CLAREMONT McKENNA COLLEGE

PERFECT SAMPLING OF VERVAAT PERPETUITIES

SUBMITTED TO

PROFESSOR MARK HUBER

AND

DEAN GREGORY HESS

BY

ROBERT TRISTAN WILLIAMS

FOR

SENIOR THESIS

SPRING 2013

APRIL 29TH, 2013

Perfect Sampling of Vervaat Perpetuities

Robert Tristan Williams

Abstract

This paper focuses on the issue of sampling directly from the stationary distribution of Vervaat perpetuities. It improves upon an algorithm for perfect sampling first presented by Fill & Huber [1] by implementing both a faster multigamma coupler and a moving value of X_{max} to increase the chance of unification. For $\beta = 1$ we are able to reduce the expected steps for a sample by 22%, and at just $\beta = 3$ we lower the expected time by over 80%. These improvements allow us to sample in reasonable time from perpetuities with much higher values of β than was previously possible.

Contents

1 Introduction

1.1 Vervaat Perpetuities

We study here the Vervaat family of perpetuities, so named for the author that described them [3]. Here, a perpetuity is the sum of a series of products of random variables, of the form

$$Y = W_1 + W_1W_2 + W_1W_2W_3 + \ldots, \tag{1}$$

where $W_1, W_2, \ldots$ and Y are independent. We know that these variables are non-negative, so $w \geq 0$, and $\mathbb{E}(w) \leq 1$. Then, because these W's are i.i.d. and have expected value less than one, we know the expected value of the sum follows the formula of a geometric series, so

$$\mathbb{E}(Y) = \mathbb{E}(W_1 + W_1W_2 + W_1W_2W_3 + \ldots) = \mathbb{E}(W)/(1 - \mathbb{E}(W)) < \infty$$

and so through multiplying and simplifying we find the equation characterizing the distribution of Y,

$$Y = W(1 + Y),$$

where W and Y are again independent. Then, to obtain the Vervaat perpetuities, of which there is one for each value of $0 < \beta < \infty$, we substitute $W = U^{1/\beta}$ to obtain

$$Y = U^{1/\beta}(1 + Y), \tag{2}$$

which we will use for our Markov model in the remainder of the paper.

1.2 Perfect Sampling

In perfect sampling, our goal is to take a random draw from the stationary distribution of a Markov chain. Unlike traditional Markov chain Monte Carlo methods, these samples will come exactly from the stationary distribution and thus be free of statistical error. Moreover, they eliminate the need for knowing the mixing time of the chain. In this paper, we seek to draw a variable that characterizes perpetuities of the Vervaat family, as seen above.

2 A Perfect Sampling Algorithm

2.1 Coupling From The Past (CFTP)

Perpetuities can be simulated as Markov chains with a continuous state space. If we can draw from the stationary distribution of these Markov chains, then

we know that we have a perfect sample.

Simulating a Markov chain forward a large number of steps to take stationary draws is time consuming, as it takes an indefinite amount of steps for one draw, and imprecise without knowledge of the chain's mixing time. Instead, using a process called coupling from the past (CFTP), we can generate perfect samples from our stationary distribution in potentially a *single* step. For $\beta = 1$, the expected time for a perfect sample is only at most 6 steps, and we can do this with a relatively simple program.
The basic format of coupling from the past is as follows:

1. Start with a minimum element X_{min} and a maximum element X_{max}
2. Generate t uniforms, $U_1, \ldots, U_t \overset{iid}{\sim}$Unif([0,1]).
3. Run the chain forward t steps using $U_1, \ldots, U_t$.
4. If $X_{max}^{(t)} = X_{min}^{(t)}$, stop algorithm and output $X_{min}^{(t)} \sim \pi$. Else, call CFTP(2t) to find X_0.
5. Run X_0 forward using $U_1, \ldots, U_t$ to get X_t
6. Output $X_t \sim \pi$.

To use CFTP, we require a monotone update function ϕ such that for $x \preceq y$ on a partially ordered state space, we have $\phi(x, w) \preceq \phi(y, w)$ for all $w \in \Omega$. Because of the monotonicity of this update function, we know that for any two values x and y with $x \prec y$, if after t steps we have $x = y$, then all numbers between x and y have been "sandwiched" by the two and must also equal that value.

In coupling from the past with our given function, we face two problems. The first is that our update function is strictly monotonic, meaning two different values will never converge. The second is that our state space has no "top" element, so X_{max} does not have a finite value at time = -t. X_{min}, on the other hand, can be set to zero since the positivity of our random variables means the perpetuity will never be negative.

2.2 Coupling Into and From The Past (CIAFTP)

We accommodate for the lack of a top element through use of a dominating chain in a modification of traditional CFTP that Wilson [4] called coupling

into and from the past (CIAFTP). The underlying principle of the dominating chain, developed by Kendall [2], is that if we can establish a process D_i that is greater than or equal to X at every timestep, then D_{-t} provides an upper bound on X and we can use its value as the initial X_{max}. If X_{min} couples with this value, then we know that it would have coupled with all possible values of X and thus provides a perfect draw.

Since the Markov chain for our perpetuity is stationary by time zero, we can draw D_0 directly from the process' stationary distribution. We then simulate the chain back in time t steps, accomplishing the "into the past" section of the coupling algorithm. X_{-t} then moves forward with the values of W_i conditional on the behavior of the dominating chain as that is predetermined. At this point, it proceeds like traditional CFTP.

2.3 The Dominating Chain

For our perpetuity function, a simple random walk on the integers with negative drift and a partially reflecting lower boundary suffices for a dominating chain. At each step, the chain with increases with probability 1/3 and decreases with probability 2/3, unless it is currently at the value $x_0 - 1$, in which case it stays in place. Here, $x_0 := \lceil 2/(1-(2/3)^{1/\beta})\rceil - 1 \geq 2$. Since $\beta \geq 0$ and $W \sim$ Unif([0,1]), we know that $\mathbb{E}(W^{1/\beta}) = \beta/(1+\beta) \leq 1$ and so for practical β the value of $\mathbb{E}(X_{-i+1}|X_{-i}) \leq X_{-i}\ \forall\ i$, making the process a supermartingale. Because the dominating chain cannot go below $x_0 - 1$ and the process is expected to decrease in value, it will remain above all values of X_{-i} for downward moves in the process. Since the maximum value of W is 1, note that $\max(X_{i+1}|X_i) = 1 \cdot (1 + X_i) = 1 + X_i$ and so any increase in the process can be no greater than 1. This is the increasing move for the dominating chain, and so D_i still remains above X_i.

To implement our dominating chain, we need to first draw a value of D_0. The partially reflecting lower boundary and negative drift mean that we can generate a stationary state as the boundary value plus a geometric variable with rate 1/2. With $G \sim$ Geom(1/2), $G \in 0, 1, 2, ...0$ we have

$$D_0 = x_0 - 1 + G. \tag{3}$$

Now, having generated D_0, we need to find D_{-t}. This Markov chain is time-reversible, and so moves backwards the same way it moves forward. Thus, we generate D_{-1} by moving up from D_0 with probability 1/3 and down from D_0 with probability 2/3, unless we are at the boundary in which case $D_{-1} = D_0$.

Moving incrementally with this process gives us D_{-t}, which we now set as $X_{max}^{(-t)}$.

2.4 The Multigamma Coupler

The second difficulty perpetuities present us with is that whereas a Markov chain with finite values could couple under normal conditions, our simulation of Vervaat perpetuities has a continuous state space. Given the monotonicity of ϕ, the update function, strictly inequal variables would theoretically never meet. While running the algorithm in a programming language like R without correcting this can still produce "coupled" results, that is only because of imprecision in the computer's number storage. The typical approach to fixing this is to use a gamma coupler, outlined in Wilson's paper [4]. However, our goal is to couple not only X_{max} and X_{min}, but all values of X in between, and so a standard gamma coupler will not suffice. Instead, we have a "multi"gamma coupler that takes in the entire space of values.

For our multigamma coupler, we use the function

$$\phi(x, W, V) := \mathbf{1}\left(W \leq \frac{1}{1+x}\right) V + \mathbf{1}\left(W \geq \frac{1}{1+x}\right) W(1+x) \qquad (4)$$

where W is the random variable imputed from the dominating chain and V is an independently generated random variable with $V \sim \text{Unif}([0,1])^{1/\beta}$. If X_{max} is ever less than $1/(1+X_{max})$, all X values couple and assume the value of V, proportionally distributed from 0 to 1. If not, then each value of X assumes the normal perpetuity value, $W(1+X)$.

It should be noted that unlike Fill and Huber's algorithm, our coupler uses values of $X_{max}^{(-i)}$ instead of D_{-i}. Conceptually, this is no different, but as $X_{max}^{(-i)}$ must be less than or equal to D_{-i} at any step, we have $1/(1+X_{max}^{(-i)}) \geq 1/(1+D_{-i})$ and so therefore a greater if not equal chance of coupling in any step. When the time increment is only one step, however, this will be a strict equality as the values will not be able to diverge in any function call.

2.5 The Formal Algorithm

Combining the previous sections, the original algorithm is as follows:

1. With $x_0 := \lceil 2/(1-(2/3)^{1/\beta}) \rceil - 1 \geq 2$, generate $G \sim \text{Geom}(1/2)$ and set $D_0 \leftarrow x_0 - 1 + G$.

2. For t' steps, having D_{-i+1}, we generate D_{-i} by moving up 1 from D_{-i+1} with probability 1/3 and down from D_{-i+1} with probability 2/3 unless we are at $x_0 - 1$, in which case we remain at $x_0 - 1$.

3. Reading the moves forward from D_{-i} to D_{-i+1}, impute $U_{-i} \sim$ Unif(2/3,1) if $D_{-i} \leq D_{-i+1}$ and $U_{-i} \sim$ Unif(0,2/3] if $D_{-i} \geq D_{-i+1}$.
Set $W_{-i} \leftarrow U_{-i}^{1/\beta}$, $X_{max} \leftarrow D_{-i}$, and $X_{min} \leftarrow 0$.

4. If $W_{-i} \leq 1/(1+X_{max}^{(-i)})$, generate $V_{-i} \sim$Unif$([0,1])^{1/\beta}$ and set $X_{max}^{(-i+1)}, X_{min}^{(-i+1)} = V_{-i}$. Else, let $X_0^{(-i+1)} = W_{-i} \cdot (1 + X_0^{(-i+1)})$.

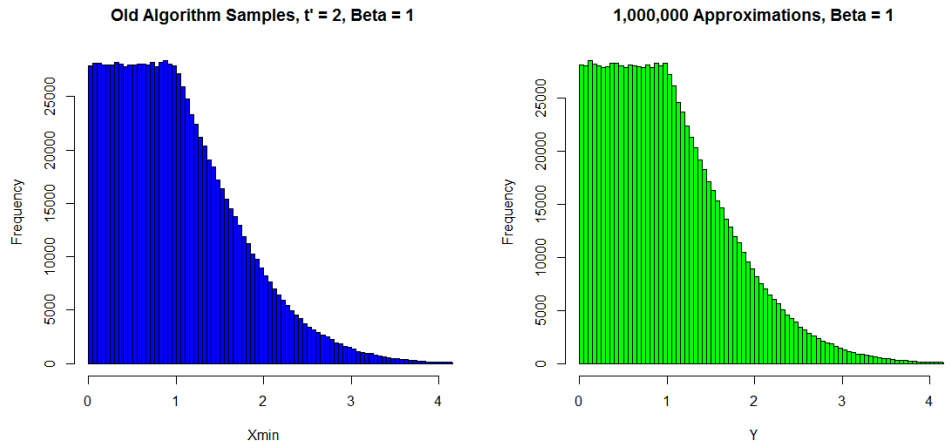
5. If X_{min} and X_{max} have not coupled by $t = 0$, call X_0 as CFTP(t). Else, return $X_{min} \sim \pi$

6. Move X_0 forward according to update function ϕ.

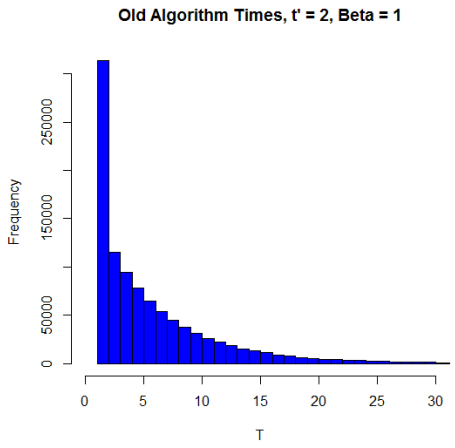
7. Return $X_0 \sim \pi$.

3 Results

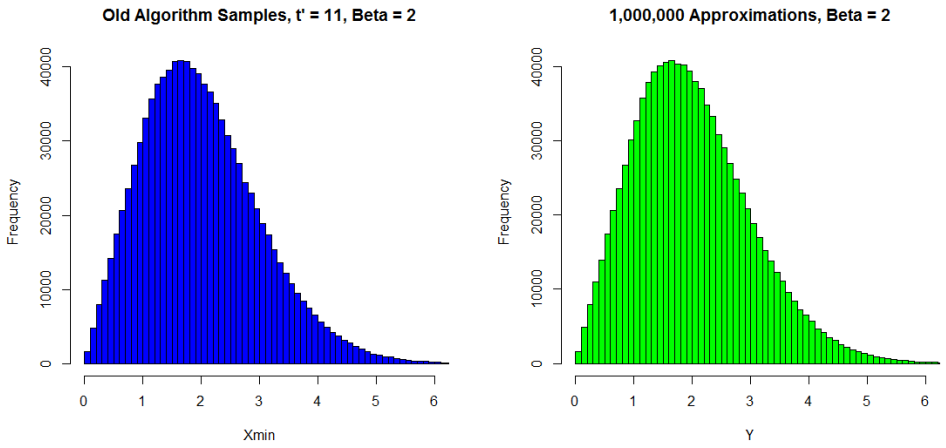
As seen in the paper by Fill and Huber [1], the expected runtime for $\beta = 1$ is just about 6 steps. A proof of this upper bound can be seen in the respective paper. Below are graphs displaying the corresponding values of X for different β's to demonstrate the expected range of perpetuity values from perfect sampling. For β values of 1, 2, and 3, we are able to find different optimal increment sizes for the process runtime. Concurrent with Fill and Huber's paper, a million samples of $t' = 1, \beta = 1$ returns an expected coupling time of 6.082 steps. However, the algorithm actually produces a lower $\mathbb{E}(T)$ with a time increment of 2, equal to 5.419 steps. The samples, compared with the approximations for $\beta = 1$, are indistinguishable.



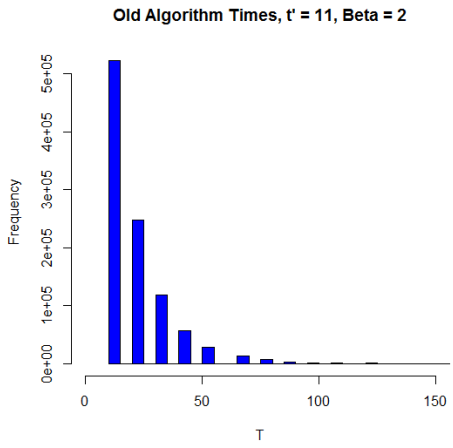
A histogram of the runtime for $\beta = 1, t' = 2$ is below. We can see that far more samples couple in the first two steps than anywhere else, but that some draws can take up to thirty steps to return an answer.



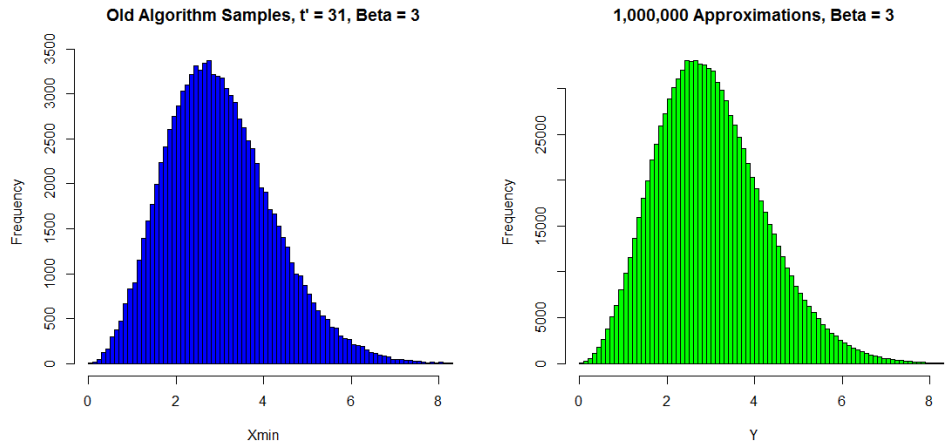
For β values of 2 and 3, we can determine their optimal runtime and t' values experimentally. For $\beta = 2$, the algorithm runs best with a t' of 11, returning 1,000,000 samples in just under 22 minutes with an expected T of 21.118 steps. The approximate solutions are displayed adjacently to confirm accuracy.



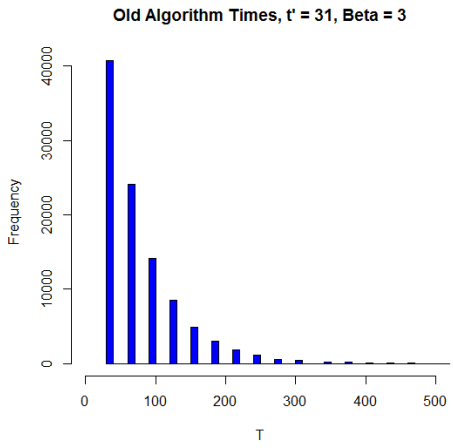
The histogram of runtimes shows that again most solutions come in one or two iterations but that rare events take upwards of 120 steps to couple and return a sample.



For $\beta = 3$, it is not as clear which is the optimal time increment, but we find very similar runtime values for t' between 30 and 32. With 100,000 samples (1,000,000 is too computationally intensive for $\beta \geq 2$), it appears that 31 is optimal, returning an $\mathbb{E}(T)$ of 76.316 steps.



A histogram of the runtime for $\beta = 3, t' = 31$ is below. Now, it appears the majority of solutions return in 4 or less iterations, but there are some that push well beyond 500 steps, greatly raising the runtime of the algorithm.



At this rate, it would be highly impractical to study β greater than at most 5, so we must consider changes that reduce this runtime.

4 An Improvement to the Algorithm

We have two improvements, but only the latter makes a substantial difference. Our first is to use an initial value of X_{max} equal to the value of D_{-t} since we know that this is the maximum value of the Markov chain at the initial position for coupling from the past. This was detailed in the previous section. As mentioned, at a time increment of one this difference will

be trivial. Furthermore, this change has been made in the old and the new algorithm, so any improvements are encompassed in both.

The primary change is an improvement to the coupler that should provably reduce the time it takes to couple. Now, the X_{min} and X_{max} values come together at timestep $i+1$ if $W_{-i} \leq (1+X_{min})/(1+X_{max})$ instead of $1/(1+X_{max})$. Therefore, our new coupler is

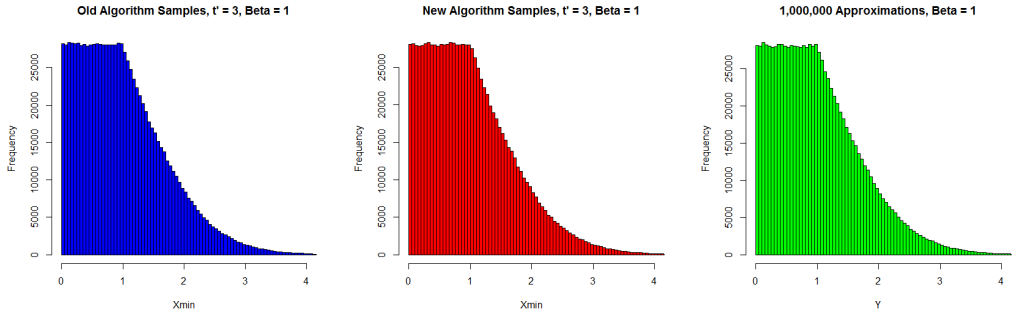
$$\phi(x, W, V) := \mathbf{1}\left(W \leq \frac{1+X_{min}}{1+X_{max}}\right) W(1+X_{max}) + \mathbf{1}\left(W \geq \frac{1+X_{min}}{1+X_{max}}\right) V, \tag{5}$$

and we now explore the effects of this change.

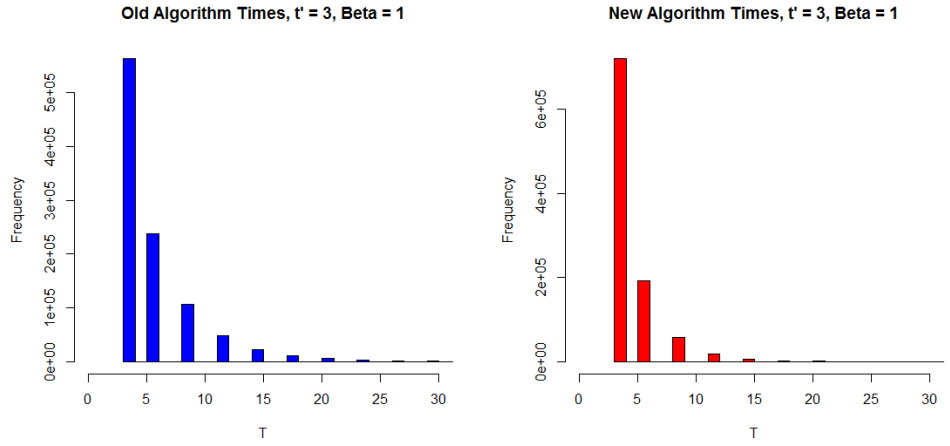
5 Improvements in Runtime

We examine the improvements by first determining experimentally the optimal time increments for both algorithms at each level of β. This process beings by taking relatively small samples of every t', starting at 1, to determine a relative lower bound on expected sampling time. These values decrease at first, because the algorithms need time for X_{min} and X_{max} to develop, but eventually bottom out and begin increasing. Once an algorithm has hit a lower bound of, say, k, it is unreasonable to check the algorithm's runtime for values of t higher than k. This is because, even in the best case, the lowest possible runtime is $t \geq k$ so the average could not possibly be lower. Moreover, the reason these times begin increasing before $t' = k$ is because, after a point, if the values do not couple in the first pass, the second one automatically puts them over the expected time. This effect is more prevalent in higher values of β. We now look at the differences in β values of 1, 2, and 3.

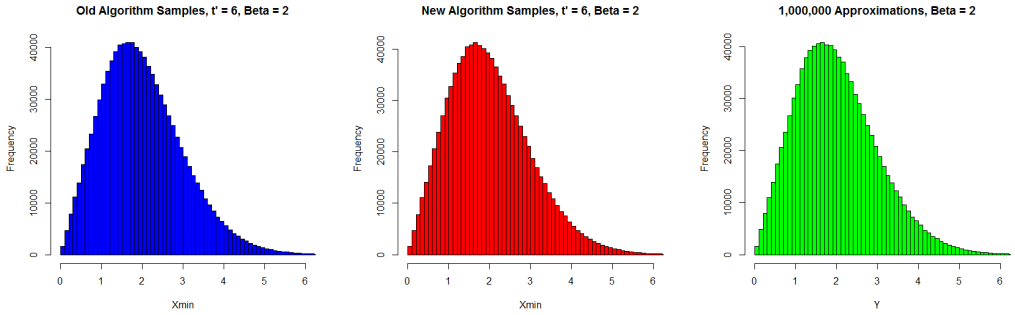
With a time increment of one, we see only trivial changes in runtime. However, at $t' = 2$, the improvements are clearly evident. For $\beta = 1$, the new algorithm runs optimally at $t' = 3$, where $\mathbb{E}(T) = 4.231$ steps, compared to the old algorithm's 5.441. Shown below are graphs of the samples from the old, new, and approximating algorithm, in blue, red, and green, respectively, to demonstrate their accuracy.



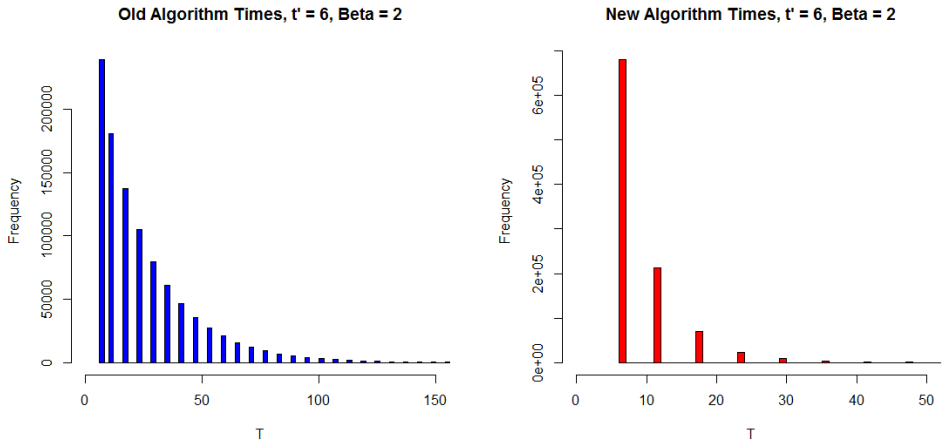
Their results are almost identical. However, when we look at their sampling times for $t' = 3$, plotted below, we notice a marked difference. Over 150,000 more samples finish in 3 steps with the new algorithm than the old, and we see 3 times as many finishing in 3 instead of 6 with the new algorithm, whereas in the old one this number was roughly 2.



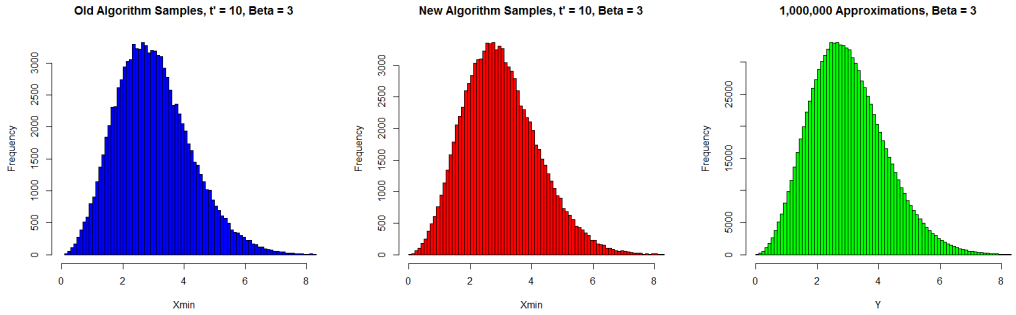
The differences become more apparent at $\beta = 2$. While the old algorithm ran optimally at $t' = 11$, with an expected sample time of 21.118 steps, the new algorithm runs best at $t' = 6$, where the mean sample time is only 8.9 steps. At this same increment, the old algorithm has a mean of 25.267 steps. The histograms of their sample values at $t' = 6$ are shown below, again next to the approximation graph.



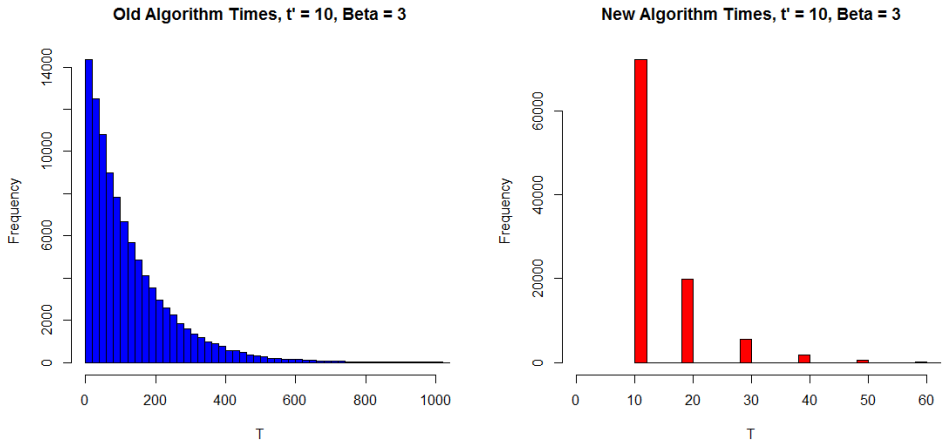
The samples are again indistinguishable. Their sample times, however, now have a sharp dichotomy. The older algorithm does not trail off in runtime until about 150 steps, meaning at least a sizeable portion of results take over 100 steps to finish. The new algorithm, meanwhile, does not go beyond 50. Indeed, the x-axis in the second graph is truncated because those longer samples made up such a small portion of results that their bars were unnoticeable.



At $\beta = 3$, the differences become even more apparent. The old algorithm runs optimally at about $t' = 31$ with an average 76.316 steps, while the new algorithm now runs best at an increment of 10 and a mere 13.899 expected time steps. At this increment, the former model takes on average 132.014 steps, almost ten times as many. Once more, in the graphs below, both algorithms return what look to be perfect answers.



Their graphs of sampling time, now, are completely different. The new times rarely make it above 60 steps, while the old ones occasionally reach over 600. Even at its optimal increment, the average of the old algorithm is longer than the sampling time for all but 22 of 100,000 draws of the new algorithm.



While we could explore runtime for higher β's, the fact that the sampling times have gone from similar at $\beta = 1$ to barely comparable at only $\beta = 3$ suggests their disparity will increase exponentially. Moreover, the old algorithm will soon be too slow to return an answer in any reasonable amount of time. At only 1000 samples of $\beta = 5$, with an arbitrary time increment of 100, the old algorithm took 150.3 seconds to run while the new algorithm took a mere 3.6. This will most likely continue for $\beta = 10, 100, 1000$, etc. We will now look more closely at why this disparity exists.

6 Analysis of Improvements

With a single step increment for CFTP, note that this model is a basic acceptance/rejection method. X_{max} will always be D_{-t} and so each step checks

only if $W_{-t} \leq 1/(1 + X_{max})$, and if not it disregards the values and moves back another step. Furthermore, since X_{min} begins at zero, $t' = 1$ will always have $1 + X_{min} = 1$ and so the algorithms have identical numerators. By increasing the time increment, we allow both algorithms to lower X_{max}, but we especially allow the new algorithm to develop the value of X_{min} and thus raise our chances of coupling even further. With a higher chance of coupling, we will have a lower expected time to do so.

To see the effect of the change in coupler on the algorithm, consider the fixed point of the equation $X = W(1 + X)$, $X \geq 0$. Since $W = U^{(1/\beta)}$ and $U \sim$Unif([0,1]), we have $\mathbb{E}(W) = \beta/(1 + \beta)$. Thus, taking the expectation of our fixed-point equation has the form $X = (\beta/(1 + \beta))(1 + X)$, yielding:

$$\frac{X}{1 + X} = \frac{\beta}{1 + \beta}. \tag{6}$$

It is obvious from (6) that the positive fixed point of this equation is $X = \beta$. Thus, as we take more steps in a single increment, X_{min} and X_{max} both progress to β and $(1 + X_{min})/(1 + X_{max})$ goes to 1. This means that for each step in the new algorithm, the chance of coupling in the next step should grow closer to 1, and for sufficiently large t' the values should definitely couple. In the old algorithm, as X_{max} progressed towards $X_{max} = \beta$, the best the expected next-step coupling probability could be was $W \leq 1/(1 + \beta)$, much lower than 1.

6.1 Future Work

While we are able to demonstrate the improvements of this algorithm exponentially, there remains to be seen a formal bound for the expected sampling time. The paper explored the values of various β's, but with an actual proof we should be able to determine this bound without the aid of a computer. This exists for the old algorithm [1], but remains to be seen for the new one.

7 Conclusion

Using perfect sampling algorithms, we were able to draw exactly from the stationary distributions of Vervaat perpetuities. For our analysis, we considered β values of 1, 2, and 3 in the old algorithm, the improved algorithm, and an approximating algorithm that 'samples' perpetuities by generating the first 100 variables of a sequence to serve as a "key" for the others. For $\beta = 1$ or 2,

we took 1,000,000 samples, but because of computational intensity we used only 100,000 samples for $\beta = 3$. At the first β, the improved algorithm only decreased the expected sampling time by about 22%. At $\beta = 2$, however, this decrease rose to 58%, and by $\beta = 3$ was already at 82%. These reductions continue to approach 1 as β goes to infinity and so allow us to sample in reasonable time from much higher valued perpetuities than were possible with the previous algorithm. Code for implementing these algorithms in R can be seen in the appendix.

8 Acknowledgements

I have to thank Professor Huber for all of his help in finding the topic, understanding it, and getting the code to work when things were at a standstill. I would also like to thank David Wilson at Rutgers for maintaining such a resourceful website on perfect sampling and my roommate William Dodds for basic help with writing in LaTeX.

References

[1] J.A. Fill and M.L. Huber. Perfect simulation of Vervaat perpetuities. *Electronic Journal of Probability.* **15** (2010), 96-109.

[2] W.S. Kendall and J. Mller. Perfect simulation using dominating processes on ordered spaces, with application to locally stable point processes. *Adv. Appl. Probab.* **32** (2000), 844-865.

[3] W. Vervaat. On a stochastic difference equation and a representation of non-negative infinitely divisible random variables. *Adv. Appl. Prob.* **11** (1979), 750-783.

[4] D.B. Wilson. Layered multishift coupling for use in perfect sampling algorithms (with a primer on cftp). *Fields Institute Communications*, **26** (2000), 141-176.

A R Code for the Algorithms

A.1 The Old Algorithm

The following code generates a sample using the algorithm in section 2.5.

```
oldalg <- function (t,beta, D0 = NULL){
        #Takes one sample from the stationary distribution of the process
        #First generates the dominating chain
        x0 <- ceiling( 2/ (1 - (2/3)^(1/beta)) ) - 1
        if(x0<2) x0 <- 2
                D <- replicate(t,0)
        if(is.null(D0)){
                G <- rgeom(1,1/2)
                D0 <- x0 - 1 + G
                }
        D <- c(D,D0)
        U <- runif(t)
        #Imputes W variables based on the dominating chain
        W <- replicate(t,0)
        for (i in 1:t) {
                if (U[i]>(2/3)) {
                D[t+1-i] <- D[t+2-i] + 1
                        }
                else if(U[i]<=(2/3)){
                D[t+1-i] <- D[t+2-i] - (D[t+2-i]>(x0-1))
                        }
                }
        for(i in 1:t){
                if(D[i]<D[i+1]) W[i] <- runif(1,2/3,1)^(1/beta)
                else W[i] <- runif(1,0,2/3)^(1/beta)
                }
        #Draws V variables independently from the dominating chain
        V <- runif(t)^(1/beta)
        xmax <- D[1]
        coup <- 0
        #Runs forward t steps to see if xmin and xmax couple
        for(i in 1:t){
                if (W[i]<=(1/(xmax+1))){
                xmax <- V[i]
                coup <- 1
                        }
                else {
                xmax <- W[i]*(1+xmax)
                        }
                }
        if(coup==1) return(c(xmax,t))
        # If not coupled, the program goes back t more steps and repeats
        else {
```

```
# Calls the algorithm recursively to get an answer
cftp <- olderalg.altered(t,beta,D[1])
xmax <- cftp[1]
for (i in 1:t) {
        if (W[i]<=(1/(xmax+1))) {
                        xmax <- V[i]
                        }
                else {
                        xmax <- W[i]*(1+xmax)
                        }
        }
t <- t + cftp[2]
return(c(xmax,t))
        }
}
```

A.2 The New Algorithm

The following code generates a sample using the algorithm in section 4.

```
newalg <- function (t,beta, D0 = NULL){
        #Takes one sample from the stationary distribution of the process
        #First generates the dominating chain
        x0 <- ceiling(2/(1-(2/3)^(1/beta)))-1
        if(x0<2) x0 <- 2
        D <- replicate(t,0)
        if(is.null(D0)){
                        G       <- rgeom(1,1/2)
                        D0      <- x0 - 1 + G
                        }
        D <- c(D,D0)
        U <- runif(t)
        for(i in 1:t){
                        if(U[i]>(2/3)){
                                D[t+1-i] <- D[t+2-i] + 1
                                }
                        else if(U[i]<=(2/3)){
                                D[t+1-i] <- D[t+2-i] - (D[t+2-i]>(x0-1))
                        }
                }
        #Imputes W variables based on the dominating chain
        W <- replicate(t,0)
        for(i in 1:t){
                if(D[i]<D[i+1]) W[i] <- runif(1,2/3,1)^(1/beta)
                else W[i] <- runif(1,0,2/3)^(1/beta)
                }
        #Draws V variables independently from the dominating chain
        V <- runif(t)^(1/beta)
        xmin    <- 0
        xmax    <- D[1]
        coup  <- 0
        #Runs forward t steps to see if xmin and xmax couple
        for(i in 1:t){
                if(W[i]<=((1+xmin)/(xmax+1))){
                        xmin <- W[i]*(xmax+1)
                        xmax <- xmin
                        coup <- 1
                        }
                else{
                        xmin <- V[i]*(1+xmin)
                        xmax <- W[i]*(1+xmax)
                        }
                }
        if(coup==1) return(c(xmin,t))
        # If not coupled, the program goes back t more steps and repeats
        else{
```

```
        # Calls the algorithm recursively to get an answer
        cftp <- neweralg(t,beta,D[1])
        xmin <- cftp[1]
        xmax <- D[1]
        for(i in 1:t){
                if(W[i]<=((1+xmin)/(xmax+1))){
                        xmin <- W[i]*(xmax+1)
                        xmax <- xmin
                        }
                else{
                        xmin <- V[i]*(1+xmin)
                        xmax <- W[i]*(1+xmax)
                        }
                }
        t <- t + cftp[2]
        return(c(xmin,t))
        }
}
```

A.3 The Approximating Algorithm

The following code generates the approximate answers in sections 2 and 4.

```
VerPerA <- function(steps,beta){
        #simulates a perpetuity forward enough steps to
        #approximate a draw
        w <- runif(steps)^(1/beta)
        perp <- 0
        for(i in 1:steps){
                perp <- perp + prod(w[i:steps])
                }
        return(perp)
        }
```