Faster Computation for Hierarchical Bayesian Models with Rcpp Packages

Lu Chen∗, #, Balgobin Nandram+, Nathan B. Cruze#

*National Institute of Statistical Sciences (NISS)
+Worcester Polytechnic Institute
#United States Department of Agriculture
National Agricultural Statistics Service (NASS)

GASP II
Washington, DC
September 23, 2019
Disclaimer and Acknowledgment

The findings and conclusions in this presentation are those of the authors and should not be construed to represent any official USDA or US Government determination or policy.

- This research was supported in part by the intramural research program of the U.S. Department of Agriculture, National Agriculture Statistics Service.
Outline

Motivation

Introduction of Rcpp Packages

Examples

Conclusion
Motivation

"Sometimes R code just isn’t fast enough. “

– Hadley Wickham

- Problem: How to combine survey and auxiliary data to improve the county-level estimates for crops?
- Application: Bayesian small area models
- Potential bottlenecks of R code: subsequent iterations, repeatedly calling functions, loops in Markov chain Monte Carlo (MCMC) algorithms
- One solution: rewriting key functions in C++ through **Rcpp** packages
Rcpp Packages

- **Rcpp** is a R package to extend R with C++ codes developed by Dirk Eddelbuettel and Romain Francois (2013).
  - Speed
  - New Things

- **RcppArmadillo** is a Rcpp extension package that provides all the functionality of Armadillo, focusing on **matrix math**.
  - Easy-to-use
  - Further speedup
Using sourceCpp() in R

- The Rcpp::sourceCpp function parses the C++ file (.cpp) and makes C++ functions available as R functions.

Example: Calculate mean of $x_i$, $i = 1, \ldots, 10^5$, where $x_i \sim U(0, 1)$.

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double meanC(NumericVector x) {
    int n = x.size();
    double total = 0;
    for(int i = 0; i < n; ++i) {
        total += x[i];
    }
    return total/n;
}
```

```r
Rcpp::sourceCpp('cpp1.cpp')
x <- runif(1e5)
benchmark(
    mean(x), #build-in R mean() function
    meanC(x) #C++ function
)
```
Using `sourceCpp()` in R

- The Rcpp::sourceCpp function parses the C++ file (.cpp) and makes C++ functions available as R functions.

Example: Calculate mean of \( x_i, i = 1, \ldots, 10^5 \), where \( x_i \sim U(0, 1) \).

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double meanC(NumericVector x) {
  int n = x.size();
  double total = 0;
  for(int i = 0; i < n; ++i) {
    total += x[i];
  }
  return total/n;
}
```

```r
Rcpp::sourceCpp('cpp1.cpp')
x <- runif(1e5)
microbenchmark(
  mean(x),  # built-in R mean() function
  meanC(x)  # C++ function
)
```
Comparison

Performance among Rcpp & R
Lower values are better.

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean</th>
<th>median</th>
<th>max</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcpp</td>
<td>105.432</td>
<td>151.448</td>
<td>109.209</td>
<td>228.249</td>
<td>100</td>
</tr>
<tr>
<td>R</td>
<td>210.845</td>
<td>237.125</td>
<td>235.366</td>
<td>396.785</td>
<td>100</td>
</tr>
</tbody>
</table>
MCMC in Bayesian Computation

- MCMC is a sampling method to draw random samples from distributions.
- Each random sample is used as a stepping stone to generate the next one (chains).
- Gibbs sampler, Metropolis-Hastings sampler and many others are widely used in Bayesian inference.
- Involve **loops** and **calling functions repeatedly** within loops.

Rcpp (C++) and RcppArmadillo are useful tools for efficient MCMC computation.
Simulated Data

Data: simulated planted acres data in Illinois (Nandram et al., 2019 and Battese et al., 1988)

- Survey estimates $\hat{\theta}_i$, $i = 1, \ldots, 102$
- Survey standard errors $\hat{\sigma}_i$, $i = 1, \ldots, 102$
- Covariates: corn and soybean planted acres from land observatory satellites (LANDSAT)
Fay-Herriot Model

Fay-Herriot Model (1979) in small area estimation:

\[ \hat{\theta}_i | \theta_i \overset{\text{ind}}{\sim} N(\theta_i, \hat{\sigma}_i^2), \]
\[ \theta_i | \beta, \delta^2 \overset{\text{ind}}{\sim} N(x'_i \beta, \delta^2), \quad i = 1, \ldots, n, \]

Priors for the parameters: \( \pi(\beta) \propto 1; \pi(\delta^2) \propto \frac{1}{\delta^2}. \)

The full conditional distributions for Gibbs sampling are:

1. \( \beta | \delta^2 \sim \text{MVN}\left( (\sum_{i=1}^{n} x_i x'_i)^{-1} (\sum_{i=1}^{n} x'_i \beta), \delta^2 (\sum_{i=1}^{n} x_i x'_i)^{-1} \right) \),
2. \( \theta_i | \beta, \delta^2 \overset{\text{ind}}{\sim} N(\lambda_i \hat{\theta}_i + (1 - \lambda_i) x'_i \beta, (1 - \lambda_i) \delta^2), \quad \lambda_i = \frac{\delta^2}{\delta^2 + \hat{\sigma}_i^2}, \)
3. \( \delta^2 | \theta, \beta \sim IG\left( \frac{n-1}{2}, \frac{1}{2} \sum_{i=1}^{n} (\theta_i - x'_i \beta)^2 \right) \).
Comparison

12,000 iterations with 2,000 burn-in and pick every 10\textsuperscript{th} sample

Performance among Rcpp & R

Lower values are better.

<table>
<thead>
<tr>
<th>name</th>
<th>min</th>
<th>mean</th>
<th>median</th>
<th>max</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcpp</td>
<td>94.863</td>
<td>97.285</td>
<td>96.621</td>
<td>106.362</td>
<td>100</td>
</tr>
<tr>
<td>R</td>
<td>4707.729</td>
<td>4856.196</td>
<td>4829.291</td>
<td>4912.025</td>
<td>100</td>
</tr>
</tbody>
</table>
Comparison - R density plots

Density of beta0

N = 1000  Bandwidth = 0.5677

Density of beta1

N = 1000  Bandwidth = 1.421

Density of beta2

N = 1000  Bandwidth = 1.275

Density of delta2

N = 1000  Bandwidth = 14.67
Comparison - Rcpp density plots

Density of beta0

N = 1000  Bandwidth = 0.5744

Density of beta1

N = 1000  Bandwidth = 1.367

Density of beta2

N = 1000  Bandwidth = 1.19

Density of delta2

N = 1000  Bandwidth = 13.9
Fay-Herriot Model with Benchmarking Constraints

In some applications, we need to incorporate benchmarking constraints into the model. For example, the county-level estimates should be summed to state target and they need to cover certain values. The model with inequality constraints:

\[
\hat{\theta}_i | \theta_i \overset{\text{ind}}{\sim} N(\theta_i, \hat{\sigma}_i^2), \quad i = 1, \ldots, n,
\]

\[
\theta_i | \beta, \delta^2 \overset{\text{ind}}{\sim} N(x_i' \beta, \delta^2), \quad \theta_i \geq c_i, \sum_{i=1}^n \theta_i \leq a,
\]

where \( C = (c_1, \ldots, c_n)' \) are known and fixed and \( a \) is state target. The priors are \( \pi(\beta) \propto 1; \delta^2 \propto \frac{1}{(1+\delta^2)^2}. \)
Joint Posterior Distribution

The posterior density is

\[ \pi(\theta, \beta, \delta^2 | \hat{\theta}, \hat{\sigma}^2) = \frac{\prod_{i=1}^{n} \phi((\theta_i - X'\beta)/\delta)\phi((\theta_i - \hat{\theta}_i)/\hat{\sigma}_i)}{\int_{\theta \in V} \prod_{i=1}^{n} \phi((\theta_i - X'\beta)/\delta) \, d\theta}, \quad \theta \in V, \]

where \(\phi(\cdot)\) is the standard normal density and the support of \(\theta\) is

\[ V = \left\{ c_i \leq \theta_i, \sum_{i=1}^{n} \theta_i \leq a, \ i = 1, \ldots, n \right\}. \]

Awkward joint posterior distribution and intractable.
Computation

Our strategy:

\[ \pi(\theta, \beta, \delta^2 | \hat{\theta}, \hat{\sigma}^2) = \pi(\beta, \delta^2 | \hat{\theta}, \hat{\sigma}^2) \times \pi(\theta | \beta, \delta^2, \hat{\theta}, \hat{\sigma}^2) \]

▶ Metropolis-Hastings Sampler
Our strategy:

$$\pi(\theta, \beta, \delta^2|\hat{\theta}, \hat{\sigma}^2) = \pi(\beta, \delta^2|\hat{\theta}, \hat{\sigma}^2) \times \pi(\theta|\beta, \delta^2, \hat{\theta}, \hat{\sigma}^2)$$

- Metropolis-Hastings Sampler
- Gibbs Sampler
Metropolis-Hastings Sampler

We will draw \((\beta, \delta^2)\) samples from \(\pi(\beta, \delta^2|\hat{\theta}, \hat{\sigma}^2)\). The proposal density is

\[
(\beta, \log(\delta^2)) \sim MVN(\hat{\beta}_p, \sigma^2\hat{\Sigma}_p)
\]

\[
\nu/\sigma^2 \sim \Gamma(\nu/2, 1/2)
\]

**Bottleneck:**

For each iteration \(h\):

- **Generate:** Generate a candidate \((\beta^c, \log(\delta^2)^c)\) from proposal density;

- **Calculate:** Calculate the acceptance ratio
  \[
  \alpha = \pi(\beta^c, \log(\delta^2)^c)/\pi(\beta^{(h)}, \log(\delta^2)^{(h)})
  \]

- **Accept or Reject** candidate based on the comparison between \(\alpha\) and \(u \sim U(0,1)\).
Comparison

10,000 iterations with 2,000 burn-in and pick every 8\textsuperscript{th} sample

Performance among Rcpp & R
Lower values are better.

<table>
<thead>
<tr>
<th>name</th>
<th>min</th>
<th>mean</th>
<th>median</th>
<th>max</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcpp</td>
<td>69.527</td>
<td>70.554</td>
<td>70.711</td>
<td>71.073</td>
<td>10</td>
</tr>
<tr>
<td>R</td>
<td>2451.688</td>
<td>2540.393</td>
<td>2519.215</td>
<td>2741.561</td>
<td>10</td>
</tr>
</tbody>
</table>
Gibbs Sampler for $\theta$

The conditional posterior density of $\theta_i$ is

$$\theta_i|\theta_{(i)}, \beta, \delta^2, \hat{\theta}, \hat{\sigma}^2 \sim N(\mu_i, \phi_i), \quad \theta_i \in V_i,$$

where $\mu_i$ and $\phi_i$ related to $\beta$ and $\delta^2$ and

$$V_i = \left\{ c_i \leq \theta_i \leq a - \sum_{j=1, j \neq i}^{n-1} \theta_j \right\}, \quad i = 1, \ldots, n.$$

**Bottleneck:**

Each $\theta_i$ related to other $\theta$s based on the $V_i$.

For one iteration, we need to loop $n$ times.
Comparison

Performance among Rcpp & R
Lower values are better.

<table>
<thead>
<tr>
<th>name</th>
<th>min</th>
<th>mean</th>
<th>median</th>
<th>max</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcpp</td>
<td>2.130</td>
<td>2.162</td>
<td>2.153</td>
<td>2.213</td>
<td>10</td>
</tr>
<tr>
<td>R</td>
<td>55.934</td>
<td>56.615</td>
<td>56.696</td>
<td>57.134</td>
<td>10</td>
</tr>
</tbody>
</table>
Convergence Diagnostics

- M-H: 1,000 samples of \((\beta^{(h)}, \delta^{2(h)})\), \(h = 1, \ldots, 1000\).
- Gibbs: for each \((\beta^{(h)}, \delta^{2(h)})\), we run 100 times Gibbs sampler and pick the last set of \(\theta\).

<table>
<thead>
<tr>
<th></th>
<th>pm</th>
<th>psd</th>
<th>lb</th>
<th>ub</th>
<th>gewe.pval</th>
<th>ess</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\beta_0)</td>
<td>116.22</td>
<td>2.01</td>
<td>112.43</td>
<td>120.06</td>
<td>0.45</td>
<td>909</td>
</tr>
<tr>
<td>(\beta_1)</td>
<td>16.74</td>
<td>4.58</td>
<td>7.75</td>
<td>25.29</td>
<td>0.40</td>
<td>870</td>
</tr>
<tr>
<td>(\beta_2)</td>
<td>-4.81</td>
<td>4.23</td>
<td>-13.14</td>
<td>3.35</td>
<td>0.24</td>
<td>968</td>
</tr>
<tr>
<td>(\delta^2)</td>
<td>325.96</td>
<td>60.78</td>
<td>206.98</td>
<td>469.46</td>
<td>0.64</td>
<td>827</td>
</tr>
</tbody>
</table>

- Rcpp vs R code: **72s** vs **2576s** for 102 samples size in the constraint Bayesian model.
Conclusion

▶ Rcpp functions can reduce the running time by a significant factor and reasonable in further production for county-level estimates in NASS.

▶ Large data set or complicated hierarchical Bayesian models: Rcpp packages
  ▶ Pros: incorporating C++ code into R workflow easily; substantially speed up MCMC computation in R
  ▶ Cons: learning curve and long coding time

▶ Small data set or simple, classic Bayesian models: such as RJags and RStan
  ▶ Pros: Easy-to-use; less coding time
  ▶ Cons: Black-box sampler; not for non-standard problems
Reference


Thank You!

lu.chen@usda.gov