Vam Dates and G7 Updates

In July 2003, I set out to extend the capability of G7 by enabling it to handle vectors and matrices of various frequencies. Until now, Vam and the programs that depend on it handled vectors and matrices only of annual frequency. Because most intersectoral models that use Inforum software are of annual frequency by nature, this limitation has not been problematic. My present work requires storage of monthly data, and so I extended the capability of G7 to create and manage high frequency databanks. The tools within G7 also now have the capability to manipulate this data. While testing these new features, I uncovered and fixed a number of existing bugs and added a few new features. The following pages describe in greater detail the changes I made to G7.

Vam Frequencies

Nearly all of the commands in G7 that depend on Vam have been updated and tested. With the exception of packed matrix binary files (*.pmx), all existing binary and text files should be compatible with this version. In all cases, existing dates (e.g. 1975) may now be replaced by dates in the GDate format (e.g. 1975 for annual dates, 1975.1 for Quarter 1 1975, 1975.001 for January 1975, and so on). Existing ASCII data files will have dates like "1975" and so naturally are compatible with the new code. Existing G7 "add" files also are compatible, because while G7 commands now work for Vam banks of high frequencies, they also continue to work with Vam banks of annual frequency. Hence, all commands should work as expected.

The new code is contained on the Sartoris ftp site (sartoris.umd.edu/horst) in "G7_files.zip," the compiled program is in "G7.zip," and test programs and data are contained in "data.zip." Download the files, unzip them, and extract the code and compiled program to the "G7" directory and the data to the "data" directory. You may copy the executable program "G7.exe" to your "pdg" directory instead; however, you first should back up or rename your existing version of G7.exe so that it is not overwritten.

To examine the capabilities of this version, start G7 with the "g.cfg" file contained in the "data" directory. You are now ready to follow along with this paper and to conduct your own trials. Some of the code contained in the "data" directory is presented here. This data is used by the test file "vt.add" to demonstrate the new capabilities of G7. The following G7 commands may be found there, and the corresponding files can be found in the "data" directory.

The test Vam bank is established in "vam.cfg." Here are a few lines:

```
# vam.cfg for nuclear data from Rust/Rothwell
1975.001 1994.012
#
HREFUEL_ 117   1     0     nuketitles.ttl
```

```
VMATDAT_ 2     1     0     matin_.ttl
MATIN_   2     2     0     matin_.ttl        matin_.ttl
PMATIN_  2     2     p     matin_.ttl        matin_.ttl
VMAKE_   117   117   0     nuketitles.ttl    nuketitles.ttl
```

Vector and matrix names are specified with capital letters and a trailing "_". The only unusual characteristics are the monthly dates at the top of the file; otherwise, this file is no different than any other Vam configuration file. The file is read and the Vam bank is created by a "vamcreate" command, the bank is loaded by a "vam" command, and the bank is assigned as default with a "dvam" command:

```
vamcreate vam.cfg nukehist
vam nukehist b
dvam b
```

The bank that has been created is empty; all values are set to zero. The databank is filled with a variety of commands. "add hrefuel.dat" loads vectors using the "vdata" command (the data presented here is not complete; the data files contain values to December 1994):

```
vdata HREFUEL1
1975.001 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0;
1976.001 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 98.8;
1977.001 672.0 744.0 96.4 0.0 0.0;

vdata HREFUEL2
1980.004 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 93.7 720.0;
```

Similar lines update the vector with the "vup" command and read additional data with the "vmatdat" and "fvread" commands:

```
vup HREFUEL1
75.001 1 2 3;

vmatdat r 1 1 1 2 0
1975.001 VMATDAT_
1 11

vmatdat c 1 2 1 2 0
VMATDAT_ 1975.003 1975.004
3 4
33 44

fvread VMATDAT_ 1975.005 0
5
55
```

The "matin" and "matin5" commands load matrix data:

```
matin MATIN_ 1975.001 1 2 1 2 0
1 2
```

```
3 4

matin5 MATIN_ 1975.005 3 0 2
MATIN_ 1 1 11 1 2 22 2 1 33 2 2 44
```

The “pmatin” and “pmatin1” commands do the same for packed matrices:

```
pmatin PMATIN_ PMATIN_.pmx
1975.001
1 1
1 1
2 1
2 2
1975.005
1 1
1 11
2 1
2 22

pmatin1 PMATIN1_ PMATIN1_.pmx
1975.001
1 0
0 2
1975.005
11 0
0 22
```

Packed matrix binary files are associated with the default Vam file with “pmfile.” Missing data can be interpolated by the “lint” command:

```
pmfile PMATIN1_ PMATIN1_.pmx
lint MATIN_
lint PMATIN_
```

The vectors and matrices can be listed with “listvecs,” graphed with “gr,” “mgr,” “lgr,” or “sgr,” and displayed with “show,” “type,” or “matty.”

```
show b.HREFUEL
show b.MATIN1N_ y 1975.001
show b.PMATIN1_ r 1
ty HREFUEL1
matty 1975.001 1993.005
  HREFUEL1 HREFUEL2;
```

Calculations are done in the normal fashion, with starting and stopping points set by “fdates”:

```
fdates 1975.001 1980.012
f dd = HREFUEL1
vf HREFUEL1 = 200 + 2* HREFUEL1
vf MATIN_1.1 = 11
vc HREFUEL = HREFUEL*HREFUEL
```

At least most "@" commands work as expected, as does the regression command:

```
f lindex3 = @log(INDEX_3)
r HREFUEL1 = HREFUEL2, lindex3
```

Frequency conversions work for vectors as they do for time series:

```
fdates 1975.1 1994.1
f temp = @mtoq(INDEX_1)
```

Vectors and matrices are constructed from other vectors and matrices with "vmake," and rows and columns can be moved within a matrix with "move":

```
vmake VMAKE_(r 1-117) HREFUEL
vmake HREFUEL VMAKE_(r 1)
move MATIN_ up 2 2 1 1975.001
```

Dynamic and static groups function as usual. Here, the vector "INDEX_" is loaded with the "load" command. Column 1, specified by a dynamic group, is guided by the time series "t" using the "index" command; "index" also works on full and packed matrices, and a vector or matrix may be used in place of the time series "t."

```
group 1
load INDEX_
index 1975.001 t :
```

A time series may be used to control the total of a vector; here, "t" is used to control the total of vector "INDEX_," and the results are stored with "store":

```
ctrl t INDEX_ 1
store
```

Diagonal elements of either a full or packed matrix may be extracted with the "diagextract" command and inserted with the "diaginsert" command. "getsum" puts the row or column sum of a full or packed matrix into a vector. "coef" converts a flow matrix to a coefficient matrix, and flow converts a coefficient matrix to a flow matrix.

```
getsum MATIN_ r INDEX_
getsum PMATIN_ c INDEX_
coef MATIN_ INDEX_ 1975.001
coef PMATIN_ INDEX_ 1975.001
flow MATIN_ INDEX_ 1975.001
flow PMATIN_ INDEX_ 1975.001
```

Similarly, "rowscale" is used to scale the rows of a matrix, and "scale" is used to scale a vector or a matrix row or column.

```
rowscale MATIN_ INDEX_ y
rowscale PMATIN_ INDEX_ y
scale t INDEX_ (1) 1975.001
```

```
scale t MATIN_ r 1 1975.001 r
```

"ras," "rdras," and "psras" are used to balance full and packed matrices:

```
ras MATIN_ INDEX_ INDEX_ 1975.001 r
ras PMATIN_ INDEX_ INDEX_ 1975.001 c
rdras MATIN_ INDEX_ INDEX_ 1975.001 c
rdras PMATIN_ INDEX_ INDEX_ 1975.001 r
psras MATIN_ INDEX_ INDEX_ 1975.001 r
psras PMATIN_ INDEX_ INDEX_ 1975.001 r
```

After a Vam bank has been created, it may be necessary to store the data as text. The commands "pmpunch" and "punch5" print full and packed matrices, and "punchvec" prints vectors.

```
pmpunch  pmpunchf.out b.MATIN_  2
pmpunch  pmpunch.out  b.PMATIN_ 2
punch5   punch5f.out  b.MATIN_
punch5   punch5p.out  b.PMATIN_
punchvec index.out    b.INDEX_  1975.001 1975.005
```

Several other commands also have been tested, and some (e.g. "monup") still have not been updated or tested at high frequencies. While problems undoubtedly remain, this version of G7 now is ready for testing on real modeling problems. The existing version of G7 contained a number of bugs. Many turned up in the present work; most of them have been fixed and are presented here.

New Features

The new capabilities center on the "dos" command for executing system commands from within G7. First, the G7 "dos" command, which passes a command to the operating system, has proven troublesome because the DOS window closes before the user can see the results of the operation. These results now are displayed in the G7 output window. For example, suppose the G7 user wanted to copy the workspace bank to another file. This can be done with "dos copy ws.* newbank.*". Before, when the command was executed in G7, a DOS window opened, the operating system tried to execute "copy ws.* newbank.*", and the window closed before the user could determine the success of the operation. Now, the results reported by the operating system are copied to the G7 output window.

Second, the "dos" command now can execute a series of commands as in a DOS batch file. To execute commands in batch mode, simply append a "{" to the "dos" command, followed by the DOS script, and end with a "}". Arguments can be passed to the "batch file" by giving the "args" command followed by the arguments. For example,

```
dos{ args 1  2  It_works!
     rem THIS IS A TEST
     if "%2" == "2" copy ws.* newws.*
     if "%1" == "1" echo "%3"
     }
```

will create a batch file containing the lines beginning with “rem” and “if”, and then this file will be executed with the options “1 2 It_works!”. These lines will be echoed on the G7 output screen (if “addprint” is on), followed by the DOS results:

```
C:\> rem THIS IS A TEST

C:\>if "2" == "2" copy ws.* newws.*
ws.bnk
ws.ind
        2 file(s) copied.

C:\>if "1" == "1" echo "It_works!"
"It_works!"
```

Of course, multiple lines can be executed without arguments, and single arguments still can be called with “dos” and no brackets. Note that results now will be displayed on the G7 output screen but not in the DOS window; this output is not displayed in the G7 window until the DOS command is finished. To work interactively in DOS, give the command “dos” with nothing following it; this opens a DOS window in which you can work.

Bug Fixes

Some bugs exist in G7; many have been fixed in the present work, a few have been detected but not fixed, and surely others remain undetected. Fortunately, many of the problems noted here are “int versus short” problems, problems with the use of “delete,” and problems with pointers.

Many packed matrix commands failed because of confusion between the types “short” and “int.” In earlier generations of computers (16 bit machines), integers (int’s) were stored in two bytes (16 bits) of memory. When 32 bit machines became available, compilers began to allocate 4 bytes (32 bits) of memory for each “int.” A new type of variable, the two byte “short,” was introduced to replace the original (two byte) “int.” Generally, this is not a problem for new programs, but it can cause confusion for programmers who try to update code written for 16 bit machines to run on 32 bit machines. A few such problems existed in G7. Packed matrix headers (“pmfid” structures) contain data such as the number of rows and columns; these data are stored as “short” integers. Packed matrices were created properly and stored as binary files. The problems arose when the binary files were read. Many times, lines such as the following were specified:

```
pmfid pmh;
int *prnc;
…
fread(&pmh, sizeof(pmh), 1, fin);
if((prnc = new int[j])==0) goto memerr;
fread(prnc, 2, row, fin);
```

The problem is that the “prnc” is an array of four-byte “ints” but the “fread” function is reading two bytes at a time. Hence, the two bytes are put into the FIRST two (from the left) bytes of the “int” instead of the LAST two. The result is that the first value of “prnc” is off by about 65,000 and the remaining values are garbage. The problem is fixed by making “prnc” an array of shorts.

A second category of problems is even more prevalent. While the problems above often cause G7 commands to fail completely, improper use of “delete” and “delete[]” cause memory leaks and other subtle problems. There were and are many variations of these problems in G7; some such occurrences have been fixed, but others remain.

When a “new” operator is used, it always should be followed with “delete” or “delete[].” Note that “delete” and “delete[]” are used ONLY with memory that has been allocated dynamically (with “new”). I found one or two cases in G7 like the following:

```
int some_ints[10];
…
delete some_ints;
```

Usually, the compiler will complain about such things, but for some reason it did not complain about those lines. I have no idea what happens when such code gets by the compiler, but this is very bad code.

Failure to call “delete” or “delete[]” can result in memory leaks. Leaks are most likely to occur when “new” is used to create a single instance or an array of objects. “delete” must be called to free a single object (including built-in data types like “int” and “float”) that has been created dynamically, and “delete[]” must be called to free an array. Both “delete” and “delete[]” call the object’s destructor. Common problems are the omission of these operators or the use of “delete” instead of “delete[];” these are not the same! Another problem is the failure to free all rows of a matrix; a loop is needed to free each row by calling “delete[] row[i]” and then another “delete[]” is needed to free the array of pointers. Finally, pointers should be initialized to zero and reset to zero immediately after they are freed. Calls to “delete” or “delete[]” then are as follows:

```
int *some_ints = 0;
some_ints = new int[10];
…
if( some_ints ) delete[] some_ints;
some_ints = 0;
```

The “ctrl” command in G7 failed because an array was initialized improperly. The problem was like this:

```
int int_array[a_class::a_constant];
for(int i=0; i<500; i++) int_array[i] = 0;
….
```

The constant probably was 500 originally but it later was increased to 1000; as a result, only half of the array was initialized to zero. A better way to initialize arrays, which will avoid this problem and probably is faster, is as follows:

```
int int_array[a_class::a_constant] = {0};
char char_array[500] = {'\0'};
float *fptr[10] = {0};
```

This does not work for dynamic arrays and it initializes the entire array to a single specified constant, but it is tidy, fast, and reliable.

If "delete" or "free()" is called for an array that has not been initialized (the pointer points to garbage), or if "fclose()" is called for a file that is not open, the program will crash, or at least the operating system will report an error. This can be avoided by initializing all pointers to zero, and by checking pointers nonzero values before calling commands like "delete," "free()," and "fclose();" following these commands, the pointers should be reset to zero. These comments also apply to pointers to objects and to destructors. I have initialized many pointers in G7 to zero, though there remains work to be done. In all cases, before "delete," "delete[]," "free()," "fclose()," or the Numerical Recipes-style functions (e.g. free_matrix()) are called, the corresponding pointer is checked for a nonzero value; following the call, the pointer is set to zero. The "show" command, for example, failed frequently because of such problems; the "vt.add" program described above has run much more reliably since I made the changes to "show." Some pointers are not initialized yet to zero. In particular, pointers declared in header files must be initialized in the class constructor; this has not been done yet. Also, pointers to objects still must be initialized to zero and checked for nonzero values before destructors are called.

"lint" failed for packed matrices because of a problem with an "if()" statement; first, IsZero() was called for an array element, and then the array index was checked. "if()" arguments are read from left to right; first, the array index must be checked, and if it is valid, then call "IsZero()."

The statement "vc VECTOR_ = 2" returned an error message reporting that vector VECTOR_ was too short; the problem has been fixed.

There remain problems with the "group" and associated commands. The statements

```
group First
1 – 2
```

seem to work if there is no groups.bin file in the directory. If the commands are executed and G7 is restarted, the command fails. Use the command cautiously; if the command fails, try using dynamic groups.

Several G7 commands to store vectors and matrices as ASCII files did not work. "pmpunch" often used the wrong string in function calls (i.e. with the bank letter remaining; e.g. "b.vorm" instead of "vorm"). It also failed to append a ";"to the end of

each section of data; when the file was read, the read command failed because it did not stop reading when it reached the end of the data; instead, it started to read the next command and failed. Also, the written commands that precede the data were not given properly and also caused failure. These and other problems were fixed for "pmpunch" and similar commands.

The command for reading matrices "matin5" lacked code to read the optional width, decimal, and index width parameters. The parameters were hard coded and the command failed for data printed with any other values; these defaults differed from the defaults in the write commands. Similarly, "pras" failed to read the "coltots," "rowtots," "year," and "r|c" parameters. "move" failed to append ".pmx" to the name of a packed matrix, and hence it could not open the packed matrix binary files. Many commands require either the inclusion or exclusion of the data bank for each vector or matrix (e.g. either "b.vec" or "vec"); while this is troublesome for users of G7, I did not extend such code to handle both.

"rowscale" has been updated for use with high frequency Vam files. In addition, several problems with its use with packed matrices (the "pmxrow1" function) have been fixed. One such problem arose from the mixing of arrays and vectors; the array indexes begin at zero, and the vector indexes begin at one. Unfortunately, the author forgot about this. There remains a note from the author stating that the code was not tested. Although I fixed some problems, the output for packed matrices still does not seem to be correct. I therefore disabled the call from "rowscale" to "pmxrow1;" when "rowscale" is called for a packed matrix, a warning is given but nothing else happens. The output of "psras" when used with packed matrices also is suspect, but I did not disable the command; the results should be verified or the command disabled.

Once again, "matty" and "gridty" failed to work (I first reported the problem several years ago, and I think it had been fixed). I think it now is functioning properly, but additional testing is necessary. The character arrays in the Vam class that hold the filenames for titles were of length 30; this proved insufficient for long titles and for paths, and so I tripled the lengths of strings that may be read. Unfortunately, only 30 characters can be stored if compatibility with older versions of Vam is to be maintained. Therefore, if the string lengths exceed 30, a warning is given and the strings are truncated. Also, the Vam constructor did not close the "vam.cfg" file. There is a "punchvec" command in this version of G7 but no "vp" command; I did not fix this problem. Titles in "show" were not displayed properly; the problem was fixed by replacing '\r' and '\n' in each character array with '\0'. The command to print equations, "ipch," failed to print extra parameters when given the "extra" option; this too has been fixed.

Notes for Future Work

G7 continues to use a version of "getint()" and "getfloat()" similar to versions that failed disastrously in Interdyme. The Interdyme version of "getfloat()" returned a zero when reading "the 1.23". It interpreted the "e" as the beginning of a number in scientific notation, and "atof("e\0")" seems to return a zero. I replaced the Interdyme routines with versions that (hopefully) are much more capable. The existing routines in G7 claim to

handle scientific notation properly, and I know of no problems with them. On the other hand, I have not examined them closely, and similar routines were not reliable. Perhaps the existing routines should be replaced by those in Interdyme. The Interdyme routines also should be reviewed. At present, they skip garbage (including digits) until they find a number that they recognize or the end of the file. At some point, it seems, the routine instead should give up and present a warning.

As was mentioned above, there remains additional work with pointers if we are to rid the G7 experience of the annoying access violations and similar problems. However, much already has been done and there now are fewer such problems.

Finally, while existing G7 code and data files are compatible with this version, binary packed matrix data (*.pmx) created by older versions of G cannot be read. The problem is that older versions of the packed matrix class store dates as shorts, which require two bytes of memory. The GDates class stores dates a floats, which require four bytes of memory. Fortunately, Vam already stored dates as floats even though it stored only whole numbers (annual data), and so older binary Vam files should be compatible. Unfortunately, there probably is no way to cram four bytes worth of data into two bytes, and so the shorts have been replaced with floats in the packed matrix class. Hence, packed matrices binaries (even of annual frequency) created by the new version cannot be read by the old, and vice versa. (The old cannot be read by the new because "sizeof(pmfid)" now is larger, and so "fread()" would read too much data. Also, the calls to "fread()" for each date must now read four bytes instead of two. If incompatibility presents a serious problem, some clever work might enable G7 to recognize the version that created the binary file and adjust the calls to "fread()" accordingly.)

Conclusions

Hopefully, these and other bug fixes will make G7 more reliable and usable. The additional capabilities of Vam begin to make possible new types of models. Before high frequency interindustry models can be built, however, more work must be done. For example, IdBuild must be updated to translate G7 code into C++ code. MacFixer and Fixer must be updated to provide fixes at high frequencies. Finally, the Interdyme classes also must be updated for use with high frequency data, and Compare might need to be modified to display the results of these models.

My own work requires that I begin immediately to update several Interdyme classes. Completion of the other work will wait until my own work requires it, until someone else assumes the responsibility, or until demand drives me to finish it (which I will be happy to do).

Please let me know if you have any questions or comments. I value especially your evaluations of the changes I made and of your experiences with them.

August 29, 2003

Ron Horst
Inforum
horst@econ.bsos.umd.edu
horst@inforum.umd.edu