Parallel Systematic Resampling*

Vicke Noren† Ingvar Strid‡

May 6, 2024

Abstract

Resampling is a crucial and computationally demanding step of the particle filter. This paper shows how to implement a parallel systematic resampling algorithm in a relatively straightforward manner. This is achieved by casting the algorithm in compact matrix form and decomposing it into a set of operations on vectors, thereby avoiding loops at least at the highest level of programming. The proposed algorithm can be parallelized such that each process can obtain replication factors for its subset of particles. The advantage of the algorithm is two-fold; it contributes to a performance boost and is relatively easy to implement as it allows for resampling output to be independent of the number of processors. Compared with common resampling algorithms such as systematic resampling, the gains of the proposed algorithm are particularly large if resampling is subject to a significant difference between the number of input and output particles.

1 Introduction

Nonlinear and non-Gaussian state-space models are complex models known for their intractability. The breakthrough in state estimation was pioneered by Gordon et al. (1993) in the context of motion tracking. With a particle filter, estimation of the state process in terms of the posterior density becomes tractable by sampling so-called particles X in a sequential manner and evaluating the likelihood to obtain (unnormalized) particle weights $\tilde{w}$. The estimation routine, however, risks to rapidly become inefficient due to weight degeneracy, i.e., the weights converge to a few particles (Cappé et al., 2005, Ch. 7). The remedy is to resample the particles in an additional stage such that particles with small weights prune away (Pitt and Shephard, 1999). Altogether, the estimation method is referred to as sequential importance sampling with resampling (SISR).

This paper focuses on the resampling operation since it is the key part in the parallelization of the particle filter. The resampling step is the bottleneck of a parallel particle filter as the sampling and importance steps of SISR are much more suitable for parallelization. In fact, in a parallel particle filter, resampling is the only part of the algorithm that creates a need for frequent interprocessor communication. To derive a parallel algorithm, Section 2 presents a couple of widely used resampling methods as well as the residual systematic resampling algorithm. Section 3 outlines how the latter can be written compactly in matrix form. The resampling algorithm is then presented in Section 4, after which we conclude our main results and their implications.

*The opinions expressed in this article are the sole responsibility of the authors and should not be interpreted as reflecting the views of Sveriges Riksbank.

†Research Division, Sveriges Riksbank, 103 37 Stockholm, Sweden. Email: vicke.noren@gmail.com.

‡Monetary Policy Department, Sveriges Riksbank, 103 37 Stockholm, Sweden. Email: ingvar.strid@riksbank.se.

2 Resampling Algorithms

With $\chi_{\text{in}} = \{X_{\text{in}}^n, w^n\}_{n=1}^{N_{\text{in}}}$ and $\chi_{\text{out}} = \{X_{\text{out}}^n, N_{\text{out}}^{-1}\}_{n=1}^{N_{\text{out}}}$, where X_{in} and X_{out} are, respectively, input and output particles, a resampling method is defined by the mapping $\chi_{\text{in}} \mapsto \chi_{\text{out}}$. The number of input and output particles are denoted by N_{in} and N_{out}. Particle weights are normalized through $w^n = \tilde{S}^{-1}\tilde{w}^n$, where $\tilde{S} = \sum_{n=1}^{N_{\text{in}}} \tilde{w}^n$. The replication factor of the nth particle, r^n, designates the number of occurrences of X_{in}^n in X_{out} such that the sum of replication factors equals the number of output particles, i.e., $\sum_{n=1}^{N_{\text{in}}} r^n = N_{\text{out}}$. Consequently, the vector of replication factors, $r = (r^1 \ \cdots \ r^{N_{\text{in}}})^\intercal$, where $r^\intercal$ denotes the transpose of r, is of length N_{in}, whereas the index vector, i, which contains the indices in X_{in} of the resampled particles, is of length N_{out}.

As a brief example to illustrate the terminology, suppose that $N_{\text{in}} = 3$, $N_{\text{out}} = 5$ and that one draws five particles from $X_{\text{in}} = (x^1 \ x^2 \ x^3)^\intercal$ and obtains $X_{\text{out}} = (x^1 \ x^1 \ x^3 \ x^3 \ x^3)^\intercal$. Then the vector of replication factors is $r = (2 \ 0 \ 3)^\intercal$ and the index vector is $i = (1 \ 1 \ 3 \ 3 \ 3)^\intercal$.

Common methods to perform resampling include multinomial sampling, residual sampling (RR), stratified sampling, and systematic resampling (SR). Carpenter et al. (1999) introduced systematic resampling, see Algorithm 1. Systematic sampling (SS) may be viewed as a special case of systematic resampling (Tillé, 2006), as the weights have to satisfy

$$0 < N_{\text{out}} w^n < 1, \qquad n = 1, \ldots, N_{\text{in}},$$

such that r^n equals either zero or one. Bolić et al. (2003) suggest a computationally efficient variant of SR, which they call residual systematic resampling (RSR), see Algorithm 2. The algorithm performs systematic resampling, i.e., RSR is equivalent to SR from a statistical perspective. Another method that is statistically equivalent to systematic resampling is the residual resampling algorithm, see Algorithm 3, where SR (or alternatively RSR) is used in the remainder resampling step. Since systematic resampling is the bottleneck of RR in all except the optimal case, that is, when $N_r = 0$, it is natural to focus on the optimization of SR. However, the inner loop of Algorithm 1 makes the implementation of pipelining difficult. By contrast, Bolić et al. (2003) point out some beneficial properties of RSR: (i) it contains only a single loop, (ii) its duration is independent of the number of output particles, (iii) its complexity is $\mathcal{O}(N_{\text{in}})$, and (iv) it is suitable for parallel processing since it can be implemented without conditional branches. The gains from using RSR are particularly large if resampling is "unbalanced," i.e., if the number of output particles from resampling significantly exceeds the number of input particles, as RSR always attains $\mathcal{O}(N_{\text{in}})$, whereas SR uses two loops and has complexity $\mathcal{O}(\max(N_{\text{in}}, N_{\text{out}}))$.

Systematic resampling is often the preferred resampling method in applications due to its computational simplicity (Cappé et al., 2005, p. 250). With this in mind, we show how to boost its performance in terms of reduced execution time obtained through parallel processing. To achieve this goal, we derive the RSR algorithm in matrix form, which is suitable for parallel implementation.

3 RSR Algorithm in Matrix Form

Let the N_{out}-scaled vector of weights be denoted by $\hat{w} = N_{\text{out}} w$, let $v = (u \ 0 \ \cdots \ 0)^\intercal$ be a vector of length N_{in} with $u \sim U(0, 1)$, and let the ceiling operator applied to a vector v of length L be $\lceil v \rceil = (\lceil v^1 \rceil \ \cdots \ \lceil v^L \rceil)$, and analogously for the floor operator. Furthermore, let the lower triangular $N_{\text{in}} \times N_{\text{in}}$ matrix A be the cumulative sum operator, with its inverse, A^{-1}, being the

Algorithm 1. Systematic resampling.

function SR(w, N_{in}, N_{out}, u)
  $u = N_{\text{out}}^{-1} u$
  $s = 0$
  for $n = 1$ to N_{in} **do**
    $k = 0$
    $s = s + w^n$
    while $s > u$ **do**
      $k = k + 1$
      $u = u + N_{\text{out}}^{-1}$
    end
    $r^n = k$
  end
  return r
end

Algorithm 2. Residual systematic resampling.

function RSR(w, N_{in}, N_{out}, u)
  $u = N_{\text{out}}^{-1} u$
  for $n = 1$ to N_{in} **do**
    $r^n = \lfloor N_{\text{out}}(w^n - u) \rfloor + 1$
    $u = u + N_{\text{out}}^{-1} r^n - w^n$
  end
  return r
end

Algorithm 3. Residual resampling.

function RR(w, N_{in}, N_{out}, u)
  $N_r = N_{\text{out}}$
  for $n = 1$ to N_{in} **do**
    $r^n = \lfloor N_{\text{out}} w^n \rfloor$
    $w_r^n = N_{\text{out}} w^n - r^n$
    $N_r = N_r - r^n$
  end
  if $N_r > 0$ **then** ▷ Remainder step.
    for $n = 1$ to N_{in} **do**
      $w_r^n = N_r^{-1} w_r^n$
    end
    $r_r = r(w_r, N_{\text{in}}, N_r, u)$ ▷ Resample with SR or RSR.
    for $n = 1$ to N_{in} **do**
      $r^n = r^n + r_r^n$
    end
  end
  return r
end

difference operator, such that

$$A_{i,j} = \begin{cases} 1 & \text{if } i \geq j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{and} \qquad A^{-1}_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ -1 & \text{if } i = j+1, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $A_{i,j}$ denotes the element of A in row i and column j. This leads us to the matrix form of RSR.

Lemma 1. *The RSR algorithm can be written compactly in matrix form as*

$$r = r_{RSR}(w, N_{in}, N_{out}, u) = A^{-1}\lceil A(N_{out}w - v)\rceil. \tag{2}$$

Proof. Unrolling the loop of Algorithm 2 yields

$$\begin{aligned} r^1 &= \lfloor N_{\text{out}} w^1 - u \rfloor + 1, \\ r^2 &= \lfloor N_{\text{out}}(w^1 + w^2) - u - r^1 \rfloor + 1, \\ &\vdots \\ r^n &= \left\lfloor N_{\text{out}} \sum_{i=1}^{n} w^i - u - \sum_{i=1}^{n-1} r^i \right\rfloor + 1 = \left\lfloor \sum_{i=1}^{n} \hat{w}^i - u \right\rfloor - \sum_{i=1}^{n-1} r^i + 1, \end{aligned} \tag{3}$$

where the last equality in (3) follows from the fact that any sum of replication factors is an integer, and $\lfloor a + b \rfloor = a + \lfloor b \rfloor$, if a is an integer and b is a real number. Since $\lceil a \rceil = \lfloor a \rfloor + 1$, if a is not an integer, one can rewrite (3) as

$$r^n = \left\lfloor \sum_{i=1}^{n} \hat{w}^i - u \right\rfloor + 1 - \left\lfloor \sum_{i=1}^{n-1} w^i - u \right\rfloor - 1 = \left\lceil \sum_{i=1}^{n} N_{\text{out}} w^i - u \right\rceil - \left\lceil \sum_{i=1}^{n-1} w^i - u \right\rceil,$$

which is the equation for the nth replication factor in (2). □

Clearly, the RSR algorithm should not be implemented directly as stated in (2), since N_{in} is typically quite large. However, the compact expression in (2) shows how to implement RSR as a sequence of operations on vectors.

Corollary 1. *Let λ be a non-zero scalar. Then*

$$r_{RSR}(w, N_{in}, N_{out}, u) = r_{RSR}(\lambda w, N_{in}, \lambda^{-1} N_{out}, u).$$

Proof. Clearly,

$$r_{\text{RSR}}(w, N_{\text{in}}, N_{\text{out}}, u) = A^{-1}\lceil A(N_{\text{out}}w - v)\rceil = A^{-1}\lceil A(\lambda^{-1}N_{\text{out}}\lambda w - v)\rceil = r_{\text{RSR}}(\lambda w, N_{\text{in}}, \lambda^{-1}N_{\text{out}}, u).$$

□

With $N_{\text{in}} = N_{\text{out}} = N$ and $\lambda = \tilde{S}$, the Corollary 1 states that the RSR algorithm can be applied without prior normalization of weights, i.e.,

$$r = r_{\text{RSR}}(w, N_{\text{in}}, N_{\text{out}}, u) = r_{\text{RSR}}(\tilde{w}, N, \tilde{S}^{-1}N, u).$$

Corollary 2. *Let r be the vector of replication factors. Then r can be decomposed into*

$$r = \lfloor N_{out} w \rfloor + r_r,$$

where the vector of residual replication factors is given by

$$r_r = r_{RSR}(w_r, N_{in}, N_r, u),$$

and where

$$w_r = \frac{N_{out} w - \lfloor N_{out} w \rfloor}{N_r} \qquad \text{and} \qquad N_r = N_{out} - \sum_{i=1}^{N_{in}} \lfloor N_{out} w^i \rfloor.$$

Proof. Making use of Lemma 1, Corollary 1 and simple properties of the floor and ceil operators gives

$$\begin{aligned} r_r &= r_{\mathrm{RSR}}(w_r, N_{\mathrm{in}}, N_r, u) & &= r_{\mathrm{RSR}}(N_{\mathrm{out}} w - \lfloor N_{\mathrm{out}} w \rfloor, N_{\mathrm{in}}, 1, u) \\ &= A^{-1} \lceil A(N_{\mathrm{out}} w - \lfloor N_{\mathrm{out}} w \rfloor - v) \rceil & &= A^{-1} \lceil A(N_{\mathrm{out}} w - v) - A \lfloor N_{\mathrm{out}} w \rfloor \rceil \\ &= A^{-1} (\lceil A(N_{\mathrm{out}} w - v) \rceil - \lceil A \lfloor N_{\mathrm{out}} w \rfloor \rceil) & &= r_{\mathrm{RSR}}(w, N_{\mathrm{in}}, N_{\mathrm{out}}, u) - \lfloor N_{\mathrm{out}} w \rfloor. \end{aligned}$$

□

4 Parallel Systemic Resampling Algorithm

The main objective of this paper is to construct a parallel systematic resampling algorithm, i.e., an algorithm that is statistically equivalent to the SR algorithm for any number of processors, P, applied with the algorithm. The derivation of a parallel SR algorithm is greatly simplified once the RSR algorithm has been cast in matrix form.

Without loss of generality, we assume that $M = N_{\mathrm{in}} P^{-1}$ is an integer. At the outset of resampling, each process p is assumed to hold an arbitrarily scaled $M \times 1$ vector of weights denoted $\tilde{w}_p$. Here we use the notation $v_p^m = v^{(p-1)M+m}$, $m = 1, \ldots, M$, for a vector $v = (v_1^\intercal \ \cdots \ v_P^\intercal)^\intercal = (v_1^1 \ \cdots \ v_1^M \ \cdots \ v_P^1 \ \cdots \ v_P^M)^\intercal = (v^1 \ \cdots \ v^{N_{\mathrm{in}}})^\intercal$ containing P subvectors such that v_p is the pth subvector and v^n is its nth element of v. Analogously to the setup of Lemma 1, the vector of N_{out}-scaled weights is partitioned as $\hat{w} = (\hat{w}_1^\intercal \ \cdots \ \hat{w}_P^\intercal)^\intercal$, where each subvector is defined as $\hat{w}_p = N_{\mathrm{out}} \tilde{S}^{-1} \tilde{w}$, hence $\tilde{S} = \sum_{p=1}^P \tilde{S}^p$, where $\tilde{S}^p = \tilde{w}_p \iota_M$ and ι_M is a vector of ones of length M, is a global quantity, i.e., it must be known to all processes. Let $v = (u \ 0 \ \cdots \ 0)^\intercal$ and $\hat{v}_p = (\hat{v}_p^1 \ 0 \ \cdots \ 0)^\intercal$ be vectors of length M, where $u \sim U(0,1)$, $\hat{v}_p^1 = u - \hat{C}^p$ and

$$\hat{C}^p = \begin{cases} \dfrac{N_{\mathrm{out}}}{\tilde{S}} \displaystyle\sum_{i=1}^{p-1} \tilde{S}^i & \text{for } p = 2, \ldots, P, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Furthermore, the cumulative sum of N_{out}-scaled weights,

$$\hat{c}^n = \sum_{i=1}^n \hat{w}^i, \quad n = 1, \ldots, N_{\mathrm{in}},$$

are related to the cumulative sum of process weights through

$$\hat{c}^{(p-1)M+j} = \hat{C}^p + \sum_{i=1}^j \hat{w}^{(p-1)M+i} = \hat{C}^p + \sum_{i=1}^j \hat{w}_p^i.$$

In particular, $\hat{v}_p^1 = u - \hat{c}^{(p-1)M}$ since $\hat{C}^p = \hat{c}^{(p-1)M}$.

Proposition 1. *In the parallel RSR algorithm, each process p obtains its vector of replication factors as*

$$r_p = \bar{r}_p - \lceil -\hat{v}_p \rceil,$$

where $\bar{r}_p = r_{RSR}(\hat{w}_p, M, 1, \hat{v}_p^1)$ is an auxiliary replication factor.

Proof. Using Lemma 1, the vector of replication factors can be written explicitly as

$$r = \begin{pmatrix} \begin{pmatrix} \lceil \hat{c}^1 - u \rceil \\ \lceil \hat{c}^2 - u \rceil - \lceil \hat{c}^1 - u \rceil \\ \vdots \\ \lceil \hat{c}^M - u \rceil - \lceil \hat{c}^{M-1} - u \rceil \end{pmatrix} \\ \begin{pmatrix} \lceil \hat{c}^{M+1} - u \rceil \\ \lceil \hat{c}^{M+2} - u \rceil - \lceil \hat{c}^{M+1} - u \rceil \\ \vdots \\ \lceil \hat{c}^{2M} - u \rceil - \lceil \hat{c}^{2M-1} - u \rceil \end{pmatrix} \\ \vdots \\ \begin{pmatrix} \lceil \hat{c}^{(P-1)M+1} - u \rceil \\ \lceil \hat{c}^{(P-1)M+2} - u \rceil - \lceil \hat{c}^{(P-1)M+1} - u \rceil \\ \vdots \\ \lceil \hat{c}^{PM} - u \rceil - \lceil \hat{c}^{PM-1} - u \rceil \end{pmatrix} \end{pmatrix} - \begin{pmatrix} \begin{pmatrix} \lceil \hat{c}^0 - u \rceil \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\ \begin{pmatrix} \lceil \hat{c}^M - u \rceil \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} \lceil \hat{c}^{(P-1)M} - u \rceil \\ 0 \\ \vdots \\ 0 \end{pmatrix} \end{pmatrix}$$

and, with $\bar{r}_p = r_{\mathrm{RSR}}(\hat{w}_p, M, 1, \hat{v}_p^1)$, be expressed in the partitioned form

$$\begin{aligned} r &= \begin{pmatrix} A_M^{-1} \lceil A_M(\hat{w}_1 - v) + \hat{C}^1 \iota_M \rceil \\ A_M^{-1} \lceil A_M(\hat{w}_2 - v) + \hat{C}^2 \iota_M \rceil \\ \vdots \\ A_M^{-1} \lceil A_M(\hat{w}_P - v) + \hat{C}^P \iota_M \rceil \end{pmatrix} - \begin{pmatrix} \lceil -\hat{v}_1 \rceil \\ \lceil -\hat{v}_2 \rceil \\ \vdots \\ \lceil -\hat{v}_P \rceil \end{pmatrix} \\ &= \begin{pmatrix} A_M^{-1} \lceil A_M(\hat{w}_1 - \hat{v}_1) \rceil \\ A_M^{-1} \lceil A_M(\hat{w}_2 - \hat{v}_2) \rceil \\ \vdots \\ A_M^{-1} \lceil A_M(\hat{w}_P - \hat{v}_P) \rceil \end{pmatrix} - \begin{pmatrix} \lceil -\hat{v}_1 \rceil \\ \lceil -\hat{v}_2 \rceil \\ \vdots \\ \lceil -\hat{v}_P \rceil \end{pmatrix} = \begin{pmatrix} \bar{r}_1 \\ \bar{r}_2 \\ \vdots \\ \bar{r}_P \end{pmatrix} - \begin{pmatrix} \lceil -\hat{v}_1 \rceil \\ \lceil -\hat{v}_2 \rceil \\ \vdots \\ \lceil -\hat{v}_P \rceil \end{pmatrix}, \end{aligned}$$

since

$$A_M(\hat{w}_p - v) + \hat{C}^p \iota_M = A_M \hat{w}_p - (\hat{C}^p - u)\iota_M = A_M(\hat{w}_p - \hat{v}_p).$$

□

Proposition 1 shows how process p can obtain the replication factors for its subset of particles. Hence, systematic resampling can be performed in parallel.

In a parallel particle filter, the importance stage of SISR, which is trivially parallelizable since there are no data dependencies between particles, is carried out in blocks of M particles, which are treated simultaneously by the processes. The suggested way to carry out the resampling stage is then in terms of implementing so-called *gather-scatter* operations as outlined by the distributed residual systematic resampling (DRSR) algorithm, see Algoirthm 4. *Scatter* partitions a vector

$v = (v_1 \ \cdots \ v_P)^\intercal$ initially stored by the root process and transmits the subvector v_p from the root process to process p for $p = 1, \ldots, P$. *Gather* is the inverse of *scatter*; the processes transmit v_p to the root process. Upon completion of *gather*, the root process is left with the full vector v.

Algorithm 4. Distributed residual systematic resampling.

1: **function** DRSR(w, N_{in}, N_{out}, u, P)
2: At the outset of resampling, each process p holds weights $\tilde{w}_p$.
3: In parallel, each process computes $\tilde{S}^p$.
4: The master process collects $\{\tilde{S}^p\}_{p=1}^P$. ▷ Gather.
5: The master process generates $u \sim U(0,1)$ and computes $\tilde{S}$, $\hat{C}^p$ and $\hat{v}_p^1$.
6: The master process distributes the pair $(\tilde{S}, \hat{v}_p^1)$ to each process. ▷ Scatter.
7: In parallel, each process computes $\bar{r}_p$ and r_p^1 using Proposition 1.
8: **return** r
9: **end**

It is straightforward to implement scattering, i.e., the penultimate step of Algorithm 4, such that the replication factors are obtained in order, which is important in the parallel particle filter in order to simplify the packaging of particles for particle routing.[1] In the final step of DRSR, only the first auxiliary replication factor in each process has to be corrected, as $\hat{v}_p^n = 0$ for $n > 1$. It is important to note that, in this step, the RSR algorithm is used merely as a computational device.

In contrast to Bolić et al. (2003), the construction of the RSR algorithm is such that it takes a uniform random number as input, thus avoiding unnecessary random number generations each time the algorithm is revoked by the parallel particle filter. Instead, a uniform random number is generated by the master process and transferred to each process through *scatter*.

To our knowledge the DRSR algorithm is the first parallel resampling algorithm with the property that the resampling output does not depend on the number of processors, P, applied with the algorithm.

5 Conclusion

Due to the fact that resampling is a key ingredient in the construction of a parallel particle filter, it is natural to put emphasize on this step in order to speed up the algorithm. Residual systematic resampling is of great interest in this context, not only because it is statistically equivalent to other common resampling methods such as RR and SR, but also because it can be run in parallel.

In the light of this, we start by presenting the algorithm and compare it with other resampling methods. Then we derive the matrix form such that it can be implemented as a sequence of vector operations. The parallelization of the RSR algorithm results in a succinct algorithm that is relatively easy to implement in terms of *gather-scatter* operations.

References

Bolić, M., P.M. Djurić, and S. Hong (2003). New Resampling Algorithms for Particle Filters, *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*

[1] Particle routing is a rebalancing process that transfers particles between processes such that each process ends up holding M particles.

2, 589—592.

Cappé, O., E. Moulines, and T. Rydén (2005). Inference in Hidden Markov Models, *Springer*, New York.

Carpenter, J., P. Clifford, and P. Fearnhead (1999). Improved Particle Filter for Nonlinear Problems, *IEE Proceedings – Radar and Sonar Navigation* 146(1), 2—7.

Gordon, N.J., D.J. Salmond, and A.F.M. Smith (1993). Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation, *IEE Proceedings F* 140(2), 107–113.

Pitt, M.K., and N. Shephard (1999). Filtering via Simulation: Auxiliary Particle Filter, *Journal of the American Statistical Association* 94(446), 590–599.

Tillé, Y. (2006). Sampling Algorithms, *Springer*, New York.

Appendix

Equivalence of RSR and RR with RSR applied for remainder step

Proposition 2. *Suppose that N_{in}, N_{out} and w are fixed. Furthermore, suppose that r_{RSR} and r_{RR} are vectors of replication factors obtained with residual systematic resampling and residual resampling with RSR applied for the remainder step. Then the methods are equivalent, i.e., $r_{RSR} = r_{RR}$.*

Proof. Let c denote the vector of cumulative sum of replication factors, with the nth element given as

$$c^n = \sum_{i=1}^{n} r^i.$$

From Lemma 1 and particularly equation (3) it is clear that one can write the vector of cumulative sum of replication factors obtained with RSR as

$$c^n_{\text{RSR}} = \left\lceil N_{\text{out}} \sum_{i=1}^{n} w^i - u \right\rceil.$$

Now, with

$$w^n_r = \frac{w^n - \lfloor N_{\text{out}} w^n \rfloor}{\sum_{i=1}^{N_{\text{in}}} N_{\text{out}} w^i - \lfloor N_{\text{out}} w^i \rfloor},$$

the nth replication factor obtained with residual resampling is

$$\begin{aligned} r^n_{\text{RR}} &= \lfloor N_{\text{out}} w^n \rfloor + \left\lceil \left(\sum_{i=1}^{N_{\text{in}}} N_{\text{out}} w^i - \lfloor N_{\text{out}} w^i \rfloor \right) \sum_{i=1}^{n} w^i_r - u \right\rceil \\ &= \lfloor N_{\text{out}} w^n \rfloor + \left\lceil \sum_{i=1}^{n} N_{\text{out}} w^i - \lfloor N_{\text{out}} w^i \rfloor - u \right\rceil \end{aligned}$$

such that, for $n = 1$,

$$r^1_{\text{RR}} = \lfloor N_{\text{out}} w^1 \rfloor + \lceil N_{\text{out}} w^1 - \lfloor N_{\text{out}} w^1 \rfloor - u \rceil = \lceil N_{\text{out}} w^1 - u \rceil$$

and, for $n > 1$,

$$\begin{aligned} r^n_{\text{RR}} &= \lfloor N_{\text{out}} w^n \rfloor + \left\lceil \sum_{i=1}^{n} (N_{\text{out}} w^i - \lfloor N_{\text{out}} w^i \rfloor) - u \right\rceil - \left\lceil \sum_{i=1}^{n-1} (N_{\text{out}} w^i - \lfloor N_{\text{out}} w^i \rfloor) - u \right\rceil \\ &= \lfloor N_{\text{out}} w^n \rfloor + \left\lceil \sum_{i=1}^{n} N_{\text{out}} w^i - u \right\rceil - \sum_{i=1}^{n} \lfloor N_{\text{out}} w^i \rfloor - \left\lceil \sum_{i=1}^{n-1} N_{\text{out}} w^i - u \right\rceil + \sum_{i=1}^{n-1} \lfloor N_{\text{out}} w^i \rfloor \\ &= \left\lceil N_{\text{out}} \sum_{i=1}^{n} w^i - u \right\rceil - \left\lceil N_{\text{out}} \sum_{i=1}^{n-1} w^i - u \right\rceil. \end{aligned}$$

Thus, the cumulative sum of replication factor obtained with residual resampling is

$$c^n_{\text{RR}} = \lceil N_{\text{out}} w^1 - u \rceil + \sum_{i=2}^{n} \left(\left\lceil N_{\text{out}} \sum_{j=1}^{i} w^j - u \right\rceil - \left\lceil N_{\text{out}} \sum_{j=1}^{i-1} w^j - u \right\rceil \right) = \left\lceil N_{\text{out}} \sum_{i=1}^{n} w^i - u \right\rceil$$

for $n > 1$. The first replication factor, r^1, and the cumulative sum of replication factors, c^n, for $n > 1$ are identical for the RSR and RR algorithms, which implies that the vectors of replication factors are identical. □

Conversion of vector of replication factors to index vector

Algorithm 5 shows how to convert a vector of replication factors, r, to an index vector, i. This scheme can easily be vectorized.

Algorithm 5. Conversion algorithm.

function CONVERT(r, N_{in}, N_{out})
  $b^1 = \cdots = b^{N_{\text{out}}} = 0$
  $k = 1$
  for $n = 1$ to N_{in} **do**
    if $r^n > 0$ **then**
      $d^k = n$
      if $k = 1$ **then**
        $a^k = r^n$
      else
        $a^k = a^{k-1} + r^n$
      end
      $b^{a^k - r^n + 1} = 1$
      $k = k + 1$
    end
  end
  $c^1 = b^1$
  $i^1 = d^{c^1}$
  for $n = 2$ to N_{out} **do**
    $c^n = c^{n-1} + b^n$
    $i^n = d^{c^n}$
  end
  return i
end