TRIP

First steps with numerical vectors

M. Gastineau
gastineau@imcce.fr

Observatoire de Paris - IMCCE
Astronomie et Systèmes Dynamiques
77, avenue Denfert Rochereau
75014 PARIS
13 March 2013

Introduction

This tutorial explains how to use numerical vectors in TRIP.

- Numerical vectors are column vectors. $v = \begin{pmatrix} 0 \\ 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{pmatrix}$
- Numerical vectors contain real or complex numbers in double precision, quadruple precision and multiprecision.
  Computations are performed in double precision by default.
- Numerical vectors have a dynamic size.
- Numerical vectors require explicit declaration only before the commands read, readbin and resize.
- Vector indices start always at 1.

Initialization

- Arithmetic progression of real numbers : *vector* $=$ *realstart, realstop, realstep;*

```
/* v0 is a vector with 10 real numbers */
v0=0,10,1;
/* v1 is a vector with 5 real numbers */
v1=-12,-4,-2;
```

$$v0 = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 10 \end{pmatrix} \quad v1 = \begin{pmatrix} -12 \\ -10 \\ -8 \\ -6 \\ -4 \end{pmatrix}$$

- Initialization with a default size

```
/* v0 is a vector with 3 real numbers */
vnumR v0([1:3]);
```

$$vr = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

- Initialization with same values

```
/* vr is a vector with 5 real numbers */
vnumR vr;
resize(vr,5,1);
/* vz is a vector with 5 complex numbers */
vnumC vz;
resize(vz,5,3+4*i);
```

$$vr = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad vz = \begin{pmatrix} 3+4\imath \\ 3+4\imath \\ 3+4\imath \\ 3+4\imath \\ 3+4\imath \end{pmatrix}$$

The commands "vnumR vr;" and " VnumC vz;" build only empty vectors. The size of the vectors are equal to 0.

- Initialization with several values

```
/* vr is a vector with 5 real numbers */
vr=vnumR[3:-5:4:1:7];
/* vz is a vector with 5 complex numbers */
vz=vnumC[i:-2+i:7-i:-i:0];
```

$$vr = \begin{pmatrix} 3 \\ -5 \\ 4 \\ 1 \\ 7 \end{pmatrix} \quad vz = \begin{pmatrix} \imath \\ -2+\imath \\ 7-\imath \\ -\imath \\ 0 \end{pmatrix}$$

Arithmetics and standard functions

Basic arithmetic between 2 vectors is available such as $+, *, -, /, **$.
If the size of 2 vectors isn't the same, the resulting vector has a size equal to the greatest size of the 2 vectors.
If one of the vector is a vector of complex numbers, the resulting vector is a vector of complex numbers.

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

Standard functions on a numerical vector are available, such as cos, sin, arg, abs, sum, max, min, sqrt, imin, imax, ...

```
/* a, b and c will have the same size as t */
/* a,b,c will be vectors of real numbers. */
t = 0, pi, pi/100;
a = cos(t);
b = sin(t);
c = a*b;
/* d will be vectors of complex numbers. */
d = i*c;
```

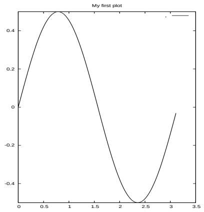
Plots

The function plot(X, Y) plots vector X versus vector Y. 3D plots could be done with an extra third argument. Plots will be done by Gnuplot or Grace depending on the value of the global variable '_graph' (by default, gnuplot).

```
/* Plot cosinus between 0 and pi */
X = 0,pi,pi/100;
plot(X, cos(X));
```

To send specific gnuplot commands, you have to insert gnuplot command between the statements "gnuplot;" "end;"
To send numbers to gnuplot or grace, you have to convert them in a string with the command 'str'. Then you can use this string in gnuplot command by inserting its name between the character '@'.

```
X = 0,pi,pi/100;
Y = cos(X)*sin(X);
ymin = str(min(Y));
ymax = str(max(Y));
gnuplot;
set yrange[@ymin@ : @ymax@];
set title 'My_first_plot'
set data style line
end;
plot(X, Y);
```



Postscript file, containing the plot, could be generated with the command 'plotps' and plotps_end'.

```
/* 3D Plot in a postscript file */
X = 0,pi,pi/100;
plotps "file1.ps";
plot(X, cos(X), sin(Y));
plotps_end;
```

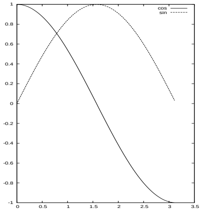
Multiple plots

The function plot can stack two or more plots one over the other. Each plot are embraced by parentheses. The list is separated by comma. The command plot((X1,Y1),(X2,Y2)) displays the plot (X2,Y2) stacked above the plot (X1,Y1).

```
/* Plot cosinus and sinus between 0 and pi */
X = 0,pi,pi/100;
plot((X, cos(X)),(X, sin(Y)));
```

If one or more components of the plot is an array of numerical vectors, one plot per component will be performed.

```
X = 0,pi,pi/100;
vnumR TY[1:2];
TY[1] = cos(X);
TY[2] = sin(X);
dim name[1:2];
name[1]="␣w␣l␣title␣'cos'";
name[2]="␣w␣l␣title␣'sin'";
plot(X, TY,name);
/*similar as plot((X,TY[1]), (X, TY[2]))*/
```



I/O functions - Read text files without a format

- Read ASCII files without a format
  The Reading of the columns of numbers is done by the function 'read'. Each column is loaded in a numerical vector. You just have to declare empty numerical vector before calling it. The size of the vector will be adjusted to the number of read lines.

```
/* Read the file 'file1.dat' which contains 4 columns */
/* t will contain the 1st column, x : 2nd, z : 3rd+i*4th */
vnumR t, x;
vnumC z;
read(file1.dat, t, x, z);
```

  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 4 to 100 every 2 lines */
vnumR t, x;
vnumC z;
read(file1.dat, [4:100:2], t, x, z);
```

  You can read only some columns in a file by just specifying the column for the vector.

```
/* z will contain the column : 2nd+i*3rd, x : 7th, y : 8th */
vnumC z;
vnumR x, y;
read(file2.dat,  (z,2,3), (x,7), y);
```

I/O functions - Read text files with a format

- Read ASCII files with a format
  The Reading of the columns of numbers or strings is done by the function 'read'. Each column is loaded in a numerical vector or an array of strings. You just have to declare empty numerical vector before calling it. The arrays of strings does not require to be declared before. The size of the vectors and arrays will be adjusted to the number of read lines. The format must be specified in a string object.

  The format specifier %g, %e %E could be used for column of numbers. The format specifier %s is used for column of strings. The number of characters inside the column may be specified using a number between the character % and the character g,e,E,s. In that case, blank characters are ignored.

```
/* Read the file 'file1.dat' which contains 4 columns  */
/* t will contain the 1st column with numbers, name :
   2nd column with strings and z : 3rd column with  numbers */
vnumR t, x;
fmt="%g %s %g";
read(file1.dat, fmt, t, name, x);
```

  You can specify the number of characters of each column.

```
/* Read the file 'file1.dat' which contains 4 columns */
/* 1st column with a number on 2 characters,
   2nd column with a string of 5 characters,
   3rd column with a number on 7 characters */
vnumR t, x;
fmt="%2g%5s%7g";
read(file1.dat, fmt, t, name, x);
```

I/O functions - Write text files

- Write ASCII files
  Writing numerical vectors is as easy as reading. You can add a format (C-format) to specify the type of output.
  If vectors don't have the same size, the columns of the shortest vectors will be filled with 0.

```
/* write X in the 1st column, Y: 2nd */
X = 0, pi, pi/100;
Y = cos(X);
write(file2.dat, X, Y);
/* write with a specific format */
write(file2.dat, "%20.8E  %20.8E\n", X, Y);
```

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

```
/* write X[1:size(X):2] and Y[ 1:size(Y):2] */
X = 0, pi, pi/100;
Y = cos(X);
write(file2.dat, [::2], X, Y);
/* write with a specific format and  X[10:50] and Y[10:50] */
write(file2.dat, [10:50], "%20.8E  %20.8E\n", X, Y);
```

- Write to the screen
  In a similar way, the content of numerical vectors could be written to the screen with the command 'writes'.

```
/* write X in the 1st column, cos(X): 2nd  to the screen */
X = 0, pi, pi/100;
writes(X, cos(X));
/* write with a specific format */
writes("%20.8E  %20.8E\n", X, cos(X));
```

I/O functions - Binary files

- Read binary files
  Binary files could be loaded in numerical vectors of real numbers. 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 records which contains :
 one 4-byte integer and one 8-byte real */
vnumR X,Y;
readbin(file4.dat,"%4d%8g", X, Y);

/* read records which contains :
 one 4-byte integer and three 8-byte reals */
vnumR X,Y1, Y2, Y3;
readbin(file4.dat,"%4d%8g%8g%8g", X, Y1, Y2, Y3);
```

- Write binary files
  Real vectors could be written to binary files with the specified record format. Numbers are encoded in the same order byte as specified by the gobal variable '_endian'.

```
/* Store 10 records which contains :
 one 4-byte integer and one 8-byte real */
_endian = big;
X = 1,10;
writebin(file4.dat,"%4d%8g", X, cos(X));
```

Access elements

- Access to a single element
  To access to an element of a numerical vector, [*indice*] must be appended to the vector's name.
  indice is an integer.

```
/* vr is a vector with 5 real numbers */
vr = vnumR[3:-5:4:1:7];
/* Affect the third element of vr in A */
A = vr[3];
/* Affect the fifth element of vr in B */
B = vr[A+2];
```

$A = 4$, $B = 7$

- Access to multiple elements
  To access to multiple elements in a numerical vector, [*indicestart* : *indicestop* : *indicestep*] must be appended to the vector's name.
  indicestart, *indicestart*, *indicestop* are integers.

  One or more of the integer values may be omitted :
  - If *indicestart* is omitted, it will be assumed to be equal to 1.
  - If *indicestop* is omitted, it will be assumed to be equal to the size of the numerical vector.
  - If *indicestep* is omitted, it will be assumed to be equal to 1.

```
vr = -10,-1,-1;
/*extract the first five elements of vr */
A = vr[:5]; /* <=>   A = vr[1:5:1]; */
/*extract elements of vr at indices 3,5,7,9*/
B = vr[3::2]; /* <=>  B = vr[3:size(vr):2];*/
```

$$A = \begin{pmatrix} -10 \\ -9 \\ -8 \\ -7 \\ -6 \end{pmatrix} B = \begin{pmatrix} -8 \\ -6 \\ -4 \\ -2 \end{pmatrix}$$

Assign a single element

To assign an element of a numerical vector, [*indice*] must be appended to the vector's name.
indice is an integer which must be between 1 and the size of the vector.

```
/* Assign real numbers to the vector vr */
vnumR vr;
resize(vr, 4);
vr[1] = 2;
A = 1;
vr[2*A] = 5;
vr[3] = 7;
```

$$vr = \begin{pmatrix} 2 \\ 5 \\ 7 \\ 0 \end{pmatrix}$$

The following example is incorrect because the size of the vector vr is 0. The command "vnumR vr;" initialize an empty vector.

```
/* INVALID CODE !! */
/* try to assign elements in vr */
vnumR vr; /* vr is empty */
vr[1] = 2;
```

If you want to assign an element after the current size of a vector, you have to increase the size of this vector with the command "resize" and you must use a temporary vector to keep the value of the vector.

```
/* Increase the size of the vector vr to assign
a real at indice 30 */
vr = 1,10;
vrtemp = vr;
resize(vr, 30);
vr[:size(vrtemp)] = vrtemp;
vr[30] = 4 ;
```

$$vr = \begin{pmatrix} 1 \\ \dots \\ 10 \\ 0 \\ \dots \\ 0 \\ 30 \end{pmatrix}$$

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
  The right vector's size must be the same as the left vector's one.
  The command "vr[indstart:indstop:indstep] = A;" is equivalent to the command "for j=indstart to indstop step indstep { vr[j] = A[j]; }"

```
/* Assign -t to the first 5 elements of vr */
/* Assign +t to the last 5 elements of vr */
t=1,5;
vnumR vr;
resize(vr,10);
vr[:5] =-t;
vr[6:] = t;

vnumC vc;
resize(vc,30);
vc[1:20:2] = 3*i*vr;
```

$$vr = \begin{pmatrix} -1 \\ -2 \\ \dots \\ -5 \\ 1 \\ \dots \\ 5 \end{pmatrix} vc = \begin{pmatrix} -3\imath \\ 0 \\ -6\imath \\ 0 \\ -9\imath \\ \dots \\ 0 \end{pmatrix}$$

The command "vr[vind] = A;" is equivalent to the command "for j=1 to size(vind) step 1 { vr[vind[j]] = A[j]; }"
vind is a numerical vector of integers. It must have the same size as the numerical vector A.

```
/* Assign -t to some elements of vr */
/* Assign +t to some elements of vr */
t=1,5;
vnumR vr;
resize(vr,9);
vind1=vnumR[1:4:8];
vind2=vnumR[2:3:5:9];
vr[vind1] =-t[1:3];
vr[vind2] = t[1:4];
```

$$vr = \begin{pmatrix} -1 \\ +1 \\ +2 \\ -2 \\ +3 \\ 0 \\ 0 \\ -3 \\ +4 \end{pmatrix}$$

Assign multiple elements (2/2)

- a number
  The command "vr[indstart:indstop:indstep] = r;" is equivalent to the command "for j=indstart to indstop step indstep { vr[j] = r; }"

```
/* Assign -1 to the first 5 elements of vr */
/* Assign +1 to the last 5 elements of vr */
vnumR vr;
resize(vr,10);
vr[:5] = -1;
vr[6:] = +1;

vnumC vc;
resize(vc,30);
vc[::2] = 1+4*i;
```

$$vr = \begin{pmatrix} -1 \\ -1 \\ \dots \\ 1 \\ \dots \\ 1 \end{pmatrix} \quad vc = \begin{pmatrix} 1+4\imath \\ 0 \\ 1+4\imath \\ 0 \\ 1+4\imath \\ \dots \\ 0 \end{pmatrix}$$

  The command "vr[vind] = r;" is equivalent to the command "for j=1 to size(vind) step 1 { vr[vind[j]] = r; }"
  vind is a numerical vector of integers.

```
/* Assign -1 to some elements of vr */
/* Assign +1 to some elements of vr */
vnumR vr;
resize(vr,9);
vind1=vnumR[1:4:8];
vind2=vnumR[2:3:5:9];
vr[vind1] = -1;
vr[vind2] = +1;
```

$$vr = \begin{pmatrix} -1 \\ +1 \\ +1 \\ -1 \\ +1 \\ 0 \\ 0 \\ -1 \\ +1 \end{pmatrix}$$

Ternary condition

Suppose you want to perform multiple assignments based on one condition.
That is, you want to do something like (cond1, cond2, src1, src2, vr are numerical vectors) :

```
for j = 1 to n
 {
  if ( cond1[j] <= cond2[j] ) then { vr[j]=src1[j]; } else { vr[j]=src2[j]; };
 };
```

it can be replaced by : vr = ?(cond1 ¡= cond2) : src1 : src2;
This statement will be much faster than the loop statement.

Now suppose, that you want to do :

```
for j = 1 to n
 {
  if ( cond1[j] <= cond2[j] ) then { vr[j]=1; } else { vr[j]=0; };
 };
```

it can be replaced by : vr = ?(cond1 ¡= cond2);

In the condition rules, one of the operands can be a real number: e.g., vr = ?(cond1 > 3) : src1 : src2;

```
t=-2,2;
v1 = ?(t < 0) : -1 : 1;
v2 = ?(sin(t) >= cos(t)) : v1 : 2*v1;
```

$$v1 = \begin{pmatrix} -1 \\ -1 \\ +1 \\ +1 \\ +1 \end{pmatrix} \quad v2 = \begin{pmatrix} -2 \\ -2 \\ -2 \\ +1 \\ +1 \end{pmatrix}$$

Select elements

You can select elements from a numerical vector :

- With a condition
  An element in a vector will be selected if the corresponding condition is true. The vector and the condition must have the same size.

```
vr = -5,5;
/* Return the elements of vr
which their magnitude <= 2 */
vp = select(abs(vr)<=2, vr);
/* Return indices */
t=1,size(vr);
vi = select(abs(vr)<=2, t);
```

$$vp = \begin{pmatrix} -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{pmatrix} vi = \begin{pmatrix} 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{pmatrix}$$

- With an indice vector
  A vector of real numbers (vi) could be used to select elements in a vector (vr) : the syntax is vr[*vi*].
  The valus of real numbers must be integers between 1 and the size of the vector.
  The previous example could be rewritten as :

```
vr = -5,5;
/* Return indices */
t=1,size(vr);
vi = select(abs(vr)<=2, t);
/* get corresponding values */
vp = vr[ vi ];
```

Another example :

```
t = vnumR[3:7:8:1];
vr =-5,5;
vp = vr[ t ];
```

$$vp = \begin{pmatrix} -3 \\ 1 \\ 2 \\ -5 \end{pmatrix}$$

Concatenate numerical vectors

To append one or more vectors or numbers to a numerical vector, you have to use the statement =vnumR[:] or = vnumC[:].

```
/* Build a vector vp with 2 vectors
                     and 2 numbers*/
vr = 1,4;
vi =-3,-2;
vp = vnumR[ vr : 7 : vi : 9];
vc = vnumC[vi: 1+i : -i ];

/* Append 3 times vi to A */
vnumR A;
for j=1 to 3 { A= vnumR[ A: vi]; };
```

$$vp = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 7 \\ -3 \\ -2 \\ 9 \end{pmatrix} vc = \begin{pmatrix} -3 \\ -2 \\ 1+\imath \\ -\imath \end{pmatrix} A = \begin{pmatrix} -3 \\ -2 \\ -3 \\ -2 \\ -3 \\ -2 \end{pmatrix}$$

Array of numerical vectors

- Declaration
  You could have arrays of numerical vectors. The number of dimension of the array is limited to 10. The indices could be any integers between -2^{31} to $2^{31}-1$.
  Array of numerical vectors require explicit declarations.

```
/* Build three array  :
 avr : array (1-dimentional) of 4 vectors of reals
 avb : array (2-dimentional) of 15 vectors of complexs
 avc : array (1-dimentional) of 5 vectors of complexs */
vnumR avr[0:3];
vnumC avb[3:5,1:2], avc[-2:2];
```

  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 vectors
  To access an vector 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 vector of the array, you just have to append two brackets :
  the first one for the indice of numerical vector in the array. The second for the indice of the number in the numerical vector.

```
/* build an array of 3 vectors */
vnumR vr[1:3];
/* each vector will contain 5 reals */
resize(vr,5);
/* assign values in the vector */
for j=1 to 5 { vr[1][j]=2*j$ vr[2][j]=3*j$ };
vr[3] = vr[2]+vr[1];
```

$$vr = \begin{pmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{pmatrix} \begin{pmatrix} 3 \\ 6 \\ 9 \\ 12 \\ 15 \end{pmatrix} \begin{pmatrix} 5 \\ 10 \\ 15 \\ 20 \\ 25 \end{pmatrix}$$

- Functions accepting array of numerical vectors : 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 vector are not affected by this change until they are assigned with a new value.

```
/* Build two vectors :
 vd : vector with double-precision
      real numbers
 vq : vector with quadruple-prec.
     real numbers */

_modenum= NUMDBL;
td = 0, pi, pi/20;
vd = sin(tq);

_modenum=NUMQUAD;
tq = 0,pi,pi/20;
vq= cos(tq);
```

```
/* Build two vectors  :
vmp100 : vector with 100
                   decimal digits
vmp30  : vector with 30
     decimal digits */

 _modenum=NUMFPMP;
_modenum_prec = 100;
tmp100 = 0,pi,pi/20;
vmp100= cos(tmp100);

_modenum=NUMFPMP;
_modenum_prec = 30;
vmp30= cos(tmp100);
```