276281
## STRM       -11.0345567   19.774674    51.951853
## STRN       -57.5970022  -24.403050     9.056510
## SA           0.2486218   28.977774    58.002116
## SB          24.4859861   57.913604    92.355236
```

```
coef_CI_par(fit,alpha = 0.05)
```

```
##            Lower_Bounds   Estimates Upper_Bounds
## Intercept -250.9593638 -161.822787   -77.756876
## MWG          2.4803281    2.909166     3.349526
## NWG          2.3378785    2.785394     3.245074
## KWG          4.6214054    6.870881     9.220991
## MDIMC      -18.6137757  -15.942214   -13.365062
## NDIMC      -18.0645940  -15.529066   -13.084430
## MDIMA        0.8616486    2.773963     4.808043
## NDIMB       -0.1020447    1.983522     4.069686
## KWI          0.8409772    5.744059    10.709685
## VWM         -1.5499689    9.779875    20.961781
## VWN         -4.8347615    4.594360    14.276281
## STRM       -11.0345567   19.774674    51.951853
## STRN       -57.5970022  -24.403050     9.056510
## SA           0.2486218   28.977774    58.002116
## SB          24.4859861   57.913604    92.355236
```

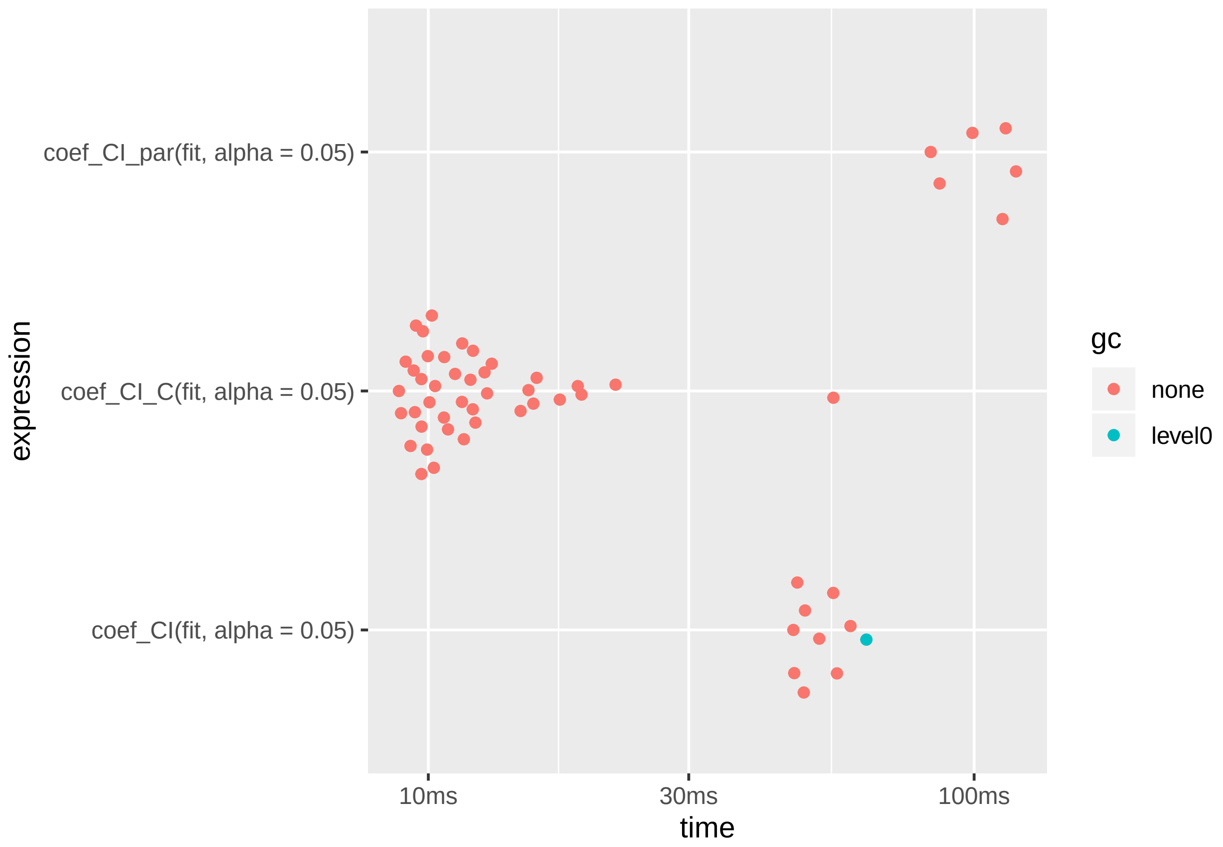
```
coef_CI_C(fit,alpha = 0.05)
```

```
##           Lower_Bounds   Estimates Upper_Bounds
## Intercept -250.9291528 -161.822787   -76.566608
## MWG          2.4804299    2.909166     3.353727
## NWG          2.3379711    2.785394     3.249154
## KWG          4.6219064    6.870881     9.246523
## MDIMC      -18.6132246  -15.942214   -13.333145
## NDIMC      -18.0640532  -15.529066   -13.067617
## MDIMA        0.8620391    2.773963     4.834013
## NDIMB       -0.1017314    1.983522     4.088236
## KWI          0.8419045    5.744059    10.754005
## VWM         -1.5468762    9.779875    21.047596
## VWN         -4.8335146    4.594360    14.337845
## STRM       -11.0248991   19.774674    52.324871
## STRN       -57.5908069  -24.403050     9.346352
## SA           0.2563944   28.977774    58.204637
## SB          24.4928834   57.913604    92.451715
```

```
(b1 = bench::mark(
  coef_CI(fit, alpha = 0.05),
  coef_CI_par(fit,alpha = 0.05),
  coef_CI_C(fit, alpha = 0.05),
  check = FALSE)
)
```

```
## # A tibble: 3 x 6
##   expression                          min   median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr>                     <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
## 1 coef_CI(fit, alpha = 0.05)      46.65ms     49ms     19.5     7.49MB     2.17
## 2 coef_CI_par(fit, alpha = 0.05)  83.27ms    106ms      9.75     7.8MB     0
## 3 coef_CI_C(fit, alpha = 0.05)     8.83ms     11ms     76.5     1.15MB     0
```

```
ggplot2::autoplot(b1)
```



Notice that `coef_CI_par` offers better memory allocation than `coef_CI`.

95% Prediction Interval

```
plan(multiprocess, workers = 4)
PI(fit, dat2[1:3, 1:14], alpha = 0.05)
```

```
##      Lower_Bounds Estimates Upper_Bounds
## [1,]    -119.5224 -64.28095    -12.19813
## [2,]     303.9097 373.82120    446.12019
## [3,]    -105.6873 -45.87250     11.50187
```

```
PI_par(fit, dat2[1:3, 1:14], alpha = 0.05)
```

```
##      Lower_Bounds Estimates Upper_Bounds
## [1,]    -119.5224 -64.28095    -12.19813
## [2,]     303.9097 373.82120    446.12019
## [3,]    -105.6873 -45.87250     11.50187
```

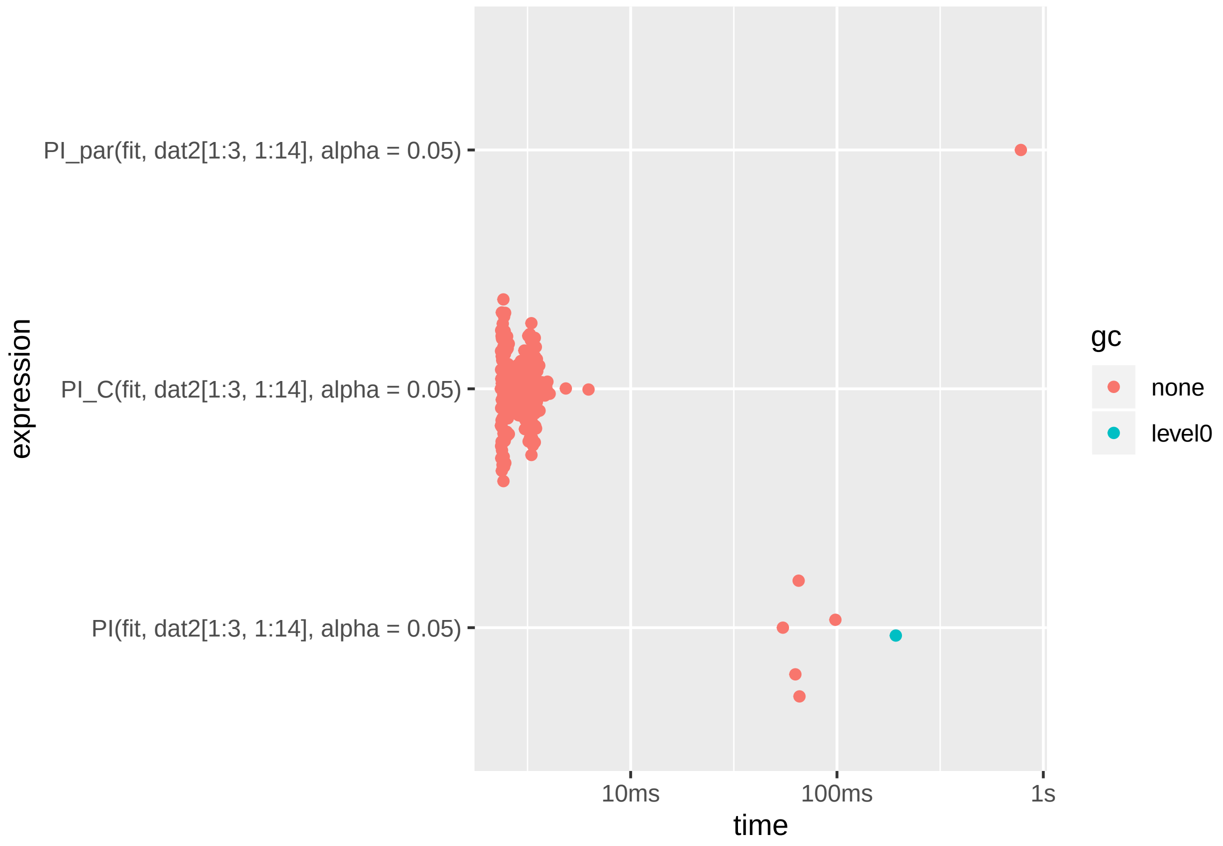
```
PI_C(fit, dat2[1:3, 1:14], alpha = 0.05)
```

```
##      Lower_Bounds Estimates Upper_Bounds
## [1,]    -119.5031 -64.28095    -11.66223
## [2,]     303.9193 373.82120    446.62104
## [3,]    -105.6698 -45.87250     12.00876
```

```
(b2 = bench::mark(
  PI(fit, dat2[1:3, 1:14], alpha = 0.05),
  PI_par(fit, dat2[1:3, 1:14], alpha = 0.05),
  PI_C(fit, dat2[1:3, 1:14], alpha = 0.05),
  check = FALSE)
)
```

```
## # A tibble: 3 x 6
##   expression                                      min   median `itr/sec`
##   <bch:expr>                                 <bch:tm> <bch:tm>     <dbl>
## 1 PI(fit, dat2[1:3, 1:14], alpha = 0.05)      54.68ms  65.18ms     14.4 
## 2 PI_par(fit, dat2[1:3, 1:14], alpha = 0.05) 778.54ms 778.54ms      1.28
## 3 PI_C(fit, dat2[1:3, 1:14], alpha = 0.05)     2.35ms   2.91ms    340.  
## # ... with 2 more variables: mem_alloc <bch:byt>, `gc/sec` <dbl>
```

```
ggplot2::autoplot(b2)
```



95 % Confindence Interval for Variance

```
s2_CI(fit, alpha = 0.05)
```

```
## Lower_Bound    Estimate Upper_Bound
##    572738.9    718571.7    858566.8
```

```
s2_CI_par(fit, alpha = 0.05)
```

```
## Lower_Bound    Estimate Upper_Bound
##    572738.9    718571.7    858566.8
```

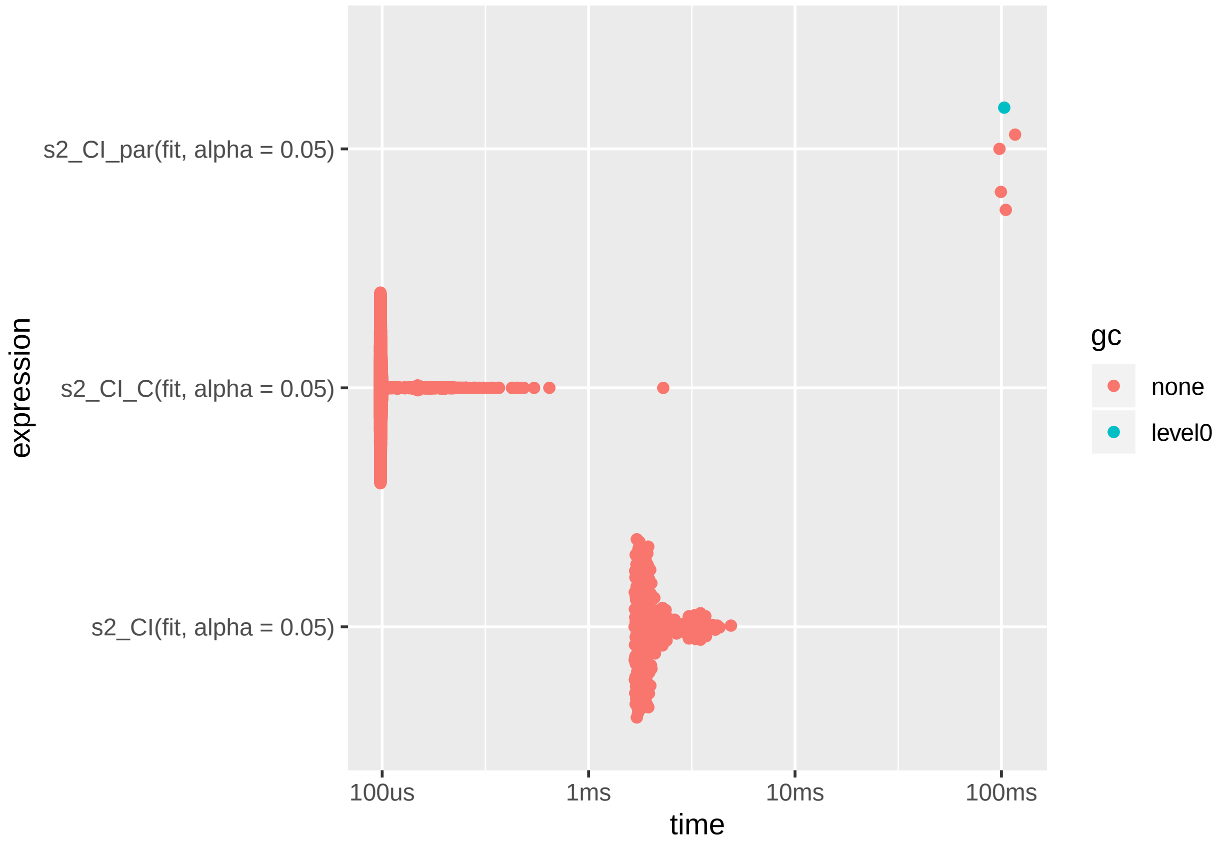
```
s2_CI_C(fit, alpha = 0.05)
```

```
## Lower_Bound    Estimate Upper_Bound
##    572781.4    718571.7    859655.6
```

```
(b3 = bench::mark(
  s2_CI(fit, alpha = 0.05),
  s2_CI_par(fit, alpha = 0.05),
  s2_CI_C(fit, alpha = 0.05),
  check = FALSE)
)
```

```
## # A tibble: 3 x 6
##   expression                        min   median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr>                   <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
## 1 s2_CI(fit, alpha = 0.05)       1.67ms   1.92ms    461.    118.12KB     0
## 2 s2_CI_par(fit, alpha = 0.05)  97.81ms 102.18ms      9.56    2.53MB     2.39
## 3 s2_CI_C(fit, alpha = 0.05)     97.4us   98.6us   8910.      2.49KB     0
```

```
ggplot2::autoplot(b3)
```



Notice that `s2_CI_par` offers better memory allocation than `s2_CI`.