STATA TECHNICAL BULLETIN

July 1992

STB-8

A publication to promote communication among Stata users

Contents of this issue

an1.1 STB categories and insert codes

Inserts in the STB are presently categorized as follows:

General Categories:			
an	announcements	*ip*	instruction on programming
cc	communications & letters	*os*	operating system, hardware, & interprogram communication
dm	data management		
dt	data sets	*qs*	questions and suggestions
gr	graphics	*tt*	teaching
in	instruction	*zz*	not elsewhere classified
Statistical Categories:			
sbe	biostatistics & epidemiology	*srd*	robust methods & statistical diagnostics
sed	exploratory data analysis	*ssa*	survival analysis
sg	general statistics	*ssi*	simulation & random numbers
smv	multivariate analysis	*sss*	social science & psychometrics
snp	nonparametric methods	*sts*	time-series, econometrics
sqc	quality control	*sxd*	experimental design
sqv	analysis of qualitative variables	*szz*	not elsewhere classified

In addition, we have granted one other prefix, *crc*, to the manufacturers of Stata for their exclusive use.

an20 Stata U.K. distributor

Charles Fox, CRC, FAX 310-393-7551

Timberlake Clark, Ltd., distributor of Stata in the United Kingdom, has changed its name to Timberlake Consultants and moved from Greenwich to West Wickham. The new address, telephone, and fax numbers are

Timberlake Consultants
47 Hartfield Crescent
West Wickham
Kent BR4 9DW U.K.
Telephone: 44-81-462-0495
Fax: 44-81-462-0493

Please contact them if you would like to be placed on their mailing list.

an21 Stata for the Apple Macintosh

Ted Anderson, CRC, FAX 310-393-7551

Since the release of Stata 3.0, we have been busy porting Stata to new platforms. Among those new ports, Stata for the Apple Macintosh should be released in September. In the meantime, STB subscribers can obtain Beta-test copies by contacting Pat Branton at CRC.

Pricing will be the same as for the DOS version of Stata and, as with the DOS version, there will be two flavors—regular and Intercooled. Capabilities will be identical to the corresponding DOS versions—the regular version will estimate models of up to 38 right-hand-side variables and allow up to 255 variables and 32,754 observations in a data set. The Intercooled version will allow up to 398 right-hand-side variables and up to 2,046 variables and a limited-only-by-memory number of observations in a data set.

Both Macintosh versions will run under either Version 6 or Version 7 of the Macintosh operating system and, under Version 7, both support multitasking. The Intercooled version uses and so requires a math coprocessor, the regular version does not. Stata/Macintosh will work on any Macintosh with a hard disk and at least 1 megabyte of free memory for the regular version and 2 megabytes of free memory for the Intercooled version.

As with all ports, data sets are transferable to other environments, so data sets created on either the DOS or Unix versions of Stata can be copied to a Mac and used without translation and vice-versa.

The Stata command language survives intact—if you know how to use Stata on any other platform, you already know how to use Stata on the Mac. On the other hand, additions have been made for the experienced Mac user, such as allowing double clicking on a data set icon to bring up Stata with the data loaded or typing '`use *`' to pull up the standard Macintosh find-file dialog box.

The Stata out-board utilities such as `gphdot` and `gphpen` are present but hidden, so graphs can be printed by double clicking on the file icon and then following regular Macintosh features for printing a document.

an22 Stata for IBM RS/6000 workstation

Ted Anderson, CRC, FAX 310-393-7551

Also among the new ports is Stata for the RISC Station/6000, IBM's Unix-based workstation. This port is complete and is shipping now. Pricing is the same as for Stata on all other Unix platforms.

There is little to say about this port because Stata/Unix is Stata/Unix. For instance, the IBM RS/6000 uses X Windows and Stata uses its standard X-window driver to support the RS/6000. Congratulations are due mainly to IBM for supporting the X Window standard and all the rest of the Unix standards.

crc13 Short describes, finding variables, and codebooks

We have implemented three new commands related to `describe`: `ds`, `lookfor`, and `codebook`. Two of these commands are useful interactively—`ds` and `lookfor`. Now that (Intercooled and Unix) Stata allow up to 2,047 variables in a data set, finding the variable you want can be difficult.

`ds` simply lists the variable names in a compact format:

```
. ds
fips      hhsamp    hh10t19   hh20t29   hh30t39   hh40t49   hh50txx   medhhinc
medfinc   famsamp   femfam    rnkhhinc  mincpc    povfam    povfamf   povper
povperd   povchld   povchldd  genrev    igrev     igrevfs   cgtaxes   cgptaxes
cgstaxes
```

The syntax for `ds` is `ds` [*varlist*].

`lookfor` helps in finding variables:

```
. lookfor tax
 23. cgtaxes      long   %10.0g            Taxes of city gov´t
 24. cgptaxes     long   %10.0g            Property taxes of city gov´t
 25. cgstaxes     long   %10.0g            Sales taxes of city gov´t
. lookfor median
  8. medhhinc     long   %10.0g            Median hsehld income 1979
  9. medfinc      long   %10.0g            Median family money income
 12. rnkhhinc     int    %8.0g             Rank of median hsehld income
```

The syntax for `lookfor` is `lookfor` *string* [*string* [...]]. `lookfor` searches for *string*, ignoring case, within the variable names and labels. Thus, '`lookfor median`' found `rnkhhinc` because the word `median` was in the variable label. If multiple strings are specified, variable names or labels containing any of the strings are listed.

`codebook` examines the variable names, labels, and data to produce a "code book" describing the data.

```
. codebook, mv

fips --------------------------------------------------------- state/place code
                 type:  numeric (long)

                range:  [10060,560050]             units:  1
        unique values:  956                coded missing:  0 / 956

                 mean:    256495
             std. dev:    156998

          percentiles:        10%       25%       50%       75%       90%
                            61462    120426    252848    391360    482530

division ------------------------------------------------------ Census Division
                 type:  numeric (int)
                label:  division

                range:  [1,9]                      units:  1
        unique values:  9                  coded missing:  0 / 956

           tabulation:  Freq.   Numeric  Label
                           69         1  N. Eng.
                           97         2  Mid Atl
                          206         3  E.N.C.
                           78         4  W.N.C.
                          115         5  S. Atl.
                           46         6  E.S.C.
                           89         7  W.S.C.
                           61         8  Mountain
                          195         9  Pacific
```

```
region ---------------------------------------------------------- Census Region
                type:  numeric (int)
               label:  region

               range:  [1,4]                       units:  1
       unique values:  4                   coded missing:  0 / 956

          tabulation:  Freq.   Numeric  Label
                         166         1  NE
                         284         2  N Cntrl
                         250         3  South
                         256         4  West

name ----------------------------------------------------------------- City name
                type:  string (str30)

       unique values:  956                 coded missing:  0 / 956

            examples:  "Corona, CA"
                       "Huntington Beach, CA"
                       "Muskegon, MI"
                       "Salinas, CA"

             warning:  variable has embedded blanks

heatdd ---------------------------------------------------- Heating degree days
                type:  numeric (int)

               range:  [0,9901]                    units:  1
       unique values:  471                 coded missing:  3 / 956

                mean:   4422.53
            std. dev:   2192.81

         percentiles:        10%       25%       50%       75%       90%
                            1510      2460      4950      6232      6919

      missing values:     cooldd==. <-> heatdd==.
                         tempjan==. --> heatdd==.
                        tempjuly==. --> heatdd==.

cooldd ---------------------------------------------------- Cooling degree days
                type:  numeric (int)

               range:  [48,4389]                   units:  1
       unique values:  438                 coded missing:  3 / 956

                mean:   1240.81
            std. dev:   937.222

         percentiles:        10%       25%       50%       75%       90%
                             411       615       940      1566      2761

      missing values:     heatdd==. <-> cooldd==.
                         tempjan==. --> cooldd==.
                        tempjuly==. --> cooldd==.

tempjan ------------------------------------------- Average January temperature
                type:  numeric (float)

               range:  [2.2,72.6]                  units:  .1
       unique values:  310                 coded missing:  2 / 956

                mean:    35.749
            std. dev:   14.1881

         percentiles:        10%       25%       50%       75%       90%
                            20.2      25.1      31.3      47.8      55.1

      missing values:   tempjuly==. <-> tempjan==.

tempjuly ---------------------------------------------- Average July temperature
                type:  numeric (float)

               range:  [58.1,93.6]                 units:  .1
       unique values:  196                 coded missing:  2 / 956

                mean:   75.0538
            std. dev:   5.49504

         percentiles:        10%       25%       50%       75%       90%
                            68.8      71.8     74.25      78.7      82.3

      missing values:    tempjan==. <-> tempjuly==.
```

The syntax for `codebook` is `codebook` [*varlist*] [`, tabulate(`*#*`) mv`].

`codebook` examines the data in producing its results. For variables that `codebook` thinks are continuous, it presents the mean, standard deviation, and the 10th, 25th, 50th, 75th, and 90th percentiles. For variables that it thinks are categorical, it presents a tabulation. In part, `codebook` makes this determination by the number of unique values of the variable. If the number is 9 or fewer, `codebook` reports a tabulation, otherwise it reports summary statistics. `tabulate(15)` would change the rule to produce tabulations whenever a variable takes on 15 or fewer unique values.

The `mv` option, which we specified above, asks `codebook` to search the data to determine the pattern of missing values. This is a cpu-intensive task, which is the only reason that `mv` is an option. The result is useful. For instance, in the case of the last variable `tempjuly`, `codebook` reported that every time `tempjan` is missing, `tempjuly` is missing and vice-versa. Looking back up the output to the `cooldd` variable, `codebook` also reports that the pattern of missing values is the same for `cooldd` and `heatdd`. In both cases, the correspondence is indicated with "`<->`".

For `cooldd`, `codebook` also states that "`tempjan==. --> cooldd==.`". The one-way arrow means that a missing `tempjan` value implies a missing `cooldd` value, but a missing `cooldd` value does not necessarily imply a missing `tempjan` value.

`codebook` has some other features worth mentioning. When `codebook` determines that neither a tabulation nor summary statistics are appropriate, for instance, in the case of a string variable or in the case of a numeric variable taking on many values all of which are labeled, it reports a few examples instead. In the example above, `codebook` did that for the variable `name`. `codebook` is also on the lookout for common errors you might make in dealing with the data. In the case of string variables, this includes leading, embedded, and trailing blanks. `codebook` informed us that `name` includes embedded blanks. If `name` ever had leading or trailing blanks, it would have mentioned that, too.

Another feature of `codebook`—this one for numeric variables—is to determine the units of the variable. For instance, `tempjan` and `tempjuly` both have units of .1, meaning that temperature is recorded to tenths. `codebook` handles precision considerations (note that `tempjan` and `tempjuly` are `float`s) in making this determination. If we had a variable in our data recorded in 100s (e.g., 21,500, 36,800, etc.), `codebook` would have reported the units as 100. If we had a variable that took on only values divisible by 5 (5, 10, 15, etc.), `codebook` would have reported the units as 5.

`codebook`, without arguments, is most usefully combined with `log` to produce a printed listing for enclosure in a notebook documenting the data. `codebook` is, however, also useful interactively, since you can specify one or a few variables:

```
. codebook tempjan, mv

tempjan ------------------------------------------- Average January temperature
                  type:  numeric (float)

                 range:  [2.2,72.6]                   units:  .1
         unique values:  310                  coded missing:  2 / 956

                  mean:   35.749
              std. dev:   14.1881

           percentiles:        10%       25%       50%       75%       90%
                             20.2      25.1      31.3      47.8      55.1

        missing values:   tempjuly==. <-> tempjan==.
```

crc14 Pairwise correlation coefficients

The already-existing `correlate` command calculates correlation coefficients using casewise deletion: when you request correlations of variables x_1, x_2, ..., x_k, any observation for which x_1, x_2, ..., x_k are missing is not used. Thus, if x_3 and x_4 have no missing values, but x_2 is missing for half the data, the correlation between x_3 and x_4 is calculated using only the half of the data for which x_2 is not missing. Of course, you can obtain the correlation between x_3 and x_4 using all the data by typing '`correlate` x_3 x_4'.

The new `pwcorr` command makes obtaining such pairwise correlation coefficients easier:

`pwcorr` [*varlist*] [*weight*] [`if` *exp*] [`in` *range*] [`,` `obs sig print(`*#*`) star(`*#*`) bonferroni sidak`]

`pwcorr` calculates all the pairwise correlation coefficients between the variables in *varlist* or, if *varlist* is not specified, all the variables in the data.

Options

`obs` adds a line to each row of the matrix reporting the number of observations used in calculating the correlation coefficient.

`sig` adds a line to each row of the matrix reporting the significance level of each correlation coefficient.

`print(#)` specifies the significance level of correlation coefficients to be printed. Correlation coefficients with larger significance levels are left blank in the matrix. Typing '`pwcorr, print(.10)`' would list only correlation coefficients significant at the 10% level or better.

`star(#)` specifies the significance level of correlation coefficients to be starred. Typing '`pwcorr, star(.05)`' would star all correlation coefficients significant at the 5% level or better.

`bonferroni` makes the Bonferroni adjustment to calculated significance levels. This affects printed significance levels and the `print()` and `star()` options. Thus, '`pwcorr, print(.05) bonferroni`' prints coefficients with Bonferroni-adjusted significance levels of .05 or less.

`sidak` makes the Šidák adjustment to calculated significance levels. This affects printed significance levels and the `print()` and `star()` options. Thus, '`pwcorr, print(.05) sidak`' prints coefficients with Šidák-adjusted significance levels of .05 or less.

Examples

```
. pwcorr mpg price rep78 for, obs sig
          |      mpg    price    rep78  foreign
----------+------------------------------------
      mpg |   1.0000
          |
          |       74
          |
    price |  -0.4594   1.0000
          |   0.0000
          |       74       74
          |
    rep78 |   0.3739   0.0066   1.0000
          |   0.0016   0.9574
          |       69       69       69
          |
  foreign |   0.3613   0.0487   0.5922   1.0000
          |   0.0016   0.6802   0.0000
          |       74       74       69       74
          |
. pwcorr mpg price hdroom rseat trun rep78 for, print(.05) star(.01)
          |      mpg    price   hdroom    rseat    trunk    rep78  foreign
----------+---------------------------------------------------------------
      mpg |   1.0000
    price |  -0.4594*  1.0000
   hdroom |  -0.4220*           1.0000
    rseat |  -0.5213*  0.4194*  0.5238*  1.0000
    trunk |  -0.5703*  0.3143*  0.6620*  0.6480*  1.0000
    rep78 |   0.3739*                                      1.0000
  foreign |   0.3613*          -0.2938  -0.2409  -0.3594*  0.5922*  1.0000
. pwcorr mpg price hdroom rseat trun rep78 for, print(.05) bon
          |      mpg    price   hdroom    rseat    trunk    rep78  foreign
----------+---------------------------------------------------------------
      mpg |   1.0000
    price |  -0.4594   1.0000
   hdroom |  -0.4220            1.0000
    rseat |  -0.5213   0.4194   0.5238   1.0000
    trunk |  -0.5703            0.6620   0.6480   1.0000
    rep78 |   0.3739                                      1.0000
  foreign |   0.3613                             -0.3594   0.5922   1.0000
```

Methods and Formulas

Each correlation coefficient is calculated as described in [5s] correlate.

Let r be a calculated correlation coefficient and n the number of observations over which it is calculated. The unadjusted significance level is calculated as $p = \texttt{invt}(n-2, r\texttt{*sqrt(1-}r^2\texttt{))}$.

Let v be the number of variables specified; then $k = v(v-1)/2$ correlation coefficients are to be estimated.

If `bonferroni` is specified, the adjusted significance level is $p' = \min(1, k \times p)$. If `sidak` is specified, $p' = \min\big(1, 1 - (1-p)^n\big)$. In both cases, see Methods and Formulas in [5s] oneway for a more complete description of the logic behind these adjustments.

crc15 Parse bug

A bug has been reported in the `parse` programming command (see [5u] parse). The following program,

```
program define silly
        local if "req"
        parse "`*´"
        di "The if expression is (`if´)"
end
```

does not work properly. It should require that an `if` *exp* be specified but, in all cases, the `parse` reports "`invalid syntax`" and returns 198. The program does work if '`local if "opt"`' is substituted for '`local if "req"`', but in that case, of course, the program no longer requires that the `if` *exp* be specified. The bug is located inside the `parse` command and cannot be fixed without recompiling and reissuing Stata. The bug will be fixed as of the next release, but in the meantime, the following construct will work around the problem:

```
program define silly2
        local if "opt"
        parse "`*´"
        if "`if´"=="" {
                di in red "if exp required"
                exit 100
        }
        di "The if expression is (`if´)"
end
```

crc16 Encapsulated PostScript driver

Although the manual ([3] printing) claims that `gphpen`'s PostScript control files produce encapsulated PostScript, the claim is false. The files produced by the PostScript drivers shipped with Stata produce files suitable for printing on a PostScript printer, but not files suitable for embedding in a document (such as a document created with WordPerfect).

The `crc16` directory on the STB-8 diskette contains true encapsulated PostScript drivers. There are four files: `eps.pen`, `eps.plf`, `epsc.pen`, and `epsc.plf`. The `eps` files are for encapsulated PostScript on monochrome printers and the `epsc` files are for color printers. The files are small, so there is no reason not to copy all four. Copy the four files to `C:\STATA` if you use DOS and to `/usr/local/stata` if you use Unix; see the `readme` file in the `crc16` directory for detailed instructions.

Assuming *filename*`.gph` is the name of your gph-file, you can then use `gphpen` to produce a true encapsulated PostScript file by typing '`gphpen` *filename* `/deps`' (monochrome) or '`gphpen` *filename* `/depsc`' (color; Unix users: '`gphpen -deps` *filename*' or '`gphpen -depsc` *filename*'). Whether you use DOS or Unix, the result will be the creation of *filename*`.eps`. (If you rename `eps.pen` or `epsc.pen` to `default.pen`, you can dispense with the `/d` (`-d`) option on the `gphpen` command.)

The encapsulated PostScript file *filename*`.eps` is *not* suitable for direct printing; it is suitable only for embedding in another document. If you simply want to print your graph on a PostScript printer, continue to use the distribution PostScript control files.

os5 Running Stata under OS/2 2.0

Joseph Hilbe, Editor, STB, FAX 602-860-1446

Both the regular and Intercooled DOS versions of Stata can be successfully run under OS/2. Instructions follow.

Regular Stata

1. Migrate `stata.exe` into the *Additional DOS programs* folder if this did not automatically happen when you installed OS/2.
2. You will need to adjust EMS_MEMORY_LIMIT and install the DOS_DEVICE `ansi.sys` driver. Both are defined on the *DOS settings* menu.

   To get to the menu, click on the *Additional DOS programs* folder and then click the right mouse button on the Stata icon. Click on *Open*, *Settings*, *Session*, and *DOS settings*.

Fill in DOS_DEVICE with "`c:\os2\mdos\ansi.sys`". Fill in EMS_MEMORY_LIMIT with the amount of expanded memory to allocate to Stata, which might be as little as 1024 or as much as 6656 (the maximum regular Stata can use). 1024 corresponds to 1 megabyte and 6656 corresponds to just over 6 megabytes.

Finally, click on *Save* to save the settings and *Close* to close the windows.

3. To use regular Stata, double click on the Stata icon in the *Additional DOS programs* folder. Once in Stata, however, you must '`set ANSI on`'.

Intercooled Stata

1. Let's pretend you have installed Intercooled Stata as `istata.exe`. You must create a BAT file containing the line "`istata /k2000`", where the 2000 means Stata is to allocate 2 megabytes to its data areas. You can make this number larger or smaller. Assume the BAT file is called `ist.bat`.
2. Migrate `ist.bat` to the *Additional DOS programs* folder.
3. You will need to adjust DPMI_DOS_API and DPMI_MEMORY_LIMIT and install the DOS_DEVICE `ansi.sys` driver. Both are defined on the *DOS settings* menu.

   To get to the menu, click on the *Additional DOS programs* folder and then click the right mouse button on the `ist.bat` icon. Click on *Open*, *Settings*, *Session*, and *DOS settings*.

   Fill in DOS_DEVICE with "`c:\os2\mdos\ansi.sys`". Set DPMI_DOS_API to enabled (not auto). Fill in DPMI_MEMORY_LIMIT with `2` if you specified `/k2000` in your `ist.bat` file; `3` if you specified `/k3000`; and so on. Intercooled Stata can allocate considerable amounts of memory. To create a session with 64 megabytes, specify `/k64000` in `ist.bat` and fill in DPMI_MEMORY_LIMIT with 64. 64 megabytes is not the upper limit so far as Intercooled Stata is concerned.

   Finally, click on *Save* to save the settings and *Close* to close the windows.
4. To use Intercooled Stata, you double click on the `ist.bat` icon in the *Additional DOS programs* folder. Once in Stata, however, you must '`set ANSI on`'. If you wish to run Intercooled Stata in the background, you must first '`set graphics off`' before escaping from the window. Also note, if you go back and change `ist.bat`, you must re-migrate the BAT file to the *Additional DOS programs* folder.

os6 Importing and exporting text files with Stata

Phil Goldberg, Systems Manager, Cardiology, Children's Memorial Hospital, Chicago, IL, FAX 312-880-3324

[Mr. Goldberg has supplied executable C programs for the Unix environment. He is working on a DOS version at present and will make them available when completed. On the current STB diskette you will find the C source code and do-files.—Ed.]

Part I. Stata data exportation

Anyone who has tried to use Stata's `outfile` command has probably discovered that it formats output lines for an 80-column page. According to Bill Gould at CRC, early DOS users requested this so that they could print `outfile` generated files. Although CRC plans an `outfile` command option to permit lines to continue regardless of length and terminate only when the record ("tuple", "observation") ends, no version exists now.

The problem lies in the fact that most other software expects one record per line. `outfile`'s writing of multiple lines wreaks havoc on most efforts to cleanly export Stata datasets. After speaking with Mr. Gould, he suggested an approach which I've implemented below.

Very simply, the approach is a two-part process. The first part is to write two files. In the first file, the number of variables (columns) is written. In the second file, the data is exported using the `outfile` command. The second part involves using the information contained in the two files to create a comma-separated export file. All of these operations are transparent to the user, making it usable by ordinary mortals.

The first part is implemented in a Stata do-file. The second part is a C program. Neither program can be considered a paragon of style, but they should work as advertised.

Part II. Stata data importation

Another issue, and one which Stata is much better at handling, is the importation of data. Using the `infile` command, the user can read a comma-separated file into a set of variables. If you have a lot of variables, this can be tedious because, very often, the names of the variables happen to be located on the first line of the file. The `infile` command becomes very unwieldy when you have to list each variable name with it. To solve this, a C program called `sep2stata` reads a "separated" file and writes it out with a data dictionary at the top (and replaces missing information with "`.`", along with other miscellany).

The program is not smart. It looks to the first non-separator character on the first line and decides whether the first line is a set of labels or not. According to the logic of the program, if that first character is a double-quote, the first line contains labels. If not, the first line just contains data. Not smart, but very simple. If there are not enough labels (which includes the no-label case), `sep2stata` puts in its own, calling them `label`*n*, where *n* is the number of the column associated with the label (e.g., `label47`).

If you have varying numbers of values on each line, `sep2stata` assumes that the longest line (as defined by number of values) in the dataset has the right number of variables, while any shorter ones are missing data.

If the data contains strings, the program will notice it. If a column contains greater than 50% strings, the software will declare the variable a string (in the dictionary) of sufficient width to handle the widest entry in the column. The logic isn't perfect, but works most of the time (you may want to fine-tune it for your datasets). Primarily, we use it to handle data exported by Lotus 1-2-3. If you are interested in how we do that, contact me.

Finally, to round out the set of software, a short do-file contains the commands to do all this processing without exiting Stata. The commands contained in `readsep.do` will cause Stata to use `sep2stata` and read in the resulting dataset.

My apologies for the crudeness of the algorithms and the lack of features in general.

Examples

Typing '`do readsep` *input_filename*' in Stata imports data from a comma-separated file into Stata.

Typing '`sep2stata` *input_filename* `>` *output_filename*`.dct`' in Unix converts a comma-separated file into a Stata-readable file.

Typing '`do s2sep` *output_filename*' in Stata exports the current dataset from Stata.

Typing '`stata2sep` *log_filename data_filename*' in Unix converts the log and data files individually (you should never need to do this, but this is how things work internally) into a single comma-separated file.

sed7.1 Resistant nonlinear smoothing using Stata

William Gould, CRC, FAX 310-393-7551

Salgado-Ugarte and Garcia (1992) presented an implementation of the `4253EH` smoother for Stata, an example of a nonlinear, robust smoother as originally defined by Tukey (1977) and further developed by Velleman (1977). In the Tukey–Velleman notation, `4253EH` means the smoother proceeds by first taking running medians of span 4, then smooths that result using running medians of span 2, followed by running medians of span 5 and span 3, applies an end-point correction, and finally smooths the overall result with a Hanning linear smoother. Thus, the notation allows description of other smoothers and there is a literature discussing the relative merits of the various smoothers that can be described in the notation (Velleman 1980).

Below, I implement nonlinear, robust smoothers more generally, allowing the smoother to be specified in the Tukey–Velleman notation. The syntax of the `nlsm` command is

`nlsm` *compound_smoother*[`,twice`] *varname*`, generate(`*newvar*`)` [`noshift`]

where *compound_smoother* is $S[S\ldots]$ and S is

{1|2|4|5|6|7|8|9}[R]

3[R]S[S|R][S|R] . . .

E

H

Examples of *compound_smoother* include `3RSSH`; `3RSSH,twice`; `4253H`; `4253H,twice`; or `43RSR2H,twice`. `nlsm`, unlike most Stata commands, does not distinguish between upper and lowercase letters, so `4253h` is equivalent to `4253H`.

Options

`generate(`*newvar*`)` is not optional; it specifies the name of the new variable to be created.

`noshift` prevents shifting the data one time period forward for each pair of even smoothers applied. This option is useful only if subsequent smoothers are to be applied in separate commands and, in that case, making required shifts of the data is the user's responsibility.

Remarks

A smoother separates a data sequence y_t, $t = 1, 2, \ldots, N$, into a smooth, $z_t = S(y_t)$, and a rough, $r_t = y_t - z_t$. A compound smoother applies smoothers sequentially from left to right; thus, if A and B are smoothers, the smoother AB is defined as $z_t = B\bigl(A(y_t)\bigr)$.

Running median smoothers of odd span

The smoother `3` defines $z_t = \text{median}(y_{t-1}, y_t, y_{t+1})$. The smoother `5` defines $z_t = \text{median}(y_{t-2}, y_{t-1}, y_t, y_{t+1}, y_{t+2})$, and so on. The smoother `1` defines $z_t = \text{median}(y_t)$ and so does nothing.

In all cases, the end-points are handled by using smoothers of shorter, odd span. Thus, in the case of `3`,

$$\begin{aligned} z_1 &= y_1 \\ z_2 &= \text{median}(y_1, y_2, y_3) \\ &\ldots \\ z_{N-1} &= \text{median}(y_{N-2}, y_{N-1}, y_N) \\ z_N &= y_N \end{aligned}$$

In the case of `5`,

$$\begin{aligned} z_1 &= y_1 \\ z_2 &= \text{median}(y_1, y_2, y_3) \\ z_3 &= \text{median}(y_1, y_2, y_3, y_4, y_5) \\ z_4 &= \text{median}(y_2, y_3, y_4, y_5, y_6) \\ &\ldots \\ z_{N-2} &= \text{median}(y_{N-4}, y_{N-3}, y_{N-2}, y_{N-1}, y_N) \\ z_{N-1} &= \text{median}(y_{N-2}, y_{N-1}, y_N) \\ z_N &= y_N \end{aligned}$$

and so on.

Running median smoothers of even span

Define the median() function as returning the linearly interpolated value when given an even number of arguments. Thus, the smoother `2` defines $z_{t+.5} = (y_t + y_{t+1})/2$. The smoother `4` defines $z_{t+.5}$ as the linearly interpolated median of $(y_{t-1}, y_t, y_{t+1}, y_{t+2})$, and so on. In all cases, end-points are handled by using smoothers of shorter, even span. Thus, in the case of `4`,

$$\begin{aligned} z_{1.5} &= \text{median}(y_1, y_2) = (y_1 + y_2)/2 \\ z_{2.5} &= \text{median}(y_1, y_2, y_3, y_4) \\ &\ldots \\ z_{N-2.5} &= \text{median}(y_{N-4}, y_{N-3}, y_{N-2}, y_{N-1}) \\ z_{N-1.5} &= \text{median}(y_{N-2}, y_{N-1}) \\ z_{N-.5} &= \text{median}(y_{N-1}, y_N) \\ z_{N+.5} &= y_N \end{aligned}$$

`nlsm` keeps track of the number of even smoothers applied to the data; it is recommended that such smoothers always be applied in pairs. After all smoothers have been applied, the data is then shifted forward one position for each pair of even smoothers. Thus, the smoother `4253` or `4523` would result in values for z_2 through z_N; z_1 would be missing. The physical shifting of the data is not performed if `noshift` is specified.

The repeat operator

`R` indicates that a smoother is to be repeated until convergence, that is, until repeated applications of the smoother produce the same series. Thus, `3` applies the smoother of running medians of span 3. `33` applies the smoother twice. `3R` produces the result of applying `3` an infinite number of times. `R` should only be used with odd-span smoothers, since even-span smoothers are not guaranteed to converge.

The smoother `453R2` applies a span-4 smoother, followed by a span-5 smoother, followed by repeated applications of a span-3 smoother, followed by a span-2 smoother.

End-point rule

The end-point rule `E` modifies the values z_1 and z_N according to the formulas:

$$z_1 = \text{median}(3z_2 - 2z_3, z_1, z_2)$$
$$z_N = \text{median}(3z_{N-2} - 2z_{N-1}, z_N, z_{N-1})$$

When the end-point rule is not applied, end-points are typically "copied-in," i.e., $z_1 = y_1$ and $z_N = y_N$.

Splitting operator

The smoothers `3` and `3R` can produce flat-topped hills and valleys. The split operator `S` is an attempt to eliminate such hills and valleys by splitting the sequence, applying the end-point rule `E`, rejoining the series, and then resmoothing by `3R`.

The `S` operator may be applied only after `3`, `3R`, or `S`.

It is recommended that the `S` operator be repeated once (`SS`) or until no further changes take place (`SR`).

Hanning smoother

`H` is the Hanning linear smoother $z_t = (y_{t-1} + 2y_t + y_{t+1})/4$. End points are copied in, $z_1 = y_1$ and $z_N = y_N$. `H` should be applied only after all nonlinear smoothers.

Twicing

A smoother divides the data into a smooth and a rough; observed = smooth + rough. If the smoothing is successful, the rough should exhibit no pattern. Twicing refers to applying the smoother to the observed, calculating the rough, and then applying the smoother to the rough. The resulting "smoothed rough" is then added back to the smooth from the first step.

Examples

As a few examples of how `nlsm` can be used:

```
. nlsm 3 coalprdn, gen(smcp)
. nlsm 3r coalprdn, gen(smcp2)
. nlsm 3rss coalprdn, gen(smcp3)
. nlsm 3rssh3rssh3 coalprdn, gen(smcp4)
. nlsm 3rssh,twice coalprdn, gen(smcp5)
. nlsm 4253eh,twice gnp, gen(sgnp)
```

Certifications

`nlsm` has been tested on most of the examples provided in Tukey (1977) and produces identical results to those reported. Salgado-Ugarte and Garcia (1992) provided in Table 1 the results of applying `4253EH` to their length-frequency data. In comparison to the results calculated by `nlsm`, the following differences were observed:

Standard body length	Frequency (individuals)	Ugarte-Garcia smoothed values	`nlsm` smoothed values
37	6	6.0000	.
38	10	6.0000	6.7500
39	3	6.0000	6.3125
40	7	6.0000	6.0625
41	5	6.0000	6.0000
42	9	5.9375	5.9375

For the remaining lengths 43–67, there were no differences in the results, so results were identical for lengths 41–67; only results for the first four lengths differ. The difference in the first observation is due to a difference of implementation; Ugarte-Garcia "copied-back" the information for length 37 from length 38 whereas `nlsm` more agnostically filled in missing (applying two even-span smoothers results in shifting the data one unit forward, so information for the first observation is lost).

I do not have an explanation for the remaining three differences except to assert that the results reported by `nlsm` are as intended, which is not to say that they are necessarily more correct. There is obviously a difference in assumptions about how the start-up tail is to be handled between the two routines although, interestingly, that difference is not reflected in how the trailing tail is handled. (Not too much should be made of that, however. Define the function $\mathrm{rev}()$ as the function reversing a sequence, e.g., $\mathrm{rev}(y_i) = y_{N-i+1}$. Let $S()$ be some smoother. One is tempted to think that $S(y_t) = \mathrm{rev}\big(S(\mathrm{rev}(y_t))\big)$. That is true for median smoothers of odd span, the Hanning smoother, and the end-point rule. It is not, however, true for median smoothers of even span.)

In any case, the tails produced by any of these smoothers should not be taken too seriously—they are based on too little data and too many approximations and fix-up rules. The purpose of the smoother is to reveal the pattern for the middle-portions of the data.

References

Salgado-Ugarte, I. H. and J. C. Garcia. 1992. sed7: Resistant smoothing using Stata. *Stata Technical Bulletin* 7: 8–11.

Tukey, J. W. 1977. *Exploratory Data Analysis*, Ch. 7. Reading, MA: Addison–Wesley Publishing Company.

Velleman, P. F. 1977. Robust nonlinear data smoothers: Definitions and recommendations. *Proc. Natl. Acad. Sci. USA* 74(2): 434–436.

——. 1980. Definition and comparison of robust nonlinear data smoothing algorithms. *Journal of the American Statistical Association* 75(371): 609–615.

sg1.3 Nonlinear regression command, bug fix

Patrick Royston, Royal Postgraduate Medical School, London, FAX (011)-44-81-740 3119

`nlpred` incorrectly calculates the predictions and residuals when `nl` is used with the `lnlsq` (log least squares) option. The bug is fixed when the update on the STB-8 diskette is installed. `nlpred` is used after `nl` to obtain predicted values and residuals much as `predict` is used after `regress` or `fit`. The mistake affected only calculations made when the log least squares option was specified during estimation.

sg7 Centile estimation command

Patrick Royston, Royal Postgraduate Medical School, London, FAX (011)-44-81-740 3119

Stata's `summarize, detail` command supplies sample estimates of the 1, 5, 10, 25, 50, 75, 90, 95 and 99th (per)centiles. To extend `summarize`, I provide an ado-file for Stata version 3.0 which estimates arbitrary centiles for one or more variables and calculates confidence intervals, using a choice of methods.

The syntax of `centile` is

`centile` [*varlist*] [`if` *exp*] [`in` *range*] [`, centile(`*#* [*#* ...]`) cci normal meansd level(`*#*`)`]

The qth centile of a continuous random variable X is defined as the value of C_q which fulfills the condition $\mathrm{P}(X \le C_q) = q/100$. The value of q must be in the range $0 < q < 100$, though q is not necessarily an integer. By default, `centile` estimates C_q for the variables in *varlist* and for the value(s) of q given in `centile(`*#...*`)`. It makes no assumptions as to the distribution of X and, if necessary, uses linear interpolation between neighboring sample values. Extreme centiles (for example, the 99th centile in samples smaller than 100) are fixed at the minimum or maximum sample value. An 'exact' confidence interval for C_q is also given, using the binomial-based method described below (see *Formulæ*). The detailed theory is given by Conover (1980, 111–116). Again, linear interpolation is employed to improve the accuracy of the estimated confidence limits, but extremes are fixed at the minimum or maximum sample value.

You can prevent `centile` from interpolating when calculating binomial-based confidence intervals by specifying the **c**onservative **c**onfidence **i**nterval option `cci`. The resulting intervals are in general wider than with the default, that is, the coverage (confidence level) tends to be greater than the nominal value (given as usual by `level(`*#*`)`, by default 95%).

If the data are believed to be normally distributed (a common case), two alternate methods for estimating centiles are offered. If `normal` is specified, C_q is calculated as just described, but its confidence interval is based on a formula for the standard error (s.e.) of a normal-distribution quantile given by Kendall and Stuart (1969, 237). If `meansd` is alternatively specified, C_q is estimated as $\bar{x} + z_q \times s$, where $\bar{x}$ and s are the sample mean and standard deviation and z_q is the qth centile of the standard normal distribution (e.g. $z_{95} = 1.645$). The confidence interval is derived from the s.e. of the estimate of C_q.

Example

Examples of using `centile` to estimate the 5th, 50th and 95th centiles of the variable `price` (car price in dollars) in the Stata example file `auto.dta` are given below.

```
. format price %8.2f
. summarize price, detail
                           Price
-------------------------------------------------------------
      Percentiles      Smallest
 1%      3291.00        3291.00
 5%      3748.00        3299.00
10%      3895.00        3667.00      Obs                  74
25%      4195.00        3748.00      Sum of Wgt.          74
50%      5006.50                     Mean            6165.26
                        Largest      Std. Dev.       2949.50
75%      6342.00       13466.00
90%     11385.00       13594.00      Variance     8699525.97
95%     13466.00       14500.00      Skewness           1.65
99%     15906.00       15906.00      Kurtosis           4.82
. centile price, centile(5 50 95)
                                                -- Binom. Interp. --
Variable |     Obs      Percent     Centile        [95% Conf. Interval]
---------+-------------------------------------------------------------
   price |      74          5       3727.75        3291.23     3914.16
         |                 50       5006.50        4593.57     5717.90
         |                 95      13498.00       11061.53    15865.30
. centile price, c(5 50 95) cci
                                                -- Binomial Exact --
Variable |     Obs      Percent     Centile        [95% Conf. Interval]
---------+-------------------------------------------------------------
   price |      74          5       3727.75        3291.00     3955.00
         |                 50       5006.50        4589.00     5719.00
         |                 95      13498.00       10372.00    15906.00
```

Notice that Stata's `summarize` formula for interpolation of centiles gives somewhat different results than that used by `centile` (see *Formulæ*). Also, the confidence limits with the `cci` option in force are defined to fall exactly on sample values and are slightly wider than those with the default (`nocci`) option.

```
. centile price, c(5 50 95) normal
                              -- Normal, based on observed centiles --
Variable |     Obs      Percent     Centile        [95% Conf. Interval]
---------+-------------------------------------------------------------
   price |      74          5       3727.75        3211.19     4244.31
         |                 50       5006.50        4096.68     5916.32
         |                 95      13498.00        5426.81    21569.19
. centile price, c(5 50 95) meansd
                              -- Normal, based on mean and std. dev.--
Variable |     Obs      Percent     Centile        [95% Conf. Interval]
---------+-------------------------------------------------------------
   price |      74          5       1313.77         278.93     2348.61
         |                 50       6165.26        5493.24     6837.27
         |                 95      11016.75        9981.90    12051.59
. sktest price
                Skewness/Kurtosis tests for Normality
                                                 ------- joint -------
 Variable |  Pr(Skewness)   Pr(Kurtosis)  adj chi-sq(2)  Pr(chi-sq)
----------+-------------------------------------------------------
    price |      0.000         0.013           21.77       0.0000
```

The above two examples assume that `price` is normally distributed. With the `normal` option, the centile estimates are by definition the same as before. The confidence intervals for the 5th and 50th centiles are similar to the previous ones, but the interval for the 95th centile is very different. The results using the `meansd` option are also very different from both previous sets of estimates. The `sktest` (see [5s] sktest) test of skewness and kurtosis reveals that `price` is definitely not normally distributed, so the normal assumption is not reasonable and the `normal` and `meansd` options are not appropriate for such data. We rely on the results from the default choice, which doesn't assume normality. If the data are normally distributed, however, the precision of the estimated centiles and their confidence intervals will be ordered (best) `meansd` $>$ `normal` $>$ (worst) [default]. The `normal` option is useful when we really do want empirical centiles (i.e., centiles based on sample order statistics rather than on the mean and s.d.) but are willing to assume normality.

Formul▌

`Default case`. I basically use the method of Mood and Graybill (1963, 408). Let $x_1 \leq x_2 \leq \ldots \leq x_n$ be a sample of size n arranged in ascending order. Denote the estimated qth centile of the x's as c_q. We require that $0 < q < 100$. Let $R = (n+1)q/100$ have integer part r and fractional part f, that is, $r = \text{int}(R)$ and $f = R - r$. (If R is itself an integer, then $r = R$ and $f = 0$.) Note that $0 \leq r \leq n$. For convenience, define $x_0 = x_1$ and $x_{n+1} = x_n$. Then C_q is estimated by

$$c_q = x_r + f \times (x_{r+1} - x_r),$$

that is, c_q is a weighted average of x_r and x_{r+1}. Loosely speaking, a (conservative) p% confidence interval for C_q involves finding the observations ranked t and u which correspond respectively to the $\alpha = (100-p)/200$ and $1-\alpha$ quantiles of a binomial distribution with parameters n and $q/100$, i.e., $\text{B}(n, q/100)$. More precisely, define the ith value ($i = 0, \ldots, n$) of the cumulative binomial distribution function to be $F_i = \text{P}(X \leq i)$, where X has distribution $\text{B}(n, q/100)$. For convenience, let $F_{-1} = 0$ and $F_{n+1} = 1$. Then t is found such that $F_t \leq \alpha$ and $F_{t+1} > \alpha$, and u is found such that $1 - F_u \leq \alpha$ and $1 - F_{u-1} > \alpha$.

With the `cci` option in force, the (conservative) confidence interval is (x_{t+1}, x_{u+1}) and its actual coverage is $F_u - F_t$.

The default case uses linear interpolation on the F_i as follows. Let

$$\begin{aligned} g &= (\alpha - F_t)/(F_{t+1} - F_t), \\ h &= [\alpha - (1 - F_u)]/[(1 - F_u) - (1 - F_{u-1})] \\ &= (\alpha - 1 + F_u)/(F_{u-1} - F_u). \end{aligned}$$

Then the interpolated lower and upper confidence limits (c_{qL}, c_{qU}) for C_q are

$$\begin{aligned} c_{qL} &= x_{t+1} + g \times (x_{t+2} - x_{t+1}) \\ c_{qU} &= x_{u+1} - h \times (x_{u+1} - x_u). \end{aligned}$$

For example, suppose we want a 95% confidence interval for the median of a sample of size 13. So $n = 13$, $q = 50$, $p = 95$, $\alpha = .025$, $R = 14 \times 50/100 = 7$, $f = 0$. The median is therefore the 7th observation. Some example data x_i and the values of F_i are as follows:

i	F_i	$1 - F_i$	x_i	i	F_i	$1 - F_i$	x_i
0	0.0001	0.9999	–	7	0.7095	0.2905	33
1	0.0017	0.9983	5	8	0.8666	0.1334	37
2	0.0112	0.9888	7	9	0.9539	0.0461	45
3	0.0461	0.9539	10	10	0.9888	0.0112	59
4	0.1334	0.8666	15	11	0.9983	0.0017	77
5	0.2905	0.7095	23	12	0.9999	0.0001	104
6	0.5000	0.5000	28	13	1.0000	0.0000	211

The median is $x_7 = 33$. Also, $F_2 \leq .025$ and $F_3 > .025$ so $t = 2$; $1 - F_{10} \leq .025$ and $1 - F_q > .025$ so $u = 10$. The conservative confidence interval is therefore

$$(c_{50L}, c_{50U}) = (x_3, x_{11}) = (10, 77),$$

with actual coverage $F_{10} - F_2 = .9888 - .0112 = .9776$ (97.8% confidence). For the interpolation calculation, we have

$$\begin{aligned} g &= (.025 - .0112)/(.0461 - .0112) = .395, \\ h &= (.025 - 1 + .9888)/(.0998 - .9539) = .395. \end{aligned}$$

So

$$\begin{aligned} c_{50L} &= x_3 + .395 \times (x_4 - x_3) = 10 + .395 \times 5 = 11.98, \\ c_{50U} &= x_{11} - .395 \times (x_{11} - x_{10}) = 77 - .395 \times 18 = 69.89. \end{aligned}$$

`normal` case. The value of c_q is as above. Its s.e. is given by the formula

$$s_q = \sqrt{q(100-q)} \Big/ \Big[100n\text{Z}(c_q; \bar{x}, s)\Big]$$

where $\bar{x}$ and s are the mean and s.d. of the x_i and

$$\mathrm{Z}(Y;\mu,\sigma) = \left[1\Big/\sqrt{2\pi\sigma^2}\right] e^{-(Y-\mu)^2/2\sigma}$$

is the density function of a normally distributed variable Y with mean μ and s.d. σ. The confidence interval for C_q is $(c_q - z_{100(1-\alpha)} s_q, c_q + z_{100(1-\alpha)} s_q)$.

`meansd` case. The value of c_q is $\bar{x} + z_q \times s$. Its s.e. is given by the formula

$$s_q^\star = s\sqrt{1/n + z_q^2/(2n-2)}.$$

The confidence interval for C_q is $(c_q - z_{100(1-\alpha)} \times s_q^\star, c_q + z_{100(1-\alpha)} \times s_q^\star)$.

References

Conover, W. J. 1980. *Practical Nonparametric Statistics.* 2d ed. New York: John Wiley & Sons.

Kendall, M. G. and A. Stuart. 1969. *The Advanced Theory of Statistics, Vol. I.* 3d ed. London: Griffin.

Mood, A. M. and F. A. Graybill. 1963. *Introduction to the Theory of Statistics.* 2d ed. New York: McGraw–Hill.

sg8 Probability weighting

William Rogers, CRC, FAX 310-393-7551

The introduction of Stata 3.0 included what to many is a new kind of weight, the `pweight` or sampling weight, along with the more well-known `fweight`s (frequency weights) and `aweight`s (analytic weights).

`fweight`s are conceptually easy—you have data where each observation reflects one or more real observations. `fweight`s are most easily thought of as a data-compression scheme. An observation might record income, age, etc., and a weight, say 5, meaning that this observation really reflects 5 people with exactly the same income, age, etc. The results of estimating a frequency-weighted regression are exactly the same as duplicating each observation so that it appears in the data as many times as it should and then estimating the unweighted regression. There are really no statistics here; just data management.

`aweight`s do solve a real statistical problem. The data you are analyzing reflect averages. You do not know each individual's income, age, etc., you know the average income in data grouped on age, etc. Weighting is important when analyzing such data because the accuracy of the averages increases as the sample size over which the average was calculated increases. An observation based on averages of 1,000 people is relatively more important than an observation in the same data based on an average of 5 people. In a regression context, for instance, mispredicting the 1,000-person average is far more serious than mispredicting, by the same amount, the 5-person average.

`pweight`s solve another statistical problem. You have data in which each observation is an individual—not a group average—it is merely that some individuals were more likely to appear in your data than others. An observation with a small probability of appearing, and therefore a large `pweight` (which is the inverse of the sampling probability) is not in any sense a more accurate measurement of, say, earnings, than is the earnings recorded in an observation more likely to appear in the data, and therefore the adjustment made to standard errors is in no way related to the adjustment made to standard errors in the `aweight` case. What is related is the adjustment made to the mean parameter estimate—`aweight`s and `pweight`s adjust means and regression coefficients in the same way. An observation with a high weight contributes more information on the mean because, in the case of `aweight`s, it is a more precise estimate and, in the case of `pweight`s, because it was less likely to be sampled and is therefore reflective of a larger underlying population.

`pweight`ed data can arise both intentionally and unintentionally. One might intentionally oversample blacks relative to whites (as is common in many social-science surveys) or the sick relative to the well (as is common in many epidemiological studies). Alternatively, imagine a survey that is administered by mail and also imagine, as is typical, that certain types of respondents are found, *ex post*, to have been more likely to respond than others. The group less likely to respond thus reflects a larger underlying population, but the measurements on the individuals we do have are no more (or less) accurate than any of our other measurements.

When one begins to consider how a sample is obtained, another issue arises, that of clustered sampling, an issue related to, but conceptually different from, `pweight`s. Let me first describe how a sample might come to be clustered and then consider the statistical issues of such clustering.

Assume you are going to survey a population and that you will do this by sending interviewers into the field. It will be more convenient (i.e., cheaper) if each interviewer can interview persons who are geographically close to each other, so you might choose geographical areas at random and then choose persons at random within the geographical area. If you do, the sample is said to be geographically clustered.

The problem in clustered samples is that they may be overly homogeneous. Let's assume that, in our example above, we choose neighborhoods at random and then choose, at random, persons within neighborhood. Assume also that we want to predict the mean of earnings for all persons living in the overall sample area. The average earnings that we observe in your sample, calculated in the standard unweighted way (summing earnings and dividing by sample size) is a fine (unbiased) estimate of the underlying mean of earnings. The estimated standard error of the mean, calculated in the unweighted way, however, is probably not a good estimate of the accuracy of our measurement of the mean. Persons who live in the same neighborhood are similar to each other, especially in earnings. Therefore, the variance of earnings within neighborhood is too small and we will underpredict the standard error of earnings.

Another way of thinking about this is that adding one more person from a neighborhood already existing in our sample does not add as much information as adding another person in a new neighborhood. Say our data contains N persons from K neighborhoods. Using standard statistical formulas, it is the N that we use in calculating the standard error of the mean. If neighborhoods were perfectly homogenous (all residents within a neighborhood had identical earnings), it should be the K that should enter our statistical formulas. We have K true observations since, once we obtain one person from the neighborhood, we know everything the neighborhood has to tell us. In reality, the neighborhoods are not perfectly homogenous, and the effective number of observations is somewhere between K and N.

Dealing with sampling issues

Stata's `pweight` modifier and "Huber" commands (see [5s] huber) deal with probability-weighted and clustered samples.

Without getting into the mathematics, it is important to understand that the formulas for analytically weighted data do not handle the problem *even though many researchers act as if they do.* Most statistical packages allow only two kinds of weights—frequency and analytical—and the non-software-developer researcher is often forced to treat probability-weighted samples as if they were analytically weighted. The justification for this, other than convenience, is that the formulas are mathematically related in that the adjustments made to the mean (or the estimated coefficients in a regression setting) are the same in either case. However, it is not the adjustment to the mean that is important—that is solely an efficiency issue. It is the adjustment to standard errors that is vitally important and, on these grounds, the adjustments are quite different.

How one deals with probability-weighted samples has generated much philosophical debate and is polarized along disciplinary lines. Survey statisticians use sampling weights and econometricians and sociologists, for instance, rarely use them (or even know about them).

Econometricians begin with the following meta-theorem. You must weight when calculating means but it is not necessary to weight when estimating regression coefficients. The following is the argument: You begin with the behavioral model

$$y_j = \mathbf{x}_j\beta + \epsilon_j$$

and *assume* that the ϵ_j are independent and identically distributed for all observations in the sample—tantamount to assuming that the model is correct. Under that assumption, no weighting is necessary, one simply estimates the coefficients and the standard errors with regression.

The Survey Statistician, on the other hand, typically has little interest in fitting a complete behavioral model—he tends to be more interested in describing the state of the world as it is today. For instance, the Survey Statistician may be interested in means and their comparison (as in public opinion polls). Although the goals and statistics may be "simple," (e.g. comparing the earnings of men and women), such comparisons are sometimes cast in a regression framework to allow for adjustment of other effects. Whether the Survey Statistician simply compares means or estimates regressions, his philosophy is to use a sampling weight which is inversely proportional to the probability of being sampled and to calculate a standard error that takes the weighting explicitly into account.

Let us consider a real problem: we want to estimate the difference in earnings between men and women, and we have data that contains 50% white respondents and 50% black respondents. The Econometrician writes down an earnings model, including all the things that might affect earnings. He estimates the model without weights. Let us first assume our Econometrician is sloppy. His earnings equation includes race, sex, education, age, and so on. He finds that women earn 83% the amount earned by what appears to be an equivalent male. Now let's assume our Econometrician is more careful. He examines the data carefully and discovers that the earnings difference of men and women is different for blacks and whites: black women earn roughly the same amount as black males, but white women earn 66% of the amount earned by white males. He therefore reports that the earnings ratios are between 66 and 100%. Given that blacks comprise roughly 12% of the population, he reports an average earnings ratio of 70%.

The Sampling Statistician runs a very simple regression, earnings on gender, and includes sampling weights to account for the oversampling of blacks in the data. He reports the difference that the ratio of female to male earnings is 70%.

The following are worth noting: (1) Using unweighted regression, the Econometrician produced an incorrect answer when sloppy—that is, when the model was wrong; (2) the Sampling Statistician's problem was easier than that of the Econometrician and he had no chance of producing the wrong answer; (3) the careful Econometrician, on the other hand, not only produced the right answer, but produced an answer that contained more information than that produced by the Sampling Statistician.

Let us now compare the approaches of the Econometrician and the Survey Statistician on the issue of weights. The Econometrician can be proved wrong by the data; given a set of sampling probabilities, the Econometrician may find that they are related to the residual and may also discover that there is no set of independent variables in his model to free the residual of this "unexplainable" correlation. On the other hand, the Sampling Statistician may be confronted by a sensitivity analysis showing that weights for which he has so carefully accounted do not matter, but in that case, he will merely argue that the inference can only be made under the assumption that they do matter and add that we merely happened to be lucky this time. The Sampling Statistician will argue that if the Econometrician wants to estimate behavioral models, that's fine, but that is still no reason for ignoring the weights. If the Econometrician wants to perform a sensitivity analysis *ex post* and finds that the weights do not matter, that's fine too. But if the Econometrician simply ignores the weights, that is not fine.

So far, we have not really distinguished between sampling weights and clustering. Mathematically, there are actually two issues. Sampling weights have to do with the issue that two observations do not have the same probability of appearing in the data. Clustering has to do with the issue that two observations may be somehow related in a way not otherwise described by the variables in the data. To adjust standard errors for both, the estimator needs to know the sampling weights and the cluster to which each observation belongs.

However, the Econometrician and the Sampling Statistician again have a characteristically different approach. The Econometrician treats the clustering as if it were another element that needs to be modeled, and then proceeds as if the revised model is correct. So he may introduce heterogeneity parameters and try to estimate them. "Variance components" models are one way this is done. The Sampling Statistician wants his regression coefficients to reflect means or differences in means. He is more interested in correcting the standard errors of the analysis he is already doing.

In attempting to estimate efficiently, the sloppy Econometrician may unwittingly downweight large clusters, since they have less information per observation. From the Survey Statistician's point of view, this potentially introduces a bias. However, the careful Econometrician gains additional information on relationships among clustered observations that may be useful in understanding the phenomenon under study.

How do you know where a given analysis fits, philosophically speaking? Econometricians sometimes use (or are forced by data availability to use) reduced form models, in which case they should behave as if they were Sampling Statisticians. Sampling Statisticians may sometimes use maximum-likelihood methods, but that does not make them Econometricians (logit analysis, for example, can be a fancy way to compare proportions with adjustment). In short, if the analysis is anything less than an all-out attempt at behavioral modeling, or if weighted analysis changes the results substantially, it needs to be considered from the viewpoint of the Sampling Statistician.

This is where Huber's method (implemented in Stata; see [5s] huber) is helpful. These commands take the philosophical approach of the Sampling Statistician. With the Huber method, weighted or clustered problems can be estimated using regression, logit, or probit estimation techniques. The calculations differ from the `aweighted` answers only in their standard errors.

If you have sampling weights or clusters, even if you think you have the "right" model, Huber's method is one way you can check your assumption. If the answers are substantially different than your weighted analysis, you know you have a problem. If your goal was to estimate a reduced form in any case, your problem is also solved. If your goal was to estimate a full behavioral model, you now know its time to reconsider the functional form, the hypothesized variables, or selection effects.

smv5.1 Loglinear analysis of cross classifications, update

D. H. Judson, DecisionQuest, 1013 James St., Newberg, OR 97132

I have made several changes to the `loglin` command (Judson 1992) to improve its operation and make it act like other estimation commands. See [4] estimate (vol. 1, p. 263) for a description of features common to estimation commands. The new syntax for the command is

```
loglin depvar varlist [weight] [if exp] [in range], fit(margins to be fitted)
    [ anova keep resid collapse ltol(#) iter(#) offset(#) level(#) irr ]
```

The changes to the `loglin` command include

1. It now works appropriately with version 3.0;
2. you can recall the results of the previous loglinear analysis by typing `loglin` without arguments;
3. tests can be performed using the `test` command;
4. weights are handled as frequency weights in version 3.0 syntax;
5. you can use `predict` after estimation to generate the log of the expected cell frequencies, if you specify the `keep` option;
6. you can obtain the variance–covariance matrix of the estimators using `correlate, _coef`;
7. you can refer to coefficients and standard errors in expressions, if you specify the `keep` option;
8. your data set is now protected from change in the event of an error or if you press the *Break* key (see [4] program_fragments, vol. 1, p. 305);
9. the `resid` option no longer leaves `resid`, `stresid`, and `cellhat` in your data. Instead, it just prints out the list of cell frequencies, expected cell frequencies, residuals, and standardized residuals;
10. `loglin` no longer calls `poilog` for estimation. Instead, it creates the design matrix and passes that matrix to `poisson`;
11. two options from `poisson`, the `irr` and `level` options, are now options for `loglin`;
12. and finally, a substantive change:

In the previous version, if you did not specify a method for constraints, regression-like constraints were assumed, which dropped the first `level` of each indicator variable for each margin. If you specified the `anova` option, ANOVA-like constraints were assumed, and the first level was set to be equal to -1 times the sum of all other levels.

There are no changes to the regression-like constraints. For ANOVA-like constraints, however, the new version is modified to drop the last level and set the last level equal to -1 times the sum of all other levels. It is nonstandard, although perfectly legal, to drop the first level. This change will make the results more comparable to other packages. At this point, I have not implemented any method to choose which level to drop, so if you prefer dropping the first level, you are (temporarily) out of luck. I'd be happy to hear any debate in the STB regarding which method is preferable.

References

Judson, D. H. 1992. smv5: Performing loglinear analysis of cross classifications. *Stata Technical Bulletin* 6: 7–17.

sqv4 Calculation of the deviance goodness-of-fit statistic after logistic

Joseph Hilbe, Editor, STB, FAX 602-860-1446

The deviance goodness-of-fit, DEV, is a summary statistic based on deviance residuals. It is interpreted in a manner similar to the Pearson χ^2 goodness-of-fit statistic (see [5s] logistic). Mathematically, DEV is calculated

$$\text{DEV} = \sum_{j=1}^{J} d^2$$

where d is the deviance residual value.

It is possible to generate this statistic by hand after `logistic`:

```
. lpredict num, number
. lpredict dev, deviance
. sort num
. by num: gen dev2=sum(dev*dev) if _n==1
. egen DEV=sum(dev2)
```

The statistic follows a χ^2 distribution with degrees of freedom equal to $J-(p+1)$, where J is the number of distinct covariate patterns and p is the number of predictor variables in the model. The `lfit` command will provide you with the number of model covariate patterns as well as the correct degrees of freedom. However, if you have used the above command list to obtain DEV, you can also type '`display num[_N]`' to determine the number of covariate patterns.

I have created a command to be used after `logistic`, similar to `lfit`, that yields the DEV statistic, significance, etc. Simply type `ldev`. (The program is on the STB-8 diskette.)

```
. logistic low age amoke race2 race3
  (output omitted)
. ldev
Logistic estimate for low, goodness-of-fit test
        no. of observations  =       189
   no. of covariate patterns =        82
           Deviance chi2(77) =        96.50
                    P>chi2 =         0.0658
```

ssi3 Continuous dynamic system modeling and simulation with Stata

Francesco Danuso, Dipartimento di Produzione Vegetale, Udine, Italy. FAX (011)-39-432-558603

Introduction

Any set composed of elements that exchange material and information and is able to regulate itself by feedback loops, can be defined as a *system*. This paper provides and describes the utilization of a new Stata command (`simula.ado`) that allows the simulation of continuous dynamic systems. *Simulation* is the calculation process performed on a mathematical model for representing a particular system behavior. Models, coded in observations of a string variable in a `.dta` file, are parsed and then simulated by `simula`. `simula` is a small language pseudo-interpreter for model listings, providing a simple tool to study the dynamic behavior of systems.

Overview on system modeling and simulation

Often, systems show complex variations that are not easily understood. *System analysis* is a powerful and general method devised for the system representation and the simulation of the dynamics. This method leads to system *modeling* by means of mathematical equations. Modeling techniques have been applied to systems of very different kinds. For example, many applications are in the social-economical (Meadows and Meadows 1972; Meadows et al. 1973), ecological (Brennan et al. 1970; de Wit and Goudriaan 1978), physiological (Smerage 1979), and agricultural (Dent and Blackie 1978) fields.

System modeling has been useful both for many practical and research purposes (examples of the former include system behavior forecasting, system control, and stability under disturbances; examples of the latter include exploration of hypothetical systems, thorough and critical review of knowledge concerning the system, and simulation experiments). The major contribution to system analysis and modeling was made by J. W. Forrester (1968) who stated the "Principles of systems." His approach deals particularly with "conservative" systems, in which matter and energy are neither created nor destroyed. His viewpoint, following a hydraulic analogy, conceives the systems conditions (called system states or levels) as water levels in reservoirs. Levels are the "memories" of the system and can vary, with time, in relation to their incoming and outcoming fluxes. The system conditions (state variables), and the parameters and the variables generated outside the system bounds (external inputs) control the fluxes (rate variables). Parameters are quantities that maintain a constant value during a simulation.

System analysis identifies the state variables, the change rate of each state variable (rates) and the relationships among state variables, rate variables and exogenous variables. Calculations of state variables proceed, starting from their initial values, by accumulation (integration) for small time increments of the incoming and outcoming rates. Rate variables depend only on state and exogenous variables but not on the other rates; state variables depend on rate variables.

When the system conditions control their rate of change, there is a feedback loop; this can be positive, when the state variable diverges from an initial value, or negative if the state variable converges toward a goal. "Delays" are other important aspects of the system dynamics. The integration of rates in a state variable can happen after a time span (material delay) or the control mechanism perceives the system conditions which retard (information delay: the present rate depends on past levels).

The application of the system analysis method does not require great mathematical skill and is suitable for all dynamic and continuous systems. Generally, these models are deterministic and the simulation results are the same if initial values, parameters and exogenous variables are the same. It is also possible to simulate stochastic behaviors by sampling the rate variable values from probability distributions or by superimposing a noise to the rate value.

System analysis begins with (a) structure identification, proceeds to (b) identification of information links and rate calculations, followed by (c) model coding and (d) specifying parameters and initial values so that the simulation can be performed.

A) System structure identification

The first phase pertains to the identifications of the variables that represent the state of the system, at least in relation to the model purposes. These are named state or level variables. In a relation diagram, the symbol used to represent state variables is the box (Figure 1). We must also define all the incoming and outcoming fluxes (rate variables) for each state variable. The model state variables are computed by the state equations. The structure of a state equation is

$$\text{svar}_{i,t} = \text{svar}_{i,t-1} + dt \times (\pm\text{rvar}_{1,t-1} \pm \text{rvar}_{2,t-1} \pm \ldots)$$

where svar_i is a state variable and appears on both sides of the equation, rvar_j are rate variables added or subtracted to the state variable, and dt the time step for the rate integration in the state variables. The above equation means that, for each instant t, the value of the state variable is obtained algebraically by adding the variation rates in the period $(t-1,t)$ to the previous value of the state variable at time $t-1$. The value of the rate variable in period $(t-1,t)$ is a function of the state variable values at time $t-1$. If dt is small enough, $\text{rvar}_{j,t-1}$ is a good approximation of the mean of rvar_j in the $(t-1,t)$ period.

B) Identification of information links and rate calculations

To calculate the state equations, we define the formulations for all the rate variables contained in the state equations. In the relational diagram, a solid arrow represents the material fluxes and a valve symbol the rate variables (Figure 1). The structure of the rate equation might be

$$\text{rvar}_{t-1} = (1/\text{TC}) \times \text{DForce}_{t-1}$$

where rvar is the name of a rate variable, TC is the time constant (in time units) and DForce is the part of the equation representing the quantity to which the rate is proportional. When a rate equation is a function of a state variable, an "information link" is said to exist. The information links are generally represented by dashed arrows exiting a state-variable box and entering a valve (the rate variable symbol) (Figure 1).

If an information link controlling a rate derives from the same state variable modified by the rates itself, there is said to be a "feedback." If DForce contains a "goal," the feedback is negative; if a goal does not exist (ex: DForce = svar), the feedback is positive and the trend of the state variable is auto-catalytic, with an increasing exponential trend.

The concepts that it is possible to find in a rate equation are

1. A goal: A parameter that represents the value toward which the state variable converges.
2. A way to express the "divergence" between the goal and the observed system condition: Depending on the system kind and the field, this divergence takes different names: error, driving force, potential differential, tension.
3. An apparent condition of the system: This can be the present value of a state variable or (if there are delays in the information fluxes) the values of that state variable in a previous time. In any case, the apparent conditions of the system (not the real conditions) determine the rate values. For example, in a prey-predator ecological system, the birth rate of predator is a function of the prey density. Really, there is a material delay due to the development time: the rate quantity is integrated to the state variable after a certain time. If the system has control mechanisms (that is has a goal) depending on the state variable values, it can detect the real "state" of the system with a delay. The state value that enters the rate equation is a past value of the state variables.
4. An effect of the exogenous variables: Systems can be sensitive to the conditions outside the defined system bound. In this case the model should also consider some external variables, named "exogenous variables." They contribute to the formulation of the rate and auxiliary equations.
5. An indication about the response intensity to the divergence between apparent states and goals: If the dimension of this parameter is time then it is called a "time coefficient" (TC). Time coefficients, depending on the model, take the names of "resistance," "relaxation time," "average residence time," "average departure or transit time" or "average life time." If the intensity parameter is expressed in quantity per unit time its meanings can be that of "relative growth rate," "decay

coefficient," "conductance coefficient," and so on (Ferrari 1978). TC relates to the well-known "half-life time" and "doubling time" concepts by the relations:

$$\text{half-life time} = \log(1/2) \times \text{TC}$$

$$\text{doubling time} = \log(2) \times \text{TC}$$

6. Auxiliary variables: Rates can be also functions of "auxiliary variables." Using them, the formulation of the rate equations becomes simpler, also permitting the linking of state variables with different measurement units. An auxiliary variable is a function of state or exogenous variables that relates fluxes of different materials, or transforms state variables in some way.

C) Model coding and simulation

To obtain simulations of real (or hypothetical) systems, it is necessary to code the mathematical model in a computer language, to perform the numerical integration and to obtain the evolution of the system states with time. The system simulation is, in effect, a method for the numerical solution of the state equations.

Many programs are available for this task. For example, DINAMO (Forrester 1968; Richardson and Pugh 1981), CSMP (IBM 1972), ACSL (ACSL 1987) and PCSMP (Jansen et al. 1988) are the most known. The equations can also be solved by using routines coded in a lower level language (as FORTRAN, C, PASCAL, BASIC, etc.).

A useful feature of models coded for specific languages (ex: DYNAMO, CSMP, and `simula`, too) is the possibility to represent all the system aspects in a conceptual rather than computational order. This makes the code more readable and errors easier to detect.

The author has implemented some of the simulation possibilities of the specific languages in a program (`simula`) that, being a Stata ado-file, looks like a real Stata command. This command works as a small simulation language, interpreting lists of statements written following a few syntax rules. Simulations are then performed. Modification of parameters and initial values is allowed. Moreover, the simulation results obtained with `simula` can be treated with all the other Stata commands. The simulation with the `simula` command is not very fast when compared with dedicated or low level languages. However, for not large models and for didactic purposes these do not seem big problems, at least when one takes into account its ease of use. `simula` could also be useful to evaluate small models before linking them to larger models.

D) Parameters and initial values

The adopted parameter values determine, in addition to the model structure and equations, the model behavior. Parameters are constant (or so considered) for the entire simulation period. Changing the parameter values also changes the model response. This allows fitting of the model to different situations by the calibration. Parameter values can be obtained by experience, by literature review, or by experiments. Also, initial conditions of the state variables are specified in the model. Sometimes, depending on the model structure, these values can be important for the model's dynamics. For a more detailed and complete description of system analysis and modeling see, for example, Forrester (1968), Ferrari (1978) and Richardson and Pugh (1981).

Procedure for system simulation with Stata

The overall procedure can be divided into four steps:

1. Development of equations governing the model (state, rate and auxiliary) and identification of the parameters and initial values.
2. Model coding into a `.dta` file. In this phase, it is possible also to state parameter and initial values.
3. Preparing of a `.dta` file containing the exogenous variables, if necessary.
4. Using the `simula` command. The command permits specification of the time solution step, the initial time, the simulation duration, the required exogenous variable's file name, the parameter values, the initial conditions and the graphic options for the results.

Step 1: Model development

The modeling process consists of the following steps: 1) identification of the relevant system variables for the representation of the system (state variables); 2) identification of the fluxes (rate variables) which determine the “speed” of the state variable variations. The “state equations” link the rate variables to the state variables; 3) development of the relations for the rates. Rates are computed as functions of the state and exogenous variables (not other rates). For a clearer analysis of the system, the rate equations can be also functions of auxiliary variables.

The third step is the most important for the modeling process. In fact, the evolution of the state variables derives from the integration of the rate equations, starting from certain “initial conditions.”

Example of model development

Let us consider a classical Volterra model on prey-predator interactions in an ecological system (see Maynard 1974 for details) and assume that the prey is a herbivorous. Two state variables describe the system conditions: the prey density in the ecosystem (X, as number of individuals per unit area) and its predator density (Y, as number of individuals per unit area). For the model development, we make the following assumptions:

1. The birth rate of the prey, without predator and without limiting factors, is a constant fraction of the prey density.
2. The environmental limits determine a maximum value for the prey density that is not possible to exceed. This limit is generally indicated as “carrying capacity” (C_c).
3. The predation rate is a fixed fraction of all the possible encounters between prey and predator (represented by the product $X \times Y$).
4. The death rate of predator is a constant fraction of the predator density.
5. The birth rate of predator is a fraction of Y density but the predator birth rate tends to zero when the X density approaches the Y density.
6. In addition, to show the use of exogenous variables, we make a further assumption: if the prey is a herbivorous, the carrying capacity of the environment (C_c) depends on the annual grass production of the range. C_c, in turn, could be dependent on the annual ratio (total rainfalls/average temperature) by the equation $C_c = 500 + K_1 \times (\text{rain}/\text{temp})$. Rain and temp are exogenous variables.

The differential equations describing the change with time of X and Y are

$$(1)\quad \underset{\substack{\text{variation}\\ \text{rate of}\\ X}}{dX/dt} = \underset{\substack{X\\ \text{birth}\\ \text{rate}}}{K_{bx} \times X} \times \underset{\substack{\text{effect of}\\ \text{carrying}\\ \text{capacity}}}{(C_c - X)/C_c} \quad \underset{\substack{\text{death rate}\\ \text{for}\\ \text{predation}}}{-K_{mx} \times (X \times Y)}$$

$$(2)\quad \underset{\substack{\text{variation}\\ \text{rate of}\\ Y}}{dY/dt} = \underset{\substack{Y\\ \text{death}\\ \text{rate}}}{K_{my} \times Y} \quad \underset{\substack{\text{effect of predation}\\ \text{on predator birth}}}{+K_{by} \times Y \times (X - Y)}$$

where $1/K_{bx}$, $1/K_{mx}$, $1/K_{my}$, and $1/K_{by}$ are the time coefficients of the rates.

Figure 2 shows the relational diagram of the described system, following the hydraulic analogy proposed by Forrester (1968). Let the initial values for prey and predator density in the area be 1000 and 100 individuals, respectively. K_{bx} is the birth coefficient of the prey population, expressed as fraction of X birth per year, and fixed to 0.05. The birth coefficient for predator is $K_{by} = 0.006$ and represents the efficiency of the prey-predator conversion. The mortality coefficients for prey and predator are, respectively, $K_{mx} = 0.001$ and $K_{my} = 0.05$. The coefficient for grass production (K_1) equals 80.

Considering a time lag, from the predation time to the new predator births, due to the development time (Devt), we can build a more realistic model by introducing a delay in the birth rate of the equation (2). The births are delayed in respect to the predation (material delay); the integrating rates are then dependent on the past values of X and Y. Equation (2) becomes

$$(2)\quad \underset{\substack{\text{variation}\\ \text{rate of}\\ Y}}{dY/dt} = \underset{\substack{Y\\ \text{death}\\ \text{rate}}}{K_{my} \times Y} \underset{\substack{\text{effect of predation}\\ \text{on predator birth}\\ \text{with material delay}}}{+K_{by} \times Y \times (X_{t-Devt} - Y_{t-Devt})}$$

where t is the time index and Devt the material delay time due to the development time of predator. We assume that the material delay for the predator birth is 0.5 year.

Step 2: Model coding

The model equations have to be written in a form understandable by the `simula` command. `simula` uses two kinds of files: the *model file* and the *exogenous variable file*. The models must be saved in the a string variable named `_model` in the model file. There are six kinds of statements recognized by `simula`: state equations (denoted by `S`), auxiliary equations (denoted by `A`), rate equations (`R`), exogenous variable declarations (`E`), initial value assignments to the state variables (`I`), and parameter value assignments (`P`).

The above symbols (`S`, `A`, `R`, `E`, `I`, and `P`), placed as the first character of each line identifies the statement type. `simula` parses the statements in a way requiring the following rules:

1. Line structure: each statement is composed of three parts:
   a. a statement identifier (the capital letters `S`, `A`, `R`, `E`, `I`, or `P`);
   b. the statement itself (equation, declaration, assignment); blanks inside are not allowed;
   c. a statement label, which is optional and must be separated by a blank, at least, from the statement.

2. Comment lines: Any character different from `S`, `A`, `R`, `E`, `I`, and `P` in the statement identifier position, denotes a comment line.

3. Line order: The model list can contain the statements in any order. The only exceptions are
   a. when an auxiliary variable (e.g., `aux2`) depends on another auxiliary variable (e.g., `aux1`); in this case, the equation for `aux1` must precede the `aux2` equation.
   b. initial values declaration must follow the related state equation.

4. All the statement kinds allow only one expression/declaration/assignment per line. For example, it's not possible to write the lines: "`E rain temp`", "`I X=100 Y=50`", "`P K1=5 K2=7`; they should be rewritten as "`E rain`", "`E temp`", "`I X=100`", etc.

5. Constant values: The `S`, `A`, `R`, `I` and `P` statements allow constant values. Arguments for the special functions are to be parameters or auxiliary variable (ex: the delay time) and are not constant.

6. Special functions: Right now `simula` has three special functions: `IDL` (information delay), `MDL` (material delay), and `SWTC` (switching a rate on/off). Functions can only be specified in the rate equations; arguments must be parameters or auxiliary variables.

7. Blank lines: Between two statements it is possible to insert up to, but not more than, one blank line. This insertion could be useful to make clear the reading of the model.

8. Parameters and initial values: It is not necessary to declare all the parameter and initial values in the `_model` variable, but this is a good practice. Whether specified or not, it is possible to assign those values as an option in the `simula` command. (Of course, all parameter and initial values need to be declared eventually so, if they are not declared in the model file, they must be declared when the `simula` command is given.)

9. Statement syntax: The syntax for each type of statement is

   `S` *svar1*`=`*svar1*`+dt*(`*rvar1* ± *rvar2* ± ...`)` *label*
   `A` *avar*`=`*exp label*
   `R` *rvar*`=`*exp label*
   `E` *evar label*
   `I` *svar*`=`*# label*
   `I` *svar*`=`*f*`(`*p*`)` *label*
   `P` *pname*`=`*# label*
   `P` *pname*`=`*f*`(`*p*`) it label`

where

svar	state variable name
avar	auxiliary variable name
rvar	rate variable name
evar	exogenous variable name
pname	parameter name
f(*p*)	function of parameters or other parameters, already declared in the model code list
#	numeric constant
`dt`	time step for the equation integration; it is inserted only to identify a state equation but, really, any character set differing by '`+`' or '`*`' is allowed.
exp	any algebraic expression containing #, *pname*, *evar*, *avar*, and *svar*. In the expression, all the Stata functions can be used. For example, it is possible to use the random generator function `uniform()`
label	a descriptor of the statement (optional)

10) Special functions: `simula` recognizes some special functions that, indicated in the rate equations, perform some specific tasks. These can be indicated only in the rate equations: moreover, each rate equation allows only one function. The syntax of the functions is

    `R` *rvar*`=`*fcn*`[`*exp*`,`*arg*`,`*arg*`,`...`]`

    where *fcn* is one of the special functions, *exp* is an algebraic expression and *arg* are the arguments needed by the function itself. The available functions are, right now:

    `IDL` information delay function. The syntax is "`R` *rvar*`=IDL[`*exp*`,`*dly*`]`", where *dly* is the delay time. The effect on *rvar* is $rvar_t = \exp(t - dly)$ if $t \geq dly$ and $rvar_t = 0$ otherwise.

    `MDL` material delay function; this function has the same effect of a further level inserted in the flux, but not available to the user. The syntax is "`R` *rvar*`=MDL[`*exp*`,`*dly*`]`", where *dly* is the delay time. The effect on *rvar* is $rvar_t = rvar_t + \exp(t - dly)$ if $t \geq dly$ and $rvar_t = 0$ otherwise. *dly* is expressed in time units. It must be a parameter or an auxiliary variable name (not a constant or expression).

    `SWTC` switch to rate on/off. The syntax is "`R` *rvar*`=SWTC[`*exp*`,`*cvar*`,`*von*`,`*voff*`]`"; sets the rate to zero if a control variable *cvar* is below a low threshold value *von* or reaches an upper threshold *voff*. The results are $rvar = 0$ if $cvar \leq von$; $rvar = exp$ if $cvar > von$ and $cvar < voff$; and $rvar = 0$ if $cvar \geq voff$. `SWTC` can be used to perform the tasks:

    Represent a state variable with a physical limit and with the inflow rate independent from the level: *von* is set to 0 and *voff* to the maximum level capacity; *cvar* is the level itself.

    Represent a process starting and/or stopping at some defined times: the control variable is `_time`; *von* and *voff* are the starting and ending times.

    Give a pulse of material at a specific time only ($von = voff$).

    Simulate processes conditionally associated to other variables.

11) Simulation set-up: Time step, time span and time of start are defined by the command and not in the program.

Example of model coding

The `.dct` file containing a Volterra example model (without delay):

```
dictionary {
* file  "volmod1.dct"
* example model for "simula" STATA command  (4/11/91)
* This is a kind of a VOLTERRA model - without delay
* Ref. J. Maynard Smith, 1974. Models in ecology. Cambridge University Press.
str70 _model   "Model statements"
}
"* Prey Predator Volterra model - without delay                 "
"* STATE equations -----------                                  "
"S X=X+dt*(Bx-Dx)               Prey population (n)             "
"S Y=Y+dt*(-Dy+By)              Predator population (n)         "
"* RATE equations ------------                                  "
"R Bx=Kbx*(Cc-X)                Birth rate of prey   (n/year)   "
"R Dx=Kmx*X*Y                   Death rate of prey   (n/year)   "
"R Dy=Kmy*Y                     Death rate of predator (n/year) "
"R By=Kby*Y*(X-Y)               Birth rate of predator (n/year) "
```

```
"* AUXILIARY equations -------                                 "
"A Cc=500+K1*rain/temp          Environm. carrying capacity (n) "
"* e.g. related to the ecosystem grass production              "
"E temp                         Aver. annual temperature (C)    "
"E rain                         Aver. annual rainfall (mm)      "
"I X=1000                       Prey initial density            "
"I Y=100                        Predator initial density        "
"P Kbx=0.05                     Fractional birth rate of X      "
"P Kby=0.006                    Birth coefficient for Y         "
"P Kmx=0.001                    Predation coefficient           "
"P Kmy=0.05                     Fractional death rate for Y     "
"P K1=80                        Effect of the rain/temp ratio   "
```

If we wished to insert a material delay on the predator birth rate, we create a new file (`volmod2.dct`) inserting the `MDL` function in the `By` rate equation. The declaration of the delay time (`Devt`) as `P` statement is optional; it is possible to declare or modify it by an option of `simula`.

```
dictionary {
* file "volmod2.dct"
* example model for "simula" STATA command  (17/1/92)
* This is a kind of a VOLTERRA model with MATERIAL DELAY
* Ref. J. Maynard Smith, 1974. Models in ecology. Cambridge University Press.
str70 _model   "Model statements"
}
"* Prey Predator Volterra model with delay                     "
Same statements as above until we get to R By
"R By=MDL[Kby*Y*(X-Y),Devt]      Birth rate of predator (n/year) "
"* Note: MDL indicates a delay of Devt years of By contrib. to Y "
(Remaining statements the same)
"P Devt=0.5                     Delay (years) for developm. time"
```

Step 3: Exogenous variable file creation

Exogenous variables required by the model must be saved in an exogenous variable file, also containing a time variable named `_time`. Before saving this file it is necessary to sort it by `_time` because `simula` merges the exogenous variable file to the model by the `_time` variable.

Example

The exogenous variable file created contains the rainfall and temperature values for ten years. We can assume that the values came from a stochastic weather model; they are inserted in the following `.dta` file:

```
. describe
Contains data from d:\climate.dta
  Obs:    10 (max= 14230)                    EXO-VARS for the Volterra model
 Vars:     3 (max=    99)
  1. _time        float  %9.0g               Year
  2. temp         float  %9.0g               Annual average temper. (C)
  3. rain         float  %9.0g               Total annual rainfalls (mm)
Sorted by: _time
. list
         _time       temp       rain
  1.      1991         18       1550
  2.      1992         17       1300
  3.      1993         15       1800
  4.      1994         21       1950
  5.      1995         15       1200
  6.      1996         22       1100
  7.      1997         19       1500
  8.      1998         20       1400
  9.      1999         15       1700
 10.      2000         16       1600
```

Step 4: Use of the simula command

`simula` requires a model file already in memory; on request, it loads and merges the exogenous variables.

`simula` performs the numerical integration of the model equations by the Euler (rectangular) method. The command gives the change with time of the system conditions (state variables) starting from the initial values and with the provided parameter values.

At the end of simulation it draws a graph of the state variables versus time. For all the time steps required, the state, rate and auxiliary variable values, the exogenous variables and the `_time` variable are available in memory.

It is possible to modify the relations or parameters interactively using '`replace _model="`*statement*`" in` *#*'.

The syntax of the `simula` command is

> `simula` [`,` `dt(`*#*`)` `tstart(`*#*`)` `tspan(`*#*`)` `exovar(`*filename*`)` *graph_options*
> `parv("`*pname1=# pname2=# ...*`")` `ival("`*svar1=# svar2=# ...*`")`]

The options are

`dt(`*#*`)` specifies the integration time step (time units). The default is 1.

`start(`*#*`)` specifies the time at which the simulation starts. The default value is 0.

`tspan(`*#*`)` specifies the simulation length. The default value is 30.

`exovar(`*filename*`)` requests a file that will be merged to the model file in memory. This file must contain the exogenous variables used in the model. This option is needed when an exogenous variable statement is declared in `_model`.

`parv()` allows the modification of parameter values, overriding the values specified in the `_model` statements. *pname* is a parameter name.

`ival()` allows the modification of the initial values of the state variables, overriding the values specified in the `_model` statements. *sname* is a name of a state variable.

graph_options are any of the `graph,twoway` options. For example, `ylog` could be useful when the expected increase is explosive; `saving()` can be useful for saving the resulting graph, and so on.

Time solution step (dt)

To ensure a stable solution, a good rule is to keep `dt()` between one-half and one-tenth of the smallest time coefficient in the model (Richardson and Pugh 1981). Frequently, it is suggested we have a `dt()` value of one-fourth to one-fifth of the smallest time coefficient (Ferrari 1978). Values higher than one-half allow computational inaccuracies while values lower than one-tenth require much computer time. Moreover, a too small `dt()` value can also increase the round-off error because the rate values added to the state variables becomes very small.

It is useful, in any case, to try a smaller `dt()` value to check for the amount of error. The solution is correct if a small change in the time step does not affect the simulation results.

The choices of time step and starting time must allow the merging of the exogenous variables; the `_time` variable created by `simula` must have the values present in the `_time` variable of the exovar file (not only those, however). If the values of the two `_time` variables do not match, the exogenous variables are not merged. For example, let the `_time` values in the exovar file be 0, 1, 2, 3, and 4 and let the choices be `dt(.4)`, `tspan(8)`, and `tstart(1)`. The values for the `_time` variable created by `simula` will be 1, 1.4, 1.8, 2.2, 2.6, 3.0, 3.4, and 3.8. The program merges only the exovar values corresponding to time 1 and 3.

Exogenous variables

When the exogenous variables' data are available with a lower frequency than specified by the time solution step, then the command generates the missing data. The initial and final values are generated by repetition of the nearest value, the intermediate ones by linear interpolation.

Parameters and initial values

It is possible to override the parameter and initial values coded in the model file by specifying them in the command options (`parv()` and `ival()`). Indeed, while it is a good practice to indicate such values in the model file, the `parv()` and `ival()` options provide a faster method to analyze the model sensitivity and the effects of the initial condition variations.

Delays

The delay functions apply to all the state variables of the equation. If in a rate equation there are state variables with different delay time, it is better to use more than one rate equation for the same process. Note that the effect of `IDL` and `MDL` is the same until the delay time (*dly*) is a constant. If *dly* is changing, the retarded information is lost while the retarded material is conserved. Only the rate equations may contain the delay functions.

Auxiliary equations

If an auxiliary variable (ex: `avar1`) depends on another auxiliary equation (ex: `avar2`), the equation for `avar2` must precede the `avar1` equation in the model listing.

Example of the `simula` command use

To use `simula`, it is necessary to have a model coded in the string variable `_model` present in memory. Using the previous example of prey-predator interactions, we can load the file `volmod1.dct` before giving the `simula` command:

```
. infile using volmod1
dictionary {
* file  "volmod1.dct"
* example model for "simula" STATA command  (4/11/91)
* This is a kind of a VOLTERRA model - without delays
* Ref. J. Maynard Smith, 1974. Models in ecology. Cambridge University Press.
str70 _model   "Model statements"
(25 observations read)
}
```

Parameter values, while defined in the `volmod1.dct` file, can be overridden with the `ival()` and `parv()` options. Suppose that the initial values for X and Y are 800 and 400, respectively, and that $K_{bx} = 0.07$ and $K_{by} = 0.003$. The command will be

```
. simula,dt(0.25) tspan(10) tstart(1991) exo(climate) par(Kbx=0.07 Kby=0.003)
> ival(X=800 Y=400) border c(ll)
Parsing:
Line: 1
Line: 2
Line: 3
  (output omitted)
Line: 24
Line: 25
Simulation:
Time=1991
Time=1991.25
Time=1991.5
  (output omitted)
Time=2000.5
Time=2000.75
```

Figure 3, top panel, shows the results. We can take into account the material delay by loading the `volmod2.dct` file and typing

```
. simula,dt(0.25) tspan(10) tstart(1991) exo(climate) par(Kbx=0.07 Kby=0.003
> Devt=0.3) ival(X=800 Y=400) border c(ll)
```

The obtained simulation results are also presented in Figure 3, lower panel. (Stage was used to combine the two graphs produced by the separate runs of `simula`.) To obtain a two-dimensional space-state graphic, it is possible to create graphics of X vs. Y with different initial values and then overlapg them by using Stage. The Appendix reports other simulation examples.

APPENDIX

Example 1: Exponential growth

a) Positive feedback

Let us simulate the process of weight (W) increasing in a living organism, particularly during the first development phase. This process is known to be, generally, exponential and the growth rate (GR) proportional to W by a parameter RGR (relative growth rate). To simulate this system, we enter the model statements and use `simula`.

```
. drop _all
. input str40 _model
                                       _model
  1. "S W=W+dt*GR   Weight (g)"
  2. "R GR=RGR*W    Growth rate (g/day)"
  3. "I W=5         Initial weight (g)"
  4. "P RGR=0.1     Relat. growth rate (g/g/d)"
  5. end
```

The time coefficient of the rate equation is $1/RGR = 1/0.1 = 10$. To ensure a stable solution, `dt()` should be less than $10 \times (1/4) = 2.5$.

```
. simula,dt(0.5) tspan(40)
Parsing:
Line: 1
Line: 2
Line: 3
Line: 4
Simulation:
Time=0
Time=.5
Time=1
Time=1.5
  (output omitted)
Time=39.5
Time=40
```

The simulation determines an exponential growth of weight W shown in Figure 4 (curve a). Obviously, in this simple model the same result could be obtained with calculus. The integration of the differential equation $dW/dt = \text{RGR} \times W$ gives the exponential growth equation $W = W_0 \times \exp(\text{RGR} \times t)$, where W_0 is the initial value of W and t the time.

b) Negative feedback

If we decide to apply an asymptotic model having a goal, we modify the rate equation in line 2 adding a goal to the system (`Wmax`). This requires also the `Wmax` value, declared in line 5 as follows:

```
. replace _model="R GR=RGR*(Wmax-W)    Growth rate (g/d)" in 2
(1 change made)
. replace _model="P Wmax=6       Maximum weight (g)" in 5
(1 change made)
. format _model %1s
. list _model in 1/5
       _model
  1.  S W=W+dt*GR   Weight (g)
  2.  R GR=RGR*(Wmax-W)    Growth rate (g/d)
  3.  I W=5         Initial weight (g)
  4.  P RGR=0.1     Relat. growth rate (g/g/d)
  5.  P Wmax=6      Maximum weight (g)
```

Again, the simulation of the above model is performed by the command '`simula, dt(.5) tspan(40)`'. The simulation results show an asymptotic exponential growth (Figure 4, curve b).

c) Physical limit to growth (rate switch-off)

If there are no growth limitations until a maximum is reached, the function `SWTC` is appropriate. The growth rate GR is turned off when the control variable (W) reaches the upper limit (Wmax). To simulate this behavior we modify the second line of the program as

```
. replace _model="R GR=SWTC[RGR*W,W,Z,Wmax]" in 2
(1 change made)
```

In `simula` we specify a new Wmax value and set Z to 0. Z is set to 0 to avoid the zeroing of the rate at low W values. Typing '`simula, dt(0.5) tspan(20) par(Wmax=15 Z=0)`' performs the simulation. The results are plotted in Figure 5.

d) Start time to accumulation (rate switch-on)

If a rate must be integrated after a control variable reaches a threshold value, we can use the `SWTC` function in the switch-on version. The growth rate GR is turned on when the control variable (for example, `_time`) reaches the threshold `Thre`). An example is the growth of a plant, starting after the seedling exits the soil. To simulate this behavior, we modify the second line of the program to be

```
. replace _model="R GR=SWTC[RGR*W,_time,Thre,H]" in 2
```

and the simulation is obtained by typing '`simula, dt(0.5) tspan(20) par(Thre=7 H=21)`'. In this case, H is so large as to avoid the rate switch-off. Figure 5 plots the results.

Example 2: Regulation of the shower water temperature

a) Negative feedback without delay

We wish to regulate the warm water flux (with temperature $T_w = 50^\circ$ C) to obtain the desired temperature ($T_{\text{opt}} = 35^\circ$ C), maintaining a cold water flux of 0.2 l/s. Let the cold water temperature be $T_c = 10^\circ$ C. The state variables of the system can be the warm water flux (W_w) and the cold water flux (W_c).

We assume no variation in the W_c flux ($\text{VA}_c = 0$) and the warm water regulation depending on temperature difference $T_{\text{opt}} - T$. T is the auxiliary variable representing the temperature of the mixed water. We assume also a perfect regulation, with no delays in perceiving the real temperature at the mixer level. The total flux (F) is considered given by the summation of W_c and W_w. The model, recorded as a `.dct` file, is as follows:

```
dictionary {
* file shower1.dct
* Shower water temperature regulation - 1 - WITHOUT DELAY
* Cold water flux constant - regulation of warm water flux
* to obtain the desired temperature (Topt).
        str70  _model
}
"* -- state variables --                                   "
"S  Wc=Wc+dt*VAc                Cold water flux (l/s)      "
"S  Ww=Ww+dt*VAw                Warm water flux (l/s)      "
"*  Note: in this case fluxes are levels, not rates.       "
"R  VAc=0                       Variat. rate cold flux (l/s/s)"
"R  VAw=K*(Topt-T)              Variat. rate warm flux (l/s/s)"
"I  Wc=0.2                                                 "
"I  Ww=0                                                   "
"P  Topt=35                     Optimal temperature (C)    "
"P  Tc=10                       Cold water temperature (C) "
"P  Tw=50                       Warm water temperature (C) "
"A  T=(Tc*Wc+Tw*Ww)/(Wc+Ww)     Shower water temperature   "
"P  K=0.03                      Adjusting speed            "
```

To read this file and perform the simulation, we type

```
. infile using shower
dictionary {
* file shower1.dct
* Shower water temperature regulation - 1 - WITHOUT DELAY
* Cold water flux constant - regulation of warm water flux
* to obtain the desired temperature (Topt).
        str70  _model
}
(15 observations read)
. simula, dt(0.2) tspan(12)
```

The simulation results reported in Figure 6(a) show that after a few seconds the system reaches the equilibrium and T equals the optimal temperature.

b) Negative feedback with a constant information delay

Really, there is a delay in perceiving the real water temperature at the mixer level. This determines a difficult regulation with temperature higher and lower than the optimal. The desired condition is obtained later and with oscillating warm water fluxes. This situation can be simulated by inserting an "information delay" in the VA_w rate calculation of the previous model. This is done by modifying our model:

```
. replace _model="R  VAw=IDL[K*(Topt-T),dly]" in 7
. replace _model="P  dly=1" in 16
```

The first command inserts a constant information delay of `dly` seconds in the VA_w rate; the second command assigns the value to the `dly` parameter. Recalling the simulation command,

```
. simula, dt(0.2) tspan(12)
```

we obtain the results shown in Figure 6(b). The constant information delay produces an oscillating behavior of the warm water flux.

c) Negative feedback with variable information delay

If we assume that information delay time is variable, as a function of the total flux ($F = W_c + W_w$) and of the volume of the pipe from the mixer to the sprinkler (V_0), we can calculate the time the mixed water needs to reach the sprinkler by the equation $dly = V_0/F$.

Now, we replace the parameter `dly` in line 16 with two auxiliary equations

```
. replace _model="A  F=Wc+Ww" in 16
. replace _model="A  dly=Vo/F" in 17
```

and we assign the value to the parameter V_0 by the line:

```
. replace _model="P  Vo=0.6"  in 18
```

Thus, it is possible to simulate the warm water flux and the mixed water temperature with a variable information delay by typing '`simula, dt(0.2) tspan(12)`'. The simulation results are shown in Figure 6(c).

d) Negative feedback with variable material delay

A different pattern of water regulation is still obtained if we consider a variable material delay. In this case the delayed quantity is not lost but is cumulated to the other delayed quantity. This could be the case when a lizard is taking a shower. It is not warm-blooded and then we can suppose its reaction time depending on the water temperature. Even if the very mixed water temperature is perceived without information delay, the regulation performed by the lizard is very slow at low temperature and fast at high temperature. Anyhow, the regulation quantity is not forgotten by the lizard but only delayed and maybe cumulated if the delay time is shortening with time. In this situation the dynamic of the water flux regulation can be complex and shown in Figure 6(d), for a time reaction coefficient of $R_{tc} = 0.05$. To obtain a variable material delay, `_model` is modified by

```
. replace _model="R  VAw=MDL[K*(Topt-T),dly]" in 7
. replace _model="A  dly=Rtc*T" in 16
. replace _model="P  Rtc=0.05" in 17
. drop in 18
. simula, dt(0.2) tspan(12)
```

The previous example with variable material delay is not completely pertinent. A better example could be the arrival rate of wares in a harbor, transported by ships. If further goods are needed (system state) the information requiring other goods is quite without retard in respect to the material delay due to the time for transport. If the ships have different speeds, the material delay is variable, and it is possible that ships starting at different times arrive at the destination on the same day. In this case the wares are not lost but are cumulated for all the ships arriving on the same day.

Figures

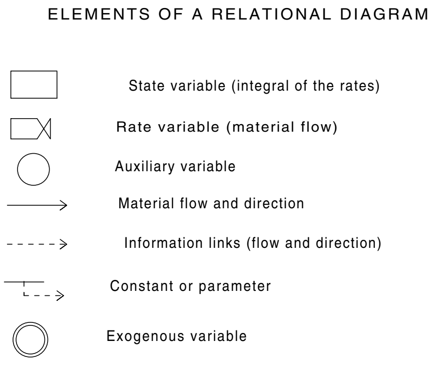
ELEMENTS OF A RELATIONAL DIAGRAM
State variable (integral of the rates)
Rate variable (material flow)
Auxiliary variable
Material flow and direction
Information links (flow and direction)
Constant or parameter
Exogenous variable

Figure 1

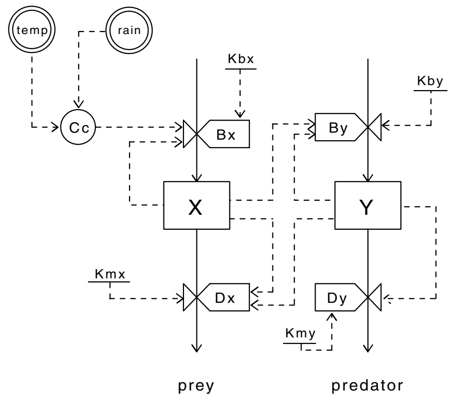

Figure 2

Figure 1 shows the graphic symbols for the relational diagram of the system. Figure 2 shows the relational diagram by the Forrester (1968) conventions of the prey-predator interaction in a "Volterra model." The state variables are X (prey density) and Y (predator density). Bx, By, Dx, Dy represent, respectively, the birth and mortality rates for prey and predator. Cc is the "carrying capacity" of the ecosystem. (Both graphs were drawn "freehand" using Stage.)

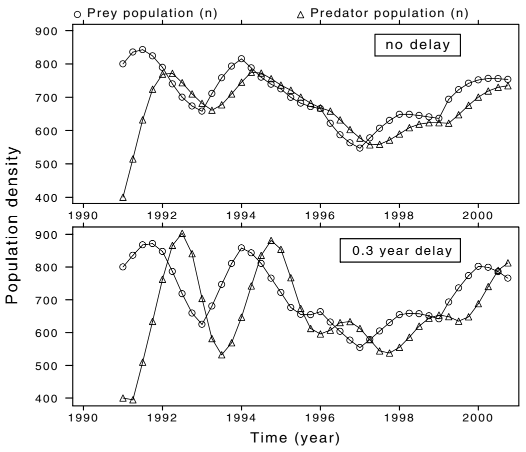

Figure 3

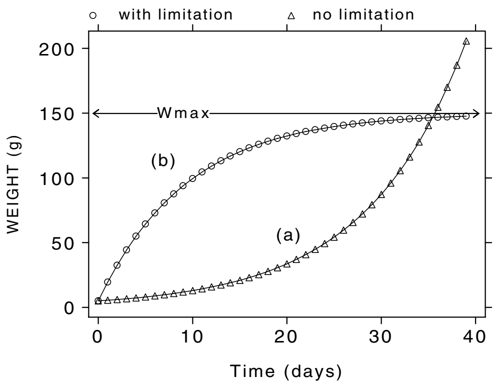

Figure 4

Figure 3 shows the simulations of the Volterra model of prey-predator interactions. In the top panel, no delay between predation and predator birth is assumed. In the bottom panel, a time lag of 0.3 years is assumed to exist between predation and the new births. Figure 4 shows the growth process of an organism, without limitations (a) and with a maximum value obtainable (b).

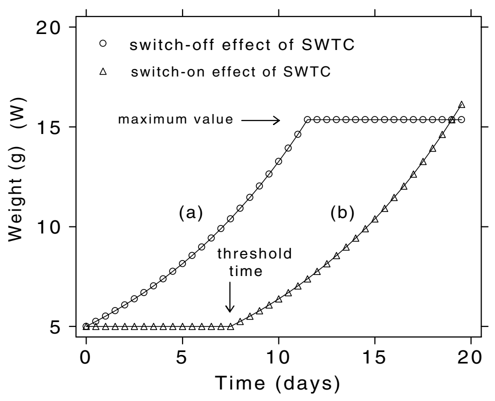

Figure 5

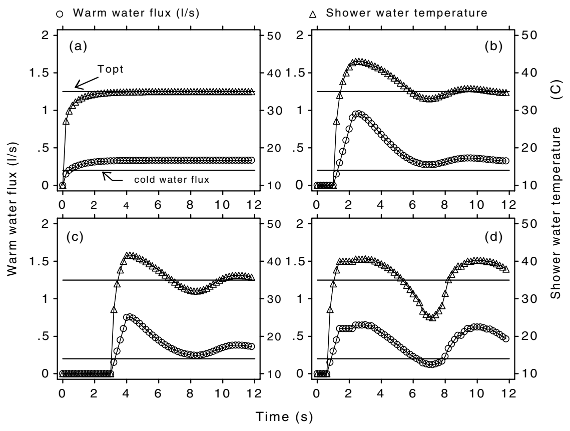

Figure 6

Figure 5 shows the growth process of an organism, switch-off (a) and switch-on (b) effects of the `SWTC` function. Figure 6 shows the shower water temperature regulation: a) perfect (no delay); b) long pipe (constant information delay); c) long pipe (information delay, variable as a function of total flux); d) lizard adjusting (material delay, variable as a function of the mixed water temperature).

References

ACSL. 1987. *Advanced Continuous Simulation Language, User Guide/Reference manual*, Edition 4.1. Concord, MA: Mitchell and Gauthier Assoc.

Brennan, R. D., C. T. De Wit, W. A. Williams, and E. V. Quattrin. 1970. *The Utility of a Digital Simulation Language for Ecological Modeling*. Oecologia (Berlin) 4: 113-132.

Dent, J. B. and M. J. Blackie. 1979. *System Simulation in Agriculture*. London: Applied Science Publishers.

de Wit C. T. and J. Goudriaan. 1978. *Simulation of ecological processes*. Wageningen: PUDOC.

Ferrari, T. J. 1978. *Elements of System-Dynamics Simulation—A textbook with exercises*. Wageningen: PUDOC.

Forrester, J. W. 1968. *Principles of Systems*. Cambridge: MIT Press.

Jansen, D. M., R. T. Dierkx, H. H. Van Laar, and M. J. Alagos. 1988. PCSMP on IBM PC-AT's or PC-XT's and compatibles. *Simul. Report no. 15*, Wageningen: CABO.

IBM. 1972. Continuous System Modeling Program III (CSMP III). *Program Reference Manual.* New York: IBM World Trade Corporation.

Meadows, D. L. and D. H. Meadows; J. Rounders and W. W. Behrens III. 1972. *The Limits to Growth.* New York: Universe Book, A Potomac Associates Book.

Meadows, D. L. and D. H. Meadows (eds.). 1973. *Toward a Global Equilibrium.* Cambridge, MA: The MIT Press.

Maynard Smith, J. 1974. *Models in Ecology.* Cambridge University Press.

Richardson, G. P. and A. L. Pugh. 1981. *Introduction to System Dynamic Modelling with Dinamo.* Cambridge, MA: MIT Press.

Smerage, G. H. 1979. Modeling Theory for Physiological Systems. *Transactions of the ASAE*, 1488–1493.