Bayesian Statistics for Genetics
10b Guided tour of software

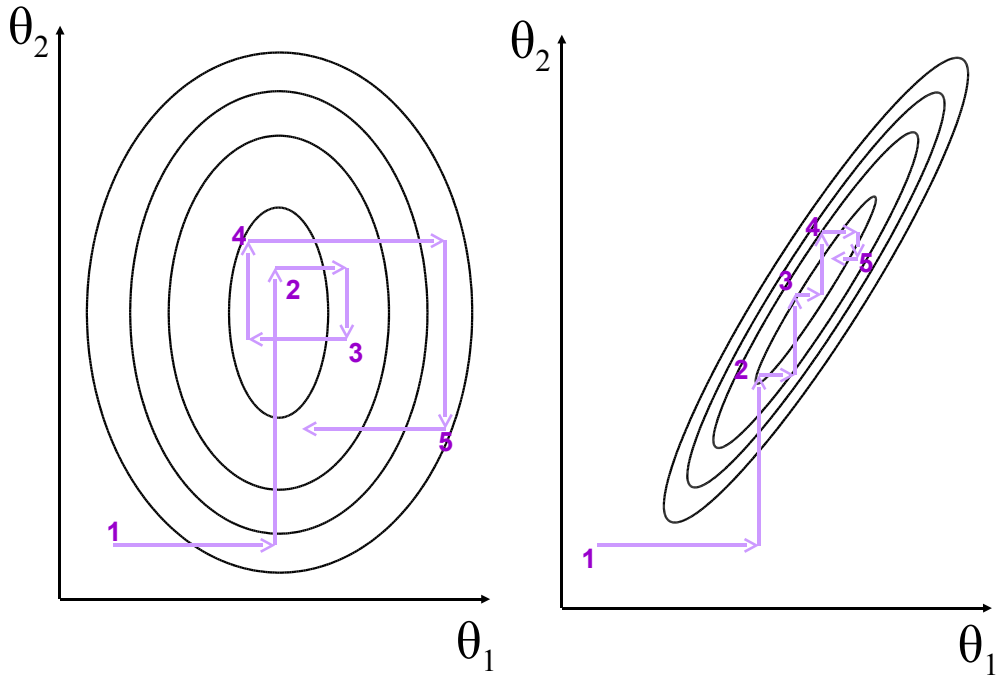
Ken Rice

UW Dept of Biostatistics

July, 2017

Off-the-shelf MCMC

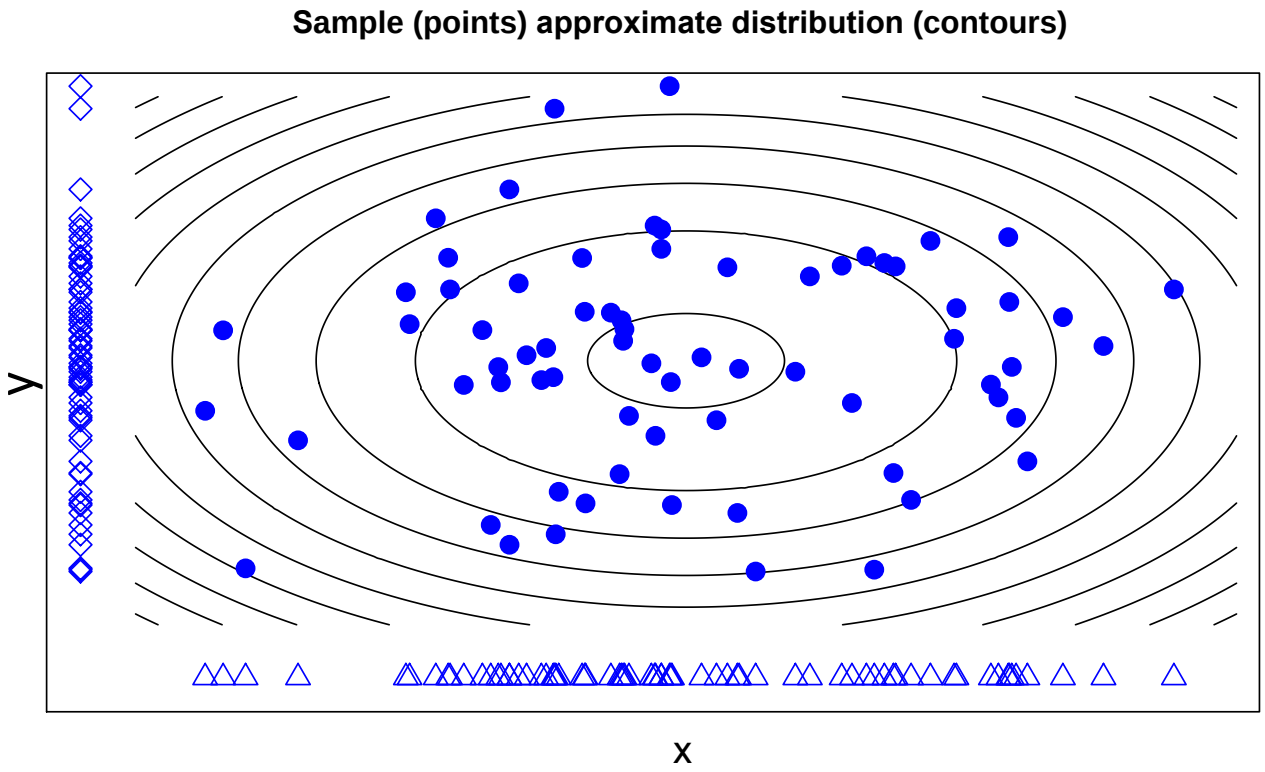
Recall the big picture of Bayesian computation;



We want a large sample from some distribution – i.e. the posterior. It **does not matter** if we get there by taking independent samples, or via some form of dependent sampling. (Gibbs Sampling, here)

Off-the-shelf MCMC

Once we have a big sample...



Any property of the actual posterior (contours) can be approximated by the *empirical* distribution of the samples (points)

Off-the-shelf MCMC

Markov Chain Monte Carlo (MCMC) is the general term for sampling methods that use Markov Chain processes to 'explore' the parameter space; the (many) random process values form our approximation of the posterior.

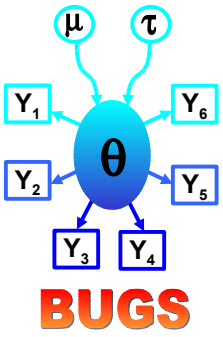
But in many settings this 'walking around' is mundane; once we specify the model and priors, the process of getting samples from the posterior can be done with no original thought – i.e. we can get a computer to do it.

Some example of this labor-saving approach;

- WinBUGS (next)
- ... or JAGS, OpenBUGS, NIMBLE and Stan
- INLA – not a Monte Carlo method

The R Task Views on Genetics and Bayesian inference may also have specialized software; see also Bioconductor.

Bayes: WinBUGS



Started in 1989, the **B**ayesian analysis **U**sing **G**ibbs **S**ampling (BUGS) project has developed software where users specify only model and prior – everything else is internal. WinBUGS is the most comprehensive version.

- The model/prior syntax is very similar to `R`
- ... with some annoying wrinkles – variance/precision, also column major ordering in matrices
- Can be 'called' from `R` – see e.g. `R2WinBUGS`, but you still need to code the model

Before we try it on GLMMs, a tiny GLM example ($n = 1, Y = 4$);

$$\begin{aligned} Y|\theta &\sim \text{Pois}\left(E\exp(\theta)\right) \\ \theta &\sim N(0, 1.797^2) \\ E &= 0.25 \end{aligned}$$

Bayes: WinBUGS

One (sane) way to code this in the BUGS language;

```
model{
  Y~dpois(lambda)            ...Poisson distribution, like R
  lambda <- E*exp(theta)     ...syntax follows R
  E <- 0.25                  ...constants could go in data
  theta~dnorm(m,tau)         ...prior for θ
  m <- 0
  tau <- 1/v                 tau = precision NOT variance!
  v <- 1.797*1.797
 }                           ...finish the model

#data
list(Y=4)                    Easiest way to input data
#inits
list(theta=0)                Same list format; or use gen.inits
```

Bayes: WinBUGS

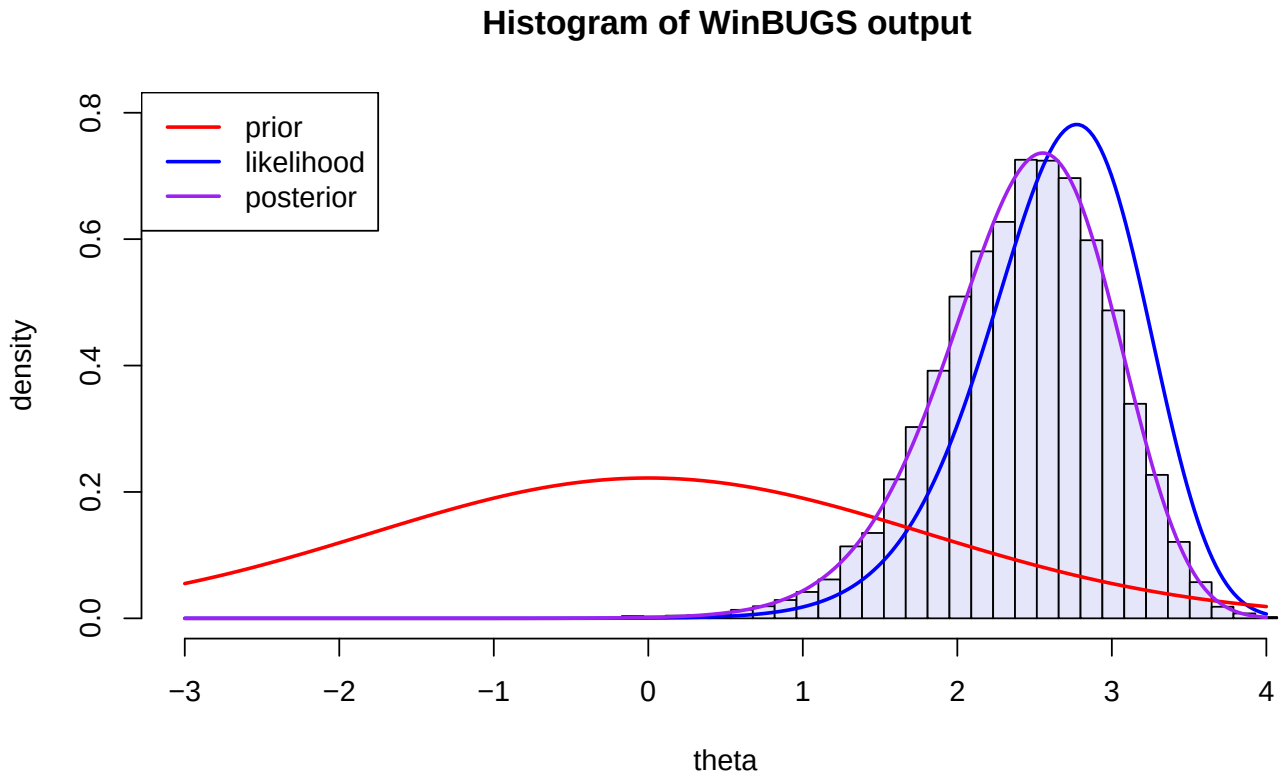
Notes on all this; (not a substitute for reading the manual!)

- This should look familiar, from the models we have been writing out. In particular '$\sim$' is used to denote distributions of data *and* parameters
- All 'nodes' appear **once** on the LHS; hard work is done on RHS
- No formulae allowed when specifying distributions
- Data nodes *must* have distributions. Non-data nodes *must* have priors – it's easy to forget these
- Write out regressions 'by hand'; `beta0 + beta1*x1 + ...`
- This language can't do everything; BUGS does not allow e.g.

```
Y <- U + V
U~dnorm(meanu,tauu); V~dt(meanv,tauv,k)
#data
list(Y=...)
```

Bayes: WinBUGS

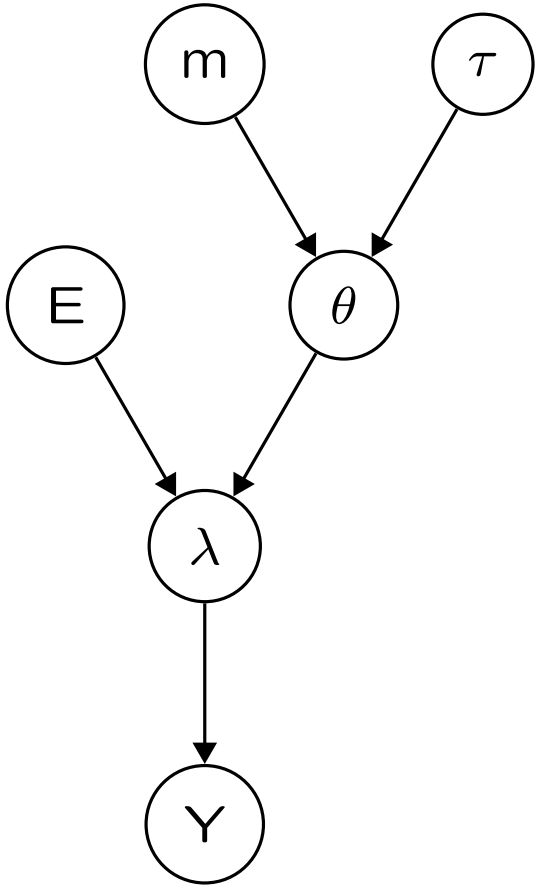
From 10,000 iterations, how do we do? (Note 'MC error' estimates Monte Carlo error in the posterior mean)



node	mean	sd	MC error	2.5%	median	97.5%
theta	2.422	0.5608	0.005246	1.229	2.466	3.388

Bayes: WinBUGS

Under the hood, here's how WinBUGS 'thinks';



- It's a DAG; arrows represent stochastic relationships (not causality)
- Some texts use square nodes for observed variables (Y, here)
- To do a Gibbs update, we need to know/work out the distribution of a node conditional on **only** its parents, children, and its children's other parents*.

* *This set is a node's 'Markov blanket'. The idea saves a lot of effort, and is particularly useful when fitting random effects models.*

WinBUGS: HWE example

A multinomial example, with a default prior;

$$\begin{aligned} \mathbf{Y} &\sim \text{Multinomial}(n, \boldsymbol{\theta}) \\ \text{where } \boldsymbol{\theta} &= (p^2, 2p(1-p), (1-p)^2) \\ p &\sim \text{Beta}(0.5, 0.5). \end{aligned}$$

And a typical way to code it in "the BUGS language";

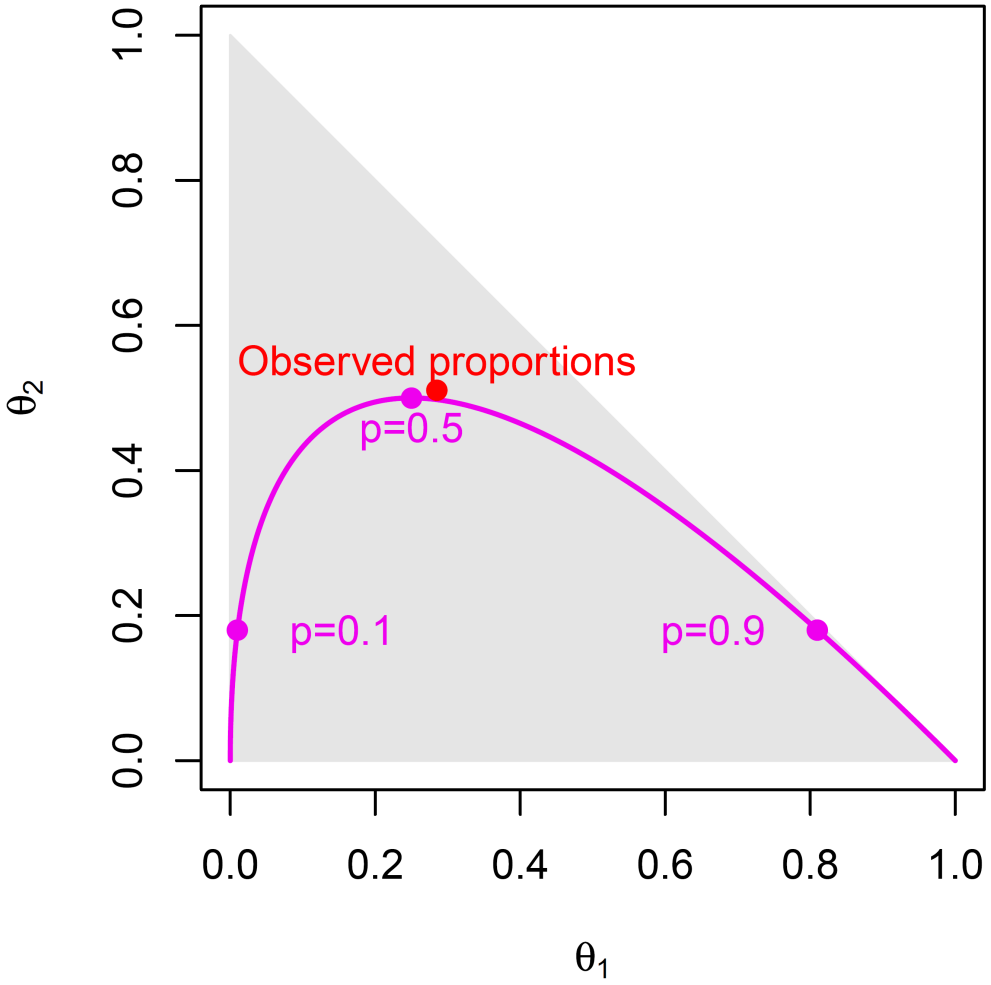
```
model{
   y[1:3] ~ dmulti(theta[], n)
   theta[1] <- p*p
   theta[2] <- 2*p*(1-p)
   theta[3] <- (1-p)*(1-p)
   p ~ dbeta(0.5, 0.5)
}
```

WinBUGS: HWE example

We have $n = 186$, and $\mathbf{Y} = (53,95,38)$.

We will run 3 chains, starting at $p = 0.5$, 0.1 and 0.9.

In WinBUGS, input these by highlighting two list objects:

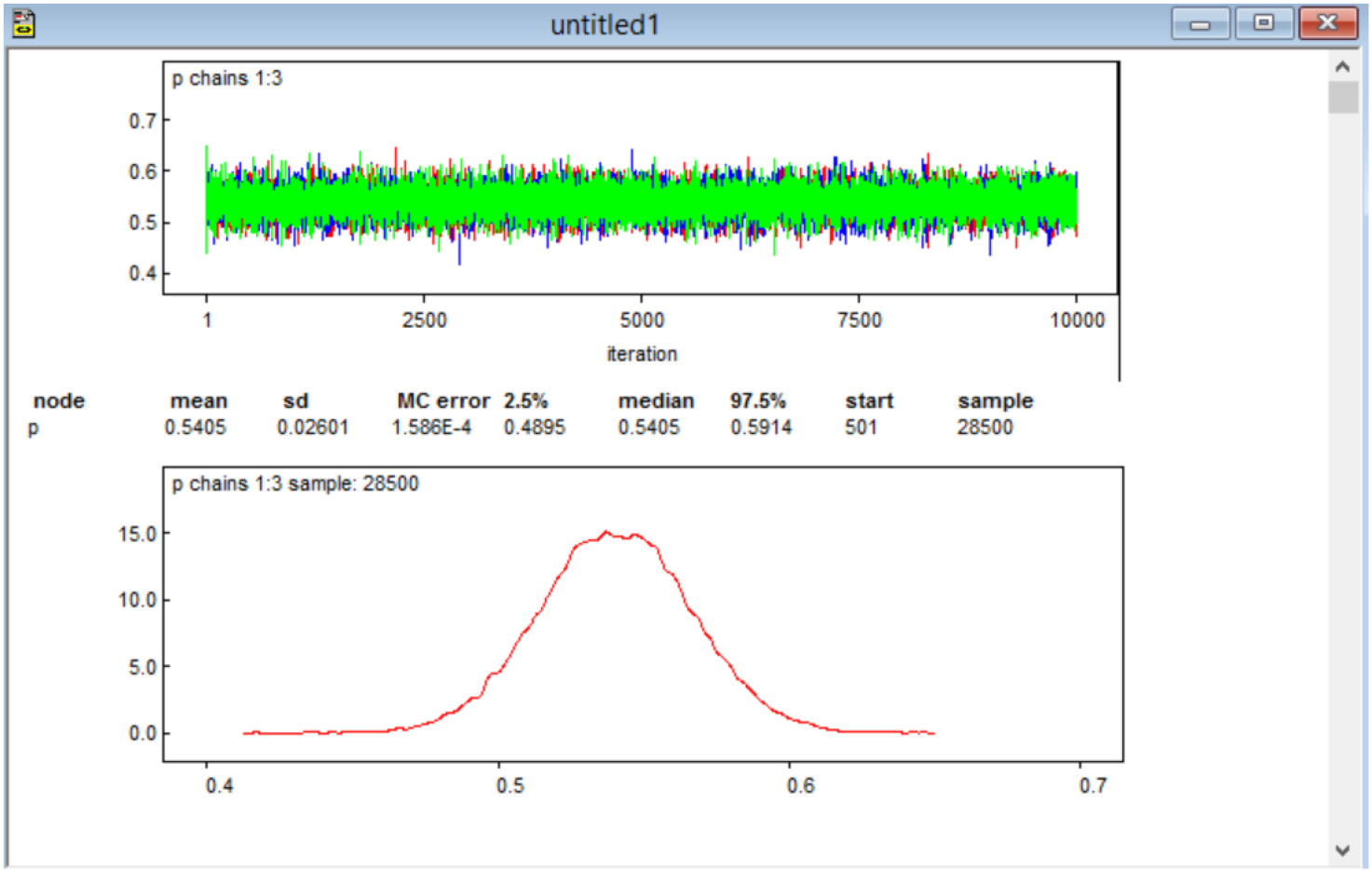



```
# Data
list(y=c(53,95,38),n=186)
```

```
# Initial values
  list(p=0.5)
  list(p=0.1)
  list(p=0.9)
```

WinBUGS: HWE example

WinBUGS unlovely but functional in-house output;



The posterior has 95% support for $p \in (0.49, 0.59)$, the posterior mean = posterior median = 0.54. Use `coda` to get the chain(s).

WinBUGS: less pointy-clicky

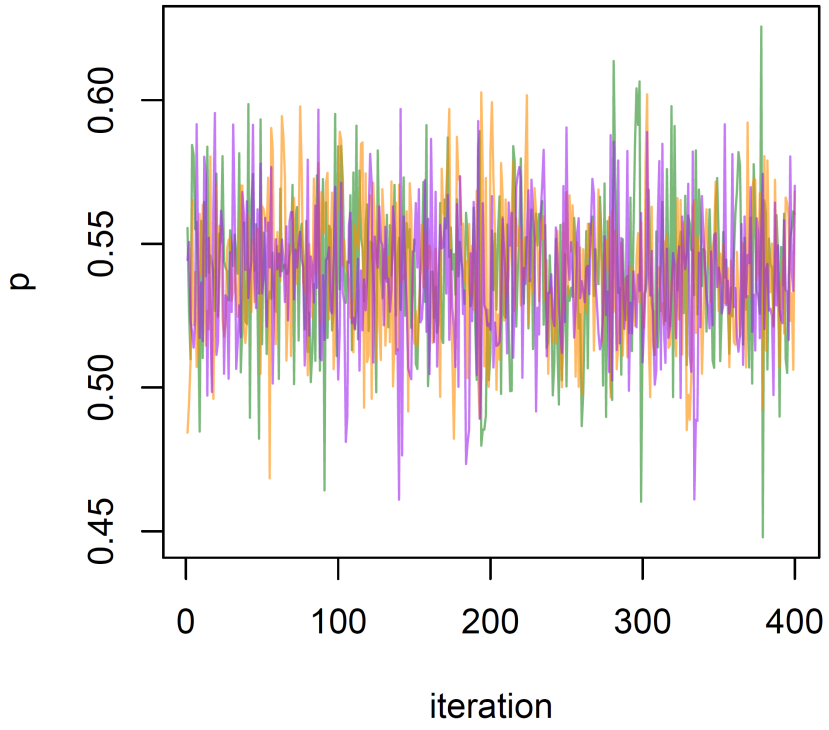
Apart from coming up with the model, everything can be automated, using R's `R2WinBUGS` package;

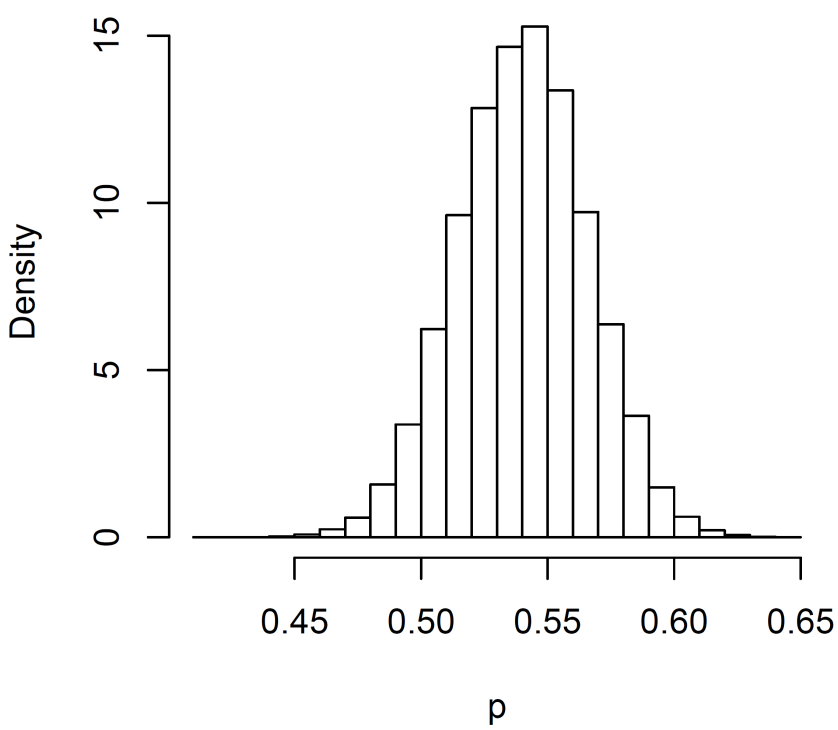
```
library("R2WinBUGS")
hweout <- bugs(data=list(y=c(53,95,38),n=186),
inits=list(p=0.5, p=0.1, p=0.9),
           parameters.to.save=c("p"),
           model.file="hweprog.txt",
           bugs.directory = "C:/Program Files/WinBUGS14",
           n.chains=3, n.iter=10000,
           n.burnin=500, n.thin=1, DIC=FALSE)
```

- Model code now in a separate file (`hweprog.txt`)
- Specify the data and initial values as R structures
- Tell R where to find WinBUGS
- The output is stored in `hweout`, an R object – no need to go via `coda`
- When debugging, pointy-clicky WinBUGS is still useful
- See next slide for less-clunky graphics

WinBUGS: less pointy-clicky

```
> print(hweout, digits=3)
Inference for Bugs model at "hweprog.txt", fit using WinBUGS,
 3 chains, each with 10000 iterations (first 500 discarded)
 n.sims = 28500 iterations saved
    mean      sd     2.5%      50%    97.5%    Rhat      n.eff
   0.540   0.026    0.490    0.541    0.590   1.001  28000.000
For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
```





WinBUGS: less pointy-clicky

- As well as the Markov blanket idea, WinBUGS uses what it knows about conjugacy to substitute closed form integrals in the calculations, where it can. (e.g. using inverse-gamma priors on Normal variances)
- Otherwise, it chooses from a hierarchy of sampling methods – though these are not cutting-edge
- Because of its generality, and the complexity of turning a model into a sampling scheme, don't expect too much help from the error messages
- Even when the MCMC is working correctly, it is possible you may be fitting a ridiculous, unhelpful model. WinBUGS' authors assume you take responsibility for that

Also, while Gibbs-style sampling works well in many situations, for some problems it's not a good choice. If unsure, check the literature to see what's been tried already.

WinBUGS: less pointy-clicky

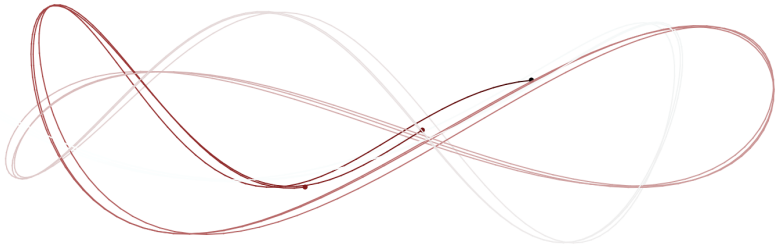
WinBUGS is no longer updated, but it's pointy-clicky interface remains a good place to get started. The BUGS language, describing models, is now used in JAGS, NIMBLE and OpenBUGS. Here's `rjags` using the **exact** same model file we just saw;

```
> library("rjags")
> jags1 <- jags.model("hweprog.txt", data=list(y=c(53,95,38),n=186) )
> update(jags1, 10000)
> summary( coda.samples(jags1, "p", n.iter=10000) )
Iterations = 11001:21000
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
     0.5398583 0.0258055    0.0002581          0.0003308
2. Quantiles for each variable:
  2.5%    25%    50%    75%    97.5%
0.4890 0.5225 0.5398 0.5576   0.5895
```

JAGS uses C, so is easier to extend than WinBUGS.

Stan

Stan is similar to BUGS, WinBUGS, JAGS etc – but new & improved;



- Coded in C++, for faster updating, it runs the *No U-Turn Sampler* – cleverer than WinBUGS' routines
- The `rstan` package lets you run chains from R, just like we did with `R2WinBUGS`
- Some modeling limitations – no discrete parameters – but becoming popular; works well with some models where WinBUGS would struggle
- Basically the same modeling language as WinBUGS – but Stan allows R-style vectorization
- Requires declarations (like C++) – unlike WinBUGS, or R – so models require a bit more typing...

Stan: HWE example

A Stan model for the HWE example

```
data {
    int y[3];
}
parameters {
   real<lower=0,upper=1> p;
}
transformed parameters {
   simplex[3] theta;
   theta[1] = p*p;
   theta[2] = 2*p*(1-p);
   theta[3] = (1-p)*(1-p);
}
model {
   p~beta(0.5, 0.5);
   y~multinomial(theta);
}
```

- More typing than BUGS!
- But experienced programmers will be used to this overhead

Stan: HWE example

With the model stored in `HWEexample.stan` (a text file) the rest follows as before;

```
> library("rstan")
> stan1 <- stan(file = "HWEexample.stan", data = list(y=c(53,95,38)),
+ iter = 10000, chains = 1)
> print(stan1)
Inference for Stan model: HWEexample.
1 chains, each with iter=10000; warmup=5000; thin=1;
post-warmup draws per chain=5000, total post-warmup draws=5000.
           mean se_mean   sd    2.5%     25%     50%     75%  97.5%  n_eff
p          0.54    0.00 0.03    0.48    0.52    0.54    0.56   0.60   5000
theta[1]   0.29    0.00 0.03    0.23    0.27    0.29    0.31   0.36   5000
theta[2]   0.49    0.00 0.01    0.48    0.49    0.50    0.50   0.50   4541
theta[3]   0.21    0.00 0.03    0.16    0.19    0.21    0.23   0.27   5000
lp__    -192.17    0.02 0.87 -194.71 -192.44 -191.81 -191.57 -191.49  2762

Samples were drawn using NUTS(diag_e) at Tue Jul 26 14:13:31 2016.
```

- Iterations in the `stan1` object can be used for other summaries, graphs, etc
- `lp__` is the log likelihood, used in (some) measures of model fit

INLA

We've already seen various examples of Bayesian analysis using Integrated Nested Laplace Approximation (INLA). For a (wide) class of models known as Gaussian Markov Random Fields, it gives a very accurate approximation of the posterior by 'adding up' a series of Normals.

- This approximation is not stochastic – it is not a Monte Carlo method
- Even with high-dimensional parameters, where MCMC works less well/badly, INLA can be practical
- INLA is so fast that e.g. 'leave-one-out' & bootstrap methods are practical – and can scale to GWAS-size analyses
- Fits **most** regression models – but not everything, unlike MCMC
- Non-standard posterior summaries require more work than manipulating MCMC's posterior sample

INLA

The `inla` package in R has syntax modeled on R's `glm()` function. And with some data reshaping, our HWE example is a GLM;

```
> y <- c(53,95,38) # 2,1,0 copies of allele with frequency "p"
> n <- 186
> longdata <- data.frame(y=rep(2:0, y), ni=rep(2, n) )
> # non-Bayesian estimate of log(p)/(1-log(p)) i.e. log odds
> glm1 <- glm( cbind(y,ni-y) ~ 1, data=longdata, family="binomial" )
> expit <- function(x){exp(x)/(1+exp(x) )}
> expit(coef(glm1))
(Intercept)
  0.5403226
> expit(confint(glm1))
    2.5 %    97.5 %
0.4895317 0.5905604
> inla1 <- inla( y~1, family="binomial", data=longdata, Ntrials=rep(2,n) )
> summary(inla1)$fixed
              mean     sd 0.025quant 0.5quant 0.975quant   mode kld
(Intercept) 0.1616 0.104    -0.0422   0.1615     0.3661 0.1612   0
> expit(summary(inla1)$fixed[,3:5]) # posterior of "p"
0.025quant  0.5quant 0.975quant
 0.4894516 0.5402875  0.5905163
```

For non-default priors, see the examples on the course site.