Comparison of MCMC Algorithms Using Convergence Diagnostics, Parallelization, GGplot2

Aaron N. Baker
Department of Statistics
Iowa State University

Overview

- Basic Motivation/Concept
- Goals
- Example Use
- Output
- Results
- Future Work

Basic Motivation/Concept

- Motivation: Functions before parallelizing and cleaning up took 4 to 5 days to run.
- Compare specific MCMC methods using useful diagnostic methods for chain convergence
  - ACF plots
    - Lower ACF means less dependence, more valuable variance estimates for chains.
  - Gelman-Rubin (GR) Diagnostics
    - A value less than one suggests chain convergence
  - Geweke Statistics
    - Checks to see if the end and beginning sections of a chain are behaving similarly

Goals

- Improve former work by
  - Reproducability
  - Parallelization
  - Improving former code via good practices (removing inefficiencies, naming)
  - Provide diagnostics for comparing any two desired algorithms

Reproducibility

- Take functions and turn them into easily executable functions
- Post these to github for reproducibility purposes

Parallelization

- Use DoParallel, Foreach, to run multiple chains at the same time.
- learned that...
  - Parallelization can be annoying as all get out
  - Windows is a pain for parallelizing
  - Must ensure that functions and libraries are explicitly loaded onto each cluster
  - I couldn't get multidplyr to work for the life of me

Improving Code

- Got rid of ugly for loops and traded in for faster functions and matrix algebra
- Annotated the code
- Used better saving practices (github), project folders, and naming

Improving Code-Example

- Former Code: Bounds for truncated draws

  -
    ```
    > for(i in 1 : length(y)){
    > if(ys[i]*X[i,j] > 0){
    > mincount = mincount + 1
    > minim[mincount] = -(η[i]+X[i,-j]β[−j, ])/X[i,j]
    > }
    > if(ys[i]*X[i,j] < 0){
    > maxcount = maxcount + 1
    > maxim[maxcount] = -(η[i]+X[i,-j]β[−j, ])/X[i,j]
    > }
    > }
    ```
- One example of a for loop inside a for loop. Eegads! There were at least 8 similar such loops that were removed through careful consideration.

Improving Code-Example

- New Code: No loop.
  - bound = bounds2(y, β, x, n) where bounds2 is

    ```
    > bounds2 = function(y, Beta, x, n){
    > bounds = array(-9999, dim = c(n,2))
    > zeroes = which(y == 0)
    > bounds[zeroes,] = cbind(rep(-Inf,length(zeroes)),rep(0,
    length(zeroes)))
    > ones = which(y == 1)
    > bounds[ones,] = cbind(rep(0, length(ones)),rep(Inf, length(ones)))
    > return(bounds)
    > }
    ```
- That's better!
- Parallelized main loop, and destroyed the inner. For the win!

Improving Code-Example

- Even worse than for loops is for loops inside for loops.
- Example
  - If there are 50,000 chain values to produce and each iteration requires an inner loop of length 100 (the length of y in my example), the loop requires 50000*100 = 5,000,000 run-throughs!
- Significant increase in time efficiency by removing inner loop.

Input and Output

- Input is the following in a list, with * if optional
  - Initial Values* (Initials)
  - MCMC* - Custom made algorithm by user with chains in list format
  - M* - The number of chains you want to run per algorithm. Set to 5 by default
  - K - The length of each chain
  - x - The covariates
  - y - The response
  - Initialize* - Set equal to one if you want the function to generate intial values for you
  - Model* - If you want to run algorithm already defined in package
  - (nu, nunot, c)* - Values with defaults set for Robit pre-defined algorithms
  - pargroups* - The number of paramter groups (means/variances for example) you want to plot separately
  - Methodname - The name of the method you are running
  - GewInit* - The proportion of each chain in the first portion of the Geweke Statistic
  - GewEnd* - The proportion of each chain in the end portion of the

Input and Output

- Output is the following in a list
  - Chains
  - ACF values
  - Gelman Rubin values
  - Geweke values
  - initial starting values
- To R for an example!

Input and Output

- Results of this project
  - Became a ggplot2 green belt (never used before)
  - Learned parallelization
  - Created significantly more efficient code
  - Created a project that is reproducible and publicly available.
  - Created function for future reproducibility in convergence diagnostics for two competing algorithms.
- To R for an example!

Future Work

- Make sure there aren't any bugs for multiple parameter groups
- Include more diagnostics
- Other bells and whistles and stuff
- Create Shiny app, because it seems like all the cool kids are doing it

Questions?