Lab 5. Simulating a posterior distribution

In this chapter I consider techniques used to simulate a posterior. I begin with some methods for sampling from a univariate nonstandard distribution. I then move to Gibbs sampling, the most common method for simulating posteriors in more complex models.

Rejection sampling

In Example 7.1 of the text, I discuss Hudgen's data, which consist of 286 observed turn angles for aphids placed in 36 ten-degree bins (variable "angle") that range from - π to π radians (-180 to 180 degrees). In this section I simulate aphid turns from these data using rejection sampling. Here are the data:

```
btmp <- read.table("hudgens.txt",header=T)
attach(btmp)
par(mfrow=c(2,1))
plot(angle,counts,type='s',ylim=c(0,20))
```

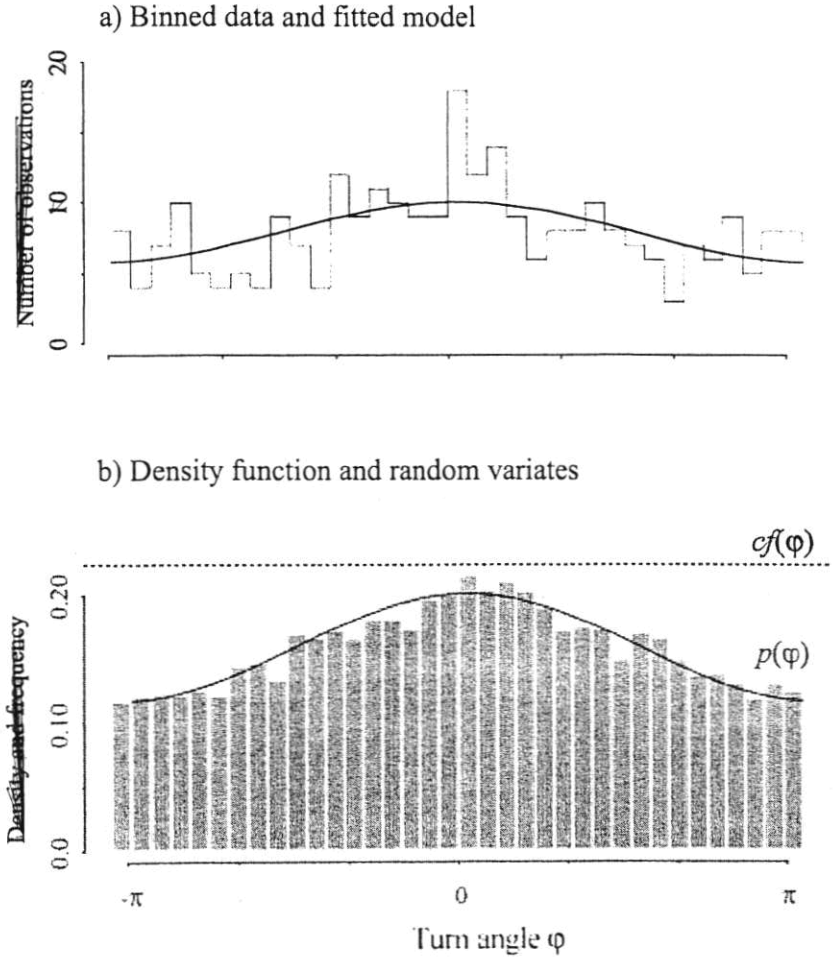

Figure 5.1. Turn angles simulated by rejection sampling. a) The counts for observed turns in each of the angle classes with a model fitted by ML (smooth curve). b) Samples from the model, represented by a histogram, with the target density $p(\varphi)$ and the uniform density used for proposals $f(\varphi)$. The fitted model (smooth curve) is normalized in b (divided by $2\pi\beta_0$), but not in a.

(Fig. 5.1a). I first fit a function to the turn data. The data consist of counts y_j, the number of the turns observed in 10-degree discrete bins that range from -π to π. The total number of bins is m. The density to simulate is

$$p(\varphi) = \frac{b_0 + b_1 \cos(b_2 + \varphi)}{2\pi b_0}$$

The parameters are phase (b_2), amplitude (b_1), and intercept (b_0). Note that this function is a density, the normalization constant being $2\pi b_0$. Because angles are recorded in discrete bins, the probability for bin j is approximately the density times the bin width,

$$p(\varphi_j)d\varphi_j = \frac{2\pi p(\varphi_j)}{m}$$

The likelihood might be taken as Poisson or multinomial. If Poisson, the likelihood for the number of turns to fall in bin j is

$$y_j \sim Pois\left(\frac{n}{m} p(\varphi_j)\right)$$

The leading coefficient n/m is the sample size divided by the number of bins for turn angles,

```
n <- sum(counts)
m <- length(angle)
```

This is the expected number of counts in a given angle class, which is scaled by the density $p(\varphi)$. A function using the Poisson likelihood could be written like this:

```
fitturn <- function(par){
  b0 <- par[1]
  b1 <- par[2]
  b2 <- par[3]
  mu <- (b0 + b1*cos(b2 + angle))/2/pi/b0    #Poisson mean
  llik <- dpois(counts,n/m*mu,log=T)         #vector of log Poisson
probabilities
  return(-sum(llik))
}
```

Alternatively, we could use a multinomial likelihood

$$y_j \sim Multinom(n, p_1, \ldots p_m)$$

where the probability that y_j turn angles fall in class j is proportional to $p(\varphi_j)$, and the multinomial log likelihood of counts falling in the j^{th} class is

$$y_j \ln\left(p(\varphi_j)d\varphi_j\right)$$

This function is written as

```
fitturn <- function(par){
  b0 <- par[1]
  b1 <- par[2]
  b2 <- par[3]
  mu <- (b0 + b1*cos(b2 + angle))/2/pi/b0 # mean
  llik <- sum(counts*log(mu))
  return(-llik)
}
```

Here is an optimization to find the MLEs using the Poisson likelihood:

```
param <- c(5,2,-1)
fit <- optim(param,fitturn,lower=c(2,1,-2),upper=c(10,5,1),
    method="L-BFGS-B",control=list(factr=1e12,maxit=3000))
```

For this data set we obtain MLEs

```
$par
[1]  6.31406160  1.69370360 -0.07516426
```

Here is the result for the multinomial likelihood

```
$par
[1]  6.18861500  1.64885893 -0.08105502
```

These differences are small. The data and fitted model shown in Figure 5.1a can be plotted with

```
b0 <- fit$par[1]; b1 <- fit$par[2]; b2 <- fit$par[3]
tseq <- seq(-pi,pi,length=100)
lines(tseq, (b0 + b1*cos(b2 + tseq))) #unnormalized function
```

Up to this point I have simply obtained parameter estimates by maximum likelihood.

To draw samples from this distribution, I use rejection sampling. Rejection sampling involves an 'envelope' for the distribution, which is some constant c times the density I will use to propose values of φ_j. I use a uniform density for proposed values of turn angle

$$f(\varphi_j) = Unif(\varphi_j|-\pi,\pi) = \frac{1}{2\pi}$$

A value of $c = 70$ safely covers the density to be estimated (Fig. 5.1b), i.e., $cf(\varphi) > f(\varphi)$ for all φ. Here I define these quantities:

```
c <- 70
f <- 1/2/pi                 #the uniform density
cf <- c*f                   #the basis for rejection
```

To generate a large number of variates I propose 10,000 values for bin angle φ_j, calculate the acceptance probabilities a, draw uniform random variates z on (0,1) and keep those proposals for which $z < a$:

```
n <- 10000
x <- runif(n,-pi,pi)
a <- (b0 + b1*cos(b2 + x))/c/f   #unnormalized function
z <- runif(n,0,1)
xturn <- x[z < a]
hist(xturn,nclass=length(angle),probability=T)
lines(tseq, (b0 + b1*cos(b2 + tseq))/2/pi/b0) #normalized function
```

The acceptance rate is the number of keepers divided by the number of proposals.

```
> length(xturn)/n
[1] 0.5692
```

So I kept over half of the proposals. I could increase the efficiency by decreasing the factor c, but I must be careful not to let $cf(\varphi)$ be exceeded by the target density. Here efficiency is not an issue, but it could be if this rejection sampler were embedded within a complex model. The target density is estimated by the histogram of values `xturn`.

Truncated normal distribution

In section 7.2.2 of the text I discuss inverse distribution sampling. Here I describe a particular type of inverse distribution sampling for a variate x that is truncated at x_1 and x_2. Here is an algorithm for drawing a value from $x \sim N(\mu,\sigma^2)I(x_1 < x < x_2)$

1. Draw a uniform random variate from $z \sim Unif(0,1)$.
2. Obtain quantiles for at truncation points $q_1 = \Phi((x_1 - \mu)/\sigma)$ and $q_2 = \Phi((x_2 - \mu)/\sigma)$
3. Transform this quantile to $x = \mu + \sigma\Phi^{-1}(q_1 + z(q_2 - q_1))$

Φ and Φ^{-1} are the standard unit normal and it's inverse function, respectively. Here is an R function:

```
tnorm <- function(n,lo,hi,mu,sig){   #normal truncated lo and hi
  z <- runif(n,0,1)
  qlo <- pnorm((lo - mu)/sig)
  qhi <- pnorm((hi - mu)/sig)
  xx <- mu + sig*qnorm(qlo + z*(qhi - qlo))
  return(xx)
}
```

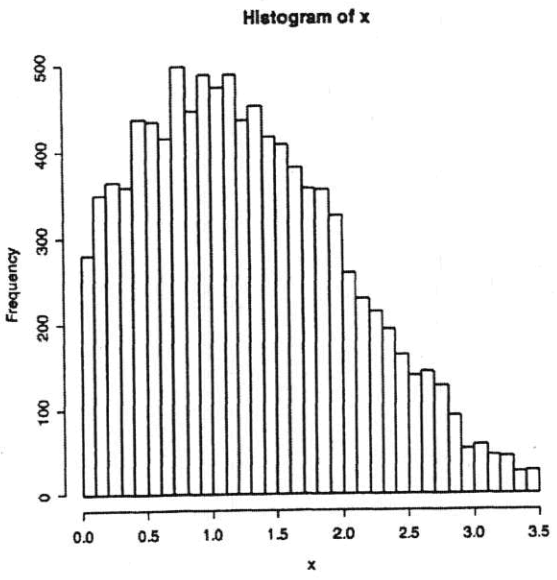

Fig. 5.2. Histogram of random variates drawn from a truncated normal.

Figure 5.2 shows a histogram of 10,000 random variates for truncation points (0, 3.5), mean $\mu = 1$, and standard deviation $\sigma = 1$:

```
x <- tnorm(10000,0,3.5,1,1)
hist(x,nclass=30)
```

Regression with Gibbs sampling

There are several techniques that I introduce here within the context of Gibbs sampling. As discussed in the text, Gibbs sampling serves as a framework for simulating a multivariate posterior, based on draws from conditional posteriors. In some cases, we can sample directly from the conditionals, as when they result from conjugate

prior/likelihood combinations. In other cases we can embed within the Gibbs sampler other types of sampling techniques. I begin by simulating a data set for linear regression, followed by a Gibbs sampler for the regression model. This provides an example for which all conditional posteriors can be directly sampled. I follow with variations on the regression theme with examples of inverse distribution sampling, Metropolis, and Metropolis-Hastings.

First simulate data for a regression model. Because the parameter values used to simulate the data are known, I can use this example to check whether the Gibbs sampler is doing what I think it should be doing. Define a sample of size $n = 20$, with regression parameters $\boldsymbol{\beta} = [10, 2]^T$ and residual standard deviation of $\sigma = 4$,

```
n <- 500
b0 <- 10
b1 <- 2
beta <- matrix(c(b0,b1),2,1)
sig <- 4^2
```

The design matrix x will have a column of ones and a column for the predictor variables, which I arbitrary draw from a unit normal distribution with means equally spaced from 1 to 20 (again, this is arbitrary). The parameter vector `beta` has dimension 2 by 1. Simulated responses y have expectation given by the regression with standard deviation $\sigma = 4$,

```
x1 <- rnorm(n,mean=seq(1,20,length=n),sd=1)   #predictor variable
x <- cbind(rep(1,n),x1)                       #assemble design matrix
y <- matrix(rnorm(n,x%*%beta,sqrt(sig)),n,1) #simulated y values
```

Figure 5.3 shows a simulated data set together with the line drawn for the underlying model used to simulate it

```
plot(x[,2],y)
abline(b0,b1)
```

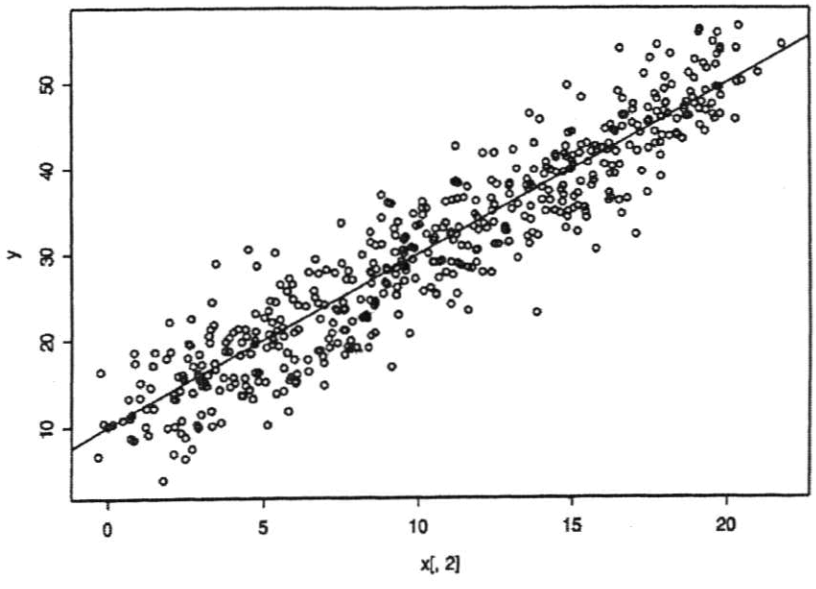

Figure 5.3. Regression of simulated data, and the line drawn for the model used for in the simulation.

In this example, I specify prior parameter values that make the prior contribution weak. I specify large prior variances on regression parameters of 1000 and small parameter values for the inverse gamma prior on the variance:

```
bprior <- as.vector(c(0,0))          #priors for regression parameters
vinvert <- solve(diag(1000,2))       #inverse prior covariance matrix
s1 <- .1                             #variance priors
s2 <- .1
```

A simple Bayesian analysis with Gibbs sampling makes use of conditional posteriors for regression parameters and for the residual variance. For conjugate normal and inverse gamma priors the conditional posteriors have forms given in Chapter 8 of the text. For the regression parameters we have

$$p(\beta|\mathbf{X},\mathbf{y},\Sigma,\ldots) = N(\beta|\mathbf{Vv},\mathbf{V})$$

where $\mathbf{v} = \mathbf{X}^T\Sigma^{-1}\mathbf{y} + \mathbf{V}_0^{-1}\mathbf{b}_0$, $\mathbf{V}^{-1} = \mathbf{X}^T\Sigma^{-1}\mathbf{X} + \mathbf{V}_0^{-1}$, and "..." refers to prior parameter values. Recall that, if samples are independent, then $\Sigma^{-1} = \sigma^{-2}\mathbf{I}_n$. For a single variance parameter the conditional posterior is

$$p(\sigma^2|\mathbf{X},\mathbf{y},\beta,\ldots) = IG(\sigma^2|u_1,u_2)$$

where $u_1 = s_1 + n/2$, and $u_2 = s_2 + \frac{1}{2}(\mathbf{y}-\mathbf{Xb})^T\mathbf{R}^{-1}(\mathbf{y}-\mathbf{Xb})$, where $\mathbf{R}$ is the correlation matrix as is used when there is autocorrelation. Because these forms apply to most of the remaining examples in this chapter, I write functions that can be reused. For the regression parameters, I will call this function with arguments for the response variable and for either i) the inverse covariance matrix or, ii) if there is a single variance parameter, $1/\sigma^2$. If samples have identical variances σ^2, as in the current example, the function knows to forego the extra matrix multiplication. Here is the function:

```
b.update <- function(y,sinv){

  if(length(sinv) == 1){        #a single variance parameter
    sx <- crossprod(x)*sinv
    sy <- crossprod(x,y)*sinv
  }
  if(length(sinv) > 1){         #a covariance matrix
    ss <- t(x) %*% sinv
    sx <- ss %*% x
    sy <- ss %*% y
  }
  bigv <- solve(sx + vinvert)
  smallv <- sy + vinvert %*% bprior
  b <- t(rmvnorm(1,bigv%*%smallv,bigv))
  return(b)
}
```

For the current example, assuming an inverse variance of 1/16, I could draw a sample from the conditional posterior for the regression parameters with this call,

```
b.update(y,1/16)
         [,1]
[1,] 9.989292
[2,] 1.976059
```

To sample from the variances, this function can be used

```
v.update <- function(y,rinverse){
```

```
  if(length(rinverse) == 1) sx <- crossprod((y - x %*% b))
  if(length(rinverse) > 1){
     sx <- t(y - x %*% b) %*% rinverse %*% (y - x %*% b)
  }
  u1 <- s1 + .5*n
  u2 <- s2 + .5*sx
  return(1/rgamma(1,u1,u2))
}
```

The 2nd argument to `v.update` is the inverse correlation matrix. If there is no correlation matrix to enter, just use 1 for the second argument. Here is a draw from the current example:

```
> v.update(y,1)
[1] 15.90060
```

Now establish the number of Gibbs steps, `ngibbs`, and define arrays to hold estimates:

```
ngibbs <- 10000
bgibbs <- matrix(NA,nrow=ngibbs,ncol=2) #for regression parameters
sgibbs <- rep(0,ngibbs)                 #for variance
sg <- v.update(y,1)                     #initial variance from prior
```

I use functions that you may need to load in your version of R,

```
library(MASS)
library(mvtnorm)
```

Within the Gibbs loop I will alternately sample for regression parameters and for the variance. Here's the Gibbs loop:

```
for(g in 1:ngibbs){

  b <- b.update(y,1/sg) #sample regression parameters
  sg <- v.update(y,1)   #sample variance

  bgibbs[g,] <- b       #save estimates
  sgibbs[g] <- sg
 }
```

The first part of the Gibbs sampler is a draw from a bivariate normal—I am sampling both regression parameters simultaneously. The second part is a draw for the variance parameter. Here is code to plot the chains of estimates and some posterior densities and to determine quantiles:

```
par(mfcol=c(3,2))
plot(bgibbs[,1] ,type='l')                          #plot Gibbs chain
  abline(h=b0)
plot(bgibbs[,2] ,type='l')
  abline(h=b1)
plot(sgibbs,type='l')
  abline(h=16)
bdens<-density(bgibbs[,1],width=.4)                 #smooth posterior for
β0
  plot(bdens,type='l')
  abline(v=quantile(bgibbs[,1],c(.5,.025,.975))) #quantiles
bdens<-density(bgibbs[,2],width=.1)
  plot(bdens,type='l')
  abline(v=quantile(bgibbs[,2],c(.5,.025,.975)))
sdens<-density(sqrt(sgibbs),width=.5)
```

```
plot(sdens,type='l')
abline(v=quantile(sqrt(sgibbs),c(.5,.025,.975)))
```

These lines plot the sequence and densities fitted to the marginal posteriors (Fig. 5.4). From the chains you can see that convergence is almost instantaneous for this linear model. For more complex models convergence can be slow, and we would discard the early 'burnin' iterations before moving to any analysis of the chains.

The 95% credible intervals (vertical lines on the densities estimates of Figure 5.4) encompass the parameter values used to simulate the data. However, one simulation is typically not sufficient to check a complex model. A full analysis would involve repeated simulations. You might then be interested to know whether the ensemble of credible intervals gives approximate 'frequentist coverage', i.e., do 95% of credible intervals include the values used to simulate the data.

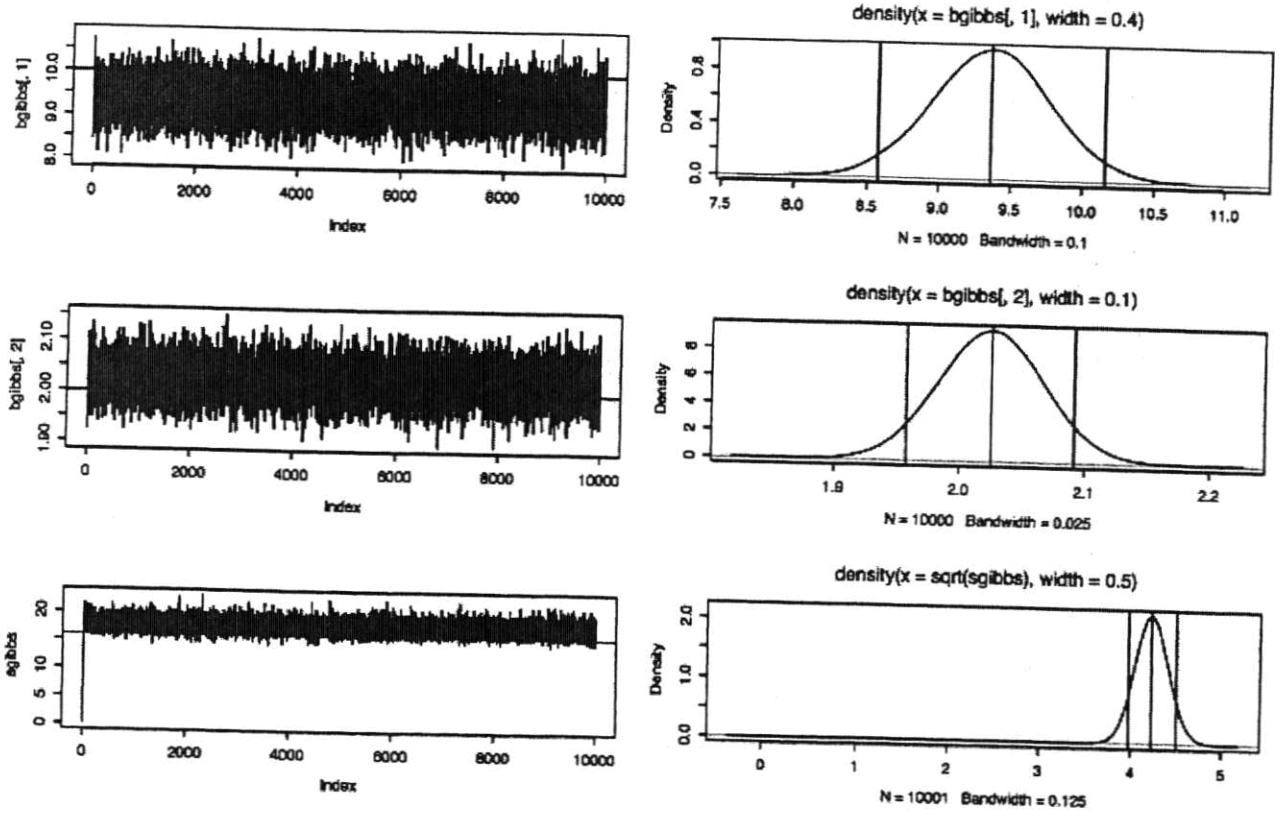

Fig. 5.4. Gibbs chains (left) and marginal posteriors (right) for regression parameters and variance parameter. Horizontal lines at left indicate the parameter values that were used to simulate data. Vertical lines on plots at right show outer 95% quantiles and median values. These intervals contain values used to simulate the data, ie., $\beta_0 = 10$, $\beta_1 = 2$, $\sigma = 4$.

We can assemble some useful summaries of the posterior in the form of a table. The following code discards the first 10 iterations, thins the Gibbs chains to every 10th value, assembles the chains into a single matrix `gmat`, and then loads means, se's, and 95% CI's in a matrix `outpar`.

```
outpar <- matrix(NA,3,4)
thin <- seq(10,ngibbs,by=10)                    #thin sequence
gmat <- cbind(bgibbs[thin,],sqrt(sgibbs[thin]))
outpar[,1] <- apply(gmat,2,mean)
outpar[,2] <- apply(gmat,2,sd)
```

```
outpar[,3:4] <- t(apply(gmat,2,quantile,c(.025,.975)))
colnames(outpar) <- c('estimate','se','0.025','0.975')
rownames(outpar) <- c('b0','b1','sigma')
signif(outpar,4)

> signif(outpar,4)
      estimate     se 0.025  0.975
b0       9.376 0.4081 8.581 10.230
b1       2.024 0.0348 1.953  2.088
sigma    4.234 0.1328 3.981  4.511
```

Given the weak priors, we would expect this analysis to not differ appreciably from a classical one. From the R function `lm` we see this to be the case:

```
summary(lm(y ~ x[,2]))
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  9.35976    0.40124   23.33   <2e-16 ***
x[, 2]       2.02540    0.03381   59.91   <2e-16 ***
Residual standard error: 4.225 on 498 degrees of freedom
```

A nonlinear model

One of the advantages of the approach is the ease with which it can be extended to more complex problems. The nonlinear model of light availability and seedling growth (Section 7.4.4 of the text) requires just a few adjustments. As part of this example, I demonstrate inverse distribution sampling.

To linearize two of the parameters in the model of Section 7.4.4, I wrote the model like this:

$$y_i = \mu_i + \varepsilon_i$$

$$\mu_i = \beta_0 + \beta_1 z_i,$$

$$z_i = \frac{x_i}{\theta + x_i}$$

$$\varepsilon_i \overset{i.i.d.}{\sim} N(0,\sigma^2)$$

The design matrix is

$$\mathbf{X} = \begin{bmatrix} 1 & z_1 \\ \vdots & \vdots \\ 1 & z_n \end{bmatrix}$$

Here is code to simulate 100 growth values `y` in response to the covariate `light`:

```
n <- 100
light <- runif(n,0,1)              #covariate vector
th0 <- .1                          #theta
z <- light/(th0 + light)
x <- cbind(rep(1,n),z)             #n by 2 design matrix

b0 <- 12                           #regression parameters
b1 <- 40
beta <- matrix(c(b0,b1),2,1)
sig <- 25
y <- matrix(rnorm(n,x%*%beta,sqrt(sig)),n,1)  #response vector
```

The data are contained in matrix x (the nonlinear function of light availability) and vector y (growth increment).

Now construct a Gibbs sampler to estimate the regression parameters, the variance, and the additional parameter θ,

```
ngibbs <- 10000
thgibbs <- rep(.1,ngibbs)                    #vector for theta's
  th <- .1
  x[,2] <- light/(th + light)                #design matrix with initial th
bgibbs <- matrix(0,nrow=ngibbs,ncol=2)       #matrix for beta's
sgibbs <- rep(1,ngibbs)                      #vector for sigma's
  sg <- 1
```

I will use this example to demonstrate inverse distribution sampling for θ. These matrices will allow us to efficiently sample θ,

```
thseq <- seq(.05,.3,length=100)                          #sequence of theta's
tmat <- matrix(rep(thseq,each=n),nrow=n,byrow=F)          #column for each
theta
lmat <- matrix(rep(light,each=100),nrow=n,byrow=T)       #row for each light
value
zmat <- lmat/(lmat + tmat)                               #light/theta
combinations
ymat <- matrix(rep(y,each=100),nrow=n,byrow=T)           #row for each y
```

The 1st of these lines defines a sequence of θ values `thseq` from 0.05 to 0.3. This means that the prior is $Unif(\theta|0.05,0.3)$. The next few lines lead to a matrix `zmat`, which contains one row of z values for each element of **y**, with different columns of `zmat` corresponding to each value of θ in the vector `thseq`. The matrix `ymat` has columns that simply repeat each value of **y** for purposes of calculation. Within the Gibbs loop, the current value of θ is assigned the name `th`. I'll return to this after the Gibbs loop. Here are the other priors:

```
bprior <- as.vector(c(0,0))          #priors for regression parameters
vinvert <- solve(diag(1000,2))       #inverse prior covariance matrix
ppart <- vinvert %*% bprior
s1 <- .1                             #variance priors
s2 <- .1
```

Here is the Gibbs sampler:

```
for(g in 1:ngibbs){

  b <- b.update(y,1/sg)
  sg <- v.update(y,1)

 #theta using inverse distribution sampling
    mmat <- b[1] + b[2]*zmat                                  #matrix of
means
    plik <- apply(dnorm(ymat,mmat,sqrt(sg),log=T),2,sum) #lnL for each
thseq
    pseq <- exp(plik - max(plik))                             #normed L
    pbin <- cumsum(pseq)                                      #cumulative L
    zp <- runif(1,pbin[1],pbin[100])
    thgibbs[g] <- thseq[findInterval(zp,pbin)]          #select new theta
    x[,2] <- light/(th + light)                               #update design
matrix
```

```
  bgibbs[g,] <- b
  sgibbs[g] <- sg
  th <- thgibbs[g]
}
```

For the inverse distribution-sampling section of this code, I determine the likelihood taken over all individuals, for a sequence of θ values (Section 8.2.2 of text). The sequence `thseq` is used to evaluate this likelihood (Fig. 5.5). Using the current values for regression parameters `b` and `zmat`, which contains the expected values for each individual at all values of `th` in the sequence, I determine the vector `plik`. This holds the ln$L$ value for the entire data set at each value of `thseq`. I then determine the normed likelihood (`pseq`), cumulate it (`pbin`), draw a uniform variate `zp` and find the new value of `th` to which it corresponds.

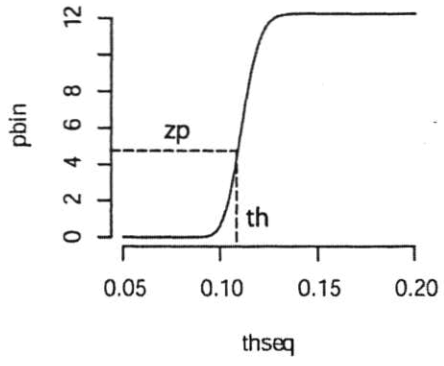

Figure 5.5. The sequence of `th` values (horizontal axis) and the cumulative likelihood `pbin`. The uniform random variate `zp` is drawn, with the corresponding updated value of `th`.

Figure 5.6 shows parameter values generated by the Gibbs sampler. These correspond to the example in section 8.2.2 of the text. Note that they recover values used to simulate the data. There is the expected correlation between regression parameters β_0 and β_1. There is also a tendency for positive correlation between β_0 and θ. This positive correlation means that there is limited capacity to identify the two parameters. The mean structure of the model,

$$\mu_i = \beta_0 + \beta_1 \left(\frac{x_i}{\theta + x_i} \right)$$

can, to some extent, 'fit' the data if β_0 and θ are both large or both small. Here is a correlation matrix:

```
thin <- seq(100,ngibbs,by=100)
signif(cor(cbind(bgibbs[thin,],sgibbs[thin],thgibbs[thin])),3)
        [,1]    [,2]    [,3]   [,4]
[1,]  1.0000 -0.9260 -0.0389  0.8070
[2,] -0.9260  1.0000  0.0126 -0.5610
[3,] -0.0389  0.0126  1.0000 -0.0286
[4,]  0.8070 -0.5610 -0.0286  1.0000
```

Figure 5.6 is produced with the R function `pairs`:

```
pairs(cbind(bgibbs[thin,],sgibbs[thin],thgibbs[thin]))
```

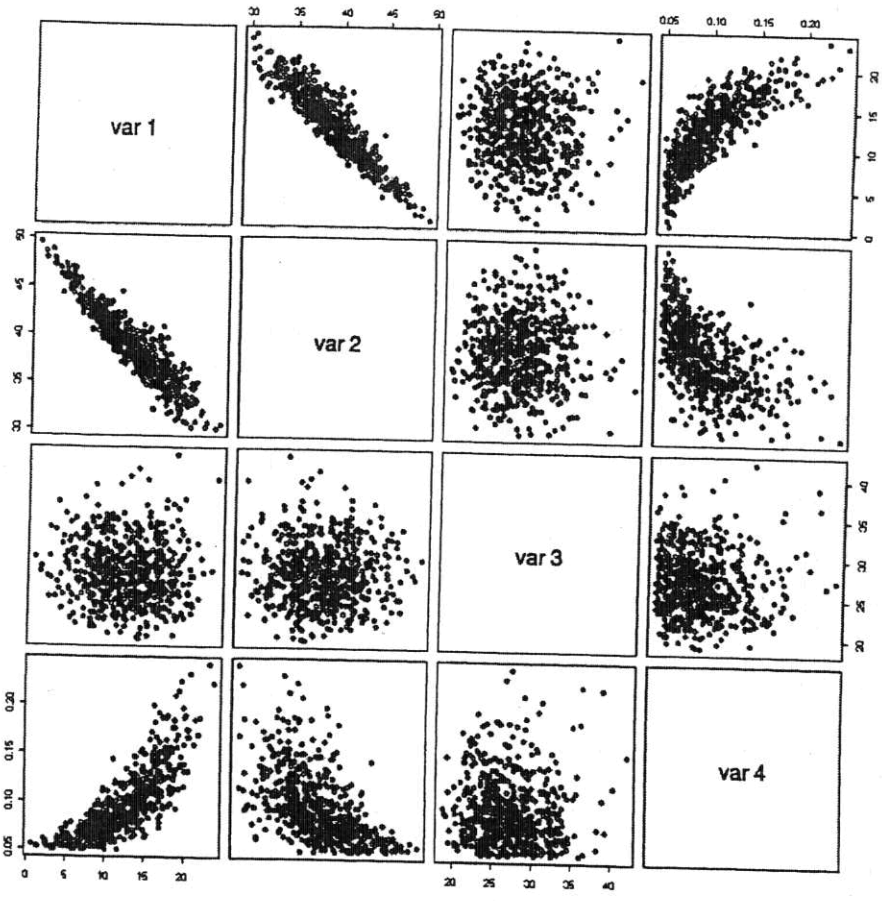

Figure. 5.6. Combinations of posterior estimates of β_0, β_1, σ^2, and θ. The two linear parameters show the expected negative correlation. Correlations among other parameters are low.

Generalized linear regression

For a standard GLM, the parameter for the sampling distribution is just a transformation of the regression parameters. Thus, there is a single, indirect sample to be drawn for the (vector) of parameters β and there are no latent variables to be estimated. Here is an example having a binomial likelihood and a logit link,

$$y_i \sim Bernoulli(\theta_i)$$

$$\text{logit}(\theta_i) = \mathbf{x}_i\beta = \beta_0 + \beta_1 x_i$$

Functions for the logit and inverse logit are useful for this Gibbs sampler:

```
logit <- function(x){log(x/(1-x))}
inv.logit <- function(x){1/(1 + exp(-x))}
```

Although I use a Gaussian prior for the regression parameters, the conditional posterior is not conjugate—the likelihood is Bernoulli. I use a Metropolis algorithm. Here is a simulated data set,

```
n <- 200
x <- cbind(rep(1,n),runif(n,2,100))     #design matrix
b0 <- 5                                 #parameter values
b1 <- -.2
beta <- matrix(c(b0,b1),2,1)            #parameter vector
ltheta <- x %*% beta                    #logit theta
```

```
y <- rbinom(n,1,inv.logit(ltheta))      #the data
```

Here is a vague prior for the slope and intercept parameters, followed by some initial values

```
bprior <- as.vector(c(0,0))    #priors for regression parameters
vmat <- diag(1000,2)           #prior covariance matrix

ngibbs <- 50000
bgibbs <- matrix(NA,nrow=ngibbs,ncol=2)  #for regression parameters
  bgibbs[1,] <- c(2,0)                   #initial values for beta's
  theta <- inv.logit(x %*% bgibbs[1,])   #initial values for theta
  cmat <- .005*rbind(c(1,-.7),c(-.7,1))  #jump distribution
```

Note that the jump distribution (a 2 by 2 covariance matrix) has negative off-diagonal elements such that proposals follow the tendency for negative correlation. Here is the Gibbs loop, which consists of a single Metropolis step:

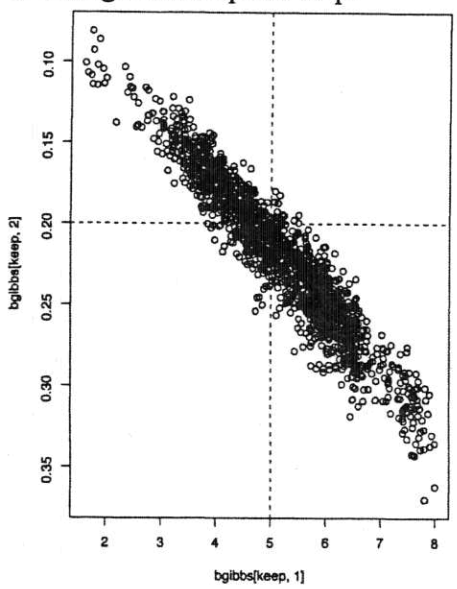

Fig. 5.7. Gibbs samples for the regression parameters in the binomial GLM with values used to simulate data (dashed lines).

```
for(g in 1:(ngibbs-1)){

    bprop <- t(rmvnorm(1,bgibbs[g,],cmat))     #proposed values
    theta.s <- inv.logit(x %*% bprop)

   #log probabilities for current (pnow) and proposed (pnew) values
    pnow <- sum(dbinom(y,1,theta,log=T)) +
dmvnorm(bgibbs[g,],bprior,vmat,log=T)
    pnew <- sum(dbinom(y,1,theta.s,log=T)) +
dmvnorm(t(bprop),bprior,vmat,log=T)

    r <- exp(pnew - pnow)                      #acceptance criterion
    z <- runif(1,0,1)
    bgibbs[(g+1),] <- bgibbs[g,]
    if(z < r)bgibbs[(g+1),] <- bprop           #accept with probability r

    theta <- inv.logit(x %*% bgibbs[(g+1),]) #update theta
}
```

The Gibbs sampler recovers the values used to simulate the data (Fig. 5.7), again with the expected negative correlation.

GLM with extra-Poisson variation

Here I provide an example showing how to implement a generalized linear model with the extension of random error on the underlying linear equation. It demonstrates the combination of Metropolis sampling within Gibbs. I also show how to construct a predictive distribution for a latent variable.

I assume a Poisson sampling distribution, a log link, and a single covariate x, and normally distributed error on the linear equation. In other words, there is extra-Poisson variation that means we also have to model the Poisson mean θ with a deterministic piece and an error term,

$$y_i \sim Pois(\theta_i)$$

$$\ln\theta_i = \mathbf{x}_i\beta = \beta_0 + \beta_1 x_i + \varepsilon_i$$

$$\varepsilon_i \sim N(0,\sigma^2)$$

A full model for n sample points could be written as

$$p(\beta,\theta,\sigma^2|\mathbf{X},\mathbf{y},\mathbf{b}_0,\mathbf{V}_0,s_1,s_2,\ldots) = \prod_{j=1}^{n} Pois(y_i|\theta_i)\prod_{j=1}^{n} N(\ln\theta_i|\mathbf{x}_i\beta,\sigma^2)$$
$$\times N_2(\beta|\mathbf{b}_0,\mathbf{V}_0)IG(\sigma^2|s_1,s_2)$$

Parameters involved in the linear regression can be sampled directly using the same functions as previously.

The θ's are latent variables that I will sample using a Metropolis step. Unlike the previous example, where θ was just a deterministic transformation of β, now θ must be modeled. Because θ must be positive, I want an asymmetric sampling distribution. So that I do not have to correct for an asymmetric jump distribution, I will propose and accept on the log scale. For each value, I draw a proposal from the jump distribution $N(\ln\theta_i^{(g)},0.05)$. The ratio

$$r = \frac{p(\ln\theta_i^*|\mathbf{x}_i,y_i)}{p(\ln\theta_i^{(g)}|\mathbf{x}_i,y_i)} = \frac{Pois(y_i|\theta_i^*)N(\ln\theta_i^*|\mathbf{x}_i\beta^{(g)},(\sigma^2)^{(g)})}{Pois(y_i|\theta_i^{(g)})N(\ln\theta_i^{(g)}|\mathbf{x}_i\beta^{(g)},(\sigma^2)^{(g)})}$$

is the basis for acceptance.

In the following I simulate $n = 100$ data points for a simple slope-intercept model. The design matrix x has a column of ones and a column of random values between 2 and 10. The intercept and slope are `b0` = 0.1 and `b1` = 0.4, respectively, with residual variance `sig` = 0.1. Here is the data simulation:

```
n <- 100
x <- cbind(rep(1,n),runif(n,2,10))          #design matrix
b0 <- .3                                    #parameter values
b1 <- .5
sig <- .1
beta <- matrix(c(b0,b1),2,1)                #parameter vector
ltheta <- rnorm(n,x %*% beta,sqrt(sig))     #ln theta
```

```
y <- rpois(n,exp(ltheta))                    #the data
```

2. Gibbs sampler—Set prior parameter values:

```
bprior <- as.vector(c(0,0))
vinvert <- solve(diag(1000,2))
ppart <- vinvert %*% bprior
s1 <- .1
s2 <- .1
```

Now define vectors and matrices to hold the MCMC output

```
ngibbs <- 10000
bgibbs <- matrix(NA,nrow=ngibbs,ncol=2)    #for regression parameters
  bgibbs[1,] <- bprior
sgibbs <- rep(1,ngibbs)                    #for variance
  sg <- 1
tgibbs <- matrix(NA,nrow=ngibbs,ncol=n)
  th <- y +.1                              #initial values > y
  tgibbs[1,] <- th                         #for theta
```

I will predict a sequence of 50 y values directly within the Gibbs sampler. Define a matrix to hold the 50 values that will be generated at each Gibbs step:

```
xseq <- seq(from=min(x[,2]),to=max(x[,2]),length=50) #x values
predy <- matrix(NA,ngibbs,50)                        #predicted y
values
```

Here's a Gibbs sampler:

```
for(g in 1:ngibbs){

 b <- b.update(log(th),1/sg)
   yp <- x %*% b                           #current mean ln(theta)
 sg <- v.update(log(th),1)

 #theta: Metropolis step
   propt <- rnorm(n,log(th),.05)           #propose n values of ln(theta)
   pstar <- dpois(y,exp(propt),log=T) + dnorm(propt,yp,sqrt(sg),log=T)
   pnow <-  dpois(y,th,log=T) + dnorm(log(th),yp,sqrt(sg),log=T)
   r <- exp(pstar - pnow)                  #acceptance ratio
   z <- runif(n,0,1)
   th[r > z] <- exp(propt[r > z])          #accept some values

  bgibbs[g,] <- b
  sgibbs[g] <- sg
  tgibbs[g,] <- th

  #predict y
    tpred <- rnorm(50,(b[1]+b[2]*xseq),sqrt(sg))
    predy[g,] <- rpois(50,exp(tpred))
 }
```

Because the θ's are conditionally independent, I proposed all values simultaneously and accepted a subset of them. Here are plots of the chains with lines drawn at the values used to simulate data.

```
par(mfrow=c(3,1))
plot(bgibbs[,1],type='l'); abline(h=b0)
plot(bgibbs[,2],type='l'); abline(h=b1)
plot(sqrt(sgibbs),type='l'); abline(h=sqrt(sig))
```

The MCMC output can be processed as before. This nonlinear model with indirect sampling can take longer to converge than the earlier examples, and it contains autocorrelation at long lags (Fig. 5.8). Note that I initialized the θ's near the values of y. Had I initialized the latent states further from the known y's, convergence would have been slowed considerably.

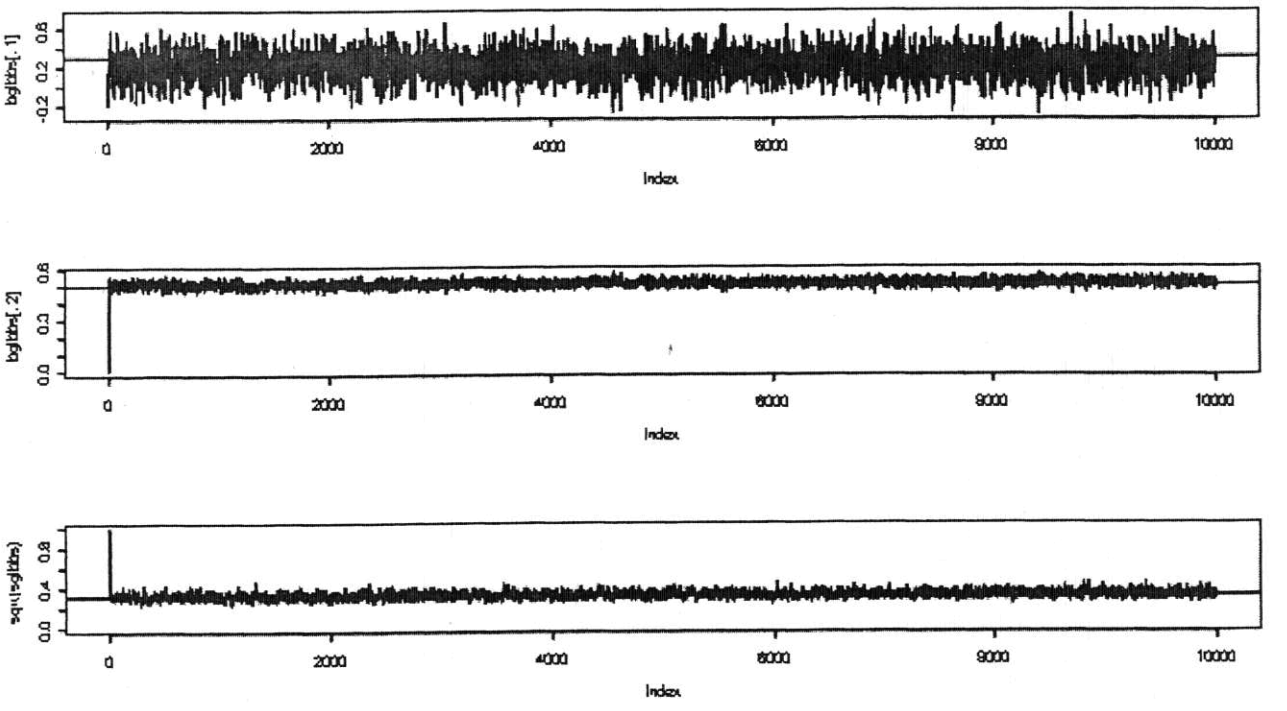

Fig. 5.8. The first 10000 iterations of the Gibbs sampler for the model with extra-Poisson variability and inverse distribution sampling of the latent states θ. Horizontal lines indicate parameter values used to simulate data.

To process the predicted values of y, you might do the following. First define a matrix with 50 rows (one for each prediction) and 3 columns (one for the median and two for the 95% quantiles):

```
ciy <- matrix(NA,50,3)
```

Now determine the quantiles for each prediction level,

```
ciy <- t(apply(predy,2,quantile,c(.5,.025,.975),na.rm=T))
```

These can be plotted with the data as follows (Fig. 5.9):

```
par(mfrow=c(1,1))
plot(xseq,ciy[,1],type='s',ylim=c(0,200))
  lines(xseq,ciy[,2],type='s')
  lines(xseq,ciy[,3],type='s')
  points(x[,2],y)
```

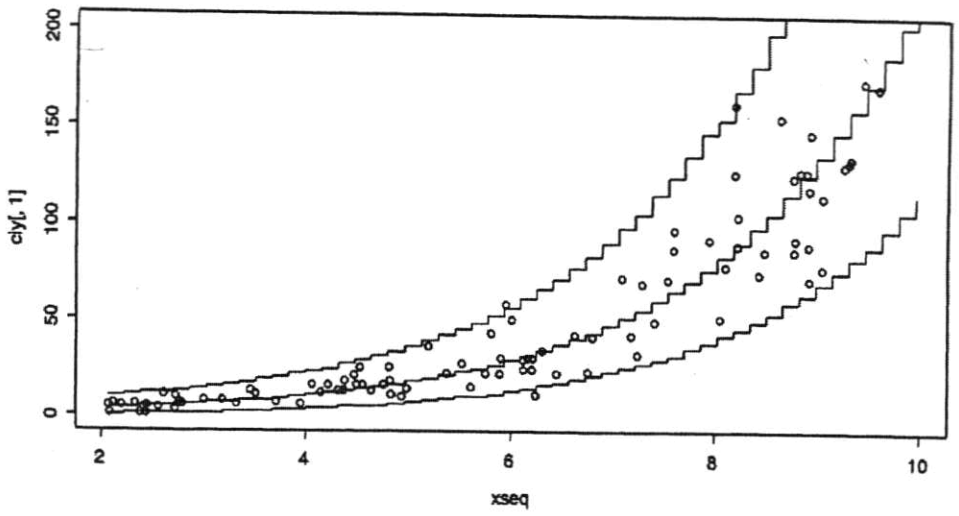

Fig. 5.9. The median prediction and 95% prediction interval for **y** plotted with the data points.

The θ_i values are not observed, so we might also be interested in how well the estimates match the simulated values that were used to general the values of y. In Figure 5.9 are shown predictive distributions plotted with this code:

```
gci <- apply(tgibbs,2,quantile,c(.5,.025,.975))
  plot(exp(ltheta[1]),gci[1,1],xlim=c(0,250),ylim=c(0,250));
lines(rep(exp(ltheta[1]),2),gci[2:3,1])
  for(i in 2:n)lines(rep(exp(ltheta[i]),2),gci[2:3,i])
  abline(0,1)
```

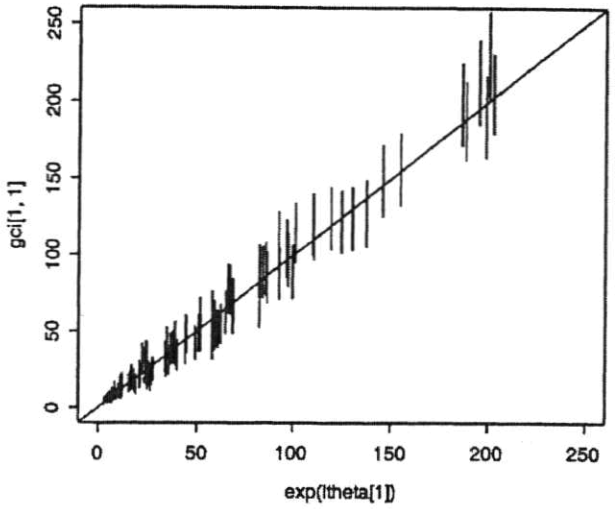

Fig. 5.10. Values of θ_i from the simulated data set (horizontal axis), which are unobserved, plotted with 95% credible intervals from the Gibbs sampler (vertical lines). The 1:1 line is shown.

Autoregression

To demonstrate autoregression, I use example 9.1 from the text. I simulate data for the AR(1) model with a covariate and analyze it with a Gibbs sampler, with an embedded

Metropolis step for the correlation parameter. To simulate data, set the length of the series n, and define parameter values,

```
n <- 50
x <- cbind(rep(1,n),runif(n,0,20))
b0 <- .5                          #intercept parameter
b1 <- 2                           #slope parameter
sig <- 1                          #residual variance
rho <- -.5                        #serial correlation
beta <- matrix(c(b0,b1),2,1)      #vector of regression parameters
```

The correlation matrix is given by eqn 9.9 in the text. The inverse correlation is simple and can be constructed with this function:

```
inv.rmat <- function(rho){
  rinv  <- diag((1+rho^2),n,n)
  rinv[1,1] <- 1; rinv[n,n] <- 1
  rinv[row(rinv) == (col(rinv)-1)] <- -rho
  rinv[row(rinv) == (col(rinv)+1)] <- -rho
  return(rinv)
}
```

From this function, the correlation matrix is obtained as

```
solve(inv.rmat(rho))
```

the inverse covariance matrix is

```
sinv <- inv.rmat(rho)/sig
```

and the covariance matrix $\mathbf{\Sigma} = \sigma^2\mathbf{R}$ is

```
smat <- sig*solve(inv.rmat(rho))
```

We are now ready to simulate y. It can be done in a loop:

```
y <- matrix(0,n,1)
error <- rnorm(1,0,sqrt(sig))
y[1,1] <- t(x[1,]) %*% beta + error
for(t in 2:n){
  error <- rnorm(1,rho*error,sqrt(sig)) #AR(1) error
  y[t,1] <- t(x[t,]) %*% beta + error
}
```

Here is a more direct way to simulate y, with a single draw from the n dimensional multivariate normal:

```
y <- t(rmvnorm(1,(x %*% beta),smat))
```

To propose values of ρ, I use Fisher's transformation,

$$z(\rho) = \frac{1}{2}\ln\left(\frac{1+\rho}{1-\rho}\right)$$

which is symmetric. For this example, I used a prior centered on zero autocorrelation, $N(z|0,1)$. The Gibbs sampler is constructed as previously. Define prior parameter values and arrays to hold estimates:

```
bprior <- c(0,0)                          #priors for regression
parameters
  vinvert <- solve(diag(1000,2))          #inverse covariance for beta's
  ppart <- vinvert %*% bprior
smu <- 2                                  #vague var prior has mean 2
  s1 <- 2
  s2 <- smu*(s1 - 1)
```

```
zprior <- 0                                    #z prior mean
  zsd <- 1                                     #z prior sd

ngibbs <- 10000
bgibbs <- matrix(NA,nrow=ngibbs,ncol=2) #for regression parameters
sgibbs <- rep(1,ngibbs)                        #for variance
rgibbs <- rep(0,ngibbs)                        #for rho
predy <- matrix(NA,ngibbs,n)                   #for predictions of y
rmat <- arlmat(0)
  rg <- 0
  sg <- 1/rgamma(1,s1,s2)                      #initial variance drawn from
prior
  smat <- rmat * sg
  rinv <- inv.rmat(rg)
  sinv <- rinv/sg
```

The Gibbs loop follows:

```
for(g in 1:ngibbs){

  b <- b.update(y,sinv)
  sg <- v.update(y,rinv)

#sample rho with Metropolis
  yp <- t(x %*% b)                             #expected y
  zg <- .5*log((1+rg)/(1-rg))                  #current z
  zs <- rnorm(1,zg,.1)                         #propose z
    rs <- (1-exp(-2*zs))/(1 + exp(-2*zs)) #proposed r
    rinvs <- inv.rmat(rs)                      #proposed inv cor matrix
    smats <- sg*solve(rinvs)                   #covariance for proposal
  pnow <- dmvnorm(t(y),yp,smat,log=T) + dnorm(zg,zprior,zsd,log=T)
  pnew <- dmvnorm(t(y),yp,smats,log=T) + dnorm(zs,zprior,zsd,log=T)
  r <- exp(pnew - pnow)
  z <- runif(1,0,1)
  if(z < r){
     rg <- rs
     smat <- smats
     sinv <- rinvs/sg
     rinv <- rinvs
  }

  bgibbs[g,] <- b
  sgibbs[g] <- sg
  rgibbs[g] <- rg

  predy[g,] <- rmvnorm(1,x%*%b,smat)          #predict y
}
```

Parameter densities are shown in Figure 5.11. Here is code for these densities together with vertical lines for quantiles and lines for values used in the simulation (dashed).

```
par(mfrow=c(2,2))
bdens<-density(bgibbs[,1],width=.5)
  plot(bdens,type='l')                                    #posterior
  abline(v=quantile(bgibbs[,1],c(.5,.025,.975)))          #CI
  abline(v=b0,lty=2)
bdens<-density(bgibbs[,2],width=.05)
  plot(bdens,type='l')
  abline(v=quantile(bgibbs[,2],c(.5,.025,.975)))
```

```
  abline(v=b1,lty=2)
sdens<-density(sgibbs,width=.4)
  plot(sdens,type='l')
  abline(v=quantile(sgibbs,c(.5,.025,.975)))
  abline(v=sig,lty=2)
rdens<-density(rgibbs,width=.2)
  plot(rdens,type='l',xlim=c(-1,1))
  abline(v=quantile(rgibbs,c(.5,.025,.975)))
 abline(v=rho,lty=2)
```

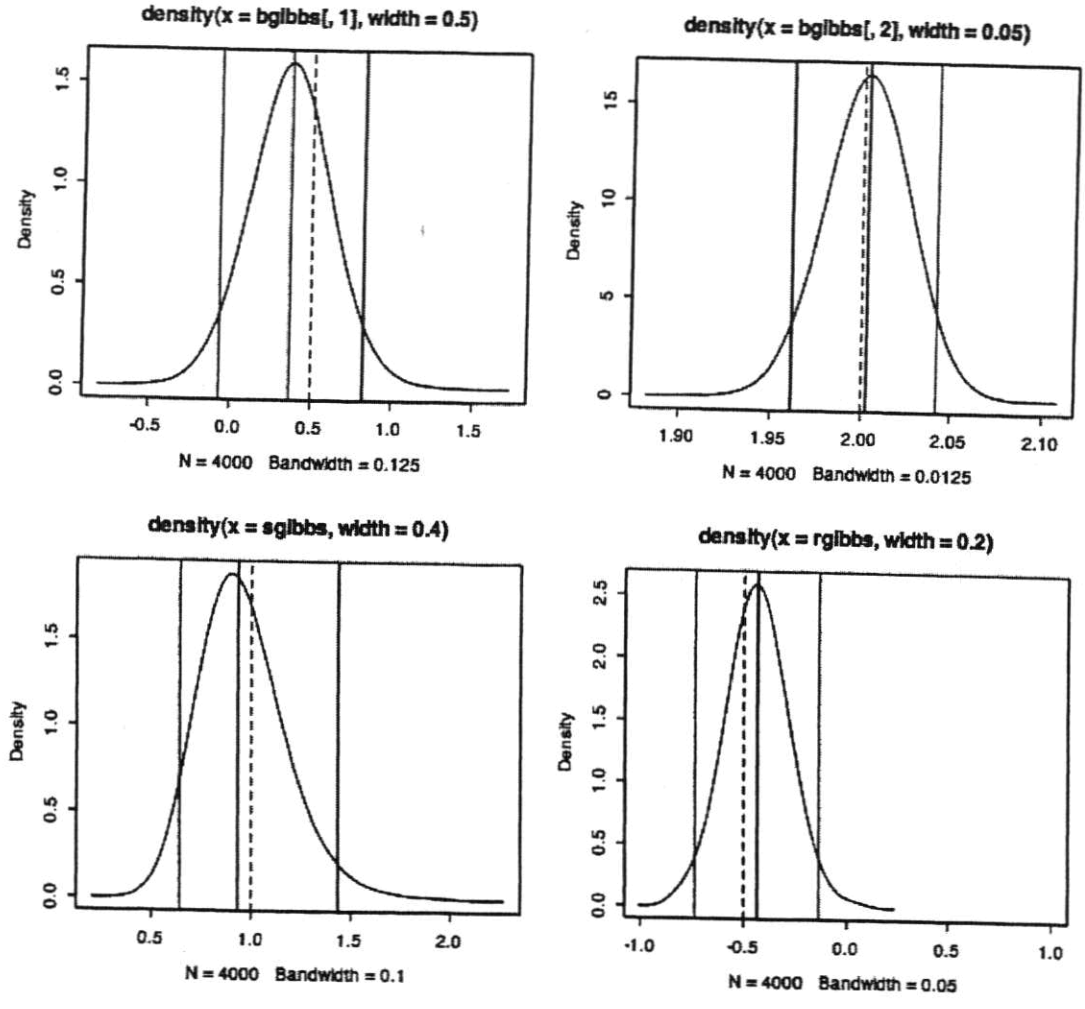

Fig. 5.11. Posterior parameter densities for the AR(1) model from simulated data. Vertical lines are median and 95% credible intervals (solid lines) and values used to simulated data (dashed).

Predictive intervals for **y** can be generated from the Gibbs sampler with some basic R functions:

```
plot(y,ylim=c(min(x,y),max(x,y)))
  lines(x[,2])
ciy <- matrix(NA,n,3)
for(i in 1:n)ciy[i,] <- quantile(predy[,i],c(.5,.025,.975),na.rm=T)
  lines(ciy[,1],lty=2)
  lines(ciy[,2])
  lines(ciy[,3])
for(i in 1:n){
```

```
  xx <- c(i,i)
  yy <- c(ciy[i,2],ciy[i,3])
  lines(xx,yy)
}
```

A plot of observations and predictions is shown in Figure 5.12.

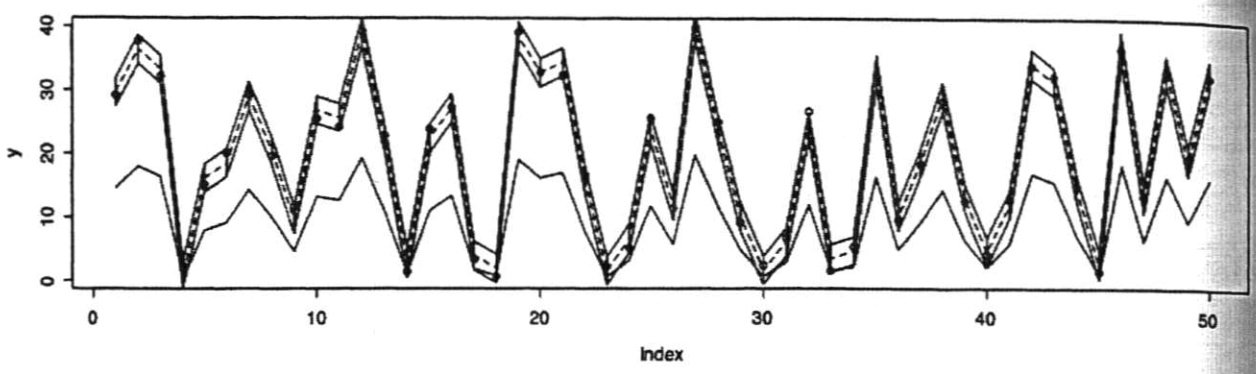

Fig. 5.12. Simulated values of y_t (dots), predicted values of y_t (upper curves bound the 95% prediction interval), and values of x_t (lower curve).

Bayesian kriging with a GLM

In this example, I consider a binomial response that depends on an underlying 'risk' surface $\theta(x(\mathbf{s}_i))$, where risk varies smoothly in space with some variable that can be measured x. The vector for the i^{th} location is given by $\mathbf{s}_i = [s_{1i}, s_{2i}]$. There is spatial correlation, which, for simplicity, is taken to be exponential,

$$\rho(r_{ij};\phi) = e^{-\phi r_{ij}} \qquad \phi > 0$$

where r_{ij} is the distance between locations i and j. The goal is to estimate this underlying risk and to generate a predictive distribution for it across a map. The only information available to us are the binomial responses $y(\mathbf{s}_i)$, which have been observed at n known locations, and the covariate x, which is known for the full map, i.e., not just for the sample locations $\mathbf{s}$, but also for the prediction grid $\mathbf{s}'$.

I also introduce here use of the truncated normal prior for situations where parameter values should be zero outside some limits. If we know that risk declines with exposure to x, then a logit model $\text{logit}(\theta(\mathbf{s}_i)) = \mathbf{x}(\mathbf{s}_i)\beta = \beta_0 + \beta_1 x(\mathbf{s}_i)$ can have priors for (β_0, β_1) that are positive and negative, respectively, i.e.,

$$N_2(\beta|\mathbf{b},\mathbf{V}_b)I(\mathbf{b}_a < \beta < \mathbf{b}_b)$$

with lower limit vector $\mathbf{b}_a = [0,-10]^{\text{T}}$ and upper limit $\mathbf{b}_b = [10,0]^{\text{T}}$. The -10 and 10 are low and high enough to be inconsequential in this example. A function to draw from this bivariate normal makes use of the univariate truncated normal introduced earlier in this chapter. Here are two functions, one of which calls `tnorm`:

```
btrunc.update <- function(y,sinv){
  ss <- t(x) %*% sinv
    sx <- ss %*% x
    sy <- ss %*% y
  bigv <- solve(sx + vinvert)
  smallv <- sy + vinvert %*% bprior
```

```
  b <- tnorm.mvt(b,bigv%*%smallv,bigv,c(0,-10),c(10,0))
  return(b)
}
tnorm.mvt <- function(amat,mumat,smat,lo,hi){ #truncated multvariate
normal
  #amat: matrix of current values (Gibbs sampling)
  #mumat: matrix of means
  #smat: covariance matrix
  #lo and hi: vector of lower and upper limits
  muvec <- as.vector(mumat)
  avec <- as.vector(amat)
  avec[avec < lo] <- lo[avec < lo]
  avec[avec > hi] <- hi[avec > hi]
  for(k in 1:length(muvec)){
    piece1 <- smat[-k,k] %*% solve(smat[-k,-k])
    muk <- muvec[k] + piece1 %*% (avec[-k] - muvec[-k])
    sgk <- smat[k,k] - piece1 %*% smat[k,-k]
    avec[k] <- tnorm(1,lo[k],hi[k],muk,sqrt(sgk))
  }
  return( matrix(avec,nrow(mumat),ncol(mumat)) )
}
```

To improve behavior of the MCMC I propose the correlation and variance parameters simultaneously. The covariance matrix $\mathbf{C}_\phi$ has elements

$$C_{ij} = \text{cov}\left(y(\mathbf{s}_i), y(\mathbf{s}_j)\right) = \sigma^2 \rho(r_{ij};\phi)$$

These parameters will have negative correlation, because large σ^2 increases variance everywhere, and ϕ translates covariance to large distances. By drawing proposals from a bivariate log normal with negative covariance we can more efficiently explore the posterior. The jump density

$$J = N_2\left(\begin{bmatrix}\ln(\sigma^2)\\ \ln(\phi)\end{bmatrix}, 0.02\begin{bmatrix}1 & -0.2\\ -0.2 & 1\end{bmatrix}\right)$$

has correlation of -0.2. I use lognormal priors for both. Here it is:

```
jump <- .02*rbind(c(1,-.2),c(-.2,1))
```

The model is

$$Bin\left(\mathbf{y}_s|\mathbf{n}_s,\theta_s\right)N_n\left(\text{logit}(\theta_s)|\mathbf{X}_s\beta,\mathbf{C}_\phi\right)N_2\left(\beta|\mathbf{b},\mathbf{V}_b\right)I\left(\mathbf{b}_a < \beta < \mathbf{b}_b\right)$$

$$N\left(\ln(\sigma^2)|\ln(3),1\right)N\left(\ln(\phi)|\ln(1),1\right)$$

I illustrate with simulated data. First is a function to calculate distances between sites. The arguments to `distmat` are vectors of coordinates for two sets of locations. The first two vectors give Cartesian coordinates for the first set of locations (`x1, y1`). The last two vectors provide the same for the second set of locations (`x2, y2`). If both sets of locations are the same, `distmat` returns the symmetric distance matrix.

```
distmat <- function(x1,y1,x2,y2){
    xd <- outer(x1,x2,function(x1,x2) (x1 - x2)^2)
    yd <- outer(y1,y2,function(y1,y2) (y1 - y2)^2)
    d <- t(sqrt(xd + yd))
    return(d)
}
```

Simulated data y come from n sites, with the number of individuals at risk at each site being neach, and located at **s** = (s1x, s2x). I am thinking of this sample area as being 10 km by 10 km. The n by n distance matrix is dist:

```
n <- 100
neach <- rpois(n,10)
s1x <- runif(n,0,10)
s2x <- runif(n,0,10)
dist <- distmat(s1x,s2x,s1x,s2x)
```

Here is an arbitrary function for a spatial trend in the covariate, represented as a map in the upper left of Figure 5.12:

```
x1 <- .5*s1x^2 + 4*s2x - .3*s1x*s2x
```

This is incorporated into the design matrix and, with assumed values for other parameters, used to simulate first the θ's, then the y's,

```
x <- cbind(rep(1,n),x1)                          #design matrix
b0 <- 3                                          #regression parameters
b1 <- -.2
beta <- matrix(c(b0,b1),2,1)
sig <- 3                                         #variance parameter s2
cpar <- 2                                        #correlation distance
parameter
smat <- sig*exp(-cpar*dist)                      #covariance matrix

ltheta.t <- matrix(rmvnorm(1,(x %*% beta),smat),n,1) #logit(theta)
theta.t <- inv.logit(ltheta.t)                       #theta
y <- rbinom(n,neach,theta.t)                         #response
```

Here are some prior and initial values and arrays to hold Gibbs chains:

```
bprior <- as.vector(c(2,-1))
vinvert <- solve(diag(1000,2))
ppart <- vinvert %*% bprior
rprior <- log(1)
  rvar <- 1
sprior <- log(3)
  svar <- 1

ngibbs <- 200000
keep <- seq(2000,ngibbs,by=20)                   #thin predictions
kp <- 0                                          #counter
bgibbs <- matrix(NA,nrow=length(keep),ncol=2)    #regression parameters
sgibbs <- rep(0,length(keep))                    #variance parameter
  sg <- rlnorm(1,sprior,sqrt(svar))              #initial variance from
prior
cgibbs <- sgibbs                                 #correlation distance
parameter
  cg <- rlnorm(1,rprior,sqrt(rvar))              #initial value from prior
cgmat <- exp(-cg*dist)                           #correlation matrix
  sgmat <- sg*cgmat                              #covariance matrix
  sginv <- solve(sgmat)                          #inv covar matrix
```

To initialize the values of θ, I arbitrarily set them equal to the fractions observed, and then set zeros and ones to 0.05 and 0.95, respectively:

```
theta <- y/neach                    #initialize theta
  theta[theta < .05] <- .05
  theta[theta > .95] <- .95
```

```
  ltheta <- logit(theta)
```

In other words, I assumed that they could be approximated by the fraction of individuals at each location that died, `y/neach`.

I now establish a prediction grid across the map and a matrix to hold a set of predictions for all grid points at each Gibbs step. I initially assumed the map spanned 10 km on a side, so the grid here will span values from zero to ten. The size of the grid is $m = 11 \times 11$. The predictive distributions will have mean vector and covariance matrix

$$E\left(\text{logit}(\theta_{s'})|\theta_s,\mathbf{X},\beta,\phi\right) = \mathbf{X}_{s'}\beta + \mathbf{C}_{s,s'}^T\mathbf{C}_s^{-1}\left(\text{logit}(\theta_s) - \mathbf{X}_s\beta\right)$$

$$\text{var}\left(\text{logit}(\theta_{s'})|\theta_s,\mathbf{X},\beta,\phi\right) = \mathbf{C}_{s'} - \mathbf{C}_{s,s'}^T\mathbf{C}_s^{-1}\mathbf{C}_{s,s'}$$

where s' indicates locations on the prediction grid, $\mathbf{C}_{s'}$ is the m by m covariance matrix for the prediction grid, and $\mathbf{C}_{s,s'}$ is the n by m matrix of covariances between the n sample locations and the m locations on the grid. The grid has a row for each Gibbs step and a column for each of m locations on the grid.

```
predgrid <- matrix(0,length(keep),(11*11))#prediction grid
  s1p <- rep(c(0:10),each=11)              #horizontal locations
  s2p <- rep(c(0:10),times=11)             #vertical locations
  distxp <- distmat(s1x,s2x,s1p,s2p)       #distance matrix for grid
  distp <- distmat(s1p,s2p,s1p,s2p)        #distance matrix for samples
to grid
  xpred <- .5*s1p^2 + 4*s2p - .3*s1p*s2p   #values of x on grid
  xp <- cbind(rep(1,11*11),xpred)          #design matrix for grid
```

Note that x is known for both the observed and predicted locations; I have simply calculated the values of x for the prediction grid.

Here is the Gibbs loop:

```
for(g in 1:ngibbs){

  b <- btrunc.update(ltheta,sginv)              # truncated normal
    yp <- x %*% b

  cpars <- rmvnorm(1,c(log(sg),log(cg)),jump) # variance and corr
distance
    ss <- exp(cpars[1])
    cs <- exp(cpars[2])
      cmats <- exp(-cs*dist)
      sgmats <- ss*cmats
    pnow <- dmvnorm(t(ltheta),yp,sgmat,log=T)  +

dmvnorm(c(log(sg),log(cg)),c(sprior,rprior),diag(c(svar,rvar)),log=T)
    pnew <- dmvnorm(t(ltheta),yp,sgmats,log=T) +

dmvnorm(c(log(ss),log(cs)),c(sprior,rprior),diag(c(svar,rvar)),log=T)
    r <- exp(pnew - pnow)
    z <- runif(1,0,1)
    if(z < r){
      sg <- ss
      cg <- cs
      cgmat <- exp(-cg*dist)
      sgmat <- sg*cgmat
      sginv <- solve(sgmat)
    }
```

```
  #theta (Metropolis)
    ltheta.s <- rnorm(n,ltheta,.01)  #propose theta's
    theta.s <- inv.logit(ltheta.s)
    pnow <- sum(dbinom(y,neach,theta,log=T)) +
            dmvnorm(t(ltheta),yp,sgmat,log=T)
    pnew <- sum(dbinom(y,neach,theta.s,log=T)) +
            dmvnorm(t(ltheta.s),yp,sgmat,log=T)
    r <- exp(pnew - pnow)
    z <- runif(1,0,1)
    if(z < r)theta <- theta.s
    ltheta <- logit(theta)

  if(g %in% keep){              #save predictions for post-burnin/thinned
   kp <- kp + 1
   cxp <- sg*exp(-cg*distxp)    #covariance, samples to grid
   cp  <- sg*exp(-cg*distp)     #covariance for grid
   mpred <- xp %*% b + cxp %*% sginv %*%(ltheta - x %*% b) #predict
mean
   vpred <- cp - cxp %*% sginv %*% t(cxp)                    #predict var
   predgrid[kp,] <- inv.logit(rmvnorm(1,mpred,vpred))
   bgibbs[kp,] <- b
   sgibbs[kp] <- sg
   cgibbs[kp] <- cg
   print(c(g,b,sg,cg))
  }
}
```

With the exception of β_1, the 95% CIs include parameter values used to simulate the data:

```
parout <- t(rbind(apply(cbind(bgibbs,sgibbs,cgibbs),2,mean),

apply(cbind(bgibbs,sgibbs,cgibbs),2,quantile,c(.025,.975))) )
  rownames(parout) <- c("b0","b1","spar","cpar")

parout
                      2.5%       97.5%
b0    2.2615062  0.8445751  3.80470925
b1   -0.1413186 -0.1877823 -0.09711453
spar  2.0962420  1.1353654  3.71326841
cpar  1.5119922  0.4750455  3.23391855
```

11:07 AMhe predictive distribution of θ is the basis for constructing the map of upper deciles. Code to produce the maps in Figure 5.13 is:

```
qpred <- apply(predgrid,2,quantile,c(.5,.1,.9))

par(mfrow=c(2,2))
contour(c(0:10),c(0:10),matrix(xpred,11,11))
  title("predictor x")
contour(c(0:10),c(0:10),matrix(qpred[1,],11,11),levels=c(.1,.3,.5,.7,.9
))
  symbols(s1x,s2x,circles=(y/neach)*.4,add=T,inches=F)
  title("risk surface theta")
contour(c(0:10),c(0:10),matrix(qpred[3,],11,11),levels=c(.1,.3,.5,.7,.9
))
  title("risk: upper decile")
```

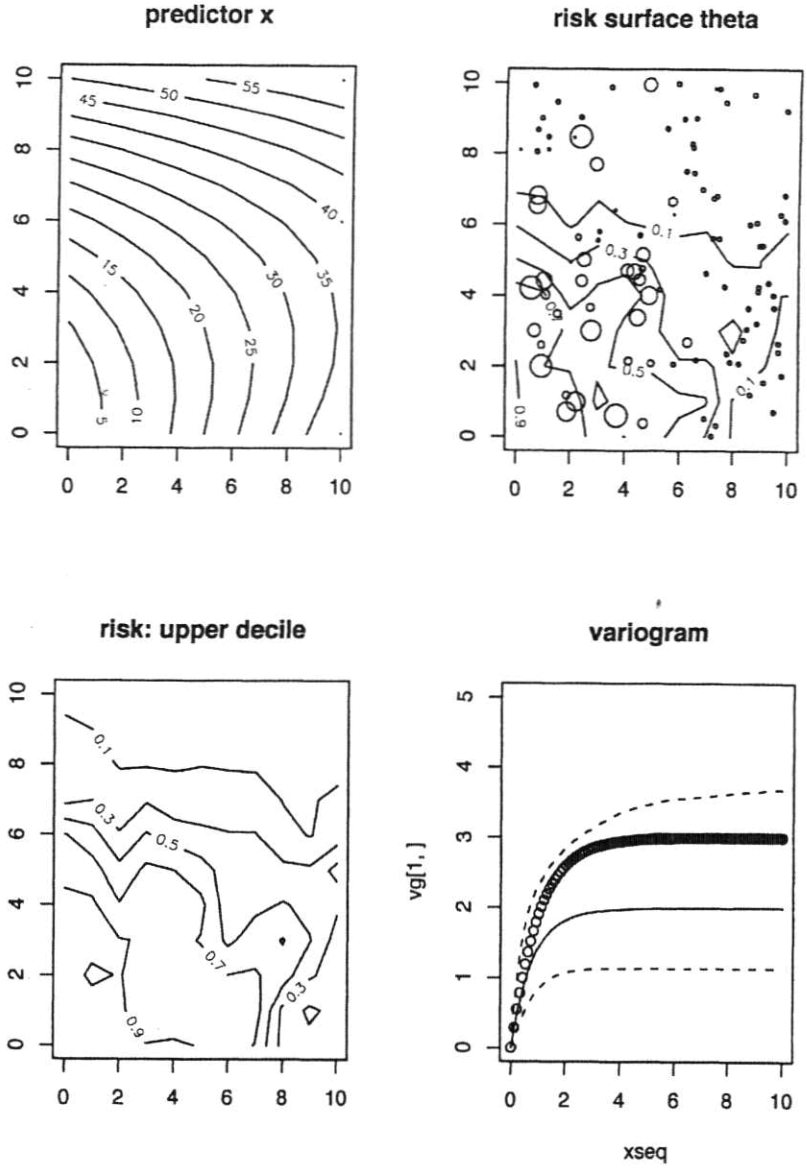

Figure 5.13. Baysian kriging example, with the surface for the predictor variable **x**, which is known (upper left), the estimated θ surface (posterior median), which decreases with increasing **x**, and observations **y** (contours and circles at upper right), the 'upper decile' for risk, or the posterior belief that risk could be as high as 90% (lower left), and estimated semivariogram (posterior median and 95% CI as lines) with that used to simulate the data (dots)(lower right).

The predictive distribution on the semivariogram function is constructed by mixing over the Gibbs chains for $[\sigma^2, \phi]$. Here is code:

```
xseq <- seq(0,10,length=100)
variogram <- matrix(0,length(keep),100)
for(g in 1:length(keep))variogram[g,] <- sgibbs[g]*(1 - exp(-
cgibbs[g]*xseq))
vg <- apply(variogram,2,quantile,c(.5,.025,.975))
plot(xseq,vg[1,],type='l',ylim=c(0,5))
  lines(xseq,vg[2,],lty=2)
  lines(xseq,vg[3,],lty=2)
  points(xseq,sig*(1 - exp(-xseq)))
  title("variogram")
```

This particular example has no "nugget" variance and, thus, has zero intercept for the semivariogram.

Exercises

Construct a Gibbs sampler for a GLM with extra-Poisson variation for the pinecone model, with covariates being tree diameter and CO2 treatment. The model is:

$$y_i \sim Pois(\theta_i)$$

$$\ln\theta_i = \mathbf{x}_i\beta = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \varepsilon_i$$

$$\varepsilon_i \sim N(0,\sigma^2)$$

where covariates are diameter (x_{1i}) and CO_2 treatment (x_{2i} is 0 for ambient and 1 for elevated CO_2). With priors, a full model could be:

$$p(\beta,\theta,\sigma^2|\mathbf{X},\mathbf{y},\ldots) = \prod_{j=1}^{n} Pois(y_i|\theta_i)\prod_{j=1}^{n} N(\ln\theta_i|\mathbf{x}_i\beta,\sigma^2)$$

$$\times N_3(\beta|\mathbf{b},\mathbf{V}_b)IG(\sigma^2|s_1,s_2)$$

The Gibbs sampler will be just like the example for the generalized linear model for the Poisson of this chapter, with the exception that you will have one more covariate. Using your Gibbs sampler, construct posterior distributions for parameters.