Working with Finance Data in SAS

Justin McCrary
Professor of Law and D-Lab Director, UC Berkeley
Faculty Research Associate, NBER

D-Lab Workshop
Tuesday, January 27, 2015

Data We Will Talk About Today: CRSP and TAQ

CRSP: 1926-2013

- Main variable: Price (`prc`) as of end of day: either true price from the closing auction (`prc`>0) or midpoint of best bid and best offer (`prc`<0)
- Universe of stocks is mostly limited to common stocks, but also things like American Depository Receipts, Real Estate Investment Trusts, and so on
- Excludes preferred shares
- Entire daily stock file 1926-2013 (`DSF`) is only 13G

TAQ: 1993-2014

- intraday information on Trades And Quotes
- timestamps down to the second or millisecond, depending on flavor
- trades and quotes come in separate files
- quote file is so-called "top of order book" data, i.e., each of the 13 regional exchanges sends best bid and best offer to Consolidated Quotation System
- Has to be processed intelligently to obtain National Best Bid or Offer (NBBO)
- Only identifier is trading symbol (recycled over time)
- Native sort order is (symbol,timestamp)
- Trade (quote) files for 2011-2014 are 2T (53T): efficiency is key

Topics:

CRSP:

1. efficiency with your data step
2. a wee bit of macro language
3. single DOW loop
4. double DOW loop
5. double DOW and arrays for CAPM data preparation

TAQ:

1. more serious macros
2. NBBO calculation
3. parallel processing
4. pipes and compression
5. sequential access versus random access and compression
6. dimension reduction

Data Step

```
options ls=256 nocenter;

proc contents data=crspa.dsf;                    *Why start with this?;
run;

libname tmpdat '/scratch/berkeley/rpb3';         *What engine is being used?;

data tmpdat.ex1;                                 *Hold a copy;
  set crspa.dsf(keep=permno date prc shrout ret
                where=(nmiss(permno,date,prc,shrout,ret)=0 &
                       year(date)>=2007));       *Why bother with these options?;
  mc=shrout*abs(prc)/1e6;                        *market cap in millions;
run;

proc summary data=tmpdat.ex1 nway;               *What is "nway"?;
  class date;
  var ret;
  weight mc;
  output out=tmp mean=;                          *What variables will tmp contain?;
run;

proc print data=tmp(obs=10);                     *Why is this always a good idea?;
```

Let's walk through this code...

The DOW loop: Example with CRSP

```
data tmpdat.ex2;   *tmpdat.ex1 and tmpdat.ex2 will be identical;
  do until (eof);  *Where does SAS get the indicator eof?;
    set crspa.dsf(keep=permno date prc shrout ret
                  where=(nmiss(permno,date,prc,shrout,ret)=0 &
                         year(date)>=2007))
        end=eof;
    mc=shrout*abs(prc)/1e6;
    output;         *Why is this necessary?;
  end;
run;

proc compare base=tmpdat.ex1 compare=tmpdat.ex2;
run;
```

Results from proc compare

```
Observation               Base      Compare

First Obs                    1            1
Last  Obs             11851091     11851091

Number of Observations in Common: 11851091.
Total Number of Observations Read from TMPDAT.EX1: 11851091.
Total Number of Observations Read from TMPDAT.EX2: 11851091.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 11851091.

NOTE: No unequal values were found. All values compared are exactly equal.
```

DOW Loops: More Basic Example

Question: What is a DOW Loop?

Answer: It's a different way of doing a data step. The name is unusual because it was named after its apparent creator, Ian Whitlock, although Paul Dorfman probably did more to expand its usage. The history is easy to look up on the SAS-L listserv.

Example: Consider the following code:

```
data one; do i=1 to 5; output; end; run; *Easy data: 1,2,3,4,5;
data two;                                *Compute the first 5 squares;
  set one;
  X=i**2;
run;
```

Compare the second data step with:

```
data three;
  do until (eof);      *this is a DOW loop;
    set one end=eof;   *end=eof creates a variable eof that indicates end of file;
    X=i**2;            *compute the square;
    output;            *DOW loops need an explicit output statement...;
  end;                 *because we don't hit the bottom of the data step until here...;
run;                   *which is too late!!!;
```

`proc compare` again shows that data sets two and three are identical. So why use the DOW loop approach?

DOW Loops: Motivation

Suppose you have a panel data set. Consider the following data:

```
data one;
  do product=1 to 3;          *3 products;
    do _n_=1 to 10000;        *10000 transactions for each;
      price=rand('Uniform');
      output;                 *spit out the price of each transaction;
    end;                      *end the loop over transactions;
  end;                        *end the loop over products;
run;
```

You have been asked to compute the average price for each product. You would probably start with `proc summary` unless told otherwise. Turns out you can do it in a data step.

```
data two;
  drop total;                  *Does it matter where this statement is?;
  do until (last.product);     *Where does last.product come from?;
    set one;
    by product;                *What is call missing()?;
    if first.product then call missing(total,n);
    total=sum(total,price); *Why don't missing values cause problems?
    n=sum(n,1);                *Later will use a trick to avoid creating n
  end;                         *Why don't we need an output statement?
  avgprice=total/n;            *How many records are going to be in two?
run;
```

Moreover, it is often faster to do means in a data step. (Less true now with automatic parallelization of `proc summary`.)

DOW Loops: Correctitude

Does it work? Let's compare the results to proc summary

Obs	product	n	avgprice
1	1	10000	0.49232
2	2	10000	0.49995
3	3	10000	0.49774

Obs	product	_FREQ_	price
1	1	10000	0.49232
2	2	10000	0.49995
3	3	10000	0.49774

So it works. Is it useful? Suppose we are doing a fixed effects model and we want to demean the data by product.
Enter... the double DOW loop

Double DOW Loops and De-meaning by Group

```
data two(drop=total);                          *different way of dropping;
  do _n_=1 by 1 until (last.product);          *note how I highjack _n_ here;
    set one;
    by product;
    if first.product then total=price;         *also now savings of call missing() irrelevant;
    else total=sum(total,price);
  end;
  avgprice=total/_n_;      *note that now avgprice exists before we step through the data;
  do until (last.product);
    set one;
    by product;
    price_demeaned=price-avgprice; *avgprice was computed before this loop started and...;
    output;                        *is constant from perspective of this second loop;
  end;
run;
```

Cool. And to be concrete, here is the output:

Obs	product	price	avgprice	price_demeaned
1	1	0.75444	0.49960	0.25484
2	1	0.09859	0.49960	-0.40101
3	1	0.97218	0.49960	0.47258
4	1	0.11213	0.49960	-0.38746
5	1	0.56501	0.49960	0.06541
6	1	0.00427	0.49960	-0.49533
7	1	0.34304	0.49960	-0.15656
8	1	0.66174	0.49960	0.16214
9	1	0.58078	0.49960	0.08118
10	1	0.85588	0.49960	0.35628

Double DOW Loops and CAPM Regressions

- In a CAPM regression, we first want to compute market returns and then run a regression of a given stock's daily return on the daily market return
- A lot of times people will use the daily return for the S&P 500 to proxy for the market return
- Suppose you didn't have that, or you wanted to get the broadest possible picture of the market
- Then you would use your CRSP data to compute the market return before running your regressions
- Think about the steps involved in a naïve implementation
  1. `proc summary` with a `class date` statement, like above
  2. sort `tmpdat.ex1` by `date`
  3. merge the results of the `proc summary` back onto `tmpdat.ex1`
  4. sort `tmpdat.ex1` by `permno`
  5. do a `proc reg` with a `by permno` statement
- Rules to live by: Never sort big datasets
- How avoid doing that? One approach is hash tables (which we don't have time for today but that are powerful and worth learning about). Another is a double-DOW combined with array statements

Double DOW Loops: CAPM Data Preparation

```
data capmdat;
  array mret{1 : 36500} _temporary_;  *these never get written to tmp;
  array mwgt{1 : 36500} _temporary_;  *NOTE: temp arrays are automatically retained;
  do until (eof1);
    set tmpdat.ex1 end=eof1;
    mret(date)=sum(ret*mc,mret(date)); *cumulate Y*W;
    mwgt(date)=sum(mc    ,mwgt(date)); *cumulate W;
  end;
  *at this point we have read the entire file once: now do sum(Y*W)/sum(W);
  do _n_=1 to dim(mret); if mwgt(_n_) ne . then mret(_n_)=mret(_n_)/mwgt(_n_); end;
  *now read the data again;
  do until (eof2);
    set tmpdat.ex1 end=eof2;  *why am I using eof1 and eof2?  why not just eof?;
    market_return=mret(date); *market return is the one corresponding to today;
    output;                   *we want the micro data and we want it now;
  end;
run;

proc reg data=capmdat outest=results noprint;
  by permno;
  model ret = market_return;
run;
proc print data=results;
  title "These are the CAPM results";
run;
```

This takes roughly 1/3 the time of the naïve implementation. For CRSP, that probably doesn't matter too much (4 seconds versus 12) but for TAQ these kinds of improvements are handy.

TAQ

Dimension Reduction

An old econometric result is that when the covariates have a grouping structure, you can actually recover your regression results from the grouped data.

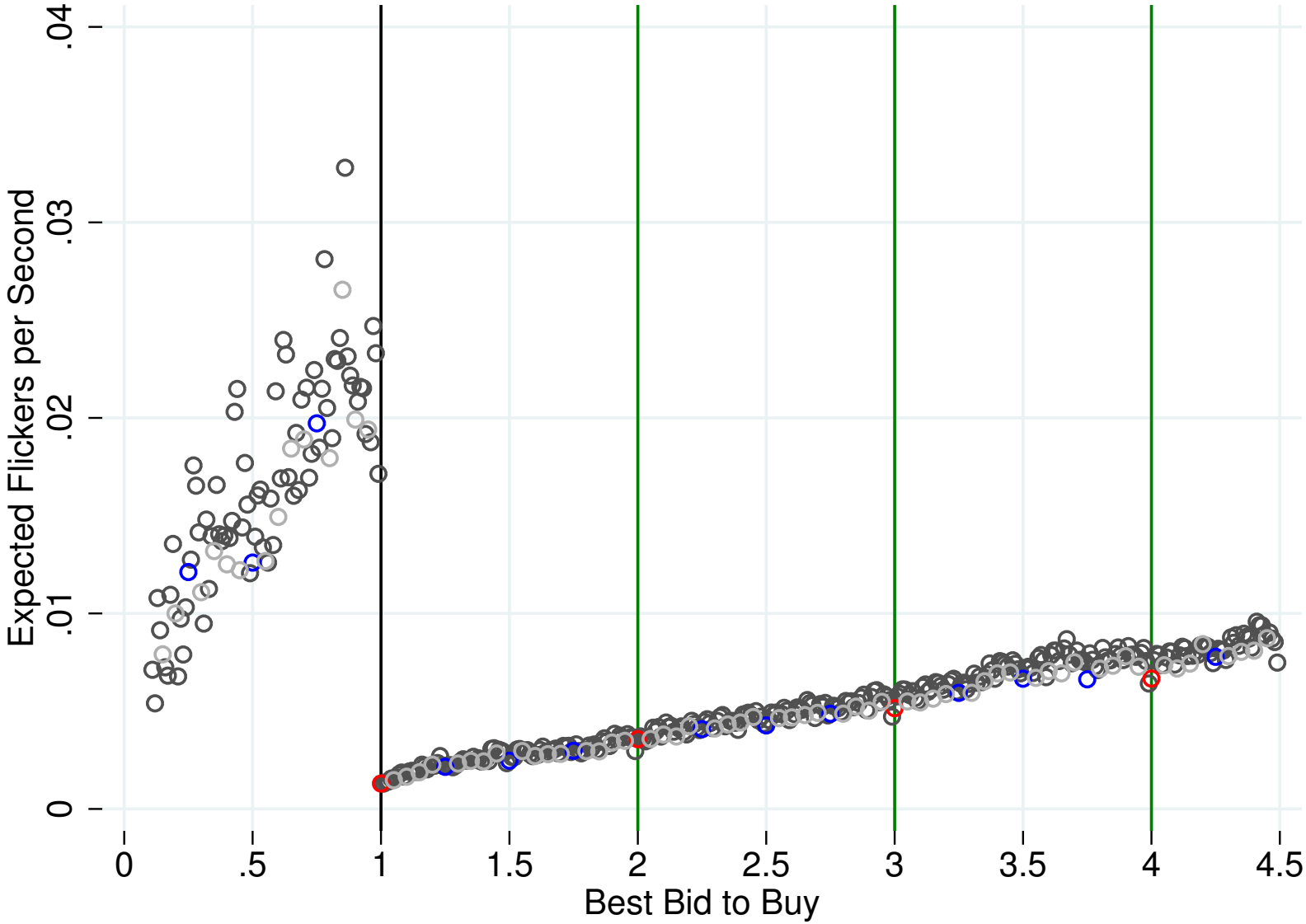
$$\widehat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \tag{1}$$

$$= \left(\sum_{j=1}^{J}\sum_{i=1}^{n_j} X_j X_j'\right)^{-1} \sum_{j=1}^{J}\sum_{i=1}^{n_j} X_j Y_{ij} \tag{2}$$

$$= \left(\sum_{j=1}^{J} n_j X_j X_j'\right)^{-1} \sum_{j=1}^{J} n_j X_j \underbrace{\frac{1}{n_j}\sum_{i=1}^{n_j} Y_{ij}}_{\overline{Y}_j} \tag{3}$$

- Obviously, the same thing works for sample means
- Standard errors from the grouped data are arguably more appropriate than those from the microdata
- TAQ data come stored one file per day
- A good idea: Loop through each day, storing up the results you need from that day, then aggregate at the end

Motivation for Dimension Reduction: Measuring Rate of "HFT Flicker"



How did I compute this conditional expectation?

Dimension Reduction

Strategy:

- Loop over days
- For each day, compute the sample mean of the outcome of interest by two-digit price (can use either a `proc summary` or a DOW-loop approach) and store those on disk
- The daily sufficient statistics for the overall problem are usually trivial to store
- Wharton Research Data Services (WRDS) currently allows for 5 jobs to be run simultaneously, so you can effectively do 1 trading week at a time
- Then aggregate those results after all the daily files are done running
- Usually the aggregator takes < 1 minute to run

- To do this, we need to proceed in several steps
  1. Use macros to store metadata about which files need processing
  2. Use a NBBO algorithm to compute the NBBO from the raw quote data
  3. Store extracts using compression
  4. Write a file to decompress on the fly, computing the daily sufficient statistics
  5. Write a file to aggregate those results

1. Getting Metadata into Macro Variables

```
%macro days_mo(yr,mo);
  filename INDAT pipe
  "ls -lg /wrds/nyse/sasdata/taqms/ct/ctm_&yr.&mo.*sas7bdat|awk '{print $8}'";
  data ct; *Data set of trade files;
    length name $8;
    infile INDAT truncover;
    input ls_ct $200.;
    name=substr(ls_ct,33,8);
  proc sort data=ct; by name; run;

  filename INDAT pipe
  "ls -lg /wrds/nyse/sasdata/taqms/cq/cqm_&yr.&mo.*sas7bdat|awk '{print $8}'";
  data cq;
    length name $8;
    infile INDAT truncover;
    input ls_cq $200.;
    name=substr(ls_cq,33,8);
  proc sort data=cq; by name; run;

  data days(where=(left(trim(name)) not in
            ("20101126","20100106","20111125","20120405","20120703","20121123","20121224")));
    merge ct(in=A) cq(in=B);
    by name;
    if A and B;
  run;
  %global numdays; *Declare the puppies to be global so other programs can access them;
  %do _t=1 %to 300; %global file&_t; %end;
  data _null_;
    set days end=eof;
    call symput('file'||left(_n_),left(trim(name))); *creates macros &file1, &file2, ...;
    if eof then call symput('numdays',left(_n_));    *creates macro &numdays;
  run;
%mend;
```

In other codes, can iterate &day from 1 to &numdays and use &&file&day.

2. NBBO Algorithm: Basic Idea

```
%global exString nEx;
%let exString=ABCDIJKMNPQTWXYZ;                                  *codes for different exchanges;
%let nEx=%length(&exString);

%macro NBBOhm(cqIn, cqOut);
data &cqOut;
  set &cqIn;
  by notsorted permnoID time_m;
  array exBid(&nEx) _temporary_;                                  *note that retain is automatic;
  array exAsk(&nEx) _temporary_;                                  *because of _temporary_;
  if bid=0 or bidsiz=0 then call missing(bid,bidsiz);             *screen out bs bids...;
  if ask=0 or asksiz=0 then call missing(ask,asksiz);             *and asks;
  if first.permnoID then call missing(of exBid(*), of exAsk(*)); *initialize array to missing;
  kEx = index("&exString",ex);                                    *kEx is location of ex in &exString;
  if kEx>0 then do;
    if (bid ne exBid(kEx)) then exBid(kEx) = bid;                 *superseded;
    if (ask ne exAsk(kEx)) then exAsk(kEx) = ask;                 *superseded;
    BBid = max(of exBid(*));                                      *best bid to buy is highest one;
    BAsk = min(of exAsk(*));                                      *best ask to sell is lowest one;
  end;
  if last.time_m then output;                                     *yes, there are multiple records per ms;
run;
%mend;
```

Basic idea can be expanded lots of directions, but they don't fit on this slide.

3A. Compression

Considerations

1. Most types of compression will reduce disk usage substantially. I usually see a factor of 3-10 reduction.
2. Compression can sometimes *reduce* run-times (most people find this surprising) as well as disk usage. Depends on size of the file and extent to which IO is the binding constraint on runtimes.
3. Compression will usually increase the complexity of your code, but sometimes by not much
4. `lz4` is *much* faster than `gzip` at decompression and recommended, even though `lz4` compressed files are not as small as `gzip` compressed files
5. SAS code for reading from compressed files is kind of a pain
6. And further requires that we store our SAS files as "sequential access" files (*.sas7sdat) rather than the more traditional "random access" files (*sas7bdat)
7. To create sequential access files, you will need to create a library using the `V9TAPE` engine and use a named pipe (broken in SAS 9.4, still works in SAS 9.3 M0, but not in SAS 9.3 M1)
8. Worth it if you are trying to stay under quota
9. Spot check to see if it also improves run times (in my experience, it won't unless the compressed files are above, say, 1G, but machine specific)

3B. Compression: Reading Fast and Slow

If you aren't thinking, you might do:

```
x 'lz4 -dc tmp.sas7sdat.lz4 > tmp.sas7bdat';  *uncompress;
proc means data=tmpdat.tmp;                   *analyze;
run;
x 'rm tmp.sas7sdat';                          *clean up;
```

Try this instead:

```
libname tmpdat V9TAPE '/scratch/berkeley/jmccrary/tmp';  *create sequential library;
x 'mknod shellpipe p';                  *exit to the shell to create the named pipe;
filename SASpipe pipe
  'lz4 -dc > shellpipe < /scratch/berkeley/jmccrary/tmp/tmp.sas7sdat.lz4 &';
data _null_;
  infile SASpipe;                   *this odd code activates the named pipe for SAS;
run;
libname pipedat 'shellpipe';              *point a library right at the named pipe;
proc means data=pipedat.tmp;      *compute means on the fly while lz4 -dc happening;
run;
x 'rm shellpipe';                         *my mama taught me to clean up after myself;
```

You can use these ideas to store your extracts from TAQ in compressed format and read from them on the fly.

3C. Here is a macro for creating extracts from TAQ

```
%macro subsett(day,fileOUT);
  data _null_; date=input("&&file&day",yymmdd8.); call symput('dateval',trim(date)); run; *get date val;
  data lusFIN(keep=sym_root sym_suffix permnoID group);
    set lusdat.lusFIN(where=(begdate<=&dateval<=findate and sample=1)); *this file contains my tickers;
  run;
  proc sort data=lusFIN;
    by sym_root sym_suffix;
  run;
  *A key-index merge is fast, but it is a bit subtle to handle because we have multiple securities for;
  *a given sym_root. We need to match on (sym_root,sym_suffix) and SAS does not do keyreset until 9.4;
  data &fileOUT(keep=permnoID group time_m ex price price2dig size tr_scond finra iso);
    set lusFIN(keep=sym_root sym_suffix permnoID group
               rename=(sym_suffix=_sym_suffix));
    length finra iso 3;
    do sym_root=sym_root, '___';          *necessary because of sym_suffix;
      do while (_iorc_=0);
        set taqmsec.ctm_&&file&day(keep=sym_root sym_suffix ex time_m price size)
            key=sym_root;                 *idea is that the index already written;
        if _iorc_= 0 and sym_suffix=_sym_suffix then do;
          finra=(ex='D');
          iso=(index(tr_scond,"F")>0);
          output;                         *spit out each match;
        end;
      end;
      _iorc_=0;                           *reset to go to next batch of matches;
    end;
    _error_=0;                            *not essential but logical;
  run;

%mend;
```

3D. Here is how to use the subsett macro

```
%macro tinytaq(yr,mo);
  %days_mo(&yr,&mo);            *get days stored as macros;
  %do day=1 %to &numdays;       *loop over days in the month;
    %subsett(&day,ctm);         *goes to subsett() to pull right subset;
    *reshuffle without sorting;
    data idx;
      set ctm(keep=permnoID);
      by permnoID notsorted;
      retain beg;
      if first.permnoID then beg=_n_; *first record for permnoID;
      if last.permnoID then do;
        fin=_n_;                      *last record for permnoID;
        output;                       *this has first and last record for permnoID;
      end;
    run;
    proc sort data=idx; by permnoID; run; *this is a quick sort if ~1,000 numbers;
    data _null_;                          *create macros with first and last numbers;
      set idx end=eof;
      call symput('begT'||left(_n_),beg); call symput('finT'||left(_n_),fin);
      if eof then call symput('nT',_n_);
    run;
    x "mknod pipe&&file&day p";
    filename nwrpipe pipe
      "lz4 -zc < pipe&&file&day > /scratch/berkeley/tinytaq/ctm_&&file&day...sas7sdat.lz4 &";
    data _null_; infile nwrpipe; run;
    libname pipedat "pipe&&file&day";
    data pipedat.ctm_&&file&day(sortedby=permnoID time_mroot time_mtrail);
      set %do i=1 %to &nT; ctm(firstobs=&&begT&i obs=&&finT&i) %end; open=defer;
    run;
    x "rm pipe&&file&day";
  %end;
%mend;
%tinytaq(&yr,&mo,&debug);
```

3E. Parallel Processing

Finally, write a script called `tinytaq_t.sh` containing the lines

```
#!/bin/sh
#$ -cwd
#
# Send grid engine standard output and error to /dev/null
#$ -o /dev/null
#$ -e /dev/null
mkdir -p logs
mkdir -p prints
sas tinytaq_t.sas -log logs/tinytaq_t$1.log -print prints/tinytaq_t$1.lst -sysparm $1
```

and then bast away

```
qsub tinytaq_t.sh 201101
qsub tinytaq_t.sh 201102
qsub tinytaq_t.sh 201103
qsub tinytaq_t.sh 201104
qsub tinytaq_t.sh 201105
qsub tinytaq_t.sh 201106
qsub tinytaq_t.sh 201107
qsub tinytaq_t.sh 201108
qsub tinytaq_t.sh 201109
qsub tinytaq_t.sh 201110
qsub tinytaq_t.sh 201111
qsub tinytaq_t.sh 201112
```

This will write your extracts in parallel

4. Analyze Directly from the Compressed Data

Consider the following macro in the file `FigureR.sas`:

```
%macro FigureR(yr,mo);
  %days_mo(&yr,&mo);
  %do day=1 %to &numdays;
    x "mknod pipeQ&&file&day p";
    filename nwrpipe pipe
      "lz4 -dc > pipeQ&&file&day < /scratch/berkeley/tinynbbo/nbbo_&&file&day...sas7sdat.lz4 &";
    libname pipedatQ "pipeQ&&file&day";
    data _null_;
      infile nwrpipe;
    run;
    proc summary data=pipedatQ.nbbo_&&file&day nway;
      class group BBid2dig BAsk2dig;
      var BBidsiz BAsksiz;
      output out=tmpdat.FigureR_&&file&day
                  mean=;
    run;
    x "rm pipeQ&&file&day";
  %end;
%mend;

%FigureR(&yr,&mo,&debug);
```

And the companion script `FigureR.sh` containing the lines

```
#!/bin/sh
#$ -cwd
#
# Send grid engine standard output and error to /dev/null
#$ -o /dev/null
#$ -e /dev/null
sas FigureR.sas -log logs/FigureR$1.log -print prints/FigureR$1.lst -sysparm $1
```

which you call in a "burst" just like before

5. Aggregation

Exercise!

Questions?