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	1.51	1.000	0.93	0.58
4	R	1000	4.44	2.940	3.28	1.09
2	lm.fit$coefficients[[2]]	1000	20.18	13.364	15.25	4.20
3	lm$coefficients[[2]]	1000	57.85	38.311	48.14	9.06

Benchmark multiplying 40*200000 by 200000*40 matrix with 5 replications

	test	replications	elapsed	relative	user.self	sys.self
1	%*%	5	1.52	1.000	1.44	0.06
4	C++ with RcppArmadillo	5	1.61	1.059	1.28	0.33
3	C++ with std::inner_product	5	2.64	1.737	2.50	0.14
2	C++ without std::inner_product	5	22.33	14.691	22.19	0.03

Benchmark multiplying 4*200000 by 200000*4 matrix

	test	replications	elapsed	relative	user.self	sys.self
1	%*%	100	0.95	1.000	0.77	0.19
3	C++ with std::inner_product	100	1.17	1.232	0.83	0.35
2	C++ without std::inner_product	100	1.42	1.495	1.19	0.15
4	C++ with RcppArmadillo	100	1.48	1.558	0.80	0.69

Benchmark inverting 40*40 matrix with 100,000 replications

	test	replications	elapsed	relative	user.self	sys.self
2	C++ with RcppArmadillo	1e+05	4.89	1.000	4.75	0.08
3	C++ with Gauss-Jordan Elimination	1e+05	12.02	2.458	12.02	0.00
1	solve	1e+05	13.52	2.765	13.08	0.06

Benchmark p = 4 and n = 200,000 Linear Regression with 100 replications

	test	replications	elapsed	relative	user.self	sys.self
6	C++ with RcppArmadillo	100	2.45	1.000	1.40	1.05
7	R	100	2.84	1.159	2.77	0.06
3	lm.fit	100	3.30	1.347	3.24	0.06
2	.lm.fit	100	3.61	1.473	3.47	0.03
4	C++ without std::inner_product	100	3.91	1.596	3.89	0.01
5	C++ with std::inner_product	100	5.12	2.090	4.91	0.19
1	lm	100	7.65	3.122	7.41	0.11

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

	test	replications	elapsed	relative	user.self	sys.self
4	C++ without std::inner_product	200	2.35	1.000	2.23	0.01
2	.lm.fit	200	2.41	1.026	2.40	0.00
7	R	200	2.49	1.060	2.45	0.03
6	C++ with RcppArmadillo	200	2.53	1.077	1.61	0.93
3	lm.fit	200	3.29	1.400	3.17	0.05
5	C++ with std::inner_product	200	3.31	1.409	3.29	0.01
1	lm	200	10.52	4.477	10.31	0.09

Benchmark p = 40 and n = 200,000 Linear Regression with 5 replications

	test	replications	elapsed	relative	user.self	sys.self
6	C++ with RcppArmadillo	5	1.78	1.000	1.52	0.26
7	R	5	2.67	1.500	2.43	0.25
3	lm.fit	5	3.00	1.685	2.91	0.09
2	.lm.fit	5	3.08	1.730	2.96	0.12
1	lm	5	3.77	2.118	3.59	0.11
5	C++ with std::inner_product	5	6.69	3.758	6.43	0.23
4	C++ without std::inner_product	5	26.44	14.854	26.26	0.12

Benchmark calling t distribution with 100,000 replications

	test	replications	elapsed	relative	user.self	sys.self
3	R	1e+05	0.52	1.000	0.52	0
1	C++ using Boost	1e+05	0.70	1.346	0.67	0
2	C++ calling R	1e+05	3.04	5.846	3.05	0

Benchmark 95% confidence interval of linear regression (p = 20, n = 200,000) result with 10,000 replications

test	replications	elapsed	relative	user.self	sys.self
C++	10000	0.12	1.000	0.13	0.00
R	10000	0.30	2.500	0.30	0.00
confint.lm	10000	120.61	1005.083	111.13	8.89

Benchmark 95% prediction interval of linear regression (p = 20, n = 200,000) result with 10,000 replications

	test	replications	elapsed	relative	user.self	sys.self
2	C++ with std::inner_product	10000	0.11	1.000	0.11	0.00
1	C++ with RcppArmadillo	10000	0.13	1.182	0.09	0.02
3	R	10000	0.33	3.000	0.32	0.00
4	predict.lm	10000	22.84	207.636	22.32	0.05

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 = 5, 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 = 100000, 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 = 200, 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 = 5, 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[,1:20]
colnames(ThirtyX) = c(paste0("X",1:20))
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, 0.05)},
          "R" = {lr_coefficient_CI(z, 0.05)},
          "confint.lm" = {confint.lm(l)},
          replications = 10000, order = "elapsed",
          columns = c("test","replications","elapsed","relative","user.self","sys.self")))

data = runif(21,0,10)
data = data.frame(t(data))
datanum = as.numeric(data)[-1]
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")))
```