REPRODUCIBLE ECONOMETRIC RESEARCH

ROGER KOENKER

ABSTRACT. These notes are an informal, first installment in an ongoing project to develop a convenient template for computational experimentation in econometrics. The approach is illustrated by means of an example based on some current research with Steve Portnoy on improving the speed of quantile regression algorithms. The computations are carried out in SPLUS, but similar techniques could be adapted for any modern computing environment designed for statistical applications. The objective is to provide a reasonably automatic, almost painless, way to make experimental results *self-documenting and reproducible*. With minor modifications the same approach could be adapted to empirical applications.

1. INTRODUCTION

A common problem in econometric research may be characterized as follows: we have several methods to do something, some criteria for performance evaluation, and some circumstances/models under which would like to compare performance of the methods. Estimation and testing are obvious applications in which asymptotic theory may provide some guidance for performance evaluation, but one often wishes to explore finite sample performance via Monte-Carlo experimentation if only as a way to validate the relevance of the theory for empirical applications.

Unfortunately, such experiments are often carried out over a long period in which hardware and software may evolve and memory affords an imperfect index to the welter of data files that remain behind. In such circumstances it is common to insert a table or a figure into a paper-in-progress and later find that it is difficult or even impossible to recreate the process which produced the object in the first instance.

In view of the difficulties of reproducing our own work, it is hardly surprising that *others* have difficulties reproducing published work in econometrics. It has been an established part of the folklore in applied econometrics that even ordinary regression results are rarely reproducible and as statistical methods become more sophisticated this has not improved. In the following pages we attempt to assemble some tools and strategies designed to counter this tendency. The methods are illustrated by their application to some recent work on improving algorithms for quantile regression. Section 2 introduces some SPLUS ideas for improving the documentation and organization of simulation results. Section 3 considers some aspects of dealing with FORTRAN components of simulation experiments. Section 4 discusses some tools for organizing SPLUS graphics and introducing them into LaTeX documents. Section 5 deals with packaging research results and making them accessible over the internet.

As with all aspects of S and SPLUS, we rely heavily on Becker, Chambers, and Wilks (1988) Spector (1994) and Venables, and Ripley (1994) for guidance on the language and its use. Throughout, we will also often rely on code contributed to STATLIB and via SNEWS to the SPLUS community. These repositories constitute a rich source of enhancements for the efficient use of the language.

Date: June 8, 2006. This paper was prepared using hardware and software supported by NSF Grants SBR 93-20555 and SBR 95-12440. The latter grant served to initiatate the Econometrics Lab at UIUC. Further details on the Lab and related research activities may be found at the URL `http://www.econ.uiuc.edu`.

2. Reproducible Simulation

The following code illustrates a function to conduct a monte-carlo experiment designed to compare the computational efficiency of several algorithms for l_1 (median) regression.

```
"monte"<-
function(run, ns = c(20000, 40000, 80000, 120000), ps = c(4, 8, 16), R = 5,
    methods = expression(lm(y ~ x), l1fit(x, y), rqfn(x, y), RQFN(x, y)),
    dfx = expression(matrix(rnorm(p * n), n, p)), dfy = expression(rnorm(n)
    ), mse = list(2, c(3, 4)))
{
    version <- 5    #function for timing experiments for rqn paper
#Input:
#    ns-a vector of sample sizes
#    ps-a vector of parameter dimensions, intercept will be appended
#    R -number of replications of each n,p pair
#    methods-methods to be compared should be of the form:
#        expression(lm(y~x),l1fit(x,y),rqfn(x,y),RQFN(x,y))
#        this is a list which can be evaluated as eval(methods[[i]])
#    dfx-expression to generate design matrix
#    dfy-expression to generate response vectors
#    mse-list describing how to evaluate accuracy:
#        1.  benchmark method (number in methods list)
#        2.  new methods  under test (numbers in methods list)
#
#Output:
#    result-data structure with the components
#        times-array of timings
#        err  -root mse of bhat for new methods vis a vis benchmark
#        seed -initial .Random.seed
#    doc-attribute of result describing in detail how it was created
#
    options(object.size = 150000000)
    #checking for dynloading now occurs in the rq functions
#do the biggest problem first in case there are memory problems
    ns <- rev(sort(ns))
    ps <- rev(sort(ps))
    times <- array(0, c(length(methods), R, length(ps), length(ns)))
    err <- array(0, c(length(mse[[2]]), R, length(ps), length(ns)))
    seed <- .Random.seed
    for(i in 1:length(ns)) {
        n <- ns[i]
        print(paste("n=", n))
        for(j in 1:length(ps)) {
            p <- ps[j]
            print(paste("p=", p))
            b <- matrix(0, p + 1, length(methods))
            x <- eval(dfx)
            m <- x %*% rep(1, p)
            for(k in 1:R) {
```

```
                y <- m + eval(dfy)
                for(l in 1:length(methods)) {
                  times[l, k, j, i] <- unix.time(b[, l] <- eval(
                    methods[[l]])$coef)[1]
                }
                err[, k, j, i] <- sqrt(apply((b[, mse[[2]]] - b[
                  , mse[[1]]])^2, 2, "mean"))
            }
        }
    }
    dimnames(times) <- list(paste(methods), NULL, paste("p=", ps, sep = ""),
        paste("n=", ns, sep = ""))
    result <- list(times = times, err = err, seed = seed, dfx = dfx, dfy =
        dfy)
    doc(result) <- how.created(paste("Test", run, "on", unix("hostname")),
        text = F)
    return(result)
}
```

Notice first that we have 4 default methods to compare: one is the function `l1fit` provided by S, two are new functions which we will describe in further detail below, `rqfn, RQFN`, and the fourth is the standard least squares function `lm(y ~ x)` which will serve as a benchmark to evaluate the performance of the l_1 algorithms. The use of the function

```
methods_expression()
```

provides a convenient general way to introduce the methods in the form of a list through which we may loop, using the function

```
eval(methods[[i]]).
```

Similarly we introduce methods for generating the data as illustrated by the specifications of `dfx` and `dfy` in the function calling sequence. Specification of the sample sizes and parametric dimension of the model are introduced simply as vectors, again to facilitate looping.

The function begins by organizing the work to be done so that the most challenging problems, the largest ones in this case, come first. This way if the simulation fails due to memory constraints, for example, it is likely to do so immediately. An array is then initialized for the results of the experiment and the current value of `.Random.seed` is stored for future reference. The storage of `.Random.seed` is critical to reproducibility. By simply reassigning `.Random.seed` at any future point to the value `seed` and reexecuting the function `monte` we may reproduce the precise results of experiment. Of course this claim must be qualified somewhat if the experiment is conducted on different hardware, but the portability of the SPLUS random number generator assures that sequence of random numbers generated will be the same up to machine precision even across machines. Thus `seed` will be a crucial component in what is returned by the function `monte.`

The next few lines loop through the configurations of the experiment with the timing result, the first component (user-time) of `unix.time`, in the array `times`. Since looping is notoriously slow in SPLUS it is worth pausing to comment briefly on efficiency considerations for the experiment at this point. It may be noted that the number of replications of the experiment is small, 5 according to the default set in the calling experiment. This is probably atypical of most experiments in econometrics. Here we are doing a small number of very large problems, while it would be more typical to do a large number of much smaller problems. In the latter case it would be *highly* desirable to incorporate

the replications into a single call, a lower level FORTRAN or C call, passing, for example, a matrix of y's to each element of the `methods`. This is unnecessary in the present instance and probably infeasible as well due to memory constraints. We will have more to say about the issue of looping in the next section. It is also important in the present application to evaluate the accuracy of the solutions computed by the new methods `rqrn` and `RQFN`. This is done in the array `err` which reports root mean squared errors for these solutions relative to the answer provided by `l1fit`.

Once the array `times` has been filled we can assign dimension labels to it. This is facilitated by the form of the vectors `ns`, `ps` and the list `methods` using the `paste` function. Then the result of the experiment is packaged as a list consisting of the components.

- the array `times`
- the array `err`
- the `seed`
- `dfx` describing how x was generated
- `dfy` describing how y was generated

Finally, we conclude the call to `monte` by creating a documentation attribute for the list `result` using the assignment,

```
doc(result)←how.created(paste(...),text=F)
```

This has the effect of appending several additional pieces of information to the outcome of the experiment which serve to identify precisely how it was created. This is illustrated in the next display. A typical call to monte might look like,

```
m.5_mont(5)
```

Typically, we wouldn't want to execute this interactively, so it would be reasonable to put this command in a file, say `mc.s` and at the system prompt, type

```
SPLUS <mc.s <& mc.o &
```

which begins an SPLUS process which is run in the background. It takes input from `mc.s` and put output, including diagnostic output, into the file `mc.o`. Since SPLUS doesn't commit assignments until it concludes successfully, one way to monitor progress of a job of this type is to put print statements of the form

```
print(paste("k=",k))
```

in the loop construct, which will allow the user to check the output file `mc.o` periodically to see where in the k-loop the job has arrived. The output `m.5` looks like this:

```
$times:

, , p=16, n=120000
                    [,1]       [,2]       [,3]       [,4]       [,5]
  lm(y ~ x)   38.37000   38.13965   38.18066   37.94043   38.05859
l1fit(x, y) 4564.16016 4742.39990 4561.49023 4574.25000 4235.11133
 rqfn(x, y)  102.37988  103.21973  103.92969   98.58008   98.35938
 RQFN(x, y)   51.75977   52.31934   55.91016   51.26953   55.97070

, , p=8, n=120000
                    [,1]       [,2]       [,3]       [,4]       [,5]
  lm(y ~ x)   22.48828   22.38086   22.11914   22.17969   21.95312
l1fit(x, y) 2768.43945 2645.08008 2488.46094 3182.05078 2590.62891
 rqfn(x, y)   67.54883   71.39062   69.29883   63.49219   66.94141
 RQFN(x, y)   31.45898   20.36914   29.95117   27.82812   29.07812
```

```
, , p=4, n=120000
                     [,1]       [,2]       [,3]       [,4]       [,5]
  lm(y ~ x)   16.10938   16.33984   15.95703   15.92188   15.91797
l1fit(x, y) 1607.01953 1761.36719 1939.10156 1376.69922 1897.66797
 rqfn(x, y)   55.43750   51.75781   55.07031   55.03906   55.53906
 RQFN(x, y)   19.03125   13.19922   13.03906   13.21875   21.17188

, , p=16, n=80000
                     [,1]       [,2]       [,3]       [,4]       [,5]
  lm(y ~ x)   24.91797   24.78125   25.00000   24.69922   24.71094
l1fit(x, y) 2079.09766 1979.44141 1974.65234 1917.89062 2106.11328
 rqfn(x, y)   61.44922   65.53906   68.65234   65.08984   65.37891
 RQFN(x, y)   37.39062   38.90234   37.66016   40.45312   25.01953

, , p=8, n=80000
                     [,1]       [,2]       [,3]       [,4]       [,5]
  lm(y ~ x)   14.98047   14.85156   14.64844   14.58984   14.51172
l1fit(x, y) 1208.23047 1235.46875 1265.23047 1265.78125 1303.05859
 rqfn(x, y)   42.51953   42.26172   42.12891   44.66016   44.67188
 RQFN(x, y)   21.81250   13.91016   20.39062   20.82031   21.41016

, , p=4, n=80000
                    [,1]      [,2]      [,3]      [,4]      [,5]
  lm(y ~ x)  10.78906  10.76172  10.68750  10.49219  10.46875
l1fit(x, y) 692.56641 838.37109 826.60938 806.97656 782.32812
 rqfn(x, y)  34.25000  38.95312  34.21094  34.21094  31.96875
 RQFN(x, y)  10.10938  10.72656  12.39062  22.81250  13.17969

, , p=16, n=40000
                    [,1]      [,2]      [,3]      [,4]      [,5]
  lm(y ~ x)  12.22656  12.14844  12.03906  12.07812  12.05469
l1fit(x, y) 500.25000 487.10156 540.10156 442.32031 513.26562
 rqfn(x, y)  30.97656  36.21094  30.70312  30.70312  32.40625
 RQFN(x, y)  19.85156  20.50000  20.21875  19.45312  19.85156

, , p=8, n=40000
                    [,1]       [,2]       [,3]      [,4]       [,5]
  lm(y ~ x)   7.15625   7.140625   7.148438   7.12500   7.132812
l1fit(x, y) 282.28906 286.421875 302.765625 274.36719 322.515625
 rqfn(x, y)  22.01562  20.804688  22.031250  19.57812  21.710938
 RQFN(x, y)   7.71875   9.859375  10.664062  11.55469   7.820312

, , p=4, n=40000
                    [,1]       [,2]       [,3]       [,4]       [,5]
  lm(y ~ x)   5.18750   5.265625   5.164062   5.164062   5.117188
l1fit(x, y) 171.76562 221.078125 206.242188 188.523438 165.507812
 rqfn(x, y)  15.52344  16.578125  19.945312  16.734375  17.664062
```

```
 RQFN(x, y)   7.40625   6.593750   5.453125   6.664062   6.984375

, , p=16, n=20000
                 [,1]       [,2]       [,3]       [,4]      [,5]
  lm(y ~ x)   5.859375   5.953125   5.890625   5.851562   5.87500
l1fit(x, y) 136.562500 136.578125 138.351562 147.945312 129.01562
 rqfn(x, y)  15.015625  14.140625  15.867188  14.187500  16.94531
 RQFN(x, y)  10.507812  11.468750   7.171875  10.570312  10.92969

, , p=8, n=20000
                [,1]      [,2]      [,3]      [,4]      [,5]
  lm(y ~ x)  3.507812  3.515625  3.515625  3.523438  3.523438
l1fit(x, y) 84.078125 77.656250 72.671875 83.773438 69.906250
 rqfn(x, y)  9.875000  9.281250  9.265625  9.296875 10.007812
 RQFN(x, y)  5.789062  6.359375  6.773438  5.757812  5.945312

, , p=4, n=20000
                [,1]      [,2]      [,3]      [,4]      [,5]
  lm(y ~ x)  2.570312  2.554688  2.593750  2.593750  2.585938
l1fit(x, y) 47.195312 44.343750 45.914062 54.007812 44.820312
 rqfn(x, y)  7.835938  8.257812  7.289062  7.812500  7.351562
 RQFN(x, y)  2.835938  4.085938  4.328125  2.648438  3.921875

$err:

, , 1, 1
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 3.414232e-05 1.566616e-05 4.964434e-06 4.395257e-05 4.359341e-05
[2,] 3.414232e-05 1.566619e-05 4.964458e-06 4.395101e-05 4.359344e-05

, , 2, 1
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 8.338684e-06 8.136083e-07 1.523099e-05 1.507634e-05 1.681484e-05
[2,] 8.338627e-06 8.135856e-07 1.523034e-05 1.507631e-05 1.681402e-05

, , 3, 1
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 2.216078e-05 3.553101e-06 1.787060e-07 1.443290e-07 1.174932e-07
[2,] 2.215855e-05 3.553141e-06 1.787061e-07 1.443649e-07 1.175176e-07

, , 1, 2
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 5.675161e-06 2.087732e-05 3.657927e-05 7.909246e-06 2.421033e-05
[2,] 5.675195e-06 2.087763e-05 3.657925e-05 7.909236e-06 2.420965e-05

, , 2, 2
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 1.085759e-06 2.920666e-06 6.352805e-06 5.974413e-06 7.780196e-06
```

```
[2,] 1.085678e-06 2.927971e-06 6.352803e-06 5.974751e-06 7.780130e-06

, , 3, 2
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 2.024797e-07 1.828511e-07 1.799722e-07 3.088179e-06 2.973498e-07
[2,] 2.074590e-07 1.892743e-07 1.799714e-07 3.088172e-06 2.973314e-07

, , 1, 3
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 5.652814e-07 1.100742e-05 3.460726e-05 1.387323e-05 4.565225e-07
[2,] 5.642479e-07 1.100742e-05 3.460725e-05 1.387323e-05 4.556660e-07

, , 2, 3
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 2.654629e-07 2.151161e-07 3.984395e-05 2.589425e-05 1.905348e-05
[2,] 2.654589e-07 2.149446e-07 3.984391e-05 2.583213e-05 1.905358e-05

, , 3, 3
             [,1]       [,2]         [,3]         [,4]         [,5]
[1,] 6.891924e-08 3.6193e-05 5.676117e-08 1.431328e-07 1.365530e-06
[2,] 6.865607e-08 3.6193e-05 5.648426e-08 1.432799e-07 1.365537e-06

, , 1, 4
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 1.334859e-06 8.979264e-07 2.673230e-06 8.003513e-07 2.444837e-06
[2,] 1.334141e-06 8.978599e-07 2.673231e-06 8.004130e-07 1.692292e-06

, , 2, 4
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 3.549152e-07 1.582878e-07 9.958417e-06 4.616344e-07 4.836977e-07
[2,] 3.549160e-07 1.582885e-07 9.954472e-06 4.614458e-07 4.702986e-07

, , 3, 4
             [,1]         [,2]         [,3]         [,4]         [,5]
[1,] 3.603852e-07 1.116334e-07 4.817329e-08 2.730610e-07 1.963181e-07
[2,] 3.598513e-07 1.115358e-07 4.821597e-08 2.824074e-07 1.964258e-07

$seed:
 [1] 21 61 59  7 40  2 33 62 52 13 55  3

$dfx:
expression(matrix(rnorm(p * n), n, p))

$dfy:
expression(rnorm(n))

attr(, "doc"):
attr(, "doc")$what:
```

```
attr(, "doc")$what$comment:
[1] "Test 5 on ragnar.econ.uiuc.edu"

attr(, "doc")$what$call:
monte(5)

attr(, "doc")$what$version:
[1] 5

attr(, "doc")$what$env:
[1] ".Data"
[2] "/usr/local/splus/splus/.Functions"
[3] "/usr/local/splus/stat/.Functions"
[4] "/usr/local/splus/s/.Functions"
[5] "/usr/local/splus/s/.Datasets"
[6] "/usr/local/splus/stat/.Datasets"
[7] "/usr/local/splus/splus/.Datasets"
[8] "/usr/local/splus/library/local/.Data"
[9] "/usr/local/splus/library/local/.Datasets"


attr(, "doc")$when:
[1] "Sun Sep  8 20:12:03 CDT 1996"

attr(, "doc")$who:
[1] "roger"
```

Note that in addition to the main components `times, err, seed, dfx, dfy` of `m.3` we have 4 documentation components:

- a comment indicating that the experiment was conducted on the machine `ragnar.econ.uiuc.edu`
- a "call" component indicating the calling sequence used
- a "version" component indicating the version numbers, if any, of the function which involved the document creation
- the search list at the time of creation
- the time of invocation
- the user who invoked it

Together the components of `result` assigned to `m.5` constitute a detailed description of how the experiment was conducted and how it would be reproduced. The functions used to create the documentation attribute are not part of SPLUS 3.3 but were contributed by Jeff Marcus and developed further by Z. Todd Taylor. They *are* accessible from `splus.local` on `ragnar.econ.uiuc.edu`, however, as are a significant number of other bells and whistles for SPLUS.

3. Linking fortran to splus

The natural question which arises immediately is, "why?" What could motivate us to return to the paleolithic land of GOTO's and DO-loops when SPLUS offers such a convenient environment already. The answer, in a word, is efficiency. It becomes painfully apparent after only a little

experience with SPLUS in simulation work, that unless the inner loops of computations are coded in some lower level language like FORTRAN or C, SPLUS can be very slow. As a rule of thumb, if you have a loop which needs to be executed 1000's of times you should begin to consider FORTRANizing it, for your own benefit, and for the welfare of other users who share the machine you happen to compute on.

Fortunately, adding FORTRAN (or C, but I will not delve into this) is quite easy in S. I will briefly describe some tools which I find helpful in the process, and some general rules that I try to follow when venturing into this land of the dinosaurs.

3.1. **Is this really necessary?** One should be sure that it is really "worth it" before embarking on a FORTRAN project. I like to have an affirmative answer to each of the following questions:

- Is this going to really save time in some appropriately discounted sense? Surprisingly, the answer to this is often, yes, since the function can easily speed things up by several orders of magnitude, base 10.
- Is this a function I'll use a year from now? This is a more stringent test.
- Am I sure that there isn't a better way to write this in S? This can only be answered definitively after considerable experience with the language, but many useful hints are available in the books by Venables and Ripley (1996) and Spector(1994) as well as in SNEWS and other places.

Once you are convinced that this is a necessary evil you should muster all the available tools. Many would say that FORTRAN, like Latin, is a dead language and puts one at an immediate disadvantage. This is certainly true for some applications, particularly those involving a serious graphics component or character string manipulation. But for purely numerical applications FORTRAN continues to serve quite well. I prefer the FORTRAN dialect Ratfor, developed at Bell Labs by Kernighan and Plauger(1976). It provides much of the syntactical structure of C, but represents only a modest investment – the entire literature on learning the dialect is the classic 25 page tutorial written by Kernighan which is a model of clarity. We will not pretend to elaborate on how to write Ratfor; we simply illustrate the language through an example of Ratfor code.

```
#This is a ratfor implementation of the floyd-revest quantile algorithm--SELECT
#Reference:  CACM 1975, alg #489, p173, algol-68 version
#As originally proposed:  mmax=600, and cs=cd=.5
#Translation by Roger Koenker August, 1996.
#Calls blas routine dswap
subroutine select(n,x,l,r,k,mmax,cs,cd)
integer n,m,l,r,k,ll,rr,i,j,mmax
double precision  x(n),z,s,d,t,cs,cd
while(r>l){
    if(r-l>mmax){
        m=r-l+1
        i=k-l+1
        fm=dfloat(m)
        z=log(fm)
        s=cs*exp(2*z/3)
        d=cd*sqrt(z*s*(m-s)/fm)*sign(1.,i-m/2)
        ll=max(l,k-i*s/fm +d)
        rr=min(r,k+(m-i)*s/fm +d)
        call select(n,x,ll,rr,k,mmax,cs,cd)
        }
```

```
    t=x(k)
    i=l
    j=r
    call dswap(1,x(l),1,x(k),1)
    if(x(r)>t)call dswap(1,x(r),1,x(l),1)
    while(i<j){
        call dswap(1,x(i),1,x(j),1)
        i=i+1
        j=j-1
        while(x(i)<t)i=i+1
        while(x(j)>t)j=j-1
        }
    if(x(l)==t)
        call dswap(1,x(l),1,x(j),1)
    else{
        j=j+1
        call dswap(1,x(j),1,x(r),1)
        }
    if(j<=k)l=j+1
    if(k<=j)r=j-1
    }
return
end
```

This is a pathbreaking algorithm development by Floyd and Rivest (1975) which computes the ordinary sample quantiles in an asymptotically linear number of comparisons in the sample size. The algorithm, as displayed, is a straightforward translation from the Algol given in the original paper. Note the recursive call.

Given the FORTRAN, how is it incorporated into SPLUS? This is inevitably system dependent; we will focus the discussion on the local environment on `ragnar`. To illustrate this we provide a simple function which calls `select` in order to compute a sample quantile.

```
"kuantile"<-
function(x, p = 0.5, mmax = 600, cs = 0.5, cd = 0.5)
{
    if(!is.loaded(symbol.For("kuantile")))
        dyn.load("src/rqfn/fn.o")
    n <- length(x)
    if(p < 0 | p > 1)
        stop("p outside [0,1]")
    z <- .Fortran("kuantile",
        as.integer(n),
        as.double(x),
        p = as.double(p),
        q = double(1),
        as.integer(mmax),
        as.double(cs),
        as.double(cs))
    return(z$q)
```

```
}
```

This function calls the following ratfor function

```
#function to compute pth quantile of a sample of n observations
subroutine kuantile(n,x,p,q,mmax,cs,cd)
integer n,k,l,r,mmax
double precision  x(n),p,q,cs,cd
if(p<0 | p>1) {call dblepr("sparsity bandwidth problem: p=",30,p,1);return}
l=1
r=n
k=nint(p*n)
call select(n,x,l,r,k,mmax,cs,cd)
q=x(k)
return
end
```

which in turns calls `select`. Note that the calling sequence requires the character name of the function to be the first argument, the remaining arguments are just as they appear in the FORTRAN. Debugging often requires us to print intermediate results from the FORTRAN, this is somewhat idiosyncratic in current versions of SPLUS, but can be accomplished with the calls

```
call intpr ("name",4,ivar,n)
call realpr ("rname",5,rvar,p)
call dblepr ("dname",5,dvar,p)
```

where the character string provides an identifying label, and the final integer argument specifies the number of elements of the variable we desire to print. See the ratfor code above for an example of the use of these calls.

In situations in which there are only one or two subroutines required for a FORTRAN function we may produce an object module corresponding to the source file `f.r` by invoking the command

```
f77 -c f.r
```

This produces a file `f.o` which can be, in most Unix systems at least, dynamically loaded into SPLUS with the command

```
dyn.load ("f.o")
```

In more complicated situations with many subroutines it is convenient to have a makefile to automate the compilation process. Now we illustrate the makefile for the function `rqn` which underlies the `rqfn` and `RQFN` functions mentioned above.

```
#This is a Makefile for the new frisch-newton RQ routine
#The compile flags are intended to optimize for ragnar
CFLAGS = -c -xarch=v8 -xchip=super2 -O4
LFLAGS = -r -dn /usr/local/SUNWspro/SC4.0/lib/v8/libsunperf.a

fn.o: fnc.o glob.o sparsity.o
ld fnc.o glob.o sparsity.o $(LFLAGS) -o fn.o
fnc.o: fnc.r
f77 $(CFLAGS) fnc.r
glob.o: glob.r
f77 $(CFLAGS) glob.r
sparsity.o: sparsity.r
```

```
f77 $(CFLAGS) sparsity.r
clean:
rm fnc.o fn.o
```

Invoking `make` induces an evaluation of what elements of the code require recomplilation, and in the load step of the present example links several `.o` files together into one dynamically loadable module.

Note also in this makefile that the flags for the compile step are chosen to optimize performance for the Sparc 20 architecture of `ragnar`. In the load step the library `libsunperf.a` is referenced, enabling access to a broad array of linear algebra routines from LAPACK and elsewhere which are again tuned for good performance on the Sparc 20. See the files in the `ragnar` directory `/usr/local/SUNWspro/READMEs` for further details on this library.

The use of the library is an excuse to emphasize the obvious, but often overlooked, fact that it is always desirable to seek out and use well-established library routines rather than risk reinventing them yourself. This is particularly true of basic linear algebra subroutines which have reached an extremely refined state of development in the `BLAS` provided by `LAPACK`.

4. Graphics

Splus provides an extremely sophisticated graphics environment which has recently been extended in important directions by the incorporation of trellis graphics, offering a general approach to the idea of "small multiples"advocated by Tufte (1992). It would be foolish to entertain a general introduction to this topic, the reader might wich to consult Cleveland (1993) which is itself a good example of reproducible research, all of the data and `S` code required to produce the figures in the book are easily available over the internet. We will only illustrate some basic ideas related to reproducibility of graphics and their incorporation into LaTeX documents.

A few general principles seem obvious:

- Try to avoid serious computation in graphics applications; isolate computationally intensive aspects into a preliminary stage in which experimental results are generated, leaving the graphics phase to concentrate on effective presentation of the results.
- Each final figure in a manuscript deserves a source file which fully describes how it was created, and therefore permits the user to recreate, or modify the figure easily.
- Final figures should be generated in postscript format so that they can be efficiently incorporated into LaTeX at the optimal available resolution.

It may appear too doctrinaire to insist on LaTeX formatting of research papers. Of course, there are many other possible text processing environments, and I will confess to resisting the transition from Troff to TeX for several years after it appeared inevitable. Other environments may be fine for literary adventures in economics, but only TeX, and preferably LaTeX is suitable for the stringent demands of serious econometrics. This document was prepared with the aid of the `amsart` formatting style described in Goossens, Mittelbach and Samarin (1994).

The following splus commands illustrate a source file used to generate Figure 4.1. `latex.table` in the following way,

```
#this is a plot to compare lm with several l1 timings
#output from m.4 and m.5 is structured in the following way:
#m.[56]$times is an array with dimensions 4 x 5  x 3 x 4
postscript("fig.4.1.ps",horizontal=F,font=7,pointsize=10,width=6.5,height=4.5)
par(mfrow=c(1,3))
```

```
ps_c(16,8,4)
for(i in 3:1){
    m.5.m_apply(m.5$times,c(1,3,4),"median")
    m.6.m_apply(m.6$times,c(1,3,4),"median")
    z_c(20,40,80,120)*100
    z_c(z,z*10)
    plot(z,c(rev(m.6.m[2,i,]),rev(m.5.m[2,i,])),type="l",
        log="xy",xlab="n",ylab="seconds")
    lines(z,c(rev(m.6.m[1,i,]),rev(m.5.m[1,i,])),lty=2)
    lines(z,c(rev(m.6.m[3,i,]),rev(m.5.m[3,i,])),lty=3)
    lines(z,c(rev(m.6.m[4,i,]),rev(m.5.m[4,i,])),lty=4)
    legend(4000,500,c("l1fit","lm","rqfn","RQFN"),lty=1:4)
    title(paste("p=",ps[i]))
    }
frame()
```

This was accomplished simply by the command

```
source("fig.4.1.s")
```

in SPLUS, which produced the file `fig.4.1.ps` which was then incorporated into the present document with the following LaTeX convention.

```
\begin{figure}[hbt]
\begin{center}
{\includegraphics{fig.4.1.ps}}
\begin{caption}
{Timing comparison of three algorithms  for median regression:
Times are in seconds for the median of five replications for iid Gaussian
data.  The solid line represents the timings for the simplex-based
Barrodale and Roberts algorithm, the {\tt rqfn} dashed line represents
a primal-dual interior point algorithm, {\tt RQFN} uses a preprocessing
step and the {\tt rqfn} algorithm, and the dotted line represents least
squares timing based on ${\tt lm(x \sim y)}$ as a benchmark}
\end{caption}
\end{center}
\end{figure}
```

Note that this approach allows one to modify the figure by rerunning the SPLUS code without having to modify the LaTeX document. Of course, other device drivers, notably `motif()` provide an indispensible service in preliminary, exploratory stages of preparing graphics. We might also note that the inclusion of code into LaTeX documents is facilitated by the use of the verbatim environoment in LaTeX and the package `alltt` written by Lamport(1992). The S function `S_to_tex` written by John Chambers and available from `Statlib` is a useful translation device in preparing code for LaTeX inclusion.

5. Tables

Tables are often a necessary evil and again SPLUS, augmented by some independently written functions provides a good set of tools. Consider the relatively simple problem of making the experimental output `m.3` described above into a table in which each group of R timings is represented by its median. In SPLUS the array of medians is easily computed as for the previous figures,

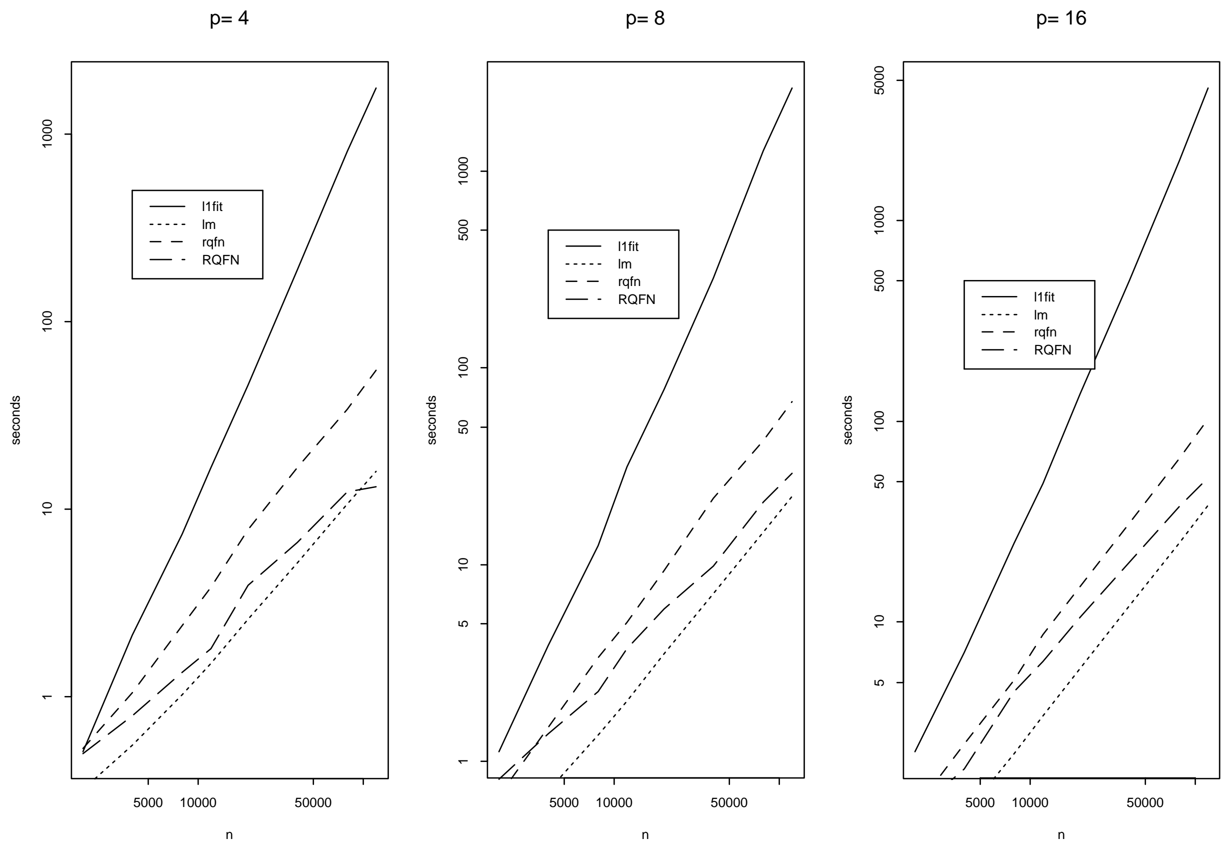

FIGURE 1. Timing comparison of three algorithms for median regression: Times are in seconds for the median of five replications for iid Gaussian data. The solid line represents the timings for the simplex-based Barrodale and Roberts algorithm, the `rqfn` dashed line represents a primal-dual interior point algorithm, `RQFN` uses a preprocessing step and the `rqfn` algorithm, and the dotted line represents least squares timing based on `lm(x ~ y)` as a benchmark

```
m.3.m_apply(m.3, c(1,3,4),"median")
```

To make this into the table appearing below, we use Alan Zalovsky's function `latex.table` in the following way,

```
#output from m.4 and m.5 is structured in the following way:
#m.[56]$times is an array with dimensions 4 x 5  x 3 x 4
methods_c("lm","RQFN","rqfn","l1fit")
ps_c(4,8,16)
z_c(20,40,80,120)*100
z_c(z,z*10)
m.5.m_apply(m.5$times,c(1,3,4),"median")
m.6.m_apply(m.6$times,c(1,3,4),"median")
#reorder elements of these arrays
```

```
m.5.m_m.5.m[c(1,4,3,2),3:1,4:1]
m.6.m_m.6.m[c(1,4,3,2),3:1,4:1]
table.4.1_rbind(cbind(m.6.m[,1,],m.5.m[,1,]),
cbind(m.6.m[,2,],m.5.m[,2,]),
cbind(m.6.m[,3,],m.5.m[,3,]))
dimnames(table.4.1)_list(rep(methods,3),paste("n=",z,sep=""))
latex.table(table.4.1,dec=3,
        rowlabel="",rgroup=paste("p=",ps),n.rgroup=c(4,4,4),label="",
        caption="Median Timings for Median Regression")
```

Like our graphics functions this produces a file, this time consisting of LaTeX commands, which can be incorporated into a document with the command,

```
\input table.4.1.tex
```

which produces the following table. Again, it is possible to alter the form of the table without chang-

	n=2000	n=4000	n=8000	n=12000	n=20000	n=40000	n=80000	n=120000
p= 4								
lm	0.320	0.550	1.030	1.510	2.586	5.164	10.688	15.957
RQFN	0.500	0.790	1.350	1.800	3.922	6.664	12.391	13.219
rqfn	0.530	1.050	2.400	3.840	7.812	16.734	34.211	55.070
l1fit	0.510	2.130	7.360	16.740	45.914	188.523	806.977	1761.367
p= 8								
lm	0.400	0.710	1.360	2.030	3.516	7.141	14.648	22.180
RQFN	0.810	1.380	2.260	3.730	5.945	9.859	20.820	29.078
rqfn	0.670	1.490	3.380	5.070	9.297	21.711	42.520	67.549
l1fit	1.120	3.870	12.530	31.270	77.656	286.422	1265.230	2645.080
p= 16								
lm	0.580	1.090	2.210	3.410	5.875	12.078	24.781	38.140
RQFN	1.130	1.860	4.500	6.350	10.570	19.852	37.660	52.319
rqfn	1.170	2.460	5.140	8.660	15.016	30.977	65.379	102.380
l1fit	2.260	7.050	24.650	48.820	136.578	500.250	1979.441	4564.160

Table 1. Median Timings for Median Regression

ing the form of the LaTeX document. As with all the SPLUS commands we have mentioned thus far, further details about the commands can be found using the SPLUS command `help(function.name)`. This raises the important point that one of the responsibilities of writing new SPLUS functions which may be used by other eventually is to provide documentation for each of them. Obviously, these `.d` documentation files should be provided in the archive associated with each project. Details on documentation generation are given in each of the basic 5 texts. We might also remind the reader that the new SPLUS interface to the help facility which can be initialized with the command

```
help.start(gui="motif")
```

on our xterms, for example, provides a key-word based search engine which is often successful in identifying new tools which are helpful in carrying a project forward.

6. Packaging and unpackaging.

Once a project is completed it is convenient to have some way to package the components so that they can be easily conveyed to others. The UNIX convention for this is `tar` + `gzip`. A hierarchy of directories including the current one is easily collected into a single file with the command

```
tar cvf - .  >/tmp/project.tar
```

which produces a file `project.tar` in the `/tmp` directory. The only delicate aspect of this is to avoid putting the target tar file into the hierarchy being tarred since this creates an infinite loop. Compression can then be done using either `compress` or `gzip`, the latter seems preferable since it is somewhat more efficient than the older `compress`. Tar files may include data, SPLUS functions, FORTRAN source, postscript files, even binary load modules. Unpackaging project archives is also straightforward. The Unix commands `uncompress` and `gunzip` may be used together with `tar`.

Of course, often, one has only a handful of files that one would like to package up for email transmission. This is most conveniently done with a shell archive, see the man page for the command *shar* for details. Most of the submissions to statlib are available in this format at the website `http://www.statlib.cmu.edu`. You might also explore the packaging of software, data, and text at the our local site `http://www.econ.uiuc.edu`.

7. Conclusions

These notes are intended as a guide to some useful "tricks of the trade" for the use of SPLUS in econometric research. They cover a wide gamet of topics from numerical simulation to transforming SPLUS arrays into LaTeX table format. It is hoped that through feedback from readers it will be possible to extend these notes further in a continuing effort ot demystify the process of describing econometric research. It is inevitable that may relevant research details will be omitted from published research papers. This should not be taken as an invitation to let them suffer in the darkness of benign neglect. One of the great potentialities of the internet, as David Donoho has recently emphasized to the wider statistical community, is to bring these details out into the light where they may be debated, tested, and refined.

REFERENCES

Becker R.A., J.M. Chambers, and A.R. Wilks, (1988) *The New S Language*, Wadsworth.

Buckheit, J. and D. Donoho, (1995). WaveLab and Reproducible Research, Technical Report 479, Department of Statistics, Stanford University.

Cleveland, W.R., (1993) *Visualizing Data*, Hobart Press.

Floyd, R.W. and R.L. Rivest (1975). Expected Time Bounds for Selection, *Communications of the ACM*, 18, 165-173.

Goossens M., F. Mittelbach and A. Samarin, (1994). *The LaTeX Companion*, Addison-Wesley.

Kernighan, B.W. and P.J. Plauger, (1976) *Software Tools*, Addison-Wesley.

Lamport, L., (1994).LaTeX, Addison-Wesley.

Spector, P., (1994) *An Introduction to* S *and* SPLUS, Duxbury.

Tufte, E.R., (1990) *Envisioning Information*, Graphics Press.

Venables, W.N. and B.D. Ripley, (1994). *Modern Applied Statistics with S-Plus*, Springer-Verlag.