Markov Chain Monte Carlo - part II

Alberto Garfagnini

Università di Padova

AA 2020/2021 - Stat Lect. 11

Locations and Scale parameters in Priors

- imagine we have a model specifying the location $x_\circ$ of some quantity and we only know it lies between limits
- our Posterior should not depend on the origin of the coordinate system we adopt to solve the problem

▷ our Prior, $P(x_\circ)$ is invariant with respect to linear transformations $x_1 = x_\circ + c$. This implies:

$$P(x_\circ)dx_\circ = P(x_1)dx_1 = P(x_\circ + c)dx_\circ$$

▷ $P(x_\circ)$ is constant inside our domain and zero outside → a uniform prior $P(x_\circ) = 1/(x_{max} - x_{min})$

- we now want to infer the scale $w_\circ$, or size, of some quantity. We know nothing about it, only that it is positive

▷ our Prior, $P(w_\circ)$ is invariant with respect to scale units (it should be the same weather we express in meters or centimeters). It is therefore invariant with respect to transformation $w_1 = aw_\circ$:

$$P(w_\circ)dw_\circ = P(w_1)dw_1 = P(aw_\circ)a\ dw_\circ$$

- the requirement implies

$$P(w_\circ) \propto \frac{1}{w_\circ} \text{ and } P(a\ w_\circ) \propto \frac{1}{a\ w_\circ}$$

Scale parameters in Priors

- a Prior invariant to scale parameter is proportional to the inverse of the parameter itself. This is equivalent to the requirement

$$P(\log w_\circ) = \text{const}$$

- from simple change of variables:

$$P(x) \propto x^b \;\rightarrow\; P(\ln x) \propto x^{b+1}$$

- therefore, by setting $b = -1$, the only power law for $P(x)$ that produces a uniform distribution in $P(\ln x)$ is $P(x) \propto x^{-1}$
- this tell us that the probability between 1 and 10 is the same as between 10 and 100 ➜ reflecting properly our complete ignorance on the scale
- this type of Prior is often called a Jeffreys Prior

Applications

- location and scale parameters are common
- ▷ if we want to infer the mean of a Gaussian, we use a uniform distribution for the prior
- ▷ when inferring the variance, we should use a Jeffreys prior

$$P(\log \sigma^2) = \text{const} \;\Rightarrow\; P(\log \sigma) = \text{const}$$

Jeffreys Priors

- are based on a general approach of setting priors, introduced by Jeffreys [1] (see also Robert et al [2])
- his goal was to produce Priors which are invariant under reparametrizations
- if θ is a J dimensional vector of parameters, and $P(x \mid \theta)$ is the Likelihood
- Jeffreys prior, $P(\theta)$ is defined to be proportional to the square root of the determinant of the *Fisher information matrix* $\mathcal{I}(\theta)$

$$P(\theta) \propto \sqrt{\left|\mathcal{I}(\theta)\right|}$$

- the *Fisher information matrix* is a $J \times J$ matrix with elements equal to the expectation of the second derivative of the log likelihood:

$$\mathcal{I}_{j,k} = -E\left[\frac{\partial^2 P(x \mid \theta)}{\partial\theta_j \partial\theta_k}\right]$$

- it can be show that, under a change of variable $\theta = \theta(\psi)$, if we set $P(\theta) \propto \sqrt{|\mathcal{I}(\theta)|}$ the transformed prior is still proportional to the square root of information $P(\psi) \propto \sqrt{|\mathcal{I}(\psi)|}$ (the Jacobian in $P(\psi) = P(\theta)\left|d\theta/d\psi\right|$ cancels out)

[1] Jeffreys H., 1961, *Theory of Probability*, Cambridge University Press, 3rd edition
[2] Robert C.P. and Rousseau J., *Harold Jeffreys's Theory of Probability Revisited*, Statist. Sci. 24 (2009) 141, https://arxiv.org/pdf/0804.3173.pdf

Example: Jeffreys prior for a binomial likelihood

- let's evaluate the Jeffreys prior for the binomial likelihood

$$P\left(r \,\middle|\, p\ n\right) = \binom{n}{r} p^r (1-p)^{r-n}$$

- which has a single parameter, p
- taking the logarithm of the likelihood and differentiating twice with respect to p gives

$$\frac{\partial \ln P(r \,|\, p\ n)}{\partial p^2} = -\frac{r}{p^2} - \frac{n-r}{(1-p)^2}$$

- evaluating the expectation value for discrete distribution, we calculate the Fisher information as

$$\mathcal{I}(p) = -\sum_{r=0}^{n} \frac{\partial \ln P(r \,|\, p\ n)}{\partial p^2} P\left(r \middle|\, p\ n\right)$$

- using the definition $E[r] = \sum_r rP(\,|\, pn) = np$ we get

$$\mathcal{I}(p) = \frac{np}{p^2} + n\frac{n-p}{(1-p)^2} = \frac{n}{(1-p)^2}$$

- remembering that Jeffreys prior is proportional to the square root of the determinant of the Fisher Information Matrix $\mathcal{I}(\theta)$, gives

$$P(\theta) \propto \sqrt{\left|\mathcal{I}(\theta)\right|} = p^{-1/2}(1-p)^{-1/2}$$

- with n absorbed by the proportionality constant

a Beta with $(\alpha, \beta) = (1/2, 1/2)$, symmetric about $p = 1/2$ and going to ∞ for $p \to 0$ and $p \to 1$

R code for the next examples

- the examples discussed in the following are taken from

  Coryn A L. Bailer-Jones, *Practical Bayesian Inference*, Cambridge University Press, 2017, ISBN 978-1-316-64221-4

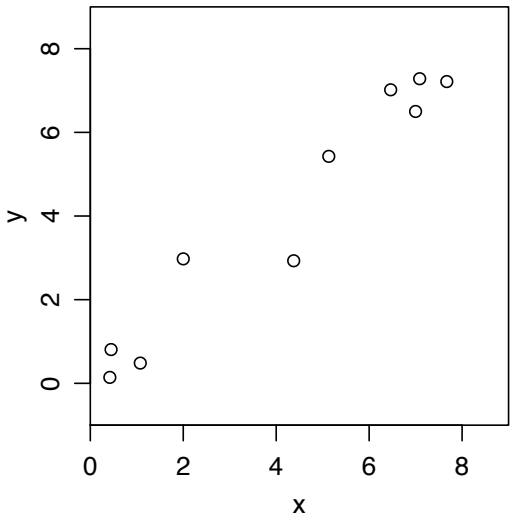
- the R code of the book can be dowloaded from
- https://github.com/ehalley/PBI/tree/master/PBI_scripts:

  - metropolis algorithm:
    https://github.com/ehalley/PBI/blob/master/PBI_scripts/metropolis.R
  - Linear model example main code:
    https://github.com/ehalley/PBI/blob/master/PBI_scripts/linearmodel_posterior.R
  - Linear Model Likelihood, Prior and Posterior probabilities:
    https://github.com/ehalley/PBI/blob/master/PBI_scripts/linearmodel_functions.R
  - quadratic model example main code:
    https://github.com/ehalley/PBI/blob/master/PBI_scripts/quadraticmodel_posterior.R
  - quadratic Model Likelihood, Prior and Posterior probabilities:
    https://github.com/ehalley/PBI/blob/master/PBI_scripts/quadraticmodel_functions.R

Fitting a straight line with noise

The Prior

- the Priors on α and $\log\sigma$ have no parameters
- the Prior on the intercept is driven by the data. A Gaussian distribution is assumed with $\mu = 0$ and a standard deviation $\sigma = 2$



R code

```
#
# parameters:
#   theta[1] -> b_0
#   theta[2] -> alpha
#   theta[3] -> log(sigma)

logprior.linearmodel <- function(theta) {
  b0Prior      <- dnorm(theta[1], mean=0, sd=2)
  alphaPrior   <- 1
  logysigPrior <- 1
  logPrior <- sum( log10(b0Prior),
                   log10(alphaPrior),
                   log10(logysigPrior) )
  return(logPrior)
}
```


Fitting a straight line with noise

The Likelihood

- the logLikelihood is

$$P\left(y_j \mid x_j, \theta, M\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{\left(y_j - f(x_j; b_0, b_1)\right)^2}{2\sigma^2}\right]$$

R code

```
# parameters:
#   theta[1] -> b_0
#   theta[2] -> alpha
#   theta[3] -> log(sigma)

loglike.linearmodel <- function(theta, obsdata) {
  # convert alpha to b_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$x)) )
  # Dimensions in mixed vector/matrix products:
  #                          [Ndat] = [P] %*% [P x Ndat]
  logLike <- (1/log(10))*sum(dnorm(modPred - obsdata$y, mean=0,
                                   sd=theta[3], log=TRUE) )
  return(logLike)
}
```

Fitting a straight line with noise

The Posterior distribution

- the Posterior is simply given by the product of the Likelihood and Prior

$$P\left(\theta \,\middle|\, D\right) \propto P\left(D \,\middle|\, \theta\right) \times P\left(\theta\right)$$

- the function is interfaced to the `metropolis()` function giving a vector with logPrior and logLikelihood values

R code

```
# Return c(log10(prior), log10(likelihood)) (each generally unnormalized)
# of the linear model
logpost.linearmodel <- function(theta, obsdata) {
  logprior <- logprior.linearmodel(theta)
  if(is.finite(logprior)) { # only evaluate model if parameters are sensible
    return( c(logprior, loglike.linearmodel(theta, obsdata)) )
   } else {
    return( c(-Inf, -Inf) )
   }
}
```


Initializing and running the MCMC process

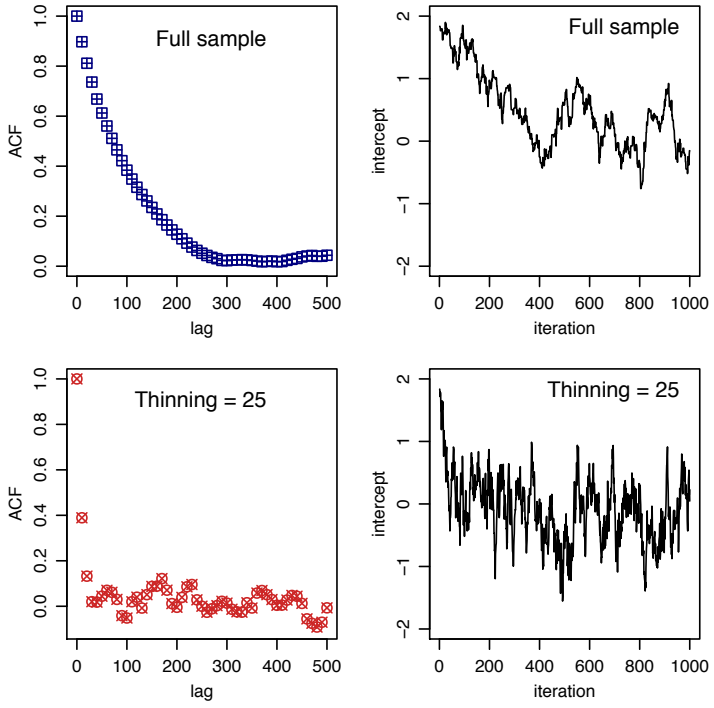
- the starting values for the Markov Chain are $b_0 = 2$, $\alpha = \pi/8$ and $\log_{10}\sigma = \log_{10}(3)$
- the step size for the evolution of the chain are 0.1, 0.02 and 0.1 (respectively for b_0, α and $\log\sigma$)

R code

```
# markov Chain initial values
thetaInit <- c(2, pi/8, log10(3))

# Markov Chain step sizes
sampleCov <- diag(c(0.1, 0.02, 0.1)^2)

set.seed(150)
allSamp <- metrop(func=logpost.linearmodel, thetaInit=thetaInit,
                  Nburnin=0, Nsamp=5e4,
                  sampleCov=sampleCov, verbose=1e3,
                  obsdata=obsdata)

  1000 of      0 +  50000  0.5826
  2000 of      0 +  50000  0.5775
  3000 of      0 +  50000  0.5689
...
 48000 of      0 +  50000  0.5629
 49000 of      0 +  50000  0.5624
 50000 of      0 +  50000  0.5627
```

Analyzing the Markov Chain

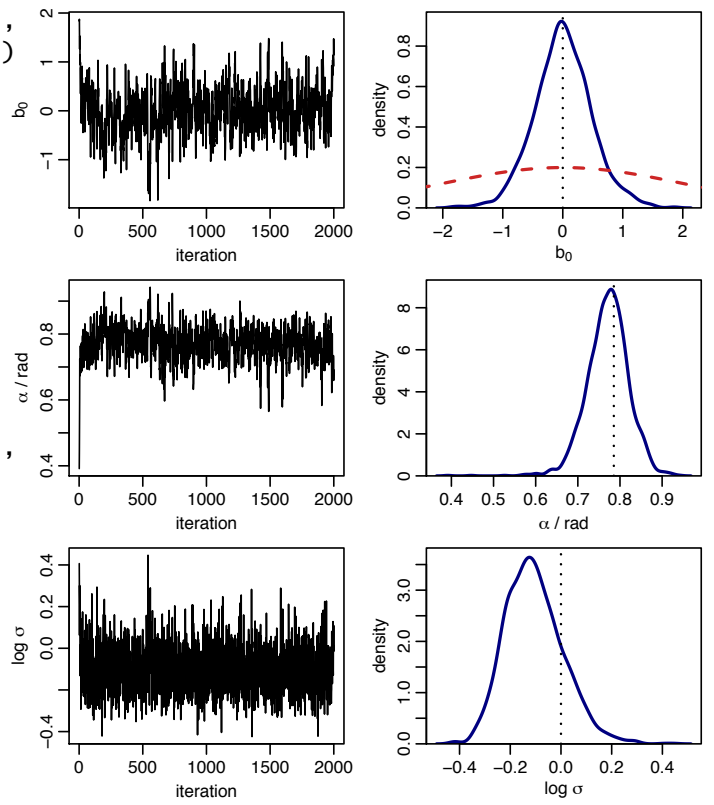
- the unormalized Posterior has been used in the MCMC, the normalization is not needed since samples are drawn with the same relative frequency, independently of the normalization
- in contrast to the Posterior, the Likelihood has to be normalized since it is a pdf over the data and therefore its normalization constant is, in general, a function of the parameters we are sampling
- data are now reduced (thinning = 25) to reduce auto-correlation in the chain
- results and plots are obtained for the last 2k events in the chain




```
allSamp <- metrop(func=logpost.linearmodel, thetaInit=thetaInit, ...)

thinSel  <- seq(from=1, to=nrow(allSamp), by=25) # thin by factor 25

postSamp <- allSamp[thinSel,]
```


Marginal Posterior pdfs

```
parname <- c(expression(b[0]),
    expression(paste(alpha, "␣/␣rad")),
    expression(paste(log, "␣", sigma)))

nr <- nrow(postSamp)
is <- nr-2000

for (j in 3:5) {

  plot(is:nr, postSamp[is:nr,j],
       type="l",
       xlab="iteration",
       ylab=parname[j-2])

  postDen <- density(postSamp[is:nr,j],
                     n=2^10)
  plot(postDen$x, postDen$y,
       col='navy', lwd = 2,
       xlab=parname[j-2],
       ylab="density")

  abline(v=thetaTrue[j-2],
         lwd=1.5, lty=3)
}
```



Posterior parameters estimation

- the joint posterior distribution is the three-dimensional distribution over the MCMC samples, and the one-dimensional marginalized distributions are obtained by making a density estimation of the samples for each parameter
- we evaluate the maximum or mean of the posterior as a single best estimate: the maximum of the posterior is not the peak in each 1-dim pdf, but of the 3-dim pdf

```
# the maximum of the sum of the log(Prior) and log(Likelihood)
posMAP    <- which.max(postSamp[,1] + postSamp[,2])
thetaMAP  <- postSamp[posMAP, 3:5]
thetaMean <- apply(postSamp[,3:5], 2, mean) # Monte Carlo integration
cov(postSamp[, 3:5]) # covariance
cor(postSamp[, 3:5]) # correlation
```

- we get:

$$(b_0, \alpha, \log\sigma) = (0.036, 0.77, -0.19)$$

- if we want to find the mean of the posterior over the original model parameters - (b_0, b_1, σ) - we must transform the individual samples first and then compute the statistic (and not vice versa)

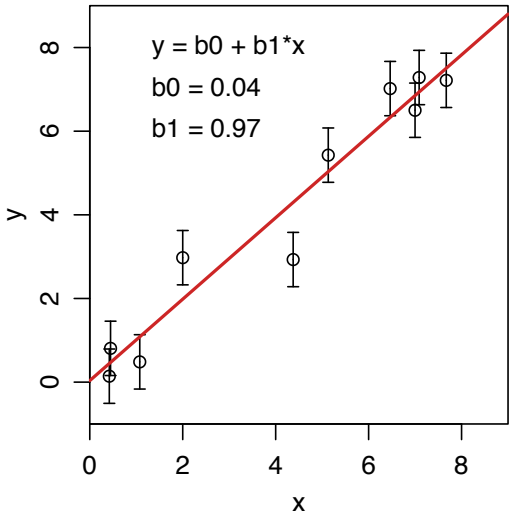
```
mean(tan(postSamp[,4])) # transform alpha to b_1
mean(10^(postSamp[,5])) # transform log10(sigma) to sigma
```

- we get:

$$(b_0, b_1, \sigma) = (0.036, 0.98, 0.81)$$

Linear Fit results

```
plotCI(obsdata$x, obsdata$y,
       xlim=c(0,9), ylim=c(-1,9),
       xaxs="i", yaxs="i",
       xlab="x", ylab="y",
       uiw=10^thetaMAP[3], gap=0)

b0 <- thetaMAP[1]
b1 <- tan(thetaMAP[2])

abline(a=b0, b=b1, lw=2, col='firebrick3')
```



- the R `plotCI()` function is used to plot error bars and Confidence Intervals (in package `gplots`)
- given a set of *x* and *y* values and interval width or upper and lower bounds, it plots the points with error bars
  - `uiw` : width of the upper or right error bar. Set to `NA` or to `NULL` to omit upper bars
  - `liw` : width of the lower or left error bar. Defaults to same value as uiw.

Posterior Predictive distribution

- once we have inferred the "best" values for the model parameters, we can use them to predict the value of the Model y_p at any specific value x_p
- the rules of probability lead us to incorporate uncertainties in parameters by marginalizing over them
- we define a posterior predictive distribution

$$P\left(y_p \mid x_p, D\right) = \int P\left(y_p \mid x_p, \theta\right) P\left(\theta \mid D\right) d\theta$$

- the distribution can be evaluated in two ways

Direct method (accurate, but slow)

- is based on evaluating $P(y_p \mid x_p, D)$ over a grid $\{y_p\}$
- at a fixed value of y_p we take our set of N_s posterior samples $\{\theta_l\}$ (obtained by MCMC), calculate the likelihood at each of these, and then average these likelihoods, i.e.

$$P\left(y_p \mid x_p D\right) \sim \frac{1}{N_s} \sum_{j=1}^{N_s} P\left(y_p \mid x_p \theta_j\right)$$

- the posterior predictive distribution is a posterior-weighted average of the predictions (the likelihood) made at each θ

Posterior Predictive distribution

Indirect method

- is based on sampling the joint distribution $P(y_p, \theta \mid x_p, D)$ directly, and marginalizing it over θ
- we can factorize the joint distribution

$$P\left(y_p \theta \mid x_p D\right) = P\left(y_p \mid x_p \theta\right) P\left(\theta \mid D\right)$$

- each of the two pdfs on the right side can be represented by samples drawn from them. The second term is the posterior pdf; we already obtained the set of samples $\{\theta_j\}$ from this with the MCMC. The first term is the likelihood
- As the likelihood is a uni-variate Gaussian, it may be sampled using a standard function. Its mean is the evaluation of the straight line at (b_0, b_1), and its standard deviation is σ
- the R code is:

```
likeSamp <- rnorm(n=length(modPred), mean=modPred, sd=10^postSamp[,5])
```

- where `modPred` (of length N_s) is the evaluations of the straight line at the posterior samples. We now have samples of θ and y_p
- we marginalize their joint distribution simply by ignoring the θ, to give the required distribution $P(y_p \mid x_p D)$

Posterior Predictive distribution - example

```
xnew <- 6

# Evaluate generative model at posterior samples (from MCMC).
# Dimensions in matrix multiplication: [Nsamp x 1] = [Nsamp x P] %*% [P x 1]
modPred <- cbind(postSamp[,3], tan(postSamp[,4])) %*% t(cbind(1,xnew))

# ---- Direct method ----
# ycand must span full range of likelihood and posterior
dy    <- 0.01
ymid  <- thetaMAP[1] + xnew*tan(thetaMAP[2]) # to center choice of ycand

ycand <- seq(ymid-10, ymid+10, dy) # uniform grid of y with step size dy
ycandPDF <- vector(mode="numeric", length=length(ycand))

for(k in 1:length(ycand)) {
  like <- dnorm(ycand[k], mean=modPred, sd=10^postSamp[,5]) # [Nsamp x 1]
  ycandPDF[k] <- mean(like) # integration by rectangle rule. Gives a scalar
}

# Note that ycandPDF[k] is normalized, i.e. sum(dy*ycandPDF)=1.
# Find peak and approximate confidence intervals at 1sigma on either side
peak.ind  <- which.max(ycandPDF)

lower.ind <- max( which(cumsum(dy*ycandPDF) < pnorm(-1)) )
upper.ind <- min( which(cumsum(dy*ycandPDF) > pnorm(+1)) )
yPredDirect <- ycand[c(peak.ind, lower.ind, upper.ind)]
```


Posterior Predictive distribution - example

```
xnew <- 6

# Evaluate generative model at posterior samples (from MCMC).
# Dimensions in matrix multiplication: [Nsamp x 1] = [Nsamp x P] %*% [P x 1]
modPred <- cbind(postSamp[,3], tan(postSamp[,4])) %*% t(cbind(1,xnew))

# ---- Indirect method ----
likeSamp <- rnorm(n=length(modPred), mean=modPred, sd=10^postSamp[,5])
likeDen  <- density(likeSamp, n=2^10)

# Find peak and confidence intervals
yPredIndirect <- c(likeDen$x[which.max(likeDen$y)], quantile(likeSamp,
                          probs=c(pnorm(-1), pnorm(+1)), names=FALSE))
```

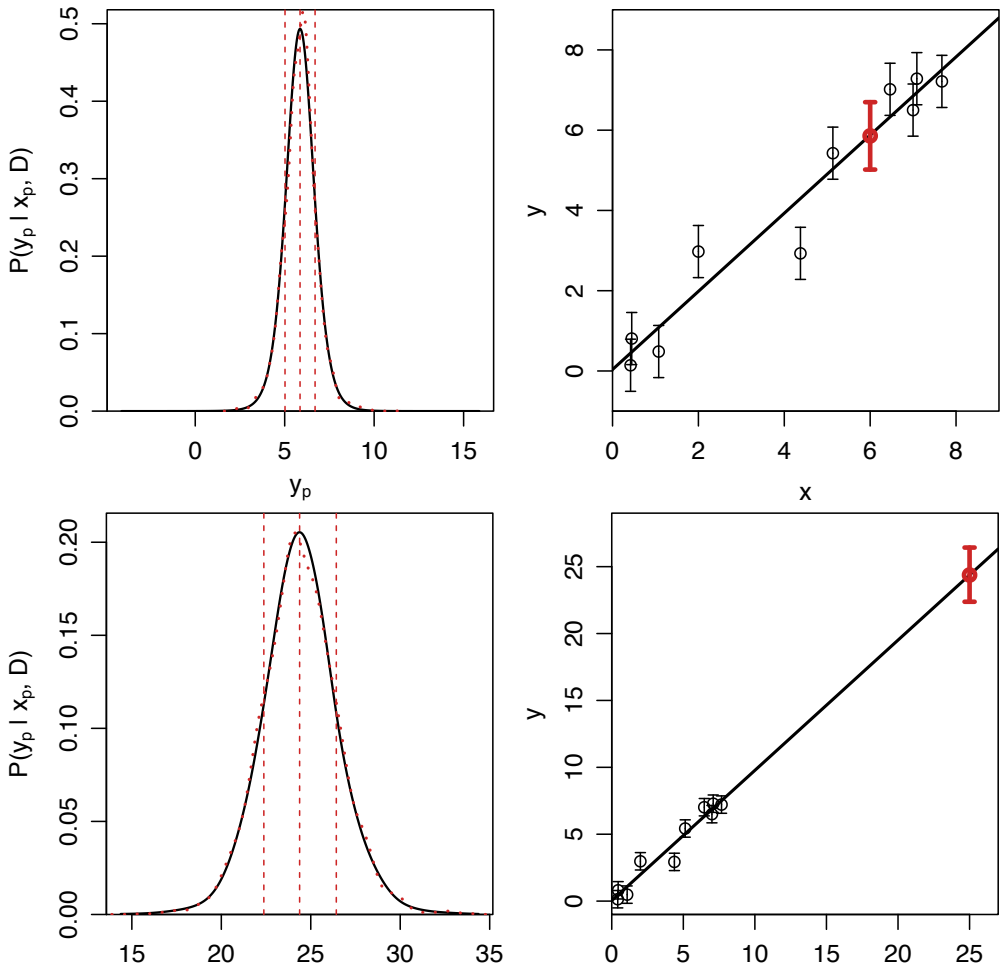
Posterior Predictive distribution - example

```
plot(ycand, ycandPDF, type="l", lwd=1.5,
     ylim=1.05*c(0,max(ycandPDF)), xlab=expression(y[p]),
     ylab=expression(paste("P(", y[p], "␣|␣", x[p], ",␣D)")))

abline(v=yPredDirect, col='firebrick3', lty=2)

# overplot result from the indirect method
lines(likeDen$x, likeDen$y, col='firebrick3', type="l", lty=3, lwd=2)
```

```
> rbind(yPredDirect, yPredIndirect)
                  [,1]     [,2]     [,3]
yPredDirect   5.858070 5.018070 6.698070
yPredIndirect 5.876795 5.037817 6.665148
```

```
# Overplot direct prediction with original data and the MAP model
plotCI(obsdata$x, obsdata$y, xlim=xlim, ylim=ylim,
       uiw=10^thetaMAP[3], gap=0, xlab="x", ylab="y")
abline(a=thetaMAP[1], b=tan(thetaMAP[2]), lwd=2) # MAP  model

plotCI(xnew, ycand[peak.ind], li=ycand[lower.ind], ui=ycand[upper.ind],
       gap=0, add=TRUE, lwd=3, col='firebrick3')
```


Linear Fit Prediction results

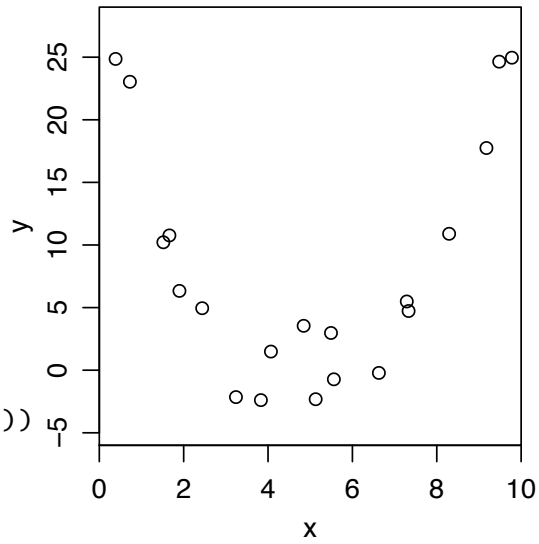


Fitting a quadratic curve with noise

- we have a new set of data we want to fit to a generative model

$$y = f(x) + \epsilon \text{ with } f(x) = b_0 + b_1 x + b_2 x^2$$

- they have been drawn at fixed x values from a straight line with $(b_0, b_1, b_2) = (25, -10, 1)$, to which zero mean Gaussian noise with $\sigma = 2$ has been added.




```
Ndat <- 20
xra <- c(0,10)
x <- sort(runif(Ndat, min=xra[1], max=xra[2]))
sigTrue <- 2

# 1 x P vector: coefficients,
#       b_p, of sum_{p=0} b_p*x^p
modMat <- c(25, -10, 1)
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 do vector addition

# finally, convert to a vector
y <- drop(y)
```


Fitting a quadratic curve with noise

The Prior

- a Gaussian prior is used on b_0, $b_0 \sim \mathcal{N}(0, 10)$
- b_1 is transformed to $\alpha = \arctan b_1$, and a uniform prior is used $\alpha \sim \mathcal{U}(0, 2\pi)$
- a Gaussian prior is used on b_2, $b_2 \sim \mathcal{N}(0, 5)$
- σ is transformed to $\log \sigma$ and an improper uniform prior is used

R code

```
#
# parameters:
#   theta[1] -> b_0
#   theta[2] -> alpha
#   theta[3] -> b_2
#   theta[4] -> log(sigma)

logprior.quadraticmodel <- function(theta) {
  b0Prior      <- dnorm(theta[1], mean=0, sd=10)
  alphaPrior   <- 1
  b2Prior      <- dnorm(theta[3], mean=0, sd=5)
  logysigPrior <- 1
  logPrior <- sum(log10(b0Prior), log10(alphaPrior),
                  log10(b2Prior), log10(logysigPrior) )
  return(logPrior)
}
```

Fitting a quadratic curve with noise

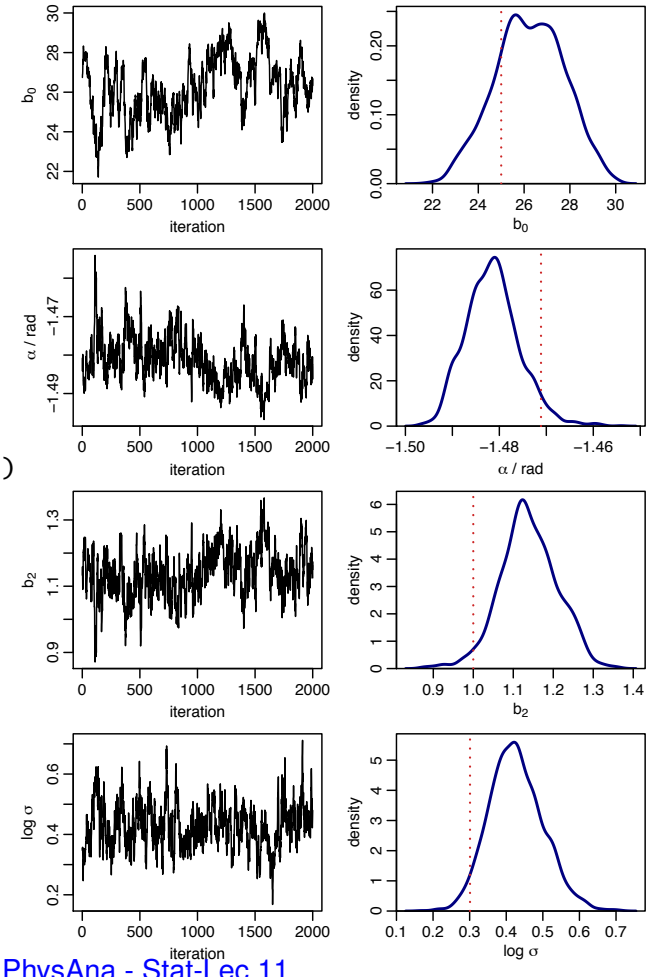
- the parameters step sizes (Gaussian standard deviations) are chosen as $(b_0, \alpha, b_2, \log\sigma) = (0.1, 0.01, 0.01,$ and $0.01)$
- as starting point any value can be in principle chosen, but it could take a large number of steps to locate the high density region of the posterior. Therefore a classical approach (`lm()` function) has been used
- to achieve good chains more iterations than in the straight line problem are needed (higher complexity of the model)
- after a burn-in of 20 k iterations, further 200 k iterations are sampled
- to reduce the auto-correlation, a thinning factor of 100, is used

R code

```
sampleCov <- diag(c(0.1, 0.01, 0.01, 0.01)^2)
thetaInit <- c(27.4, atan(-11.7), 1.18, log10(2.4))
set.seed(250)
allSamp <- metrop(func=logpost.quadraticmodel, thetaInit=thetaInit,
                  Nburnin=2e4, Nsamp=2e5,
                  sampleCov=sampleCov, verbose=2e3, obsdata=obsdata)
  2000 of  20000 +  2e+05  0.0729
  4000 of  20000 +  2e+05  0.0940
...
216000 of  20000 +  2e+05  0.1039
218000 of  20000 +  2e+05  0.1039
220000 of  20000 +  2e+05  0.1041
```

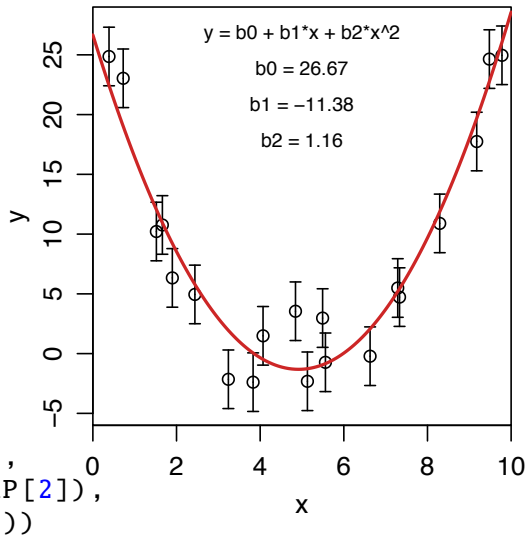

Quadratic curve : posterior pdfs

```
parnames <- c(expression(b[0]),
     expression(paste(alpha, "␣/␣rad")),
     expression(b[2]),
     expression(paste(log, "␣", sigma)))

for (j in 3:6) {

  plot(1:nrow(postSamp), postSamp[,j],
     type="l", xlab="iteration",
     ylab=parnames[j-2])

  postDen <- density(postSamp[,j], n=2^10)

  plot(postDen$x, postDen$y, type="l",
     lwd=2, yaxs="i", col='navy',
     ylim=1.05*c(0,max(postDen$y)),
     xlab=parnames[j-2], ylab="density")

  abline(v=thetaTrue[j-2],
       lwd=1.5, lty=3, col='firebrick3')
}
```



Quadratic Fit results

```
plotCI(obsdata$x, obsdata$y,
       xlim=xrange, ylim=c(-6,29),
       xaxs="i", yaxs="i",
       xlab="x", ylab="y",
       uiw=10^thetaMAP[4], gap=0)

b0 <- thetaMAP[1]
b1 <- tan(thetaMAP[2])
b2 <- thetaMAP[3]

ysamp <- cbind(1,
          xsamp,
          xsamp^2) %*% as.matrix(c(thetaMAP[1],
                                   tan(thetaMAP[2]),
                                   thetaMAP[3]))
lines(xsamp, drop(ysamp), lwd=2, col='firebrick3')
```



- the R `plotCI()` function is used to plot error bars and Credibility Intervals
- given a set of *x* and *y* values and interval width or upper and lower bounds, it plots the points with error bars
- `uiw` : width of the upper or right error bar. Set to `NA` or to `NULL` to omit upper bars
- `liw` : width of the lower or left error bar. Defaults to same value as uiw.

Appendix: scalar product of two vectors with R

- given two vectors, $\mathbf{a} = (1,2,3)$ and $\mathbf{b} = (2,3,4)$, we want to evaluate the scalar product $\mathbf{a} \cdot \mathbf{b} = 20$

```
# In R this will produce a matrix
> 1:3 %*% 2:4
     [,1]
[1,]   20

> class(1:3 %*% 2:4)
[1] "matrix"

# The drop() function deletes the dimensions of an array
#  which have only one level

> drop(1:3 %*% 2:4)
[1] 20

> class(drop(1:3 %*% 2:4))
[1] "numeric"
```