Markov Chain Monte Carlo - part I

Alberto Garfagnini

Università di Padova

AA 2023/2024 - Stat Lect. 9

Integration in Bayesian inference

- there are four cases, in Bayesian inference, that require integration:

Marginalization

- given a two dimensional (or higher) posterior pdf over the parameters (θ_1, θ_2), we can determine the posterior over just one parameter by integration

$$P\left(\theta_1 \mid D, M\right) = \int P\left(\theta_1, \theta_2 \mid D, M\right) d\theta_2$$

Expectation values

- this is defined as

$$E\left[\theta\right] = \int \theta P\left(\theta \mid D, M\right) d\theta$$

- if the posterior pdf, $P\left(\theta \mid D, M\right)$ is normalized, or by

$$E\left[\theta\right] = \frac{1}{Z^*} \int \theta P^*\left(\theta \mid D, M\right) d\theta$$

- with $Z^* = \int P^*\left(\theta \mid D, M\right) d\theta$ when $P^*\left(\theta \mid D, M\right)$ is the unnormalized posterior pdf distribution

Integration in Bayesian inference

Model Comparison

- for comparing models, we need to evaluate the *evidence* of Bayesian theorem

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

Data prediction

- given a data set $D = \{y_j\}$ obtained at fixed $\{x_j\}$, we have determined the posterior pdf over the model parameters. Now, we are looking for the prediction y_p over a new point x_p
- the Bayesian approach is to find the posterior pdf over y_p, i.e.

$$P\left(y_p \mid x_p, D, M\right) \text{ , a posterior predicative distribution}$$

$$\begin{aligned} P\left(y_p \mid x_p, D, M\right) &= \int P\left(y_p \theta \mid x_p, D, M\right) d\theta \\ &= \int P\left(y_p \mid x_p, \theta D, M\right) P\left(\theta \mid x_p, D, M\right) d\theta \\ &= \int P\left(y_p \mid x_p, \theta M\right) P\left(\theta \mid D, M\right) d\theta \end{aligned}$$

- notice that $P\left(y_p \mid x_p, \theta D, M\right) = P\left(y_p \mid x_p, \theta M\right)$ since it is independent of our data set D, once we have determined the model parameters
- in a similar way, $P\left(\theta \mid x_p, D, M\right) = P\left(\theta \mid DM\right)$ because our knowledge of the model parameters does not depend on where we want to make a prediction

How to compute the posterior distribution

1) The easy way

- select a Prior distribution function which is conjugate to the Likelihood function:

  Examples:

Prior	Likelihood
Beta	Bernoulli / Binomial
Gamma	Poisson
Beta	Geometric
Normal	Normal (with known σ^2)
Inverse Gamma	Normal (with known μ)

2) The brute force approach

- define the Prior on a dense grid of points spacing the range of your parameter θ
- compute the Posterior numerically by summing the product Likelihood $\times$ Prior on the grid

1) works well in a very limited number of cases
2) has severe limitations (memory, computation time) for multiparameter spaces

The Markov Chain Monte Carlo

- is a very powerful, generic method, for *approximately* generating samples from any Posterior distribution
- the Prior distribution, $P(\theta)$, is specified by a function that can be easily evaluated (analytically or numerically)
- the Likelihood function, $P(D \mid \theta)$, can be computed for any values of D and θ
- the method demands that Prior and Likelihood can be computed up to a multiplicative constant ➜ it is not required to compute the Evidence (i.e. the denominator of the Bayes' theorem)
- an approximation of the Posterior distribution, $P(\theta \mid D)$, is produced
- since the Posterior distribution is estimated by randomly generating a large samples from it, it is called a Monte Carlo method (by analogy to 'standard' Monte Carlo methods)

The Island example

- 10 islands of different size form an archipelago
- the number of people living in each island is proportional to its area
- a doctor is continuosly traveling among the islands and she wants to remain in an island for a time proportional to that island's population
- at the beginning of each week, the doctor can

1) stay on the current island
2) move to an adjacent island

- to simplify the problem,
  - we label the island from 1 to 10 and place them on a circle
  - The number of inhabitants is equal to the island label (in some arbitrary unit)

The Island example : the algorithm

- at the beginning of each week, the doctor
  - flips a coin to decide on which island she can go: HEAD ➜ East, TAIL ➜ West
  - if the proposed island has a larger population with respect to her current position, she goes to that island
  - if the proposed island has a smaller population, the probability of moving there is proportional to the ratio of populations

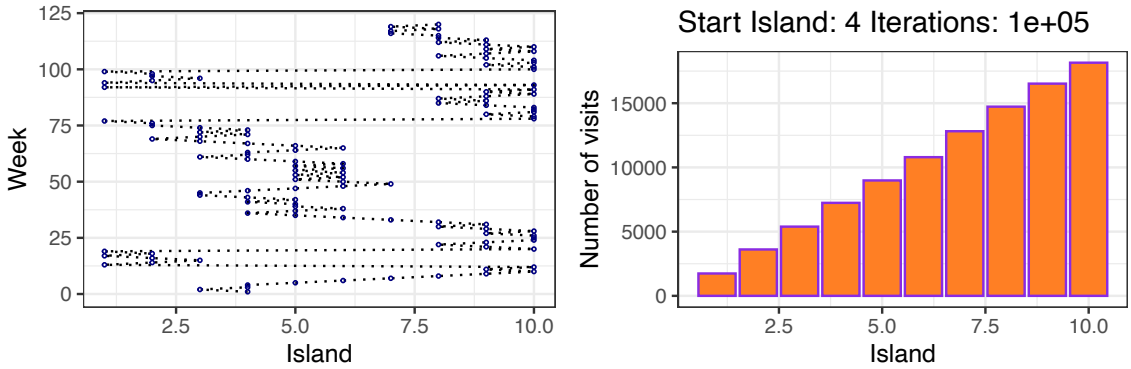
$$P_{\text{proposed}}/P_{\text{current}}$$

- the moving probability is

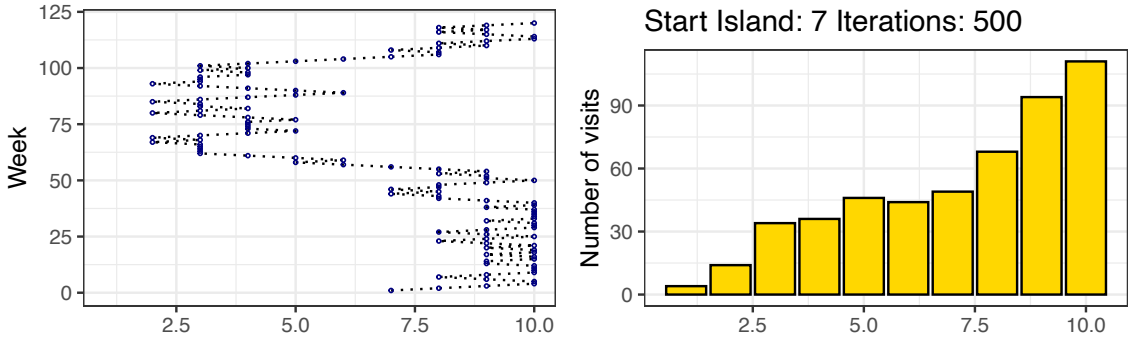
$$P_{move} = \text{MIN}\left(\frac{P\left(\theta_{proposed}\right)}{P\left(\theta_{current}\right)}, 1\right)$$

The Island example : test runs

- a long run (10^5 weeks) has been performed:
  - adjacent islands are visited proportionally to their population size (i.e. target distribution)



- a shorter run (500 weeks) has been done:
  - the obtained distributions is a bad approximation of the target distribution



Markov Chain Monte Carlo

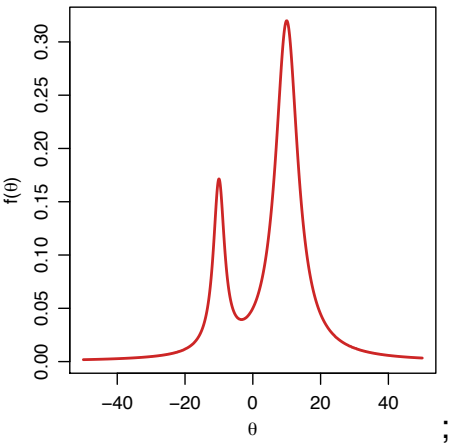
- is a very powerful, generic method, for *approximately* generating samples from any arbitrary distribution
- the MCMC method is due to Metropolis et al [1] and was motivated by computational methods in statistical physics
- it uses the idea of generating a Markov chain whose limiting distribution is equal to to desired target distribution
- many modifications and enhancement were proposed, most notably the one of Hastings [2]
- today, any approach the produces an ergodic Markov chain whose stationary distribution is the target distribution is referred to as MCMC or Markov chain sampling
- the most prominent MCMC algorithms are the Metropolis-Hastings and the Gibbs sampler

[1] N. Metropolis, et al., *Equations of state calculations by fast computing machines*, J Chem Phys, **21**, (1953), 1087

[2] W.K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika 57, (1970), 92

Markov Chain Monte Carlo

- let's assume we want to sample from a complex distribution $f(\theta)$
- the 'standard' Monte Carlo methods we have discussed would not be very efficient since they would 'waste time' in sampling $f(\theta)$ in regions where the value is small
- we would like to make samples in the region where $f(\theta)$ is high, but keeping the full sample still representative of $f(\theta)$
- this can be done if we relax the constraint of drawing samples independently



;

- the principle behind a Markov Chain Monte Carlo is to setup a random walk over the parameter space which explores the regions of high probability density of $f(\theta)$
- the random walk is done through a Markov Chain:

  a random process in which the probability of evolving from a state $\theta_t \rightarrow \theta_{t+1}$ is defined by a transition probability $Q(\theta_{t+1} \mid \theta_t)$ which does not depend on the previous states
- this is also called a memory-less process

Markov Chain Monte Carlo Algorithm

- as with the rejection-sampling, the MCMC uses a proposal distribution $Q(s|\theta)$
- it is a distribution from which we can easily draw a candidate sample s for the next point in the chain, θ_{t+1}, given the current parameter value θ_t

Algorithm

(0) initialize the chain at some value

(1) draw a random sample from the distribution $Q(s|\theta)$

This is often a multivariate Gaussian where θ_t is the mean and the covariance matrix specifies the the typical size of steps in the chain in each dimension of the parameters θ

(2) decide whether to accept or not the new candidate sample on the basis of the Metropolis ratio

$$\rho = \frac{f(s)}{f(\theta_t)} \frac{Q(\theta_t \mid s)}{Q(s \mid \theta_t)}$$

if $\rho \geq 1$ the new candidate is accepted and $\theta_{t+1} = s$

if $\rho < 1$ we only accept it with probability ρ:

▷ draw $u \sim \mathcal{U}(0,1)$ and set $\theta_{t+1} = s$ only if $u \leq \rho$

if s is not accepted, we set $\theta_{t+1} = \theta_t$, i.e. the existing sample in the chain is repeated

Markov Chain Monte Carlo Algorithm

- the algorithm goes on for a certain number of iterations
- the typical number of steps required to have a good sampling depends on the problem. Typical values are between 10^4 and 10^6
- if a symmetric proposal distribution function is used (like a Gaussian) the term $Q(\theta_t \mid s)/Q(s \mid \theta_t)$ in the definition of ρ is always unity
- this is referred to as the Metropolis algorithm
- depending on the initialization of the chain, the initial samples may not be representative of it and they should be discarded
- the discarded initial samples are called the burn-in
- with a good initialization the burn-in may only be a few percent of the chain

MCMC Exercise 1

MCMC sampling : 1-dim

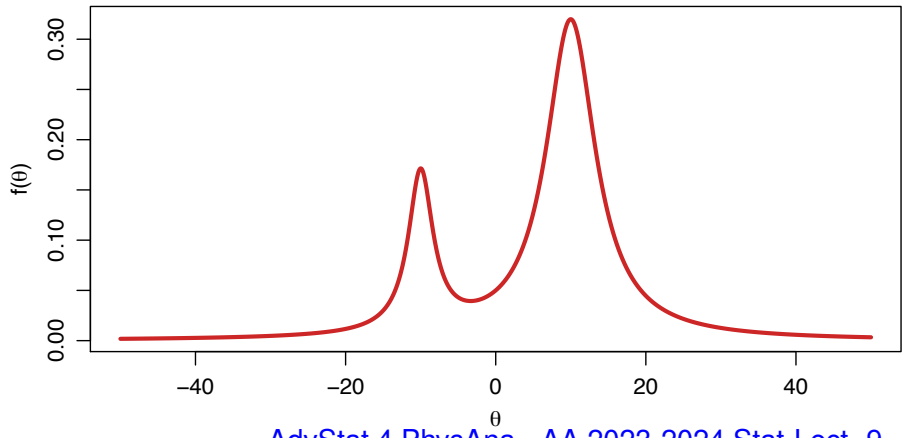
- given a Cauchy distribution function

$$\text{Cauchy}(x \mid x_\circ, \gamma) = \frac{1}{\pi\gamma} \frac{1}{1 + ((x - x_\circ)/\gamma)^2}$$

- where γ and $x_\circ$ are called scale and location parameters, let's consider

$$f(x) = \text{Cauchy}(x_\circ = -10, \gamma = 2) + 4 * \text{Cauchy}(x_\circ = 10, \gamma = 4)$$

- we want to sample from the distribution using a MCMC algorithm



MCMC Metropolis R code (1dim functions)

```
# Parameters:
# func : a function whose first argument is a real vector of parameters
#        func returns a log10 of the likelihood function
# theta.init : the initial value of the Markov Chain (and of func)
# n.sample: number of required samples
# sigma : standar deviation of the gaussian MCMC sampling pdf
metropolis.1dim <- function(func, theta.init, n.sample, sigma) {
  theta.cur <- theta.init
  func.Cur <- func(theta.cur)
  func.Samp <- matrix(data=NA, nrow=n.sample, ncol=2+1)
  n.accept <- 0
  rate.accept <- 0.0

  for (n in 1:n.sample) {

    theta.prop <- rnorm(n=1, mean = theta.cur, sigma)
    func.Prop <- func(theta.prop)
    logMR <- func.Prop - func.Cur  # Log10 of the Metropolis ratio

    if ( logMR>=0 || logMR>log10(runif(1)) ) {
       theta.cur <- theta.prop
       func.Cur  <- func.Prop
       n.accept <- n.accept + 1
    }
    func.Samp[n, 1] <- func.Cur
    func.Samp[n, 2] <- theta.cur
  }
  return(func.Samp)
}
```

MCMC Metropolis R code (1dim functions)

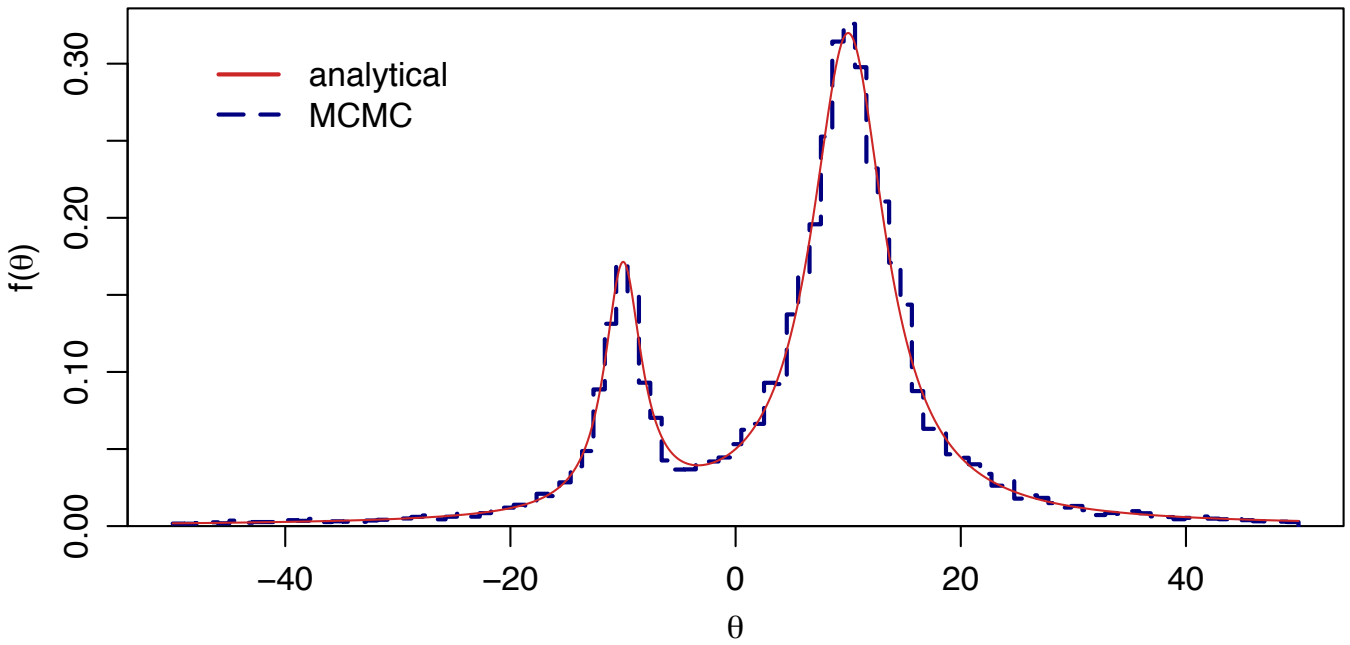
```
#
# Our test function
#
testfunc <- function(theta) {
   return(dcauchy(theta, -10, 2,) + 4*dcauchy(theta, 10, 4))
}

#
# - interface for the metropolis function, gets the log10 of test function
testfunc.metropolis <- function(theta) {
  return(log10(testfunc(theta)))
}

### Running parameters
theta.init <- -5
sample.sig <- 10
n.sample   <- 10^5
demo       <- TRUE

set.seed(20190513)
chain <- metropolis.1dim(func=testfunc.metropolis,
                         theta.init = theta.init,
                         n.sample = n.sample,
                         sigma = sample.sig^2, demo)
```


MCMC Metropolis R code results

```
#
# Here are the plots
#
par(mfrow=c(2,2), mgp=c(2,0.8,0), mar=c(3.5,3.5,1,1), oma=0.1*c(1,1,1,1))

x <- seq(-50, 50, length.out=10^4)
y <- testfunc(x)
ymax <- 1.05 * max(y)
plot(x, y, ylim=c(0,max(y)*1.10),
     type='l', lwd=2, col='firebrick3',
     xlab=expression(theta), ylab=expression(paste('f(',theta,')', sep='')))

plot(x, y, type="n", yaxs="i", ylim=c(0, 1.05*max(y)),
     xlab=expression(theta), ylab=expression(paste('f(',theta,')', sep='')))
sa <- which(chain$func.Samp[,2]>=min(x) & chain$func.Samp[,2]<=max(x))
hist <- hist(chain$func.Samp[sa,2], breaks=seq(from=min(x), to=max(x),
                                    length.out=100), plot=FALSE)
Zhist <- sum(hist$counts)*diff(range(hist$breaks))/(length(hist$counts))
lines(hist$breaks, c(hist$counts*Zfunc/Zhist,0),
      col='navy', type="s", lwd=2, lty=5)
lines(x, y, col='firebrick3', lwd=1, lty=1)

leg.labels = c('analytical', 'MCMC')
leg.ltype = c(1, 5)
leg.colors = c('firebrick3','navy')
legend("topleft", inset=.05, bty='n',
       legend = leg.labels, lty=leg.ltype, col=leg.colors,
       lwd = 2)
```

MCMC plot results

- the histogram reproduces the behavior of our test function
- the acceptance rate of the samples is only 15.84%
- by changing the σ of the proposal distribution function, we get a better rate (40.81% for $\sigma = 5$)



MCMC chain analysis

- the proposed algorithm works in principle, but it may not produce a representative sample ➜ it is important to inspect the chain and check its property
- open points in the recipe are: the covariance matrix of the proposal distribution, how long should the burn-in period be, how many iterations are expected before convergence, etc.
- one of the simplest ways to check whether the chain has reached a steady state is to rerun the sampling several times, with different starting points ➜ all chains should converge to the same region of parameter space
- various metrics exist. One way is to compute an auto-correlation function of the elements of the chain:
- given a chain of length N, at lag h, from the definition of covariance it follows:

$$\mathrm{ACF}(h) = \frac{\frac{1}{N-h}\sum_{t=1}^{N-h}(\theta_t - \overline{\theta})(\theta_{t+h} - \overline{\theta})}{\frac{1}{N-1}\sum_{t=1}^{N}(\theta_t - \overline{\theta})^2}$$

- where θ_{t+h} is the chain offset by h steps

➜ $\mathrm{ACF}(h)$ measures how closely the chain is correlated with itself h steps later

MCMC chain analysis : the coda R package

CODA

- provides functions for summarizing and plotting the output from Markov Chain Monte Carlo (MCMC) simulations, as well as diagnostic tests of convergence to the equilibrium distribution of the Markov chain

➜ https://cran.r-project.org/web/packages/coda/coda.pdf

- the function `mcmc` and `as.mcmc` are used to create a Markov Chain Monte Carlo object, that can be digested by the CODA methods and functions. The input data are taken to be a vector, or a matrix with one column per variable

- useful functions are

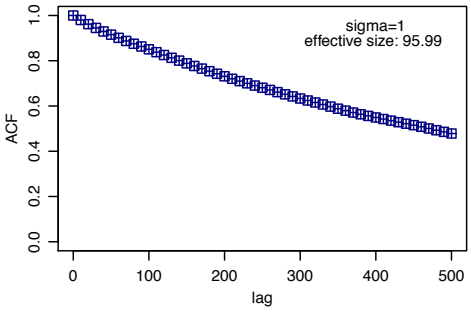
▷ `autocorr()` : calculates the auto-correlation function for the Markov chain object at the lags given by parameter `lags`. High auto-correlations within chains indicate slow mixing and, usually, slow convergence

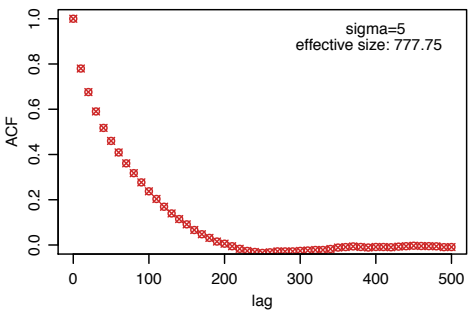
▷ `effectiveSize()` : computes the sample size adjusted for auto-correlation

MCMC chain analysis example

- we run our previous example changing the σ parameter of the proposal distribution function and analyze the MCMC chain

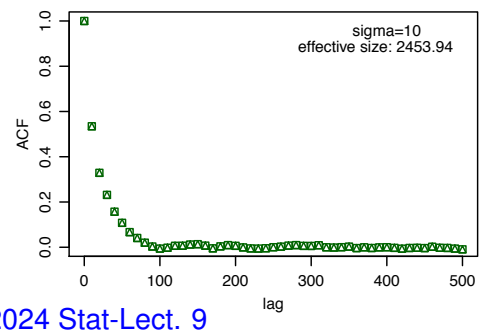
```
library(coda)
c.chain1 <- as.mcmc(chain.r1$func.Samp[,2])

my.lags = seq(0,500,10)
y1 <- autocorr(c.chain1, lags=my.lags)

plot(my.lags, y1, ylim=c(0,1),
     pch=12, col='navy',
     xlab='lag', ylab='ACF', cex=1.3)
text(400,0.9, paste('sigma=1'))
text(400,0.85,
     sprintf("effective size: %.2f",
             effectiveSize(c.chain1)))
```





- the first sample is strongly correlated

σ	$R_{acceptance}$	N_{eff}
1	0.9256	95.99
5	0.4127	777.75
10	0.1585	2453.94



MCMC and parameter transformation

- sometimes it is more efficient to sample over a transformed parameter
- let's consider, as an example, a parameter $\theta > 0$; we could sample $\ln\theta$ since it ensures the parameter θ cannot be negative
- but this means drawing from $P(\ln\theta)$ and not from $P(\theta)$. Since

$$P(\theta)d\theta = P(\ln\theta)d(\ln\theta) \Rightarrow P(\ln\theta) = \theta P(\theta)$$

- when we are using a symmetric proposal distribution

$$Q(\theta_t \mid s) = Q(s \mid \theta_t)$$

- the Metropolis ratio becomes

$$\rho = \frac{sP(s)}{\theta_t P(\theta_t)}$$

- the base of the logarithm in the transformation is irrelevant, since it corresponds to a constant factor that cancels in the ratio
- in general, for a transformation from $(\theta_1, \ldots, \theta_J)$ to $(\phi_1, \ldots, \phi_J)$ we need the Jacobian determinant of the original parameters versus the transformed ones

$$\mathcal{J}_\theta = \left| \frac{\partial(\theta_1, \ldots, \theta_J)}{\partial(\phi_1, \ldots, \phi_J)} \right|$$

- and the Metropolis ratio becomes

$$\rho = \frac{P(s)}{P(\theta_t)} \frac{\mathcal{J}_s}{\mathcal{J}_\theta}$$

Parameter estimation with MCMC

- we will show how to sample posteriors with more than two parameters using MCMC
- as an example we will consider a fit to data, both linear and quadratic, whereby we will infer also the noise on the data

The problem requirements

- we have a 2-dim set of N points $\{x_j; y_j\}$
- the model M predicts:
  - $y = f(x) + \epsilon$
  - where $f(x) = b_0 + b_1 \cdot x$
- $f(x)$ is the generative model, it gives noise-free prediction of the data, given the parameters
- the residuals $\epsilon = y - f(x)$ are modeled as a zero-mean Gaussian function with standard deviation σ. This is the noise model
- assuming $\{x_j\}$ are noise-free, and $\theta = (b_0, b_1, \sigma)$, the likelihood is

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

Parameter estimation with MCMC

- we try to infere σ from the data
- although the $\{x_j\}$ values are supplied with the data, they are assumed to be fixed and not described by a measurement model. Therefore $D = \{y_j\}$
- assuming data points are independent, the log-likelihood for all the data points is

$$\ln P\left(\{y_j\} \,\middle|\, \{x_j\}, \theta, M\right) = \sum_{j=1}^{N} \ln P\left(y_j \,\middle|\, x_j, \theta, M\right)$$

- in general, none of the parameters is known in advance and we want to infer the posterior from the data

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

- given the data, D, the procedure to compute the posterior is as follows:

(1) define the prior pdf for the parameters. Use reasonable and plausible priors and make use, if needed, of variable transformation

(2) define the covariance matrix of the proposal distribution. (a diagonal, multivariate Gaussian distribution)

(3) define the starting point of the MCMC

(4) define the number of burn-in and sampling interactions

Parameter estimation with MCMC

- once the MCMC data have been collected, perform the following analysis

(5) make the chains thinner

(6) plot the chains and the one one-dimensional marginal posterior pdf over the parameters

(7) plot the two-dimensional posterior distributions of all three parameters, simply by plotting the samples, and look for correlations between the parameters

(8) calculate the maximum a posteriori values of the model parameters from the MCMC chains, calculate and plot the resulting model, and compare to the original data

(9) calculate the predictive posterior distribution over y at new data points

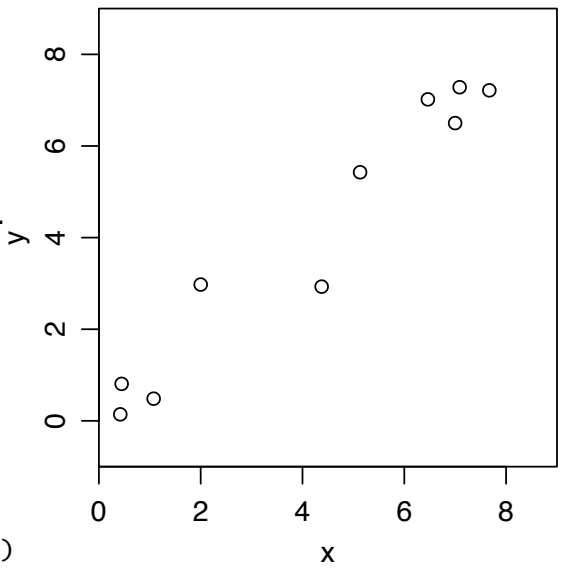
- since we have samples drawn from the posterior, we don't need the actual values of the posterior density in order to plot the posteriors. For the same reason, we don't have to perform any integration to get the one-dimensional marginal distributions

Parameter Priors

- for the intercept, b_0 : $P(b_0) = \mathrm{N}(\mu, \sigma)$, a Gaussian with mean μ and standard deviation σ
- for the gradient, b_1 : we can write it as $b_1 = \tan\alpha$, where α is the angle, in radians, between the horizontal and the model line. Since we have no prior knowledge of the slope, we should use a uniform distribution $P(\alpha) = 1/2\pi$
- standard deviation, σ : in the absence of any other information, a scale parameter such as the standard deviation of a Gaussian should be assigned a Jeffreys prior, $P(\sigma) \propto \log\sigma$. This also prevents σ from becoming negative.
- given these priors, the model parameters are now $(b_0, \alpha, \log\sigma)$. These are the parameters that the Monte Carlo algorithm will sample over. The prior distributions are likewise defined over the parameters, as Gaussian (b_0), uniform (α), and uniform ($\log\sigma$, respectively

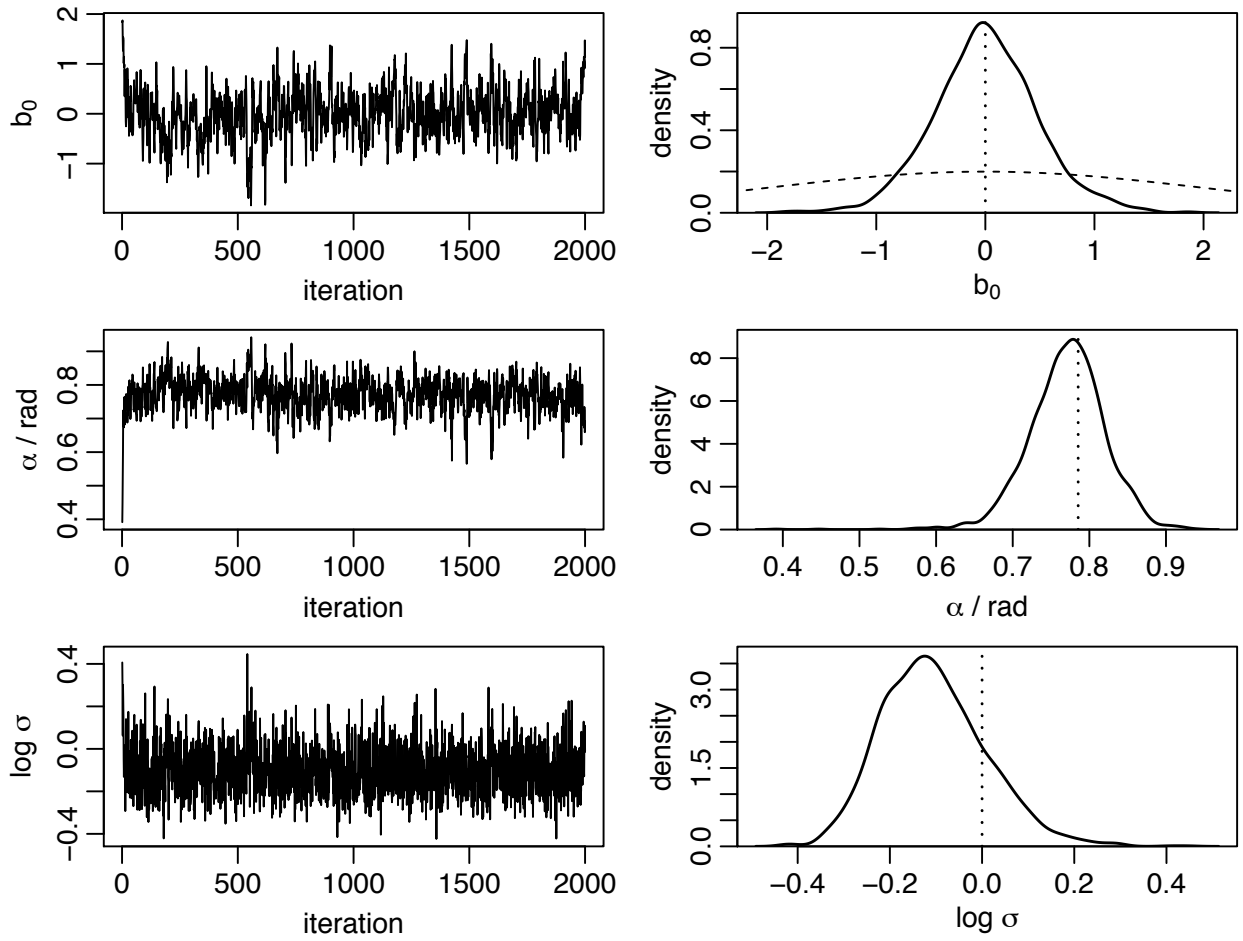
Example: the data

- these are the 10 data points we want to fit to our model
- they have been drawn at fixed x values from a straight line with $b_0 = 0$ and $b_1 = 1$, to which zero mean Gaussian noise with $\sigma = 1$ has been added.




```
Ndat <- 10
x <- sort(runif(Ndat, 0, 10))
sigTrue <- 1

# 1 x P vector: coefficients,
# b_p, of sum_{p=0} b_p*x^p
modMat <- c(0,1)
y <- cbind(1,x) %*% as.matrix(modMat) +
                    rnorm(Ndat, 0, sigTrue)
# Dimensions in the above:
#    [Ndat x 1] = [Ndat x P] %*% [P x 1] + [Ndat]
# cbind does the logical thing when combining
# a scalar and vector, then do vector addition

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

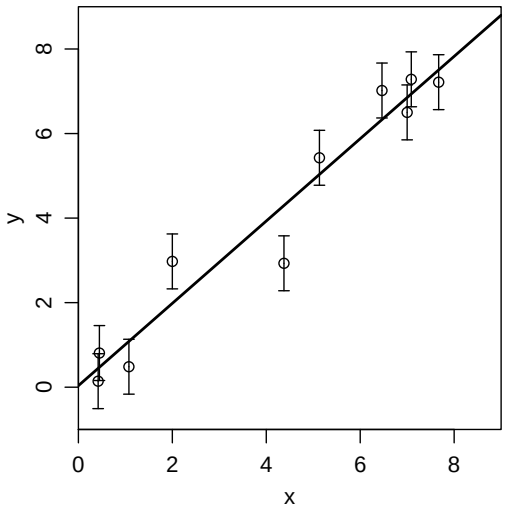
Example: the MCMC

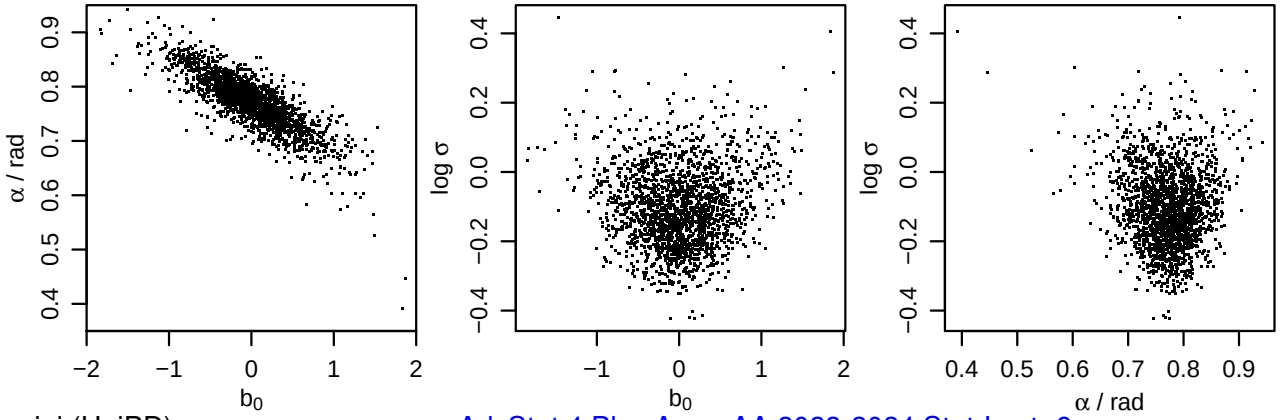


Example: fit and parameters correlations

- the mean of the posterior is $(b_0, \alpha, \log\sigma) = (0.0042, 0.77, -0.11)$
- the covariance of the posterior pdf is

	b_0	α	$\log\sigma$
b_0	0.48		
α	-0.83	0.050	
$\log\sigma$	0.038	-0.073	0.11





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

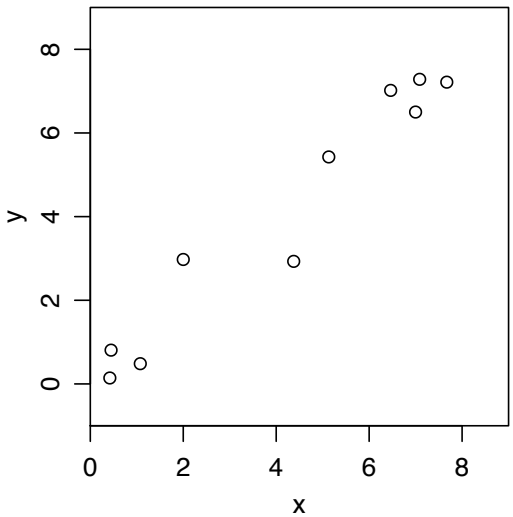
- the R code of the book can be dowloaded from
- https://github.com/ehalley/PBI/tree/master/PBI_scripts:

  - 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

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

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

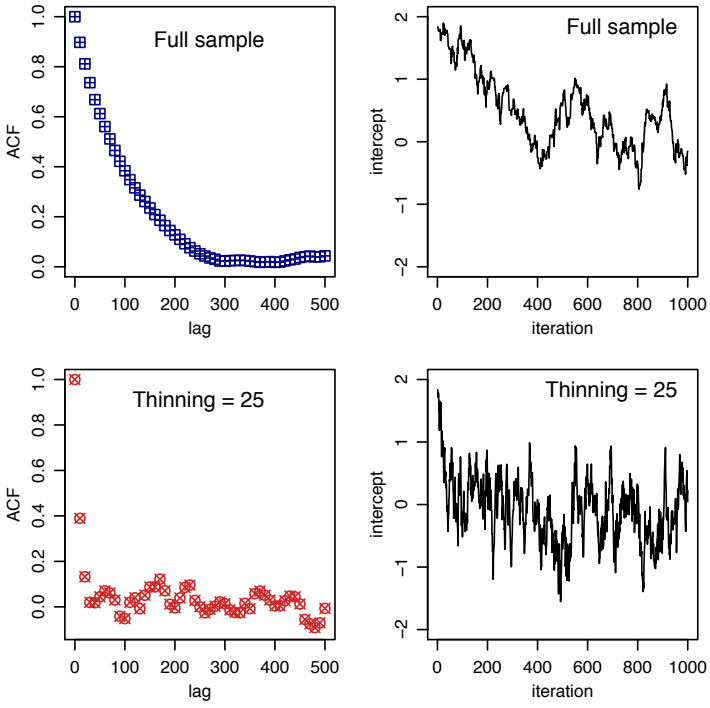
- the starting values for the Markov Chain are $b_0 = 2$, $\alpha = \pi/8$ and $\log_{10}\sigma = \log_{10}(3)$
- the step size for the evolution of the chain are 0.1, 0.02 and 0.1 (respectively for b_0, α and $\log\sigma$)

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

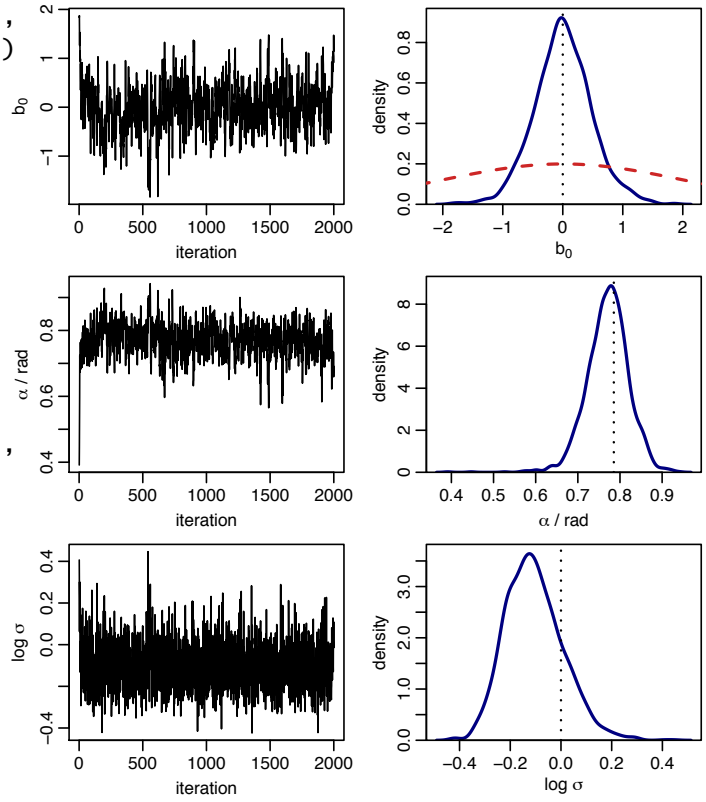
- the unormalized Posterior has been used in the MCMC, the normalization is not needed since samples are drawn with the same relative frequency, independently of the normalization
- in contrast to the Posterior, the Likelihood has to be normalized since it is a pdf over the data and therefore its normalization 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)

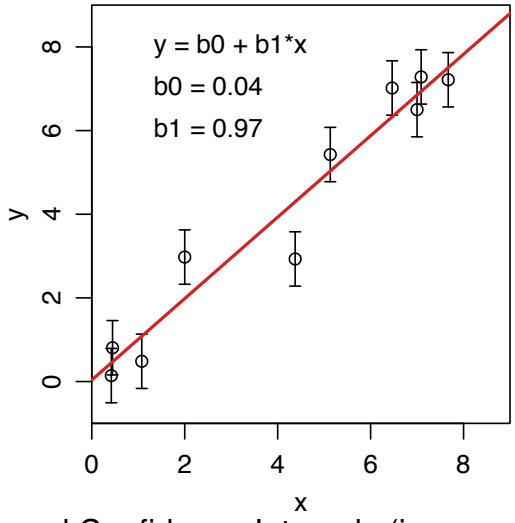
```
mean(tan(postSamp[,4])) # transform alpha to b_1
mean(10^(postSamp[,5])) # transform log10(sigma) to sigma
```

- 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)]
```

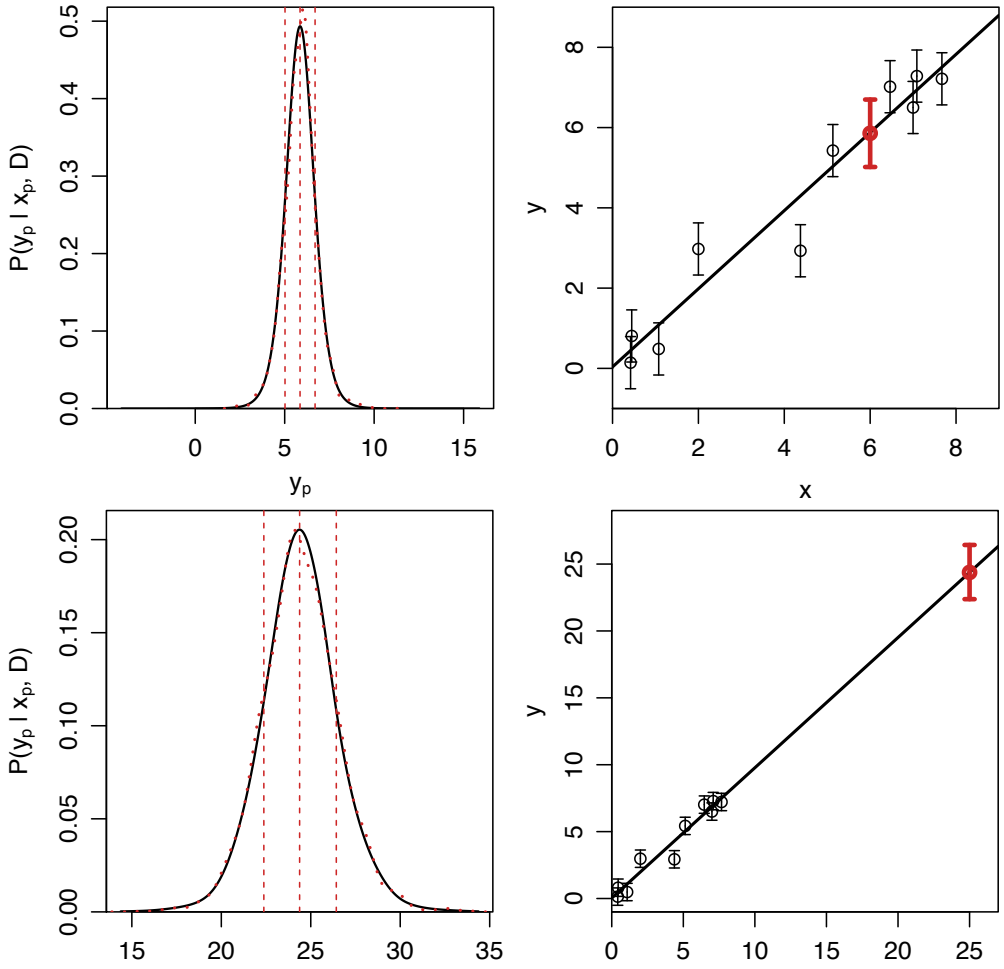
Posterior Predictive distribution - example

```
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

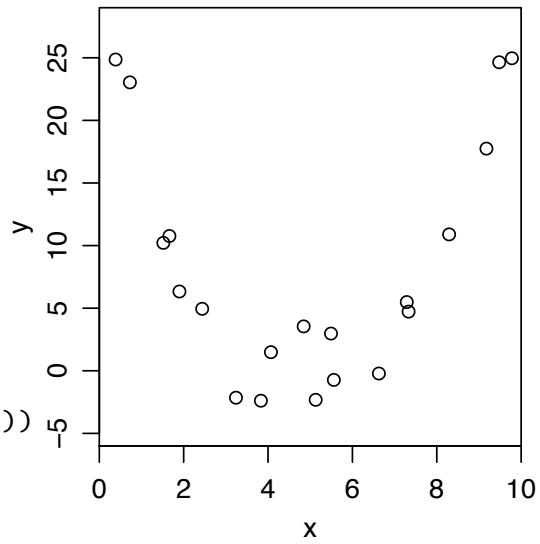


Fitting a quadratic curve with noise

- 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

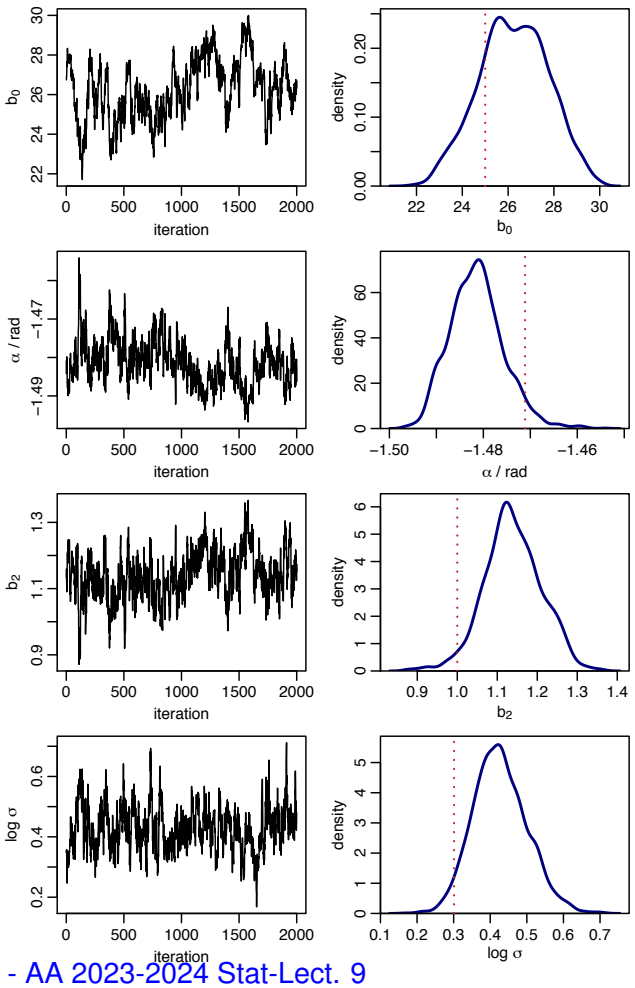
- the parameters step sizes (Gaussian standard deviations) are chosen as $(b_0, \alpha, b_2, \log \sigma) = (0.1, 0.01, 0.01,$ and $0.01)$
- as starting point any value can be in principle chosen, but it could take a large number of steps to locate the high density region of the posterior. Therefore 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
```

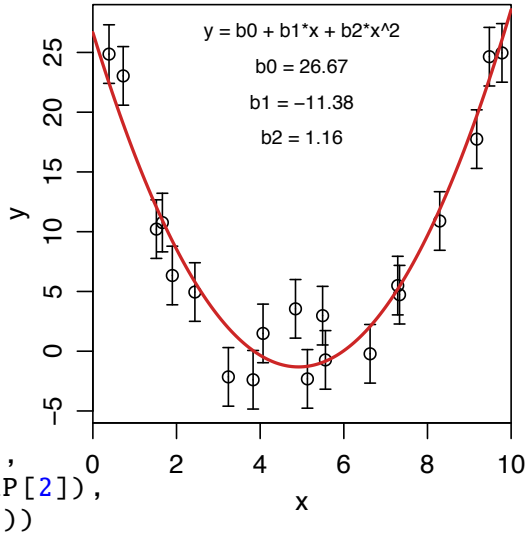
Quadratic curve : posterior pdfs

```
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.