Adapted from SAS notes of P. Smith

12 Simulation Loops in Splus vs. SAS

12.1 Random number generation in SAS.

SAS has a variety of pseudo-random number generation functions which can be invoked during a DATA step ('pseudo' because computer software implements deterministic algorithms and so does not produce genuinely random numbers). In SAS, one must supply an initial random number or *seed* to generate a pseudorandom sequence. (In **Splus**, recall, one *can* specify or record the seed using the reserved vector **.Random.seed**, but one need not unless it is desirable to re-use the same seed later, e.g. for debugging.) In SAS, the seed is a nonzero integer with absolute value less than $2^{31} - 1$. Each call to a random number generating function must use the seed as a function argument, and the seed is updated on each call. The resulting sequence should appear to be a sequence of independent random variables with a preassigned distribution. As an example, consider the following SAS program. The sample moments and quantiles are very close to the theoretical values.

```
data random ;
  seed = 4055061 ;
  do i=1 to 100 ;
    uu = ranuni(seed) ;
    output ;
  end ;
title 'RANDOM UNIFORM NUMBERS' ;

proc univariate data=random plot ;
  var uu ;
```

The SAS function RANUNI returns a random variable with a uniform distribution on $[0, 1]$. Here is the edited output of our little simulation.

```
                    RANDOM UNIFORM NUMBERS
                             Univariate Procedure
Variable=UU

                                  Moments

                  N                1000  Sum Wgts       1000
                  Mean         0.492469  Sum         492.4692
                  Std Dev      0.288889  Variance    0.083457
                  Skewness     0.082374  Kurtosis    -1.19532
...
                               Quantiles(Def=5)

                   100% Max   0.998822        99%   0.992526
                    75% Q3    0.743291        95%   0.956327
                    50% Med   0.484571        90%    0.90731
                    25% Q1    0.240905        10%   0.108907
                     0% Min    0.00117         5%   0.055209
                                               1%   0.007447
                   Range      0.997652
                   Q3-Q1      0.502387
                   Mode        0.00117
```

According to the logic of SAS, we can operate using a PROC only on the column(s) of a SAS dataset. So if we want to do a large number of simulation-iterations, e.g. to check the statistical distribution of the studentized average of 40 independent normally or exponentially distributed variables, we should generate the data in 40 columns and operate on the column entries within a data-step, and then process the result in a single PROC UNIVARIATE :

```
data student ;
  seedz = 314159 ; seedw = 271828 ;
  array zz[40] _TEMPORARY_ ;
  array ww[40] _TEMPORARY_ ;
  do r=1 to 1000 ;
    sum_z = 0 ; sum_zsq = 0 ;
    sum_w = 0 ; sum_wsq = 0 ;
    do i=1 to 40 ;
      zz[i] = rannor(seedz) ;
      ww[i] = ranexp(seedw)-1 ;
```

```
      sum_z = sum_z + zz[i] ;
      sum_zsq = sum_zsq + zz[i]*zz[i] ;
      sum_w = sum_w + ww[i] ;
      sum_wsq = sum_wsq + ww[i]*ww[i] ;
    end ;
    t_z = sum_z/sqrt(40)/sqrt((sum_zsq-sum_z*sum_z/40)/39);
    t_w = sum_w/sqrt(40)/sqrt((sum_wsq-sum_w*sum_w/40)/39) ;
    output ;
  end ;
title 'SIMULATED T FOR NORMAL AND EXPONENTIAL DATA';
proc univariate data=student;
  var t_z t_w ;
run;
```

We know that Student's t with 39 d.f. has a symmetric, bell-shapeddistribution when the underlying sample is normal. This distribution is not too far from Normal, and the Central Limit Theorem says that also for non-normal data, the studentized averages should be roughly normal. But how roughly ? We next display our (edited) simulated output.

The theoretical quantiles from 1% through 99% of the t_{39} distribution as displayed on the SAS output are

$$-2.426,\ -1.685,\ -1.304,\ -0.681,\ 0.000,\ 0.681,\ 1.304,\ 1.685,\ 2.426$$

and the theoretical mean, variance, skewness, and kurtosis are respectively 0, 1.0541, 0, 0.1905. The simulated values for normal input-data agree quite well with the values in the Student t tables. For the exponential data, we have a somewhat different story. The mean is close to zero because we have centered the exponential data-values, but the distribution of the studentized values is still skewed by comparison with a t_{39} distribution.

```
     SIMULATED T FOR NORMAL AND NONNORMAL DATA
Variable=t_z
                        Moments
N                 1000          Sum Weights               1000
Mean           0.00711654       Sum Observations    7.11653888
Std Deviation  1.05915437       Variance            1.12180798
Skewness       0.00522282       Kurtosis            0.35739736
```

```
                  Quantiles (Definition 5)
        100% Max    4.49569        99%         2.41896
         95%        1.73602        90%         1.34262
         75% Q3     0.72300        50% Median -0.00984
         25% Q1    -0.70482        10%        -1.32652
         5%        -1.73805        1%         -2.57299
         0% Min    -3.84347        Range  8.33916

Variable=t_w
                        Moments
N                  1000        Sum Weights               1000
Mean            -0.1882496     Sum Observations    -188.24957
Std Deviation    1.1091563     Variance            1.23022763
Skewness        -0.6548093     Kurtosis              1.201935

                     Quantiles(Def=5)
         100% Max   3.01113        99%         1.90066
          95%       1.44562        90%         1.15071
          75% Q3    0.57106        50% Median -0.07586
          25% Q1   -0.86782        10%        -1.65168
          5%       -2.15754         1%        -3.20561
          0% Min   -6.34213        Range       9.35325
```

We can display the discrepancies between the histograms in either SAS or Splus. The following code produces a high-quality scaled histogram in SAS.

```
 proc gchart data=student ;
title "Histograms for LOGBILI" ;
  vbar t_w / LEVELS=30 type=percent;
RUN ;
```

But as we see in the next subsection, it is very convenient in **Sp[lus** both to plot the histogram and an overlaid theoretical function.

12.2 Histograms and Overlaid Densities in Splus

Before proceeding to more complicated simulations in SAS and Splus, let us quickly re-capitulate the last graph through a simulation in Splus.

```
> motif()
> rmat <- matrix(rnorm(4e4), ncol=40)
  nrmt <- apply(rmat,1, function(mrow)
        sqrt(40)*mean(mrow)/sqrt(var(mrow)))
> round(quantile(nrmt, c(.01,.05,.1,.25,.5,.75,.9,.95,.99)),3)
    1%     5%    10%    25%    50%    75%    90%    95%    99%
-2.558 -1.789 -1.390 -0.727  0.022  0.737  1.367  1.697  2.356
> rmat <- matrix(rexp(4e4)-1, ncol=40)
  expt <- apply(rmat,1, function(mrow)
        sqrt(40)*mean(mrow)/sqrt(var(mrow)))
  round(quantile(expt, c(.01,.05,.1,.25,.5,.75,.9,.95,.99)),3)
    1%     5%    10%    25%    50%    75%    90%    95%    99%
-3.648 -2.302 -1.654 -0.781 -0.045  0.622  1.157  1.502  2.019
> par(mfrow=c(2,1))
> hist(nrmt, , nclass=40, prob=T)
> lines(seq(-3,3,.01), dt(seq(-3,3,.01),39), lty=3)
> hist(expt, nclass=40, prob=T)
> lines(seq(-3,3,.01), dt(seq(-3,3,.01),39), lty=3)
```

The histograms produced in the last few command-lines, summarizing the distribution of the simulated output values, are displayed in the following Figure. Note that in order to overlay a theoretical density on a plotted histogram, the option **prob=T** scaling the histogram like a probability density (total area in histogram bars equal to 1) must be chosen.

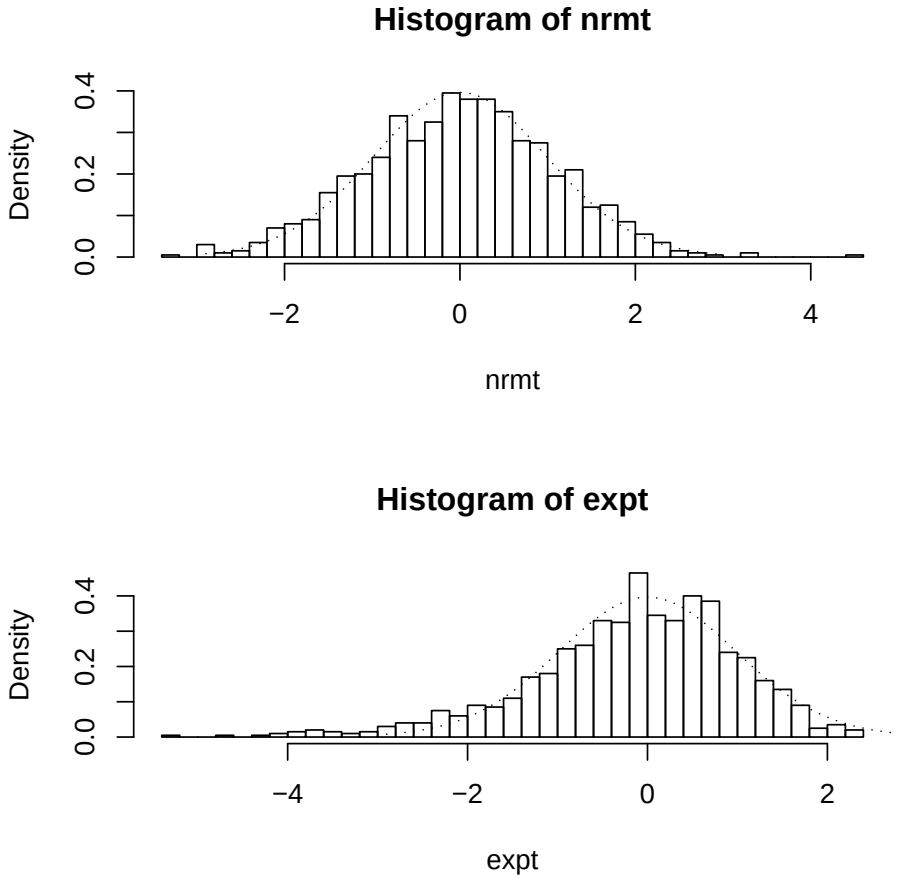

Figure 1: Scaled relative-frequency histograms of Splus-simulated t Values, for normal (**nrmt**) and centered-exponential (**expt**) data, n=40. Simulations each consisted of 1000 iterations. In each plotted histogram, the true t_{39} is overlaid as a dashed line.

12.3 More Elaborate Simulations

The key complication we address here, in comparing simulations in Splus versus SAS, is the very familiar possibility that each iteration of a large simulation may require some data-analysis step or model-fit which we do not want to program from scratch. For purposes of illustration, we simulate the standardized coefficient and associated t-distribution p-value for slope in a simple linear regression with non-normal (uniform) predictor and errors.

```
> BigReg <- array(0, dim=c(1000,20,2), dimnames=list(
       NULL,1:20,c("X","Y")))
  BigReg[,,1] <- runif(20000)
  BigReg[,,2] <- 0.3 + rnorm(20000)*.4
### NOTE that BigReg[,,2] is now UNrelated to BigReg[,,1]
> unix.time(slopvec <- apply(BigReg,1, function(smat)
         summary(lm(smat[,2] ~ smat[,1]))$coef[2,3]))
[1] 50.69  0.27 52.00  0.00  0.00
> summary(slopvec)
   Min. 1st Qu.   Median     Mean 3rd Qu.  Max.
 -3.618 -0.7371 -0.02168 -0.03306  0.7038 3.765
> pvals <- 1-pt(slopvec,18)  ## This is the one-sided p-value for
 ###the standardized slope-coefficient, which is t_{18} distributed
> summary (pvals)              #### Should be approx. Uniform[0,1]
      Min. 1st Qu.  Median      Mean  3rd Qu.     Max.
 0.000709 0.245281 0.513005 0.508529 0.764719 0.999017
## NB: the two-sided p-values would be 2*(1-pt(abs(slopvec),18))
```

So even this little simulation took nearly a minute of computer time in Splus!! We next do a larger version in order to test the limits of efficiency of Splus looping, to compare with a pure linear-algebra method, and to compare the timing with R.

```
> unix.time({BigReg <- array(0, dim=c(1000,500,2), dimnames=list(
       NULL,1:500,c("X","Y")))
  BigReg[,,1] <- runif(5e5)
  BigReg[,,2] <- 0.3 + rnorm(5e5)*.4
  slopvec <- apply(BigReg,1, function(smat)
```

```
         summary(lm(smat[,2] ~ smat[,1]))$coef[2,3])})
[1] 70.64  0.91 76.00  0.00  0.00
## Only 50% longer with batch-size 500 than with batch-size 20 !
> summary(slopvec)
   Min. 1st Qu.  Median    Mean 3rd Qu.  Max.
 -2.936 -0.6481 0.04752 0.03481   0.727 3.616
```

We have already mentioned that the method of 'parallel' calculation using **apply** is hardly better than directly coding a for-loop. But here, we can compare with the much more efficient method using linear algebra in a truly parallel way.

```
> unix.time({
 ###   BigReg <- array(0, dim=c(1000,500,2))
 ###   BigReg[,,1] <- runif(5e5)
 ###   BigReg[,,2] <- 0.3 + rnorm(5e5)*.4
  cnstvc <- rep(.002,500)
  Xsum <- c(BigReg[,,1] %*% cnstvc)
  XtX <- matrix(c(rep(1,1000), Xsum, Xsum, BigReg[,,1]^2 %*%
      cnstvc), ncol=4)   ## scaled down by factor n=500
  Ysum <- c(BigReg[,,2] %*% cnstvc)
  XYsum <- c((BigReg[,,2]*BigReg[,,1]) %*% cnstvc)
  Yvar <- (BigReg[,,2]^2 %*% cnstvc - Ysum^2)*(500/499)
  Xvar <- (XtX[,4]-Xsum^2)*(500/499)
  XYcor <- (500/499)*(XYsum-Xsum*Ysum)/sqrt(Yvar*Xvar)
  sigsq <- Yvar*(1-XYcor^2)
  dtvec <- Xvar*(499/500)    ## scaled down by 500^2
  slopvec2 <- c(XYsum - Ysum * Xsum)/sqrt(sigsq*Xvar/500)})
[1] 4.350006 0.200000 4.000000 0.000000 0.000000
```

Note the incredible difference in speed: the factor is > 15.

When I ran exactly the same programs in R (on my same Sun-terminal), the last (linear-algebra) method took 6.07 seconds machine-time. But the first method (calculating *slopvec* using **apply**) took 104 seconds !! So to a first approximation, looping in R is no better than in Splus. Recall that although the syntax of the two languages is the same, their internal; workings are programmed differently.

Now we turn to SAS to attempt to reproduce the last simulation:

```
data sampreg (keep= xx yy);
    seed = 4055067 ;
  do i=1 to 30 ;
    xx = ranuni(seed) ;
    yy = -1.2 + 0.6*xx + 0.5*rannor(seed);
    output ;
  end ;

PROC REG  outest=regests;
  model yy = xx;
run;
The data set WORK.REGESTS has 1 observations and 7 variables.

Obs _MODEL_  _TYPE_  _DEPVAR_  _RMSE_  Intercept  xx     yy
1   MODEL1   PARMS     yy      0.48755 -1.42935  0.78730 -1
```

The -1 value for yy is just an indication that it is the dependent variable. The list of output statistic values can be very much expanded: for example, by issuing the TABLEOUT option, we get standard errors of estimates, t values, etc. But now the output file REGESTS would have 6 records instead of one, respectively with the TYPE variable equal to PARMS, STDERR, T, PVALUE, L95B, U95B.

The next step is to try to produce an output file along the same lines to contain estimated quantities from each of a number of simulation iterations.

```
data sampreg (keep = xx yy iter);
    seed = 401067 ;
  do iter = 1 to 100;
  do i=1 to 30 ;
    xx = ranuni(seed) ;
    yy = -1.2 + 0.6*xx + 0.5*rannor(seed);
    output ;
  end ;
 end;
run;
```

```
PROC SORT ;
  by iter;

PROC REG  outest=regests2 TABLEOUT NOPRINT;
  model yy = xx;
  BY iter;
run;
```

Note that the SAMPREG dataset used in this computation now had 3000 record-lines. The NOPRINT option is needed to generate the OUTEST file without a lot of needless output printed to the OUTPUT window. Messages about analysis by BY-group were issued for all 100 iterations; the REG step now took 3.23 seconds of real time and 0.34 seconds of CPU time, and the output data file REGEST2 has 600 records and 8 variables. (The variables are the seven from before plus ITER (because the is the BY-group variable), and we have 6 records of different TYPE for each BY-group value ITER= $1, \ldots, 100$. It remains to re-process the last data-file into a summary. First we need a data-step which grabs just the required fields (xx in records of TYPE T and TYPE PVALUE).

```
data slopvals (keep = Tval Pval iter);
  RETAIN Tval;
  set REGESTS2 (keep = iter _TYPE_ xx);
  if _TYPE_ EQ "T" then Tval = xx;
  if _TYPE_ EQ "PVALUE" then do;
      Pval = xx; output; end;
run;
PROC PRINT;
  where iter < 3;
run;
...
       Obs      Tval     iter      Pval
        1    1.77870      1     0.086149
        2    2.80599      2     0.009024
```

The RETAIN statement is needed here in order that SAS not re-initialize *Tval* to 'missing' each time it reads a new line.

We conclude this subsection by doing a timing run in SAS for the same (1000 iterations of simple linear regression with 20 observations) simulation which was done in 6 seconds in **R**, and 4.5 seconds in **Splus3.4** using *apply* and *lm*.

```
data sampreg (keep = xx yy iter);
    seed = 401067 ;
  do iter = 1 to 1000;
    do i=1 to 20 ;
      xx = ranuni(seed) ;
      yy = -1.2 + 0.5*rannor(seed);
      output ;
    end ;
  end;
     run;
PROC SORT ;
  by iter;
PROC REG  outest=regests3 TABLEOUT NOPRINT;
  model yy = xx;
  BY iter;
data slopvals (keep = Tval Pval iter);
  RETAIN Tval;
  set REGESTS3 (keep = iter _TYPE_ xx);
  if _TYPE_ EQ "T" then Tval = xx;
  if _TYPE_ EQ "PVALUE" then do;
      Pval = xx; output; end;
PROC UNIVARIATE;
  Var Tval Pval;
run;
```

The total CPU and real times spent in this simulation run are respectively 0.21 and 24.74 seconds, on a detective-cluster machine. When I tried to run the same simulation in SAS with 500 observations in each simple linear regression, I ran out of SAS resources. When I tried to do the same thing in 10 separate chunks: the average time was 1.86 seconds for each chunk, or around 18.6 seconds in all. So the computations do not take particularly long !