ENGINEERING RESEARCH INSTITUTE
UNIVERSITY OF ICELAND

Pretty-printing matrices in standard Fortran 95 using Dispmodule

Kristján Jónasson

Report VHÍ-02-2008
Reykjavík, May 2008

Report VHÍ-02-2008, Reykjavík May 2008

Kristján Jónasson. *Pretty-printing matrices in standard Fortran 95 using Dispmodule*. Engineering Research Institute, University of Iceland, Technical report VHI-02-2008, May 2008.

Kristján Jónasson, associate professor, Department of Computer Science, Hjardarhagi 4, IS-107 Reykjavik, Iceland. Email: jonasson@hi.is.

Engineering Research Institute, University of Iceland, Hjarðarhagi 4, IS-107 Reykjavík, Iceland

ABSTRACT

A standard Fortran 95 module for printing scalars, vectors and matrices to external files is described. The module can display variables of default kind of all intrinsic types (integer, real, complex, logical and character), and add-on modules are provided for data of non-default kind. The main module is self-contained and incorporating it only requires that it be compiled and linked with a program containing a 'use dispmodule' statement. A generic interface and optional parameters are used, so that the same subroutine name, `DISP`, is used to display items of different data type and rank, irrespective of display options. The subroutine is quite versatile, and hopefully can improve Fortran's competitiveness against other array programming languages. The module also contains a function `TOSTRING` to convert numerical scalars and vectors to strings. The module is distributed under a public domain licence and can be freely used, even in commercial packages.

Key Words: *Fortran 95, Matrix pretty-printing, Matrix printing, Output utilities, Array programming language*

ÁGRIP

Lýst er einingu eða forritasafni skrifuðu í Fortran 95 sem ætlað er til að skrifa tölur, vigra og fylki í skrár eða birta þessa hluti á skjá. Einingin (ásamt viðbótareiningum) getur birt breytur af öllum gagnatögum sem eru innbyggð í Fortran 95 (heiltölur, kommutölur, tvinntölur, rökstærðir og táknastrengi). Einingin er sjálfstæð og til að nota hana nægir einfaldlega að þýða hana og tengja við forrit sem inniheldur *use dispmodule* setningu. Margræðni og valfrjálsir stikar eru notaðir, þannig að sama undirforritsnafnið, `DISP`, er notað til að birta stærðir af mismunandi gagnatögum og víddum, óháð því hvaða valkostir eru notaðir við birtinguna. Einingin er fjölhæf og getur ef vel tekst til bætt samkeppnishæfni Fortrans gegn öðrum fylkjaforritunarmálum. Einingin inniheldur ennfremur fallið `TOSTRING` sem breytir talnabreytum í táknastrengi. Forritasafninu er dreift með opnu leyfi sem leyfir frjálsa notkun, jafnvel í hugbúnaði sem er seldur.

CONTENTS

1. INTRODUCTION

One of the reasons for the great popularity of numerical computation systems such as Matlab [Moler 2004], S-plus [Krause and Olson 2000] and their open source clones (e.g. Octave [Eaton 2002], Scilab [Mrkaic 2001] and R [Chambers 2007]) stems from the ability of these systems to treat vectors and matrices as fundamental data types. At the heart of these systems are languages that may be classified as *array programming languages*. The first array programming language, APL, was developed in 1957 and it, along with its successor, J, still enjoy considerable popularity [Iverson 1991]. A feature of all these languages is that the syntax for displaying arrays in traditional mathematical format is very simple. As an example the Matlab command `disp(A)` will pretty-print the matrix `A` using a fairly versatile default format. The Matlab commands `A = exp([3,2;-3,-2]); disp(A)` will display:

```
   20.0855    7.3891
    0.0498    0.1353
```

It may be argued that the array programming features account for the widespread use of these languages for writing prototype programs, which are later superseded by programs written in compiled high level languages such as C or Fortran.

In the 1990's array programming in Fortran became possible with compilers for Fortran 90 and Fortran 95 [ISO/IEC 1997]. The future of Fortran is in many respects bright [Reid 2003 and 2006, Metcalf et al. 2003, ISO/IEC 2004]. However, even with recent enhancements Fortran still lacks the ability to print matrices with a concise command. To display a general matrix A, one needs a sequence of statements such as:

```
do i=1,size(A,1)
  write(*,'(20G12.4)') A(i,:)
end do
```

which for the matrix given above will produce a rather mis-aligned output:

```
    20.09        7.389
  0.4979E-01   0.1353
```

Using an F edit descriptor an aligned display similar to the one given by Matlab is obtained, but this comes at a cost, as matrices with large elements can no longer be displayed. Fortran also offers "asterisk-format" for writing with default format. To take an example, the statement "`write(*,*) A`" might display `20.085537 0.049787067 7.389056 0.13533528`. However, aligned output of matrices with multiple rows is not possible using this, and in addition the result is compiler dependent.

The purpose of the current subroutine package is to remedy this deficiency, by providing a module written in standard Fortran, that will, with minimal fuss, display a vector or a matrix with sensible default format. At the same time the module offers considerable flexibility in controlling the format of the output (much more than Matlab's `disp`). It is possible to specify the edit descriptor, the displayed items may be labelled, rows and columns may be numbered, and the output may be directed to a file, to the screen, or even (when calling Fortran from Matlab) to the Matlab command window. Several matrices / vectors may be written side by side.

The package is especially useful for debugging purposes, and for preliminary display of numerical results, but it is general enough that it may in many cases be used for final display of such results. It uses a generic interface and has optional parameters, so that the same subroutine name, `DISP`, is used to display items of different data types and ranks, with or without labels, and using default or specified format (in Fortran terminology scalars have rank 0, vectors rank 1, matrices rank 2 etc.). All the intrinsic data types of Fortran are supported, i.e. integer, real (both single and double precision), complex, logical and character. The module is accompanied by documentation in a comprehensive user manual [Jonasson 2008] in four different formats (plain text, html, pdf and Microsoft Word). The package may be downloaded from the author's web page, www.hi.is/~jonasson.

Among related earlier work one can mention the Fortran 95 package *Matran* [Stewart 2003], which contains a pretty print subroutine for items of type *Matran matrix*, but that routine is of much more limited scope than the present one. More versatile matrix printing routines are in the NAG Libraries

[NAG 2000 and 2006], but these are not free, and even if they have a few features that are absent in the current package, the latter is in most respects more flexible.

2. PACKAGE COMPONENTS

The present package consists of one principal Fortran 95 module, `Dispmodule` which handles the display of data types of default kind (that are guaranteed to exist by the Fortran standard), and several add-on modules to handle data types of non-default kind (such as byte integers and quadruple precision reals). The main procedures are the subroutine `DISP` which is used to display a scalar, vector or matrix, and the function `TOSTRING` which is used to change scalars or vectors into character strings. In addition there are four subroutines to control settings for how the data are edited and one function to retrieve settings for `DISP`.

2.1 Subroutine DISP

Simple ways to call the subroutine `DISP` are:

```
CALL DISP(X)
CALL DISP(TITLE, X)
CALL DISP(X, FMT)
CALL DISP(TITLE, X, FMT)
```

where `X` is the item to be displayed, `TITLE` is a character string used to label the displayed item, and `FMT` is an edit descriptor to control the display. Assume that $a_{ij} = \exp(i + j - 1)$ and $b_{ij} = \exp(i^j)$, i, j = 1,…,3. Then examples of calls are `CALL DISP('A = ', A)` which will write out:

```
A =  2.718   7.389   20.086
     7.389  20.086   54.598
    20.086  54.598  148.410
```

`CALL DISP(B)` which displays:

```
2.71828E+00  2.71828E+00  2.71828E+00
7.38906E+00  5.45981E+01  2.98096E+03
2.00855E+01  8.10308E+03  5.32048E+11
```

and `CALL DISP(A(1:2,:),'F0.5')` which displays

```
2.71828   7.38906  20.08554
7.38906  20.08554  54.59815.
```

A call with a complete list of arguments is `CALL DISP(TITLE, X, FMT, FMT_IMAG, ADVANCE, DIGMAX, LBOUND, ORIENT, SEP, STYLE, TRIM, UNIT, ZEROAS)`. All the arguments are optional, and the purpose, data type and possible values of each argument are given in Table 1 in the user manual. Not all arguments are compatible with all data types or ranks of `X`. It is for instance only possible to specify `FMT_IMAG` for complex `X` and `ORIENT` for vectors. Argument association for arguments after (or starting with) `FMT` will usually be realized with argument keywords, e.g. `CALL DISP('A=', A, UNIT=3, ZEROAS='.')`. Examples demonstrating some of the possibilities are:

```
CALL DISP('MATRIX', A, STYLE='UNDERLINE & NUMBER', UNIT=8, DIGMAX=4)
```

which with the matrix *A* given above will send the following to a formatted file on unit 8,

```
        MATRIX
--------------------
     1     2      3
1   2.7   7.4   20.1
2   7.4  20.1   54.6
3  20.1  54.6  148.4
```

and

```
K = [-3,0,12,14,0]
CALL DISP('K', K, STYLE='PAD', ORIENT='ROW', SEP=' ', ZEROAS='.')
```

which will display on the asterisk unit (usually the screen):

```
------K-----
-3 . 12 14 .
```

Further examples and details can be found in the user manual.

2.2 The function TOSTRING

Many programming languages have built-in functions that change numbers to strings. It is for instance possible with Java to write

```
System.out.println("The square of " + x " is " + x*x);
```

(Java also has a function, `Float.toString` that may be used instead). In Matlab one may write

```
disp(['The square of ' num2str(x) ' is ' num2str(x*x)])
```

If `x` is 1.5, both commands display "The square of 1.5 is 2.25". The function `TOSTRING` offers similar functionality. The statement

```
call disp('The square of '//tostring(x)//' is '//tostring(x*x))
```

will produce the same output as the Java and Matlab commands. As shown in the user manual it is possible to achieve a similar effect in Fortran using internal files and list-directed output, but this is cumbersome and the result is compiler-dependent. `TOSTRING` can change numeric and logical scalars and vectors into character strings, and it is possible to specify how to format the string. One application of `TOSTRING` is to create variable format, e.g.:

```
fmt = '(F'//tostring(w)//'.'//tostring(d)//')'
write(*,fmt) A
```

For further examples and description, see the user manual.

2.3 Changing and retrieving settings for DISP and TOSTRING

The default settings used by `DISP` may be changed with the subroutine `DISP_SET`, which with a complete list of arguments may be called with:

```
CALL DISP_SET(ADVANCE, DIGMAX, MATSEP, ORIENT, SEP, STYLE, UNIT, ZEROAS)
```

All the arguments are optional, and the effect of calling `DISP_SET` with a specified argument present is to change the default value for the corresponding argument of `DISP`. For instance, after calling `CALL DISP_SET(UNIT=7)`, subsequent `DISP`-calls will send output to a file connected to unit 7. The only argument that is unique to `DISP_SET` is `MATSEP` ('matrix separator') which is a character string that is used to separate items displayed with `ADVANCE='NO'` (the default separator is a string with three spaces). To take an example, the sequence of calls:

```
CALL DISP_SET(MATSEP = ' | ')
CALL DISP([11,12,13], ADVANCE='NO')
CALL DISP([.TRUE., .FALSE., .TRUE.], ADVANCE='N0')
CALL DISP(['A','B','C'])
```

will display:

```
11 | T | A
12 | F | B
13 | T | C
```

Similarly, there is a function `TOSTRING_SET` to change default-settings for `TOSTRING`. There are also functions to return the settings to the original (or *factory*) defaults. These are called `DISP_SET_FACTORY` and `TOSTRING_SET_FACTORY`. The current default settings for `DISP` may be retrieved with the function `DISP_GET` which returns a structure (say `DS`) of type `DISP_SETTINGS` (declared in `Dispmodule`) with these settings. The settings may be re-applied to `DISP` by calling `DISP_SET(DS)`. The user manual contains details.

2.4 Testing

The package contains a test-program, `Test_Dispmodule`, which applies modern testing techniques to ascertain the correctness of `Dispmodule`. The test-program calls all the procedures of the module, and all arguments are tried. Testing of all possible data types for `X` is allowed for. For arguments that have a limited range of allowed values, all these values are tried. In addition all the examples given in the user manual and in this paper are tested. It is however impossible to test all possible combinations of `X` data type, arguments and argument values, so one must be content with a partial test, accompanied by examination of the source code of the module(s) and the test program.

There is a parameter declared at the beginning of the test program to control the verbosity level, or amount of output. It has three possible settings: minimal output, a short report, and a detailed report (including the result of all `DISP` and `TOSTRING` invocations). In all cases the report ends with “OK” if all tests are passed. Further instructions for operation of the test program are given in the user manual.

The package has been tested successfully with recent versions of the following compilers: under Microsoft Windows: *gfortran*, *g95*, *f95* from NAG, and *ifort* from Intel; under Ubuntu Linux: *gfortran* and *g95*; under Sun Solaris: *g95*; under Suse Linux: *ifort*. It passes tests of g95, gfortran and ifort of conforming to the Fortran 95 standard.

2.5 Package limitations and assumptions on the compiler

Among major limitations of the module is that `DISP` cannot display very wide matrices, as there is no provision for breaking matrices (or matrix rows) vertically. The compilers tried can however all handle record lengths of 1024 characters or more, and since the output of `DISP` is intended for human reading this should not be too serious drawback. Another important limitation is that displaying arrays of rank ≥ 3 is not supported. Some minor limitations are that *zeroas* is not supported for complex variables, `TOSTRING` cannot handle character variables and 2-byte logicals are not directly supported.

Among the things which the NAG matrix printing routines can do and `DISP` cannot, is writing matrices with special storage schemes, such as triangular, band or symmetric. Another ability of the NAG routines missing from `DISP` is specifying character strings for matrix row and column labels.

An assumption on the compiler in the design of the package is, that if `A` and `B` are real variables with |`A`| < |`B`|, and writing `B` with F*w*.*d* edititing does not produce *w* asterisks, then neither does writing `A`. One possible problem that hopefully does not occur involves compilers that automatically prepend a plus-sign when writing positive numbers. Every effort has been made to use SS editing to suppress potential plus-signs, but no compiler that defaults to SP editing is known to the author.

The feature of writing matrices side by side (with `ADVANCE='NO'`) is implemented by allocating memory for the blocks waiting to be written out. To avoid memory leaking, care must be taken to clear the output queues on each unit (by calling `DISP` with `ADVANCE='YES'` in effect) before exiting the program.

REFERENCES

CHAMBERS, J. M. 2007. *Software for Data Analysis: Programming with R*. Springer-Verlag, New York.

EATON, J. W. 2002. *GNU Octave Manual*. Network Theory Limited, Bristol.

ISO/IEC. 1997. *Information Technology---Programming Languages---Fortran-Part 1: Base Language (ISO/IEC 1539-1:1997)*. ISO, Geneva.

ISO/IEC. 2004. *Information Technology---Programming Languages---Fortran-Part 1: Base Language (ISO/IEC 1539-1:2004)*. ISO, Geneva

IVERSON, K. E. 1991. A personal view of APL, *IBM Systems Journal 30*, 4, 582−593.

JONASSON, K. 2008. *Dispmodule User Manual*. Report VHI-01-2008, Engineering Research Institute, University of Iceland.

KRAUSE, A. AND OLSON, M. 2000. *The Basics of S and S-PLUS (second ed.)*, Springer-Verlag, New York.

METCALF, M., REID, J. K. AND COHEN, M. 2004. *Fortran 95/2003 explained*. Oxford University Press.

MOLER, C. 2004. *Numerical Computing with MATLAB*, SIAM, Philadelphia.

MRKAIC, M. 2001. Scilab as an econometric programming system, *J. Appl. Econometr. Systems 16*, 4, 553−559.

NAG 2000. *NAG Fortran 90 Library Manual*. Numerical Algorithms Group, Oxford.

NAG 2006. *NAG Fortran Library Manual*. Numerical Algorithms Group, Oxford.

REID, J. K. 2003. The future of Fortran. *Computing in Science and Engineering, 5*, 4, 59−67.

REID, J. K. 2006. Fortran is getting more and more powerful. In *Applied Parallel Computing, State of the Art in Scientific Computing*, ed. Dongarra, J., Madsen, K. and Wasniewski, J., Lecture Notes in Computer Science 3732, Springer, 33−42.

STEWART, G. W. 2003. *MATRAN: A Fortran-95 Matrix Wrapper*, University of Maryland, Department of Computer Science Technical Report 4522.