Bayesian Regression

Paul Johnson[1]

[1] Center for Research Methods and Data Analysis

2018

KU

Outline

KU

Outline

I'm Totally Enthusiastic!

- I swallowed the Bayes pill. I want to be like that!
- software frameworks
  - BUGS
  - JAGS
  - Stan
- I'm an R (R Core Team, 2017) user, what to do?
  - The R Bayesian Task View: R Bayesian

Understand your model

- Necessary to understand which unknown parameters you are estimating
- Necessary to understand jargon about prior beliefs for parameters

I understand regression, sorta

$$y_i = \beta_0 + \beta_1 x1_i + \beta_2 x2_i + \ldots + \varepsilon_i, \;\; for\, i \in 1, \ldots N$$

The usual assumption is that ε_i is drawn from $N(0, \sigma_\varepsilon^2)$

- The parameters to estimate are β_0, β_1, β_2, ..., and σ_ε
- Data will be y, $x1$, $x2$, etc
- Now, state your prior beliefs about the β's and σ_ε.

Do you like vague priors?

- Use the prior information to make priors that are "in the general vicinity" of current beliefs, but leave a reasonable variation
- If you really have no idea at all about most likely value, choose a prior distribution that is wide and relatively flat.
  - Betas: choose normal centered on 0 with a ridiculously huge standard deviation, say $N(0,\, 1000^2)$, or adopt Student's t prior with small degrees of freedom.
  - Sigma: less clear what we ought to do.
    - $\tau = \frac{1}{\sigma_\varepsilon^2}$ is called precision
    - A common prior for precision has been $Gamma(0.1, 0.1)$ which has mean 1 and variance 10.
    - Gelman suggest prior for σ_ε should be simple $Uniform(0.1, some\, big\, number)$.

Outline

The Ordinary Least Squares Regression

- Some data Zack Roman obtained

```
ddir <- "data"
fn <- "OLS_data.csv"
dat <- read.csv(file.path(ddir, fn))
# Make text variable into R factor
dat$x3f <- factor(dat$x3, levels = c(0, 1, 2),
                  labels = c("lo", "med", "hi"))
```

```
head(dat, 10)
```

```
            x1          x2          y x3 x3f
1   0.04308983 -0.8373452 -0.3461347  0  lo
2   0.26042677  0.1869587 -0.1634261  1 med
3  -1.27029419  0.6448636  0.6919094  2  hi
4  -1.35175600  1.7507298  0.7995129  0  lo
5   1.80028444 -0.4759012  1.6761407  1 med
6   1.88627714 -1.2192776  1.7799701  2  hi
7  -1.34221493  1.0640451 -0.4041325  0  lo
8  -1.37435358 -0.8777185 -0.2136798  1 med
9   1.50847983  1.9537122 -4.9849412  2  hi
10 -0.38709828 -1.0223294 -0.4816288  0  lo
```

The Ordinary Least Squares Regression

- We are going to estimate a regression with an interaction between two predictors, x1 and x2, and a categorical variable x3.
- Putting aside x3 for the moment, our theory is that there is some nonlinearity in the relationship.
  - Linear model :

$$y_i = \beta_0 + \beta_1 x1_i + \beta_2 x2_i + \varepsilon_i$$

    seems not right
  - The effect of $x1$ depends on $x2$, and vice versa
  - Needs some "wiggly" curvatures, some wavy function like

$$y_i = \beta_0 + f(x1_i, x2_i) + \varepsilon_i$$

  - Models for that are out of our reach right now, we are "stuck" with a traditional OLS calculation

The Ordinary Least Squares Regression ...

- The simplest nonlinear formula is standard "interaction" model.

$$y_i = \beta_0 + \beta_1 x1_i + \beta_2 x2_i + \beta_3 x1_i \cdot x2_i + \varepsilon_i$$

- Why $x1_i \cdot x2_i$? This is an effort to include curvature, a nonlinearity in the predictive relationship.
- Could say effect of x2 depends on x1. Here, it is re-written so x2's slope includes x1,

$$y_i = \beta_0 + \beta_1 x1_i + (\beta_2 + \beta_3 x1_i) \cdot x2_i + \varepsilon_i$$

The OLS estimate

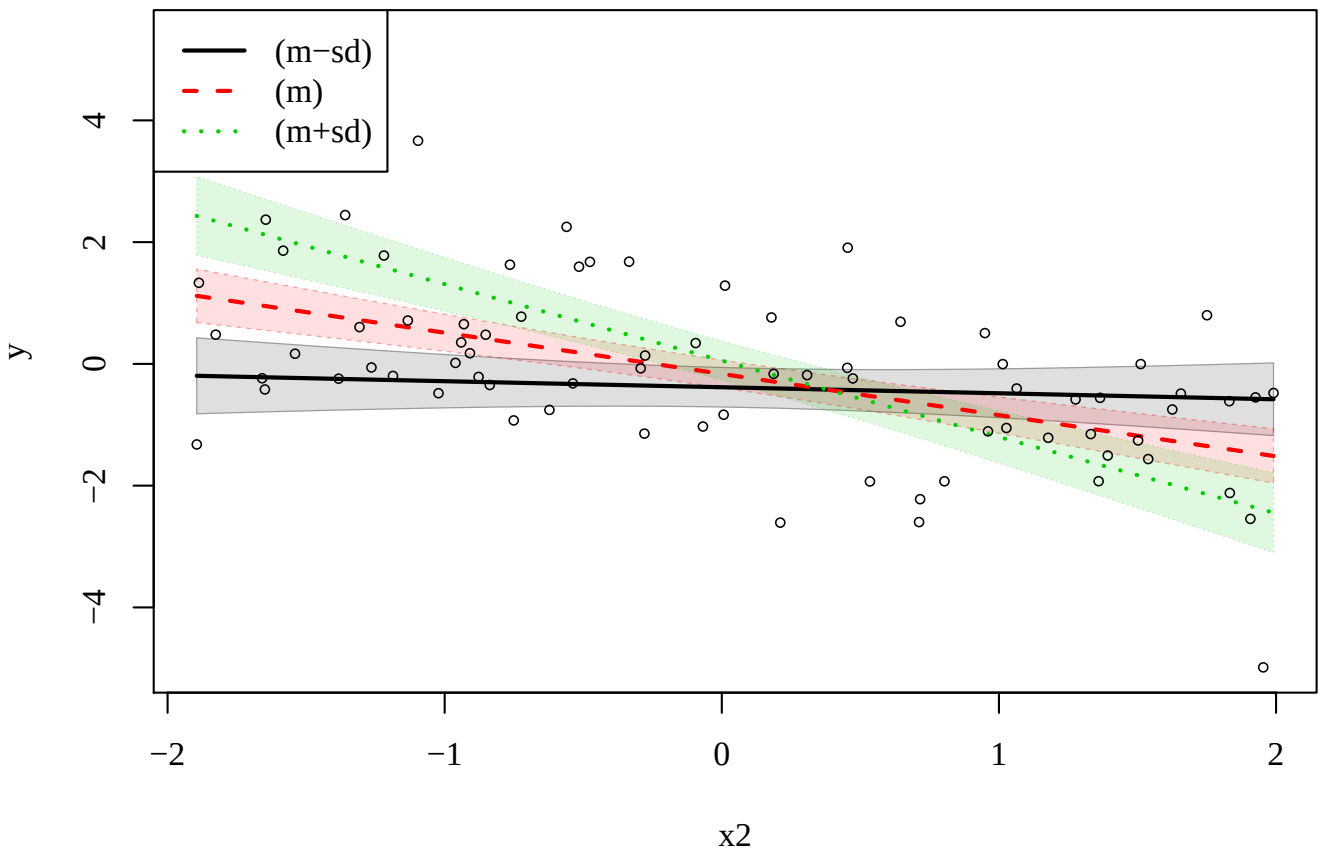
```
m0 <- lm(y ~ x1 * x2, data = dat)
summary(m0)
```

```
Call:
lm(formula = y ~ x1 * x2, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.45205 -0.51588  0.05467  0.48869  2.37923

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.15901    0.11402  -1.395   0.1675
x1           0.19033    0.10050   1.894   0.0623 .
x2          -0.69192    0.09837  -7.034 1.02e-09 ***
x1:x2       -0.50231    0.08698  -5.775 1.88e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9864 on 71 degrees of freedom
Multiple R-squared:  0.5199,  Adjusted R-squared:  0.4996
F-statistic: 25.63 on 3 and 71 DF,  p-value: 2.411e-11
```

Visualize that, 1

```
library(rockchalk)
ps20 <- plotSlopes(m0, plotx = "x2", modx = "x1",
    modxVals = "std.dev.", interval =
    "confidence")
```

Visualize that, 1 ...

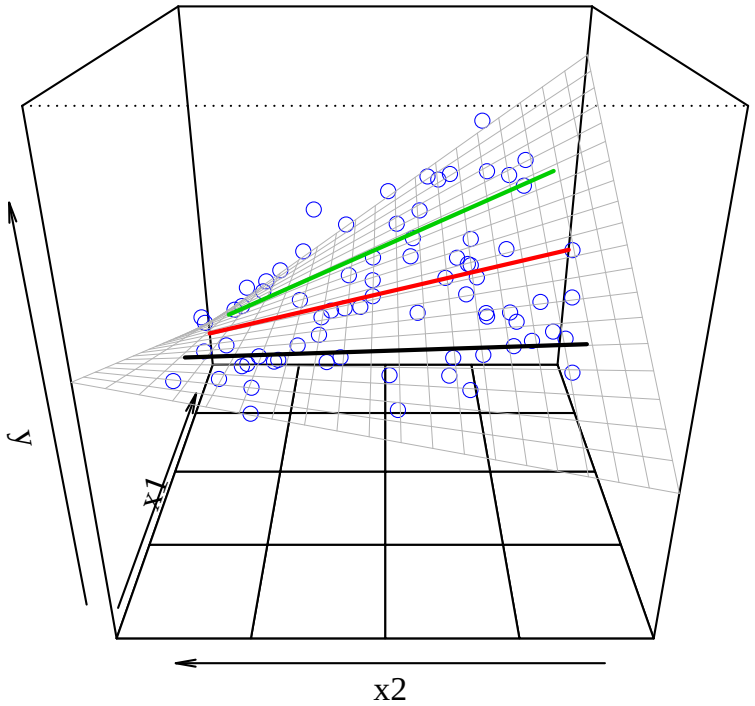


Visualize that, 2

```
ps30 <- plotPlane(m0, plotx1 = "x1", plotx2 =
    "x2", lcol = gray(.7), theta = 270, phi = 25)
addLines(from = ps20, to = ps30, col = ps20$col)
```

```
NULL
```

Visualize that, 2 ...



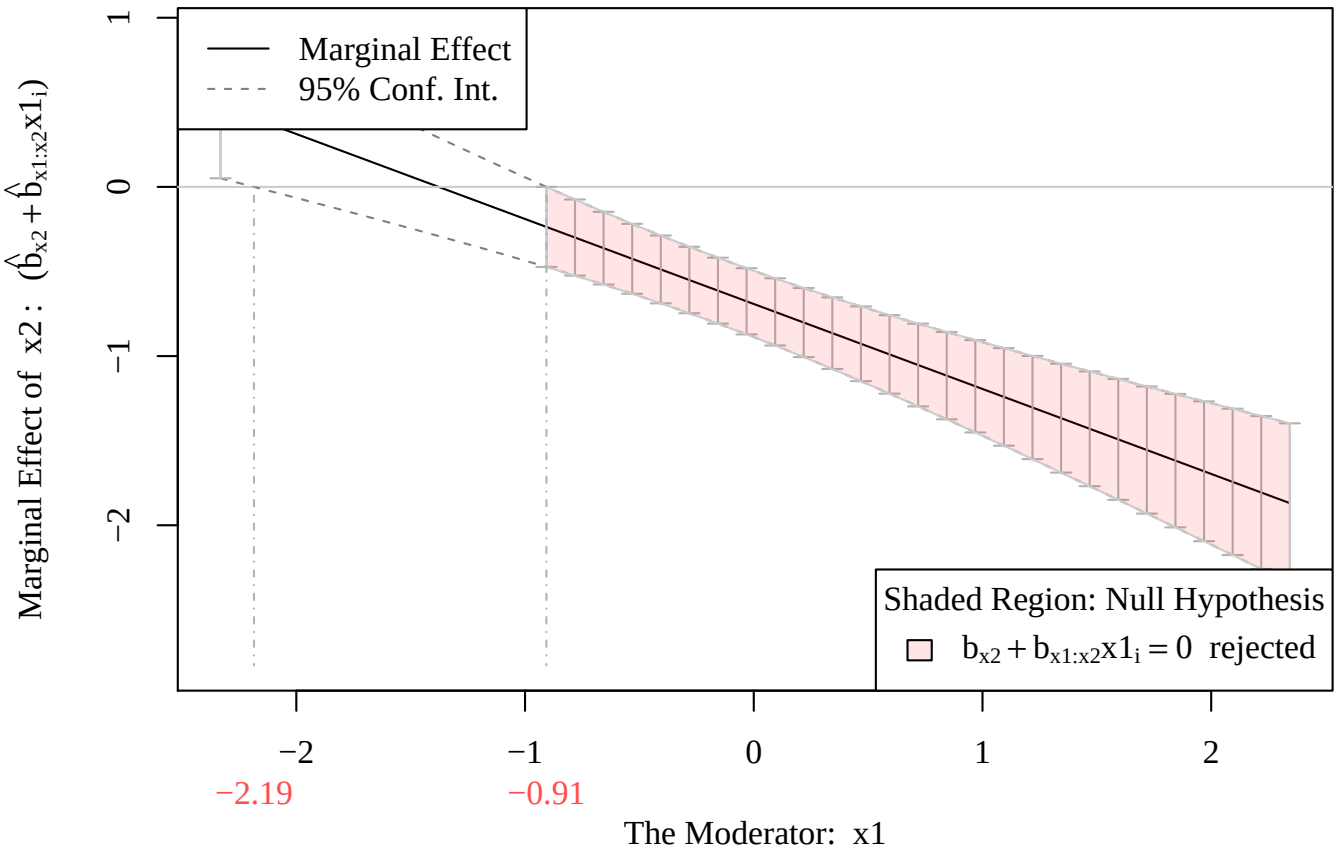
Visualize that, 3

```
ps40 <- testSlopes(ps20)
```

```
Values of x1 OUTSIDE this interval:
        lo         hi
-2.1851709 -0.9066175
cause the slope of (b1 + b2*x1)x2 to be statistically significant
```

```
plot(ps40)
```

Visualize that, 3 ...



Visualize that, 4

Marginal predictions

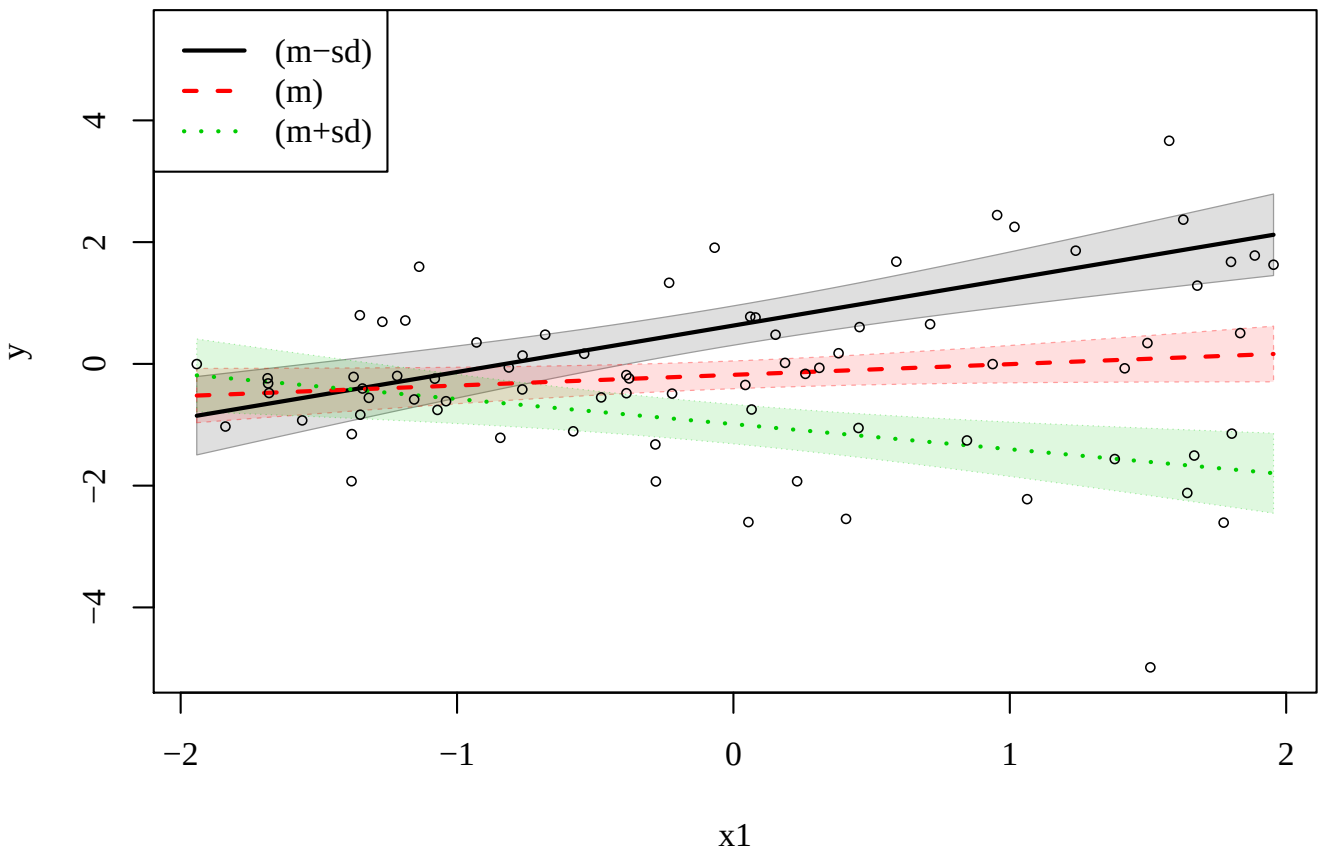
```
predictOMatic(m0, interval = "confidence")
```

```
$x1
          x1          x2          fit         lwr          upr
1 -1.9416696 0.03172283 -0.51957810 -0.9649748 -0.07418135
2 -1.0756371 0.03172283 -0.36854585 -0.6778744 -0.05921733
3 -0.0681562 0.03172283 -0.19284562 -0.4202329  0.03454168
4  0.9460424 0.03172283 -0.01597385 -0.3150284  0.28308066
5  1.9542074 0.03172283  0.15984570 -0.2967488  0.61644019

$x2
           x1          x2        fit         lwr         upr
1 -0.02845962 -1.89454252  1.1193641  0.6799328  1.5587955
2 -0.02845962 -0.93557042  0.4695400  0.1742902  0.7647898
3 -0.02845962 -0.06809952 -0.1182805 -0.3462894  0.1097283
4 -0.02845962  1.04552832 -0.8729033 -1.1749847 -0.5708220
5 -0.02845962  1.99089046 -1.5135050 -1.9596881 -1.0673218
```

Marginal plot from other point of view

```
ps70 <- plotSlopes(m0, plotx = "x1", modx = "x2",
    modxVals = "std.dev.", interval =
    "confidence")
```

Marginal plot from other point of view ...



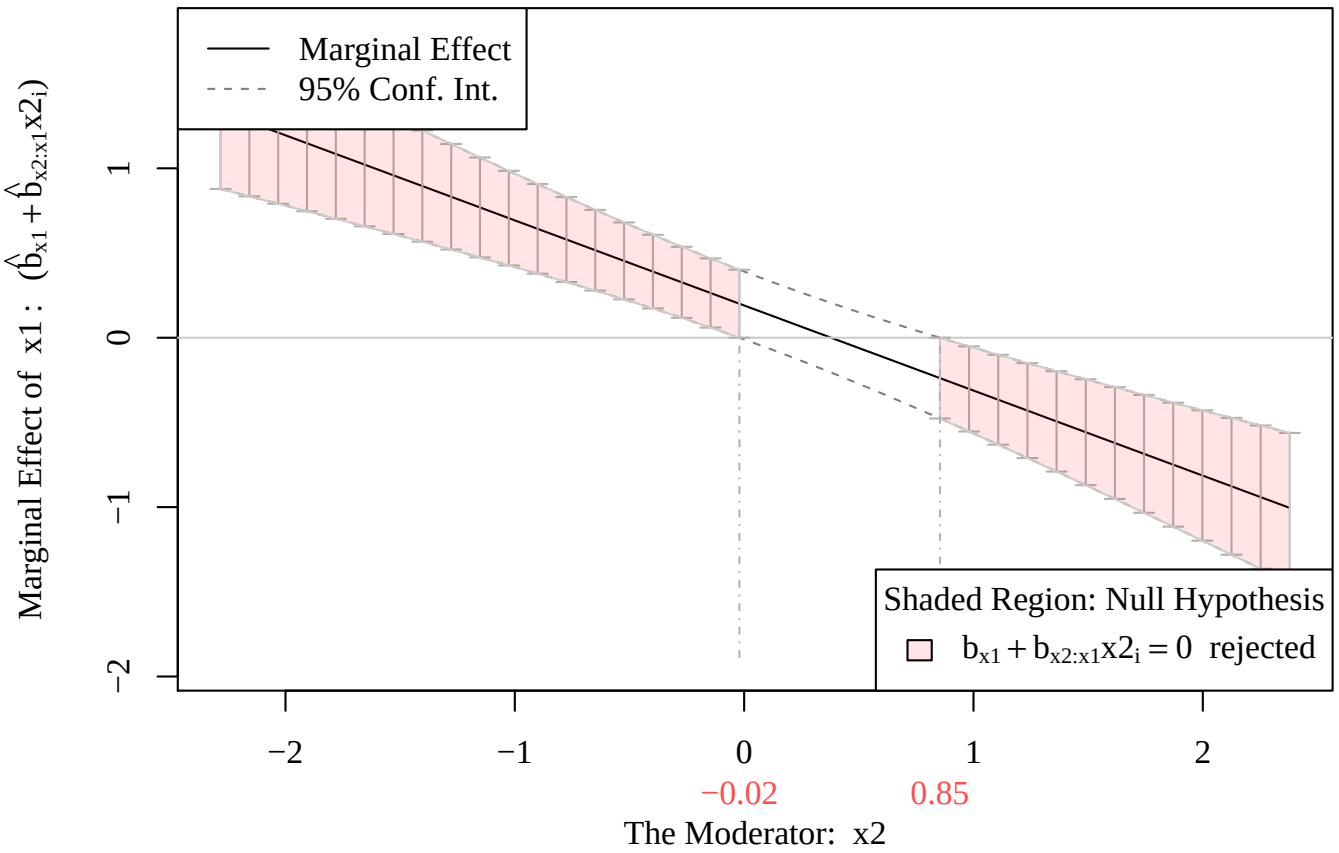
Testslopes on that

```
ps80 <- testSlopes(ps70)
```

```
Values of x2 OUTSIDE this interval:
         lo          hi
-0.02071143  0.85376194
cause the slope of (b1 + b2*x2)x1 to be statistically significant
```

```
plot(ps80)
```

Testslopes on that ...



Add in x3f

```
m0 <- lm(y ~ x1 * x2 + x3f, data = dat)
summary(m0)
```

```
Call:
lm(formula = y ~ x1 * x2 + x3f, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.50415 -0.53255  0.04636  0.51527  2.30997

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.12175    0.20212  -0.602   0.5489
x1           0.19990    0.10269   1.947   0.0557 .
x2          -0.69230    0.10062  -6.880 2.19e-09 ***
x3fmed       0.03265    0.28643   0.114   0.9096
x3fhi       -0.14434    0.28593  -0.505   0.6153
x1:x2       -0.50779    0.08849  -5.738 2.34e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9974 on 69 degrees of freedom
Multiple R-squared:  0.5229,  Adjusted R-squared:  0.4883
F-statistic: 15.13 on 5 and 69 DF,  p-value: 5.153e-10
```

Add in x3f ...

- Because `x3f` is a categorical variable with 3 levels, the standard coding creates 2 dummy variables, `x3fmed` and `x3fhi`

```
confint(m0)
```

```
                  2.5 %      97.5 %
(Intercept) -0.524962270  0.2814709
x1          -0.004972956  0.4047652
x2          -0.893032200 -0.4915659
x3fmed      -0.538768748  0.6040747
x3fhi       -0.714759429  0.4260814
x1:x2       -0.684335100 -0.3312497
```

Outline

Easy Routes for Standard Models

- For "garden variety" statistical models, there are *more-or-less automatic* ways to estimate with Bayes
- Mplus, Stata, SPSS, etc offer Bayesian variants of some models.
- The only problem is that they make value judgments for you on the priors and the chain calculations.

MCMCpack

- The first-and-foremost among R packages for canned Bayesian models is `MCMCpack` (Martin et al., 2011)
- In MCMCpack, use the function called `MCMCregress()`

```
library(MCMCpack)
mcmcpack0 <- MCMCregress(y ~ x1 * x2 + x3f, b0 =
    0, B0 = 0.1, sigma.mu = 1, sigma.var = 10,
    data = dat)
```

- Priors
  - betas are "flat" (improper uniform) if we use B0 = 0, here I set precision at 0.1 for all betas (variance = 10).
  - error variance is assumed to be inverse Gamma distribution with parameters `c0` and `d0`, which can also be specified by providing parameters `sigma.mu` and `sigma.var`.

MCMCpack ...

```
summary(mcmcpack0)
```

```
Iterations = 1001:11000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

               Mean      SD  Naive SE Time-series SE
(Intercept) -0.1212 0.20292 0.0020292      0.0020292
x1           0.1990 0.10227 0.0010227      0.0010227
x2          -0.6907 0.10080 0.0010080      0.0010080
x3fmed       0.0302 0.28756 0.0028756      0.0028756
x3fhi       -0.1448 0.28247 0.0028247      0.0028247
x1:x2       -0.5070 0.08911 0.0008911      0.0008911
sigma2       0.9953 0.17123 0.0017123      0.0018657

2. Quantiles for each variable:

                2.5%     25%      50%     75%   97.5%
(Intercept) -0.516407 -0.2585 -0.12029 0.01372  0.2756
x1          -0.004466  0.1320  0.20017 0.26735  0.3997
```

MCMCpack ...

```
x2          -0.885047 -0.7578 -0.69021 -0.62320 -0.4914
x3fmed      -0.536367 -0.1593  0.03006  0.22492  0.5954
x3fhi       -0.703359 -0.3332 -0.14648  0.04558  0.4140
x1:x2       -0.681713 -0.5669 -0.50679 -0.44745 -0.3315
sigma2       0.713725  0.8723  0.97997  1.09870  1.3802
```

```
if(interactive()) plot(mcmcpack0)
```

Specialty Packages

- "MNP" (Imai, 2005, 2017) is MCMC multinomial probit, using all homemade C++ code libraries. Great learning opportunity
  Imai and van Dyk (2005). "MNP: R Package for Fitting the Multinomial Probit Model." Journal of Statistical Software, Vol. 14, No. 3 (May), pp. 1-32. <doi:10.18637/jss.v014.i03>.

Hierarchical Models

- `MCMCpack` offers many variants of hierarchical models
- A similar competing set of hierarchical (and other) Bayesian models was implemented in the R package `bayesm` (Rossi, 2017).
- A state-of-the-art, flexible Bayes modeling is R package `MCMCglmm`. This includes 2 excellent vignettes, including course notes.
  Jarrod D Hadfield (2010). MCMC Methods for Multi-Response Generalized Linear Mixed Models: The MCMCglmm R Package. Journal of Statistical Software, 33(2), 1-22. URL http://www.jstatsoft.org/v33/i02/.
- Although Stan is a general purpose, syntax-driven project, the Stan team has decided to package up "idiot proof" examples of many typical models in a package called "rstanarm": lots of stats models! Similar in spirit to MCMCpack.
  - Similar R package `brms` allows users to write R commands and a Stan code model "pops out".

Concluding comment

- Should you use the pre-made Bayesian estimator?
  - yes, IF its priors are understandable to you and you can write what you want
  - no, IF you want to run a customized model that is not included in a pre-wrapped function

Outline

What is a library?

- library = collection of routines. Compiled languages like C, Fortran. High performance. Reusable
- Challenge 1: develop high quality numeric code that has a workable interface for programmers
- Challenge 2: get the programmers to write more libraries and R packages so that non-programmers can use the libraries as well.

KU

Period of Transition

- Many pioneering authors write their own Fortran/C++ algorithms, but we see less of that today
- Many would choose a library/toolkit and focus on that/
- Those people starting today should concentrate on learning JAGS and Stan.

BUGS=Bayesian Updating with Gibbs Sampling

- The BUGS project was a block-buster (summarized Lunn & Spiegelhalter, 2013)
- Created a path-breaking, freely available tools (BUGS, WinBUGS, OpenBUGS). Set the paradigm.
- HOWTO: user creates a model text file, specifies settings, launches MCMC simulation
- Nice training notes online:
  - The OpenBUGS site
  - WinBUGSTraining_smcs.pdf

Example model with 1 predictor regression

$y_i = b0 + b1 \cdot x_i + \varepsilon_i$, $\varepsilon \sim N(0, sigma^2)$.

```
model{
  ## likelihood for each case
  for (i in 1:n){
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- b0 + b1*x[i]
  }
  # Priors
  b0 ~ dnorm(0, 0.001) # 0.001 is precision
  b1 ~ dnorm(0, 0.001)
  # precision of error term is tau
  tau ~ dgamma(0.01, 0.01)
  sigma <- 1/sqrt(tau)
  ## alternatively
  ## sigma ~ dunif(0.001, 100)
  ## tau <- 1/(sigma*sigma)
}
```

Example model with 1 predictor regression ...

Other components needed to run

- model control information
  - how many chains, iterations for each one
  - how long is the "burn in" period
  - thinning
- initial values for parameters in each of the "chains"
- requests for diagnostic information
  - name the parameters to be saved and tracked
- SUMMARY: lots of details to correct

WinBUGS

- To facilitate user experience, WinBUGS added a graphical interface
- Model text file still needed, GUI can drive and monitor MCMC
- Wildly successful
- OpenBUGS is last iteration of BUGS, it is an open-sourced version of WinBUGS
- Project now frozen, only making new releases to fix truly cataclysmic bugs, no new features.

Then R came along

- Many R users wanted easy way to interact with OpenBUGS engine.
- Packages developed to interface R with OpenBUGS. Can still be used today.
- Because R has tools for data management, it is to send data and init info, along with a request
- R2OpenBUGS now works fine!
- BRugs (live interactive connector R OpenBUGS) also mostly works fine
- In the writeups folder for this session, I'm leaving an example of a model that can be run with R's lme4, OpenBUGS, and Stan for comparison.

JAGS = Just Another Gibbs Sampler

- The original BUGS project was not open source. It was free to use, but internal algorithm code was not.
  - What they had was written in a now-extinct variant of "Object Pascal"
- A competing implementation in C++ was developed by Martyn Plummer called JAGS.
- A JAGS model file uses most of syntax used by BUGS
- R connector package "rjags" (Plummer, 2010,1).

KU

Choose JAGS

- If you are using Gibbs Sampling to write your own model, I recommend JAGS over OpenBUGS. C++ faster, will be supported longer
- If a parameter is discrete (latent trait model), the JAGS platform is preferred (even over Stan)
- The R package "`blavaan`" is a user-friendly SEM package that is similar in nature to MCMCpack.
  - blaavan writes a JAGS syntax file and then runs it via rjags.

Stan is latest

- Research team headed by Andrew Gelman, Columbia.
- Stan proposes an updated version of the Metropolis algorithm, known as the Hamiltonian No U-Turn Sampler (NUTS) that seems to run faster and explore the parameter space more efficiently.
- Stan team has implemented many examples, including the examples that were provided with BUGS.

Stan for the regression model

- Look in the stan folder distributed with these notes.
  - elementary stan "OLS_reg0.stan"
  - still elementary but more flexible "OLS_reg.stan"

OLS_reg.stan

```
//  Zack Roman <zroman@ku.edu>
//  Simple OLS regression for Rstan Workflow
data {
  int N; //the number of observations
  int K; //the number of columns in the model
      matrix
  real y[N]; //the response
  matrix[N,K] X; //the model matrix
}
parameters {
  vector[K] beta; //the regression parameters
  real sigma; //the standard deviation
}
transformed parameters {
  vector[N] linpred;
  linpred = X*beta;
}
```

OLS_reg.stan ...

```
model {
  beta[1] ~ cauchy(0,10); //prior for the intercept

  for(i in 2:K)
   beta[i] ~ cauchy(0,1);//prior for the slopes
  y ~ normal(linpred,sigma);
}
```

Run through stan with this R stanza

```
library(rstan)
set.seed(234234)
## Manufacture data structures
X <- model.matrix(~ x1 * x2 + x3f, dat)
y <- dat$y
head(X)
```

```
  (Intercept)          x1         x2 x3fmed x3fhi       x1:x2
1           1  0.04308983 -0.8373452      0      0 -0.03608106
2           1  0.26042677  0.1869587      1      0  0.04868904
3           1 -1.27029419  0.6448636      0      1 -0.81916651
4           1 -1.35175600  1.7507298      0      0 -2.36655951
5           1  1.80028444 -0.4759012      1      0 -0.85675755
6           1  1.88627714 -1.2192776      0      1 -2.29989542
```

Run through stan with this R stanza ...

```
svn <- "OLS_reg.stan"
rt1 <- stanc(file = file.path(sdir, sfn))
sm1 <- stan_model(stanc_ret = rt1, verbose =
    FALSE)
stanm0 <- sampling(sm1, data = list(N =
    nrow(dat), K = ncol(X),
      y = y, X = X), pars = c("beta", "sigma"),
      chains = 4, iter = 2000, warmup = 1000)
```

- You'd need Stan to run that, of course.
- It takes a little while (20-30 seconds), so we saved a copy of the output object. Load it like so

```
wdir = "workingdata"
soutfn <- "stanm0.rds"
stanm0 <- readRDS(file.path(wdir, soutfn))
```

summary

- summary. Note it uses parameter names

```
summary(stanm0)
```

```
$summary
                mean     se_mean          sd          2.5%          25%          50%          75%
beta[1]   -0.14157030 0.004241371 0.19240240  -0.51631912  -0.2750561  -0.1405447  -0.01170728
beta[2]    0.19152033 0.001636877 0.10352521  -0.01209155   0.1235989   0.1939034   0.26108976
beta[3]   -0.67803898 0.001634730 0.10338941  -0.88718513  -0.7464410  -0.6762589  -0.61160576
beta[4]    0.05279162 0.005318138 0.26499924  -0.46100767  -0.1263369   0.0521137   0.22984083
beta[5]   -0.11668442 0.005284024 0.26593605  -0.63878495  -0.2907040  -0.1215888   0.05796778
beta[6]   -0.50098213 0.001401620 0.08864621  -0.67142319  -0.5619895  -0.5011020  -0.44185388
sigma      1.01529744 0.001406670 0.08896559   0.85778819   0.9526128   1.0096773   1.07297121
lp__     -38.54245418 0.045897759 1.94342626 -43.27617602 -39.6019515 -38.2214183 -37.10667130
              97.5%    n_eff      Rhat
beta[1]    0.2300102 2057.825 1.0003020
beta[2]    0.3889523 4000.000 0.9998801
beta[3]   -0.4759030 4000.000 0.9994049
beta[4]    0.5755834 2482.962 1.0006005
beta[5]    0.4134304 2532.940 1.0003923
beta[6]   -0.3248961 4000.000 1.0001695
sigma      1.2099515 4000.000 0.9994242
lp__     -35.7251147 1792.888 1.0009303

$c_summary
, , chains = chain:1

          stats
parameter          mean          sd           2.5%          25%           50%           75%        97.5%
  beta[1]   -0.13333278 0.19276812  -0.516384812  -0.2574546  -0.13556649  -0.01179914   0.2452291
```

summary ...

```
  beta[2]    0.19231493 0.10345588   0.004835102   0.1209143   0.19223304   0.26271391   0.3920030
  beta[3]   -0.67570219 0.10712585  -0.896553307  -0.7425049  -0.67675338  -0.60588099  -0.4787271
  beta[4]    0.04465561 0.26197118  -0.458925442  -0.1278580   0.05220658   0.22680419   0.5468439
  beta[5]   -0.12925919 0.27521639  -0.668153124  -0.3009059  -0.13343989   0.05395322   0.4123755
  beta[6]   -0.49803874 0.08734198  -0.663810519  -0.5611975  -0.49999789  -0.43474229  -0.3268878
  sigma      1.01611126 0.09382281   0.845904622   0.9490164   1.01212873   1.07898392   1.2122880
  lp__     -38.63341418 1.93888592 -43.092004105 -39.7679447 -38.29029068 -37.24501203 -35.7823666

, , chains = chain:2

         stats
parameter          mean          sd         2.5%         25%          50%          75%        97.5%
  beta[1]   -0.14805878 0.19064101  -0.50391579  -0.2864628  -0.14744765  -0.01110713   0.2138094
  beta[2]    0.18934883 0.10085118  -0.01118942   0.1214870   0.19062860   0.25236555   0.3854645
  beta[3]   -0.68037506 0.10136161  -0.88409118  -0.7483953  -0.67926879  -0.61544368  -0.4762840
  beta[4]    0.05879432 0.26929012  -0.46136958  -0.1256825   0.05028764   0.25203113   0.5920933
  beta[5]   -0.10445083 0.26727102  -0.60837727  -0.2850316  -0.11269278   0.06905388   0.4204522
  beta[6]   -0.50151131 0.08910141  -0.66913590  -0.5644356  -0.50160309  -0.44076293  -0.3250381
  sigma      1.01648305 0.08805561   0.85776717   0.9544239   1.00965660   1.06910752   1.2245269
  lp__     -38.52042831 1.87033218 -42.88930285 -39.5012429 -38.29573556 -37.10438397 -35.7601793

, , chains = chain:3

         stats
parameter          mean          sd          2.5%          25%          50%          75%        97.5%
  beta[1]   -0.14745493 0.18546501  -0.526775707  -0.27413760  -0.14265192  -0.01060526   0.1973395
  beta[2]    0.19652524 0.10505521  -0.001915868   0.12627377   0.19872688   0.27198780   0.3933224
  beta[3]   -0.67706313 0.10347443  -0.887269378  -0.74435733  -0.67352874  -0.61066460  -0.4718810
  beta[4]    0.06537512 0.25798514  -0.459590505  -0.08985958   0.06399977   0.22858175   0.5689778
  beta[5]   -0.11103637 0.25559181  -0.594014172  -0.28559332  -0.11601959   0.05768219   0.4164890
  beta[6]   -0.50400932 0.08694446  -0.674279480  -0.56390705  -0.50144597  -0.44790441  -0.3335026
  sigma      1.01510742 0.08419115   0.865060440   0.95473222   1.01082344   1.06906518   1.1940680
  lp__     -38.39554677 1.93801983 -43.232344290 -39.38388313 -38.00855219 -36.97117343 -35.7005305
```

summary ...

```
, , chains = chain:4

         stats
parameter          mean          sd         2.5%          25%          50%          75%        97.5%
  beta[1]   -0.13743471 0.20030784  -0.51682710   -0.2760259   -0.13413950   -0.01379842    0.2712258
  beta[2]    0.18789232 0.10463165  -0.03479283    0.1247124    0.19608024    0.25462707    0.3867111
  beta[3]   -0.67901555 0.10158575  -0.88002592   -0.7487031   -0.67498436   -0.61460816   -0.4762111
  beta[4]    0.04234143 0.27025308  -0.45910579   -0.1468834    0.04191593    0.21344879    0.5889127
  beta[5]   -0.12199131 0.26500389  -0.65305834   -0.2877754   -0.12422412    0.04781666    0.3921245
  beta[6]   -0.50036914 0.09116290  -0.67476376   -0.5604889   -0.50115637   -0.44362699   -0.3146862
  sigma      1.01348804 0.08962829   0.85832400    0.9510656    1.00583674    1.07292100    1.2005119
  lp__     -38.62042747 2.01721733 -43.43542909 -39.7647415 -38.24194768 -37.13088958 -35.6871775
```

Can re-insert variable column names

```
names(stanm0)[1:ncol(X)]  <- colnames(X)
stanm0
```

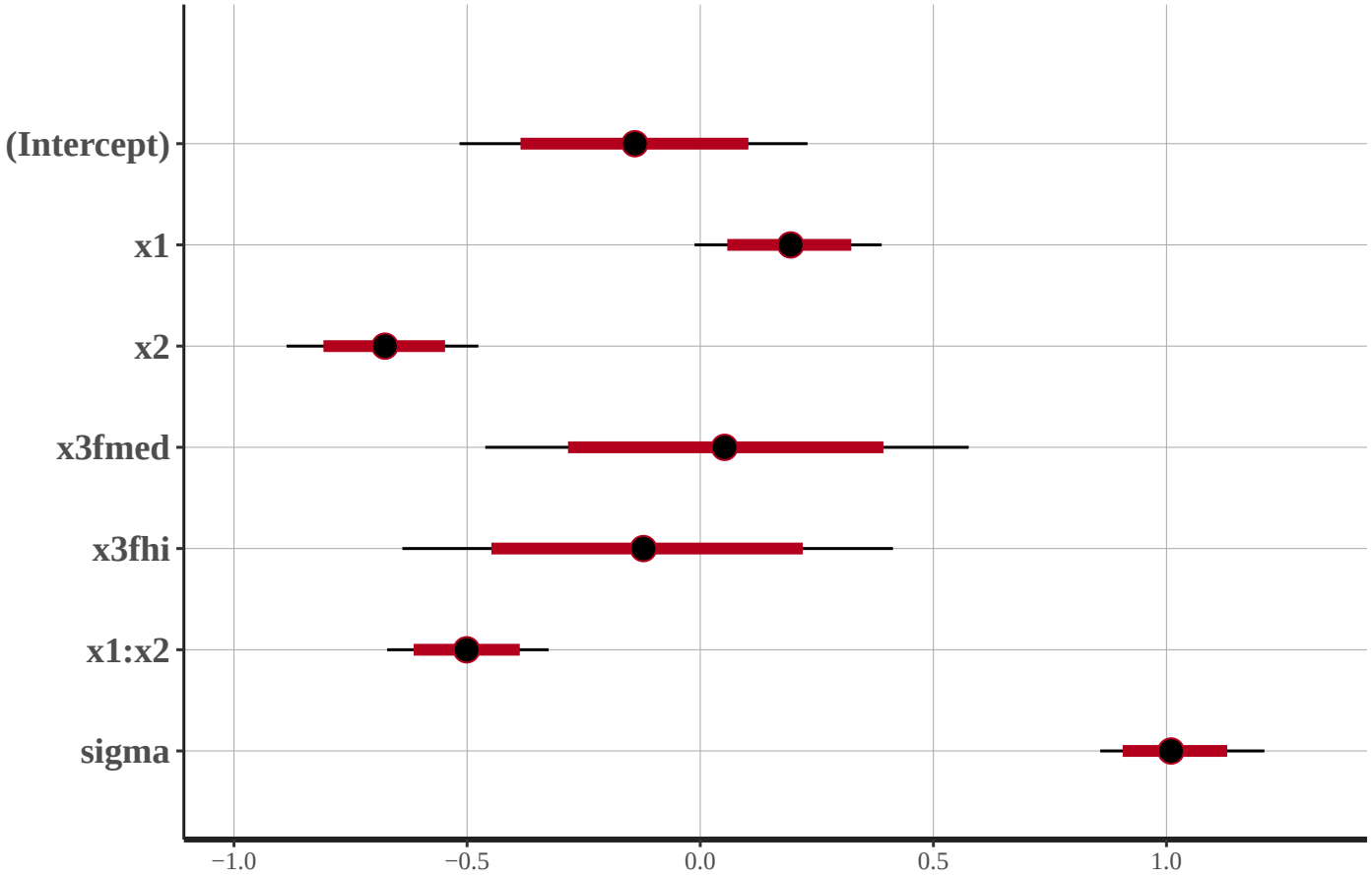
```
Inference for Stan model: OLS_reg.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

                mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
(Intercept)    -0.14    0.00 0.19  -0.52  -0.28  -0.14  -0.01   0.23  2058    1
x1              0.19    0.00 0.10  -0.01   0.12   0.19   0.26   0.39  4000    1
x2             -0.68    0.00 0.10  -0.89  -0.75  -0.68  -0.61  -0.48  4000    1
x3fmed          0.05    0.01 0.26  -0.46  -0.13   0.05   0.23   0.58  2483    1
x3fhi          -0.12    0.01 0.27  -0.64  -0.29  -0.12   0.06   0.41  2533    1
x1:x2          -0.50    0.00 0.09  -0.67  -0.56  -0.50  -0.44  -0.32  4000    1
sigma           1.02    0.00 0.09   0.86   0.95   1.01   1.07   1.21  4000    1
lp__          -38.54    0.05 1.94 -43.28 -39.60 -38.22 -37.11 -35.73  1793    1

Samples were drawn using NUTS(diag_e) at Wed May 30 17:57:07 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Prodigious Potpourri Plotting Possibilities

- Somewhat unfortunately, the plotting functions for stan objects are tied into the rstan package, so it is not possible to explore the package very well without installing rstan.
  - That's different from MCMCpack, which outputs an R-standard "coda" object, which we can explore even if MCMCpack is not installed.
- If you do have rstan, you will see they are consolidating their plotting support with functions named "`stan_xxx`" where xxx might be one of ("plot", "trace", "dens", "hist", "diag", "rhat").

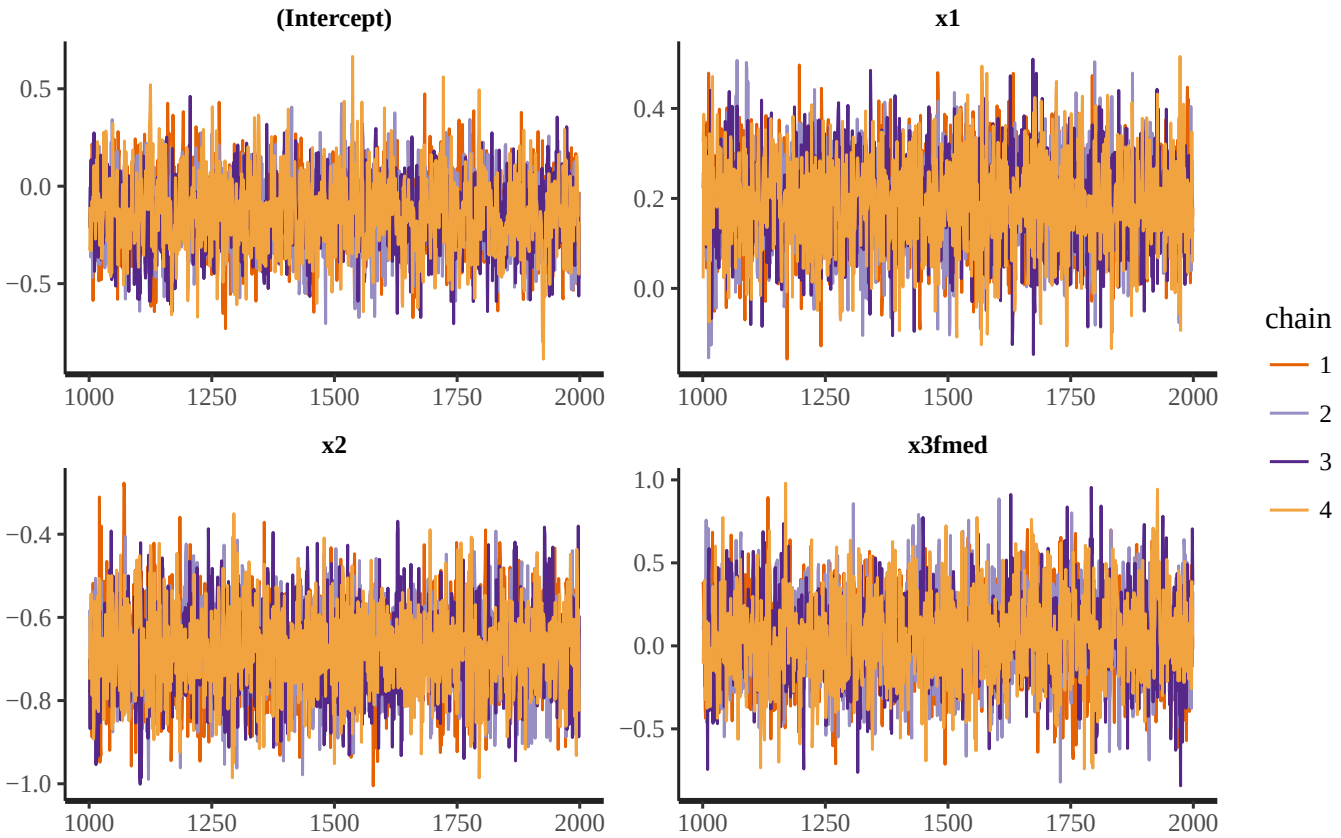
Prodigious Potpourri Plotting Possibilities

```
stan_plot(stanm0)
```



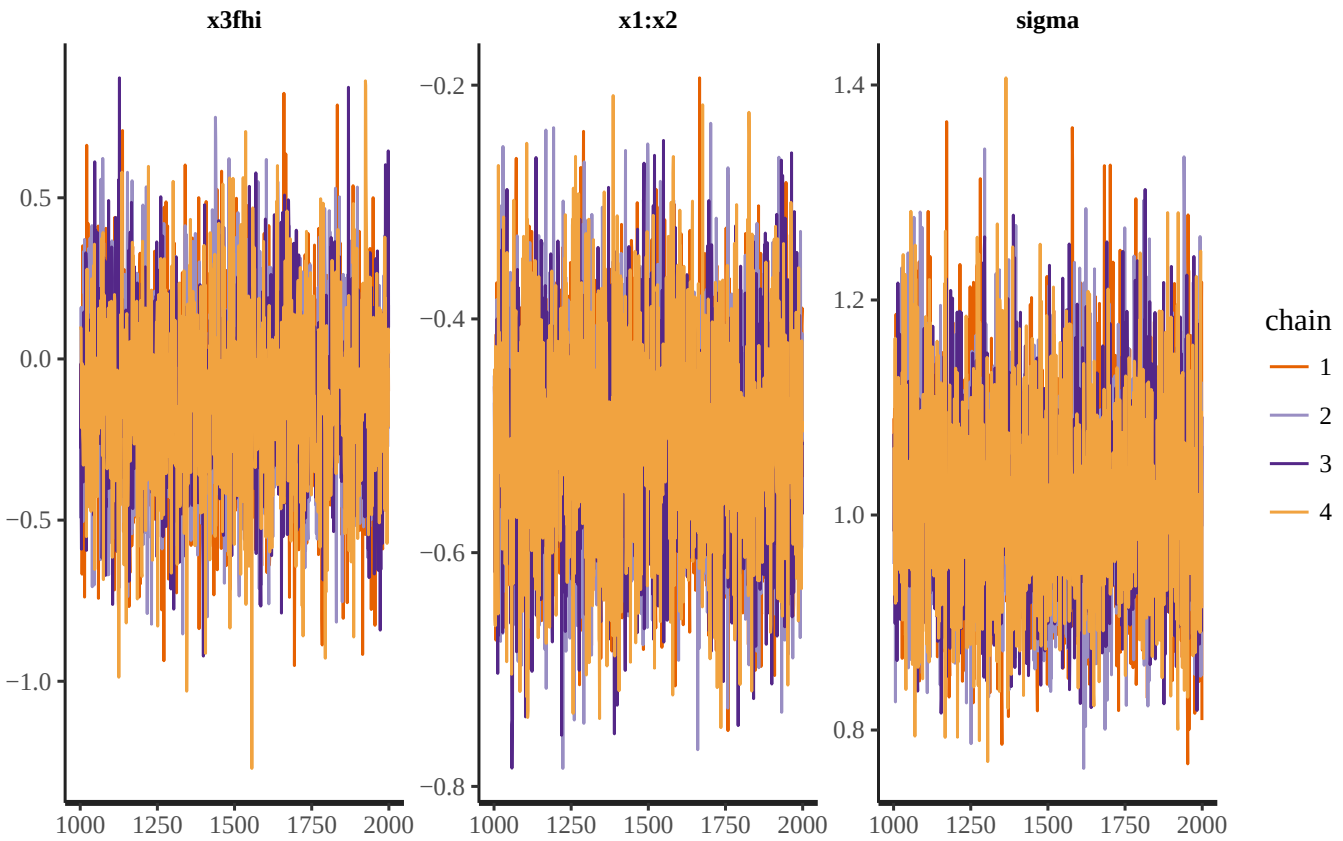
Prodigious Potpourri Plotting Possibilities

```
stan_trace(stanm0, pars = names(stanm0)[1:4])
```



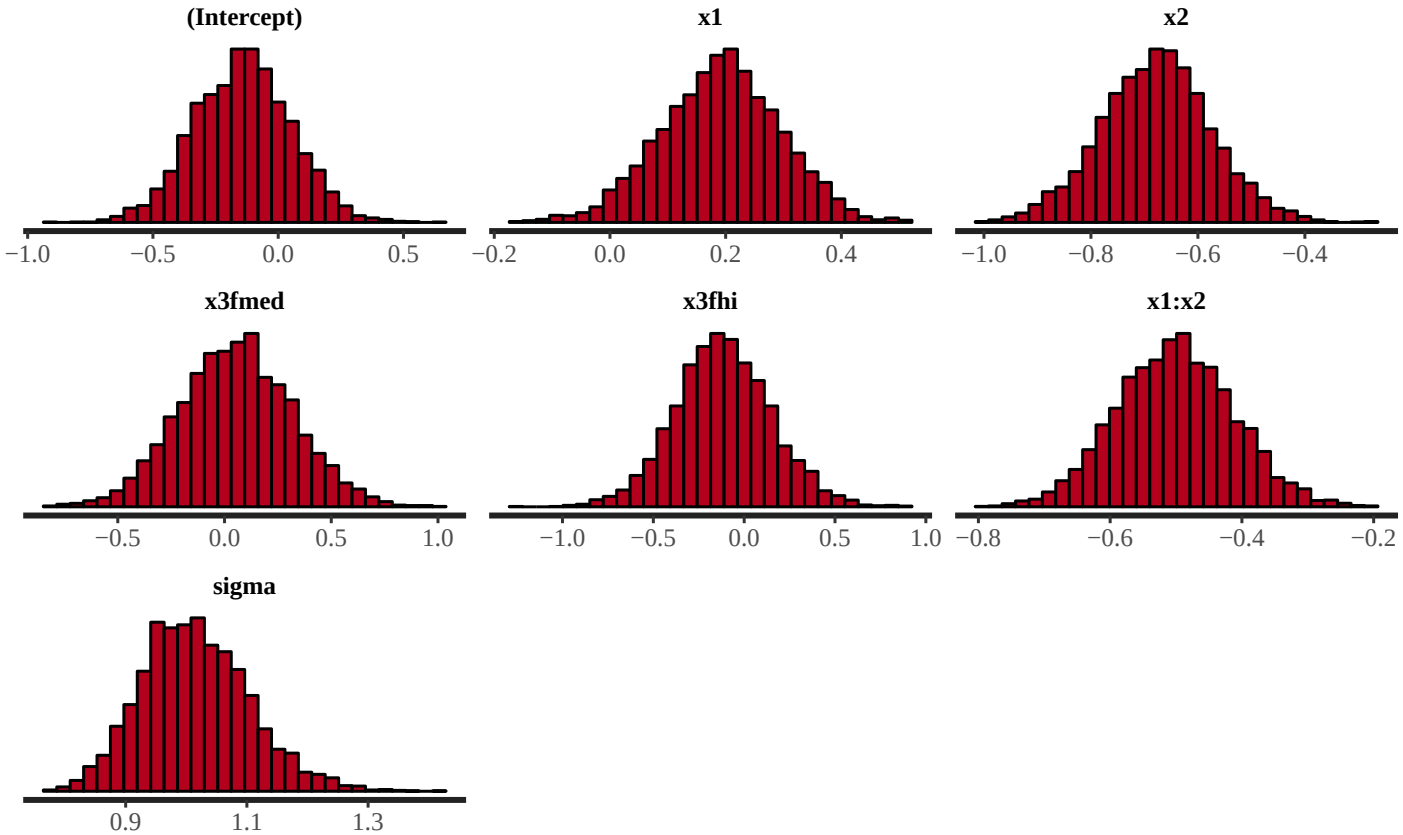
Prodigious Potpourri Plotting Possibilities

```
stan_trace(stanm0, pars = names(stanm0)[5:7])
```



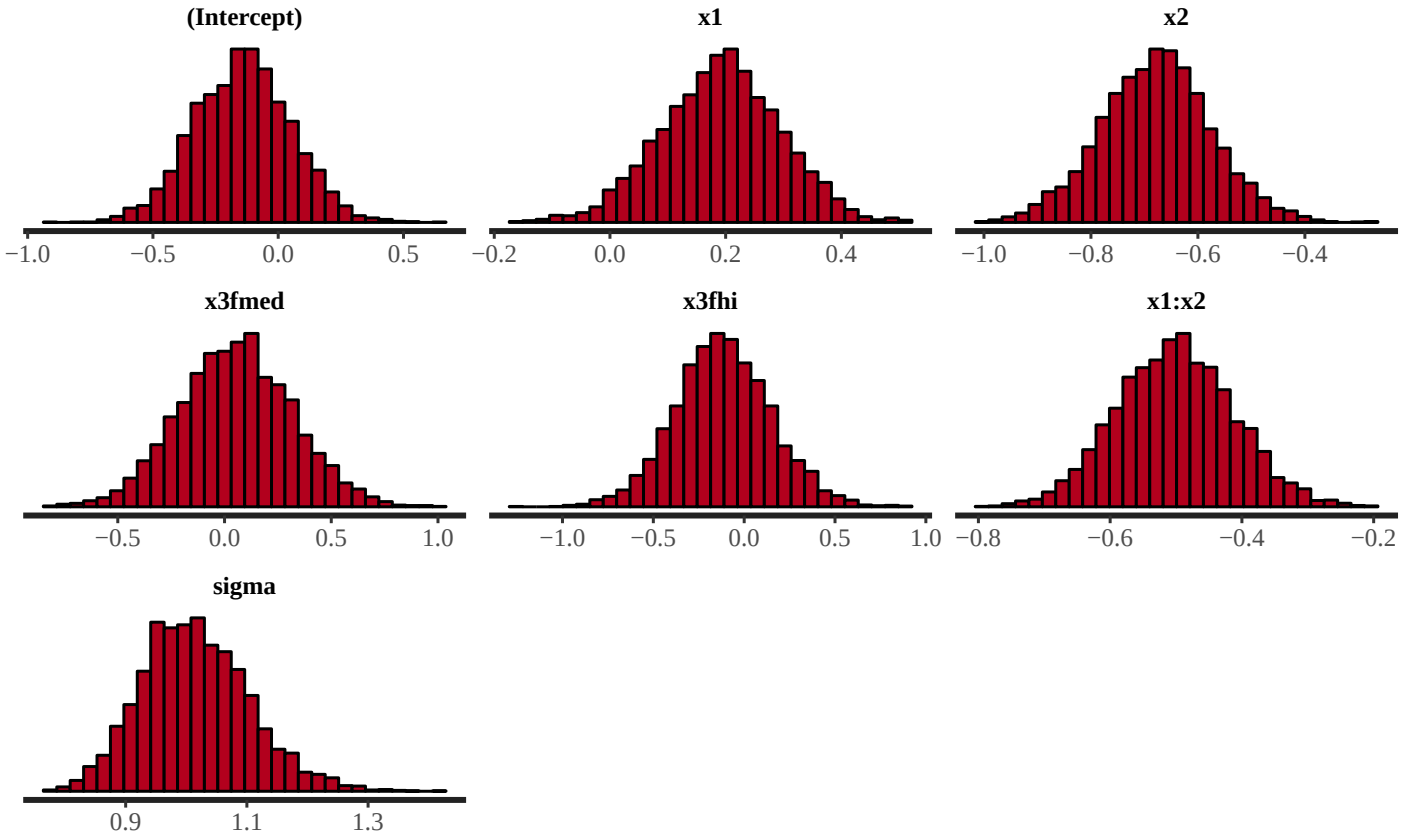
Prodigious Potpourri Plotting Possibilities

```
stan_hist(stanm0)
```



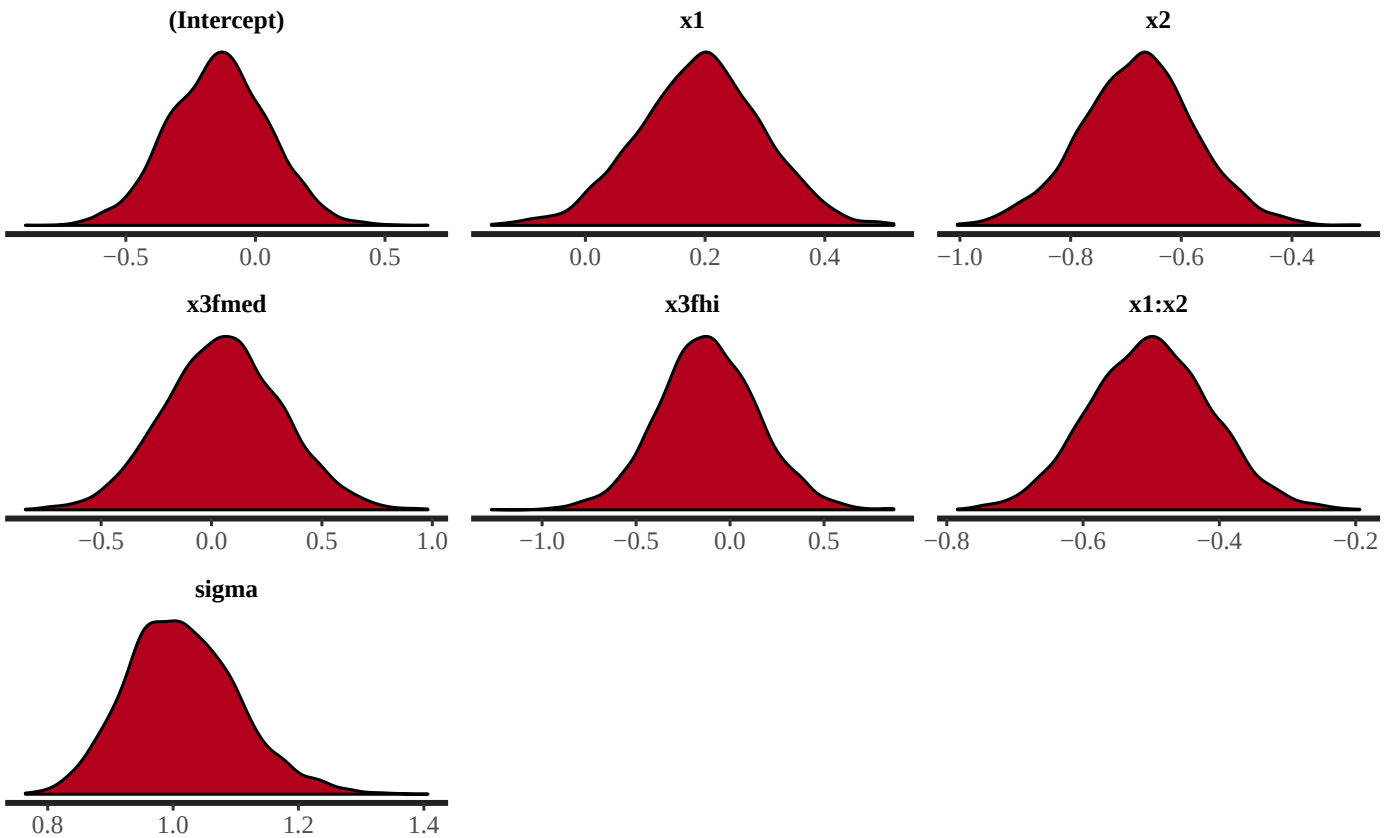
Prodigious Potpourri Plotting Possibilities

```
stan_hist(stanm0)
```



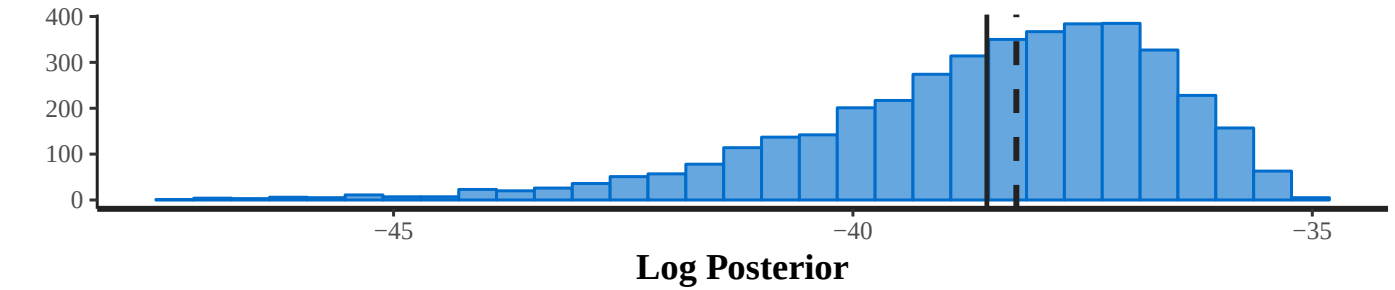
Prodigious Potpourri Plotting Possibilities

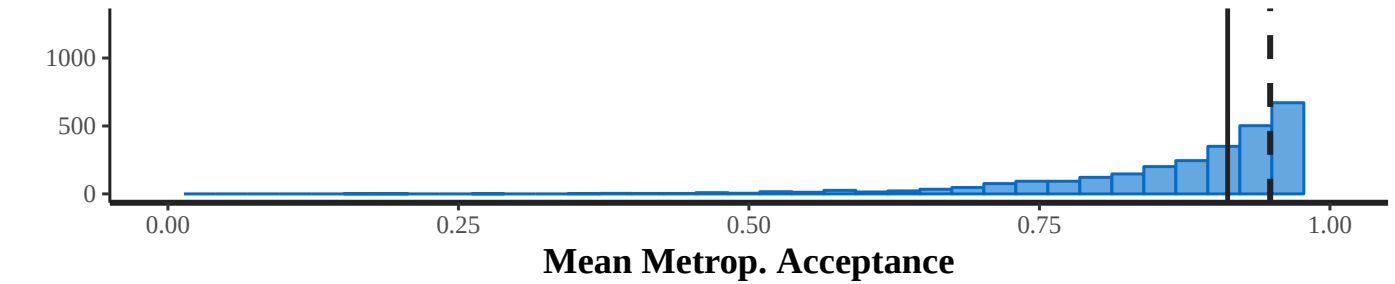
```
stan_dens(stanm0)
```

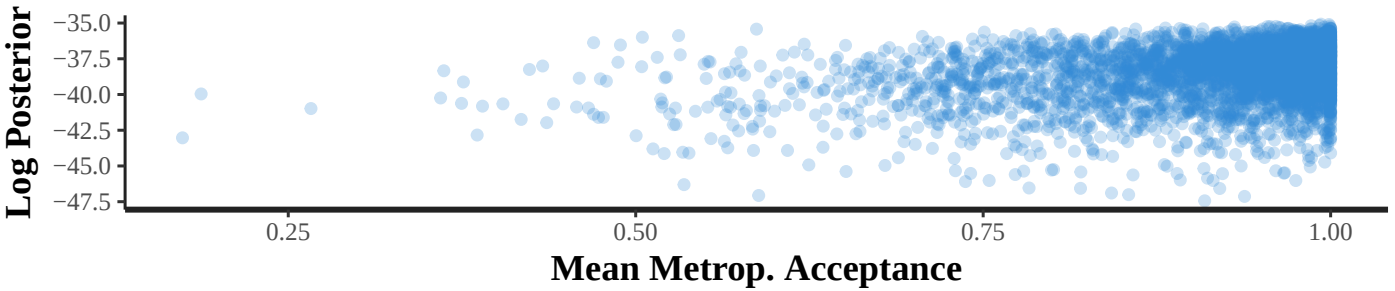


Prodigious Potpourri Plotting Possibilities

```
stan_diag(stanm0)
```

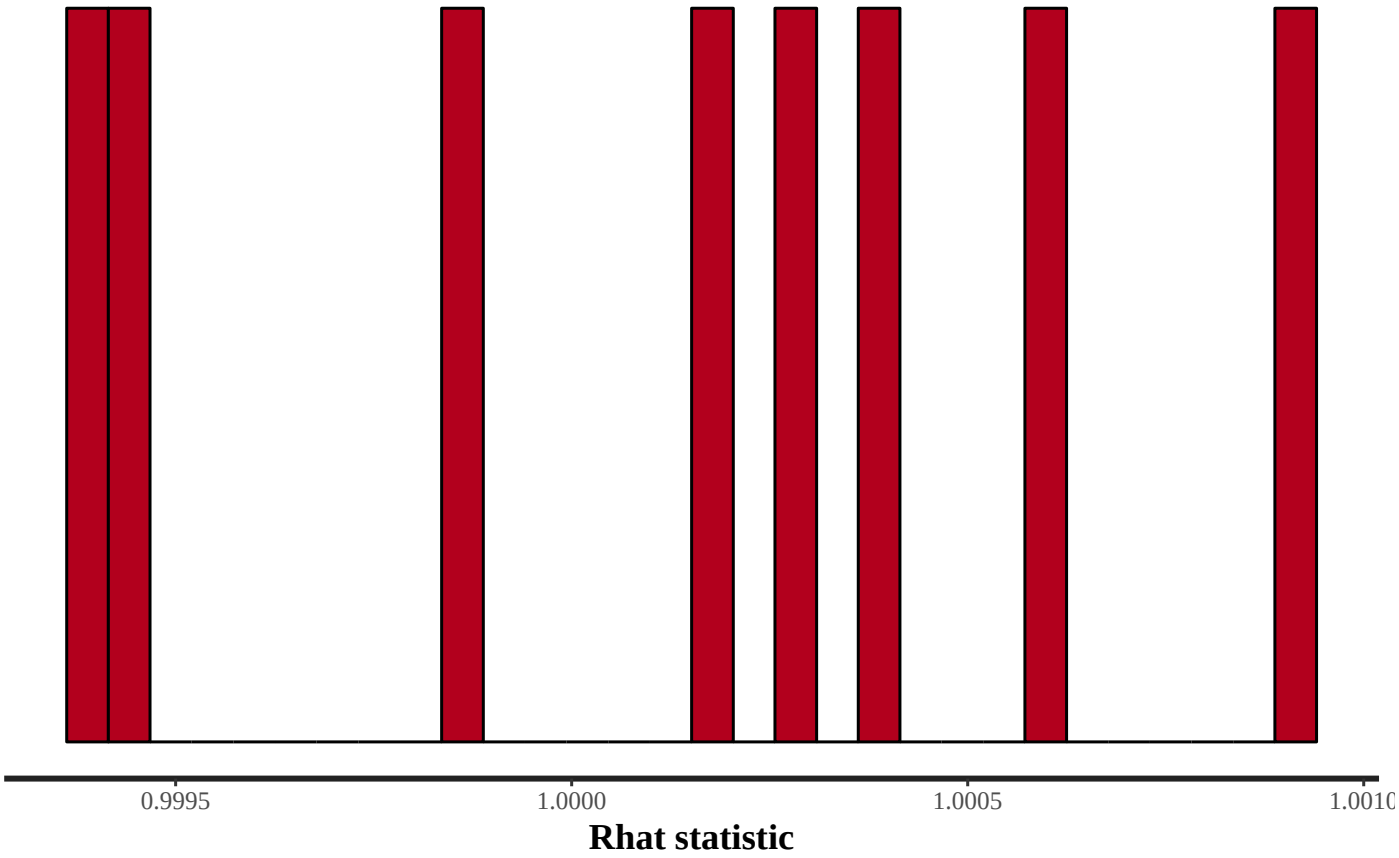






Prodigious Potpourri Plotting Possibilities

```
stan_rhat(stanm0)
```

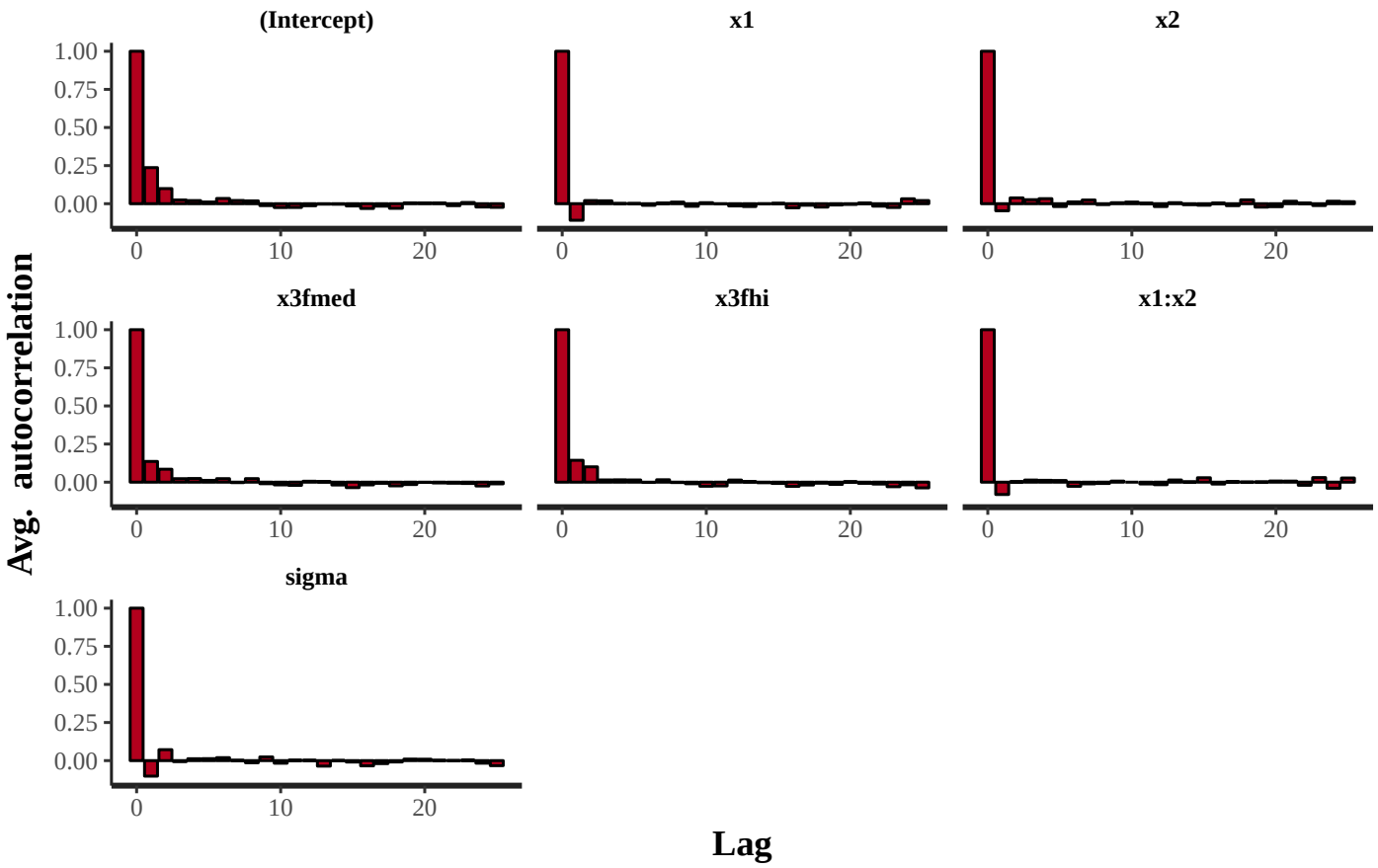


Prodigious Potpourri Plotting Possibilities

```
stan_mcse(stanm0)
```

Prodigious Potpourri Plotting Possibilities

```
stan_ac(stanm0)
```

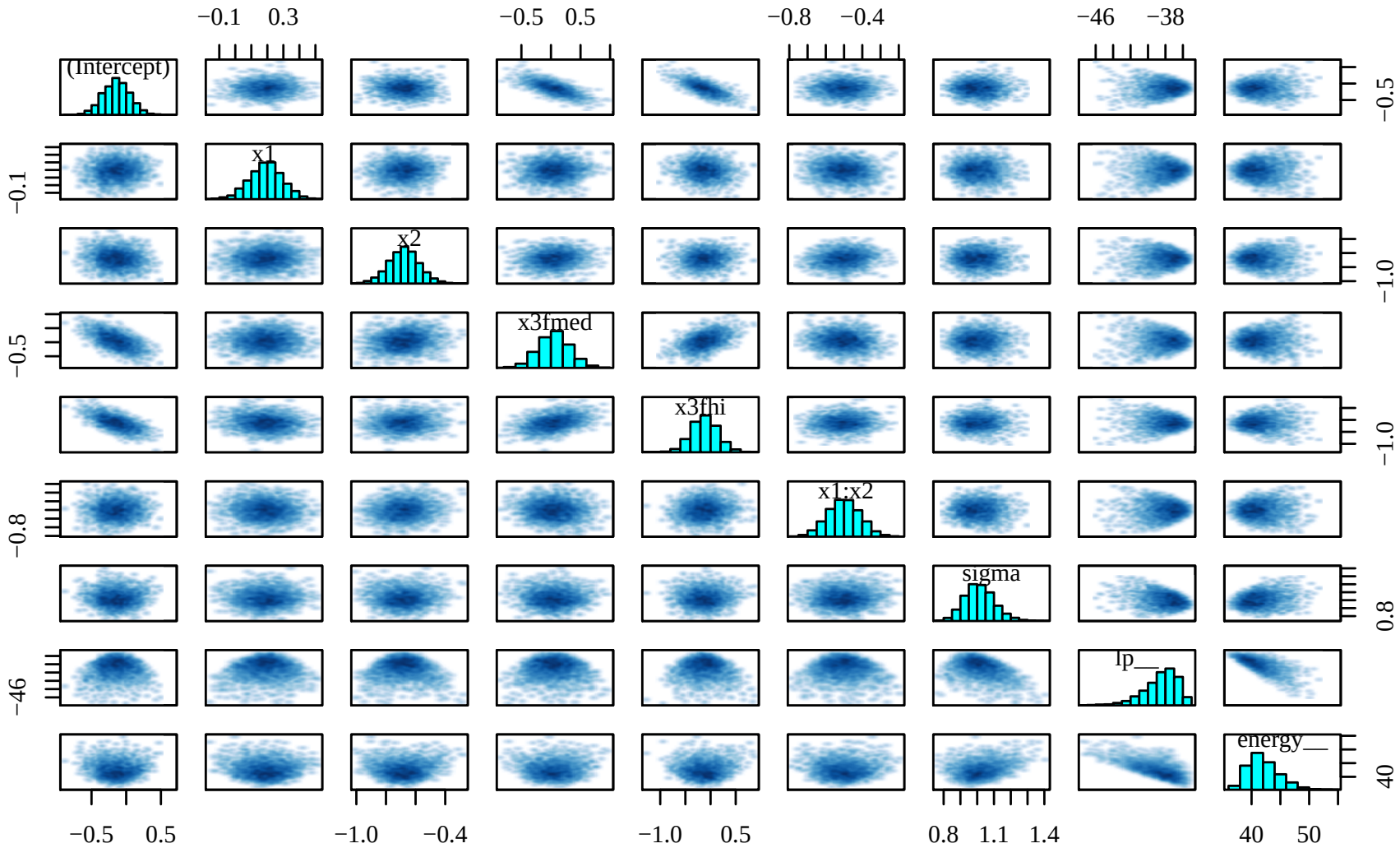


Prodigious Potpourri Plotting Possibilities

- `pairs()` is a standard R function that creates a scatterplot matrix

```
pairs(stanm0)
```

Prodigious Potpourri Plotting Possibilities ...



Did you wonder. . .

What is inside `stanm0`?

```
library(methods)
slotNames(stanm0)
```

```
[1] "model_name" "model_pars" "par_dims"   "mode"        "sim"
    "inits"      "stan_args"
[8] "stanmodel"  "date"       ".MISC"
```

```
stanm0@inits
```

Did you wonder. . . ...

```
[[1]]
[[1]]$beta
[1] -1.5807274  0.3399832  1.8385338  1.7618753 -0.9423795 -0.9462381

[[1]]$sigma
[1] 1.548685

[[1]]$linpred
 [1] -3.0714238  0.5673470 -0.9942555  3.4178029  0.7289505 -1.9472387  1.2706204 -3.0412673
 [9] -1.2069749 -3.9663877 -3.5785477 -1.6133143 -0.5005789  1.7833999  1.1951056 -1.4248036
[17] -1.3527515  2.2923197 -0.7759911  1.1385930 -3.9150260  1.3293299  1.0222583 -1.4940197
[25] -0.2916053 -2.6909589 -2.6683219  4.6737320 -4.5851937 -2.1879300 -0.9439036 -0.6873884
[33] -3.4910673 -0.9082170  0.9028212 -1.9496384 -5.5445230  2.9996281 -2.7512309  1.3292309
[41] -3.7800483 -0.6749405  1.7688077  2.2767360 -1.9470262 -0.2977052 -0.4530751 -8.7912500
[49] -6.0635359  3.4992612 -1.2324333 -0.2005089 -0.7661835 -2.4606058 -5.6664140 -3.3918904
[57] -2.0535818 -3.9970902 -0.6614822 -4.2070980 -3.4694223  1.1113622  1.7270592  2.1839439
[65] -1.9568242 -5.5427168  3.3157227  3.9888462 -3.7912256 -5.9036112 -1.0333172 -6.3249594
[73] -1.2379550 -0.2573993 -1.9761734


[[2]]
[[2]]$beta
[1]  1.277812 -1.213191  1.995320 -1.242699  1.604965  1.153057

[[2]]$sigma
[1] 1.443994

[[2]]$linpred
 [1] -0.48683925  0.14834859  4.76604991  3.68223831 -4.08644245 -4.49039795  3.38251590  1.34206491
 [9]  8.34919812  0.16387514  0.76544751  4.07826896  6.41424328  2.07120973  5.09479701 -4.81464132
[17] -2.31647072  5.60259574  1.96604771  1.05086177  0.84790316  5.48544388  0.98830626  4.11666796
[25]  5.12208017  0.29452324 -0.12554664  3.43150754 -1.34809203  0.10325289 -0.02156828  2.27342204
[33]  3.49966544 -4.34109955  1.71155285  0.89222599 -1.69846138  2.26292499  1.27271065  4.56487693
[41]  0.80801624  6.31849425  3.52604225  2.01283809 -0.44249507  3.25910597 -6.88584395  4.84143704
```

Did you wonder. . . ...

```
[49]  0.36701869  3.18563068  4.28187058  2.81472943 -5.32993973 -5.46425305 -1.54251189 -0.31683097
[57]  0.71440465  3.29121566 -3.44904743 -0.96710561  2.30479860  3.46863853  6.24342203  3.52553213
[65]  0.86109103  4.62831594  3.26528488  2.25813056  1.31426753  1.55235447 -2.63862303  1.42381236
[73]  1.55458851  1.67594922  3.82941628


[[3]]
[[3]]$beta
[1]  1.3802911 -1.3878559  1.6342230  1.2168373 -1.1237471 -0.4752688

[[3]]$sigma
[1] 1.227014

[[3]]$linpred
 [1] -0.03077192  2.51808531  3.46270447  7.24216830 -0.27194560 -3.26083984  5.66074687  2.49683072
 [9] -0.04488870  0.05872972  1.75363438  1.63776394  0.66629151  3.42648035  5.22257703 -1.77775577
[17]  0.85235972  5.59766231  1.62160749  1.93036753 -1.28578695  3.56271292  3.44863318  1.59067146
[25]  0.97089795  1.93504901 -1.79949721  8.55573230 -0.03326168 -1.98961662 -0.91440553  4.97725033
[33]  0.76282461 -1.87118388  0.77648060 -2.06292176 -1.58904395  6.16449848 -1.01454180  3.89486679
[41]  1.66034687  0.93575939  5.77043646  5.23578863 -2.46298043  1.28344696 -0.77670074 -1.44381477
[49] -0.85257706  5.78723889  1.32624749  2.28537175 -0.33198054 -3.41878322 -1.57810618  1.17055347
[57] -1.90931049  2.40869290  0.40324861 -2.22966101  1.83770647  1.45563331  4.50601779  6.29735870
[65]  2.75014942  0.63801887  7.94042185  7.62967432 -0.98958766 -0.08968606  0.74816515 -1.90605695
[73]  1.55517706  4.48585408  1.35389086


[[4]]
[[4]]$beta
[1] -0.7791828 -1.6918797 -0.5211262 -0.6484995  1.7113056  0.3751601

[[4]]$sigma
[1] 0.3744146

[[4]]$linpred
```

Did you wonder. . . ...

```
 [1] -0.42925925 -1.94745586  2.43793392 -0.29236415 -4.54696366 -2.48666280  0.40138657  1.80751485
 [9] -1.53252631  0.55697019  1.67593956  1.25728395 -3.38345237 -2.55421313  1.88459572 -3.52361509
[17] -1.31009831  1.02268336 -1.48831002 -3.31328584  1.06959443 -2.17091297 -1.56065865  1.07212296
[25] -3.11897374  0.96308664 -0.71007346 -0.22886148  1.14317759 -1.46584050 -3.74961546  1.76446475
[33]  2.44637325 -4.24755206 -4.37155561 -1.90598783  0.76263521 -0.98707126  0.03524920 -1.69745593
[41]  1.79060286 -0.80395257 -0.04325848 -1.15640747 -2.16193681 -2.53759870 -3.43348101  5.69696207
[49]  1.84497191 -2.05323433  0.48387785 -1.51829888 -2.82027736 -1.96916402  0.88688753  0.99166246
[57] -1.60596977  2.68860994 -2.39500420  0.61764647  1.63316609 -4.10312946  0.39237161  0.06585647
[65]  0.95805851  4.40274664  0.61697318 -0.50328291  1.18978169  2.32978079 -1.72742704  2.96019561
[73] -1.00228448  0.84980954  1.38252493
```

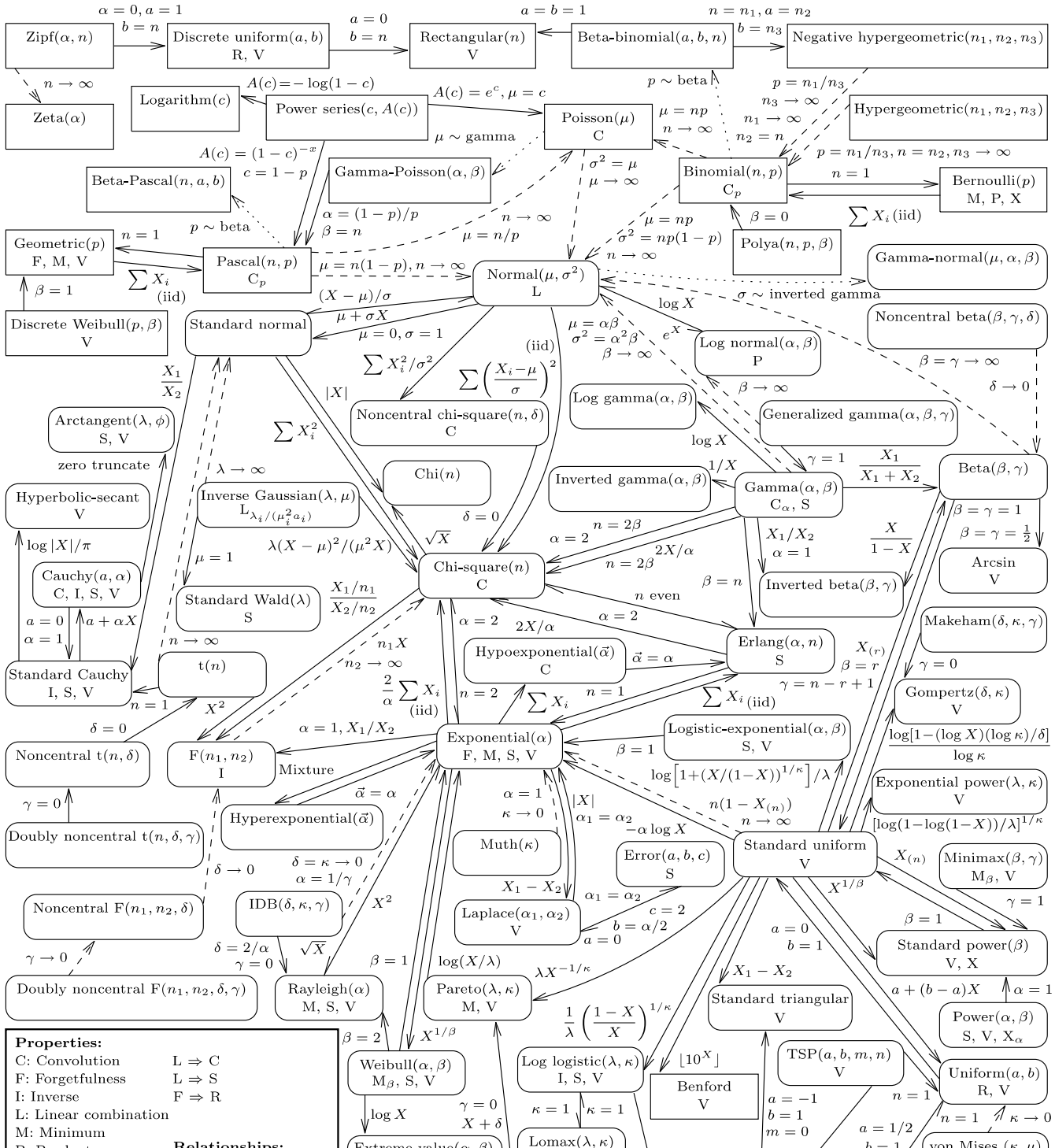
Outline

We get caught up in details

- We already have regression tools that are widely used.
- Get same results as Bayes method.
- What's the point again?

Practical Answer 1

- We don't have estimators for a lot of models we might dream up
- We have estimators for normally distributed things, and subject to some restrictions, related distributions
- The mosaic of distributions is immense


Zipf(α, n)
Discrete uniform(a, b)
R, V
Rectangular(n)
V
Beta-binomial(a, b, n)
Negative hypergeometric(n1, n2, n3)
Zeta(α)
Logarithm(c)
Power series(c, A(c))
Poisson(μ)
C
Hypergeometric(n1, n2, n3)
Beta-Pascal(n, a, b)
Gamma-Poisson(α, β)
Binomial(n, p)
Cp
Bernoulli(p)
M, P, X
Geometric(p)
F, M, V
Pascal(n, p)
Cp
Polya(n, p, β)
Discrete Weibull(p, β)
V
Normal(μ, σ2)
L
Gamma-normal(μ, α, β)
Standard normal
Noncentral beta(β, γ, δ)
Log normal(α, β)
P
Log gamma(α, β)
Generalized gamma(α, β, γ)
Arctangent(λ, φ)
S, V
Noncentral chi-square(n, δ)
C
Chi(n)
Inverted gamma(α, β)
Gamma(α, β)
Cα, S
Beta(β, γ)
Hyperbolic-secant
V
Inverse Gaussian(λ, μ)
Lλi/(μi2ai)
Chi-square(n)
C
Inverted beta(β, γ)
Arcsin
V
Cauchy(a, α)
C, I, S, V
Standard Wald(λ)
S
Makeham(δ, κ, γ)
Standard Cauchy
I, S, V
t(n)
Hypoexponential(α⃗)
C
Erlang(α, n)
S
Gompertz(δ, κ)
V
Noncentral t(n, δ)
F(n1, n2)
I
Exponential(α)
F, M, S, V
Logistic-exponential(α, β)
S, V
Exponential power(λ, κ)
V
Doubly noncentral t(n, δ, γ)
Hyperexponential(α⃗)
Muth(κ)
Error(a, b, c)
S
Standard uniform
V
Minimax(β, γ)
Mβ, V
Noncentral F(n1, n2, δ)
IDB(δ, κ, γ)
Laplace(α1, α2)
V
Standard power(β)
V, X
Doubly noncentral F(n1, n2, δ, γ)
Rayleigh(α)
M, S, V
Pareto(λ, κ)
M, V
Standard triangular
V
Power(α, β)
S, V, Xα
Weibull(α, β)
Mβ, S, V
Log logistic(λ, κ)
I, S, V
Benford
V
TSP(a, b, m, n)
V
Uniform(a, b)
R, V
Extreme value(α, β)
Lomax(λ, κ)
von Mises (κ, μ)
Properties:
C: Convolution
F: Forgetfulness
I: Inverse
L: Linear combination
M: Minimum
L ⇒ C
L ⇒ S
F ⇒ R
Relationships:

Practical Answer 2

- All ML estimators have the same weakness/frustration:

  We Don't Know How Accurate The Estimates Are.

- We have "asymptotic" theory and hypothesis tests (likelihood ratio, t-ratio) based tests that are all conducted "as if" the sample size is infinite
- Categorical outcomes (logistic regression) ML parameter estimates are biased and the "asymptotic" standard errors are not accurate in small sample.
  - Primary impetus behind Bayesian interest in item response theory (Albert, 1992; Albert & Chib, 1993). Note the dates on those famous publications.
- Could bootstrap to get standard errors, but bootstrap is justified as an approximation to Bayesian posterior analysis. Why bother with that?

KU

Practical Answer 3

- With exotic predictive models, we can't make any progress with Maximum Likelihood because there are too many parameters, too much nonlinearity.
- In Item Response Theory, we might estimate 1000s of parameters, ML simply does not work.
- Use Bayesian MCMC as a way to obtain ML estimates.

References

Albert, J. H. (1992). Bayesian estimation of normal ogive item response curves using gibbs sampling. *Journal of Educational and Behavioral Statistics*, 17(3), 251 –269.

Albert, J. H. & Chib, S. (1993). Bayesian Analysis of Binary and Polychotomous Response Data. *Journal of the American Statistical Association*, 88(422), 669–679. ArticleType: primary_article / Full publication date: Jun., 1993 / Copyright © 1993 American Statistical Association.

Lunn, D. & Spiegelhalter, D. J. (2013). *The BUGS book: a practical introduction to Bayesian analysis*. Texts in statistical science series. Boca Raton: CRC Press, Taylor & Francis Group. OCLC: ocn808810636.

Martin, A. D., Quinn, K. M., & Park, J. H. (2011). MCMCpack: Markov chain monte carlo in R. *Journal of Statistical Software*, 42(9), 22.

References ...

Plummer, M. (2010). JAGS - Just Another Gibbs Sampler.

Plummer, M. (2016). *rjags: Bayesian Graphical Models using MCMC*. R package version 4-6.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Session

```
sessionInfo()
```

```
R version 3.4.4 (2018-03-15)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04 LTS

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
     LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8
     LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8
    LC_IDENTIFICATION=C

attached base packages:
[1] methods   stats     graphics  grDevices utils     datasets  base

other attached packages:
```

Session ...

```
[1] rstan_2.17.3         StanHeaders_2.17.2 ggplot2_2.2.1
    MCMCpack_1.4-2       MASS_7.3-49
[6] coda_0.19-1          rockchalk_1.8.111

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.15         compiler_3.4.4     pillar_1.1.0
     nloptr_1.0.4         plyr_1.8.4
 [6] tools_3.4.4          lme4_1.1-17        digest_0.6.15
     tibble_1.4.2         gtable_0.2.0
[11] nlme_3.1-137         lattice_0.20-35    mgcv_1.8-23
     rlang_0.1.6          Matrix_1.2-14
[16] parallel_3.4.4       SparseM_1.77       gridExtra_2.3
     MatrixModels_0.4-1   stats4_3.4.4
[21] grid_3.4.4           nnet_7.3-12        inline_0.3.14
     minqa_1.2.4          car_2.1-6
[26] scales_0.5.0         mcmc_0.9-5         splines_3.4.4
     pbkrtest_0.4-7       colorspace_1.3-2
[31] labeling_0.3         quantreg_5.35      KernSmooth_2.23-15
     lazyeval_0.2.1       munsell_0.4.3
```