Interindustry Forecasting at the University of Maryland

THE G7 USER GUIDE AND REFERENCE FOR G7.3

AUGUST 2011

INTERINDUSTRY FORECASTING PROJECT

ECONOMICS DEPARTMENT

UNIVERSITY OF MARYLAND

COLLEGE PARK, MD 20742

(301) 405-4609

www.inforum.umd.edu

TABLE OF CONTENTS

G7 Reference

This reference serves as the complete documentation on the commands available in *G6* for DOS and *G7* for Windows, explaining their syntax and providing examples to make the syntax clearer. When referring to commands or features that apply to both programs, we will use the generic name "*G*". Where commands are particular to one version or the other, we will make note of that.

The first part of the reference covers some general topics in *G7*. The second part consists of an alphabetical list of commands.

Each *G7* command is listed in bold. If an abbreviated version of the command is available, that part of the name is enclosed in parentheses, or the abbreviation is listed separately. For historical reasons, some commands have two names. When this is the case, only the command name of the second version is listed on the second line. Command line parameters are enclosed in angle brackets (< >). Optional parameters are enclosed in square brackets ([ ]). References to other related commands are provided at the end of each command entry. Within text, names of commands are mentioned in double-quotes (“ “). Names of time series in *G7* are also in double-quotes, when referenced in the text. Names of filenames are in ALL CAPS. Names of programs are in *Italics*. Examples are given in `Courier` font.

What is G7?

G7 is the regression analysis member of a closely integrated package of programs for building economic models. It can be used look at data, to estimate a single equation or a system of equations, to build a simultaneous equation model, or to create a multisectoral model involving hundreds of equations. It has extensive capabilities for:

- Making, using, and exploring data banks. Over twenty different banks can be assigned at once. Data can be shown in graphs or in grids resembling spreadsheets.

- Rapid drawing and annotating graphs and saving them so that they can be imported into documents.

- Producing tables of results.

- Editing and displaying text files.

- Combining and transforming variables. Transformations available include algebraic, exponential, logarithmic, sine, cumulation with decay, interpolation, change of frequency, linking and benchmarking of series, cutting off negatives, converting all positive elements to 1.

- Looping, such as over sector numbers and titles.

- Performing regressions by ordinary least squares, least squares with soft constraints and

distributed lags, stacked regression with constraints across equations, pooled regression, SUR, 2SLS, 3SLS, recursive regression, ARIMA, and non-linear least squares. Regression results include -- besides the usual regression coefficients and t-values, R^2, Δ, and DW statistics -- for each variable its marginal explanatory value, elasticity at the mean, beta coefficient, and the normalized sum of squared residuals after its introduction. F statistics for groups of variables are available, as is the leverage variable for finding outlying observations. If requested, the matrix of simple correlations among all the variables, the derivatives of regression coefficients with respect to one another, and the variance-covariance matrix of the regression coefficients can be shown. Statistics for testing whether the residuals are normal are available. Information necessary for stochastic simulation can be saved. Missing values in any variable are noted.

- Building simultaneous equation macro-economic models. The building process checks the model for inconsistencies in the definition of variables. A simple check on the correctness of the model's identities is provided. Trend projections of all exogenous variables can be automatically generated. The building process writes a C program for solving the model. This program is then automatically compiled and linked, usually in a few seconds. It then operates extremely rapidly. Also, any calculations not easily expressed in *G7* commands can be accomplished by C code that is passed through to the program for solving the model. Models can be easily run with various sorts of modifications to the results of the regression equations. Stochastic simulations can be run.

- Building interindustry models or other models that require extensive matrix and vector operations. Such models can include a linked system of national or regional models, or a model of bilateral world trade at a detailed commodity level. Results of these models can be shown in grids resembling spreadsheets.

G7 7.0 and higher require the Windows95 or higher operating system. *G* 6.4 can be run in a DOS window. Versions are also available for Linux and other versions of Unix.

Starting G7

Double click on the *G7* icon to start the program. After an initial splash screen, you will be given a choice of folder (directory) in which to start *G7*. This step is necessary because *G*7 looks for a C.CFG file in the folder from which it starts, and this file determines the characteristics of the workspace which *G7* creates and the initially assigned data bank. For example, to follow the initial tutorial, you should select the C:\AMI folder. If you exit *G7* normally, your choice of folder will be remembered next time you start. If you wish to use the same one, you have only to confirm the initial selection.

You then see the *G7* main form. It has a menu at the top, a few speed buttons, a white, one-line command box, and a large blue results area. The blinking vertical line in the command box indicates that *G7* is ready to receive any of its commands from this box. For example, if you have selected C:\AMI as the folder, you can give the command "type gdp". You will see values

of U.S. GDP appear in the blue "results" window. (The first number on each line is the quarter of the first datum on the line.)

The default font for the results window is rather small so as to allow all of the results of a regression to be seen in a form only half the width of the screen. Many users, however, prefer to use a larger font. Click File | Font to pick your font. Some displays assume that the font is monospaced, so FixedSys is a good choice, as is Courier in a higher point size. If you exit *G7* normally, your choice of font will be remembered next time you start *G7*. Try it; after changing the font, give *G7* the "q" command in the white command box (or use 'Alt-F4', or File | Exit).

To start *G*6.4 for DOS, just type "G" at any DOS command window. This assumes that *G* has been installed correctly on your system, and that the PATH variable includes C:\PDG, or whatever directory you installed the Inforum software into. The opening screen introduces *G* and then returns a command prompt (:). At this point, the full selection of *G* commands are available to you. You can get help at any time by using the "help" command. If you want a list of *G* commands in the DOS version, just type "commands". If you would like help on any individual command, type "help <commandname>" where <commandname> is the name of the command.

autocomplete [<setting>]

The command box provides an auto-complete feature to speed the typing of repeated commands. Sometimes this feature proves troublesome. It can be turned off by clicking File | Autocomplete, or it can be turned off by typing the autocomplete command in the command box.

Settings may be "yes" or "no". If no setting is given, then the current setting is displayed. The setting will be saved when *G7* is closed, and the same setting will be restored when *G7* next is run.

Command Cache

These menu items provide control over the command-line cache, where each command-box entry is recorded. Available menu items allow printing of the cached commands, clearing of the cache, and execution of the cached commands.

G7 Topics

The Basic 12 G7 Commands

This is a quick review of the most commonly used commands. Each is described in greater detail in the command reference . In this reference, the full name of the command is given, and the abbreviated version, if available, is enclosed in quotes.

(ti)tle <text>

Provides a title for regressions and graphs.

Example:

```
title Consumption Function
```

f <variable> = <expression>

f <variable>{ <date1> [- <date2>] } = <expression>

Defines variable on left in terms of variables in expression on right.

Examples:

```
f x = gnp/(1. + unemp[1]*@exp(.25*time))
f index = x/x{85.1}
```

Note: [ ] denotes lags; (t-1) is denoted [1]; @ introduces a function (see list of functions); and {date} denotes a specific observation. A date or range of dates on the left-hand side determines the dates for which the calculations will be made.

If the left-hand variable already is in the workspace bank, then its values may be modified with the "+=", "-=", "*=", or "/=" operators in place of the standard '='. By using these operators, the right-hand side expression will be added or subtracted from the left-hand side, or the left-hand side will be multiplied by the right-hand side values, or the left-hand side values will be divided by right-hand side values. These operators follow the syntax for the C++ programming language, though here they operate over a range of values. To add *expression* to a *variable*, where *variable* already exists in the workspace bank, the syntax is

f <variable> += <expression>

(lim)its <date1> <date2> [date3]

Sets limits for regressions and for @sum() commands. Estimation begins at date1; ends at date2. Simulation or forecast to date3. Quarterly dates require .1, .2, .3 or .4. Monthly dates require .001 ....012.

Example:

```
lim 75.1 84.4 86.3
```

(r)egress <y> = <x1>, <x2>, <x3>, ...

Regresses y on x1, x2, etc. The x's may be simple expressions.

Example:

```
r gnp$ = g$, v$-vi$
```

(gr)aph <y1> [y2] [y3] [y4] ... [y7] [date1] [date2] (note: pl == gr)

Graphs up to six series from date1 to date2. Dates are retained from previous "gr" commands.

Examples:

```
gr v$ c$ g$ 79.1 86.3
gr *      (This graphs actuals and predicted from last regression.)
```

(ty)pe <y> [date1] [date2] (note: pr == ty)

Displays the values of y from date1 to date2 on the screen. Dates need not be repeated if unchanged from previous "ty" or "pr" commands.

Example:

```
ty gnp$ 79.1 86.3
```

data <name>
<date> <observation1> ... <observationN>

Introduces data into workspace. First number on each line is date of first observation on line.

Example:

```
data sales
   85.1  117 123 134 142
   86.1  137 143 145;
```

(ba)nk <bankname>

Assigns a data bank. What this means is discussed more fully in the topic on assigned banks.

Example:

```
ba mybank
```

(lis)tnames [-srgv] <w | a> [wildcard]

Types the names of the variables in the workspace (w) or assigned bank (a). If the *save* command is on, the list will then appear both on the screen and in the saved file. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS. The *lnc* routine prints the series as a single column. Option *s* sorts the series in alphabetical order, and *r* reverses the order. If a Vam file is associated with the G bank, then option *g* prints only macro series and option *v* prints only Vam bank series.

(ad)d <filename> [arguments]

Takes commands from named file.

Example:

```
 add invest.reg
```

help
Calls up the help menu.

(q)uit
Quits and exits *G7*.

Dates in G7

Dates in *G7* can be either 4-digit or 2-digit. A date is indicated by a year, followed by a decimal point, followed by an optional "period", which is used with quarterly or monthly data. For example, "83" or "83.0" or "1983" all refer to the year 1983. With annual data, the period is either "0", or should be left off. Note that when using "2-digit" dates, the year 2000 should be represented by "100", and the year 2010 by "110".

Quarterly data is indicated by a one-digit period, from 1 to 4. Thus, "83.1" and "1983.4" refer to the first and fourth quarters of 1983.

Monthly data is indicated by a three-digit period, from 001 to 012. So, "83.001" is January 1983, and 1999.012 is December 1999.

Note: if "gnp" is a quarterly series, the command

```
type gnp 65 85
```

will NOT print "gnp" for the years 1965 to 1985, but return the error message "Dates do not match frequency of variable." The correct command is

```
type gnp 65.1 85.4
```

freq <series name> [frequency]
This command displays a series' frequency and allows you to change it. The series and its new frequency will be put into the workspace bank. Note that this is true even if the series was originally in an assigned bank. (See the topic "Assigned Banks" for more information on data banks. See the function list for functions in *G7* that will change the frequency of data, performing conversions from quarterly to annual and back, etc..)

dfreq [0|1|2|4|6|12]
Set a default frequency for the left-hand side variable of an f commands when the right hand side does not have a frequency. The default "default" is 0, no frequency.

fdates [date1 [date2]]
fdates off
Sets or resets the dates used by subsequent "f" commands. When an "fdates" command is issued, it defines the time period in which subsequent "f" commands act. See the detailed help on "fdates" for more information. The "freq" command displays the frequency of a series.

tdates <date1> <date2>

Sets the dates used by subsequent “type” commands. The command

```
tdates 80.1 90.4
```

means that the next "type" or "print" command will display quarterly data from the first quarter of 1980 until the fourth quarter of 1990.

gdates <date1> <date2> [date3]

Sets the dates used by subsequent “graph” (or “plot”) commands. With two dates provided, the series will be graphed from date1 to date2. If a third date is given, the series will be graphed from date1 to date3, with a vertical line drawn at date2.

gdates a
tdates a

Selects "automatic" dates for “graph” and “type” commands. The automatic dates are the first and last date of the series actually present. The default setting in "look" is for automatic dates, unless specific dates have been previously specified. Automatic dates also adjust automatically to the frequency of the series. This feature is not to be trusted when more than one series is being placed on the same graph.

Assigned Data Banks

G databanks consist of at least two files: an index file that contains a list of series names, and a data file. A third file, called a “stubfile”, giving code names of the variables and full titles is optional. *G7* always has access to two databanks, the assigned bank and the workspace bank.

The assigned banks may be changed during a *G7* session by the various“bank” commands explained below. The workspace banks is not normally swapped during a session. (See the Workspace Bank topic.) All data introduced into *G7* or created during a session will be stored in the workspace (not an assigned bank), no matter how many different banks are assigned during the session. When *G7* needs a series, it scans the name list of the workspace and then, only if it doesn’t find the variable there, it scans the name list of the assigned bank. (Preceeding the name of a variable with “a.” , “b.”, etc. forces *G7* to skip looking in the workspace and to look only in that assigned bank.) The assigned bank to which *G7* has access initially as bank ‘a’ is specified in the G.CFG file in the default directory.

All series created with “f” commands or introduced with “data” commands, are put into the workspace, even if the “update” and “mupdate” commands are used. By design *G7* provides no facility for writing directly into the assigned bank. This limitation protects against inadvertently changing a series in a data bank. Building or modifying banks is explained below.

The assigned bank may be of five types:

Standard banks—exactly the same as the workspace banks. All series begin in the same period, all have the same number of observations, and all data points are represented in 4-byte floating point numbers. The file with the body of the bank has the extension .bnk, while the file

with the index has the .ind extension. The number of series is limited to 3000. Simply renaming the files of the workspace after exiting from *G7* turns it into a standard bank.

Compressed banks—the body of the file has been compressed by recording the starting and ending date of each series and eliminating from each series periods for which no data are available. Where possible, the series is stored as first differences represented by two-byte integers. The body of the bank has the extension .cbk, while the index has the extension .cin. The number of series is limited by the requirement that the total number of letters in the names of all the variables should be less than 65,000. In practice, that limit is about 7000 series. The *Press* program converts a bank to a compressed bank. Do not used a compressed bank with dummy variables, of values only 0 or 1, as the 0 observations are treated asvariables, of values only 0 or 1, as the 0 observations are treated as missing values.

Hashed banks—the body is similar to the Compressed bank, but the index file has been put through a "hashing" process which permits such banks to contain millions of series, so that the number of series is no longer an effective limitation on the size of banks. The body file has the extension .hbk; and the index file, .hin. The same warning about dummy variables applies. Hashed banks usually require less memory than compressed banks.

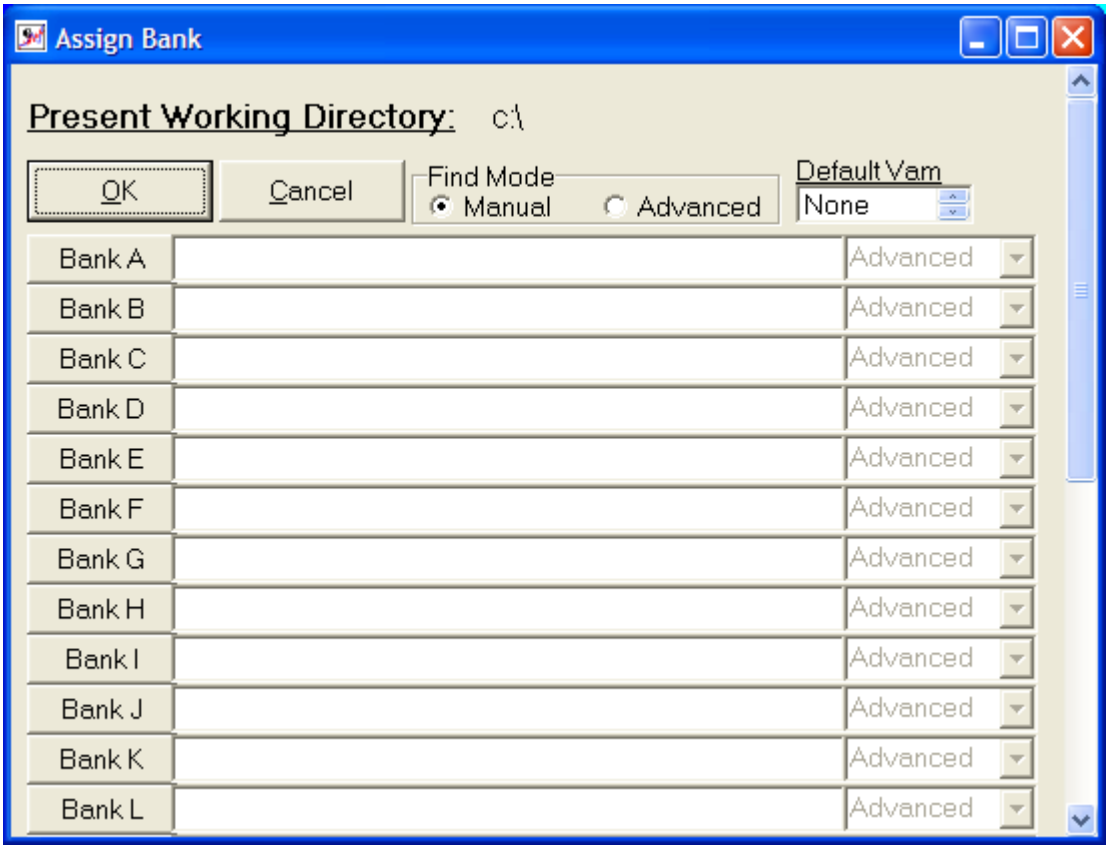
Vam files—Vam files are files that hold vector and matrix data, and are used in the development of *Interdyme* models. *Interdyme* is the system of programs and C++ classes used by Inforum to develop macroeconomic interindustry models. The vector data in these files can be read, manipulated, typed and graphed in *G7*. If a vector is called "out", to show the series for sector one, you could refer to it as "out1".

Dirfor files—Dirfor files are another format used by Inforum in conjunction with macroeconomic interindustry models. These files hold multiple vectors of data,as well as a set of "macrovariables". Each Dirfor file needs a companion file in the current directory called DIRFOR.DAT, or it cannot be read.

The five different banks are readily distinguished by the extension part of their file names. These extensions are as follows:

Type of Bank	Data File Extension	Index File Extension
Standard bank (workspace)	.bnk	.ind
Compressed bank	.cbk	.cin
Hashed bank	.hbk	.hin
Vam file	.vam	none
Dirfor file	.dfr	File should be DIRFOR.DAT.

Several methods are available to load additional or alternative banks. One method is to select Bank | Assign Data Banks from the *G7* menu. The following window will appear, allowing you to browse for banks, type in the name of a bank, or to remove a bank that already is loaded. In addition, findmode may be set, a default vam bank declared, and the vammode setting specified for each vam bank.


Assign Bank
Present Working Directory: c:\
OK
Cancel
Find Mode
Manual
Advanced
Default Vam
None
Bank A
Bank B
Bank C
Bank D
Bank E
Bank F
Bank G
Bank H
Bank I
Bank J
Bank K
Bank L
Advanced

Here are some commonly-used commands for working with banks.

(ba)nk <bank_name> [<bank_location>]
Makes the named standard *G* databank the assigned bank. Example: If a bank consisting of the two files PRICE.BNK and PRICE.IND is in the default directory, the command:

```
ba price
```

makes PRICE the assigned bank. If the bank is in another directory, use the entire path name in the command. The <bank_location> argument is optional, and may be any letter between 'a' and 'z'. If it is not given, the default is 'a'. *G7* may have up to 25 assigned banks, of any of the five types discussed above. To assign the price bank in position c, you would use the command:

```
ba price c
```

cbk <bank_name> [<bank_location>]
Make the named compressed bank the assigned bank.

hbk <bank_name> [<bank_location>]
Make the named hashed bank the assigned bank.

vam <bank_name> [<bank_location>]
Makes the named Vam file the assigned bank.

dfr <bank_name> [<bank_location>]
Maked the named Dirfor file the assigned bank.

(lis)tnames [-srgv] <w | a> [wildcard]
Types the names of the variables in the workspace (w) or assigned bank (a). If the *save* command is on, the list will then appear both on the screen and in the saved file. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS. The lnc routine prints the series as a single column. Option *s* sorts the series in alphabetical order, and *r* reverses the order. If a Vam file is associated with the G bank, then option *g* prints only macro series and option *v* prints only Vam bank series.

listnamescol <bank_location> [<wildcard>]
lnc [-srgv] <w | a> [wildcard]
The *listnamescol* or *lnc* command works just like the *lis* command, but all series names in the workspace (w) or assigned bank (a) are output in one column. You may find it convenient to capture output to a file, and then use this command to get a list of all the series in the databank in that file. Then, using an editor that supports macros, you can change all lines to create an addfile that does the same thing with

each series in the databank. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS. Option *s* sorts the series in alphabetical order, and *r* reverses the order. If a Vam file is associated with the G bank, then option *g* prints only macro series and option *v* prints only Vam bank series.

(btit)le <bank_location>

Displays the title and configuration of the workspace (w) or assigned bank ('a' through 'j'). In the case of the workspace, you can supply a new title or change the existing one.

Updating a Data Bank

Suppose you have just received new data for your price bank. How can you correct or extend the bank without completely remaking it? If the new data has come to you as a hashed bank—as data from the EconData web site does -- then, at the DOS prompt, type

```
hsplice <old> <new> <combined>
```

where <old>, <new>, and <combined> are the names of the old hashed bank, the new hashed bank, and the combined hashed bank. The combined bank will contain all data in either the old or the new bank, but the data in <new> will replace that in <old> wherever both banks have a data point.

If the new data comes in some other form, follow the directions in the " Workspace bank" topic for getting it into a compressed G bank. Then proceed as above.

Looking at a Data Bank

To look at the series in one of the assigned data banks, click on the Bank menu and select Look (we will refer to this sequence as Bank | Look). A list of the currently assigned banks will appear in a list box. Click on one of these banks, then a window opens in the upper right corner of the screen. It has a list of the data series in that assigned bank. Run the cursor down the list until you find one that interests you. Tap 'Enter'. Below you will see a graph and to the left, in the results area, the numerical values of the series. (The first number in each line is the date of the first observation in that line; then follow the data for four successive quarterly observations.) The cursor will have gone back to the command box, but if you wish to see another series, click on the look window again and continue looking. If you tap a letter key, the cursor will move to the next entry beginning with that letter. For a deeper search, click Find on the menu bar, and fill in the word or phrase which you wish to find. Notice that you can search up or down, with or without sensitivity to case. A case-sensitive search for "income" will not find "Income" or "INCOME"; without case sensitivity, the same search would find both. Another way to get the "look" command is from the command box. Type "look" if there is only one assigned bank, or "look <bank_letter>" if there is more than one. In *G* 6.4 for DOS, the "look" command works similarly, but the screen changes to contain the series in the stub file. At this point you can pick

"t" for "type", "g" for "graph", or "p" for "paste", which puts up a simple spreadsheet of data on the screen, for the series currently visible from the stub file.

It is not necessary to close the look window in *G7* while doing other things. It may be closed, however, by clicking the x in the upper right corner.

The "look" command will work only if there exists in the directory with the bank a file with the same root name as the bank has but with the extension .STB. This file is called a "stub file", and has lines containing the series name, a semi-colon, and a the full descriptive title of the series, for example:

```
gdp   ; Gross Domestic Product
```

For an example, see the QUIP.STB file in the AMI directory.

Forming Variables

Most transformations of existing variables into new variables are done with the "f" command. Such transformations alter the values of a variable over a period specified by an "fdates" command. The default fdates are determined by the maximum number of observations that are defined in the G.CFG file.

The general form of an "f" command is

f <variable> = <expression>

The resulting variable is placed in the workspace bank.

Example:

```
f x  =  y*(z[3] + v*@log(w))/s{77.1}
```

Variable x is calculated and placed in the workspace using the variables y, z, v, w, and s. These variables must already exist. Note in the example:

z[3] means z lagged three periods, i.e. z(t-3) in a common notation,

s{77.1} means the value of s in the first quarter of 1977,

@log(w) means the natural logarithm function of the variable w.

The full list of @ functions are shown in the list of functions.

Rules for "f" statements

Any algebraically legal expression is allowed, including nested parentheses.

The left side variable must be a single variable, or a variable with a subscript to indicate a specific time. For example, z and z{78.1} are valid variables on the left side. However, z[1] is invalid on the left side.

For a series which exists in the workspace or in the assigned bank, the "f" command modifies the value within the period specified by fdate without affecting the existing values outside of that period, and puts the series in the workspace. Otherwise, the "f" command creates a series, determines the values using the expression on the right hand side of the "f" command for the period specified by fdate, and sets missing values, (-0.000001), for those observations outside the period. If "missing n" is in effect, the observations outside of the fdates period will be set to zero, instead of -0.000001.

For example, if we have in effect the following "fdates" command:

```
fdates 78 84
```

then for an "f" command

f x = (expression)

If x is an existing series in either the workspace or the assigned bank, x will be placed in the workspace with values from 1978 through 1984 being taken from the expression. Otherwise, x is created with values taken from the expression from 1978 through 1984, and taken as missing for the rest of the period.

Variable names must begin with a letter and may contain up to 32 letters, digits, or the '$' or '_'characters. Do not use a digit as the first character.

Powers are obtained using the @sq and @pow functions, not ** or ^.

Division by zero in an "f" command is legal and yields zero as a result.

Right-hand-side expressions too long to fit on a single line may be continued by using a +, -, *, or / as the last character on a line.

The commands "f", "fex", and "fdup" have exactly the same effect in *G7* but very different effects in *Build*. See the model building section for the differences.

Other commands for transforming variables

(miss)ing <y|n>

This is 'y' by default. If it is set to 'n', then all values that are missing in the databank will be set to zeroes. Otherwise, they will remain as missing values (i.e. -.000001).

ls <x> <y> <date> [direction]

This "linkseries" command takes series x and y from the workspace or the assigned bank, calculates their ratio at the specified linking date. This ratio is then used to move x with non-zero entries from series y. The linking direction can be 'f' for

forwards or 'b' for backwards. The default is 'f'. The result is stored with name x in the workspace.

ctrl <x> <concept> <group>

Imposes the values of <x> series as a control total on a <group> of sectors of a given <concept> (such as exports). If the variables to be controlled are found in the workspace, then the results will be stored in workspace with names formed by <concept> and given sectors. If a bank letter is provided for <concept>, and if this bank is the default Vam file, then the modified values will be stored in the default Vam file. Otherwise, the results will be stored to the workspace.

For example, the command

```
ctrl tot out 1-10 (4-7) 13 15
```

imposes the values of tot as a control total on the named sectors out, i.e. sectors 1 to 10, except for 4 through 7, and then 13 and 15.

Functions

The following arithmetic @ functions are available:

@log(x) natural logarithm of x

@exp(x) exponential function of x

@gr(x) the period-to-period growth rate of x

@yoy(x) the year-on-year growth rate of x

@ggr(x,start,end) the geometric growth rate of x from the starting period to the ending period

@pos(x) = x if x > 0, otherwise 0

@if(<exp1> <condition> <exp2>, <exp3>, <exp4>)

creates a series dependant on the period-by-period comparison of expressions exp1 and exp2, where valid comparisons are <, <=, ==, >=, and >. If the result is true for a given period, then the result of expression exp3 is assigned. If instead the result is false, then the value of exp4 is assigned. Expressions are any legal equation for the the f command.

@ifpos(x) = 1 if x > 0, else 0

@min(expression, start, end)

returns the minimum value of the expression between period start and period end. If these dates are not given, then the fdates are assumed to apply.

@max(expression, start, end)

returns the maximum value of the expression between period start and period end. If these dates are not given, then the fdates are assumed to apply.

@fabs(x) absolute value of x

@sq(x) the square of x

@sqrt(x) the square root of *x*

@sin(x) sine of x

@round(x,n) rounds x to n decimal places

@lint(x) fills in any *missing observations* (-0.00001) in the series x by linear interpolation except at the beginning and end.

@bmk(x,y) benchmark function, like @lint, fills in missing values, using y as a mover series. The new series preserves the movement of y as much as possible, while smoothly passing through non-missing values of x.

@rand() generates random numbers with uniform distribution over (0,1)

@normal() generates random numbers with normal distribution: N(0,1).

@mean(x) the arithmetic mean of *x*

@stdev(x) the (sample) standard deviation of *x*

@sum(x) computes the sum of x from dates specified in the last "lim" command. The sum appears in the observation of the last date.

@zero(x) replaces all missing observation signs in x with true zeroes.

@miss(x) replaces all true zeroes in x with missing observations

@peak(y,x,z) y equals previous peak of x with decline rate of z for peak

@pcl(y,x) computes y as y[t] = y[t-1] * (1 + x[t] / 100).

Note: x is the percentage increase over last period, and t starts from the first forecasting period as defined by "fdates".

@pct(x) calculates the exponential growth rate of x (also "@gr()")

@cum(y,x,z) y equals cumulation of x with spill rate of z

@pow(x,z) raises x to the power z

@hpfilter(x,[lambda]) Implement the Hodrick-Prescott filter for *x*. Default values for *lambda* are 14400 for monthly data, 1600 for quarterly data, and 100 for annual data.

NOTE: *z* in these three functions may be a number or a variable; if it is a number, it must contain a decimal point: .0 not 0

@diff(x) the first difference of x, i.e. x - x[1] (can be abbreviated "@d())

@dlog(x) the first difference of the log, i.e. @log(x) - @log(x[1])

Note: z in these three functions may be a number or a variable; if it is a number, it must contain a decimal point: i.e., .0 not 0

The following functions convert the frequency of a variable by aggregation:

@<high>to<low>(x)	converts the high-frequency series x to a lower frequency series by forming the arithmetic mean.
@<high>to<low>e(x)	converts the high-frequency series x to a lower frequency series by applying the end-of-period values.
@<high>to<low>max(x)	converts the high-frequency series x to a lower frequency series by taking the maximum.
@<high>to<low>min(x)	converts the high-frequency series x to a lower frequency series by taking the minimum.
@<high>to<low>s(x)	converts the high-frequency series x to a lower frequency series by taking the sum.

where *high* and *low* must be replaced with *m* (monthly), *q* (quarterly), *s* (semiannual), or *a* (annual). For example, @mtoa(x) converts the monthly series x to annual frequency by forming the average of monthly values. Note that sums and averages are formed over non-missing values, so that missing values are ignored.

The following functions convert the frequency of a variable by interpolation:

@atoq(x)	converts the annual series x to a quarterly series by interpolation. The new quarterly series will have the correct annual total. (The annual series is cumulated; a cubic polynomial is fit to each successive set of four points; the values of the polynomial are calculated at the ends of the quarters. As for the middle two points; these values are differenced to give the quarterly series consistent with the annual totals.
@atoqi(x,y)	similar to @atoq but uses the quarterly indicator series y to pick the points for interpolation. To work correctly, if any value of y is available in a particular year, all values of y must be available in that year.
@qtom(x)	converts the quarterly series x to a monthly series by interpolation.
@stoq(x)	converts the semi-annual series to a quarterly series.

The functions @atoqe and @qtome convert periodicities similarly to the function of the same name except for the 'e' on the end. The 'e' functions, however, apply to end-of-period series, such as the value of assets.

The conversion functions may be used in *G7* but should not be included in Models built with *Build*.

Csum() and Groups

In industry models, you often need to deal with aggregates of many sectoral variables, and this is commonly for well-defined sets of sectors, defined by groups. The format of the @csum function is:

@csum(<name>,<group definition>)

For example:

```
f outsum = @csum(out,1-3 5-10 (7-8) )
```

In this example, outsum is calculated as the sum of out1, out2, out3, out5, out6, out9, and out10. A pair of numbers separated by a dash specifies a range of sectors or numbers to include. Single numbers separated by spaces or commas specify single sectors to include. Finally, any group specification surrounded by parenthesis will be excluded from the summed group. If no group definition is provided, then *G7* will calculate the sum over all vector elements, either for the vector in the vam bank specified by a bank letter or, if no bank letter is given, for the vector in the default vam file. The function also can sum individual series in the workspace or other bank, but the desired sector numbers must be listed.

A named group can also be used in the @csum() function, by prefixing its name with a colon (:). For example, in working with IdLift, we may find the following example convenient:

```
group Durables
   29-58
f empdur = @csum(emp,:Durables)
```

Chain Weighting: @pchain() and @qchain()

Since 1996, the NIPA have used Fisher chain-weighting to calculate aggregate variables in constant prices, and the corresponding price indices. A Fisher index attempts to avoid distortions in index numbers formed by using weights that are inappropriate. For period-to-period movement, it is the geometric mean of a Laspeyres and Paasche index. For longer periods, the period-to-period indexes are chained together. For example, if we have quantity (*Q*) and price (*P*) data on several variables for period 0 and period 1, the Laspeyres quantity index can be written using price weights of period 0:

$$L = \frac{\sum_i P_{i0} Q_{i1}}{\sum_i P_{i0} Q_{i0}}$$

The Paasche index is written using price weights of period 1:

$$P = \frac{\sum_i P_{i1} Q_{i1}}{\sum_i P_{i1} Q_{i0}}$$

The Fisher quantity index is then simply:

$$F = \sqrt{LP}$$

The Fisher price index is calculated analogously, except that the Laspeyres and Paasche components use fixed quantity weights. The convention when creating constant price chain aggregates is to define a base year, in which the price is equal to 1.0, and the quantity is equal to the nominal value. With this convenient definition, the chained quantity multiplied by the chained price yields the nominal value.

The syntax of the chain-weighting function in *G7* is:

@xchain(<list of N quantity variables>, <list of N price variables>, base year)

where 'x' may be 'p' or 'q', and the specification of the quantity and price variables may use group expressions. The function only checks that there is an even number of variables in total. The user is responsible for ensuring that the proper quantity and price variables are entered. Also, take note that the original quantity and price variables must be in the same base year as specified in the function. Here is an example that creates aggregates of personal consumption from the NIPA bank:

```
f qi = @qchain(c030(3-5,7-9), c03(10,11,13,15-19),
               d04(22-24,26-30,32,34-38), 1992 )
```

Both the @pchain() and the @qchain() create two variables in the workspace bank. "chwpi" is the chain-weighted price index, and "chwqi" is the chain-weighted quantity index. The use of the name "@pchain" or "@qchain" is only relevant to which series the function expression will return.

Alternative Chain Weighting: @pchain() and @qchain()

An alternative chain weighting routine also is available in G7. Instead of specifying groups of quantity and price indexes as with the @qchain and @pchain functions, this routine requires groups of nominal levels and groups of quantity indexes. The syntax is given by:

@qchwt(<list of N nominal variables>, <list of N quantity variables>, desired base date [, zero])

@pchwt(<list of N nominal variables>, <list of N quantity variables>, desired base date [, zero])

The @qchwt() function returns an aggregate quantity index, and the @pchwt() function returns an aggregate price index. Note that both this routine and the original chaining routines now support groups of up to 1000 data series. A base date must be provided in order to scale the result, but it need not be consistent with the base date of the source data. Finally, a "zero" option indicates whether missing values should be interpreted as true zeros. At present, it also controls the routine's ability to skip zero aggregates that may precede the actual data and following the actual data; this problem occurs when the fdates interval is too wide. If these features can be made dependable, then the detection routine need not be optional and may be made standard. The routine adds three values to the workspace: chwqi, chwpi, and chwni, which are the aggregate quantity index, the aggregate price index, and the nominal aggregate, respectively. Please keep in mind that this routine is brand new, that it needs additional testing, and that the syntax and specification for this command are subject to change. An example is

```
pce_real = @qchwt( pcez(1-92), pce(1-92), 2000, z )
```

Interpolation

If you do not find the function you want in the above list, but can find the function in some table, you can then help *G7* to provide your function by the general @interp() interpolation function. The format is:

@interp(<filename>,x)

It applies to x whatever function is specified by interpolation points in the named file. Up to 100 interpolation points may be given. For example, to get the vector of cumulative normal probabilities, y, corresponding to the vector of normal deviates x, do

```
f  y = @interp(cumnorm,x)
```

where the file "cumnorm" contains interpolation points for the cumulative normal curve. The "cumnorm" file might contain these lines.

```
-3.3  .0000
-3.0  .0013
-2.5  .0062
-2.0  .0228
-1.5  .0668
-1.0  .1587
-0.5  .3085
 0.0  .5000
 0.5  .6915
 1.0  .8413
 1.5  .9332
 2.0  .9772
 2.5  .9938
 3.0  .9987
 3.3 1.0000
```

Command Files, Groups and Do Lists

Files containing *G7* commands can be prepared with a text editor and then introduced to the program by the command

add <filename> [Arguments]

The commands in the named file then will be executed as if they were being typed in at the keyboard. Add files can contain "add" commands. Add files can "add" themselves. They are commonly used to introduce data, to set up complicated regressions, and to supplement *G7*'s functions. The arguments are optional in an "add" command.

Arguments in Add Files

Add files may have up to 9 arguments. Consider the problem of calculating the percentage change of a quarterly series over the 3-quarter moving average of the series a year earlier. If *x* is the original series, this growth rate, *grx*, could be computed by

```
f ma   = (x + x[1] + x[2])/3
f grx  = (x - ma[3])/ma[3]
```

Those are fairly long formulas, so if you had to do this calculation for many different series, you would find it advantageous to make up a file with a short name like AGR (for annual growth rate) as follows:

```
f ma   = (%1 + %1[1] + %1[2])/3
f gr%1 = (%1 - ma[3])/ma[3]
```

You would use it like this:

```
add agr x
```

The effect will be exactly as if the original pair of lines had been executed. Every occurrence of %1 will be replaced by the variable *x*. You now can calculate the growth rate of any variable; if the name of the variable were *n37z2y*, the command would be

```
add agr n37z2y
```

and the variable *grn37z2y* would be created.

Up to 99 arguments may be used. Here is an example with two. In the example, investment of sector 32 depends on imports of sector 57 in a regression equation.

Make the file INVEST.REG as follows:

```
f d%1   = out%1 - out%1[1]
r inv%1 = out%1, d%1[1], d%1[2], rtb, rep%1, import%2
```

The command "add invest.reg 32 57" will execute these “f” and “r” commands replacing all occurrences of %1 with 32 and all occurences of %2 by 57. What would be executed would be

```
f d32   = out32 - out32[1]
r inv32 = out32, d32[1], d32[2], rtb, rep32, import57
```

Arguments are useful for estimating a similar equation form for different industries, states, or other entities.

An argument may be a text string enclosed in double quotation marks ("). Example: Let the file OUTEXP be

```
ti %1 %2
gr out%1 exp%1
```

then

```
add outexp 17 "Printing and Publishing"
```

would have the same effect as

```
ti 17 Printing and Publishing
gr out17 exp17
```

The last line of the configuration file for *G7*, G.CFG, may contain an "add" command. A typical example might be:

```
Initial add command; add initial.add
```

Such a file would typically be used to select a printer or specify screen or graphics options.

Groups and Group Arguments in Add commands

A special type of argument is a group. The concept "group" comes from multi-sectoral modelling where sector variables have names such as out1, out2,.., out85. A group is a set of specific sector numbers. For example, all industries include sectors 1-85; non-durable industries include sectors 5-13, and durable industries include sectors 14-24, and so on. Here we have three groups of sector numbers: 1-85, 5-13, and 14-24. To use a group as an argument, we must enclose those sectors in parentheses like this:

```
(1-85), (5-13), and (14-24).
```

Groups also may specify sets of spreadsheet columns. For example, to construct a group of the first 30 columns of a spreadsheet, the group should be

```
(A-AD)
```

Groups also may contain named groups. For example, suppose to construct a group of all non-chemical manufacturing sectors, where manufacturing and chemical sectors are defined as

```
group Manufacturing
1-58
group Chemicals
```

```
20-27
```

then a group to include all manufacturing sectors except chemicals may be specified as

```
( :Manufacturing ( :Chemicals ))
```

This is equivalent to

```
( 1-58 ( 20-27 ) )
```

Note finally that these various specifications may be combined. For example, the following is another alternative to the example above.

```
( :Manufacturing ( 20-27 ) )
```

An "add" command allows its last two arguments to be groups. Take the earlier example of calculating the percentage change of a series over a 3-quarter moving average. Suppose we have 85 quarterly output series: *out1* through *out85*. We may modify the AGR file in the earlier example to become the following:

```
f ma     = (%1%2 + %1%2[1] + %1%2[2])/3
f gr%1%2 = (%1%2 - ma[3])/ma[3]
```

we then could write out 85 "add" commands as follows:

```
add agr out 1
add agr out 2

add agr out 85
```

Equivalently, we can write them out in only one "add" command:

```
add agr out (1-85)
```

In this example, the variable *ma* acts like a temporary variable. And 85 new series, *grout1* through *grout85* are created in the workspace. If we similarly want to compute the moving average series for another variable, such as industry sales, *sale1* through *sale55*, we may use the command:

```
add agr sale (1-55)
```

It should be stressed that up to two group arguments are allowed. However, group arguments can only be the last ones on the argument list. If two group arguments are used, by default, the

outer loop is controlled by the first group argument, and the inner loop by the second group argument. For example,

```
add invest.reg (32-34) (52-54)
```

is equivalent to

```
add invest.reg 32 (52-54)
add invest.reg 33 (52-54)
add invest.reg 34 (52-54)
```

There is an alternative to the double loop, parallel matching, with an 'm' option after the second group. For example,

```
add invest.reg (32-34) (52-55(53)) m
```

is equivalent to

```
add invest.reg 32 52
add invest.reg 33 54
add invest.reg 34 55
```

That is, the first members of each group are matched to be the first pair of arguments, the second member of each group to be the second pair, and so on. The two groups must have equal number of members in parallel matching.

Loop with Do Command

A "do" command allows you to run a set of *G7* commands like an add command without creating the add file. Its format is

do <{G commands with variables}> [Arguments]

where variables are %1, %2, etc., just as in an add file. A do command can continue on several lines. However, the arguments should be placed on the same line as the closing brace mark '}'. As with the add command, no more than 99 arguments are allowed, and the last two arguments may be groups. For example, the AGR example can be rewritten with a do command like this:

```
do {f ma     = (%1%2 + %1%2[1] + %1%2[2])/3
    f gr%1%2 = (%1%2 - ma[3])/ma[3]
    } out (1-85)
```

In this case, however, the file AGR need not be created. A "do" command thus provides an easy way to have a loop.

Nesting of do loops is possible. A loop may be nested within an add file or within another do loop. For example, the following code computes the sums of the rows of matrix 'X'. The values are stored in vector rowsum. Note that an argument is passed from the outer loop to the inner loop. This is accomplished by using a variable from the outer loop to specify a group for the inner loop.

```
do{f sum = 0
   do{f sum = sum + X%2.%1
      }(%1) (1-5)
   vf rowsum%1 = sum
   }(1-5)
```

Command File with an Argument File

fadd <CommandFile> <ArgumentFile> [<arg> [<arg>] ...]

The named CommandFile is first executed with the arguments from the first line of named ArgumentFile, then the CommandFile is executed again with the arguments from the second line of the ArgumentFile, and so on until the ArgumentFile is exhausted. Additional arguments may be supplied with the fadd command, and these will be added to each line of arguments provided in the arguments file. For example, the CommandFile might be

```
ti %2
gr %1
```

while the ArgumentFile was

```
gnp$    "Gross National Product"
vfnre$  "Investment in producer durable equipment"
vfnrs$  "Investment in non-residential structures"
```

Then the result would be three properly titled graphs. The argument file may not have any comments.

User-defined Functions

function <function name> {G commands with variables}

G7 allows the user to define functions so that similar operations can be repeated easily. This feature is very similar to the add command described above, but it allows the entire script to be specified in a single file. Once a function is defined, it may be used repeatedly.

Functions may contain arguments; see the description of the add command for details. A good practice is to specify function names that contain at least one capital letter. This will avoid confusion between the user's functions and G7 keywords. Once it is defined, a function may be called by entering the function name in the same manner as other G7 commands. If arguments are to be passed to the function, they should be listed after the function name.

Note that the addprint setting automatically is set to no within each function, so that the function contents are not printed to screen during processing. For debugging purposes, addprint commands can be given within the function to override this practice.

G7 supports recursive techniques. For example, the function command may be employed to construct a Fibonacci sequence. The only restrictions for this sequence are the inherent limits of the 32-bit software, but even so, up to 47 iterations are possible before the limit is reached.

The following code repeats the example above while employing the function command.

```
function GR{
            ti %2
           gr %1
  }
GR gnp$    "Gross National Product"
GR vfnre$  "Investment in producer durable equipment"
GR vfnrs$  "Investment in non-residential structures"
```

function list

This routine will print a list of names of the user-defined functions.

function print [<function_name>]

This routine will print the specification of the function function_name. If no name is provided, then the specification of every function will be printed.

Other Commands That Are Useful in Add Files

catch <file_name>

Captures screen displays (except graphs) to the named file. Used to capture the set up of a regression and its results into a single file. A regression display is caught as it stands when you proceed to the next command. To turn off the catching type "catch off"

(addt)ype<n | y>

(addpr)int

Invoked with an "n" for "no", this command turns off the typing on the screen of the contents of the "add" files. It speeds up the processing of large data files. Turn printing back on with "addtype y" for "yes". This command is also known as "(addp)rint".

zip

After the "zip" command has been given, the program will not do the graphs in "add" files and therefore not pause. It also does not calculate the "lever" variable after each regression. It is particularly useful for rapid re-estimation of an entire model when data has been updated. The command to turn off "zip" is "zip off".

Strings in G7

Strings may be defined and referenced by name. These same tools may be used to open an arbitrary text file and parse the content found there.

str <name> = <rhs>
Create a string named *name* with definition *rhs*. The string definition *rhs* may be specified as text presented within quotation marks, given as the name of a previously-defined string, or defined as a sequence of text and/or strings linked by '+'.

```
str concept  = "Output"
str industry = "Coal Mining"
str display  = concept + "of the " + industry " industry."
```

str save <filename>
Store all defined strings in a text file *filename*. Afterward, the strings may be loaded into memory again.

```
str save myStrings.txt
add myStrings.txt
```

str print
Print each string to the screen.

str clear [<stringname>]
Erase all defined strings from memory. If a string name is given, then that string will be removed from the list of user-defined strings.

str replace [<stringname>] <"search_text"> <"replacement_text">
This function operates on the string *stringname*, if provided, or the last line read by *str getline*. Any text in this string matched by the string *search_text* is replaced by the string *replacement_text*. Matching employs regular expressions as defined in the C++ Boost library. More details for regular expressions are provided on the following page.

NOTE: The Visual C++ Redistributable package must be installed in order to use some of the newest features. The VC++ installer should be available at C:\PDG\C++Install. Run this installer (run as Administrator if using Vista or Windows 7) before attempting to employ the *str replace* command.

The following routines are employed in the parsing of a text file.

str open <filename>
Open a file for parsing with the *string* command.

str close

Close the file that was opened with the *str open* command.

str getline [<string_name>]
Read a line of text from the file opened by *str open*. Store the text as a string named *string_name*. Store a copy of the string with the name "line;" this string may be referenced by related routines, but it does not appear in the list of user-defined strings.

str parse [<string>] ["<separators>" [“<string_name>”]]
Split the string named *string* into words separated by any one of the characters listed as *separators*. If no string name is provided, then the routine acts on the last line read by str getline. The default separators list is composed of the space and tab characters, or " \t". The words are stored as a list of strings that may be accessed with the %w() function, which has a syntax similar to the %s() function described below. If a string name is suppied, it will be used as the root name of the stored words.

NOTE: The Visual C++ Redistributable package must be installed in order to use some of the newest features. The VC++ installer should be available at C:\PDG\C++Install. Run this installer (run as Administrator if using Vista or Windows 7) before attempting to employ the *str parse* command.

Several keywords and functions aid in the parsing of text files.

%linelen
Returns the length of the last line read by *str getline*. If *getline* encounters the end of the file, then a value of -1 is returned by *%linelen*.

%line
The *%line* keyword allows the user to access the string last read by *str getline*. It especially is useful when no string name was provided for the string.

%numwords
The *%numwords* keyword returns the number of words parsed from the line currently in memory or the string last processed by the *str parse* command. After calling *str getline*, the number of words is 1 (*%line* == *%w(1)* and *%numwords* == 1) until a subsequent command *str parse* is issued.

%w(<index> [,<first>[,<last>]])
The *%w()* function provides access to the list of words created by the *str parse* routine. *index* is an integer that specifies the position of the desired string within the list of words created by *str parse*. The optional first and last parameters are integers that specify the positions of the first and last characters to define a substring of the word in position *index*.

The definition of a string may be recovered at virtually any point within a *G7* script by using the %s(<name>) function. To employ our example string “display”, which is

defined as “Output of the Coal Mining industry.”, as a graph title, we simply issue the following command.

```
gr %s( display )
```

Additional functions are available that compare strings, calculate the length of the string, and perform other operations.

Variables and Arguments in G7

Variables in *G7* were introduced earlier for use in add files, fadd files, do loops, and elsewhere. This section provides additional details.

The treatment of variables in *G7* are similar to the treatment in DOS batch files. Up to 99 variables typically may be passed to an add or fadd file, within a do loop or function, and so on. The first argument that is passed to a file is referenced by '%1', the second by '%2', …, and '%99'.

The appearance of a percent symbol (%) within a *G7* script always signifies that the following character or word should be interpreted as a variable, a keyword, or a function. In each case, when the variable, keyword, or function is encountered and evaluated, the value of the result effectively will be inserted into the script and subsequently processed by *G7*.

Variables

Variables in *G7* were introduced earlier. All arguments are passed as text to the add file, do loop, or other routine. Variables are referenced as %1, …, %99.

%{}

Integers often are passed to an add file or function, and integers are used as indexes for a do loop. Sometimes a calculation is needed that is based on that integer, but some adjustment is needed. The %{} operation allows such calculations. Numerical variables and constants may be referenced within the brackets, and addition, subtraction, multiplication, and division of variables and constants is permitted. For example, suppose we need to print vector data to a spreadsheet. Suppose the vector elements range from 1-10, and we want to print these series in rows 6-15. We could use the "xl vecwrite" command, or we could construct a do loop like the following:

```
do{ xl write A %{5 + %1} vec%1 2000 2010 }(1-10)
```

Note that for each iteration '%1', element '%1' is referenced in the "vec" vector, and that series is written to column '5+%1' in the spreadsheet.

Another common use of %{} is to provide unambiguous specification of variables within a script. Recall that variables '%1' through '%99' are legal. However, there may be ambiguity between a two-digit variable and a single-digit variable that is followed by another digit that is not part of the variable. For example, if '%9' is

followed by a '9', it looks the same as '%99' and so *G7* will attempt to fill in the 99-th variable definition. To remove such ambiguity, use the following approach:

```
f x = %{%9}9   # Variable 9, followed by '9'
f x = %99      # Variable 99
```

%floor(<number>, <number>)

%ceiling(<number>,<number>)

These function return the value of nearest integer, either rounding down or up, respectively.

%min(<number>,<number>)

%max(<number>,<number>)

This function returns the minimum or maximum values of the two numbers provided.

%mod(<number>,<number>)

The modulus function returns the remainder of dividing its first argument by the second.

Keywords

Several keywords were introduced earlier. The following is a summary of keywords in *G7*. Availability of particular regression statistics depends on the type of regression performed.

- %date Insert the date.
- %time Insert the time.
- %NARGS The number of arguments that currently are in scope.
- %% Treat this as text, evaluated as a single '%'.
- %fdateX Recover the first or last *fdates* setting, where X = {1,2}.
- %gdateX Recover the *gdates* setting, where X = {1,2,3}.
- %tdateX Recover the first or last *tdates* setting, where X = {1,2}.
- %limX Recover the *limits* setting, where X = {1,2,3}.
- %title Recover the current graph title.
- %subtitle Recover the current graph subtitle.
- %vaxtitle Recover the current graph vertical axis title.
- %ncoef Recover the number of coefficients in the last regression.
- %betaX Recover parameters from the last regression, X={1,…,*ncoef*}.
- %mexvalX Recover the mexvals from the last regression, X={1,…,*ncoef*}.
- %see Recover the standard error of the estimate of the last regression.

- %rsq Recover the r-square from the last regression.
- %rho Recover rho from the last regression.
- %rbarsq Recover the r-bar-square from the last regression.
- %mape Recover the mean absolute percentage error of last regression.
- %dw Recover the Durbin-Watson statistic from the last regression.
- %nobs Recover the number of observations in the last regression.
- %obsinregression Recover the number of observations in the last regression.
- %logl Recover the log likelihood statistic from the last regression.

Functions

A variety of functions also are available. The following is a summary of such functions in *G7*.

%xlcol(<int>)

This function maps positive digits to letters. Typically, <int> will be a variable (e.g. '%1') that is assigned an integer value. For integer arguments greater than 26, the results will follow the convention for identifying Excel spreadsheet columns. The following is the mapping for the positive sequence of integers: A…Z AA … AZ BA … BZ ….

%getval(<expression>, <date>)

The value of *expression* in period *date* is retrieved. Any algebraic expression that is legal for the *f* command may be given, or a variable may be named either from the workspace or from the specified bank.

%s(<string> [, <first>[, <last>]])

Insert the value of the named string *string*, where *string* previously was declared and defined with the str command. Optional parameters *first* and *last* indicate the positions of the first and last characters of a desired substring of *string*.

The following functions operate on strings. The strings either may named strings that were defined by the str command or they may be provided as text or a variable that is surrounded by quotation marks (e.g. "x1" or "%1").

%strlen(<string>)

Calculates the length of *string*. This follows the standard C libraries.

%strcmp(<string1>,<string2>)

Compares *string1* and *string2*, with case sensitivity. Value is 0 if strings match, following the standard C libraries.

%strcmpi(<string1>,<string2>)
Compares *string1* and *string2*, without case sensitivity. Value is 0 if strings match, following the standards of the C libraries.

%strncmp(<string1>,<string2>,<N>)
Compares the first *N* characters of *string1* and *string2*, with case sensitivity. Value is 0 if the first *N* characters of the strings match, following the standards of the C libraries.

%strncmpi(<string1>,<string2>,<N>)
Compares the first N characters of *string1* and *string2*, without case sensitivity. Value is 0 if the first *N* characters of strings match, following the standards of the C libraries..

%strstr(<string1>,<string2>)
Looks for the first instance of s*tring2* within *string1*. If found, then return the portion of *string1* beginning at that point. Otherwise, return an empty string. This is similar to the standard C libraries.

%strnset(<string>, <N>)
Construct a string composed of *N* repetitions of *string*. The length of the resulting string is limited to 90 characters. This function loosely follows the standard C libraries.

%lower(<string>)
%upper(<string>)
%titlecase(<string>)
Returns a copy of *string* after converting it to lowercase letters, capital letters, or title case (where the first letter of each word is capitalized).

%trim(<string>[,<separators>])
%trimleft(<string>[,<separators>])
%trimright(<string>[,<separators>])
Returns a copy of *string* after removing extraneous characters from the beginning, end, or both ends of *string*. The default behavior is to remove whitespace and control characters. Other characters *separators* optionally may be given in the syntax of the *%eliteral()* routine.

%literal("<string>")
The *%literal()* function prevents the characters of *string* from modification by the *G7* parsing engine. This allows character sequences to be passed through that otherwise might prompt an error or other undesirable action.

%eliteral("<string>")
The *%literal()* function is identical to the *%literal()* function except that it escapes the sequences '\0', '\a', '\b', '\t', '\f', '\n', '\r', '\\', converting each to its corresponding ASCII code. For example, '\t' is converted to the ASCII code for the tab character.

The Resector Command

The purpose of these tools is to provide a convenient way to aggregate and disaggregate data from various sectoral levels. The routine begins by reading a file that contains concordances between various sectoral aggregation schemes. The column heading on the first line must specify the number of sectors in that aggregation scheme. This number of sectors is also the "handle" that is used to refer to that column in the resector routines.

Please see the demo section of the Inforum web site for examples of the resector capabilities.

rs create <filename> [<Number of Columns> <Maximum Number of Sectors>]

The create function opens the key file and reads in up to 10 columns of aggregation information (6 is the default). An optional argument can be supplied to limit the reading to more or less columns than the default. In principle, the routines handle aggregation or disaggregation between any two of the schemes read from the file. With six aggregation schemes, this results in 36 possible combinations of aggregation.

The beginning of a sample file is displayed below:

```
575  495  360  432   85  BEA82
  1    1    1    1    1  10100   Dairy farm products
  2    2    2    2    1  10200   poultry and eggs
  3    3    3    3    1  10301   Meat animals
  4    4    3    3    1  10302   Miscellaneous livestock-horses,bees,ho
  5    5    4    4    1  20100   Cotton
  6    6    5    5    1  20201   Food grains: wheat,rye,rice,buckwheat
  7    7    5    6    1  20202   Feed grains: corn,oats,barley,hay,sorg
  8    8    5    6    1  20203   Grass seeds
  9    9    6    7    1  20300   Tobacco
 10   10    7    8    1  20401   Fruits
 11   11    7    8    1  20402   Tree nuts
 12   12    7    9    1  20501   Vegetables
 13   13    7   10    1  20502   Sugar crops
```

rs formagg <Number From> <Number To>]

This is the first function that should be called after creation. It performs the most important initialization tasks. It sets up all of the information that is needed to aggregate from the scheme indicated as Number From to the scheme indicated as Number To. It sets up concordance lists and "split lists" that will be used by other functions listed below. In some programs, you may need to issue this command several times, especially if you need to use some of the "cross-aggregation" techniques described below.

One of the functions of initialization is to set up lists of correspondences between sectors, as well as lists of splits, where one sector from one scheme corresponds to one or more sectors from the other scheme. Since this initialization is time and memory consuming, an explicit function called *formagg* performs this task, and this function is called for only needed aggregation relationships. *formagg* takes as its arguments the maximum sector numbers of the source and destination schemes. Until *formagg* has been called with a certain aggregation pair, no other functions using that pair are allowed.

rs aggvector <Number From> <Number To> <Input Vector> <Output Vector> <Split Vector>]

This function is designed to aggregate or split a vector from one aggregation scheme to a vector of another aggregation scheme. Number From should be the number of sectors of the source vector, and Number To should be the number of sectors of the destination vector, as shown in the heading in the key file that was read from the create routine. Split Vector must be of the same aggregation level as the destination vector. It is used by the routine where there is a one-to-many relationship going from source to destination sectors. If you purely are aggregating, the SplitVector will not be used and its values may be set to arbitrary levels.

rs ctrlvec <Number From> <Number To> <Detailed Vector> <Aggregate Vector>

This function controls a detailed vector to a more aggregate vector.

rs rdctrlvec <Number From> <Number To> <Detailed Vector> <Aggregate Vector>

This function controls a detailed vector to a more aggregate vector, using right-direction scaling.

rs crossaggvector <Number From> <Number To> <Number In Between> <Input Vector> <Output Vector> <Split Vector>

Sometimes conversion of a sectoral scheme is more complicated than merely a combination of aggregations and splits. For example, there may exist a many-to-many relationship, where a SplitVector of either sectoring level would not provide enough information on how the flows should be allocated. The name "cross-aggregation" indicates the method of translation used by this function, whereby the source vector is split to the level of a more detailed intermediary vector, which then can be aggregated directly to the destination vector. It must be possible to aggregate the intermediate sectoring level both to the source and to the destination levels for this function to work.

In the argument list, Number From is the number of sectors of the source vector, Number To is the number of sectors of the destination vector, and Number In Between is the number of sectors of the intermediary vector. Input Vector is the source vector and Output Vector is the destination vector. In this function, Split Vector is a vector of length Number In Between that is used to split the source vector.

rs aggmatrows <Number From> <Number To> <Input Matrix> <Output Matrix> <Split Vector>

rs aggmatrows <Number From> <Number To> <Input Packed Matrix> <Output Matrix> <Split Vector>

This function is similar to *aggvector* but aggregates the Input Matrix by row to obtain the Output Matrix. The routine also will accept a packed matrix as the input, though the Output Matrix must be a full matrix.

rs aggmatrix <Number From> <Number To> <Input Matrix> <Output Matrix>

This function aggregates a matrix both by rows and columns, and assumes that both the source and destination matrix are square.

rs ctrlmatrows <Number From> <Number To> <Input Matrix> <Output Matrix>
rs ctrlmatrows <Number From> <Number To> <Input Packed Matrix> <Output Packed Matrix>

This function controls a more detailed matrix to an aggregate matrix.

rs ctrlmat <Number From> <Number To> <Input Matrix> <Output Matrix>
rs ctrlmat <Number From> <Number To> <Input Packed Matrix> <Output Packed Matrix>

This function controls all cells within a block of the more detailed matrix to a single cell of the less detailed matrix. The function handles all of the different cases: single cell to single cell, row vector to single cell, column vector to single cell, and sub-block to single cell. The function is useful for updating a more detailed matrix such as a benchmark IO table to a more aggregate (and more current) matrix.

rs crossaggmatrows <Number From> <Number To> <Number In Between> <Input Matrix> <Output Matrix> <Split Vector>

This function works much like *crossaggvector* but aggregates rows of a matrix.

Commenting Add Files

Add files are like small computer programs, and should be commented. A line eginning with a pound sign, '#', is treated as a comment. A line beginning ith a colon, ':', causes that line and all subsequent lines to be treated as a comment until a line beginning with a colon is again encountered. (The line of the second colon is also a comment.)

Examples:

```
# This is a one line comment.
: This comment, however, is longer.
Clearly, it is a much more important comment,
for it takes three lines.
: Or four.
```

Use of the Group Titles Feature

It is often the case that you know the sector number of a series, and need to know a title. This is where the group titles features comes in handy. There are two commands that work together, "gtfile" and "gtitle".

gtfile < titles file | "off" >

This command scans the specified titles file and stores the starting position of each line in the file. It makes it possible to use the "gtitles" command, which will supply a title from the titles file if given a number. To close the titles file, thus freeing up some memory, or to allow for a different titles file to be used, do "gtitle off".

Example titles file:

```
1 "Agriculture, forestry, and fisheries"
2 "Metal mining"
3 "Coal mining"
4 "Natural gas extraction"
5 "Crude petroleum"
6 "Non_metallic mining"
```

gtitle <number>

This command assumes that a "gtfile" command has been given, with a valid titles file as an an argument. The "gtitle" command will then use the n'th line of the titles file to put a title in the following "graph" command. These commands, "gtfile" and "gtitle" are particularly handy in combination with the "do" command, or with an add file with group arguments. Since in these cases only sector numbers are being passed to the add file, the "gtitle" command allows you to specify a sector number, and get back the title for that sector.

Pausing to View Output in Results Window

If you have a very long add file, the results may all rush by before you have a chance to examine them. There are a number of ways to solve this problem. The simplest is just to scroll the results window up to examine the output that has flashed before you. A second is to use the "catch" command to capture most of the output to a file, where it can then be examined with the editor. Occasionally it is convenient to just pause the operation of the add file for a moment. One way to do this is with the "pause" command. Just include the command "pause" on a single line in your add file, and *G7* will stop when it reaches that point, and display a message box on the screen. At this point, hit an 'ESC' to stop the addfile, or any other key to continue.

pause [<“message”>]

The pause command causes *G7* to pause, print an optional message, and prompt the user to continue or to cancel the process. Note that if more than one word is to be included, the message must be surrounded by quotes.

Another way to pause is to use the "addpause" command:

addpause < y | n>

This command cause *G7* to pause just before the beginning of each add file, with a message box to the screen. This pause feature also allows you to hit an 'ESC', or click on the 'ESC' button to stop the add file sequence.

Reading Data into G7

G7 has seven commands related to introducing data:

"data"	introduces a single series prepared in *G7*'s data input format
"matdat"	introduces one or more series in a column format usually found in spreadsheet programs
"update"	adds new observations or revises observations in series already in a data bank
"matup"	updates from column-format data
"monup"	updates a quarterly series with monthly data
"wrindex"	turns on or off the writing of the index file of the workspace
"addtype"	turns on or off the printing of data as it comes in

More detail on the commands is given below.

(da)ta <series_name>

Introduces data into the workspace data bank. There are 2 forms the command can take.

Form 1:

```
data sales
      83.1  34.0 56.8 44.5 55.6
      84.1  39.3 41.2 43.9 47.0 ;
```

The first number on each line is the date of the first observation on that line. Input is terminated by a ";". The ";" may be omitted in an "add" file, except on the last line of the file.

Form 2:

```
data sales 83.1
      34.0 56.8 44.5 55.6
      39.3 41.2 43.9 47.0 ;
```

The date of the first observation is given on the command line immediately after the series name. No dates appear on subsequent lines.

Form 1 is easiest if the data is being entered by hand. Form 2 is easier if the data is generated by a program.

Input data may contain floating point numbers in exponential form. Thus, the number 3 may be represented as 3.0, 3.0E+00, .300E+01, or 30E-01. Only E, not e, is recognized in this context. Any number of spaces between data observations is allowed (the data are free format).

A missing value may be represented by a single question mark ?. Therefore

```
data sales
      83.1  34.0 ? 44.5 55.6
```

indicates that the sales for 83.2 quarter is missing.

(up)date <series_name>

Works like the data command (Form 1) but updates an existing series. The data following the command contain only the data points to be changed. The updated series is placed in the workspace only.

monup(mup) <series_name>

This is used for updating a quarterly series with monthly data. Example:

```
mup rtb
   83.3  9.120 9.390 9.050   8.710 8.710 8.960
   84.1  8.930 9.030 9.440;
```

rtb is a quarterly series being updated with monthly data. 9.120 is the value of *rtb* in July 1983; 9.390 is the value in August 1983, etc.. Note that quarterly dates are the first numbers on each line and three monthly observations must be present for each quarter.

matdat [date]

Brings data into *G7* prepared by a spreadsheet program. There are two major forms in which the series can be arranged. The series can be arranged either in vertical columns with the name of the series at the top of the column, or in horizontal rows with the names appear on the same line of the matdat command. Up to 20 columns of numbers occupying up to 160 characters per line may be given. The two forms of the command are:

Form 1:

```
matdat
          gnp      c        cd       cnd      cs
 81.1     1513     950      146      359      445
 81.2     1512     949      140      361      448
 81.3     1522     956      143      361      450 ;
```

Dates appear as the first number on each line. Here 1513 is the value of *gnp* in 1981.1, 1522 is the value in 1981.2, etc.

or

```
matdat 81.1
gnp      c        cd       cnd      cs
1513     950      146      359      445
1512     949      140      361      448
1522     956      143      361      450 ;
```

The date of the first observation appears on the command line, and no dates appear on subsequent lines. Use a semicolon to end the data. (*You must have a semi-colon on the last line of data for either form.*)

Form 2:

```
matdat  gnp out(1-4) 81.1
 1513    1512    1522
 950     949     956
 359     140     143
 359     361     361
 445     448     450;
```

Here, series names are given immediately after "matdat" with starting date given as the last argument on the "matdat" line. The first series reads from the first line after the "matdat", the second series reads from the second line, and so on. Note also that group definitions are allowed here.

matup [date]

Uses the same format as the "matdat" command, but updates series already in the bank.

To use the matdat or matup commands effectively, create an "add" file containing the data or "matdat" statements necessary to bring in the data. For example, to transfer data from a spreadsheet where the data already exist in columns, use the following steps.

Print the spreadsheet data to a file. (In LOTUS, you would print the data range to a file, creating a PRN file.) Add the necessary *G7* data commands to the file just created using an the editor. (Alternatively, include the commands in the spreadsheet before you print the range.) If the column data do not include dates, use Form 2 of the matdat command.

From the *G7* console, use the "add" command to bring in the data stored in the file created by STEP 1.

```
add filename_created_in_STEP_1
```

Speeding Processing of Long Data Files

When adding long data files to *G7*, two commands can considerably accelerate processing the data. Include these commands as part of your add file to introduce data, but only after you are sure the add file is trouble free.

(addt)ype n

Normally, *G7* writes to the screen all of the input file when an "add" is underway. Turning screen writing off with this command speeds up the operation greatly. Turn screen writing back on again at the end with

```
addt y
```

(wri)ndex <y | n>

Normally *G7* writes the entire index file of the workspace as each variable is added. When many series are added, this feature lengthens processing time. To defer writing the index until the entire add file is processed,"wri n" turns off writing and "wri y" turns it on again and writes the index. A "q" with writing off will cause the index to be written before quitting, so there is no danger of losing everything by forgetting "wri y".

Writing Text Files

Here are some *G7* commands commonly used for writing data to ASCII output files.

tdates <date1> <date2>

Sets the dates for subsequent typing commands without actually printing out any data.

(ty)pe <series_name> [date1] [date2]

Displays values for the named series from date1 to date2 on the screen. The first number on each line is the date of the first observation on the line. Dates may be omitted if they are unchanged from the previous type command. If the "save" command (see below) is on, the displayed values are transferred to the save file where they are preceeded with an "update" command.

(sty)pe <series_name> [date1] [date2]

"Silent" type writes the series to the currently open "save file" without displaying data on the screen. This can speed processing a long add file.

matty [file <filename>] <var1> [dp1] [var2] [dp2] ... [var400]

The command's name abbreviates "matrix type". It displays time series data for up to 400 variables (var1, var2, ...) in vertical columns, with time running down the page. Data are shown for the range of dates specified by the preceding tdates command. Following each variable name, there can be an optional digit, dp, specifying the number of decimal places with which it is shown. The specified number of decimal places continues to apply to subsequent variables until a new value is specified.

For example, with the Quip data base assigned, the commands

```
tdates 1990.1 1991.4
      matty gdp 1 c v fe fi g
```

shows the results:

Date	gdp	c	v	fe	fi	g
1990.100	5660.6	3759.2	822.7	541.6	615.9	1153.0
1990.200	5750.8	3811.8	835.0	554.8	615.1	1164.3
1990.300	5782.2	3879.2	804.7	555.5	634.1	1176.9
1990.400	5781.7	3907.0	736.3	577.3	649.2	1210.4
1991.100	5821.9	3910.7	723.5	577.4	610.3	1220.6
1991.200	5892.5	3961.0	716.4	602.7	615.0	1227.4
1991.300	5950.2	4001.6	744.1	602.6	624.5	1226.5
1991.400	6002.1	4027.1	760.7	624.4	639.3	1229.2

If the first word after the command is "file", then the next word is the name of a file to which the results will also be written. Example:

```
matty file nipa.dat gdp 1 c 2 cd  v 1 vfnre 2 vfnrs
```

The results are shown on the blue Results screen and then displayed in a grid. If saving is on (as a result of a "save <filename>" command explained below), the results go also into the

save file. The grid display is convenient for scrolling about to look at the data. One can also copy data from the grid display to the Windows clipboard.

save <filename> [<type>]

Opens a file of the given name to receive output produced by type, stype, matty, regression, and other commands. Most commands executed by *G7* are stored in the "save" file. A save file is closed and saving terminated by

```
save off
```

The optional <type> argument is the command that will be printed when using "ty" or "sty" commands. The default is "update", but it may also be "data", "vdata", "ovr", "ind", "cta" or "gro".

Writing to Lotus .WK1 Files

There are two related commands for writing many series of data to Lotus .WK1 files, either in row or in column format. The Lotus .WK1 file can hold up to 256 columns and 4096 rows. Therefore, if you want to write many series, with less than 256 observations, use “rp123”, which writes the series by row. If you have a few series with many (>256) observations, use “p123”, which writes by column.

p123 <file_name> <date1> <date2> [width] [decs]<ser1> [<ser2> <ser3> ... <sern>] ;

or

p123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> ... <sern>] ;

This command is similar to the matty command, except that the data is written directly to a *Lotus* worksheet. Do not include the WK1 extension with the filename; *G7* automatically appends the extension WK1. <date1> and <date2> are the desired starting and ending dates. Up to 239 series names can be entered in free format. End the list with a ';'. Width and decs are optional; width specifies the display width within *Lotus* for all series transferred, decs specifies the number of decimals displayed. Defaults are 9 and 3 respectively. p123 worksheets are compatible with *Lotus*, *Excel*, *Quattro Pro* and many other programs.

Example 1:

```
p123 natacct 60.1 91.1 8 1
gnp c v vf vi fe fi g
```

Example 2:

```
p123 natacct 60.1 91.1 8 1 out(1-85)
```

rp123 <file_name> <date1> <date2> [width] [decs]

<ser1> [<ser2> <ser3> ... <sern>] ;

or

rp123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> ... <sern>] ;

This command works like "p123" to write series into a *Lotus* 1-2-3 worksheet; but, whereas p123 writes the series as columns, "rp123" writes them as rows. With "p123", one can have up to 255 series, each with up to 4096 observations. With "rp123", one can have up to 4096 series, each with up to 255 observations. For many annual and quarterly data banks, "rp123" may prove the more efficient format.

Making Tables

The *Compare* program is used to make tables with the results of running the model. As the name suggests, it is particularly adapted to comparing results of several runs of a model, but it can also list the contents of a data bank or the results of a single run of a model. When being used to show a base case and several alternatives, it can show the alternatives as actual values, as deviations from the base, or as percentage deviations from the base. The results are written to a file which can then be sent to a printer. To use *Compare*, one must first prepare a stub file. Details are given below.

Compare is a separate program but can be run inside *G7* or by selecting Model | Tables.

Selection of this menu item opens a form asking for the information necessary for *Compare* to run. Clicking OK then executes *Compare*. If you know that you want to run *Compare* with exactly the same entries on the form as the last time *Compare* was run, you can skip form by clicking Model | Express Tables.

Details on the Stub File

Here is an example of a stub file

```
\dates 1995.0 1996.0 1997.0 1996.1 1996.2 1996.3 1996.4 1995-1996
1996-1997
*
;
;                       THE AMI MODEL
;
&
gdpR         ;Gross national product
cR           ;Personal consumption
pibgR        ;Pers. income w/o gov't
pidisR       ;Personal disposable income
taxrate*100 ;taxrate*100 *
gtnisR       ;Gov trans, net int, sub.
```

```
cpcR        ;Consumption per capita
pop*1000    ;Population (in millions) *
emp*.001    ;Employment
```

The first line may give the dates of the periods which are to be printed. In a quarterly data bank or model, an annual date such as "1998.0" will print the annual average for the year. Dates shown with a hyphen will display the growth rate between the two periods shown. For instance, the above stub file will make a table with the annual growth rates between 1995 and 1996 and then 1996 and 1997. Alternatively, the first line may say "Ask dates"; *Compare* will then ask the user to provide the dates from the keyboard.

Compare has several editing features which are controlled by special characters at the beginning of a line. They are:

* Go to the top of a new page and print the titles of the alternatives runs as given in the ".fix" file.

* *m* If there is not room for m more items on the page, go to a new page and print the dates across the top.

* *m n* If there is not room on the page for *m* more items plus *n* lines, go to a new page and print the names of the base and alternative runs.

; Print the line just as it stands.

& Print the dates across the page above the appropriate columns.

\ *fw dp pl tm bm tw*

where *fw dp pl tm bm* and *tw* are all numbers. This is an optional command to set the field width of each printed number to *fw*, the number of decimal places to be printed to *dp*, the page length to *pl* lines, the top margin to *tm* lines, the bottom margin to *bm* lines, and the width of the titles to *tw*. The last three may be omitted. The default values are 7 1 60 3 9 32.

\ g where g is the letter g. Show the growth rates in percent for the variables named on the following lines. The growth rate will be for the time period between adjacent columns in the table

\dates Do not use this if you have specified growth rates in your \dates command using the hyphenated dates.

\ n where n is the letter n. Cancel a previous \ g.

\ ar use if monthly or quarterly data is given at annual rates (the default). (ar = annual rates)

\ pr use if monthly or quarterly data is given at monthly or quarterly rates (pr = period rates).

Many other commands that begin with the '\' character are available. A line beginning in any other way will be presumed to begin with a variable name, such as *gnp$*, or an expression. For the expressions, all the functions available in *G7* are also available in *Compare*. After the name

or expression comes a ';'and after the ";" come up to thirty characters which will be printed at the left side of the line. Leading blanks -- blanks between the ";" and the first visible letter on the line -- will be printed and can be used to indent the titles.

Please see the *Compare* documentation for more details on using *Compare*.

Drawing Graphs

G7 can make a variety of graphs, which can then be printed, saved into Windows metafiles to be included in documents, or copied to the windows clipboard to be pasted into other applications. Here are the basic graph commands and related subsidiary commands. Detailed explanations appear on the following pages.

title	Provides the title for the graph.
subti	Provides the subtitle.
vaxti	Provides the vertical axis title.
vaxl	Puts vertical axis labels inside or outside the rectangle of the graph.
graph	Graphs up to six series on one graph with one scale.
mgr	Multi-scale graph. This is most useful with two series, since the left axis scale will be for the first series, and the right axis scale for the second.
sgraph	Scatter graph. Graphs one series against another.
lgraph	Graphs variables which are in logarithms, while displaying the y axis in natural units
vrange	Controls the vertical range of graphs (the y-axis scale).
hrange	Controls the horizontal range for scatter graphs or the right-vertical range for multi-scale graphs.
legend	Controls printing of legend on graphs.

The characteristics of each line on the graph are set by selecting Graph|Settings from the main menu.

Graph Titles

(ti)tle <title of graph or regression>

Provides a main title that appears on subsequents graphs. Use "ti" with no following text string to remove the title.

(subt)itle <text string>

Provides a subtitle on the subsequent graphs. Use "subt" with no following text string to remove the subtitle.

(vaxt)itle <text string>

Provides a vertically printed title for the y (left) axis. Use "vaxt" with no following text to remove the left axis title.

vaxl <in | out>

Causes the vertical axes numbers to be printed inside or outside the axes. The default is "vaxl in".

Displaying Graphs

Standard Line Graph:

(gr)aph <name1> [name2] [name3] ... [name6] <date1> <date2> [date3]

Graph the named series from date1 to date2 or date3, if present. If date3 is present, a vertical line (to separate history from forecast) will appear at date2. If dates have not changed since the last "graph" command, they need not be repeated. After a regression, the actual and predicted values may be plotted by "gr *". The actual data will be marked by squares; the predicted, by plusses, unless the marks have been changed by a previous "line" command.

Example:

```
title RTB -- The Treasury Bill Rate
gr rtb 70.1 85.1
```

Multi-scale Graph:

(mg)raph <name1> [name2] [name3] [name4] <date1> <date2> [date3]

Exactly like graph command except that the series has its own vertical scale chosen so that its graph extends from the top to the bottom of the screen. The scale shown on the left is for the first series; the scale shown on the right is for the second.

Semi-logarithmic Graph:

(lgr)aph <name1> [name2] [name3] [name4] <date1> <date2> [date3]

Exactly like the graph command except that the series are expected to be in logarithms. The vertical scale will be marked in the natural units, not the units of the logarithms. If

vertical range control is in effect (see below), the vertical ranges will be presumed to be in natural units.

Scatter Graph:

(sgr)aph <series 1> <series 2>

Displays a scatter diagram of series 1 and series 2. Graph is done with line 1 characteristics. Use the "0 width" option on the "line" command to plot only scatter points. Otherwise the points will be connected.

Style of Graph -- Line, Range, Legend Control

line <number> <color> <width> <style> <mark> <fill> <left> <right>

The easy way to use the "line" command is interactively. Use Graph | Settings from the main menu..

For use in command files, however, the following reference information is provided. The options are:

Number	the line number: 0 for the frame and annotation, 1-4 for the series graphed
Color	color of the line; a number from 1 to 63 for EGA and VGA monitors
Width	width of the line in pixels; 0 for no line (only marks or bars are printed). With a width above 1, all lines will be solid.
Style	line style: 0 for a solid line. 1 for a dashed line, 2 for a dotted line, 3 for a dashed line, 4 for dash dot, 5 for dash dot dot, 6 for none.
Mark	point markers: + for plus signs, x for x's, s for squares, d for diamonds, ^ for up pointing triangles, v for down pointing triangles, > for arrows in direction of line, b for bars, f for bars without a rectangle around them. (Use f with 0 fill for the "low" line in "high-low" graphs.)
fill	fill pattern for bars, 0 for Solid, 1 for Clear; 2 horizontal hatching; 3 for vertical hatching; 4 for forward diagonal; 5 for backward diagonal; 6 for cross hatching; and 7 for diagonal cross hatching.
left	
right	these decimal fractions (between 0 and 1) specify where the left and right edges of the bar fall in the available space for each observation. Parallel bars are drawn by assigning non-overlapping intervals to different series. Stacked bars or high-low graphs are produced with overlapping intervals.

Vertical Range Control

(vr)ange <bottom> [top]

vr <bottom> [line1] [line2] ... line[8] <top>

vr off

The graphing program normally sets the vertical range so the series extend from the bottom to the top of the graph. Sometimes, it is desirable to set the range independently. This is done with the "vrange" command. For example "vr 0 100" will make all subsequent graphs have 0 at the bottom and 100 at the top until the "vr off" command is encountered, restoring *G7* to its normal mode. If the <top> is omitted, then only the bottom of the graph is set and the program finds the top so that the graph fits on the screen.

If one or more of the optional [line] entries are present, horizontal lines will be marked at those levels. The lighted pixels which mark these lines are located above the long marks on the horizontal axis, so theyalso serve as meaningful vertical lines.

Legend Control

(le)gend <a> [b]

Controls printing a legend at the bottom of a graph.

a = y for yes, prints the legend (default)

a = n for no, do not print the legend

a = s leave space for legend but do not print. Useful for allowing the user to annotate the legend.

b = y for yes, mark the dates (default)

b = n for no, do not mark dates. Useful for making bar graphs of data occurring at arbitrary intervals.

Background color

The background color can be set interactively with Graph|Settings menu item or by the following command.

color <background>

```
Example: color 4
Sets colors for graphs. The numbers may range from 0 to 7.
```

Graphing Distributed Lags

For graphing distributed lags, it is helpful to specify the "dates" as coefficient numbers. Thus, to plot the coefficients a3 through a16 do "gr rcoef :3 16". The ":" is the signal to take the following numbers as regression coefficient numbers.

Setting Dates Without Actually Graphing

gdates <date1> <date2> [date3]

Sets the dates used by subsequent graph commands. With two dates provided, the series will be graphed from date1 to date2. If a third date is given, the series will be graphed from date1 to date3, with a vertical line drawn at date2.

gdates a

Selects "automatic" dates for graph commands. The automatic dates are the first and last date of the series actually present. The default setting in "look" is for automatic dates, unless specific dates have been previously specified. Automatic dates also adjust automatically to the frequency of the series. Not to be trusted when more than one series is being placed on the same graph.

Printing and Saving Graphs

gprint

This command prints the current graph to the printer. It is equivalent to picking Graph | Print from the main menu.

autoprint < y | n>

This command turns the autoprint flag on or off. It is off by default. When this flat is set on, then every graph that is displayed is also automatically printed.

gsave <filename>

This command is equivalent to picking Graph | Save from the main menu. It will create a Windows Metafile with the name given, and the .WMF extension. Do not supply the file extension on the command line.

Annotating Graphs

anntext <x-coordinate> <y-coordinate> <text to be inserted>

This command adds text to a graph. The *x* and *y* are values 0.000 to 1.000. The *x* values start from the left side, where .250 would be 1/4 of the way from the left. The *y* values start from the top of the graph, where .25 would be 1/4 of the way down from the top.

Example:

```
anntext 0.549 0.128 Hurricane Katrina
```

annline < start, x-coordinate><start, y-coordinate><end, x-coordinate><end, y-coordinate><symbol>

This command adds an arrow to a graph. *x* and *y* are values 0.000 to 1.000. The *x* values start from the left side, where .250 would be 1/4 of the way from the left. The *y* values start from the top of the graph, where .25 would be 1/4 of the way down from the top. *symbol* is a character added to the end of the line. Typically, the symbol is the ">" character that forms an arrow point.

Example:

```
annline  0.260  0.145  0.63  0.79 >
```

Ordinary Regression

Here are some commands related to the running of an ordinary regression:

(ti)tle <the title you want for your regression or graph>

Supplies the title for the display of your regression results.

(lim)its <date1> <date2> [date3]

Specifies the starting date for the fit, date1, the ending date for the fit, date2, and the ending date for the test or forecast period, date3. If date3 is omitted, it is assumed to be the same as date2.

mode <test | forecast> or mode <t | f>

Determines what is to be done between date2 and date3. Default mode is test. In that case, the values "predicted" by the equation for the period from date2 to date3 are compared with the actual data and a SEE and MAPE computed for the test period. For forecast mode to work, the series for all independent variables except lagged values of the independent variable must extend to date3. Then, in forecast mode, two forecasts are made for the period from date2 to date3. The first is simply the predicted value; the second, the predicted value plus the "rho adjustment" to take account of the error in the last period of fit.

cmtype <yes | no>

If "yes", the simple correlation matrix and the standard deviation of each variable will be displayed in the results window before the regression results. After the regression, the variance-covariance matrix of the regression coefficients is displayed, as is the matrix of derivatives of the regression coefficients with respect to one another. For each coefficient, this matrix shows how all the other coefficients would change if the given coefficient where forcibly changed by a "constraint" ("con") command. The default value for "cmtype" is 'no'.

See also the Regression Tests topic.

OLS Regression Command

r <y> = <x1>, <x2>, <x3>, ..., <xn>

r <y> = ! <x1>, <x2>, <x3>, ..., <xn>

Both forms regress the dependent variable y on the independent variables x1, ..., xn. The first form automatically supplies a constant term, the second does not. The total number of variables, counting the constant, must not exceed 30. Independent variables may be expressions, as on the right side of an "f" command. Dependent variable should be a single variable. If the line ends with a ',' the command continues on the next line.

If lagged values of the dependent variable occur among the explanatory variables, the are automatically supplied from previously calculated values when forecasting. Regression coefficients and other values are stored in the *rcoef* variable in the workspace bank.

Use "gr *" to graph results. When you have the graph of the regression and want to look back at the regression results, just move to the regression results window by navigating through the open windows of *G7*.

Use "gr rcoef :3 8 " to graph regression coefficients 3 through 8. This is especially useful with distributed lags.

Use "gr resid" to graph the residuals. Use "gr lever" to graph the "leverage" series, which shows the derivative of the predicted value of each observation with respect to the observed value of that observation. It is useful for spotting outlying observations.

Terms Appearing on the Regression Display

Characteristics of the entire regression. These appear at the top.

SEE	Standard error of estimate, or the square root of the average of the squares of the misses or "residuals of the equation.
RSQ	(pronounced r-square) = Coefficient of multiple determination.
RHO	(Greek rho) = Autocorrelation coefficient of the residuals.
Obser	Number of observations.
SEE+1	The SEE for forecasts one period ahead using rho adjustment.

RBSQ	Coefficient of multiple determination adjusted for degrees of freedom.
DW	Durbin-Watson statistic; contains same information as does RHO.
DH	Durbin H statistic; replaces DW if a lagged value of the dependent variable is present.
DoFree	(Degrees of freedom) =
Obser -	number of independent variables.from to the dates covered by the regression.
MAPE	Mean absolute percentage error.
JarqBer	A test of normality due to Jarque and Bera.

Values relating to individual variables. In the Regression | Settings you may specify which of these are to be displayed. The default display is:

Name	Code name of the variable.
Reg-coef	Regression coefficient for this variable.
Mexval	(Marginal explanatory value) The percentage increase in SEE if this variable is left out of the regression. A factual alternative to the t statistic.
T-value	Student t values.
Elas	Elasticity of the dependant variable with respect to this variable, evaluated at the means of both.
NorRes	Normalized residuals. The ratio of the sum of squares residuals after the introduction of this variable to the sum of squared residuals after all variables have been introduced. A factual alternative to F statistics.
Mean	The mean value of this variable.
Beta	What the regression coefficient would be if both the independent and dependent variables were scaled so that they had unit standard deviations.
Fstat	The Fisher F statistic for testing the significance of this and the following independent variables.

Additionally, if you give the command “cmtype y”, *G7* will display the correlation matrix of the original variables and their standard deviations, the matrix of variances and covariance of the regression coefficients, and a matrix of the derivatives of one regression coefficient with respect to another. The derivatives with respect to variable 3, for example, show the rate of change of each other coefficient if the coefficient on variable 3 is changed by a constraint. They show how the estimate of one coefficient depends on the estimate of another. You can turn the display of the T-statistic on or off with "showt y" or "showt n".

Recursive OLS Regression

limit date1 date2 date3

recur <y> = <x1>, <x2>, <x3>, ..., <xn>

recur <y> = ! <x1>, <x2>, <x3>, ..., <xn>

The indicated regression will be done from date2 to date3, then from date2-1(the period before date2) to date3, then from date2-2 to date3, and so on back to date1 to date3. The regression coefficients are made into time series and stored in the workspace. b1 is the series for the first regression coefficient; b2, for the second; and so on. The regression coefficients are stored in the date corresponding to the first date in their regression. Thus, b1{date1} would be b1 for the regression over the period date1 to date3. Similarly, the standard errors of the regression

coefficients are stored in s1, s2, s3, etc.

If a "zip" command preceeds the "recur" command, no regressions will be displayed, but the zip command will be turned off at the end of the "recur" command, for you will surely want to do graphs after the "recur" and they cannot be done with zip on.

Example:

```
zip
limit 80.1 85.1 91.2
recur gnp$ = g$,v$,fe$,fi$
gdates 80.1 85.1
fadd bounds (1 - 5)
```

where "bounds" is the file

```
ti Estimate and 2-Sigma Limits for Coefficient %1
f upper = b%1 + 2*s%1
f lower = b%1 - 2*s%1
gr b%1 upper lower
```

See also the following topics:

Distributed lags

Soft Constraints

Seemingly Unrelated Regression (SUR)

Homogeneity and normality tests

Non-linear regression

Soft Constraints

Soft constraints are a way of combining *a priori* theoretical information or opinions about the desired value of estimated parameters with what the data suggests the value of those parameters should be. The commands available for applying soft constraints are the "con" command, which applies a constraint directly to a single parameter or to a linear function of a number of parameters; and the "sma" command, which softly constrains a number of distributed lag weights to lie close to a certain degree of polynomial. The "con" command is explained below. See the distributed lags topic for the "sma" command.

con <top> <c> = <linear function of ai's>

Examples:

```
con 100 1 = a3 + a4 + a5
con 200 0 = a7 - 3a6 + 3a7 - a8
```

This command imposes "softly" the given constraint on the regression. The "top" (trade-off paramater) determines how "soft" the constraint with the convention, "the higher, the harder." Specifically, the constraint counts as top*T observations, where T is the number of regular observations in the regression. Soft constraints are also know as "Theil's mixed estimation",as "stochastic constraints", and are also a particular type of Bayesian regression.

Ridge Regression

This technique amounts precisely to imposing a soft constraint that one or more of the coefficients should each be zero. For example,

```
con 100 0 = a2
con 100 0 = a3
```

will put a2 and a3 on a 'ridge'. Such constraints may increase the t statistic for a variable, but they almost certainly distort the regression coefficients.

Quadratic Programming Regression

G7 also can estimate regression parameters subject to "hard" constraints. These may be combined with soft constraints.

Suppose matrix *A* specifies linear combinations of the regression parameters β that is less than or equal to vector *b*. Then we have the conditions that

$A \times \beta \leq b$

$\beta \geq 0$

Constraints *A* and *b* are added with the *qpcon* command, and the parameter vector β is estimated with the *r* command. Soft constraints also may be added with the *con* and *sma* commands. These constraints are imposed on the next linear regression, which is specified by the *r* command. If no constraints are imposed with the *qpcon* command, then *G7* finds parameters using OLS. If constraints are imposed with *qpcon*, then G7 finds parameters using a quadratic programming algorithm.

The syntax for the *qpcon* command is as follows:

qpcon b <sign> [constant][*]ai [< > [constant][*] aj …]

where *sign* is <, =, or > (<= and >= also are accepted but are recorded as strict inequality constraints). *constant* are optional scalar multiples for the respective parameters. An asterisk may be included to clarify that the parameters are multiplied by the corresponding constant. The parameters are denoted by *ai*, where *i* is the position in the regression equation of the corresponding variable. If a constant is included in the regression, then the parameters are denoted as *a1*, *a2*,… Right-hand-side terms are separated by either a + or -.

Examples:

```
qpcon 25.0 > a1          # constrain the constant
qpcon 13.0 = 1*a2 + 1*a3 # constrain the sum of slope parameters
```

Distributed Lags

To easily estimate a polynomial distributed lag, in which the lag coefficients are softly constrained to lie along a certain degree of polynomial, use the "sma" command, described here.

sma <top> <first> <last> <order> [f]

Examples:

```
sma 100 a4 a9 3 f
sma 100 a8 a16 1
```

The "sma" command imposes a series of constraints which "softly" require the regression coefficients between 'first' and 'last' to lie on a polynomial of degree 'order'. If the 'f' is present at the end, the polynomial is "free" at the end; without the f, the polynomial will be assumed to be zero for the coefficient after 'last'.

Tests of Homogeneity

chow <n>

where <n> is the number of regressions involved in the test.

A "Chow" test is a special case of the Fisher F test used to test the homogeneity of a sample. Typically, one runs two or more regressions covering parts of a sample and compares the sum of squared errors with that of a single regression covering the whole sample. If we were going to break up the sample into two parts, the command would be

```
chow 3
```

Then we run first the two regressions over the separate halves of the sample, and finally the single regression over the whole sample. After this last regression, one will be greeted by the results of the Chow test.

There is a special case in which the second "half" of the sample is too small to run the regression. In that case, give the command

```
chow 2
```

and then run the regression over the reduced sample and then again over the whole sample. After the second regression, the Chow tests appear.

Test of Normality of Errors

(norm)ality <y | n | f>

This command turns on (or off) tests of normality of the error terms. If the option chosen is 'y', then the Jarque-Bera test appears on the standard regression result screen labeled "JarqBer". Under the hypothesis of normality, it has a chi-square distribution with two degrees of freedom.

If the option 'f' (for full) is chosen, then before the regression results are shown, a table appears with moments and measures of symmetry and peakedness, as well as the Jarque-Bera statistics. If a "catch" file is active, this table will go into it.

Editing

Although you may use any ASCII text editor to edit the add files, fadd files, etc. which you may need to create for use with *G7*, *G7* comes with its own text editor, which works much like Windows Notepad. (Notepad itself is available under the File item of the main menu.)

To start the *G7* editor, choose File | Editor. The editor will appear in the upper right corner of the screen. To edit an existing file, choose Open ..., and then pick a file from the dialog box. When you have finished working on the file, use Save or Save as from the File menu. There is no need to close the editor before using the file. However, if you wish to close the editor window, you may do so by a click in the upper right corner or with 'Ctrl-F4'.

By clicking Editor on the *G7* main menu, you can open a new editor window while leaving the existing one open. The new one, however, will be in the same upper right corner; to see them both, you must drag the top one to a new position.

Either clicking the Find menu item on the Editor main window or tapping 'Ctrl-F' will bring up a standard Find window for searching for a string.

Clicking Save on the Editor's menu saves the current contents of the editor. "Save as ..." is on the File menu, as are commands for changing the font and background color. Although you can choose special fonts and colors, none of that formatting is saved when you click the Save button, only the ASCII text. This behavior is because the formatting information would stop the files from working correctly as input to *G7*.

Clicking Run on the Editor's menu saves the current contents and executes the file as a *G7* command file.

In *G* 6.4 for DOS, you can use many DOS editors from within *G*. Simply copy the .EXE file of your favorite editor into the file ED.EXE. The freely available editors *SEE*, *FTE* and *TED* are made available in the *G* 6.4 distribution.

Hildreth-Lu

hl <rho1> <rho2> <incr> <y> = <x1>, [x2,] [x3,] ...[xn]

In the Hildreth-Lu procedure, a guess of the autocorrelation coefficient of the errors, rho, is chosen, multiplied by the equation lagged once and subtracted from the unlagged equation. The resulting equation is then estimated by ordinary regression. Another value of the guess of rho is then chosen and the process repeated. In this command, rho1 is the starting guess of rho, incr is the amount by which it is incremented on each iteration and rho2 is an upper limit on the guess. The y and x1, ..., xn are as in the r command. At the end of the process, you will get a table with this heading:

```
RHO-HL     SEE 1 AHEAD  RHO-EST     SEE LONG-RUN
```

The RHO-HL shows the assumed rho, the SEE 1 AHEAD shows the standard error of estimate (SEE) of the estimated equation, RHO-EST shows the rho of the estimated equation, and SEE LONG-RUN shows the standard error of using the fitted equation on the original, undifferenced data, without a knowledge of the true lagged value of the dependent variable, as

must be done in forecasts of more than a few periods ahead. If the "save" command is on, all of the estimated equations will placed in the ".sav" file as undifferenced equations suitable for going into a model. You must choose which one you want.

For graphing the HL output, one should run the equation one last time with the chosen value of rho as both the starting and ending rho. Then also run the equation with just the "r" command (no autocorrelation) Then do two graphs:

```
gr depvar predp1 hlshort
gr depvar predic hllong
```

The first will compare the actual with two one-period ahead forecasts, the one from ordinary least squares (+'s) and one from Hildreth-Lu (x's). The second compares the actual with two forecasts not relying on the lagged value of the dependent variable. Again, the uncorrected least squares fit is shown by +'s and the fit using the H-L correction is shown by x's.

The pauses at the end of each fit in the HL procedure can be eliminated by the "zip" command. If you use zip, however, you must remember to do "zip off" before using the "graph" command.

G7 does not provide the Corchrane-Orcutt correction, since Hildreth-Lu is better.

ARIMA

ac <series> [n]

```
Example: ac vin$ 11
```

Calculate the autocorrelation function for the named series over the period specified by the latest "limits" command. The optional [n] is the maximum number of terms to be calculated; its default value is 20. The output shows the autocorrelation function and the autoregression coefficients determined by the Yule-Walker equations. Taking the rightmost elements of each line of the autoregression coefficients gives the partial autocorrelation function. The function is also placed into the workspace with a name derived by adding "_ac" to the variable's name. This variable can then be used to graph the function. If the command was "ac vi 20", then the graphing command would be "gr vi_ac :0 20".

bj <q> <y> = <x1>, [x2,] ..., [xn]

Example:

```
bj 2 vif$ = vif$[1], vif$[2]
```

Performs a Box-Jenkins estimation for autoregressive moving average models. Here q is the number of lags in the moving average error terms, e.g., with q = 2, u[t] = e[t] + b1e[t-1] + b2e[t-2]. This q must be no more than 4. The program will only check for the stability of the solution if q <= 2. No more than 10 independent variables, please. At each iteration of the

"Newtonian" non-linear regression algorithm, you will be allowed to intervene to try new values for the theta's, the coefficients for the moving average of error terms. If "save" is on, the first and the final equation will be in the .SAV file with the theta values appearing as coeffients of "theta". These must be removed before building with *Build*.

SUR

The format for the "sur" command is shown by this example:

```
sur
r y1 = x11, x12, ...
...
r yn = xn1, xn2, ...
con 10 1 = a2 + 4b5 + etc
sma 100 a6 a12 1
do
```

After the "sur" command come all the individual regressions. Any constraint or "sma" commands must come after the regressions. In the "con" and "sma" lines, a's denote coefficients in the first regression; b's coefficients in the second, and so on. Thus, a2 denotes the second coefficient of the first regression, and b5 denotes the fifth coefficient of the second regression.

The predicted values, dependent variables, and regression coefficients of the successive equations appear in the workspace file as predic1,depvar1, rcoef1, predic2, depvar2, rcoef2, etc. The results of the individual equations can be plotted by "gr *1", "gr *2", etc. The graph commands should follow the "do" command and each should be preceeded by an appropriate "title" command.

stack

The format for "stack" is exactly like that for "sur" except that "stack" replaces "sur". With "stack", no attention is paid to contemporaneous covariances. The point of "stack" is solely to impose soft constraints across regressions.

limitation: *G7*'s limitation of 30 variables, counting dependent, intercepts,and independent variables, can quickly become binding in sur and stack, since it applies to the total variables in all equations involved in the "sur" or "stack".

Miscellaneous

Recovery from catastrophe
If by some misfortune you are thrown out of *G7* against your will, you can continue right from where you were. Start *G7* again and recover the workspace of the previous run of *G7* by typing "wsb ws". If you then do a "lis w" you will see that you still have all the variables you had created before *G7* died. Repeat any "limits" or "mode" commands you have used.

(q)uit

Quit, that is, close up shop in *G7*. This is equivalent to picking File, Exit from the menu.

time

Display the date and time.

dos

dos [options] <arguments>

dos [options] [batch file arguments] { … }

The dos command passes commands to the operating system. Three versions are offered, along with an important option.

To open a DOS window, simply enter dos with no arguments. G7 pauses until the window is closed. The window may be closed by clicking the button in the upper right-hand corner or by typing "exit".

To pass single commands to the system, type the desired system commands after dos. For example, to save the workspace bank as "newws", enter the following:

```
dos copy ws.* newws.*
```

To pass a group of commands to the operating system, use the dos{} command. The brackets tell G7 to create a temporary batch file containing the system commands between the brackets. The temporary file is destroyed automatically upon completion. Arguments may be passed to the batch file by listing them after dos and before the open bracket. Here is an example that again copies the workspace.

```
dos ws newws{
   rem Copying bank %%1 to %%2.
   copy %%1.ind %%2.ind
   copy %%1.bnk %%2.bnk
   }
```

Note that the syntax required for text within the dos{} command is the same syntax required elsewhere in G7. In particular, in order to pass a '%' character to the system, "%%" must be specified in the G7 script as is shown in the example above.

When the dos command is given with arguments (i.e. as in the second and third cases listed above), the output is captured that otherwise would appear in the DOS window. When the command window closes, this output is displayed in the G7 window. No output is displayed on the DOS window. While this often is acceptable and desirable, it is not suitable for running programs or commands that require user input. For this reason, an option is given to execute the dos command in interactive mode. Specify the option by entering "-i" (for interactive) immediately after dos. For example,

```
dos –i idbuild master
```

opens a command window and executes the program IdBuild with "master" as a command-line argument. This command is displayed in the DOS window, together with output from IdBuild. If IdBuild requires user input (for example if no configuration file is available), the user will see the prompt in the DOS window and can enter a response. The same option is available in batch mode.

```
dos -i ws newws{
   rem Copying bank %1 to %2.  Type <ctrl> c to cancel.
   pause
   copy %1.ind %2.ind
   copy %1.bnk %2.bnk
   pause
   }
```

Multiple commands may be entered on a single line. The commands must be separated by a semicolon and enclosed with brackets. This capability works when there is only one line of G7 input; everything, including the brackets, must be entered on the same line. In particular, this is useful for entering multiple DOS commands in the G7 command box. The following command

opens a DOS window, displays the contents of the directory one page at a time, and pauses at the end:

```
dos –i {dir /p; pause}
```

Note that the command interpreter also may offer the ability to enter more than one command per line. In Windows 2000, multiple commands may be separated with an ampersand (&). The above command may be entered as

```
dos –i dir /p & pause
```

findmode < m|a>

This command controls how series are searched for when there is more than one databank assigned. Normally, the mode is set to 'm', for manual. By default, the bank at position 'a' and the default Vam file are searched. Series in any other banks must be searched for by prefixing the series name with the bank letter and a period. For example, if there are 3 banks assigned, 'a', 'b' and 'c', and a series called "gnp" is in bank 'c', then you must specify "c.gnp". If findmode is set to 'a' (automatic), then G7 will search sequentially through each bank in turn, until it finds a series named "gdp".

vammode <s|a>

The Vam mode may be set to either 's' (simple) or 'a' (advanced). In the simple mode, when a Vam file is assigned, only that file is assigned. The 'a' option is used by builders and users of *Interdyme* models, for which there is a G bank containing the macrovariables in the model. Each time a new simulation is made, both a Vam file and a G bank are created with the same root name. When vammode is 'a' and a Vam file is assigned, G7 looks to see if there is a G bank with the same name. If the G bank exists, then G7 loads both banks as a single unit. For example, if there is a Vam file named HIOIL.VAM, with associated macro bank HIOIL.BNK, the following code will assign the two files as a unit:

```
vammode a
vam hioil a   # Load both hioil.bnk and hioil.vam
```

If the series "gnp" is in the G bank, you now can view the macro series *gnp* with "ty a.gnp". If the Vam bank contains a vector *output*, then the vector can be shown with “show a.output”. Note that both commands refer to bank *a*. The user alternatively might want to load the banks sepparately. The following code will load the banks separately.

```
vammode s
vam  hioil a   # Load hioil.vam
bank hioil b   # Load hioil.bnk
```

Non-linear Estimation

G7 offers two algorithms for non-linear regression. The "nl" command uses the simplex method (sketched below and not to be confused with the simplex method of linear programming) and the Powell method. The form for the two is exactly the same except that the first is invoked by the "nl" command, while the second is invoked by the "nlp" command. We illustrate the with "nl" here.

nl <y> = <non-linear function involving n parameters, a0, a1, ...an-1>
n
<starting values of the parameters>
<initial variations>

Example:

```
nl y = @exp(a0*@log(a1 + a2*x))
3
1.5  25  2.0
.1   1   .05
```

Since the non-linear function must be evaluated repeatedly, anything the user can do to speed the evaluation is helpful. For example, put all the variables involved into the workspace; make the workspace no larger than necessary; put the workspace on a ram disk; take any functions of variables, such as logs or squares, which do not change through the calculations and put the results in the workspace and use them instead of, say, taking the log over and over.

In the simplex algorithm, S, the sum of squared errors, is initially calculated at the initial value of each parameter. Then, one-by-one, the parameters are changed by adding the initial variations and S is recalculated at each point, thus yielding values at n+1 points (a simplex). By the algorithm sketched below, points in the simplex are replaced by better points or the simplex is shrunk towards its best point. The process continues until no point differs from the best point by more than one-tenth of the initial variation in any parameter. Each time a replacement is done, the simplex and the operation that yield it is reported on screen. Like all non-linear algorithms, this one is not guaranteed to work on all problems. It has, however, certain advantages. It is easily understood, no derivatives are required, the programming is easy, the process never forgets the best point it has found so far, and the process either converges or goes on improving forever. The display show the regression coefficients, their standard errors, t-values, and variance-covariance matrix. The Powell method also requires no derivatives and has the property of "quadratic convergence." That is, applied to a quadratic form, it will find the minimum in a finite number of steps.

Following the "nl" command, there can be f commands, r commands, and con commands before the non-linear equation itself. These may contain the parameters a1, a2, etc.; they should each be terminated by ';'. The non-linear search then includes the execution of these lines. E.g.:

```
nl f x1 = @cum(s,v,a0);
y = a1 + a2*x1
```

If the name of the left side variable is "zero", no squaring of the difference between the left and right side will be done. Instead, the squaring must be done by the @sq() function. This feature allows soft constraints to be built into the objective function. For example,

```
nl zero = @sq(y - ( a0 + a1*x1 + a2*x2)) + 100*@sq(@pos(-a2))
```

will "softly" require a2 to be positive in the otherwise linear regression of y on x1 and x2.

If the name of the left-side variable is "last", no summing will be performed. Instead, the value of the function on the right at the "last" period will be minimized. This "last" period is the one specified by the second date on the preceding "limits" command. In order to combine summing of some components of the function on the right with the use of the "last" option, the @sum() function is used. It places the sum of the elements from the first date to the second date in the second date position of the output series, which is otherwise zero. This device is useful in some maximum liklihood calculations.

Both the simplex method and Powell's method are described in the book,*Numerical Recipes in C*, by Wm. H. Press, et al., Cambridge University Press. The Simplex Method is sketched below.

Sketch of Simplex Method of Non-linear Regression

The method begins with a point in the parameter space of, say, *n* dimensions and step sizes for each parameter. Initially, new points are found by taking one step in each direction. Actually, each step is made both forwards and backwards, and the better of the two positions is chosen. This initial operation gives *n+1* points, the initial point and n others. These *n+1* points in *n* dimensions are known as a simplex. The algorithm then goes as follows.

Reflect the old worst point, W, through mid-point, M, of the other points to R (reflected). (R is on the straigth line from W to M, the same distance as W from M, but on the other side.)

```
If R is better than old best, B {
 expand to E by taking another step of length MR in the same direction.
 if E is better than R,
      replace W by E in the simplex.
 else
      replace W by R. (reflected)
 }
Else if R is better than W {
 replace W by R in the simplex. (reflected)
 }
Else {
 contract W half way to mid-point of other points, to C. (contracted)
 if C is better than W,
      replace W by C.
 Else
      Shrink all points except B half way towards B. (shrunk)
```

Two-stage Least Squares

G7 needs and has no special command for ordinary two-stage least squares. Consider the following system of equations:

```
y1 = f(y2, x1, x3)
y2 = g(y1, x2, x3)
```

To obtain the 2SLS estimates, first do:

```
r y1 = x1, x2, x3
rename pred z1
r y2 = x1, x2, x3
rename pred z2
```

The z's are now the second stage regressors, so we do:

```
r y1 = z2, x1, x3
r y2 = z1, x2, x3
```

or, if we are building a model and want to have the correct names on the independent variables in the second stage regression, we can replace the last two lines by

```
rename z1 y1
rename z2 y2
r y1 = y2, x1, x3
r y2 = y1, x2, x3
```

Systemic Two-Stage Least Squares

The bane of two-stage least squares has been that there are usually so many exogenous variables and lagged values of endogenous variables in the system that the first-stage regressions fit so closely that there is no substantial difference between the OLS and the 2SLS estimates, especially if nonlinearities in the model are allowed for by including powers and cross products of the exogenous variables. *G7* and *Build* offer a new way to deal with this problem. The procedure is simple. Estimate the equations of the model by OLS or other single-equation method. "Build" the model with these equations and simulate it over the historical period. The simulated values of the endogenous variables are not influenced by the errors in the structural equations. They may therefore be used as regressors in the second stage to obtain estimates free of simultaneous equation bias. There are two variants of this procedure. In the first, the simulation is done using the simulated values for lagged values of endogenous variables; the other uses the actual values of these variables.

To perform the first variant, perform the simulation without any special command, copy the bank of simulation results to the *G7* workspace, start *G7*, select the b option on the opening menu, do "bank bws" to put the actual values in the assigned bank, give the command "second on" so that "f" commands are ignored, and re-estimate the equations with the same add files as originally used. With "second" on, the dependent variable of the regression will be taken from the assigned bank; all others come from the workspace, which contains the simulated values. To turn off the "second" feature, the command is "second off".

For the second variant, give the command "first" to the Run program before giving it the "run" command. Run will now store the simulated values in the workspace as before, but it will use actual values of all lagged variables. You may now proceed as before to start *G7* and give the "second" command. You will then be using simulated values of all independent variables. Alternatively, you can choose to use actual values of all lagged variables. To do so, proceed as before, but place a "a." in front of the name of every lagged endogenous variable in an "r" command. The "a." in front of a variable name causes *G7* to look for the variable only in the assigned bank, where it will find the actual value.

The command "second off" will restore *G7* to its normal condition in which it processes "f" commands.

Three Stage Least Squares

G7 performs three stage least squares by combining the two stage procedure explained above with the seemingly unrelated regression procedure described in the help topic for SUR. Consider again the system of equations:

y1 = f(y2, x1, x3)

y2 = g(y1, x2, x3)

Obtain the 2SLS instrumental variables by:

```
r y1 = x1, x2, x3
rename pred z1
r y2 = x1, x2, x3
rename pred z2
```

Insert the second stage regressors(z's) into the system and estimate with SUR:

```
title Two-stage and Three-stage Least Squares Estimates
sur
     y1 = z2, x1, x3
     y2 = z1, x2, x3
do
```

The first set of regression results are the 2SLS estimates, the second set are the 3SLS estimates. To graph the results use the same statements as in SUR.

Model Building

This help topic discusses some of the *G7* commands useful for model building. See *The Craft of Economic Modeling* by Clopper Almon for more on the use of *G7* and *Build* to build macroeconomic models.

save <filename>

Opens the named file which then receives all “title”, “f”, “fex”, “id”, and “cc” commands exactly as they appear on the screen. It also catches the output of “type” commands but with the "type" changed to "update". Most important, the file receives the results of regressions in the form of equations with the variable names and values of the regression coefficients. Use "save off" to close the saved file and stop saving. By convention, not necessity, all "save" files are given the extension ".SAV".

“f”, “fex”, and “fdup” all do the same thing in *G7*, but in *Build*:

f computes the left_side variable and puts it and all the right_side variables into the workspace and puts an identity into the model.

fex computes the left_left side variable and puts it into the work_space. It does NOT put the right_side variables into the workspace and does NOT put an identity into the model. It is used to define historical values of EXogenous variables, such as tax rates, and historical values of dependent variables of behavioral equations.

fdup has no effect whatsoever in *Build*.

id <identity>
This has no effect in *G7* but, when passed to *Build* puts all variables (without computing the left_side one) into the workspace and puts an identity into the model.

check <variable name> <tolerance>
This command can only be placed in a "master" file for input into *Build*. It causes the named variable to be checked for convergence to within the specified tolerance. The values of the checked variables will be printed on the screen as the model runs. It is generally necessary to check only a few of the variables in a model.

cc <C language statement>
This has no effect in *G7* or *Build*, but the C_language statement is passed through to the file heart.c, which contains the model written in the C language and ready for compilation. Such passed_through statements may be used to do special tasks for which *G7* and *Build* have no convenient mechanism __ such as scaling one group of variables to sum to another variable. For many models, no cc statements are needed.

sty <variable name>

Silent type. Prints the variable to the file being saved; do not show it on the screen.

Matrix operations in models

The following matrix commands can only be placed in a MASTER file for input into *Build*. When they are used, the value of DIMMAX in COMMON.INC must be set equal to the largest dimension of the largest matrix.

vec <y> = <n1>,<n2>,<x1>,<x2>, ..., <xn>

vec <y> = <n1>,<n2>

Defines the vector y with n1 rows and n2 columns. The two dimensional specification is used to distinguish between row vectors and column vectors. In the example, fd is a column vector and nulv is a row vector. The vec command forms the vector y by putting the variables x1,x2,...,xn into the corresponding components of y. When the variables x1,x2,...,xn are not specified, the vec command defines a zero vector.

Example:

```
vec fd = 2,1,auto,food
vec nulv = 1,4
```

mat <y> = <n1>,<n2>,<filename>

mat <y> = <n1>,<n2>

Defines the matrix y with n1 rows and n2 columns dimension n2. The entries of y are fetched from the given file row by row. When the file name is not specified, the mat command defines a zero matrix.

Example:

```
iomat = 78,78,io77.usa
cmat = 78,34
```

mop <y> = <x1> + <x2>

mop <y> = <x1> _ <x2>

mop <y> = <x1> * <x2>

mop <y> = <x1>' transposition

mop <y> = <x1> ^ <x2> vector component_wise multiplication

Performs matrix operations. When addition, subtraction, multiplication, or transpose is done, the resulting matrix is stored in matrix y. With the ^ operation, the ith component of the vector y will be the product of the ith components of vectors x1 and x2. WARNING! All matrices and vectors should be defined by mat command and vec command before using "mop" command.

Common Coefficient, Panel Data, or Pooled Regressions

It fairly frequently happens that we have similar data sets collected at different times or in different places. We may then wish to estimate regressions which combine the data sets in various ways. The "comcoef" command in *G7* is designed for this purpose. It allows one to specify a number "common coeficients" in the regressions which follow. The method may be best explained with an example. In the following example, "cps78" and "cps85" are data banks of from the Current Population Surveys of 1978 and 1985, respectively. The first has data on 550 individuals, the second on 534 individuals. We want to regress the logarithm of an individual's wage, lnwage, on the individuals education, ed. We want the coefficient of lnwage to be the same in both samples, but the interecepts may be different in the two years. Here is what we do.

```
f one = 1
comcoef 1 4
hbk cps78
lim 1 550
r lnwage = ! ed, one
hbk cps85
lim 1 534
r lnwage = ! ed, one
do
```

In the "comcoef" command, the "1" tells the program to use the same coefficient for the first 1 variable(s) in all the following regressions. The "4" is the total number of variables, counting the dependent variable, in the combined or "pooled" regression. Once the "comcoef" command has been received, *G7* accumulates regressions until the "do" command is reached.

Only a limited list of commands are a permitted between the "comcoef" command and the "do" command. They are the "bank", "hbk", "lim", "f", "add", "pause", "quit", and "r" commands. The "quit" command only quits "comcoef", not *G7*.

The "comcoef" command can be used under the chow command and under the catch command. It also puts the estimated coefficients into the rcoef series in the workspace bank, as usual. It does not enter a series of predicted values into the workspace bank, so "gr *" will not work following a comcoef regression. Nor are the variable names shown on the display of the results, for in general the variables with a common coefficient may have different names while variables with different coefficients may have the same name, such as "one" in the above example.

The Workspace Bank

At any one time, *G7* has one workspace bank, and 0 to 25 assigned banks. The workspace bank is the same physical bank throughout any one run of *G7*, unless changed using the "wsbank" command explained below. Whenever *G7* needs a series, it looks first in the workspace and then, only if it doesn't find the variable there, it looks through the assigned banks. The original assigned bank, at position ‘a’ is specified in the G.CFG file in the default directory. As *G7* forms variables with "f" commands or introduces variables with "data" commands, the

newly created or introduced series are put into the workspace bank. Even the "update" and "mupdate" commands do not change the series in the assigned bank but put the revised series in the workspace.

listnames <ws | assigned> [wildcard]
lis [-srgv] <w | a> [wildcard]
lnc [-srgv] <w | a> [wildcard]

Types the names of the variables in the workspace (w) or assigned bank (a). If the *save* command is on, the list will then appear both on the screen and in the saved file. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS. The lnc routine prints the series as a single column. Option *s* sorts the series in alphabetical order, and *r* reverses the order. If a Vam file is associated with the *G7* bank, then option *g* prints only macro series and option *v* prints only Vam bank series.

btitle <ws | assigned>
bti <w | a>

Displays the title and configuration of the workspace (w) or assigned bank (a). In the case of the workspace, you can change the name of the bank by providing the new name after the "bti w" command. If you have a bank with no title, one can be supplied via cautious use of the "wsbank"command explained below.

wsbank <bank_name>
wsb

Makes the named bank the workspace. The bank name is expressed as in the "bank" command, above. Beware: the workspace now becomes the named bank and this bank will be changed by almost anything you do. In particular, a "zap" (see below) will totally destroy the bank. Use this command with utmost caution. It is best to backup a bank before using it with this command.

zap

Sets the number of series in the workspace to zero.

del <series name>

Delete the named series from the workspace data bank. The last series is moved to the place of the deleted series. The "ws.bnk" file is not physically reduced in size by a deletion; but if another variable is added to it after the deletion, it will be put in the space formerly occupied by the deleted variable.

rename <old_name> <new_name>
ren

Changes the name of a series in the workspace bank.

bupdate <x> = <y>
bu

"bupdate" stands for bank update. This command takes the series x from the workspace, updates it with non_zero entries from series y in the assigned bank, and stores it as x in the workspace. If x is not previously in workspace, "bup" will put it there with values from the assigned bank. y may be an expression. For updating or combining entire data banks using this command, use the procedure described below.

Making Data Banks: An Example

You want to make a G bank, to be called "PRICES" with data from a file called PRICES.DAT, which contains data in the form ready for use by *G7*. Each series has one observation per year. Data begin in 1965, and you want to save room for data or forecasts through 2010 (46 years). First, edit the g.cfg file to have the workspace begin in 1965 and the number of observations to 46. Exit from *G7* and start it again. When *G7* starts, do "add prices.dat". Then do "dos copy ws.* prices.*" The Prices bank should now be ready to assign.

Compressing a Bank

Before returning to *G7*, you may wish to compress the bank with the *Cpress* program. The command is just:

cpress <bank_name>

In our example, the command would be

```
cpress price
```

and the result would be the two files price.cbk and price.cin. Generally, they will take up much less space than the original price.bnk and price.ind, which can now be deleted.

In this form the bank would be assigned in *G7* by the command

```
cbk price
```

To create a hashed bank, rather than a compressed bank, use *HPress*.

Updating a Standard Data Bank

The easiest way to update is with the Splice programs described in the help file on assigned banks. An alternative, older route is described here for updating standard banks, while the "Splice" programs apply to compressed banks or hashed banks. In the method described here,

the new data must not extend beyond the number of observations specified when the databank was created. In this alternative way, to update data in the PRICE bank, use the following procedure.

1. Put the new price data with appropriate "update" commands (**not** "data" commands) into a file (example: HOT.DAT.)
2. Use the wsb command to make the databank files the *G7* workspace. Example:
   ```
   wsb  price
   ```
3. Type
   ```
   add hot.dat
   ```
   When the add file ends, check that series have been updated.
4. Use the "wsb" command to move back to the usual workspace:
   ```
   wsb ws
   ```

Updating A Standard Bank From Another Bank

The "bupdate" command updates one standard bank from another. Suppose that we have the existing "price" bank and a short bank of recent information in "newprice". Outside *G7*, run the *bups* program. *bups* asks for the name of the databank and the name of the output file that will contain the "bup" commands. The lines of this output file will be of the form:

bup variable_name = variable_name

Example:

```
bup gnp = gnp
```

There will be one line for each variable in the bank. Let us suppose that you choose to call the output file PRICE.BUP.

In *G7*, do

```
wsb price
bank newprice
add price.bup
wsb ws
```

The series in the "price" bank will now have all the data the "newprice" bank, plus any data they previously contained that was not updated.

Working with Vam Files in G7

G7 is part of the *Interdyme* software for building dynamic interindustry models. These models store their vector and matrix data in files called Vam (for vectors and matrices) files.

Usually a model has a single Vam file which contains all vectors and matrices used in that model. To speed reading and writing during the execution of the model, these files are organized quite differently from the usual *G* data bank. The usual bank places the data for successive observations on a single variable in adjacent memory positions. The Vam file, on the contrary, puts all the cells of a vector for one year before going on to the next year. This difference is largely transparent to the user of *G7*, but it explains why some ways of working are faster than others.

Vam files can be created and viewed in *G7* and used as a source of data for regression or graph. They are just another type of data bank as far as *G7* is concerned. Several Vam files can be attached at the same time and use as sources of data. Data input and manipulation, however, is possible in only one Vam file at a time; this one is called the default Vam file. The next sections explain how to create a Vam file, how to make it the default Vam file, how to load data into it, and how to transform it in several ways.

The Vam Configuration File VAM.CFG

Every Vam file begins from a Vam configuration file, often called Vam.cfg VAM.CFG, which sets out the data content of the model in so far as the data is best thought of as vectors and matrices. This configuration file contains some very vital information -- namely, the starting date and ending dates for all the Vam file -- the range of years over which the model can be run or graphs and tables made. Then, for each vector or matrix in the model, there is a line containing

- its name,
- its number of rows,
- its number of columns,
- the number of lagged values with which a vector can be used in the model,
- if a matrix, whether or not it is "packed", so that only non-zero elements are stored,
- the file name of the titles for the rows of a vector or matrix,
- the file name of the titles for the columns of a matrix.
- a # followed by a comment usually explaining the economic nature of the variable and possibly the name of the file with historical data.

The format is free. Here is part of the VAM.CFG for the Mudan model of China:

```
#         Matrices and Vectors of the Mudan Model
#
1980 2010 # Starting and ending years
#
#Name |Number of  |Files of titles of|   Description
#     |row col lag|  rows   cols     |
#
# Matrices
am      33  33   0 sectors.ttl sectors.ttl # the input output matrix
bmcr    33  11   0 sectors.ttl hhrural.ttl # the bridge matrix for cr
bmcu    33  19   0 sectors.ttl hhurban.ttl # the bridge matrix for cu
bmv     33  40   p sectors.ttl bmv.ttl     # the bridge matrix for investment
# final demand vectors
```

```
hcr     11   1   0 hhrural.ttl            # Consumption of Rural Hh, Hh sectors
hcu     19   1   0 hhurban.ttl            # Consumption of Urban Hh, Hh sectors
out     33   1   3 sectors.ttl            # Output
# Vectors related to fixes
dump     33  1  0 sectors.ttl                # Discrepancy of Fixed Outputs
pdump    33  1  0 sectors.ttl           # Discrepancy in prices
fix     100  1  0 fix.ttl        # Fixes, to be filled in by Fixer
```

Note that the starting and ending dates do not control when a particular run of the model starts or stops, but define the range of the Vam files. The model in the example cannot start earlier than 1980 or run past 2010, but it certainly does not have to run over the whole span on each simulation. In this example, the model will have a vector named "hcr" which will have 11 rows and 1 column, as indicated by the first two numbers on the line; only current year values of this vector are needed in computing the results of the model in the present year, so the number of lags used, the third number on the line, is 0. In the example above, only the "out" vector requires Lagged values lagged values. Note the 'p' in this position of the "bmv" matrix. This 'p' marks a matrix that is to be " Packed matricesin vam.cfg packed" so that only non-zero items are stored. (This is not presently actually the case in the Mudan model.) The next item on the line is the name of the file with the sector names to be used for the spreadsheet-like displays of the vector. Everything following the # is a comment.

Names of vectors may contain up to 16 letters or numbers and may contain the underscore mark, "_". They must not, however, end in a number. This restriction is necessary because it is sometimes necessary to use the sector number as a suffix to the vector name and to convert between the suffix and subscript forms of the name. For example, we have to be able to recognize that pce[23] and pce23 are the same series. If we had a vector named g2, then g2[3] would convert to g23, and g23 would convert back to g[23], which is wrong. So no numbers at the end of vector names, please!

Note that *G7* is case sensitive; Q is not the same variable as q.

When using a Vam file, you may need a reminder of the names of the various vectors and matrices. You can use *G7*'s editor to look back to the VAM.CFG file or, if the names alone are sufficient reminder, you can use the "listvecs" command.

listvecs

This command shows a list of the vectors and matrices in the model.

Creating, Assigning, Defaulting, and Closing a Vam File

To create a Vam file from a Vam configuration file the command in *G7* is

vamcreate <vam configuration file> <vam file>

For example, to create the Vam file hist.vam from the configuration file vam.cfg, the command is

```
vamcreate  vam.cfg  hist
```

The "vamcreate" command may be abbreviated to "vamcr," thus:

```
vamcr   vam.cfg  hist
```

At this point, the newly created Vam file has zeroes for all of its data. The following pages deal with how to put data into it.

The first step is to assign it as a bank. The command is

```
vam  <filename> <letter name of bank>
```

For example,

```
vam hist b
```

will assign hist.vam as bank b. Letters *a* through *v* may be used to designate banks. However, it is generally a good practice to leave *a* as the G bank which was initially assigned.

In order not to have to continually repeat the bank letter, most commands for working with Vam files use the default Vam file. It is specified by the "dvam" command

dvam <letter name of bank>

For example

```
dvam b
```

A Vam file must already be assigned as a bank before it can be made the default. However, if several Vam files are assigned, the default can be switched from one to another as often as needed.

To review, the commands

```
vamcr  vam.cfg hist
vam hist b
dvam b
```

will create an all-zero Vam file as specified by the vam.cfg file, assign this file as bank b, and make it the default Vam file.

Assigning a Vam file to *G7* opens the file for writing as well as reading. The operating system insures that no two programs can ever have the same file open for writing. Hence, if after looking at model results in the file dyme.vam, we decide to run the model again, we must first close DYME.VAM with the *G7* "close" command. Thus the sequence might be

```
vam dyme c
gr c.out1
...
close c
```

and we are then free to run the model again and write to the DYME.VAM file.

Input of Time Series into Vam Files

There are a number of ways of reading data into Vam files. This help page shows ways to intoduce a single time series for one element of a vector or matrix. Following pages show how to introduce whole vectors or matrices.

vdata <series_name>

This command works just like "data" except that the series introduced goes to the default Vam file. Recall that if *fd* is a vector in the Vam file, fd23 will be the 23rd element of it. If *am* is a matrix, *am12.14* is the element in row 12, column 14.

vupdate
vup

This command works just like "update" except that the introduces series goes into the default Vam file.

Loading the Vam File from G7 banks

It is also possible to introduce data from G banks into the Vam file. Just as the "f" command will form a variable and store it in *G7*'s workspace, the "vf" command forms a variable and puts it into the default Vam file. The format is

vf <name> = <expression>

Example:

```
vf out1 = out1
```

This seemingly tautological example actually does something useful. It will look in *G7*'s workspace bank for out1 and if it fails to find it, it will look in the bank assigned as *a*. Let us say it finds out1 there. It will then copy this series over to the default Vam file. The right side of the vf command can be any valid *G7* expression, including the various @ functions, such as @log, @exp, @pos, @ifpos, and @csum. To specify that a variable on the right should come from the bank assigned as *b*, put b. in front of the variable name. The "vf" command works over the range of dates specified by the "fdates" command.

fdates <fdate1> [fdate2]

This command specifies dates which will be used as the range of action of the *f, vf, vc, index, ctrl* and other commands described below, and of the @cum() function. The default values of the fdates are the beginning and ending dates of the Vam file, respectively.

Example:

```
fdates 1980 1999
```

Input of Vectors into Vam Files

vmatdata

The "vmatdata" command puts whole vectors into the default Vam file. It can be used to bring in data in a variety of formats commonly used by statistical agencies in releasing input-output tables. It can read a file which shows any one of the following:

one vector in columns for several years

one vector in rows for several years

several vectors in columns for one year

several vectors in rows for one year

The "vmatdat" command requires at least two lines and three if a format for reading is specified. The form of the first line is always the same, as is the form of the third line, if present. The second line has two different forms, one if several vectors are being read for one year and another if data for one vector is being read for several years. The form of the command is

vmatdat <r|c> <nv> <nyrs> <first> <last> [skip]
<year> <vec1> <vec2> ... <vecnv>
OR --
<vec> <year1> <year2> ... <yearnyrs>
[form]

where

r|c	is 'r' or 'c' according to whether the vectors are rows or columns.
nv	is the number of vectors.
nyrs	is the number of years.
first	is the position in the vector in the Vam file of the first item in the data which follows.
last	is the position in the vector in the Vam file of the last item in the data which follows.
skip	is the number of spaces on each line which should be skipped before beginning to read data. If no value for skip is provided, then a form line (explained below) will be expected as the third line. If no spaces are to be skipped and no form line provided, then give skip a value of 0.
year	is the year of the vectors or the first year if nyrs > 1.
vec1	is the name of the first vector, vec2 is the name of the second vector, etc.
form	If no value for skip is given, then this form line must be provided. It should have the letter r in every position which is to be read in the following table. This

device allows direct reading of tables that have columns of data separated by | or other symbols or which have subtotals which are not to be read. A typical form line and a matching data line are:

```
form:                rrrrrrrrrrrrrrrrrrrr         rrrrr      rrrr
data:Agriculture     | 3450  3718  249   0  | 6780|  8102 |**| 1598
```

Columns which do not have an *r* are simply totally ignored. Consequently, they cannot be used to separate data fields. Thus, reading the following

```
    rrrrrrrr    rrrr
      851       2971
```

would produce one number, 8512971, not two numbers, 851 and 2971.

The first version of the second line is used if there are several vectors for one year; the second is used when giving data for one vector in several years. In both cases, the item of which there is only one is specified first and then the items of which there are several.

Any line beginning with a # is skipped completely.

For example, the lines

```
# Example of several vectors in columns for one year:
vmatdat c 6 1 1 57 3
1986      cons   gov    inv    stk    exp    imp
1         21           0            2            -3           6            7
...       ...          ...          ...          ...          ...          ...
57        38           401          0            0            17           15
```

would read the 6 columns of data into positions 1 through 57 of the named vectors (cons, gov, ...) for the year 1986. Positions 1 - 3 of each line, which contain the sector number for easy visual reference, are skipped.

Here is another example which reads the vectors as rows, as may often be the case when reading printed tables of primary inputs.

```
# Example of several vectors in rows for one year:
vmatdat  r 6 1 1 6  16
1986 labinc propinc deprec inter profit tax
#                  1     2     3     4     5     6
labor income    7303    21   368  1203  1302   820
proprietor inc  1215     9   100   107   303    92
depreciation     950     3    82   104   121    73
interest income  845     7    83    54    95    52
profits           50     3    25   217   337    38
indirect taxes   121     1    32   178   294    29
```

Note the use of the line beginning with a # to provide labels for the columns.

The second form of the second line is illustrated by the following example, which reads one vector, cons, for five years.

```
# Example of a single vector in columns for several years:
```

```
vmatdat c 1 5  1 57 2
cons 1985 1986 1987 1988 1989 1990
 1   31.8 37.2 32.5 38.7 41.2 43.1
....
57   1.2  1.7  1.8    2.0  2.1  2.2
```

Finally, here is an example which reads a single vector in columns for several years.

```
# Example of reading a single vector in columns for many years.
vmatdat r 1 28 1 8  12
demog 1989 1990 1991 1992 1993 1994 1995 1996 1997
#  Date  ncent south  west college  twoy  fs1   fs2   fs5   head1 head3
 89.000  0.243 0.345 0.208  0.220  0.469 0.243 0.323 0.106  0.274 0.350
 90.000  0.241 0.346 0.209  0.222  0.476 0.250 0.320 0.105  0.267 0.350
 91.000  0.240 0.347 0.211  0.224  0.482 0.247 0.320 0.105  0.263 0.348
 92.000  0.238 0.348 0.212  0.226  0.487 0.245 0.319 0.104  0.260 0.346
 93.000  0.237 0.349 0.213  0.227  0.493 0.244 0.319 0.102  0.257 0.345
 94.000  0.236 0.349 0.215  0.227  0.495 0.243 0.321 0.106  0.257 0.340
 95.000  0.233 0.351 0.218  0.228  0.507 0.252 0.320 0.102  0.257 0.343
 96.000  0.233 0.351 0.219  0.229  0.509 0.253 0.320 0.101  0.257 0.342
 97.000  0.232 0.352 0.219  0.231  0.511 0.254 0.321 0.099  0.256 0.342
```

It must be emphasized that, precisely because it can read in free format, the “vmatdat” command cannot interpret blanks as zero entries. There must be a numerical value for all cells in the tables, especially the 0's.

The flexibility of “vmatdat” often makes it possible to read in final demand columns or primary inputs as they appear in printed tables or on the diskettes provided by statistical offices.

For large models, it is convenient to be able to read "folded" vectors into the default Vam file. This is the function of the fvread command. The usage is:

fvread <vector_name> <year> <skip>

The <skip> is the number of columns to skip at the beginning of each line. There must be present as many elements as are in the vector, and they must be in order.

Example:

```
fvread fd 1997 10
finaldem 1   23 36 83 92 32
finaldem 6    8 23 73 80 75
```

This example will put the vector (23, 36, 83, 92, 32, 8, 23, 73, 80, 75) int the fd vector of the default Vam file for the year 1997.

Input of Matrices to Vam Files

There are four commands to introduce into the default Vam file a matrix from an ASCII file. The first two use different input formats to put matrices into the Vam file. The other two put data into packed matrix files, files from which the cells which are zero in all years have been eliminated.

The first, and most used, command is “matin”. It has the form

matin <matrix_name> <year> <firstrow> <lastrow> <firstcol> <lastcol> [skip] [form]

There should then follow a rectangular array of numbers matching the <firstrow>, <lastrow>, <firstcol>, and <lastcol> parameters of the command. As in vmatdata, the optional "skip" parameter is the number of spaces to be skipped in reading each line. The skip parameter and the form line work together exactly as described above for vmatdat: the absence of a skip indicates the presence of a form line. A ‘#’ in the first space of a line means to skip the whole line. Use of "matin" does not affect entries outside the specified area, so it can be used to update a matrix as well as to introduce it originally.

(It is also possible to introduce a matrix that is in a Windows spread sheet by a “Copy” and “Paste” operation into the window created by the *show* command described below. While this route may be a quick expedient, if the Vam file is changed and the process has to be repeated, it quickly loses its appeal.)

The second command for reading matrices is solely for conversion of models previously built with the Slimforp program. If you are not converting a model from Slimforp, skip this command. Its form is

matin5 <matrix_name> <year>[<width> <decs> <IndexWidth>]

and is followed by a matrix in the "punch5" format used in Slimforp. The end of the matrix is signaled either by a ';' in the first space on a line or by the end of the file. For those not familiar with the "punch5" format, it is a fixed length format, with 5 matrix entries to a line, each entry of the form <row> <col> <number>. One possible format in Fortran would be (5X,5(2I3,F9.5)). In this case, width = 9, decs = 5, and IndexWidth = 3. Here is a sample from the beginning of an A-matrix:

```
matin5  am 1977 9 5 3
# A_matrix for 1977. 83 rows and 83 columns.
AM     1  1 0.245790  1  4 0.000220  1  8 0.000410  1  9 0.281300  1 10 0.058360
AM     1 11 0.002070  1 12 0.005680  1 13 0.000380  1 15 0.001700  1 16 0.003060
AM     1 22 0.089770  1 24 0.000070  1 48 0.001210  1 49 0.000130  1 50 0.000030
```

Packed Matrices

There are two special commands for introducing data for a "packed" matrix. Packed matrices become important in models with many sectors, in which the matrices -- especially the various bridge matrices -- often become very sparse, that is, they have relatively few non-zero elements. *Interdyme* has the ability to carry the matrices in a packed form in which only the non-zero elements are stored. In most places in *G7*, the packed matrix can be used nterchangeably with the "full" matrix. For example, in the "index", "lint", and "show" commands described below, either full or packed matrix names can be used without having to remember which kind of matrix is being used. For the input of data, however, there is a special format for packed matrices. Moreover, packed matrices are not included in the Vam file. Only the file name of the packed matrix is in the Vam file; the actual data is in a special binary file for which we use the extension .PMX. This separation of the .PMX from the Vam file was done because (a) the .PMX files are fairly big and (b) they often do not change at all between various alternative runs of the model. Thus we might have five runs of the model, each with its own Vam file but all with the same pmx file. The savings in disk space over having five copies of the .PMX files could be substantial.

The first command for introducing data into packed matrices assumes that the elimination of the zero elements has already been done, as is likely to be the case if the matrix comes from Inforum programs for updating a matrix. The second command reads the matrix in full rectangular form and eliminates the zeroes itself. The result is the same in either case. The first command is:

pmatin <matrix_name> <packed_matrix_filename>

where <matrix_name> is the name of a matrix in the vam.cfg file which must have the letter p in the "lags" position. The <packed_matrix_filename> is name the DOS name of the .PMX file. (The .PMX should be included in the name.) The format of the data which follows is then:

<year>

<row> <num_non-zero_elements>

<col1> <element1> <col2> <element2> ...

For example

```
pmatin bm bm.pmx
1985
1  5  # row 1 has 5 non-zero elements
1  .656  7 .352  11 .038      23 .019  27 .218
3  2  # row 3 has 2 non-zero elements
7  .098  51 .923
.... ;
```

If the matrix is available for several years, the data for all years should be introduced with only one “pmatin” command, with each year's data preceded by the number of the year. It is essential

that the year numbers not be abbreviated; use 1987 please, not 87. If a cell is non-zero in any year for which there is data, the packed matrix will have a place reserved for it in all years. To signify the end of the data, provide a ';'.

pmatin1 <matrix_name> <packed_matrix_filename>

An alternative way of introducing data into packed matrices is the "pmatin1" command. The format is just like that of the "pmatin" command, except that the data should be arranged in rectangular rows and columns like the "matin" command. For example, if you wish to treat as a packed matrix a 3 x 3 matrix known as B in the Vam file, you can introduce it with the "pmatin1" command as follows:

```
pmatin1 B B.pmx
1995
   1  0  0
   5  0  1
   6  2  0
1996
  1.5 0  0
  6   0  2
  7   3  1
```

Note that in the above example, only those cells that are zero in all years will be left out of the packed matrix. Cell (3,3) will actually be stored, since it has a non-zero value in 1996. As with the "pmatin" command, you should put the data for all years you want to read in the same file, and use only one "pmatin1" command per matrix.

pmfile <matrix_name> <packed_matrix_filename>

Although packed matrix files often do not change between runs, they may. If, for example, we wished to study the effects of changes in the A matrix and the A matrix was packed, then we would need two different .pmx files. If the original was called AM.PMX, then to create the alternative, say, AMALT.PMX, we would go to the DOS prompt via File | DOS and do

```
copy am.pmx amalt.pmx
copy hist.vam vamalt.vam
```

Then from *G7*'s command box do

```
vam vamalt b
dvam b
pmfile am amalt.pmx
```

This *pmfile* command will both assign AMALT.PMX to be used whenever the am matrix is referred and put AMALT.PMX into AMALT.VAM as the name of the file to be used for the packed am matrix.

pmmode { “simple”| “advanced” }

Note that the “packed matrix filename” setting is limited to 30 characters, including any path specification. Sometimes more than one vam bank is loaded in G7, where the vam banks include packed matrix files with the same name but that are stored in different locations. Sometimes it is not convenient to reset the link for each vam bank using the pmfile command, or perhaps inclusion of a full path specification would be needed to distinguish between the files but the full specification exceeds the limit of 30 characters. In such cases, it might be useful to specify a link to each packed matrix file relative to the placement of the corresponding vam bank. By default, G7 will assume that the packed matrix file is in the current working directory, or, if a relative path is specified, that the specification is relative to the current location. This behavior can be controlled with the pmmode command. The “pmmode simple” command corresponds to the default behavior. The “advanced” mode instructs G7 to attempt to find PMX files in the same location as the corresponding vam bank, or if a relative path is specified for the PMX file, the root location should be the location of the parent vam bank.

Display of Vam File Data

All of the *G7* commands for the display of data all work for data in any assigned Vam file; it is just necessary to prefix the bank’s letter designation followed by a period to the variable name. Thus

```
vam dyme b
ti Output, Export, and Import of Agriculture
gr b.out1 b.ex1 b.im1
type b.out1
matty b.out1 b.ex1 b.im1
```

will graph or type out the time series of the first element of the out, ex, and im vectors from the Vam file assigned as bank b.

For Vam files, however, there is another command which shows vectors and matrices as if one were in a spreadsheet program. This is the *show* command. For vectors it format is just

show <bank letter>.<vector_name>

The values of the vector in successive years will appear as columns; dates run across the top and sector numbers down the side. The Options menu item allows the user to control the number of decimal places, fonts, and colors of the display. The display can be copied to the clipboard in the usual way for Windows programs. The contents of the clipboard can then be copied into a spreadsheet such as *Lotus* 1-2-3 or into a table in a word processor. On the Copy command, you get to specify how much you want copied — just the numbers or also the frame.

It is also possible to go in the other direction, from the spread sheet into the show window and thus into the Vam file.

For matrices, the "show" command has a somewhat different format

show <bank letter>**.**<matrix_name> <view> <line>

The "view" argument must be one of the following:

r for row

c for column

y for year.

If view is 'r', then <line> is the number of the row to be displayed. (A row is displayed as if it were a column.) If view is 'c', then <line> is the number of the column to be displayed. If view is 'y', <then> line is the year number. Examples:

```
show b.am r 5
show b.am c 7
show b.am y 1997
```

Cutting and pasting works as with vectors.

Vector Calculations

vc <vector_name> = <expression>

The "vc", or vector calculation, command evaluates the expression on the right and puts the results into the named vector. Only a few matrix and vector operations are implemented with *vc*. It can add or subtract any number of vectors and multiply a matrix times a vector. It cannot yet add or subtract matrices; parentheses are not yet supported. However, scalar values (either constants or G bank variables) can be used to premultiply or postmultiply a vector or matrix. Also, a scalar value can appear all alone on the right hand side of a *vc* equation, which indicates that the entire vector will be set equal to that scalar value. Between a matrix and a vector, an * means matrix-vector multiplication. Between two vectors, the * means element-by-element multiplication. Between a vector and a matrix, the / means multiplication by the transpose of the matrix or vector on the left. Between two vectors, the / means element-by-element division. The *vc* command is performed only over the interval defined by the current "fdates" command Example: If *am* and *cm* are matrices and *out*, *pdm*, *qcu*, and *cprices* are vectors, we could write

```
vc out = am*out + out    # Matrix multiplication, vector addition
vc qcu = out*pdm         # Element-by-element vector multiplication
vc out = qcu/pdm         # Element-by-element vector division
vc cprices = pdm/cm      # This actually is pdm'cm.
vc pdm = pdm / pdm{2000} # Normalize prices to 1.0 in 2000.
vc index = index{2000}   # Set equal to values in 2000 for all periods.
```

The vector_name must be a valid vector name in the default Vam file; matrices and packed matrices are not allowed. All right-hand-side vectors and matrices must be in the default Vam file; bank letters are not allowed.

The vc command is the only tool available for vector calculations that allows general (though limited) operations on the right-hand-side. Note that the vf command allows far more flexibility, but it operates on just one element at a time. While vc operates only on vectors, several standard matrix operations are offered. The following technique provides another means of copying vectors and matrices, in whole or in part, and allows addition and subtraction.

<vmake|vplus|vminus> [startyr [endyr]] <target> <source> [source ...]

The **vmake**, **vplus** and **vminus** commands make a vector out of another vector or part of a matrix, or make a matrix out of one or more vectors and/or other matrices. *vmake* replaces selected values in the target with the values from the source; *vplus* adds the values in the source to the values is the target; and *vminus* subtracts the values in the source from the values is the target. Up to 200 sources may be taken from one or more assigned Vam files. The command is as follows:

```
<vmake|vplus|vminus> [startyr [endyr]] <target> <source> [source ...]
```

<*startyr*> and <*endyr*> are optional parameters. If not given, then the process will work over the range of dates common to the target Vam file and all the source Vam files.

target is a matrix or a vector, and sources are matrices and/or vectors. Each matrix, whether a target or a source, is represented by

```
MatrixName(<r|c> [lines])[(<c|r> elements)]
```

<*r*|*c*> indicates whether the data will be taken by row or by column, lines the selected rows or columns, and elements the items selected within the specified row or column. The total number of lines selected from the sources must equal the number of lines selected from the target, and the number of elements selected from *each* source must equal the number of elements selected in the target. If lines or elements are omitted, the default is all rows/columns or all elements

Each vector, whether a target or a source, is represented by

```
VectorName [(elements)]
```

where *elements* are the items selected within the vector. If elements are omitted, the default is all elements. The number of elements selected from *each* source must equal the number of elements selected in the target.

Here are some examples:

The first example replaces all elements in rows 1, 2 and 3 of matrix *A* with all elements in vectors *a*, *b*, and *c*.

```
vmake A(r 1-3) a b c
```

The second example adds the values of all elements in vectors *a*, *b*, and *c* to the first five elements of columns 1, 2 and 3 in matrix *A* (assuming that vectors *a*, *b*, and *c* each have five elements.

```
vplus A(c 1-3)(r 1-5) a b c
```

If vectors *a, b*, and *c* had more than five elements, the command would have to be

```
vplus A(c 1-3)(r 1-5) a(v 1-5) b(v 1-5) c(v 1-5)
```

The next example subtracts column 3 of matrix *A* from the vector *c*.

```
vminus c A(c 3)
```

The final example is more complicated. It puts all rows of matrix *B* (starting from row 1) followed by the vector *c* into columns 2-100 of the matrix *A*, thus transposing the data in the sources.

```
vmake A(c 2-100) B(r) c
```

Groups of Sector Numbers

We have already seen how, with the "add" and "do" commands, it was useful to be able to specify a group of integers as arguments. In the "lint", "index", and "ctrl" commands which we are about to explain, this concept of a *group* of integers is carried further. For use in these commands, the groupis specified first and then used in the commands. *G7* has a dynamic group that can be specified and respecified over and over as necessary. It can also use the static groups defined by the *Fixer* program.

The dynamic group is defined by the command:

group <group definition>

Define the content of the dynamic group. The sector numbers are specified as in the *add* command. Example:

```
group 19-25 (20 22)
```

The name of this dynamic group is " : ". The names and content of the static groups defined in *Fixer* are preceded by a : in the following commands.

Named groups can also be introduced with the "group" command. The format is:

group <group name>

<group definition>

This version of the "group" command adds a named group to the GROUPS.BIN file. This group can then be used in other commands requiring group expressions by prefixing the group name with a colon ':'.

For example:

```
group Manufacturing
      1-58
f empmfg = @csum(emp,:Manufacturing)
```

listgroups

List the names of all the groups currently in the GROUPS.BIN file.

glist <name>

List the sectors in a group. The <name> specifies the group name. After the above *group* command, the command

```
glist :
```

would give the answer

```
19 21 23 24 25
```

If the "listgroups" command gives the answer

```
Ag Min Mfg Trans Trade Util Serv
```

Then

```
glist Ag
```

might, for example, give an answer like

```
1  2  3  4
```

Linear Interpolation of Vectors and Matrices

Linear interpolation of missing values in a vector is done by the “lint” command.

lint <names>

Replaces zeros and missing values in the time series of the vector or matrix elements with linearly interpolated values.. Zeroes before the first or after the last observation are not replaced. This command applies only to series in the Vam file. Groups may be used in the “names” field to refer to sectors of the currently loaded vector. Example:

```
lint pce1
```

This example loads the pce vector and then fills in the missing values in sector 1 by linear interpolation. We could similarly fill in the values for all the sectors in the present dynamic group by

```
load pce
lint :
```

or all the values for the static group Ag by

```
load pce
lint :Ag
```

Entire matrices may be interpolated with one command. For example

```
lint am
```

will interpolate the entire `am` matrix. This interpolation also works for packed matrices.

The “lint” command works on the entire range of the Vam file without regard to fdates.

Moving Vectors and Matrices by Indexing

A quick way to fill in values of a vector so that they all grow at the same rate is the use of the "index" command. It is used in conjunction with a guide series, a previously established single-variable time series, whose growth rates will be applied to the elements of the vector. It acts over the range of the currently specified "fdates". The form is

index <base date> <guide> <vector name>

For example, if “years” is the name of a series of the numbers of years — 1980, 1981, etc. — then to make all elements of the vector “pce” grow by 2.0 percent per year from 1999 to 2010, the following commands can be used:

```
f  g02 = @exp(0.02*(years - 1980))
index  1999  g02  pce
```

The use of groups is allowed in the names of series to be affected. For example, to move only the sectors 1,7, and 9 of pce, we could -- recalling that : is the name of the dynamic group -- do

```
group 1 7 9
load pce
index  1999 g02  :
```

To move just the sectors in the static group Mfg, the command would be

```
load pce
index 1999 g02 :Mfg
```

With a specific vector+sector name like *exp2*, command would be

```
index 1999 g02 exp2
```

to store the current loaded vector, load the *exp* vector, and move the series *exp2* by the index of *g02.*

A vector or matrix name not followed by a sector number means to move the whole vector or matrix by the index. This device can be used to project a whole input-output matrix, say, *am*, to be constant, as in this example:

```
fdates 1998 2020
f one = 1
index 1998 one am
```

A matrix name followed without space by a sector number means to move that row of the matrix by the guide. Thus, *am2* means move the second row of the *am* matrix by the guide.

If <guide> is the name of a vector and <name> is a matrix, then the rows of the matrix are indexed by the corresponding elements of the vector. If an element of the vector is zero in the base year, the corresponding rows of the matrix are unchanged. This feature works for both standard and packed matrices. It can be used to apply across-the-row coefficient change to an input-output matrix. For example, if *movers* is a vector, we can make each row of `the am` matrix follow the index of the corresponding element of *movers* by

```
fdates 1998 2020
index 1998 movers am
```

In applying this sort of across-the-row coefficient change, one usually does *not* want it to apply to the diagonal elements, for they have little to do with the technological changes affecting other coefficients. To avoid changing them, the following two commands are convenient.

diagextract <matrix_name> <vector_name>

Extract diagonal elements from the matrix and put them into the named vector. The command works for both regular and packed matrices.

diaginsert <matrix_name> <vector_name>

Insert elements in the named vector into the diagonal of the matrix. The diagonal element will remain zero for a packed matrix which has zero coefficient in that position for all years.

This example combines these last three commands

```
fdates 1998 2020
diagextract  am  diags
index 1998 movers am
diaginsert  am diags
```

Controlling Totals of Vectors and Subvectors

In specifying scenarios, it is important to be able to control the total of the projected values of a vector or of certain elements within the vector. The "ctrl" command gives this ability.

ctrl [basedate] <guide> <sectors>

This command imposes the values of the <guide> series as a control total on the named sectors of the currently loaded vector. The control is imposed over the period specified by the current fdates. If no basedate is present -- note that it is an optional element in the command -- the absolute values of the guide series are the control total. If the basedate is present, the guide series will be scaled to the total of the series in that period before being used as a control total. In naming the sectors, the allowable names are illustrated by:

0	for all the sectors in the loaded vector.
:	for the sectors in the present dynamic group
:Mfg	for the sectors in the static Mfg
1 7 9	for the group composed of sectors 1, 7, 9

load <vector_name>

When *G7* is working on a vector in a Vam file, it pulls the whole time series for the vector into a sort of vector workspace. Usually, this is done implicitly by simply referring to the

vector or one of its elements. The "load" command enables the user to do this loading explicitly. It is used principally in connection with the "index" command described below. Example:

```
load pce
```

store

Store the currently loaded vector back to the Vam file with the modifications that have been made. This "store" is automatic when a new "load" or implicit load is encountered. When a "quit" is encountered with an unstored loaded vector, the program asks whether it should store that vector. Answering “no” is a chance to escape if you know you have made a mistake and do not wish to mess up your Vam file. On the other hand, giving the “store” before the “quit” is a way to avoid this question. Long add files will, therefore, frequently end with a “store”.

Matrix Balancing by the RAS Method, Coefficients and Flows

In work with input-output analysis, one frequently needs to balance a table to known row and column totals or to convert flows to coefficients or vice-versa. Here are the commands to do so.

ras <matrix> <row> <col> [year] <r | c>

Balance the named distribution matrix by the rAs method so that <row> = <matrix>*<col> with each column of the matrix having a column sum of 1.0. It does not actually matter whether <row> and <col> are columns or rows, though in the above equation they are to be thought of as columns. If the [year] parameter is missing, the command works over the fdates range. If the sum of the elements of <col> does not equal the sum of the elements of <row>, the vector to govern is specified by the ‘r’ or ‘c’ at the end of the command. The other vector will be scaled to have the right total. The rows will then be scaled to get the correct row totals, then the columns scaled to get the correct column totals. Every five iterations, there is a report on the row and column in which the maximum scaling occurs. When the maximum scale factor differs from 1.0 by less than .00002, convergence is declared to have been reached. The last scaling will be of the rows. If convergence does not occur after 100 iterations, the program reports the problem and continues to the next command. Once the balance is achieved, the columns are scaled to sum to 1.0. The reason for this normalizing and how to deal with it if it is not wanted is explained after the next two commands.

coef <matrix> <vec> [year]

Convert the named flow matrix to a coefficient matrix by dividing each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the fdates range.

flow <matrix> <vec> [year]

Multiply each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the fdates range. This command is the opposite of coef command.

In our experience, the matrices most frequently in need of balancing are the distribution matrices, such as those that determine how personal consumption expenditure by budget category is to be distributed to industries. For this reason, the ras command, after balancing, automatically insures that the column sums are 1.0. Of course, if the matrix is the intermediate coefficient matrix, one does not want the columns to sum to 1.0. What do we do then? Let us suppose that am is an initial coefficient matrix, q is the vector of outputs, cs is the vector of column sum controls and rs is the vector of row sum controls. Then we would do

```
flow am q
coef am cs
ras am rs cs r
flow am cs
coef am q
```

getsum <matrix> <r|c> <vector>

Get the sum of the rows or columns of a given matrix for the range of periods specified by the *fdates*. The first argument must be the matrix or vector for which we want to obtain the sums. The matrix is assumed to be stored in the default vam file unless a bank letter is provided. The second argument must be 'r' or 'c' (to obtain the sum by rows or by columns). If only a partial sum is desired, then the columns or rows are specified next. The last argument is the vector where the result is to be stored; this vector must be located in the default vam file.The sum is calculated and stored for all years, so that we cannot have mixed information in the target vector.

Writing Vam File Data to ASCII Files

It is sometimes necessary to take data from a Vam file and output it to an ASCII file. For example, we may have used the Windows cut and paste commands to move data from a spreadsheet into the window of the "show" command, but we want an ASCII version of the file so we will not need to repeat this hand procedure if we rebuild the Vam file from scratch. The following commands provide various ways to output matrices and vectors to files that can be easily read back by *G7* or, in some cases, by other programs..

pmpunch <filename> <matname> [decs]

The non_zero elements of the matrix <matname> are written into the named file as input for the “pmatin” command. The [decs] argument gives the number of decimal places. The years written are determined by the current values of “fdates”. This command is especially useful

for converting a rectangular matrix in a Vam file into a packed matrix. One writes it out as an ASCII file with this command and then reads it in with the "pmatin" command.

punchvec <filename> <vecname> <startyear> <endyear> [<width> <decs>]
pv

Print out a vector to a file for all sectors, for the years specified. The optional width and decs arguments again specify the field width, and number of decimal places. The output file starts with a number of comment lines, telling the name of the vector, starting and ending years, and format information. Then one comment line gives the years as column headings. Each following line of the file contains the time series for one sector, with the name of the series, followed by the time series of data.

punch5 <filename> <vecname> [<width> <decs><IndexWidth>]
p5

Prints out a matrix or packed matrix to a file in "punch5" format. The years that are written are determined by the current value of fdates. Each year is written with a header that is a "matin5" command, which tells "matin5" also what field width, decs and IndexWidth to use. (The IndexWidth parameter is the width of the row and column numbers in the file. Thus, a file created with "punch5" can be read back into *G7* using the "matin5" command. This provides a convenient way to convert packed matrices into normal matrices. The "punch5" format prints 5 non-zero matrix cells to a line, with row number, column number and cell value for each cell.

vp <vector_name> [r|c] [field_width] [decimal_points]
or
vp <matrix_name><view><line> [field_width] [decimal_points]

Write the named matrix to the currently open "save" file. As you can see, this command has a format and options like those of the "show" command. Example:

```
save am.dat
vp am y 2000  9  6
save off
```

Note that the use of save files in conjunction with the "matty" or "type" commands are also useful ways of capturing Vam data in ASCII form file. Furthermore, since the *Compare* program can work with Vam files, you can use the "\gdata" command in that program to make an neat ASCII file printout of vector or matrix time series from a Vam file.

Titles for Vam Files

vtitle <title>

This command allows you to give a title to the default Vam file. This title is displayed when using *Compare.*

Running an *Interdyme* Model in *G7*

The *Interdyme* software is distributed a very simple model based on Chinese data. This model contains only the output and price solution, and a simple imports equation. This simple model, called " Slimdyme", should be installed and compiled first to make sure that everything is working correctly. The Slimdyme model file contains comments pointing out where you need to add the code for the functions of a typical interindustry model, such as final demand equations, employment equations, value added equations, etc. To install, create a directory called \SLIMDYME, and unzip the Slimdyme files into that directory. The Slimdyme.zip file will be named DYMEVxxx.ZIP, where the "xxx" indicates the current version of *Interdyme*. You should also have installed a Borland Builder, which includes a 32-bit Borland C++ compiler called by "bcc32".

Here is a sequence of steps to operate the model. It also tests that everything is installed properly.

1. From the *G7* command line, do

```
add initial
```

The file *initial* has the following contents.

```
# INITIAL for SLIMDYME
# Create vam file
vamcreate vam.cfg nohist
# Assign as bank b
vam hist b
# Make b the default vam file
dvam b
# Set up a constant A_matrix, using the lint command:
add am.dat
fdates 1980 2000
f mover = 1
index 1980 mover am
# Bring into the VAM file from the Mudan
# G bank data for some vectors.
ba mudan
fadd getout.add   sec33.fad
fadd getfd.add    sec33.fad
fadd getim.add    sec33.fad
fadd getprice.add sec33.fad
fadd getva.add    sec33.fad
# Move 1990 final demand vector, fd,
# forward by 2% per year.
add indexfor.add 1990 .02 fd
# Store the loaded vector, fd.  This is
# necessary only at the end of the work
# with the vam file.
store
# Before you can do Model | VecFixer, you
# must do "close b", but you may first want
# to "show b.fd" or show other vectors or
# matrices.
```

When the program finishes and the blinking prompt returns to the *G7* command line, you may, as indicated by the comments at the end of the *Initial* file, either first look at some of the vectors and matrices, or proceed to build the rest of the model. Before proceeding, however, you must first give the command

```
close b
```

Build the macro part of the model by the command

Model | IdBuild

This step also compiles and links the simulation model.

Prepare the macro fixes:

Model | MacFixer

From the information on the form which this command opens, *G7* will prepare the file MACFIXER.CFG and then execute the *MacFixer* program. For Slimdyme, you should always accept the default information in the form which opens in this and the following menus.

Prepare the vector fixes:

Model | VecFixer

From the information on the form which this command opens, *G7* will prepare the file FIXER.CFG and then execute the *Fixer* program.

Run the model

Model | Run Dyme

From the information on the form which this command opens, *G7* will prepare the file xxx.CFG where xxx is the name of the model as you indicate on the form. The model will then be executed with this control file.

Execute the *Compare* program with the command

Model | Tables

The data in the form which then opens is used to make a control file for the *Compare* program. If you are running *Compare* again with exactly the same input control, you can use

Model | Express Tables

Once you have verified that Slimdyme is working correctly, you can create a directory for your own model, named whatever you want. To get only the necessary files into that directory, first copy the file GETDYME.BATfrom the \SLIMDYME directory to your new directory, and then type "getdyme \slimdyme" from that directory. You should also look into the GETDYME.BAT file to see what it is copying. Before proceeding much further, you should study the rest of this manual to learn how to build an *Interdyme* model.

Alphabetical List of *G7* Commands

:

A ':' at the beginning of a line begins comment which extends through the next line beginning with a ':'. Text in comments does not cause any action by *G7*, except that the comments are printed to the screen when an add file runs.

Related topics: *#, “*, *add*, *Command files*.

#

Any text on a line that follows a hash sign (‘#’) will be treated as a comment. In addition to providing descriptive text for an add file, the # can be a method of temporarily removing lines from an add file that you may want to try again later.

Related topics: *“*, *:*, *add*, *Command files.*

“

This is also a comment character, if used as the first character of a line. Its use is deprecated, as the ‘#’ comments are more powerful.

ac <series> [<n>]

Calculate the autocorrelation function for the named series over the period specified by the latest "limits" command. The optional [n] is the maximum number of terms to be calculated; its default value is 20. The output shows the autocorrelation function and the autoregression coefficients determined by the Yule-Walker equations. Taking the rightmost elements of each line of the autoregression coefficients gives the partial autocorrelation function. The function is also placed into the workspace with a name derived by adding "_ac" to the variable's name. This facilitates graphing of the function. If the command was "ac vi 20", then the graphing command would be "gr vi_ac :0 20".

Example:

```
ac vin$ 11
```

Related topics: *bj*, *ARIMA*

(ad)d <addfilename> [Arguments]

The add file is one of the most powerful features of *G7*, and is used intensively by those who are familiar with the *G7* command set. Although the graphical user interface is a useful tool for teaching and for remembering features, the add file is necessary when you need to get large jobs done quickly, in unattended operation. Think of the add file as the batch file for *G7*. Only the basic syntax of the "add" command will be described in this section, as a

complete discussion of its use, with examples, is provided in the section entitled “Command Files, Groups and Do Lists”.

The commands in the <addfilename> are executed almost as if they were being typed in at the keyboard. *G7* knows whether the commands are coming from an add file or not, so sometimes it may behave slightly differently to help speed up the operation. As indicated by the above syntax, the arguments are optional. There may be 99 arguments, and they are passed as text strings. If a long text string with separators such as spaces or arithmetic operators needs to be passed, it should be enclosed in double quotes (""). The limit of the length of an argument is 90 characters.

Within the add file, the places where the arguments are to be substituted are indicated by "%1" up to "%99". Long strings passed in quotes will be inserted without the quotes.

Add files may also have a group on the command line. A group is a certain way of coding a list of numbers, which may represent industries, categories or other numbers. For example, 1-85 the list of numbers from 1 to 85 inclusive. To use a group as an argument, we must enclose those sectors in parentheses like this:

```
(1-85)
```

An "add" command allows its last two arguments to be groups. For example one could write out 85 "add" commands as follows:

```
add agr out 1
add agr out 2
.
.
add agr out 85
```

or equivalently, write them out in only one "add" command:

```
add agr out (1-85)
```

Up to two group arguments are allowed. However, group arguments can only be the last ones on the argument list. If two group arguments are used, by default, the outer loop is controlled by the first group argument, and the inner loop by the second group argument. For example,

```
add invest.reg (32-34) (52-54)
```

is equivalent to:

```
add invest.reg 32 (52-54)
add invest.reg 33 (52-54)
add invest.reg 34 (52-54)
```

There is an alternative to the double loop, parallel matching, with an 'm' option after the second group. For example:

```
add invest.reg (32-34) (52-55(53)) m
```

is equivalent to:

```
add invest.reg 32 52
add invest.reg 33 54
add invest.reg 34 55
```

That is, the first members of each group are matched to be the first pair of arguments, the second member of each group to be the second pair, and so on. The two groups must have equal number of members in parallel matching.

Related topics: *Command Files*, *Groups and Do Lists*, *do*, *fadd*, *gadd*

(addp)rint <n|y> (addt)ype <n|y>
(addt)ype

The "addprint" command is used to control the quantity of output going to the screen. If it is invoked with an "n" or "no", this command turns off the typing on the screen of the contents of the "add" files. This significantly speeds up the processing of large data files. If you want to turn printing back on, use "addp y" for "yes".

Related topics: *add*, *do*, *fadd*, *gadd*, *(ty)pe*.

autocomplete [<setting>]

The command box provides an auto-complete feature to speed the typing of repeated commands. Sometimes this feature proves troublesome. It can be turned off by clicking File | Autocomplete, or it can be turned off by typing the autocomplete command in the command box.

Settings may be "yes" or "no". If no setting is given, then the current setting is displayed. The setting will be saved when *G7* is closed, and the same setting will be restored when *G7* next is run.

(autop)rint <y | n>

This command sets the global autoprint flag in *G7*, which is 'n' (no), or false, by default. When you set the flag to true, with:

```
autoprint y
```

then graphs will be "printed" as soon as they are drawn.

Related topics: *(gr)aph*.

(ba)nk <bankname> [<location>]

This command assigns a *G7* workspace style bank (.bnk, .ind). The workspace style banks are the simplest type of bank in *G7*, and can be created by copying the files WS.BNK and

WS.IND to two files with a different root name. If <location> is not given, it is assumed to be 'a', which is the first slot. Up to 26 data banks of the various types may be assigned, in positions 'a' through 'z'.

In *G7*, the equivalent function can also be reached via the Bank, Assign menu item, and choosing Files of type "Workspace bank".

Examples:

```
bank mybank
ba h:\idlift\model\dyme d
```

Related topics: *Assigned Bank*s, *cbk*, *dfr*, *hbk*, *vam*.

bj <q> <y> = <x1>, [<x2>,] ..., [<xn>]

The "bj" command performs a Box-Jenkins estimation for autoregressive moving average models. Here q is the number of lags in the moving average error terms, e.g., with q = 2, u[t] = e[t] + b1e[t-1] + b2e[t-2]. This q must be no more than 4. The program will only check for the stability of the solution if q <= 2. No more than 10 independent variables are allowed at the current time.

At each iteration of the "Newtonian" non-linear regression algorithm, you will be allowed to intervene to try new values for the theta's, the coefficients for the moving average of error terms. If "save" is on, the first and the final equation will be in the ".sav" file with the theta values appearing as coeffients of "theta". These must be removed before building with *Build*.

Example:

```
bj 2 vif$ = vif$[1], vif$[2]
```

Related topics: *ac*, *ARIMA*

break

The "break" command terminates execution of an add file. If an add file calls a second add file, and the secnod contains the "break" command, then upon processing the "break" command *G7* will return to the first add file.

(bt)itle <ws | [a-z]>

The "btitle" command will report the title of either the assigned bank or of the bank in location 'a' through 'z'. If you ask for the title of a location that has no bank assigned, *G7* will print an error message.

The "btitle" command also allows you to change the title of the workspace bank. After the information is printed, *G7* asks if you would like to give a new title. If you hit [Enter] at this point, the old title will remain.

Related topics: *Assigned data banks*, *Workspace bank*

(bu)pdate <x> = <y>

The "bupdate" (bank update) command starts with a series <x> from the workspace bank, updates it with non-zero entries from the series <y> in the assigned bank, and replaces the series <x> in the workspace bank.

If <x> was not previously in workspace, the "bupdate" will put it there with values from the assigned bank. Note that <y> may be an expression.

For updating or combining entire data banks using this command, use the *bups* program, a small utility that comes with *G7*. If you type

```
bups <bankname>
```

bups will ask you what kind of bank this is, and then write out a text file of bups commands, one for each series in the bank. A similar file can be created using the "listnamescol(lnc)" command.

Related topics: *listnamescol*

(cat)ch [-a|-w] <catchfile>

The "catch" command captures most screen output (except graphs) to the named <catchfile>. The default flag "-w" causes a new file to be created. The optional "-a" flag causes G7 to open an existing file and append text to the end. This command is used to capture the set up of a regression and its results into a file. A regression display is caught as it stands when you proceed to the next command. To turn off the catching type "catch off". The "catch" command can also be used as a clever way to create new add files. Simply catch the results of a session, go in and edit out the extraneous output, and voila, a new add file!

Related topics: *add*, *(r)egress*.

(cb)k <bank_name> [<bank_location>]

The "cbk" command assigns a compressed bank (.CBK, .CIN). The <bank_location> is optional, and may be a letter between 'a' and 'z'. If no letter is given, the position defaults to 'a'. The compressed bank comes in two files. The first file, with extension .CBK, holds the data series in compressed format, and the second file, with extension .CIN, holds an index of names of series. When you give the <bank_name>, give only the "rootname", that is, without the file extension.

Related topics: *Assigned banks.*

cc <C language statement>

This has no effect in *G7* or *Build*, but the C-language statement is passed through to the file HEART.C, which contains the model written in the C language and ready for compilation. Such passed-through statements may be used to do special tasks for which *G7* and *Build* have no convenient mechanism -- such as scaling one group of variables to sum to another variable. For many models, no "cc" statements are needed.

Related topics: *Model Building*

cd <directory>

Normally when starting *G7*, you are asked to specify the location of the G.CFG you would like to use for identifying the default assigned bank and current working directory. Using the "cd" command, you can change the current working directory. After giving this command, *G7* will look in that directory when searching for .add files, .reg files and databanks.

Related topics: *G.CFG*.

(ch)ow <n>

The "chow" command is a test of homogeneity. In the calling format above, <n> is the number of regressions involved in the test.

A "Chow" test is a special case of the Fisher F test used to test the homogeneity of a sample. Typically, one runs two or more regressions covering parts of a sample and compares the sum of squared errors with that of a single regression covering the whole sample. If we were going to break up the sample into two parts, the command would be

```
chow 3
```

Then we run first the two regressions over the separate halves of the sample, and finally the single regression over the whole sample. After this last regression, one will be greeted by the results of the Chow test.

There is a special case in which the second "half" of the sample is too small to run the regression. In that case, give the command

```
chow 2
```

and then run the regression over the reduced sample and then again over the whole sample. After the second regression, the Chow tests appear.

Related topics: *Regression tests.*

(cle)ar

The "clear" command can be used to clear the contents of the results window. This window

can be useful for viewing the history of a *G7* session, but as it gets larger it may scroll more slowly. This command is equivalent to the File | Clear Results command from the menu.

close <bank letter> [<bank letter> <bank letter> …]
close all
The close command is used to close a bank, multiple banks, or all banks.

Examples:
close all # close all open banks, except the workspace bank
close a # close bank 'a'
close c d e # close banks 'c', 'd' and 'e'

coef <matrix> <vec> [year]
Convert the named flow matrix to a coefficient matrix by dividing each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the fdates range.

Related topics: *flow*, *fdates.*

(comc)oef <common_variables> <total_variables>
It fairly frequently happens that we have similar data sets collected at different times or in different places. We may then wish to estimate regressions which combine the data sets in various ways. The "comcoef" command in *G7* is designed for this purpose. It allows one to specify a number of "common coefficients" in the regressions which follow. The method may be best explained with an example. In the following example, "cps78" and "cps85" are data banks of from the Current Population Surveys of 1978 and 1985, respectively. The first has data on 550 individuals, the second on 534 individuals. We want to regress the logarithm of an individual's wage, lnwage, on the individuals education, ed. We want the coefficient of lnwage to be the same in both samples, but the interecepts may be different in the two years. Here is what we do.

```
f one = 1
comcoef 1 4
hbk cps78
lim 1 550
r lnwage = ! ed, one
hbk cps85
lim 1 534
r lnwage = ! ed, one
do
```

In the "comcoef" command, the "1" tells the program to use the same coefficient for the first 1 variable(s) in all the following regressions. The "4" is the total number of variables, counting the dependent variable, in the combined or "pooled" regression. Once the "comcoef" command has been received, *G7* accumulates regressions until the "do" command is reached. Only a limited list of commands are a permitted between the "comcoef" command

and the "do" command. They are the "bank", "hbk", "lim", "f", "add", "pause", "quit", and "r" commands. The quit command only quits comcoef, not *G7*.

The "comcoef" command can be used under the "chow" command and under the "catch" command. It also puts the estimated coefficients into the rcoef series in the workspace bank, as usual. It does not enter a series of predicted values into the workspace bank, so "gr *" will not work following a comcoef regression. Nor are the variable names shown on the display of the results, for in general the variables with a common coefficient may have different names while variables with different coefficients may have the same name, such as "one" in the above example.

cmtype <yes | no>
vcprint <yes | no>

If "yes", the simple correlation matrix and the standard deviation of each variable will be typed before the regression results. The default is 'no'.

con <top> <c> = <linear function of ai's>

The "con" command imposes a "soft constraint" on a regression. The "top" (trade-off parameter) determines how "soft" the constraint is. The higher this number is, the harder the constraint. Specifically, the constraint counts as top*T observations, where T is the number of regular observations in the regression. Soft constraints are also know as "Theil's mixed estimation", as "stochastic constraints", and are also a particular type of Bayesian regression.

Remember to put the value <c> of the constraint on the left hand side. Note that one parameter may appear in one or more constraints, but be careful not to impose constraints which are inconsistent!

We often use soft constraints when theory suggests that a parameter should be near a certain value, but ordinary regression gives a raw parameters which is not consistent with what theory suggests. Often, the theoretically sensible parameters are also those which lend stability to our models. The use of soft constraints is also implicit in the estimation of Almon lags, imposed using the "sma" command.

The "con" commands must be given before the regression to which they apply, and they will be cleared after that regression is estimated.

Examples:

```
con 100 1 = a3 + a4 + a5
con 200 0 = a7 - 3a6 + 3a7 - a8
```

Related topics: *sma*, *Soft Constraints*

(ct)rl <x> <concept> <group>

The "ctrl" command imposes the values of <x> series as a control total on a <group> of sectors of a given <concept> (such as exports). Results are stored in the workspace bank with names formed by <concept> and the given sectors.

For example, the command

```
ctrl tot out 1-10 (4-7) 13 15
```

imposes the values of tot as a control total on the named sectors out, i.e. sectors 1 to 10, except for 4 through 7, and then 13 and 15.

Related topics: *Forming Variables*, *Groups*, *scale*, *group*

cd <path>

Change the current working directory to <path>. This is equivalent to the menu option File | Directory.

Related topics: *pwd*

data <name>
<date> <observation1> ... <observationN>

The "data" command introduces data into the workspace data bank. There are 2 forms the command can take.

Form 1:

```
data sales
  83.1  34.0 56.8 44.5 55.6
  84.1  39.3 41.2 43.9 47.0 ;
```

The first number on each line is the date of the first observation on that line. Input is terminated by a ";". The ";" may be omitted in an "add" file, except on the last line of the file.

Form 2:

```
data sales 83.1
  34.0 56.8 44.5 55.6
  39.3 41.2 43.9 47.0 ;
```

The date of the first observation is given on the command line immediately after the series name. No dates appear on subsequent lines.

Form 1 is easiest if the data is being entered by hand. Form 2 is easier if the data is generated by a program.

Input data may contain floating point numbers in exponential form. Thus, the number 3 may be represented as 3.0, 3.0E+00, .300E+01, or 30E-01. Only E, not e, is recognized in this context. Any number of spaces between data observations is allowed (the data are free format).

A missing value may be represented by a single question mark (?). Therefore

```
data sales
   83.1  34.0 ? 44.5 55.6
```

indicate that the sales for 83.2 quarter is missing.

Related topics: *matdata*, *update*

del <series name>

Delete the named series from the workspace data bank. The last series is moved to the place of the deleted series. The workspace bank file is not physically reduced in size by a deletion, but if another variable is added to it after the deletion, it will be put in the space formerly occupied by the deleted variable.

Related topics: *(ren)ame*.

dfr <bank_name> <bank_location>

The "dfr" command is used to assign a Dirfor file as a *G7* assigned bank. Dirfor files are files containing forecast results and historical data from a run of the INFORUM LIFT model, or from one of the family of SlimForp international models. (SlimForp is the predecessor to *Interdyme*.)

These Dirfor files have a format described by a DIRFOR.DAT, or DIRFOR.CRD file. After assigning a Dirfor file with this command, you will be asked to provide the name of the appropriate DIRFOR.DAT file. Note that the DIRFOR.DAT file must point to the appropriate DIRHIS.DHS history file, as well as a correct MACRONAM.BIN list of macrovariable names. The series names for *G7* will be formed by the concept name in the DIRFOR.DAT file, joined with the sector or category number.

Related topics: *Assigned Banks*, *bank*, *cbk*, *hbk*, *vam*.

dfreq [0|1|2|4|6|12]

The "dfreq" command sets a default frequency for the left-hand side variable of an "f" command, when the right hand side does not have a frequency. The default is 0, which means no frequency. Otherwise, frequencies are determined by the frequencies of the series on the right hand side of the "f" command.

(diage)xtract <matrix_name> <vector_name)

This command takes the diagonal elements from the matrix and inserts them into the vector. The vector and the matrix must be of the same dimension, and the matrix must be square. One common use of this command is when you want to use the "index" command to move the A-matrix coefficients forward in time, excluding the diagonal. In this case, extract the diagonal to a scratch vector, do the coefficient indexing, and then insert the diagonal back again.

Example:

```
fdates 1998 2020
diagextract am z   # am is the A-matrix, z is a scratch vector
index 1998 mover am  # mover is a vector of coefficient change indexes
diaginsert am z
```

Related topics: *diaginsert*

(diagi)nsert <matrix_name> <vector_name>

This command is just the reverse of the previous one. It inserts the elements of the vector into the diagonal of the matrix.

dir [arguments]

The "dir" command prints the contents of the current working directory. The arguments are legal DOS arguments for the system "dir" command. Note, however, that any '#' characters are interpreted as the *G7* comment delimiter, and so any text after '#' is ignored.

Example:

```
dir t*.* # This command displays all files beginning with 't'.
```

Note that the /p option will not function properly, because at present there is no means for the *G7* user to interact with the operating system in this case (where the user is prompted to hit a key after every page). Because the *G7* window may be scrolled, the /p option is probably not needed. If necessary, the following *G7* commands will allow use of the option, but the results will not appear in the *G7* output window.

```
dos –i {dir t*.* /p; pause}
dos –I dir t*.* & pause
```

do <{*G7* commands with variables}> [<Arguments>]

A "do" command allows you to run a set of *G7* commands like an "add" command without actually creating an add file. Argument substitution is done just as in add files, where variables are %1, %2, etc. A "do" command can continue on several lines. However, the arguments should be placed on the same line as the closing brace mark '}'. As with the "add" command, no more than 99 arguments are allowed, and the last two arguments may be

groups.

Example 1:

```
do { gr out%1 65 97 } (1-87)  # graph output of sectors 1 through 87
```

Example 2:

The "do" loop can also make use of an if test in the loop. Useful examples of the need for this capability would be: 1) special operations needing to be performed for particular industries; or 2) report displayed for particular iterations. In this example, we iterate over values 1 to 10, and display the iteration numbers 1-3, 5 and 10.

```
do{
  if( %1 < 3 || %1 == 5 || %1 == 10){
    ic Iteration %1
    }
    }(1-10)
```

Related topics: *add, Command Files, Groups*

dos [<dos command>]
dos [<options>] [<batch file arguments>] <{>

The "dos" command allows for the passing of a DOS command to a temporary DOS shell. Batch files can also be run. When the DOS shell starts, you will see its output in the *G7* output window. If the "dos" command is given with no arguments, then a DOS command window will open, and will remain open until you type "exit".

Example:

```
dos runmodel.bat
```

To pass a group of commands to DOS, use the "dos{}" command. The braces tell *G7* to create a temporary batch file containing the dos commands between the braces. The temporary file is destroyed automatically upon completion. Arguments may be passed to the batch file by listing them after the "dos" and before the opening brace. Here is an example that copies a workspace bank to another bank:

```
dos: ws newws {
   rem Copying bank %1 to %2
   copy %1.ind %2.ind
   copy %1.bnk %2.bnk
   }
```

When this "dos" command is given, the three lines after the opening brace are written to a temporary batch file, and then that batch file is called with the arguments "ws" and "newws", which replace "%1" and "%2" in the batch file, respectively. The output of the

batch file is written to the *G7* output window.
If you would like to run DOS commands in interactive mode (where user input is required), use the "-i" option. For example:

```
dos –I idbuild master
```

opens a command window and executes the program *IdBuild* with `master` as a command line argument. If *IdBuild* requires user input, the user will see the prompt in the DOS window and can enter a response. The same option is available in batch mode (using the braces.)

(dv)am <bank_letter>

This command sets the *default vam file*. *G7* can have many banks open at once, and some of these may be Vam files. Most commands that involve using vectors or matrices in a Vam file will operate on the default Vam file. Each bank is opened at a different bank letter position, and that is that bank letter that should be given with this command. A Vam file must have already been assigned as a bank before it can be made the default. However, if several Vam files are assigned, the default can be switched from one to another as often as needed.

Example:

```
vamcr vam.cfg hist # Create hist.vam with VAM.CFG as configuration file
vam hist b         # Assign hist.vam in position "b"
dvam b             # Make hist.vam the default vam file
```

Related topics: *VAM.CFG file*, *vam*, *vamcr*.

ed [<filename>]

The "ed" command is equivalent to opening up an existing or new file using the File Open or File New commands from the editor menu. The editor opened up is internal to *G7*, and supports multiple files being open simultaneously, cut and paste between files, and editor windows can be open in the background while you are working with the *G7* Console.

Related topics: *Editing files.*

eqpunch <filename> | "off"

The "eqpunch" command is for writing equation results out to a file in a tabular format. We follow the precedent of giving those files an ".eqp" extension, but of course any file name will do. The use of the "eqpunch" command is analogous to the "ipch" command, and in fact both are often used in the same regression .reg file. The "eqpunch" command sets up a file by writing by the "titpch" and "tpch" commands. The "titpch" writes out a header for the table, and the "tpch" command writes out regression results, one line per sector. See the documentation on the "tpch" command for a complete example.

Related topics: *titpch*, *tpch*, *ipch.*

f <variable> = <expression>
f <variable>{<date1> [- <date2>]} = <expression>

This command is one of the most commonly used as well as most powerful and useful commands in *G7*. It is equivalent to the LET or GENR commands in other packages, and its job is to form a variable on the left hand side from an expression on the right hand side of the equals sign. The syntax of the expressions used by "f" or similar to those in most programming languages. Any algebraically legal expression is allowed, including nested parentheses.

The left side variable must be a single variable, or a variable with a subscript to indicate a specific time. For example, z and z{78.1} are valid variables on the left side. However, z[1] (z lagged once) is invalid on the left side. A range of dates also may be provided to specify the periods to carry out the calculations.

For a series which exists in the workspace or in the assigned bank, the "f" command modifies the value within the period specified by "fdates" without affecting the existing values outside of that period, and puts the series in the workspace. Otherwise, the "f" command creates a series, determines the values using the expression on the right hand side of the "f" command for the period specified by "fdate", and sets missing values, (-0.000001), for those observations outside the period. If "missing n" is in effect, the observations outside of the fdates period will be set to zero, instead of -0.000001.

In addition to the assignment operator (i.e. "="), the *f* command can add <expression> to the left-hand series using the "+=" operator, or it can subtract <expression> from the left-hand series with the "-=" operator, or multiply with "*=" or divide with "/=". These operators follow the syntax for the C++ programming language, though here they operate over a range of values.

For example, if we have in effect the following "fdates" command:

```
fdates 78 84
```

then for an "f" command

```
f x = <expression>
```

if x is an existing series in either the workspace or the assigned bank, x will be placed in the workspace with values from 1978 through 1984 being taken from the expression. Otherwise, x is created with values taken from the expression from 1978 through 1984, and set to be missing for the rest of the period.

Variable names must begin with a letter and may contain up to 32 letters, digits, or the '$' or '_'characters. Do not use a digit as the first character.

Example:

```
f x  =  y*(z[3] + v*@log(w))/s{77.1}
```

Here, the variable x is calculated and placed in the workspace using the variables y, z, v, w, and s. These variables must already exist. Note in the example:

z[3] means z lagged three periods, i.e. z(t-3) in a common notat
s{77.1} means the value of s in the first quarter of 1977,
@log(w) means the natural logarithm function of the variable w.

The full list of @ functions are shown in the list of functions.

Powers are obtained using the @sq and @pow functions, not ** or ^.

Right-hand-side expressions too long to fit on a single line may be continued by using a +, -, *, or / as the last character on a line.

The commands "f", "fex", and "fdup" have exactly the same effect in *G7* but very different effects in *Build*. See the model building topic for the differences.

Related topics: *fdup*, *fex*, *Missing values*, *Model building*

fadd <CommandFile> <ArgumentFile> [<arg> [<arg>] ...]

The named <CommandFile> is first executed with the arguments from the first line of named <ArgumentFile>, then the <CommandFile> is executed again with the arguments from the second line of the <ArgumentFile>, and so on until the <ArgumentFile> is exhausted. Additional arguments may be supplied with the fadd command, and these will be added to each line of arguments provided in the arguments file.

For example, the <CommandFile> could be

```
ti %2
gr %1
```

and the <ArgumentFile>

```
gnp$    "Gross National Product"
vfnre$  "Investment in producer durable equipment"
vfnrs$  "Investment in non-residential structures"
```

Then the result would be three properly titled graphs.

Related topics: *add*, *Command files*

fdates [date1 [date2]]
fdates off

fdates ±n1 ±n2

This command sets or resets the dates in effect for subsequent "f" commands. For a series which exists in the workspace or in the assigned bank, it modifies the value within that period without affecting the existing values outside of the period, and puts the series in the workspace. Otherwise, it creates a series, sets the values from the right hand side of the f command within the specified period, and sets missing values, that is, -0.000001, for those outside the period.

Example:

```
fdates 80.1 90.4
```

In this case, subsequent "f" commands only alter the value of a series between the first quarter of 1980 through the last quarter of 1990.

The default fdates are implicitly defined in G.CFG: fdate1 is the first period of the "Default base year of workspace file" as defined in G.CFG, fdate2 is the period implied by "Default maximum number of observations per series in workspace" in G.CFG.

"fdates off" resets the fdates to the default.

If valid dates have been specified, then they can be adjusted, moving either or both dates either forward or backward in time.

Example:

```
fdates +0 -2
```

If the command above follows the command in the first example, then the adjusted fdates will specify the period quarter one of 1980 through quarter 2 of 1990.

Related topics: *Dates in G7*, *f, G.CFG file*.

fdup <variable> = <expression>

In the context of the *G7* program, the "fdup" command is equivalent to the "f" command. However, when passed to the *Build* program through a .SAV file, it does nothing. It's main purpose is to do a calculation in one .REG file that is already done in another .REG file in a model, which does not need to be repeated. ("dup" stands for "duplicate").

Related topics: *f*, *fex*, *id*, *Model Building*

fex <variable> = <expression>

The "fex" command is just like the "f" command in the context of *G7*. However, when passed to the *Build* program in a .SAV file, it only computes the left-left side variable and puts it into the workspace. It does not put the right-side variables of the expression into the

workspace and does not put code for this calculation into the model. It is often used to define historical values of EXogenous variables, such as tax rates, and historical values of dependent variables of behavioral equations.

Related topics: *f*, *fdup*, *id*, *Model Building.*

findmode < m | a>

When there are multiple databanks assigned, this command sets the mode for searching through banks when looking for a time series. There are two modes: *manual* (m) and *automatic* (a). Manual is the default. In this mode, *G7* searches first the workspace bank, and then the bank in position 'a'. If you want to access or type a series in any of the other banks, you must explicitly prefix the series name with the bank letter, followed by a period. For example, if "listbanks" shows that you have four banks assigned, in positions 'a', 'c', 'e' and 'f', and you want to type the series gnp in bank f, then you must type:

```
ty f.gnp
```

In automatic mode, *G7* will automatically search through the workspace bank, then the assigned banks, in alphabetical order, until it finds a series with the specified name. If the name is a common one, such as "gnp", then be warned that you may not get the series from the bank you expected! In other words, if there was a monthly series in bank 'c' named "gnp" and a quarterly series in bank 'f', then typing

```
ty gnp
```

would get the monthly version.

Related topics: *Assigned banks*, *listbank (lb)*, *(ty)pe*.

flow <matrix> <vec> [year]

Multiply each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the fdates range. This command is the opposite of "coef" command.

Related topics: *coef*, *fdates*.

format {<width> <decs> <obsPerLine>} | "off"

This command is used to override the default settings of the "ty" and "sty" commands for the width of the data, the number of decimal points, and the number of observations printed on each line. The <width> and <decs> arguments also apply to the "matty" or "matpr" command. If you give the "format off" command, this returns *G7* to its default style of formatting.

Without using this command, *G7* decides on the format to use based on the absolute value of

selected elements of the series. This can sometimes result in a messy .SAV file if all data is printed at different widths and decimal points.

Related topics: *(ty)pe*, *(sty)pe*, *matty*

(freq)uency <series name> [frequency()<1|2|4|12>]

Some of the most Frequently Asked Questions about *G7* involve the topic of series frequencies.

Each time series in a G data bank has associated with it a one byte integer representing the frequency of the series. This frequency is set at the time of the creation of the series, but may be changed with the "freq" command. In a "data" statement, *G7* determines the frequency of the series by looking at the dates. An annual date (1995, 95.0) will indicate that the series is annual. Quarterly dates (91.3) and monthly dates (94.005) work accordingly.

Certain functions in *G7* will change the frequency of data, performing conversions from quarterly to annual and back, or from monthly to quarterly. These functions, of course, create the new series in the target frequency.

A formula or expression calculated with the "f" command must involve series that are all of the same frequency (or else no frequency), and the resulting series will have that same frequency. However, note that the "f" command can be used to set a series to a constant, or to a linear time trend, using the @cum() function. In these cases, the frequency is 0, or undefined. As just mentioned, *G7* is quite happy to use series created in this way in expressions with other series. However, the *Compare* program will complain if it is asked to print out a series with no frequency. To ensure that such series are assigned a frequency, use the "freq" command, or use the "dfreq" command in the beginning of your session or add file.

The "freq" command displays a series' frequency and allows you to change it. The series and its new frequency will be put into the workspace bank. Note that this is true even if the series was originally in an assigned bank.

Related topics: *Dates in G7*, *G7 functions*, *data*, *dfreq*, *f*, *Compare*

function <function name> { <function body> }

The “function” command enables you to define a function which will be available for the rest of the *G7* session. A function may be passed arguments, in which case they are referenced in the same way as in add files, with “%1”, “%2”, etc. Functions also have some flow of control capabilities, such as “if” and “else”. The following example is a function that just states its arguments, and how many arguments there are.

Example 1.

```
function TEST {
```

```
if (NARGS == 0) {
      ic Zero arguments were passed to the function TEST.
      }
else if (NARGS == 1) {
      ic The argument %1 was passed to the function TEST.
      }
else {
      ic Two arguments, %1 and %2, were passed to TEST.
      }
}
```

The second example demonstrates several tests, first for the comparison of strings and then for the comparison of numbers. The arguments may be replaced by the standard *G7* variables (i.e. %1, %2, etc.) defined for "add" files and loop iterators.

Example 2.

```
if( Clopper != Almon ){
   ic String inequality test failed.
   ic Clopper is not the same as Almon.
   }
if( A <= C ){
   ic Character less than or equal to comparisons.
   }
if( -1 < -0.5 ){
   ic Integer evaluation with negatives.
   }
if( ( 2 < 3 && 3 <= 4 ) || A <= C ){
   ic Multiple evaluations with AND, OR, and parentheses.
   }
```

Related topics: add, Command files, do

fvread <vector_name> <year> <skip>

The "fv" in this command name stands for *folded vector*. The input file read in this format is "folded" across several lines. Each line can begin with several columns of comment, and <skip> argument is the number of columns to skip at the beginning of each line. After the comment follows the elements of the vector, divided across several lines. There must be present as many elements as are in the vector, and they must be in order.

Example:

```
fvread fd 1997 10
finaldem 1   23 36 83 92 32
finaldem 6    8 23 73 80 75
```

This example will put the vector (23, 36, 83, 92, 32, 8, 23, 73, 80, 75) int the fd vector of the default Vam file for the year 1997.

gdates <gdate1> <gdate2> [gdate3]

Sets the dates used by subsequent "graph" (or "plot") commands. With two dates provided, the series will be graphed from <gdate1> to <gdate2>. If a third date is given, the series will be graphed from <gdate1> to <gdate3>, with a vertical line drawn at <gdate2>.

```
gdates a
```

Selects "automatic" dates for graph and type commands. The automatic dates are the first and last date of the series actually present. The default setting in "look" is for automatic dates, unless specific dates have been previously specified. Automatic dates also adjust automatically to the frequency of the series. This feature is not to be trusted when more than one series is being placed on the same graph.

Related topics: *Dates in G7*, *fdates*, *tdates*

getsum <matrix> <r|c> [<group>] <vector>

Get the sum of the rows or columns of a given matrix for the range of periods specified by the *fdates*. The first argument must be the matrix or vector for which we want to obtain the sums. The matrix is assumed to be stored in the default vam file unless a bank letter is provided. The second argument must be 'r' or 'c' (to obtain the sum by rows or by columns). If only a partial sum is desired, then the columns or rows are specified next. The last argument is the vector where the result is to be stored; this vector must be located in the default vam file.

Example:

```
getsum amf c intcol
```

This example puts the column sum of the A-matrix in flows (“amf”) into the vector intcol.

The sum is calculated and stored for all the years in the Vam file, so that we cannot have mixed information in the target vector.

Related topics: *flow*, *coef*.

glist <groupname>

This command lists the sectors in a group. The <groupname> specifies the group name. For named groups to be available, there must be a GROUPS.BIN file in the current directory, created by the *Fixer* program. (See the *Fixer* documentation for more on how to create named groups.) Otherwise, there is one dynamic group, named “:”, which is availlable after using the “group” command.

Example:

```
group 19-25 (20 22)
```

The name of this dynamic group is " : ". The names and content of the static groups defined in *Fixer* are preceded by a : in commands which use groups.. After the above "group" command, the command:

```
glist :
```

would give:

```
19 21 23 24 25
```

In *IdLift*, the command:

```
glist Health
```

would give:

```
83 84 85 86
```

Related topics: *group*, *listgroups*.

(gn)ame <prefix> <number>

In working with multisectoral models, it is common to have add files which draw graphs for many sectors. In this case, the "gname" can get a series of graphs started with a name and number, as follows:

```
gname out 1
```

The name of the save file for the first graph drawn will then be OUT1.WMF, for the second OUT2.WMF, and so on. Note that the numbers increase with each graph drawn whether or not it is saved.

Related topics: *Drawing graphs*, *gsave*.

(gp)rint

Print the displayed graph to the currently assigned default printer. This command is useful when you would like to print a large number of graphs without user intervention. Set up the default printer setup with Graph | Printer setup, then use "gprint" to print each graph in turn.

Related topics: *Drawing graphs.*

(gs)ave <filename>

Save the current graph to a Windows Metafile. Only give the rootname of the file in <filename>, the extension .WMF will be automatically appended. The file will be saved to

the current directory unless you supply a full path name.

Related topics: *Drawing graphs*, *gname*.

(gr)aph <name1> [name2] [name3] ... [name6] <date1> <date2> [date3]
(pl)ot

The "graph" command will graph the named series from date1 to date2 or date3, if present. If date3 is present, a vertical line (to separate history from forecast) will appear at date2. If dates have not changed since the last "graph" command, they need not be repeated. After a regression, the actual and predicted values may be plotted by "gr *". The actual data will be marked by squares; the predicted, by plusses, unless the marks have been changed by a previous "line" command.

Example:

```
title RTB -- The Treasury Bill Rate
gr rtb 70.1 85.1
```

Related topics: *Dates in G7*, *Drawing graphs in G7*, *(hr)ange*, *(le)gend*, *(lgr)aph*, *line*, *(mgr)aph*, *printer*, *(sgr)aph*, *(subti)tle*, *(ti)tle*, *vaxlab*, *(vaxti)tle*, *(vr)ange*.

(gridty)pe [file <filename>] [<date1> <date2>] <ser1> [<dp1> ...<seriesN> <dpN>] **;**

This command displays a scrollable spreadsheet, or grid in a new window. The each column is a variable, and each row is a separate date. Column and row headings are also included. The grid display is convenient for scrolling around to look at the data. One can also copy data from the grid display to the Windows clipboard. If the optional "file <filename>" part of the command is given, the results are written to an output file, exactly as with "matty". The dates are optional. If not given, they will defautl to the current "tdates". After each series name there is an optional "decimal places" number (<dp1> ... <dpN>). Each series can be specified to have a different number of decimal places. If not specified, the default is 3, unless the "decs" or "format" command has been given to specify a different default. If a value for <dp> is given for one series, it will be in effect for all the other series in the list. This command is available only in *G7*.

Example:

```
# GRIDTY.TST - Test the operation of the "gridty" command.
hbk quip
gridty gnp c v;
```

Related topics: *matty*, *show*.

(gro)up <group definition>

Define the content of the dynamic group. A group definition works as follows. Any list of numbers separated by blanks or commas may be included. Two numbers joined by a dash ('-

‘) indicate a consecutive range of numbers to be included. A group definition within a group definition, enclosed by parenthesis indicates a group of sectors to be excluded from the overall group.

For example:

group 1-10 (4, 7-9)

Indicates a group consisting of the numbers (1,2,3,5,6,10). The name of a group created by the “group” command in *G7* is “ : ”. This name is used in the various commands that may use groups. These commands may also use *named groups*, if they have been created in a GROUPS.BIN file by the *Fixer* program.

Related topics: *glist*, *listgroups*, *Groups.*

(gtf)ile <stubfile | “off”>

The “gtfile” command scans the specified stub file and stores the starting position of each line in the stub file. This command is used only in preparation for giving a sequence of “gtitle” commands. The titles from the stub file which has just been scanned are now available to be used as titles, as specified by “gtitle”. To close the “gtfile” command, and free up some memory, give “gtfile off”.

Related topics: *gtitle*.

(gti)tle <n>

This command requires that a “gtfile” command has been given with a valid stub file as an argument. The “gtitle” command will then use the n’th line of the stub file to take a title in the following graph command. (The title will be the text after the semicolon (‘;’) in the stub file.) These commands, “gtfile” and “gtitle” are particularly handy in combination with the “do” command, or with an add file with group arguments. Since in these cases only sector numbers are being passed, the “gtitle” command allows you to specify a sector number, and get back the title for that sector.

Related topics: *gtfile*.

help

The “help” command can be given from the command line, in which case it will bring up the on-line version of the *G7 Command Reference*. That reference is organized by subject, rather than alphabetically, like this one. Other help menu items available are *Contents* and *Tutorial*.

Related topics: *Help menu*.

hbk <bank_name> <bank_location>

The "hbk" command assigns a hashed bank (.HBK, .HIN). The <bank_location> parameter is optional, and may be any letter between 'a' and 'z'. If no letter is given, the position defaults to 'a'. The hashed bank comes in two files. The first file, with xtension .HBK, holds the data series in compressed format, and the second file, with extension .HIN, holds a hash table that allows for quick access to a large number of series. When you give the <bank_name> to the "hbk" command, give only the "rootname" of the file, that is, without the file extension.

Note that when you do a "listnames" command on a hashed bank, the series names will not be in the order in which the bank was created using the *Banker* program. This is because the hashing algorithm puts the series names into various "bins", based on the hash value of the series name. When reading the series names back again, *G7* just goes through each bin in sequence, which is not the sequence in which the bank was created. See the help on *Banker* for more on how to create compressed and hashed banks.

Related topics: *Assigned Banks*, *bank*, *Using Banker*, *cbk*, *dfr*, *vam*.

(hes)itate < y | n >

This command is almost the opposite of the "zip" command. If you give "hes y", then *G7* pauses before displaying each regression so that you can read the commands which prepared the regression. To turn off hesitation, use "hes n". Remember, if you wish to have the add file stop so that you can see what has just gone by on the screen, use the "pause" command in your add file. After using "pause", tap the [Esc] key to continue.

Related topics: *add*, *pause*, *zip*.

hl <rho1> <rho2> <incr> <y> = <x1>, [<x2>,] [<x3>,] ...[<xn>]

The "hl" command applies the Hildreth-Lu procedure for correcting for serial correlation of errors. It is a special case of Generalized Least Squares (GLS). In the Hildreth-Lu procedure, a guess of the autocorrelation coefficient of the errors, rho, is chosen, multiplied by the equation lagged once and subtracted from the unlagged equation. The resulting equation is then estimated by ordinary regression. Another value of the guess of rho is then chosen and the process repeated. In this command, <rho1> is the starting guess of rho, <incr> is the amount by which it is incremented on each iteration and <rho2> is an upper limit on the guess. The <y> and <x1>, ..., <xn> are as in the r command. At the end of the process, you will get a table with this heading:

```
RHO-HL    SEE 1 AHEAD  RHO-EST    SEE LONG-RUN
```

The RHO-HL shows the assumed rho, the SEE 1 AHEAD shows the standard error of estimate (SEE) of the estimated equation, RHO-EST shows the rho of the estimated equation, and SEE LONG-RUN shows the standard error of using the fitted equation on the original, undifferenced data, without a knowledge of the true lagged value of the dependent variable, as must be done in forecasts of more than a few periods ahead. If the "save" command is on,

all of the estimated equations will placed in the ".sav" file as undifferenced equations suitable for going into a model. You must choose which one you want.

For graphing the "hl" output, one should run the equation one last time with the chosen value of rho as both the starting and ending rho. Then also run the equation with just the "r" command (no autocorrelation) Then do two graphs:

```
gr depvar predp1 hlshort
gr depvar predic hllong
```

The first will compare the actual with two one-period ahead forecasts, the one from ordinary least squares (+'s) and one from Hildreth-Lu (x's). The second compares the actual with two forecasts not relying on the lagged value of the dependent variable. Again, the un-corrected least squares fit is shown by +'s and the fit using the H-L correction is shown by x's.

The pauses at the end of each fit in the HL procedure can be eliminated by the "zip" command. If you use zip, however, you must remember to do "zip off" before using the "graph" command, if you want to see graphs.

(hr)ange <bottom> [top]
hr <bottom> [line1] [line2] ... line[8] <top>
hr off

The syntax of the "hrange" command is identical to that of the "vrange" command. However, it is used only when doing "sgraph", where each axis corresponds to the value of one variable. In this case, it defines the range or locations of tick marks for the horizontal axis.

Example, from the *G7* Demo:

```
ti The Phillips Curve -- so to Speak
subti from 70.1 to 96.1
gdates 70.1 96.1
vaxl out
vaxti Inflation
legend s
vr 0 2 4 6 8 10 12
hr 4 6 8 10 12
f infl = 400*(dgdp - dgdp[1])/dgdp[1]
sgr infl unempc
```

In this example, inflation (infl) is measured on the vertical axis, which runs from 0 to 12, and unemployment (unempc) is measured on the horizontal axis, which runs from 4 to 12. Axis labels and tick marks on both axes are every two percentage points.

Related topics: *Drawing graphs*, *vrange*, *sgraph*.

ic <comment>

The "ic" (insert comment) command is used to put comments (lines beginning with '#') into save files. This can be useful when you are writing out a *G7* data file, and would like to put titles or documentation next to each data series.

Related topics: *comments*, *save*.

id <identity expression>

The "id" command has no effect in *G7* but, when passed to *Build* puts all variables (without computing the left-side one) into the workspace and puts an identity into the model. The syntax for the expression is the same as the "f" command.

Example:

```
ti 4. Additions and Alterations
fex cst4h = cst4$/hhld
r cst4h = ddi87h, rcmor, ratdif[1], hhead, singsh
id cst4$ = cst4h*hhld
gr *
```

This example is one of the construction equations from the *IdLift* model. The "id" statement is used here so that there will be a statement in the model to create cst4$ after calculating the regression. However, we don't want the estimation regression runstream to recalculate the value of cst4$ and put it into the workspace bank, since the proper values of this series are already put into the bank by the previous "fex" statement.

Related topics: *Model building*, *f*, *fex*, *fdup*.

if([comparisons]){ [...] }
[else if([comparisons]){ [...] }]
[else{ [...] }]

The if command evaluates a series of logical expressions. If the arguments are true, then the following block of code, contained in a set of brackets, is executed. If the argument is false, then else if statements are processed in the same way. If all if and else if arguments are false, then the code following the final else command is processed.

Three types of comparisons are legal. First, a number may be compared to a second number. Second, a single number may be given and implicitly compared to zero. Finally, a string may be compared to a second string.

Valid comparisons between two strings, with the comparison based on the lexicographical ordering, or between two numbers are

<	is LESS THAN
<=	is LESS THAN OR EQUAL TO
==	is EQUALS
>=	is GREATER THAN OR EQUAL TO
>	is GREATER THAN

!= is NOT EQUALS

and

! is the logical negation operator. If given before a number, then the result is FALSE if the number is nonzero and the result is TRUE if the number is zero. If given before a set of parentheses, then the result is FALSE if the expression within the parentheses is TRUE, and the result is TRUE if the expression within the parentheses is FALSE.

Additional string comparison tools include:

strcmpi(<string1>,<string2>)

Compares string1 and string2, without case sensitivity. Value is TRUE if strings match.

strncmp(<string1>,<string2>,<N>)

Compares the first N characters of string1 and string2, with case sensitivity. Value is TRUE if the first N characters of the strings match.

strncmpi(<string1>,<string2>,<N>)

Compares the first N characters of string1 and string2, without case sensitivity. Value is TRUE if the first N characters of strings match.

Multiple comparisons may be performed, with the following operators joining them

|| Logical OR, which returns TRUE if either the left hand side or the right hand side is TRUE.

&& Logical AND, which returns TRUE if both the left hand side and the right hand sides are TRUE, and returns FALSE otherwise.

The following list of keywords may be used

NARGS

The number of arguments passed to a function or to an add file.

The presence of a series in a data bank can be tested.

exists

exists(a.x) returns *true* if variable *x* is found in bank *a*.

A number may be retrieved from a data bank for comparison.

eval(series, date)

The value of series in period date is retrieved, either from the workspace or from the specified bank. The value may be compared to a scalar or to a second eval result.

Comparisons may be evaluated as a group by surrounding them with parentheses. Blocks of code following the arguments must be surrounded by brackets {}.

Examples:

```
if( 1 - 2 < -0.5 )  { ic Subtraction }
if( 2 < 3 && 3 <= 4){ ic Multiple evaluations }
```

```
if( 2 < 3 || 3 <= 4){ ic Multiple evaluations with OR }
if( 1 == 1 )        { ic Equality }
else if( !0 )       { ic Else if and negation }
else                { ic Else }
if( D != N ){ ic String inequality comparisons }
if( D <= N ){ ic String less than or equal to }
if( mod(7,2) == 1 ) { ic Modulo arithmetic }
do{
   if( %1 == 3){
      ic Continuing on Iteration %1
      continue
      }              if( %1 == 5 ){
      ic Breaking on Iteration 5
      break
      }
    if( %1 > 5 ){ return ERR }
    }( 1-10 )
```

return [arg]

This command interupts execution of the G7 script. If G7 is in the midst of processing a do loop or an add file, then the arguments determine the next actions of G7. Arguments may be given in upper or lower case.

Arguments:

OK The default argument. Exits a do loop or add file, but processing continues.

ESC Processing halts entirely, but no error messages are shown.

ERR Processing halts entirely, with error messages displayed.

Examples:

```
return ERR
```

(in)dex <base date> <guide> <vector name>
or
(in)dex <base data> <guide> <groupname>

A quick way to fill in values of a vector so that they all grow at the same rate is the use of the "index" command. It is used in conjunction with a *guide series*, a previously established single-variable time series, whose growth rates will be applied to the elements of the vector. It acts over the range of the currently specified fdates.

For example, if "years" is the name of a series of the numbers of years — 1980, 1981, etc. — then to make all elements of the vector "pce" grow by 2.0 percent per year from 1999 to 2010, the following commands can be used:

```
f  g02 = @exp(0.02*(years - 1980))
index 1999 g02 pce
```

The use of *groups* is allowed in the names of series to be affected. To use groups, we must first use the "load" command to load the vector into memory. For example, to move only the sectors 1,7, and 9 of pce, we could do:

```
group 1 7 9
load pce
index 1999 g02 :
```

Recall that ":" is the name of the dynamic group, if it exists. To move just the sectors in the static or named group ConsumerDurables, the command would be:

```
load pce
index 1999 g02 :ConsumerDurables
```

To index only a specific vector element like "exp2", the command would be

```
index 1999 g02 exp2
```

A vector or matrix name not followed by a sector number means to move the whole vector or matrix by the index. This device can be used to project a whole input-output matrix, say, am, to be constant, as in this example:

```
fdates 1998 2020
f one = 1
index 1998 one am
```

A matrix name followed without a space by a sector number means to move that row of the matrix by the guide. Thus, "am2" means move the second row of the am matrix by the guide.

If <guide> is the name of a vector and <name> is the name of a matrix, then the rows of the matrix are indexed by the corresponding elements of the vector. If an element of the vector is zero in the base year, the corresponding rows of the matrix are unchanged. This feature works for both standard and packed matrices. It can be used to apply across-the-row coefficient change to an input-output matrix. For example, if mover is a vector, we can make each row of the am matrix follow the index of the corresponding element of mover with:

```
fdates 1998 2020
index 1998 movers am
```

In applying this sort of across-the-row coefficient change, one usually does not want it to apply to the diagonal elements, for they have little to do with the technological changes affecting other coefficients. For this, we would use the "diagextract" and "diaginsert" commands to first save the diagonal, then do the indexing, and then put the diagonal back in, as in the following example:

```
fdates 1998 2020
diagextract am diags
index 1998 movers am
```

```
diaginsert am diags
```

Related topics: *group*, *fdates*, *diagextract*, *diaginsert*.

ipch [<which>] <label> <sector> [<type>] [<psn1>] ... [<psnN>] [extra <var1> .. <varN>]

The "ipch" (*Interdyme* punch) command is used for creating equation files for use in building *Interdyme* models.

where:

<which> is the suffixed number
<label> is the name of the vector for which the equation is estimated
<sector> is the number of the sector to which the equation applies
<type> is a character ('a', 'b', etc.) that signals the type of form of the equation. The type should not be a numeral.
<psn1> (optional) the column number in the matrix where the first regression coefficient belongs.
<psnN > The column number in the matrix where the n'th regression coefficient belongs.

At the end of the command, type the word "extra" (without the quotes), and then type several variable names that hold the values of extra parameters you would like to pass to the file. For example, for an investment equation, which calculates capital stocks, and replacement investment, a physical service life is needed to calculated the spill rate. The following code would write the physical life at the end of the parameter list:

```
f plf1 = 4.9
r eqi1 = out1, out1[1], out1[2]
ipch eqi 1 a extra plf1
```

The "ipch" is not useful unless an equation file has already been opened with the "punch" command. See the *Interdyme* manual, chapter 4, for more information on the "punch" and "ipch" commands, and about detached-coefficient equations in *Interdyme*.

Related topics: *Interdyme models*, *punch*

intvar <prefix> <date1> <date2> [<date3> <date4> … <dateN>]

The “intvar” command makes up linear interpolation functions between the given dates. Each linear interpolation function begins at 0 and remains at 0 until its particular time interval comes; then it rises by 1 each period until the end of its interval. After that, it remains constant at whatever value it has reached. For example, if the <prefix> is “tax”, and we have 6 dates, 6 linear interpolation functions will be created, with names “tax1”, “tax2”, … “tax6”. Then, if we regress a policy variable, such as the federal tax rate, on these functions, the predicted values from the regression take the shape of a set of spliced line

segments, approximating the curve of the left hand side variable. For example, to approximate the federal personal tax rate in the *Quest* model, use:

```
r pitfBR = tax1, tax2, tax3, tax4, tax5, tax6
```

Related topics: *Optimization*

(le)gend <a> [<b>]

The "(le)gend" command controls printing a legend at the bottom of a graph.

<a> = y for yes, prints the legend (default)
<a> = n for no, do not print the legend
<a> = s, leave space for legend but do not print. Useful for allowing the user to annotate the legend.

<b> = y for yes, mark the dates (default)
<b> = n for no, do not mark dates. Useful for making bar graphs of data occurring at arbitrary intervals.

Related topics: *Drawing graphs in G7*

(lgr)aph <name1> [name2] [name3] [name4] <date1> <date2> [<date3>]

The "lgraph" command is exactly like the "graph" command except that the series are expected to be in logarithms. The vertical scale will be marked in the natural units, not the units of the logarithms. If vertical range control is in effect (see below), the vertical ranges will be presumed to be in natural units. This type of graph is also called a semi-log graph.

Related topics: *(gr)aph*

(lim)its <date1> <date2> [<date3>]
(lim)its <±n1> <±n2> [<±n3>]

The "limits" command sets limits for regressions and for @sum() commands. Estimation begins at <date1> and ends at <date2>. Simulations or forecasts will be made to <date3>. Remember that quarterly dates require .1, .2, .3 or .4. Monthly dates require .001 ... .012. Annual dates are integer values.

Example:

```
lim 75.1 84.4 86.3
```

If valid dates have been specified, then they can be adjusted, moving either or both dates either forward or backward in time.

Example:

```
lim +0 -2 +1
```

If the command above follows the command in the first example, then the adjusted limits specify the period quarter one of 1975 through quarter 2 of 1984, and a forecast period through quarter four of 1986.

Related topics: *Dates in G7*, *Regression*

line <linenumber> <color num | name> <thickness> <mark> <style> <barfill> <left> <right>
line <linenumber>

The "line" command is used to control the appearance of lines and bars in graphs. The command may be typed interactively, or used in add files to change the appearance of graphs in a "show" file. In *G7*, the "line" command can be written out from the Graph | Settings dialog box, if you click the Save to file button. This brings up another dialog which asks you the name of the add file that you would like to save the graph settings to. Let's say you call this file GRAPH.SET. The contents of GRAPH.SET might look something like:

```
line 1      255  2 +  0  0  0.000  1.000
line 2 16711680  2 s  1  0  0.000  1.000
line 3    32768  2 x  2  0  0.000  1.000
line 4 16776960  2 d  3  0  0.000  1.000
line 5  8388736  2 ^  4  0  0.000  1.000
line 6  8388608  2 v  0  0  0.000  1.000
line 7  5315660  2 n  0  0  0.000  1.000
```

Alternatively, providing the line command followed by a single "linenumber" argument causes the settings for the corresponding line to be printed to screen. Settings for up to 7 series can be specified by the "line" command. The first argument, <linenumber> corresponds to the series given in that position on the "gr" command line. The first series after "gr" is line 1. The second argument <color_number> can be a decimal number representing the components of red, green and blue of the color, or it can be a color name. Both *G6* for DOS and *G7* use more or less the same set of color names, so if you want your add file to work under both programs, you should use names instead of numbers. The third argument <thickness>, may be set to anything you like, but something in the range of 2 to 4 is generally pleasing. The fourth argument specifies the type of marker on the line, if any. Here is a short table of the possible characters for this argument, and what they mean:

Symbol	*Meaning*
+	Plus signs
x	X marker
s	Square
d	Diamond
>	Arrow in direction of line
v	Downware pointing triangles

b	Bars
f	Bars with no surrounding rectangle

The next argument is the <style> of the line, possible numeric values for this argument are:

Value	*Meaning*
0	Solid
1	Dashed
2	Dotted
3	Dash dot
4	Dash dot dot
5	Clear (blank line)

The next argument, <barfill> indicates the type of fill pattern that a bar will take, if a bar graph has been chosen. Possible numeric values for this are:

Value	*Meaning*
0	Solid fill
1	Clear (empty)
2	Horizontal lines
3	Vertical lines
4	Forward diagonal fill (/////)
5	Backward diagonal fill (\\\\\)
6	Cross hatch
7	Diagonal cross hatch

The <left> and <right> parameters, must be between 0 and 1. They show where, within the space allocated for the bar, the left and right edges of the bar go. For example, with left = .1 and right = .9, the bar will be in the center of the allowed space, and the bars will be four times as wide (.8 units) as the space between them (.2 units). To eliminate the line connecting points marked by bars, set its width to zero. Parallel bars are drawn by assigning non-overlapping intervals to different series. Stacked bars or high-low graphs are produced with overlapping intervals. The table below shows some of the legal color names that are valid in either *G7* or *G6*. Color names may be capitalized or not, it doesn't matter. Where names have "light" or "dark" these can be abbreviated as "lt" or "dk". For example, "Lightgray" is the same as "ltgray".

Basic	Light/Dark	Others	
black	lightgray	olive	pumpkin
white	darkgray	lavender	seagreen
red	darkblue	forest	chocolate
green	darkred	aqua	slategray
blue	darkgreen	chartreuse	skyblue
yellow	lightblue	maroon	tomato
orange	lightred	fuchsia	teal

purple	lightgreen	azure	coral
brown	darkbrown	midnight	gold

Related topics: *Drawing graphs*, *Graph settings*.

lint <name>
or
lint <group>

Linear interpolation of missing values in a vector is done by the "lint" command. The "lint" command replaces missing values in the time series by linearly interpolated values. Zeroes before the first or after the last observation are not replaced. This command applies only to series in the default Vam file. Groups may be used in the <names> field to refer to sectors of the currently loaded vector.

Example:

```
lint pce1
```

This example loads the pce vector and then fills in the missing values in sector 1 by linear interpolation. We could similarly fill in the values for all the sectors in the present dynamic group by

```
load pce
lint :
```

or all the values for the static group Services by:

```
load pce
lint :Services
```

Entire matrices may be interpolated with one command. For example

```
lint am
```

will interpolate the entire am matrix. This interpolation also works for packed matrices. The "lint" command works on the entire range of the Vam file without regard to fdates.

Related topics: *group*.

linv [bank letter.]<matrix A> [period]

The "linv" command calculates the Leontief inverse and places it in the source matrix. It works over the current "fdates", or for the [period] specified on the command line.

For example

```
linv A 1995
```

replaces A with its Leontief inverse for 1995.

Related topics: Matrix operations, madd, mcopy, mtrans, mmult, minv, vc.

listbanks (lb)

This function provides information about the banks that are currently assigned in *G7*. Up to 26 banks of various types may be assigned, to positions 'a' through 'z'. The "lb" command reviews which banks are assigned to which position, and what type of bank they are.

Related topics: *Assigned banks*, *bank*, *cbk*, *dfr*, *hbk*, *vam*.

(listg)roups

List the names of all the groups currently in the GROUPS.BIN file.

Related topics: *group*.

(lis)tnames <bank_location> [wildcard]

Displays the series names in the workspace (w) or the assigned bank (a).

lis a

This command also has a "wildcard" option, which allows you to list variable names matching a certain pattern. For example:

lis a out*

will list all series in the assigned bank that start with out (such as out1 through out85)

Related topics: *listnamescol.*

listnamescol (lnc) <bank_location>

The *listnamescol* or *lnc* command works just like the *lis* command, but all series names in the workspace (w) or assigned bank (a) are output in one column. You may find it convenient to capture output to a file, and then use this command to get a list of all the series in the databank in that file. Then, using an editor that supports macros, you can change all lines to create an addfile that does the same thing with each series in the databank. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS. Option *s* sorts the series in alphabetical order, and *r*

reverses the order. If a Vam file is associated with the G bank, then option *g* prints only macro series and option *v* prints only Vam bank series.

Related topics: (lis)tnames, save

listvecs
lv [-sr] **[**<bank_location> [<wildcard>]]
lvc [-sr] [<bank_location> [<wildcard>]]

The *listvecs* or *lv* command works just like the *lis* command, but all vector and matrix names in the default vam file or specified bank are printed. The *lnc* is the same but prints names in one column. Option *s* sorts the series in alphabetical order, and *r* reverses the order. By default, the command will operate on the default vam bank, but other bank locations may be specified. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS.

load <vector_name>

When *G7* is working on a vector in a Vam file, it pulls the whole time series for the vector into a sort of vector workspace. Usually, this is done implicitly by simply referring to the vector or one of its elements. The load command enables the user to do this loading explicitly. It is used principally in connection with the index command described below.

Example:

```
load pce
```

Related topics: *store*.

look <bank_location>

This command from the *G7* console is equivalent to the Bank, Look menu command. As usual, <bank_location> is a letter between 'a' and 'z' to which you have assigned the bank you would like to look at. For "look" to work, there must be a stub file created with the same root name as the bank. For example, the stub file for the NIPAA.HBK, databank, which is a hashed bank, is NIPAA.STB.

The "look" command brings up the stub file in a scrolling list box, from which you can select lines of the file to print out and graph the various series in the bank. If you double-click on the series name, the series will be printed out using the current graph dates ("gdates"), and the title of the graph will be the line of the stub file. The series will also be printed to the *G7* console.

Related topics: *Dates in G7*, *gdates*, *graph*, *Stub files*, *tdates*.

ls [-<flag>] <x> <y> <date> [direction]

This *linkseries* command takes series x and y from the workspace or the assigned bank, calculates their ratio at the specified linking date. This ratio is then used to move 'x' with non-zero entries from series 'y'. The linking direction can be 'f' for forwards or 'b' for backwards. The default is 'f'. If the variable x is found in the workspace, then the result is stored with name x in the workspace. If a bank letter is provided for x, and if this bank is the default Vam file, then the modified value of x is stored in the Vam file. Otherwise, the result is stored to the workspace.

If the signs of the values in *x* and *y* are not the same, then the result will move in the opposite direction of the guide series; this usually is not desirable and a warning will be given. Three alternatives to the basic routine may be implemented by using a flag.

-f: the calculation using standard algorithm without reporting problems.

-i: if x and y are of opposite sign in the base year, then ensure positive correlation between x and y such that

x = a + b*y

b = x{date} / y{date} * sign(x{date}*y{date})

a = x{date} - b * y{date}

-a: if x and y are of opposite sign in the base year, and the problem is ill-conditioned such that *y* lies close to zero in the link year, then ensure positive correlation between x and y such that

x = a + b*y

b = x{date}/ AVG(y) * sign(x{date}*AVG(y))

a = x{date} - b * y{date}

where AVG(y) is calculated from fdate1 to <date>, if direction is 'b', or from <date> to fdate2, if direction is 'f'. This may be useful when y{date} is close to zero so that scaling parameter b is large, thus scaling x excessively. If AVG(y) is close to zero, then b = x{date} * sign(x{date}*AVG(y)). It is the user's responsibility to ensure that the results are useful.

Related topics: *@bmk function*

madd [bank letter.]<matrix A> = [bank letter.]<matrix B> + [bank letter.]<matrix C> [period]

The "madd" command performs addition or subtraction of two matrices, and stores the results in a third. If the optional [period] is absent, the operation is done over the "fdates" range. If no bank letter is specified, the matrices are assumed to be in the default Vam file

Example:

```
fdates 1972 2020
madd A = B + C
```

This example forms A as the sum of B and C, over the period 1972 to 2020, with all matrices in the default Vam file.

Example:

```
fdates 1972 2020
madd d.A = e.B – e.C
```

This example forms A in bank 'd' as the difference of B and C from bank 'e', over the period 1972 to 2020.

Related topics: *@bmk function*

(matd)ata
(matu)pdate

This command brings data into *G7* in a rectangular format, which could easily be prepared using Excel or 123. There are three alternative forms in which the series can be arranged. The series can be arranged either in vertical columns with the name of the series at the top of the column, or in horizontal rows with the names appear on the same line of the "matdat" command. Up to 20 columns of numbers occupying up to 160 characters per line may be given. The "matdata" command is used to provide data for a new series. If the series is already present in the databank, it overwrites it. The "matupdate" command assumes that the data is already present in the workspace bank, and merges the new data with the existing series. The three forms of the command are:

Form 1:

```
matdat
             gnp      c        cd       cnd      cs
    81.1    1513    950      146      359      445
    81.2    1512    949      140      361      448
    81.3    1522    956      143      361      450 ;
```

Dates appear as the first number on each line. Here 1513 is the value of "gnp" in 1981.1, 1522 is the value in 1981.2, etc.

Form 2:

```
matdat 81.1
    gnp      c        cd       cnd      cs
    1513    950      146      359      445
    1512    949      140      361      448
    1522    956      143      361      450 ;
```

The date of the first observation appears on the command line, and no dates appear on subsequent lines. Use a semicolon to end the data. (You must have a semi-colon on the last

line of data for either form.)

Here is yet another version of legal format for “matdata”.

Form 3:

```
matdat  gnp out(1-4) 81.1
    1513    1512    1522
    950     949     956
    359     140     143
    359     361     361
    445     448     450;
```

Here, series names are given immediately after "matdat" with starting date given as the last argument on the "matdat" line. The first series reads from the first line after the "matdat", the second series reads from the second line, and so on. Note also that group definitions are allowed in this version.

matin <matrix_name> <year> <firstrow> <lastrow> <firstcol> <lastcol> [<skip>] [form]

The “matin” command is a command for reading matrix data for one year into a Vam file. For it to work, a Vam file must already be assigned and set as the default Vam file. There should then follow a rectangular array of numbers matching the <firstrow>, <lastrow>, <firstcol>, and <lastcol> parameters of the command. As in the “vmatdata” command, the optional <skip> parameter is the number of spaces to be skipped in reading each line. The <skip> parameter and the form line work together exactly as for “vmatdata”: the absence of a skip indicates the presence of a form line. A ‘#’ in the first space of a line means to skip the whole line. Use of "matin" does not affect entries outside the specified area, so it can be used to update a matrix as well as to introduce it originally.

Example:

```
matin govstr 1987 1 25 1 5 4
   1     48      0      0      0     61
   2    495      0     32      0    453
   3      0      0      5      0      0
   4   1344      0      0      0   1651
   5    812      0    201      0      0
   6    859    555      0      0   3012
                .
                .
```

The above example is actually used to introduce the distribution of government structures by 25 categories to 5 government types in *IdLift*. The data presented is for 1987, and covers rows 1 to 25 and columns 1 to 5 of the matrix. Four columns are to be *skipped* at the beginning of each line, and this is where we keep the row number, which serves to make the file easier to read.

Related topics: *vmatdata*, *matin5*, *pmatin*, *pmatin5*.

matin5 <matrix_name> <year>[<width> <decs> <IndexWidth>]

The "matin5" command is another command for reading in data for a matrix. This data is in a format called "punch5". This is an input format for matrices that is useful when a significant number of the matrix elements may be zero. On each line, there are 5 cells (hence the name), and for each cell, the row, column and value are specified. For this command to work a Vam file must be assigned and set as the default Vam file. In the above syntax, <matrix_name> is the name of the matrix as shown in VAM.CFG, <year> is the year of the data, <width>, <decs> and <IndexWidth> are optional arguments that specify the width of the data field for each cell, the number of decimal points, and the width of the field for the row and column indexes. The end of the matrix data is signaled either by a ';' in the first space on a line or by the end of the file.

Here is a sample from the beginning of an A-matrix:

```
matin5  am 1977 9 5 3
# A-matrix for 1977. 83 rows and 83 columns.
AM     1  1 0.245790  1  4 0.000220  1  8 0.000410  1  9 0.281300  1 10 0.058360
AM     1 11 0.002070  1 12 0.005680  1 13 0.000380  1 15 0.001700  1 16 0.003060
AM     1 22 0.089770  1 24 0.000070  1 48 0.001210  1 49 0.000130  1 50 0.000030
```

Related topics: *matin*, *pmatin*, *pmatin5*.

matty | matpr [<file_name>] <date1> <date2> [<option>]
<series1> <series2> <series3> .... <seriesN> ;

The "matpr", or "matty" command creates or overwrites a file of data in parallel columns for input into spreadsheets or other software. The <file_name> is the name of the file created, and <date1> and <date2> are the starting and ending dates of the transfer. The only option currently available the "dump" option, which prints data in a compact format. *G7* responds to this command with "Give series names >." Type in the names of the series and terminate with a carriage return. Up to 20 series may be transferred in one file. If no filename is given, the data is written to the screen and the "save" file, if one is currently open. Width and decimal points are set by the current values given by either the "format" command or the "width" and "decs" commands. The year format is given by the "yearformat (yf)" command.

Example:

```
#  MATTY.TST -- Test the output of the "matty" command.

hbk quip a
matty nipa.prn 1985.1 2000.2
gnp c v g;
```

Related topics: p123, rp123, Writing data from *G7*

maxobs <number>

This command sets the maximum number of observations that can be in any series in the workspace bank.

Example:

```
maxobs 2000
```

allows series with up to 2000 observations. The default value of maxobs is 600. Increasing the value slows down most data operations, and may reduce the number of variables that can be included in any regression. If long series are to be stored in the workspace bank, the maximum number of observations in this bank should also be increased. This number is set at the end of the G.CFG file.

Note that you may delete the workspace bank and set all new parameters all in one fell swoop with the "zap" command.

Related topics: *G.CFG file*, *zap*

mcopy [bank letter.]<destination> = [bank letter.]<source> [<period>]

The "mcopy" command copies the matrix (or vector) from the source to the matrix (or vector) destination. If no bank letters are specified, then both the source and destination are assumed to be in the default Vam file. If a value is given for [period] then only the values for that period are copied. Otherwise, the starting period and ending period are determined by the "fdates" command. An '=' may be included for clarity, but is not required.

Example:

```
vam \idlift\devel\hist b
vam \idlift\model\hist a
fdates 1972 2040
mcopy a.am = b.am
```

This example copies the matrix "am" from the *IdLift* development Vam file to the production Vam file, for the period 1972 to 2040.

Note that the "vc" command will also copy vectors within the same bank, and the "vc" also is capable of evaluating certain algebraic equations involving matrices.

Related topics: *Matrix operations, madd, mmult, mtrans, minv, linv, vc*

(mgr)aph <series1> <series2> [<gdate1>] [<gdate2>] [<gdate3>]
(mpl)ot

The "mgraph" command causes a multi-scale graph to be displayed. This is most useful with

two series, since the left axis scale will be for the first series, and the right axis scale for the second. If any more than two series are displayed, the other series will also be plotted on their own scale, but that scale will not be displayed.

Related topics: *Drawing graphs*

minv [bank letter.]<matrix A> [period]

The "minv" command performs in-place matrix inversion. It works over the current "fdates", or for the [period] specified on the command line.

Related topics: Matrix operations, madd, mcopy, mtrans, mmult, linv, vc.

missing <y | n> [<missing_text>]

This command does two things. First it tells *G7* whether or not to rely on missing values to be present when performing regressions, interpolations or other functions. When missing is set to "y", then *G7* will complain if missing values are in any of the data in a regression. Functions such as @benchmark and @lint rely on missing values to tell which periods of a series need to be filled in. If missing is "n", then *G7* treats zeroes as missing values, and the regression command will not complain if there are zeroes.

The second function of the "missing" command is to supply some text to be displayed when missing values are present. A missing value is equal to -.0000001, so by default it will display as -0.0. However, using the "missing" command, you can tell *G7* to display missing values as "miss", "N/A", or some other character string. Just make sure that the missing_text> is short enough to line up well with the other data being displayed.

You can convert the zeroes in a series to missing values by using the @miss() function. Conversely, you can convert missing values to zeroes using the @zeroes function.

Related topics: *Functions in G7*, *Missing values.*

mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>*[bank letter.]<matrix C> [period]
mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>'[bank letter.]<matrix C> [period]
mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>/[bank letter.]<matrix C> [period]
mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>&[bank letter.]<matrix C> [period]

The "mmult" command is actually a family of commands, with four operations available:

1. The asterisk (*) operator is matrix multiplication. If the matrices are conformable, then

```
mmult A = B*C
```

multiples matrix B by matrix C and stores the result in matrix A. If a date is specified on the command line, the operation will be done for that period only. Otherwise, it is performed over the current "fdates".

2. The apostrophe (') operator does a transpose of the first matrix, and then does

multiplication. If the matrices are conformable, then

```
mmult d.A = e.B'e.C
```

multiples the transpose of matrix B in bank 'e' by matrix C in bank 'e' and stores the result in matrix A in bank 'd'.

3. The slash (/) operator does element-by-element division. So,

```
mmult A = B / C
```

divides each element of B by the corresponding element in C, and places the result in A.

4. The ampersand (&) operator does element-by-element multiplication. So,

```
mmult A = B & C
```

multiplies each element of B by the corresponding element in C, and places the result in A.

Related topics: Matrix operations, madd, mcopy, mtrans, minv, linv, vc.

mode <test | forecast> or mode <t | f>

Determines what is to be done between <rdate2> and <rdate3>. The default mode is test. In that case, the values "predicted" by the equation for the period from <rdate2> to <rdate3> are compared with the actual data and a SEE and MAPE computed for the test period. For forecast mode to work, the series for all independent variables except lagged values of the independent variable must extend to <rdate3>. Then, in forecast mode, two forecasts are made for the period from <rdate2> to <rdate3>. The first is simply the predicted value; the second, the predicted value plus the "rho adjustment" to take account of the error in the last period of fit.

Related topics: *(lim)its*, *ordinary regression*, *(r)egress*.

monup(mup) <series_name>

This command is used for updating a quarterly series with monthly data. Example:

```
mup rtb
  83.3  9.120 9.390 9.050   8.710 8.710 8.960
  84.1  8.930 9.030 9.440;
```

"rtb" is a quarterly series being updated with monthly data. 9.120 is the value of rtb in July 1983; 9.390 is the value in August 1983, etc.. Note that quarterly dates are the first numbers on each line and three monthly observations must be present for each quarter.

move <matrix> <up|down|left|right> <first> <last> <newfirst> [year]

This command moves a block a rows up or down, or moves a block of columns left or right.

where:
<matrix> is the name of a matrix whose rows or columns are to be moved.
<up|down|left|right> indicates the direction of movement
<first> indicates the first row or column of the block that is to be moved.
<last> indicates the last row or column of the block that is to be moved.
<newfirst> indicates the first row or column to which the block is to be moved.
[year] specifies a single year. If not specified, the function will do the move for all years between the active fdates.

Example:

```
move fact left 2 4 1 1997
```

In this example, columns 2 through 4 are moved left, and overwrite columns 1 through 3. Column 4 will be zero.

Related topics: *flow*, *coef, rdras*.

mtrans [bank letter.]<matrix A> [bank letter.]<matrix B> [period]

The "mtrans" command takes the transpose of the 2nd matrix (B) and places it in the first matrix (A). The two matrices can be in different Vam files. If no bank letters are specified, they are assumed to be in the default Vam file.

```
mtrans d.A e.B
```

sets A in bank 'd' to the transpose of B in bank 'e'.

Related topics: Matrix operations, madd, mcopy, mmult, minv, linv, vc.

nl or nlp <y> = <non-linear function involving n parameters, a0, a1, ...an-1>
n
<starting values of the parameters>
<initial variations>

G7 offers two algorithms for non-linear regression. The "nl" command uses the simplex method (sketched below and not to be confused with the simplex method of linear programming) and the Powell method. The form for the two is exactly the same except that the first is invoked by the "nl" command, while the second is invoked by the "nlp" command. We illustrate the with "nl" here.

Example:

```
nl y = @exp(a0*@log(a1 + a2*x))
3
 1.5  25  2.0
  .1   1   .05
```

Since the non-linear function must be evaluated repeatedly, anything the user can do to speed

the evaluation is helpful. For example, put all the variables involved into the workspace; make the workspace no larger than necessary; put the workspace on a *ram disk*; take any functions of variables, such as logs or squares, which do not change through the calculations and put the results in the workspace and use them instead of, say, taking the log over and over.

In the simplex algorithm, S, the sum of squared errors, is initially calculated at the initial value of each parameter. Then, one-by-one, the parameters are changed by adding the initial variations and S is recalculated at each point, thus yielding values at n+1 points (a simplex). By the algorithm sketched below, points in the simplex are replaced by better points or the simplex is shrunk towards its best point. The process continues until no point differs from the best point by more than one-tenth of the initial variation in any parameter. Each time a replacement is done, the simplex and the operation that yield it is reported on screen. Like all non-linear algorithms, this one is not guaranteed to work on all problems. It has, however, certain advantages. It is easily understood, no derivatives are required, the programming is easy, the process never forgets the best point it has found so far, and the process either converges or goes on improving forever. The display show the regression coefficients, their standard errors, t-values, and variance-covariance matrix. The Powell method also requires no derivatives and has the property of "quadratic convergence." That is, applied to a quadratic form, it will find the minimum in a finite number of steps.

Following the "nl" command, there can be f commands, r commands, and con commands before the non-linear equation itself. These may contain the parameters a1, a2, etc.; they should each be terminated by ';'. The non-linear search then includes the execution of these lines. E.g.:

```
nl f x1 = @cum(s,v,a0);
y = a1 + a2*x1
```

If the name of the left side variable is "zero", no squaring of the difference between the left and right side will be done. Instead, the squaring must be done by the @sq() function. This feature allows soft constraints to be built into the objective function. For example,

```
nl zero = @sq(y - ( a0 + a1*x1 + a2*x2)) + 100*@sq(@pos(-a2))
```

will "softly" require a2 to be positive in the otherwise linear regression of y on x1 and x2.

If the name of the left-side variable is "last", no summing will be performed. Instead, the value of the function on the right at the "last" period will be minimized. This "last" period is the one specified by the second date on the preceding "limits" command. In order to combine summing of some components of the function on the right with the use of the "last" option, the @sum() function is used. It places the sum of the elements from the first date to the second date in the second date position of the output series, which is otherwise zero. This device is useful in some maximum liklihood calculations.

Both the simplex method and Powell's method are described in the book, *Numerical Recipes in C*, by Wm. H. Press, et al., Cambridge University Press.

Sketch of Simplex Method of Non-linear Regression
The method begins with a point in the parameter space of, say, n dimensions and step sizes for each parameter. Initially, new points are found by taking one step in each direction. Actually, each step is made both forwards and backwards, and the better of the two positions is chosen. This initial operation gives n+1 points, the initial point and n others. These n+1 points in n dimensions are known as a simplex. The algorithm then goes as follows.

Reflect the old worst point, W, through mid-point, M, of the other points to R (reflected). (R is on the straight

If R is better than old best, B {
expand to E by taking another step of length MR in the same direction.

if E is better than R,
replace W by E in the simplex.
else
replace W by R. (reflected)
}
Else if R is better than W {
replace W by R in the simplex. (reflected)

}
Else {
contract W half way to mid-point of other points, to C. (contracted)
if C is better than W,
replace W by C.
Else
Shrink all points except B half way towards B. (shrunk)
}

(norm)ality <y | n | f>
This command turns on (or off) tests of normality of the error terms. If the option chosen is 'y', then the Jarque-Bera test appears on the standard regression result screen labeled "JarqBer". Under the hypothesis of normality, it has a chi-square distribution with two degrees of freedom.

If the option 'f' (for full) is chosen, then before the regression results are shown, a table appears with moments and measures of symmetry and peakedness, as well as the Jarque-Bera statistics. If a "catch" file active, this table will go into it.

Related topics: *Regression tests*

p123 <file_name> <date1> <date2> [width] [decs]

<ser1> [<ser2> <ser3> ... <sern>] ;

or

p123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> ... <sern>] ;
This command is similar to the "matty" command, except that the data is written directly to a LOTUS worksheet. Do not include the WK1 extension with the filename; *G7* automatically appends the extension WK1. <date1> and <date2> are the desired starting and ending dates. Up to 239 series names can be entered in free format. End the list with a ';'. Width and decs are optional; width specifies the display width within LOTUS for all series transferred, decs specifies the number of decimals displayed. Defaults are 9 and 3 respectively. p123 worksheets are compatible with LOTUS 1-2-3, Excel and many other programs.

Example 1:

```
p123 natacct 60.1 91.1 8 1
   gnp c v vf vi fe fi g
```

Example 2:

```
p123 natacct 60.1 91.1 8 1 out(1-85)
```

Related topics: *Writing WK1 files.*

pch <name> <number>
The "pch" command is an older form of the "ipch" command, and is used for generating equation results for SlimForp style models. As with "ipch", first an equation file is opened with the "punch" command, then, after doing a reression, use "pch", and it will write out some lines to the equation file in the format expected by SlimForp.

Related topics: *ipch*, *punch*.

pmatin <matrix_name> <packed_matrix_filename>
The "pmatin" command is a command for introducing data for *packed matrices* into a Vam file. A Vam file must have already been assigned and set as the default Vam file. A packed matrix is a matrix with all zero elements removed, and the format is efficient for presenting the nonzero cells of the matrix. The packed matrix resides outside of the Vam file itself, and the Vam file contains only a pointer to the filename.

In the above syntax, <matrix_name> is the name of a matrix in the VAM.CFG file, which must have the letter p in the "lags" position. The <packed_matrix_filename> is name the DOS name of the packed matrix file (these files always have the .PMX extension, but the .PMX should be included in the name.) The format of the data which follows is then:

<year>
<row> <num_non-zero_elements>
<col1> <element1> <col2> <element2> ...

For example:

```
pmatin bm bm.pmx
1985
      1  5  # row 1 has 5 non-zero elements
      1  .656  7 .352  11 .038      23 .019  27 .218
      3  2  # row 3 has 2 non-zero elements
      7  .098  51 .923
      etc.
```

If the matrix is available for several years, the data for all years should be introduced with only one "pmatin" command, with each year's data preceded by the number of the year. It is essential that the year numbers be in 4-digit format, i.e. use 1987 please, not 87. If a cell is non-zero in any year for which there is data, the packed matrix will have a place reserved for it in all years.

Related topics: *pmatin1*, *pmfile*, *VAM.CFG file*.

pmatin1 <matrix_name> <packed_matrix_filename>

An alternative way of introducing data into packed matrices is the "pmatin1" command. The format is just like that of the "pmatin" command, except that the data should be arranged in rectangular rows and columns like the "matin" command. For example, if you wish to treat as a packed matrix a 3 x 3 matrix known as B in the Vam file, you can introduce it with the "pmatin1" command as follows:

```
pmatin1 B B.pmx
1995
    1  0  0
    5  0  1
    6  2  0
1996
    1.5 0  0
    6   0  2
    7   3  1
```

Note that in the above example, only those cells that are zero in all years will be left out of the packed matrix. Cell (3,3) will actually be stored, since it has a non-zero value in 1996. As with the "pmatin" command, you should put the data for all years you want to read in the same file, and use only one "pmatin1" command per matrix.

Related topics: *pmfile*, *pmatin*.

pmfile <matrix_name> <packed_matrix_filename>

The "pmfile" command associates a matrix name with a particular packed matrix file. This is useful when you need more than one versin of a matrix. For example, one copy might hold coefficients, and the other copy hold flows. Or, suppose we wished to study the effects of changes in the A matrix and the A matrix was packed, then we would need two different .PMX files. If the original was called AM.PMX, then to create the alternative, say, AMALT.PMX, we would go to the DOS prompt via File | DOS and do:

```
copy am.pmx amalt.pmx
copy hist.vam vamalt.vam
```

Then from *G7*'s command box do:

```
vam vamalt b
dvam b
pmfile am amalt.pmx
```

This "pmfile" command will both assign AMALT.PMX to be used whenever the am matrix is referred to, and put AMALT.PMX into AMALT.VAM as the name of the file to be used for the packed am matrix.

Related topics: *pmatin*, *pmatin1*, *vam*, *dvam*.

pmpunch <filename> <matname> [decs]

The non-zero elements of the matrix <matname> are written into the named file as input for the "pmatin" command. The [decs] argument gives the number of decimal places. The years written are determined by the current values of "fdates". This command is especially useful for converting a rectangular matrix in a Vam file into a packed matrix. One writes it out as an ASCII file with this command and then reads it in with the "pmatin" command.

Related topics: *punch*, *pmatin*.

qpcon b <sign> [constant][*]ai [< > [constant][*] aj …]

Where *sign* is <, =, or > (<= and >= also are accepted but are recorded as strict inequality constraints). *constant* are optional scalar multiples for the respective parameters. An asterisk may be included to clarify that the parameters are multiplied by the corresponding constant. The parameters are denoted by *ai*, where *i* is the position in the regression equation of the corresponding variable. If a constant is included in the regression, then the parameters are denoted as *a1*, *a2*,… Right-hand-side terms are separated by either a + or -.

Examples:

```
qpcon 25.0 > a1          # constrain the constant
qpcon 13.0 = 1*a2 + 1*a3 # constrain the sum of slope parameters
```

Related Topics: Soft Constraints

punch <punchfile | "off">

The "punch command is used to open up equation files for writing using the "ipch" command. These files usually have the extension .eqn, and are used in *Interdyme* models to construct an Equation object, which contains regression parameters, values of rho, equation types and other information for multisectoral regression equations. To close the file, use "punch off".

The “punch” command is usally given at the beginning of a large regression add file. It is often useful to use the "fadd" command, along with an argument file that contains sector numbers, titles, equation types, starting dates and other information that changes with each sector. Within the body of the "fadd", a "(r)egress" command will be given followed by an "ipch" command. This will estimate the regression, and put the equation results into the equation file.

Related topics: *fadd*, *Interdyme*, *ipch*, *(r)egress*.

punch5 <filename> <vecname> [<width> <decs><IndexWidth>]
p5

This command prints out a matrix or packed matrix to a file in “punch5" format. The years that are written are determined by the current value of fdates. Each year is written with a header that is a “matin5" command, which tells “matin5" also what field width, decs and IndexWidth to use. (The IndexWidth parameter is the width of the row and column numbers in the file.) Thus, a file created with “punch5" can be read back into Vam using the “matin5” command. This provides a convenient way to convert packed matrices into normal matrices. The “punch5" format prints 5 non-zero matrix cells to a line, with row number, column number and cell value for each cell.

Related topics: *matin5*, *fdates*.

punchvec <filename> <vecname> <startyear> <endyear> [<width> <decs>]
pv

The “punchvec” command is used to print out a vector to a file for all sectors, for the years specified. The optional <width> and <decs> arguments again specify the field width, and number of decimal places. The output file starts with a number of comment lines, telling the name of the vector, starting and ending years, and format information. Then one comment line gives the years as column headings. Each following line of the file contains the time series for one sector, with the name of the series, followed by the time series of data.

pwd

The “pwd” command prints the path of the current working directory.

Related topics: *cd*

quit

This command quits the execution of *G7*. Note that if any editor windows are open, they will be closed. If you have any unsaved changes in these files, *G7* will ask you if you want to save the file. Also, before exiting *G7* will write out one more copy of the current workspace bank index, to make sure it is updated.

ras [-f] <matrix> [r <group_exp>] [c <group_exp>] <rowcon> <colcon> [year] <r | c> [cnt]

Balance the named matrix by the *rAs* method so that matrix will have the row sum of <rowcon> and the column sum of <colcon>. It does not actually matter whether <rowcon> and <colcon> are columns or rows, though in the above equation they are to be thought of as columns. If the [year] parameter is missing, the command works over the current "fdates" range. If the sum of the elements of <colcon> does not equal the sum of the elements of <rowcon>, the vector to govern is specified by the 'r' or 'c' at the end of the command. The other vector will be scaled to have the right total. The rows will then be scaled to get the correct row totals, then the columns scaled to get the correct column totals. Every five iterations, there is a report on the row and column in which the maximum scaling occurs. When the maximum scale factor differs from 1.0 by less than .00002, convergence is declared to have been reached. The last scaling will be of the rows. The [cnt] parameter at the end determines maximum number of iterations. If no [cnt] is specified, the default is 100. If convergence does not occur after the maximum number of iterations, the program reports the problem and continues to the next command. Once the balance is achieved, the columns are scaled to sum to 1.0. The reason for this normalizing and how to deal with it if it is not wanted is explained below.

The optional arguments [r <group_exp>] and [c <group_exp>] allow you to focus on doing a ras on a submatrix, where <group_exp> is a legal group expression. If you specifiy the 'r', the group expression will determine which rows of the matrix are scaled. If you specify the 'c', the group expression will determine which columns of the matrix are scaled.

In our experience, the matrices most frequently in need of balancing are the distribution matrices, such as those that determine how personal consumption expenditure by budget category is to be distributed to industries. For this reason, the "ras" command, after balancing, automatically insures that the column sums are 1.0. Of course, if the matrix is the intermediate coefficient matrix, one does not want the columns to sum to 1.0. What do we do then? Let us suppose that am is an initial coefficient matrix, q is the vector of outputs, cs is the vector of column sum controls and rs is the vector of row sum controls. Then we would do:

```
flow am q
coef am cs
ras am rs cs r
flow am cs
coef am q
```

Note that the scaling algorithm used by the "ras" command is simple scaling. If the control

total is *C* and the vector is **X**, then the scaling factor *S* is applied to each element, as is calculated as:

$$S = \frac{C}{\sum X_i}$$

Related topics: *flow*, *coef, rdras, psras*.

rdras [-f] <matrix> [r <group_exp>] [c <group_exp>] <rowcon> <colcon> [year] <r | c> [cnt]
This command works identically to the "ras" command, except in the case where some of the matrix cells are negative. In this case, it performs *right-direction scaling* on the rows and columns. The essence of right-direction scaling is that both positive and negative numbers are scaled in the same direction. A scaling factor *s* is found such that when multiplying the positive numbers by *s* and dividing the negative numbers by *s*, the numbers add to the correct total. This algorithm is also used when applying group fixes in *Interdyme*.

Note that the right-direction scaling is a bit more complicated than the simple scaling technique used by the "ras" command. Right direction scaling preservers the relative ordering of the elements, and scales them all in the same direction, the "right" direction. Assume that *A* is the sum of all positive elements of the vector **X**, *B* is the sum of all negative elements, and *C* is the control total. The problem is formulated so as to find the number *S* for which the following equation is true:

$$AS + B/S = C$$

This can be rewritten as the quadratic equation in *S*:

$$AS^2 - CS + B = 0$$

To solve for *S*, we use the general quadratic formula to obtain:

$$S = \frac{C + \sqrt{C^2 - 4AB}}{2A}$$

See the *Interdyme Report #3* /by Douglas Meade for a more detailed description (http://www.inforum.umd.edu/WorkPaper/INFORUM/InterdymeReports/Report3.pdf)

Related topics: *ras, psras*.

psras [-f] <matrix> [r <group_exp>] [c <group_exp>] <rowcon> <colcon> [year] <r | c> [cnt]
This is anoher form of the "ras" command. However, in the case of where some of the matrix

cells are negative, the proportional scaling changes all elements in the same proportion.

The procedure can be seen most easily by writing down what the **X** vector is after scaling, call this $\mathbf{X}^*$. This is determined in proportional scaling by the following equation:

$$X_i^* = X_i + |X_i| \left(\frac{C - \sum X_i}{\sum |X_i|} \right)$$

The intuition behind this scaling is that it preserves the proportion between the difference of the old and new valuesand the final scaled values. One of its strengths is that it is able to provide a result when for example, all the elements of **X** are negative, yet the control total is positive.

.

(rec)ursive <y> = <x1>, <x2>, <x3>, ..., <xn>
or
rec <y> = ! <x1>, <x2>, <x3>, ..., <xn>

This command is used to perform recursive OLS regression. What this means is that if we have set up limits as:

```
limits <date1> <date2> <date3>
```

Then the indicated regression will be done from date2 to date3, then from date2-1(the period before date2) to date3, then from date2-2 to date3, and so on back to date1 to date3. The regression coefficients are made into time series and stored in the workspace. b1 is the series for the first regression coefficient; b2, for the second; and so on. The regression coefficients are stored in the date corresponding to the first date in their regression. Thus, b1{date1} would be b1 for the regression over the period date1 to date3. Similarly, the standard errors of the regression coefficients are stored in s1, s2, s3, etc.

If a "zip" command preceeds the "recur" command, no regressions will be displayed, but the zip command will be turned off at the end of the "recur" command, for you will surely want to do graphs after the "recur" and they cannot be done with zip on.

Example:

```
zip
limit 80.1 85.1 91.2
recur gnp$ = g$,v$,fe$,fi$
gdates 80.1 85.1
fadd bounds (1 - 5)
```

where "bounds" is the file

```
ti Estimate and 2-Sigma Limits for Coefficient %1
```

```
f upper = b%1 + 2*s%1
f lower = b%1 - 2*s%1
gr b%1 upper lower
```

Related topics: *regress*, *zip*, *limits*.

(r)egress <y> = <x1>, <x2>, <x3>, ..., <xn>
or
(r)egress <y> = ! <x1>, <x2>, <x3>, ..., <xn>

Both forms of the "r" command regress the dependent variable <y> on the independent variables <x1>, ..., <xn>. The first form automatically supplies a constant term, the second does not. The total number of variables, counting the constant, must not exceed 30. Independent variables may be expressions, as on the right side of an "f" command. The dependent variable should be a single variable. If the line ends with a ',' the command continues on the next line.

If lagged values of the dependent variable occur among the explanatory variables, the are automatically supplied from previously calculated values when forecasting. Regression coefficients and other values are stored in the 'rcoef' variable in the workspace bank.

Related topics: *Ordinary regression.*

(ren)ame <old_series_name> <new_series_name>

This command is usually used to rename a series in the workspace bank to another name. For example, it is used in the example on 2SLS to rename to predicted values from the first stage regression which will be used on the right hand side in the second stage regression.

Related topics: *del*, *2SLS and 3SLS*.

(rows)cale <matrix> <row> [y|n]

This command is another method for converting flow matrices to coefficients. <matrix> is the name of a matrix whose rows are to be multiplied by the elements of the vector <row>. The last argument indicates whether or not to skip the diagonal elements. The default for this argument is 'n'. Note that this command is different from the "flow" command, as each element of the vector multiplies all elements in the corresponding *row* of the matrix, instead of the corresponding column. This command could thus be used to implement across-the-row coefficient change, with the coefficient change indicator vector in <row>. The command takes effect over the current fdates.

Example:

```
rowscale am ami y
```

In this example, **am** is the A-matrix, **ami** is a vector which indicates the time path of

coefficient change, and which is equal to 1 in the base year of the A-matrix. In the current example, we are requesting not to have the diagonal elements moved.

Related topics: *flow, coef, index*.

rp123 <file_name> <date1> <date2> [width] [decs]
<ser1> [<ser2> <ser3> ... <sern>] ;
or
rp123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> ... <sern>] ;

The format of the "rp123" command is identical to that of the "p123" command. The only difference is that the "rp123" command writes the series to the WK1 file by row. In other words, rows are series, and columns are dates. If you have very many series, with not so many observations (<250), "rp123" is better than "p123".

Related topics: *p123*, *Writing WK1 files*.

save < savefile | "off"> [<datacommand>]

The "save" command opens a file of the name <savefile> to receive output produced by many of the command of *G7*, including "type", "f", "fex", "id", "cc", "(r)egress", "matty", and other commands. The save file is closed and saving is terminated by:

```
save off
```

The default flag "-w" causes a new file to be created. The optional "-a" flag causes G7 to open an existing file and append text to the end. The save file represents a crucial step in the procedure of model building using the *Build* program. Normally, the procedure is as follows. One or more .REG (regression) files may be "add"ed during a *G7* session to estimate the equations of a model. Each .REG file will include a "save" command, to save the regression results, as well as the steps of forming variables, using the various flavors of the "f" command. These files are usually given the file extension .SAV. Then the .SAV files are "add"ed by "add" commands in *Build*, or they may included in an *Interdyme* model using *IdBuild*'s "iadd" command.

Note that the "save" command is also a powerful tool for using *G7* to create data files. You can use the *G7* "format" command to control the width, decimal points, and observations per line of data series typed using either the "type" or "stype" commands.

The optional <datacommand> argument specifies how series will be printed out when using "type". The default is for *G7* to create an "update" command, to use for updating series in *G7*. However, other legal values of <datacommand> are:

"data" - write out "data" cards
"ovr" - write out "ovr" fixes

"cta" - write out "cta" fixes
"mul" - write out "mul" fixes
"ind" - write out "ind" fixes
"gro" - write out "gro" fixes
"stp" - write out "stp" fixes
"vdata" - write out "vdata" cards for Vam
"vupdate" - write out "vupdate" cards for Vam
"fix" - write out LIFT style index fixes.
"dump" - write out data in compact format.

Since Vam likes 4-digit dates, you can specify "yf 4" before using "save" to print out "vdata" cards. Also, remember that you can insert comments into the save file, using the "ic" command.

The "ovr", "cta", "mul", "ind", "gro" and "stp" options above are for writing out macro or vector fixes for *Interdyme* models. Remember that if you write out a vector fix, you need to hand-edit the file and change names like "out5" to "out 5".

Related topics: *Build*, *format*, *ic*, *Interdyme*, *Model building*, *stype*, *yearformat(yf)*.

scale <control> <vectorname> [<group>] [year] [rdscale]
scale <control> <matrixname> <r rnum> [colgroup] [year] [rdscale]
scale <control> <matrixname> <c cnum> [rowgroup] [year] [rdscale]

This command can scale a vector, or scale a row or column of a matrix.
where:
<control> is a series from any bank (a.ctrl) or from any vector <b.vec2) or from any matrix(c.mat1.1) that contain the value to which the vector or matrix row or column is to be scaled.
For a vector:
<vectorname> is the vector whose elements are to be scaled.
Optionally:
[group] specifies which elements of the vector are to be scaled.
[year] specifies a single year for which the elements of the vector are to be scaled. If not specified, the operation will be done over the current fdates.
[rdscale] Giving an argument of 'r' specifies that right direction scaling is to be used.

For a matrix:
<matrixname> is the name of the matrix whose row or column is to be scaled.
<r rnum> or <c cnum> specifies that a row or column is to be scaled and gives the number of the row or column.
Optionally:
[colgroup] or [rowgroup] specifies which elements of the column or row are to be scaled.
[year] and [rdscale] are as before.

Example:

```
scale eqi2 bmf c 2
```

This example scales column 2 of the matrix **bm** to sum to the control *eqi2*, for all years in the current fdates.

Example:

```
scale cont mat r 5 1-10 2000
```

This example scales row 5, elements 1 to 10 of the matrix **mat** to sum to the control *cont* for the year 2000 only.

Related topics: flow, coef, rowscale, ctrl

second <on | off>

This command is used in estimating systemic two-stage least squares. This procedure is simple. Estimate the equations of the model by OLS or other single-equation method. "Build" the model with these equations and simulate it over the historical period. The simulated values of the endogenous variables are not influenced by the errors in the structural equations. They may therefore be used as regressors in the second stage to obtain estimates free of simultaneous equation bias. There are two variants of this procedure. In the first, the simulation is done using the simulated values for lagged values of endogenous variables; the other uses the actual values of these variables.

To perform the first variant, perform the simulation without any special command, copy the bank of simulation results to the *G7* workspace, start *G7*, select the b option on the opening menu, do "bank bws" to put the actual values in the assigned bank, give the command "second on" so that f commands are ignored, and re-estimate the equations with the same add files as originally used. With "second" on, the dependent variable of the regression will be taken from the assigned bank; all others come from the workspace, which contains the simulated values. To turn off the "second" feature, the command is "second off".

Related topics: *Build*, *Model building*.

seed(<integer>)

The “seed” command allows the user to reinitialize the random number generator. The random number generator is used with the @rand() and @normal() functions. The @rand() function draws from the uniform (0,1) distribution, and the @normal() function draws from the normal(0,1) distribution. If the “seed” command is called with identical arguments between calls to @rand() or @normal(), then these random number generators should return identical series.

Example:

```
seed(20707)
f uni1 = @rand()
seed(20707)
f uni2 = @rand()
```

In this case, the series *uni1* and *uni2* will contain identical elements.

Related topics: Functions, particularly the @rand() and @normal().

(sgr)aph <series1> <series2> [<date1>] [<date2>]
(spl)ot

The "scatter graph" command, "sgraph", allows two series to be plotted against one another. Each point on the two-dimensional graph measures the value of the first variable on the vertical axis, and the value of the second variable on the horizontal axis. For use only with the "sgraph" command, there is an "hrange" command, which works just like the "vrange" command, except for the horizontal axis. A common use of the "sgraph" command would be to show a Phillips Curve, as in the following example:

```
vr 0 2 4 6 8 10 12
hr 4 6 8 10 12
sgr infl unempc
```

Related topics: *hrange*, *Drawing graphs*.

(sh)ow <bank letter>.<vector_name>
or
(sh)ow <bank letter>.<matrix_name> < view> <line>

The "show" command shows vectors and matrices in a grid similar to a spreadsheet. For matrices, the format is the simpler one listed first. The values of the vector in successive years will appear as columns; dates run across the top and sector numbers down the side. The Options menu item allows the user to control the number of decimal places, fonts, and colors of the display. The display can be copied to the clipboard in the usual way for Windows programs. The contents of the clipboard can then be copied into a spreadsheet such as Lotus 1-2-3 or into a table in a word processor. On the Copy command, you get to specify how much you want copied — just the numbers or also the frame.

It is also possible to go in the other direction, from the spread sheet into the show window and thus into the Vam file.

For matrices, the "show" command has second format

The "view" argument must be one of the following:

r	for row
c	for column
y	for year.

If view is 'r', then <line> is the number of the row to be displayed. (A row is displayed as if it were a column.) If view is 'c', then <line> is the number of the column to be displayed. If view is 'y', <then> line is the year number.

Examples:

```
show b.am r 5
show b.am c 7
show b.am y 1997
```

Cutting and pasting works as with vectors.

sma <top> <first> <last> <order> [f]

To easily estimate a polynomial distributed lag, in which the lag coefficients are softly constrained to lie along a certain degree of polynomial, use the "sma" command, described here.

The "sma" command imposes a series of constraints which "softly" require the regression coefficients between 'first' and 'last' to lie on a polynomial of degree 'order'. If the 'f' is present at the end, the polynomial is "free" at the end; without the f, the polynomial will be assumed to be zero for the coefficient after 'last'.

Examples:

```
sma 100 a4 a9 3 f
sma 100 a8 a16 1
```

Related topics: *SoftConstraints*, *Distributed lags*.

stack

The format for "stack" is exactly like that for "sur" (see below) except that "stack" replaces "sur". With "stack", no attention is paid to contemporaneous covariances. The point of "stack" is solely to impose soft constraints across regressions.

Limitation of “stack” and “sur”: *G7*'s limitation of 30 variables, counting dependent, intercepts,and independent variables, can quickly become binding in “sur” and “stack”, since it applies to the total variables in all equations involved in the “sur” or “stack”.

Related topics: *sur*.

stochastic < y | n >

Set stochastic to ‘y’ if you want to save the results of regressions as stochastic regressions. See the section on stochastic regressions for the details of how to do this.

Related topics: Stochastic regression.

store

This command stores the currently loaded vector back to the Vam file with the modifications that have been made. This store is automatic when a new load or implicit load is encountered. When a "quit" is encountered with an unstored loaded vector, the program asks whether it should store that vector. Answering "no" is a chance to escape if you know you have made a mistake and do not wish to mess up your Vam file. On the other hand, giving the "store "before the "quit" is a way to avoid this question. Long add files will, therefore, frequently end with a "store".

Related topics: *load*.

(sty)pe <series_name> [<date1>] [<date2>]
(spr)int

"Silent" type writes the series to the currently open "save file" without displaying data on the screen. This can speed processing a long add file. Otherwise, it is identical to the "type" command.

Related topics: *save*, *type*.

(subt)itle <text string>

Provides <text string> as a subtitle on subsequent graphs. Use "subt" with no following text string to remove the subtitle.

Related topics: *Drawing graphs*, *title*.

sur

The format for the "sur" command is shown by this example:

```
sur
r y1 = x11, x12, ...
...
r yn = xn1, xn2, ...
con 10 1 = a2 + 4b5 + etc
sma 100 a6 a12 1
do
```

After the "sur" command come all the individual regressions. Any constraint or "sma" commands must come after the regressions. In the "con" and "sma" lines, a's denote coefficients in the first regression; b's coefficients in the second, and so on. Thus, a2 denotes the second coefficient of the first regression, and b5 denotes the fifth coefficient of the second regression.

The predicted values, dependent variables, and regression coefficients of the successive equations appear in the workspace file as predic1,depvar1, rcoef1, predic2, depvar2, rcoef2, etc. The results of the individual equations can be plotted by "gr *1", "gr *2", etc. The graph

commands should follow the "do" command and each should be preceeded by an appropriate "title" command.

Related topics: *stack*.

tdates <date1> <date2>
tdates a

The "tdates" command sets the dates used by subsequent "type" (or "print") commands. For example, the command

```
tdates 80.1 90.4
```

means that the next "type" or "print" command will display quarterly data from the first quarter of 1980 until the fourth quarter of 1990. The tdates are set implicitly when you do a type command with date arguments. They are then saved for the next type commands, and not changed until you give another "tdates", or "ty" with date arguments.

```
tdates a
```

Selects "automatic" dates for "graph" and "type" commands. The automatic dates are the first and last date of the series actually present. Automatic dates also adjust automatically to the frequency of the series.

Related topics: *Dates in G7*, *gdates*, *type*.

time

Display the date and time.

(**ti)tle** <title for regression or graph>

Provides a main title that appears on subsequent graphs. Use "ti" with no following text string to remove the title. The title may be up to 90 characters in length. However, on graphs that is probably too long! Note that titles can be passed as arguments when using the "add" or "fadd" command. In this case, the title must be enclosed within double quotes. Note that on graphs, there may also be a "subtitle" command given.

Related topics: *add*, *fadd*, *(gr)aph*, *(r)egress*, *(subt)itle*.

titpch [-<options>] [<stub_len>] ["str1" "str2" .."strn"]

The “titpch” command defines a header line that will be provided for a regression table, and also defines the regression output that will be displayed on subsequent “tpch” commands. Before using either the “titpch” or “tpch” commands, an equation table punch file (.eqp) must first be opened using the “eqpunch” command.

The <stub_len> is a number that indicates how long the “stub” or title for the line should be.

You should ensure that the width you specify in the “tpch” command is the same, so that the table will line up properly. See the example below for the “tpch” command.

The option string starts with a ‘-‘ character, and may contain one or more of the following:

b: rbar-squared
o: rho
r: r-squared
s: see
d: double-line format
f: floating point f format, instead of g format.

"str1", "str2", .., "strn" are strings to be printed to explain the coefficients. Therefore, "str1" might be "intercept", "str2" is the second explanatory variable name, and so on.

Related topics: *tpch, eqpunch*.

tpch [<sur which>] <sector> [<“label”>] [str_len] [(coef numbers)]

The “tpch” command is the command that writes out a line of an equation table file (.eqp). In order to use this command, a file should have first been opened using the “eqpunch” command, and a header definition supplied by the “titpch” command. The command line arguments of this command are as follows:

[<sur which>] is used only when an equation has been estimated with the “stack” or “sur” commands to that the coefficients are in rcoef1, rcoef2, etc.
<sector> is the number of the sector or category for which the equation is estimated.
<”label”> is a sector or category title in quotes. If used, you should also specify the length you want printed in the <str_len> argument
<str_len> is the length of the sector or category label in the printout. Note that this should be equal to the “stub length” specified in the “titpch” command.
[(coef numbers)] are used when there is a superset of regression parameters possible, and each equation uses some subset of that.

Example:

```
eqpunch ven.eqp

# Note: 30 is stub length.  Be sure to use the same for tpch!
titpch -rsf 30 const use usedif
add vena.reg   1 "Oilseed farming"
      .
      .
      .
eqpunch off
```

[**contents of vena.reg**]

```
ti %1 %2
subti Inventory Change Regression
f usedif = use%1-use%1[1]
```

```
r ven%1 = use%1, usedif
#gr *
ipch ven %1 a
tpch %1 "%2"  30
```

Related topics: eqpunch, titpch..

try{...} catch{...}

The try command must be followed by the catch command. These routines loosely clone similar routines in C++. Within the brackets following each command may be any collection of G7 routines, including other try-catch blocks. If execution of any of the commands within the try block produces an internal error signal, then execution of the code in the try block ceases and execution of the code in the catch block begins. Otherwise, the catch block is ignored. For example, consider the link series command ls. The routine will fail if the value of the guide series is zero in the base period. Ordinarily, this will cause G7 to stop execution of the script and report an error. If, on the other hand, the ls command is nested within a try block, then if an error occurs the alternative block of code following the catch command will be executed. In the following example, suppose that we prefer to use guide series y when it is available, and when it is not we rely on guide series z. We first try to extend series x using series y beginning in 2005, and if the value of y is zero in 2005, then we repeat the exercise to extend x using series z:

```
try{
   ls x y 2005 f
   }
catch{
   try{
      ls x z 2005 f
      }
   catch{
      ic Sorry.  No data in y or z.
      pause
      }
   }
```

The try-catch routine works well with many other commands, and has a wide variety of useful applications. Note again that try-catch blocks can be nested, so that several alternatives can be tried before G7 gives up in despair.

(ty)pe <series_name> [<date1>] [<date2>]
(pr)int

Displays values for the named series from <date1> to <date2> on the screen. The first number on each line is the date of the first observation on the line. Dates may be omitted if they are unchanged from the previous type command. If the "save" command (see below) is on, the displayed values are transferred to the save file where they are preceeded with an "update" command by default. The word "update" can be replaced by other commands, such as "data", "vdata", and various fix commands, by giving the appropriate word as the 3rd argument of the "save" command.

If you are printing out a lot of data to a file, you may want to use the "sty" command, which

will save some time by not writing output to the console.

Note that the formatting of the "ty" and "sty" commands can be changed with the "format" command, which can control the width, decimal points, and number of data points per line printed. When outputting data to programs that read fixed width data fields, this can be useful.

Related topics: *format*, *save*, *sty*, *tdates*.

(up)date <series_name>

Works like the "data" command but updates an existing series. The data following the command contain only the data points to be changed. The updated series is placed in the workspace only. Note that if you skip values as in the example below, that they will be assigned as missing values.

Example:

```
update um
  94.1  34.2 35.3
  95.1  37.0
  97.1  38.
```

You can later interpolate the missing values for a series by using the @lint function.

Related topics: *data*, *Missing values.*

vam <bankname> [<location>]

This command has the same format as "bank", and assigns a Vam file at the default location. The <location> must be a letter. If <location> is not given, it is assumed to be 'a', which is the first slot. Up to 26 data banks of the various types may be assigned, in positions 'a' through 'z'.

In *G7*, the equivalent function can also be reached via the Bank, Assign menu item, and choosing Files of type "Workspace bank".

Related topics: *Assigned Bank*s, bank, cbk, *dfr*, *hbk.*.

(vamcr)eate <vam configuration file> <vam file>

This command is used to create a Vam file from a Vam configuration file the command in *G7*. For example, to create the Vam file HIST.VAM from the configuration file VAM.CFG, the command is:

```
vamcreate vam.cfg hist
```

At this point, the newly created Vam file has zeroes for all of its data.

vammode [a | s]

The simulation files from *Interdyme* models usually consist of a logical pair. The Vam file contains the matrices and vectors, and the *G7* bank contains all macro variables. If you use the *Compare* program to make tables, and ask it to read data from a Vam file, it will treat the Vam file–*G7* bank pair as one logical entity. *G7* on the other hand, can treat the two as one logical bank like *Compare*, or require that you load them as separate banks. The first option, which is the default, is 'a' (advanced mode), and the alternative option, which you are free to use if you prefer, is 's' (simple mode). If you are working with several alternative simulations from *Interdyme* models such as *IdLift*, *Jidea* or *Intimo*, you most certainly will prefer to use the 'a' mode.

(vaxl)ab <in | out>

Causes the vertical axes numbers to be printed inside or outside the axes. The default is "vaxl in".

Related topics: *Drawing graphs*, *(vaxt)itle*.

(vaxt)itle <text string>

Provides a vertically printed title for the y (left) axis. Use "vaxt" with no following text to remove the left axis title.

Related topics: *Drawing Graphs*, *(subt)itle*, *(ti)tle*, *(vaxl)ab*.

vc <vector_name> = <expression>

The "vc", (vector calculate) command evaluates the expression on the right and puts the results into the named vector. Only a few matrix and vector operations are implemented with "vc". It can add or subtract any number of vectors and multiply a matrix times a vector. It cannot yet add or subtract matrices; parentheses are not yet supported. However, scalar values (either constants or *G7* bank variables) can be used to premultiply or postmultiply a vector or matrix. Also, a scalar value can appear all alone on the right hand side of a vc equation, which indicates that the entire vector will be set equal to that scalar value. Between a matrix and a vector, an * means matrix-vector multiplication. Between two vectors, the * means element-by-element multiplication. Between a vector and a matrix, the / means multiplication by the transpose of the matrix or vector on the left. Between two vectors, the / means element-by-element division. The vc command is performed only over the interval defined by the current fdates command Example: If am and cm are matrices and out, pdm, qcu, and cprices are vectors, we could write

```
vc out = am*out + out    # matrix multiplication, vector addition
vc qcu = out*pdm         # element by element vector multiplication
vc out = qcu/pdm         # element by element vector division
```

```
vc cprices = pdm/cm      # This is actually pdm'cm.
```

The <vector_name> must be a valid vector name in the Vam file, as must all names in the expression.

vdata <series_name>

This command works just like "data" except that the series introduced goes to the default Vam file. Recall that if fd is a vector in the Vam file, fd23 will be the 23rd element of it. If am is a matrix, am12.14 is the element in row 12, column 14.

Related topics: *data*, *update*, *vupdate*.

vf <name> = <expression>
vf <name>{<date1> [- <date2>]} = <expression>
vf <name> <operator> <expression>

This command is exactly like the "f" command, except that the destination for the calculated series is the default Vam file. Therefore, the <name> must be a legal name, i.e., a vector defined in that Vam file, with a sector number within the range of elements for that vector. A range of dates also may be provided to specify the periods to carry out the calculations.

Example:

```
ba outemp a
vam hist b
dvam b
vf out1=out1
```

This seemingly tautological example actually does something useful. It will look first in the workspace bank for a series named out1, and if it fails to find it, will look in the bank assigned in position 'a', which is OUTEMP.BNK. Let's say it finds the series there. It will then copy this series over to the default Vam file, which is HIST.VAM. The right hand side <expression> is any valid *G7* expression, and the operation is effective over the current fdates.

In addition to the assignment operator (i.e. "="), the *vf* command can add <expression> to the left-hand series using the "+=" operator, or it can subtract <expression> from the left-hand series with the "-=" operator, or multiply with "*=" or divide with "/=". These operators follow the syntax for the C++ programming language, though here they operate over a range of values.

Related topics, *f*, *vdata*, *vupdate*, *vam*, *dvam*.

vmake [<startyr> [<endyr>] <target> <source> ...

This command is used to make a vector out of other vectors, or a matrix out of vectors. The <startyr> and <endyr> parameters are optional. If not given, then the process will work over the entire range of the Vam file. Each matrix, whether it is a target or a source, is represented by:

MatrixName(r|c, < lines>)

where 'r' or 'c' indicate row or column, and <lines> are the selected rows or columns. If the <lines> specification is omitted, then the default is all rows or columns. Each vector in the list is represented simply by its name. If the total number of lines in the sources is different from that of the target, then "vmake" uses the smaller number. The target may be a packed matrix, in which case *the packed matrix file must have already been created, and the positions of the non-zero elements already determined*. In this case, "vmake" will only store elements in the non-zero positions. Here are some examples:

The first example puts the matrices a, b and c in rows 1, 2 and 3 of A.

```
vmake A(r 1-5) a b c
```

The next example puts column 3 of A into the vector c.

```
vmake c A(c 3)
```

The final example puts fows of B starting from row 1 followed by the vector c into rows 2 to 100 of the matrix A.

```
vmake A(r 2-100) B(r) c
```

(vmatd)ata <r|c> <nv> <nyrs> <first> <last> [skip]
<year> <vec1> <vec2> ... <vecnv>

or

<vec> <year1> <year2> ... <yearnyrs>
[form]

The "vmatdata" command puts whole vectors into the default va, file. It can be used to bring in data in a variety of formats commonly used by statistical agencies in releasing input-output tables. It can read a file which shows any one of the following:

one vector in columns for several years
one vector in rows for several years
several vectors in columns for one year
several vectors in rows for one year

The "vmatdat" command requires at least two lines and three if a format for reading is specified. The form of the first line is always the same, as is the form of the third line, if present. The second line has two different forms, one if several vectors are being read for one year and another if data for one vector is being read for several years.

In the above syntax, the arguments have the following meaning:

<r|c> is 'r' or 'c' according to whether the vectors are rows or columns.
<nv> is the number of vectors.
<nyrs> is the number of years.
<first> is the position in the vector in the Vam file of the first item in the data which follows.
<last> is the position in the vector in the Vam file of the last item in the data which follows.
<skip> is the number of spaces on each line which should be skipped before beginning to read data. If no value for <skip> is provided, then a form line (explained below) will be expected as the third line. If no spaces are to be skipped and no form line provided, then give skip a value of 0.
<year> is the year of the vectors or the first year if nyrs > 1.
<vec1> is the name of the first vector, vec2 is the name of the second vector, etc.
<form> If no value for skip is given, then this form line must be provided. It should have the letter r in every position which is to be read in the following table. This device allows direct reading of tables that have columns of data separated by | or other symbols or which have subtotals which are not to be read. A typical form line and a matching data line are:

```
form:                  rrrrrrrrrrrrrrrrrrr         rrrrr      rrrr

data:Agriculture     | 3450  3718  249   0  | 6780|  8102 |**| 1598
```

Columns which do not have an r are simply totally ignored. Consequently, they cannot be used to separate data fields. Thus, reading the following

```
  rrrrrrrr    rrrr
    851       2971
```

would produce one number, 8512971, not two numbers, 851 and 2971.

The first version of the second line is used if there are several vectors for one year; the second is used when giving data for one vector in several years. In both cases, the item of which there is only one is specified first and then the items of which there are several.

Any line beginning with a # is skipped completely.

For example, the lines

```
# Example of several vectors in columns for one year:
   vmatdat c 6 1 1 57 3
   1986    cons    gov     inv     stk     exp     imp
   1               21              0               2               -3              6         7
   …               …               …               …               …               …       ...
```

```
   57            38            401    0             0             17            15
```

would read the 6 columns of data into positions 1 through 57 of the named vectors (cons, gov, ...) for the year 1986. Positions 1 - 3 of each line, which contain the sector number for easy visual reference, are skipped.

Here is another example which reads the vectors as rows, as may often be the case when reading printed tables of primary inputs.

```
# Example of several vectors in rows for one year:
   vmatdat  r 6 1 1 6  16
   1986 labinc propinc deprec inter profit tax
   #                  1     2     3     4     5     6
   labor income    7303    21   368  1203  1302   820
   proprietor inc  1215     9   100   107   303    92
   depreciation     950     3    82   104   121    73
   interest income  845     7    83    54    95    52
   profits           50     3    25   217   337    38
   indirect taxes   121     1    32   178   294    29
```

Note the use of the line beginning with a # to provide labels for the columns.

The second form of the second line is illustrated by the following example, which reads one vector, cons, for five years.

```
# Example of a single vector in columns for several years:
   vmatdat c 1 5  1 57 2
   cons 1985 1986 1987 1988 1989 1990
    1   31.8 37.2 32.5 38.7 41.2 43.1
   ....
   57       1.2  1.7  1.8      2.0  2.1  2.2
```

Finally, here is an example which reads a single vector in columns for several years.

```
# Example of reading a single vector in columns for many years.
   vmatdat r 1 28 1 8  12
   demog 1989 1990 1991 1992 1993 1994 1995 1996 1997
   #  Date  ncent south  west college  twoy  fs1   fs2   fs5   head1 head3
    89.000  0.243 0.345 0.208  0.220  0.469 0.243 0.323 0.106  0.274 0.350
    90.000  0.241 0.346 0.209  0.222  0.476 0.250 0.320 0.105  0.267 0.350
    91.000  0.240 0.347 0.211  0.224  0.482 0.247 0.320 0.105  0.263 0.348
    92.000  0.238 0.348 0.212  0.226  0.487 0.245 0.319 0.104  0.260 0.346
```

It must be emphasized that, precisely because it can read in free format, the “vmatdat” command cannot interpret blanks as zero entries. There must be a numerical value for all cells in the tables, especially the 0's.

The flexibility of “vmatdat” often makes it possible to read in final demand columns or primary inputs as they appear in printed tables or on the diskettes provided by statistical offices.

vp <vector_name> [r|c] [field_width] [decimal_points]

or

vp <matrix_name><view><line> [field_width] [decimal_points]

The "vp" command writes the named matrix to the currently open "save" file. As you can see, this command has a format and options like those of the "show" command.

Example:

```
save am.dat
vp am y 2000  9  6
save off
```

Related topics: *punch*, *pmpunch*, *punchvec*.

(vr)ange <bottom> [top]

vr <bottom> [line1] [line2] ... line[8] <top>

vr off

The "graph" command normally sets the vertical range so the series extend from the bottom to the top of the graph. Sometimes, it is desirable to set the range independently. This is done with the "vrange" command. For example "vr 0 100" will make all subsequent graphs have 0 at the bottom and 100 at the top until the "vr off" command is encountered, restoring *G7* to its normal mode. If the <top> is omitted, then only the bottom of the graph is set and the program finds the top so that the graph fits on the screen.

If one or more of the optional [line] entries are present, horizontal lines will be marked at those levels. The lighted pixels which mark these lines are located above the long marks on the horizontal axis, so they also serve as meaningful vertical lines.

Related topics: *Drawing graphs, line*, *vaxlab, hrange.*

vtitle <title>

This command allows you to give a title to the default Vam file. This title is displayed when using *Compare*.

vupdate

vup

This command works just like "update" except that the series goes into the default Vam file. For this to work, a Vam file must be assigned with "vam", and it must be set to the default with "dvam".

Related topics: *vam*, *dvam*.

(wri)ndex <y | n>

Normally *G7* writes the entire index file of the workspace as each variable is added. When

many series are added, this feature lengthens processing time. To defer writing the index until the entire add file is processed, "wri n" turns off writing and "wri y" turns it on again and writes the index. A "q" with writing off will cause the index to be written before quitting, so there is no danger of losing everything by forgetting "wri y".

Related topics: *Tips and tricks*, *Workspace banks*.

(wsb)ank <bank_name>

This command makes the named bank the workspace bank. Make sure you have a backup of the bank that you assign as a workspace, since it will be modified if you create any data series during the current *G7* session. In addition, a "zap" command will totally destroy the bank.

In traditional *G7* usage, the workspace bank is called ws.bnk, and is in the current directory. However, by editing the first two lines of the G.CFG file you can change the default location and name of the workspace bank to be any bank you like. A commonly used tip is to set up the workspace bank on a RAM disk, to speed the writing of series created in *G7*.

Related topics: *(ba)nk*, *The G.CFG File*, *Workspace Bank*, *zap*.

(wsi)info

This command prints information to the screen about the current workspace bank, including its bank title (if any), the number of series in the bank, the maximum number of bservations per series, and the default starting year (base year) and period.

Related topics: *(ba)nk*, *(wsb)ank*, *zap*.

xl

The "xl" command is actually a whole family of two or three word commands that enable the reading and writing of Excel worksheets from within *G7*. Actually, *G7* does not do the reading and writing itself, but makes use of a DLL file that comes with Excel. Therefore, Excel must already be installed on your machine if these "xl" commands are going to work.

Next, each member of this family of commands will be described. Then some examples will be provided that show how to read from and write to Excel files.

xl open <xlsfile>

This is the command for opening an Excel file, either for writing or reading.

xl open worksheet <worksheet number>

This command opens a worksheet with the Excel file for reading or writing. The number to be used is the order of the worksheet within the file. (The first tab is '1').

xl create

xl create workbook [<filename> [<filetype>]]

An *xl create* command with no arguments launches the Excel server and opens a new workbook with one worksheet.

An *xl create workbook* command may provide the name for a new Excel *filename*. If no name is provided, a new workbook will be created with the default name. The option *workbook* may be abbreviated *wb*.

The default filetype is XLS. Other available file types include AddIn (.xlam), CSV (.csv), CSVMac (.csv), CSVMSDOS (.csv), CSVWindows (.csv), DBF2 (.dbf), DBF3 (.dbf), DBF4 (.dbf), DIF (.dif), Excel2 (.xls), Excel2FarEast (.xls), Excel3 (.xls), Excel4 (.xls), Excel5 (.xls), Excel7 (.xls), Excel9795 (.xls), Excel4Workbook (.xls), IntlAddIn (.xla), IntlMacro (.xlsm), WorkbookNormal (.xls), SYLK (.slk), Template (.xltx), TextMac (.txt), TextMSDOS (.txt), TextPrinter (.txt), TextWindows (.txt), WK1 (.wk1), WK1ALL (.wk1), WK1FMT (.wk1), WK3 (.wk3), WK4 (.wk4), WK3FM3 (.wk3), WKS (.wks), WQ1 (.wq1), UnicodeText (.txt), Html (.html), and XLS (.xls). If no extension is provided with the filename, then the appropriate extension will be determined according to the file type and will be appended to the file name.

xl create worksheet [<name>]

An *xl create worksheet* command may provide a name for a new worksheet to be added to the open workbook. The instruction *worksheet* may be replaced with *ws*. See also the *xl name* command.

xl save [<namefile> [filetype]]

A *save* command must provide a name for the open workbook, if the workbook has not already been named.

If a file type is specified that is different than the current setting, then the spreadsheet will be saved as the new file type. The default filet ype is XLS. Other available file types include AddIn (.xlam), CSV (.csv), CSVMac (.csv), CSVMSDOS (.csv), CSVWindows (.csv), DBF2 (.dbf), DBF3 (.dbf), DBF4 (.dbf), DIF (.dif), Excel2 (.xls), Excel2FarEast (.xls), Excel3 (.xls), Excel4 (.xls), Excel5 (.xls), Excel7 (.xls), Excel9795 (.xls), Excel4Workbook (.xls), IntlAddIn (.xla), IntlMacro (.xlsm), WorkbookNormal (.xls), SYLK (.slk), Template (.xltx), TextMac (.txt), TextMSDOS (.txt), TextPrinter (.txt), TextWindows (.txt), WK1 (.wk1), WK1ALL (.wk1), WK1FMT (.wk1), WK3 (.wk3), WK4 (.wk4), WK3FM3 (.wk3), WKS (.wks), WQ1 (.wq1), UnicodeText (.txt), Html (.html), and XLS (.xls). If no extension is provided with the filename, then the appropriate extension will be determined according to the file type and will be appended to the file name.

xl write <column letter> <row number> <label>

This command writes a label of text to a cell in the worksheet.

xl write <column letter> <row number> <down | right > <series name> <starting date> <ending date>

This command writes a time series into the Excel worksheet, starting at the given location, going down a column, or across a row, for the dates given.

xl vecwrite <vector> < v(index) > < c(cols) > < r(rows) > < direction > [< start > [end]]

G7 can print entire vectors of data for multiple years using a single command. The vecwrite command must be used with a VAM bank. Vector is the root name of a vector stored in a vam bank. Bank letters are allowed. Index is the the range of vector elements that are to be printed. For example:

```
xl vecwrite a.output v(1-10) c(B) r(5-14) right 2008 2010
```

xl formula <column> <row> "< formula >"

Insert an Excel formula in the specified cell.

```
xl formula A 5 "=sum(A1:A4)"
```

xl setfont <settings>
xl font <column1> <row1> <column2> <row2> <settings>

The setfont routine sets the font for all subsequent printing. The font command sets the font for the specified range of cells.

The settings are a listing of one or more font settings.

- clear — Clear the fonts set by the setfont command.
- bold
- italic
- underline — See the list of underlining options.
- horizontal alignment — "justify", "center", “right", or "left"
- vertical alignment — "top", "vcenter", "bottom", or "vjustify"
- color — See the list of available text colors.
- type face — See the list of available type faces.
- sizeX — Set the font size to integer X.

Underline

Underlining may be specified in several ways.

underline [-]
UnderlineSingle
-
Single underlining.

underline [=]
UnderlineDouble
=
Double underlining.

underline [_]
UnderlineSingleAccounting

_
Single accounting underlining.

underline [~]
UnderlineDoubleAccounting
~
Double accounting underlining.

underline n
UnderlineNone
No underlining.

Font Colors

"None"
"Aqua"
"Black"
"Blue"
"Cream"
"Dark Gray"
"Fuchsia"
"Gray"
"Green"
"Lime green"
"Light Gray"
"Maroon"
"Medium Gray"
"Mint green"
"Navy blue"
"Olive green"
"Purple"
"Red"
"Silver"
"Sky blue"
"Teal"
"White"
"Yellow"

Font Types

"Agency FB"
"Agency FB Bold"
"Algerian"
"Arial"
"Arial Black"
"Arial Black Italic"
"Arial Bold"
"Arial Bold Italic"
"Arial Italic"
"Arial Narrow"
"Arial Narrow Bold"
"Arial Narrow Bold Italic"
"Arial Narrow Italic"
"Arial Rounded MT Bold"
"Arial Unicode MS"
"Baskerville Old Face"
"Batang"
"Bauhaus 93"
"Bell MT"
"Bell MT Bold"
"Bell MT Italic"
"Berlin Sans FB"
"Berlin Sans FB Bold"
"Berlin Sans FB Demi Bold"
"Bernard MT Condensed"
"Blackadder ITC"
"Bodoni MT"
"Bodoni MT Black"
"Bodoni MT Black Italic"
"Bodoni MT Bold"
"Bodoni MT Bold Italic"
"Bodoni MT Condensed"
"Bodoni MT Condensed Bold"
"Bodoni MT Condensed Bold Italic"
"Bodoni MT Condensed Italic"

"Bodoni MT Italic"
"Bodoni MT Poster Compressed"
"Book Antiqua"
"Book Antiqua Bold"
"Book Antiqua Bold Italic"
"Book Antiqua Italic"
"Bookman Old Style"
"Bookman Old Style Bold"
"Bookman Old Style Bold Italic"
"Bookman Old Style Italic"
"Bradley Hand ITC"
"Britannic Bold"
"Broadway"
"Brush Script MT Italic"
"Californian FB"
"Californian FB Bold"
"Californian FB Italic"
"Calisto MT"
"Calisto MT Bold"
"Calisto MT Bold Italic"
"Calisto MT Italic"
"Castellar"
"Centaur"
"Century"
"Century Gothic"
"Century Gothic Bold"
"Century Gothic Bold Italic"
"Century Gothic Italic"
"Century Schoolbook"
"Century Schoolbook Bold"
"Century Schoolbook Bold Italic"
"Century Schoolbook Italic"
"Chiller"
"Colonna MT"
"Comic Sans MS"
"Comic Sans MS Bold"
"Cooper Black"
"Copperplate Gothic Bold"
"Copperplate Gothic Light"
"Courier New"
"Courier New Bold"
"Courier New Bold Italic"
"Courier New Italic"
"Curlz MT"
"Edwardian Script ITC"
"Elephant"
"Elephant Italic"
"Engravers MT"
"Eras Bold ITC"
"Eras Demi ITC"
"Eras Light ITC"
"Eras Medium ITC"
"Felix Titling"
"Footlight MT Light"
"Forte"
"Franklin Gothic Book"
"Franklin Gothic Book Italic"
"Franklin Gothic Demi"
"Franklin Gothic Demi Cond"
"Franklin Gothic Demi Italic"
"Franklin Gothic Heavy"
"Franklin Gothic Heavy Italic"
"Franklin Gothic Medium"
"Franklin Gothic Medium Cond"
"Franklin Gothic Medium Italic"
"Freestyle Script"
"French Script MT"
"Garamond"
"Garamond Bold"
"Garamond Italic"
"Gigi"
"Gill Sans MT"
"Gill Sans MT Bold"
"Gill Sans MT Bold Italic"
"Gill Sans MT Condensed"
"Gill Sans MT Ext Condensed Bold"
"Gill Sans MT Italic"
"Gill Sans Ultra Bold"
"Gill Sans Ultra Bold Condensed"
"Gloucester MT Extra Condensed"
"Goudy Old Style"
"Goudy Old Style Bold"
"Goudy Old Style Italic"
"Goudy Stout"
"Haettenschweiler"
"Harlow Solid Italic"
"Harrington"
"High Tower Text"
"High Tower Text Italic"
"Impact"
"Imprint MT Shadow"
"Informal Roman"
"Jokerman"
"Juice ITC"
"Kristen ITC"
"Kunstler Script"
"Lucida Bright"

"Lucida Bright Demibold"
"Lucida Bright Demibold Italic"
"Lucida Bright Italic"
"Lucida Calligraphy Italic"
"Lucida Fax Demibold"
"Lucida Fax Demibold Italic"
"Lucida Fax Italic"
"Lucida Fax Regular"
"Lucida Handwriting Italic"
"Lucida Sans Demibold Italic"
"Lucida Sans Demibold Roman"
"Lucida Sans Italic"
"Lucida Sans Regular"
"Lucida Sans Typewriter Bold"
"Lucida Sans Typewriter Bold Oblique"
"Lucida Sans Typewriter Oblique"
"Lucida Sans Typewriter Regular"
"Magneto Bold"
"Maiandra GD"
"Map Symbols"
"Matura MT Script Capitals"
"Mistral"
"Modern No. 20"
"Monotype Corsiva"
"MS Mincho"
"MS Outlook"
"MT Extra"
"Niagara Engraved"
"Niagara Solid"
"OCR A Extended"
"Old English Text MT"
"Onyx"
"Palace Script MT"
"Palatino Linotype"
"Palatino Linotype Bold"
"Palatino Linotype Bold Italic"
"Palatino Linotype Italic"
"Papyrus"
"Parchment"
"Perpetua"
"Perpetua Bold"
"Perpetua Bold Italic"
"Perpetua Italic"
"Perpetua Titling MT Bold"
"Perpetua Titling MT Light"
"Playbill"
"PMingLiU"
"Poor Richard"
"Pristina"
"Rage Italic"
"Ravie"
"Rockwell"
"Rockwell Bold"
"Rockwell Bold Italic"
"Rockwell Condensed"
"Rockwell Condensed Bold"
"Rockwell Extra Bold"
"Rockwell Italic"
"Script MT Bold"
"Showcard Gothic"
"SimSun"
"Snap ITC"
"Stencil"
"Symbol"
"Tahoma"
"Tahoma Bold"
"Tempus Sans ITC"
"Times"
"Times New Roman"
"Times New Roman Bold"
"Times New Roman Bold Italic"
"Times New Roman Italic"
"Trebuchet MS"
"Trebuchet MS Bold"
"Trebuchet MS Bold Italic"
"Trebuchet MS Italic"
"Tw Cen MT"
"Tw Cen MT Bold"
"Tw Cen MT Bold Italic"
"Tw Cen MT Condensed"
"Tw Cen MT Condensed Bold"
"Tw Cen MT Condensed Extra Bold"
"Tw Cen MT Italic"
"Verdana"
"Verdana Bold"
"Verdana Bold Italic"
"Verdana Italic"
"Viner Hand ITC"
"Vivaldi Italic"
"Vladimir Script"
"Wide Latin"
"Wingdings"
"Wingdings 2"
"Wingdings 3"

xl read <column letter> <row number> “”

Read the label at the specified column and row. The string of text will be printed to the screen.

xl mkseries <frequency> [text] <column> <row> [text]

Read a text string from a worksheet cell, and optionally attach additional text to the string. A new series with the created name is added to the workspace. If a following xl read command is given, and the series name is omitted in the read command, then the series created by the previous mkseries command will be assumed.

xl read <column letter> <row number> <direction > <series name> <starting date> <ending date>

Read a time series from the Excel worksheet, starting at the given location and proceeding in the direction specified, for the dates given. *direction* may be right, down, left, or up.

xl matread c(column index group) **r**(row index group) <matname> **c**(column index group) **r**(row index group) <date>

The “xl matread” command is rather complicated, but very powerful. If you have a matrix of data in a spreadsheet, and you want to read it into a matrix in a Vam file, use the first “c” and “r” expressions to indicate the numbers of the columns and rows in which the matrix is contained in the worksheet. Then give the name of the matrix where you want to place the data, and then the column and row group expressions where you want to put the data. The example below will make this more clear.

xl missing [symbol]

This sets missing value symbols for the spreadsheet. When *G7* is reading the spreadsheet file, a “missing value” entry is recorded in the *G7* data bank for any spreadsheet cell containing this *symbol*. The symbol may be a word, number, or a string, where strings must be specified in quotes in the *xl missing* command. Up to 10 missing value codes may be stored, but each must be entered separately. If the command is given without arguments, then previous entries are reported. The following example allows *G7* to recognize 0.0, NA, and _N/A_ as missing value codes when they are read from a spreadsheet file. See also the *replace* command.

xl print missing “<value>”

This command provides a character or string to indicate a missing value when G7 writes data to a spreadsheet.

```
xl print missing "N/A"
```

xl clear missing

This command clears missing value codes specified in *xl missing*. It also resets the replacement value to the default setting, where replacement values are specified with the *xl replace* command.

xl replace <value>

This command sets a replacement value for missing values in the spreadsheet file. If *G7* reads a cell that matches an entry set with *xl missing*, then the value is replaced by *value*. By default, this replacement value is the *G7* missing value code (-0.0000001). *value* must be a number. The following example replaces missing value codes read from a spreadsheet with zeros in the *G7* data bank. See also the *xl missing* command.

xl visible
xl invisible

This command makes visible (invisible) the Excel program, provided that Excel is running. Making Excel visible (invisible) may decrease (increase) the execution speed of your script. By default, the Excel window is not visible when *G7* launches the Excel server and opens a spreadsheet file.

xl name worksheet <sheetname>

This command names the worksheet that currently is open.

xl column width <column> <width>

This command sets the column *column* in the selected worksheet to width *width*.

xl close

This command closes the Excel file.

xl exit

This command closes an open workbook and disconnects *G7* from the Excel server that is running in the background. If the Excel server was started by *G7*, then it will be stopped. Otherwise, the Excel server may continue running in the background; it can be closed by opening the Task Manager, selecting "Excel," and terminating the process.

Example 1:

In this example, the file "xltest.xls" must already exist in the current directory (*G7* cannot create the Excel file). The time series are written to the spreadsheet by giving the starting cell, the direction to write the data (down or right), the name of the series, and the range of periods to write.

```
fdate 1975 2010                         # Create sample data.
f Year = 1974 + @cum(t, 1.0, 0.0)
f Data = 1975 / (t - 1975)

xl open xltest.xls              # Start Excel server, open the xltest.xls
                                # workbook.
xl open worksheet 1             # Open worksheet 1.
xl write A 1 "Writing text to file:"
xl write A 3 "Year"             # Record label for the year
xl write A 4 down Year 1976 2010       # Record the year
xl write B 3 "Data"             # Record series name
xl write B 4 down Data 1976 2010       # Write series 'Data'
```

```
xl close                                # Close the workbook.
```

Example 2:

In this example, we re-open the same file, and read one of the series back into *G7* with a different name.

```
xl open xltest.xls              # Open the workbook.
xl open worksheet 1             # Open worksheet 3.
xl read A 1 ""                  # Read the string in position A1; string
                                # will be printed on screen.
xl read B 4 right Year2 1976 2010      # Read data into workspace.
xl exit                         # Close workbook, close connection to
                                # Excel server.
```

Example 3:

In this example, we must be working with a Vam file, and it must be opened and declared as the default. The data read from the spreadsheet are stored as vector elements in the Vam file. Writing to the Vam file, instead of to the *G7* workspace bank, is forced by providing a bank leter ('c') corresponding to the open Vam bank in front of the name of the vector. Note that columns may be specified either by the Excel column letters or by the column number.

```
xl open C0301e.xls
xl open worksheet 1
do{
   xl read %1 27 down c.gdpN%2 1990 1990
   }(3-4 6-7 9-10)(1-6)m
xl exit
```

Example 4:

In this final example, we show the use of the "xl matread" command. The command must all be on one line, even though it seems to span two lines in this example. First the blocks of data in the Excel spreadsheet are specified by listing their rows and columns. Next, the name of the matrix is given in which the data should be sotred, along with the matrix rows and columns. Finally, the year is given for which the matrix should be stored.

```
xl open C0319e.xls
xl open worksheet 1
xl matread c(2-18) r(14-17, 19-20, 22-24,26-29,31-32, 34-35)...
   c.AM c(1-17) r(1-17) 2000
```

Related topics: *p123, 123toG*.

yearformat(yf) <format>

G7 solved the "year 2000" problem long ago, by representing the year 2000 as "100", 2010 as "110" and so on. However, more recent versions of *G7* are even smarter, in that they can take either a 2-digit or 4-digit date, and convert it internally into a "G Date".

You can provide control over how you would like these dates to print out in "print", "matpr" or "graph" commands by using the "yearformat(yf)" command.

The possible values for <format> are:

'2' -- All dates 2-digit, i.e. 77, 00, 05
'4' -- All dates 4-digit, i.e. 1977, 2000, 2005
'3' -- 2 or 3 digit dates, 77, 100, 105

Related topics: *Dates in G7.*

zap [<baseyr>] [<starting period>] [<nobservations>]

The "zap" command is used to delete the old workspace bank, and create a new bank, with the name specified in G.CFG. The three command line arguments are all optional. If they are given, they override the corresponding values in G.CFG.

If <baseyr> is given, it overrides the line labeled "Default base year of workspace file. If <starting period> is given, it overrides the line labeled "First month covered", or "First period covered". If <nobservations> is given, it overrides the line labeled "Default maximum number of observations per series in workspace bank".

Related topics: *G.CFG file*, *workspace bank*.

zip

After the "zip" command has been given, the program will not pause after regression commands. It is particularly useful for rapid re-estimation of an entire model when data has been updated. The command to turn off "zip" is "zip off".

The "zip" command is used in production mode, when you are sure that your add files of regressions or other procedures is correct, and you want to run a large job in batch mode, with no pauses or interruptions. For example, when creating many graphics files, such as .wpg or .wmf files, you may turn on "autoprint" and use "zip". It is also used when creating large equation parameter files using the "punch" and "ipch" command for *Interdyme* models.

Related commands: *autoprint*, *(cat)ch*, *ipch*, *punch*, *save*.