Monte Carlo Methods

Lecture notes for MAP001169

Based on Script by Martin Sköld

adopted by Krzysztof Podgórski

Contents

Part I

Simulation and Monte-Carlo Integration

Chapter 1

Simulation and Monte-Carlo integration

1.1 Issues in simulation

1.2 Raw ingredients

Chapter 2

Simulating from specified distributions

2.1 Transforming uniforms

2.2 Transformation methods

2.3 Rejection sampling

2.4 Conditional methods

Chapter 3

Monte-Carlo integration

3.1 Generic Monte Carlo integration

3.2 Bias and the Delta method

3.3 Variance reduction by rejection sampling

3.4 Variance reduction by importance sampling

3.4.1 Unknown constant of proportionality

Chapter 4

Markov Chain Monte-Carlo

4.1 Markov chains - basic concepts

4.2 Markov chains with continuous state-space

4.3 Markov chain Monte-Carlo integration

We now turn to the actual construction of the Markov chains.

4.4 Two simple continuous time Markov chain models

4.5 The Metropolis-Hastings algorithm

How do you choose transition density $\tilde{q}$ in order to satisfy (**??**)? The idea behind the Metropolis-Hastings algorithm is to start with an (almost) arbitrary transition density q. This density will not give the correct asymptotic distribution f, but we could try to repair this by rejecting some of the moves it proposes. Thus, we construct a new transition density $\tilde{q}$ defined by

$$\tilde{q}(x,y) = \alpha(x,y)q(x,y) + (1-\alpha(x,y))\delta_x(y), \tag{4.1}$$

where $\delta_x(y)$ is a point-mass at x. This implies that we stay at the level x if the proposed value is rejected and we reject a proposal y^* with probability $1-\alpha(x,y^*)$. Simulating from the density $y \mapsto \tilde{q}(x,y)$ works as follows

1. Draw y^* from $q(x,\cdot)$.
2. Draw u from $U(0,1)$.
3. If $u < \alpha(x,y^*)$ set $y = y^*$,
   else set $y = x$.

We now have to match our proposal density q with a suitable acceptance probability α. The choice of the *Metropolis-Hastings algorithm*, based on satisfying the global balance equation (**??**), is

$$\alpha(x, y) = \min(1, \frac{f(y)q(y, x)}{f(x)q(x, y)}), \tag{4.2}$$

you might want to check that this actually satisfies (**??**).

Algorithm 4.1 (The Metropolis-Hastings algorithm)**.**

```
1. Choose a starting value $x^{(0)}$.
2. Repeat for $i = 1, \ldots, N$:
    i.1 Draw $y^*$ from $q(x^{(i-1)}, \cdot)$.
    i.2 Draw $u$ from $U(0, 1)$.
    i.3 If $u < \alpha(x^{(i-1)}, y^*)$ set $x^{(i)} = y^*$, else set $x^{(i)} = x^{(i-1)}$.
3. $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$ is now a sequence of dependent draws, approximately from $f$.
```

There are three general types of Metropolis-Hastings candidate generating densities q used in practise; the *Gibbs sampler*, *independence sampler* and the *random walk sampler*. Below we will discuss their relative merits and problems.

4.6 The Gibbs-sampler

The Gibbs-sampler is often viewed as a separate algorithm rather than a special case of the Metropolis-Hastings algorithm. It is based on partitioning the vector state-space $\mathbf{R}^d = \mathbf{R}^{d_1} \times \cdots \mathbf{R}^{d_m}$ into blocks of sizes d_i, $i = 1, \ldots, m$ such that $d_1 + \cdots + d_m = d$. We will write $z = (z_1, \ldots, z_m) = (x_1, \ldots, x_d)$, where $z_i \in \mathbf{R}^{d_i}$ (e.g. $z_1 = (x_1, \ldots, x_{d_1})$. With this partition, the Gibbs-sampler with target f is now:

Algorithm 4.2 (The Gibbs-sampler)**.**

```
1. Choose a starting value $z^{(0)}$.
2. Repeat for $i = 1, \dots, N$:
    i.1 Draw $z_1^{(i)}$ from $f(z_1|z_2^{(i-1)}, \dots, z_m^{(i-1)})$.
    i.2 Draw $z_2^{(i)}$ from $f(z_2|z_1^{(i)}, z_3^{(i-1)}, \dots, z_m^{(i-1)})$.
    i.3 Draw $z_3^{(i)}$ from $f(z_3|z_1^{(i)}, z_2^{(i)}, z_4^{(i-1)}, \dots, z_m^{(i-1)})$.
     ⋮
    i.m Draw $z_m^{(i)}$ from $f(z_m|z_1^{(i)}, z_2^{(i)}, \dots, z_{m-1}^{(i)})$.
3. $z^{(1)}, z^{(2)}, \dots z^{(N)}$, is now a sequence of dependent draws approximately from $f$.
```

This corresponds to an MH-algorithm with a particular proposal density and $\alpha = 1$. What is the proposal density q?

Note the similarity with Algorithm **??**. The difference is that here we draw each z_i conditionally on all the others which is easier since these conditional distributions are much easier to derive. For example,

$$z_1 \mapsto f(z_1|z_2, \dots, z_m) = \frac{f(z_1, \dots, z_m)}{f(z_2, \dots, z_m)} \propto f(z_1, \dots, z_m).$$

Hence, if we know $f(z_1, \dots, z_m)$, we also know all the conditionals needed (up to a constant of proportionality). The Gibbs-sampler is the most popular MCMC algorithm and given a suitable choice of partition of the state-space it works well for most applications to Bayesian statistics. We will have the opportunity to study it more closely in action in the subsequent part on Bayesian statistics of this course. Poor performance occurs when there is a high dependence between the components Z_i. This is due to the fact that the Gibbs-sampler only moves along the coordinate axes of the vector $(z_1, \dots, z_m)$, illustrated by Figure 4.1. One remedy to this problem is to merge the dependent components into a single larger component, but this is not always practical.

Example 4.1 (Bivariate Normals)**.** Bivariate Normals can be drawn with the methods of Examples **??** or **??**. Here we will use the Gibbs-sampler instead. We want to draw from

$$(X_1, X_2) \sim N_2\left(0, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right),$$

and choose partition $(X_1, X_2) = (Z_1, Z_2)$. The conditional distributions are given by

$$Z_1|Z_2 = z_2 \sim N(\rho z_2, \sqrt{1-\rho^2}) \text{ and } Z_2|Z_1 = z_1 \sim N(\rho z_1, \sqrt{1-\rho^2}).$$

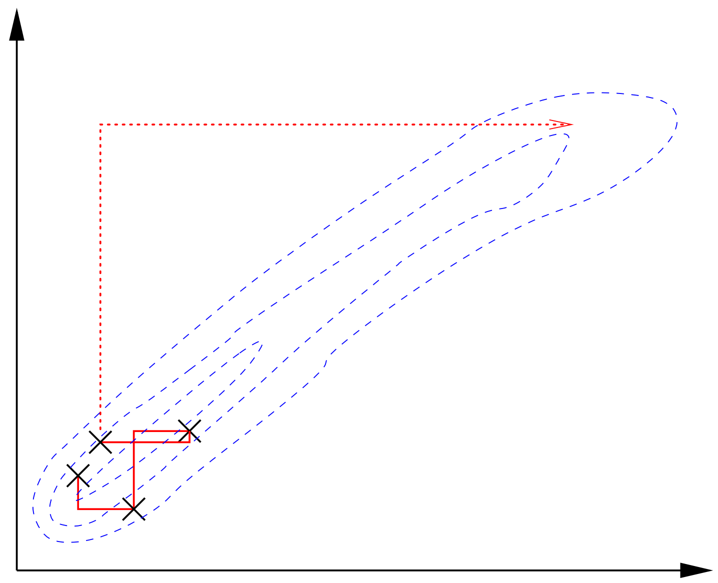

Figure 4.1: For a target (dashed) with strongly dependent components the Gibbs sampler will move slowly across the support since “big jumps”, like the dashed move, would involve first simulating a highly unlikely value from $f(z_1|z_2)$.

The following script draws 1000 values starting from $(z_1^{(0)}, z_2^{(0)}) = (0, 0)$.

```
function z=bivngibbs(rho)
z=zeros(1001,2);
for i=2:1000
    z(i,1)=randn*sqrt(1-rho^2)+rho*z(i-1,2);
    z(i,2)=randn*sqrt(1-rho^2)+rho*z(i,1);
end
z=z(2:1001,:);
```

In Figure **??** we have plotted the output for $\rho = 0.5$ and $\rho = 0.99$. Note the strong dependence between successive draws when $\rho = 0.99$. Also note that each individual panel constitute approximate draws from the same distribution, i.e. the $N(0, 1)$ distribution.

4.7 Independence proposal

The independence proposal amounts to proposing a candidate value y^* *independently* of the current position of the Markov Chain, i.e. we choose $q(x, y) = q(y)$. A necessary requirement here is that $\text{supp}(f) \subseteq \text{supp}(q)$; if this is not satisfied, some parts of the support of f will never be reached. This candidate is then accepted with probability

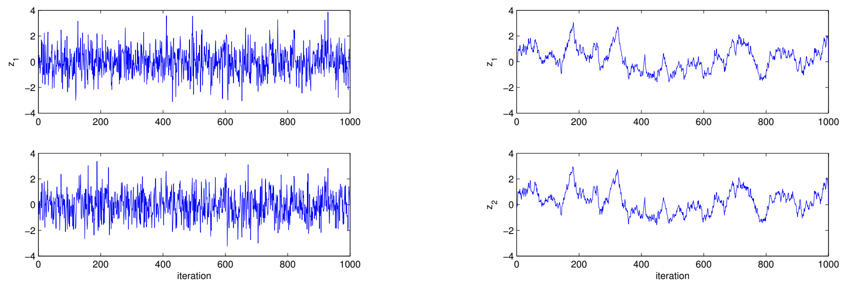

Figure 4.2: Gibbs-draws from Example 4.1, left with $\rho = 0.5$ and right with $\rho = 0.99$.

$$\alpha(x, y^*) = \begin{cases} \min\{\frac{f(y^*)q(x)}{f(x)q(y^*)}, 1\} & \text{if } f(x)q(y^*) > 0 \\ 1 & \text{if } f(x)q(y^*) = 0 \end{cases}$$

And we immediately see that if $q(y) = f(y)$, $\alpha(x, y) \equiv 1$, i.e. all candidates are accepted. Of course, if we really could simulate a candidate directly from $q(y) = f(y)$ we would not have bothered about implementing an MH algorithm in the first place. Still, this fact suggests that we should attempt to find a candidate generating density $q(y)$ that is as good an approximation to $f(y)$ as possible, similarily to the rejection sampling algorithm. The main difference is that we don't need to worry about deriving constants M or K such that $Mf < Kq$ when we do independence sampling. To ensure good performance of the sampler it is advisable to ensure that such constants *exists*, though we do not need to derive it explicitly. If it does not exist, the algorithm will have problems reaching parts of the target support, typically the extreme tail of the target. This is best illustrated with an example; assume we want to simulate from $f(x) \propto 1/(1 + x)^3$ using an Exp(1) MH independence proposal. A plot of (unnormalised) densities q and $1/(1+x)^3$ in Figure **??** does not indicate any problems — the main support of the two densities seem similar. The algorithm is implemented as follows

```
function [x,acc]=indsamp(m,x0)
x(1:m)=x0;
acc=0;
for i=1:m
    y=gamrnd(1,1);
    a=min(((y+1)^(-3))*gampdf(x(i),1,1)...
               /gampdf(y,1,1)/((x(i)+1)^(-3)),1);
    if (rand<a)
        x(i+1)=y;
        acc=acc+1;
    else
        x(i+1)=x(i);
    end
end
```

```
acc=acc/m;
```

and 1000 simulated values are shown in the left panel of Figure ?? with starting value `x0=1`. Looking at the output does not immediately indicate any problems either. However, a second run, now with starting value `x0=15`, is shown in the right panel of the same figure; 715 proposed moves away from the tail are rejected.

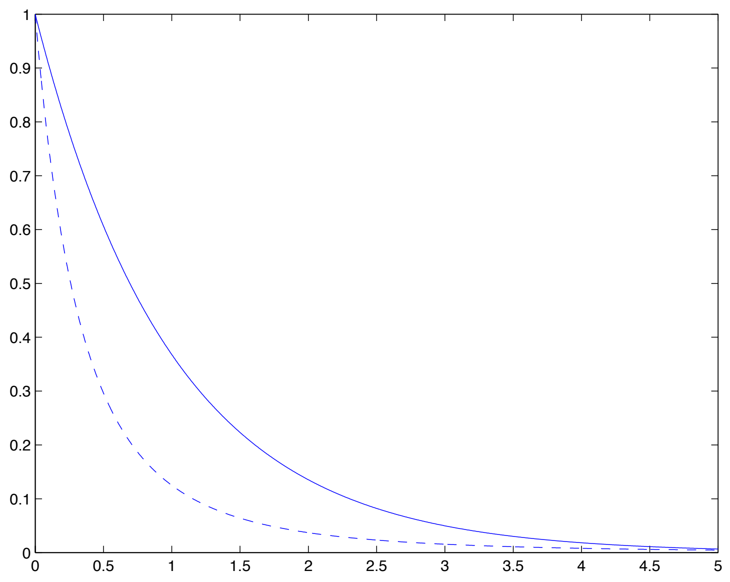

Figure 4.3: Unnormalized target $1/(1+x)^3$ (dashed) and Exp(1) independence proposal (solid).

The problem here is that since the light-tailed proposal density generates large values too seldom, the chain needs to stay in the tail for a very long time once it gets there to preserve stationarity. As a consequence this induces large autocorrelation which reduces the information contained in the output.

The main problem with the independence sampler is that unless q is a good approximations of f (and *especially so in high dimensional problems*), most proposals will be rejected and as a consequence autocorrelations high.

4.8 Random walk proposal

While the independence sampler needs careful consideration when choosing a candidate generating kernel q, random walk kernels are more "black box". As a drawback they will never be quite as efficient as a finely tuned independence proposal.

Random walk proposals are characterised by $q(x, y) = g(|x-y|)$, i.e. they are symmetric centered around the current value, and as a consequence α

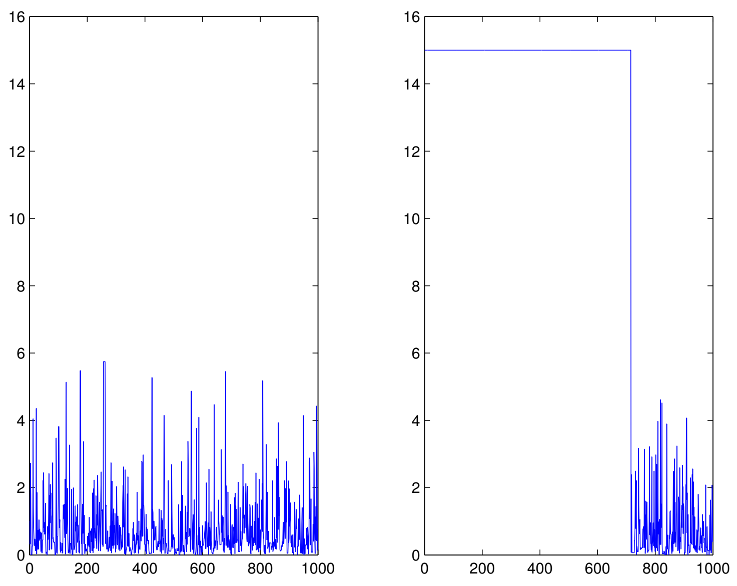

Figure 4.4: Output from independence sampler. Left with small starting value and right starting from the tail.

simplifies to

$$\alpha(x, y) = \min\{f(y)/f(x), 1\}.$$

Note especially that moves to areas with higher density, $f(y^*) > f(x)$, are always accepted.

A common choice is to let g be an $\mathrm{N}(0, s^2\Sigma)$ density with a suitably chosen covariance matrix Σ and a scaling constant s. In this case proposals are generated by adding a zero-mean Gaussian vector to the current state, $y^* = x + s\epsilon$ where $\epsilon \in N(0, \Sigma)$. What remains for the practitioner in this case is the choice of scaling constant s and covariance matrix Σ. The latter choice is difficult and in practise Σ is often chosen to be a diagonal matrix of ones. For s some general rules of thumb can be derived though. Lets first look at a simple example:

The function `rwmh.m` implements a Gaussian random-walk for a standard Gaussian target density given input `N` number of iterations, `x0` starting value and `s` scaling constant:

```
function [x,acc]=rwmh(N,x0,s);
x(1:N)=x0;
acc=0;
for i=1:N-1
    y=x(i)+s*randn;
    a=min(exp(-y^2/2+x(i)^2/2),1);
    if rand<a
        x(i+1)=y;
```

```
            acc=acc+1;
        else
            x(i+1)=x(i);
        end
    end
```

In Figure **??** we have plotted outputs from 1000 iterations with `x0=0` and scales `s` set to 0.3, 30 and 3. For the smallest scaling constant almost all of the proposed moves are accepted (908 out of 1000) but they are too small and the chain travels slowly across the support of f. For $s = 30$ very few proposals are accepted (48 out of 1000), but they are all very "innovative". Finally the result using $s = 3$ looks most promising. The methods can be compared by estimating the auto-correlation functions of the output as is done in Figure **??**. Here it is clear that if our goal is to estimate the mean, this will be most efficient for $s = 3$.

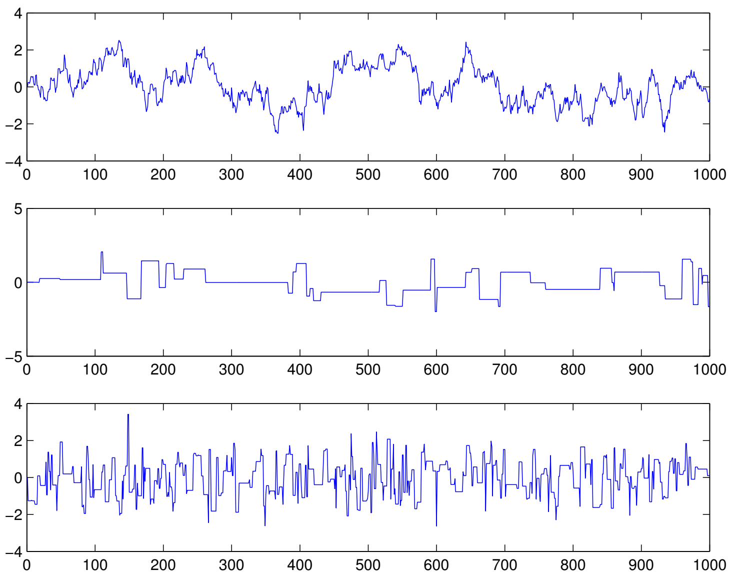

Figure 4.5: Random walks with too small s (top), too large s (middle) and well-tuned s (bottom)

Are there any general rules of thumb on how to choose random-walk scale s? The answer is yes, at least asymptotically. It turns out monitoring the acceptance rate is the key, and that for a large class of densities $f : \mathbf{R}^d \mapsto \mathbf{R}$, asymptotically, as $d \to \infty$, it is optimal to choose s in such a way that 23.4. . . % of the proposed moves are accepted (when $d = 1$ a slightly higher acceptance rate is often favourable). This is optimal for any fixed Σ.

Returning to the choice of Σ, note in Figure **??** that if we set Σ to be a diagonal matrix of ones, the random-walk sampler will have similar problems with dependent components as the Gibbs-sampler.

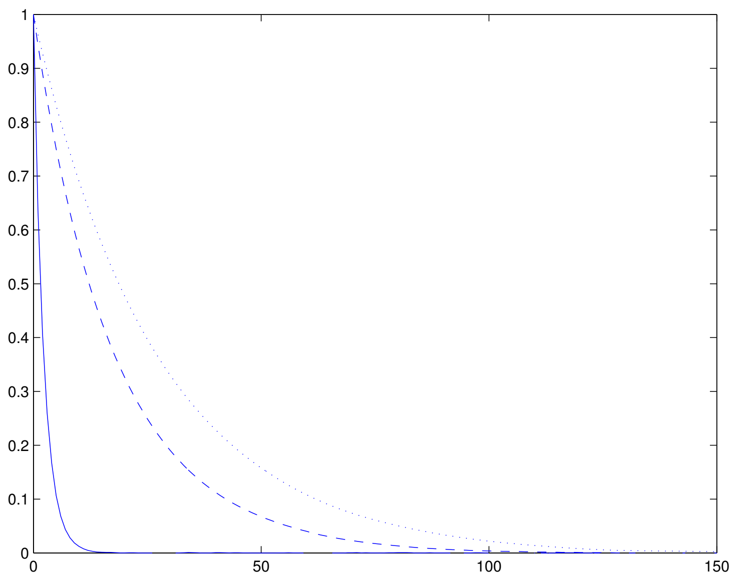

Figure 4.6: Autocorrelation functions for the output with $s = .3$ (dotted) $s = 30$ (dashed) and $s = 3$ (solid)

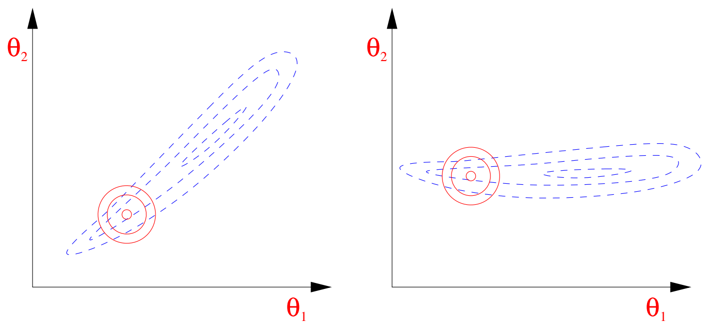

Figure 4.7: Left panel: With a symmetric random-walk proposal (solid contours) tuned to the optimal acceptance rate, movement will be slow if correlation of the posterior (dashed contours) is strong or variances different.

In addition, orthogonalising does not help unless we also standardise variances. A good choice of Σ is one that is similar to the covariance matrix of the target (up to a proportionality constant). An approximation that is often useful is to let Σ be proportional to the Hessian matrix of $\log f$ (i.e. $H(x)$ with entries $H_{i,j} = d(\log f)^2/(dx_i\, dx_j)$) evaluated at e.g. a mode of f if this can be found.

4.8.1 Multiplicative random walk

If the target has a very heavy tail, the random walk proposal will generally perform poorly. In a similar fashion to the independence sampler with a light-tailed proposal, since it takes the chain a long time to travel all the way to the tail, it will stay there for a long time when it reaches it.

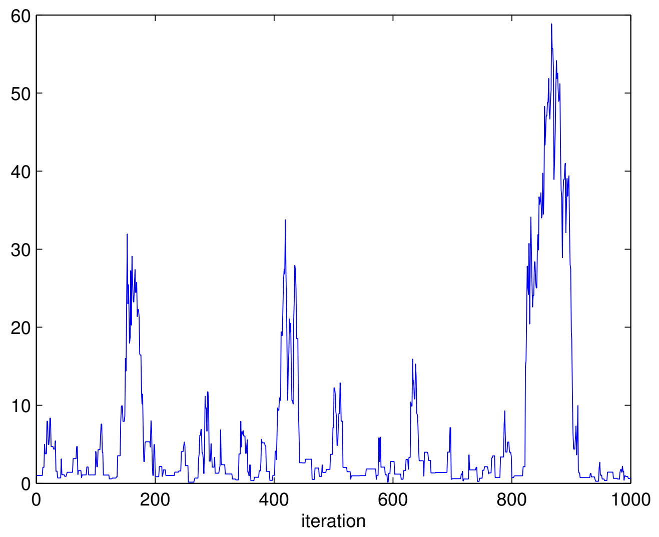

Figure 4.8: Typical behaviour of a random walk Metropolis Hastings on a heavytailed target. Seemingly stable behaviour is exchanged with long excursions in the tails.

In this situation, a *Multiplicative random-walk proposal* is often more efficient. The random-walk proposal is formed by adding an independent component, $y^* = x^{(i-1)} + \epsilon$, the *multiplicative* random-walk instead multiplies with an independent component, $y^* = x^{(i-1)}\epsilon$. If we denote the density of ϵ by g, the proposal-generating density is $q(x, y) = g(y/x)/x$ and we accept/reject with probability

$$\alpha(x, y) = \min(1, \frac{f(y)g(x/y)x}{f(x)g(y/x)y}).$$

4.9 Hybrid strategies

In statistical problems there is often a natural choice of partition for the Gibbs-sampler, however one or more of the conditional distributions in Algorithm 4.2 might be difficult to sample from directly. In this case, an exact draw from the tricky conditionals can be replaced by one iteration of the Metropolis-Hastings algorithm. This strategy is often necessary in complex problems and is sometimes referred to as the Metropolis–within–Gibbs algorithm.