Bayesian Statistics in Educational Research: Day 3
Howard Seltman, PIER Program
12:30-4:30 June 21-25, 2010

Review: Bayesian analysis, the main steps

- Study design and choice of variables (same as classical analysis)
- Construct a data model
- Choose prior distributions
- Construct a complete Bayesian model (make a DAG)
- Use one of the two methods for finding p(θ | y) from p(y | θ) and p(θ)
- Evaluate the quality of the posterior sample.
- Evaluate the model quality
- Report results

Bayesian Analysis: Posterior Distribution Simulation Methods, continued

- Concerns for MCMC: Burn-in, autocorrelation, poor mixing, insufficient chain length, lack of convergence, convergence to minor modes.

---

alpha[5]
DAGs and Full Conditional Distributions

- Rasch example
  - The N latent student proficiencies, $\theta_i$, are scaled to have mean 0 and variance 1, and assumed to be normally distributed (CLT applies). The T latent question difficulties, $\alpha_j$, are fixed effects with very weakly informative priors. The $\beta$ parameter puts the scale on the $\alpha$'s and has a large-variance, censored normal prior (>0). The priors on $\alpha$ and $\beta$ are taken to be normal with mean 0 and small precision (large variance). The log odds of student $i$ correctly answering question $j$ is taken to be $\beta \theta_i - \alpha_j$.
  - Rasch Math: natural log of the odds of success is $\theta_i - \alpha_j$ where $\theta_i - \alpha_j$ is student proficiency, $i$ is student number out of $N$, $\alpha$ is question difficulty, $j$ is question number out of $T$. (Log odds is abbreviated “logit”.) Assumes independence of students and questions, and that question difficult ordering is the same for all students. The problem is “unidentifiable” in the sense that three equally spaced student proficiencies could be labeled 0, 1, 2 or 7, 9, 11, etc. To fix this we can make the student proficiencies have mean 0 and variance 1. The alphas are now “pinned to” this scale such that a difficulty of 1.3 indicates that the question will be answered correctly by 50% of students with a proficiency of $+1.3$ s.d. above the mean proficiency. A beta parameter is needed to scale the relationship. We will use the form $LO(S) = \beta \theta - \alpha$ where the difficulty is $\alpha/\beta$. 

\[ \text{Rasch Model for Difficulty 0.5 (solid) and 1.5 (dashed)} \]

\[ \text{beta = 0.8, alpha = beta*difficulty} \]
\[ \text{beta = 2.2, alpha = beta*difficulty} \]

\[ \text{P(success)} \]

\[ \text{meta} \]
○ DAG:

○ The joint distribution can be written easily from the DAG:
  \[ f(y|\beta, \theta, \alpha)f(\beta) = f(y|\theta, \alpha) = f(\beta|\theta, \alpha) = f(\beta) \]

○ The full conditionals are easily obtained (up to the normalizing constant)
  \[ f(\beta|y, \theta, \alpha) = \frac{f(\beta, y, \theta, \alpha)}{f(y, \theta, \alpha)} \]

Since this is a function of \( \beta \) with the other variables fixed, we can just say that this is proportional to all of the (product) terms in the joint distribution that contain \( \beta \). If we recognize the form we can simulate directly. If not, even without calculating the normalizing constant, we can use, e.g. M-H or slice sampling to simulate from \( \beta \) given everything else. Iterating this process over all parameters gives an MCMC sample from the posterior.

Software: R, WinBUGS, and rube

- R ([http://www.r-project.org](http://www.r-project.org)) is the open-source multi-platform version of S (commercial versions are also available). R is a non-menu oriented statistical programming system that includes functions for data manipulation, the most common classical statistical algorithms, and highly flexible graphics. Links to various sources of help are at [http://www.stat.cmu.edu/~hseltman/RTips.html](http://www.stat.cmu.edu/~hseltman/RTips.html). Many non-built-in algorithms are available in hundreds of user-written “packages” that are easily and automatically installed from the web.

R’s strengths include the scope of what it can do, the free and open nature, the robustness of its statistical algorithms, flexibility for representing data and models, ability to “compute on the language”, and the presences of a worldwide community of users. Weaknesses include relative slowness due to its being an interpreted language (but critical calls use optimized compiled code), potential for user error, and relatively poor ability to efficiently handle large datasets.
Key features are that the natural data types are all vectors, and that it has a class orientation with automatic method dispatch. For non-technical users this means that the object returned from, say a regression call, can be printed, plotted, or summarized with the same superficial code as the object returned from a hierarchical clustering call. Also, the components of the object are accessible as needed. In addition, objects persist across sessions.

Hand-built MCMC algorithms are often prototyped in R or just run in R. This requires a combination of great programming skills and statistical knowledge. Instead, we will use R to explore prior distributions, read in and manipulate data, do EDA, call WinBUGS, and evaluate, explore and summarize the posterior.

- **WinBUGS** ([http://www.mrc-bsu.cam.ac.uk/bugs/](http://www.mrc-bsu.cam.ac.uk/bugs/)) is an open source package that is for the Windows OS. **JAGS** ([http://www-fis.iarc.fr/~martyn/software/jags/](http://www-fis.iarc.fr/~martyn/software/jags/)) is a similar multi-platform program. **Open Bugs** ([http://www.openbugs.info/w/](http://www.openbugs.info/w/)) is the multi-platform future of WinBUGS. WinBUGS is intended to stand alone, but IMHO has insufficient flexibility and is hard to work with. It has an interactive DAG drawing program called “doodle” that is very clunky. Its error reporting is horrendous. Its core MCMC algorithms are very good.

Model specification is very R-like (but is declarative rather than procedural). There are many important limitations to the model specification system for serious MCMC work.

- **R2WinBUGS** is an R package that allows R users to avoid having to directly interact with WinBUGS.

- My package “rube” ([http://www.stat.cmu.edu/~hseltman/rube/](http://www.stat.cmu.edu/~hseltman/rube/)) is an R add-in that is designed to reduce cognitive overload in the complex world of Bayesian analysis. It makes use of R2WinBUGS (and a text processing package called stringr). Key features include:
  - Syntax checking of WinBUGS model declaration code
  - Extension of model code to include default constants (for hyperparameters), conditional coding (e.g., for alternate prior choices), and flexible R-like use of covariates in the structural model. All of these result in maintenance of a single model file rather than many.
  - Interactive graphics for exploring priors and posteriors.
  - More flexible use of data generation (for simulation studies) and starting value generation.
  - Better self documentation of results
  - The ability to display the relationships between models, data, and initial values, which is key to detecting non-syntax errors.
  - A function to read the WinBUGS examples into R.
Example: Rasch test for five LSAT items and 1000 students

```r
library(rube)
args(rube)
function (model, data = NULL, inits = NULL, parameters.to.save = NULL, 
n.chains = 3, n.iter = 2000, n.burnin = floor(n.iter/2), 
n.thin = max(1, floor(n.chains * (n.iter - n.burnin)/n.sims)), 
n.sims = 1000, bin = (n.iter - n.burnin)/n.thin, debug = FALSE, 
modelCheck = c("onFail", "never", "always"), cullData = TRUE, 
cullInits = TRUE, cullPnts = TRUE, cullWarn = TRUE, DIC = TRUE, 
digits = 5, codaPkg = FALSE, 
subs = NULL, cases = NULL, varList = list(), 
wd = getwd(), bugs.seed = NULL, over.relax = FALSE, dataParams = list(), 
initExtra = NULL, warnUseless = FALSE, modelFile = "tempModel.txt", 
ignore = NULL)

> r = rube("RaschModel.txt")
Rube Results:
Stochastics:

<table>
<thead>
<tr>
<th>Distr</th>
<th>Size</th>
<th>Initial Value(s) [Range]</th>
<th>Parameters (mean, sd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>dbern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta</td>
<td>dnorm</td>
<td>0, 1 (0, 1)</td>
<td></td>
</tr>
<tr>
<td>alpha</td>
<td>dnorm</td>
<td>0, 0.001 (0, 31.623)</td>
<td></td>
</tr>
<tr>
<td>beta</td>
<td>dnorm_I(0,</td>
<td>0, 0.001 (25.152, 18.966)</td>
<td></td>
</tr>
</tbody>
</table>
```
# Get data
gt LSAT = read.csv("LSAT.csv")
gt rube("RaschModel.txt", LSAT)

Rube Results:

Data:

<table>
<thead>
<tr>
<th>Size</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.92</td>
</tr>
<tr>
<td>Q2</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.71</td>
</tr>
<tr>
<td>Q3</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.55</td>
</tr>
<tr>
<td>Q4</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.76</td>
</tr>
<tr>
<td>Q5</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Stochastics:

<table>
<thead>
<tr>
<th>Distr</th>
<th>Size</th>
<th>Initial Value(s) [Range]</th>
<th>Parameters (mean, sd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>dbern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta</td>
<td>dnorm</td>
<td>0, 1 (0, 1)</td>
<td></td>
</tr>
<tr>
<td>alpha</td>
<td>dnorm</td>
<td>0, 0.001 (0, 31.623)</td>
<td></td>
</tr>
<tr>
<td>beta</td>
<td>dnorm_I(0,)</td>
<td>0, 0.001 (25.571, 19.157)</td>
<td></td>
</tr>
</tbody>
</table>

# Initialization function
gt abinit = function() list(alpha=rnorm(5,0,1), beta=abs(rnorm(1)))
gt abinit()

$alpha
[1] 1.3617904 1.1708600 0.6168696 -1.9631928 0.8912635

$beta
[1] 0.6228842

> abinit()

$alpha
[1] -0.8103875 2.0272602 -0.3488002 0.7009656 -1.1229658

$beta
[1] 2.042201

# Check all together
gt rube("RaschModel.txt", LSAT, abinit)

Rube Results:

Data:

<table>
<thead>
<tr>
<th>Size</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.92</td>
</tr>
<tr>
<td>Q2</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.71</td>
</tr>
<tr>
<td>Q3</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.55</td>
</tr>
<tr>
<td>Q4</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.76</td>
</tr>
<tr>
<td>Q5</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Stochastics:

<table>
<thead>
<tr>
<th>Distr</th>
<th>Size</th>
<th>Initial Value(s) [Range]</th>
<th>Parameters (mean, sd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>dbern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theta</td>
<td>dnorm</td>
<td>0, 1 (0, 1)</td>
<td></td>
</tr>
<tr>
<td>alpha</td>
<td>dnorm</td>
<td>5 0.803 +/- 0.526 [0.236, 1.559]</td>
<td>0, 0.001 (0, 31.623)</td>
</tr>
<tr>
<td>beta</td>
<td>dnorm_I(0,)</td>
<td>1 0.443 [0.443, 0.443]</td>
<td>0, 0.001 (25.542, 19.203)</td>
</tr>
</tbody>
</table>
# Choose "parameters to save" and submit to WinBUGS

```r
raschB = rube("RaschModel.txt", LSATdata, abinit, c("alpha","beta","theta"),
                n.chains=1, n.burn=0, n.iter=300, DIC=FALSE)
p3(raschB)
```

![Graph of alpha[4]](image1)

![ACF plot](image2)

![Histogram of alpha[4]](image3)
raschB2 = rube("RaschModel.txt", rasch$data1, abinit,
c("alpha","theta","a12","a12.34","a12.35","Q1P1","Q1M1"),
n.burn=100, n.iter=1000, n.thin=5)

95% Post. Int. alpha ~ dnorm

Signif. Autocor. Lags

rHat

theta ~ dnorm

95% Post. Int.

Signif. Autocor. Lags

rHat
I n f e r e n c e  f o r  B u g s  m o d e l  a t  " t e m p M o d e l . t x t " ,  f i t  u s i n g  W i n B U G S ,
3  c h a i n s ,  e a c h  w i t h  1 0 0 0  i t e r a t i o n s  ( f i r s t  1 0 0  d i s c a r d e d ) ,  n . t h i n  =  5
n . s i m s  =  5 4 0  i t e r a t i o n s  s a v e d
m e a n      s d      2 . 5 %       2 5 %       5 0 %       7 5 %     9 7 . 5 %   R h a t  n . e f f
a l p h a [ 1 ]       - 2 . 7 3 3   0 . 1 2 8    - 2 . 9 8 3    - 2 . 8 1 7    - 2 . 7 2 7    - 2 . 6 4 8    - 2 . 4 9 9  1 . 0 0 9    2 0 0
a l p h a [ 2 ]       - 1 . 0 0 5   0 . 0 8 2    - 1 . 1 5 3    - 1 . 0 6 3    - 1 . 0 0 7    - 0 . 9 4 7    - 0 . 8 5 7  1 . 0 3 0     7 1
a l p h a [ 3 ]       - 0 . 2 4 5   0 . 0 7 0    - 0 . 3 8 2    - 0 . 2 9 3    - 0 . 2 4 3    - 0 . 1 9 5    - 0 . 1 1 0  0 . 9 9 9    5 4 0
a l p h a [ 4 ]       - 1 . 3 1 2   0 . 0 8 6    - 1 . 4 8 2    - 1 . 3 6 9    - 1 . 3 0 6    - 1 . 2 5 3    - 1 . 1 4 5  1 . 0 1 2    5 4 0
a l p h a [ 5 ]       - 2 . 1 1 0   0 . 1 0 7    - 2 . 3 2 2    - 2 . 1 8 4    - 2 . 1 0 8    - 2 . 0 3 7    - 1 . 8 9 1  1 . 0 0 0    5 4 0
r h e t a [ 1 ]       - 1 . 8 5 5   0 . 7 9 0    - 3 . 3 2 6    - 2 . 4 5 3    - 1 . 8 3 0    - 1 . 3 0 1    - 0 . 2 7 1  1 . 0 0 3    4 6 0
r h e t a [ 2 ]       - 1 . 9 0 9   0 . 7 7 7    - 3 . 2 7 7    - 2 . 4 1 7    - 1 . 8 3 0    - 1 . 3 0 1    - 0 . 2 7 1  0 . 9 9 9    5 4 0
…
theta[999]  0.638  0.882  -1.064  0.105  0.600  1.220  2.390  1.000  540
theta[1000]  0.676  0.861  -0.885  0.084  0.670  1.220  2.518  1.000  540
a12  -1.727  0.144  -2.017  -1.827  -1.719  -1.629  -1.458  0.999  540
a12.34  -2.181  0.178  -2.532  -2.297  -2.170  -2.062  -1.855  1.015  130
a12.35  -1.384  0.189  -1.796  -1.500  -1.372  -1.257  -1.052  1.022  110
Q1P1  0.970  0.005  0.961  0.967  0.970  0.973  0.978  1.013  130
Q1M1  0.877  0.014  0.850  0.868  0.877  0.887  0.904  1.008  540

> print(raschB2,digits=3)
Lab 1: LSAT data with the Rasch model

Step 1: Get your computer ready

- In the Psych lab, either logon as “pier” (not yourself) “on this computer” (not Kerberos realm) or on your own Windows machine.
- Create a “Lab1” directory, e.g., under My Documents.
- Here is one way to keep your data separated by projects and to correct the poor “mdi” installation choice. Copy the R desktop icon (shortcut) into your Lab1 directory. From the top of the folder, highlight and copy the full path name for Lab1. Right-click on the new R icon in Lab1 and select properties. Under “Target” add “-sd1” after the final quote that is already there (without adding a new set of quotes). Under “Start in”, replace what is inside the quotes by pasting your full Lab1 directory name. Click OK, then double click on the R icon to start R in the Lab 1 directory in single document interface mode.
- Download these files into the Lab1 folder: rube_0.2-x.zip (“rube current version”), RaschModel.txt, LSAT.csv, and the RaschEx.R files.

Step 2: Install packages and load packages

- Ask if you don’t understand the difference
- Try package vs. function help
  - library(help=rube) and/or help(“rube-package”)
  - ?rube and/or args(rube)

Step 3: Check the model

- Use rube() with only a model argument to check for syntax errors
- Fix the errors in RaschModel.txt. (Ask for help if you have no programming experience.)

Step 4: Choose/check prior distributions

- Try priorExplore() for general practice. Here, just calculate the sds.

Step 5: Load the data into R and check the data

- Be sure you know what each step does.
- Run rube() with both model and data arguments.

Step 6: Initial values for the MCMC

- Generally a good idea
- May be in the form of a list or a function that generates a list. The latter is better for multiple chains.
- Here we will try to let WinBUGS generate the 1000 thetas. We’ll set alpha and beta. }

Step 7: Check that the initializations are reasonable

- Run rube with the initializations as the third argument.
Step 8: Choose which parameters to monitor and run a short chain with no burn-in and no thinning

- Adding the parameters.to.save argument to rube() causes it to actually run WinBUGS.
- One of several things may happen:
  - rube() may tell you that it's a waste of time to run WinBUGS because it would fail, and it tells you why
  - rube() may tell you that it tried to run WinBUGS but WinBUGS fails. A good choice is to rerun rube(debug=TRUE) to see the WinBUGS error. The error message is occasionally useful, but mostly you just can see if the failure was while checking the model, reading the data, checking the initializations, or running the MCMC iterations. It may also help to look at the thisModel.txt file in case the problem relates to how you told rube to preprocess the model. Most often, studying the result of summary() of the returned object will help pinpoint the problem.
  - If WinBUGS was not successfully run, rube() returns a bugsCheck object. If it was successful, a bugs() object is returned.

Step 9: Examine the bugs result

- Use p3() to carefully examine the WinBUGS result.

Step 10: Run the model “for keeps”

- Run multiple chains with burn-in and thinning, long enough to get good results.
- Re-check with p3(). Examine the $summary component of the result for standard inference output.
- Use standard parallel R math on $sims.array[,] to get the posterior of derived quantities that were not monitored. The positions of the array are iteration, chain, and parameter with full notation for vector or matrix parameters, e.g. “alpha[5]”.

Step 11: Check prior sensitivity

- Change hyperparameters and/or distribution types and compare results.
- That’s enough to process for one day!!!