STAT-S610
Final Project Writings, 2025

Updating a function from EM Algorithm to make it faster

Nahid Hasan Sagar

Indiana University Bloomington

College of Arts and Sciences, Department of Statistics

December 13, 2025

Abstract

While learning R in STAT-S610, I realized that R is very good at specific tasks but not very good at computation or efficiency. That is why it sometimes takes a long time to run computationally intensive analyses in R. People try to overcome this hurdle by developing parts of the code in C++, and I am trying to do precisely that with this project of mine.

I am trying to develop a C++ function using RcppArmadillo for the BayesianBrainMap package. This package is used in the analysis of fMRI data, especially time-series data, to map the Human brain using Bayesian techniques. Like many Bayesian techniques, it requires multiple runs of the same function to achieve better results, making this a very computationally intensive task. So re-coding some of the computation-intensive functions in this package will make it much more time-efficient, saving significant computation time and energy.

Here, for this project, I am working with only one function. The Bayesian Brain Map (BBM) package uses the EM algorithm, and there is a function named 'UpdateTheta_BrainMap.independent'. It takes a few matrices as input, performs the calculations, and returns a new theta, which is used in multiple places to perform the analysis. I am here trying to replicate this function's output and return the exactly same theta value using the same calculation, but in C++ using RcppArmadillo. And there is a use of 'for' loop in the function, which is not very efficient in R by Replicating the same thing in C++ will increase the efficiency of the code and reduce the computation time.

Keywords
RcppArmadillo, BayesianBrainMap, EM Algorithm

1 Introduction

At the beginning of the process, the main challenge was to use the BBM package to obtain the test value of the input variable for the 'UpdateTheta_BrainMap.independent' function. The next challenge was to run the function in the BBM package and obtain the theta values. The last part of the challenge was to replicate the same result using the same input with the function developed in C++ using RcppArmadillo.

To run the BBM package, the Human Connectome Project data were obtained from the IU data center. Then the browser is used to observe the variable's input value at each step, gather test data, and verify the function's output theta against the one developed with RcppArmadillo.

Then Rcpp and RcppArmadillo were use to develop the new version of the 'UpdateTheta_BrainMap.independent' function. It was then tested using the test values accumulated in the earlier process, and the replication was completed.

The final step was to use this function in the main BBM package and compare its run time with the original function that was built in R. Since there were quite a few uses of the 'for' loop, and the 'for' loop makes R much slower, this new development for the function 'UpdateTheta_BrainMap. Independent should be significantly faster.

One final note is that, instead of using Rcpp and RcppArmadillo, if I had used raw C++,

then the function would have been more versatile. That means it can be used elsewhere if necessary, not just inside an R environment. Also, the result would have been a little faster since Rcpp and RcppArmadillo themselves use some R functions that sacrifice some computation time. But since I was under a time constraint, I would not have enough time to complete the whole project, including development and testing.

2 Materials & Methods

To run the BBM package, I used the Human Connectome Project (HCP) dataset from the IU data center, and the calculations were performed on IU's supercomputer, the Research Desktop, also known as the RED.

Using the dataset, the first hurdle was overcome. Even though it was running on the RED with 5 cores and 20GB of RAM per core, totaling 100GB of RAM. Because of the heavy, calculation-intensive nature of the Bayesian method, it took almost 20 minutes per subject just to run.

But the caveat was that I ran it in the cloud, where I could not access the argument information I needed to test my developed function. I needed to run it locally, and even though that should not have been a challenging task given the time frame, I was unable to do so because I had already spent a lot of time learning to implement Rcpp and RcppArmadillo to convert the function into C++.

So, I made a slight change to the plan and used a large language model [1] to generate some test data, with which I can run the UpdateTheta_BrainMap.independent and my developed UpdateTheta_BrainMap_independent_cpp functions, and compare the outputs. So, I was able to successfully replicate the output with both of the functions. The code is shown below to demonstrate how I did that using the example dimensions for the test.

```
# required packages
#install.packages("Rcpp")
#install.packages("RcppArmadillo")
library(Rcpp)
library(RcppArmadillo)

# Dimensions for test example
nV <- 50   # number of voxels
nL <- 30   # number of latent components
nT <- 40   # number of time points

prior_mean <- matrix(rnorm(nV * nL), nV, nL)
prior_var  <- matrix(rexp(nV * nL, rate = 1), nV, nL)
BOLD       <- matrix(rnorm(nV * nT), nV, nT)

theta <- list(
  A = matrix(rnorm(nT * nL), nT, nL),
  nu0_sq = 1.5
)

C_diag <- runif(nT, 0.5, 2)

H <- NULL
Hinv <- NULL

source("UpdateTheta_BrainMap.independent.R")
out <- UpdateTheta_BrainMap.independent(
  prior_mean = prior_mean,
  prior_var  = prior_var,
  BOLD       = BOLD,
  theta      = theta,
  C_diag     = C_diag,
  H          = H,
  Hinv       = Hinv,
  update_nu0sq = TRUE,
  return_MAP = FALSE,
  verbose = TRUE
)
out

Rcpp::sourceCpp("updateTheta_BrainMap_independent.cpp")

out2 <- UpdateTheta_BrainMap_independent_cpp(
  prior_mean = prior_mean,
  prior_var  = prior_var,
  BOLD       = BOLD,
  theta      = theta,
  C_diag     = C_diag,
  H          = H,
  Hinv       = Hinv,
  update_nu0sq = TRUE,
  return_MAP = FALSE,
  verbose = TRUE
)
out2
```

Then the replication was checked using the lines of code below

```
tol <- 1e-8  # numerical tolerance

# Helper function for matrix comparison
compare_matrix <- function(M1, M2, name) {
  if (!all(dim(M1) == dim(M2))) {
    cat(name, ": DIMENSION MISMATCH\n")
    return(FALSE)
  }
  max_diff <- max(abs(M1 - M2))
  cat(sprintf("%s: max | diff | = %.3e\n",
      name, max_diff))
  max_diff < tol
}

# Helper function for scalar comparison
compare_scalar <- function(x1, x2, name) {
  diff <- abs(x1 - x2)
  cat(sprintf("%s: | diff | = %.3e\n",
      name, diff))
  diff < tol
}

# Helper function for 3D array comparison
```

```r
compare_array3 <- function(A1, A2, name) {
  if (!all(dim(A1) == dim(A2))) {
    cat(name, ": DIMENSION MISMATCH\n")
    return(FALSE)
  }
  max_diff <- max(abs(A1 - A2))
  cat(sprintf("%s: max |diff| = %.3e\n",
      name, max_diff))
  max_diff < tol
}

## -------------------------
## 1. Compare A
## -------------------------
A_ok <- compare_matrix(out$A, out2$A, "A")

## -------------------------
## 2. Compare nu0_sq
## -------------------------
nu0_ok <- compare_scalar(out$nu0_sq,
    out2$nu0_sq, "nu0_sq")

## -------------------------
## 3. Compare Estep components
## -------------------------

E_v_inv_ok <- compare_matrix(
  out$Estep$E_v_inv,
  out2$Estep$E_v_inv,
  "Estep$E_v_inv"
)

Sigma_ok <- compare_matrix(
  out$Estep$Sigma_s_v,
  out2$Estep$Sigma_s_v,
  "Estep$Sigma_s_v"
)

miu_s_ok <- compare_matrix(
  out$Estep$miu_s,
  out2$Estep$miu_s,
  "Estep$miu_s"
)

miu_ssT_ok <- compare_array3(
  out$Estep$miu_ssT,
  out2$Estep$miu_ssT,
  "Estep$miu_ssT"
)

var_s_ok <- compare_matrix(
  out$Estep$var_s,
  out2$Estep$var_s,
  "Estep$var_s"
)

## -------------------------
## 4. Final summary
## -------------------------
cat("\n====== SUMMARY ======\n")
results <- c(
  A = A_ok,
  nu0_sq = nu0_ok,
  E_v_inv = E_v_inv_ok,
  Sigma_s_v = Sigma_ok,
  miu_s = miu_s_ok,
  miu_ssT = miu_ssT_ok,
  var_s = var_s_ok
)

print(results)

if (all(results)) {
  cat("SUCCESS: All outputs match within tolerance.\n")
} else {
  cat("WARNING: Some outputs do NOT match.\n")
}
```

After that, the micro-benchmark was used to compare the relative speeds of the functions with a default (100)-iteration count.

Listing 1: Microbenchmark setup for runtime comparison.

```r
library(microbenchmark)
library(Rcpp)

source("UpdateTheta_BrainMap.independent.R")
Rcpp::sourceCpp("updateTheta_BrainMap_independent.cpp")

microbenchmark(
  R = UpdateTheta_BrainMap.independent(
    prior_mean, prior_var, BOLD, theta, C_diag,
    H, Hinv, update_nu0sq = TRUE, return_MAP = FALSE
  ),
  Rcpp = UpdateTheta_BrainMap_independent_cpp(
    prior_mean, prior_var, BOLD, theta, C_diag,
    H, Hinv, update_nu0sq = TRUE, return_MAP = FALSE
  ),
  times = 100
)
```

3 Results

The runtime performance of the original R implementation and the C++ implementation of `UpdateTheta_BrainMap.independent` using RcppArmadillo was evaluated using microbenchmarking with 100 iterations across identical input dimensions. The results are summarized in Table 1.

Across all summary statistics, the Rcpp implementation consistently outperformed the original R version. In particular, the median runtime decreased from 10.77 ms in R to 2.73 ms in C++, corresponding to an approximate fourfold speedup. The maximum runtime was also substantially reduced, indicating improved stability and reduced sensitivity to occasional slow executions.

These results demonstrate that reimplementing the most computationally intensive components of the EM algorithm in C++ can yield substantial performance gains without altering the numerical output. Such improvements are especially valuable in Bayesian

Table 1: Runtime comparison between R and Rcpp implementations of the `UpdateTheta_BrainMap.independent` algorithm.

Implementation	Min (ms)	Median (ms)	Mean (ms)	Max (ms)
R	10.30	10.77	15.37	270.62
Rcpp (C++)	2.57	2.73	2.88	14.01

fMRI analyses, where this function is executed repeatedly across voxels and iterations.

Conclusions

This project demonstrates that substantial computational gains can be achieved by re-implementing performance-critical components of an EM algorithm in C++ using RcppArmadillo. By converting the function `UpdateTheta_BrainMap.independent` from its original R implementation to a C++ version while preserving identical numerical results, the runtime was reduced by approximately a factor of four under identical benchmarking conditions.

The observed performance improvement is primarily attributable to more efficient handling of nested loops, matrix algebra, and memory management in C++, which are known bottlenecks in interpreted R code. Since the EM algorithm requires repeated evaluation of the same update steps across many iterations and voxels, even modest per-iteration speedups translate into significant overall reductions in computation time. This is particularly important in neuroimaging applications, where datasets are enormous and computational demands are high.

Beyond the immediate performance gains, this project highlights the practical value of integrating R with lower-level languages for scientific computing. RcppArmadillo provides a flexible interface that allows developers to retain R's usability while leveraging the efficiency of optimized C++ linear algebra routines. The ability to seamlessly replace an R function with a compiled alternative makes this approach especially attractive for existing packages such as `BayesianBrainMap`, where performance improvements can be achieved without altering the user-facing workflow.

There are several directions for future work. First, additional functions within the BayesianBrainMap package could be profiled and selectively rewritten in C++ to further reduce runtime. Second, numerical stability and memory usage could be investigated for larger datasets and higher-dimensional models. Finally, extending the implementation to exploit parallelism or GPU acceleration may yield further performance improvements for large-scale neuroimaging analyses.

Overall, this project illustrates how targeted use of RcppArmadillo can significantly improve the efficiency of Bayesian algorithms while maintaining correctness and reproducibility, making it a valuable tool for computational statistics and applied data analysis.

Discussion

The challenge of doing this project was learning new stuff. At first, I thought my current C++ knowledge would be sufficient to complete the project. But as I proceed through the project, I need to learn new things and implement them step by step. Because of this step-by-step approach, I didn't see any output until the very last day before our presentation.

Then I saw the output and thought I could replicate it with my newly developed C++ version of the function. But then, when I looked more closely, I saw a mismatch, and at that point I felt again that I might not be able to fix it in time. But I kept working, then wrote a few lines of code to check the match, and I found that quite a few of the outputs weren't matched.

Then I started debugging the main code where a mismatch occurred. And after spending a whole day nicking the code, I finally managed to replicate the output within the tolerance I set.

I could have done the final project the conventional way, started it a week before the final, and gotten it done with that. Still, since I chose to do something with real-world applications, I started months ago and kept learning and working on it, which at least yielded a positive result in the end. It's feeling terrific in the end. Even though I still have to run the test on the real data set from the Human Connectome Project, I am hopeful it will work since it has been working with my test data set.

It has been a terrific journey throughout the project. I look forward to doing more of this challenging work in the future.

Acknowledgments

Thank you, Prof. Julia Fukuyama, Mason Griswold, and Paul Hunt, for helping me out and

keeping me going on the project. Also, thank you, Dr. Amanda Mejia, for allowing me to work on her Bayesian Brain Map package.

A Code Availability

All source code developed for this project, including the original R implementation of `UpdateTheta_BrainMap.independent`, the RcppArmadillo-based C++ implementation, numerical validation scripts, and benchmarking code, is publicly available on GitHub:

github.com/Sagar-DU/S610-FinalProject

The repository contains:

- The original R implementation of the EM update step
- The optimized C++ implementation using RcppArmadillo
- Scripts used for numerical equivalence testing
- Benchmarking code used to generate the runtime comparisons

This repository ensures the full reproducibility of the results presented in this report and provides a foundation for future extensions of the `BayesianBrainMap` package.

References

[1] OpenAI. Chatgpt (version 5.1). https://chat.openai.com/, 2025. Large language model.