Paper

Outline

- General Bayesian computational settings
  - Chains
    - Greta
  - Iterations
  - Warmup
    - STAN/Bugs uses warmup as a part of iterations, Jags/Greta/Nimble
  - Thinning
  - initial value specification
- Each sampler
  - BUGS
    - WINE
    - Debugging
  - Jags
  - Nimble
    - Crashes
  - STAN
    - RDS file
  - Greta
- Common coding techniques for each language
  - Truncated normal
  - Specifiying data and constants
  - Matrix multiplication
  - Initial value specification?
  - sigma / tau
- Benchmarking
  - Run time
  - Memory allocation
  - Computational accuracy
  - Benchmarking files
- Folder design
  - General model type
    - Simple conjugate models
    - Regression models
    - Time Series models
    - Survival models
    - 8 Schools
  - Specific model type
  - Scripts for each method
  - Benchmarking
- Here package
- Models Used

Bayesian Samplers

BUGS

OpenBugs is the oldest of the Bayesian samplers that we use. OpenBugs is designed for Windows machines, but there is a workaround to run it on MacOS and Linux, using WINE, a program designed to run Windows software on other operating systems. OpenBugs can be used through the actual click-and-point application, or through `R`, using the `R2OpenBugs` package, which allows for the user to send model scripts and data to OpenBugs without needing to leave the `R` interface. The `R` command to run OpenBugs is `bugs()`. Using this method causes OpenBugs to run invisibly in the background and return the results, unless an argument is specified to debug code in the actual application itself, which can be very useful. When using R to run OpenBugs, there is little way to know what went wrong if the code errors somewhere without interacting with the OpenBugs program itself. Using the `debug = T` argument within `bugs()` will cause the application to run in the forefront as if it were actually being used directly by the user. This way, any messages given by the application while running can be viewed directly. It is recommend to use `debug = T` only when debugging, as it does slow down the sampling process slightly. OpenBugs has many tools to visualize the performance the sampler which are easy to use. The biggest drawback of using the OpenBugs application directly is that specifiying the data can be tedious if the data is not very small in size or design. Using R to specify the data is much easier. One unique feature in OpenBugs is the Doodle tool, which allows for model specification through the creation of a diagram of nodes representing the parameters. Though using Doodle can be helpful for illustration, it is not used often in practice.

JAGS

In terms of model specification, JAGS is very similar to OpenBugs, but works more seemlessly in R using the package `rjags`. The `runjags` is an optional package that allows for the JAGS model script to be inputted as a string, and run using the `run.jags()` function. Since JAGS runs inside `R` itself, it is very easy to work with. Helpful errors are given when there is an issue with the model code or sampling process. JAGS is often a very fast sampler, especially for less complicated models.

Nimble

Nimble is an extension of Bugs that allows for bayesian computation within `R`, but uses C++ under the hood to compile models and algorithms for speed. Using the `nimble` package, Nimble can perform MCMC bayesian computations, but also allows for other types of algorithms to be loaded or custom-built. It can also compile other code without BUGS models into C++ for other uses. For the purpose of bayesian sampling using MCMC, Nimble is used very similarly to BUGS in practicality. The flexibility of Nimble is not needed in most cases when doing MCMC. The model specification language is almost identical to OpenBugs. A model can be both compiled and run using the function `nimbleMCMC()`. Because the model is compiled using C++ each time, this can drastically increase the total computation time as compared to other sampling methods. Occasionally, Nimble can also freeze up or fail, which can take a bit of time to fix.

STAN

STAN is a language in-and-of itself used for statistcal modeling and data analysis. STAN can be used in several programming languages, including `R` and Python. While many bayesian samplers use Gibbs sampling, STAN uses Hamiltonian Monte Carlo under the hood, a more computationally efficient sampler that is based on an advanced physics simulation. This can greatly reduce the computational time for many types of statistical models, especially for more complicated ones. Like Nimble, STAN compiles each model, but one big advantage is that it can store each compiled model in a .Rds file, which allows for repeated use without the need to re-compile each time. In `R`, STAN is used through the `rstan` package, which allows for data to be loaded through normal `R` data structures. The biggest practical difference in using STAN compared to other bayesian computational methods is in the model specification process. STAN requires a model object in the form of a .stan file, which is created in a very unique language that is more formal than in BUGS. A .stan file, either created through a character string in R or in a seperate text file, can be used by the `stan()` function, which takes the file along with the data to perform the sampling. There is a very robust set of manuals for STAN users, compared to other methods.

Greta

Greta, a very recent addition to the list of bayes computational methods, is made specifically for use in `R`, making the model and data specification process as easy as possible. It also uses Google TensorFlow to perform its calculations, and allows for more flexible use of CPUs and GPUs. Unfortunately, as of the writing of this, it is still in need of lots of development, and lacks an appropriate amount of documentation and support for users. Computationally, it is very slow compared to the other bayes methods, especially in simple cases. There are several types of semi-common models that Greta seems unable to properly handle in model specfication at the moment. Greta also attempts to make model specification very user-friendly, but uses some very common function names in the process that might override other often-used `R` functions. Greta does offer a model visualization tool that does not exist in any of the other Bayesian samplers, in that it can provide a graphical representation of the specified model through the use of nodes and arrows. This tool can be very useful in trying to explain how all the components of a model work together.

Common Bayesian Computation Settings

Chains

When using Markov Chain Monte Carlo methods to estimate Bayesian posteriors of parameters, the number of independent Markov chains to be run needs to be specified. It is recommended to use several chains to explore the space of the target distribution, as a single chain may not fully traverse the entire region on its own. For some simpler models that don't have irregular features in the target distribution, using one chain can suffice. Each sampler has an argument in its function call that specifies the number of chains. Some of the samplers have a default value if the number of chains isn't given.

Iterations

The number of steps that each Markov chain will take is a very important setting to get right. If each chain is not set up to run long enough, the parameter space may not be fully explored and the estimates will be inaccurate. Checking convergence statistics can be helpful in determining if the number of iterations is sufficient. An argument specifiying the number of iterations for each chain can be inputted in the function call for each sampler.

Warmup

Since our inferences are based on the assmumption that our draws are from the target distribution, it is wise to discard the samples at the beginning of each chain, since it takes time for the sampler to "settle down". The terms "warmup" and "burn-in" are often used to indicate the number of iterations that we will discard at the beginning of each chain when calculating statistics about our posterior distribution. We can choose this number in the function call for each sampler. In `OpenBUGS`, `Nimble`, and `Jags`, the argument is referred to as burnin, but in `STAN` and `Greta` it is called warmup. Some samplers (`Greta`, `Jags, Nimble`) treat this warmup argument as a number of iterations to run before starting to track the sampling results, but others (`STAN`, `BUGS`) have this be the actual number of iterations to discard out of the total number of specified iterations, so it is important to know which designation a sampler uses.