# MathMatrixPack package documentation

**Wolfram Mathematica® 10.0 package for matrices with labeled entries**

Renato Maia Matarazzo Orsino

November 13, 2015

# 1 Formatting rules

This section presents the following formatting rules in the kernel of Mathematica when Package MathMatrixPack is used.

## 1.1 Trigonometric functions

```
1   $PrePrint = # /. {
2      Csc[xArgument_] :> 1/Defer @ Sin[xArgument],
3      Sec[xArgument_] :> 1/Defer @ Cos[xArgument],
4      Tan[xArgument_] :> Defer @ Sin[xArgument]/Defer @ Cos[xArgument],
5      Cot[xArgument_] :> Defer @ Cos[xArgument]/Defer @ Sin[xArgument]
6      } &;
7
8   Unprotect[Cos, Sin];
9   Format[Cos[xArgument_]] := Subscript[c, xArgument]
10  Format[Sin[xArgument_]] := Subscript[s, xArgument]
11  Protect[Cos, Sin];
```

This piece of code modifies the default display notation for trigonometric functions in Mathematica: $\sin(*)$, $\cos(*)$, $\tan(*)$, $\cot(*)$, $\sec(*)$, $\csc(*)$ are denoted respectively as $s_*$, $c_*$, $s_*/c_*$, $c_*/s_*$, $1/c_*$ and $1/s_*$ for any (assigned or unassigned) variable used in the code.

## 1.2 Derivatives

```
1 Format[Subscript[ Subscript[xArgument_, xIndexes1__], xIndexes2__]'[t_]]
     :=
2    Subscript[ Subscript[ Overscript[xArgument, "."], xIndexes1],
        xIndexes2][t]
3 Format[Subscript[ Subscript[xArgument_, xIndexes1__], xIndexes2__]''[t_
    ]] :=
4  Subscript[ Subscript[ Overscript[xArgument, ".."], xIndexes1],
      xIndexes2][t]
5 Format[Subscript[xArgument_, xIndexes1__]'[t_]] :=
6  Subscript[ Overscript[xArgument, "."], xIndexes1][t]
```

```
 7 Format[Subscript[xArgument_, xIndexes1__]''[t_]] :=
 8   Subscript[ Overscript[xArgument, ".."], xIndexes1][t]
 9 Format[xArgument_'[t_]] :=
10   Overscript[xArgument, "."][t]
11 Format[xArgument_''[t_]] :=
12   Overscript[xArgument, ".."][t]
13
14 SymbolReplacements = {
15   Subscript[ Subscript[xBase_, xIndexes__], xIndexes2__]'[t] ->
16     Subscript[ Subscript[ Overscript[xBase, "."], xIndexes], xIndexes2],
17   Subscript[ Subscript[xBase_, xIndexes__], xIndexes2__]''[t] ->
18     Subscript[Subscript[ Overscript[xBase, ".."], xIndexes], xIndexes2],
19   Subscript[xBase_, xIndexes__]'[t] ->
20     Subscript[ Overscript[xBase, "."], xIndexes],
21   Subscript[xBase_, xIndexes__]''[t] ->
22      Subscript[ Overscript[xBase, ".."], xIndexes],
23   xVariable_'[t] -> Overscript[xVariable, "."],
24   xVariable_''[t] -> Overscript[xVariable, ".."],
25   xVariable_[t] -> xVariable
26   };
```

This piece of code modifies the display notation for first and second order time derivatives: $\zeta'[t]$ and $\zeta''[t]$ are denoted respectively as $\dot{\zeta}[t]$ and $\ddot{\zeta}[t]$.

SymbolReplacements is a list of rules for formatting first and second order time derivatives. Whenever this list of rules is used, $\zeta'[t]$ and $\zeta''[t]$ will be replaced respectively by $\dot{\zeta}$ and $\ddot{\zeta}$.

## 1.3 Round-off rules

```
1 RoundOffRules = {
2   xNumber_?NumericQ /; Abs[xNumber] < 10^-12 -> 0,
3   xNumber_?NumericQ /; Abs[xNumber - 1] < 10^-12 -> 1,
4   xNumber_?NumericQ /; Abs[xNumber + 1] < 10^-12 -> -1
5   };
```

RoundOffRules is a list of rules for formatting numbers. Whenever this list of rules is used, numbers in the ranges $]-10^{-12}, +10^{-12}[$ and $]1-10^{-12}, 1+10^{-12}[$ will be displayed as $0$ and $1$, respectively.

## 1.4 Displaying matrices

```
1  SMatrixForm[xA_Association, xL_: Automatic] :=
2   If[ KeyExistsQ[xA, "Column␣Labels"],
3     MatrixForm[
4       xA["Matrix"],
5       TableHeadings -> ({xA["Row␣Labels"], xA["Column␣Labels"]} /.
         SymbolReplacements),
6       TableAlignments -> xL
7       ],
8     MatrixForm[
9       xA["Matrix"],
10      TableHeadings -> ({xA["Row␣Labels"], None} /. SymbolReplacements),
11      TableAlignments -> xL
12      ]
13    ]
```

```
1  STableForm[xA_Association, xL_: Left] :=
2   If[ KeyExistsQ[xA, "Column␣Labels"],
3     TableForm[
4       xA["Matrix"],
5       TableHeadings -> ({xA["Row␣Labels"], xA["Column␣Labels"]} /.
         SymbolReplacements),
6       TableAlignments -> xL
7       ],
8     TableForm[
9       xA["Matrix"],
10      TableHeadings -> ({xA["Row␣Labels"], None} /. SymbolReplacements),
11      TableAlignments -> xL
12      ]
13    ]
```

`SMatrixForm` and `STableForm` extend the application of the built-in functions `MatrixForm` and `TableForm` to matrices given by `Association` elements.

# 2 General purpose functions

This section presents some general purpose functions that can be used for other applications than the modelling of multibody systems.

## 2.1 Set complement

```
1 SetComplement[xMainSet_, xDiffSet_] :=
2   Select[xMainSet, Not[MemberQ[xDiffSet, #]] &]
```

`SetComplement` returns the elements of the list `xMainSet` that are not in `xDiffSet` in the same order of occurrence in `xMainSet` (unlike the built-in function `Complement` that does the same opperation but sorts the output list).

## 2.2 Delete redundant expressions

```
1 RedundantElim[xX_] := DeleteDuplicates @ (DeleteCases[Simplify @ xX, 0])
    ;
```

`RedundantElim` deletes all repeated elements and all exact zeros (with head `Integer`) of a list.

## 2.3 Simplify Associations

```
1 SSimplify[xA_Association] := Association @
2   MapThread[#1 -> #2 &, {First /@ (Normal @ xA), Simplify@(Last /@ (
      Normal @ xA))}, 1]
3
4 SSimplify[xX_] := Simplify[xX]
```

`SSimplify` is an extension of the built-in function `Simplify` applicable to `Association` elements.

## 2.4 Replacements in Associations

```
 1 SReplaceRepeated[xA_Association, xL_List] :=
 2   Association @ MapThread[ #1 -> #2 &, {
 3     First /@ (Normal@xA),
 4     ReplaceRepeated[(Last /@ (Normal @ xA)), xL]
 5     }, 1]
 6
 7 SReplaceRepeated[xX_, xL_List] :=
 8   ReplaceRepeated[xX, xL]
 9
10 SReplaceFullSimplify[xA_Association, xRules_List] :=
11   Association @ MapThread[ #1 -> #2 &, {
12       First /@ (Normal @ xA),
13       FullSimplify[ FullSimplify[ Expand[(Last /@ (Normal @ xA)) //.
14           xRules] //. xRules] //. xRules]
14       }, 1]
15
16 SReplaceFullSimplify[xX_, xRules_List] :=
17   FullSimplify[ FullSimplify[ Expand[(Flatten @ {xX}) //. xRules] //.
18       xRules] //. xRules]
18
19 SReplaceSimplify[xA_Association, xRules_List] :=
20   Association @ MapThread[ #1 -> #2 &, {
21       First /@ (Normal @ xA),
22       Simplify[ Simplify[ Expand[(Last /@ (Normal @ xA)) //. xRules] //.
23           xRules] //. xRules]
23       }, 1]
24
25 SReplaceSimplify[xX_, xRules_List] :=
26   Simplify[ Simplify[ Expand[(Flatten @ {xX}) //. xRules] //. xRules]
26       //. xRules]
```

SReplaceRepeated is an extension of the built-in function ReplaceRepeated applicable to Association elements.

`SReplaceFullSimplify` and `SReplaceSimplify` are functions that simultaneously perform replacements and simplify the resulting expressions. They apply the built-in functions `ReplaceRepeated`, `Expand` and `FullSimplify` or `Simplify` to the corresponding expressions (normally `List` or `Association` elements).

## 2.5 Rename keys and values in Associations

```
1 SRename[xIn_Association, xNamingRules_, xExtraRules_: {}] :=
2   Association @ MapThread[ #1 -> #2 &, {
3     If[Head[#] === String,
4       StringReplace[#, xNamingRules], #] & /@ (First /@ Normal @ (xIn)),
5     Map[SReplaceRepeated[#, xExtraRules] &, Map[SReplaceRepeated[#,
         xNamingRules] &,
6       Map[SReplaceRepeated[#, xExtraRules] &, (Last /@ Normal @ xIn), All
         ], All], All]
7     }, 1]
```

`SRename[xIn, xNamingRules]` replaces, according to `xNamingRules`, string ocurrences both in the keys and values of the `Association` element `xIn`.

`SRename[xIn, xNamingRules, xExtraRules]` also applies replacements according to `xExtraRules` to the values of the `Association` element `xIn`.

## 2.6 List variables in expressions

```
1 GetVariables[xX_List, xExcept_List: {}] :=
2   Complement[ DeleteDuplicates @ Cases[xX, xVariable_[t], Infinity],
     xExcept]
3
4 GetVariables[xX_Association, xExcept_List: {}] :=
5   Complement[ DeleteDuplicates @ Cases[xX["Matrix"], xVariable_[t],
     Infinity], xExcept]
```

`GetVariables` returns a list of all time dependent variables in a given symbolic expression `xX` (which can be either a `List` or an `Association`). With the optional argument `xExcept_List` the user can list the variables that must not be listed in the output.

```
1 HeadList = {
```

```
 2    Or, And,
 3    Equal, Unequal, Inequality
 4    Less, LessEqual,
 5    Greater, GreaterEqual
 6    };
 7
 8  GetAllVariables[xNumber_?NumericQ] :=
 9    Sequence[]
10
11  GetAllVariables[{}] :=
12    Sequence[]
13
14  GetAllVariables[xRelationalOperator_] /; MemberQ[HeadList,
        xRelationalOperator] :=
15    Sequence[]
16
17  GetAllVariables[x_List] :=
18    DeleteDuplicates @ (Flatten @ (Union @ (GetAllVariables[#] & /@ x)))
19
20  GetAllVariables[Derivative[xNumber_Integer][xFunction_][xArgument__]] :=
21    Module[{xVariable},
22        If[MemberQ[Attributes[xFunction], NumericFunction] || MemberQ[
            HeadList, xFunction],
23      (*-TRUE-*)
24      xVariable = GetAllVariables[{xArgument}],
25      (*-FALSE-*)
26        xVariable = Derivative[xNumber][xFunction][xArgument]
27        ];
28      xVariable
29    ]
30
31  GetAllVariables[xFunction_Symbol[xArgument__]] :=
32    Module[{xVariable},
33      If[MemberQ[Attributes[xFunction], NumericFunction] || MemberQ[
          HeadList, xFunction],
```

```
34      (*-TRUE-*)
35      xVariable = GetAllVariables[{xArgument}],
36      (*-FALSE-*)
37      xVariable = xFunction[xArgument]
38      ];
39   xVariable
40   ];
41
42 GetAllVariables[xOther_] :=
43   xOther
```

`GetAllVariables` returns a list of all symbolic variables (both time dependent variables and non-numeric parameters) in a given symbolic expression.

# 3 Matrix calculus

In package MathMatrixPack, matrices must have row and column labels in order to perform correctly the operations of matrix sum/assemble and matrix multiplication. Thus, in this package a matrix is represented by an `Association` element with 3 keys:

- `"Matrix"`: a two dimensional array (`List` element) representing the matrix itself.

- `"Row␣Labels"`: an ordered `List` providing the indexes of the respective rows of the declared matrix.

- `"Column␣Labels"`: an ordered `List` providing the indexes of the respective columns of the declared matrix.

## 3.1 Sum, assemble and partitioning of matrices - `AngleBracket` operator

Wolfram Mathematica has some operators without built-in meanings. In MathMatrix-Pack, the operator `AngleBracket`, displayed as ⟨X,Y,...⟩, is used to perform the operations of sum, assemble and partitioning of matrices. The definitions for this operator are shown in the following piece of code:

```
1 Matrix2Rule[xA_Association] /; (ArrayDepth[xA["Matrix"]] > 1) :=
2   Association @ Flatten @ MapThread[ (#1 -> #2) &, {
```

```
 3     Outer[{#1, #2} &, xA["Row␣Labels"], xA["Column␣Labels"]],
 4     xA["Matrix"]
 5     }, 2]
 6
 7 Matrix2Rule[xA_Association] :=
 8   Association @ Flatten @ MapThread[ (#1 -> #2) &, {
 9     xA["Row␣Labels"],
10     xA["Matrix"]
11     }, 1]
```

```
 1 AngleBracket[xA__Association] :=
 2   Module[{xAList, xRowLabels, xColumnLabels, xRList},
 3     xAList = List[xA];
 4     xRowLabels = Union @ (Join @@ ((#["Row␣Labels"]) & /@ xAList));
 5     If[ And @@ (KeyExistsQ[#, "Column␣Labels"]& /@ xAList),
 6       xColumnLabels = Union @ (Join @@ ((#["Column␣Labels"]) & /@ xAList)
         );
 7       xRList = Association @ ((# -> Plus @@ DeleteCases[# /. (Matrix2Rule
          /@ xAList), #]) & /@
 8         Flatten[Outer[{#1, #2} &, xRowLabels, xColumnLabels], 1]);
 9       Association[
10         "Matrix" -> Outer[{#1, #2} &, xRowLabels, xColumnLabels] /.
            xRList,
11         "Row␣Labels" -> xRowLabels,
12         "Column␣Labels" -> xColumnLabels
13         ],
14       xRList = Association @ ((# -> Plus @@ DeleteCases[# /. (Matrix2Rule
          /@ xAList), #]) & /@
15         xRowLabels);
16       Association[
17         "Matrix" -> xRowLabels /. xRList,
18         "Row␣Labels" -> xRowLabels
19         ]
20       ]
21     ]
```

```
22
23 AngleBracket[xMatrix_List, xLabels_List] /; (ArrayDepth[xMatrix] === 1)
      :=
24    AngleBracket @ Association[
25      "Matrix" -> xMatrix,
26      "Row␣Labels" -> xLabels
27      ]
28
29 AngleBracket[xMatrix_List, xColumnLabels_List, xRowLabels_List: {}] /; (
      ArrayDepth[xMatrix] > 1) :=
30    AngleBracket @ Association[
31      "Matrix" -> xMatrix,
32      "Row␣Labels" -> If[xRowLabels === {},
33        Range @ (First @ (Dimensions @ xMatrix)),
34        xRowLabels
35        ],
36      "Column␣Labels" -> xColumnLabels
37      ]
38
39 AngleBracket[0, xRowLabels_List] :=
40    AngleBracket @ Association[
41      "Matrix" -> Array[0 &, {Length @ xRowLabels}],
42      "Row␣Labels" -> xRowLabels
43      ]
44
45 AngleBracket[0, xColumnLabels_List, xRowLabels_List] :=
46    AngleBracket @ Association[
47      "Matrix" -> Array[0 &, {Length @ xRowLabels, Length @ xColumnLabels
          }],
48      "Row␣Labels" -> xRowLabels,
49      "Column␣Labels" -> xColumnLabels
50      ]
51
52 AngleBracket[1, xLabels_List] :=
53    Association[
```

```
54      "Matrix" -> IdentityMatrix[Length @ xLabels],
55      "Row␣Labels" -> xLabels,
56      "Column␣Labels" -> xLabels
57      ]
58
59  AngleBracket[1, xColumnLabels_List, xRowLabels_List] :=
60    Module[{xId},
61      xId = Intersection[xColumnLabels, xRowLabels];
62      AngleBracket[
63        Association[
64          "Matrix" -> IdentityMatrix[Length @ xId],
65          "Row␣Labels" -> xId,
66          "Column␣Labels" -> xId
67          ],
68        AngleBracket[0, xColumnLabels, xRowLabels]
69        ]
70      ]
71
72  AngleBracket[xA_Association, xRowLabels_List] /; KeyExistsQ[xA, "Column␣
        Labels"] :=
73    AngleBracket[
74      AngleBracket[0, xA["Column␣Labels"], xRowLabels],
75      Association[
76        "Matrix" -> Part[xA["Matrix"],
77          Flatten @ (Position[xA["Row␣Labels"], #] & /@ Intersection[
                xRowLabels, xA["Row␣Labels"]]), All],
78        "Row␣Labels" -> Intersection[xRowLabels, xA["Row␣Labels"]],
79        "Column␣Labels" -> xA["Column␣Labels"]
80        ]
81      ]
82
83  AngleBracket[xA_Association, xRowLabels_List] :=
84    AngleBracket[
85      AngleBracket[0, xRowLabels],
86      Association[
```

```
87      "Matrix" -> Part[xA["Matrix"],
88        Flatten@(Position[xA["Row␣Labels"], #] & /@ Intersection[
                xRowLabels, xA["Row␣Labels"]])],
89      "Row␣Labels" -> Intersection[xRowLabels, xA["Row␣Labels"]]
90        ]
91      ]
92
93  AngleBracket[xA_Association, xRowLabel_] /; KeyExistsQ[xA, "Column␣
        Labels"] :=
94    If[First @ Dimensions @ (xA["Column␣Labels"]) == 1,
95      Part[xA["Matrix"], First @ (Flatten @ (Position[xA["Row␣Labels"],
          xRowLabel])), 1],
96      AngleBracket @ Association[
97        "Matrix" -> Part[xA["Matrix"], Flatten @ (Position[xA["Row␣Labels"
            ], xRowLabel]), All],
98        "Row␣Labels" -> {xRowLabel},
99        "Column␣Labels" -> xA["Column␣Labels"]
100       ]
101     ]
102
103 AngleBracket[xA_Association, xRowLabel_] :=
104   Part[xA["Matrix"], First @ (Flatten @ (Position[xA["Row␣Labels"],
        xRowLabel]))]
105
106 AngleBracket[xA_Association, xRowLabels_List, xColumnLabels_List] :=
107   AngleBracket[
108     AngleBracket[0, xColumnLabels, xRowLabels],
109     Association[
110       "Matrix" -> Part[xA["Matrix"],
111         Flatten @ (Position[xA["Row␣Labels"], #] & /@ Intersection[
                xRowLabels, xA["Row␣Labels"]]),
112         Flatten @ (Position[xA["Column␣Labels"], #] & /@ Intersection[
                xColumnLabels, xA["Column␣Labels"]])],
113       "Row␣Labels" -> Intersection[xRowLabels, xA["Row␣Labels"]],
114       "Column␣Labels" -> Intersection[xColumnLabels, xA["Column␣Labels"]]
```

```
115      ]
116    ]
117
118 AngleBracket[xA_Association, All, xColumnLabels_List] :=
119   AngleBracket[
120     AngleBracket[0, xColumnLabels, xA["Row␣Labels"]],
121     Association[
122       "Matrix" -> Part[xA["Matrix"], All,
123         Flatten @ (Position[xA["Column␣Labels"], #] & /@ Intersection[
124           xColumnLabels, xA["Column␣Labels"]])],
124       "Row␣Labels" -> xA["Row␣Labels"],
125       "Column␣Labels" -> Intersection[xColumnLabels, xA["Column␣Labels"]]
126     ]
127   ]
128
129 AngleBracket[xA_Association, xRowLabels_List, xColumnLabel_] :=
130   AngleBracket[
131     AngleBracket[0, {xColumnLabel}, xRowLabels],
132     Association[
133       "Matrix" -> Transpose @ ({Part[xA["Matrix"],
134         Flatten @ (Position[xA["Row␣Labels"], #] & /@ Intersection[
135           xRowLabels, xA["Row␣Labels"]]),
135         First @ (Flatten @ (Position[xA["Column␣Labels"], xColumnLabel]))
136         ]}),
136       "Row␣Labels" -> Intersection[xRowLabels, xA["Row␣Labels"]],
137       "Column␣Labels" -> {xColumnLabel}
138     ]
139   ]
140
141 AngleBracket[xA_Association, All, xColumnLabel_] :=
142   AngleBracket @ Association[
143     "Matrix" -> Transpose @ ({Part[xA["Matrix"], All,
144       First @ (Flatten @ (Position[xA["Column␣Labels"], xColumnLabel]))
145       ]}),
145     "Row␣Labels" -> xA["Row␣Labels"],
```

```
146      "Column␣Labels" -> {xColumnLabel}
147      ]
148
149  AngleBracket[xA_Association, xRowLabel_, xColumnLabels_List] :=
150    AngleBracket [
151      AngleBracket[0, xColumnLabels, {xRowLabel}],
152      Association[
153        "Matrix" -> Part[xA["Matrix"], First @ Flatten @ (Position[xA["Row␣
               Labels"], xRowLabel]),
154         Flatten @ (Position[xA["Column␣Labels"], #] & /@ Intersection[
               xColumnLabels, xA["Column␣Labels"]])],
155        "Row␣Labels" -> {xRowLabel},
156        "Column␣Labels" -> Intersection[xColumnLabels, xA["Column␣Labels"]]
157        ]
158      ]
159
160  AngleBracket[xA_Association, xRowLabel_, xColumnLabel_] :=
161    Part[xA["Matrix"], First @ Flatten @ (Position[xA["Row␣Labels"],
         xRowLabel]),
162      First @ Flatten @ (Position[xA["Column␣Labels"], xColumnLabel])]
163
164  AngleBracket[xFunction_, xA_Association] :=
165    If[KeyExistsQ[xA, "Column␣Labels"],
166      AngleBracket @ Association[
167        "Matrix"-> xFunction @ (xA["Matrix"]),
168        "Column␣Labels"-> xA["Column␣Labels"],
169        "Row␣Labels"-> xA["Row␣Labels"]
170        ],
171      AngleBracket @ Association[
172        "Matrix"-> xFunction @ (xA["Matrix"]),
173        "Row␣Labels"-> xA["Row␣Labels"]
174        ]
175      ]
176
177  SApply[xFunction_, xA_Association] := AngleBracket[xFunction, xA]
```

```
178
179  SAssemble = SPart = AngleBracket;
```

When `AngleBracket` is called with a sequence of matrices (sequence of `Association` elements, $\langle$X,Y,...$\rangle$), the output is a new `Association` element (representing a matrix) consisting of an assemble of the inputs in which elements having simultaneously the same row and column labels are added up. The "Row␣Labels" and "Column␣Labels" lists of the output consist of an sorted version of the union of all the respective lists of the inputs. Thus, in this usage, `AngleBracket` operator performs the opetations of matrix sum and assemble.

Let `xV` be a `List` element representing a column-matrix. `AngleBracket[xV, xRowLabels]` or $\langle$xV, xRowLabels$\rangle$ gives the `Association` element representing the column-matrix `xV` whose rows are labeled by `xRowLabels`.

Let `xM` be a `List` element representing a matrix (two-dimensional array). `AngleBracket[xM, xColumnLabels]` or $\langle$xM, xColumnLabels$\rangle$ gives the `Association` element representing the matrix `xM` whose rows are labeled by a sequence of integers and the columns are labeled by `xColumnLabels`. `AngleBracket[xM, xColumnLabels, xRowLabels]` or $\langle$xM, xColumnLabels$\rangle$ gives the `Association` element representing the matrix `xM` whose rows are labeled by `xRowLabels` and the columns are labeled by `xColumnLabels`.

`AngleBracket[0, xRowLabels]` or $\langle$0, xRowLabels$\rangle$ gives the `Association` element representing the null column-matrix whose rows are labeled by `xRowLabels`. `AngleBracket[0, xColumnLabels, xRowLabels]` or $\langle$0, xColumnLabels, xRowLabels$\rangle$ gives the `Association` element representing the null matrix whose rows are labeled by `xRowLabels` and the columns are labeled by `xColumnLabels`. `AngleBracket[1, xLabels]` or $\langle$1, xLabels$\rangle$ gives the `Association` element representing the identity whose rows and columns are labeled by `xLabels`. `AngleBracket[1, xColumnLabels, xRowLabels]` or $\langle$1, xColumnLabels, xRowLabels$\rangle$ gives the `Association` element representing the matrix defined by the Kronecker Delta function whose rows are labeled by `xRowLabels` and the columns are labeled by `xColumnLabels`.

`SApply[xFunction, xX]` or $\langle$xFunction, xX$\rangle$ applies the unary function `xFunction` to the entry whose key is "Matrix" in the `Association xX`.

All the other uses of `AngleBracket` correspond to partitioning of matrices. In these cases `AngleBracket` is called with a sequence of two or three arguments (the third argument is optional), in which the first one must correspond to a matrix (`Association` element), the second one can be a list of row labels, a single row label or the keyword `All` and the third (optional) can be a list of column labels or a single column label. When

16

a the first argument represents a column-matrix and the second is a single row label, or when the first represent a matrix, the second is a single row label and the third, a single column label, then the output of the operator is a the expression of the corresponding element (i.e., not a `List` nor an `Association`). In all the other cases, the output is an `Association` representing a matrix constituted only by the corresponding rows and columns of the input matrix. When the keyword `All` is used in the second argument, all rows of the original matrix are selected. When the third argument is not used, all the columns of the original matrix are selected.

## 3.2 Matrix multiplication and multiplication of a matrix by a scalar

```
1  CircleDot[xX_Association, xY_Association] /;
2    And[xX["Matrix"] === {}, ArrayDepth[xY["Matrix"]] === 1] :=
3    Association[
4      "Matrix" -> {},
5      "Row Labels" -> {}
6      ]
7
8  CircleDot[xX_Association, xY_Association] /;
9    And[xX["Matrix"] === {}] :=
10   Association[
11     "Matrix" -> {},
12     "Row Labels" -> {},
13     "Column Labels" -> xY["Column Labels"]
14     ]
15
16 CircleDot[xX_Association, xY_Association] /;
17   And[ArrayDepth[xX["Matrix"]] === 1, ArrayDepth[xY["Matrix"]] === 1] :=
18   Module[{xA, xB, xU},
19     xU = Association[
20       "Matrix" -> Array[0&, Length @ #],
21       "Row Labels" -> #
22       ] & @ Union[xX["Row Labels"], xY["Row Labels"]];
23     xA = SAssemble[xX, xU];
24     xB = SAssemble[xY, xU];
```

```
25      xA["Matrix"].xB["Matrix"]
26      ]
27
28  CircleDot[xX_Association, xY_Association] /;
29    And[ArrayDepth[xY["Matrix"]] === 1] :=
30    Module[{xA, xB},
31      xA = Association[
32        "Matrix" -> Array[0&, {Length @ #1, Length @ #2}],
33        "Row␣Labels" -> #1,
34        "Column␣Labels" -> #2
35        ] & @@ {xX["Row␣Labels"], Union[xX["Column␣Labels"], xY["Row␣Labels
            "]]};
36      xB = Association[
37        "Matrix" -> Array[0&, Length @ #],
38        "Row␣Labels" -> #
39        ] & @ Union[xX["Column␣Labels"], xY["Row␣Labels"]];
40      xA = SAssemble[xX, xA];
41      xB = SAssemble[xY, xB];
42      Association[
43        "Matrix" -> xA["Matrix"].xB["Matrix"],
44        "Row␣Labels" -> xA["Row␣Labels"]
45        ]
46      ]
47
48  CircleDot[xX_Association, xY_Association] :=
49    Module[{xA, xB},
50      xA = Association[
51        "Matrix" -> Array[0&, {Length @ #1, Length @ #2}],
52        "Row␣Labels" -> #1,
53        "Column␣Labels" -> #2
54        ] & @@ {xX["Row␣Labels"], Union[xX["Column␣Labels"], xY["Row␣Labels
            "]]};
55      xB = Association[
56        "Matrix" -> Array[0&, {Length @ #1, Length @ #2}],
57        "Row␣Labels" -> #1,
```

```
58        "Column␣Labels" -> #2
59        ] & @@ {Union[xX["Column␣Labels"], xY["Row␣Labels"]], xY["Column␣
             Labels"]};
60     xA = SAssemble[xX, xA];
61     xB = SAssemble[xY, xB];
62     Association[
63       "Matrix" -> xA["Matrix"].xB["Matrix"],
64       "Row␣Labels" -> xA["Row␣Labels"],
65       "Column␣Labels" -> xB["Column␣Labels"]
66       ]
67     ]
68
69 CircleDot[xX_Association, xY_List] :=
70   xX["Matrix"].xY
71
72 CircleDot[xX_Association, xY_] :=
73   SApply[(xY #)&, xX]
74
75 CircleDot[xY_, xX_Association] :=
76   SApply[(xY #)&, xX]
77
78 SDot = CircleDot;
```

In the package MathMatrixPack, the operator `CircleDot`, denote by X⊙Y or the function `SDot[X, Y]` is used to denote the operations of matrix multiplication and multiplication of a matrix by a scalar. In the case of matrix multiplication, either both `CircleDot` arguments are `Association` elements or the first one is an `Association` element and the second one a `List` element. If the second argument is an `Association`, the output will be an `Association` representing the matrix multiplication between both input arguments. If the second argument is a `List`, the output will be a `List` representing the matrix multiplication between both input arguments. In the case of multiplication of a matrix by a scalar, one argument must be an `Association` and the other an scalar. The order of the arguments is not relevant in this case, and the output is an `Association` representing the corresponding multplication of the matrix by the scalar.

## 3.3 Matrix transposition

```
1 SuperDagger[xX_Association] :=
2   Association[
3     "Matrix"->Transpose @ xX["Matrix"],
4     "Column␣Labels"-> xX["Row␣Labels"],
5     "Row␣Labels"-> xX["Column␣Labels"]
6     ]
7
8 STranspose[xX_Association] := SuperDagger[xX]
```

In the package MathMatrixPack, the operator `SuperDagger`, denote by $X^\dagger$ is used to denote the operation of transposition of matrices. It extends the use of the built-in function `Transpose` (that is applicable to `List` elements representing matrices) to `Association` elements representing matrices. The unary function `STranspose` does the same as the operator `SuperDagger`.

## 3.4 Affine Transformations

```
1 BracketingBar[xX_Association] :=
2   AffineTransform[xX["Matrix"]]
3
4 BracketingBar[xX_List /; Dimensions[xX]=={3,3}] :=
5   AffineTransform[xX]
6
7 BracketingBar[xX_List /; Dimensions[xX]=={4,4}] :=
8   LinearFractionalTransform[xX]
```

In the package MathMatrixPack, the operator `BracketingBar`, denote by $\lfloor X \rfloor$ is used to convert matrices into affine operators. Whenever the (single) argument of the operator is an `Association`, the output is a `TransformationFunction` given by the application of the built-in `AffineTransform` to the `Association` entry whose key is `"Matrix"`. The same kind of output will be obtained if the argument of the operator is a $3 \times 3$ `List` element. However, when the argument is a $4 \times 4$ `List` element, the corresponding `TransformationFunction` is obtained by the application of the built-in `LinearFractionalTransform` function (whose output represents a homogeneous transformation).

## 3.5 Coefficient arrays

```
1  SCoefficientArrays[xA_Association, xVariables_List, xRules_List:{}] :=
2    Module[{x},
3      x["Row␣Labels"] = xA["Row␣Labels"];
4      x["Expressions"] = Flatten @ (xA["Matrix"]);
5      x["Coefficient␣Arrays"] = CoefficientArrays[x["Expressions"] //.
         xRules, xVariables];
6      {
7      Association[
8        "Matrix" -> Part[#, 1]& @ x["Coefficient␣Arrays"],
9        "Row␣Labels" -> x["Row␣Labels"]
10       ],
11     Association[
12       "Matrix" -> Part[#, 2]& @ x["Coefficient␣Arrays"],
13       "Row␣Labels" -> x["Row␣Labels"],
14       "Column␣Labels" -> xVariables
15       ]
16     }
17     ]
18
19
20 SMatrixCoefficientArrays[xA_Association, xRules_List: {}] :=
21   Module[{xxMatrix, xxVariables, xxRowLabels, xxColumnLabels,
         xxCoefficientMatrices},
22     xxMatrix = xA["Matrix"] //. xRules;
23     xxRowLabels = xA["Row␣Labels"];
24     xxColumnLabels = xA["Column␣Labels"];
25     xxVariables = Union @ GetVariables[xxMatrix];
26     xxCoefficientMatrices = CoefficientArrays[xxMatrix,xxVariables];
27     {
28     Association[ Union@@{
29       {
30       1->
31       Association[
32       "Matrix"->Normal@Part[xxCoefficientMatrices,1],
```

```
33          "Column␣Labels"->xxColumnLabels,
34          "Row␣Labels"->xxRowLabels
35          ]
36        },
37        MapThread[ (#1-> Association[
38          "Matrix"->Normal@Part[xxCoefficientMatrices,2,All,All,#2],
39          "Column␣Labels"->xxColumnLabels,
40          "Row␣Labels"->xxRowLabels
41          ])&, {#,Range@Length@#}, 1]& @ xxVariables
42        }],
43      xxVariables
44      }
45    ]
```

`SCoefficientArrays` is an extension of the built-in function `CoefficientArrays` that is applicable to matrices represented by `Association` elements. This function can be called with two or three arguments (being the third optional), i.e., both syntaxes `SCoefficientArrays[M, V, R]` and `SCoefficientArrays[M, V]` are valid. In both cases, the function transforms the `Association` element `M` in a `List` of expressions `E`, applies to this list the transformation rules `R` whenever they are defined, and returns a `List` element {`K`, `H`}, containing two `Association` elements, `K` and `H`, such that the affine part of `E` (i.e., terms of the expressions in `E` that are either independent or linear dependent of the variables in `V`) is given by ⟨`K`, `H`⊙`V`⟩.

In order to understand how the function `SMatrixCoefficientArrays` works, consider a matrix $M$ that may be dependent of some scalar variables $(v_1, \ldots, v_r)$, i.e., $M = \underline{M}(v_1, \ldots, v_r)$. If $M$ is affine with respect to these variables, then there is a list of constant matrices $M_1, M_{v_1}, \ldots, M_{v_r}$ such that:

$$M = 1\,M_1 + \sum_{k=1}^{r} v_k\,M_{v_k}$$

`SMatrixCoefficientArrays[M]` or `SMatrixCoefficientArrays[M,R]` are valid syntaxes for this function, with `M` being an `Association` element representing a matrix $M$ and with `M` being an optional `List` of replacement rules, to be applied to this matrix. The output is the `List` {`X`, `V`}, with `X` being an `Association` element of the form

```
1  Association[1 -> M₁, v₁ -> M_{v₁}, ..., v_r -> M_{v_r} ]
```

($M_1$, $M_{v_1}$, ..., $M_{v_r}$ are the `Association` elements representing the corresponding coefficient matrices $M_1, M_{v_1}, \ldots, M_{v_r}$) and with `V` being the `List` $\{v_1, \ldots, v_r\}$.

## 3.6 Linear Solve

```
1 SLinearSolve[xX_Association,xY_Association] :=
2   Module[{xA,xB},
3     xA = SAssemble[xX];
4     xB = SAssemble[xY];
5     If[xA["Row Labels"] === xB["Row Labels"],
6       Association[
7         "Matrix" -> LinearSolve[xA["Matrix"], xB["Matrix"]],
8         "Column Labels" -> xB["Column Labels"],
9         "Row Labels" -> xA["Column Labels"]
10        ],
11        "Error"
12      ]
13    ]
```

`SLinearSolve` extends the application of the built-in function `LinearSolve` (originally applicable to a pair of `List` elements representing matrices) to pairs of `Association` elements representing matrices. The output of `SLinearSolve[A, B]` is an `Association` element `Z` such that `A⊙Z == B`.

```
1 SLeastSquares[xX_Association,xY_Association] :=
2   Module[{xA,xB},
3     xA = SAssemble[xX];
4     xB = SAssemble[xY];
5     If[xA["Row Labels"] === xB["Row Labels"],
6       Association[
7         "Matrix" -> LeastSquares[xA["Matrix"], xB["Matrix"]],
8         "Column Labels" -> xB["Column Labels"],
9         "Row Labels" -> xA["Column Labels"]
10        ],
11        "Error"
12      ]
13    ]
```

SLeastSquares extends the application of the built-in function `LeastSquares` (originally applicable to a pair of `List` elements representing matrices) to pairs of `Association` elements representing matrices. The output of `SLeastSquares[A, B]` is an `Association` element Z which is a least squares solution for X in the matrix equation $\langle A\odot X, B\rangle$ == 0.

```
 1  LSSolver[xEquations_List, xGenVariables_List, xIndVariables_List,
 2    xRules_List: {}, xExtraRules_List: {}, xSize_Integer: 0,
          xTestParameters_List: {}, xSymmetry_: Automatic] :=
 3    Module[{xIn, xSol, xXe, xXi, xC},
 4      xXi = Union[Complement[GetVariables @ xEquations, xGenVariables],
            xIndVariables];
 5      xIn = Select[xEquations, (Length[#] <= xSize) &];
 6      xXe = Complement[GetVariables @ xIn, xXi];
 7      xSol = Flatten @ (Quiet @ Solve[(# == 0)& /@ xIn, xXe]);
 8      (*  xS = Jacobi[
 9        xGenVariables //. xSol,
10        Union[Complement[xGenVariables, First /@ xSol], xXi],
11        xGenVariables
12        ]; *)
13      xIn = Collect[RedundantElim @ (Expand @ (xEquations //. xSol) //.
            xExtraRules), xX_[t], Simplify] //. xRules;
14      xC = LSReferenceOrthogonalComplement[
15        Jacobi[#, Union[GetVariables[#], xXi]]& @ xIn ,
16        xXi, xSymmetry, xTestParameters
17        ];
18      (* xS = xS ~SDot~ xC; *)
19      {
20        Union[
21          xSol,
22          Select[ MapThread[(#1 -> #2) &,
23            {#["Row␣Labels"], (#["Matrix"].#["Column␣Labels"])} /.
24              (xSymmetry["Extra␣Rules"] //. Missing[xX__] -> {})},
25            1] & @ xC, Not @ (Expand[First[#] - Last[#]] === 0) &]],
26        xC["Test␣Parameters"]
27        }
```

```
28      ]
```

## 3.7 Jacobians

```
1  Jacobi[xExpressionsList_, xVariablesList_] :=
2    Association[
3      "Matrix" -> D[xExpressionsList, {xVariablesList}],
4      "Column␣Labels" -> xVariablesList,
5      "Row␣Labels" -> Range @@ Dimensions @ xExpressionsList
6      ]
7
8  Jacobi[xExpressionsList_,xVariablesList_, xExpressionsLabels_] :=
9    Association[
10     "Matrix" -> D[xExpressionsList, {xVariablesList}],
11     "Column␣Labels" -> xVariablesList,
12     "Row␣Labels" -> xExpressionsLabels
13     ]
```

Jacobi obtains the Jacobian matrix of a given list of expressions with respect to a list of variables. The syntax of this function is Jacobi[E, V, L] or Jacobi[E, V] (i.e., the third argument is optional). E is an expression or a list of symbolic expressions, V is a list of variables and L is a list of labels for the corresponding expressions. The output is an Association element, representing the Jacobian matrix of E with respect to the variables in V. The "Column␣Labels" entry of the output is the list V and the "Row␣Labels" entry is L, if it is an input argument, or a list of positive integer indexes, otherwise.

## 3.8 Orthogonal complement

```
1  OrthogonalComplement[xJacobian_] :=
2    Module[{x},
3      x["Null␣Space␣Matrix"] = Transpose @ NullSpace[xJacobian["Matrix"]];
4      x["Independent␣Variations"] = (Range @ Part[Dimensions[x["Null␣Space
          ␣Matrix"]], 2]);
5      Association[
```

```
 6        "Matrix" -> x["Null␣Space␣Matrix"],
 7        "Column␣Labels" -> x["Independent␣Variations"],
 8        "Row␣Labels" -> xJacobian["Column␣Labels"]
 9        ]
10     ]
11
12 OrthogonalComplement[xJacobian_, xLabel_String] :=
13   Module[{x},
14     x["Null␣Space␣Matrix"] = Transpose @ NullSpace[xJacobian["Matrix"]];
15     x["Independent␣Variations"] = Subscript[OverTilde[q],xLabel,#][t]& /
         @
16     (Range @ Part[Dimensions[x["Null␣Space␣Matrix"]], 2]);
17     Association[
18       "Matrix" -> x["Null␣Space␣Matrix"],
19       "Column␣Labels" -> x["Independent␣Variations"],
20       "Row␣Labels" -> xJacobian["Column␣Labels"]
21       ]
22     ]
23
24 OrthogonalComplement[xJacobian_, xIndependentVariablesList_List] :=
25   Module[{x, xOrthogonalComplement},
26     {x["Number␣of␣Constraints"], x["Number␣of␣Variables"]} = Dimensions[
         xJacobian["Matrix"]];
27     x["Number␣of␣Degrees␣of␣Freedom"] = x["Number␣of␣Variables"] - x["
         Number␣of␣Constraints"];
28     If[ x["Number␣of␣Degrees␣of␣Freedom"] === Length @
         xIndependentVariablesList,
29     (*-TRUE-*)
30       x["Independent␣Variables␣Column␣Indexes"] =
31         Flatten[ Position[xJacobian["Column␣Labels"], #]& /@
             xIndependentVariablesList, Infinity];
32       x["Redundant␣Variables␣Column␣Indexes"] =
33         Complement[Range @@ Dimensions @ xJacobian["Column␣Labels"], x["
             Independent␣Variables␣Column␣Indexes"]];
34       xOrthogonalComplement = Association[
```

```
35        "Matrix" -> Array[0&, {x["Number␣of␣Variables"], x["Number␣of␣
            Degrees␣of␣Freedom"]}],
36        "Column␣Labels" -> xIndependentVariablesList,
37        "Row␣Labels" -> xJacobian["Column␣Labels"]
38        ];
39      xOrthogonalComplement[["Matrix", x["Independent␣Variables␣Column␣
            Indexes"]]] =
40        IdentityMatrix @ x["Number␣of␣Degrees␣of␣Freedom"];
41      xOrthogonalComplement[["Matrix", x["Redundant␣Variables␣Column␣
            Indexes"]]] =
42        LinearSolve @@ {
43          xJacobian[["Matrix", All, x["Redundant␣Variables␣Column␣Indexes
              "]]],
44          -xJacobian[["Matrix", All, x["Independent␣Variables␣Column␣
              Indexes"]]]
45          };
46      xOrthogonalComplement,
47      (*-FALSE-*)
48      "Error"
49      ]
50    ]
```

OrthogonalComplement calculates an orthogonal complement of a (Jacobian) matrix. Two syntaxes are possible for this function:

- OrthogonalComplement[A] calculates *an* orthogonal complement for the matrix represented by the Association element A using the built-in NullSpace function. The output is an Association element C whose "Row␣Labels" entry is equal to the "Column␣Labels" entry of the input argument and whose "Column␣Labels" entry is a list of positive integer indexes; also, A⊙C == 0.

- OrthogonalComplement[A, V] calculates *the* orthogonal complement for the matrix represented by the Association element A with respect to the independent set of variables represented by the List element V using the built-in LinearSolve function. The output is an Association element C whose "Row␣Labels" entry is equal to the "Column␣Labels" entry of the input argument and whose "Column␣Labels" entry is equal to V; also, A⊙C == 0.

```
1 LSNumericalOrthogonalComplement[xJacobian_,
    xIndependentVariablesList_List] :=
2  Module[{x, xOrthogonalComplement},
3    {x["Number of Constraints"], x["Number of Variables"]} = Dimensions[
        xJacobian["Matrix"]];
4    x["Number of Degrees of Freedom"] = Part[Dimensions @
        xIndependentVariablesList, 1];
5    x["Independent Variables Column Indexes"] =
6      Flatten[Position[xJacobian["Column Labels"], #]& /@
          xIndependentVariablesList, Infinity];
7    x["Redundant Variables Column Indexes"] =
8      Complement[Range @@ Dimensions @ xJacobian["Column Labels"], x["
          Independent Variables Column Indexes"]];
9    xOrthogonalComplement = Association[
10     "Matrix"->Array[0&, {x["Number of Variables"], x["Number of Degrees
          of Freedom"]}],
11     "Column Labels" -> xIndependentVariablesList,
12     "Row Labels" -> xJacobian["Column Labels"]
13     ];
14   xOrthogonalComplement[["Matrix", x["Independent Variables Column
        Indexes"]]] =
15     IdentityMatrix @ x["Number of Degrees of Freedom"];
16   xOrthogonalComplement[["Matrix", x["Redundant Variables Column
        Indexes"]]] =
17     LeastSquares @@ {
18       xJacobian[["Matrix", All, x["Redundant Variables Column Indexes"
          ]]],
19       -xJacobian[["Matrix", All, x["Independent Variables Column
          Indexes"]]]
20       };
21   xOrthogonalComplement
22   ]
23
```

```
24 LSReferenceOrthogonalComplement[xJacobian_Association,
     xIndependentVariables_List,
25   xSymmetry_: Automatic, xTestParameters_List: {}, xNZero_Rational:1
       10^-5] :=
26   Module[{x, xNC, xSC, xNTestParameters},
27     xNTestParameters = Union[
28       xTestParameters,
29       If[xSymmetry === Automatic,
30         (#-> RandomReal[1])& /@ (GetAllVariables[(Flatten @ (Union @@ (
             Normal @
31         (#["Matrix"])& @ xJacobian))) //. xTestParameters]),
32         xSymmetry["Function"] /@ (GetAllVariables[(Flatten @ (Union @@ (
             Normal @
33         (#["Matrix"])& @ xJacobian))) //. xTestParameters])
34       ]
35     ];
36     xNC = LSNumericalOrthogonalComplement[
37       SReplaceRepeated[xJacobian, xNTestParameters],
38       xIndependentVariables
39     ];
40     xNC = AppendTo[xNC, "Matrix" -> Round[xNC["Matrix"], xNZero]];
41     x["Column␣Labels"] = xNC["Column␣Labels"] //. SymbolReplacements;
42     x["Row␣Labels"]= xNC["Row␣Labels"] //.SymbolReplacements;
43
44     x["New␣Parameters"]={};
45     If[xSymmetry === Automatic,
46       (*-TRUE-*)
47       Function[xRowLabel,
48         x["Row␣Number"] = First @ (Flatten @ Position[x["Row␣Labels"],
             xRowLabel]);
49         x["Parameters␣Values"] = Flatten @ (Part[xNC["Matrix"], x["Row␣
             Number"]]);
50         x["Parameters␣Names"] = Flatten @
51           ((Function[{xColumnLabel}, Subscript[OverBar[\[CapitalGamma]],
               xRowLabel,xColumnLabel]]) /@
```

```mathematica
52            x["Column␣Labels"]);
53        x["New␣Parameters:1"] = MapThread[(#2-> #1)&, {x["Parameters␣
              Values"], x["Parameters␣Names"]}, 1];
54        x["New␣Parameters:2"] = (Flatten @ (Normal @ DeleteCases[
              Association[x["New␣Parameters:1"]], _Integer]));
55        xNTestParameters = Union[xNTestParameters, N[x["New␣Parameters:2"
              ]]];
56        x["New␣Parameters"] = Union[
57          x["New␣Parameters"],
58          (Reverse /@ x["New␣Parameters:2"]),
59          (Reverse /@ x["New␣Parameters:2"]) /. ((xA_->xB_ )->(-xA->-xB )
              )
60          ];
61        ]/@ x["Row␣Labels"],
62      (*-FALSE-*)
63      Function[{xRowLabel},
64        If[Intersection[xSymmetry["Secondary"],
65          Flatten @ (Characters /@ Select[xRowLabel /. {Subscript[xV_,
              xS__] -> {xV, xS}, xV_ -> {xV}}, StringQ])
66          ] === {}
67          (* Not @ And[
68          StringQ[Quiet @ Last[xRowLabel]],
69          StringTake[Last[xRowLabel], -1] === xSymmetry["Secondary"]
70          ] *),
71          x["Row␣Number"] = First @ (Flatten @ Position[x["Row␣Labels"],
              xRowLabel]);
72          x["New␣Subscript"] = If[Intersection[xSymmetry["Primary"],
73            Flatten @ (Characters /@ Select[xRowLabel /. {Subscript[xV_,
                xS__] -> {xV, xS}, xV_ -> {xV}}, StringQ])
74            ] === {}
75            (* Quiet @ (StringTake[Last[xRowLabel], -1] === xSymmetry["
                Primary"]) *),
76            xRowLabel,
77            Subscript @@ (If[StringQ[#], StringReplace[#, (# -> "")& /@
                xSymmetry["Primary"]], #] & /@
```

```
78              (xRowLabel /. (Subscript[xV_, xS__] -> {xV, xS})))
79            ];
80          x["Parameters␣Values"] = Flatten @ (Part[xNC["Matrix"], x["Row␣
              Number"]]);
81          x["Parameters␣Names"] = Flatten @ ((Function[{xLabel},
82              Subscript[OverBar[\[CapitalGamma]], x["New␣Subscript"],
                xLabel]
83            ]) /@ x["Column␣Labels"]);
84          x["New␣Parameters:1"] = MapThread[(#2 -> #1) &,
85            {x["Parameters␣Values"], x["Parameters␣Names"]}, 1];
86          x["New␣Parameters:2"] = (Flatten @ (Normal @
87            DeleteCases[Association[x["New␣Parameters:1"]], _Integer]));
88          xNTestParameters = Union[xNTestParameters, N[x["New␣Parameters
              :2"]]];
89          x["New␣Parameters"] = Union[
90            x["New␣Parameters"],
91            (Reverse /@ x["New␣Parameters:2"]),
92            (Reverse /@ x["New␣Parameters:2"]) /. ((xA_ -> xB_) -> (-xA
              -> -xB))
93            ];
94          ];
95        ] /@ x["Row␣Labels"];
96      ];
97     xSC = Association[xNC, "Matrix"-> (xNC["Matrix"] //. x["New␣
        Parameters"]),
98      "Test␣Parameters" -> xNTestParameters];
99     xSC
100     ]
101
102
103 LSLinearizedOrthogonalComplement[xLinearizedJacobian_Association,
      xIndependentVariables_List,
104   xCoordinatesReplacements_: {}, xSymmetry_: Automatic,
        xTestParameters_List: {}, xNZero_Rational:1 10^-5] :=
105   Module[{x, xE, xLSOC, xCoordinates, xLinearizedJacobianCoefficients,
```

```
106    xNTestParameters, xNA1, xNC1, xNCq, xSC1, xSCq},
107
108    {xLinearizedJacobianCoefficients,xCoordinates} =
109      SMatrixCoefficientArrays[xLinearizedJacobian];
110    xNTestParameters = Union[
111      xTestParameters,
112      If[xSymmetry === Automatic,
113        (#-> RandomReal[1])& /@ (GetAllVariables[(Flatten @ (Union @@ (
             Normal @
114        (#["Matrix"])& /@ xLinearizedJacobianCoefficients))) //.
             xTestParameters]),
115      xSymmetry["Function"] /@ (GetAllVariables[(Flatten @ (Union @@ (
             Normal @
116        (#["Matrix"])& /@ xLinearizedJacobianCoefficients))) //.
             xTestParameters])
117      ]
118    ];
119
120    xNA1 = SReplaceRepeated[xLinearizedJacobianCoefficients[1],
           xNTestParameters];
121    xNC1 = LSNumericalOrthogonalComplement[xNA1, xIndependentVariables];
122    xNC1 = AppendTo[xNC1, "Matrix" -> Round[xNC1["Matrix"], xNZero]];
123    x["Column␣Labels"] = xNC1["Column␣Labels"] //. SymbolReplacements;
124    x["Row␣Labels"]= xNC1["Row␣Labels"] //. SymbolReplacements;
125
126    xE = 1 10^-3;
127    xNCq = Association[
128      (#->SAssemble[
129        (+1/(2 xE)) ~SDot~ SAssemble[ LSNumericalOrthogonalComplement[
130          SAssemble[ xNA1, (+xE) ~SDot~ SReplaceRepeated[
               xLinearizedJacobianCoefficients[#],
131          xNTestParameters]], xIndependentVariables],
132        (-1) ~SDot~ xNC1],
133      (-1/(2 xE)) ~SDot~ SAssemble[ LSNumericalOrthogonalComplement[
```

32

```
134      SAssemble[ xNA1, (-xE) ~SDot~ SReplaceRepeated[
             xLinearizedJacobianCoefficients[#],
135        xNTestParameters]], xIndependentVariables],
136        (-1) ~SDot~ xNC1]
137       ]
138     )& /@ xCoordinates
139     ];
140
141   (xNCq[#] = AppendTo[xNCq[#], "Matrix" -> Round[xNCq[#]["Matrix"],
         xNZero]])& /@ xCoordinates;
142
143   x["New␣Parameters"]={};
144   If[xSymmetry === Automatic,
145    (*-TRUE-*)
146    Function[xRowLabel,
147      x["Row␣Number"] = First @ (Flatten @ Position[x["Row␣Labels"],
           xRowLabel]);
148      x["Parameters␣Values"] = Flatten @ (Join[Part[xNC1["Matrix"], x["
           Row␣Number"]],
149        Join @@ ((Part[xNCq[#]["Matrix"],x["Row␣Number"]])& /@
             xCoordinates)]);
150      x["Parameters␣Names"] = Flatten @ (Join[
151        ((Function[{xColumnLabel}, Subscript[OverBar[\[CapitalDelta
             ]],1,xRowLabel,xColumnLabel]]) /@
152          x["Column␣Labels"]),
153        Join @@ (
154          ((Function[{xColumnLabel}, Subscript[OverBar[\[CapitalDelta
               ]],#,xRowLabel,xColumnLabel]]) /@
155            x["Column␣Labels"])& /@ (xCoordinates //.
                 SymbolReplacements)
156          )
157        ]);
158      x["New␣Parameters:1"] = MapThread[(#2-> #1)&, {x["Parameters␣
           Values"], x["Parameters␣Names"]}, 1];
```

```
159    x["New␣Parameters:2"] = (Flatten @ (Normal @ DeleteCases[
           Association[x["New␣Parameters:1"]], _Integer]));
160    xNTestParameters = Union[xNTestParameters, N[x["New␣Parameters:2"
           ]]];
161    x["New␣Parameters"] = Union[
162      x["New␣Parameters"],
163      (Reverse /@ x["New␣Parameters:2"]),
164      (Reverse /@ x["New␣Parameters:2"]) /. ((xA_->xB_ )->(-xA->-xB )
           )
165      ];
166    ]/@ x["Row␣Labels"],
167    (*-FALSE-*)
168    Function[{xRowLabel},
169      If[Intersection[xSymmetry["Secondary"],
170        Flatten @ (Characters /@ Select[xRowLabel /. {Subscript[xV_,
             xS__] -> {xV, xS}, xV_ -> {xV}}, StringQ])
171         ] === {}
172        (* Not @ And[
173        StringQ[Quiet @ Last[xRowLabel]],
174        MemberQ[xSymmetry["Secondary"], StringTake[Last[xRowLabel],
             -1]]
175        ] *),
176        x["Row␣Number"] = First @ (Flatten @ Position[x["Row␣Labels"],
             xRowLabel]);
177        x["New␣Subscript"] = If[Intersection[xSymmetry["Primary"],
178          Flatten @ (Characters /@ Select[xRowLabel /. {Subscript[xV_,
               xS__] -> {xV, xS}, xV_ -> {xV}}, StringQ])
179           ] === {}
180          (* Quiet @ (StringTake[Last[xRowLabel], -1] === xSymmetry["
               Primary"]) *),
181          xRowLabel,
182          Subscript @@ (If[StringQ[#], StringReplace[#, (# -> "")& /@
               xSymmetry["Primary"]], #] & /@
183            (xRowLabel /. (Subscript[xV_, xS__] -> {xV, xS})))
184          ];
```

34

```
185    x["Parameters␣Values"] = Flatten @ (Join[
186      Part[xNC1["Matrix"], x["Row␣Number"]],
187      Join @@ ((Part[xNCq[#]["Matrix"], x["Row␣Number"]]) & /@
             xCoordinates)
188      ]);
189    x["Parameters␣Names"] = Flatten @ (Join[
190      (Function[{xLabel},
191        Subscript[OverBar[\[CapitalDelta]], 1, x["New␣Subscript"],
             xLabel]
192      ]) /@ x["Column␣Labels"],
193      Join @@ (((
194        Function[{xLabel},
195          Subscript[OverBar[\[CapitalDelta]], #, x["New␣Subscript"],
               xLabel]
196        ]) /@ x["Column␣Labels"]
197      ) & /@ (xCoordinates //. SymbolReplacements))
198      ]);
199    x["New␣Parameters:1"] = MapThread[(#2 -> #1) &,
200      {x["Parameters␣Values"], x["Parameters␣Names"]}, 1];
201    x["New␣Parameters:2"] = (Flatten @ (Normal @
202      DeleteCases[Association[x["New␣Parameters:1"]], _Integer]));
203    xNTestParameters = Union[xNTestParameters, N[x["New␣Parameters
          :2"]]];
204    x["New␣Parameters"] = Union[
205      x["New␣Parameters"],
206      (Reverse /@ x["New␣Parameters:2"]),
207      (Reverse /@ x["New␣Parameters:2"]) /. ((xA_ -> xB_) -> (-xA
          -> -xB))
208      ];
209    ];
210    ] /@ x["Row␣Labels"];
211  ];
212
213  xSC1 = Association[xNC1, "Matrix"-> (xNC1["Matrix"] //. x["New␣
      Parameters"])];
```

```
214    (xSCq[#] = Association[xNCq[#], "Matrix"-> (xNCq[#]["Matrix"] //. x[
           "New␣Parameters"])])& /@ xCoordinates;
215    xLSOC = SAssemble[xSC1, Inner[#1 ~SDot~ #2&, xCoordinates, xSCq /@
           xCoordinates, SAssemble]];
216    xLSOC["Test␣Parameters"] = xNTestParameters;
217    xLSOC
218    ]
```

LSLinearizedOrthogonalComplement provides an symbolic expression for the linearized form of the orthogonal complement of a non-linear Jacobian matrix. The syntax for this function is LSLinearizedOrthogonalComplement[J,V,R,C,Z,P]:

- J is an Association element representing a non-linear Jacobian matrix.

- V is a List of the variables among the "Column␣Labels" of J that are considered as independent.

- R is a List element consisting of replacement rules for the reference values of the generalized variables in the expression of J (*optional argument whose default value is an empty* List).

- C is a List element consisting of replacement rules for the linearized expressions of some of the generalized variables in the expression of J (*optional argument whose default value is an empty* List).

- Z is a rational number expressing the precision of the numerical algorithms present in the function; numbers whose difference is less than Z are considered as equal during the execution of the algorithm (*optional argument whose default value is* $1 \cdot 10^{-5}$).

- P is a List element consisting of replacement rules for the values of some of the parameters in the expression of J (*optional argument whose default value is an empty* List).

The output of this function is a List element {A,C,T} in which:

- A is an Association element representing the symbolic linearized expression of J.

- C is an Association element representing the symbolic linearized expression of an orthogonal complement of J.

- T is a `List` element consisting of replacement rules for the test values (i.e., random or prescribed values used in the algorithm for obtaining the expression of `C`) of the parameters of the symbolic expression of `J`.

# 4 Rotation and homogeneous transformations

## 4.1 Rotation transformation

```
1  Rotation = Function @ Module[{x},
2    x["TransformList"] = List[##] /. {
3      "x" -> (RotationTransform[#, {1,0,0}]&),
4      "y" -> (RotationTransform[#, {0,1,0}]&),
5      "z" -> (RotationTransform[#, {0,0,1}]&),
6      "X" -> (RotationTransform[#, {1,0,0}]&),
7      "Y" -> (RotationTransform[#, {0,1,0}]&),
8      "Z" -> (RotationTransform[#, {0,0,1}]&)
9      };
10   Function[(TransformationMatrix @ (Simplify @ Inner[(#1 @ #2)&, x["
        TransformList"], List[##], Dot]))[[1;;3,1;;3]]]
11   ];
12
13 HomogToRot = #[[1;;3,1;;3]]&;
14
15 QuatToRot = {
16   {#1^2-#2^2-#3^2+#4^2,2 #1 #2-2 #3 #4,2 #1 #3+2 #2 #4},
17   {2 #1 #2+2 #3 #4,-#1^2+#2^2-#3^2+#4^2,2 #2 #3-2 #1 #4},
18   {2 #1 #3-2 #2 #4,2 #2 #3+2 #1 #4,-#1^2-#2^2+#3^2+#4^2}
19   }&;
```

`Rotation[$\mathbf{e}_1, \ldots, \mathbf{e}_r$][$\theta_1, \ldots, \theta_r$]` gives the transformation matrix associated to successive rotations around the axes $\mathbf{e}_1, \ldots, \mathbf{e}_r$ (being $\theta_1, \ldots, \theta_r$ the corresponding rotation angles). In this syntax, an axis $\mathbf{e}_k$ can be defined either by a `List` element representing the three Cartesian coordinates of a vector aligned to the axis of rotation in the local basis coordinates or by a `String` element "x", "y" or "z" whenever any of the canonical local axis is the corresponding axis of rotation.

## 4.2 Homogeneous transformation

```
1  Homogeneous = Function @ Module[{x},
2    x["TransformList"] = List[##] /. {
3      "Rx" -> (RotationTransform[#, {1,0,0}]&),
4      "Ry" -> (RotationTransform[#, {0,1,0}]&),
5      "Rz" -> (RotationTransform[#, {0,0,1}]&),
6      "R"[xVector_] -> (RotationTransform[#, xVector]&),
7      "Tx" -> (TranslationTransform[# {1,0,0}]&),
8      "Ty" -> (TranslationTransform[# {0,1,0}]&),
9      "Tz" -> (TranslationTransform[# {0,0,1}]&),
10     "T"[xVector_] -> (TranslationTransform[# xVector]&)
11     };
12   Function[TransformationMatrix @ (Simplify @ Inner[(#1 @ #2)&, x["
        TransformList"], List[##], Dot])]
13   ];
```

Homogeneous[H$_1$, ..., H$_r$][$\xi_1, \ldots, \xi_r$] gives the homogeneous transformation matrix associated to sucessive rotations or translations H$_1$, ..., H$_r$ (being $\xi_1, \ldots, \xi_r$ the corresponding rotation angles or displacements). In this syntax, H$_k$ can be defined either a rotation "R"[$\mathbf{e}_k$] around an axis defined by $\mathbf{e}_k$ or a translation "T"[$\mathbf{e}_k$] in the direction of $\mathbf{e}_k$ (being $\mathbf{e}_k$ a List element representing the three Cartesian coordinates of a vector in the local basis coordinates). When the rotation is around a canonical local axis, the following syntaxes are allowed for the H$_k$: "Rx", "Ry" or "Rz". Analogously, when a translation is in the directions of a canonical local axis, the following syntaxes are allowed for the H$_k$: "Tx", "Ty" or "Tz".

## 4.3 Angular velocity

```
1  SkewToVec = If[ And @@ (Flatten @ PossibleZeroQ[# + Transpose[#]]),
       {#[[3,2]], #[[1,3]], #[[2,1]]}]&;
2  VecToSkew = {{0, -#[[3]], #[[2]]}, {#[[3]], 0, -#[[1]]}, {-#[[2]],
       #[[1]], 0}}&;
3
4  AngularVelocity[xRotationMatrix_List] /; (Dimensions[xRotationMatrix
       ]==={3,3}) :=
```

```
5   Simplify @ (SkewToVec @ ((Transpose @ xRotationMatrix).D[
        xRotationMatrix, t]))
```

`SkewToVec` converts any $3 \times 3$ skew-symmetric `List` element representing a matrix into a $3$ entries `List`. `VecToSkew` is its corresponding inverse function.

`AngularVelocity` obtains the angular velocity, in terms of local basis components ($3$ entries `List`), given the corresponding $3 \times 3$ `List` element representing a rotation transformation.

# 5 Plotting and visualization

## 5.1 General options

```
1  SetOptions[Plot, BaseStyle -> {FontFamily -> "Arial", FontSize -> 16}];
2  SetOptions[Plot3D, BaseStyle -> {FontFamily -> "Arial", FontSize ->
       14}];
3  SetOptions[ParametricPlot, BaseStyle -> {FontFamily -> "Arial", FontSize
        -> 16}];
4  SetOptions[ParametricPlot3D, BaseStyle -> {FontFamily -> "Arial",
       FontSize -> 14}];
5  SetOptions[ListPlot, BaseStyle -> {FontFamily -> "Arial", FontSize ->
       16}];
```

Package MathMatrixPack sets the `FontFamily` and `FontSize` for the following built-in plot functions:

- `Plot`: Arial, 16

- `Plot3D`: Arial, 14

- `ParametricPlot`: Arial, 16

- `ParametricPlot3D`: Arial, 14

- `ListPlot`: Arial, 16

## 5.2 Custom plot

```
1  Style8 = {
2    {Hue[0.6, 1, 1], Thickness[0.005]},
3    {Hue[0.3, 1, 1], Thickness[0.006], Dashed},
4    {Hue[1, 1, 1], Thickness[0.007], Dotted},
5    {Hue[0.1, 1, 1], Thickness[0.005]},
6    {Hue[0.9, 1, 1], Thickness[0.006], Dashed},
7    {Hue[0.5, 1, 1], Thickness[0.007], Dotted},
8    {Hue[0.2, 1, 1], Thickness[0.005]},
9    {Hue[0.8, 1, 1], Thickness[0.006], Dashed}
10   };
11
12 SPlot[xExpression_, xInterval_, xFrameLabel_ : {}, xLegend_ : {},
        xPlotLabel_String : "", xScale_ : 1.15] :=
13   Module[{xStyle = Style8},
14     TableForm[ {
15       Plot[
16         xExpression,
17         xInterval,
18         PlotStyle -> Style8,
19         PlotRange -> Full,
20         Frame -> True,
21         GridLines -> Automatic,
22         ImageSize -> xScale {500, 300},
23         FrameLabel -> xFrameLabel,
24         PlotLabel -> xPlotLabel
25         ],
26       Graphics[ {
27         Black,
28         Directive[FontFamily -> "Arial", FontSize -> 16],
29         MapIndexed[ Text[#1, {10 (First[#2] - 1) + 6, 0}] &, xLegend],
30         MapIndexed[ Join[
31           Last[xStyle = RotateLeft @ xStyle],
32           {Line[{{10 (First[#2] - 1), 0}, {10 (First[#2] - 1) + 3, 0}}]}]
33           ] &,
```

40

```
34        xLegend
35        ]
36      },
37     ImageSize -> 1.15 {500, 30}]]}
38     ]
39   ];
```

`SPlot` is a customized version of the built-in function `Plot` for showing in the same frame up to 8 plots with their respective legends. The corresponding list of styles used in this plot are set in the `List` element `Style8`. `SPlot` syntax requires 5 arguments:

- The first argument must be a `List` of functions to be plot.

- The second argument must be a `List` of three elements: the first is the symbol denoting the independent variable, and the second and the third defining the range of this variable.

- The third argument is a `List` of 2 `String` elements representing representing the labels of the axes.

- The fourth argument is the title of the plot.

- The fifth argument is a `List` of legend labels.

For example, consider the following usage of the function:

```
1   SPlot[Sin[# t] & /@ #, {t, 0, Pi/2}, {"t", "Sin(nt)"},
2     "Sin(nt)␣for␣several␣values␣of␣n", #] & @ Range[8]
```
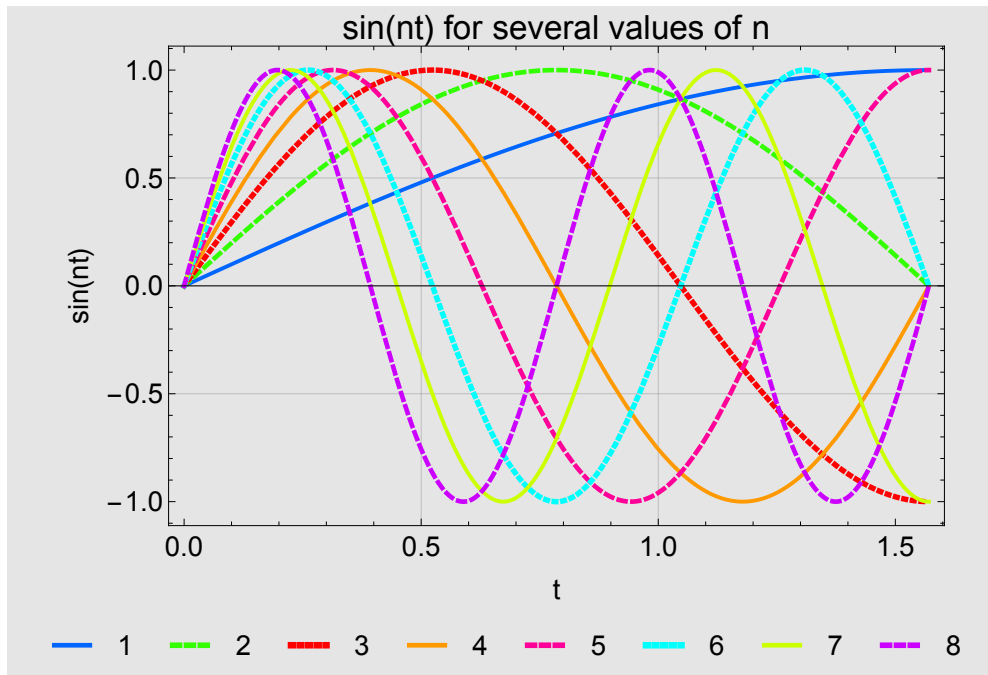
The corresponding output is presented in Figure 1.

Figure 1: Example of output of the function `SPlot`

## 5.3 SMatrixPlot

```
1  SMatrixPlot[xA_Association] /; KeyExistsQ[xA, "Column␣Labels"] :=
     MatrixPlot[
2    xA["Matrix"],
3    FrameTicks -> {
4      Transpose[{Range @@ Dimensions @ xA["Row␣Labels"], xA["Row␣Labels"
         ]}],
5      Transpose[{Range @@ Dimensions @ xA["Column␣Labels"], xA["Column␣
         Labels"]}]
6      } /. SymbolReplacements,
7    ColorFunction -> (RGBColor[
8      (0.00130 (1 - #) + 0.99985 #) (2 # - 1)^2,
9      (0.35656 (1 - #) + 0.085864 #) (2 # - 1)^2,
10     (0.56796 (1 - #)) (2 # - 1)^2,
11     0.13 + (2 # - 1)^2
12     ] &),
```

```
13  ColorRules -> {xN_ /; Not @ NumberQ[xN] -> RGBColor[0.63521, 0.99995,
         0.19208]}
14  ]
15
16 SMatrixPlot[xA_Association] := MatrixPlot[
17   Transpose @ {xA["Matrix"]},
18   FrameTicks -> {
19     Transpose[{Range @@ Dimensions @ xA["Row␣Labels"], xA["Row␣Labels"
          ]}],
20     Transpose[{Range @@ Dimensions @ {""}, {""}}]
21     } /. SymbolReplacements,
22   ColorFunction -> (RGBColor[
23     (0.00130 (1 - #) + 0.99985 #) (2 # - 1)^2,
24     (0.35656 (1 - #) + 0.085864 #) (2 # - 1)^2,
25     (0.56796 (1 - #)) (2 # - 1)^2,
26     0.13 + (2 # - 1)^2
27     ] &),
28   ColorRules -> {xN_ /; Not @ NumberQ[xN] -> RGBColor[0.63521, 0.99995,
         0.19208]}
29   ]
```

SMatrixPlot extends the application of the built-in functions MatrixPlot to matrices given by Association elements.

# Index