TRIP

First steps with numerical matrices

M. Gastineau

gastineau@imcce.fr

Observatoire de Paris - IMCCE
Astronomie et Systèmes Dynamiques
77, avenue Denfert Rochereau
75014 PARIS

22 April 2013

Introduction

This tutorial explains how to use numerical matrices in TRIP.

- Numerical matrices are rectangular array of numbers. $M = \begin{pmatrix} 1 & -5 & 13 \\ 6 & 9 & 11 \end{pmatrix}$
- Numerical matrices contain real or complex numbers in double precision, quadruple precision and multiprecision.
  Computations are performed in double precision by default.
- Numerical matrices have a static size.
- Numerical matrices require explicit declaration only if a specific element is assigned.

Initialization

- Initialization with a default size

```
/* real matrix with 2 rows and 3 columns */
matrixR M0([1:2, 1:3]);
/* complex matrix with 2 rows and 2 columns */
matrixC M1([1:2, 1:2]);
```

$$M0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, M1 = \begin{pmatrix} 0+0i & 0+0i \\ 0+0i & 0+0i \end{pmatrix}$$

- Identity matrix

```
/*I_3 : Identity matrix of size 3 */
I3 = identitymatrix(3);
```

$$I3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Initialization with several values

```
/* real matrix with 2 rows and 3 columns */
M0=matrixR[1,-5,13:6,9,11];
/* complex matrix with 2 rows and 2 columns */
M1=matrixC[i,-2+i:7-i,-i];
```

$$M0 = \begin{pmatrix} 1 & -5 & 13 \\ 6 & 9 & 11 \end{pmatrix}, M1 = \begin{pmatrix} i & -2+i \\ 7-i & -i \end{pmatrix}$$

Access elements

- Access to a single element
  To access to an element of a numerical matrix, $[i, j]$ must be appended to the matrix's name.
  i is an integer which specifies the row and the integer j specifies the column of the matrix.

```
M0=matrixR[1,-5,13
           :6,  9,11];
A = M0[1,3];
B = M0[2,2];
```

$A = 13, B = 9$

- Access to multiple elements
  To access to multiple elements in a numerical matrix, $[istart : istop : istep, jstart : jstop : jstep]$ must be appended to the matrix's name.
  istart, *istop*, *istep*, *jstart*, *jstop* and *jstep* are integers.

  One or more of the integer values may be omitted :
  - If *istart* or *jstart* is omitted, it will be assumed to be equal to the lower bound of the corresponding dimension.
  - If *istop* or *jstop* is omitted, it will be assumed to be equal to the upper bound of the corresponding dimension.
  - If *istep* or *jstep* is omitted, it will be assumed to be equal to 1.

```
M0=matrixR[1,-5,13:6,9,11];
A = M0[:,:2];  /* <=>    A=M0[1:2:1,1:2:1];  */
B = M0[::2,::2];  /* <=>
B=M0[1:2:2,1:3:2];  */
```

$A = \begin{pmatrix} 1 & -5 \\ 6 & 9 \end{pmatrix}, B = \begin{pmatrix} 1 & 13 \end{pmatrix}$

Assign a single element

To assign an element of a numerical matrix, $[i, j]$ must be appended to the matrix's name.
i is an integer which specifies the row and the integer j specifies the column of the matrix.

```
/* Assign real numbers to the matrix M0 */
matrixR M0([1:2,1:3]);
M0[1,1] = 7;
M0[1,2] = 5;
M0[2,1] = 9;
M0[2,2] = 8;
M0[2,3] = 11;
```

$$M0 = \begin{pmatrix} 7 & 5 & 0 \\ 9 & 8 & 11 \end{pmatrix}$$

```
/* Assign complex numbers to the matrix M0 */
matrixC M1([1:2,1:3]);
M1[1,1] = 7+4*i;
M1[1,2] = 5-11*i;
M1[2,1] = 9+3*i;
M1[2,2] = 8+2*i;
M1[2,3] = 11+3*i;
```

$$M0 = \begin{pmatrix} 7+4\imath & 5-11\imath & 0 \\ 9+3\imath & 8+2\imath & 11+3\imath \end{pmatrix}$$

Assign multiple elements (1/2)

The assignment of multiple elements depends on the type of the right value (value after the symbol =).
The right value can be :

- a numerical vector as a column

The right vector's size must be the same as the number of matrix's row.
The command "M[istart:istop:istep,j] = A;" is equivalent to the command
"for ii=istart to istop step istep { M[ii,j] = A[ii]; }"

```
/* Assign -t to the first column of M */
/* Assign +t to the last column of M */
t=1,3;
matrixR M([1:3,1:2]);
M[:,1] = -t;
M[:,2] = t;
```

$$M = \begin{pmatrix} -1 & 1 \\ -2 & 2 \\ -3 & 3 \end{pmatrix}$$

- a numerical vector as a line

To assign a numerical vector as a line of a matrix, the function 'transposematrix' should be used.

```
/* Assign -t to the first line of M */
/* Assign +t to the last line of M */
t=1,3;
matrixR M([1:2,1:3]);
mt = matrixR(t);
M[1,:] = transposematrix(-mt);
M[2,:] = transposematrix(mt);
```

$$M = \begin{pmatrix} -1 & -2 & -3 \\ 1 & 2 & 3 \end{pmatrix}$$

Assign multiple elements (2/2)

- a number

The command "vr[istart:istop:istep, jstart,jstop,jstep] = r;" is equivalent to the command "for ii=istart to istop step istep { for j=jstart to jstop step jstep { M[ii,j] = r; }; };"

```
/* Assign -1 to the first column of MR  */
/* Assign +1 to the last column of MR */
matrixR MR([1:2,1:3]);
MR[:,1] = -1;
MR[:,2] = +1;

/* Assign -1-i to the first line of MC  */
/* Assign +1+i to the last line of MC */
matrixC MC([1:2,1:2]);
MC[1,:] = -1-i;
MC[2,:] = +1+i;
```

$$MR = \begin{pmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \end{pmatrix}, MC = \begin{pmatrix} -1-i & -1-i \\ 1+i & 1+i \end{pmatrix}$$

Arithmetics and standard functions

The matrix multiplication between two matrices is performed using the operator &∗ .

The basic arithmetic, such as $+, *, -, /$ between 2 matrices and standard functions, such as cos, sin, ..., are performed on each element of the matrices. The matrices must have the same dimension.

Basic arithmetic between a matrix and a number is available such $+, *, -, /$.
If the matrix or the number contains complex number, then the resulting matrix is a matrix of complex numbers.

The invert of a matrix is computed by the function 'invertmatrix'.
The matrix transposition is computed by the function 'transposematrix'.

```
A=matrixR[1,2:
          3,4];
B=matrixR[5,6:
          7,8];
/* perform the matrix multiplication */
C = A&*B;
/* perform a multiplication of each element of A and B   */
D = A*B;
/* invert of A */
E = invertmatrix(A);
```

$$C = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}, D = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}, E = \begin{pmatrix} -2 & 1 \\ 1.5 & -0.5 \end{pmatrix}$$

I/O functions - Write text files

- Write to the screen
  The content of numerical matrix could be written to the screen with the command 'writes'.
  You can add a format (C-format) to specify the type of output.

```
/* write M  */
M=matrixR[1,2:
          3,4];
writes(M);
/* write with a specific format */
writes("%20.8E  %20.8E\n", M);
```

- Write ASCII files
  If matrices don't have the same size, the columns of the shortest matrices will be filled with 0.

```
/* write M0 in the column 1 and 2, M1 in column 3 and 4 and 5 */
M0 = matrixR[1,2:
             3,4];
M1 = matrixR[1,-5,13:
             6, 9,11];
write(file2.dat, M0,M1);
/* write with a specific format */
write(file2.dat, 5*"%20.8E "+"\n", M0,M1);
```

You could write only some lines of the numerical matrices with the range specifier [*istart* : *istop* : *istep*] where it specifies the first line to write, the last line and the step (must be $\geqslant 1$).

```
/* write the line 1 and 3 of M */
M = matrixR[1,-5:
            13,6:
            9,11];
write(file2.dat, [::2], M);
/* write with a specific format and  M[1:2] */
write(file2.dat, [1:2], "%20.8E  %20.8E\n", M);
```

I/O functions - Read text files

- Read ASCII files without a format
  ASCII files could be loaded into numerical matrices using an array of numerical vectors.
  You just have to declare the vectors of real or complex numbers before calling 'read' with the specified record format.
  Then you convert the array of vectors to a matrix using 'matrixR' or 'matrixC'.

```
/* read a real matrix of 2 columns and store to MR */
vnumR VR[1:2];
read(file4.dat, VR);
MR = matrixR(VR);

/* read a complex matrix of 2 columns and store to MC */
vnumC VC[1:2];
read(file5.dat, VC);
MC = matrixC(V);
```

You could read only some parts of the file with the range specifier [*indicestart* : *indicestop* : *indicestep*] where it specifies the first line to read (must be $\geqslant 1$), the last line and the step line (must be $\geqslant 1$).

```
/* Read  from the line 1 to 100 */
vnumR VR[1:100];
read(file1.dat, [1:100], VR);
/* MR is a 100x100 matrix */
MR = matrixR(VR);
```

I/O functions - Binary files

- Read binary files
  Binary files could be loaded into numerical matrices of real numbers using an array of numerical vectors. You just have to declare the vectors of real numbers before calling 'readbin' with the specified record format.
  Data in the file could be stored as integers (on 2, 4, 8 bytes) or as real numbers (on 4, 8, 16 bytes).
  If the data aren't in the same order (endianess) as your computer architecture, you could specify the endianess of this file by setting the global variable '_endian'. Real numbers must be encoded as IEEE number.

```
/* read a matrix of 2 columns and store to M */
vnumR V[1:2];
readbin(file4.dat,"%8g%8g", V);
M = matrixR(V);
```

- Write binary files
  Matrices could be written to binary files with the specified record format. The format must contain the number of specifiers must be the same as the number of columns. Numbers are encoded in the same order byte as specified by the global variable '_endian'.

```
/* Store a 2x3 matrix  */
_endian = big;
M = matrixR[1,-5,13:6,9,11];
writebin(file4.dat,size(M,2)*"%g", M);
```

Array of numerical matrices

- Declaration
  You could have arrays of numerical matrices. The number of dimensions of the array is limited to 10. The indices could be any integer between -2^{31} to $2^{31}-1$.
  Arrays of numerical matrices require explicit declarations.

```
/* Build three arrays :
 A : array (1-dimentional) of 4 real matrices of size 10x10
 B : array (2-dimentional) of 15 complex matrices of size 3x3
 C : array (1-dimentional) of 5 complex matrices of size 8x8 */
matrixR A[0:3]([1:10,1:10]);
matrixC B[1:15]([1:3,1:3]);
matrixC C[1:5]([0:7,0:7]);
```

  The size of the array can't be change after declaration : If you declare again the array with a different size, its content will be destroyed.

- Accessing and assignment matrices
  To access an matrix in the array, you just have to append indices between [] to the name of the array. Each dimension must separated by a comma.
  To access an element in a numerical matrix of the array, you just have to append two brackets : the first one for the indice of numerical matrix in the array and the second one for the 2 indices of the number in the numerical matrix.

```
/* build an array of 3 matrices of size 2x2 */
matrixR A[1:3]([1:2,1:2]);
/* assign values in the matrix */
for j=1 to 2 { A[1][j,j]=2*j$ A[2][j,j]=3*j$ };
A[3] = A[2]+A[1];
```

$$A_1 = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}, A_2 = \begin{pmatrix} 3 & 0 \\ 0 & 6 \end{pmatrix}, A_3 = \begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix}$$

- Functions accepting array of numerical matrices : I/O functions, operators (+ - * /), ...

Numerical precision

- Setting the numerical precision
  The global variable '_modenum' specifies the current numerical precision.

  By default, computations are performed in double precision (_modenum = NUMDBL).
  Computations could be performed in quadruple precision (_modenum = NUMQUAD) and multiprecision (_modenum= NUMFPMP).
  The number of significant decimal digits for the multiprecision floating-point numbers is specified by the global variable '_modenum_prec'.

  The numerical precision could be changed at any time. Already initialized elements of the matrices are not affected by this change until they are reinitialized.

```
/* Build two matrices :
 Md : matrix with double-precision
      real numbers
 Mq : matrix with quadruple-prec.
     real numbers */

_modenum= NUMDBL;
matrixR Md([1:10,1:10]);

_modenum=NUMQUAD;
matrixR Mq([1:10,1:10]);
```

```
/* Build two matrices  :
Mmp100 : matrix with 100
                 decimal digits
Mmp30  : matrix with 30
     decimal digits */

 _modenum=NUMFPMP;
_modenum_prec = 100;
matrixR Mmp100([1:10,1:10]);

_modenum=NUMFPMP;
_modenum_prec = 30;
matrixR Mmp30([1:10,1:10]);
```