Astrostats 2013 Lecture 1
Bayesian parameter estimation and model comparison

C.A.L. Bailer-Jones
Max Planck Institute for Astronomy, Heidelberg
http://www.mpia.de/~calj/
Last updated: 2013-06-18 08:11

Contents

1 Key points

- The fundamental principle of Bayesian analysis of data is to treat data and parameters probabilistically. Our goal is to infer probability distributions over model parameters of interest, the probabilities of models and/or probability distributions over predicted data.
- Bayesian parameter estimation is a universal approach to fitting models to data. We define a generative model for the data, a likelihood function, and a prior distribution over the parameters. The posterior rarely has closed form, so we characterize it by sampling, from which we can find the best model parameters and their uncertainties.
- Model comparison comparison compares models as a whole, not at specific value of the parameters. This is achieved by some kind of averaging over the model parameters, for example with the evidence (likelihood averaged over the parameter prior PDF), or the K-fold cross validation likelihood (likelihood of a partition of the data averaged over the posterior, averaged over all partitions of the data). Both of these methods balance the power of the model to fit the data with the complexity of the model.
- Efficient sampling of multidimensional distributions is key to applying Bayesian inference in practice. This may be done with Monte Carlo methods.

2 Recap on Bayesian parameter estimation

Bayesian parameter estimation involves inferring the posterior probability density function (PDF) over model parameters (θ) given the data (D) for some model (M). The posterior PDF is given by Bayes' theorem

$$P(\theta|D,M) = \frac{P(D|\theta,M)P(\theta|M)}{P(D|M)} \tag{1}$$

where the terms in the numerator are called the *likelihood*, $P(D|\theta,M)$, and the *prior*, $P(\theta|M)$, and the denominator is the *evidence*. As the evidence does not depend on the model parameters, we can consider it as a normalization constant for the posterior. In that case the unnormalized posterior is just the product of the likelihood and prior. For a uniform prior the posterior is proportional to the likelihood. But where as the likelihood is a (normalized) function of the data for fixed parameters, the posterior is considered a function of the parameters for fixed data. This is an important difference.

3 Is this coin fair?

We are given a coin and we toss it n times. It lands heads in r of them. Is the coin fair?

This question has no definitive answer. We can only answer it probabilistically. We therefore interpret it to mean "what is the posterior PDF over p?", i.e. "what is $P(p|D,M)$, where $D = (n$ heads, $n-r$ tails)?" Equation 1 tells us that this is the product of the likelihood and prior, divided by a normalization constant.

Let's first deal with the likelihood. We set up a model M, which specifies that the coin lands heads in a single toss with parameter p, and that all tosses are independent. This is the only parameter of the model. The probability of getting r heads in n tosses is the binomial distribution in r

binomial distribution

$$P(D|p,M) = {}_nC_r\, p^r(1-p)^{n-r} \tag{2}$$

where $r < n$ and ${}_nC_r$ is independent of p.[1]

We'll now adopt two different forms of the prior, and then derive and plot the posterior, and also work out the expectation value of p from this.

3.1 Uniform prior

Let us first adopt a uniform prior

$$P(p|M) = \begin{cases} 1 & \text{if } \ 0 \le p \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

[1]

$${}_nC_r = \binom{n}{r} = \frac{n!}{r!(n-r)!} \qquad r \le n$$

"n choose r", is the number of ways of selecting r things from n without replacement.

In that case equation the posterior PDF is

$$P(p|D,M) = \frac{1}{Z} p^r (1-p)^{n-r} \tag{4}$$

where Z is a normalization constant. Imagine we had $n = 20$, $r = 7$. We can plot the posterior PDF using

```
n <- 20
r <- 7
p <- seq(from=0,to=1,length.out=201)
plot(p, dbinom(x=r, size=n, prob=p), type="l")
```

Note that the PDF peaks well away from $p = 1/2$, although a fair coin is not strongly disfavoured. Remember that this curve is *not* binomial in p. (It is binomial in r.) Furthermore, the posterior as plotted – as a function of p – is *unnormalized*, i.e. does not integrate over p to unity. As normalization would just rescale the whole curve by a scalar multiple, it does not change the peak (the mode) or relative probabilities of solutions. But normalization is necessary if you want to calculate expectation values, e.g. the mean or variance. If we denote the unnormalized posterior PDF as $P^u(p|D,M)$ – which in this case is $p^r(1-p)^{n-r}$ – then the mean of p is

$$E[p] = \int_p p\, P(p|D,M) dp = \frac{1}{Z} \int_p p\, P^u(p|D,M) dp \tag{5}$$

where $Z = \int_p P^u(p|D,M)dp$. We can approximate both of these integrals (the one above, and Z) with a sum. Here I use a simple, but adequate, approximation

$$\int_0^1 f(p)dp \simeq \delta p \sum_i f(p_i) \tag{6}$$

for some fixed (small) step size δp. The step size appears in both the numerator and denominator of equation 5 and so cancels, i.e.

$$E[p] \simeq \frac{\sum_i p P^u(p|D,M)}{\sum_i P^u(p|D,M)} \,. \tag{7}$$

Implementing this in R and plotting

```
pdense <- dbinom(x=r, size=n, prob=p) # unnormalized in p
p.mean <- sum(p*pdense)/sum(pdense)
abline(v=p.mean, col="red")
```

It is instructive to repeat this example for $n = 20$ for a range of values of r.

```
n <- 20
Nsamp <- 201 # no. of points to sample at
p <- seq(from=0,to=1,length.out=Nsamp)
deltap <- 1/(Nsamp-1) # step size between samples of p
par(mfrow=c(3,4), mgp=c(1.8,0.6,0), mar=c(3.5,3.5,1,1), oma=c(1,1,1,1))
for(r in seq(from=0,to=20,by=2)) {
  pdense <- dbinom(x=r, size=n, prob=p)
  pdense <- pdense/(deltap*sum(pdense)) # normalize posterior
  plot(p, pdense, type="l", xlab="p", ylab="P(p|D,M)")
  title(main=paste("r=",r), line=0.3, cex.main=1.0)
  p.mean <- sum(p*pdense)/sum(pdense)
  abline(v=p.mean, col="red")
}
```

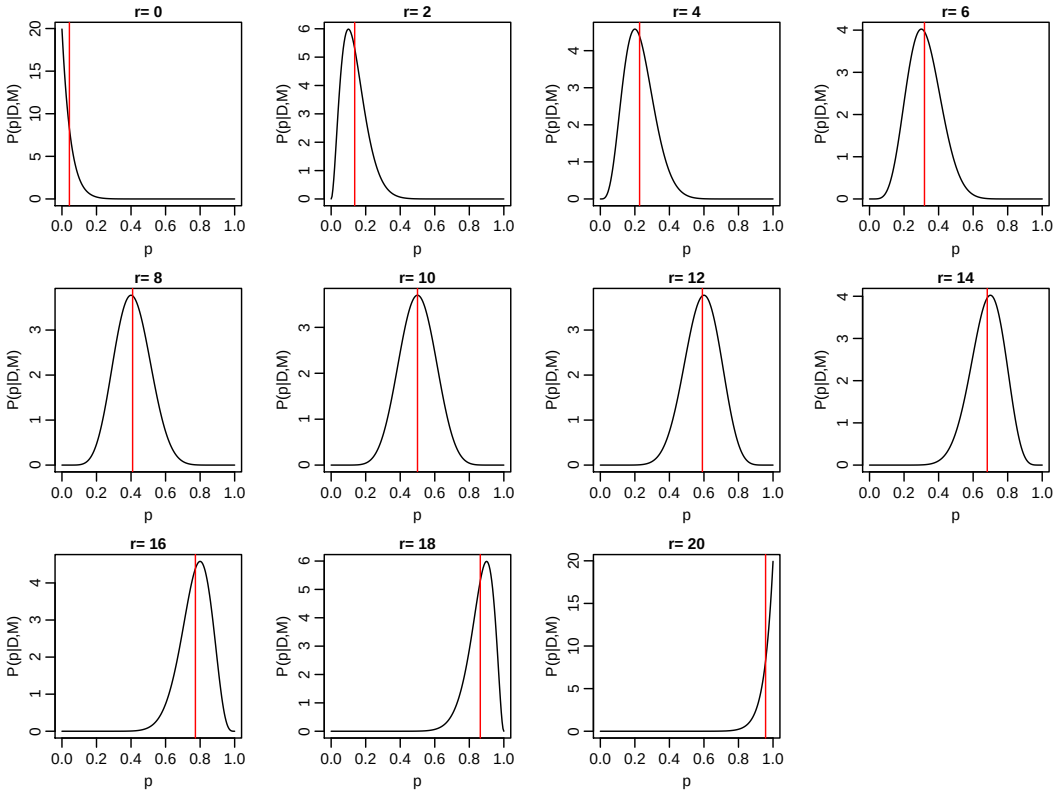

Figure 1: The posterior PDF for the probability, p, of a coin giving heads on a single toss, when r of n observed tosses are heads. A uniform prior over p is used. The red vertical line is the mean.

The plot is shown in Figure 1. Here I have normalized the posterior, `pdense`, in the usual way, i.e. to make the area under the curve unity. This ensures that `pdense` is a true density. Note that `sum(pdense)` is then equal to `Nsamp-1` and not to unity. Therefore we still need to divide by `sum(pdense)` to calculate the mean (or any other expectation values) in order that δp cancels. (If we omitted the term `deltap` in the above then `pdense` would sum to one, but it would no longer be a true density.)

3.2 Beta prior

In practice you are very unlikely to have a personal prior for p which is uniform. For example, you may believe that the coin is probably fair, or close to fair. An appropriate density function for parameters constrained to lie in the range 0–1 is the beta distribution. This is described by two parameters, α and β. Its PDF is

$$P(p) = \frac{1}{B(\alpha,\beta)} p^{\alpha-1}(1-p)^{\beta-1} \quad \text{where} \quad \alpha > 0,\ \beta > 0,\ 0 \le p \le 1 \tag{8}$$

and $B(\alpha,\beta) = \int p^{\alpha-1}(1-p)^{\beta-1}dp$ is the normalization constant (the Beta function). The mean is $\alpha/(\alpha+\beta)$ and the mode (for $\alpha > 1, \beta > 1$) is $(\alpha-1)/(\alpha+\beta-2)$. If $\alpha = \beta$ then the

beta distribution

function is symmetric, the mean and mode are 0.5, and the larger the value of α the narrower the distribution.

Let's now redo the coin analysis with a beta prior with $\alpha = \beta = 10$

```
p <- seq(from=0, to=1, length.out=201)
alpha.prior <- 10
beta.prior <- 10
plot(p, dbeta(p, shape1=alpha.prior, shape2=beta.prior), type="l")
```

In R, `shape1` is α and `shape2` is β.

The (unnormalized) posterior is the product of the prior and likelihood. The likelihood function is unchanged. It turns out that with a binomial likelihood function, the beta distribution is the conjugate prior, i.e. the prior and posterior have the same functional form (beta distribution). The parameters of the posterior PDF are

conjugate prior

$$\begin{aligned} \alpha_{\rm posterior} &= \alpha_{\rm prior} + r \\ \beta_{\rm posterior} &= \beta_{\rm prior} + n - r \ . \end{aligned} \tag{9}$$

We now repeat the above plots for a range of values of r, with this beta prior

```
n <- 20
alpha.prior <- 10
beta.prior  <- 10
Nsamp <- 201 # no. of points to sample at
p <- seq(from=0,to=1,length.out=Nsamp)
deltap <- 1/(Nsamp-1) # step size between samples of p
par(mfrow=c(3,4), mgp=c(1.8,0.6,0), mar=c(3.5,3.5,1,1), oma=c(1,1,1,1))
for(r in seq(from=0,to=20,by=2)) {
  pdense <- dbeta(x=p, shape1=alpha.prior+r, shape2=beta.prior+n-r)
  plot(p, pdense, type="l", xlab="p", ylab="P(p|D,M)")
  title(main=paste("r=",r), line=0.3, cex.main=1.0)
  p.mean <- sum(p*pdense)/sum(pdense)
  abline(v=p.mean, col="red")
  # Verify that this is the same as the direct calculation
  pdense2 <- dbinom(x=r, size=n, prob=p) * dbeta(x=p,shape1=alpha.prior,shape2=beta.prior)
  pdense2 <- pdense2/(deltap*sum(pdense2)) # normalize posterior
  lines(p, pdense2, col="cyan", lty=2)
}
```

The plot is shown in Figure 2. Note that we no longer need to normalize `pdense`, because it's a beta function in p calculated by the R function `dbeta`, which is a proper (i.e. normalized) density function. But `pdense2` is not normalized, so we must do that ourselves.

It is instructive to plot the likelihood, prior and posterior together:

```
n <- 20
alpha.prior <- 10
beta.prior  <- 10
Nsamp <- 201 # no. of points to sample at
p <- seq(from=0,to=1,length.out=Nsamp)
deltap <- 1/(Nsamp-1) # step size between samples of p
prior <- dbeta(x=p, shape1=alpha.prior, shape2=beta.prior)
par(mfrow=c(3,4), mgp=c(1.8,0.6,0), mar=c(3.5,3.5,1,1), oma=c(1,1,1,1))
for(r in seq(from=0,to=20,by=2)) {
  like  <- dbinom(x=r, size=n, prob=p)
```

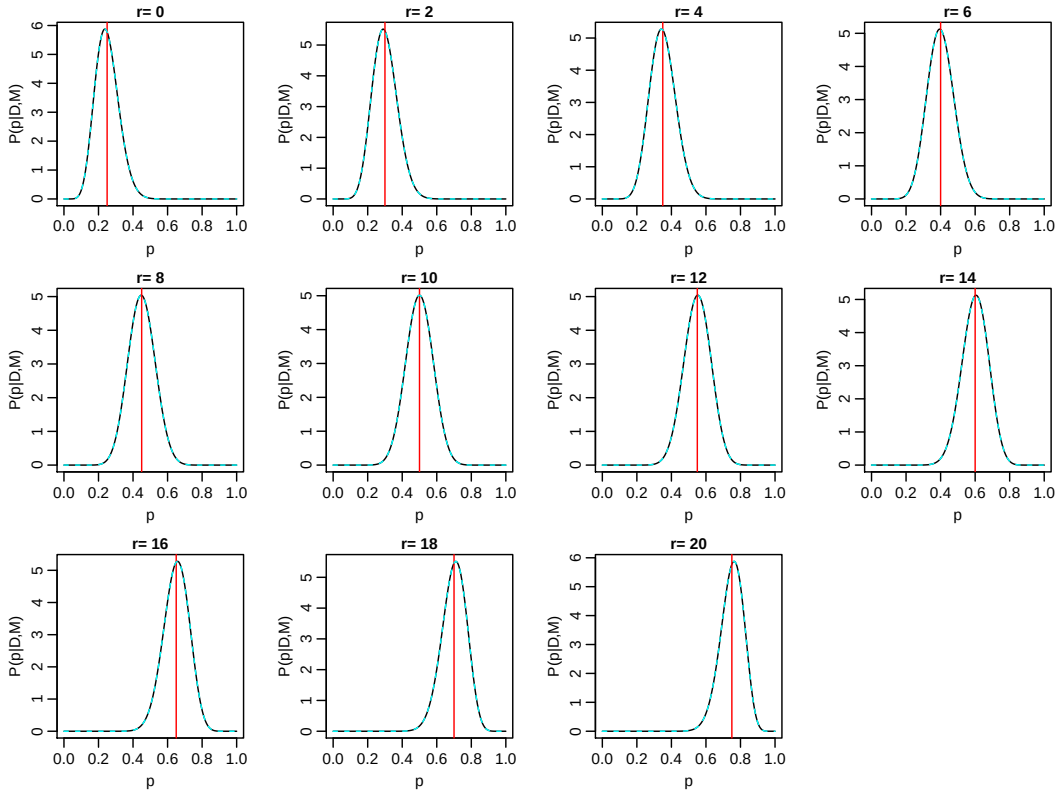

Figure 2: As Figure 1, but now using a beta prior on p with $\alpha = \beta = 10$. Both the black line and cyan line show the posterior, the black line using the beta distribution (i.e. taking advantage of the fact that we have a conjugate prior), the cyan using a direct calculation. They are of course identical.

```
  like  <- like/(deltap*sum(like)) # for plotting convenience only: see below
  post  <- dbeta(x=p, shape1=alpha.prior+r, shape2=beta.prior+n-r)
  plot(p, post, type="l", xlab="p", ylab="PDF", ylim=c(0,6))
  # to get full y-scale use: ylim=c(0,max(prior,post,like))
  lines(p, like, col="red")
  lines(p, prior, col="blue")
  title(main=paste("r=",r), line=0.3, cex.main=1.0)
}
```

The plot is shown in Figure 3. We can see how the (fixed) prior combines with the likelihood to form the posterior. Note that I have plotted with the same y-axis scale in all panels: the likelihood extends higher in the first and last panels. It doesn't really make sense to normalize the likelihood, as it is not a PDF over p, but rather a PDF over the data D given p. It is done here just for plotting convenience (so the area under all curves is unity). In each panel, the posterior (black) is the product of the likelihood (red) and the prior (blue), and then renormalized (which is what Bayes' theorem tells us to do).

As we get more and more data, the prior stays the same, but the likelihood becomes more peaked, and so the posterior is influenced more by the likelihood than by the prior. Let's start with the same prior as above, a beta distribution with $\alpha = \beta = 10$. Imagine we have $(r, n) = (2, 3)$, i.e. two heads and one tails. Let's increase the amount of data in steps of factors

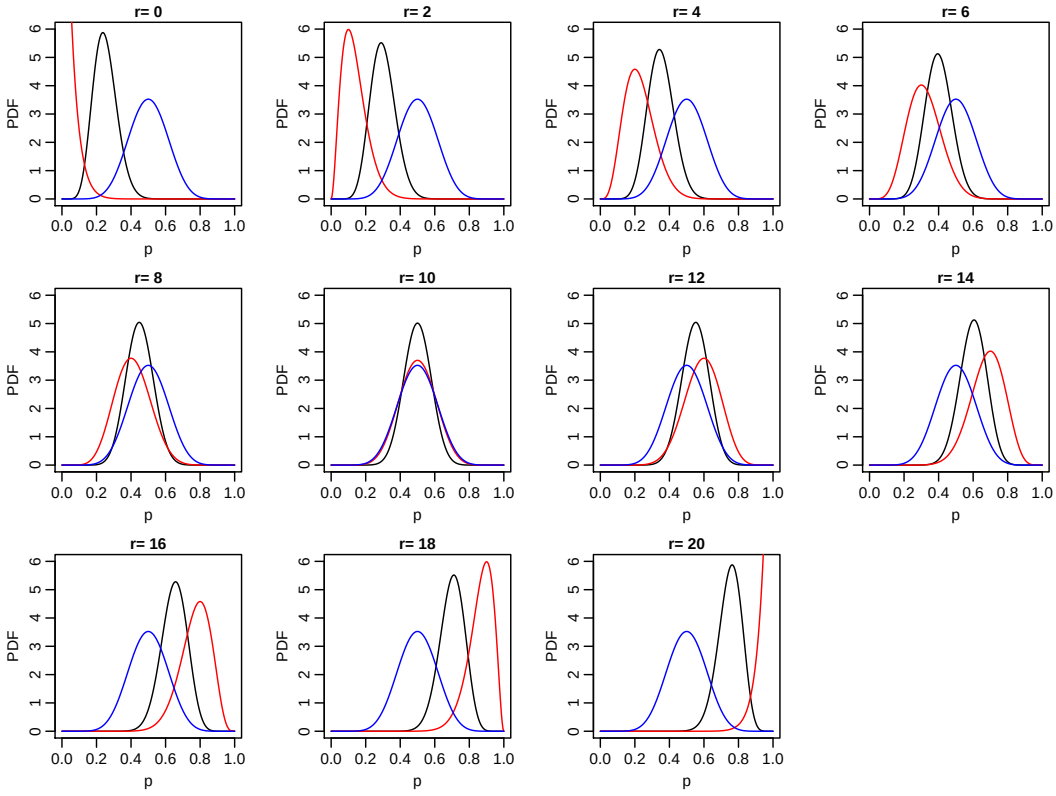

Figure 3: As Figure 2, but now showing in addition to the posterior PDF (in black), the likelihood (red) and prior (blue).

of two, keeping the proportion of heads and tails the same, i.e. $(r, n) = (2, 3), (4, 6), (8, 12), \ldots$. It is interesting to see how the likelihood and posterior evolve as we get more data.

```
alpha.prior <- 10
beta.prior  <- 10
Nsamp <- 201 # no. of points to sample at
par(mfrow=c(3,3), mgp=c(1.5,0.6,0), mar=c(3.5,3.5,1,1), oma=c(2,2,2,2))
p <- seq(from=0,to=1,length.out=Nsamp)
deltap <- 1/(Nsamp-1) # step size between samples of p
prior <- dbeta(x=p, shape1=alpha.prior, shape2=beta.prior)
for(i in 1:9) {
  r <- 2^i
  n <- (3/2)*r
  like  <- dbinom(x=r, size=n, prob=p)
  like  <- like/(deltap*sum(like)) # for plotting convenience only
  post  <- dbeta(x=p, shape1=alpha.prior+r, shape2=beta.prior+n-r)
  plot(p, post, type="l", xlab="p", ylab="PDF", ylim=c(0,max(prior,post,like)))
  lines(p, like, col="red")
  lines(p, prior, col="blue")
  title(main=paste("r/n=",r,"/",n), line=0.3, cex.main=1.0)
  readline("Pause. Press <Enter> to continue...")
}
```

The prior (blue) is the same in all panels (the y-axis scale scale is changing). Initially we

have little data, the likelihood (red) is broad, and the posterior (black) is dominated by the prior. As the amount of data increases, the likelihood starts to dominate the posterior: the posterior shifts from the prior towards the likelihood. By the time we have a lot of data, the prior is basically irrelevant. This is all a consequence of the posterior being the product of the prior and likelihood (then renormalized).[2]

If we compare equation 2 with equation 8, then we see that, when considered as a function of p, the beta function is just the binomial function in which we have had $\alpha - 1$ successes (heads), and $\beta - 1$ failures (tails). As both the prior and posterior have the beta functional form, the effect of the prior is equivalent to adding $\alpha_{\rm prior}$ successes to the actual observed number of successes, and adding $\beta_{\rm prior}$ failures to the actual observed number of failures.

Note that the uniform distribution (equation 3) is just a beta distribution with $\alpha = \beta = 1$, so the posterior PDF in the previous section (section 3.1) was actually a beta PDF as well.

Note that in none of this have we considered the order of heads and tails. If we wanted to drop the assumption of the independence of the coin tosses, then we would need to take into account this ordering. This corresponds to a more complex (but legitimate) model for the coin, in which the binomial distribution is no longer the appropriate likelihood function.

4 Why we need efficient sampling

In the coin tossing example we had just one parameter. The posterior was therefore a one-dimensional PDF and it was straight forward to determine it (e.g. for plotting) simply by evaluating it on a dense, uniform grid of values of θ. (It needs to be dense enough to capture the variation of the PDF.) We further saw that if we adopted a conjugate prior then the posterior had the same functional form as the prior, and so it was easy to plot using a standard function.

For more general problems, however, the situation is more complicated. First, we usually do not have a conjugate prior. Second, we will typically have a larger number of parameters, so the posterior PDF is defined over a higher dimensional parameter space. This makes it much more time consuming to determine it by evaluating it as we have been, namely on a regular, but now multidimensional parameter grid. This is because of the *curse of dimensionality*. If we need N samples in order to sample one parameter with sufficient density, then for two parameters we will need N^2 samples. For J-dimensions, a rectangular grid will require N^J samples. With $N = 1000$ and $J = 5$ (quite modest; we not infrequently have models with many more parameters), we already need 10^{15} samples, an infeasible number of calculations of the likelihood (which is normally more "expensive" to compute than the prior). Moreover, the likelihood and/or prior (and therefore the posterior) is likely to be vanishingly small at the overwhelming majority of these points, thereby wasting most of the computation time.

curse of dimensionality

Evaluating on a regular grid is therefore far too inefficient to be used in practice. To overcome this, it would be desirable to *sample* the posterior PDF directly. That is, we would like to be able to draw samples from an arbitrary PDF – let's call it $g(\theta)$ – in such a way that the frequency distribution of the samples is equal to $g(\theta)$ in the limit of a very large number of

[2] Of course, if the prior is zero at any point, then no matter how much data we collect, the posterior will always be zero there too. Be careful about setting the prior to exactly 0 or 1: no amount of evidence will ever convince you to change!

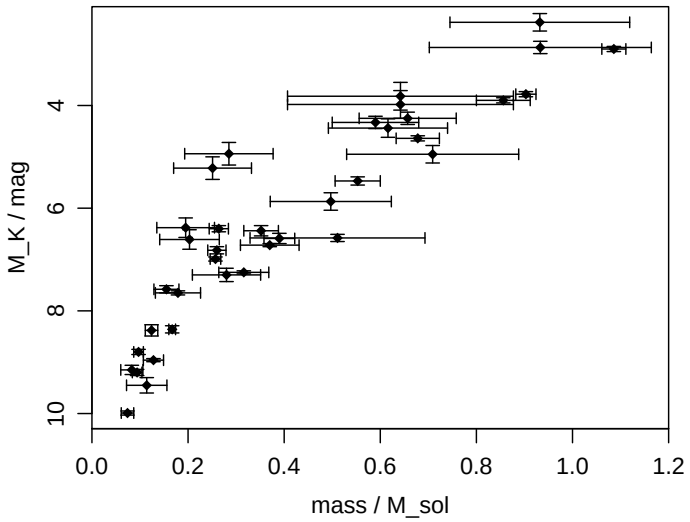

Figure 4: Mass–luminosity relation (actually, K band absolute magnitude) for cool stars from Henry et al. 1993

draws. Let's assume that we can evaluate $g(\theta)$ up to some multiplicative constant, that is, we do not require $g(\theta)$ to be normalized. How can we sample from it? This is not self-evident, because in order to sample efficiently we would need to draw preferentially from regions of high relative probability. It is not obvious how we could locate these regions without explicitly evaluating the function everywhere, which is no better than evaluating on a regular grid.

It turns out that there are efficient ways to sample an arbitrary probability distribution using *Monte Carlo* methods. A quick introduction to these is given in section C in the appendix.

The resulting samples from the posterior PDF are all we need to do parameter estimation. The examples in the following two sections demonstrate this.

5 Fitting a linear model with unknown noise

A typical task is to fit a model to two-dimensional data. An example data set is shown in Figure 4. Here we have error bars on both axes. A general method for finding the best fitting line in such cases will be covered in lecture 2 (and is described in Bailer-Jones 2012). For now we will assume that we only have uncertainties in one variable, but that the magnitude of this uncertainty is unknown. Here we will set up the model and use MCMC to estimate the model parameters.

Let $D = \{x_r, y_r\}$ denote the data with R data points. $D_r = (x_r, y_r)$ is one data point. The model, M, predicts the values of y as being

$$y = f(x) + \epsilon \quad \text{where} \tag{10}$$

$$f(x) = a_0 + a_1 x \tag{11}$$

is just the linear model with parameters a_0 (intercept) and a_1 (gradient). ϵ is a zero mean

Gaussian random variable with standard deviation σ, i.e. the residuals are modelled as $\epsilon = y - f = \mathcal{N}(0, \sigma)$. We assume that the $\{x\}$ are noise free. This tells us that the likelihood function is

$$P(D_r|\theta, M) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{[y_r - f(x_r; a_0, a_1)]^2}{2\sigma^2}\right] \tag{12}$$

and so the log (base e) likelihood for all R data points is

$$\ln P(D|\theta, M) = \sum_{r=1}^{r=R} \ln P(y_r|x_r, \theta, M) \tag{13}$$

where $\theta = (a_0, a_1, \sigma)$ are the parameters of the model.[3] In general none of these are known in advance, so we want to infer their posterior PDF from the data, $P(\theta|D, M)$. To do this we

1. define the prior PDF over the parameters. We will use plausible but convenient priors. (This requires that we make some transformation of variables, defined in the code);
2. define the covariance matrix of the proposal distribution (here we use a multivariate Gaussian);
3. define the initial parameters, as the starting point of the MCMC;
4. define the number of burn-in iterations and the number of sampling iterations.

In addition to this we generate some simulated data to work with. Once we have run the MCMC we

5. plot the chains and the 1D (projected) posterior PDFs over the parameters;
6. plot the 2D posterior distribution of (a_0, a_1), simply by plotting the samples (there are more precise ways to do this, e.g. with 2D density estimation);
7. calculate the mean and MAP values of the model parameters, and overplot these model solutions on the data;
8. calculate the predictive posterior distribution over y at a new data point (see section E.3).

This is all done in the following R script, with explanations provided as comments. It applies the method to data simulated from a linear model, so we can see how well we retrieve the true parameters. The script looks long, but a reasonable chunk of it is actually concerned with analysing the results. It makes use of functions in two external files, `linearmodel.R`, which provides the function to calculate the posterior PDF , and `monte_carlo.R` which provides the Metropolis algorithm. (The former is listed at the end of this section. The latter is generic and is listed in section C.3). These are loaded at the beginning of the following file. The just work through the rest of the script by cut and pasting blocks.

The main plots generated by this code are shown in Figures 5 and 6.

[3] Equation 13 can be written

$$\ln P(D|\theta, M) = -\frac{R}{2}\ln(2\pi) - R\ln\sigma - \frac{1}{2}\sum_r \left[\frac{y_r - f(x_r; a_0, a_1)}{\sigma}\right]^2 . \tag{14}$$

However, we don't actually need to use this, as `dnorm()` can be made to return the log likelihood, then we just sum these.

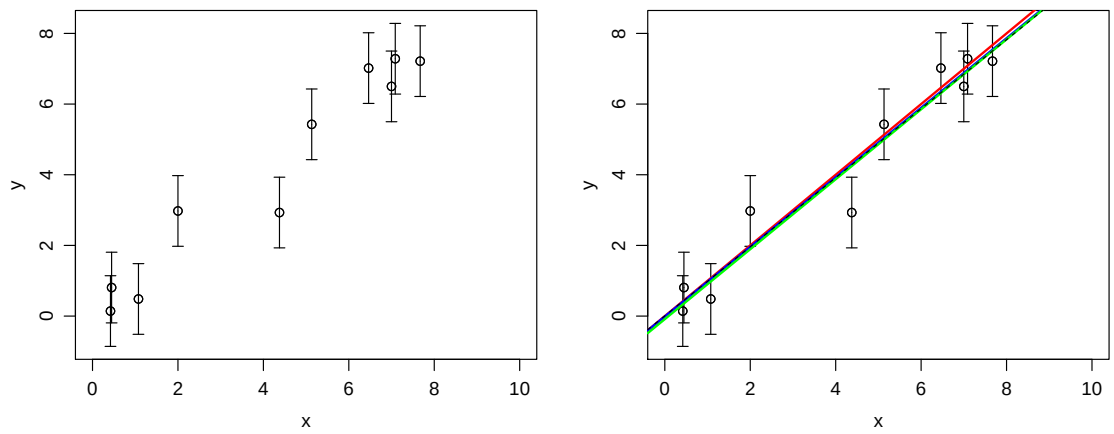

Figure 5: Linear model fitting. The red line is the true model, equation 11, with $a_0 = 0$, $a_1 = 1$, from which the black data points have been drawn and noise (with unit standard deviation) has been added. The data points and error bars are the same in both panels. The error bars show the true noise standard deviation (not used in the inference). The blue and green lines are the model fit with the parameters set to the maximum and mean of the posterior PDF respectively. The black line is the least squares (maximum likelihood) solution.

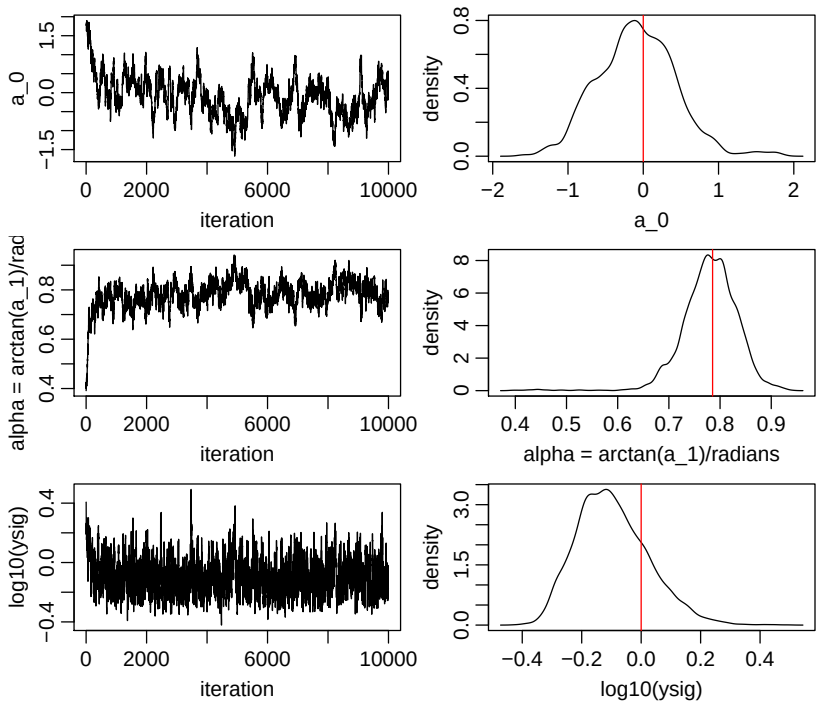

Figure 6: Example of the posterior PDF sampling of the sinusoidal model for the data shown in Fig. 5. The panels in the left column show the chains for the three parameters, those in the right the 1D PDFs generated from the samples via density estimation. The vertical red lines indicate the true parameters.

R file: `Rcode/exp_linearmodel.R`

```
# C.A.L. Bailer-Jones
# Astrostats 2013
# This file: exp_linearmodel.R
# R code to do Bayesian inference of a 3-parameter linear model to 2D data

library(gplots) # for plotCI()
source("monte_carlo.R") # provides metrop() and make.covariance.matrix()
source("linearmodel.R") # provides logpost.linearmodel()

########## Define true model and simulate experimental data from it

set.seed(50)
Ndat <- 10
x <- sort(runif(Ndat, 0, 10))
sigTrue <- 1
modMat <- c(0,1) # 1 x P vector: coefficients, a_p, of polynomial sum_{p=0} a_p*x^p
y <- cbind(1,x) %*% as.matrix(modMat) + rnorm(Ndat, 0, sigTrue)
# Dimensions in matrix multiplication: [Ndat x 1] = [Ndat x P] %*% [P x 1] + [Ndat]
# cbind does the logical thing combining a scalar and vector; then vector addition
y <- drop(y) # converts into a vector
pdf("linearmodel_data.pdf", width=5, height=4)
par(mfrow=c(1,1), mar=c(3.5,3.0,0.5,0.5), oma=c(0.5,0.5,0.5,0.5), mgp=c(2.2,0.8,0), cex=1.0)
plotCI(x, y, xlim=c(0,10), uiw=sigTrue, gap=0)
abline(a=modMat[1], b=modMat[2], col="red") # true model
dev.off()
# True parameters, transformed to be conformable with model to be used below
thetaTrue <- c(modMat[1], atan(modMat[2]), log10(sigTrue))
obsdata <- data.frame(cbind(x,y)) # only this is used in the analysis
rm(x,y)

########## Define model and infer the posterior PDF over its parameters

# Model to infer: linear regression with Gaussian noise
# Parameters: intercept a_0, gradient a_1; Gaussian noise sigma, ysig.
# Prior PDFs over model parameters:
# a_0:  intercept. P(a_0) ~ N(mean=m, sd=s)
# a_1:  gradient, a_1 = tan(alpha), alpha is angle between horizontal and model line.
#       P(alpha) ~ 1 [uniform] => P(alpha) ~ 1/(1+tan(alpha)^2) but it is easier if we
#       use alpha (in radians) as the model parameter, alpha = atan(a_1)
# ysig: Jeffreys prior P(ysig) ~ 1/ysig, or equivalently P(log10(ysig)) ~ 1.
#       This is an "improper prior", which means its integral does not converge.
# theta is 1 x J vector of all model parameters (J=3): theta = c(a_0, alpha, log10(ysig)).
# The sampling is performed on theta. It is defined in this way to make sampling from a
#   Gaussian more convenient: (1) linear steps in alpha better than in tan(alpha);
#   (2) ysig cannot be negative, and additive steps in log10(ysig) - multiplicative steps
#   in ysig - are more natural for a standard deviation.

# define covariance matrix of MCMC sampling PDF: c(a_0, alpha, log10(ysig))
sampleCov <- make.covariance.matrix(sampleSD=c(0.1, 0.02, 0.1), sampleCor=0)
# set starting point
thetaInit <- c(2, pi/8, log10(3))
# run the MCMC to find postSamp, samples of the posterior PDF
set.seed(150)
```

```
postSamp <- metrop(func=logpost.linearmodel, thetaInit=thetaInit, Nburnin=0, Nsamp=1e4,
                   verbose=1e3, sampleCov=sampleCov, obsdata=obsdata)
# 10^(postSamp[,1]+postSamp[,2]) is the unnormalized posterior at each sample

# Plot MCMC chains and use density estimation to plot 1D posterior PDFs from these.
# Note that we don't need to do any explicit marginalization to get the 1D PDFs.
parnames <- c("a_0", "alpha = arctan(a_1)/radians", "log10(ysig)")
pdf("linearmodel_mcmc.pdf", width=7, height=6)
par(mfrow=c(3,2), mar=c(3.0,3.0,0.5,0.5), oma=c(1,1,1,1), mgp=c(1.8,0.6,0), cex=1.0)
for(p in 3:5) { # columns of postSamp
  plot(1:nrow(postSamp), postSamp[,p], type="l", xlab="iteration", ylab=parnames[p-2])
  postDen <- density(postSamp[,p], n=2^10)
  plot(postDen$x, postDen$y, type="l", xlab=parnames[p-2], ylab="density")
  abline(v=thetaTrue[p-2], col="red")
}
dev.off()

# Plot gradient and intercept samples in 2D
par(mfrow=c(1,2))
plot(postSamp[,3], postSamp[,4], xlab="intercept a_0", ylab="alpha = atan(a_1) / radians", pch=".")
plot(postSamp[,3], tan(postSamp[,4]), xlab="intercept a_0", ylab="gradient a_1", pch=".")

# Find MAP solution and mean solution.
# MAP = Maximum A Posteriori, i.e. peak of posterior.
# MAP is not the peak in each 1D PDF, but the peak of the 3D PDF.
# mean is easy, because samples have been drawn from the (unnormalized) posterior.
posMAP     <- which.max(postSamp[,1]+postSamp[,2])
(thetaMAP  <- postSamp[posMAP, 3:5])
(thetaMean <- apply(postSamp[,3:5], 2, mean)) # Monte Carlo integration
# Overplot these solutions with original data and true model
pdf("linearmodel_fits.pdf", width=5, height=4)
par(mfrow=c(1,1), mar=c(3.0,3.0,0.5,0.5), oma=c(0.5,0.5,0.5,0.5), mgp=c(2.2,0.8,0), cex=1.0)
plotCI(obsdata$x, obsdata$y, xlim=c(0,10), xlab="x", ylab="y", uiw=sigTrue, gap=0)
abline(a=modMat[1],    b=modMat[2],         col="red",   lw=2) # true model
abline(a=thetaMAP[1],  b=tan(thetaMAP[2]),  col="blue",  lw=2) # MAP  model
abline(a=thetaMean[1], b=tan(thetaMean[2]), col="green", lw=2) # mean model
# Compare this with the result from ML estimation from lm()
abline(lm(obsdata$y ~ obsdata$x), col="black", lty=2)
dev.off()

########## Make prediction: determine PDF(ycand | xnew, obsdata)

# Set up uniform grid of candidate values of y, ycand[], and at each of these
# calculate the probability density (a scalar) by integrating the likelihood
# over the posterior. We do this with the Monte Carlo approximation of integration.
# Model and likelihood used here must be consistent with logpost.linearmodel() !

xnew  <- 6 # then repeat again with xnew=15
dy    <- 0.01
ymid  <- thetaMAP[1] + xnew*tan(thetaMAP[2]) # for centering choice of ycand
ycand <- seq(ymid-5, ymid+5, dy) # uniform sampling of y with step size dy
ycandPDF <- vector(mode='numeric', length=length(ycand))
# predict y at all values of parameters drawn from posterior by applying model.
# Dimensions in matrix multiplication: [Nsamp x 1] = [Nsamp x P] %*% [P x 1]
modPred <- cbind(postSamp[,3], tan(postSamp[,4])) %*% t(cbind(1,xnew))
```

```
for(k in 1:length(ycand)) {
  like <- dnorm(modPred - ycand[k], mean=0, sd=10^postSamp[,5]) # [Nsamp x 1]
  ycandPDF[k] <- mean(like) # Monte Carlo integration. Gives a scalar
}
# Note that ycandPDF[k] is normalized, i.e. sum(dy*ycandPDF) = 1
plot(ycand, ycandPDF, type="l")
# find peak and approximate confidence intervals at  1sigma
peak.ind  <- which.max(ycandPDF)
lower.ind <- max( which(cumsum(dy*ycandPDF) < pnorm(-1)) )
upper.ind <- min( which(cumsum(dy*ycandPDF) > pnorm(+1)) )
abline(v=ycand[c(peak.ind, lower.ind, upper.ind)])
# Overplot this prediction with original data and the models
par(mfrow=c(1,1))
plotCI(obsdata$x, obsdata$y, xlim=range(c(xnew,obsdata$x)), ylim=c(-1,11), uiw=sigTrue, gap=0)
abline(a=modMat[1],    b=modMat[2],         col="red",   lw=2) # true model
abline(a=thetaMAP[1],  b=tan(thetaMAP[2]),  col="blue",  lw=2) # MAP  model
abline(a=thetaMean[1], b=tan(thetaMean[2]), col="green", lw=2) # mean model
plotCI(xnew, ycand[peak.ind], li=ycand[lower.ind], ui=ycand[upper.ind],
       gap=0, add=TRUE, col="magenta")
```

R file: `Rcode/linearmodel.R`

```
# C.A.L. Bailer-Jones
# Astrostats 2013
# This file: linearmodel.R
# Functions to provide evaluations of prior, likelihood and posterior for linear model
# plus sampling from the prior

# theta is vector of parameters; obsdata is 2 column dataframe [x,y].
# the priors are hard-wired into the functions

# return log10(unnormalized posterior) of the linear model
# (see notes on the functions called)
logpost.linearmodel <- function(theta, obsdata, ind=NULL) {
  logprior <- logprior.linearmodel(theta)
  if(is.finite(logprior)) { # only evaluate model if parameters are sensible
    return( loglike.linearmodel(theta, obsdata, ind) + logprior )
  } else {
    return(-Inf)
  }
}

# return log10(likelihood) for parameters theta for rows ind in obsdata
# (do all if ind not specified)
# dnorm(..., log=TRUE) returns log base e, so multiply by 1/ln(10) = 0.4342945
# to get log base 10
loglike.linearmodel <- function(theta, obsdata, ind=NULL) {
  if(is.null(ind)) ind <- 1:nrow(obsdata)
  # convert alpha to a_1 and log10(ysig) to ysig
  theta[2] <- tan(theta[2])
  theta[3] <- 10^theta[3]
  modPred <- drop( theta[1:2] %*% t(cbind(1,obsdata[ind,]$x)) )
  # Dimensions in mixed vector matrix multiplication: [Ndat] = [P] %*% [P x Ndat]
  logLike <- (1/log(10))*sum( dnorm(modPred - obsdata[ind,]$y, mean=0,
```

```
                                    sd=theta[3], log=TRUE) ) # scalar
  return(logLike)
}

# return log10(unnormalized prior)
logprior.linearmodel <- function(theta) {
  a0Prior      <- dnorm(theta[1], mean=0, sd=2)
  alphaPrior   <- 1
  logysigPrior <- 1
  logPrior <- sum( log10(a0Prior), log10(alphaPrior), log10(logysigPrior) ) # scalar
  return(logPrior)
}

# return Nsamp samples from prior
# NOTE: This uses a proper gamma prior on ysig instead of an improper Jeffreys prior,
# which is inconsistent with logprior.linearmodel(). This is generally a BAD IDEA
# and really we should modify logprior.linearmodel() to use proper priors.
sampleprior.linearmodel <- function(Nsamp) {
  a0 <- rnorm(Nsamp, mean=0, sd=4)
  a1 <- tan(runif(Nsamp, min=-pi/2, max=pi/2))
  logysig <- rgamma(Nsamp, shape=1.5, scale=1)
  return(cbind(a0, a1, logysig))
}
```

6 Fitting a quadratic model with unknown noise

We repeat the same procedure as in the previous section, but now using a quadratic model

$$y = f(x) + \epsilon \quad \text{where} \tag{15}$$

$$f(x) = a_0 + a_1 x + a_2 x^2 \tag{16}$$

The following scripts are similar to the linear model case, but with a quadratic term. I assign a Gaussian prior to this. The main plots generated by this code are shown in Figures 7 and 8.

R file: `Rcode/exp_quadraticmodel.R`

```
# C.A.L. Bailer-Jones
# Astrostats 2013
# This file: exp_quadraticmodel.R
# R code to do Bayesian inference of a 4-parameter quadratic model to 2D data

library(gplots) # for plotCI()
source("monte_carlo.R") # provides metrop() and make.covariance.matrix()
source("quadraticmodel.R") # provides logpost.quadraticmodel()

########## Define true model and simulate experimental data from it

set.seed(70)
Ndat <- 10
xrange <- c(0,10)
x <- sort(runif(Ndat, xrange[1], xrange[2]))
sigTrue <- 1
```

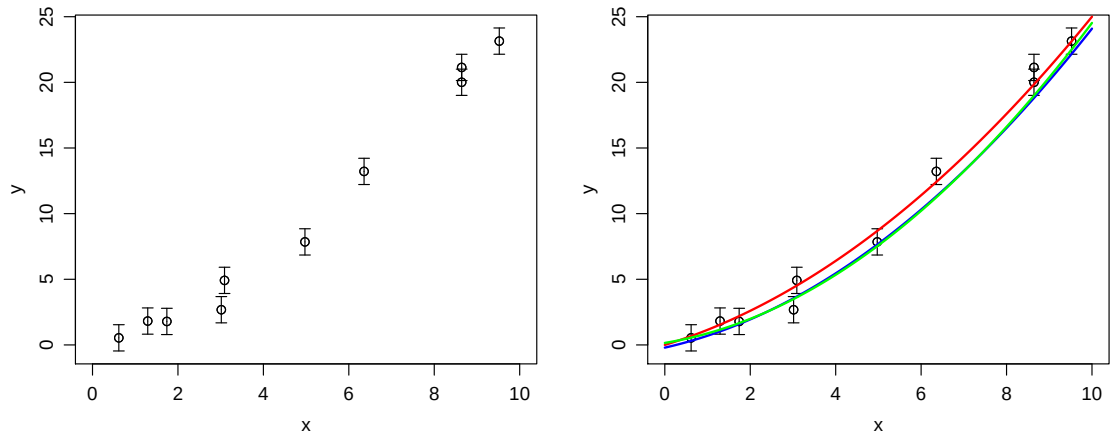

Figure 7: Quadratic model fitting. The red line is the true model, equation 16, with $a_0 = 0$, $a_1 = 1$, $a_2 = 0.15$, from which the black data points have been drawn and noise (with unit standard deviation) has been added. The data are the same in both panels. The blue and green lines are the model fit with the parameters set to the maximum and mean of the posterior PDF respectively.

```
modMat <- c(0,1,0.15) # 1 x P vector: coefficients, a_p, of polynomial sum_{p=0} a_p*x^p
y <- cbind(1,x,x^2) %*% as.matrix(modMat) + rnorm(Ndat, 0, sigTrue)
# Dimensions in matrix multiplication: [Ndat x 1] = [Ndat x P] %*% [P x 1] + [Ndat]
# cbind does the logical thing combining a scalar and vector; then vector addition
y <- drop(y) # converts into a vector
pdf("quadraticmodel_data.pdf", width=5, height=4)
par(mfrow=c(1,1), mar=c(3.0,3.0,0.5,0.5), oma=c(0.5,0.5,0.5,0.5), mgp=c(2.2,0.8,0), cex=1.0)
plotCI(x, y, xlim=xrange, uiw=sigTrue, gap=0)
# plot true model
xsamp <- seq(from=xrange[1], to=xrange[2], length.out=500)
ysamp <- cbind(1,xsamp,xsamp^2) %*% as.matrix(modMat)
lines(xsamp, drop(ysamp), col="red", lw=2)
dev.off()
# True parameters, transformed to be conformable with model to be used below
thetaTrue <- c(modMat[1], atan(modMat[2]), modMat[3], log10(sigTrue))
obsdata <- data.frame(cbind(x,y)) # only this is used in the analysis
rm(x,y)

########## Define model and infer the posterior PDF over its parameters

# Model to infer: quadratic regression with Gaussian noise
# Parameters: intercept a_0, gradient a_1, quadratic term a_2; Gaussian noise sigma, ysig.
# Prior PDFs over model parameters:
# a_0:  intercept. P(a_0) ~ N(mean=m, sd=s)
# a_1:  gradient, a_1 = tan(alpha), alpha is angle between horizontal and model line.
#       P(alpha) ~ 1 [uniform] => P(alpha) ~ 1/(1+tan(alpha)^2) but it is easier if we
#       use alpha (in radians) as the model parameter, alpha = atan(a_1)
# a_2:  quadratic term. P(a_0) ~ N(mean=m, sd=s)
# ysig: Jeffreys prior P(ysig) ~ 1/ysig, or equivalently P(log10(ysig)) ~ 1.
#       This is an "improper prior", which means its integral does not converge.
# theta is 1 x J vector of all model parameters (J=4): theta = c(a_0, alpha, a_2, log10(ysig)).
# The sampling is performed on theta.
```

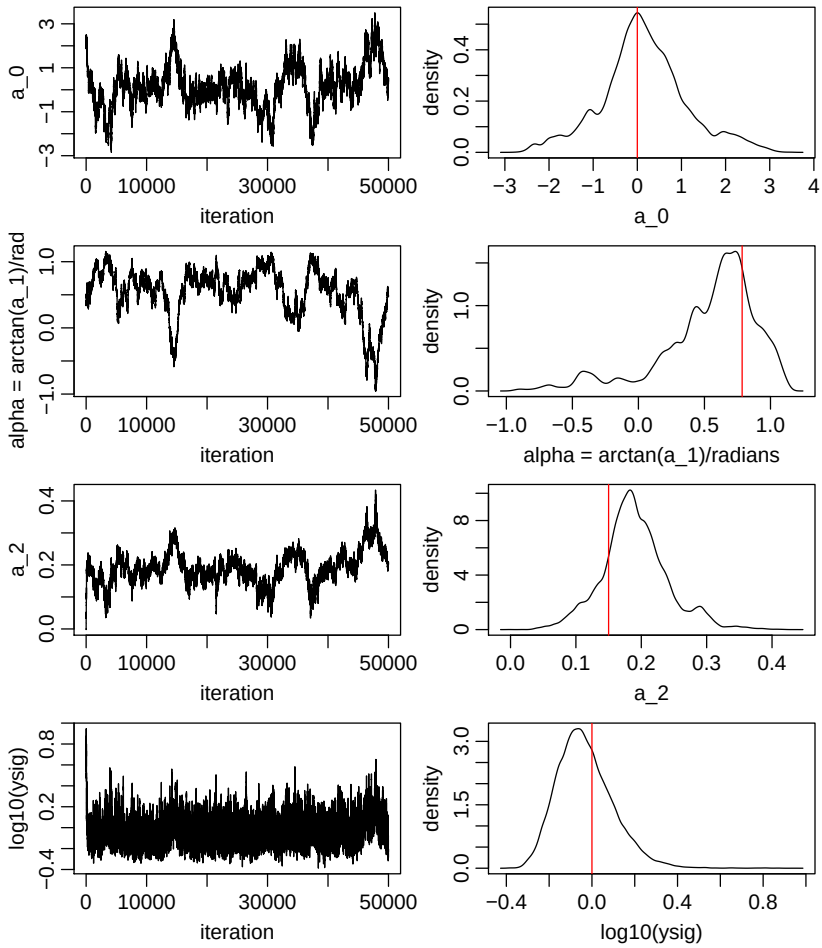

Figure 8: Example of the posterior PDF sampling of the sinusoidal model for the data shown in Fig. 7. The panels in the left column show the chains for the four parameters, those in the right the 1D PDFs generated from the samples via density estimation. The vertical red lines indicate the true parameters.

```
# define covariance matrix of MCMC sampling PDF: c(a_0, alpha, a_2, log10(ysig))
sampleCov <- make.covariance.matrix(sampleSD=c(0.1, 0.02, 0.01, 0.1), sampleCor=0)
# set starting point
thetaInit <- c(2, pi/8, 0, log10(3))
# run the MCMC to find postSamp, samples of the posterior PDF
set.seed(150)
postSamp <- metrop(func=logpost.quadraticmodel, thetaInit=thetaInit, Nburnin=0, Nsamp=5e4,
                   verbose=1e3, sampleCov=sampleCov, obsdata=obsdata)
# 10^(postSamp[,1]+postSamp[,2]) is the unnormalized posterior at each sample

# Plot MCMC chains and use density estimation to plot 1D posterior PDFs from these.
# Note that we don't need to do any explicit marginalization to get the 1D PDFs.
parnames <- c("a_0", "alpha = arctan(a_1)/radians", "a_2", "log10(ysig)")
pdf("quadraticmodel_mcmc.pdf", width=7, height=8)
```

```
par(mfrow=c(4,2), mar=c(3.0,3.0,0.5,0.5), oma=c(1,1,1,1), mgp=c(1.8,0.6,0), cex=1.0)
for(p in 3:6) { # columns of postSamp
  plot(1:nrow(postSamp), postSamp[,p], type="l", xlab="iteration", ylab=parnames[p-2])
  postDen <- density(postSamp[,p], n=2^10)
  plot(postDen$x, postDen$y, type="l", xlab=parnames[p-2], ylab="density")
  abline(v=thetaTrue[p-2], col="red")
}
dev.off()

# Plot a_0, a_1, and a_2 samples in 2D
par(mfrow=c(2,2))
plot(postSamp[,3], postSamp[,4], xlab="intercept a_0", ylab="alpha = atan(a_1) / radians", pch=".")
plot(postSamp[,3], postSamp[,5], xlab="intercept a_0", ylab="a_2", pch=".")
plot(postSamp[,4], tan(postSamp[,5]), xlab="alpha = atan(a_1) / radians",
     ylab="gradient a_2", pch=".")

# Find MAP solution and mean solution.
# MAP = Maximum A Posteriori, i.e. peak of posterior.
# MAP is not the peak in each 1D PDF, but the peak of the 4D PDF.
# mean is easy, because samples have been drawn from the (unnormalized) posterior.
posMAP     <- which.max(postSamp[,1]+postSamp[,2])
(thetaMAP  <- postSamp[posMAP, 3:6])
(thetaMean <- apply(postSamp[,3:6], 2, mean)) # Monte Carlo integration
# Overplot these solutions with original data and true model
pdf("quadraticmodel_fits.pdf", width=5, height=4)
par(mfrow=c(1,1), mar=c(3.0,3.0,0.5,0.5), oma=c(0.5,0.5,0.5,0.5), mgp=c(2.2,0.8,0), cex=1.0)
plotCI(obsdata$x, obsdata$y, xlab="x", ylab="y", xlim=xrange, uiw=sigTrue, gap=0)
ysamp <- cbind(1,xsamp,xsamp^2) %*% as.matrix(modMat)
lines(xsamp, drop(ysamp), col="red", lw=2) # true model
ysamp <- cbind(1,xsamp,xsamp^2) %*% as.matrix(thetaMAP[1:3])
lines(xsamp, drop(ysamp), col="blue", lw=2) # MAP model
ysamp <- cbind(1,xsamp,xsamp^2) %*% as.matrix(thetaMean[1:3])
lines(xsamp, drop(ysamp), col="green", lw=2) # mean model
dev.off()
```

R file: `Rcode/quadraticmodel.R`

```
# C.A.L. Bailer-Jones
# Astrostats 2013
# This file: quadraticmodel.R
# Functions to provide evaluations of prior, likelihood and posterior for quadratic model
# plus sampling from the prior

# theta is vector of parameters; obsdata is 2 column dataframe [x,y].
# the priors are hard-wired into the functions

# return log10(unnormalized posterior) of the quadratic model
# (see notes on the functions called)
logpost.quadraticmodel <- function(theta, obsdata, ind=NULL) {
  logprior <- logprior.quadraticmodel(theta)
  if(is.finite(logprior)) { # only evaluate model if parameters are sensible
    return( loglike.quadraticmodel(theta, obsdata, ind) + logprior )
  } else {
    return(-Inf)
```

```
  }
}

# return log10(likelihood) for parameters theta for rows ind in obsdata
# (do all if ind not specified)
# dnorm(..., log=TRUE) returns log base e, so multiply by 1/ln(10) = 0.4342945
# to get log base 10
loglike.quadraticmodel <- function(theta, obsdata, ind=NULL) {
  if(is.null(ind)) ind <- 1:nrow(obsdata)
  # convert alpha to a_1 and log10(ysig) to ysig
  theta[2] <- tan(theta[2])
  theta[4] <- 10^theta[4]
  modPred <- drop( theta[1:3] %*% t(cbind(1,obsdata[ind,]$x,obsdata[ind,]$x^2)) )
  # Dimensions in mixed vector matrix multiplication: [Ndat] = [P] %*% [P x Ndat]
  logLike <- (1/log(10))*sum( dnorm(modPred - obsdata[ind,]$y, mean=0, sd=theta[4], log=TRUE) )
  return(logLike)
}

# return log10(unnormalized prior)
logprior.quadraticmodel <- function(theta) {
  a0Prior      <- dnorm(theta[1], mean=0, sd=4)
  alphaPrior   <- 1
  a2Prior      <- dnorm(theta[3], mean=0, sd=1)
  logysigPrior <- 1
  logPrior <- sum( log10(a0Prior), log10(alphaPrior), log10(a2Prior), log10(logysigPrior) )
  return(logPrior)
}

# return Nsamp samples from prior
# NOTE: This uses a proper gamma prior on ysig instead of an improper Jeffreys prior,
# which is inconsistent with logprior.quadraticmodel(). This is generally a BAD IDEA
# and really we should modify logprior.quadraticmodel() to use proper priors.
sampleprior.quadraticmodel <- function(Nsamp) {
  a0 <- rnorm(Nsamp, mean=0, sd=4)
  a1 <- tan(runif(Nsamp, min=-pi/2, max=pi/2))
  a2 <- rnorm(Nsamp, mean=0, sd=1)
  logysig <- rgamma(Nsamp, shape=1.5, scale=1)
  return(cbind(a0, a1, a2, logysig))
}
```

7 Model comparison using the Bayesian evidence

Often we are interested in which model is a better description of the data, regardless of the actual values of the model parameters. In terms of probabilities, that means we would like to know $P(M|D)$, the posterior model probability (given the data). How can we calculate this? From Bayes' theorem we have

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)} \ . \tag{17}$$

The first term on the right-hand-side we can write as

marginal likelihood

$$P(D|M) = \int P(D,\theta|M)d\theta = \int P(D|\theta,M)P(\theta|M)d\theta \ . \tag{18}$$

This is just the likelihood averaged over the model parameters (weighted by their prior probabilities). This is called the marginal likelihood, or evidence. $P(D)$ does not involve the models, so does not help us discriminate them (it can just be considered as a normalization constant here). $P(M)$ is the model prior probability. If we have no reason to favour one model over another, then we just set all of these to be equal. In that case, the evidence is sufficient for model selection: the "best" model is the one which achieves the highest evidence.

Note that the evidence is just the denominator in equation 1, where it was "just" a normalization constant (integration of the numerator over all θ) in the context of parameter estimation. As such it was uninteresting, because there we were dealing with a single model. Now we have moved up a level in the hierarchy of inference in order to compare models, so this normalization constant becomes relevant.

If we have a set of W models, $\{M_i\}$, $i = 1 \ldots W$, which are exhaustive and mutually exclusive, then we can calculate the actual model posterior probabilities. Noting that $P(D) = \sum_i P(D|M_i)P(M_i)$, then for one particular model M_1

$$P(M_1|D) = \frac{P(D|M_1)P(M_1)}{\sum_i P(D|M_i)P(M_i)} \ . \tag{19}$$

Dividing top and bottom by the numerator and setting all the $P(M_i)$ equal (no model preference), we have

$$P(M_1|D) = \left(1 + \sum_{i=2}^{i=W} \frac{P(D|M_i)}{P(D|M_1)}\right)^{-1} \ . \tag{20}$$

Each of the terms $P(D|M_k)/P(D|M_1)$ is just the ratio of evidence of each of the models with respect to the evidence for M_1, known as the *Bayes factor*. Bayes factor

Note that the evidence depends on the choice of prior. That choice may not be unique. Thus when using the evidence for model selection, we should investigate its sensitivity to the choice of prior. Furthermore, the "exhaustive and mutually exclusive" condition for the set of models is sometimes hard to achieve, although Bayes factors can still be used to compare pairs of models.

7.1 Understanding the evidence

The evidence tells us how well the data are predicted given the model. Imagine comparing a simple model, M_s, such as the linear one, with a more complex model, M_c, such as a higher order polynomial. For some specific data sets, the more complex model will give better predictions – will fit the data better – than the simpler model. But for many more data sets the simpler model will give better predictions. This is illustrated schematically in Figure 9. The important point is that the evidence is a normalized PDF over the data, so improved predictive power (a larger probability density) over some part of the data space must be traded off against a lower one elsewhere.

When doing model comparison, one might be tempted to simply look at which model predicts the data better, e.g. which gives the higher maximum likelihood solution (maximum of $P(\theta|D, M)$ for that particular model). But this is very wrong, because it will generally favour the more complex model on account of such models being more flexible and thus better able

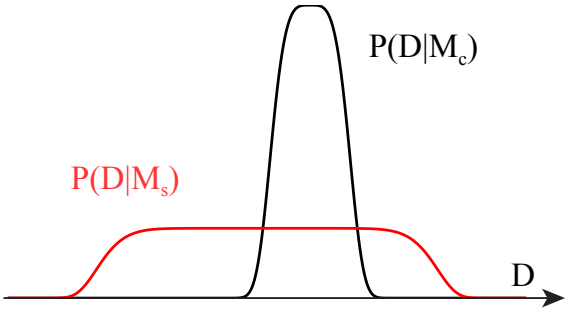

Figure 9: Illustration of the evidence for a simpler (M_s) and a more complex (M_c) model.

to fit the data. (A seventh order polynomial will generally give a much better fit – have a higher maximum likelihood – to ten arbitrary data points than a linear model. Does that make it the better model?) Using the maximum likelihood would also not be answering the right question, because when we ask which model is better, we are asking which model is more probable given the data, but regardless of any specific values of the parameters. That is, we want to compare values of $P(M|D)$.

Accommodating different model complexities is therefore a central aspect of model comparison. We can get an intuitive feel for this by taking the logarithm of equation 1 and rearranging it

$$\begin{aligned} \log P(D|M) &= \log P(D|\theta, M) + [\log P(\theta|M) - \log P(\theta|D, M)] \\ \log(\text{evidence}) &= \log(\text{likelihood}) + [\log(\text{prior}) - \log(\text{posterior})] \ . \end{aligned} \tag{21}$$

A more complex model will generally have a posterior which is larger than the prior. This is because a model with more parameters will have to spread its prior probability (which must integrate to unity) over a larger volume of parameter space than a model with fewer parameters, thus lowering the typical value of its prior PDF. Consequently, the quantity in square brackets tends to be more negative for a more complex model. This term acts as a penalty which reduces the evidence *for a given likelihood*. Thus for a more complex model to achieve a higher evidence, it has to achieve a high enough likelihood in order to overcome this penalty. Thus the evidence takes into account model complexity when used to compare models.

Occam factor

It is not the case that more complex models are penalized simply on the grounds of being complex. What counts is how the plausibility of the model is changed in light of the data. This can be understood by the concept of the *Occam factor*. If the likelihood function is dominated by a single peak, then we can approximate the evidence (average likelihood) as

$$\underbrace{P(D|M)}_{\text{Evidence}} = \underbrace{\mathcal{L}(\hat{\theta})}_{\text{best fit likelihood}} \times \underbrace{\frac{\Delta\theta_{\text{posterior}}}{\Delta\theta_{\text{prior}}}}_{\text{Occam factor}} \tag{22}$$

where $\mathcal{L}(\hat{\theta}) = P(D|\hat{\theta}, M)$ is the likelihood at the best fit solution, $\Delta\theta_{\text{prior}}$ is the prior parameter range and $\Delta\theta_{\text{posterior}}$ is the posterior parameter range (the width of the likelihood peak), see Figure 10. The Occam factor (which is always less than or equal to one) measures

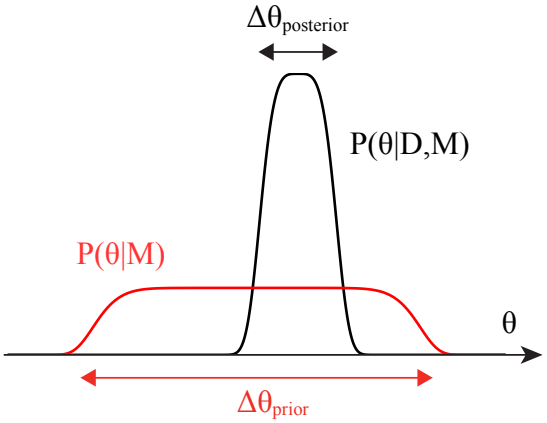

Figure 10: Illustration of the concept of the Occam factor.

the amount by which the plausible parameter volume shrinks on account of the data. For given $\mathcal{L}(\hat{\theta})$, a simple or general model will fit over a large part of the parameter space, so $\Delta\theta_{\rm prior} \sim \Delta\theta_{\rm posterior}$ and the Occam factor is not significantly less than one. In contrast, a more complex model, or one which has to be more finely tuned to fit the data, will have a larger shrinkage, so $\Delta\theta_{\rm posterior} \ll \Delta\theta_{\rm prior}$. In this case the Occam factor is small and the evidence is reduced. Of course, if the fit is good enough then $\mathcal{L}(\hat{\theta})$ will be large, perhaps large enough to dominate the Occam factor and to give the model a large evidence.

This concept helps us to understand how the Bayesian approach accommodates model complexity, something generally lacking in frequentist approaches. If we assess a model's evidence only by looking at the maximum likelihood solution (or the maximum over one parameter, the period), then we artificially compress the prior parameter range, increasing the Occam factor.

8 Monte Carlo integration

To calculate the evidence, or indeed any normalization constant, we need to integrate some function $f(\theta)$ over θ. It can rarely be expressed in closed form. A simple solution is to evaluate it numerically over a dense, regular grid. If the N values in this grid are represented with the set $\{\theta_n\}$, then in general

$$\int_\theta f(\theta)d\theta \approx \sum_{n=1}^{n=N} f(\theta_n)\delta\theta = \frac{\Delta\Theta}{N}\sum_{n=1}^{n=N} f(\theta_n) \tag{23}$$

for some function $f(\theta)$, in which $\Delta\Theta$ is the total range of the grid of θ values, and $\delta\theta = \Delta\Theta/N$ is the spacing between them. If we set $f(\theta) = P(D|\theta, M)P(\theta|M)$, the unnormalized posterior, then this summation gives us a numerical approximation of the evidence

Equation 23 actually holds for a non-regular grid ($\{\theta_n\}$ drawn from a uniform distribution) in which case $\delta\theta$ is the average spacing between the points. It also holds – in principle – where θ is a vector, i.e. a higher dimensional parameter space. However, as explained in section 4, this numerical estimate of the integral suffers from the curse of dimensionality.

We can do integration more efficiently than this (i.e. with fewer samples) once we have managed to sample explicitly from a PDF. The Monte Carlo approximation of an integration of some function $f(\theta)$ over $g(\theta)$ is

$$\langle f(\theta) \rangle \;=\; \frac{\int_\theta g(\theta) f(\theta) d\theta}{\int_\theta g(\theta) d\theta} \;\approx\; \frac{1}{N} \sum_{n=1}^{n=N} f(\theta_n) \tag{24}$$

where the sample $\{\theta_n\}$ has been drawn from the PDF $g(\theta)$, which we assume to be unnormalized. If it were in fact normalized, then the denominator would be unity and would vanish. Note that the set of samples we get from $g(\theta)$ by Monte Carlo is the same whether it is normalized or not. Hence, if we want to integrate $f(\theta)$ over a normalized PDF, $Zg(\theta)$ for some unknown normalization constant Z, then we can happily sample from the unnormalized PDF $g(\theta)$. This is because that constant won't make any difference to the relative frequency with which samples are drawn at different values of θ.

One particular case is worth noting. If $g(\theta)$ is the uniform distribution over the range $\Delta\Theta$, then $g(\theta) = 1$ and $\int_\theta g(\theta) d\theta = \Delta\Theta$. Equation 24 then just reduces to equation 23. In this case we see that normalization constant, $\Delta\Theta$, which is just a length scale (or a volume in the general, higher dimensional case) is still present in the Monte Carlo approximation of the integral. The reason for this is that unless we constrain its range, the uniform distribution stretches over an infinite range but with a finite value, so its integral is not finite. This is known as an *improper distribution*. In contrast, we usually use a sampling distribution, $g(\theta)$ which is proper, i.e. normalizable, even though we do not necessarily know its normalization constant.

Monte Carlo estimation of the evidence

With $g(\theta)$ equal to the prior and $f(\theta)$ equal to the likelihood, the equation 24 gives the evidence (equation 18). That is, by drawing samples from the prior and then calculating the likelihood at these, the average of these likelihoods is the evidence.

There are at least two other uses for Monte Carlo integration in Bayesian inference

1. with $g(\theta)$ equal to the posterior and $f(\theta) = \theta$, the integral gives the expectation value of θ (equation 32 in section E.2). Indeed, equation 24 just tells us that the expectation value of θ with respect to $g(\theta)$ is just the average of samples drawn from $g(\theta)$ (even if it's unnormalized).[4]
2. with $g(\theta)$ equal to the posterior and $f(\theta)$ equal to the likelihood of a new data point, we obtain the posterior predictive distribution (equation 35 in section E.3). That is, we average the likelihood over the (samples drawn from the) parameter posterior PDF.

9 Model comparison alternatives to the evidence

One drawback of the evidence is that is it sensitive to the prior adopted. This is not a problem per se, but becomes one if we have little idea what the prior should be. As an example, if the likelihood were constant over the range $0 < \theta < 1$ but essentially zero outside this (a somewhat extreme, admittedly), then the evidence calculated using a prior uniform

[4] In section 3 we *did* need the normalization constant to calculate the expectation values. That was because we did not have samples drawn from the posterior (normalized or not), but rather from a (regular) uniform distribution.

over $0 < \theta < 2$ would be half that calculated using a prior uniform over $0 < \theta < 1$. In a model with p such parameters the factor would be 2^{-p}. In cases where the parameters have a physical interpretation and/or where we have reasonable prior information, then we may be able to justify a reasonable choice for the prior. But one should always explore the sensitivity of the evidence to "fair" changes in the prior, i.e. ones which we have no particular reason not to make. If the evidence changes enough to alter significantly the Bayes factors when making fair changes, then the evidence is over-sensitive to the choice of prior, making it impossible to draw robust conclusions without additional information.

In such situations we might resort to one of the many "information criteria" which have been defined, such as the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) or the Deviance Information Criterion (DIC). These all use the maximum likelihood, and as it often faster to locate this than it is to adequately sample the likelihood, they are usually quicker to calculate. But they all make rather simple and possibly unreasonable assumptions regarding how to represent the complexity of the model, and all have been criticized in the literature.

When doing parameter estimation we draw from the posterior. When calculating the evidence we (normally) draw from the prior. Both require separate evaluations of the likelihood, which are often quite expensive. Ideally we would like to do model comparison using the posterior samples. The following approach allows us to do this, and to reduce the sensitivity to the prior.

9.1 K-fold cross validation likelihood

An alternative approach to model comparison is a form of K-fold cross validation (CV). We split the data set (I events) into K disjoint partitions, where $K \leq I$. Denote the data in the k^{th} partition as D_k and its complement as D_{-k}. The idea is to calculate the likelihood of D_k using D_{-k}, without having an additional dependence on a specific choice of model parameters. That is, we want $P(D_k|D_{-k}, M)$, which tells us how well, in model M, some of the data are predicted using the other data. Combining these likelihoods for all K partitions gives an overall measure of the fit of the model. By marginalization

$$\begin{aligned} P(D_k|D_{-k}, M) &= \int_\theta P(D_k|D_{-k}, \theta, M) P(\theta|D_{-k}, M) d\theta \\ &= \int_\theta \underbrace{P(D_k|\theta, M)}_{\text{likelihood}} \underbrace{P(\theta|D_{-k}, M)}_{\text{posterior}} d\theta \end{aligned} \tag{25}$$

where D_{-k} drops out of the first term because the model predictions are independent of these data when θ is specified. If we draw a sample $\{\theta_n\}$ of size N from the posterior $P(\theta|D_{-k}, M)$, then the Monte Carlo approximation of this integral is

$$L_k \equiv P(D_k|D_{-k}, M) \approx \frac{1}{N} \sum_{n=1}^{n=N} P(D_k|\theta_n, M) \tag{26}$$

the mean of the likelihood of the data in partition k. I will call L_k the *partition likelihood*. Note that here the posterior is sampled using the data D_{-k} only.

Because L_k is the product of event likelihoods, it scales multiplicatively with the number of events in partition k. An appropriate combination of the partition likelihoods over all

partitions is therefore their product

$$L_{\rm CV} \;=\; \prod_{k=1}^{k=K} L_k \quad \text{or} \quad \log L_{\rm CV} \;=\; \sum_k \log L_k \tag{27}$$

which I call the *K-fold cross validation likelihood*, for $1 \leq K \leq I$. If $K > 1$ and $K < I$ then its value will depend on the choice of partitions. If $K = I$ there is one event per partition (a unique choice). This is *leave-one-out CV (LOO-CV)*. If $K = 1$, we just use all of the data to calculate both the likelihood and the posterior. This is not a very correct measure of goodness-of-fit, however, because it uses all of the data both to draw the posterior samples and to calculate the likelihood.

The posterior PDF required in equation 25 is given by Bayes' theorem. It is sufficient to use the unnormalized posterior (as indeed we must, because the normalization term is the evidence), which is

$$P(\theta|D_{-k}, M) \;\propto\; P(D_{-k}|\theta, M)P(\theta|M) \tag{28}$$

i.e. the product of the likelihood and the prior. $L_{\rm CV}$ therefore still depends on the choice of prior. However, the likelihood will often dominate the prior (unless the data are very indeterminate), in which case $L_{\rm CV}$ should be less sensitive to the prior than is the evidence.

There is a close relationship between the partition likelihood and the evidence. Whereas the evidence involves integrating the likelihood (for D) over the *prior* (equation 18), the partition likelihood involves integrating the likelihood (for D_k) over the *posterior* (for D_{-k}) (equation 25). This is like using D_{-k} to build a new prior from "previous" data. We can use the product rule to write the partition likelihood as

$$L_k \;\equiv\; P(D_k|D_{-k}, M) \;=\; \frac{P(D|M)}{P(D_{-k}|M)} \tag{29}$$

which shows that it is equal to the ratio of the evidence calculated over all the data to the evidence calculated on the subset of the data used in the posterior sampling. As the same prior PDF enters into both terms, it will, in some vague sense, "cancel" out, although I stress that there is still a prior dependence.

It is important to realize that the model complexity is taken into account by the model comparison with the K-fold CV likelihood, just as it is with the Bayesian evidence. That is, more complex models are not penalized simply on account of having more parameters. It is, as usual, the prior plausibility of the model which counts.

10 Comparing the linear and quadratic models

As we saw in section 8, the evidence can be estimated by averaging the likelihood over samples drawn from the prior (which is why it is also called the marginal likelihood). One often uses prior distributions – such as uniform, Gaussian, gamma, and beta – which are easy to sample, so MCMC is not required. Calculating the evidence can nonetheless be time consuming because we often need a lot of samples (with the time dominated by the likelihood calculations).

We now calculate both the evidence for both the linear and quadratic models for a given set of data. To do this we must make one modification, however. In order to sample from a prior distribution it must be proper, i.e. normalizable. This is not true for the Jeffrey's prior adopted for the noise parameter (σ, in equation 12). I therefore change the prior on that parameter to a gamma prior with scale=1 (which is just the exponential function).

We will also calculate the K-fold CV likelihood (section 9.1) for both models. This involves sampling the posterior, and for this we will use MCMC. (Note that this does not need a proper prior, but we'll stick to the same prior in order to compare the methods.)

In the R script below, data are simulated from a quadratic model. By setting the value of a_2 in `modMat` to zero we change this to a linear one. The (hyper)parameters of the prior are hard-wired into the functions, so change these if you want to investigate the effect of changing the priors.

R file: `Rcode/model_comparison_1.R`

```
# C.A.L. Bailer-Jones
# Astrostats 2013
# This file: model_comparison_1.R
# R code to calculate evidence and K-fold CV likelihood for linear and quadratic models

library(gplots) # for plotCI()
source("linearmodel.R")
source("quadraticmodel.R")
source("monte_carlo.R")
source("kfoldCV.R")

########## Define true model and simulate experimental data from it

set.seed(50)
Ndat <- 10
xrange <- c(0,10)
x <- sort(runif(Ndat, xrange[1], xrange[2]))
sigTrue <- 1
modMat <- c(0,1,0.3) # 1 x P vector: coefficients, a_p, of polynomial sum_{p=0} a_p*x^p
y <- cbind(1,x,x^2) %*% as.matrix(modMat) + rnorm(Ndat, 0, sigTrue)
# Dimensions in matrix multiplication: [Ndat x 1] = [Ndat x P] %*% [P x 1] + [Ndat]
# cbind does the logical thing combining a scalar and vector; then vector addition
y <- drop(y) # converts into a vector
par(mfrow=c(1,1))
plotCI(x, y, xlim=xrange, uiw=sigTrue, gap=0)
# plot true model
xsamp <- seq(from=xrange[1], to=xrange[2], length.out=500)
ysamp <- cbind(1,xsamp,xsamp^2) %*% as.matrix(modMat)
lines(xsamp, drop(ysamp), col="red", lw=2)
# True parameters, transformed to be conformable with model to be used below
thetaTrue <- c(modMat[1], atan(modMat[2]), modMat[3], log10(sigTrue))
obsdata <- data.frame(cbind(x,y)) # only this is used in the analysis
rm(x,y,xsamp,ysamp)

########## Calculate evidences

set.seed(100)
```

```
# linear model
Nsamp <- 1e5
priorSamples <- sampleprior.linearmodel(Nsamp)
logLike <- vector(length=Nsamp)
for(i in 1:Nsamp) {
  logLike[i] <- loglike.linearmodel(priorSamples[i,], obsdata)
}
evLM <- mean(10^logLike)
# quadratic model
Nsamp <- 1e5
priorSamples <- sampleprior.quadraticmodel(Nsamp)
logLike <- vector(length=Nsamp)
for(i in 1:Nsamp) {
  logLike[i] <- loglike.quadraticmodel(priorSamples[i,], obsdata)
}
evQM <- mean(10^logLike)
#
cat("Bayes factor [Quadratic/Linear] = ", evQM/evLM, "\n")
cat("log10 Bayes factor [Quadratic - Linear] = ", log10(evQM/evLM), "\n")

########## Calculate K-fold CV likelihoods

set.seed(100)
# linear model: c(a_0, alpha, ysig)
sampleCov <- make.covariance.matrix(sampleSD=c(0.1, 0.02, 0.1), sampleCor=0)
thetaInit <- c(2, pi/8, log10(3))
kcvLM <- kfoldcv(Npart=5, obsdata=obsdata, logpost=logpost.linearmodel,
                 loglike=loglike.linearmodel, sampleCov=sampleCov, thetaInit=thetaInit,
                 Nburnin=1e3, Nsamp=1e4)
# quadratic model: c(a_0, alpha, a_2, log10(ysig))
sampleCov <- make.covariance.matrix(sampleSD=c(0.1, 0.02, 0.01, 0.1), sampleCor=0)
thetaInit <- c(2, pi/8, 0, log10(3))
kcvQM <- kfoldcv(Npart=5, obsdata=obsdata, logpost=logpost.quadraticmodel,
                 loglike=loglike.quadraticmodel, sampleCov=sampleCov, thetaInit=thetaInit,
                 Nburnin=1e3, Nsamp=1e4)
#
cat("Difference log10 K-fold CV likelihood [Quadratic - Linear]", kcvQM - kcvLM, "\n")
```

11 Exercises

1. Investigate the linear (section 5) and quadratic (section 6) parameter estimation examples, making the following changes

   - different initial values of the MCMC and/or random number seed, and thus a different sequence of samples
   - different lengths of the chain (`Nsamp` and `Nburnin`) (by factors of more than two)
   - different values of the standard deviations, `sampleSD`, in the covariance matrix of the proposal distribution. Try doubling the step sizes, for example
   - different parameters for the priors
   - different values for the model parameters used for data generation (e.g. the degree of nonlinearity in the data)

- different amounts of data (e.g. by factors of two, up and down). Try also using very little data ...

What impact do these changes have on the results? Produce the same plots as in the example: the chains and posterior PDFs; the model solutions for the mean and MAP parameters.

2. Repeat the linear and quadratic model comparison example using the evidence and K-fold CV likelihood (section 10) making the changes outlined as in exercise 1. See `model_comparison_1.R` for details. Investigate in particular the dependence of these on the priors. Calculating the K-fold CV likelihood is much more time consuming than computing the evidence, but if you have time, investigate the effect of different data partition sizes.

3. (A) Look at the mass–luminosity data file `MLR.dat` used to produce Figure 4. Fit linear and quadratic models to it using the methods (and code) described above. Use the evidence to decide whether a quadratic fit is preferred over a linear one.
Note: the uncertainties in mass are much larger than those in the K-band magnitude, so for the sake of this exercise just consider the mass to have an (unknown) uncertainty.
(B) Modify the code to take into account the measured mass uncertainties, σ_i, treating the parameter σ as an extra (positive) degree of uncertainty. How might you achieve this?

4. (Extra) Look into Bayesian prediction, as described in section E.3 in the appendix (and see the code in `exp_linearmodel.R`). How does the accuracy and precision of the prediction vary with: the amount of data used to fit the model; how "far" the position of prediction is from the data (i.e. how much of an extrapolation it is).

A Further reading

Bailer-Jones C.A.L., 2012, *A Bayesian method for the analysis of deterministic and stochastic time series*, A&A 546, A89
Here I introduce and use the K-fold CV likelihood. See also this technical note for another way to implement a linear model and to deal with arbitrary error bars on both axes

Gregory P.C., 2005, *Bayesian logical data analysis for the physical sciences*, Cambridge University Press
One of the relatively few good books I know of which give a reasonably comprehensive treatment for a physicist

Jaynes E.T., 2003, *Probability theory. The logic of science*, Cambridge University Press
Idiosyncratic and opinionated (even polemic), this is great book by one of the architects of Bayesian statistics, for those who want to go deeper

Kadane J.B., Lazar N.A., 2004, *Methods and criteria for model selection*, Journal of the American Statistical Association 99, 465
Defines and discusses the AIC and BIC

Kass R., Raftery A., 1995, *Bayes factors*, Journal of the American Statistical Association 90, 773

Almost everything you want to know about using the Bayesian evidence

MacKay D.J.C., 2003, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press
Has a good chapter on Monte Carlo methods, and much else besides

Spiegelhalter D.J., Best N.G., Carlin B.P., van der Linde A., 2002, *Bayesian measures of model complexity and fit*, J. R. Statist. Soc. B, 64, 583
Detailed discussion of the DIC

Vehtari A., Lampinen J., 2002 *Bayesian model assessment and comparison using cross-validation predictive densities*, Neural Computation 14, 2439
They introduce something very similar to the K-fold CV likelihood

von Toussaint U., 2011, *Bayesian inference in physics*, Rev. Mod. Physics 83, 943
A nice overview

B Laplace's law of succession

Imagine we have observed an event n times, and this event can have just two possible outcomes, which we can call success and fail. Suppose we have observed r successes. What is our best estimate for the probability of success, p, for each event, assuming it has remained constant?

A reasonable guess is r/n. Laplace, however, suggested $(1+r)/(2+n)$, and this is referred to as *Laplace's law of succession*. Why did he suggest this? One reason is that it gives a better estimate in the limit as r and n tend to zero. For example, with $n = 1$ and $r = 0$, our reasonable guess gives $r/n = 0$. But this doesn't seem to be a very good estimate of p given so little data. Laplace would say $p = 1/3$, which seems nicer. Furthermore, when $r = n = 0$, r/n isn't even defined, but Laplace would say $p = 1/2$, which seems acceptable if we have no data.

The basis behind this is, of course, the inference of the probability p for the binomial likelihood discussed in section 3. If we adopt a beta distribution prior PDF with parameters (α_p, β_p), then the posterior PDF is also a beta distribution with mean and mode given by

$$\text{mode} = (\alpha - 1)/(\alpha + \beta - 2) = (\alpha_p + r - 1)/(\alpha_p + \beta_p + n - 2)$$

$$\text{mean} = \alpha/(\alpha + \beta) = (\alpha_p + r)/(\alpha_p + \beta_p + n)$$

For a uniform prior, $\alpha = \beta = 1$, so

$$\text{mode} = r/n$$

This is intuitive, but not defined for no data.

$$\text{mean} = (1 + r)/(2 + n)$$

This is the origin Laplace's law of succession, and it's the Bayesian thing to do, because it behaves sensibly in the limit of no data. Of course, this does not imply that the mean is always (or even generally) a better summary of a distribution than is the mode. Always look at your posterior PDF!

C Monte Carlo sampling

In its general sense, "Monte Carlo" means random samples to solve what would otherwise be hard combinatorial problems. The idea goes back at least to the 1940s, but its use in inference probably only dates back to the late 1980s. Here we use it to mean sampling from an arbitrary probability density function, $g(\theta)$, using random numbers drawn from a simpler distribution, i.e. one we *can* easily draw from. In the context of Bayesian models, $g(\theta)$ might be the (unnormalized) posterior PDF.

Note that having samples drawn *from* $g(\theta)$ is quite different from defining an arbitrary set of values (perhaps a regular grid) of θ at which we then evaluate $g(\theta)$. This is what we mean by *sampling*. The frequency distribution of such samples is equal to $g(\theta)$ in the limit of a very large number of samples.

Once we have obtained a set of samples from a posterior PDF $g(\theta)$, then these are sufficient to represent the posterior PDF for plotting it. That is, we do not need to know the actual value of the posterior probability (normalized or not). We simply use *density estimation* instead.

We will now look at two widely-used method of sampling $g(\theta)$. We assume that we can evaluate $g(\theta)$. We do not require that $g(\theta)$ be normalized: it need only be known up to a multiplicative constant.

C.1 Rejection sampling

The idea behind rejection sampling is to define a simpler function which we *can* draw samples from, the *proposal distribution*. Let's call this $Q(\theta)$ (it need not be normalized) and define it such that $Q(\theta) > g(\theta) \,\forall\, \theta$. We now draw samples from $Q(\theta)$ but reject each sample with a probability

$$\frac{Q(\theta) - g(\theta)}{Q(\theta)} = 1 - \frac{g(\theta)}{Q(\theta)}$$

or equivalently, only retain each with a probability $g(\theta)/Q(\theta)$. The set of samples retained is equivalent to having been drawn from $g(\theta)$.

Unfortunately, this method is often inefficient, because often we need $Q(\theta)$ to be much larger than $g(\theta)$ in order for the former to be a sufficiently simple function. Consequently we get a very large number of rejections so need a very large number of draws and function evaluations.

C.2 Markov Chain Monte Carlo: the Metropolis–Hastings algorithm

Rejection sampling generated independent samples $g(\theta)$. But it turns out that we don't actually need the samples to be independent. We may in fact generate them from any process provided it produces a distribution which is the same as $g(\theta)$, i.e. gives samples in the correct proportions. This allows us to sample much more efficiently.

The principle of a *Markov Chain Monte Carlo* (MCMC) method is to set up a random walk over the parameter space θ in such a way that one preferentially samples regions of high probability. The random walk is achieved using a *Markov chain*, which means that the

probability of moving from sample θ_t to θ_{t+1} is defined by a transition probability $Q(\theta_{t+1}|\theta_t)$ which is independent of the "time", or step number, t. It only depends on the current position. (The "chain" is the sequence of samples.) The reason this works is that for a sufficiently large number of samples, the Markov chain generates samples which have a PDF equal to $g(\theta)$. In other words, $g(\theta)$ is the stationary distribution of the Markov chain.

An important, simple and widely-used MCMC method is the Metropolis–Hastings algorithm. It again uses a proposal distribution, $Q(\theta)$, from which we (easily) draw a candidate sample for the next step given the current parameter value, θ_t. The algorithm iterates the following two steps

1. Draw at random a candidate value of θ_{t+1}, which we'll call s, from the proposal distribution $Q(s|\theta_t)$. This could be (and often is) a multivariate Gaussian, with mean θ_t and some covariance matrix which specifies the typical size of steps in the chain in each dimension of θ. Generally we want a proposal distribution which gives a higher probability of picking nearby points than far away ones, and usually the covariance matrix is fixed.

2. Decide whether or not to accept this candidate based on the value of

$$\rho = \frac{g(s)}{g(\theta_t)} \frac{Q(\theta_t|s)}{Q(s|\theta_t)} \ . \tag{30}$$

   If $\rho \geq 1$ then we accept the transition and set $\theta_{t+1} = s$. If $\rho < 1$ we only accept with a probability of ρ (i.e. we draw a number from a uniform distribution $U(0,1)$ and compare to ρ). If we don't accept then we set $\theta_{t+1} = \theta_t$ (i.e. repeat θ_t in the chain).

The algorithm iterates between these two steps and is stopped after some large number (typically 10^4 to 10^6) steps.

It turns out that, depending on where in the distribution the chain is initialized, the initial samples are not representative of $g(\theta)$. These samples are therefore not retained in the final set of samples. These initial samples are called the "burn in" samples, and may typically be 10% or more of the total chain.

The algorithm is named after the people who first invented it – Metropolis et al. 1953; who used symmetric proposal distributions, i.e. $Q(\theta_t|s) = Q(s|\theta_t)$ – and the person who improved it (Hastings 1970). Note that the multivariate Gaussian is symmetric, so when we use that, we don't need the term $Q(\theta_t|s)/Q(s|\theta_t)$ in the definition of ρ.

There are many variations on the algorithm (Gibbs sampling, slice sampling, Hamiltonian Monte Carlo, ...), and there is a lot of theory and practice in what to use as the covariance matrix of the proposal distribution, how long the burn-in period should be, how many iterations we expect to need before convergence etc. It turns out that for many proposal distributions, the above procedure (Markov chain) does have the essential property that its stationary distribution is the one desired.

One thing to be aware of is that the samples in the chain are generally correlated. You can inspect this by plotting the autocorrelation function. The degree of correlation depends on the typical step size, i.e. on the covariance matrix of the proposal distribution. If the steps are large, then the correlation is small (or rather, the autocorrelation function decays to

small values after a small lag), yet such large steps might fail to sample $g(\theta)$ finely enough. Conversely, if the steps are small then the correlation is large, and the resulting PDF may not be representative of $g(\theta)$. One way around this is then to use "thinning", which means to retain only every k^{th} sample in the chain, thereby yielding a less correlated sequence of samples.

C.3 R code for the Metropolis algorithm

R file: `Rcode/monte_carlo.R`

```
# C.A.L. Bailer-Jones
# Astrostats 2013
# This file: monte_carlo.R
# R code for Metropolis algorithm (and for constructing covariance matrix)

library(mvtnorm) # for rmvnorm()

# Metropolis (MCMC) algorithm to sample from function func()
# The first argument of func must be a real vector of parameters, the initial values
# of which are provided by the real vector thetaInit.
# func() returns a two-element vector, the logPrior and logLike (log base 10), the sum
# of which is taken to be the log of the density function (i.e. unnormalized posterior).
# The MCMC sampling PDF is the multivariate Gaussian with fixed covariance, sampleCov.
# A total of Nburnin+Nsamp samples are drawn, of which the last Nsamp are kept.
# As the sampling PDF is symmetric, the Hasting factor cancels,
# leaving the basic Metropolis algorithm.
# Diagnostics are printed very verboseˆth sample: sample number, acceptance rate so far.
# ... is used to pass data, prior parameters etc. to func()
# Return a Nsamp * (2+Ntheta) matrix (no names), where the columns are
# 1:  log10 prior PDF
# 2:  log10 likelihood
# 3+: Ntheta parameters
# (The order of the parameters in thetaInit and sampleCov must match, of course.)
metrop <- function(func, thetaInit, Nburnin, Nsamp, verbose, sampleCov, ...) {

  Ntheta   <- length(thetaInit)
  thetaCur <- thetaInit
  funcCur  <- func(thetaInit, ...) # log10
  funcSamp <- matrix(data=NA, nrow=Nsamp, ncol=2+Ntheta) # this will be filled and returned
  nAccept  <- 0
  acceptRate <- 0

  for(n in 1:(Nburnin+Nsamp)) {

    # Metropolis algorithm. No Hastings factor for symmetric sampling distributions.
    thetaProp <- rmvnorm(n=1, mean=thetaCur, sigma=sampleCov, method="eigen")
    funcProp  <- func(thetaProp, ...)
    logMR <- sum(funcProp) - sum(funcCur) # log10 of the Metropolis ratio
    if(logMR>=0 || logMR>log10(runif(1, min=0, max=1))) {
      thetaCur   <- thetaProp
      funcCur    <- funcProp
      nAccept    <- nAccept + 1
      acceptRate <- nAccept/n
```

```
    }
    if(n>Nburnin) {
      funcSamp[n-Nburnin,1:2] <- funcCur
      funcSamp[n-Nburnin,3:(2+Ntheta)] <- thetaCur
    }

    # diagnostics
    if( is.finite(verbose) && (n%%verbose==0 || n==Nburnin+Nsamp) ) {
      sdump1 <- noquote( formatC(n,          format="d", digits=5, flag="") )
      sdump2 <- noquote( formatC(Nburnin,    format="g", digits=5, flag="") )
      sdump3 <- noquote( formatC(Nsamp,      format="g", digits=5, flag="") )
      sdump4 <- noquote( formatC(acceptRate, format="f", width=7, digits=4, flag="") )
      cat(sdump1, "of", sdump2, "+", sdump3, sdump4, "\n")
    }

  }

  return(funcSamp)

}

# Return a covariance matrix given a vector of the standard deviations
# and the global (i.e. scalar) correlation coefficient.
# This is inefficient, but matrix is very small and only done once.
make.covariance.matrix <- function(sampleSD, sampleCor) {
  Len <- length(sampleSD)
  covMatrix <- matrix(nrow=Len, ncol=Len)
  if(abs(sampleCor)>1) {stop("|sampleCor| > 1")}
  for(i in 1:(Len)) {
    for(j in 1:(Len)) {
      if(i==j) {
        covMatrix[i,j] <- sampleSD[i]^2
      } else {
        covMatrix[i,j] <- sampleCor*sampleSD[i]*sampleSD[j]
      }
    }
  }
  return(covMatrix)
}
```

D Fewer data points than parameters?

When fitting a model with J parameters, conventional wisdom tells us that we need to have at least J independent data points. With Bayesian parameters inference, we have no such limitation. Consider the problem discussed above of fitting the linear model with unknown noise, which has three parameters.

What happens if you reduce the size of the data set to just two points, or even one?
Even though you now have fewer parameters than data points, the likelihood function is still defined, so you still get a posterior PDF. The posterior is likely to be a lot wider now, because with less data you will generally have less certainty about the values of the parameters. But you still have a distribution from which you can estimate the mean or mode of the parameters.

If you don't believe me, try it.[5] However, look carefully at the MCMC chains: they are likely to be poor, i.e. not in a steady state, with few data. If that is the case, then they are not representative of the posterior. This raises an important point about MCMC which we have not looked at, namely analysing the chains to ensure that they have reached steady state. You may well need a lot more samples, a longer burn-in, and/or a different covariance matrix for the proposal distribution.

But how, logically thinking, can you estimate three parameters at all given only one data point (or two data points)? Surely the solution is somehow "underdetermined?"
No, it's not. That's because you're not only using the data in your inference. You also have a prior. The prior on the α (= atan(a_1)) is uniform, but the prior on the intercept (a_0) is not. We used a Gaussian with mean zero and standard deviation 2. This is a constraint on the model. It is, if you like, a fuzzy data point (y value) at $x = 0$. If we had set the standard deviation of this prior PDF to be much larger, then the posterior over this parameters would also be broader. Alternatively, if the single data point were also at $x = 0$, then the prior on the intercept would be much less informative.[6] Note that the maximum likelihood (or least squares solution), as produced by `lm`, is not defined when there are fewer data points than parameters.

What if I have no data?
In that case the likelihood function is not defined (it does not even exist). The posterior is then just equal to the prior. Indeed, that's exactly what the prior is: our knowledge of the PDF over the parameters in the absence of data.[7]

E Other uses of integration in Bayesian inference

E.1 Marginal parameter distributions

Consider a two-parameter model, $\theta = (\theta_1, \theta_2)$. If we have determined the 2D posterior PDF, we may want to know the posterior PDF over just one parameter regardless of the value of the other parameter. We achieve this by integrating – also known as marginalizing – over the other parameter

$$P(\theta_1|D, M) = \int_{\theta_2} P(\theta_1, \theta_2|D, M) d\theta_2 \ . \tag{31}$$

[5] The R code will work with just one data point. This is not immediately obvious, however, because when a vector or matrix only has one element, then R (sometimes) automatically converts it into a scalar, which may then be an incompatible data type for some calculation (e.g. in a matrix multiplication). But it seems that this particular R code is working fine.

[6] In practice you might have little idea, a priori, of the intercept, making it hard to put a prior on it. But you might instead have some idea of the mean of the data (y). In that case we could re-parametrize the model and replace the parameter a_0 with this mean, on which we can more easily place a prior.

[7] The R code will not work properly if you set `data` to be `NULL`. This is because the code does not accommodate the likelihood being undefined (although that would be easy to fix). The code actually does return a value, but it is not what you want.

E.2 Expectation values (parameter estimation)

The expectation value of θ is defined as

$$< \theta >= E[\theta] = \int_\theta \theta P(\theta|D,M)d\theta \ . \tag{32}$$

This requires $P(\theta|D,M)$ to be normalized. If $P^u(\theta|D,M)$ is the unnormalized PDF, then

$$E[\theta] = \frac{1}{\int_\theta P^u(\theta|D,M)d\theta} \int_\theta \theta P^u(\theta|D,M)d\theta \ . \tag{33}$$

If θ is a vector (multiple parameters), then this involves a multidimensional integration.

E.3 Prediction

Suppose we have a two-dimensional data set, $D = (x, y)$, and have determined the posterior PDF over the parameters in model M. Given a new data point, x_p, what is the model prediction of y, call it y_p? The maximum likelihood approach to this would be to find the single "best" parameters of the model, then use these in the model equation to predict y, and finally attempt to propagate the errors in some (often dubious) way. The Bayesian approach is to find the entire posterior PDF over y_p, taking into account the fact that the uncertainty in the model parameters is reflected by the posterior PDF over these parameters. We achieve this by averaging over all possible sets of parameters of the model, with each "solution" weighed by how well it is supported by the data.

The Bayesian interpretation of the question is "what is $P(y_p|x_p, y, x, M)$"? This is the posterior PDF given the new data point, x_p, and all of the original data, (x, y), used to fit model M. The approach follows from the laws of probability, by writing down this PDF as a marginalization over the model parameters. First, note that

$$P(D) = P(y, x) = P(y|x)P(x) = P(y|x) \ . \tag{34}$$

The reason for this is that in this example the x values have no noise, and thus have probability one. In some sense only the $\{y\}$ are the data (things with noise) and the $\{x\}$ are like fixed (noise-free) parameters.

With this in mind, we now write down the posterior we want in terms of a marginalization. I will drop M for simplicity; it's implicit everywhere.

$$\begin{aligned} P(y_p|x_p, y, x) &= \int_\theta P(y_p, \theta|x_p, y, x)\ d\theta \\ &= \int_\theta P(y_p|\theta, x_p, y, x)\ P(\theta|x_p, y, x)\ d\theta \\ &= \int_\theta \underbrace{P(y_p|x_p, \theta)}_{\text{likelihood}}\ \underbrace{P(\theta|y, x)}_{\text{posterior}}\ d\theta \end{aligned} \tag{35}$$

where we have just used the general relation $P(y_p, \theta) = P(y_p|\theta)P(\theta)$ – with everything also conditioned on (x_p, y, x) – and then removed variables on the right of the "|" symbol when the variables on the left are *conditionally independent* of them. Thus for the first term:

$P(y_p|\theta, x_p, y, x)$ is independent of (x, y) once we specify the model parameters, because these parameters contain all the information. For the second term: our knowledge of θ is, by construction, independent of the new data point, and dependent only on the "old" data.

Equation 35 tells us that the predictive PDF over y_p is just the posterior-weighted average of the likelihood of the data. Another way of looking at this: for each θ we get a predictive distribution over y_p. We then average all of these, with a weighting factor (the posterior) which tells us how well that θ is supported by the data.