Fast Two-Sample Permutation Tests, Even When One Sample is Large, that Efficiently Maximize Power Under Crude Monte Carlo Sampling

J.D. Opdyke, Economic Consulting Group – Andersen, LLP, Boston, MA

ABSTRACT

I present a method for quickly performing multiple nonparametric two-sample permutation tests on continuous data in SAS®, even when one sample is large. I maximize statistical power (within the context of a crude Monte Carlo approach) by "oversampling" – drawing more permutation samples than desired, deleting duplicates, and then selecting the desired number of samples from the remainder. I determine the optimal number of samples to "oversample" based on sampling probability and the runtime of a sampling procedure (PROC PLAN). Implementing "oversampling" with nearly optimal numbers of samples increases start-to-finish runtime typically by only 5%, and always by less than 10%. Using telecommunications performance measurement data from multiple sources with a wide range of sample size pairs, I benchmark start-to-finish runtime against a) another SAS® procedure (PROC MULTTEST), b) another SAS® program written for the same purpose, and c) Cytel's PROC TWOSAMPL®, with very favorable results. The relative benchmark speeds would be identical if applied to data from randomized controlled studies.

INTRODUCTION

Permutation tests are as old as modern statistics,[1] and their statistical properties are well understood and thoroughly documented in the statistical literature.[2] Though not always as powerful as their parametric counterparts that rely on asymptotic theory, they sometimes have equal or even greater power.[3] Often they can be used when asymptotic theory falls short (e.g. small samples and the Central Limit Theorem), and when fully enumerated, they provide gratifyingly exact results (as opposed to approximations based on asymptotic theory).[4] Most important, however, is their reliance on very few distributional assumptions,[5] giving permutation tests a much broader range of application.

Until recently the major drawback of permutation tests has been their high computational demands. Fully enumerating a permutation test requires calculating the test statistic appropriate for the hypotheses being tested for every possible two-sample combination of the data points.[6] Then the value of the test statistic based on the original two samples must be compared to those based on all the "permutation" samples to obtain a p-value[7] – the result of the test.[8] Drawing only a sample of all possible samples, as is typically done, still has been associated with prohibitive computer runtimes. Recent advances in computing capacity and speed, however, increasingly have relaxed this constraint. But efficient statistical code still is needed to most effectively exploit these advances and to ensure that the choice of method is driven as much by statistical theory, and as little by technological constraint, as possible. The goal of the methods described below is to contribute to this effort.

IMPLEMENTING PERMUTATION TESTS IN SAS®

Two procedures in SAS® can be used to perform two-sample nonparametric permutation tests – PROC MULTTEST[9] and PROC PLAN. The former directly samples the input dataset itself, while the later generates a record-by-record list identifying those records on the input dataset to include in the samples. This list subsequently must be merged with the original data to obtain the corresponding data points. In addition, PROC MULTTEST actually conducts the permutation test and provides a p-value (assuming that, for continuous data, a pooled-variance t-test is the appropriate test statistic); the results of PROC PLAN, on the other hand, must be manipulated "manually" to calculate the value of the test statistic associated with the original sample pair, and then compare it to those associated with all the permutation samples to obtain a p-value. However, this entire process using PROC PLAN still is much faster than PROC MULTTEST under most conditions, as shown in the benchmark section below, and it also provides more flexibility in the definition of the test statistic.

However, PROC PLAN has a sample size constraint – the product of the sum of the two sample sizes ($n_1 + n_2$) and the number of samples being drawn (T) cannot exceed 2^{31} or the procedure terminates. Yet this can be circumvented by inserting calls to PROC PLAN in a loop which cycles roundup($(n_1 + n_2)$* T / 2^{31}) times, each loop drawing T * [roundup($(n_1 + n_2)$* T / 2^{31})]$^{-1}$ samples until T samples have been drawn (see code in Appendix A). And this looping in and of itself does not slow the total runtime of the procedure. The relative speed of PROC PLAN when samples are large enough to require such looping is at least 20 times faster than PROC MULTTEST.

Another important issue regarding the implementation of these procedures is the sampling method: both PROC MULTTEST and PROC PLAN can perform crude Monte Carlo sampling without replacement *within a sample*, as required of a permutation test, but neither can avoid the possibility of drawing the same sample

1 Permutation tests were advocated by one of the fathers of modern statistics, Sir R.A. Fisher, as early as the 1920s.

2 Pesarin (2001) and Mielke and Berry (2001) contain extensive bibliographies.

3 For just one example, see Andersen and Legendre (1999).

4 In fact, it was Fisher who correctly characterized parametric tests relying on asymptotic theory as mere approximations to the exact results of fully enumerated permutation tests (Good (1994), p.4, citing Fisher).

5 Exchangeability of the data under both the null and alternate hypotheses is the major requirement of permutation tests. Good (1994) states that a permutation test requires only, "that the underlying distributions are symmetric, and/or the alternatives are simples [sic] shifts in value ", though there are cases satisfying these criteria where the basic, nonparametric permutation test discussed here is not the most appropriate method and should not be relied upon (e.g. the Behrens-Fisher problem -- see Pesarin (2001), Ch. 10).

6 Of course, the sample sizes of all these possible combinations are the same as the original two samples.

7 The p-value is simply a proportion – the percentage of "permutation" sample test statistic values at least as large as that based on the original data.

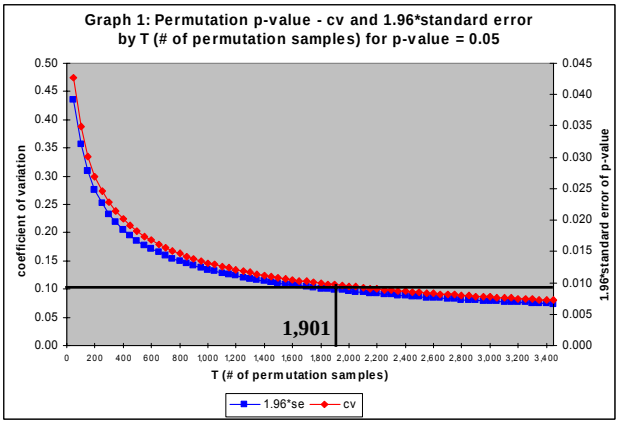
8 This paper addresses only the two-sample permutation test, though its methods readily can be applied to permutation tests with more complex study designs.

9 PROC MULTTEST is a versatile procedure with many functions – I address only its specific application to two-sample nonparametric permutation tests on continuous data.

more than once. In other words, when drawing a sample of samples, both procedures can only sample the entire samples *with replacement*. This problem of drawing duplicate samples, its effect on the statistical power of the permutation test, and a proposed solution that maximizes power under crude Monte Carlo sampling[10] are discussed below.

DETERMINING THE NUMBER OF PERMUTATION SAMPLES

Full enumeration of permutation tests quickly becomes infeasible as sample sizes increase because the number of possible sample combinations becomes very large, even for relatively small sample sizes.[11] However, full enumeration is unnecessary as the permutation test can be based on only a sample of all possible samples. Recognizing that the resulting p-value is simply an estimated proportion distributed binomially, we can determine its coefficient of variation (cv)[12] and confidence interval as functions of the number of samples (T) drawn. For example, to achieve cv>0.10 with a p-value equal to the critical value of the test (p-value = α = 0.05), T = 1,901.[13] This yields a 95% confidence interval of within 0.01 of the estimated p-value,[14] which for most practical purposes is a sufficiently precise estimate of the fully enumerated p-value.[15] Larger values of T will yield greater precision, but the marginal increases in precision decrease rapidly and nonlinearly (see Graph 1 below).[16]



THE PROBLEM OF DRAWING DUPLICATE SAMPLES

The above calculation assumes, however, that the T "permutation" samples are drawn without replacement – i.e. that no sample was drawn more than once.[17] More importantly, the permutation test will lose statistical power if there are any duplicate samples among these T samples.[18] Thus, maximizing power requires a draw of a unique set of samples. This can be accomplished in different ways, but the fastest when using PROC PLAN as described herein is to simply "oversample" by drawing more than T samples (say, r samples), deleting any duplicate samples, and then randomly selecting T samples from the remaining set. This approach does not violate any statistical assumptions required by a nonparametric permutation test – specifically, that the distribution of all possible samples is uniform with

$$\Pr(Y=y)=\frac{1}{\frac{(n_1+n_2)!}{n_1!n_2!}} \quad (1)$$

– so it is a valid approach for obtaining T unique samples. However, it brings up two interrelated questions, one regarding sampling probability, the other regarding computational efficiency: how many samples more than T (r-T) are required so that we are left with at least T unique samples once duplicates are deleted? r = 2T? r = 3T? And what is the runtime tradeoff of drawing larger factors of T samples, which takes more time simply because the number of samples is larger, but also reduces the chance of coming up short with less than T unique samples and having to redraw samples by calling PROC PLAN more than once? Solving the problem of minimizing expected runtime addresses both of these questions below.

10 These are simple, uniform random draws as opposed to more complicated sampling algorithms, such as importance sampling (see Mehta et. al. (1988)).

11 The number of possible sample combinations is given by $\frac{(n_1+n_2)!}{n_1!n_2!}$, which for n_1 = 3 and n_2 = 3 is just 20, but for the slightly larger sample pair of n_1 = 29 and n_2 = 30, is a sizeable 59,132,290,782,430,700.

12 The coefficient of variation is a unitless measure of relative spread. It is simply the standard error of a statistic divided by its mean (see Zar (1999), p. 40).

13 Solving for

$$cv=\frac{\sqrt{\frac{p-value\cdot(1-(p-value))}{T}}}{p-value}=\frac{\sqrt{\frac{0.05\cdot(1-0.05)}{T}}}{0.05}=0.10$$

yields T=1,900, so to obtain cv>0.10, we must increase T to 1,901. This calculation also is performed in Efron and Tibshirani (1993), pp. 208-211.

14 This calculation assumes that both the sample size (in this case, T) is sufficiently large and the value of the proportion (in this case, the p-value at p-value = α = 0.05) is sufficiently nonextreme (i.e. not close to 0 or 1) for the binomially distributed p-value to closely approximate the normal distribution. If these criteria are satisfied, the 95% confidence interval is defined by the range of p-value ± 1.96 standard errors. T = 1,901 and p-value = 0.05 satisfy a very strict of standard in the literature (npq>25, or T*p-value*(1-p-value)>25 – see Evans, Hastings, and Peacock (1993), p. 39) for such an approximation to be considered valid.

15 This approximate level of precision for an estimate of a proportion was deemed "excellent" (for T = 2,000) by Braun and Feng (2001) in their recent study of optimal permutation tests.

16 An efficient alternative is to increase T only if the p-value is close to the critical value of the test.

17 As mentioned above, permutation tests require that each sample be drawn "without replacement" – i.e. with no item drawn more than once *within* a sample. To maximize power, the sample of "permutation" samples also must be drawn "without replacement" – i.e. no entire sample among the T samples drawn may be drawn more than once.

18 Imagine, in the extreme, that the same sample pair is drawn all T times, or close to all T times. It would be impossible, or nearly so, to detect with a specified level of statistical confidence a difference between the two samples even if such a difference did, in fact, exist. In other words, it would be impossible to reject the null hypothesis of no difference even if there was a difference, and the extent to which a test can correctly reject the null hypothesis is its level of statistical power.

CHOOSING r TO MINIMIZE EXPECTED RUNTIME

Expected runtime is the product of the time it takes to draw r samples and the expected number of times r samples must be drawn to obtain at least T unique samples.[19] Addressing the latter factor first, the number of times r samples must be drawn before at least T unique samples are obtained follows the geometric distribution, which identifies the number of events occurring before the first success:

(2) $$\Pr(S=s)=p(1-p)^{(s-1)}$$

where p indicates the probability of success (of obtaining at least T unique samples) for each event (each call to PROC PLAN). The expected value of the geometric distribution is simply 1/p, and p is simply derived from a general form of the familiar collector's problem. This problem asks the question, "How many samples are required, when sampling with replacement, to obtain ('collect') all samples from the sampling distribution?" or more generally, "How many samples are required, when sampling with replacement, to obtain T samples from the sampling distribution?"[20] The number of samples "required" obviously will follow a probability density function, which in fact, is the sum of geometric random variables, and is presented below:

(3)

$$\Pr(\#\text{ unique samples}=j)=\frac{\frac{(n_1+n_2)!}{n_1!n_2!}}{j!\left(\frac{(n_1+n_2)!}{n_1!n_2!}-j\right)!}\sum_{i=0}^{j}\frac{(-1)^i\,j!(j-i)^r}{i!(j-i)!\left(\frac{(n_1+n_2)!}{n_1!n_2!}\right)^r}$$

where r = # of samples drawn and j ≤ r

The probability of obtaining at least T unique samples is simply the cumulative probability of obtaining T, T+1, T+2, … , r-1, and r unique samples, as shown below:

(4)

$$p=\Pr(j\geq T)=\sum_{j=T}^{r}\left[\frac{\frac{(n_1+n_2)!}{n_1!n_2!}}{j!\left(\frac{(n_1+n_2)!}{n_1!n_2!}-j\right)!}\sum_{i=0}^{j}\frac{(-1)^i\,j!(j-i)^r}{i!(j-i)!\left(\frac{(n_1+n_2)!}{n_1!n_2!}\right)^r}\right]$$

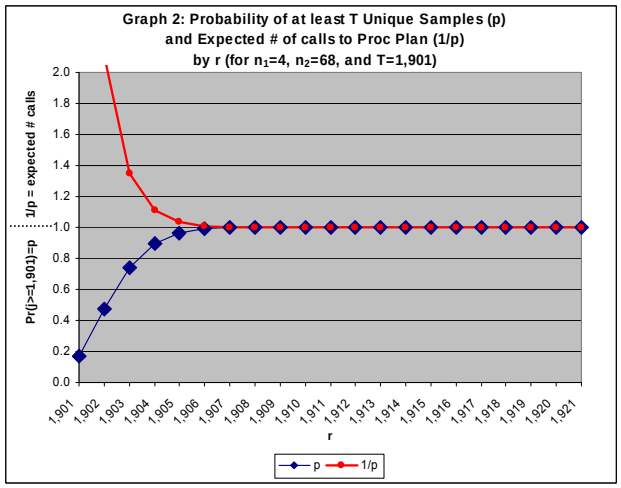
where T ≤ r.

Thus, the expected number of times r samples must be drawn to obtain at least T unique samples is a function of the number of possible sample combinations and r, and is provided below:

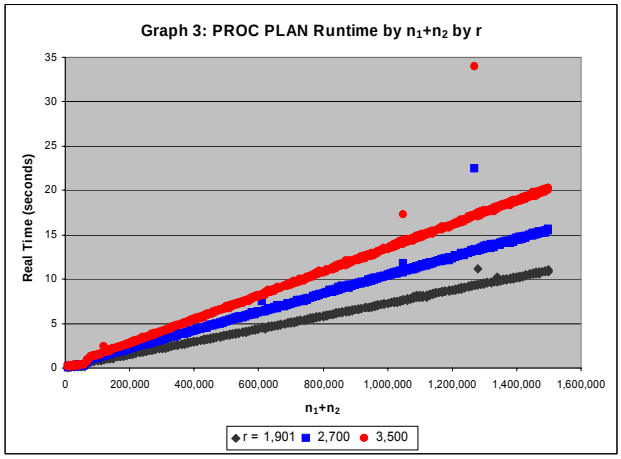
(5) expected # of calls to PROC PLAN =

$$\left(\frac{1}{p}\right)=\left(\sum_{j=T}^{r}\left[\frac{\frac{(n_1+n_2)!}{n_1!n_2!}}{j!\left(\frac{(n_1+n_2)!}{n_1!n_2!}-j\right)!}\sum_{i=0}^{j}\frac{(-1)^i\,j!(j-i)^r}{i!(j-i)!\left(\frac{(n_1+n_2)!}{n_1!n_2!}\right)^r}\right]\right)^{-1}$$

Graph 2 below illustrates this functional relationship between p, 1/p, and r for n_1 = 68, n_2= 4, and T = 1,901:



Now to return to the first factor determining expected sampling runtime – the time it takes PROC PLAN to draw a sample of r samples. This is simply the runtime of PROC PLAN as a function of, interestingly, not the number of possible two-sample combinations, but rather the sum of the two sample sizes (n_1 + n_2), as well as the number of samples drawn, r. This is shown in Graph 3 below.[21]



Obviously, r and (n_1 + n_2) are correlated, but runtime is very well predicted (adjusted R^2 = 0.9884) by the simple ordinary least squares multivariate regression equation below:

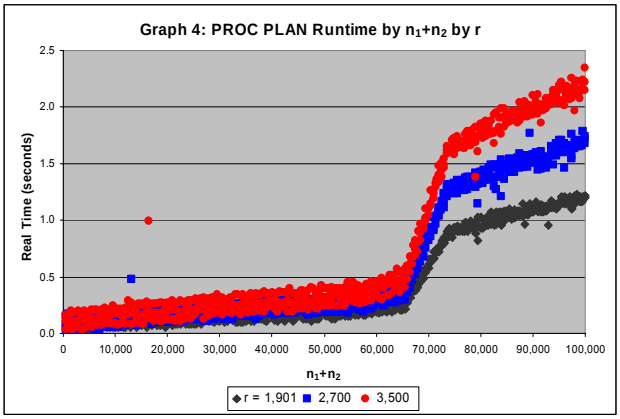
19 Investigating the possibility of drawing fewer than r samples if PROC PLAN is called more than once increases the complexity of this problem beyond the scope of this paper. Given the proximity to T of the nearly optimal values for r provided in Table 1 below, and the very fast runtime of PROC PLAN when the probability of "recalling" it is anything but infinitesimal, I suspect small, if any, gains in runtime would result from such an inquiry, though I intend to explore the issue.

20 Of course, the sampling distribution here is all possible two-sample combinations based on the two original samples of data.

21 PROC PLAN was run in SAS® v.8.2 on a desktop PC with 2GB RAM and a 2GHz Pentium processor. Sample sizes were generated by assigning values of 3, 16, 27 to the smaller of the two samples, and, beginning at 100, assigning values by 100 increments to the larger sample up to 100,000, after which point increments of 10,000 were used up to 1.5 million (though the program has been run on sample pairs as large as 29 and 5,000,029). Three values of r were used: 1,901, 2,700, and 3,500.

PROC PLAN Runtime =
PPRT(n_1, n_2, r) = $\beta_0 + \beta_1*(n_1 + n_2) + \beta_2*r + \beta_3*(n_1 + n_2)*r$

Nonlinearity at about (n_1 + n_2) = 65,500 and (n_1 + n_2) = 73,500 prompted the inclusion of dummy and interaction terms, leading to the near perfect prediction (adjusted R^2 = 0.9927) for PPRT(n_1, n_2, r) presented in Appendix B (see Graph 4, which is simply a magnification of Graph 3 up to (n_1 + n_2)=100,000).

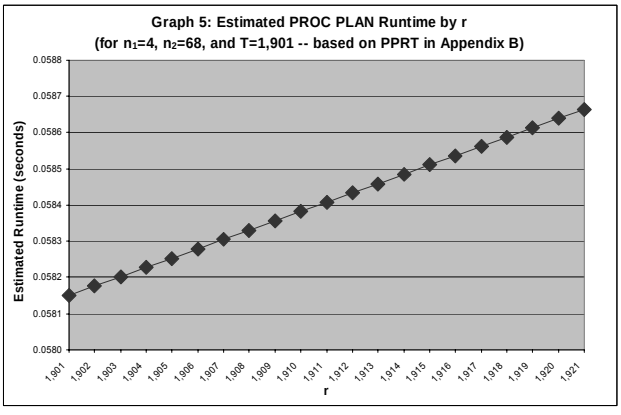


Thus, expected runtime g(n_1, n_2, r, T) is the product of PROC PLAN Runtime and the expected number of calls to PROC PLAN:

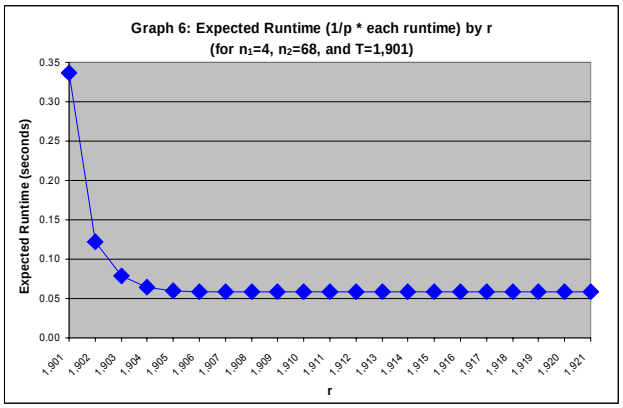
(6) expected runtime = g(n_1, n_2, r, T) = PPRT(n_1, n_2, r) * expected # of calls to PROC PLAN = PPRT(n_1, n_2, r) *

$$\left(\sum_{j=T}^{r} \left[\frac{\frac{(n_1+n_2)!}{n_1!n_2!}}{j!\left(\frac{(n_1+n_2)!}{n_1!n_2!} - j\right)!} \sum_{i=0}^{j} \frac{(-1)^i\, j!(j-i)^r}{i!(j-i)!\left(\frac{(n_1+n_2)!}{n_1!n_2!}\right)^r} \right] \right)^{-1}$$

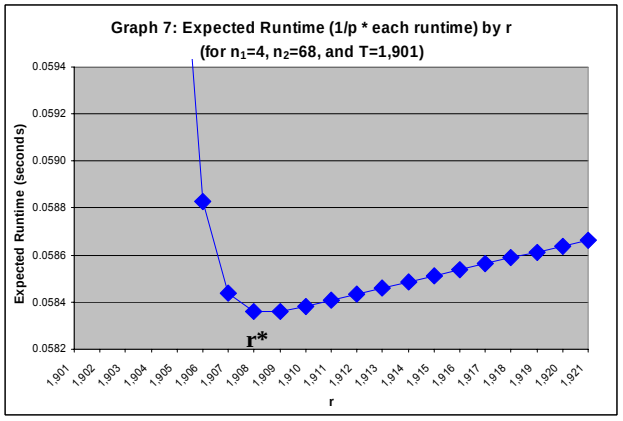
To get an intuitive feel for r as a function of n_1 and n_2 (for a given T), note again that the second term of (6) above is a combinatorial function of the sample sizes while the first term is merely a linear function of the sample sizes (see Graph 5 below).



The combinatorial terms in the second term of (6) end up dominating as sample sizes increase, asymptotically converging to 1.0 (one call to PROC PLAN) faster than the first term (each PROC PLAN runtime) diverges. Hence, for all but the smallest sample sizes, an optimal r, in terms of expected runtime (where $\partial g/\partial r = 0$), will be fairly close to T. Graphs 6 and 7 below present g(n_1, n_2, r, T) (the product of 1/p in Graph 2 and PPRT in Graph 5 above) and demonstrate an optimal r, r* = 1,908, for T = 1,901, n_1 = 4, and n_2 = 68 (and number of possible sample combinations = 1,028,790).



Graph 7 magnifies the relevant expected runtime range.



Unfortunately, the high level of precision needed to calculate numeric solutions for r* for different sample sizes (and T) requires use of a symbolic programming language, and thus, cannot be implemented "on the fly" in SAS®.[22] However, for all practical purposes, r* need not be calculated for every combination of values of n_1 and n_2 – nearly optimal r can be calculated for ranges of the number of possible sample combinations because, as shown in Graph 7, the marginal runtime cost of drawing r slightly larger than r* is negligible (though the marginal runtime cost of drawing r smaller than r* is relatively large). Thus, if we create appropriate ranges of the number of sample combinations, and for the low end of each range identify the lowest corresponding (n_1 + n_2) (because this gives us the lowest marginal increase in PPRT as r increases, and thus, the largest r*), then the corresponding "low-end" r* will never be lower than any other r* for any of the sample pairs within that range. In other words, though not optimal for every combination of sample sizes within the range, the "low-end" r* will be nearly optimal because it will be slightly larger (never smaller) than all other r* for sample size pairs within that range, and the marginal cost of being slightly larger than r* is small, if not negligible. Table 1 below shows the values of r used in the program – the "low-end" r*'s – for different ranges of the number of possible sample combinations.

22 I used Mathematica® v.4.1 to calculate p in Table 2 from the cumulative distribution function (4) of the generalized collector's problem. This code is available upon request. Good approximations to (4), however, do exist – see Read (1998) and Kuonen (2000).

TABLE 1:
Nearly Optimal r ("low-end" r*) Used in Program, Probability (p) of Obtaining T = 1,901 Unique Samples, and Expected # of Calls to PROC PLAN (1/p) by Ranges of # of Possible Sample Combinations, C

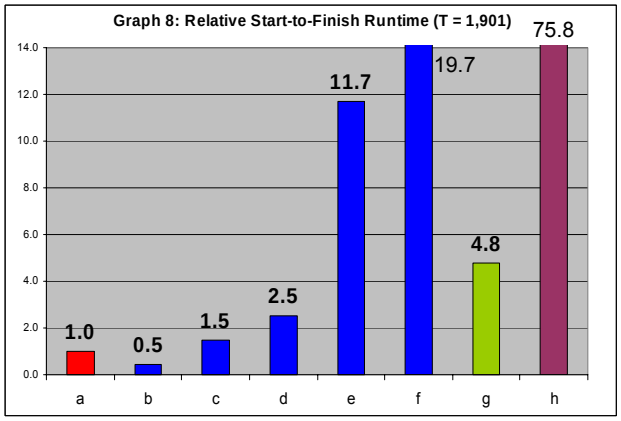
$C = \frac{(n_1+n_2)!}{n_1!n_2!}$	"low-end" r*	p (lower bound)	1/p (lower bound)
C < 10,626	C	1.0 (assuming C ≥ T)	1.0
10,626 ≤ C < 52,360	2,155	0.99990058618 0442 (approx.)	1.000099423703 650
52,360 ≤ C < 101,270	1,960	0.99928750023 4577 (approx.)	1.000713007783 300
101,270 ≤ C < 521,855	1,934	0.99942971769 2296	1.000570607715 190
521,855 ≤ C < 1,028,790	1,912	0.99972655524 0808	1.000273519551 680
1,028,790 ≤ C < 10,009,125	1,908	0.99951283912 0371	1.000487398321 020
10,009,125 ≤ C < 25,637,001	1,904	0.99996159418 0711	1.000038407294 350
25,637,001 ≤ C < 100,290,905	1,903	0.99994461537 6581	1.000055387691 050
100,290,905 ≤ C < 5,031,771,045	1,902	0.99983969137 9204	1.000160334323 770
5,031,771,045 ≤ C	1,901	0.99964115494 0541	1.000358973875 460

POWER AT WHAT COST?

A good metric for the "cost" of employing "oversampling" via Table 1 above in order to maximize statistical power is its start-to-finish runtime compared to that associated with just drawing T samples and ignoring the duplicate sample problem. Employing Table 1 increased start-to-finish runtime typically by 5% but always by less than 10%. Maximizing power arguably is worth this relatively small increase in runtime.

HOW FAST IS IT?
RELATIVE SPEED – SOME BENCHMARKS

The start-to-finish runtime of the program using the "oversampling" method with PROC PLAN as described above is fast relative to other programs and procedures, as seen in Graph 8 below:[23]



where
a = PROC PLAN with "oversampling"
b = PROC MULTTEST, (n_1+n_2)<10,000, 1 Study per Control
c = PROC MULTTEST, (n_1+n_2)<10,000, Many Study per Control
d = PROC MULTTEST, 10,000<(n_1+n_2)<100,000
e = PROC MULTTEST, 100,000<(n_1+n_2)<150,000
f = PROC MULTTEST, 1,000,000<(n_1+n_2)<1,500,00
g = Cytel's PROC TWOSAMPL®[24]
h = looping in SAS®[25]

The only procedure or program faster than PROC PLAN with "oversampling" is PROC MULTTEST with small samples *and* one study group per control group.[26] On the one hand, smaller sample ranges are where one is most likely to need permutation tests. However, this is where the speed differential matters the least in absolute terms – even when comparing two hundred permutation tests with these smaller sample sizes and a study/control group ratio equal to one, PROC MULTTEST was never more than five minutes faster than PROC PLAN. So the tradeoff is several minutes per run with PROC MULTTEST, versus maximum power and flexibility in the definition of the test statistic with PROC PLAN with "oversampling."

In addition to the speed of PROC PLAN itself, a number of factors contribute to the speed of the entire program using PROC PLAN with "oversampling," including:

- Use of PROC APPEND to "set" two large datasets together (one on top of the other) whenever possible.
- Judicious use of multiple PROC TRANSPOSE's to evaluate the summarized results of the permutation sampling.

23 Of course, these alternatives do not delete duplicate samples and thus, for the same number of samples T, have less power than PROC PLAN with "oversampling" to the extent they draw duplicate samples.

24 Time did not permit a thorough investigation into the effect of lowering the study/control group ratio to one for PROC TWOSAMPL, though early indications are that its relative runtime decreases somewhat, though not nearly as much as that of PROC MULTTEST.

25 See Jackson (1998) for a copy of the SAS® code employing looping to sample the permutation samples. Beware, however, that the code enters an infinite loop if the number of possible sample combinations for a given sample pair is less than T. Also note that the code, unlike the appropriate definition of a permutation test which includes "ties" in the numerator of the p-value, splits ties at the boundary after assuming exactly one tie at the boundary (apparently in an attempt to make the test less statistically conservative).

26 For $(n_1 + n_2)$ beyond approximately 10,000, the study/control group ratio made less of a difference in PROC MULTTEST's runtime. The times reported for larger $(n_1 + n_2)$ are for a study/control group ratio = 1, which would be the fastest time for PROC MULTTEST.

- Most test statistics can be constructed based on just one of the two samples in a pair and, if necessary, the pooled summary statistics of the pair. Thus, when conducting permutation sampling, sample only the smaller of the two samples, but keep track of which sample is used (study or control) when constructing the test statistics based on the permutation samples.

- Setting together the many output datasets from PROC PLAN by entering their names into a long string assigned to a global variable using CALL SYMPUT, and then placing the global variable in a set statement so that all the datasets are set together all at once. Alternatives, such as cumulatively setting or even PROC APPENDing the datasets together one at a time in a loop, can consume a fair amount of runtime.

- If the dataset is large and contains a large percentage of records with the same response variable value (say, zero), delete these records to avoid sorting and later merging them with the PROC PLAN output list. After merging the remaining data with the PROC PLAN output and retaining all PROC PLAN records in the merge, reassign this value to the response variable when it is missing (i.e. when that record did not merge with the PROC PLAN output because it had been deleted).

- Most importantly, if the data contains multiple study groups per control group, there is no need to output control group records multiple times for each corresponding study group when using PROC PLAN as outlined above. The original data simply can be divided into two datasets – one for control group(s) and one for study groups – and each merged separately to the PROC PLAN output (then (PROC) APPENDed together after the merges). Unless one constructs a separate dataset for each permutation test, both PROC MULTTEST and PROC TWOSAMPL require control group records duplicated in the input dataset for each study group against which they are being compared. If the dataset is large, this can take a tremendous amount of sort-time as both procedures require sorted input data.

ABSOLUTE SPEED

When run on data containing 220 sample pairs where the smaller sample was less than 30 observations but the larger sample was sometimes over 64,000 observations, the runtime of the program was 7 minutes, 45 seconds.[27] For data containing 6,682 sample pairs where the smaller sample was less than 30 observations but the larger was sometimes over 5,000,000 observations, the runtime was 8 hours, 36 minutes. The former example obviously is more typical of the contexts in which permutation tests are used, but the latter is instructive because it is important to know the limits of the methods and software being relied upon. This study shows that the runtime of PROC PLAN with "oversampling" is not prohibitive even when applied to sample sizes as large (if not far larger) than would ever be used with permutation tests.[28]

The same cannot be said for the three alternate methods.

CONCLUSION

I have presented a method for quickly performing multiple two-sample nonparametric permutation tests on continuous data in SAS®, even when one sample is large. I also have presented a practical and efficient method for maximizing power (within the context of a crude Monte Carlo approach) with little increase in runtime (5-10%). The program utilizing these two methods is at least several times faster than several widely available alternatives in almost all circumstances, does not have the sample size constraints of all of these alternatives, and has more flexibility in the definition of the test statistic used compared with two of these alternatives (the two procedures). Only the two-sample test was explored in this study, but the methods presented are flexible and can be adapted to more complex study designs. To quote a medical study that empirically and favorably compares permutation tests to their asymptotic theory counterparts, with continual improvements in computing speed "…there seems no important argument remaining against preferred use of this elementary but exact [when fully enumerated] and flexible method" (Bullmore et al. (1999), p.42). Use of the methods presented in this paper hopefully will contribute to the effort of efficiently exploiting continual advances in computing speed and capacity so that the choice and implementation of statistical methods will be driven by applicable statistical theory and not technological constraint.

27 All programs were run in SAS® v.8.2 on a desktop PC with 2GB RAM and a 2GHz Pentium processor.

28 One such context is the telecommunications regulatory arena. As a requirement of the Telecommunications Act of 1996, Regional Bell Operating Companies (RBOCs), in order to be permitted to enter the long distance phone service market, must open their local phone service monopolies to competition. To initially establish competitive economic markets, this requires RBOCs to provide service to their competitors' customers that is equivalent to the service they provide to their own customers. Proof of this service "parity" has been required in the form of thousands of statistical tests on hundreds of service performance metrics (e.g. how quickly is a phone line installed; how quickly is a phone line repaired, etc.). When one of the two samples being compared is small (typically that of the Competitive Local Exchange Carrier's customers) and the Central Limit Theorem cannot be relied upon, State Public Service Commissions often have required the use of permutation tests, even if the sample corresponding to the RBOC's own customers is millions of observations. Of the methods examined in this study, only PROC PLAN with "oversampling" realistically can be relied upon to handle such comparisons (for continuous data performance metrics) in a timely manner.

APPENDIX A

```
*** The code below has been run, and its results
verified, many times.  Translating file formats,
however, can always cause typos.  Please notify
the author if you spot one.;

*** set global variables;

%let npermsampT=1901;
%let seednum   =1;
%let seedorig  =1;

*** The PROC PLAN seed(s) can be set to -1 to
key off the time of day (but they still must be
incremented when PROC PLAN is looped below or
the same draw of samples will result).  The seed
generated can be saved and reused if necessary
for verification / process control purposes.;

*** summarized data contains # study group obs,
# control group obs;

*** use # of obs on summarized input data to
obtain # of permutation tests;

proc summary data=sumdinpt;
  var nobscntrl;
  output out=npermtst(drop=_TYPE_) n=nperms;
  run;

data npermtst
     error
  ;
  set npermtst(keep=nperms _FREQ_);
  if nperms~=_FREQ_ then output error;
  else do;
    output npermtst;
    call symput('npermtst',compress(nperms));
  end;
  run;

title "Missing NOBS Control Group";
proc print data=error;
  run;
title " ";


*** create permutation samples one sample pair
at a time with the macro makesamp;

*** variables passed:
exprment  = an identifier of the experiment (any
            number of such "by variables" can be
            inserted into the macro)
bigcomb   = identifies which is larger: combins
            or r (the size of the permutation
            sample)
nobsmalr  = min(nobstudy, nobscntrl)
combins   = comb(sumofnobs, nobsmalr)
minrcomb  = min(combins, r)
sumofnobs = sum(nobstudy, nobscntrl)
ncalls2pp = number of times to loop on PROC PLAN
            if nobstudy+nobscntrl * size of draw
            > 2^31
topdraws  = # of samples to draw in all but last
            loop
lastdraw  = # of samples to draw in last loop
            (takes can of modulus)
smaller   = identifies which sample - study or
            control - is smaller and thus,
            corresponds with the permutation
            samples
recountr  = counter for record # of summarized
            data fed into makesamp ... should
            count up to global variable npermtst

%macro makesamp(exprment  = ,
                bigcomb   = ,
                nobsmalr  = ,
                combins   = ,
                minrcomb  = ,
                sumofnobs = ,
                ncalls2pp = ,
                topdraws  = ,
                lastdraw  = ,
                smaller   = ,
                recountr  =
               );

*** if combins <= r, choose all sample
combinations, then select T samples from them;

%if &bigcomb=0 %then %do;

  proc plan seed=&seednum;
  factors drawnum   = &combins
          dataobsid = &nobsmalr of &sumofnobs
                      comb / noprint;
  output  out       = psamp&recountr;
  run;

  %if &combins>&npermsampT %then %do;

    proc plan seed=&seednum;
    factors drawnum   = 1
            dataobsid = &npermsampT of &combins
                        random / noprint;
    output  out       = choosmp;
    run;

    data choosmp(keep=drawnum);
      set choosmp(drop=drawnum);
      drawnum=dataobsid;
      run;

    proc sort data=choosmp;
      by drawnum;
      run;

    proc sort data=psamp&recountr;
      by drawnum;
      run;

    data psamp&recountr;
       merge psamp&recountr
             choosmp(in=inchoos)
       ;
       by drawnum;
       if inchoos then output psamp&recountr;
       run;

    data psamp&recountr(drop=drawnum2 temp);
      set psamp&recountr(drop=drawnum);
      retain drawnum2;
      temp=mod(_n_,&nobsmalr);
      if _n_=1 then drawnum2=1;
      else if temp=1 then drawnum2=drawnum2+1;
      drawnum=drawnum2;
      run;

  %end;
```

```
%end;

*** if combins > r, check whether PROC PLAN
needs to be looped multiple times -- if not,
simply select r samples, delete duplicates, and
keep T samples.  If so, loop it first to select
r samples.  In either case, redraw samples if
fewer than T unique samples are drawn the first
time around.;

%if &bigcomb=1 %then %do;

  %redraw3:
  %if &ncalls2pp=1 %then %do;

    proc plan seed=&seednum;
    factors drawnum   = &minrcomb
            dataobsid = &nobsmalr of &sumofnobs
                        random / noprint;
    output  out       = psamp&recountr;
    run;

    proc sort data=psamp&recountr;
      by drawnum;
      run;

    proc transpose data=psamp&recountr out=temp
                        prefix=stdy;
      var dataobsid;
      by drawnum;
      run;

    proc sort data=temp out=temp nodupkey;
      by stdy1-stdy&nobsmalr;
      run;

    proc sort data=temp;
      by drawnum;
      run;

    data temp;
      set temp;
      by drawnum;
      if last.drawnum;
     call symput("uniqsampn",compress(drawnum));
      run;

     %if &uniqsampn<&npermsampT %then %do;
       data temp;
         run;
       data temp;
         set temp;
         drawsize="&minrcomb";
         combins =
          "&sumofnobs"||" choose "||"&nobsmalr";
         nuniqdrw=&uniqsampn;
         experiment ="&exprment";
         seednum2=&seednum+1;
      call symput('seednum',compress(seednum2));
         run;
       title "Fewer than &npermsampT Unique
Permutation Samples Selected for &exprment.:
Redraw Performed";
       proc print data=temp;
         run;
       title " ";
       %goto redraw3;
     %end;

     %if &uniqsampn>&npermsampT %then %do;

        data psamp&recountr;
          set psamp&recountr;
          if drawnum<=&npermsampT;
        run;

     %end;

  %end;


  %redraw4:
  %if &ncalls2pp>1 %then %do q=1 %to &ncalls2pp;

    %if &q<&ncalls2pp %then %do;

      proc plan seed=&q;
        factors drawnum   = &topdraws
                dataobsid = &nobsmalr of
                            &sumofnobs random /
                            noprint;
        output  out       = ptemp&q;
        run;

    %end;

    %if &q=&ncalls2pp %then %do;

      proc plan seed=&q;
        factors drawnum   = &lastdraw
                dataobsid = &nobsmalr of
                            &sumofnobs random /
                            noprint;
        output  out       = ptemp&q;
        run;

      data psamp&recountr;
        set ptemp1;
        run;

      %do k=2 %to &q;
        data psamp&recountr;
          set psamp&recountr ptemp&k(in=in&k);
          if in&k then
               drawnum=drawnum+(&k-1)*&topdraws;
          run;
      %end;

      proc sort data=psamp&recountr;
        by drawnum;
        run;

      proc transpose data=psamp&recountr
                          out=temp prefix=stdyn;
        var dataobsid;
        by drawnum;
        run;

      proc sort data=temp out=temp nodupkey;
        by stdyn1-stdyn&nobsmalr;
        run;

      proc sort data=temp;
        by drawnum;
        run;

      data temp;
        set temp;
        by drawnum;
        if last.drawnum;
     call symput("uniqsampn",compress(drawnum));
        run;

      %if &uniqsampn<&npermsampT %then %do;
        data temp;
```

```
          run;
        data temp;
          set temp;
          drawsize="&minrcomb";
          combins =
          "&sumofnobs"||" choose "||"&nobsmalr";
          nuniqdrw=&uniqsampn;
          experiment ="&exprment";
          seednum2=&seednum+1;
      call symput('seednum',compress(seednum2));
          run;
        title "Fewer than &npermsampT Unique
Permutation Samples Selected for &exprment.:
Redraw Performed";
        proc print data=temp;
          run;
        title " ";
        %goto redraw4;
      %end;

      %if &uniqsampn>&npermsampT %then %do;

         data psamp&recountr;
           set psamp&recountr;
           if drawnum<=&npermsampT;
           run;

      %end;
    %end;
  %end;
%end;

*** set lengths of any "by variables" in PROC
PLAN output equal to those in the original
dataset for a smooth merge of the PROC PLAN
output with the original dataset;

data psamp&recountr(drop=seedback);
  length experiment $10. whichsmlr $4.;
  set psamp&recountr;
  seedback="&seedorig";
  call symput('seednum',compress(seedback));
  experiment ="&exprment";
  whichsmlr  ="&smaller";
  run;

%mend;


*** use nested macro to call PROC PLAN to
generate random samples for each record of the
summarized input dataset;

%macro loopingmacro(numloops);

%do i=1 %to &numloops;

  data temp;
    set sumdinpt(keep=experiment studynobs
                      contrlnobs
                 );
    if &i=_n_ then do;

    n=sum(studynobs,contrlnobs);
    k=min(studynobs,contrlnobs);
    combin=comb(n,k);
    drop n k;

*** for versions of SAS 6.12 and older, comb(,)
will only go up to about 1*10E70 before
terminating, so use the loop below;
*      combin=1;
*      do j=k to 1 by -1;
*         combin=combin*(n-j+1)/j;
*      end;
*      drop n k j;
*** The current numeric size limit of SAS® is
1.8*10E308 – if one sample of the pair being
tested is less than 30 observations, the larger
sample would have to approach 500,000,000,000
observations for the number of possible two-
sample combinations to exceed this limit.  Equal
sample sizes of n1 = n2 = 514 approach this limit
(# combinations = 7.1560510549*10E307), but one
should examine closely whether permutation tests
are necessary and/or appropriate under these
circumstances.;

*** the 'table' below was calculated based on
    npermsampT=1901;

    if combin<10626 then nsamp=combin;
    else if combin<52360 then nsamp=2155;
    else if combin<101270 then nsamp=1960;
    else if combin<521855 then nsamp=1934;
    else if combin<1028790 then nsamp=1912;
    else if combin<10009125 then nsamp=1908;
    else if combin<25637001 then nsamp=1904;
    else if combin<100290905 then nsamp=1903;
    else if combin<5031771045 then nsamp=1902;
    else if combin>=5031771045 then nsamp=1901;

    call symput('ncomb',combin);
    x=min(combin,nsamp);
    call symput('mincombr',compress(x));

    if combin=x then combbigr=0;
    else if combin>x then combbigr=1;
    call symput('combbig',compress(combbigr));

 nloops=ceil(x*sum(studynobs,contrlnobs)/2**31);
    modplus=mod(&npermsampT,nloops);
    toploops=floor(&npermsampT/nloops);
    botmloop=toploops+modplus;
    call symput('ncalls',compress(nloops));
    call symput('topndraw',compress(toploops));
    call symput('lastndraw',compress(botmloop));

*** assign names of by variables;
    call symput('exprmnt',experiment);

    if studynobs<=contrlnobs
       then whichsml="stdy";
       else whichsml="cntl";

*** keep track of which sample (size)
corresponds to the permutation samples for
calculating the test statistic;
    call symput('whichsmlr',compress(whichsml));

call symput('nsmaller',
          compress(min(studynobs,contrlnobs)));
call symput('sumnobs',
          compress(sum(studynobs,contrlnobs)));
    output;
  end;
  run;

  % makesamp(exprment  =&exprmnt,
            bigcomb   =&combbig,
            nobsmalr  =&nsmaller,
            combins   =&ncomb,
            minrcomb  =&mincombr,
            sumofnobs =&sumnobs,
            ncalls2pp =&ncalls,
            topdraws  =&topndraw,
            lastdraw  =&lastndraw,
```

```
            smaller   =&whichsmlr,
           recountr  =&i
          );
%end;

%mend;

%loopingmacro(&npermtst);

*** now using CALL SYMPUT, place all the PROC
PLAN output datasets (psamp1, psamp2, etc.) into
a long string in a global variable so they can
all be set together at once (much faster than
setting or appending them cumulatively in a
loop).  Then merge this entire PROC PLAN output
dataset with the original dataset by the
appropriate "by variables" (e.g. experiment) and
the record id variable (above, dataobsid).;
```

APPENDIX B

PROC PLAN RunTime, PPRT(n_1, n_2, r), regression results:
Left hand side variable = runtime seconds
adjusted R^2 = 0.9927

Variable Key	Parameter Estimate	t value
A	0.0432387277000000	1.80
B	-0.0000001298032000	-2.88
C	0.0000838185000000	9.68
D	0.0000000038095955	234.72
E	-0.0340413560000000	-0.89
F	0.0000004543242500	0.58
G	-0.0000581740000000	-4.24
H	-0.0000000024994500	-8.86
I	-0.4873557050000000	-0.38
J	0.0000071862352000	0.39
K	-0.0016941670000000	-3.70
L	0.0000000228154240	3.47

Variable Key	Variable
A	Intercept
B	(n_1 + n_2)
C	r
D	(n_1 + n_2) * r
E	(dummy: (n_1 + n_2)<65.5K)
F	(dummy: (n_1 + n_2)<65.5K) * (n_1 + n_2)
G	(dummy: (n_1 + n_2)<65.5K) * r
H	(dummy: (n_1 + n_2)<65.5K) * (n_1 + n_2) * r
I	(dummy: 65.5K ≤ (n_1 + n_2) ≤ 73.5K)
J	(dummy: 65.5K ≤ (n_1 + n_2) ≤73.5K) * (n_1 + n_2)
K	(dummy: 65.5K ≤ (n_1 + n_2) ≤73.5K) * r
L	(dummy: 65.5K ≤ (n_1 + n_2) ≤ 73.5K) * (n_1 + n_2) * r

REFERENCES

Andersen, M.J. and P. Legendre, "An empirical comparison of permutation methods for tests of partial regression coefficients in a linear model," *Journal of Statistical Computation and Simulation*, Vol. 62, No. 3, 1999.

Bullmore, Edward T., John Suckling, Stephan Overmeyer, Sophia Rabe-Hesketh, Eric Taylor, and Michael J. Brammer, "Global, Voxel, and Cluster Tests, by Theory and Permutation, for a Difference Between Two Groups of Structural MR Images of the Brain," *IEEE Transactions on Medical Imaging*, Vol. 18, No. 1, 1999.

Braun, Thomas M. and Ziding Feng, "Optimal Permutation Tests for the Analysis of Group Randomized Trials." *Journal of the American Statistical Association*, Vol. 96, No. 456, December, 2001.

Evans, Merran, Nichols Hastings, and Brian Peacock, Statistical Distributions, 2nd ed., New York: John Wiley & Sons, 1993.

Good, Phillip, Permutation Tests, Springer-Verlag, New York, 1994.

Efron, Bradley and Robert Tibshirani, An Introduction to the Bootstap, Chapman & Hall, London & New York, 1993.

Affidavit of John Jackson, On Behalf of MCI-Worldcom, Before the Michigan Public Service Commission, Case No. U-11830, November 18, 1998, ATTACHMENT A, "Using Permutation Tests to Evaluate the Significance of CLEC vs. ILEC Service Quality Differentials"

Kuonen, Diego, "A Saddlepoint Approximation for the Collector's Problem," *The American Statistician*, Vol. 54, No. 3, August 2000.

Mehta, Cyrus R., Nitin R. Patel, Pralay Senchauduri, "Importance Sampling for Estimating Exact Probabilities in Permutational Inference," *Journal of the American Statistical Association*, Vol. 83, No. 404, 1988.

Mielke, Paul W. and Kenneth J. Berry, Permutation Methods: A Distance Function Approach, Springer-Verlag, New York, 2001.

Pesarin, Fortunado, Multivariate Permutation Tests with Applications in Biostatistics, John Wiley & Sons, Ltd., New York, 2001.

Read, K.L.Q., "A Lognormal Approximation for the Collector's Problem," *The American Statistician*, Vol. 52, No. 2, May 1998.

Zar, Jerrold H., Biostatistical Analysis, 4th ed., Upper Saddle River, NJ: Prentice Hall, 1999.

ACKNOWLEDGMENTS

I owe special thanks to Geri Costanza, M.S., who provided me with a number of valuable insights. Any errors are my own.

CONTACT INFORMATION

Please contact the author with your questions and comments at:

J.D. Opdyke
Economic Consulting Group - Andersen, LLP
225 Franklin Street, 13th Floor
Boston, MA 02110
Work Phone: 203-249-4837, 617-330-5794
Fax: 617-330-4398
Email: jdopdyke@usa.net
john.d.opdyke@andersen.com