# Chapter 13
# Introduction to Coupling-from-the-Past using *R*

Wilfrid S. Kendall

**Abstract** The purpose of this chapter is to exemplify construction of selected coupling-from-the-past algorithms, using simple examples and discussing code which can be run in the statistical scripting language *R*. The simple examples are: symmetric random walk with two reflecting boundaries, a very basic continuous state-space Markov chain, the Ising model with external field, and random walk with negative drift and a reflecting boundary at the origin. In parallel with this, a discussion is given of the relationship between coupling-from-the-past algorithms on the one hand, and uniform and geometric ergodicity on the other.

## 13.1 Introduction

Propp and Wilson's coupling-from-the-past algorithm (CFTP) [324, 325] and its generalisations are based on simple but delicately balanced ideas. The purpose of this chapter is to give a careful discussion of exactly how some simple examples of CFTP can be implemented in the popular and flexible statistical language *R* [327]. Of course *R* is a scripting language and therefore is not ideal for implementing simulation algorithms; serious work should use compiler-based languages or at least scripting languages with substantial support for numerics (for example, *Python* with the *Numpy* extension, or *Matlab*). However *R* has several advantages if one wishes to demonstrate precisely what is going on without worrying much about efficiency; in particular *R* is not only free and open-source but also widely popular within the statistical community. Moreover, by its very nature *R* allows direct access to statistical procedures, making it easier to check statistical correctness. In addition, the algorithms for CFTP are most clearly communicated if the reader can actually run, test, and vary them within a suitable stable computing environment of this kind. To

Wilfrid S. Kendall
Department of Statistics, University of Warwick, Coventry CV4 7AL, UK, e-mail: `w.s.kendall@warwick.ac.uk`





facilitate this, the chapter was written using the *R* report-generation utility software `Sweave` [250], so as to ensure that the *R* code presented here is identical to the code used to generate the reported results and to test algorithms (though graphics-related code is largely suppressed). Actual *R* code is presented in boxed displays (sometimes continued over consecutive pages); occasional *R* output is presented in similar boxes distinguished by their shaded edges.

After this introductory section, the chapter commences (Sect. 13.2) with a discussion of the classic Propp-Wilson version of CFTP, and then demonstrates its use in a (rather inefficient) implementation of CFTP for image analysis using the Ising model, before turning (Sect. 13.3) to the more subtle and generally less understood notion of dominated CFTP. A brief concluding Sect. 13.4 discusses the relationship of CFTP and dominated CFTP to uniform and geometric ergodicity.

Throughout much of the exposition, algorithms are illustrated by short self-contained *R* scripts, which themselves are discussed in the text. The *R* scripts (concatenated together into one long *R* script file) can be found at `go.warwick.ac.uk/wsk/perfect_programs#Ulm-notes`. The exposition focusses on specific examples rather than general principles and theory (which is covered in [228]), though comments on the general theory occur throughout.

## 13.2 Classic Coupling-from-the-Past

The objective of CFTP is to produce exact draws from the equilibrium distribution of suitable Markov chains $X$ defined on specified state-spaces $\mathcal{X}$. Conventional Markov chain Monte Carlo runs a single realization of a suitable $X$ from the present (time 0) to some distant future (time $T = n$ where $n$ is large), and relies on convergence theorems which assert (under suitable conditions) that the distribution of $X_n$ approximates the equilibrium distribution when $n$ is suitably large. In contrast, CFTP typically seeks to generate a realization of $X$ which starts in the indefinite past (time $T = -n$ for indefinitely large $n$) and runs until the present (time 0). In the case of classic CFTP, this is achieved by realizing a single simulation of multiple trajectories of $X$ running from all possible starting locations $x \in \mathcal{X}$ and all possible past times $-n$. This is done by constructing a stochastic flow $\{F_{-n,-n+t} : \mathcal{X} \to \mathcal{X} \,:\, n,t \geq 0\}$. Here $F_{-n,-n+t} : \mathcal{X} \to \mathcal{X}$ is a random map, and for each $x \in \mathcal{X}$ and each $-n < 0$ the process $\{F_{-n,-n+t}(x) : t = 0,1,\ldots,n\}$ is a realization of the Markov chain $X$ run from time $T = -n$ (with initial state $x$) through to time 0.

The classic CFTP algorithm succeeds exactly when the stochastic flow is constructed so that the map $F_{-n,0}$ has a one-point random image for all large enough $n$; we can view this as complete coalescence of the coupled realizations $\{F_{-n,-n+t}(x) : t = 0,1,\ldots,n\}$ for varying $x \in \mathcal{X}$, once $n$ is sufficiently large. Because of the random flow property ($F_{-n-m,0} = F_{-m,0} \circ F_{-n-m,-m}$) it then follows that the random image $\{F_{-n,0}(x) : x \in \mathcal{X}\}$ stabilizes for large enough $n$, and the point in the stable one-point image is actually an exact draw from the equilibrium distribution of $X$. See Theorem 3 of [228] for a formal statement and proof.

Practical issues are: firstly, how to construct coalescing stochastic flows; secondly, how to identify a random time such that coalescence has definitely occurred. If the state-space is finite then it is possible in principle to proceed by exhaustive enumeration, but this would typically be unbearably inefficient and computationally infeasible. In this section we will discuss three thematic examples which allow for efficient solutions.

### *13.2.1 Random Walk CFTP*

We begin by illustrating classic CFTP [324] in the very special case of simple symmetric random walk on the integer segment $\{1,2,\ldots,10\}$, with (reversible) reflection at both boundaries. This expands on and refines the discussion in Sect. 1.2 of [228]. Here the random flow is implemented as synchronous coupling of random walks begun at different starting points, and the monotonicity of this coupling can be used to detect coalescence in an efficient way. Moreover in this simple case the equilibrium distribution is uniform (this follows directly from a detailed balance calculation) and so the validity of CFTP can be confirmed empirically using a statistical $\chi^2$ test.

This example is completely trivial, but can be viewed as a prototype for a rather less trivial Ising model application, which is discussed below in Sect. 13.2.4.

#### 13.2.1.1 Helper Functions for the Simulations

We begin by describing various *R* functions useful for constructing the random walk CFTP algorithm. First of all, we need a function which generates a sequence or block of innovations for the underlying stochastic flow. Since the flow is composed of synchronously coupled simple symmetric random walks, the innovations can be realized as binary random variables which are equally likely to take the values $\pm 1$. The function `make_block` generates a single block (of prescribed length) of realizations of independent binary random variables of this form, using the *R* simulation function `rbinom` to generate $\mathrm{Ber}(\frac{1}{2})$ random variables (equivalently, $\mathrm{Binom}(1,\frac{1}{2})$) and then transforming them appropriately.

```
make_block <- function(block_length) {
    return(2 * rbinom(block_length, 1, 1/2) - 1)
}
```

The flow for this implementation of random walk CFTP is based on the `update` function. This uses a single $\pm 1$ innovation `innov` as a candidate for the jump of the chosen random walk at a particular instant. If the random walk is prevented by a boundary from using this jump then it simply stays where it is.



```
update <- function(x, innov, lo = 1, hi = 10) {
    return(min(max(x + innov, lo), hi))
}
```

The consequence of this `update` definition is that all random walks are synchronously coupled; they move in parallel except where boundaries prevent them from so doing. In particular the random walk trajectories depend monotonically on their starting positions.

The function `cycle` lies at the heart of the algorithm. It employs a block of $\pm 1$ innovations `innovations` (as supplied by `make_block` above) to construct a synchronously coupled pair of upper and lower random walks using `update`. One of the walks (`upper`) starts from the upper boundary `hi` of the random walk, the other (`lower`) starts from the lower boundary `lo`. Consider all other synchronously coupled simple symmetric walks based on the same stream of innovations started either at the same time as these two, or earlier. By monotonicity of the synchronous coupling, at any given time all these other walks must lie above `lower` and below `upper`. The `cycle` function returns `NA` (*R*'s "not a number" value) if coalescence has not yet occurred at time 0, and otherwise returns the common coalesced value. Since coalescence is achieved exactly when `upper` and `lower` meet before time 0, it is detected easily.

```
cycle <- function(innovations, lo = 1, hi = 10) {
    lower <- lo
    upper <- hi
    for (i in 1:length(innovations)) {
        lower <- update(lower, innovations[i],
            lo = lo, hi = hi)
        upper <- update(upper, innovations[i],
            lo = lo, hi = hi)
    }
    if (upper != lower)
        return(NA)
    return(upper)
}
```

#### 13.2.1.2 A Typical Run of the Random Walk CFTP Algorithm

We illustrate the CFTP algorithm by carrying out classic random walk CFTP, run statement-by-statement, without regard for efficiency.

First we initialize the seed of the random number generator (to facilitate repeatability of the simulation), and use `make_block` to create an initial vector of $\pm 1$ `innovations`.

```
set.seed(1)
innovations <- make_block(2)
```

Then we repeatedly apply `cycle`, extending the `innovations` block to the left, until `upper` and `lower` random walks coincide at time zero.

```
while (is.na(cycle(innovations))) {
    innovations <- c(
        make_block(length(innovations)),
        innovations)
}
```

The doubling of the length of the `innovations` block corresponds to a simple binary search for the coalescence time.

Finally we report the result of the CFTP algorithm, which we obtain by re-running `cycle` on the total block of `innovations` which has been accumulated in the previous `while` loop.

```
cycle(innovations)
```

Output is as follows.

```
[1] 4
```

#### 13.2.1.3 Implementing the Algorithm

The random walk CFTP simulation can now be put together succinctly as a single *R* function, using the helper functions defined above.

```
rw_cftp <- function(initial_range, lo = 1,
    hi = 10) {
    innovations <- make_block(initial_range)
    result <- NA
    while (is.na(result)) {
        innovations <- c(
            make_block(length(innovations)),
            innovations)
        result <- cycle(innovations, lo = lo,
            hi = hi)
    }
    return(result)
}
```



#### 13.2.1.4 Statistical Analysis

This *R* implementation of random walk CFTP is of course rather slow. We are however now able to investigate the algorithm output statistically; we base this on 10000 successive runs of the algorithm

```
data <- sapply(rep(100, 10000), rw_cftp)
tabulate(data)
```

Output is as follows.

```
[1]  959  957 1029 1054 1051 1007  958 1002  987  996
```

Computations took 15 seconds on a notebook computer running Ubuntu 10.10 using an Intel quad processor i7 Core M 620 at 2.67GHz using 2 GB RAM. Fig. 13.1 (a) presents a histogram of these results. A $\chi^2$-test confirms that the output has the correct distribution (uniform on $\{1,2,\ldots,10\}$).

```
chisq.test(tabulate(data), p = rep(1/10, 10))
```

Output is as follows.

```
        Chi-squared test for given probabilities

data:  tabulate(data)
X-squared = 11.89, df = 9, p-value = 0.2196
```

#### 13.2.1.5 Graphical Visualization of Random Walk CFTP Algorithm

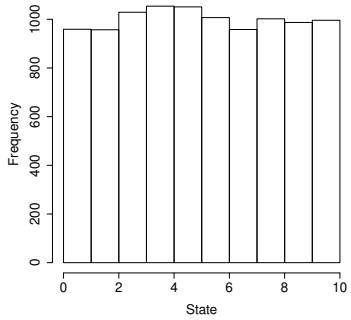
It is helpful to visualize the algorithm using graphics. Re-writing the algorithm to generate graphical output is now a straightforward exercise; and this can be invaluable when testing for correctness of implementation. The changes are as follows:

1. The `cycle` part of the algorithm is augmented graphically so that at each cycle it plots the trajectories of upper and lower processes;
2. The main body of the algorithm initializes the graphics, then cycles through the CFTP algorithm using the graphically augmented version of `cycle`.

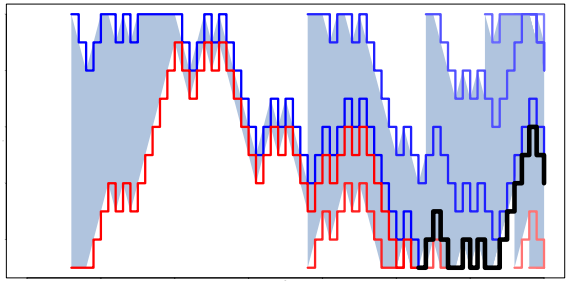
We can now use the amended algorithm to obtain a graphical demonstration of how the algorithm runs. The result is presented in Fig. 13.1 (b).

#### 13.2.1.6 More Details

The reader is referred to the tutorial [77] (see also [398, 421]) for further discussion of classic CFTP, including illustration of the defective simulation results arising from (a) not re-using randomness, or (b) stopping the CFTP cycle early at the first




(a) Histogram of results from repeated runs of random walk CFTP. The impression of uniform distribution is confirmed by a $\chi^2$-test.



(b) Visualization of a single random walk CFTP run. Initial time range is $\{-1,0\}$; the algorithm detects coalescence after 8 cycles at which point the time range is $\{-64,-63,\ldots,0\}$ The thick line indicates the coalesced trajectory. The range between successive `upper`/`lower` pairs is indicated by the shaded regions. Note how earlier `upper`/`lower` pairs "funnel" between later `upper`/`lower` pairs.

**Fig. 13.1** Random walk CFTP output: histogram and visualization



point when upper and lower simulations coalesce. See also Exercises 13.1 and 13.3 below.

Note that, despite appearances, the validity of the CFTP algorithm described here actually does *not* depend on the symmetry of the random walk. Indeed the algorithm can be modified to deal with instances in which the target Markov chain is no longer a random walk, nor even a reversible Markov chain; It suffices that the target chain exhibit a stochastic monotonicity sufficient to make sense of being able to couple realizations of the chain starting at all possible initial points so that there is an upper chain and a lower chain, and such that coalescence eventually occurs. For more on the nature of the required monotonicity, see [120, 121]. An alternative general approach to perfect simulation is presented in [118], using a rejection sampling approach, with the advantages of requiring less stringent monotonicity structure and avoiding possible correlation between output and algorithm run-time (compare Exercise 13.2); the relationship between this and classic CFTP has been elucidated in [122].

**Exercise 13.1.** Consider the function `rw_cftp`. In each cycle of the following loop,

```
while(is.na(result)) {...},
```

the previous block of `innovations` is extended backwards in time. Investigate the statistical consequences of (incorrectly) generating completely new blocks instead, using the following construction.

```
innovations <- make_block(2 * length(innovations))
```

**Exercise 13.2.** Consider the function `rw_cftp`. What would be the statistical consequences if the

```
while(is.na(result)) {...}
```

loop were abandoned after (say) 2 iterations, in favour of re-starting the function `rw_cftp` using the `initial_range`?

**Exercise 13.3.** Consider the function `rw_cftp`. What would be the consequences of extending the previous block of `innovations` *forwards* in time rather than back in time? That is to say, generating new blocks using the following construction.

```
innovations <- c(
     innovations,
     make_block(length(innovations)))
```

**Exercise 13.4.** Modify the function `make_block` so that `rw_cftp` produces a perfect draw from the equilibrium distribution of a specified *asymmetric* random walk on the integer segment $\{1,2,\ldots,10\}$. Compute the true equilibrium using detailed balance, and test the modified algorithm statistically.

### *13.2.2 Read-once CFTP*

In this section we consider *read-once CFTP* (RO-CFTP). This idea goes back to [420], and may be thought of as a variation on classic CFTP which works block-by-block using successive draws of whole blocks of innovations, and additionally runs *forwards* in time rather than backwards. For the sake of clarity we will describe this in terms of random walk CFTP, although the same principles apply to any instance of classic CFTP. This is indicated, for example, as part of the treatment of the small-set CFTP discussed in Sect. 13.2.3.

The implementation uses the same helper functions as are described for random walk CFTP in Sect. 13.2.1.

#### 13.2.2.1 A Typical Run of RO-CFTP

The idea is to group together fixed sequences of innovations in separate *blocks*, block length $n$ being chosen to ensure a positive chance of coalescence within the block.

```
n <- 100
```

Before discussing the details, we work through an instance of read-once CFTP run statement-by-statement. We begin by repeatedly sampling an initial block of innovations and rejecting the sample until coalescence is achieved. The resulting draw is therefore of a block of innovations, viewed as running over the time interval $\{-n, -n+1, \ldots, 0\}$, which is *conditioned* so that the flow map $F_{-n,0}$ has a one-point image.

```
block <- make_block(n)
while (is.na(cycle(block))) {
    block <- make_block(n)
}
```

This initial (coalescent) block is stored as the first (left-most) contribution to a whole sequence of `innovations`.

```
innovations <- block
```

Now we generate a sequence of further blocks corresponding to a sequence of flow functions $\{F_{kn,(k+1)n} : k = 0, 1, \ldots\}$; while no coalescence is achieved these blocks are successively appended to `innovations`. This iteration is finished once a further coalescent block is obtained, stored separately in `coalescent_block`.

```
block <- make_block(n)
while (is.na(cycle(block))) {
    block <- make_block(n)
```



```
    append(innovations, block)
}
coalescent_block <- block
```

Finally we return the final value from the current sequence of `innovations`. A further RO-CFTP run can now start off with the most recent `coalescent_block`, to save the work of repeatedly sampling to generate a suitably conditioned initial block.

```
cycle(innovations)
```

Output is as follows.

```
[1] 6
```

The validity of this algorithm may be deduced from the following argument. Suppose we carry out classic CFTP, but limit ourselves to detecting coalescence block-by-block and stop when we obtain a coalescent block. We then generate blocks of innovations in succession, $B_{-1}$, $B_{-2}$, …, $B_{-(N-1)}$, $B_{-N}$. By construction these blocks are all non-coalescent except for the final block, $B_{-N}$. Moreover, conditional on $N$, the blocks $B_{-1}$, $B_{-2}$, …, $B_{-(N-1)}$ are independent draws of blocks conditioned to be non-coalescent, while independently the final block $B_{-N}$ is conditioned to be coalescent. Moreover $N$ has a Geometric distribution with success probability given by the probability of a block being coalescent. For classic CFTP these blocks are arranged thus:

$$B_{-N} \cdot B_{-(N-1)} \cdot \ldots \cdot B_{-2} \cdot B_{-1}, \tag{13.1}$$

and this delivers an exact draw from the equilibrium distribution.

On the other hand the RO-CFTP procedure described above generates an arrangement

$$\widetilde{B}_1 \cdot \widetilde{B}_2 \cdot \ldots \cdot \widetilde{B}_{\widetilde{N}-1} \cdot \widetilde{B}_{\widetilde{N}}, \tag{13.2}$$

where, conditional on $\widetilde{N}$, the *first* block $\widetilde{B}_1$ is conditioned to be coalescent and independently the blocks $\widetilde{B}_2$, $\widetilde{B}_3$, …, $\widetilde{B}_{\widetilde{N}}$ are independent draws of blocks conditioned to be non-coalescent. Finally, $\widetilde{N}$ again has a Geometric distribution with success probability given by the probability of a block being coalescent.

Thus the innovation sequences corresponding to the concatenations of (13.1) and (13.2) are statistically identical. It follows that the distributions of the images of the corresponding flow maps are the same, and hence RO-CFTP delivers a draw with the same statistics as a draw from classic CFTP.

This algorithm may be refined to economize on storage: it suffices to establish the coalesced value of the initial coalesced block, and then to keep a record only of the most recent result of updating this coalesced value. Moreover, in a very natural way, RO-CFTP can be made to deliver a sequence of independent exact draws from a sequence of draws of blocks.

#### 13.2.2.2 Packaging the Algorithm

As before, we package the RO-CFTP process as a single *R* function.

```
ro_cftp <- function(block_length) {
    block <- make_block(block_length)
    while (is.na(cycle(block))) {
        block <- make_block(block_length)
    }
    innovations <- block
    block <- make_block(block_length)
    while (is.na(cycle(block))) {
        block <- make_block(block_length)
        append(innovations, block)
    }
    return(cycle(innovations))
}
```

We ignore here the possibility of re-using the final coalescent block, which would economize on the initial rejection-sampling step. Also we do not economize on storage by storing only the most recent value during the iteration loop; this is because the less economical version presented here is easier to augment to produce visualization graphics.

#### 13.2.2.3 Statistical Analysis

As with random walk CFTP, computation is slow but manageable (in [420] it is shown that run-time can be arranged to be comparable to that of classic CFTP). The output appears to be uniformly distributed (Fig. 13.2 (a)).

```
data <- sapply(rep(100, 2000), ro_cftp)
tabulate(data)
```

Output is as follows.

```
[1] 205 212 192 192 212 208 193 204 192 190
```

Uniformity is confirmed by a $\chi^2$-test.

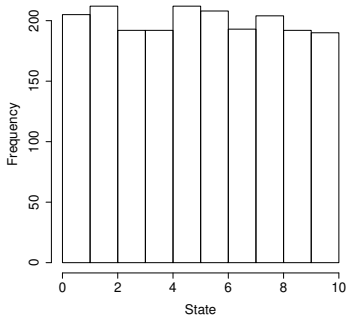
```
chisq.test(tabulate(data), p = rep(1/10, 10))
```

Output is as follows.

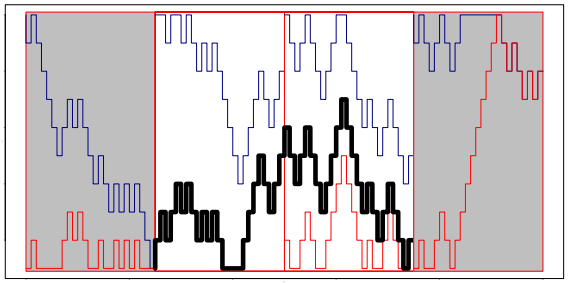
```
        Chi-squared test for given probabilities

data:  tabulate(data)
X-squared = 3.67, df = 9, p-value = 0.9318
```






(a) Histogram of results from repeated runs of RO-CFTP. The impression of uniform distribution is confirmed by a $\chi^2$ test.



(b) Visualization of an RO-CFTP run producing a single result. The two non-coalescing blocks have white backgrounds. The thick line indicates the coalesced trajectory. This trajectory starts at the end of the first coalesced block, at the coalesced value, and ends at the start of the second coalesced block, there delivering the sampled value.

**Fig. 13.2** Illustrations of RO-CFTP output: histogram and visualization

A typical run is visualized graphically in Fig. 13.2 (b)

#### 13.2.2.4 More Details

We have mentioned (Sect. 13.2.2.1) that it is easy to adapt RO-CFTP so as to generate a stream of exact draws for its target distribution. This and other ways of generating streams of exact CFTP draws have been discussed in [298]. Indeed this work led directly to the link [122] between classic CFTP and Fill's version of perfect simulation.

**Exercise 13.5.** Consider the function `ro_cftp`. Using the modification of the function `make_block` from Exercise 13.4, arrange for `ro_cftp` to produce a perfect draw from the equilibrium distribution of a specified *asymmetric* random walk on the integer segment $\{1,2,\ldots,10\}$. Test the modified algorithm statistically.

**Exercise 13.6.** Modify the function `ro_cftp` to produce a sequence of independent exact draws, with the length of the sequence specified by a second argument of `ro_cftp`, as indicated in Sect. 13.2.2.1.

### *13.2.3 Small-set (Green-Murdoch) CFTP*

So far we have only described CFTP for discrete probability models. But in fact CFTP can be applied to continuous probability models as well. The clue as to how to do this is to be found in the theory of small sets for continuous state-space Markov chains [9, 275, 306].

**Definition 13.1 (Small set).** Suppose that $X = \{X_0, X_1, X_3, \ldots\}$ is a Markov chain on a state-space $\mathcal{X}$ which is a measurable space. We say $C \subseteq \mathcal{X}$ is *small of lag k* if there is a probability measure $\nu$ on $\mathcal{X}$ and $0 < \rho < 1$ such that the following minorization condition is obeyed for all $x \in C$:

$$\mathbf{P}(X_k \in \cdot \mid X_0 = x) \quad \geq \quad \rho\,\nu(\cdot)\,.$$

(Technically one usually requires $X$ is $\varphi$-irreducible and insists that $\varphi(C) > 0$.)

In case $k = 1$ then $X$ can be viewed as having positive probability $p$ of regeneration at every visit to the small set $C$. If $C = \mathcal{X}$ is the whole state-space then it is shown in [297] how to generate a CFTP algorithm.

#### 13.2.3.1 A Simple Example of Small-Set CFTP

A simple example is given by the Markov chain $X$ on $[0,1]$ with "triangular" transition density: the conditional distribution of $X_{n+1}$ given $X_n = x$ has probability density $p(x,y)$ where



$$p(x,y) \quad = \quad \begin{cases} 2\,(x/y) & \text{if } 0 < x \leq y\,, \\ 2\,((1-x)/(1-y)) & \text{if } y < x < 1\,. \end{cases}$$

Thus the density forms a triangle with base the unit segment on the $x$-axis, and third vertex at $(y,2)$. A visual appreciation of the fact that the whole state-space is small is given by the graph of densities in Fig. 13.3, using the following *R* function: all possible triangles intersect in a "regeneration triangle".

```
triangle <- function(x, y = 0) {
    if (y == 0)
        return(2 * (1 - x))
    if (y == 1)
        return(x)
    return((x <= y) * (2 * x/y) + (x > y) * (2 *
        (1 - x)/(1 - y)))
}
```

For the purposes of CFTP we need to be able to generate coupled realizations from all triangle densities. We begin by noting that a simultaneous draw from all densities can be obtained by regarding $p(\cdot,y)$ as obtained from $p(\cdot,0)$ by a horizontal affine shear, while we can draw from $p(\cdot,0)$ by first drawing $(x,z)$ from the rectangle $(0,1) \times (0.2)$ and then folding the rectangle over using the diagonal from $(0.2)$ to $(1,0)$ as in the following *R* function and illustrated in Fig. 13.4 (a).

```
left_draw <- function(n) {
    x <- runif(n, min = 0, max = 1)
    z <- runif(n, min = 0, max = 2)
    reflect <- (z > (2 - 2 * x))
    return(reflect * cbind(1 - x, 2 - z) + (1 -
        reflect) * cbind(x, z))
}
```

Fig. 13.4 (b) presents the result of shearing the points to produce draws from a different triangular density $p(\cdot,0.75)$.

This coupling construction needs to be modified so that points falling in the "regeneration triangle" are not sheared at all, while points that would fall into or to the right of this triangle after shearing are subject to a further affine shear: see Fig. 13.4 (c) and Fig. 13.4 (d). We compile all this into a single function to deliver the resulting coalescing flow.

```
coalescing_flow <- function(y) {
    draw <- left_draw(1)
    x <- rep(draw[, 1], length(y))
    z <- rep(draw[, 2], length(y))
    regeneration <- (2 * x > z) & (2 * (1 - x) >
        z)
```

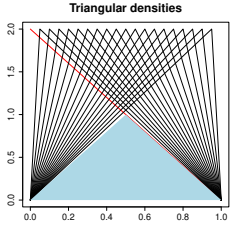



**Fig. 13.3** Triangular densities. Note the common area under the densities, indicated by the shaded triangle. This implies that the entire state-space $[0, 1]$ is a small set.

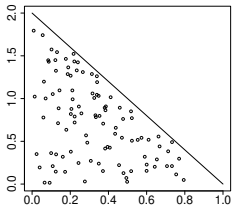
```
    sheared_x <- x + 0.5 * z * y
    needs_second_shear <- (2 * sheared_x > z) &
        (z < 1)
    return(regeneration * x + (1 - regeneration) *
        (sheared_x + needs_second_shear * (1 -
            z)))
}
```

RO-CFTP may be applied to the output to harvest a sequence of exact draws from the target distribution, which is the equilibrium distribution of the Markov chain $X$. In the following code, we exceptionally include the graphical directives, which produce Fig. 13.5.

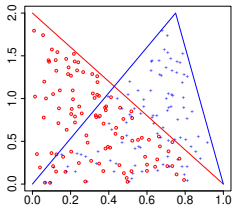
```
T1 <- 30
start <- seq(0, 1, 0.2)
state <- start
predecessor <- start[1]
initial <- TRUE
store <- state
t0 <- 1
```

```
plot(0:T1, rep(0, T1 + 1), xlim = c(0, T1),
    ylim = c(0, 1), type = "l",
    main = "Coalescing flow",
    xlab = "Time", ylab = "State")
for (t in 1:T1) {
    if (length(unique(state)) == 1) {
        for (j in 1:dim(store)[2]) lines(t0:t +
            1, store[, j])
```

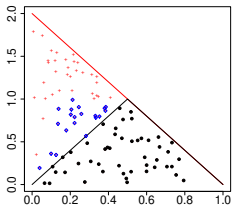




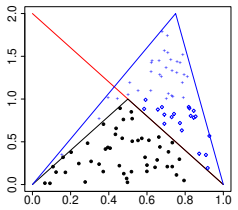

(a) Draws of points lying beneath the left-most triangular density




(b) Draws of points as above, and also of their locations after being subjected to a shear so that they are drawn from a different triangular density




(c) Draws of points lying beneath the left-most triangular density, marking points in "regeneration triangle" and also points which would fall into or to right of "regeneration triangle" after first shearing




(d) Draws of sheared points, but leaving untouched those points originally in "regeneration triangle" and applying a further shear to points which would fall into or to right of "regeneration triangle" after first shearing

**Fig. 13.4** Illustration of triangular densities example

```
        if (!initial)
            points(t, predecessor, col = "red",
                pch = 19, cex = 20)
        initial <- FALSE
        state <- c(unique(state), start)
        t0 <- t
        store <- state
    }
    predecessor <- state[1]
```

```
    state <- coalescing_flow(state)
    store <- rbind(store, state)
}
```

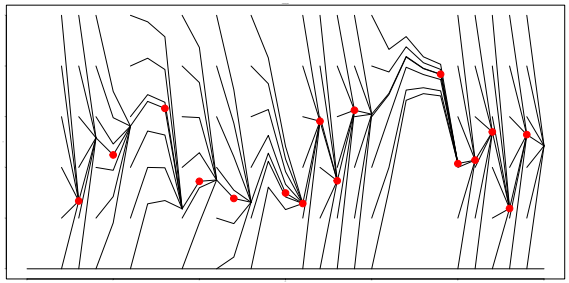


**Fig. 13.5** Flow induced by triangular densities example. The circular dots indicate the instants where RO-CFTP permits sampling of exact draws from the equilibrium distribution. These occur at the right-hand ends of trajectories running strictly between adjacent pairs of coalesced blocks.

#### 13.2.3.2 Extensions of Small-set CFTP

In real-world applications either the whole state-space may not form a small set, or the resulting regeneration probability may be too small to be of practical use. In [297] it is shown how to overcome this when the state-space can be partitioned into disjoint small sets of useful regeneration probability. Furthermore, in [10] it is described how the small-set CFTP construction may be adapted to produce a representation of the equilibrium probability distribution (see also [175]). Small-set CFTP provides a strong motivation for the natural question, how prevalent are small sets of small lag? Consider the case of Markov chains with measurable transition densities: in [230] it is shown that there exist examples with no small sets of lag 1 at all, but that all such Markov chains possess so many small sets of lag 2 that, when sub-sampled with periodicity 2, they may be represented in terms of latent discrete-time Markov chains.

**Exercise 13.7.** Modify the code in this section by removing the graphics directives, and arranging for it to produce a sequence of specified length of perfect draws from the equilibrium distribution of the Markov chain $X$ using RO-CFTP (as applied in Sect. 13.2.3.1). Use *R* to construct a kernel density estimate of the equilibrium density of $X$.



### *13.2.4 Image Analysis and Ising CFTP*

One of the earliest instances of perfect simulation concerned the Ising model [324]. Here the major application used Sweeny's algorithm for exact simulation of Fortuin-Kastelyn-Potts models near the critical temperature; however we will describe the easier method used in [324] to produce exact draws from high-temperature Ising models. This method is well-adapted to image analysis problems.

Readers unfamiliar with the Ising model will find a useful and accessible introduction in [233]. Recall that an Ising model is formed by assigning spins $S_i = \pm 1$ to each node $i$ of a fixed graph $G$ (we shall take $G$ to be a finite planar lattice $\{1,\ldots,N\}^2$). Then the Ising model is defined as having the following probability mass function for assignations of spins:

$$p(S_i : i \in G) \quad = \quad Z(J)^{-1} \exp\left( J \sum_{i \sim j} S_i S_j \right) . \tag{13.3}$$

Here $i \sim j$ if $i$ and $j$ are neighbours in the graph $G$ and the sum is taken over all unordered pairs; we shall take $J > 0$ (the *ferromagnetic* case), so that neighbouring sites with the same spin lead to higher values of the probability mass function. Finally, $Z(J)$ is the normalizing constant.

For purposes of image analysis it is interesting also to investigate the case when an "external field" $\{\widetilde{S}_i : i \in G\}$ is applied; given the external field, the probability mass function is then taken to be proportional to

$$\exp\left( J \sum_{i \sim j} S_i S_j + H \sum_i S_i \widetilde{S}_i \right) . \tag{13.4}$$

In the context of binary image analysis the external field represents the observed noisy image $\{\widetilde{S}_i : i \in G\}$, while $\{S_i : i \in G\}$ is the "true" image. The probability mass function defined by (13.4) may then be taken to be the Bayesian posterior distribution of the "true" image, based on a Bayesian prior and likelihood determined by an Ising model connecting $\{\widetilde{S}_i : i \in G\}$, and $\{S_i : i \in G\}$.

We can draw from the Ising model (whether given by (13.3) or by (13.4)) by viewing it as the equilibrium distribution of the Markov chain given by the single-site update heat bath algorithm; sites are updated either at random or in systematic order by re-sampling from their conditional distribution given the configuration at all other sites. (It follows by detailed balance that the equilibrium will be a draw from the Ising model.) Thus under (13.3) we update site $i$ as follows:

$$\text{re-sampled } S_i \leftarrow +1, \text{ probability proportional to } \exp\left( +J \sum_{j : j \sim i} S_j \right) ,$$

$$\text{re-sampled } S_i \leftarrow -1, \text{ probability proportional to } \exp\left( -J \sum_{j : j \sim i} S_j \right) .$$

Similarly in the case of (13.4)

$$\text{re-sampled } S_i \leftarrow +1, \text{ probability proportional to } \exp\left(+H\widetilde{S}_i + J\sum_{j:j\sim i} S_j\right),$$

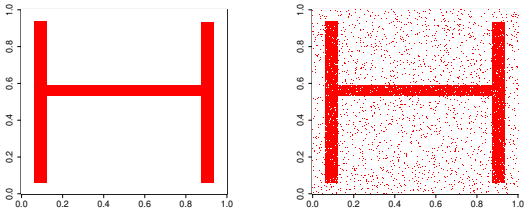
$$\text{re-sampled } S_i \leftarrow -1, \text{ probability proportional to } \exp\left(-H\widetilde{S}_i - J\sum_{j:j\sim i} S_j\right).$$

Since we take $J > 0$, it follows that the probability of the re-sampled $S_i$ being equal to 1 is monotonically increasing in the configuration $\{S_i : i \in G\}$ using the natural partial order. This is the key observation for the purposes of CFTP: as a result we can implement a monotonic coupling of the heat bath process which is a simple generalization of that used for random walk CFTP as described in Sect. 13.2.1.

A further refinement arises from the bipartite structure of the finite lattice $\{1,\ldots,N\}^2$, which allows us to implement a "coding" scheme for the updates. Viewing the lattice as a chessboard, we update all the sites on the black squares, then all the sites on the white squares, and so forth; whatever the detailed order of implementation, the statistical result will be the same. This permits extraction of the heat-bath simulation (in fact, two copies of it; see below) from *simultaneous* updates of all sites. (This refinement is no longer available for more general graphs, such as that arising from the lattice if we choose to make additional connections between diagonally adjacent sites.)

#### 13.2.4.1 Implementation

We begin by constructing a test image, represented as a $256 \times 256$ binary matrix. Noisy and clean versions of this test image are illustrated in Fig. 13.6.



(a) Original "true" image



(b) Noisy image

**Fig. 13.6** The test image for Ising CFTP

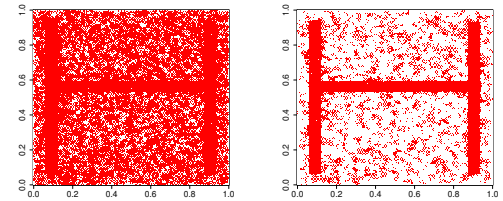


The heat-bath algorithm is based on a single `update` using an array of innovations. Note that the update is really a parallel update of *two* separate instances of the algorithm: one update confined to the "black squares", one to the "white squares" of the chessboard; this corresponds to an inbuilt period 2 for the heat-bath algorithm for this particular version of the Ising model. The `state` of the algorithm is an $M \times M$ matrix of $\pm 1$, bordered by a single strip of zeros. (This corresponds to free boundary conditions: the construction and algorithm are easily varied to produce other boundary conditions.)

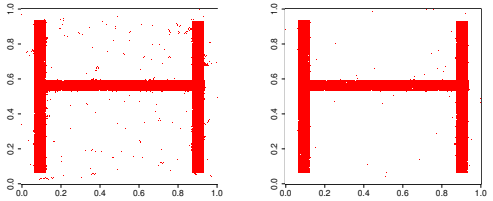
```
update <- function(state, image, innov, J = 0.9,
    H = 1.5) {
    M <- dim(image)[1]
    line <- (1:M) + 1
    work <- cbind(0, rbind(0, state, 0), 0)
    threshold <- 1/(1 + exp(-2 * (J * (work[line,
        line + 1] + work[line, line - 1] +
        work[line + 1, line] +
        work[line - 1, line]) +
        H * image)))
    state[1:M, 1:M] <- -1
    state[innov < threshold] <- +1
    return(state)
}
```

```
M <- dim(im1)[1]
J <- 0.9
H <- 1.5
state <- matrix(data = +1, nrow = M, ncol = M)
line <- (1:M) + 1
for (iter in 1:36) {
    innov <- matrix(data = runif(M * M), nrow = M,
        ncol = M)
    state <- update(state, im1, innov, J = J,
        H = H)
}
```

A sub-sequence of outputs from the parallel heat-bath algorithm is given in Fig. 13.7. Here the initial state was fixed with all spins at $+1$, so the first few updates overcome the discrepancies arising from the fact that most of the noisy image yields spins of $-1$. By update 24 it appears that equilibrium may nearly be attained. Note that (a) post-processing of the image is required in order to undo the "chess-board" encoding; (b) this post-processing yields *two* draws from equilibrium; (c) the encoding means that each of the two draws entails visiting of each node 12 times in the first 24 updates. In fact the CFTP algorithm will produce exact draws after order of 64 updates or fewer.



(a) Iteration 6 (parallel heat-bath) (b) Iteration 12



(c) Iteration 18 (d) Iteration 24

**Fig. 13.7** Parallel heat-bath reconstruction ($J = 0.9$ and $H = 1.5$). Post-processing is required.

We now package up a CFTP algorithm, which largely follows the pattern of the original random walk CFTP algorithm discussed above.

```
make_block <- function(block_length, M) {
    ilist <- list()
    for (iter in 1:block_length) {
        ilist <- c(list(matrix(data = runif(M *
            M), nrow = M, ncol = M)), ilist)
    }
    return(ilist)
}
```

```
cycle <- function(innovations, image, J = 0.6,
    H = 1.5) {
    M <- dim(image)[1]
    upper <- matrix(data = +1, ncol = M, nrow = M)
    lower <- matrix(data = -1, ncol = M, nrow = M)
    for (innov in innovations) {
        upper <- update(upper, im1, innov, J = J,
            H = H)
        lower <- update(lower, im1, innov, J = J,
            H = H)
    }
    if (sum(upper - lower) != 0)
        return(NA)
    else return(upper)
}
```



```
ising_cftp <- function(image, initial_range,
    J = 0.6, H = 1.5) {
    innovations <- make_block(initial_range,
        dim(image)[1])
    result <- NA
    while (is.na(result)) {
        innovations <- c(
            make_block(length(innovations),
            dim(image)[1]), innovations)
        result <- cycle(innovations, image, J = J,
            H = H)
    }
    return(result)
}
```

We need to post-process the result of the CFTP function `ising_cftp`, since its result interlaces two independent draws from the posterior distribution using the black/white chessboard encoding. Note that the two independent post-processed draws are guaranteed to be taken from the posterior distribution, but are *not* necessarily good results from the point of view of statistical image analysis, which would involve considerations of model adequacy and whether the parameters $J$ and $H$ have been chosen appropriately.

```
innov <- matrix(data = runif(M * M), nrow = M,
    ncol = M)
result1 <- update(result0, im1, innov, J = J,
    H = H)
chess1 <- outer(1:M, 1:M, function(x, y) ((x +
    y)%%2))
chess0 <- 1 - chess1
image0 <- result0 * chess0 + result1 * chess1
image1 <- result0 * chess1 + result1 * chess0
```

#### 13.2.4.2 More Details

There is room for considerable improvement in this algorithm as far as image analysis is concerned! Certainly it would be helpful to use more general neighbourhood structures, the better to capture boundaries which are non-rectilinear (eg: diagonal of second-nearest neighbours as well as nearest neighbours). However periodicity 2 fails for most more general lattice structures, meaning that the coding technique will also fail. This means that one must use (potentially less efficient) sequential update schemes. Performance is greatly enhanced by using a scripting language with substantial support for numerics: for example *Python* with the *Numpy* extension provides matrix operations which are implemented in a manner that allows one to encode an entire sequential image update in a single command. On the other hand, close inspection of successive updates of the CFTP algorithm (both upper and lower states) makes it apparent that much of the running time is taken up with dealing with small deviations from the equilibrium. In [141] modifications of the CFTP algorithm has been considered which are based on producing draws which are within a set distance from the perfect equilibrium draw, using Wasserstein metric.

We have noted that the major application of CFTP in [324] was to critical Ising models without external field, using Sweeney's algorithm. In [176] it is shown how to use the "bounding chain" technique to produce a CFTP algorithm for the Swendsen-Wang algorithm. Recent results [404] describe bounds for variants on heat-bath dynamics for all temperatures.

**Exercise 13.8.** Modify the function `update` for suitable $M$ to implement the case of *periodic boundary conditions*: pixels $(i_1, i_2)$ and $(j_1, j_2)$ are neighbours if $i_1 = j_1 \pm 1 \mod M$ and $i_2 = j_2 \pm 1 \mod M$. For which values of $M$ does the "parallel update" chessboard coding scheme still work correctly?

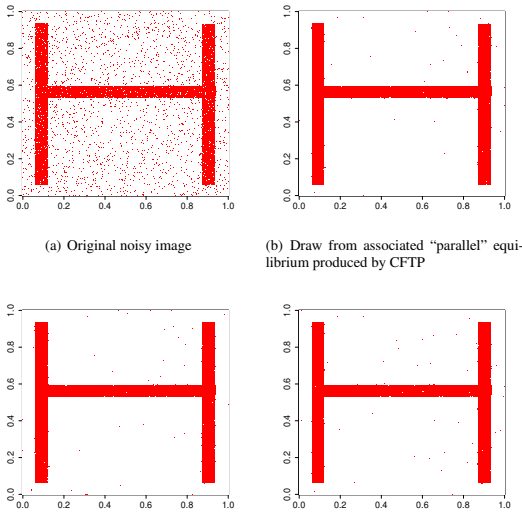
**Exercise 13.9.** Modify the functions involved in `ising_cftp` to work with rectangular images of size $M_1 \times M_2$ for suitable $M_1$ and $M_2$.

### *13.2.5 Other Remarks*

Space precludes treatment of the significant technique of the "multishift sampler", introduced in [421] and developed in [83]. A brief description is given in Sect. 2.4 of [228]; the fundamental observation is that one can draw simultaneously from all $\mathrm{Unif}([x, x+1))$ distributions $(-\infty < x < \infty)$ simply by simulating a uniformly randomized integer lattice $X + \{0, \pm 1, \pm 2, \ldots\}$ (for $X$ being $\mathrm{Unif}([0,1))$), and selecting the single point of this random lattice lying in the interval $[x, x+1)$. This is an important tool in CFTP, allowing one to reduce a continuous range of possibilities to a locally finite range.

Very recent work on convergence rates for Gibbs samplers for the $n$-simplex [375] makes intriguing use of coupling which is *non-co-adapted*; as noted in [375], this can be modified to produce CFTP algorithms (though controlling the range





(a) Original noisy image

(b) Draw from associated "parallel" equilibrium produced by CFTP

(c) First independent draw from posterior image, produced by CFTP

(d) Second independent draw from posterior image, produced by CFTP

**Fig. 13.8** Results of Ising CFTP using $J = 0.9$ and $H = 1.5$, compared with noisy image

of possibilities then produces a substantial slow factor which probably cannot be avoided).

## 13.3 Dominated Coupling-from-the-Past

In the first section we introduced classic CFTP by treating three examples (simple symmetric random walk, a Markov chain with triangle densities, the Ising model). The treatment of the first and third of these examples depends heavily on monotonicity, and moreover on there being upper and lower chains. Monotonicity is not evident in the second example, though (as indicated by Green and Murdoch) one may take a set-valued approach to this case, so that suitable monotonicity is seen

to follow from the partial ordering imposed by set-inclusion. More generally, the monotonicity requirement can to some extent be avoided by the use of bounding chains [173, 176, 226, 229].

In all three cases the significant constraint is that one is dealing with *uniform ergodicity*. We now state this notion formally, together with the associated notion of geometric ergodicity, formally. First recall the notion of *distance in total variation*

$$\mathrm{dist}_{\mathrm{TV}}(\mu,\nu) = \sup\{|\mu(A) - \nu(A)| : \text{ measurable } A\},$$

which provides a measure of the amount of agreement between probability measures $\mu$ and $\nu$.

**Definition 13.2 (Uniform ergodicity).** The Markov chain $X$ is said to be *uniformly ergodic* with equilibrium distribution $\pi$ if its distribution converges to the equilibrium distribution $\pi$ in total variation distance *uniformly in the starting point* $X_0 = x$: for some fixed $C > 0$ and for fixed $\gamma \in (0,1)$,

$$\sup_{x\in\mathcal{X}} \mathrm{dist}_{\mathrm{TV}}(\mathbf{P}(X_n \in \cdot \mid X_0 = x), \pi) \quad \leq \quad C\,\gamma^n,$$

where $\mathbf{P}(X_n \in \cdot \mid X_0 = x)$ is the conditional distribution of $X_n$ given $X_0 = x$.

Uniform ergodicity is an easy consequence of the apparently weaker assertion that as $n \to \infty$ so

$$\sup_{x\in\mathcal{X}} \mathrm{dist}_{\mathrm{TV}}(\mathbf{P}(X_t \in \cdot \mid X_0 = x), \pi) \quad \to \quad 0\,.$$

**Definition 13.3 (Geometric ergodicity).** The chain $X$ is *geometrically ergodic* with equilibrium distribution $\pi$ if there is a $\pi$-almost surely finite function $V : \mathcal{X} \to [0,\infty]$, and fixed $\gamma \in (0,1)$, such that

$$\mathrm{dist}_{\mathrm{TV}}\left(P^{(n)}(x,\cdot), \pi\right) \quad \leq \quad V(x)\,\gamma^n \qquad \text{for all } n,x\,,$$

where $P^{(n)}(x,\cdot)$ denotes the $n$-th step transition kernel of $X$.

It is evident that the very existence of coalescing upper and lower chains implies uniform ergodicity. In [124] it has been shown that the reverse implication also holds at least in principle. Thus the scope of classic CFTP is strictly limited to uniformly ergodic chains. This is a severe limitation; many chains arising in practice are geometrically ergodic but not uniformly ergodic (random walks with negative drift and reflected in the origin, birth-death-immigration processes, queues which do not turn customers away, storage processes with no upper storage limit, ...).

However there are three ways to extend CFTP to extend beyond instances of uniform ergodicity:

1. **Truncation of state-space.** One can approximate the equilibrium distribution *via* constraining the target chain by forbidding any moves which take the chain out of a large but finite subset of state-space (so that the chain does not move at all instead of carrying out a forbidden move). Under suitable conditions the equilibrium of the constrained chain can be shown to approximate that of the original chain. Particularly in case the target chain is reversible, this holds if the subset is irreducible for the constrained chain, and moreover the constrained equilibrium probabilities are proportional to the original probabilities within the constrained region. However it seems somewhat self-defeating to build an exact simulation of an approximation.
2. **Conversion to uniformly ergodic case.** In [296] it has been noted that one can sometimes convert a geometrically ergodic Markov chain into a uniformly ergodic Markov chain, simply by occasionally substituting in an independence sampler update.
3. **Domination by an amenable process.** It is shown in [226] how (in suitable cases) one can replace the upper bound (and lower bound, if required), by using a stationary random process which can be coupled to the target Markov chain so as to continue to lie above any realization of the target if it does so initially. If one can simulate this stationary process *backwards in time*, and if one can realize the coupling together with some variant on stochastic monotonicity, then this *dominated coupling-from-the-past* (domCFTP) can produce an exact simulation.



In this section we explore the method of domCFTP by describing the algorithm in the special context of a particular and very simple example. We refer to [228, Sect. 3.4, Theorem 31] for a general description of the method and a proof.

### *13.3.1 Random Walk domCFTP*

We demonstrate domCFTP by applying it to the extremely elementary example of a simple random walk with negative drift, reflected in the origin. This has the advantage of being closely related to the random walk discussed in Sect. 13.2.1; therefore the *R* code will be closely related to the code in that section.

#### 13.3.1.1 Helper Functions for domCFTP Simulation

Here is the `update` function which we will use for dominating and target Markov chains. The $\pm 1$ `innovations` need to be constructed so that the probability of a $+1$ innovation is less than $\frac{1}{2}$, in order for an equilibrium to exist (and in this case it is a straightforward exercise to show that geometric ergodicity applies).

```
update <- function(x, innov) return(max(x +
    innov, 0))
```

Note that the reflection carried out here is the reversible form of reflection also used in Sect. 13.2.1; the recipe can be adapted to other forms of reflection.

The truncation method would amount to replacing this update by an update of the form used in Sect. 13.2.1. Not only does this give only approximate answers, but also coalescence involves waiting for the upper random walk to sink to zero, or the lower random walk to rise to the top level. Clearly good approximations will result in increasing computational demands, unnecessary if compared with the domCFTP method demonstrated below. Conversion to a uniformly ergodic chain is feasible in this simple one-dimensional case, but our purpose here is to demonstrate domCFTP. For dominating process, we choose the same reflected random walk but with a larger value $p^{\mathrm{dom}} \in (p, \frac{1}{2})$ of positive jump. The dominating random walk is reversible, and its equilibrium distribution is easily computed as geometric using detailed balance: setting $\rho^{\mathrm{dom}} = p^{\mathrm{dom}}/(1-p^{\mathrm{dom}})$,

$$\pi_x^{\mathrm{dom}} \quad = \quad (1-\rho^{\mathrm{dom}})(\rho^{\mathrm{dom}})^x \quad \text{for } x = 0,1,2,\ldots . \tag{13.5}$$

We need a function `evolve`, which uses `update` to build up a trajectory from increments using an initial value.

```
evolve <- function(x, dx) {
    result <- c(x)
    for (u in dx) {
        x <- update(x, u)
        result <- append(result, x)
    }
    return(result)
}
```

We can now simulate the dominating process (in statistical equilibrium) by drawing its height at $t = 0$ from the equilibrium distribution and then simulating the same process backwards in time (initially we work back to the initial time specified by the invocation of the algorithm). We base this on generation of innovations, but now these innovations are to be viewed as occurring in reverse time.

```
make_block <- function(block_length, p_dom = 0.4) {
    return(2 * rbinom(block_length, 1, p_dom) - 1)
}
```

#### 13.3.1.2 A Typical Run of the Random Walk domCFTP algorithm

As in Sect. 13.2.1, we build up the algorithm in a sequence of steps. First we determine the trajectory of the dominating process, by using `evolve` to build up the trajectory (in reverse-time) using final value and innovations. The distribution of the dominating process at time 0 is geometric, as given in the detailed balance calculations resulting in (13.5).



```
set.seed(5)
p_dom <- 0.4
p <- 0.25
T <- -9
```

So we work here in this example with $p^{\mathrm{dom}} = 0.4$. We generate reversed dominating innovations `r_d_innovs` for the dominating process *in reversed time* and evolve the dominating `trajectory` accordingly:

```
dom0 <- rgeom(1, 1 - p_dom/(1 - p_dom))
r_d_innovs <- make_block(-T, p_dom = p_dom)
trajectory <- rev(evolve(dom0, r_d_innovs))
```

Next, we need to run the target process forwards in a way which is coupled to the dominating process. Note that the innovations `r_d_innovs` for the dominating process *cannot* be the innovations for the target process; dominating innovations are run backwards in time for the stationary dominating process, so the independence structure is quite different. We construct innovations for the target process by exploiting the reversibility of the dominating process; thus when run forwards in time the dominating process jumps by $+1$ with probability $p^{\mathrm{dom}} = 0.4$. We wish to work with probability $p < p^{\mathrm{dom}}$ for a $+1$ jump; hence $-1$ jumps of the dominating process remain negative, 0 jumps are converted to $-1$ jumps, while $+1$ jumps are converted into $-1$ jumps with probability $1 - p/p^{\mathrm{dom}}$, as performed in the following function.

```
generate_innovations <- function(trajectory,
    old_innovs, p_dom = 0.4, p = 0.25) {
    innovs <- trajectory[2:length(trajectory)] -
        trajectory[1:(length(trajectory) - 1)]
    innovs[innovs == 0] <- -1
    innovs[innovs == 1] <- 1 - 2 *
        rbinom(length(innovs[innovs ==
            1]), 1, 1 - p/p_dom)
    return(c(innovs[1:(length(innovs) -
        length(old_innovs))], old_innovs))
}
```

These innovations can now be used to generate `upper` and `lower` processes. Note the initial value for the `upper` process is given by the trajectory of the dominating process.

```
lower <- evolve(0, innovations)
upper <- evolve(trajectory[1], innovations)
```

Coalescence not yet having occurred, we need to extend the dominating process. We work in reversed time as before.

```
old_upper <- upper
old_lower <- lower
old_T <- T
extend_r_d_innovs <- function(innovs,
    p_dom = 0.4) {
    return(c(innovs, make_block(length(innovs),
        p_dom = p_dom)))
}
trajectory <- rev(evolve(dom0,
    extend_r_d_innovs(r_d_innovs,
    p_dom = p_dom)))
T <- 1 - length(trajectory)
```

Having done this, we may now simulate the new upper and lower processes. Care must be taken to re-use randomness, but this is ensured by the construction of the `generate_innovations` function invoked above.

```
lower <- evolve(0, innovations)
upper <- evolve(trajectory[1], innovations)
```

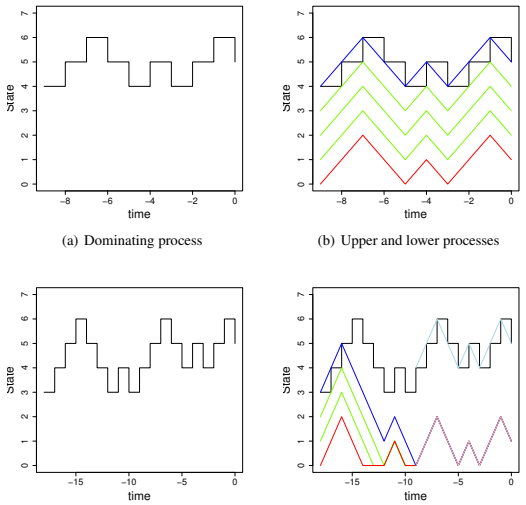
We now achieve coalescence. Note that the consequence of re-use of randomness is that old upper and lower processes "funnel" between newer upper and lower processes.

#### 13.3.1.3 Implementing the Algorithm

Finally we package the domCFTP algorithm in a function. Note that the output of `dom_cftp(T)` is *not* stable under variations of the start-time parameter `T` even if the seed of the random number generator is held fixed: this arises because each cycle of this particular domCFTP requires the use of the random number generator to impute innovations for the target process. In order to keep the code simple, this implementation does not synchronize each imputation with the generation of the corresponding entry in `r_d_innovs`, which is what would be required to stabilize the algorithm.

```
dom_cftp <- function(T, p_dom = 0.4, p = 0.25) {
    dom0 <- rgeom(1, 1 - p_dom/(1 - p_dom))
    r_d_innovs <- make_block(-T, p_dom = p_dom)
    trajectory <- rev(evolve(dom0, r_d_innovs))
    innovs <- generate_innovations(trajectory,
        c(), p_dom = p_dom, p = p)
    lower <- evolve(0, innovs)
    upper <- evolve(trajectory[1], innovs)
    while (lower[1 - T] != upper[1 - T]) {
```






(a) Dominating process (b) Upper and lower processes

(c) Extended dominating process (d) Extended upper and lower processes

**Fig. 13.9** domCFTP output for a simple random walk example

```
        old_innov <- innovs[]
        old_lower <- lower[]
        old_upper <- upper[]
        r_d_innovs <- extend_r_d_innovs(r_d_innovs,
            p_dom = p_dom)
        T <- -length(r_d_innovs)
        trajectory <- rev(evolve(dom0, r_d_innovs))
        innovs <- generate_innovations(trajectory,
            innovs, p_dom = p_dom, p = p)
        old_innov_range <- (length(innovs) -
            length(old_innov) + 1):length(innovs)
        lower <- evolve(0, innovs)
        upper <- evolve(trajectory[1], innovs)
        old_range <- (length(lower) + 1 -
```

```
            length(old_lower)):length(lower)
    }
    return(upper[length(upper)])
}
```

#### 13.3.1.4 Statistical Analysis

We now test the algorithm. It is worth paying particular attention to whether the actual implementation maintains the fundamental "funnelling" relationship

$$\texttt{old\_lower} \leq \texttt{lower} \leq \texttt{upper} \leq \texttt{old\_upper} \leq \texttt{trajectory}.$$

This can be checked during evolution of the algorithm by using the *R* function `stopifnot`.

Detailed inspection of the operation of the domCFTP algorithm reveals that in the following 10000 runs the algorithm reaches back on occasion to time $-120$ in order to achieve coalescence.

```
N <- 10000
data <- sapply(rep(-30, N), dom_cftp)
est <- sum(data > 0)/N
sd <- sqrt(est * (1 - est)/N)
print(paste("Empirical estimate of p / (1 - p) (=",
    format(p/(1 - p), digits = 3), "): ",
    format(est, digits = 3), "+/-", format(2 *
        sd, digits = 1), sep = ""))
```

This yields output giving an empirical estimate for $p/(1-p)$:

```
[1] "Empirical estimate of p / (1 - p) (=0.333): 0.331+/-0.01"
```

The above empirical estimate forms part of a larger analysis, comparing log-frequencies with their theoretical expectations, which is graphed in Fig. 13.10.

```
rho <- p/(1 - p)
counts <- tabulate(1 + data)
k <- 5
probs <- c(dgeom(0:k, 1 - rho), pgeom(k, 1 -
    rho, lower.tail = FALSE))
counts
```

The above code fragment produces output tabulating the output of the domCFTP algorithm.

```
 [1] 6688 2183  742  268   80   21   13    2    1    1
[11]    0    1
```



We now compute the expected output:

```
N * probs
```

with output as follows.

```
[1] 6666.66667 2222.22222  740.74074  246.91358
[5]   82.30453   27.43484   13.71742
```

Finally we perform a $\chi^2$ test,

```
chisq.test(tabulate(1 + data, nbins = k +
    2), p = probs)
```

with output as follows.

```
        Chi-squared test for given probabilities

data:  tabulate(1 + data, nbins = k + 2)
X-squared = 4.1744, df = 6, p-value = 0.6531
```

The $\chi^2$ test has been carried out on samples of size $10^6$, and still reports no significant deviations from the predicted equilibrium distribution.

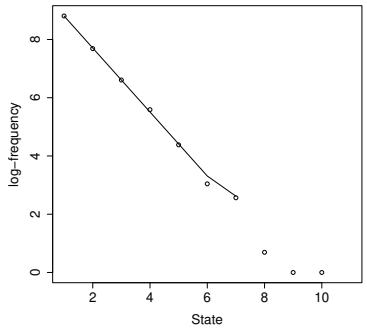



**Fig. 13.10** Plot of log-frequencies of domCFTP output for the simple random walk example: the line indicates the theoretical expectation. Note that sampling variation is high at low log-frequencies, so the theoretical line is not plotted there.

While conveniently simple, this example is unrepresentative in a number of ways. Firstly, the dominating process often visits 0, thus forcing coalescence of upper and lower processes at such times. In more general applications of domCFTP such convenient coalescence will be very rare. Speaking technically, one should generally not expect the dominating process in substantial applications to exhibit a practically useful regenerative atom. Secondly, the target process is a random walk (and we are using a rather simple synchronous coupling of upper and lower processes), and this means that upper and lower processes themselves can only coalesce when the upper process hits 0 (in contrast to the continuous-time non-linear birth-death example discussed in [225, 228]). Nevertheless, the simplicity of this example permits clear demonstration of the underlying principles of domCFTP unhindered by technicalities.

### *13.3.2 Other Remarks*

While the random walk domCFTP example is trivial, it serves as a useful prototype for much more complicated and useful examples. There are a variety of instances of more realistic examples of domCFTP. We have already noted the continuous-time non-linear birth-death example discussed in [225, 228]; this extends naturally to treatment of spatial point processes (such as area-interaction point processes and Strauss point processes), to be found in [225, 226, 229]; see also the tutorial paper [36]. Note that many spatial point processes are produced by non-attractive spatial birth-and-death processes; in this case a simple crossover trick can be applied to ensure upper and lower processes envelope the perfectly simulated trajectory [226, 229]. A nice application to the (super-stable) M/G/s queue is given in [372]. In [284] the ideas of domCFTP have been applied to produce an intriguing model diagnostic for a spatial point process. Finally, in [119] it is shown how to use domCFTP to produce exact draws from Vervaat's perpetuity.

**Exercise 13.10.** Modify the function `dom_cftp` to work for a target process which evolves as a *modified* random walk $X$ with negative bias: at each time $n$ such that $X_n > 1$ the next step $X_{n+1} - X_n$ is replaced by a double jump $-2$ with some small positive probability, independently of the past and of the step $X_{n+1} - X_n$.

## 13.4 Conclusion

As noted in Sect. 13.3, the equivalence of (a) uniform ergodicity and (b) existence (at least in principle) of a classical CFTP algorithm has been shown in [124]. The implication "classical CFTP implies uniform ergodicity" is an immediate calculation, using the consideration that bounds on coalescence time will generate bounds on convergence in total variation, and it is well-known in the field how to improve such bounds to establish geometric decay. The reverse implication follows by noting



the fact that uniform ergodicity implies that the whole state-space is a small set of lag $k$ for some $k > 0$. But then the $k$-sub-sampled chain makes the whole state-space small of lag 1, and then small set CFTP may be applied as in Sect. 13.2.3.

Of course this small set CFTP only works in principle. To make it work in practise, one would need to determine the lag $k$ and the regeneration measure $\rho\nu(\cdot)$ of the state-space, and one would also need to know how to draw from the $k$-step transition probability conditioned on regeneration *not* occurring. In practise one would at least need $k = 1$; moreover it would be crucial to consider whether the actual run-time of the algorithm made it a feasible means of achieving draws from the equilibrium. However, whether practical or impractical, the existence of small set CFTP is surely of interest.

It is then natural to ask whether there is a relationship between geometric ergodicity and domCFTP. Even the simple example of Sect. 13.3 provides an example of a geometrically ergodic chain which is not uniformly ergodic, and yet possesses a domCFTP algorithm. However general geometrically ergodic chains do not possess natural monotonicity structure, and this appears to be an obstacle to the existence of domCFTP.

However for Markov chains satisfying the weak property of $\varphi$-irreducibility it is actually the case that geometric ergodicity is equivalent to the existence of a geometric Foster-Lyapunov criterion (for $\varphi$-irreducibility and this equivalence result we refer to the famous monograph of Meyn and Tweedie [275]). Namely, let us consider the following condition.

**Definition 13.4 (Geometric Foster-Lyapunov condition).** The Markov chain $X$ satisfies a geometric Foster-Lyapunov condition if there are: a small set $C$; constants $\alpha \in (0,1)$, $b > 0$; and a scale or drift function $\Lambda : \mathcal{X} \to [1,\infty)$ bounded on $C$; such that $C$ is a $\Lambda$-sub-level set, and for $\pi$-almost all $x \in \mathcal{X}$

$$\mathbf{E}\left(\Lambda(X_{n+1}) \mid X_n = x\right) \quad \leq \quad \alpha\Lambda(x) + b\mathbf{1}_{(x\in C)}\,,$$

where $\pi$ is the equilibrium measure of $X$.

The following can then be established [227]: stochastic comparisons using the Markov inequality establish that $\Lambda(X_n)$ can be dominated by $D$, the scalar multiple of the exponential of the workload of a D/M/1 queue sampled at arrivals. The dominating chain will not itself necessarily be recurrent. However under $k$-sub-sampling for suitable $k$ and variation of the small set $C = \{x : \Lambda(x) \leq c\}$ the domination can be refined to produce a dominating Markov chain $D$ which *is* recurrent. Although $D$ is not reversible, nonetheless one can use duality to compute its time-reversal under equilibrium. Refining the sub-sampling if necessary, one can then show that a sub-sampling of $\Lambda(X)$ is dominated by a stationary dominating Markov chain $D$, for which the equilibrium and time-reversed statistics are known, and such that regeneration occurs whenever $D$ sinks below the level $c$. This suffices to establish a domCFTP algorithm, impractical in the same way that the Foss-Tweedie CFTP is impractical for the case of uniform ergodicity.

Despite its impracticality, this queue-based domCFTP algorithm bears a clear resemblance to the practical domCFTP algorithms discussed in the literature. It

suggests the following challenging question: if one can prove that a chain is geometrically ergodic, then shouldn't one try to exhibit a domCFTP algorithm?

It is natural to ask whether geometric ergodicity is equivalent to domCFTP. However the answer to this is negative. It has been shown in [79] that domCFTP can be established for classes of non-geometrically ergodic Markov chains (specifically, polynomially ergodic Markov chains satisfying a technical condition known as "tameness"). This has in turn led to theoretical advances in the theory of polynomially ergodic Markov chains [78].