TestingRandC

William Shih

3/17/2020

```r
library(Rcpp)
library(RcppArmadillo)
library(rbenchmark)
set.seed(121)
sourceCpp("C/calc_slope.cpp")
sourceCpp("C/linreg.cpp")
sourceCpp("C/lr_coefficient_CI.cpp")
source("R/calc_slope.R")
source("R/linreg.R")
source("R/lr_coefficient_CI.R")
```

```r
Random = runif(200000,0,100)
AT = data.frame(x = 1:200000, y = 1:200000 + Random)


x1 = runif(200000,0,1000) + runif(200000, 500, 1000)
x2 = runif(200000,0,2000) + runif(200000, 1000, 2000)
x3 = runif(200000,2000,3000) + runif(200000, 1000, 2000)
x4 = runif(200000,2500,2750) + runif(200000, 1000, 2000)
y = x1 + x2 + x3 + x4 + Random^2 + runif(20000, 500, 777)
xFrame = as.matrix(data.frame(x1,x2,x3,x4))




knitr::kable((benchmark("C++" = {calc_slopeC(AT)},
          "R" = {calc_slope(AT)},
          replications = 1000,
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
```

test	replications	elapsed	relative	user.self	sys.self
C++	1000	1.64	1.000	1.22	0.39
R	1000	5.05	3.079	3.75	1.23

```r
#p = 4
knitr::kable((benchmark("lm" = {lm(y ~ xFrame)},
          ".lm.fit" = {.lm.fit(as.matrix(data.frame(1, xFrame)), y)},
          "lm.fit" = {lm.fit(as.matrix(data.frame(1, xFrame)), y)},
          "C++ without std::inner_product" = {linear_regC(xFrame, y)},
          "C++ with std::inner_product" = {linear_regC2(xFrame, y)},
          "C++ with RcppArmadillo" = {linear_regC3(xFrame, y)},
          "R" = {linear_reg(xFrame, y)},
            replications = 100,
            columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
```

	test	replications	elapsed	relative	user.self	sys.self
2	.lm.fit	100	3.78	1.405	3.00	0.78
6	C++ with RcppArmadillo	100	2.69	1.000	1.66	1.03
5	C++ with std::inner_product	100	14.36	5.338	13.70	0.63
4	C++ without std::inner_product	100	4.89	1.818	4.36	0.50
1	lm	100	9.29	3.454	7.57	1.64
3	lm.fit	100	4.50	1.673	3.59	0.91
7	R	100	3.92	1.457	3.14	0.78

```r
# p =1
knitr::kable((benchmark("lm" = {lm(AT[[2]] ~ AT[[1]])},
          ".lm.fit" = {.lm.fit(as.matrix(data.frame(1, AT[[1]])), as.vector(AT[[2]]))},
          "lm.fit" = {lm.fit(as.matrix(data.frame(1, AT[[1]])), as.vector(AT[[2]]))},
          "C++ without std::inner_product" = {linear_regC(as.matrix(AT[[1]]), as.vector(AT[[2]]))},
          "C++ with std::inner_product" = {linear_regC2(as.matrix(AT[[1]]), as.vector(AT[[2]]))},
          "C++ with RcppArmadillo" = {linear_regC3(as.matrix(AT[[1]]), as.vector(AT[[2]]))},
          "R" = {linear_reg(as.matrix(AT[[1]]), as.vector(AT[[2]]))},
          replications = 100,
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
```

	test	replications	elapsed	relative	user.self	sys.self
2	.lm.fit	100	1.83	1.220	1.49	0.33
6	C++ with RcppArmadillo	100	1.50	1.000	0.84	0.66
5	C++ with std::inner_product	100	5.08	3.387	4.53	0.55
4	C++ without std::inner_product	100	1.58	1.053	1.14	0.42
1	lm	100	6.67	4.447	5.64	0.98
3	lm.fit	100	2.29	1.527	1.86	0.41
7	R	100	1.90	1.267	1.38	0.53

```r
#p = 40
x1 = runif(8000000,0,1000) + runif(8000000, 500, 1000)
FourtyX = matrix(x1, nrow = 200000, ncol = 40)
knitr::kable((benchmark("lm" = {lm(y ~ FourtyX)},
          ".lm.fit" = {.lm.fit(as.matrix(data.frame(1, FourtyX)), y)},
          "lm.fit" = {lm.fit(as.matrix(data.frame(1, FourtyX)), y)},
          "C++ without std::inner_product" = {linear_regC(FourtyX, y)},
          "C++ with std::inner_product" = {linear_regC2(FourtyX, y)},
          "C++ with RcppArmadillo" = {linear_regC3(FourtyX, y)},
          "R" = {linear_reg(FourtyX, y)},
          replications = 1,
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
```

	test	replications	elapsed	relative	user.self	sys.self
2	.lm.fit	1	0.62	1.676	0.58	0.05
6	C++ with RcppArmadillo	1	0.37	1.000	0.31	0.07
5	C++ with std::inner_product	1	5.26	14.216	5.25	0.00

	test	replications	elapsed	relative	user.self	sys.self
4	C++ without std::inner_product	1	5.45	14.730	5.39	0.03
1	lm	1	0.75	2.027	0.72	0.03
3	lm.fit	1	0.62	1.676	0.58	0.05
7	R	1	0.51	1.378	0.50	0.01

```r
knitr::kable(benchmark("%*%" = {t(FourtyX) %*% FourtyX},
          "multiply without std::inner_product" = {multiply(t(FourtyX), FourtyX)},
          "multiply with std::inner_product" = {multiply2(t(FourtyX), FourtyX)},
          "multiply with RcppArmadillo" = {armamultiply(t(FourtyX), FourtyX)},
          replications = 1,
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))
```

	test	replications	elapsed	relative	user.self	sys.self
1	%*%	1	0.25	1.00	0.25	0.00
4	multiply with RcppArmadillo	1	0.34	1.36	0.25	0.09
3	multiply with std::inner_product	1	2.48	9.92	2.42	0.02
2	multiply without std::inner_product	1	4.56	18.24	4.45	0.01

```r
knitr::kable(benchmark("%*%" = {t(xFrame) %*% xFrame},
          "multiply without std::inner_product" = {multiply(t(xFrame), xFrame)},
          "multiply with std::inner_product" = {multiply2(t(xFrame), xFrame)},
          "multiply with RcppArmadillo" = {armamultiply(t(xFrame), xFrame)},
          replications = 100,
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))
```

	test	replications	elapsed	relative	user.self	sys.self
1	%*%	100	1.27	1.024	1.03	0.18
4	multiply with RcppArmadillo	100	1.63	1.315	0.98	0.62
3	multiply with std::inner_product	100	3.06	2.468	2.75	0.31
2	multiply without std::inner_product	100	1.24	1.000	1.08	0.15

```r
knitr::kable(benchmark("C++ using Boost" = {tc(0.99, 55)},
          "C++ calling R" = {tr(0.99,55)},
          "R" = {qt(0.99, 55)},
          replications = 100000),
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))
```

	test	replications	elapsed	relative	user.self	sys.self	user.child	sys.child
2	C++ calling R	1e+05	2.95	5.784	2.94	0.00	NA	NA
1	C++ using Boost	1e+05	0.70	1.373	0.68	0.01	NA	NA
3	R	1e+05	0.51	1.000	0.52	0.00	NA	NA

```r
z = linear_reg(FourtyX, y)
knitr::kable(benchmark("C++" = {lr_coefficient_CI_C(z, 0.95)},
          "R" = {lr_coefficient_CI(z, 0.95)},
          replications = 10000),
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))
```

test	replications	elapsed	relative	user.self	sys.self	user.child	sys.child
C++	10000	0.11	1.000	0.09	0.02	NA	NA
R	10000	0.30	2.727	0.29	0.00	NA	NA