TestingRandC

William Shih

3/17/2020

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

Benchmark calculating slope of p = 1, n = 200,000 with 1000 replications

	test	replications	elapsed	relative	user.self	sys.self
1	C++	1000	2.04	1.000	1.31	0.42
4	R	1000	4.66	2.284	3.56	1.10
2	lm.fit$coefficients[[2]]	1000	20.96	10.275	16.18	4.09
3	lm$coefficients[[2]]	1000	59.22	29.029	49.52	9.36

Benchmark multiplying 40*200000 by 200000*40 matrix

	test	replications	elapsed	relative	user.self	sys.self
4	C++ with RcppArmadillo	1	0.35	1.000	0.30	0.05
1	%*%	1	0.47	1.343	0.39	0.01
3	C++ with std::inner_product	1	2.69	7.686	2.65	0.00
2	C++ without std::inner_product	1	5.55	15.857	5.30	0.05

Benchmark multiplying 4*200000 by 200000*4 matrix

	test	replications	elapsed	relative	user.self	sys.self
1	%*%	100	0.99	1.000	0.86	0.12
2	C++ without std::inner_product	100	1.30	1.313	1.21	0.09
4	C++ with RcppArmadillo	100	2.30	2.323	1.22	0.81
3	C++ with std::inner_product	100	3.39	3.424	3.11	0.18

Benchmark inverting 40*40 matrix

	test	replications	elapsed	relative	user.self	sys.self
2	C++ with RcppArmadillo	10000	0.54	1.000	0.55	0.00
3	C++ with Gauss-Jordan Elimination	10000	1.48	2.741	1.43	0.03
1	solve	10000	1.80	3.333	1.59	0.06

Benchmark p = 400 and n = 200,000 Linear Regression with 100 replications

	test	replications	elapsed	relative	user.self	sys.self
6	C++ with RcppArmadillo	100	2.70	1.000	1.56	1.14
2	.lm.fit	100	3.80	1.407	3.03	0.70
7	R	100	3.93	1.456	3.11	0.81
3	lm.fit	100	4.53	1.678	3.58	0.94
4	C++ without std::inner_product	100	4.93	1.826	4.49	0.42
1	lm	100	9.27	3.433	7.79	1.45
5	C++ with std::inner_product	100	14.36	5.319	13.56	0.77

Benchmark p = 1 and n = 200,000 Linear Regression with 100 replications

	test	replications	elapsed	relative	user.self	sys.self
6	C++ with RcppArmadillo	100	1.53	1.000	1.03	0.50
4	C++ without std::inner_product	100	1.84	1.203	1.20	0.58
7	R	100	1.94	1.268	1.42	0.50
2	.lm.fit	100	2.22	1.451	1.64	0.45
3	lm.fit	100	2.54	1.660	1.92	0.54
5	C++ with std::inner_product	100	5.23	3.418	4.64	0.56
1	lm	100	6.63	4.333	5.56	1.05

Benchmark p = 40 and n = 200,000 Linear Regression with 1 replication

	test	replications	elapsed	relative	user.self	sys.self
6	C++ with RcppArmadillo	1	0.37	1.000	0.30	0.08
7	R	1	0.54	1.459	0.53	0.02
3	lm.fit	1	0.64	1.730	0.62	0.02
2	.lm.fit	1	0.77	2.081	0.73	0.02
1	lm	1	0.85	2.297	0.75	0.08
5	C++ with std::inner_product	1	5.53	14.946	5.45	0.04
4	C++ without std::inner_product	1	5.89	15.919	5.76	0.01

Benchmark calling t distribution with 100,000 replications

	test	replications	elapsed	relative	user.self	sys.self
3	R	1e+05	0.54	1.000	0.53	0
1	C++ using Boost	1e+05	0.75	1.389	0.75	0
2	C++ calling R	1e+05	3.14	5.815	3.09	0

Benchmark 95% confidence interval of linear regression (p = 30, n = 200,000) result with 10,000 replications

test	replications	elapsed	relative	user.self	sys.self
C++	10000	0.11	1.000	0.09	0.02
R	10000	0.33	3.000	0.33	0.00
confint.lm	10000	146.38	1330.727	123.83	19.28

Benchmark 95% prediction interval of linear regression (p = 30, n = 200,000) result with 10,000 replications

test	replications	elapsed	relative	user.self	sys.self
C++ with RcppArmadillo	10000	0.12	1.000	0.08	0.05
C++ with std::inner_product	10000	0.14	1.167	0.14	0.00
R	10000	0.26	2.167	0.25	0.00
predict.lm	10000	32.05	267.083	26.91	4.39

Code Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(Rcpp)
library(RcppArmadillo)
library(rbenchmark)
set.seed(121)
sourceCpp("C/calc_slope.cpp")
sourceCpp("C/linreg.cpp")
sourceCpp("C/lr_coefficient_CI.cpp")
sourceCpp("C/lr_prediction_interval.cpp")
source("R/calc_slope.R")
source("R/linreg.R")
source("R/lr_coefficient_CI.R")
source("R/lr_prediction_interval.R")

Random = runif(200000,0,100000)
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))
x1 = runif(8000000,0,1000) + runif(8000000, 500, 1181)
FourtyX = matrix(x1, nrow = 200000, ncol = 40)
knitr::kable((benchmark("C++" = {calc_slopeC(AT)},
          "lm.fit$coefficients[[2]]" = {lm.fit(as.matrix(data.frame(1, AT[[1]])), AT[[2]])$coefficients
          "lm$coefficients[[2]]" = {lm(y ~ x, data = AT)$coefficients[[2]]},
          "R" = {calc_slope(AT)},
```

```
          replications = 1000, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
knitr::kable(benchmark("%*%" = {t(FourtyX) %*% FourtyX},
          "C++ without std::inner_product" = {multiply(t(FourtyX), FourtyX)},
          "C++ with std::inner_product" = {multiply2(t(FourtyX), FourtyX)},
          "C++ with RcppArmadillo" = {armamultiply(t(FourtyX), FourtyX)},
          replications = 1, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))
knitr::kable(benchmark("%*%" = {t(xFrame) %*% xFrame},
          "C++ without std::inner_product" = {multiply(t(xFrame), xFrame)},
          "C++ with std::inner_product" = {multiply2(t(xFrame), xFrame)},
          "C++ with RcppArmadillo" = {armamultiply(t(xFrame), xFrame)},
          replications = 100, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))

InvertThis = t(FourtyX) %*% FourtyX
knitr::kable(benchmark("solve" = {solve(InvertThis)},
          "C++ with RcppArmadillo" = {armainverse(InvertThis)},
          "C++ with Gauss-Jordan Elimination" = {inverse(InvertThis)},
          replications = 10000, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))
#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, order = "elapsed",
            columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
# 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, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
#p = 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, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self"))))
knitr::kable(benchmark("C++ using Boost" = {tc(0.99, 55)},
          "C++ calling R" = {tr(0.99,55)},
```

```
          "R" = {qt(0.99, 55)},
          replications = 100000, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))

ThirtyX = FourtyX[,1:30]
colnames(ThirtyX) = c(paste0("X",1:30))
l = lm(y ~ ., data = data.frame(cbind(ThirtyX,y)))
l$effects = NULL
z = linear_reg(ThirtyX, y)

knitr::kable(benchmark("C++" = {lr_coefficient_CI_C(z, 0.95)},
          "R" = {lr_coefficient_CI(z, 0.95)},
          "confint.lm" = {confint.lm(l)},
          replications = 10000, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))

data = runif(30,0,10)
data = data.frame(t(data))
datanum = as.numeric(data)
colnames(data) = names(l$coefficients)[-1]
knitr::kable(benchmark("C++ with RcppArmadillo" = {lr_prediction_interval_C(z, datanum, 0.05)},
          "C++ with std::inner_product" = {lr_prediction_interval_C2(z, datanum, 0.05)},
          "R" = {lr_prediction_interval(z, datanum, 0.05)},
          "predict.lm" = {predict(l, data, interval = "prediction", level = 0.95)},
          replications = 10000, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))
```