Simulating from the Polya posterior

Glen Meeden (gmeeden@umn.edu)
Charles Geyer (geyer@umn.edu)

October 6, 2021

1 Introduction

The Polya posterior is an objective Bayesian approach to finite population sampling. In its simplest form it assumes little prior information is available about the population and that the sample, however chosen, is "representative." That is the unobserved or unseen units in the population are assumed to be similar to the observed or seen units in the sample. It is appropriate when a classical survey sampler would be willing to use simple random sampling as their sampling design. Let `ysamp` $= (y_1, \ldots, y_n)$ be the vector of observed values in the sample of size n from a population of size N.

Given the data, the Polya posterior is a predictive joint distribution for the unobserved units in the population conditioned on `ysamp`, the values in the sample. For a sample `ysamp` we now define this distribution for the unseen units. Consider two urns where the first urn contains the n observed `ysamp` values while the second urn contains the $N - n$ unsampled units. We begin by choosing one unit at random from each of the two urns. We then assign the observed y value of the unit selected from the first urn to the unit selected from the second urn and then place them both in the first urn. The urns now contain $n + 1$ and $N - n - 1$ items respectively. This process is repeated until all the units have been moved from the second urn to the first and have been assigned a value. At each step in the process all the units in the first urn have the same probability of being selected. That is, unobserved units which have been assigned a value are treated just like the ones that actually appeared in the sample. Once this is done, we have generated one realization of the complete population from the Polya posterior distribution. This simulated, completed copy contains the n observed values along with the $N - n$ simulated values for the unobserved members of the population. Hence, simple Polya sampling yields a predictive distribution for the unobserved given the observed. One can use the Polya posterior in the usual Bayesian manner to find point and interval estimates of various population parameters. In practice such estimates are usually found approximately by considering many simulated copies of the completed population.

A good reference for Polya sampling is Feller (1968). The Polya posterior

is related to the Bayesian bootstrap of Rubin (1981). See also Lo (1988). For more discussion on the Polya posterior see Ghosh and Meeden (1997).

The Polya posterior can be modified to take into account available prior information about the population. This leads to the constrained Polya posterior. More discussion about the theory underlying the constrained Polya posterior can be found in Lazar, Meeden and Nelson (2008). The discussion in this vignette will focus on using the constrained Polya posterior to simulate completed copies of the population.

2 R

```
> library(polyapost)
```

This document was processed with R, version 4.1.0, using the CRAN packages `polyapost`, version 1.7, and `rcdd`, version 1.2.2.

```
> set.seed(313)
```

3 Simulating from the Polya posterior

This package has a function, `polyap`, which given a sample `ysamp` will simulate one completed copy of the population. To begin suppose we have a sample of size 2 from a population of size 10 where the values in the sample are 0 and 1. The next bit of code simulates one completed copy of the population.

```
> ysamp<-c(0,1)
> K<-10-2
> polyap(ysamp,K)

 [1] 0 1 0 0 1 0 1 1 0 1
```

For a more realistic example we construct a population of size 500, select a random sample of 25 units, generate 10 completed copies of the population and return the 10 means of the simulated populations.

```
> y<-rgamma(500,5)
> mean(y)

[1] 5.021535

> samp<-sample(1:500,25)
> ysamp<-y[samp]
> mean(ysamp)

[1] 5.459067
```

```
> K<-500-25
> simmns<-rep(0,10)
> for(i in 1:10){simmns[i]<-mean(polyap(ysamp,K))}
> round(simmns,digits=2)

 [1] 5.65 5.64 5.02 5.34 5.94 6.02 5.28 5.96 4.76 5.01
```

When the sample size n is small compared to the population size N a well known approximation can be used to generate simulated copies of the population given the sample values. For simplicity assume the sample values `ysamp` are all distinct. For $j = 1, \ldots, n$ let p_j be the proportion of units in a complete simulated copy of the entire population which take on the jth value of `ysamp`. Then, under the Polya posterior, $p = (p_1, \ldots, p_n)$ has approximately a Dirichlet distribution with a parameter vector of all ones, i.e., it is uniform on the $n-1$ dimensional simplex, where $\sum_{j=1}^{n} p_j = 1$. So rather than using simple Polya sampling to randomly assign each unseen member of the population an observed value we can construct a completed copy of the population by generating an observation from the uniform distribution on the appropriate simplex. This second approach will be very useful when we consider the constrained Polya posterior.

4 The constrained Polya posterior

4.1 The basic idea

The Polya posterior can be modified to take into account prior information about the population. For example, suppose that attached to each unit there is an auxiliary variable, x say. Moreover suppose that the population mean of x is known and for units in the observed sample we learn both their y and x values. Then given the sample values `ysamp` and `xsamp` one should restrict the Polya posterior to only generate completed populations which satisfy the population mean constraint for x. If `xsamp` $= (x_1, \ldots, x_n)$ then when using the approximate form of the Polya posterior we should only consider the subset of the simplex where $\sum_{j=1}^{n} p_j x_j$ equals the known population mean. More generally, when using the Polya posterior to simulate completed copies of the entire population, one should restrict it to yield only completed copies which satisfy all the known constraints coming from prior information about the auxiliary variables. The appropriately constrained Polya posterior can then be used to make inferences about the population parameters of interest.

We assume that prior information about the population can be expressed through a set of linear equalities and inequalities for the probability vector p. In particular we suppose that p satisfies the following equations

$$A_1 p = b_1 \qquad A_2 p \leq b_2 \qquad A_3 p \geq b_3$$

Here the A_i's are matrices, p and the b_i's are column vectors and they have the correct dimensions so all the equations make sense. In each system of equations

the equalities or inequalities are assumed to hold componentwise. Furthermore all the components of the b_i's must be nonnegative.

To see how this can work we first construct a population using an auxiliary variable x to construct the y values.

```
> x<-sort(rgamma(500,10))
> y<-rnorm(500,20 + 2*x,3)
> cor(x,y)

[1] 0.89672

> mean(y)

[1] 39.86569

> mnx<-mean(x)
> mnx

[1] 9.892615
```

We divide the population into two strata, the first 250 units and the remaining 250 units. Our sampling plan will select 8 units at random from the first stratum and 17 from the second. In addition we assume prior knowledge indicates that the population mean of x lies between 9.7 and 10.0. The next chuck of code selects the stratified random sample and constructs the A_i matrices and the the b_i vectors which incorporate the prior information.

```
> samp<-sort(c(sample(1:250,8),sample(251:500,17)))
> ysamp<-y[samp]
> xsamp<-x[samp]
> mean(ysamp)

[1] 41.67035

> mean(xsamp)

[1] 11.05348

> A1<-rbind(rep(1,25),c(rep(1,8),rep(0,17)))
> b1<-c(1,0.5)
> A2<-rbind(xsamp,-diag(25))
> b2<-c(10.0,rep(0,25))
> A3<-matrix(xsamp,1,25)
> b3<-9.7
```

Note that the first row of A_1 represents the fact that the p_i's, the weights or probabilities assigned to the units in the sample, must sum to one. The second row represents the fact that the sum of the first eight must be 0.5 since their stratum is half the population. The last 25 rows of A_2 guarantee that all the

p_i's are greater than zero. Its first row represents the prior information that the population mean of x is less than 10. The single row of A_3 represents the prior information that the population mean of x is greater than 9.7.

The next step is to find a probability distribution on the sample units which satisfies all of the constraints. The function `feasible` will do this. It uses a simplex algorithm in `R` to a find solution. The function `feasible` is a function of the A_i's, the b_i's and a positive real number `eps` which is close to zero. This is a lower bound which all the p_i's in the solution must satisfy since our method will not work if any of the p_i's are zero. One must be careful to not choose a value of `eps` which is too large. We let `initsol` denote the solution found by `feasible`. The next bit of code finds `initsol` for our example. In the solution all but the third, ninth and twenty-fifth have the value 0.001. These remaining values are given just below.

```
> eps<-0.001
> initsol<-feasible(A1,A2,A3,b1,b2,b3,eps)
> initsol[c(3,9,25)]

[1] 0.001000 0.372716 0.112284
```

We are ready to generate completed copies of the population and calculate their respective means. This will be done using the function `constrppmn`. Starting at the initial distribution, `initsol`, the function `constrppmn` uses the Metropolis-Hastings algorithm to generate a Markov chain of `reps` dependent observations from the subset of the simplex defined by the constraints. The Markov chain generated in this way converges in distribution to the uniform distribution over the subset. If we wish to approximate the expected value of some function defined on the subset under the uniform distribution then the average of the function computed at the simulated values converges to its actual value. For example, if we were estimating the population mean then for each simulated probability vector p we would compute the p expectation of `ysamp`. This allows one to find point estimates of population parameters approximately.

The function `constrppmn` returns a list of 3 objects. To call it you also must specify `reps` and `burnin`. For now we will only be concerned with the first item in the list which is just the simulated population means created after the `burnin`. That is, it ignores the first `burnin` $-$ 1 simulated population means and returns the remaining `reps` $-$ `burnin` $+$ 1 simulated population means. In this case we are keeping all of the simulated means because `burnin` $= 1$. We generated a total of `reps` $= 200,001$ means. The average of these 200,001 is given in the output. The plot of the first and each successive two hundredth simulated population mean is given in Figure 1. From the plot it appears that the chain is mixing well and standard diagnostics indicate that this is so.

```
> burnin<-1
> reps<-200001
> out<-constrppmn(A1,A2,A3,b1,b2,b3,initsol,reps,ysamp,burnin)
> mean(out[[1]])
```

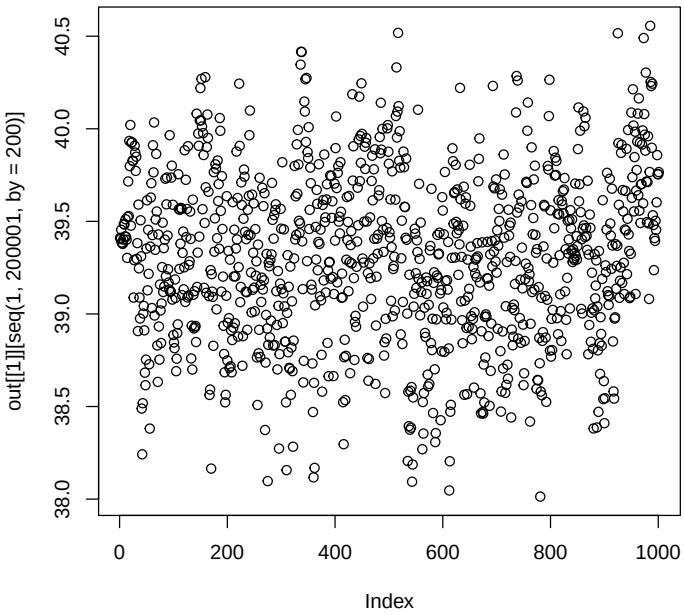

Figure 1: Plot of a subsequence of 1,000 simulated population means where every two hundredth was taken from a dependent sequence of length 200,001. The constrained Polya posterior uses the strata information and assumes the population mean of the dependent variable x lies between 9.7 and 10.0.

```
[1] 39.31708
```

The second component of the output from the function `constrppmn` is the mean of the first component, i.e. the Polya estimate of the population mean. The third component is the 2.5th and 97.5th quantiles of the first component, i.e. an approximate 95 percent confidence interval of the population mean.

```
> out[[2]]

[1] 39.31708

> out[[3]]

    2.5%    97.5%
38.40241 40.18371
```

Qu, Meeden and Zhang (2015) and Strief and Meeden (2014) discuss two situations where the constrained Polya posterior yields good inference procedures.

In the first, they demonstrate that for some small area estimation problems the constrained Polya posterior yields more robust procedures than standard methods. In the second, they demonstrate that for each unit in a sample the constrained Polya posterior can be used to find a weight. This weight can be given the usual interpretation as the number of units in the population the sampled unit represents. These weights can then be used to find good estimates of population parameters.

4.2 Some computing issues

4.2.1 How long to run the chain

The Markov chain generated by `constrppmn` converges in distribution to the uniform distribution over the polytope. The convergence result of such mixing algorithms was proven by Smith (1984). If we wish to approximate the expected value of some function defined on the polytope then the average of the function computed at the simulated values converges to its actual value. This allows one to compute point estimates of population parameters. Finding the 0.95 Bayesian credible interval approximately is more difficult.

One possibility is to run the chain for a long time; for example, we may generate 4.1 million values, throw away the first 100,000 values, and find the 0.025 and 0.975 quantiles of the remaining values. These two numbers will form our approximate 0.95 credible interval. For sample sizes of less than 100 we have found that chains of a few million suffice.

How fast a chain mixes can depend on the constraints and the parameter being estimated. It seems to take longer to get good mixing when estimating the median rather than the mean. Another approach which can work well is to run the chain for a long time and then just use every mth point where m is a large integer. Although this is inefficient it can give good answers when finding a 0.95 credible interval for the median.

To get a sense of how fast these chains can mix we run the following comparison. Although it would be silly, we could use the function `constrppmn` to generated dependent samples from the unconstrained Polya posterior. The following bit of code does that and a plot of a subsequence of the simulate population means are given in Figure 2.

```
> A1<-rbind(rep(1,25))
> A2<--diag(25)
> b1<-1
> b2<-rep(0,25)
> initsol<-rep(0.04,25)
> reps<-1000001
> out<-constrppmn(A1,A2,NULL,b1,b2,NULL,initsol,reps,ysamp,burnin)
> mean(out[[1]])

[1] 41.68159
```

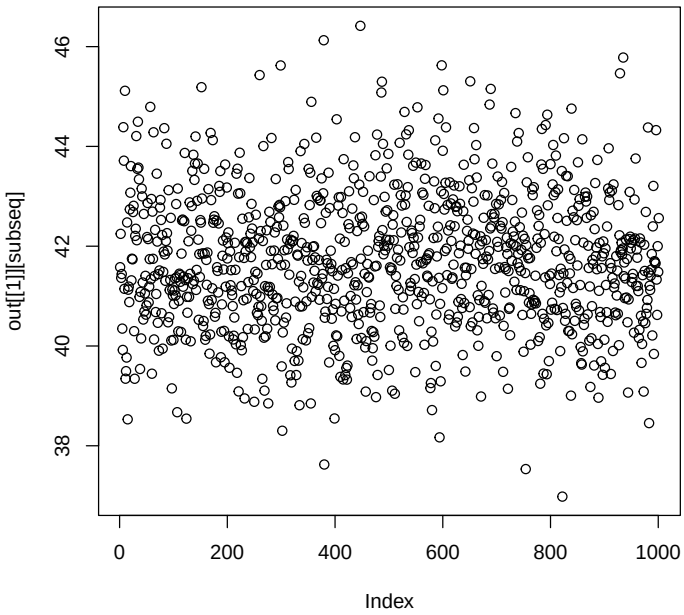

Figure 2: Plot of a subsequence of 1,000 simulated population means where every thousandth was taken from a dependent sequence of length 1,000,001. In this case the constrained polytope was the entire simplex.

```
> subseq<-seq(1,1000001,by=1000)
> mean(out[[1]][subseq])

[1] 41.6968

> sqrt(var(out[[1]][subseq]))

[1] 1.370147
```

We can now compare this sequence of 1,000 dependent simulated population means to a sequence of 1,000 truly independent copies generated by `polyap`. We see from the plots in Figures 2 and 3 that the subsequence of dependent means look very much like the sequence of independent means.

```
> K<-500-25
> simmns<-rep(0,1000)
> for(i in 1:1000){
```

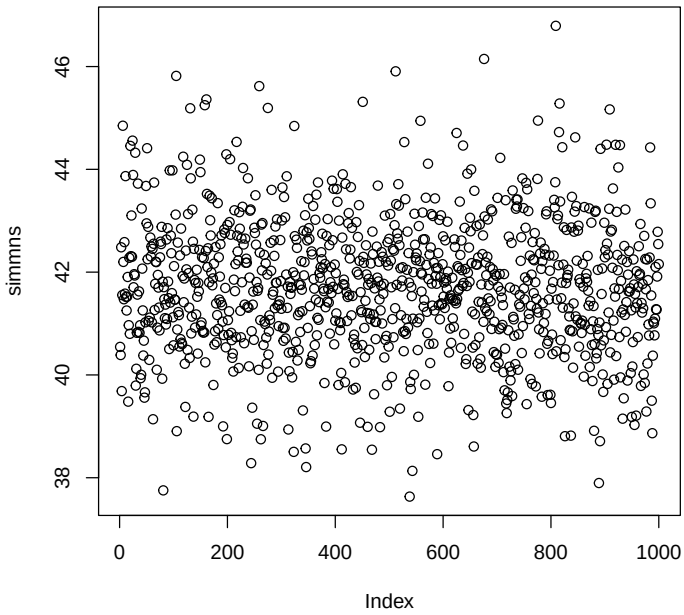

Figure 3: Plot of 1,000 independent simulated population means generated from the uniform distribution over the simplex.

```
+         simmns[i]<-mean(polyap(ysamp,K))
+ }
> mean(simmns)

[1] 41.69104

> sqrt(var(simmns))

[1] 1.311089
```

4.2.2 Estimating parameters other than the mean

The function `constrppmn` is setup for estimating a population mean. Often one is interested in estimating other population quantities, like the median. In order to do this one needs to have observations from the polytope of the entire probability distribution. With these one can compute the parameter of interest from the simulated populations and find point and interval estimates.

The function `constrppprob` returns a subsequence of dependent observations from the from the polytope. A simple example is given just below.

```
> A1<-rbind(rep(1,6),1:6)
> A2<-rbind(c(2,5,7,1,10,8),diag(-1,6))
> A3<-matrix(c(1,1,1,0,0,0),1,6)
> b1<-c(1,3.5)
> b2<-c(6,rep(0,6))
> b3<-0.45
> initsol<-rep(1/6,6)
> out<-constrpppprob(A1,A2,A3,b1,b2,b3,initsol,2000,5)
> round(out,digits=5)
```

```
           [,1]    [,2]    [,3]    [,4]    [,5]    [,6]
initsol 0.16037 0.19878 0.13807 0.19067 0.10750 0.20461
initsol 0.05095 0.23939 0.17355 0.38326 0.00047 0.15237
initsol 0.09005 0.20213 0.15903 0.29465 0.17485 0.07929
initsol 0.22338 0.02755 0.28794 0.16770 0.07367 0.21976
initsol 0.10953 0.20315 0.20953 0.21140 0.08833 0.17805
```

4.2.3 The weighted Polya posterior

In the simple Polya posterior each member of the sample is treated the same and assigned a weight of one. In some situations it is sensible to assign different weighs to the units in the sample. A detailed discussion of the weighted Polya posterior can be found in Meeden (1999). The process proceeds in much the same way as before. Consider an urn containing a finite set of n sampled values selected from a population of size N. In addition associated with each sampled unit is a positive weight. An item is selected at random from the urn with probability proportional to its weight. Then it is returned to the urn and its weight is increased by one. The process is repeated on the adjusted urn. We continue until the total weight in the urn has been increased by $N - n$. The original composition of the urn along with the $N - n$ selected values, in order, are returned. A simple example follows.

```
> ysamp<-c(1,2,3)
> wts<-c(1,2,3)
> wtpolyap(ysamp,wts,25)
```

```
 [1] 1 2 3 2 3 3 3 2 3 3 2 2 2 3 3 3 3 3 3 3 3 2 3 3 3 3 2 2
```

5 An example using the hitrun function

A limitation of the function `constrppprob` is that one cannot add any constraints. In practice one would like to be able to find approximately the expectation of the components of a constrained Dirichlet distribution. The `hitrun` function handles this more general problem.

For a survey sampling example suppose we have a population that is divided into two strata. For $h = 1, 2$ let N_h be the number of units that belong to stratum h. Suppose a random sample of size n_h is taken from stratum h and n_h/N_h is small. In addition suppose that each unit in the population belongs to a class, or poststratum that may cut across the design strata. Let N_c denote the number of units that belongs to the cth class. If for each h and c we knew the number of units that belonged to class c in stratum h then this is just the standard poststratification problem with a textbook solution. When this is not the case the standard approach is to adjust the sampling weights by using the information contained in the N_c's. For our example we assume that the number of classes is three.

Let p_{ch} be the proportion of the units in the population that belong to class c in stratum h. Our goal is to estimate the p_{ch}'s using the prior information about the class sizes and strata sizes. Let N be the total size of the population. For each class c we know that

$$p_{c1} + p_{c2} = N_c/N$$

while for each stratum h we have

$$p_{1h} + p_{2h} + p_{3h} = N_h/N$$

where

$$p_{11} + p_{21} + p_{31} + p_{12} + p_{22} + p_{32} = 1$$

There is no need to include the last constraint since that is done automatically in the `hitrun` function. For the class constraints we will only include two of them since the third is redundant. For the same reason we will only include one constraint for the strata sizes.

We set the class sizes to be 5,000, 3,000 and 2,000 while the stratum sizes are 6,000 and 4,000. The vector

```
> smp<-c(20,10,10,25,15,20)
```

gives the observed number of units in each class and stratum combination for a sample of size 40 from the first stratum and of size 60 from the second. Since the sample size is small compared to the population size and the sampling design is simple random sampling without replacement within the strata, the theory underlying the Polya posterior justifies taking as the posterior the Dirichlet distribution with parameter `smp` restricted to the polytope defined by the constraints.

The next bit of code shows how the function `hitrun` can find the posterior expectation of the p vector given the sample and the constraints.

```
>  mxcst<-rbind(c(1,0,0,1,0,0),c(0,1,0,0,1,0),c(1,1,1,0,0,0))
>  mncst<-c("5000/10000","3000/10000","6000/10000")
>  out<-hitrun(smp, a2=mxcst, b2=mncst, nbatch=20, blen=1000)
```

Note that `a1` and `b1` are omitted because there are no inequality constraints in this example.

Also note that `hitrun` requires no initial distribution. It figures out a point in the relative interior of the constraint set to start at all by itself.

The reason why `mncst` is given as character strings is so `hitrun` will use infinite-precision rational arithmetic to calculate the constraint set, thus assuring no errors due to the inexactness of ordinary computer arithmetic. The reason why `mxcst` did not need the same trick is that it is integer-valued and integers are exact in ordinary computer arithmetic. The reason why 3000/10000 is not a round number to computers is that they use binary arithmetic rather than decimal as shown by

```
> foo <- d2q(3000/10000)
> bar <- q2q("3000/10000")
> baz <- qmq(foo, bar)
> foo # three tenths decimal converted to binary converted to rational

[1] "5404319552844595/18014398509481984"

> bar # three tenths rational

[1] "3/10"

> baz # the difference

[1] "-1/90071992547409920"
```

The difference is not large, but such differences can change not just the position of vertices of the constraint polytope but the actual number of vertices.

The (Monte Carlo approximations of) posterior means are

```
> round(colMeans(out$batch), digits=3)

[1] 0.322 0.186 0.092 0.178 0.114 0.108
```

and the Monte Carlo standard errors are

```
> round(apply(out$batch, 2, sd) / sqrt(out$nbatch), digits=3)

[1] 0.000 0.001 0.000 0.000 0.001 0.000
```

Thus we see that the posterior means have almost the reported three-significant-figure accuracy, so long as the batch means have no serial correlation, which is shown by Figure 4, which is made by the following code.

```
> i <- 1
> acf(out$batch[,i], main=paste("Batch Means for Component", i))
```

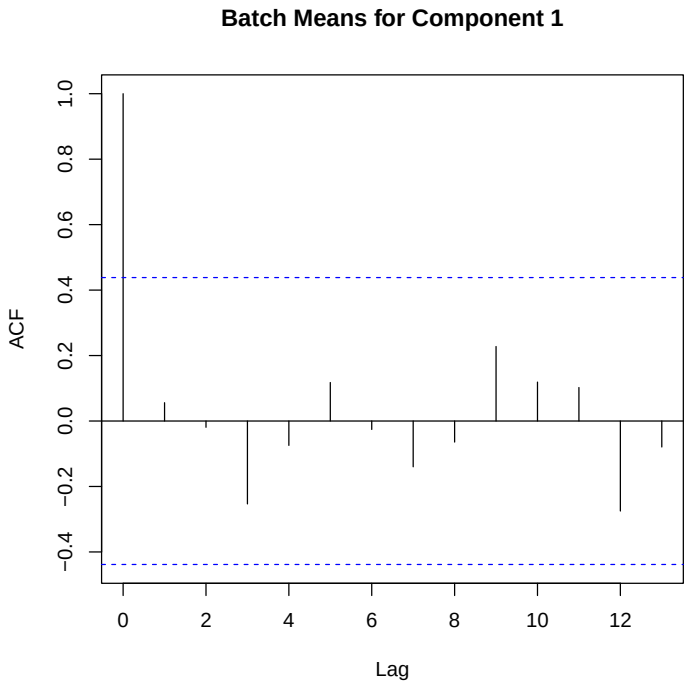

Figure 4: Autocorrelation Plot for Batch Means

This figure only shows the code for component 1 of the state vector. The other components (not shown) could be seen by changing the variable `i` and rerunning the `Sweave` source for the vignette (which is found in the package source tarball on CRAN). None of the components have any statistically significant autocorrelation for this batch size (1000). (We looked at them.)

Even though it is more general than `constrppprob` we recommend that `hitrun` be used even in problems where `constrppprob` is applicable, because `hitrun` has many features that `constrppprob` lacks.

References

Feller, W. (1968). *An Introduction to Probability Theory and its Applications*, volume I. New York: Wiley.

Ghosh, M. and Meeden, G. (1997). *Bayesian Methods for Finite Population Sampling*. London: Chapman and Hall.

Lazar, R., Meeden, G., and Nelson, D. (2008). A noninformative Bayesian approach to finite population sampling using auxiliary variables. *Survey Methodology*, 34:51–64.

Lo, A. (1988). A Bayesian bootstrap for a finite population. *Annals of Statistics*, 16:1684–1695.

Meeden, G. (1999). Interval estimators for the population mean for skewed distributions with a small sample size. *Journal of Applied Statistics*, 26:81–96.

Qu, Y., Meeden, G., and Zhang, B. (2015). An objective stepwise Bayes approach to small area estimation. *Journal of Statistical Computation and Simulation*, 85:1474–1494.

Rubin, D. (1981). The Bayesian bootstrap. *Annals of Statistics*, 9:130–134.

Smith, R. L. (1984). Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32:1296–1308.

Strief, J. and Meeden, G. (2014). Objective stepwise Bayes weights in survey sampling. *Survey Methodology*, 39:1–27.