Sampling methods

Søren Højsgaard

Department of Mathematical Sciences

Aalborg University, Denmark

June 3, 2014

Contents

1 Introduction – Bayesian modelling

- In a Bayesian setting, parameters are treated as random quantities on equal footing with the random variables.
- The joint distribution of a parameter (vector) θ and data (vector) y is specified through a prior distribution $\pi(\theta)$ for θ and a conditional distribution $p(y \mid \theta)$ of data for a fixed value of θ.
- This leads to the joint distribution for data AND parameters

$$p(y, \theta) = p(y \mid \theta)\pi(\theta)$$

- The prior distribution $\pi(\theta)$ represents our knowledge (or uncertainty) about θ before data have been observed.
- After observing data y, the posterior distribution $\pi^*(\theta)$ of θ is obtained by conditioning with data which gives

$$\pi^*(\theta) = p(\theta|y) = \frac{p(y|\theta)\pi(\theta)}{p(y)} \propto L(\theta)\pi(\theta)$$

  where $L(\theta) = p(y \mid \theta)$ is the likelihood and the marginal density $p(y) = \int p(y \mid \theta)\pi(\theta)d\theta$ is the normalizing constant.
- Often we are interested in the posterior mean of some function $g(\theta)$:

$$\mathbb{E}(g(\theta)|\pi^*) = \int g(\theta)\pi^*(\theta)d\theta$$

  Examples: $\mathbb{E}(\theta|\pi^*)$ or $\mathbb{V}\text{ar}(\theta|\pi^*)$.
- However, usually $\pi^*(\theta)$ can not be found analytically because the normalizing constant $p(y) = \int p(y \mid \theta)\pi(\theta)d\theta$ is intractable.
- In such cases one will often resort to sampling based methods: If we can draw samples $\theta^{(1)}, \dots, \theta^{(N)}$ from $\pi^*(\theta)$ we can do just as well:

$$\mathbb{E}(g(\theta)|\pi^*) \approx \frac{1}{N}\sum_i g(\theta^{(i)})$$

- The question is then how to draw samples from $\pi^*(\theta)$ where $\pi^*(\theta)$ is only known up to the normalizing constant.
- There are many methods for achieving this; these methods are known as Markov Chain Monte Carlo (MCMC) methods and will be described elsewhere.
- Sections marked with "*" in the following can be skipped at first reading.

2 Computations using Monte Carlo methods

Consider a random vector X with density / probability mass function $p(x)$ which is the TARGET DISTRIBUTION (from which we want to sample).

In many real world applications

- we can not directly draw samples from p.
- p is only known up to a constant of proportionality; that is

  $$p(x) = k(x)/c$$

  where $k()$ is known and the normalizing constant c is unknown.

We reserve $h(x)$ for a PROPOSAL DISTRIBUTION which is a distribution from which we can draw samples.

2.1 Rejection sampling

Let $p(x) = k(x)/c$ be a density where $k()$ is known and c is unknown. Let $h(x)$ be a proposal distribution.

Suppose we can find a constant M such that $k(x) < Mh(x)$ (i.e. $k(x)/M < h(x)$ and hence $\frac{k(x)/M}{h(x)} < 1$) for all x. The algorithm is then

1. Draw sample $x \sim h()$. Draw $u \sim U(0, 1)$.
2. Set $\alpha = \frac{k(x)/M}{h(x)}$
3. If $u < \alpha$, accept x.

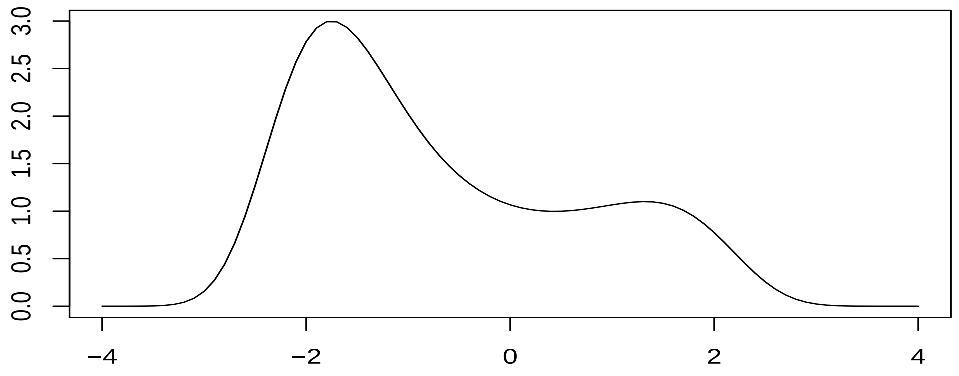
The accepted values $x^1, \dots x^N$ is a random sample from $p(\cdot)$.

Notice:

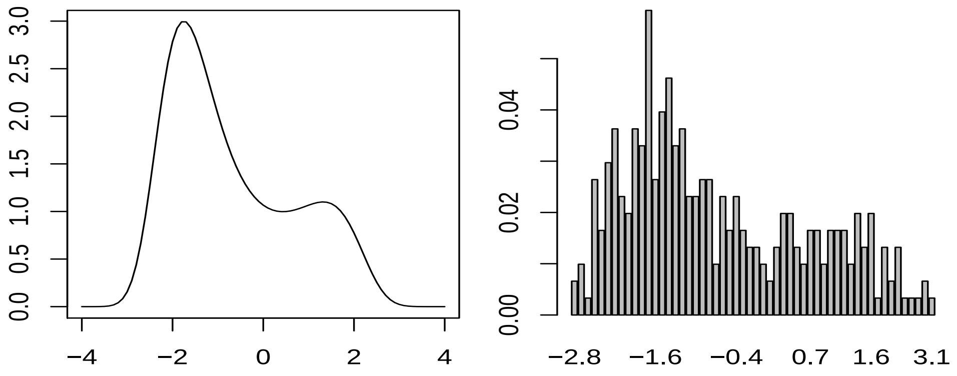
- It is tricky to choose a good proposal distribution $h()$. It should have support at least as large as $p()$ and preferably heavier tails than $p()$.
- It is desirable to choose M as small as possible which is difficult in practice. Hence one often chooses a large value of M whereby only few proposed values are accepted so it is difficult to make rejection sampling efficient.

2.2 Example: Rejection sampling

```
> k <- function(x, a=.4, b=.08){exp(a*(x-a)^2 - b*x^4)}
> x <- seq(-4, 4, 0.1)
> plot(x,k(x),type="l")
```



```
> # uniform proposal on [-4,4]:
> h <- function(x){rep.int(0.125,length(x))}
> # we can find M in this case:
> M   <- round(max(k(x)/h(x))) + 1; M

[1] 25

> # number of samples
> N   <- 1000
> # generate proposals and u
> x.h <- runif( N, -4, 4 )
> u   <- runif( N )
> acc <- u < k(x.h) / (M * h(x.h))
> x.acc <- x.h[ acc ]
> # how many proposals are accepted
> sum( acc ) /N

[1] 0.335

> # calculate some statistics
> c(m=mean(x.acc), s=sd(x.acc))

       m        s
-0.60006  1.42031

> par(mfrow=c(1,2), mar=c(2,2,1,1))
> plot(x,k(x),type="l")
> barplot(table(round(x.acc,1))/length(x.acc))
```



2.3 QUIZ

Reuse the code from above to answer these questions, but please think about what the results would be before executing the code.

- Suppose we could not easily determine M and hence had to make a conservative choice; say $M = 100$ or $M = 500$ in this context.

  Which effect would that have on the number of accepted samples, and how would you have to compensate?

- Suppose we take the proposal distribution $h()$ to be uniform om $[-10, 10]$. Which effect would that have on the acceptance rate? What if the proposal distribution is an $N(0, 1)$? What is the quality of the samples in this case? Hint: Use dnorm() to evaluate the normal density.

2.4 Sampling importance resampling (SIR)*

When M is not readily available, we may generated approximate samples from p as follows.

1. Draw samples $x^1, \dots x^N \sim h(x)$.
2. Calculated importance weights $w_i = p(x^i)/h(x^i)$.
3. Normalize the weights as $q_i = w_i / \sum_j w_j$.
4. Resample from $\{x^1, \dots x^N\}$ where y^i is drawn with probability q_i.

The samples obtained in 4. are approximately samples from p.

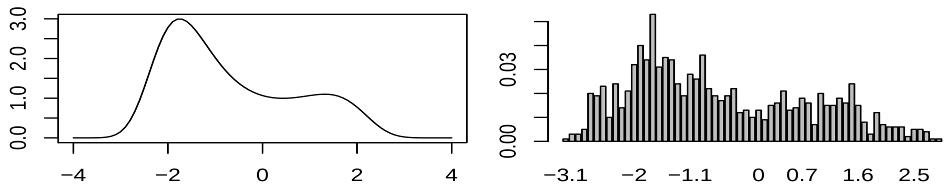
Notice:

- This scheme works also if p is only known up to proportionality (because the normalizing constant cancels out in step 3. above).

- Samples from h which "fits best to p" are those most likely to appear in the resample. However, if h is a poor approximation to p then the "best samples from h" are not necessarily good samples in the sense of resembling p.

2.5 Example: Sampling importance resampling (SIR)*

```
> h <- function(x){rep.int(0.125,length(x))}
> N   <- 1000
> x.h <- runif( N, -4, 4 )
> u   <- runif( N )
> ww  <- k(x.h) / h(x.h)
> qq  <- ww / sum(ww)
> x.acc <-sample(x.h, prob=qq, replace=T)
> par(mfrow=c(2,2), mar=c(2,2,1,1))
> plot(x,k(x),type="l")
> barplot(table(round(x.acc,1))/length(x.acc))
```



3 Markov Chain Monte Carlo methods

- A drawback of the rejection algorithm and the SIR–algorithm is that it is difficult to suggest a proposal distribution h which leads to an efficient algorithm.
- For the rejection algorithm, it is also difficult to find M.
- A way around this problem is to let the proposed values depend on the last accepted values: If x' is a "likely" value from p then so is probably also a proposed value x which is "close" to x'.
- Hence the proposal distribution will now be conditional on the last accepted value and have the form $h(x|x')$.
- This leads schemes (described below) for drawing samples $x^1, \dots, x^N$ and these samples will, under certain conditions, form an ergodic Markov chain with $p(x)$ as its stationary distribution.

- Hence, the expected value of any function of x can be calculated approximately as

$$\int g(x)p(x)dx \approx \frac{1}{N}\sum_i g(x^i).$$

- The samples $x^1, \ldots, x^N$ will typically be correlated because the value x^j will be generated from $h(\cdot|x^{j-1})$ and will hence depend on x^{j-1}.

3.1 The Metropolis–Hastings (MH) algorithm

Given an accepted value x^{t-1}:

1. Draw $x \sim h(\cdot|x^{t-1})$. Draw $u \sim U(0,1)$.
2. Calculate acceptance probability $\alpha = \min\left(1, \frac{p(x)}{p(x^{t-1})}\frac{h(x^{t-1}|x)}{h(x|x^{t-1})}\right)$
3. If $u < \alpha$ then set $x^t = x$; else set $x^t = x^{t-1}$.

After a burn–in period the samples $x^1, x^2, \ldots$ will be samples from $p(\cdot)$.

Notice:

- The samples $x^1, x^2, \ldots$ will be correlated.
- The algorithm also works if p is only known up to proportionality (because the normalizing constant cancels when calculating the acceptance probability).
- We must be able to both sample from $h()$ and evaluate the density.

3.2 Special cases of the Metropolis–Hastings algorithm

Metropolis algorithm (a special case of the Metropolis-Hastings algorithm) The proposal distribution is symmetrical, i.e. $h(x|x') = h(x'|x)$ for all pairs (x, x'). Hence the acceptance probability is $\alpha = \min\left(1, \frac{p(x)}{p(x^{t-1})}\right)$.

Random–walk Metropolis A popular choice for proposal in a Metropolis algorithm is $h(x|x') = g(x - x')$ where g is symmetric, e.g.

$$x = x' + e \quad e \sim g$$

Example: $x = x' + N(0, \sigma^2)$

The independence sampler (A special case of the Metropolis–Hastings algorithm) The proposal $h(x|x') = h(x)$ does not depend on x'.) The acceptance probability becomes $\alpha = \min\left(1, \frac{p(x)}{p(x^{t-1})}\frac{h(x^{t-1})}{h(x)}\right)$. For this sampler to work well, h should be a good approximation to p.

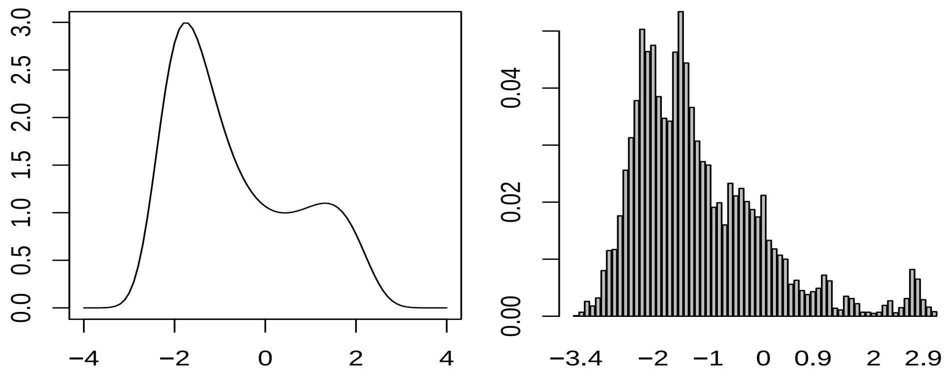
3.3 Example: Metropolis–Hastings algorithm

Random walk Metropolis is straight forward to implement

```
> N         <- 10000
> x.acc5    <- rep.int(NA, N)
> u         <- runif(N)
> acc.count <- 0
> std       <- 0.05  ## Spread of proposal distribution
> xc        <- 0;    ## Starting value
> for (ii in 1:N){
    xp    <- rnorm(1, mean=xc, sd=std) ## proposal
    alpha <- min(1, (k(xp)/k(xc)) *
                 (dnorm(xc, mean=xp,sd=std)/dnorm(xp, mean=xc,sd=std)))
    x.acc5[ii] <- xc <- ifelse(u[ii] < alpha, xp, xc)
    ## find number of acccepted proposals:
    acc.count  <- acc.count + (u[ii] < alpha)
  }
> ## Fraction of accepted *new* proposals
> acc.count/N
```

```
[1] 0.9846
```

```
> par(mfrow=c(1,2), mar=c(2,2,1,1))
> plot(x,k(x),type="l")
> barplot(table(round(x.acc5,1))/length(x.acc5))
```



3.4 Capture–recapture revisited

Consider again the capture–recapture model for estimating population size.

	recaptured	not recaptured	
marked	m=20	n-m=80	n=100
unmarked	u=180	?	U ?
total	R=200	?	N ?

We assume

$$m \sim bin(n, \theta), \quad u \sim bin(U, \theta)$$

So we get

$$p(m|\theta) \sim bin(n, \theta) \quad p(u|\theta, U) \sim bin(U, \theta)$$

Hence as before we get

$$p(m, u|\theta, U) = p(m|\theta)p(u|\theta, U)$$

The likelihood is:

$$\begin{aligned} p(m, u|\theta, U) &= L(\theta, U) \\ &= \binom{n}{m}\theta^m(1-\theta)^{n-m}\binom{U}{u}\theta^u(1-\theta)^{U-u} \\ &\propto \binom{U}{u}\theta^{m+u}(1-\theta)^{n+U-(m+u)} \end{aligned}$$

To complete the model specification we must specify prior distributions for θ and U. These must reflect our prior knowledge of the problem.

The joint density of data (m, u) and the parameters (θ, U) is then

$$p(m, u, \theta, U) \propto \binom{U}{u}\theta^{m+u}(1-\theta)^{n+U-(m+u)}\pi_\theta(\theta)\pi_U(U)$$

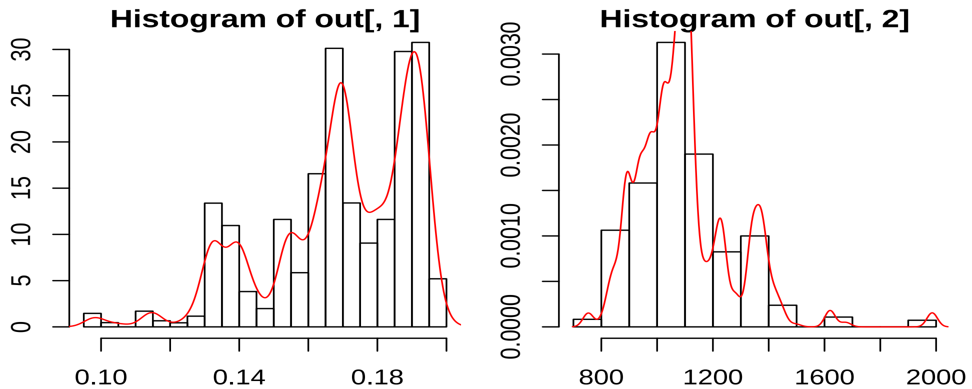
The posterior is proportional to the joint density

$$p(\theta, U|m, u) \propto \binom{U}{u}\theta^{m+u}(1-\theta)^{n+U-(m+u)}\pi_\theta(\theta)\pi_U(U)$$

To fit in with the current notation let $x_1 = \theta$, $x_2 = U$ and $x = (x_1, x_2)$. Also notice that data (m, u) is fixed so we need not write that in the posterior.

$$p^*(x_1, x_2) \propto \binom{x_2}{u}x_1^{m+u}(1-x_1)^{n+x_2-(m+u)}\pi_{x_1}(x_1)\pi_{x_2}(x_2) = k(x_1, x_2)$$

```
> logk <- function(x1, x2, n_, m_, u_){
      R_  <- m_ + u_
      R_*log(x1) + (n_+x2-R_)*log(1-x1) + lchoose (x2, u_) +
          + log(dunif(x1, .0, .2)) + log( disc.pmf(x2, 500, 2000))
  }
> disc.pmf <- function(x, a, b){
      ifelse (x>=a & x<=b, 1/(b-a+1), 0)
  }
> n_ <- 100
> m_ <- 20
> u_ <- 180
> NN <- 10000  ## Number of samples
> u  <- runif(NN)
> th.prop <- runif(NN, .0, 0.5)
> U.prop  <- sample(300:3000, NN, replace=T)
> out <- matrix(NA, NN,2)
> xc <- c(0.2, 1500)
> acc.count <- 0
> for (i in 1:NN){
      xp <- c( th.prop[i], U.prop[i] )
      alpha <- min(1, exp(logk(xp[1], xp[2], n_, m_, u_) -
                          logk(xc[1], xc[2], n_, m_, u_)))
      xc <- if(u[i]<alpha) xp else xc
      out[i, ] <- xc
      acc.count <- acc.count + (u[i]<alpha)
  }
> acc.count / NN ## Not impressive acceptance ratio
[1] 0.0128
> summary(out[,1])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.103   0.158   0.168   0.166   0.180   0.200
> summary(out[,2])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    813     991    1090    1110    1200    1750
> par(mfrow=c(1,2), mar=c(2,2,1,1))
> hist(out[,1], prob=T); lines(density(out[,1]), col="red")
> hist(out[,2], prob=T); lines(density(out[,2]), col="red")
```



3.5 Quiz

Using the code from the slides, experiment with the following:

- Set $m = 2$ and $u = 18$. How does that effect the posterior distribution? What if you set $m = 40$ and $u = 360$?
- Experiment with narrowing and widening the range of the proposal distributions. Which effect does that have on the output?
- Try chaning the prior for U to a poisson distribution. Hint: `dpois` is your friend.
- Experiment with changing the number of samples. How many do you need to produce "nice" histograms?

3.6 Single component Metropolis–Hastings

Instead of updating the entire vector x it is often more convenient and computationally efficient to update x in blocks.

We partition x into blocks, for example $x = (x_1, x_2, x_3)$.

Suppose that we have a sample $x^{t-1} = (x_1^{t-1}, x_2^{t-1}, x_3^{t-1})$ and also that x_1 has also been updated to x_1^t in the current iteration. The task is to update x_2.

To do so we specify a proposal distribution h_2 from which we can sample candidate values for x_2:

1. Draw $x_2 \sim h_2(\cdot|x_1^t, x_2^{t-1}, x_3^{t-1})$. Draw $u \sim U(0, 1)$.
2. Calculate acceptance probability $\alpha = \min\left(1, \frac{p(x_2|x_1^t, x_3^{t-1})}{p(x_2^{t-1}|x_1^t, x_3^{t-1})} \frac{h_2(x_2^{t-1}|x_1^t, x_2, x_3^{t-1})}{h_2(x_2|x_1^t, x_2^{t-1}, x_3^{t-1})}\right)$
3. If $u < \alpha$ set $x_2^t = x_2$; else set $x_2^t = x_2^{t-1}$.

Notice:

- Item 3. can be restated as: With probability α set $x_2^t = x_2$; with probability $1 - \alpha$ set $x_2^t = x_2^{t-1}$.
- If we can choose h_2 such that α is close to 1 then we have an efficient sampler.

3.7 The Gibbs sampler

Consider the acceptance probability for single component Metropolis–Hastings for updating x_2:

$$\alpha = \min\left(1, \frac{p(x_2|x_1^t, x_3^{t-1})}{p(x_2^{t-1}|x_1^t, x_3^{t-1})} \frac{h_2(x_2^{t-1}|x_1^t, x_2, x_3^{t-1})}{h_2(x_2|x_1^t, x_2^{t-1}, x_3^{t-1})}\right)$$

The Gibbs sampler is a special case of single component Metropolis–Hastings, namely the case where the proposal distribution $h_2(x_2|x_1^t, x_2^{t-1}, x_3^{t-1})$ for updating x_2 is chosen to be

$$p(x_2|x_1^t, x_3^{t-1})$$

Hence for the Gibbs sampler the proposed values are always accepted.

One version of the algorithm is as follows. Suppose a sample $x^t = (x_1^t, x_2^t, x_3^t)$ is available.

1. Sample $x_1^{t+1} \sim p(x_1|x_2^t, x_3^t)$
2. Sample $x_2^{t+1} \sim p(x_2|x_1^{t+1}, x_3^t)$
3. Sample $x_3^{t+1} \sim p(x_3|x_1^{t+1}, x_2^{t+1})$
4. Set $x^{t+1} = (x_1^{t+1}, x_1^{t+2}, x_1^{t+3})$

The sequence $x^1, x^2, \dots$ then consists of (correlated) samples from $p(x)$.

Notice:

- The proposed values are always accepted (because $\alpha = 1$), so the sampler is very efficient.
- The sampler requires that we can sample from the conditionals $p(x_i|x_{-i})$. In some cases this is easy; in some cases this is difficult. In general; slice sampling can be used (and this is what JAGS does).

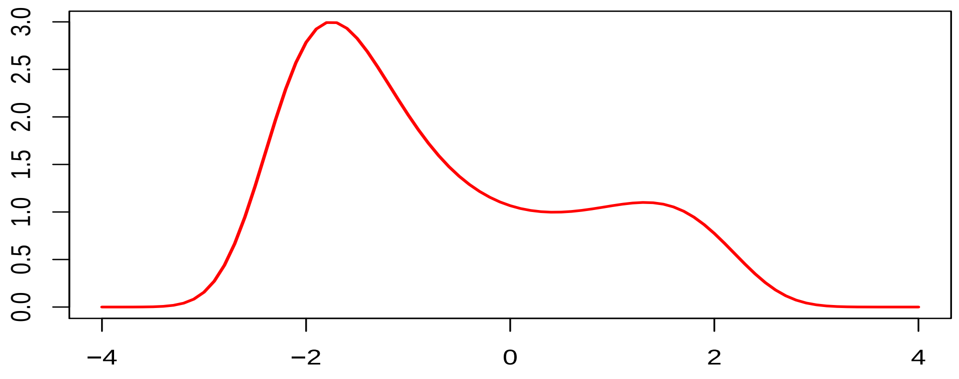
3.8 Slice sampling

Suppose we want to sample from $p(x_i|x_{-i})$ where $x_{-i} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots x_K)$.

Since x_{-i} is fixed we can regard $p(x_i|x_{-i})$ as a function of x_i alone; call this function $k_i(x_i)$ and recall that $k_i()$ is an unnormalized density.

Slice sampling is a simple approach for sampling from an unnormalized density.

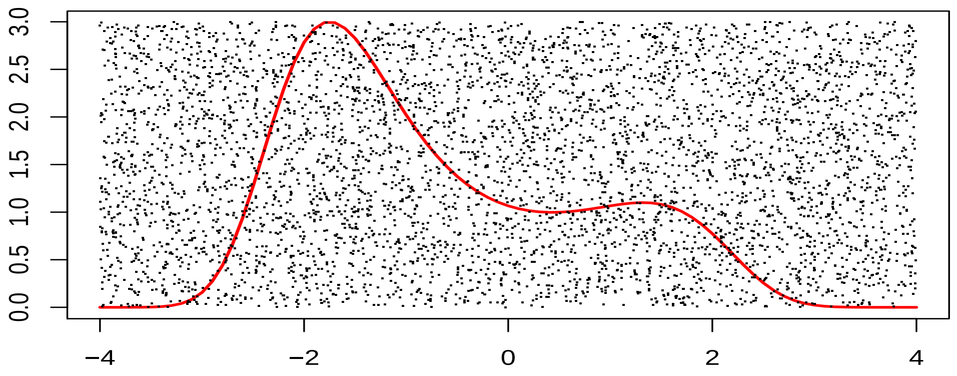
```
> k <- function(x, a=.4, b=.08){exp(a*(x-a)^2 - b*x^4)}
> plot(x,k(x), type='l', lwd=2, col=2)
```



Notice: $k()$ is practically zero outside $[-4, 4]$ and in this interval $k()$ takes values in, say $[0, 3]$.

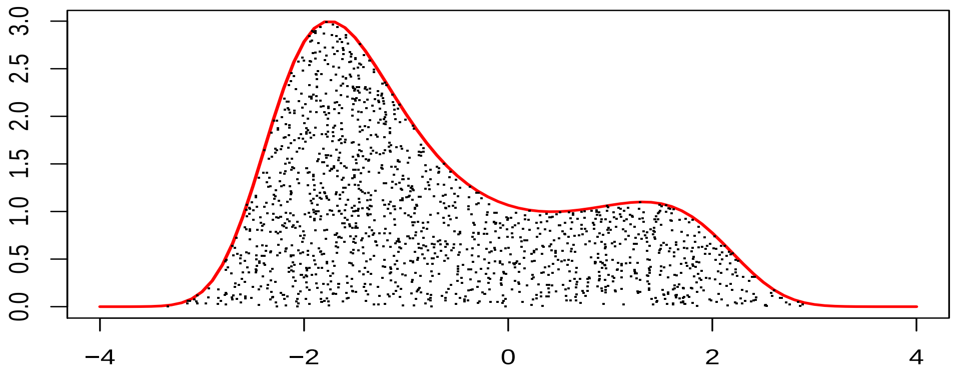
Slice sampling is based on the following idea: Sample uniformly in a "large enough" window:

```
> N <- 5000
> xs <- runif(N, -4, 4)
> ys <- runif(N, 0, 3)
> plot(x,k(x), type='l', lwd=2, col=2)
> points(xs,ys, pch=".")
```

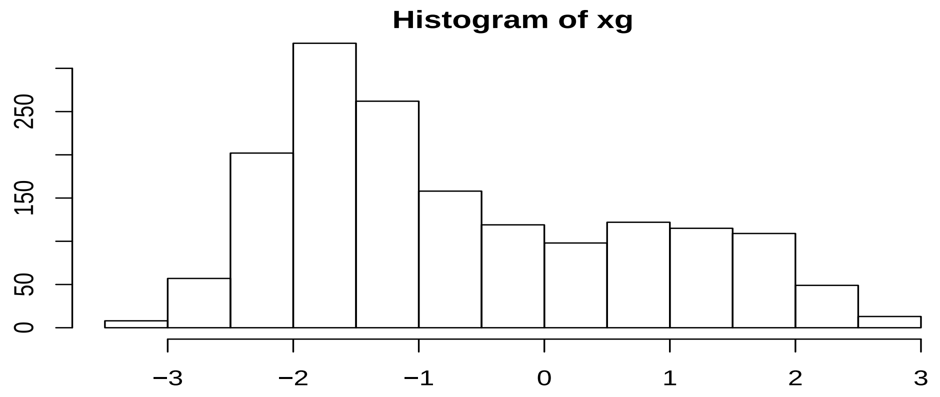


Keep those samples that fall under the curve.

```
> good <- ys<k(xs)
> xg <- xs[good]
> yg <- ys[good]
> plot(x,k(x), type='l', lwd=2, col=2)
> points(xg, yg, pch='.')
```

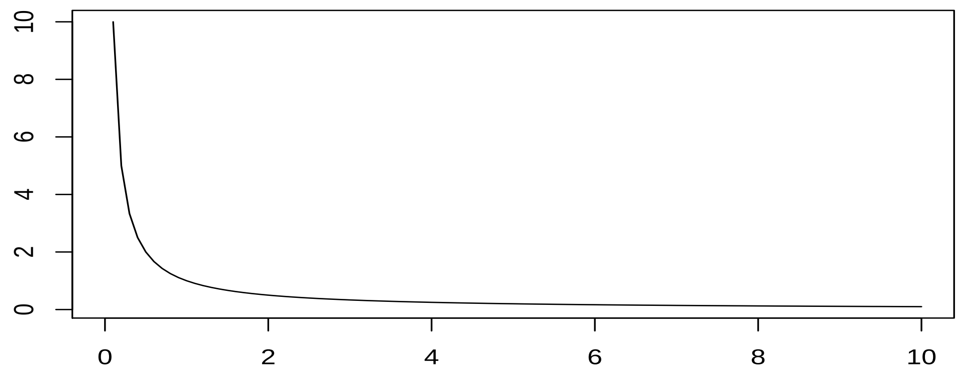


```
> hist(xg)
```



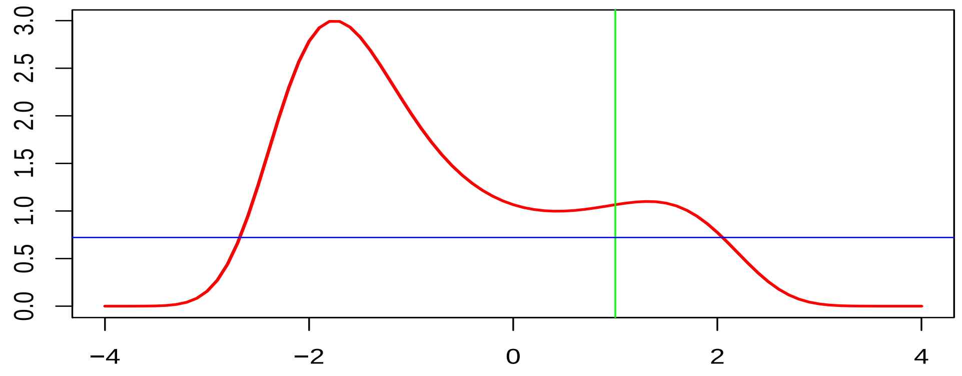
Obviously inefficient for other type of distributions, for example

```
> x2 <- seq(0,10, 0.1)
> k2 <- function(x){1/x}
> plot(x2, k2(x2), type='l')
```



Algorithm goes as follows: Given sample x^t. Pick y uniformly in $[0, k(x^t)]$.

```
> xt<-1; y <- runif(1, 0, k(xt))
> plot(x,k(x), type='l', lwd=2, col=2)
> abline(v=xt, col='green'); abline(h=y, col='blue')
```



Let set $S = \{x : k(x) \geq y\}$ of x–values for which $k(x)$ is below the curve. Sample x^{t+1} uniformly from S.

1. Sample y uniformly from $]0, k(x^t)]$. Defines a horisontal "slice" $S = \{x : k(x) \geq y\}$.
2. Find interval $I = [L, R]$ containing all (or much) of the slice.
3. Sample x^{t+1} uniformly from the part of the slice within I.

The last two steps can be implemented in many ways. We need an interval width w (chosen by us).

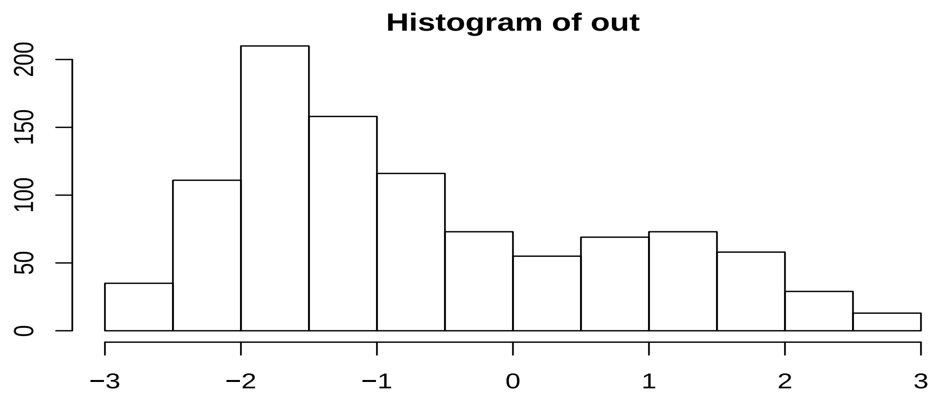
"Stepping out": Position interval of length w randomly around x^t. Denote interval by $I = [L, R]$. Expand both ends in steps of size w until both ends are outside the slice, i.e. until $k(L) < y$ and $k(R) < y$. Sample x^{t+1} from the part of the slice within I. (That is, sample uniformly from I; if a sample is outside S just sample again).

"Doubling": Position interval of length w around x^t. Denote interval by $I = [L, R]$. Double the interval until both ends are outside the slice, i.e. until $k(L) < y$ and $k(R) < y$. Sample x^{t+1} from the part of the slice within I.

```
> sliceSample_real<- function(k, xc, w){
      kc<-k(xc)
      y <-runif(1, 0, kc)
      a <- runif(1)  ## place w randomly around xc
      l <- xc-a*w
      u <- xc+(1-a)*w
      kl <- k(l)
      while (kl>y){  ## expand interval to the left if necessary
          l <- l-w; kl <- k(l)
      }
      ku <- k(u)
      while(ku>y){   ## expand interval to the right if necessary
          u <- u+w; ku <- k(u)
      }
      xp <- runif(1, l, u) ## propose xp
      kp <- k(xp)
      while(kp<y){   ## shrink interval if possible
          if (xp<xc) l <- xp else u <- xp
          xp <- runif(1, l, u)
          kp <- k(xp)
      }
      xp
  }
```

```
> N   <- 3000
> out <- rep.int(NA, N)
> x   <- 1
> for (i in 1:1000){
      x <- sliceSample_real(k, x, w=1)
      out[i] <- x
  }
```

```
> hist( out )
```



3.9 Towards omnibus software

With the slice sampling method, it is now clear why general purpose software (such as JAGS) can be constructed.

We need a collection of slice samplers:

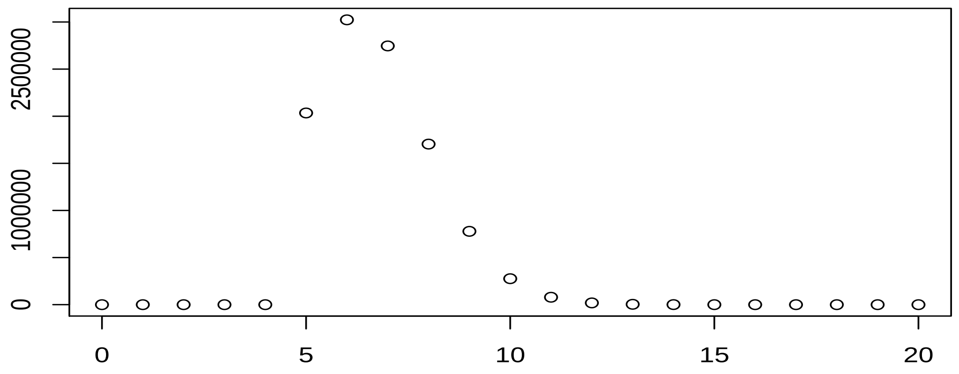
- Sampling on the real line
- Sampling on the unit interval $[0, 1]$
- Sampling on non–negative real values
- Sampling on the non–negative integers $0, 1, 2, \dots$
- Sampling on $a, a + 1, a + 2, \dots a + b$
- ... and more

3.10 Sampling from a (truncated) poisson

Suppose we want to sample from a (truncated) poisson distribution given as

$$p(x) \propto k(x) = I(x \geq L)I(x \leq U)\frac{\lambda^x}{x!}e^{-\lambda} \propto I(x \geq L)I(x \leq U)\frac{\lambda^x}{x!}$$

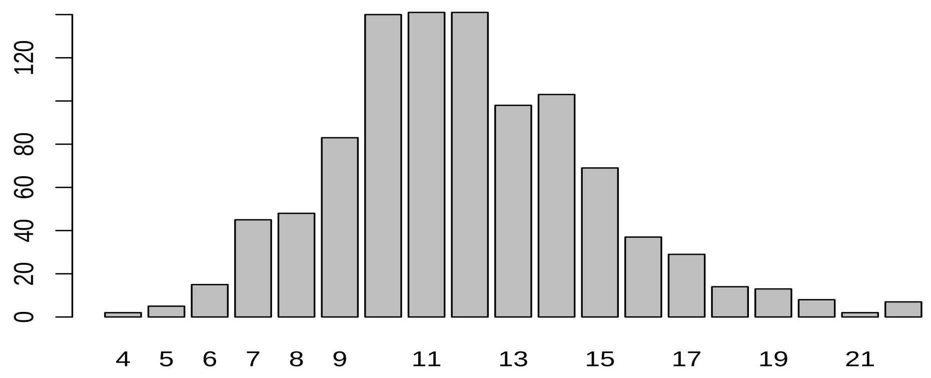
```
> k <- function(x, lambda=12, L=5, U=15){
      (x>=L)*(x<=U)*x^lambda/factorial(x)
  }
> x <- seq(0,20)
> plot(x, k(x))
```



```
> sliceSample_int <- function(k, xc, w=5){
      a <- runif(1, 0, 1)
      l <- floor( xc-a*w )
      u <- ceiling( xc+(1-a)*w )
      #cat(sprintf("init  l=%d, u=%d\n",l,u))
      y <- runif(1, 0, k(xc))
      kl <- k(l)
      while(kl>y){
          l <- l-w; kl <- k(l)
          #cat(sprintf("lower: l=%d, u=%d\n",l,u))
      }
      ku <- k(u)
      while(ku>y){
          u <- u+w; ku <- k(u)
          #cat(sprintf("upper: l=%d, u=%d\n",l,u))
      }
      z <- l:u # sample uniformly on [l,u]
      xp <- sample(z, 1)
      kp <- k(xp)
      while(kp<y){xp <- sample(z, 1);kp <- k(xp)}
      xp
  }
```

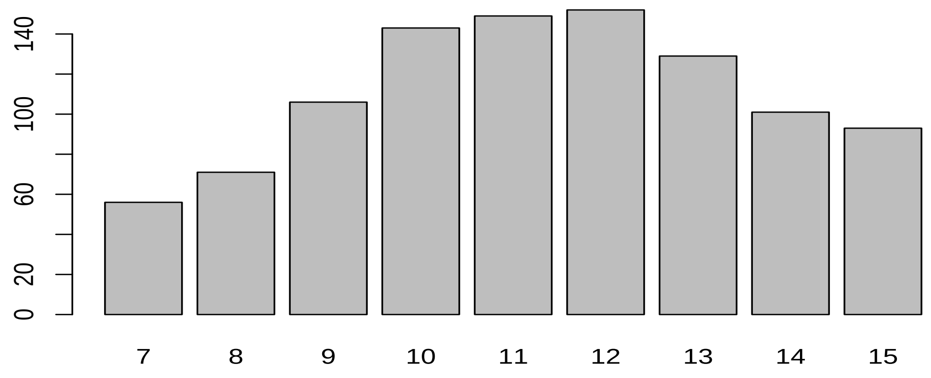
Sampling from poisson:

```
> N <- 1000
> out <- rep.int(NA, N)
> x <- 10 # Some starting value
> k <- function(x, lambda=12, L=0, U=150){
      if(x<0) 0
      else
          (x>=L)*(x<=U)*lambda^x/factorial(x)
  }
> for (i in 1:N){
      out[i] <- sliceSample_int(k, x, w=5)
  }
> barplot(table(out))
```



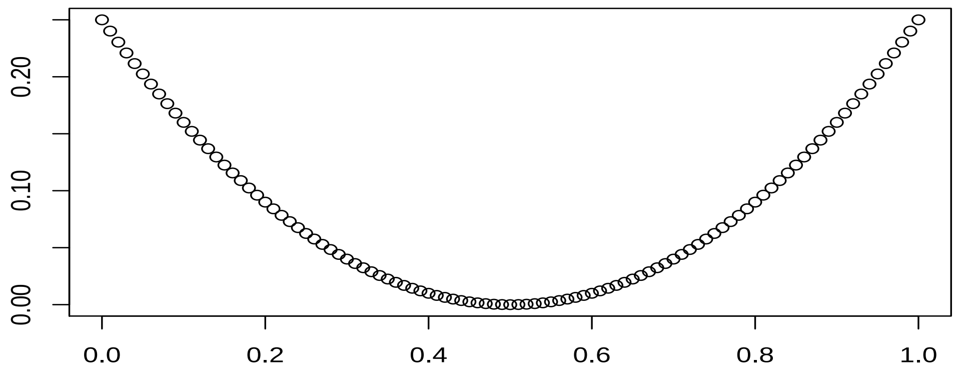
Sampling from truncated poisson:

```
> N <- 1000
> out <- rep.int(NA, N)
> x <- 10 # Some starting value
> k <- function(x, lambda=12, L=7, U=15){
      if(x<0) 0
      else
          (x>=L)*(x<=U)*lambda^x/factorial(x)
  }
> for (i in 1:N){
      out[i] <- sliceSample_int(k, x, w=5)
  }
> barplot(table(out))
```

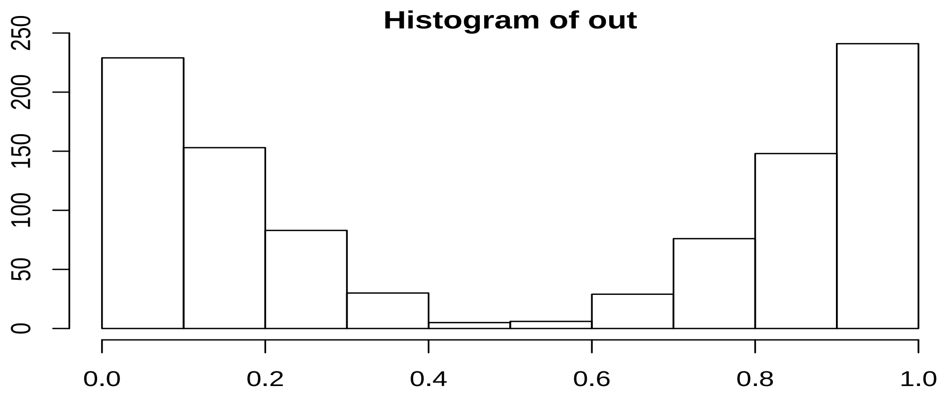


3.11 Sampling on the unit interval

```
> k <- function(x){-x*(1-x)+.25}
> x <- seq(0,1,0.01)
> plot(x,k(x))
```



```
> sliceSample_unit<- function(k, xc, w){
      kc<-k(xc)
      y <-runif(1, 0, kc)
          xp <-runif(1, 0, 1)
          kp <- k(xp)
          while (kp<y){xp<-runif(1,0,1); kp<-k(xp)}
          xp
  }
> N <- 1000
> out <- rep(NA, N)
> for (i in 1:N){
      x <- sliceSample_unit(k, x, w);
      out[i] <- x
  }
> hist(out)
```



3.12 Capture–recapture revisited

Consider again the capture–recapture model for estimating population size.

We assume

$$m \sim bin(n, \theta), \quad u \sim bin(U, \theta)$$

	recaptured	not recaptured	
marked	m=20	n-m=80	n=100
unmarked	u=180	?	U ?
total	R=200	?	N ?

So we get

$$p(m|\theta) \sim bin(n,\theta) \quad p(u|\theta,U) \sim bin(U,\theta)$$

Hence as before we get

$$p(m,u|\theta,U) = p(m|\theta)p(u|\theta,U)$$

The likelihood is:

$$\begin{aligned} p(m,u|\theta,U) &= L(\theta,U) \\ &= \binom{n}{m}\theta^m(1-\theta)^{n-m}\binom{U}{u}\theta^u(1-\theta)^{U-u} \\ &\propto \binom{U}{u}\theta^{m+u}(1-\theta)^{n+U-(m+u)} \end{aligned}$$

To complete the model specification we must specify prior distributions for θ and U. These must reflect our prior knowledge of the problem.

The joint density of data (m,u) and the parameters (θ,U) is then

$$p(m,u,\theta,U) \propto \binom{U}{u}\theta^{m+u}(1-\theta)^{n+U-(m+u)}\pi_\theta(\theta)\pi_U(U)$$

The posterior is proportional to the joint density

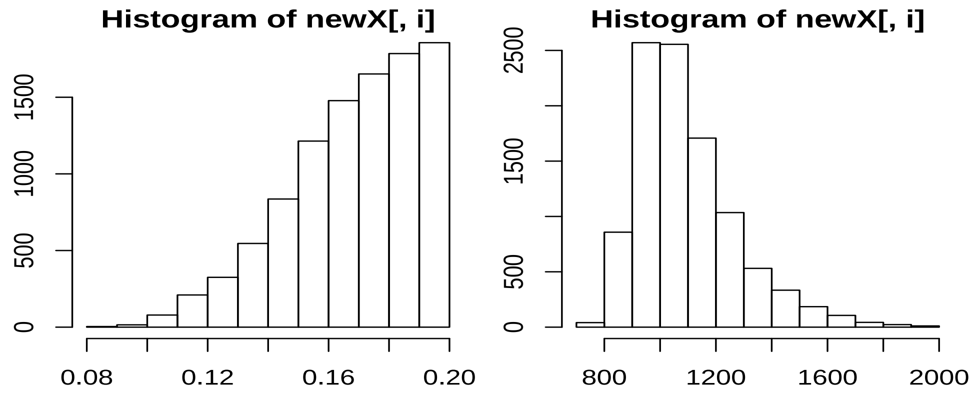
$$p(\theta,U|m,u) \propto \binom{U}{u}\theta^{m+u}(1-\theta)^{n+U-(m+u)}\pi_\theta(\theta)\pi_U(U)$$

To fit in with the current notation let $x_1 = \theta$, $x_2 = U$ and $x = (x_1,x_2)$. Also notice that data (m,u) is fixed so we need not write that in the posterior.

$$p^*(x_1,x_2) \propto \binom{x_2}{u}x_1^{m+u}(1-x_1)^{n+x_2-(m+u)}\pi_{x_1}(x_1)\pi_{x_2}(x_2) = k(x_1,x_2)$$

```
> k <- function(x1, x2, n_, m_, u_){
      R_  <- m_ + u_
      z<-R_*log(x1) + (n_+x2-R_)*log(1-x1) + lchoose (x2, u_) +
          + log(dunif(x1, .0, .2)) + log( disc.pmf(x2, 500, 2000))
      exp(z)
  }
> disc.pmf <- function(x, a, b){
      ifelse (x>=a & x<=b, 1/(b-a+1), 0)
  }
> n_ <- 100
> m_ <- 20
> u_ <- 180
> library(doBy)
> kk <- specialize(k, list(n_=n_, m_=m_, u_=u_))
> # Now kk is function only of x1, x2
> args(kk)
function (x1, x2)
NULL
> kk
function (x1, x2)
{
    R_ <- 20 + 180
    z <- R_ * log(x1) + (100 + x2 - R_) * log(1 - x1) + lchoose(x2,
        180) + +log(dunif(x1, 0, 0.2)) + log(disc.pmf(x2, 500,
        2000))
    exp(z)
}
<environment: 0x07e1c188>
> N <- 10000
> x1t <- .1    # initial values
> x2t <- 1000  # initial values
> out <- matrix(NA, N, 2)
> kk1 <- specialize(kk, list(x2=x2t)); kk1
function (x1)
{
    R_ <- 20 + 180
    z <- R_ * log(x1) + (100 + 1000 - R_) * log(1 - x1) + lchoose(1000,
        180) + +log(dunif(x1, 0, 0.2)) + log(disc.pmf(1000, 500,
        2000))
    exp(z)
}
<environment: 0x07db16e0>
> kk2 <- specialize(kk, list(x1=x1t)); kk2
function (x2)
{
    R_ <- 20 + 180
    z <- R_ * log(0.1) + (100 + x2 - R_) * log(1 - 0.1) + lchoose(x2,
        180) + +log(dunif(0.1, 0, 0.2)) + log(disc.pmf(x2, 500,
        2000))
    exp(z)
}
<environment: 0x07d95200>
```

```
> for (i in 1:N){
      x1t <- sliceSample_unit(kk1, x1t, w=1)
      kk2 <- specialize(kk, list(x1=x1t))
      x2t <- sliceSample_int(kk2, x2t, w=10)
      kk1 <- specialize(kk, list(x2=x2t))
      out[i,] <-     c(x1t,x2t)
  }
> par(mfrow=c(1,2))
> z<-apply(out, 2, hist)
```



3.13 Work on the log–scale

For numerical reasons it is generally better to work on a log scale, i.e. with $\log k(x)$ instead of $k(x)$.

Sampling y from a $U(0, k(x^t)$ distribution is the same as taking $y = k(x^t)u$ where $u \sim u(0,1)$.

Instead of doing this work on the log scale. Take

$$z = \log k(x^t) + \log u \text{ where } u \sim u(0,1)$$

We need to sample x^{t+1} uniformly from the slice $S = \{x : k(x) \geq y\}$. But this is the same as sampling x^{t+1} uniformly from the slice $S = \{x : \log k(x) \geq \log y = z\}$.