Advanced Programming in Quantitative Economics

Introduction, structure, and advanced programming techniques

Charles S. Bos

VU University Amsterdam

c.s.bos@vu.nl

20 – 24 August 2012, Aarhus, Denmark

Outline

Day 5 - Morning

9.00L Data handling

- Difference in formats
- Reading large datasets
- Selecting and transforming

10.00 Handing out exam

10.30P Selected exercises. Choice of

- Reading HF data
- Implement HF Autoregressive Duration Model
- Using graphing package
- AR(p) estimation with ARFIMA package

12.00 Lunch

Input and Output

file type	*extension*	*Names*	*Remark*
ASCII matrix file	.mat	-	Convenient input
ASCII data file with load information	.dat	+	
PcGive/OxMetrics data file	.in7 (with .bn7)	+	Retains variable names
Excel spreadsheet file	.xls	+	Common format, take care
Lotus spreadsheet file	.wks/.wk1	+	
Gauss data file	.dht (with .dat)	+	
Gauss matrix file	.fmt	-	Small, save results
Stata data file	.dta	-	Input only
text file using `fscan`/`fprint` functions		?	Lots of control
binary file using `fread`/`fwrite` functions		?	

```
main()
{
  decl mX, asVar;

  mX = loadmat("data/data.xls", &asVar);
  println("Saving data in 5 types of files: ...");
  savemat("excl/lm_data.mat", mX);
  savemat("excl/lm_data1.dat", mX, asVar);
  savemat("excl/lm_data.dht", mX, asVar);
  savemat("excl/lm_data.in7", mX, asVar);
  savemat("excl/lm_data.fmt", mX);
  println("done.");
}
```

Input: mat format

- Text-based format, very convenient for inputting your data
- Starts with two numbers, rows and columns of matrix
- Followed by numbers; output matrix is filled row-by-row
- Numbers are separated by space, comma or new-line
- '.', 'm', 'M' and '.NaN' are considered missing
- '.Inf' is infinity
- Other text leads to skipping the remainder of the line

Paths

- You may specify a full path (but relative paths are easier). Use either '/' (preferably) or '\\' in your path.
- The file is searched first starting from the present directory, then along the value of the `OX6PATH` environment variable (i.e., usually in the main Ox directory and its `include` subdirectory).

stackloss.mat example

Listing 1: stack/data/stackloss.mat

```
4 21
//  Stackloss.mat
//
//  Hoeting, J. A., Madigan, D., and Raftery, A.E. (1996) ``A Method for
//  Simultaneous Variable Selection and Outlier Identification in Linear
//  Regression,'' {\em Journal of Computational Statistics and Data
//  Analysis}, {\bf 22}, 251-270.
//
//  Data source:
//    Brownlee, K. A. (1965), "Statistical Theory and Methodology in
//    Science and Engineering", 2nd edition, New York:Wiley.
//
//  Rows contain:
//    Air Flow, Water Temperature, Acid Concentration, Stack Loss
80 80 75 62 62 62 62 62 58 58 58 58 58 58 50 50 50 50 50 56 70
27 27 25 24 22 23 24 24 23 18 18 17 18 19 18 18 19 19 20 20 20
89 88 90 87 87 87 93 93 87 80 89 88 82 93 89 86 72 79 80 82 91
42 37 37 28 18 18 19 20 15 14 14 13 11 12  8  7  8  8  9 15 15
```

Great format to save your data, easy reference etc.

Spreadsheet files

	A	B	C	D	E
1		CONS	INC	INFLAT	OUTPUT
2	1953-1	890.45	908.21	3.66	1203.77
3	1953-2	886.54	900.68	2.76	1200.36
4	1953-3	886.33	899.8	2.52	1193.63
5	1953-4	884.88	898.48	1.72	1193.04

Rules:

- Variables in columns
- Row of variable names
- Column of dates, unlabelled, with year and period: YYYYxP
- First sheet only
- Periods are only useful when loading the file into the Ox Database class (see later)
- Names can be loaded along, with `mX= loadmat("data/data.xls", &asVar)`.

OxMetrics files

OxMetrics files are combination of `.in7` file (Names of variables) and `.bn7` file (Binary data). Small format, containing names, flexible for use within Ox. Can be used in

1. OxMetrics: Analyse data in spreadsheet environment
2. Ox: `mX= loadmat("data/stackloss.in7", &asNames);`, allows for loading including names
3. Ox-Database: Convenient format to put in database, using Object Oriented programming (see elsewhere)

Hint:

- Prepare/analyse your data in small Ox program
- Save treated series (demeaned, log-transform, including dummies, without unneccessary variables) in `.in7` format
- Use `.in7` for input in estimation program

$\Rightarrow$ Less probability of mistakes, makes sure you use correct data.

Intermezzo: Stack loss

Prepare (and check!) data:

Listing 2: stack/loadstack.ox

```
main()
{
  decl mX, asNames, i;

  mX= loadmat("data/stackloss.mat");
  asNames= {"AirFlow", "WaterTemperature",
            "AcidConcentration", "StackLoss"};
  savemat("data/stackloss.in7", mX', asNames);  // Save data in columns
  print ("%r", asNames, meanr(mX)~varr(mX));    // Check data

  for (i= 0; i < 3; ++i)        // Plot data, correctly read?
    DrawXMatrix(i, mX[i][], asNames[i], mX[3][], asNames[3], 1);
  ShowDrawWindow();
}
```

Intermezzo: Stack loss II

Estimate using OxMetrics file:

```
mStackloss= loadmat("data/stackloss.in7", &asNames);    // Load the data
// Find the index of the stack loss variable
i= strifind(asNames, "StackLoss");
vY= mStackloss[][i];            // Read out column Stack loss
i= strifind(asNames, {"AirFlow", "WaterTemperature", "AcidConcentration"});
mX= mStackloss[][i];            // Get other columns
ir= olsc(vY, mX, &vBeta);       // Run OLS on columns
```

Or using a database:

```
#import <database>

main()
{
  decl db, mStackloss, asNames, vY, mX, ir, vBeta, i;

  db= new Database();
  db.LoadIn7("data/stackloss.in7");    // Load the data

  vY= db.GetVar("StackLoss");          // Read out column Stack loss
  mX= db.GetVar({"Constant",           // Get other columns
                 "AirFlow", "WaterTemperature", "AcidConcentration"});
  ir= olsc(vY, mX, &vBeta);            // Run OLS on columns

  print ("Ols estimates of Beta: ", vBeta');
}
```

High frequency data

Biggest trouble: Data has not been saved for your specific purposes...

- Enormous files (Euronext Amsterdam: Tick-by-tick data, only stocks in Amsterdam, >6GB per month)
- Dirty: Not always saved exactly in order, sorting/cleaning/linking necessary
- Order book: What is best order available? Depends on orders as they arrive *and orders as they are fulfilled; but those aren't indicated clearly in datafile...* Lots of processing needed
- Timing: What effect does daylight saving time have? What happens at market opening? Intra-day periodicity

Not really something to solve here...

Step one: Reading first file

Technicality:

> *32-bit Operating systems (Windows XP and before; many versions of Vista; older versions of Linux/OSX) can at most handle a file of 2GB length*

Even though a file is smaller, you might not have sufficient memory to read it... Have to work in smaller chunks.

Linux/OSX tools, available for Windows as well

`head -n file.csv`	Show first *n* lines
`tail -n file.csv`	Show last *n* lines
`grep ABN file.csv`	Show lines containing 'ABN'

Step one.B: Reading first file

First get a smaller file, to test your procedures: Write 10.000 lines as `eu_short.csv`

```
head -n 10000 EuronextAmsterdam-TemporaryOrders-200711-1.csv > eu_short.csv
```

or choose a specific stock (Fortis, in this case):

```
head -n 1 EuronextAmsterdam-TemporaryOrders-200711-1.csv > ft0711.csv
grep BE0003801181 EuronextAmsterdam-TemporaryOrders-200711-1.csv >> ft0711.csv
```

Pipe/append the outcome of grep to a file, using `> ft0711.csv` or `>> ft0711.csv`

Resulting file looks like

```
Internal code;ISIN code;Name;Quotation place;Order entry date;Order entry time;Order number;Chain Order nu
66694;BE0003801181;FORTIS           ;Amsterdam;2007-09-25;10:07:15;15381;0;L;D;2007-12-31;0;A;1000;0;19.1
66694;BE0003801181;FORTIS           ;Amsterdam;2007-09-25;11:13:00;27943;0;L;D;2008-09-24;0;A;1000;0;20.0
66694;BE0003801181;FORTIS           ;Amsterdam;2007-09-26;12:36:11;12968;0;L; ;.;0;V;50;0;25.500000;2007-
```

Step two: Reading smaller file

Several options:

1. Use editor to cut out unnecessary columns, get it into easily readable format
2. Use spreadsheet for this purpose
3. Write a script in Perl or Python
4. Write a program in Ox or other

Step two: Editor + Ox

Could you get it into `.mat` or `.csv` format ?
Try

```
mX= loadmat("myfile.csv");
```

- Advantage: Little ox-coding needed
- Disadvantage: You have to hand-edit the downloaded file...
  (else dates go wrong, you loose string indicators etc)

Step two: Reading in Ox

Bring out Swiss army knife: Read line by line, analyse block-by-block, store in prefered format

```
fh= fopen("ft0711.csv", "r");
while (!feof(fh))
  {
    // Read a line
    ir= fscan(fh, "%z", &sLine);

    // Analyse line

    // Store line
  }
```

Analyse: Build a routine reading in the items you need. Check out examples on `sscan()` and tokens.

Step two: Read a line

With `ir= fscan(fh, "%z", &sLine);` you have one line as a string.

Droste-effect: Take small steps...

- One option: Split at ';'
- Get all separate strings
- Easier to handle afterwards

Listing 3: hf/readft.ox

```
SplitLine(const aasElem, const sLine, const sSep)
{
  decl i, j;

  aasElem[0]= {};
  i= j= 0;
  while (i < sizeof(sLine))
    {
      j= strifind(sLine[i:], sSep); // Find sep after i
      if (j < 0)                    // If not found
        j= sizeof(sLine) - i;       //   indicate end of string
      aasElem[0]~= sLine[i:i+j-1];  // Read out next element
      i+= j+1;
    }
}
```

Step two: Read the elements

Now we have an array with strings: How to convert a string to a number?
Check `sscan`:

```
ir= sscan("2007-11-04", "%d", &iYY, "-%d", &iMM, "-%d", &iDD);
```

This

- Reads the string
- Assigns first integer (format='%d') to `iYY`
- Skips the minus, and assigns second integer to `iMM`
- etc
- and returns the number of values read correctly (here: 3)

Step two: Read the elements II

Read date (element 4), time (element 5), size (element 13) and price (element 15)

```
ReadElements(const avX, const asX)
{
  decl ir, iYY, iMM, iDD, iH, iM, iS, dSize, dPrice, dDate;

  ir= sscan(asX[4], "%d", &iYY, "-%d", &iMM, "-%d", &iDD) +
      sscan(asX[5], "%d", &iH, ":%d", &iM, ":%d", &iS) +
      sscan(asX[13], "%d", &dSize) +
      sscan(asX[15], "%g", &dPrice);
  // Get date number, including time-of-day}
  dDate= dayofcalendar(iYY, iMM, iDD)+timeofday(iH, iM, iS);

  avX[0]= dDate~dSize~dPrice;

  return ir;
}
```

Step two: Simpler alternative

If the input file has fixed-width fields, you could just count what columns you are interested in. Say you want the date, time, and the last price.

```
06/25/1979,1759,1.3580,1.3760,1.3580,1.3725,0,0
06/26/1979,1759,1.3725,1.3725,1.3554,1.3564,0,0
06/27/1979,1759,1.3564,1.3645,1.3564,1.3635,0,0
```

Then you want to read an integer ("%2i"), a slash+integer ("/%2i"), slash+integer ("/%4i"), comma, two integers of width two, (",%2i", "%2i"), bunch of nothing, ("%*22c") which is not assigned to anything, and a double ("%g")...

```
  ir= sscan(sLine, "%2i", &iMM, "/%2i", &iDD, "/%2i", &iYY,
            ",%2i", &ihh, "%2i", &imm, "%*22c", "%g", &dPrice);
  print ("Number of elements: ", ir,
         "Date: ", iYY~iMM~iDD,
         "Time and price: ", ihh~imm~dPrice);
```

Step two: Store the results

Remember speed-example: Concatenation is bad... Don't do

```
mX= <>;
while (!feof(fh))
  {
    ...
    ReadElements(&vX, asX);
    mX|= vX;
  }
```

How can you initialise `mX` in that case?

Step two: Store the results II

Do it in batches...

- Choose batch-size `iB`
- Initialise `mX` at the batch size: `mX= zeros(iB, 3);`
- Fill in the lines `mX[iL++][]= vX;`
- If you run out of space, extend `mX`: `mX|= zeros(iB, 3);`
- At the end, throw out empty lines: `mX= mX[:iL-1][];`

⇒ Relatively low number of concatenations

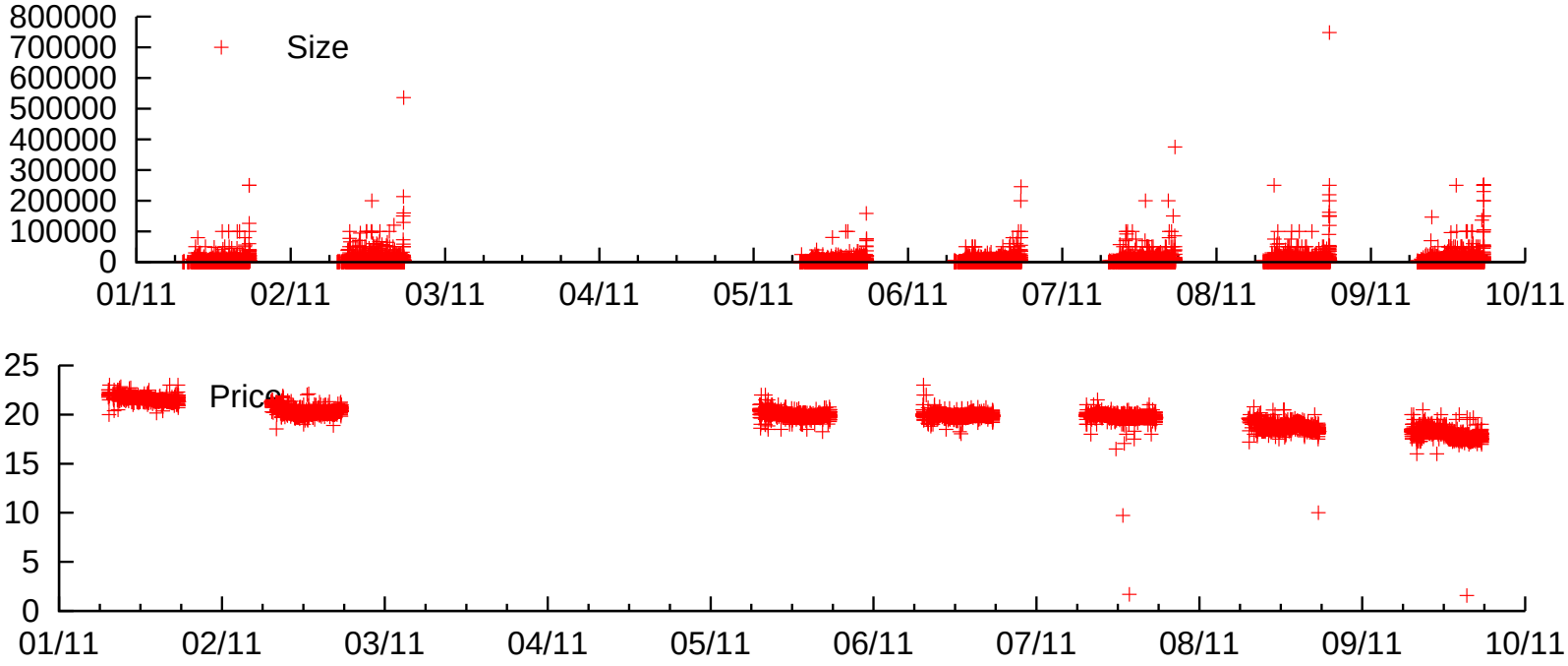
Quick code, especially for HF data!

Warning:

- HF data tends to be dirty
- Cleaning is a necessity
- Hard work, have to know the source, and construction of dataset...

HF Results

After reading Fortis data over the first days of November 2007 (during discussion of sale to ABNAMRO), 115461 'good' trades out of 307924 result.



Fortis trade size and price, November 2007

HF Alternative

Using 'Private' package `packages/loadcsv/loadcsv.ox`:

- Similar to `loadmat()`, simple loading
- Loads all data, including strings
- No hassle with `SplitLine` etc.
- But far slower...

Program	readft2.ox	readft3.ox
Method	`SplitLine`	`loadcsv`
Time	0m24s	2m01s

Data selection

Standard situation

- Large dataset of observations, many individuals
- Only those
  - older than 18,
  - female,
  - with an income between 25-50k

  should go into the sample...

Three words: `selectifr, deleteifr, vecrindex`
Some others: `thinc, range`

Data: selectifr

First two either select or delete specific rows from a matrix.

Listing 4: seldata/seldata.ox

```
main()
{
  ...
  // Load data including names
  mX= loadmat("../data/duration.in7", &asNames);

  // Find out what are indices in data set
  vSel= strfind(asNames, {"Age", "Sex", "Income"});

  mX= selectifr(mX, mX[][vSel[0]] .> 18); // Older than 18
  mX= selectifr(mX, !mX[][vSel[1]]);      // Only female
  mX= selectifr(mX, (mX[][vSel[2]] .>= 25000) .&&
        (mX[][vSel[2]] .<= 50000));       // 25k-50k income
  ...
}
```

`selectifr(mA, mB)`: Select only those rows of `mX` which have (at least one) non-zero element in the corresponding rows of the second argument.

Data: deleteifr

You could also do it the other way around:

Listing 5: seldata/deldata.ox

```
main()
{
  ...
  // Load data including names
  mX= loadmat("../data/duration.in7", &asNames);

  // Find out what are indices in data set
  vSel= strfind(asNames, {"Age", "Sex", "Income"});

  mX= deleteifr(mX, mX[][vSel[0]] .<= 18); // Not older than 18
  mX= deleteifr(mX, mX[][vSel[1]]);        // Not female
  mX= deleteifr(mX, (mX[][vSel[2]] .< 25000) .||
        (mX[][vSel[2]] .> 50000));         // <25k or >50k income
  ...
}
```

Data: vecrindex

For more extensive situations: Use `vecrindex`?

Listing 6: seldata/inddata.ox

```
// Create vector of zero/ones, indicating observations to use
vI= (mX[][vSel[0]] .> 18) .&&
    !mX[][vSel[1]] .&&
    (mX[][vSel[2]] .>= 25000) .&&
    (mX[][vSel[2]] .<= 50000);
// Re-using vSel, to indicate observations to use}
vSel= vecrindex(vI);
print ("Limits of indexed observations: ", limits(mX[vSel][]));

// Or, old tricks:
mX= selectifr(mX, vI);
```

- These functions are used extensively in `Initialise()`
- Warning: Don't do this within likelihood function...
- Changing big data matrices, sorting, whatever: Relatively **slow**

Data: thinr/thinc

If a matrix is too large to handle, thinr can select a number of rows from a large matrix:

```
mXSmall= thinr(mX, 1000);      // Get 1000 rows from mX, if possible
```

A similar effect, with more control, can be obtained using `range`:

```
iN= rows(mX);
vI= range(0, iN-1, 100);
mXSmall= mX[vI][];             // Get row 0, 100, 200, ... from mX
```