The **simples-matrices** package[*]

Yvon Henel[†]

Released 2022/07/03

Résumé

Cette extension fournit des commandes pour définir et écrire des matrices en donnant leurs coefficients, par ligne, dans une vliste (liste de valeurs séparées par des virgules).

La documentation française pour l'utilisateur de l'extension simples-matrices est disponible sous le nom de `simples-matrices-fra`.

Abstract

This package provides macros to define and write matrices the coefficients of which are given, row by row, in a clist (list of values separated by commas).

The English documentation for the final user of the package simples-matrices is available in the file `simples-matrices-eng`.

---

1   Summary of Syntax of Document Commands

`\matrice`   `\matrice(⟨prefix⟩)<⟨clist of key-value pairs⟩>[⟨type⟩]{⟨clist of coefficients⟩}`

`\declarermatrice`   `\declarermatrice*(⟨prefix⟩)<⟨clist of key-value pairs⟩>{⟨matrix name⟩}[⟨type⟩]{⟨clist of coefficients⟩}`

`\lamatrice`   `\lamatrice(⟨prefix⟩)<⟨clist of key-value pairs⟩>{⟨matrix name⟩}`

`\MatriceInterieur`   `\MatriceInterieur`

`\LaMatriceInterieur`   `\LaMatriceInterieur{⟨matrix name⟩}`

---

[*]This file describes v1.0.1, last revised 2022/07/03.
[†]E-mail: le.texnicien.de.surface@yvon-henel.fr

\simplesmatricessetup	\simplesmatricessetup{⟨*clist of key-value pairs*⟩}

\matid	\matid[⟨*diagonal coefficient, default 1*⟩]{⟨*size of the square matrix*⟩}

\matnulle	\matid[⟨*diagonal coefficient, default 0*⟩]{⟨*size of the square matrix*⟩}

2   simples-matrices implementation

2.1   Loading And Greeting

```
1 ⟨@@=SMPLMTRC⟩

2 \RequirePackage{l3keys2e,xparse}
3 \ProvidesExplPackage
4 {simples-matrices} {2022/06/08} {1.0} {defining and printing matrices}
5 \RequirePackage{amsmath}
```

amsmath is the only auxiliary package loaded. We need it to write the matrices.

2.2   Keys And Options

I'm using the key management mechanism of LaTeX3.

```
6 \keys_define:nn {simples-matrices}
7 {
8   envir .tl_set:N = \g__SMPLMTRC_nom_environnement_tl,
9   envir .initial:n = {matrix},
10  prefix .tl_set:N = \g__SMPLMTRC_prefix_environnement_tl,
11  prefix .initial:n = {p},
12  argopt .tl_set:N = \g__SMPLMTRC_argument_environnement_tl,
13  argopt .initial:n = { },
14  typeord .str_set:N = \g__SMPLMTRC_type_ordinaire_str,
15  typeord .initial:n = { C },
16 }
17 \ProcessKeysOptions {simples-matrices}
```

The keys above defined the package options. The next one may be usefull to return to a kind of neutral state.

```
18 \keys_define:nn {simples-matrices}
19 {
20   out-of-box .meta:n = {prefix=p, envir=matrix, argopt=, typeord=C},
21 }
```

2.3   Constants And Variables

To handle the syntax of xcas matrices, I resort to *regex* so I have to create some constant *regexes*. Some comments are just there to prevent emacs to go too far in formating the code.

```
22 \regex_const:Nn \c__SMPLMTRC_xcas_debut_regexp {\[\[} %\]\]
23 \regex_const:Nn \c__SMPLMTRC_xcas_fin_regexp {\]\]}
24 \regex_const:Nn \c__SMPLMTRC_xcas_milieu_regexp{\],\ *\[} %\]
```

Now come error messages: the first deals with bad numbers of coefficient for matrices of type I, J, S or T; the second appears when dealing with an x type matrix.

```
25 \msg_new:nnnn{simples-matrices}{errorIJST}
26 {The~number~of~coefficients~does~not~fit~the~matrix-type}
27 {I~was~expecting~#1~coefficients~and~obtain~only~#2.}
28
29 \msg_new:nnnn{simples-matrices}{errorx}
30 {The~data~do~not~seem~to~match~the~x-type}
31 {I~was~expecting~the~data~to~begin~with~"[["}
```

To contain the inner part of the matrix, we need a token list global variable. I chose to built it with a non-letter to try to prevent name clash.

```
32 \tl_new:c {g__SMPLMTRC_contenu_anonyme*_tl}
```

__SMPLMTRC_creer_matrice_aux: When this macro is called, \g_tmpa_int contains the number of columns of the matrix and \g_tmpa_seq the *sequence* of coefficients.

```
33 \cs_new:Nn \__SMPLMTRC_creer_matrice_aux:
34 {
35   \tl_gclear:c {g__SMPLMTRC_contenu_anonyme*_tl}
36
37   \seq_map_indexed_inline:Nn \g_tmpa_seq
38   {
39     \tl_gput_right:cn {g__SMPLMTRC_contenu_anonyme*_tl} { ##2  }
40     \int_compare:nNnTF { \int_mod:nn {##1} { \g_tmpa_int } } = { 0 }
41     {
42       \tl_gput_right:cn {g__SMPLMTRC_contenu_anonyme*_tl} { \\ }
43     }
44     {
45       \tl_gput_right:cn {g__SMPLMTRC_contenu_anonyme*_tl} { &  }
46     }
47   }
48 }
```

At the end of this macro, \g__SMPLMTRC_contenu_anonyme*_tl contains the inner part of the matrix.

(*End definition for* __SMPLMTRC_creer_matrice_aux:.)

For I need to match a text against a regex constant, I create a variant of one of the function of LaTeX3:

```
49 \cs_generate_variant:Nn \regex_match:nnF { nVF }
```

The next macro does the bulk of the work in reading the coefficients in the second argument and managing them according to the data-type which is given in the first argument.

__SMPLMTRC_creer_matrice:nn   To be on the safe side, I begin with clearing the variable.

```
50 \cs_new:Nn \__SMPLMTRC_creer_matrice:nn
51 {
52   \tl_gclear:c {g__SMPLMTRC_contenu_anonyme*_tl}
```

I then read the first argument inside \l_tmpa_str with \str_set:Nx in case the string is contained in a variable.

```
53   \str_set:Nx \l_tmpa_str { #1 }
```

If `#1` is equal to `O`, I fetch the data-type in the string valued through the key `typeord`.

```
54    \str_if_eq:VnT \l_tmpa_str { O }
55    {
56       \str_set:NV \l_tmpa_str \g__SMPLMTRC_type_ordinaire_str
57    }
```

We enter now the first `case` which depends on the data-type. The first case deals with the `x`-type. Here we obtain the inner part of the matrix by replacing, according to *regex*. There is no need to calculate anything.

If ever there are other types of the same kind —e. g. to deal with `maxima` syntax or the like— those types would be dealt with here.

```
58    \str_case:VnF \l_tmpa_str
59    {
60       { x } {
61          \tl_set:Nn \l_tmpa_tl { #2 }
62
63          \regex_match:nVF {\A\s*\[\[} %\]\]
64          \l_tmpa_tl
65          {
66             \msg_fatal:nn{simples-matrices} {errorx}
67          }
68
69          \regex_replace_all:NnN \c__SMPLMTRC_xcas_debut_regexp { } \l_tmpa_tl
70          \regex_replace_all:NnN \c__SMPLMTRC_xcas_fin_regexp { \c{\\} } \l_tmpa_tl
71          \regex_replace_all:NnN \c__SMPLMTRC_xcas_milieu_regexp { \c{\\} } \l_tmpa_tl
72          \regex_replace_all:nnN { , }{ & } \l_tmpa_tl
73
74          \tl_set_eq:cN {g__SMPLMTRC_contenu_anonyme*_tl} \l_tmpa_tl
75       }
76    }
```

The `x` -type is done. Now for the `false` part of the `case` where I will take care of the other kind of data-types, those which require calculations.

I prepare the work by reading the second argument inside the *sequence* `\g_tmpa_seq` and cleaning `\g_tmpb_seq`.

```
77    {
78       \seq_set_from_clist:Nn \g_tmpa_seq { #2 }
79       \seq_clear:N \l_tmpb_seq
```

Now come the second `case`:

```
80       \str_case:VnF \l_tmpa_str
81       {
```

The first case is that of the square matrices. I calculate the number of columns as the square root of the number of coefficients. There is, as for now, no check for correctness because in case of bad number of coefficients there will just be a strange matrix, with void cells.

```
82       { C } {
83          \int_set:Nn \g_tmpb_int { \seq_count:N \g_tmpa_seq }
84          \int_set:Nn \g_tmpa_int { \fp_to_int:n { sqrt(\g_tmpb_int) } }
85       }
```

In case of a diagonal matrix, the number of columns is simply the number of coefficients but we have to create the suitable *sequence*. The trick is just to add as many zeros as there are columns between each given coefficient.

```
86        { D } {
87          \int_set:Nn \g_tmpa_int { \seq_count:N \g_tmpa_seq }
88          \seq_pop_left:NN \g_tmpa_seq \l_tmpa_tl
89          \seq_put_right:NV \l_tmpb_seq \l_tmpa_tl
90
91          \bool_until_do:nn { \seq_if_empty_p:N \g_tmpa_seq }
92          {
93            \int_step_inline:nn { \g_tmpa_int }
94            {
95              \seq_put_right:Nn \l_tmpb_seq { 0 }
96            }
97            \seq_pop:NN \g_tmpa_seq \l_tmpa_tl
98            \seq_put_right:NV \l_tmpb_seq \l_tmpa_tl
99          }
100         \seq_set_eq:NN \g_tmpa_seq \l_tmpb_seq
101       }
```

The code for the next four cases is very similar. We have to calculate the number of columns from the number of coefficient and then complete the *sequence* of coefficients.

Here I check the correctness because, otherwise, in case of error LATEX doesn't behave kindly.

The number of coefficients *non-necessarily* null in a triangular matrix with n columns is $n(n+1)/2$ or $n(n-1)/2$ if the diagonal is *null*. Therefore the computing of \g_tmpa_-int from \g_tmpb_int which is equal to the number of coefficients, that is the number of items in the *sequence* \g_tmpa_seq.

In the first case (upper triangular) there are coefficients on the diagonal so we have to append $L-1$ zeros before $N+1-L$ coefficients where L is the line number and N the number of columns.

```
102       { S } {
103         \int_set:Nn \g_tmpb_int { \seq_count:N \g_tmpa_seq }
104         \int_set:Nn \g_tmpa_int
105         {
106           \fp_to_int:n { ( sqrt(1 + 8*\g_tmpb_int) - 1 ) / 2 }
107         }
```

We check for correctness and abort in case of discrepancy.

```
108         \int_compare:nNnF
109         {2*\g_tmpb_int} = { \g_tmpa_int * (\g_tmpa_int + 1)}
110         {
111           \msg_fatal:nnxx{simples-matrices}
112           {errorIJST}
113           {\int_to_arabic:n{(\g_tmpa_int * (\g_tmpa_int + 1))/2}}
114           {\int_to_arabic:n{\g_tmpb_int}}
115         }
```

##1 is the line number of the matrix

```
116
117         \int_step_inline:nn { \g_tmpa_int }
118         {
119           \int_step_inline:nn { ##1 - 1 }
120           {
```

```
121        \seq_put_right:Nn \l_tmpb_seq {0}
122      }
123      \int_step_inline:nn { \g_tmpa_int - ##1 + 1 }
124      {
125        \seq_pop:NN \g_tmpa_seq \l_tmpa_tl
126        \seq_put_right:NV \l_tmpb_seq \l_tmpa_tl
127      }
128    }
129
130    \seq_set_eq:NN \g_tmpa_seq \l_tmpb_seq
131
132  }
```

When there are zeros on the diagonal, the formula changes a tiny bit: append L zeros before $N - L$ coefficients.

```
133    { T } {
134      \int_set:Nn \g_tmpb_int { \seq_count:N \g_tmpa_seq }
135      \int_set:Nn \g_tmpa_int
136      {
137        \fp_to_int:n { (1 + sqrt(1 + 8*\g_tmpb_int) ) / 2 }
138      }
139      \int_compare:nNnF
140      {2*\g_tmpb_int} = { \g_tmpa_int * (\g_tmpa_int - 1)}
141      {
142        \msg_fatal:nnxx{simples-matrices}
143        {errorIJST}
144        {\int_to_arabic:n{(\g_tmpa_int * (\g_tmpa_int - 1))/2}}
145        {\int_to_arabic:n{\g_tmpb_int}}
146      }
147      \int_step_inline:nn { \g_tmpa_int }
148      {
149        \int_step_inline:nn { ##1 }
150        {
151          \seq_put_right:Nn \l_tmpb_seq {0}
152        }
153        \int_step_inline:nn { \g_tmpa_int - ##1 }
154        {
155          \seq_pop:NN \g_tmpa_seq \l_tmpa_tl
156          \seq_put_right:NV \l_tmpb_seq \l_tmpa_tl
157        }
158      }
159      \seq_set_eq:NN \g_tmpa_seq \l_tmpb_seq
160    }
```

Now come the cases for lower triangular matrices. First *with diagonal*.

```
161    { I } {
162      \int_set:Nn \g_tmpb_int { \seq_count:N \g_tmpa_seq }
163      \int_set:Nn \g_tmpa_int
164      {
165        \fp_to_int:n { ( sqrt(1 + 8*\g_tmpb_int) - 1 ) / 2 }
166      }
167      \int_compare:nNnF
168      {2*\g_tmpb_int} = { \g_tmpa_int * (\g_tmpa_int + 1)}
169      {
170        \msg_fatal:nnxx{simples-matrices}
```

```
171        {errorIJST}
172        {\int_to_arabic:n{(\g_tmpa_int * (\g_tmpa_int + 1))/2}}
173        {\int_to_arabic:n{\g_tmpb_int}}
174      }
175      \int_step_inline:nn { \g_tmpa_int }
176      {
177        \int_step_inline:nn { ##1 }
178        {
179          \seq_pop:NN \g_tmpa_seq \l_tmpa_tl
180          \seq_put_right:NV \l_tmpb_seq \l_tmpa_tl
181        }
182        \int_step_inline:nn { \g_tmpa_int - ##1 }
183        {
184          \seq_put_right:Nn \l_tmpb_seq {0}
185        }
186      }
187      \seq_set_eq:NN \g_tmpa_seq \l_tmpb_seq
188    }
```

Secondly *without diagonal.*

```
189      { J } {
190        \int_set:Nn \g_tmpb_int { \seq_count:N \g_tmpa_seq }
191        \int_set:Nn \g_tmpa_int
192        {
193          \fp_to_int:n { (1 + sqrt(1 + 8*\g_tmpb_int) ) / 2 }
194        }
195        \int_compare:nNnF
196        {2*\g_tmpb_int} = { \g_tmpa_int * (\g_tmpa_int - 1)}
197        {
198          \msg_fatal:nnxx{simples-matrices}
199          {errorIJST}
200          {\int_to_arabic:n{(\g_tmpa_int * (\g_tmpa_int - 1))/2}}
201          {\int_to_arabic:n{\g_tmpb_int}}
202        }
203        \int_step_inline:nn { \g_tmpa_int }
204        {
205          \int_step_inline:nn { ##1 -1}
206          {
207            \seq_pop:NN \g_tmpa_seq \l_tmpa_tl
208            \seq_put_right:NV \l_tmpb_seq \l_tmpa_tl
209          }
210          \int_step_inline:nn { \g_tmpa_int - ##1 + 1}
211          {
212            \seq_put_right:Nn \l_tmpb_seq {0}
213          }
214        }
215        \seq_set_eq:NN \g_tmpa_seq \l_tmpb_seq
216      }
217    }
```

Here we attain the `false` part of the second `case`. We assume that we have a number in `#1`.

```
218      {
219        \int_set:Nn \g_tmpa_int { \l_tmpa_str }
220      }
```

If all is well, here it ends well with the number of columns in `\g_tmpa_int` and all the coefficients of the matrix in `\g_tmpa_seq`. So it's time to call

```
221     \__SMPLMTRC_creer_matrice_aux:
222   }
223 }
```

(*End definition for* `\__SMPLMTRC_creer_matrice:nn.`)

`\MatriceInterieur` A document command to access the inner part of the **last** printed —via `\matrice`— or defined —via `\declarermatrice`— matrix.

This macro can be placed inside an `array`-like environment.

```
224 \cs_new:Npn \MatriceInterieur
225 {
226   \tl_use:c { g__SMPLMTRC_contenu_anonyme*_tl }
227 }
```

(*End definition for* `\MatriceInterieur`. *This function is documented on page* *1.*)

`\LaMatriceInterieur` A version to access the inner part of a named matrix:

```
228 \cs_new:Npn \LaMatriceInterieur #1
229 {
230   \tl_use:c { g__SMPLMTRC_contenu_nom_#1_tl }
231 }
```

(*End definition for* `\LaMatriceInterieur`. *This function is documented on page* *1.*)

The next macro is used to print a matrix. Its name means *prepare (the) presentation* of the matrix. It builds the name of the environment, taking care of possible arguments of the said environment.

`\__SMPLMTRC_preparer_presentation:nn` At the end of the macro, `\g_tmpa_tl` contains the *overture* of the environment and `\g_tmpb_tl` the *grand final*.

```
232 \cs_new:Nn \__SMPLMTRC_preparer_presentation:nn
233 {
234   \IfValueT{#1}
235   {
236     \keys_set:nn {simples-matrices} {prefix = #1}
237   }
238   \IfValueT{#2}
239   {
240     \keys_set:nn {simples-matrices} { #2 }
241   }
242   \tl_set:NV \l_tmpa_tl \g__SMPLMTRC_prefix_environnement_tl
243   \tl_put_right:NV \l_tmpa_tl \g__SMPLMTRC_nom_environnement_tl
244   \tl_set:Nn \g_tmpa_tl { \begin{\l_tmpa_tl} }
245   \tl_put_right:NV \g_tmpa_tl \g__SMPLMTRC_argument_environnement_tl
246   \tl_set:Nn \g_tmpb_tl {\end{\l_tmpa_tl} }
247 }
```

(*End definition for* `\__SMPLMTRC_preparer_presentation:nn.`)

Now comes the first main document command which prints the matrix from the data-type and coefficients. It assigns the different arguments to the auxiliary macros and finishes with using the token lists created by the auxiliaries.

\matrice

```
248 \NewDocumentCommand{\matrice}{ d() d<> O{O} m }
249 {
250   \__SMPLMTRC_creer_matrice:nn { #3 } { #4 }
251   \__SMPLMTRC_preparer_presentation:nn { #1 } { #2 }
252   \tl_use:N \g_tmpa_tl
253   \tl_use:c { g__SMPLMTRC_contenu_anonyme*_tl }
254   \tl_use:N \g_tmpb_tl
255 }
```

(*End definition for* \matrice. *This function is documented on page 1.*)

I provide a way of changing the keys inside the document with the usual *setup* macro:

\simplesmatricessetup

```
256 \NewDocumentCommand{\simplesmatricessetup}{m}
257 {
258   \keys_set:nn {simples-matrices} { #1 }
259 }
```

(*End definition for* \simplesmatricessetup. *This function is documented on page 2.*)

Another main document command to define a named matrix. The starred version defines and prints.

The first mandatory argument —the 4th of all— should contain the "name" of the matrix with which one can recall the matrix using \lamatrice.

\declarermatrice

```
260 \NewDocumentCommand{\declarermatrice}{ s d() d<> m  O{O} m }
261 {
262   \__SMPLMTRC_creer_matrice:nn { #5 } { #6 }
263   \tl_gset_eq:cc {g__SMPLMTRC_contenu_nom_#4_tl} {g__SMPLMTRC_contenu_anonyme*_tl}
264   \IfBooleanT { #1 }
265   {
266     \__SMPLMTRC_preparer_presentation:nn { #2 } { #3 }
267     \tl_use:N \g_tmpa_tl
268     \tl_use:c { g__SMPLMTRC_contenu_anonyme*_tl }
269     \tl_use:N \g_tmpb_tl
270   }
271 }
```

(*End definition for* \declarermatrice. *This function is documented on page 1.*)

To print a named matrix:

\lamatrice

```
272 \NewDocumentCommand{\lamatrice}{ d() d<> m }
273 {
274   \__SMPLMTRC_preparer_presentation:nn { #1 } { #2 }
275   \tl_use:N \g_tmpa_tl
276   \tl_use:c { g__SMPLMTRC_contenu_nom_#3_tl }
277   \tl_use:N \g_tmpb_tl
278 }
```

(*End definition for* \lamatrice. *This function is documented on page 1.*)

For the next commands, I need a variant:

```
279 \cs_generate_variant:Nn \__SMPLMTRC_creer_matrice:nn { nV }
```

The auxiliary macro to print an identity matrix. `#1` will receive the value to print in the diagonal —defaulting to 1— and `#2` the number of columns.

```
280 \cs_new:Nn \__SMPLMTRC_creer_matrice_identite:nn
281 {
282   \clist_clear:N \l_tmpa_clist
283   \int_step_inline:nn { #2 }
284   {
285     \clist_put_left:Nn \l_tmpa_clist { #1 }
286   }
287   \__SMPLMTRC_creer_matrice:nV {D} \l_tmpa_clist
288 }
```

(*End definition for* __SMPLMTRC_creer_matrice_identite:nn.)

I can then use it to define the document command:

\matid

```
289 \NewDocumentCommand{\matid}{ d() d<> O{1} m}
290 {
291   \__SMPLMTRC_creer_matrice_identite:nn { #3 } { #4 }
292   \__SMPLMTRC_preparer_presentation:nn  { #1 } { #2 }
293   \tl_use:N \g_tmpa_tl
294   \tl_use:c { g__SMPLMTRC_contenu_anonyme*_tl }
295   \tl_use:N \g_tmpb_tl
296 }
```

(*End definition for* \matid. *This function is documented on page 2.*)

I follow the same pattern for null matrices. First the auxiliary macro, `#2` is the number of columns, `#1` the value of all the coefficients —defaulting to 0—.

```
297 \cs_new:Nn \__SMPLMTRC_creer_matrice_nulle:nn
298 {
299   \clist_clear:N \l_tmpa_clist
300   \int_step_inline:nn { #2*#2 }
301   {
302     \clist_put_left:Nn \l_tmpa_clist { #1 }
303   }
304   \__SMPLMTRC_creer_matrice:nV {C} \l_tmpa_clist
305 }
```

(*End definition for* __SMPLMTRC_creer_matrice_nulle:nn.)

I can again use it to define the document command:

\matnulle

```
306 \NewDocumentCommand{\matnulle}{ d() d<> O{0} m}
307 {
308   \__SMPLMTRC_creer_matrice_nulle:nn {#3} {#4}
309   \__SMPLMTRC_preparer_presentation:nn { #1 } { #2 }
310   \tl_use:N \g_tmpa_tl
311   \tl_use:c { g__SMPLMTRC_contenu_anonyme*_tl }
312   \tl_use:N \g_tmpb_tl
313 }
```

(*End definition for* \matnulle. *This function is documented on page 2.*)

Change History

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.