Lec 10: Bayesian Statistics for Genetics

Imputation and software

July, 2021

Overview

In this final session:

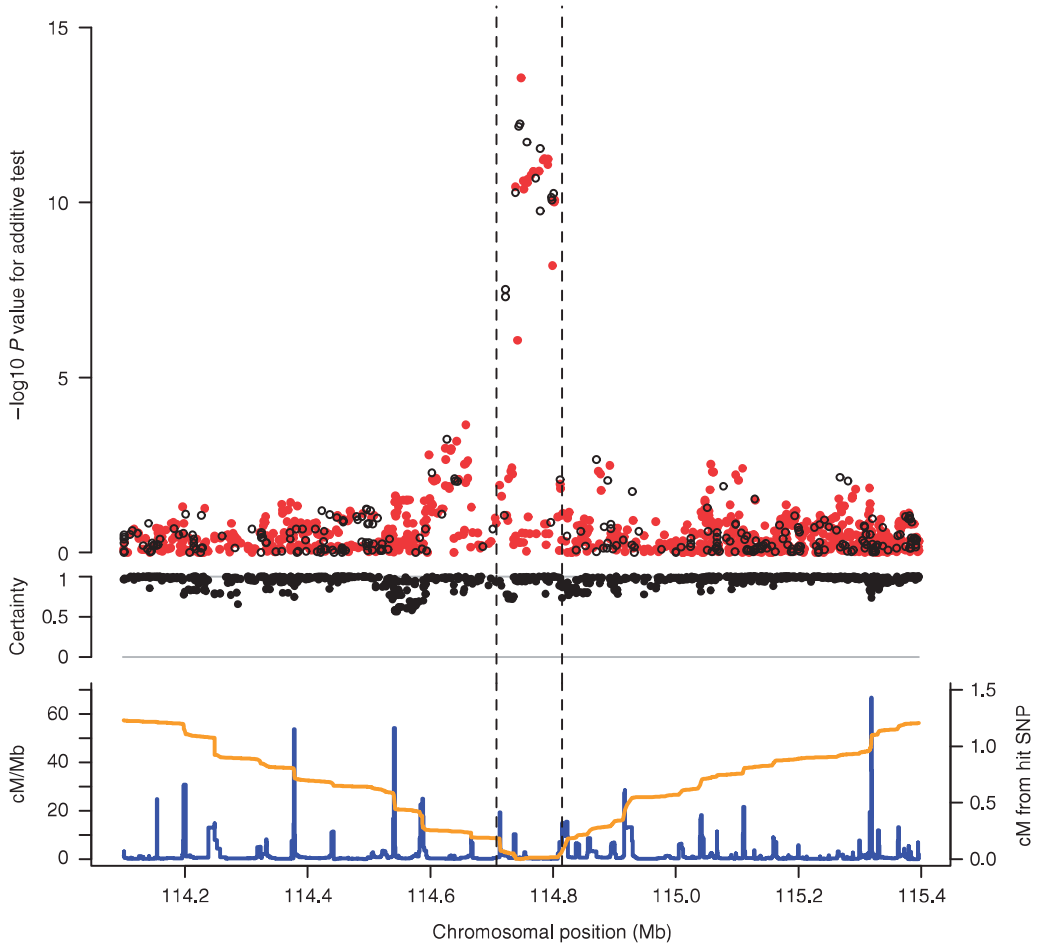
- Overview of imputation – where Bayesian (or approximately-Bayesian) methods are used to address a complex measurement error/missing data problem
- More details of off-the-shelf MCMC software
- No exercises – time for questions

Motivation for Imputation

- *Imputation* is the prediction of missing genotypes, using measured genotypes and prior information
- It is used widely in both GWAS and in fine-mapping studies, since it can:
  - Increase **power** in GWAS
  - Facilitate **meta-analysis** in which it is required to combine information from different panels which have different sets of SNPs. (This also helps power)
  - Fine-map **causal variants**. Imputed SNPs that show large associations can be better candidates for replication studies.
- The key idea in the approaches we describe is the use of data on haplotypes from a relevant population to build a *model* for the missing data – basically the models leverage linkage disequilibrium.

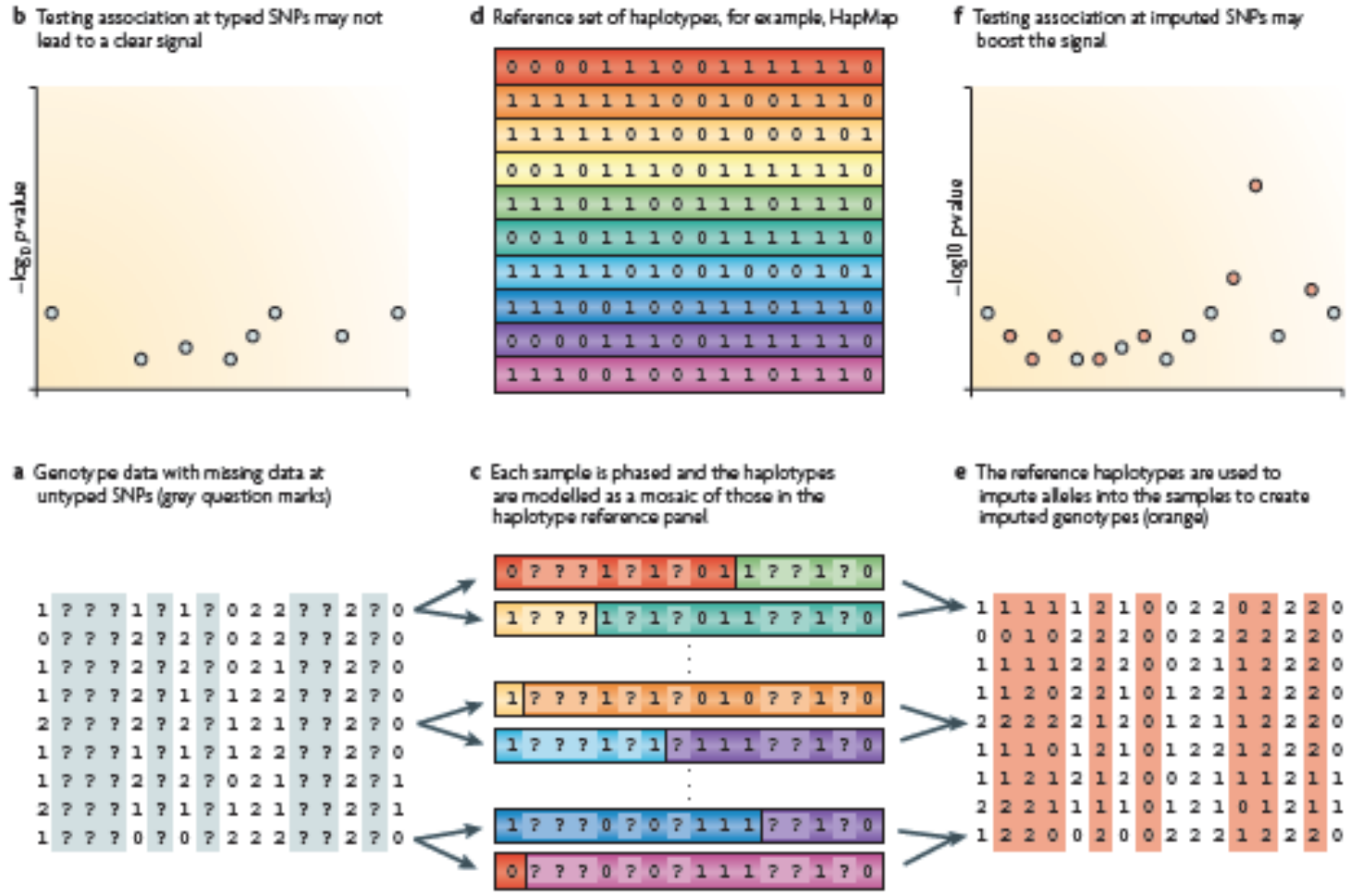
Motivation for Imputation

Imputation for the TCF7L2 gene, from Marcini et al (2007). Imputed SNP signals are in red and observed SNPs in **black**.



Imputation Overview

From Marchini & Howie (2010):



Statistical Framework

- Suppose we wish to estimate the association between a phenotype and m genetic markers in n individuals.
- Let G_{ij} represent the genotype of individual i at SNP j with G_{ij} unobserved for some SNPs.
- We consider diallelic SNPs so that G_{ij} can take the value 0, 1 or 2 depending on whether the pair of constituent SNPs are $\{0,0\}$, $\{0,1\}$, $\{1,0\}$ or $\{1,1\}$.
- If G_{ij} is observed then for SNP j we simply model $p(y_i|G_{ij})$
- For example, for continuous phenotype y_i we might assume a normal model:

$$\mathbb{E}[Y_i] = \beta_0 + \beta_1 G_{ij},$$

and if y_i is binary, a logistic model is an obvious candidate:

$$\frac{p_i}{1-p_i} = \exp(\beta_0 + \beta_1 G_{ij})$$

where p_i is the probability of disease for individual i.

Statistical Framework

- Let $\boldsymbol{H} = (\boldsymbol{H}_1, ..., \boldsymbol{H}_N)$ represent haplotype information at m SNPs in a relevant reference-panel, with N distinct haplotypes.
- Let $\boldsymbol{G}_i$ be the *observed* genotype information for individual i.
- If G_{ij} is unobserved then for SNP j we have the model

$$p(y_i|\boldsymbol{H}, \boldsymbol{G}_i) = \sum_{k=0}^{2} p(y_i|G_{ij} = k) \times \Pr(G_{ij} = k|\boldsymbol{H}, \boldsymbol{G}_i)$$

- The big question is how to obtain the predictive distribution

$$\Pr(G_{ij} = k|\boldsymbol{H}, \boldsymbol{G}_i)$$

- A common approach is to take as prior a *Hidden Markov Model (HMM)*
- ...so what's an HMM?

Hidden Markov Models: simple example

- A common problem is how to model count data over time. A Poisson model is the standard choice but we need to introduce:
  1. overdispersion and
  2. dependence over time.
- First consider the model:

  Stage 1: $Y_t|\lambda_t \sim$ Poisson(λ_t), $t = 1, 2, ...$

  Stage 2: $\lambda_t|Z_t \sim_{iid} \begin{cases} \lambda_0 & \text{if } Z_t = 0 \\ \lambda_1 & \text{if } Z_t = 1 \end{cases}$

  Stage 3: $Z_t|p \sim_{iid}$ Bernoulli(p).
- This adds overdispersion – the mean varies between λ_0 and λ_1, with probabilities $1 - p$ and p respectively

Hidden Markov Models: simple example

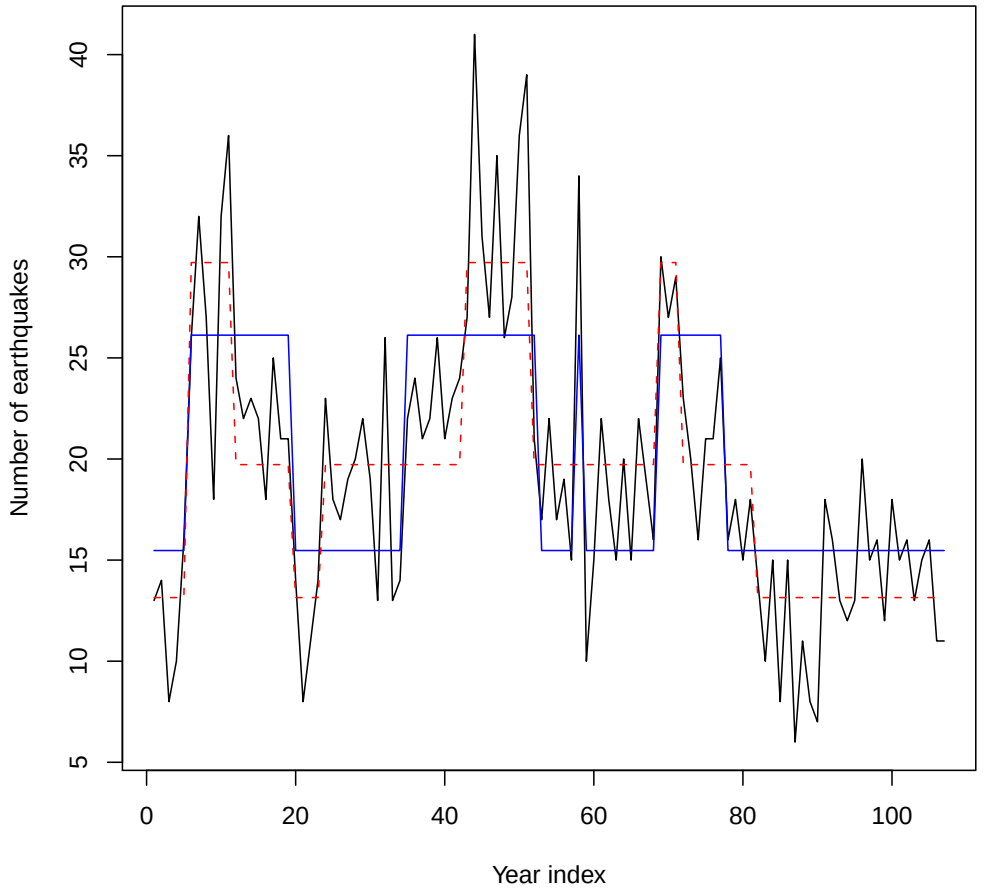
To introduce dependence over time, we replace *Stage 3* with a (first-order) *Markov chain model*, i.e, $\Pr(Z_t|Z_1, ..., Z_{t-1}) = \Pr(Z_t|Z_{t-1})$:

$$\begin{aligned} \Pr(Z_t = 0|Z_{t-1} = 0) &= p_0 \\ \Pr(Z_t = 1|Z_{t-1} = 1) &= p_1 \end{aligned}$$

- Z_t is an unobserved ('hidden' or 'latent') state.
- The next value of Z depends on the current Z, but no further back – the chain is 'memoryless'
- As an example we consider the number of major earthquakes (magnitude 7 and above) for the years 1990–2006.
- We illustrate the fit of this model with two or three underlying states.

HMM: Earthquake Data

Earthquake data along with the underlying states for the two and three state HMMs, in blue and red, respectively.



IMPUTE v1

- Marchini et al (2007) consider a HMM for the vector of genotypes for individual i:

$$\Pr(\boldsymbol{G}_i|\boldsymbol{H},\theta,\rho) = \sum_{\boldsymbol{z}_i=(\boldsymbol{z}_i^{(1)},\boldsymbol{z}_i^{(2)})} \Pr(\boldsymbol{G}_i|\mathbf{Z}_i,\theta) \times \Pr(\mathbf{Z}_i|\boldsymbol{H},\rho)$$

  where $\mathbf{Z}_i^{(1)} = \{Z_{i1}^{(1)},...,Z_{iJ}^{(1)}\}$ and $\mathbf{Z}_i^{(2)} = \{Z_{i1}^{(2)},...,Z_{iJ}^{(2)}\}$.
- The $(\mathbf{Z}_i^{(1)},\mathbf{Z}_i^{(2)})$ are the pair of haplotypes for SNP j from the reference panel that are copied to form the genotype vector. These are the hidden states.
- $\Pr(\mathbf{Z}_i|\boldsymbol{H},\rho)$ models how the pair of copied haplotypes for individual i changes along the sequence. This probability changes according to a *Markov chain* with the switching of states depending on the *fine-scale recombination rate*, which is denoted ρ.

IMPUTE v1

- The term $\Pr(\boldsymbol{G}_i|\mathbf{Z},\theta)$ allows the observed genotypes to differ from the pair of copied haplotypes through mutation; the mutation parameter is θ.
- `IMPUTE v2` (Howie et al, 2009) is a more flexible version that alternates between phasing and haploid imputation.

fastPHASE and BIMBAM

- We describe the model of Scheet & Stephens (2006)
- A *Hidden Markov Model (HMM)* is used to determine $\Pr(G_{ij} = k|\boldsymbol{\alpha}, \boldsymbol{\theta}, r)$.
- The basic idea is that haplotypes tend to cluster into groups of similar haplotypes; suppose there are K clusters.
- The unobserved *hidden* or *latent* state is the haplotype cluster from which this SNP arose from. Each cluster has an associated set of allele frequencies $\boldsymbol{\theta}_{kj}$.
- With K underlying states we have, for SNP j, α_{kj} being the probability of arising from haplotype k, with

$$\sum_{k=1}^{K} \alpha_{kj} = 1$$

`fastPHASE` and `BIMBAM`

- The model is

$$\Pr(\boldsymbol{G}_i|\boldsymbol{\alpha},\boldsymbol{\theta},r) = \sum_{\boldsymbol{z}} \Pr(\boldsymbol{G}_i|\mathbf{Z}_i,\boldsymbol{\theta}) \times \Pr(\mathbf{Z}_i|\boldsymbol{\alpha},r)$$

  with Z_{ij} the haplotype of origin for individual i and SNP j.
- A Markov chain is constructed for Z_{ij} with the strength of dependence being based on the recombination rate r at a given location.
- Given $Z_{ij} = k$, the genotype assigned depends on the allele frequencies of the k-th haplotype at the j-th SNP.

Use in Association Studies

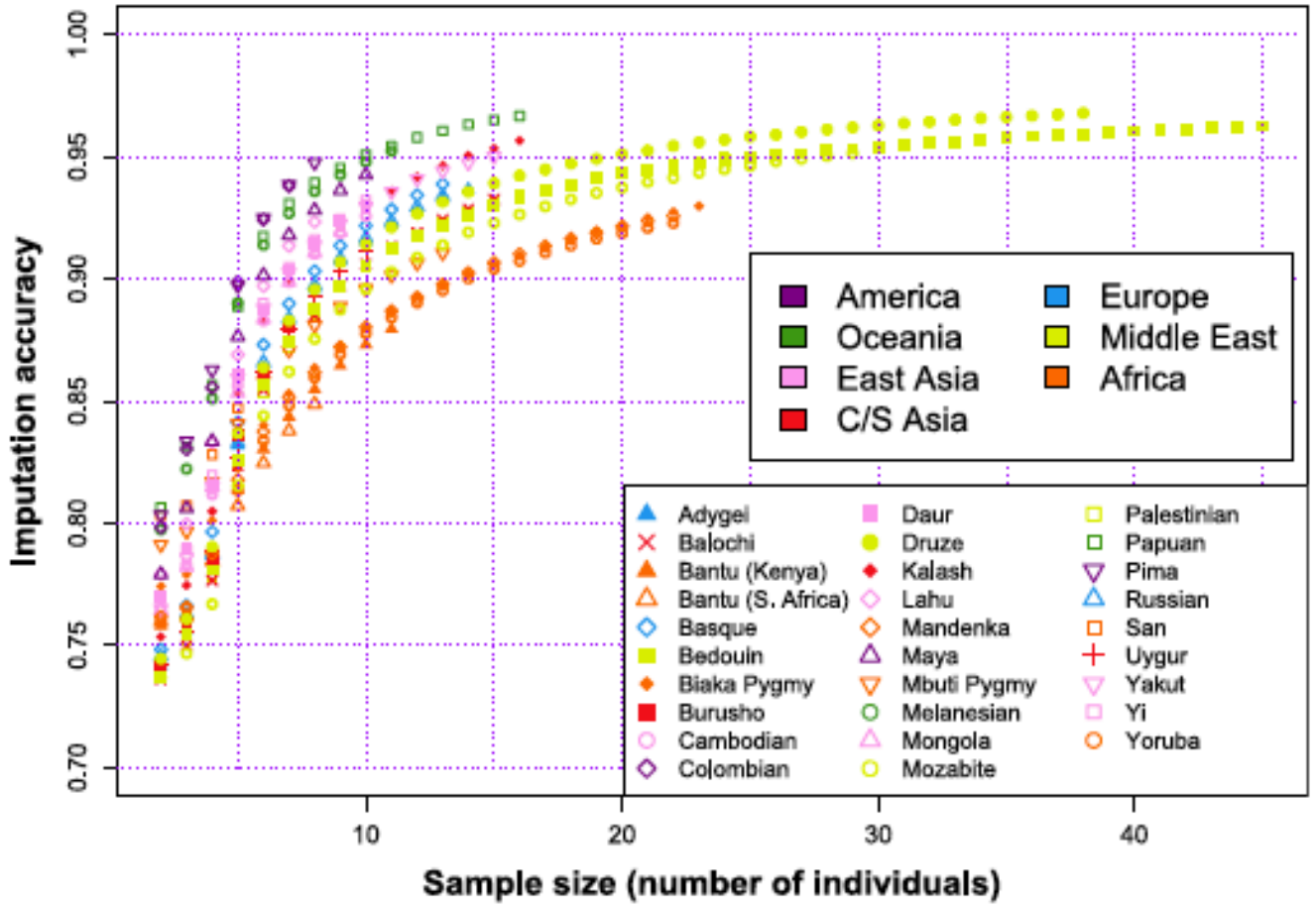
- The simplest approach to using imputed SNPs is to substitute $\widehat{G}_{ij}$ (a number between 0 and 2) into the phenotype association model – and it gives valid tests
- Imputation quality is also used. This is based on the variance of the imputations – it's zero if everyone got the same $\widehat{G}$
- A set of probabilities $\Pr(G_{ij} = k|\boldsymbol{G}, \boldsymbol{H})$ for $k = 0, 1, 2$ are produced and these may be used to average over the uncertainty in the phenotype model.
- Within `BIMBAM` the unknown genotype is sampled from its posterior distribution, within an MCMC framework.
- Other approaches:
  - `MACH`: (Li et al 2010) similar methodology to `IMPUTE`
  - `Beagle`: (Browning & Browning, 2009) uses a graphical model for haplotypes

Practical Issues

- One may attempt to match the haplotype panel (e.g. from HapMAP 2) with the study individuals.
- An alternative approach is to use all available haplotypes, and assigning equal prior probabilities to each.
- Many studies, for example Huang et al (2009), have examined SNP imputation accuracy in different populations.

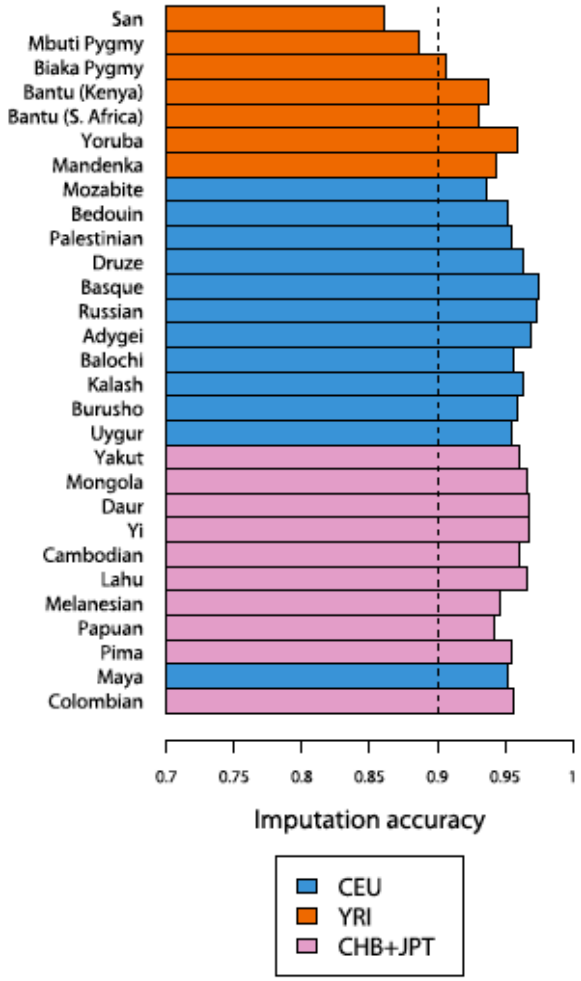
Practical Issues

Imputation accuracy as a function of sample size, from Huang et al (2009); (more recent approaches impute everyone together)



Practical Issues

Imputation accuracy for different populations with a reference-panel of 120 haplotypes, also from Huang et al (2009)



Practical Issues

Table 1. Association Analysis results.

Locus	SNPname	Type	Effect Allele/ Other	Freq Effect Allele	Effect (SE)[a]	P-value	Genomic Annotation	Variance explained by the locus	Top GWAS SNP	Effect Allele/ Other	Freq Effect Allele	Effect (SE)[a]	P-value	r2	Adjusted P-value	Variance explained by the locus
PCSK9	rs11591147	Metabochip	T/G	0.037	−0.380 (0.048)	2.90×10^{-15}	**missense (R46L)**	1.19%	rs11206510	C/T	0.243	−0.106 (0.023)	5.71×10^{-07}	0.101	0.013	0.23%
	rs2479415	1000G	C/T	0.413	0.076 (0.019)	7.50×10^{-05}	8 Kb from *PCSK9*									
SORT1	rs583104	Metabochip	T/G	0.177	0.149 (0.024)	1.28×10^{-09}	31 Kb from *SORT1*[b]	0.63%	rs599839	G/A	0.276	−0.148 (0.025)	1.43×10^{-09}	0.991	0.90	0.61%
B3GALT4	rs28361085	1000G	C/T	0.073	0.114 (0.036)	0.00169	146 Kb from *B3GALT3*	0.22%	rs2254287	G/C	0.492	0.005 (0.018)	0.771	0.413	0.84	0.02%
B4GALT4	rs34507110	1000G	G/A	0.154	0.122 (0.030)	4.99×10^{-05}	83 Kb from *B4GALT4*	0.48%	rs12695382	A/G	0.075	−0.074 (0.035)	0.035	0.795	0.48	0.03%
APOB	rs547235	1000G	A/G	0.187	−0.144 (0.024)	1.69×10^{-09}	140 Kb from *APOB*	0.51%	rs562338	A/G	0.173	−0.139 (0.025)	1.43×10^{-8}	0. 878	0.98	0.43%
LDLR	rs73015013	Metabochip	T/C	0.138	−0.155 (0.027)	1.12×10^{-08}	9 kb from *LDLR*	1.17%	rs6511720	T/G	0.132	−0.160 (0.027)	1.71×10^{-08}	0.934	0.97	0.59%
	rs72658864	Metabochip	C/T	0.005	0.626 (0.136)	3.90×10^{-06}	**missense (V578A)**									
APOC1/C2/E	rs7412	Metabochip	T/C	0.037	−0.563 (0.048)	1.80×10^{-31}	**missense (R176C)** *APOE*	3.33%	rs4420638[c]	G/A	0.097	0.218 (0.031)	4.67×10^{-12}	0.0003	6.41×10^{-10}	1.07%
	rs429358	Affy+Sanger	C/T	0.071	0.260 (0.036)	5.82×10^{-11}	**missense (C130R)** *APOE*									

The left panel shows the association results at 7 loci. For each gene, the strongest variant is listed first, and any second detected independent signal is listed with results from the conditional analysis (Materials and Methods). The column Type indicates whether the SNP was directly genotyped (Metabochip) or imputed using 1000G reference haplotype (1000G) or the Sardinian reference panel (Affy+Sanger). The right panel shows the association results for the GWAS SNPs previously described [5], the correlation with the top SNP listed in the left panel, and its p-value in the conditional analysis (Adjusted P-value).

[a]Effect sizes are standardized (see Materials and Methods), and represent the change in trait LDL-C values associated with each copy of the reference allele, measured in standard deviation units.

[b]SNP rs583104 is also 1 Kb from *PSRC1* transcript.

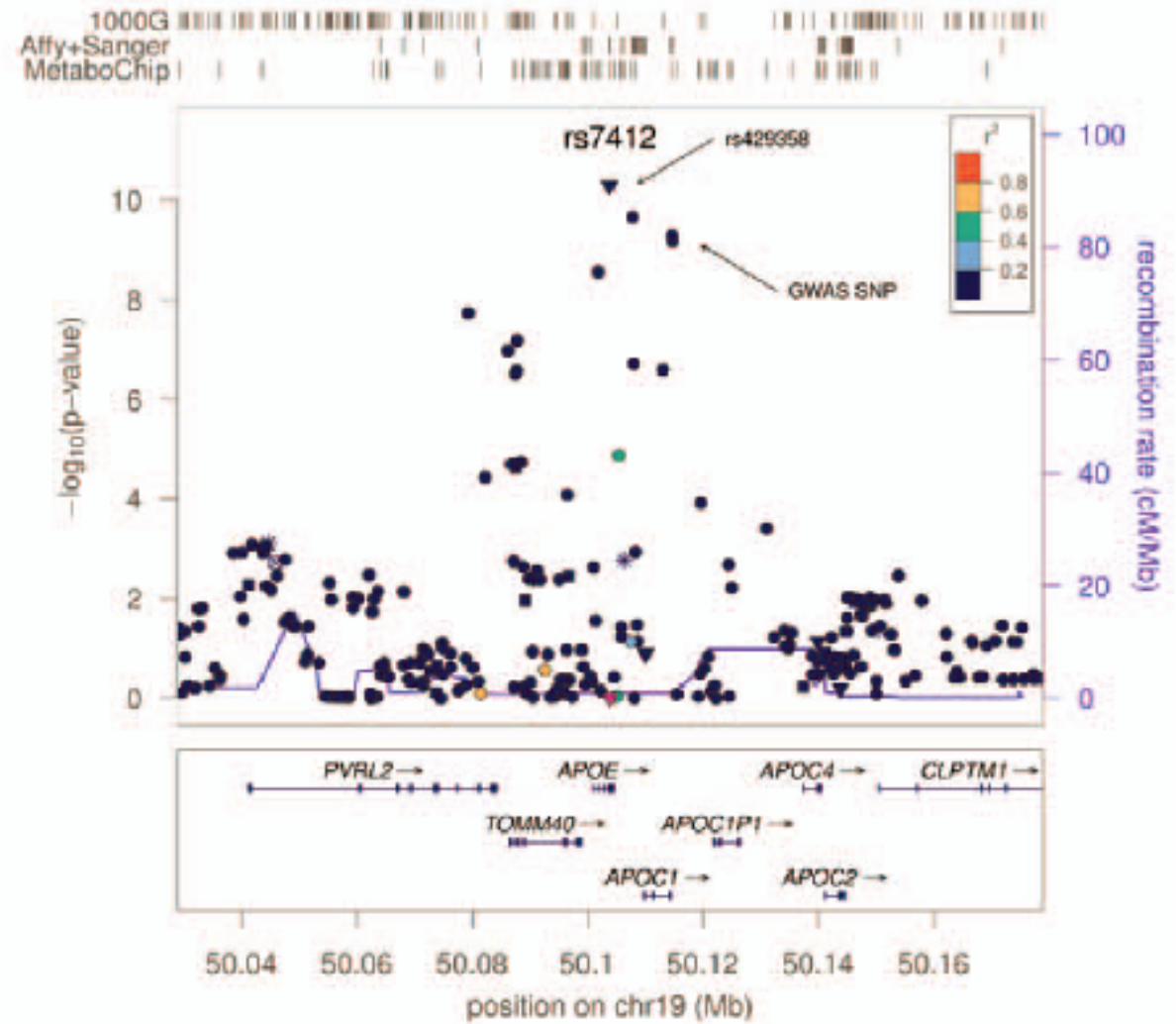
[c]$r^2=0.967$ with Metabochip second-independent SNP, rs429358. After adjusting for the two independent SNPs, rs7412 and rs429358, the p-value for rs4420638 was 0.5.

doi:10.1371/journal.pgen.1002198.t001

Example results from Sanna et al (2011), with imputation carried out using the `MACH` software.

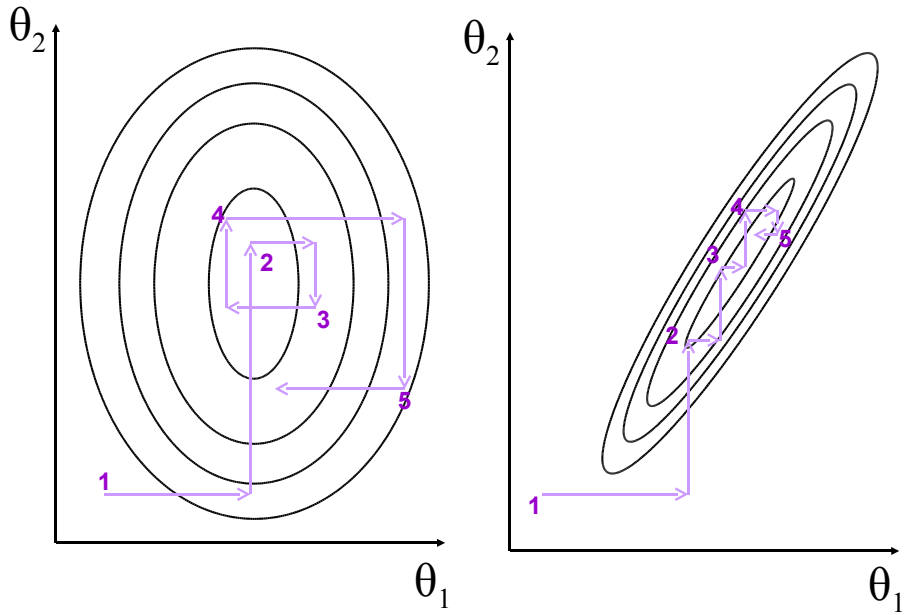
Practical Issues

Regional association plot from Sanna et al (2011).



Off-the-shelf MCMC

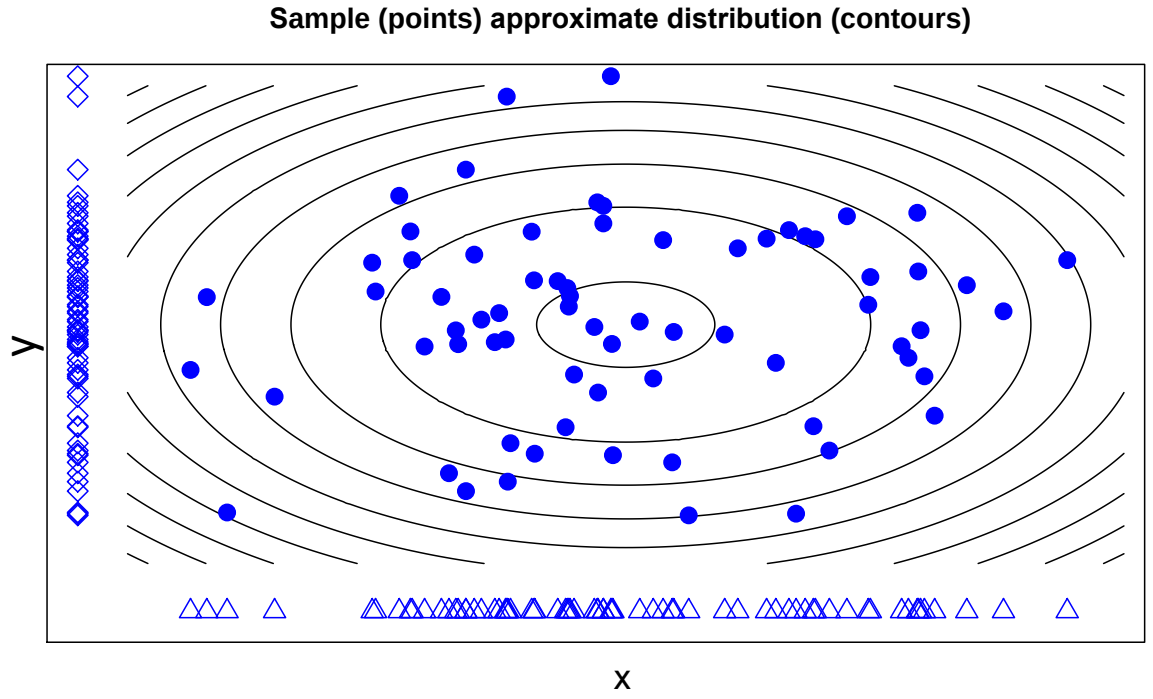
Recall the big picture of Bayesian computation;



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.

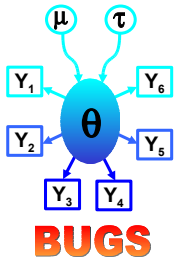
But in many settings this 'walking around' is mundane; once we specify the model and priors, the process of getting samples from the posterior can be done with no original thought – i.e. we can get a computer to do it.

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 popular version.

- The model/prior syntax is very similar to `R`
- ...with some wrinkles – variance/precision, also column/rows in matrices
- Can be 'called' from `R` – see e.g. `R2WinBUGS`, much like `rstan`, `rjags`

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

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

...the BUGS code will look like an R representation of these statements.

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

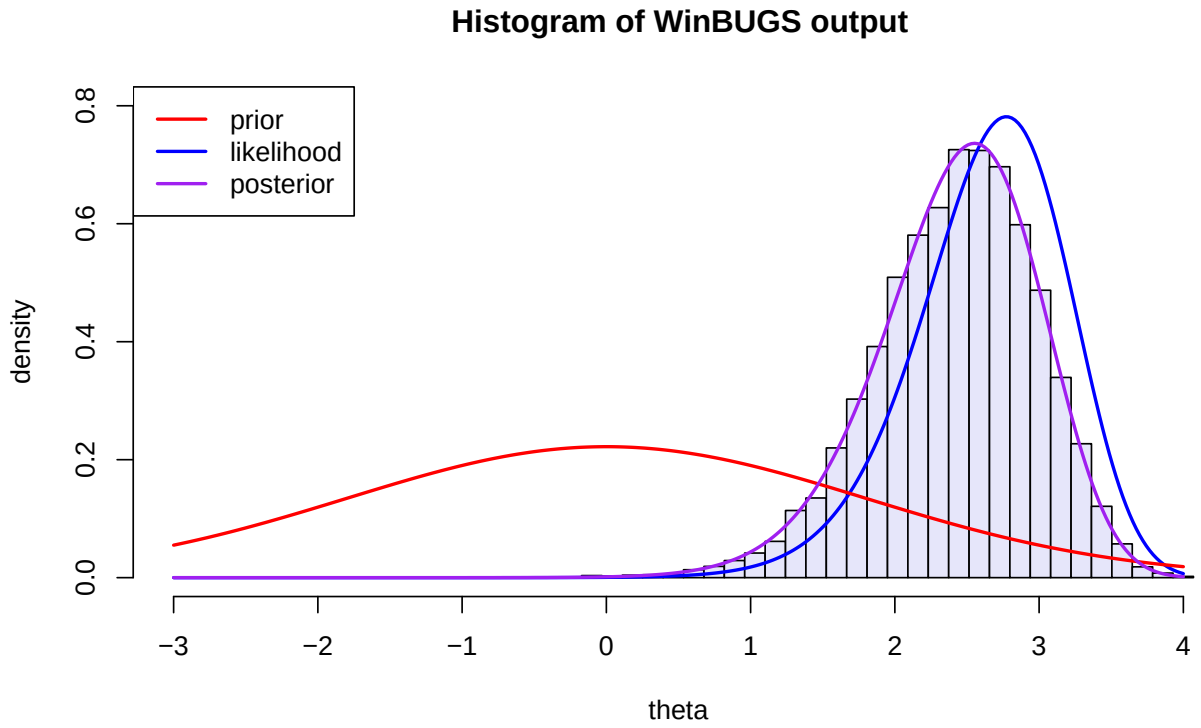
Notes on all this; (not a substitute for reading the manual!)

- 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

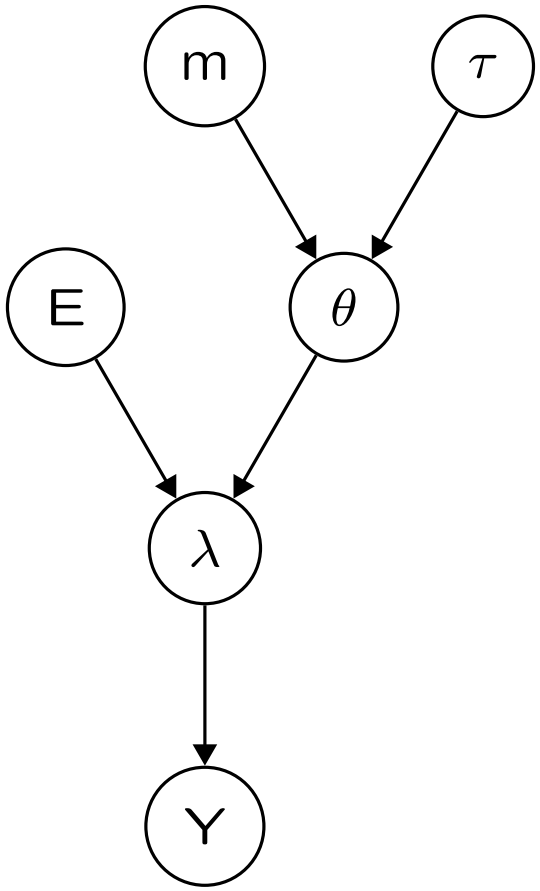
From 10,000 iterations, how do we do? (Note 'MC error' estimates Monte Carlo error in the posterior mean)



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

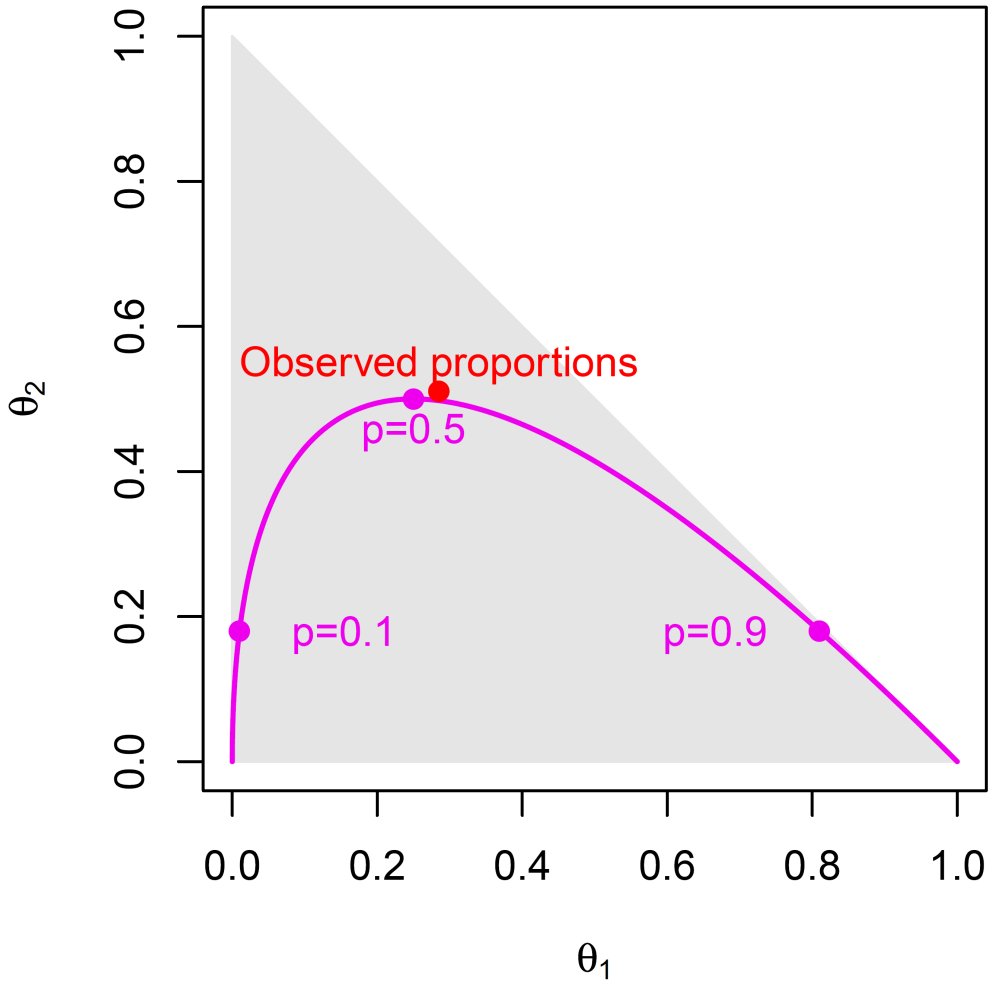
We have $n = 186$, and
$\mathbf{Y} = (53,95,38)$.

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

In WinBUGS, input these by highlighting two list objects:

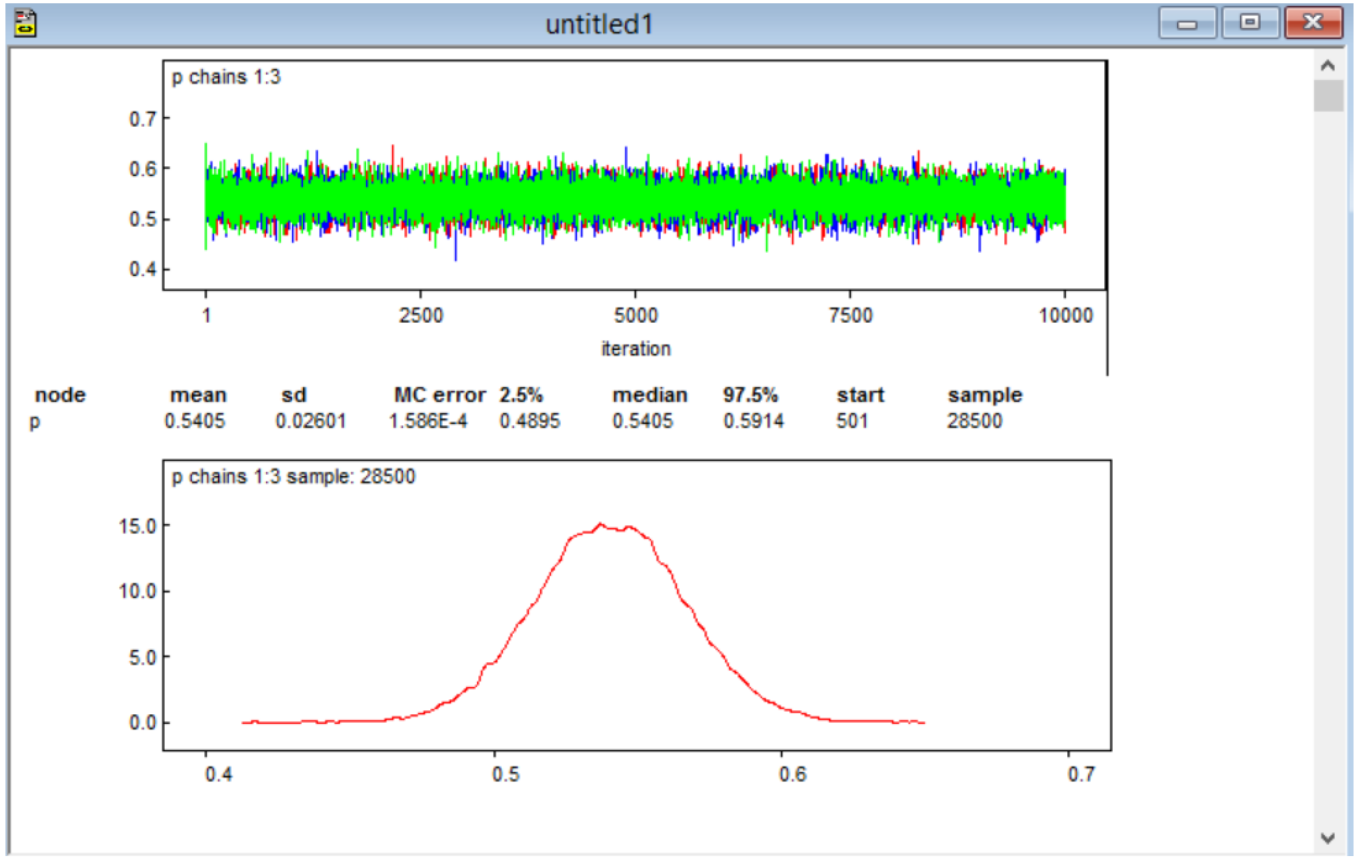
```
# Data
list(y=c(53,95,38),n=186)
```

```
# 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;

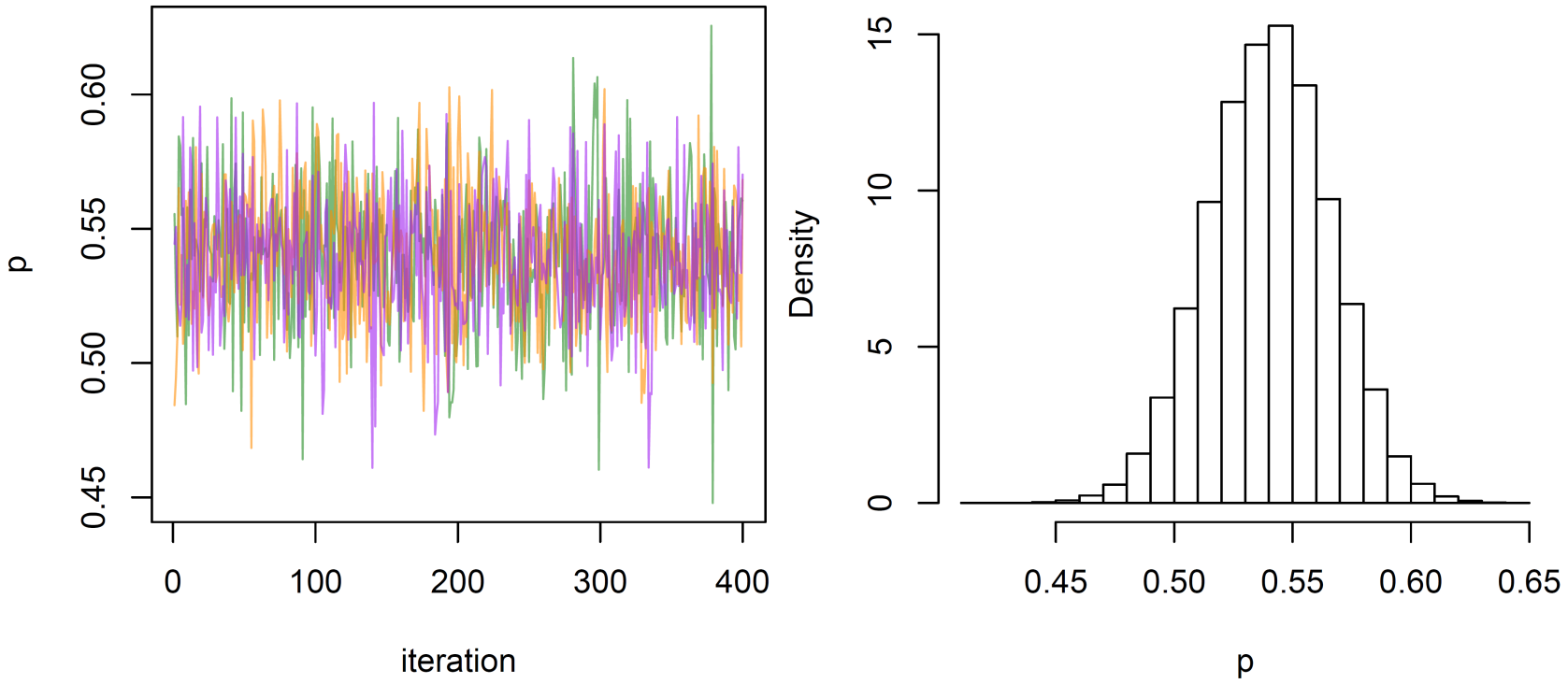
```
library("R2WinBUGS")
hweout <- bugs(data=list(y=c(53,95,38),n=186),
inits=list(p=0.5, p=0.1, p=0.9),
           parameters.to.save=c("p"),
           model.file="hweprog.txt",
           bugs.directory = "C:/Program Files/WinBUGS14",
           n.chains=3, n.iter=10000,
           n.burnin=500, n.thin=1,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



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.

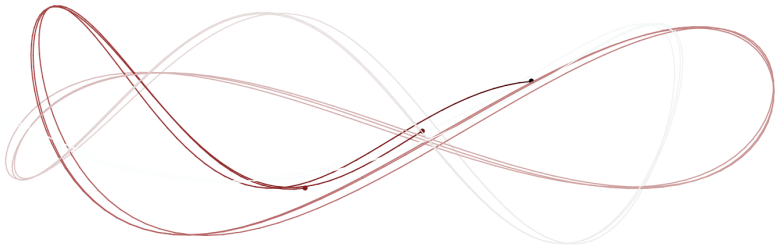
WinBUGS: less pointy-clicky

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 `rjags` uses the **exact** same model file;

```
> 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
```

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
```

INLA

Compare these to the non-Bayesian point estimate and confidence interval:

```
> 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.

Model comparison

We saw some model comparison tools in Session 6 – evaluating posterior probability for different submodels, indexed by different $\boldsymbol{z}$ values.

An alternative approach evaluates different models by how well they predict new outcomes. In Session 6 we examined the Mean Squared Error – for n 'new' observations this is

$$\frac{1}{n}\sum_i (\tilde{Y}_i - \widehat{Y}_i)^2.$$

A more general criteria is the log *posterior predictive density*, which for a single new observation $\tilde{Y}$ we write as

$$\begin{aligned} lpd &= \log p_{\text{ppost}}(\tilde{Y}) = \log p(\tilde{Y}|\mathbf{Y}) \\ &= \log \int p(\tilde{Y}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{Y}) d\boldsymbol{\theta} \end{aligned}$$

where $p(\boldsymbol{\theta}|\mathbf{Y})$ is the posterior distribution for parameter(s) $\boldsymbol{\theta}$.

Model comparison

Some notes on this:

- If new data $\tilde{Y}$ isn't likely under the posterior values of $\boldsymbol{\theta}$, expect to get a low 'score'
- If new data $\tilde{Y}$ is likely under the posterior, typically get a big score
- Confusingly, it's traditional to use $-2 \times lpd$ as the criteria – so **smaller** values are better

Model comparison

But we don't know the values of future observations $\tilde{Y}$, so ideally we would average lpd over the **true** sampling distribution of new observations;

$$elpd = \mathbb{E}[\log p(\tilde{Y}|\mathbf{Y}); \boldsymbol{\theta}] = \int \log p(\tilde{Y}|\mathbf{Y}) p(\tilde{Y}|\boldsymbol{\theta}) d\tilde{Y}$$

for true density $q(\tilde{Y})$ – i.e. sampling data under the true $\boldsymbol{\theta}$ values.

And for predicting a whole new dataset, we could consider the total score

$$\begin{aligned} elpd_n &= \sum_{i=1}^{n} \mathbb{E}[\log p(\tilde{Y}_i|\mathbf{Y}); \boldsymbol{\theta}] \\ &= \sum_{i=1}^{n} \int \log p(\tilde{Y}_i|\mathbf{Y}) p(\tilde{Y}_i|\boldsymbol{\theta}) d\tilde{Y}_i. \end{aligned}$$

– adding up the score for each observation.

Unfortunately these quantities depend on the true $\boldsymbol{\theta}$, which is unknown.

Model comparison

The first widely-used approximation to $elpd_n$ 'plugs-in' a simple non-Bayesian maximum likelihood point estimate $\hat{\boldsymbol{\theta}}_{MLE}$ for $\boldsymbol{\theta}$, and assumes the future data $\tilde{Y}_i$ can be approximated by current data Y_i.

This double use of the data (in both $\hat{\boldsymbol{\theta}}_{MLE}$ and $\tilde{Y}_i \approx Y_i$) results in overfitting, but this can be easily corrected. We approximate $elpd_n$ by

$$\widehat{elpd}_{AIC} = \left(\sum_{i=1}^{n} p(Y_i|\hat{\boldsymbol{\theta}}) \right) - k$$

and (confusingly!) we report

$$AIC = -2 \times \widehat{elpd}_{AIC} = -2 \left(\sum_{i=1}^{n} p(Y_i|\hat{\boldsymbol{\theta}}) \right) + 2k,$$

where k is the number of parameters. AIC, developed by Akaike (1973), is *An Information Criterion*.

Model comparison

In Bayesian and/or hierarchical models, there is no well-defined count of parameters – because we typically learn about one as we learn about all the others. The widely-used *Deviance Information Criterion* instead uses a Bayesian posterior mean $\mathbb{E}[\boldsymbol{\theta}|\mathbf{Y}]$ for its estimate $\widehat{\boldsymbol{\theta}}_{\text{Bayes}}$, giving

$$\begin{aligned} \widehat{elpd}_{DIC} &= \left(\sum_{i=1}^{n} \log p(Y_i|\widehat{\boldsymbol{\theta}}_{\text{Bayes}}) \right) - pD, \\ \text{where } pD &= 2\left(\log p(\mathbf{Y}|\widehat{\boldsymbol{\theta}}_{\text{Bayes}}) - \mathbb{E}_{\boldsymbol{\theta}|\mathbf{Y}}[\log p(\mathbf{Y}|\boldsymbol{\theta})] \right) \end{aligned}$$

is another bias-correction term, accounting for double use of data in the 'score' $\log p(\mathbf{Y}|\boldsymbol{\theta})$ and also in the posterior $p(\boldsymbol{\theta}|\mathbf{Y})$.

In the usual confusing manner, we actually use

$$DIC = -2 \left(\sum_{i=1}^{n} \log p(Y_i|\widehat{\boldsymbol{\theta}}_{\text{Bayes}}) \right) + 2pD.$$

Model comparison

DIC – easily calculated using the WinBUGS software and similar packages – is popular in practice. Some examples from genetics;

- Shriner and Yi 2009 use DIC in the context of multiple QTL Mapping – to select how many QTLs there are, and their locations
- Yu et al, 2012 use DIC studying gene×environment interactions, with a model that 'clusters' nearby* variants, so they have similar interaction effects. DIC is used to choose how many clusters

* ...using the Potts model

Model comparison

But DIC has been heavily criticized by statisticians;

- p_D is not invariant to parameterization
- DIC is not consistent for choosing the correct model
- DIC has an *ad hoc* theoretical justification and can't be used for all models
- DIC has been shown to under-penalize complex models (Plummer (2008), Ando (2007)
- See Spiegelhalter et al (2014) for a recent review

Model comparison

A currently-preferred alternative is the Widely-Applicable Information Criterion (WAIC) which is identical to DIC by uses bias-correction term

$$p_{WAIC} = 2\sum_{i=1}^{n} \log(\mathbb{E}_{\boldsymbol{\theta}|\mathbf{Y}}[\, p(Y_i|\boldsymbol{\theta})\,]) - \mathbb{E}_{\boldsymbol{\theta}|\mathbf{Y}}[\, \log p(Y_i|\boldsymbol{\theta})\,]$$

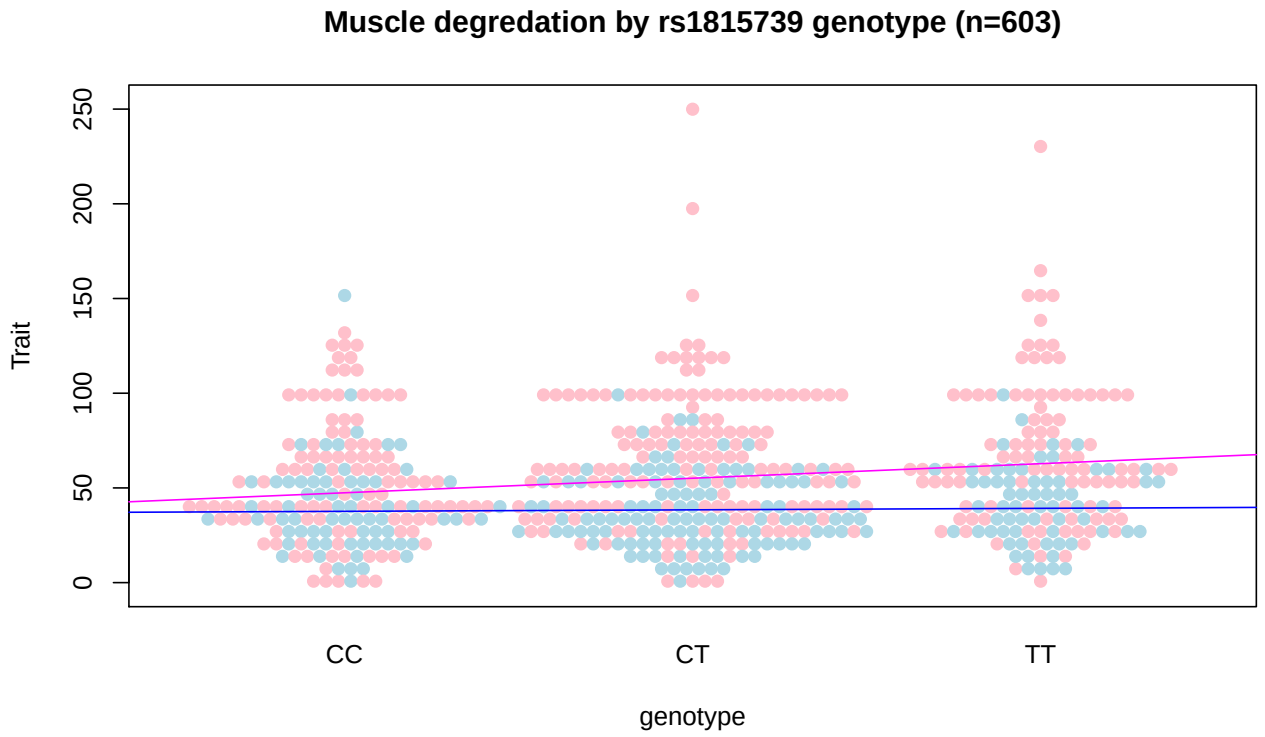
... and then reports

$$DIC = -2\left(\sum_{i=1}^{n} \log p(Y_i|\hat{\boldsymbol{\theta}}_{\text{Bayes}})\right) + 2p_{WAIC}.$$

- This has better theoretical properties, but can be unstable in small samples
- If you find all the log-densities confusing, use of these criteria is similar in practice to using leave-one-out prediction measures, averaged over each observation – but they can be computed much, much more quickly

Model comparison

To illustrate these criteria, we use a linear regression example – similar to the FTO one, but with a larger dataset;



Model comparison

Using `R2WinBUGS` for a model with a genotype×sex interaction – and calculating its DIC;

```
library("R2WinBUGS")
# full model
cat(file="linregprog1.txt", "model{
for(i in 1:n){
   Trait[i] ~ dnorm(mu[i], tau)
   mu[i] <- beta[1] + beta[2]*Geno[i] + beta[3]*Male[i] + beta[4]*Geno[i]*Male[i]
}
tau <- 1/(sigma*sigma)
sigma <- abs(Z)/sqrt(chi2)
Z ~ dnorm(0,1)
chi2 ~ dchisqr(1) # sneaky way to get a cauchy prior
for(j in 1:4){
   beta[j] ~ dnorm(0,0.001) }
}")
```

Model comparison

```
linregout1 <- bugs(data=fms.list, inits=NULL,
                parameters.to.save=c("beta"),
                   model.file="linregprog1.txt",
                   bugs.directory = "C:/Program Files/WinBUGS14",
                   n.chains=3, n.iter=10000,
                   n.burnin=500, n.thin=1, DIC=TRUE, debug=FALSE)
```

Model comparison

We also use WinBUGS to fits a reduced model – `linregout2` with no interaction but everything else the same. Comparing it to classical methods and AIC;

```
> lm1 <- lm(Trait~Geno+Male+Geno*Male, data=fms.clean)
> lm2 <- lm(Trait~Geno+Male, data=fms.clean)

> -2*c(logLik(lm1), logLik(lm2))
[1] 5841.214 5845.255
> c(AIC(lm1),       AIC(lm2))
[1] 5851.214 5853.255         # lower is better (i.e. better predictions)

> c(linregout1$DIC, linregout2$DIC)
[1] 5851.174 5853.489
> c(linregout1$pD, linregout2$pD)
[1] 5.014797 4.189147
```

- AIC and DIC values are extremely close (expected with non-hierarchical model and large n)
- DIC's pD approximates the number of parameters very accurately

Model comparison

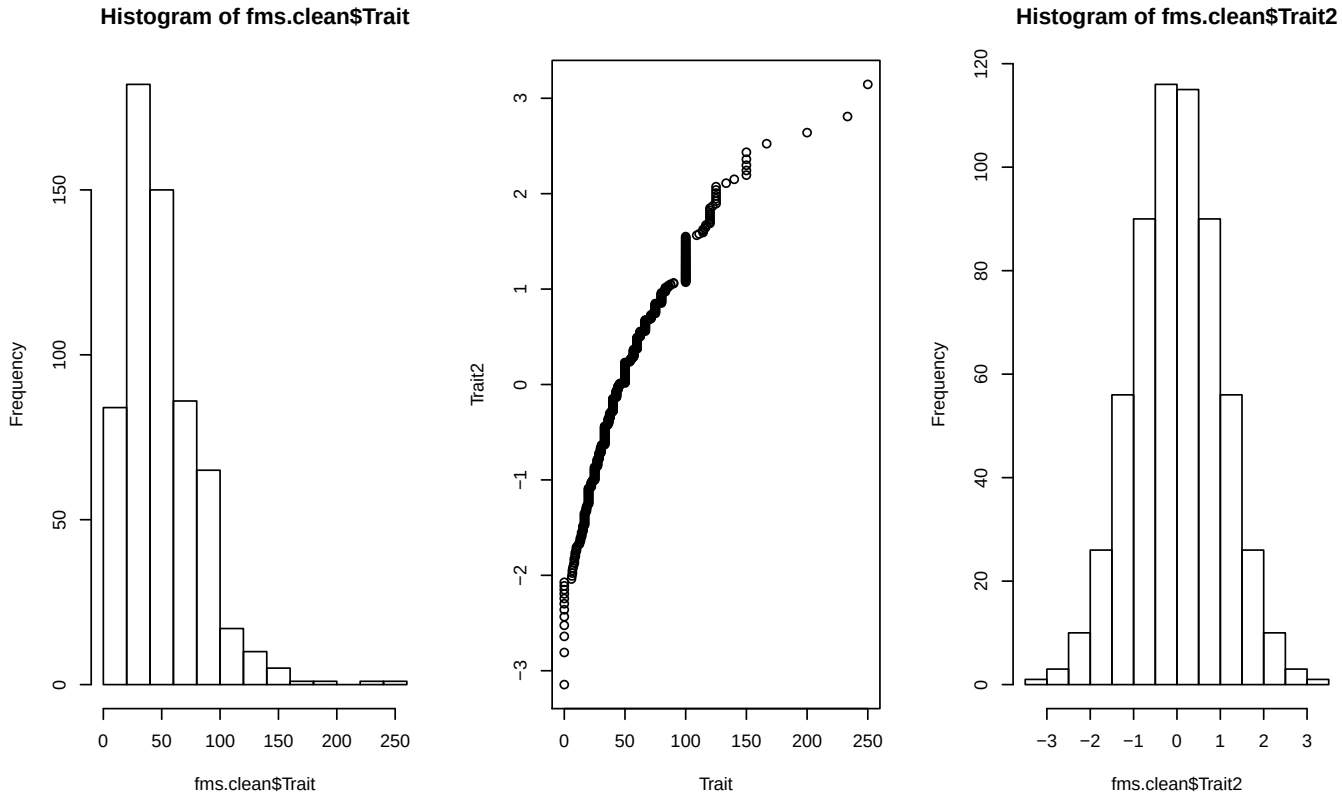
Using INLA, this is even simpler – and we can also obtain WAIC;

```
> library("INLA")
> inla1 <- inla(Trait~Geno+Male+Geno*Male, data=fms.clean, control.compute = list(waic=TRUE, dic=TRUE
> inla2 <- inla(Trait~Geno+Male, data=fms.clean, control.compute = list(waic=TRUE, dic=TRUE))

> c(inla1$waic$waic, inla2$waic$waic)
[1] 5853.014 5855.292
> c(inla1$dic$dic,   inla2$dic$dic  )
[1] 5851.213 5853.323

> c(inla1$waic$p.eff, inla2$waic$p.eff) # not so great
[1] 6.654282 5.860704
> c(inla1$dic$p.eff,  inla2$dic$p.eff ) # unreasonably good!
[1] 4.995477 4.031116
```

- INLA's (sensible) default priors are used here
- Part of the issue with WAIC's effective number of parameters is that the model – which assumed Normality – doesn't fit the spread of the data well

Model comparison

Just to illustrate this property, we transform the data to look more Normal;



Model comparison

Re-running all the analyses with the transformed, more-Normal trait;

```
> inla1b <- inla(Trait2~Geno+Male+Geno*Male, data=fms.clean, control.compute = list(waic=TRUE, dic=TR
> inla2b <- inla(Trait2~Geno+Male, data=fms.clean, control.compute = list(waic=TRUE, dic=TRUE))

> c(inla1b$waic$waic, inla2b$waic$waic)
[1] 1637.344 1638.347
> c(inla1b$dic$dic,   inla2b$dic$dic  )
[1] 1637.252 1638.192

> c(inla1b$waic$p.eff, inla2b$waic$p.eff) # much better
[1] 5.082174 4.162125
> c(inla1b$dic$p.eff,  inla2b$dic$p.eff ) # still unreasonably good!
[1] 5.044638 4.042928
```

Conclusions

- Stan, INLA and others allow almost any model to be fit, given some coding based on a description of the model and data, and some care using MCMC

- No universally agreed upon approach to carrying out model comparison. (Also true in non-Bayesian work)

- The Widely Applicable Information Criteria (WAIC) is growing in popularity, and both DIC and WAIC are available off-the-shelf, for many frequently-used models.