MoSs package documentation

Wolfram Mathematica® 10.0 package for modular modelling of multibody systems

Renato Maia Matarazzo Orsino

September 17, 2015

1 Introduction

MoSs, acronym for **Mo**dular **Mo**delling of Multibody Systems Based on **S**ub**s**ystems Models, is a Mathematica Package developed by Renato Maia Matarazzo Orsino based on the modular modeling methodology for multibody systems presented in [1].

The package, developed in Wolfram Mathematica 10.0, aids in the implementation of a modular modelling algorithm in which, the user only needs to provide the mathematical models of subsystems of a multibody system (i.e., systems of differential-algebraic equations of motion of the subsystems when there are no constraints among them) and some description of the constraints among these subsystems (i.e., holonomic or non-holonomic constraint equations) to obtain the equations of motion of the whole system (satsifying all the existing physical constraints).

Consider a mechanical system $\mathscr{M}$ consisting of a finite set of constrained subsystems generally denoted by $\mathscr{S}_n$.

Define $\boldsymbol{q}_n^{\langle 0\rangle}$ as the column-matrix of 0-th order generalized variables of $\mathscr{S}_n$ (which also can be called generalized coordinates of $\mathscr{S}_n$); $\boldsymbol{q}_n^{\langle 0\rangle}$ represents a set of variables that is enought to parametrize the description of every configuration of this subsystem. That is, all positions and orientations of $\mathscr{S}_n$ when it is not constrained to any other subsystem, can be described as functions of $\boldsymbol{q}_n^{\langle 0\rangle}$ and of geometrical of this subsystem. Analogously, define $\boldsymbol{q}_n^{\langle 1\rangle}$ as the column-matrix of 1-st order generalized variables of $\mathscr{S}_n$ (which also can be called quasi-velocities of $\mathscr{S}_n$); $\boldsymbol{q}_n^{\langle 1\rangle}$ represents a set of variables that is enough to parametrize, along with $\boldsymbol{q}_n^{\langle 0\rangle}$, the description of any state of $\mathscr{S}_n$. All components velocities, angular velocities, linear and angular momenta of $\mathscr{S}_n$ as well as an expression for the kinetic energy of this subsystem when it is not constrained to any other subsystem can be described as functions of $\boldsymbol{q}^{\langle 0\rangle}$ and $\boldsymbol{q}^{\langle 1\rangle}$. Actually, $\boldsymbol{q}^{\langle 1\rangle}$ can be interpreted as a set of variables that replace $\dot{\boldsymbol{q}}_n^{\langle 0\rangle}$ in the description of any state of $\mathscr{S}_n$.

Generally, α-th order generalized variables ($\boldsymbol{q}_n^{\langle\alpha\rangle}$) can be similarly defined as being a set of variables that replace the time derivatives of $(\alpha-1)$-th order generalized variables ($\dot{\boldsymbol{q}}_n^{\langle\alpha-1\rangle}$) in the parametric description of some motion variable. Define also $\boldsymbol{q}_n^{\langle\!\langle\alpha\rangle\!\rangle}$ as the column-matrix constituted by all generalized variables of $\mathscr{S}_n$ up to α-th order ($\boldsymbol{q}_n^{\langle 0\rangle},\dots,\boldsymbol{q}_n^{\langle\alpha\rangle}$).

Define also the column-matrix $\boldsymbol{u}_n$ consisting of some control inputs or external disturbances that influence on the components of active forces and torques of $\mathscr{S}_n$. Consider that the mathematical model of $\mathscr{S}_n$ is already known and given by the following system of equations:

$$\begin{cases} \dot{\boldsymbol{q}}_n^{\langle\kappa\rangle} = \underline{\dot{\boldsymbol{q}}}_n^{\langle\kappa\rangle}\big(t,\boldsymbol{q}_n^{\langle\!\langle\kappa+1\rangle\!\rangle}\big) \qquad \text{for} \qquad 0\le\kappa\le\sigma-1 \\ \bar{\boldsymbol{q}}_n^{\langle\sigma\rangle} = \tilde{\boldsymbol{A}}_n\big(t,\boldsymbol{q}_n^{\langle\!\langle\sigma-1\rangle\!\rangle}\big)\,\boldsymbol{q}_n^{\langle\sigma\rangle} + \tilde{\boldsymbol{b}}_n^{\langle\sigma-1\rangle}\big(t,\boldsymbol{q}_n^{\langle\!\langle\sigma-1\rangle\!\rangle}\big) = \boldsymbol{0} \\ \bar{\boldsymbol{d}}_n^{\langle\sigma\rangle}(t,\boldsymbol{q}_n^{\langle\!\langle\sigma\rangle\!\rangle},\boldsymbol{u}_n) = \boldsymbol{0} \end{cases} \tag{1}$$

Define $\boldsymbol{q}^{\langle\alpha\rangle}$ and $\boldsymbol{q}^{\langle\!\langle\alpha\rangle\!\rangle}$ as the block-column-matrices constituted respectively by the $\boldsymbol{q}_n^{\langle\alpha\rangle}$ and $\boldsymbol{q}_n^{\langle\!\langle\alpha\rangle\!\rangle}$ of all the subsystems $\mathscr{S}_n$. Suppose that all the constraints among the subsystems can be described by equations of the form:

$$\bar{\bar{\boldsymbol{q}}}^{\langle\sigma\rangle} = \sum_n \tilde{\tilde{\boldsymbol{A}}}_n(t,\boldsymbol{q}^{\langle\!\langle\sigma-1\rangle\!\rangle})\,\boldsymbol{q}_n^{\langle\sigma\rangle} + \tilde{\tilde{\boldsymbol{b}}}^{\langle\sigma-1\rangle}(t,\boldsymbol{q}^{\langle\!\langle\sigma-1\rangle\!\rangle}) = \boldsymbol{0} \tag{2}$$

Suppose without loss of generality that the subsystems $\mathscr{S}_n$ of $\mathscr{M}$ are indexed by consecutive positive integers, i.e., $n\in\{1,2,\dots,\nu_{\mathscr{S}}\}$. In this case the jacobian of the constraint equations that most be satisfied in order to a motion be compatible with both internal constraints of the subsystems and external constraints among subsystems is given by:

$$\boldsymbol{A} = \begin{bmatrix} \tilde{\boldsymbol{A}}_1 & \dots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \dots & \tilde{\boldsymbol{A}}_{\nu_{\mathscr{S}}} \\ \tilde{\tilde{\boldsymbol{A}}}_1 & \dots & \tilde{\tilde{\boldsymbol{A}}}_{\nu_{\mathscr{S}}} \end{bmatrix}$$

Let $\tilde{\boldsymbol{C}}_n$ denote an orthogonal complement of $\tilde{\boldsymbol{A}}_n$. Depending on the methodology used to derive the mathematical model of $\mathscr{S}_n$, some expression for $\tilde{\boldsymbol{C}}_n$ may already be known. Define the matrix $\tilde{\tilde{\boldsymbol{A}}}$ by the expression:

$$\tilde{\tilde{\boldsymbol{A}}} = \left[\begin{array}{cccc} \tilde{\tilde{\boldsymbol{A}}}_1 \tilde{\boldsymbol{C}}_1 & \tilde{\tilde{\boldsymbol{A}}}_2 \tilde{\boldsymbol{C}}_2 & \dots & \tilde{\tilde{\boldsymbol{A}}}_{\nu_\mathscr{S}} \tilde{\boldsymbol{C}}_{\nu_\mathscr{S}} \end{array}\right]$$

Define $\tilde{\boldsymbol{d}}^{\langle\sigma\rangle}$ as the block-column-matrix constituted by the $\tilde{\boldsymbol{d}}_n^{\langle\sigma\rangle}$ of all the subsystems $\mathscr{S}_n$ and let $\tilde{\tilde{\boldsymbol{C}}}$ be an orthogonal complement of $\tilde{\tilde{\boldsymbol{A}}}$. It can be stated that, the equations of motion of system $\mathscr{M}$, compatible with all its physical constraints, are given by [1]:

$$\begin{cases} \dot{\boldsymbol{q}}_n^{\langle\kappa\rangle} = \underline{\dot{\boldsymbol{q}}}_n^{\langle\kappa\rangle}\left(t, \boldsymbol{q}_n^{\langle\!\langle\kappa+1\rangle\!\rangle}\right), \text{ for } 0 \leq \kappa \leq \sigma - 1, \forall\, n \\ \bar{\boldsymbol{q}}_n^{\langle\sigma\rangle} = \tilde{\boldsymbol{A}}_n\, \boldsymbol{q}_n^{\langle\sigma\rangle} + \tilde{\boldsymbol{b}}_n^{\langle\sigma-1\rangle} = \boldsymbol{0}, \forall\, n \\ \bar{\bar{\boldsymbol{q}}}^{\langle\sigma\rangle} = \sum_n \tilde{\tilde{\boldsymbol{A}}}_n\, \boldsymbol{q}_n^{\langle\sigma\rangle} + \tilde{\tilde{\boldsymbol{b}}}^{\langle\sigma-1\rangle} = \boldsymbol{0} \\ \bar{\bar{\boldsymbol{d}}}^{\langle\sigma\rangle} = \tilde{\tilde{\boldsymbol{C}}}^{\mathsf{T}}\, \bar{\boldsymbol{d}}^{\langle\sigma\rangle} = \boldsymbol{0} \end{cases} \tag{3}$$

Package MoSs consists of functions developed in Wolfram Mathematica 10.0 that enable the implementation of the algorithm for obtaining the system of equations (3) from already known expressions for (1) and (2).

2 Auxiliary functions and configurations

This section presents the functions and configurations of MoSs package that aid in the implementation of the algorithms involved in the modular modelling of multibody systems as well as in the settings of the main functions.

2.1 Formatting rules

This subsection presents the following formatting rules in the kernel of Mathematica when Package MoSs is used.

2.1.1 Trigonometric functions

```
$PrePrint = # /. {
   Csc[◇Argument_] :> 1/Defer @ Sin[◇Argument],
   Sec[◇Argument_] :> 1/Defer @ Cos[◇Argument],
   Tan[◇Argument_] :> Defer @ Sin[◇Argument]/Defer @ Cos[◇Argument],
   Cot[◇Argument_] :> Defer @ Cos[◇Argument]/Defer @ Sin[◇Argument]
   } &;

Unprotect[Cos, Sin];
Format[Cos[◇Argument_]] := Subscript[c, ◇Argument]
Format[Sin[◇Argument_]] := Subscript[s, ◇Argument]
Protect[Cos, Sin];
```

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

2.1.2 Derivatives

```
Format[
  Subscript[Subscript[◇Argument_, ◇Indexes1__], ◇Indexes2__]'[t_]] :=
  Subscript[Subscript[Overscript[◇Argument, "."], ◇Indexes1],
    ◇Indexes2][t]
Format[
  Subscript[Subscript[◇Argument_, ◇Indexes1__], ◇Indexes2__]''[t_]] :=
  Subscript[Subscript[Overscript[◇Argument, ".."], ◇Indexes1],
    ◇Indexes2][t]
Format[Subscript[◇Argument_, ◇Indexes1__]'[t_]] :=
  Subscript[Overscript[◇Argument, "."], ◇Indexes1][t]
Format[Subscript[◇Argument_, ◇Indexes1__]''[t_]] :=
  Subscript[Overscript[◇Argument, ".."], ◇Indexes1][t]
Format[◇Argument_'[t_]] := Overscript[◇Argument, "."][t]
Format[◇Argument_''[t_]] := Overscript[◇Argument, ".."][t]
```

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 = {
    Subscript[Subscript[◇Base_, ◇Indexes__], ◇Indexes2__]'[t] ->
      Subscript[Subscript[Overscript[◇Base, "."], ◇Indexes],
        ◇Indexes2],
    Subscript[Subscript[◇Base_, ◇Indexes__], ◇Indexes2__]''[t] ->
      Subscript[Subscript[Overscript[◇Base, ".."], ◇Indexes],
        ◇Indexes2],
    Subscript[◇Base_, ◇Indexes__]'[t] ->
      Subscript[Overscript[◇Base, "."], ◇Indexes],
    Subscript[◇Base_, ◇Indexes__]''[t] ->
      Subscript[Overscript[◇Base, ".."], ◇Indexes],
    ◇Variable_'[t] -> Overscript[◇Variable, "."],
    ◇Variable_''[t] -> Overscript[◇Variable, ".."],
    ◇Variable_[t] -> ◇Variable
  };
```

`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 $\ddot{\zeta}$.

2.1.3 Round-off rules

```
RoundOffRules = {◇Number_?NumericQ /; Abs[◇Number] < 10^-12 -> 0,
  ◇Number_?NumericQ /; Abs[◇Number - 1] < 10^-12 -> 1};
```

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.

2.2 General purpose functions

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

2.2.1 Set complement

```
SetComplement = {◇MainSet, ◇DiffSet} \[Function]
  Select[◇MainSet, Not[MemberQ[◇DiffSet, #]] &];
```

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

2.2.2 Delete redundant expressions

```
RedundantElim = DeleteDuplicates @ (DeleteCases[Simplify @ #, 0]) &;
```

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

2.2.3 Simplify Associations

```
SSimplify[◇A_Association] :=
  Association @ MapThread[#1 -> #2 &, {First /@ (Normal @ ◇A),
    Simplify @ (Last /@ (Normal @ ◇A))}, 1]
SSimplify[◇X_] := Simplify[◇X]
```

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

2.2.4 Replacements in Associations

```
SReplaceRepeated[◇A_Association, ◇L_List] :=
  Association @ MapThread[#1 -> #2 &, {First /@ (Normal @ ◇A),
    ReplaceRepeated[(Last /@ (Normal @ ◇A)), ◇L]}, 1]
 SReplaceRepeated[◇X_, ◇L_List] := ReplaceRepeated[◇X, ◇L]
```

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

2.2.5 Replacements and Simplifications in Associations

```
SReplaceFullSimplify[◇A_Asssociation, ◇Rules_List] :=
  Association@MapThread[#1 -> #2 &, {
    First /@ (Normal @ ◇A),
    FullSimplify[FullSimplify[
      Expand[(Last /@ (Normal @ ◇A)) //.◇Rules] //.◇Rules] //.◇Rules]
  }, 1]

SReplaceFullSimplify[◇X_, ◇Rules_List] :=
  FullSimplify[FullSimplify[
    Expand[(Flatten @ {◇X}) //.◇Rules] //.◇Rules] //.◇Rules]

SReplaceSimplify[◇A_Asssociation, ◇Rules_List] :=
  Association@MapThread[#1 -> #2 &, {
    First /@ (Normal @ ◇A),
    Simplify[Simplify[
      Expand[(Last /@ (Normal @ ◇A)) //.◇Rules] //.◇Rules] //.◇Rules]
  }, 1]

SReplaceSimplify[◇X_, ◇Rules_List] :=
  Simplify[Simplify[
      Expand[(Flatten @ {◇X}) //.◇Rules] //.◇Rules] //.◇Rules]
```

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).

2.2.6 Rename keys and values in Associations

```
SRename[◇In_Association, ◇NamingRules_, ◇ExtraRules_: {}] :=
  Association@MapThread[
    #1 -> #2 &,
    {If[Head[#] === String,
    StringReplace[#, ◇NamingRules], #] & /@
        (First /@ Normal @ (◇In)),
    Map[SReplaceRepeated[#, ◇ExtraRules] &,
      Map[SReplaceRepeated[#, ◇NamingRules] &,
        Map[SReplaceRepeated[#, ◇ExtraRules] &,
          (Last /@ Normal @ ◇In), All], All], All]}, 1];
```

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

`SRename[◇In, ◇NamingRules, ◇ExtraRules]` also applies replacements according to `◇ExtraRules` to the values of the `Association` element `◇In`.

2.2.7 List variables in expressions

```
GetVariables[◇X_List, ◇Except_List: {}] :=
  Complement[ DeleteDuplicates @ Cases[◇X, ◇Variable_[t], Infinity],
    ◇Except];

GetVariables[◇X_Association, ◇Except_List: {}] :=
  Complement[ DeleteDuplicates @ Cases[◇X["Matrix"], ◇Variable_[t],
    Infinity], ◇Except];
```

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

```
HeadList = {Or, And, Equal, Unequal, Less, LessEqual, Greater,
  GreaterEqual, Inequality};

GetAllVariables[◇Number_?NumericQ] := Sequence[]
GetAllVariables[{}] := Sequence[]
GetAllVariables[◇RelationalOperator_] /; MemberQ[HeadList,
  ◇RelationalOperator] := Sequence[]
GetAllVariables[◇_List] :=
  DeleteDuplicates@(Flatten@(Union@(GetAllVariables[#] & /@ ◇)))

GetAllVariables[
  Derivative[◇Number_Integer][◇Function_][◇Argument__]
  ]:=
  Module[{◇Variable},
   If[MemberQ[Attributes[◇Function], NumericFunction] ||
    MemberQ[HeadList, ◇Function],
    (*-TRUE-*)
    ◇Variable = GetAllVariables[{◇Argument}],
    (*-FALSE-*)
    ◇Variable = Derivative[◇Number][◇Function][◇Argument]
    ];
   ◇Variable
   ];

GetAllVariables[◇Function_Symbol[◇Argument__]] :=
  Module[{◇Variable},
   If[MemberQ[Attributes[◇Function], NumericFunction] ||
    MemberQ[HeadList, ◇Function],
    (*-TRUE-*)
    ◇Variable = GetAllVariables[{◇Argument}],
    (*-FALSE-*)
    ◇Variable = ◇Function[◇Argument]
    ];
   ◇Variable
    ];

GetAllVariables[◇Other_] := ◇Other
```

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

2.3 Matrix calculus

In package MoSs, 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.

2.3.1 Sum, assemble and partitioning of matrices - `AngleBracket` operator

Wolfram Mathematica has some operators without built-in meanings. In MoSs, 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:

```
Matrix2Rule[◇A_Association] :=
  Association @ Flatten @ MapThread[
      (#1 -> #2) &,
      {Outer[{#1, #2} &, ◇A["Row␣Labels"], ◇A["Column␣Labels"]],
        ◇A["Matrix"]},
      2
      ]

AngleBracket[◇A__Association] :=
  Module[{◇AList, ◇RowLabels, ◇ColumnLabels, ◇RList},
      ◇AList = List[◇A];
      ◇ColumnLabels = Union@(Join@@((#["Column␣Labels"])&/@ ◇AList));
      ◇RowLabels = Union@(Join @@ ((#["Row␣Labels"]) & /@ ◇AList));
      ◇RList = Association@((# -> Plus @@ DeleteCases[# /.
        (Matrix2Rule /@ ◇AList), #]) & /@
        Flatten[Outer[{#1, #2} &, ◇RowLabels, ◇ColumnLabels], 1]);
      Association[
        "Matrix" -> Outer[{#1, #2} &, ◇RowLabels, ◇ColumnLabels] /.
```

```
         ◇RList,
       "Row␣Labels" -> ◇RowLabels,
       "Column␣Labels" -> ◇ColumnLabels
       ]
     ]

AngleBracket[◇A_Association, ◇RowLabels_List] :=
  AngleBracket @ Association[
     "Matrix" -> Part[◇A["Matrix"],Flatten@(Position[◇A["Row␣Labels"],
       #] & /@ ◇RowLabels), All],
     "Row␣Labels" -> ◇RowLabels,
     "Column␣Labels" -> ◇A["Column␣Labels"]
     ];

AngleBracket[◇A_Association, ◇RowLabel_] :=
  If[First @ Dimensions@(◇A["Column␣Labels"]) == 1,
     Part[◇A["Matrix"], First@(Flatten@(Position[◇A["Row␣Labels"],
       ◇RowLabel])), 1],
     AngleBracket @ Association[
       "Matrix" -> Part[◇A["Matrix"], Flatten@(Position[
         ◇A["Row␣Labels"], ◇RowLabel]), All],
       "Row␣Labels" -> {◇RowLabel},
       "Column␣Labels" -> ◇A["Column␣Labels"]
       ]
     ];

AngleBracket[◇A_Association, ◇RowLabels_List, ◇ColumnLabels_List] :=
  AngleBracket @ Association[
     "Matrix" -> Part[◇A["Matrix"], Flatten@(Position[
       ◇A["Row␣Labels"], #] & /@ ◇RowLabels), Flatten @ (Position[
       ◇A["Column␣Labels"], #] & /@ ◇ColumnLabels)],
     "Row␣Labels" -> ◇RowLabels,
     "Column␣Labels" -> ◇ColumnLabels
     ];

```

```
AngleBracket[◇A_Association, All, ◇ColumnLabels_List] :=
  AngleBracket @ Association[
      "Matrix" -> Part[◇A["Matrix"], All, Flatten@(Position[
        ◇A["Column␣Labels"], #] & /@ ◇ColumnLabels)],
      "Row␣Labels" -> ◇A["Row␣Labels"],
      "Column␣Labels" -> ◇ColumnLabels
      ];

AngleBracket[◇A_Association, ◇RowLabels_List, ◇ColumnLabel_] :=
  AngleBracket @ Association[
      "Matrix" -> {Part[◇A["Matrix"], Flatten @ (Position[
        ◇A["Row␣Labels"], #] & /@ ◇RowLabels), First @ Flatten @
        (Position[◇A["Column␣Labels"], ◇ColumnLabel])]}\[Transpose],
      "Row␣Labels" -> ◇RowLabels,
      "Column␣Labels" -> {◇ColumnLabel}
      ];

AngleBracket[◇A_Association, All, ◇ColumnLabel_] :=
  AngleBracket @ Association[
      "Matrix" -> {Part[◇A["Matrix"], All, First @ Flatten @
      (Position[◇A["Column␣Labels"], ◇ColumnLabel])]}\[Transpose],
      "Row␣Labels" -> ◇A["Row␣Labels"],
      "Column␣Labels" -> {◇ColumnLabel}
      ];

AngleBracket[◇A_Association, ◇RowLabel_, ◇ColumnLabels_List] :=
  AngleBracket @ Association[
      "Matrix" -> Part[◇A["Matrix"], First @ Flatten@(Position[
        ◇A["Row␣Labels"], ◇RowLabel]), Flatten@(Position[
        ◇A["Column␣Labels"], #] & /@ ◇ColumnLabels)],
      "Row␣Labels" -> {◇RowLabel},
      "Column␣Labels" -> ◇ColumnLabels
      ];

AngleBracket[◇A_Association, ◇RowLabel_, ◇ColumnLabel_] :=
  Part[◇A["Matrix"], First@Flatten@(Position[
      ◇A["Row␣Labels"], ◇RowLabel]), First@Flatten@(Position[
      ◇A["Column␣Labels"], ◇ColumnLabel])];
```

When `AngleBracket` is called with a sequence of matrices (sequence of `Association` elements, ⟨`X,Y,...`⟩), 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.

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 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.

2.3.2 Apply unary functions to matrices

```
SApply[◇Function_, ◇X_Association] :=
  Association[
    "Matrix" -> ◇Function@(◇X["Matrix"]),
    "Column␣Labels" -> ◇X["Column␣Labels"],
    "Row␣Labels" -> ◇X["Row␣Labels"]
    ]
```

`SApply` applied the unary function `◇Function` to the entry whose key is `"Matrix"` in the `Association ◇X`.

2.3.3 Matrix multiplication and multiplication of a matrix by a scalar

```
CircleDot[◇X_Association, ◇Y_Association] :=
  Module[{◇A, ◇B},
    ◇A = AngleBracket @ ◇X;
      ◇B = AngleBracket @ ◇Y;
    If[◇A["Column␣Labels"] === ◇B["Row␣Labels"],
      Association[
        "Matrix" -> (◇A["Matrix"].◇B["Matrix"]),
        "Column␣Labels" -> ◇B["Column␣Labels"],
        "Row␣Labels" -> ◇A["Row␣Labels"]
        ],
      "Error"
      ]
    ]

CircleDot[◇X_Association, ◇Y_List] := (◇X["Matrix"].◇Y)

CircleDot[◇X_Association, ◇Y_] :=
  Association[
    "Matrix" -> ((AngleBracket @ ◇X)["Matrix"]) ◇Y,
    "Column␣Labels" -> (AngleBracket @ ◇X)["Column␣Labels"],
    "Row␣Labels" -> (AngleBracket @ ◇X)["Row␣Labels"]
    ]
CircleDot[◇Y_, ◇X_Association] :=
  Association[
    "Matrix" -> ((AngleBracket @ ◇X)["Matrix"]) ◇Y,
    "Column␣Labels" -> (AngleBracket @ ◇X)["Column␣Labels"],
    "Row␣Labels" -> (AngleBracket @ ◇X)["Row␣Labels"]
    ]
```

In the package MoSs, the operator `CircleDot`, denote by `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.

2.3.4 Matrix transposition

```
SuperDagger[◇X_Association] :=
  Association[
    "Matrix" -> Transpose@(◇X["Matrix"]),
    "Column␣Labels" -> ◇X["Row␣Labels"],
    "Row␣Labels" -> ◇X["Column␣Labels"]
    ]

STranspose[◇X_Association] :=
  SuperDagger[◇X]
```

In the package MoSs, 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.

2.3.5 Affine Transformations

```
BracketingBar[◇X_Association] :=
  AffineTransform[◇X["Matrix"]]

BracketingBar[◇X_List /; Dimensions[◇X] == {3, 3}] :=
  AffineTransform[◇X]

BracketingBar[◇X_List /; Dimensions[◇X] == {4, 4}] :=
  LinearFractionalTransform[◇X]
```

In the package MoSs, the operator BracketingBar, denote by ⊣ X ⊢ 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 3×3 List element. However, when the argument is a 4×4 List element, the corresponding TransformationFunction is obtained by the application of the built-in LinearFractionalTransform function (whose output represents a homogeneous transformation).

2.3.6 Coefficient arrays

```
SCoefficientArrays[◇A_Association, ◇Variables_List, ◇Rules_List: {}]:=
  Module[{◇},
    ◇["Row␣Labels"] = ◇A["Row␣Labels"];
    ◇["Expressions"] = Flatten @ (◇A["Matrix"]);
    ◇["Coefficient␣Arrays"] = CoefficientArrays[◇["Expressions"] //.
      ◇Rules, ◇Variables];
    {
      Association[
        "Matrix" -> {Part[#, 1] & @ (◇["Coefficient␣Arrays"])}
          \[Transpose],
        "Row␣Labels" -> ◇["Row␣Labels"],
        "Column␣Labels" -> {""}
        ],
      Association[
        "Matrix" -> Part[#, 2] & @ (◇["Coefficient␣Arrays"]),
        "Row␣Labels" -> ◇["Row␣Labels"],
        "Column␣Labels" -> ◇Variables
        ]
    }
    ]
```

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`⟩.

```
SMatrixCoefficientArrays[◇A_Association, ◇Rules_List: {}] :=
  Module[{◇◇Matrix, ◇◇Variables, ◇◇RowLabels,
    ◇◇ColumnLabels, ◇◇CoefficientMatrices},
    ◇◇Matrix = ◇A["Matrix"] //. ◇Rules;
    ◇◇RowLabels = ◇A["Row␣Labels"];
    ◇◇ColumnLabels = ◇A["Column␣Labels"];
    ◇◇Variables = Union @ GetVariables[◇◇Matrix];
    ◇◇CoefficientMatrices = CoefficientArrays[◇◇Matrix, ◇◇Variables];
    {
      Association[ Union @@
        {
        {1 -> Association[
          "Matrix" -> Normal @ Part[◇◇CoefficientMatrices, 1],
          "Column␣Labels" -> ◇◇ColumnLabels,
          "Row␣Labels" -> ◇◇RowLabels
          ]},
        MapThread[ (#1 ->
          Association[
            "Matrix" -> Normal @ Part[◇◇CoefficientMatrices, 2,
              All, All, #2],
            "Column␣Labels" -> ◇◇ColumnLabels,
            "Row␣Labels" -> ◇◇RowLabels
            ]) &,
          {#, Range @ Length @ #},
          1
          ] & @ ◇◇Variables
        }
        ],
      ◇◇Variables
    }
    ]
```

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

```
Association[1 -> M_1, v_1 -> M_v_1, ..., 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 $\{v_1, \ldots, v_r\}$.

2.3.7 Linear Solve

```
SLinearSolve[◇X_Association, ◇Y_Association] :=
  Module[{◇A, ◇B},
    ◇A = AngleBracket @ ◇X;
    ◇B = AngleBracket @ ◇Y;
    If[◇A["Row␣Labels"] === ◇B["Row␣Labels"],
      Association[
        "Matrix" -> LinearSolve[◇A["Matrix"], ◇B["Matrix"]],
        "Column␣Labels" -> ◇B["Column␣Labels"],
        "Row␣Labels" -> ◇A["Column␣Labels"]
        ],
      "Error"
      ]
    ]
```

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`.

```
LSOCSolver[◇Jacobian_Association, ◇Remainder_Association] :=
  Association[
    "Matrix" -> - LeastSquares[◇Jacobian["Matrix"],
      ◇Remainder["Matrix"]],
    "Row␣Labels" -> ◇Jacobian["Column␣Labels"],
    "Column␣Labels" -> ◇Remainder["Column␣Labels"]
    ]
```

`LSOCSolver` 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 `SLinearSolve[A, B]` is an `Association` element `Z` which is a least squares solution for `X` in the matrix equation `⟨A⊙X, B⟩ == 0`.

2.3.8 Jacobians

```
Jacobi[◇ExpressionsList_, ◇VariablesList_] :=
  Module[{◇Jacobian},
    ◇Jacobian = Association[
      "Matrix" -> D[◇ExpressionsList, {◇VariablesList}],
      "Column␣Labels" -> ◇VariablesList,
      "Row␣Labels" -> Range @@ Dimensions@◇ExpressionsList
      ]
    ]

Jacobi[◇ExpressionsList_, ◇VariablesList_, ◇ExpressionsLabels_] :=
  Module[{◇Jacobian},
    ◇Jacobian = Association[
      "Matrix" -> D[◇ExpressionsList, {◇VariablesList}],
      "Column␣Labels" -> ◇VariablesList,
      "Row␣Labels" -> ◇ExpressionsLabels
      ]
    ]
```

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.

2.3.9 Orthogonal complement

```
OrthogonalComplement[◇Jacobian_] :=
  Module[{◇, ◇OrthogonalComplement},
    ◇["Null␣Space␣Matrix"] =
      Transpose @ NullSpace[◇Jacobian["Matrix"]];
    ◇["Independent␣Variations"] =
      (Range @ (Dimensions[◇["Null␣Space␣Matrix"]])[[2]]);
    ◇OrthogonalComplement = Association[
      "Matrix" -> ◇["Null␣Space␣Matrix"],
      "Column␣Labels" -> ◇["Independent␣Variations"],
      "Row␣Labels" -> ◇Jacobian["Column␣Labels"]
      ]
    ]

OrthogonalComplement[◇Jacobian_, ◇IndependentVariablesList_List] :=
  Module[{◇, ◇OrthogonalComplement},
    {◇["Number␣of␣Constraints"], ◇["Number␣of␣Variables"]} =
      Dimensions[◇Jacobian["Matrix"]];
    ◇["Number␣of␣Degrees␣of␣Freedom"] =
      ◇["Number␣of␣Variables"] - ◇["Number␣of␣Constraints"];
    If[{◇["Number␣of␣Degrees␣of␣Freedom"]} ==
      Dimensions @ ◇IndependentVariablesList,
      (*-TRUE-*)
      ◇["Independent␣Variables␣Column␣Indexes"] =
        Flatten[Position[◇Jacobian["Column␣Labels"], #] & /@
          ◇IndependentVariablesList, Infinity];
      ◇["Redundant␣Variables␣Column␣Indexes"] =
        Complement[Range @@ Dimensions@◇Jacobian["Column␣Labels"],
          ◇["Independent␣Variables␣Column␣Indexes"]];
      ◇OrthogonalComplement = Association[
        "Matrix" -> Array[0 &, {◇["Number␣of␣Variables"],
          ◇["Number␣of␣Degrees␣of␣Freedom"]}],
        "Column␣Labels" -> ◇IndependentVariablesList,
        "Row␣Labels" -> ◇Jacobian["Column␣Labels"]
        ];
      ◇OrthogonalComplement[["Matrix",
        ◇["Independent␣Variables␣Column␣Indexes"]]] =
        IdentityMatrix @ ◇["Number␣of␣Degrees␣of␣Freedom"];
      ◇OrthogonalComplement[["Matrix",
        ◇["Redundant␣Variables␣Column␣Indexes"]]] =
        LinearSolve @@ {◇Jacobian[["Matrix", All,
          ◇["Redundant␣Variables␣Column␣Indexes"]]],
          -◇Jacobian[["Matrix", All,
            ◇["Independent␣Variables␣Column␣Indexes"]]]};
      ◇OrthogonalComplement,
      (*-FALSE-*)
      "Error"
      ]
    ]
```

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.

```
LSOrthogonalComplement[◇Jacobian_, ◇IndependentVariablesList_List] :=
  Module[{◇, ◇OrthogonalComplement},
    {◇["Number␣of␣Constraints"], ◇["Number␣of␣Variables"]} =
      Dimensions[◇Jacobian["Matrix"]];
    ◇["Number␣of␣Degrees␣of␣Freedom"] =
      Part[Dimensions @ ◇IndependentVariablesList, 1];
    ◇["Independent␣Variables␣Column␣Indexes"] =
      Flatten[Position[◇Jacobian["Column␣Labels"], #] & /@
        ◇IndependentVariablesList, Infinity];
    ◇["Redundant␣Variables␣Column␣Indexes"] =
      Complement[Range @@ Dimensions @ ◇Jacobian["Column␣Labels"],
        ◇["Independent␣Variables␣Column␣Indexes"]];
    ◇OrthogonalComplement = Association[
      "Matrix" -> Array[0 &, {◇["Number␣of␣Variables"],
        ◇["Number␣of␣Degrees␣of␣Freedom"]}],
      "Column␣Labels" -> ◇IndependentVariablesList,
      "Row␣Labels" -> ◇Jacobian["Column␣Labels"]
      ];
    ◇OrthogonalComplement[["Matrix",
      ◇["Independent␣Variables␣Column␣Indexes"]]] =
      IdentityMatrix @ ◇["Number␣of␣Degrees␣of␣Freedom"];
    ◇OrthogonalComplement[["Matrix",
      ◇["Redundant␣Variables␣Column␣Indexes"]]] =
      LeastSquares @@ {◇Jacobian[["Matrix", All,
        ◇["Redundant␣Variables␣Column␣Indexes"]]],
        -◇Jacobian[["Matrix", All,
          ◇["Independent␣Variables␣Column␣Indexes"]]]};
    ◇OrthogonalComplement
    ]
```

LSOrthogonalComplement uses least squares algorithms to obtain an exact or approximate orthogonal complement of a (Jacobian) matrix. LSOrthogonalComplement[A, V] is similar to OrthogonalComplement[A, V] apart from the fact that the built-in function

`LeastSquares` is used instead of `LinearSolve`.

```
LSLinearizedOrthogonalComplement[◇Jacobian_Association,
  ◇IndependentVariables_List, ◇ReferenceMotion_List: {},
  ◇CoordinatesReplacements_: {}, ◇NZero_Rational: 1 10^-5,
  ◇TestParameters_List: {}] :=
  Module[{◇, ◇LSOC, ◇LinearizedJacobian, ◇Coordinates,
    ◇LinearizedJacobianCoefficients, ◇NTestParameters, ◇NA1, ◇NC1,
    ◇NCq, ◇SC1, ◇SCq},

    ◇LinearizedJacobian = SSimplify @ (SReplaceRepeated[
      Linearize[◇Jacobian, ◇ReferenceMotion],
      ◇CoordinatesReplacements]);
    {◇LinearizedJacobianCoefficients, ◇Coordinates} =
      SMatrixCoefficientArrays[◇LinearizedJacobian];

    ◇NTestParameters = Union[
      ◇TestParameters,
      (# -> RandomReal[1]) & /@ (GetAllVariables[(Flatten @ (Union @@
        (Normal @ (#["Matrix"]) & /@ ◇LinearizedJacobianCoefficients))
        //. ◇TestParameters])];

    ◇NA1 = SReplaceRepeated[◇LinearizedJacobianCoefficients[1],
      ◇NTestParameters];
    ◇NC1 = LSOrthogonalComplement[◇NA1, ◇IndependentVariables];
    ◇["ϵ"] = 1 10^-3;

    ◇NCq = Association[ (# -> ⟨
      (+1/(2 ◇["ϵ"]))⊙⟨LSOrthogonalComplement[⟨◇NA1, (+◇["ϵ"])⊙
        SReplaceRepeated[◇LinearizedJacobianCoefficients[#],
          ◇NTestParameters]⟩, ◇IndependentVariables], (-1)⊙◇NC1⟩,
      (-1/(2 ◇["ϵ"]))⊙⟨LSOrthogonalComplement[⟨◇NA1, (-◇["ϵ"])⊙
        SReplaceRepeated[◇LinearizedJacobianCoefficients[#],
          ◇NTestParameters]⟩, ◇IndependentVariables], (-1)⊙◇NC1⟩
    ⟩) & /@ ◇Coordinates];

```

```
◇NC1 = AppendTo[◇NC1, "Matrix" -> Round[◇NC1["Matrix"], ◇NZero]];
(◇NCq[#] = AppendTo[◇NCq[#],
  "Matrix" -> Round[◇NCq[#]["Matrix"], ◇NZero]]) & /@ ◇Coordinates;

◇["Column␣Labels"] = ◇NC1["Column␣Labels"] //. SymbolReplacements;
◇["Row␣Labels"] = ◇NC1["Row␣Labels"] //. SymbolReplacements;
  ◇["New␣Parameters"] = {};

Function[◇RowLabel,
  ◇["Row␣Number"] = First @ (Flatten @ Position[◇["Row␣Labels"],
    ◇RowLabel]);
  ◇["Parameters␣Values"] = Flatten@(Join[
      Part[◇NC1["Matrix"], ◇["Row␣Number"]],
        Join @@ ((Part[◇NCq[#]["Matrix"], ◇["Row␣Number"]]) & /@
          ◇Coordinates)]);
  ◇["Parameters␣Names"] = Flatten @ (Join[
    ((Function[{◇ColumnLabel},
      Subscript[Δ̄, 1, ◇RowLabel, ◇ColumnLabel]]) /@
      ◇["Column␣Labels"]),
    Join @@ (((Function[{◇ColumnLabel},
      Subscript[Δ̄, #, ◇RowLabel, ◇ColumnLabel]]) /@
      ◇["Column␣Labels"]) & /@
        (◇Coordinates //. SymbolReplacements))
    ]);

◇["New␣Parameters:1"] = MapThread[(#2 -> #1) &,
  {◇["Parameters␣Values"], ◇["Parameters␣Names"]}, 1];
◇["New␣Parameters:2"] = (Flatten @ (Normal @ DeleteCases[
  Association[◇["New␣Parameters:1"]], _Integer]));
◇NTestParameters = Union[
  ◇NTestParameters,
  N[◇["New␣Parameters:2"]]
  ];
◇["New␣Parameters"] = Union[
  ◇["New␣Parameters"],
  (Reverse /@ ◇["New␣Parameters:2"]),
  (Reverse /@ ◇["New␣Parameters:2"]) /.
    ((◇A_ -> ◇B_) -> (-◇A -> -◇B))];
  ] /@ ◇["Row␣Labels"];

◇SC1 = Association[◇NC1,
  "Matrix" -> (◇NC1["Matrix"] //. ◇["New␣Parameters"])];
  (◇SCq[#] = Association[◇NCq[#],
    "Matrix" -> (◇NCq[#]["Matrix"] //. ◇["New␣Parameters"])]) & /@
      ◇Coordinates;
◇LSOC = ⟨◇SC1, Inner[#1⊙#2 &, ◇Coordinates,
  ◇SCq /@ ◇Coordinates, SPart]⟩;

{◇LinearizedJacobian, ◇LSOC, ◇NTestParameters}
];
```

`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.

2.4 Rotation and homogeneous transformations

2.4.1 Rotation transformation

```
Rotation = Function @ Module[{◇},
  ◇["AxesList"] = List[##] /.
    {"x" -> {1, 0, 0}, "y" -> {0, 1, 0}, "z" -> {0, 0, 1}};
  Function[(TransformationMatrix @ Simplify @
    (Dot @@ (ComplexExpand[
      MapThread[RotationTransform, {List[##], ◇["AxesList"]}],
      TargetFunctions -> {Re, Im}])))[[1 ;; 3, 1 ;; 3]]
  ];
```

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.

2.4.2 Homogeneous transformation

```
Homogeneous = Function @ Module[{◇},
  ◇["TransformList"] = List[##] /. {
    "Rx" -> (RotationTransform[#, {1, 0, 0}] &),
    "Ry" -> (RotationTransform[#, {0, 1, 0}] &),
    "Rz" -> (RotationTransform[#, {0, 0, 1}] &),
    "R"[◇Vector_] -> (RotationTransform[#, ◇Vector] &),
    "Tx" -> (TranslationTransform[# {1, 0, 0}] &),
    "Ty" -> (TranslationTransform[# {0, 1, 0}] &),
    "Tz" -> (TranslationTransform[# {0, 0, 1}] &),
    "T"[◇Vector_] -> (TranslationTransform[# ◇Vector] &)
    };
  Function[TransformationMatrix @ (Simplify @
    Inner[(#1 @ #2) &, ◇["TransformList"], List[##], Dot])]
  ];
```

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".

2.4.3 Angular velocity

```
SkewToVec = If[And @@ (Flatten @ PossibleZeroQ[# + Transpose[#]]),
  {#[[3, 2]], #[[1, 3]], #[[2, 1]]}] &;
VecToSkew = {{0, -#[[3]], #[[2]]}, {#[[3]], 0, -#[[1]]},
  {-#[[2]], #[[1]], 0}} &;

AngularVelocity[◇RotationMatrix_List /;
  Dimensions[◇RotationMatrix] == {3, 3}] :=
  Simplify @ (SkewToVec @ ((Transpose @ ◇RotationMatrix).
    D[◇RotationMatrix, t]))
```

SkewToVec converts any 3×3 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 3×3 List element representing a rotation transformation.

2.5 Plotting and visualization

2.5.1 General options

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

Package MoSs 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

2.5.2 Custom plot

```
Style8 = {
  {Hue[0.6, 1, 1], Thickness[0.005]},
  {Hue[0.3, 1, 1], Thickness[0.006], Dashed},
  {Hue[1, 1, 1], Thickness[0.007], Dotted},
  {Hue[0.1, 1, 1], Thickness[0.005]},
  {Hue[0.9, 1, 1], Thickness[0.006], Dashed},
  {Hue[0.5, 1, 1], Thickness[0.007], Dotted},
  {Hue[0.2, 1, 1], Thickness[0.005]},
  {Hue[0.8, 1, 1], Thickness[0.006], Dashed}
  };

SPlot = Module[{st = Style8},
  TableForm[{
    Plot[#1, #2, PlotStyle -> Style8, PlotRange -> Full,
      Frame -> True, FrameLabel -> #3, PlotLabel -> #4,
      GridLines -> Automatic, ImageSize -> 1.15 {500, 300}],
    Graphics[
      {Black, Directive[FontFamily -> "Arial", FontSize -> 16],
      MapIndexed[Text[#1, {10 (First[#2] - 1) + 6, 0}] &, #5],
      MapIndexed[Join[Last[st = RotateLeft @ st],
        {Line[{{10 (First[#2] - 1), 0}, {10 (First[#2] - 1) + 3, 0}}]}]
          &, #5]},
    ImageSize -> 1.15 {500, 30}]
    }]
  ] &;
```

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:

```
SPlot[Sin[# t] & /@ #, {t, 0, Pi/2}, {"t", "Sin(nt)"},
  "Sin(nt)␣for␣several␣values␣of␣n", #] & @ Range[8]
```

The corresponding output is presented in Figure 1.

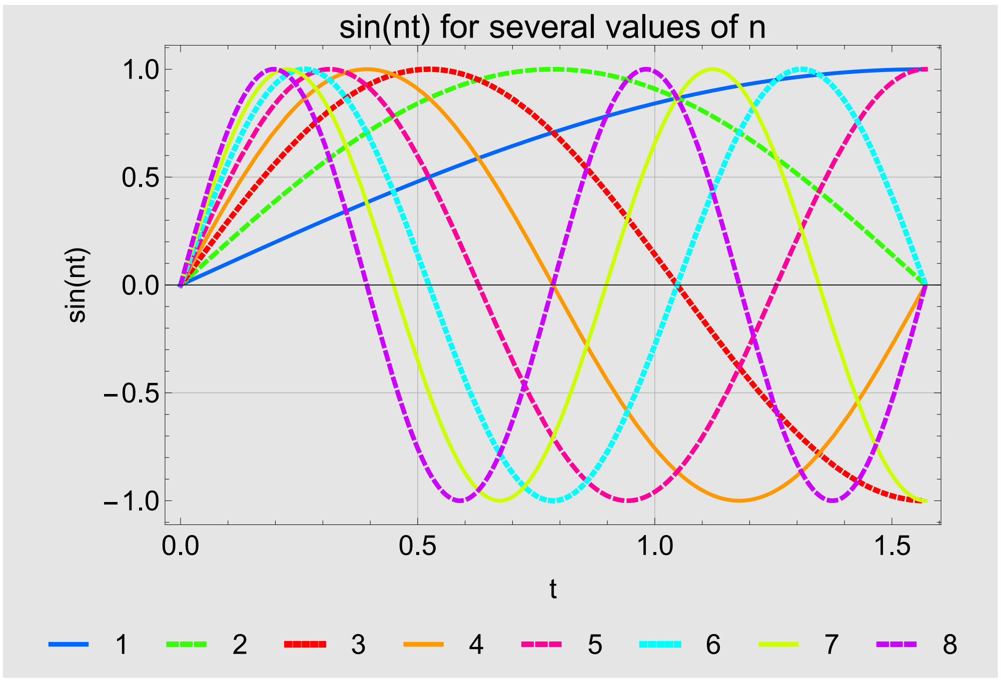


Figure 1: Example of output of the function `SPlot`

2.5.3 Displaying and plotting matrices

```
SMatrixPlot [◇A_Association] :=
  (MatrixPlot[◇A["Matrix"],
    FrameTicks -> ({Transpose[{Range @@ Dimensions @ ◇A["Row␣Labels"],
      ◇A["Row␣Labels"]}],
    Transpose[{Range @@ Dimensions@◇A["Column␣Labels"],
      ◇A["Column␣Labels"]}]} /. SymbolReplacements),
    FrameTicksStyle -> Directive[Orange],
    FrameStyle -> Directive[Orange],
    ColorFunction -> "SolarColors"])

SMatrixForm [◇A_Association] :=
  (MatrixForm[◇A["Matrix"],
    TableHeadings -> ({◇A["Row␣Labels"], ◇A["Column␣Labels"]} /.
      SymbolReplacements)])

STableForm [◇A_Association] :=
  (TableForm[◇A["Matrix"],
    TableHeadings -> ({◇A["Row␣Labels"], ◇A["Column␣Labels"]} /.
      SymbolReplacements)])
```

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

2.6 Typing palette

In order to ease the typing of some of the symbols used in the codes, a typing plalette is created whenever package MoSs is used.

```
CreatePalette[
 Grid[Partition[
   PasteButton[Style[#, 12], RawBoxes[#], ImageSize -> {45, 30}] & /@ {
     "◇", "#", "§", "£",
     "q", "q_#", "q̄", "q̲",
     "q̲°", "d", "A", "C",
     "r̲", "[1]_□", "|", "○"
     "■̇", "■̈", "■̄", "■̲",
     "■̃", "■^□", "■_□", "■̊",
     "(■)", "{■}", "[[■]]", "[■]",
     "< |■| >", "⟨■⟩", "\"■\"", "(*■*)"
     }, 4], Spacings -> {0, 0}]];
```

In this piece of code, □ and ■ represent \[Placeholder] and \[SelectionPlaceholder] elements respectively.

3 Main functions

This section presents the main functions of the package MoSs, which are directly related to the application of the algorithm presented in Section 1.

3.1 Modular Modelling

```
MoSs[◇System_, ◇SubSystems_List: {}] :=

  Module[{◇In, ◇Out, ◇Rules, ◇Keys, ◇A, ◇Timer},
    ◇Timer = AbsoluteTime[];
    ◇In = ◇System;
    ◇Out = If[AssociationQ[◇In], ◇In, Association[]];
    Quiet @ (
    ◇Out["System␣Label"] = ◇In /. {
      ◇X_List /; Length[◇X] >= 1 -> ◇X[[1]],
      ◇X_Association -> ◇X["System␣Label"]
      };
    ◇Out["Subsystems␣Labels"] = ◇In /. {
      ◇X_Association /; KeyExistsQ[◇X, "Subsystems␣Labels"] ->
        ◇X["Subsystems␣Labels"],
      ◇X_ -> {}
      };
    ◇Out["Description"] = ◇In /. {
      ◇X_List /; And[Length[◇X] >= 2,
      StringQ[◇X[[2]]]] -> ◇X[[2]],
      ◇X_Association /; And[
        KeyExistsQ[◇X, "Description"], StringQ[◇X["Description"]]] ->
        ◇X["Description"],
      ◇X_ -> ""
      };
    ◇Out["Replacement␣Rules"] = ◇In /. {
      ◇X_List /; Length[◇X] >= 3 -> ◇X[[3]],
      ◇X_Association /; KeyExistsQ[◇X, "Replacement␣Rules"] ->
        ◇X["Replacement␣Rules"],
      ◇X_ -> {}
      };
    ◇Out[r] = ◇In /. {
      ◇X_List /; Length[◇X] >= 4 -> ◇X[[4]],
      ◇X_Association /; KeyExistsQ[◇X, r] -> ◇X[r],
      ◇X_ -> {}
      };
    );


    Quiet@(
    ◇In = (◇SubSystems[[#]]) /. {
      ◇X_List /; AssociationQ[◇X[[1]]] -> ◇X[[1]]
      };
    ◇Rules["Replacement␣Rules"] = Join[
      (◇SubSystems[[#]]) /. {
        ◇X_List /; And[
          Length[◇X] >= 2, Or[ListQ[◇X[[2]]], AssociationQ[◇X[[2]]]]]
          -> Normal @ (◇X[[2]]),
        ◇X_ -> {}
        },
      ◇Out["Replacement␣Rules"]
      ];
    ◇Rules[r] = Join[
      (◇SubSystems[[#]]) /. {
        ◇X_List /; And[
          Length[◇X] >= 3, Or[ListQ[◇X[[3]]], AssociationQ[◇X[[3]]]]]
          -> Normal @ (◇X[[3]]),
        ◇X_ -> {}
        },
      ◇Out[r]
      ];
    ◇In = SRename[◇In, ◇Rules["Replacement␣Rules"], ◇Rules[ r]];

    ◇Out["Subsystems␣Labels"] = Union[
      ◇Out["Subsystems␣Labels"], {◇In["System␣Label"]}];
```

```
◇Out[◇In["System␣Label"]] = ◇In;
◇Out[r] = Union[
  ◇Out[r], ◇Rules[r]];
) & /@ Range @ (Length @ ◇SubSystems);

◇In = ◇Out;

◇Out["q:Order"] = If[
  KeyExistsQ[◇In, "q:Order"],
  ◇In["q:Order"],
  Max @ (((◇In[#]["q:Order"]) & /@ ◇In["Subsystems␣Labels"]) //.
    Missing[◇X__] -> {})];

(
If[◇In[#]["q:Order"] < ◇Out["q:Order"],
  ◇In[#]["q:Order"] = ◇Out["q:Order"]];
◇Out[#] = MoSs[◇In[#]];
) & /@ ◇In["Subsystems␣Labels"];

If[◇Out["Debug␣Mode"] === "On",
Print[StringForm["``:``:Subsystems:OK",
  NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
  ◇Out["System␣Label"]]]];

◇Keys = Part[#, 1] & /@
  Union @ (Flatten@{(Select[Keys @ ◇In, Part[#, 0] == q &]),
    (Select[Keys @ ◇In[#], Part[#, 0] == q &]) & /@
    ◇In["Subsystems␣Labels"]});
Function[◇Key,
  ◇Out[q[◇Key]] = (Union @ (Flatten @ (
    {◇In[q[◇Key]],
    Function[◇Sub, ◇In[◇Sub][q[◇Key]]] /@
      ◇In["Subsystems␣Labels"]} //. Missing[◇X__] -> {})
    ))] /@ ◇Keys;
Function[◇Key,
     ◇In[q[◇Key]] = Complement[◇Out[q[◇Key]]//.
      Missing[◇X__] -> {},
      (Union @ (Flatten @ ({Function[◇Sub,
       ◇In[◇Sub][q[◇Key]]] /@ ◇In["Subsystems␣Labels"]} //.
      Missing[◇X__] -> {})
      ))]
     ] /@ ◇Keys;

    ◇Out["q:Def:Order"] = If[
     KeyExistsQ[◇In, "q:Def:Order"],
     ◇In["q:Def:Order"],
     Max @ ToExpression @ Flatten @ (StringSplit[#, {":", "|"}] & /@
      ◇Keys)];

    (◇Out[q[ToString @ #]] =
    D[◇Out[q[ToString @ ◇Out["q:Def:Order"]]],
     {t, (# - ◇Out["q:Def:Order"])}]) & /@
     Complement[Range[0, Max[2, ◇Out["q:Order"]]],
     Range[0, ◇Out["q:Def:Order"]]];

    ◇Keys = Union[ReplaceRepeated[#, {{◇A_, ◇B_} :>
     (ToString[◇A] <> "|" <> ToString[◇B])}] & @
     (Select[Flatten[#, 1], (Part[#, 1] > Part[#, 2]) &] & @
      (Outer[List, #, #]))] & @ Range[0, Max[2, ◇Out["q:Order"]]];
    (◇Out[q[#]] = D[◇Out[q[Part[#, 2]]] //.
     Missing[◇X__] -> {},
     {t, ((ToExpression @ Part[#, 1]) - (ToExpression @ Part[#, 2]))}]
     & @ StringSplit[#, {":", "|"}];
    ) & /@ ◇Keys;

    If[◇Out["Debug␣Mode"] === "On",
     Print[StringForm["``:``:q:OK",
     NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
     ◇Out["System␣Label"]]]];

```

```
   ◇Keys = Part[#, 1] & /@
    Union @ (Flatten @ {(Select[Keys @ ◇In, Part[#, 0] == c̄ &]),
     (Select[Keys @ ◇In[#], Part[#, 0] == c̄ &]) & /@
     ◇In["Subsystems␣Labels"]});
   Function[◇Key,
    ◇Out[c̄[◇Key]] = (Union @ (Flatten @
     ({◇In[c̄[◇Key]],
     Function[◇Sub, ◇In[◇Sub][c̄[◇Key]]] /@
     ◇In["Subsystems␣Labels"]} //. Missing[◇X__] -> {})
    ))] /@ ◇Keys;
   (◇Out[c̄[#]] = {}) & /@ Complement[ToString /@
    Range[0, ◇Out["q:Order"]], ◇Keys];

   ◇Rules = (Union @ (Flatten @ ({◇In[c̄],
    Function[◇Sub, ◇In[◇Sub][c̄]] /@
     ◇In["Subsystems␣Labels"]} //. Missing[◇X__] -> {})
    ));
   (◇Out[q[(ToString @ #) <> "|" <> (ToString @ (# - 1))]] =
    (Union @ (Flatten @ ({◇In[q[(ToString @ #) <> "|" <>
     (ToString @ (# - 1))]],
    Function[◇Sub,
     ◇In[◇Sub][q[(ToString @ #) <> "|" <> (ToString @ (# - 1))]]] /@
     ◇In["Subsystems␣Labels"]} //. Missing[◇X__] -> {})
    ));
   If[And[Length[Complement[◇Out[q[(ToString @ #) <> "|" <>
    (ToString @ (# - 1))]],
    ◇Out[q[ToString @ #]], First /@ ◇Rules]] > 0],
    ◇Out[q[(ToString @ #) <> "|" <> (ToString @ (# - 1))]] =
     Union @@ {
      ◇Out[q[(ToString @ #) <> "|" <> (ToString @ (# - 1))]],
      (Simplify @ Flatten @ (Quiet @ Solve[(# == 0) & /@
       (RedundantElim @((RedundantElim @ (Union @@ {D[
       ◇Out[c̄[ToString @ (# - 1)]], t], ◇Out[c̄[ToString @ #]]} //.
       ◇Rules)) //. ◇Rules)),
      Complement[Complement[◇Out[q[(ToString @ #) <> "|" <>
       (ToString @ (# - 1))]], ◇Out[q[ToString @ #]]],
       First /@ ◇Rules]])) //. ◇Rules
     }];
   ◇Rules = Union @@ {◇Rules, ◇Out[q̲[(ToString @ #) <> "|" <>
    (ToString @ (# - 1))]]};) & /@ Range[1, ◇Out["q:Def:Order"]];
   ◇Out[c̄] = Union[◇Rules, ◇Rules /.
    {(◇A_ -> ◇B_) -> (-◇A -> -◇B)}];
   ◇Out[r̲] = Union[#, # /.
    {(◇A_ -> ◇B_) -> (-◇A -> -◇B)}] & @
    (Union[#, # /. ◇Out[c̄]] & @ ◇In[r̲]);

   If[◇Out["Debug␣Mode"] === "On",
    Print[StringForm["``:``:q̲:OK",
    NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
    ◇Out["System␣Label"]]]];

   ◇A = {};

   If[KeyExistsQ[◇In, f],
    AppendTo[◇A, ◇In[f]]];
    If[KeyExistsQ[◇Out[#], d̄],
     AppendTo[◇A, ◇Out[#][d̄]]] & /@
      ◇In["Subsystems␣Labels"];
     ◇Out[d̄] = ◇Out[d] = ⟨##⟩ & @@ (RedundantElim @ ◇A);

   If[◇Out["Debug␣Mode"] === "On",
    Print[StringForm["``:``:d:OK",
    NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
    ◇Out["System␣Label"]]]];

   ◇Keys = Part[#, 1] & /@
    (Select[Keys @ ◇In, Part[#, 0] == q̄ &]);

   If [Length[◇Keys] > 0,

      ◇Keys = Part[#, 1] & /@ Select[Keys @ ◇In, Part[#, 0] == q̄ &];
      ◇Out["q̄:Def:Order"] =
        If[KeyExistsQ[◇In, "q̄:Def:Order"],
          ◇In["q̄:Def:Order"],
          Max @ ToExpression @ Flatten @ (StringSplit[#, {":", "|"}] & /@
          ◇Keys)];
          Function[◇Key, ◇Out[q̄[◇Key]] =
            (◇In[q̄[◇Key]] //. Missing[◇X__] -> {});
            ] /@ ◇Keys;
          (◇Out[q̄[#]] = {}) & /@ Complement[
            ToString /@ Range[0, Max[2, ◇Out["q:Order"],
              ◇Out["q̄:Def:Order"]]], ◇Keys];

      If[Not @ (◇Out["q̄?"] === "No"),
        ◇Keys = Part[#, 1] & /@ (Select[Keys @ ◇Out,
          Part[#, 0] == c̄ &]);
        (◇Out[q̄[#]] = (Union @@ ({
          ◇Out[q̄[#]], ◇In[c̄[#]]} //.
          Missing[◇X__] -> {}))) & /@ ◇Keys;
        (◇Out[q̄[ToString @ #]] = (RedundantElim @(( Union @@ {D[
          ◇Out[q̄[ToString @ (# - 1)]], t], ◇Out[q̄[ToString @ #]]}) //.
          ◇Out[c̄]));
        ◇Out[q̄[ToString @ #]] = Union @ (RedundantElim @
          (◇Out[q̄[ToString @ #]] //. ◇Out[c̄]));
        ) & /@ Range[1, Max[2, ◇Out["q:Order"]]];
        ,
        (◇Out[q̄[ToString @ #]] =
          D[◇Out[q̄[ToString @ (# - 1)]], t] //. ◇Out[c̄]) & /@
          (Complement[Range[0, Max[2, ◇Out["q:Order"]]],
            Range[0, ◇Out["q̄:Def:Order"]]]);
        ];

      If[◇Out["Debug␣Mode"] === "On",
        Print[StringForm["‘‘:‘‘:q̄:OK",
        NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
        ◇Out["System␣Label"]]]];

      ◇Keys = (ToString @ ◇Out["q:Order"]);
      ◇A = {};
      If[And[KeyExistsQ[◇In, q[◇Keys]],
        Length[◇In[q[◇Keys]]] > 0],
        AppendTo[◇A, Jacobi[◇Out[q̄[◇Keys]], ◇In[q[◇Keys]]]]];
      If[And[KeyExistsQ[◇Out[#], q[◇Keys]],
        Length[◇Out[#][q[◇Keys]]] > 0],
        If[KeyExistsQ[◇Out[#], Subscript[C, \[NumberSign]]],
        AppendTo[◇A,
          Jacobi[◇Out[q̄[◇Keys]],
            ◇Out[#][q[◇Keys]]]⊙ ◇Out[#][Subscript[C, \[NumberSign]]]]],
        AppendTo[◇A,
          Jacobi[◇Out[q̄[◇Keys]],
            ◇Out[#][q[◇Keys]]]]]
        ] & /@ ◇In["Subsystems␣Labels"];
      ◇Out[A] = ⟨##⟩ & @@ ◇A;

      If[◇Out["Debug␣Mode"] === "On",
        Print[StringForm["``:``:A:OK",
        NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
        ◇Out["System␣Label"]]]];

      If[Not @ (◇Out["C?"] === "No"),
        If[KeyExistsQ[◇Out, Subscript[q, \[NumberSign]][#]],
          ◇Out[C] = OrthogonalComplement[◇Out[A],
            ◇Out[Subscript[q, \[NumberSign]][#]]],
          ◇Out[C] = OrthogonalComplement[◇Out[A],
            ToString @ ◇Out["System␣Label"]]] & @
            (ToString @ ◇Out["q:Order"]);
        ◇A = {};
        If[KeyExistsQ[◇Out[#], Subscript[C, \[NumberSign]]],
          AppendTo[◇A, ◇Out[#][Subscript[C, \[NumberSign]]]]] & /@
          ◇In["Subsystems␣Labels"];
        If[◇A === {},
          ◇Out[Subscript[C, \[NumberSign]]] = ◇Out[C],
          If[Not @ (# === {}),
            AppendTo[◇A, <|
              "Matrix" -> IdentityMatrix[Length @ #],
              "Row␣Labels" -> #,
              "Column␣Labels" -> #|>]] & @
              Complement[(◇Out[C]["Row␣Labels"]),
                Union @@ (((◇Out[#][Subscript[q, \[NumberSign]][
                ToString @ ◇Out["q:Order"]]]) & /@
                ◇In["Subsystems␣Labels"]) //. Missing[◇X__] -> {})];
            ◇Out[Subscript[C, \[NumberSign]]] =
            (⟨##⟩ & @@ ◇A)⊙◇Out[C]];
          If[◇Out["Debug␣Mode"] === "On",
            Print[StringForm["``:``:C:OK",
            NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
            ◇Out["System␣Label"]]]];
        ];

      If[And[KeyExistsQ[◇Out, d],
        KeyExistsQ[◇Out, C],
          Complement[⟨◇Out[d]⟩["Row␣Labels"],
          ⟨◇Out[C]⟩["Row␣Labels"]] === {}],
        ◇Keys = Complement[⟨◇Out[C]⟩["Row␣Labels"],
          ⟨◇Out[d]⟩["Row␣Labels"]];
        If[Not @ (◇Keys === {}),
          ◇Out[d] = ⟨
          ◇Out[d], <|
            "Matrix" -> ({0} & /@ (Range @ (Length @ ◇Keys))),
            "Column␣Labels" -> {""},
            "Row␣Labels" -> ◇Keys|>
          ⟩];
        ◇Out[d̄] = STranspose[◇Out[C]]⊙◇Out[d];
        If[◇Out["Explicit␣EOM"] === "Yes",
          (◇Out[d̲[#]] = SReplaceFullSimplify[
            Solve[(# == 0) & /@ Flatten@(Union @@ {⟨◇Out[d̄]
            ⟩["Matrix"], ◇Out[q̄[#]]}), ◇Out[q[#]]],
            ◇Out[r̲]]) & @
            (ToString @ (Max[2, ◇Out["q:Order"]]));
          If[◇Out["Debug␣Mode"] === "On",
            Print[StringForm["‘‘:‘‘:d̲:OK",
            NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
            ◇Out["System␣Label"]]]];
          ];
        ]
      ];

    If[◇Out["Debug␣Mode"] === "On",
      Print[StringForm["‘‘:‘‘:d̄:OK",
      NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
      ◇Out["System␣Label"]]]];

    If[◇Out["Timer"] === "On",
      Print[StringForm["‘‘:‘‘:OK",
      NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
      ◇Out["System␣Label"]]]];

    ◇Out
    ]
```

`MoSs` is a function that implements the modular modelling algorithm. Once enough information is provided (models of subsystems and descriptions of external constraint equations), its output is an `Association` element representing the complete model of a multibody system (dynamic equations and ν°-th order constraint equations). Two syntaxes are admissible for this function:

- `MoSs[S]`
  `S` must be an `Association` element representing a multibody system. If `S` is already a complete model, then the output of this function will be `S`.

- `MoSs[S,Ss]`

`S` can be:

(a) An `Association` element representing a multibody system.

(b) A `String` element representing the label of output multibody system.

(c) Or a `List` element with up to 4 elements:

   i. The first element is a `String` element representing the label of multibody system.

   ii. The second element (optional) is a `String` element providing a description of the system.

   iii. The third element (optional) is a `List` of replacement rules for nomenclature, applicable both to the keys and values of `Association` elements within the scope of the function.

   iv. The fourth element (optional) is a `List` of replacement rules to be applicable to the values of `Association` elements within the scope of the function.

`Ss` is a `List` element providing the models of the subsystems of this system. The elements of `Ss` can be:

(a) `Association` elements representing multibody systems which are subsystems of the output system.

(b) `List` elements with up to 3 elements:

   i. The first element is `Association` element representing a multibody system which is a subsystem of the output system.

   ii. The second element (optional) a `List` of replacement rules for nomenclature, applicable both to the keys and values of `Association` elements related to the associated subsystem within the scope of the function.

   iii. The third element (optional) is a `List` of replacement rules to be applicable to the values of `Association` elements within the scope of the function.

Some keys in `S` can have its values setted to control the execution of the internal algorithms of the function `LinearizeSystem`. These keys are:

- `"Debug␣Mode"`: whenever its value is `"On"` messages indicating the progress of the execution of the internal algorithms are shown.

- `"Timer"`: whenever its value is `"On"` a message shows the total computation time of the function.

- "$\bar{\boldsymbol{q}}$?": whenever its value is `"No"` it means that the algorithm must not complete the list of constraint equations (i.e., all the forms of the constraint equations necessary for the correct execution of the modular modelling algorithm were already provided).

- "$\bar{\boldsymbol{C}}$?": whenever its value is `"No"` the algorithm for calculating the matrix $\tilde{\tilde{\boldsymbol{C}}}$ is not executed.

- `"Explicit␣EOM"`: whenever its value is `"Yes"`, explicit forms of the differential equations of motion (EOM) are shown, i.e., the system of EOM is presented in the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x})$.

3.2 Linearization of equations of motion

3.2.1 Reference motion

```
ReferenceMotion[◇System_, ◇ReferenceValues_: {}] :=
  Module[{◇Out, ◇Keys, ◇Variables},
    ◇Keys = Part[#, 1] & /@ Union @ (Flatten @
      {(Select[Keys @ ◇System, Part[#, 0] == q &]),
      (Select[Keys @ ◇System[#], Part[#, 0] == q &]) & /@
      ◇System["Subsystems␣Labels"]
      });
    ◇Variables = Union @@ (Function[◇Key, (Union @@ (
      ({◇System[q[◇Key]],
      Union @@ (Function[◇Sub, ◇System[◇Sub][q[◇Key]]] /@
        ◇System["Subsystems␣Labels"])} //. Missing[◇X__] -> {})))] /@
        ◇Keys);
    ◇Out = Association @ (Flatten @ Outer[
      (#1 -> #2) &,
      (Superscript[#, \[EmptySmallCircle]]) & /@
        (◇Variables /. SymbolReplacements), {0}]);
      AssociateTo[◇Out, (Superscript[First[#], \[EmptySmallCircle]] ->
        Last[#]) & /@ (◇ReferenceValues /. SymbolReplacements)];
    ◇Out // Normal
  ]
```

ReferenceMotion identifies all the generalized variables in a mathematical model and creates a List of replacement rules for the reference values of these variables. Two syntaxes are admissible for this function:

- ReferenceMotion[S]: simply set all the reference values of all the generalized variables of system S to zero.
- ReferenceMotion[S,L]: L is a List of replacement rules for reference values of some of the generalized variables provided by the user; in this case, the output is a List consisting of the union of L with another List setting null reference values for all the variables that are not in L.

3.2.2 Linearization procedures

```
LinearExpansion[◇E_] = {
  Derivative[2][Subscript[Subscript[◇Argument_, ◇Indexes1__], ◇
      Indexes1◇Indexes1__]][t] ->
    Superscript[Subscript[Subscript[Overscript[◇Argument, ".."], ◇
        Indexes1], ◇Indexes1◇Indexes1], ○]
    + ◇ϵ Derivative[2][Subscript[Subscript[◇Argument, ◇Indexes1], ◇
        Indexes1◇Indexes1]][t],
  Derivative[1][Subscript[Subscript[◇Argument_, ◇Indexes1__], ◇
      Indexes1◇Indexes1__]][t] ->
    Superscript[Subscript[Subscript[Overscript[◇Argument, "."], ◇
        Indexes1], ◇Indexes1◇Indexes1], ○]
    + ◇ϵ Derivative[1][Subscript[Subscript[◇Argument, ◇Indexes1], ◇
        Indexes1◇Indexes1]][t],
  Subscript[Subscript[◇Argument_, ◇Indexes1__], ◇Indexes1◇Indexes1__][
      t] ->
    Superscript[Subscript[Subscript[◇Argument, ◇Indexes1], ◇Indexes1◇
        Indexes1], ○]
    + ◇ϵ Subscript[Subscript[◇Argument, ◇Indexes1], ◇Indexes1◇Indexes1
        ][t],
  Derivative[2][Subscript[◇Argument_, ◇Indexes1__]][t] ->
```

```
    Superscript[Subscript[Overscript[◇Argument, ".."], ◇Indexes1], ∘]
    + ◇ϵ Derivative[2][Subscript[◇Argument, ◇Indexes1]][t],
  Derivative[1][Subscript[◇Argument_, ◇Indexes1__]][t] ->
    Superscript[Subscript[Overscript[◇Argument, "."], ◇Indexes1], ∘]
    + ◇ϵ Derivative[1][Subscript[◇Argument, ◇Indexes1]][t],
  Subscript[◇Argument_, ◇Indexes1__][t] ->
    Superscript[Subscript[◇Argument, ◇Indexes1], ∘]
    + ◇ϵ Subscript[◇Argument, ◇Indexes1][t],
  Derivative[2][Subscript[◇Argument_, ◇Indexes1__]][t] ->
    Superscript[Subscript[Overscript[◇Argument, ".."], ◇Indexes1], ∘]
    + ◇ϵ Derivative[2][Subscript[◇Argument, ◇Indexes1]][t],
  Derivative[1][Subscript[◇Argument_, ◇Indexes1__]][t] ->
    Superscript[Subscript[Overscript[◇Argument, "."], ◇Indexes1], ∘]
    + ◇ϵ Derivative[1][Subscript[◇Argument, ◇Indexes1]][t],
  Subscript[◇Argument_, ◇Indexes1__][t] ->
    Superscript[Subscript[◇Argument, ◇Indexes1], ∘]
    + ◇ϵ Subscript[◇Argument, ◇Indexes1][t],
  Derivative[2][◇Argument_][t] ->
    Superscript[Overscript[◇Argument, ".."], ∘]
    + ◇ϵ Derivative[2][◇Argument][t],
  Derivative[1][◇Argument_][t] ->
    Superscript[Overscript[◇Argument, "."], ∘]
    + ◇ϵ Derivative[1][◇Argument][t],
  ◇Argument_[t] ->
    Superscript[◇Argument, ∘] + ◇ϵ ◇Argument[t]
  };

Linearize[◇A_Association, ◇ReferenceMotion_: {}] :=
  Association[
    "Matrix" -> ((Series[((((◇A["Matrix"]) /.
      LinearExpansion[◇ϵ]) /. ◇ReferenceMotion) /.
        {Superscript[◇Argument_,∘] -> 0}),
        {◇ϵ, 0, 1}] // Normal) /. {◇ϵ -> 1}),
    "Row␣Labels" -> ◇A["Row␣Labels"],
    "Column␣Labels" -> ◇A["Column␣Labels"]
    ];

Linearize[◇L_, ◇ReferenceMotion_: {}] :=
  ((Series[(((◇L /. LinearExpansion[◇ϵ]) /. ◇ReferenceMotion) /.
    {Superscript[◇Argument_,○] -> 0}), {◇ϵ, 0, 1}] // Normal)
    /. {◇ϵ -> 1});
```

Linearize obtains the linearized version of an expression (given either by a List or by an Association element) with respect to some reference values set for its generalized variables. Two syntaxes are admissible for this function:

- Linearize[E]: linearizes the expression E assuming that the reference values for all its variables are null.
- Linearize[E,R]: linearizes the expression E with respect to the reference values R (which is a list of rules similar to the outputs of function ReferenceMotion).

3.2.3 Linearized model

```
LinearizeSystem[◇System_, ◇ReferenceValues_: {},
  ◇LinSubsystemsModels_: Association[], ◇ExtraRules_: {}] :=
  Module[{◇In, ◇Out, ◇ReferenceMotion, ◇Keys, ◇A, ◇Timer},
    ◇Timer = AbsoluteTime[];
    ◇In = ◇Out = MoSs @ ◇System;
    (◇Out[#] = ◇LinSubsystemsModels[#]) & /@
      Intersection[◇Out["Subsystems␣Labels"],
        Keys[◇LinSubsystemsModels]];
    (◇Out[#] = LinearizeSystem[◇In[#], ◇ReferenceValues, ◇ExtraRules])
      & /@ Complement[◇Out["Subsystems␣Labels"],
        Keys[◇LinSubsystemsModels]];
    ◇ReferenceMotion = ReferenceMotion[◇In, ◇ReferenceValues];
    ◇Out[q°] = ◇ReferenceMotion;

    If[◇Out["Debug␣Mode"] === "On",
      Print[StringForm["``:``:q°:OK",
      NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
      ◇Out["System␣Label"]]]];

    ◇Keys = First /@ (Select[Keys @ ◇In,
      Part[#, 0] == Subscript[q, \[NumberSign]] &]);
    ◇Out["q#:Def:Order"] = If[KeyExistsQ[◇In, "q#:Def:Order"],
      ◇In["q#:Def:Order"],
      Max @ ToExpression @ Flatten @
        (StringSplit[#, {":", "|"}] & /@ ◇Keys)];

    (◇Out[Subscript[q, \[NumberSign]][ToString @ #]] =
      D[◇Out[Subscript[q, \[NumberSign]][
        ToString @ ◇Out["q#:Def:Order"]]],
      {t, (# - ◇Out["q#:Def:Order"])}]) & /@
      Complement[ Range[0, Max[2, ◇Out["q:Order"]]],
        Range[0, ◇Out["q#:Def:Order"]]];

    ◇Keys = Union[ReplaceRepeated[#, {{◇A_, ◇B_} :> (
      ToString[◇A] <> "|" <> ToString[◇B])}] & @
        (Select[Flatten[#, 1], (Part[#, 1] > Part[#, 2]) &] & @
          (Outer[List, #, #]))] & @ Range[0, Max[2, ◇Out["q:Order"]]];
    (◇Out[Subscript[q, \[NumberSign]][#]] =
      D[◇Out[Subscript[q, \[NumberSign]][Part[#, 2]]],
        {t, ((ToExpression @ Part[#, 1]) -
          (ToExpression @ Part[#, 2]))}] & @ StringSplit[#, {":", "|"}]
    ) & /@ ◇Keys;

    ◇Keys = Part[#, 1] & /@
      Union@(Select[Keys @ ◇In, Part[#, 0] == q̄ &]);
    (◇Out[q̄[#]] = Linearize[◇In[q̄[#]] //.
      ◇In[c̲], ◇ReferenceMotion] //. ◇ExtraRules) & /@
      ◇Keys;
    (◇Out[q̄[#]] = {}) & /@ Complement[ToString /@
      Range[0, Max[2, ◇Out["q:Order"]]], ◇Keys];

    ◇Out[c̲] = Union @@ (((◇Out[#][c̲]) & /@
      ◇Out["Subsystems␣Labels"]) //. Missing[◇X__] -> {});
```

```

◇Out[c] = Union @@ {
  ◇Out[c], Union[#, # /. {(◇A_ -> ◇B_) ->
    (-◇A -> -◇B)}] & @(((# -> 0) & /@
    RedundantElim @ ((Union @@ (◇Out[q̄[#]] & /@
    ◇Keys)) //. {◇X_[t] -> 0} //. ◇ExtraRules)) /.
    {({} -> 0) -> {}})
  };

◇Out[c] = Union @@ {
  ◇Out[c], ((# -> 0) & /@
    (RedundantElim @ ((Linearize[◇In[c] /.
      {(◇X_ -> ◇Y_) -> ◇X - ◇Y},
    ◇ReferenceMotion] //. ◇ExtraRules) //. ◇Out[c])))
  };

(◇Out[q̄[#]] = ◇Out[q̄[#]] //. ◇Out[c] //.
  ◇ExtraRules) & /@ ◇Keys;

If[◇Out["Debug␣Mode"] === "On",
  Print[StringForm["``:``:q̄:OK",
  NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
  ◇Out["System␣Label"]]]];

◇Keys = Part[#, 1] & /@
  Union @ (Select[Keys @ ◇In, Part[#, 0] == q &]);
  Module[{◇First, ◇Last},
    ◇First = Linearize[(First /@ ◇In[q[#]]), ◇ReferenceMotion] //.
      ◇Out[c] //. ◇ExtraRules;
    ◇Last = Linearize[(Last /@ ◇In[q[#]]), ◇ReferenceMotion] //.
      ◇Out[c] //. ◇ExtraRules;
    ◇Out[q[#]] = MapThread[(#1 - (#1 //. {◇X_[t] -> 0})) ->
      (#2 - (#2 //. {◇X_[t] -> 0})) &, {◇First, ◇Last}, 1];
    ◇Out[c] = Select[Union @@ {
      ◇Out[c], ◇Out[q[#]],
   Union[#, # /. {(◇A_ -> ◇B_) -> (-◇A -> -◇B)}] & @
   MapThread[#1 -> #2 &, {◇First, ◇Last} //. {◇X_[t] -> 0}, 1]
   }, (Not@(First[#] - Last[#] === 0)) &];
  ] & /@ ◇Keys;

Module[{◇Equations},
 ◇Equations = RedundantElim @(◇Out[q̄[#]] //. ◇Out[c̲]);
 ◇Out[q̲[#]] = If[Or[◇Equations === {},
  SetComplement[◇In[q[#]],
  ◇Out[Subscript[q, \[NumberSign]][#]]] === {}],
  {},
  Function[{◇X},
   MapThread[(#1 -> #2) &, {◇X, (Flatten@(-LinearSolve @@
   Reverse @ CoefficientArrays[◇Equations, ◇X]))}, 1]
   ] @ SetComplement[Intersection[◇In[q[#]],
    GetVariables @ ◇Equations],
    ◇Out[Subscript[q, \[NumberSign]][#]]] //. ◇ExtraRules];
 ◇Out[c̲] = Complement[Union @@ {◇Out[c̲], \
  ◇Out[q̲[#]]}, {0 -> 0}];
 ] & /@ (ToString /@ Range[0, Max[2, ◇Out["q:Order"]]]);

If[◇Out["Debug␣Mode"] === "On",
 Print[StringForm["‘‘:‘‘:c̲:OK",
 NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
 ◇Out["System␣Label"]]]];

If[KeyExistsQ[◇In, A],
 ◇Out[A] = Association[
  "Matrix" -> Simplify@(Linearize[◇In[A]["Matrix"],
   ◇ReferenceMotion] //. ◇Out[c̲] //. ◇ExtraRules),
  "Column␣Labels" -> ◇In[A]["Column␣Labels"],
  "Row␣Labels" -> ◇In[A]["Row␣Labels"]
  ];

 If[◇Out["Debug␣Mode"] === "On",
  Print[StringForm["``:``:A:OK",
  NumberForm[
  Round[AbsoluteTime[] - ◇Timer, 0.01], {5,
  2}], ◇Out["System␣Label"]]]];

 If[KeyExistsQ[◇In, C],
  ◇Out[C] = Association[
   "Matrix" -> Simplify @ (Linearize[◇In[C]["Matrix"],
    ◇ReferenceMotion] //. ◇Out[c] //. ◇ExtraRules),
   "Column␣Labels" -> ◇In[C]["Column␣Labels"],
   "Row␣Labels" -> ◇In[C]["Row␣Labels"]],
  ◇Out[C] = LSLinearizedOrthogonalComplement[◇Out[A],
   ◇Out[Subscript[q, \[NumberSign]][(ToString @
    ◇Out["q:Order"])]], ◇Out[c]]
  ];
 ];

If[◇Out["Debug␣Mode"] === "On",
 Print[StringForm["``:``:C:OK",
 NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
 ◇Out["System␣Label"]]]];

◇A = {};
◇Keys = ToString /@ Range[◇Out["q:Order"],
 (Max[2, ◇Out["q:Order"]])];
If[KeyExistsQ[◇In, f],
 ◇Out[f] = Association[
  "Matrix" -> Collect[Simplify@(Linearize[◇In[f]["Matrix"],
   ◇ReferenceMotion] //. ◇Out[c] //. ◇ExtraRules),
   Union @@ (◇Out[q[#]] & /@ ◇Keys), Simplify],
  "Column␣Labels" -> ◇In[f]["Column␣Labels"],
  "Row␣Labels" -> ◇In[f]["Row␣Labels"]
  ];
 AppendTo[◇A, ◇Out[f]];
 ];
If[KeyExistsQ[◇Out[#], d̄],
  AppendTo[◇A, SApply[(# //. ◇Out[c̲] //. ◇ExtraRules) &,
  ◇Out[#][d̄]]]
  ] & /@ ◇In["Subsystems␣Labels"];
◇Out[d̄] = ◇Out[d] = ⟨##⟩ & @@
  (RedundantElim @ ◇A);

If[And[KeyExistsQ[◇Out, d], KeyExistsQ[◇Out, C],
  Complement[⟨◇Out[d]⟩["Row␣Labels"],
  ⟨◇Out[C]⟩["Row␣Labels"]] === {}],
  ◇Keys = Complement[⟨◇Out[C]⟩["Row␣Labels"],
    ⟨◇Out[d]⟩["Row␣Labels"]];
  If[Not @ (◇Keys === {}), ◇Out[d] =
    ⟨ ◇Out[d],
    <|"Matrix" -> ({0} & /@ (Range @ (Length @ \
    ◇Keys))), "Column␣Labels" -> {""},
    "Row␣Labels" -> ◇Keys|> ⟩
    ];
  ◇Out[d̄] =
    Linearize @ (STranspose[◇Out[C]]⊙
    ◇Out[d]);
  If[◇Out["Explicit␣Linearized␣EOM"] === "Yes",
    (◇Out[d̲[#]] =
    SReplaceFullSimplify[Solve[(# == 0) & /@
      Flatten @ (Union @@ {⟨◇Out[d̄]⟩["Matrix"],
        ◇Out[q̄[#]]}), ◇Out[q[#]]], ◇Out[r̲]]) & @
        (ToString @ (Max[2, ◇Out["q:Order"]]));
    If[◇Out["Debug␣Mode"] === "On",
      Print[StringForm["‘‘:‘‘:d̲:OK",
        NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
        ◇Out["System␣Label"]]]];
    ];
  ];

If[◇Out["Debug␣Mode"] === "On",
   Print[StringForm["‘‘:‘‘:d̄:OK",
    NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
    ◇Out["System␣Label"]]]];

  ◇Out["d̄:q"] = Union @ (GetVariables @ ◇Out[d̄]);
  ◇Out["System␣Parameters"] = Union @@ {
   Union @@ (◇Out[#]["System␣Parameters"] & /@
    ◇Out["Subsystems␣Labels"]),
   RedundantElim @ (Quiet @ GetAllVariables[Join @@ {
    ◇Out[d̄]["Matrix"],
    Join @@ (◇Out[q̄[#]] & /@ (ToString /@
      Range[0, ◇Out["q:Order"]]))}] //. ◇X_[t] -> 0)};

  If[◇Out["Timer"] === "On",
   Print[StringForm["‘‘:‘‘:OK",
   NumberForm[Round[AbsoluteTime[] - ◇Timer, 0.01], {5, 2}],
   ◇Out["System␣Label"]]]];

  ◇Out
  ];
```

`LinearizeSystem` obtains the linearized version of a model given its nonlinear version. The syntax for this function is `LinearizeSystem[S,R,LM,X]`:

- `S` is an `Association` element representing a nonlinear mathematical model (e.g.: any output of `MoSs`)

- `R` is an optional argument, *whose default value is an empty* `List`, that may be a `List` element of replacement rules setting the non-zero reference values of the generalized variables of the model.

- `LM` is an optional argument, *whose default value is an empty* `Association`, that may be an `Association` element whose values correspond to linearized models of some of the subsystems of the system (whenever linearized models for subsystems are already known, it makes the linearization algorithm faster).

- `X` is an optional argument, *whose default value is an empty* `List`, that may be a `List` element of replacement rules for other symbolic variables in the linearized model (affects only the values in the output `Association` element, not its keys).

Some keys in `S` can have its values setted to control the execution of the internal algorithms of the function `LinearizeSystem`. These keys are:

- `"Debug␣Mode"`: whenever its value is `"On"` messages indicating the progress of the execution of the internal algorithms are shown.
- `"Timer"`: whenever its value is `"On"` a message shows the total computation time of the function.
- `"Explicit␣Linearized␣EOM"`: whenever its value is `"Yes"`, explicit forms of the differential equations of motion (EOM) are shown, i.e., the system of EOM is presented in the form $\dot{x} = f(t, x)$.

3.3 Auxiliar parameters evaluation

```
ParametersEval[◇System_Association, ◇PhysicalParameters_List,
  ◇ExtraRules_List: {}] :=
  Module[{◇AuxiliarParameters, ◇Invariants, ◇Variables, ◇CoeffA,
    ◇VarA, ◇CoeffC, ◇VarC},
    {◇CoeffA, ◇VarA} = SMatrixCoefficientArrays@(◇System["Ã"]);
    {◇CoeffC, ◇VarC} = SMatrixCoefficientArrays@(◇System["C̃"]);
    ◇Invariants = Expand /@ RedundantElim @ (Expand /@ (Flatten @
      (⟨◇CoeffA[1] ⊙ ◇CoeffC[1]⟩["Matrix"])
      //. ◇ExtraRules //. ◇PhysicalParameters));
    ◇Variables = Union @ GetAllVariables[◇Invariants];
    ◇AuxiliarParameters = Union @ MapThread[#1 -> #2 &,
      {◇Variables, -LeastSquares @@ (Reverse @
        CoefficientArrays[◇Invariants, ◇Variables])}, 1];
    (◇Invariants = Expand /@ RedundantElim @ (Chop @ (Expand /@
      (Flatten @ (⟨◇CoeffA[1] ⊙ ◇CoeffC[#],
        ◇CoeffA[#] ⊙ ◇CoeffC[1]⟩["Matrix"])
      //.◇ExtraRules //.◇PhysicalParameters //.◇AuxiliarParameters)));
    ◇Variables = Union @ GetAllVariables[◇Invariants];
    If[Not[◇Invariants === {}],
      ◇AuxiliarParameters = Union[◇AuxiliarParameters,
      MapThread[#1 -> #2 &, {◇Variables, -LeastSquares @@ (Reverse @
        CoefficientArrays[◇Invariants, ◇Variables])}, 1]]];
    ) & /@ ◇VarC;
    Union[◇PhysicalParameters, ◇AuxiliarParameters]
    ];
```

`ParametersEval` evaluates eventual auxiliar symbolic parameters in the linearized expressions of matrix $\tilde{C}$ (due to the use of least squares algorithm for the calculations of orthogonal complements). Its syntax is `ParametersEval[S,P,X]`:

- `S` is an `Association` element representing the model of the system.
- `P` is a `List` element of replacement rules for the values of the physical parameters of the system.
- `X` is an optional `List` element (*whose default value is an empty* `List`) for declaring extra replacement rules.

3.4 Newton-Euler equations

```
NewtonEuler[
  ◇Label_,
  ◇PositionOrientationDescription_String: "None",
  ◇GravitationalField_: "Default",
  ◇InertiaSymmetry_: "Central",
  ◇ExternalActiveTorque_List: {0, 0, 0},
  ◇ExternalActiveForce_List: {0, 0, 0}
  ] :=
  Module[{◇Out},
    ◇Out = <|
      "System␣Label" -> ◇Label,
      "Description" -> ToString @ StringForm[
         "Newton-Euler␣equations␣of␣the␣free␣rigid␣body␣``", ◇Label],
      "q:Order" -> 1
      |>;

    ◇Out[q["1"]] = ◇Out[Subscript[q, \[NumberSign]]["1"]] = {
        Subscript[v, ◇Label, "x"][t], Subscript[v, ◇Label, "y"][t],
        Subscript[v, ◇Label, "z"][t], Subscript[ω, ◇Label, "x"][t],
        Subscript[ω, ◇Label, "y"][t], Subscript[ω, ◇Label, "z"][t]
        };

     If[StringMatchQ[(ToUpperCase @ ◇PositionOrientationDescription),
       ___ ~~ "POSITION" ~~ ___],
       ◇Out[𝒒["0"]] = {
         Subscript[p, ◇Label, "x"][t], Subscript[p, ◇Label, "y"][t],
         Subscript[p, ◇Label, "z"][t]
         };
       ◇Out[c̄["1"]] = {
         Subscript[v, ◇Label, "x"][t] - Subscript[p, ◇Label, "x"]'[t],
         Subscript[v, ◇Label, "y"][t] - Subscript[p, ◇Label, "y"]'[t],
         Subscript[v, ◇Label, "z"][t] - Subscript[p, ◇Label, "z"]'[t]};
       ◇Out[c̲["1|0"]] = {
         Subscript[p, ◇Label, "x"]'[t] -> Subscript[v, ◇Label, "x"][t],
         Subscript[p, ◇Label, "y"]'[t] -> Subscript[v, ◇Label, "y"][t],
         Subscript[p, ◇Label, "z"]'[t] -> Subscript[v, ◇Label, "z"][t]};
        ];

     If[StringMatchQ[(ToUpperCase @ ◇PositionOrientationDescription),
       ___ ~~ "QUATERNION" ~~ ___],
         ◇Out[𝒒["0"]] = {
         Subscript[p, ◇Label, "x"][t], Subscript[p, ◇Label, "y"][t],
         Subscript[p, ◇Label, "z"][t], Subscript[q, ◇Label, "x"][t],
         Subscript[q, ◇Label, "y"][t], Subscript[q, ◇Label, "z"][t],
         Subscript[q, ◇Label, "t"][t]
         };
         ◇Out[
         ToString @
         StringForm["[𝟏]𝒩|“", ◇Label]] = QuatToRot @@ {
         Subscript[q, ◇Label, "x"][t], Subscript[q, ◇Label, "y"][t],
         Subscript[q, ◇Label, "z"][t], Subscript[q, ◇Label, "t"][t]
         };
```

```
    ◇Out[c̲] = {
      Subscript[q, ◇Label, "t"][t]^2
      + Subscript[q, ◇Label, "x"][t]^2
      + Subscript[q, ◇Label, "y"][t]^2
      + Subscript[q, ◇Label, "z"][t]^2 -> 1,
      1/2 Subscript[q, ◇Label, "t"][t]^2
      + 1/2 Subscript[q, ◇Label, "x"][t]^2
      + 1/2 Subscript[q, ◇Label, "y"][t]^2
      + 1/2 Subscript[q, ◇Label, "z"][t]^2 -> 1/2,
      1 - (Subscript[q, ◇Label, "x"][t]^2
      + Subscript[q, ◇Label, "y"][t]^2
      + Subscript[q, ◇Label, "z"][t]^2)
      -> Subscript[q, ◇Label, "t"][t]^2
        };
    ◇Out[c̄["0"]] = {
      -1 + Subscript[q, ◇Label, "t"][t]^2 + Subscript[q, ◇Label, "x"
         ][t]^2
      + Subscript[q, ◇Label, "y"][t]^2 + Subscript[q, ◇Label, "z"][t
         ]^2
      };
    ◇Out[c̄["1"]] = {
      Subscript[v, ◇Label, "x"][t] - Subscript[p, ◇Label, "x"]'[t],
      Subscript[v, ◇Label, "y"][t] - Subscript[p, ◇Label, "y"]'[t],
      Subscript[v, ◇Label, "z"][t] - Subscript[p, ◇Label, "z"]'[t],
      Subscript[ω, ◇Label, "z"][t]
      + 2 Subscript[q, ◇Label, "z"][t] Subscript[q, ◇Label, "t"]'[t]
      + 2 Subscript[q, ◇Label, "y"][t] Subscript[q, ◇Label, "x"]'[t]
      - 2 Subscript[q, ◇Label, "x"][t] Subscript[q, ◇Label, "y"]'[t]
      - 2 Subscript[q, ◇Label, "t"][t] Subscript[q, ◇Label, "z"]'[t],
      Subscript[ω, ◇Label, "y"][t]
      + 2 Subscript[q, ◇Label, "y"][t] Subscript[q, ◇Label, "t"]'[t]
      - 2 Subscript[q, ◇Label, "z"][t] Subscript[q, ◇Label, "x"]'[t]
      - 2 Subscript[q, ◇Label, "t"][t] Subscript[q, ◇Label, "y"]'[t]
      + 2 Subscript[q, ◇Label, "x"][t] Subscript[q, ◇Label, "z"]'[t],
      Subscript[ω, ◇Label, "x"][t]
```

```
        + 2 Subscript[q, ◇Label, "x"][t] Subscript[q, ◇Label, "t"]'[t]
        - 2 Subscript[q, ◇Label, "t"][t] Subscript[q, ◇Label, "x"]'[t]
        + 2 Subscript[q, ◇Label, "z"][t] Subscript[q, ◇Label, "y"]'[t]
        - 2 Subscript[q, ◇Label, "y"][t] Subscript[q, ◇Label, "z"]'[t]
         };
      ◇Out[c["1|0"]] = {
        Subscript[p, ◇Label, "x"]'[t] -> Subscript[v, ◇Label, "x"][t],
        Subscript[p, ◇Label, "y"]'[t] -> Subscript[v, ◇Label, "y"][t],
        Subscript[p, ◇Label, "z"]'[t] -> Subscript[v, ◇Label, "z"][t],
        Subscript[q, ◇Label, "t"]'[t] ->
        1/2 (-Subscript[q, ◇Label, "x"][t] Subscript[ω, ◇Label, "x"][t]
        - Subscript[q, ◇Label, "y"][t] Subscript[ω, ◇Label, "y"][t]
        - Subscript[q, ◇Label, "z"][t] Subscript[ω, ◇Label, "z"][t]),
        Subscript[q, ◇Label, "x"]'[t] ->
        1/2 (Subscript[q, ◇Label, "t"][t] Subscript[ω, ◇Label, "x"][t]
        + Subscript[q, ◇Label, "z"][t] Subscript[ω, ◇Label, "y"][t]
        - Subscript[q, ◇Label, "y"][t] Subscript[ω, ◇Label, "z"][t]),
        Subscript[q, ◇Label, "y"]'[t] ->
        1/2 (-Subscript[q, ◇Label, "z"][t] Subscript[ω, ◇Label, "x"][t]
        + Subscript[q, ◇Label, "t"][t] Subscript[ω, ◇Label, "y"][t]
        + Subscript[q, ◇Label, "x"][t] Subscript[ω, ◇Label, "z"][t]),
        Subscript[q, ◇Label, "z"]'[t] ->
        1/2 (Subscript[q, ◇Label, "y"][t] Subscript[ω, ◇Label, "x"][t]
        - Subscript[q, ◇Label, "x"][t] Subscript[ω, ◇Label, "y"][t]
        + Subscript[q, ◇Label, "t"][t] Subscript[ω, ◇Label, "z"][t])
        };
        ];

    If[StringMatchQ[(ToUpperCase @ ◇PositionOrientationDescription),
      ___ ~~ "EULER␣ANGLES" ~~ ___] ,
      ◇Out[q["0"]] = {
        Subscript[p, ◇Label, "x"][t], Subscript[p, ◇Label, "y"][t],
        Subscript[p, ◇Label, "z"][t], Subscript[ψ, ◇Label][t],
        Subscript[ϕ, ◇Label][t], Subscript[θ, ◇Label][t]
       };
    ◇Out[ToString @ StringForm["[1]ℕ|“", ◇Label]] =
      (Rotation @@ (Characters @ (First @
        StringSplit[◇PositionOrientationDescription,
        {":", "|", "␣"}])))[Subscript[ψ, ◇Label][t],
          Subscript[ϕ, ◇Label][t], Subscript[θ, ◇Label][t]];
    If[StringMatchQ[(ToUpperCase @ ◇PositionOrientationDescription),
      ___ ~~ "REDUNDANT" ~~ ___] ,
      ◇Out[q["1"]] = {
        Subscript[v, ◇Label, "x"][t], Subscript[v, ◇Label, "y"][t],
        Subscript[v, ◇Label, "z"][t], Subscript[ω, ◇Label, "x"][t],
        Subscript[ω, ◇Label, "y"][t], Subscript[ω, ◇Label, "z"][t],
        Subscript[ψ, ◇Label]'[t], Subscript[ϕ, ◇Label]'[t],
        Subscript[θ, ◇Label]'[t]
        };
      ◇Out[Subscript[q, \[NumberSign]]["1"]] = {
        Subscript[v, ◇Label, "x"][t], Subscript[v, ◇Label, "y"][t],
        Subscript[v, ◇Label, "z"][t], Subscript[ψ, ◇Label]'[t],
        Subscript[ϕ, ◇Label]'[t], Subscript[θ, ◇Label]'[t]
        };
      ◇Out[c̄["1"]] = {
        Subscript[v, ◇Label, "x"][t] - Subscript[p, ◇Label, "x"]'[t],
        Subscript[v, ◇Label, "y"][t] - Subscript[p, ◇Label, "y"]'[t],
        Subscript[v, ◇Label, "z"][t] - Subscript[p, ◇Label, "z"]'[t]};
        ◇Out[q̄["1"]] = Union @@ {
          ({Subscript[ω, ◇Label, "x"][t], Subscript[ω, ◇Label, "y"][t
             ],
          Subscript[ω, ◇Label, "z"][t]} -
          (AngularVelocity @ ◇Out[ToString @ StringForm[
            "[1]ℕ|“", ◇Label]]))
           }
        ,
      ◇Out[
        c̄["1"]] = Union @@ {{
          Subscript[v, ◇Label, "x"][t] - Subscript[p, ◇Label, "x"]'[t],
          Subscript[v, ◇Label, "y"][t] - Subscript[p, ◇Label, "y"]'[t],
          Subscript[v, ◇Label, "z"][t] - Subscript[p, ◇Label, "z"]'[t]
          },
          ({Subscript[ω, ◇Label, "x"][t],
          Subscript[ω, ◇Label, "y"][t],
          Subscript[ω, ◇Label, "z"][t]} -
          (AngularVelocity @ ◇Out[ToString @ StringForm[
            "[1]𝒩|“", ◇Label]]))
          }
      ];
    ];

    Module[{◇I, ◇g},
      ◇I["Spherical"] = ◇I["S"] = ({
        {Subscript[l̄, ◇Label], 0, 0},
        {0, Subscript[l̄, ◇Label], 0},
        {0, 0, Subscript[l̄, ◇Label]}
        });
      ◇I["Cylindrical␣x"] = ◇I["Cx"] = ({
        {Subscript[l̄, ◇Label, "a"], 0, 0},
        {0, Subscript[l̄, ◇Label, "r"], 0},
        {0, 0, Subscript[l̄, ◇Label, "r"]}
        });
      ◇I["Cylindrical␣y"] = ◇I["Cy"] = ({
        {Subscript[l̄, ◇Label, "r"], 0, 0},
        {0, Subscript[l̄, ◇Label, "a"], 0},
        {0, 0, Subscript[l̄, ◇Label, "r"]}
        });
      ◇I["Cylindrical␣z"] = ◇I["Cz"] = ({
        {Subscript[l̄, ◇Label, "r"], 0, 0},
        {0, Subscript[l̄, ◇Label, "r"], 0},
        {0, 0, Subscript[l̄, ◇Label, "a"]}
        });
      ◇I["Central"] = ◇I["xyz"] = ◇I["C"] = ({
        {Subscript[l̄, ◇Label, "x"], 0, 0},
        {0, Subscript[l̄, ◇Label, "y"], 0},
        {0, 0, Subscript[Ī, ◇Label, "z"]}
        });
      ◇I["xy␣Plane"] = ◇I["xy"] = ◇I["z"] = ({
        {Subscript[Ī, ◇Label, "xx"],
        Subscript[Ī, ◇Label, "xy"], 0},
        {Subscript[Ī, ◇Label, "xy"],
        Subscript[Ī, ◇Label, "yy"], 0},
        {0, 0, Subscript[Ī, ◇Label, "zz"]}
        } );
      ◇I["xz␣Plane"] = ◇I["xz"] = ◇I["y"] = ({
        {Subscript[Ī, ◇Label, "xx"], 0,
        Subscript[Ī, ◇Label, "xz"]},
        {0, Subscript[Ī, ◇Label, "yy"], 0},
        {Subscript[Ī, ◇Label, "xz"], 0,
        Subscript[Ī, ◇Label, "zz"]}
        });
      ◇I["yz␣Plane"] = ◇I["yz"] = ◇I["x"] = ({
        {Subscript[Ī, ◇Label, "xx"], 0, 0},
        {0, Subscript[Ī, ◇Label, "yy"],
        Subscript[Ī, ◇Label, "yz"]},
        {0, Subscript[Ī, ◇Label, "yz"],
        Subscript[Ī, ◇Label, "zz"]}
        });
      ◇I[◇X_] := ({
        {Subscript[Ī, ◇Label, "xx"],
        Subscript[Ī, ◇Label, "xy"],
        Subscript[Ī, ◇Label, "xz"]},
        {Subscript[Ī, ◇Label, "xy"],
        Subscript[Ī, ◇Label, "yy"],
        Subscript[Ī, ◇Label, "yz"]},
        {Subscript[Ī, ◇Label, "xz"],
        Subscript[Ī, ◇Label, "yz"],
        Subscript[Ī, ◇Label, "zz"]}
        });

```

```
    ◇g["Default"] = ḡ {Sin[ξ̄], 0, Cos[ξ̄]};
    ◇g["None"] = {0, 0, 0};
    ◇g["x"] = ḡ {1, 0, 0};
    ◇g["-x"] = ḡ {-1, 0, 0};
    ◇g["y"] = ḡ {0, 1, 0};
    ◇g["-y"] = ḡ {0, -1, 0};
    ◇g["z"] = ḡ {0, 0, 1};
    ◇g["-z"] = ḡ {0, 0, -1};
    ◇g[◇L_List] := ◇L;
    ◇g[◇X_] := ḡ {Sin[◇X], 0, Cos[◇X]};

    ◇Out[d̄] = ◇Out[d] = ◇Out[f] = <|
      "Matrix" -> Transpose @ {Join @@ {
        -Subscript[m̄, ◇Label] (D[#, t] & /@
          {Subscript[v, ◇Label, "x"][t], Subscript[v, ◇Label, "y"][t
             ],
          Subscript[v, ◇Label, "z"][t]})
        + Subscript[m̄, ◇Label] ◇g[◇GravitationalField]
        + ◇ExternalActiveForce,
        -◇I[◇InertiaSymmetry].(D[#, t] & /@
          {Subscript[ω, ◇Label, "x"][t],
          Subscript[ω, ◇Label, "y"][t],
          Subscript[ω, ◇Label, "z"][t]})
        - {Subscript[ω, ◇Label, "x"][t],
          Subscript[ω, ◇Label, "y"][t],
          Subscript[ω, ◇Label, "z"][t]}×
          (◇I[◇InertiaSymmetry].
          {Subscript[ω, ◇Label, "x"][t],
          Subscript[ω, ◇Label, "y"][t],
          Subscript[ω, ◇Label, "z"][t]})
          + ◇ExternalActiveTorque
        }},
      "Row␣Labels" -> {
        Subscript[v, ◇Label, "x"][t],
        Subscript[v, ◇Label, "y"][t],
        Subscript[v, ◇Label, "z"][t],
        Subscript[ω, ◇Label, "x"][t],
        Subscript[ω, ◇Label, "y"][t],
        Subscript[ω, ◇Label, "z"][t]
      },
      "Column␣Labels" -> {""}
      |>;
    ];

  ◇Out
  ]
```

NewtonEuler provides the Newton-Euler equations based model of a single free rigid-body. The syntax for this function is `NewtonEuler[L,PO,GF,IS,T,F]`:

- `L` is a label for identifying the system (typically a `String` element).
- `PO` is an optional argument for choosing the generalized coordinates for describing position and orientation of the rigid body. Its possible values are the following (non case sensitive) `String`s:
  - `"None"` (*default value*): defines no generalized coordinates.
  - `"Position"` or `"Position␣only"`: defines 3 generalized coordinates only - 3 Cartesian coordinates of the centre of mass of the rigid body (with respect to a coordinate system fixed to an inertial reference frame); no coordinates are defined for the orientation description.
  - `"Quaternion"`: defines a set of 7 generalized coordinates - 3 Cartesian coordinates of the centre of mass with respect to a coordinate system fixed to an inertial reference frame and 4 quaternion components for describing the orientation of a coordinate system attached to the inertial reference frame with respect to the one fixed to an inertial reference frame.
  - `"xyx␣Euler␣Angles"`, `"xyz␣Euler␣Angles"`, `"zyx␣Euler␣Angles"`, etc.: defines a set of 6 generalized coordinates - 3 Cartesian coordinates of the centre of mass with respect to a coordinate system fixed to an inertial reference frame and 3 Euler angles for describing the orientation of a coordinate system attached to the inertial reference frame with respect to the one fixed to

an inertial reference frame; the convention adopted to define the Euler angles must be set by the first 3 characters of the `String`.

  - `"xyx␣Euler␣Angles␣Redundant"`, `"xyz␣Euler␣Angles␣Redundant"`, `"zyx␣Euler␣Angles␣Redundant"`, etc.: does the same as the previous case, but also defines as quasi-velocities the time derivatives of the Euler angles (thus, the set of quasi-velocities will be redundant consisting of 3 components of velocity of the centre of mass, 3 components of the angular velocity of the rigid body with respect to an inertial reference frame and 3 time derivatives of Euler angles).

- `GF` is an optional argument for defining the gravitational field. Its possible values are ($\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ are the unity vectors of the coordinate system fixed to an inertial reference frame):
  - `"Default"` (*default value*): $\mathbf{g} = \bar{g}(\sin\bar{\xi}\,\hat{\mathbf{x}} + \cos\bar{\xi}\,\hat{\mathbf{z}})$
  - `"None"`: $\mathbf{g} = \mathbf{0}$
  - `"x"`: $\mathbf{g} = \bar{g}\,\hat{\mathbf{x}}$
  - `"-x"`: $\mathbf{g} = -\bar{g}\,\hat{\mathbf{x}}$
  - `"y"`: $\mathbf{g} = \bar{g}\,\hat{\mathbf{y}}$
  - `"-y"`: $\mathbf{g} = -\bar{g}\,\hat{\mathbf{y}}$
  - `"z"`: $\mathbf{g} = \bar{g}\,\hat{\mathbf{z}}$
  - `"-z"`: $\mathbf{g} = -\bar{g}\,\hat{\mathbf{z}}$
  - Any 3 elements `List` setting the components $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ of $\mathbf{g}$.

- `IS` is an optional argument for defining the inertia symmetry of the rigid body. Its possible values are:
  - `"Central"` (*default value*): the inertia tensor with respect to the centre of mass is represented by a diagonal matrix.
  - `"Spherical"`: the inertia tensor with respect to the centre of mass is represented by a multiple of the identity matrix.
  - `"Cylindrical␣x"` or `"Cx"`: the inertia tensor with respect to the centre of mass is represented by a diagonal matrix in which the entries associated to $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ are equal.

  - `"Cylindrical␣y"` or `"Cy"`: the inertia tensor with respect to the centre of mass is represented by a diagonal matrix in which the entries associated to $\hat{\mathbf{x}}$ and $\hat{\mathbf{z}}$ are equal.
  - `"Cylindrical␣z"` or `"Cz"`: the inertia tensor with respect to the centre of mass is represented by a diagonal matrix in which the entries associated to $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are equal.
  - `"-x"`: $\mathbf{g} = -\bar{g}\,\hat{\mathbf{x}}$
  - `"y"`: $\mathbf{g} = \bar{g}\,\hat{\mathbf{y}}$
  - `"-y"`: $\mathbf{g} = -\bar{g}\,\hat{\mathbf{y}}$
  - `"z"`: $\mathbf{g} = \bar{g}\,\hat{\mathbf{z}}$
  - `"-z"`: $\mathbf{g} = -\bar{g}\,\hat{\mathbf{z}}$
  - Any 3 elements `List` setting the components $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ of $\mathbf{g}$.

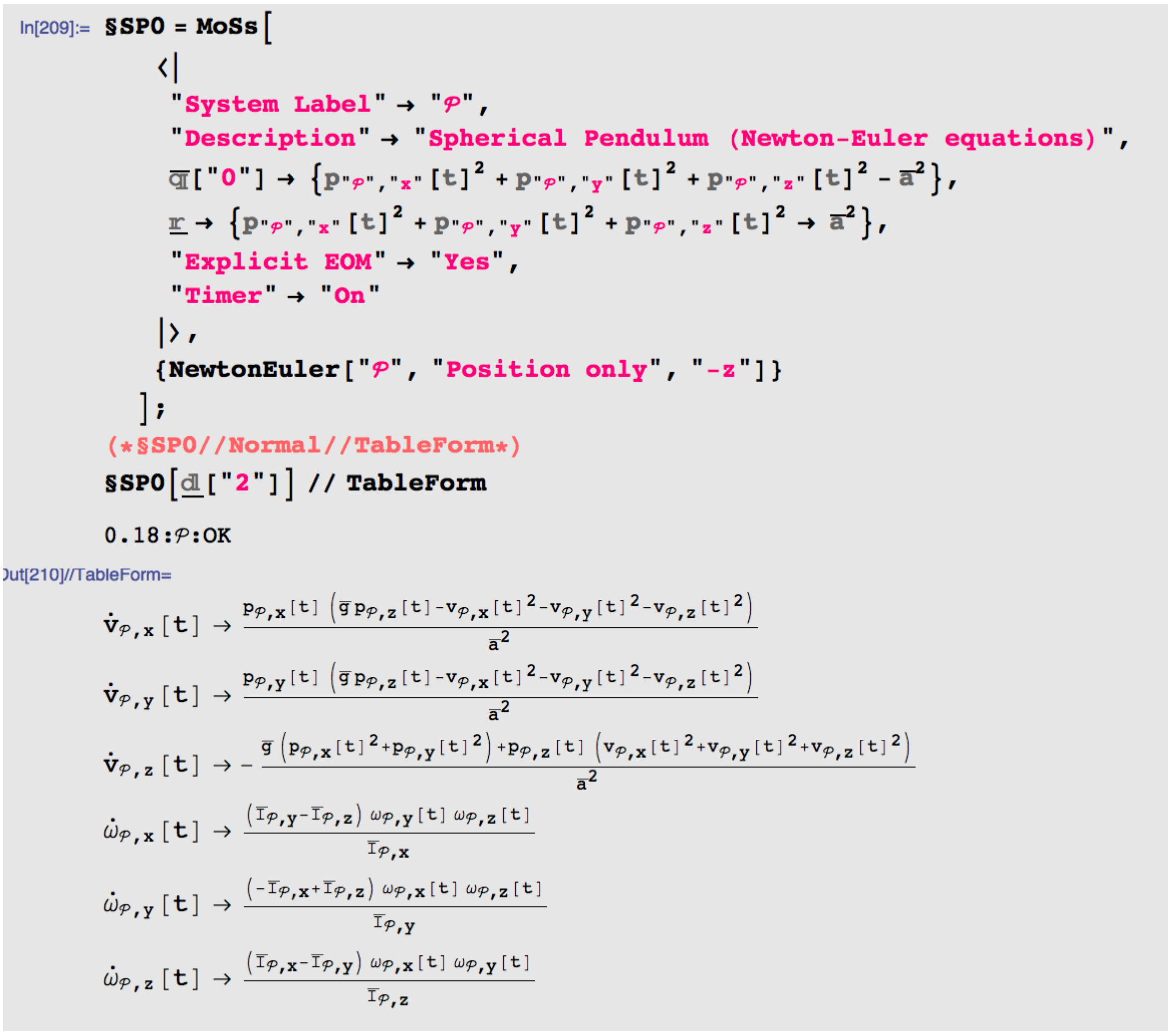
- `T` is an optional argument for setting the 3 components, with respect to a coordinate system fixed to the body, of any external torque actuating in the rigid body. Its default value is `{0,0,0}`.

- `F` is an optional argument for setting the 3 components, with respect to a coordinate system fixed to an inertial reference frame, of any external force actuating in the rigid body. Its default value is `{0,0,0}`.

4 Commented examples

4.1 Spherical pendulum modelling

Figure 2 illustrates the use of the package MoSs for obtaining a mathematical model of a spherical pendulum.



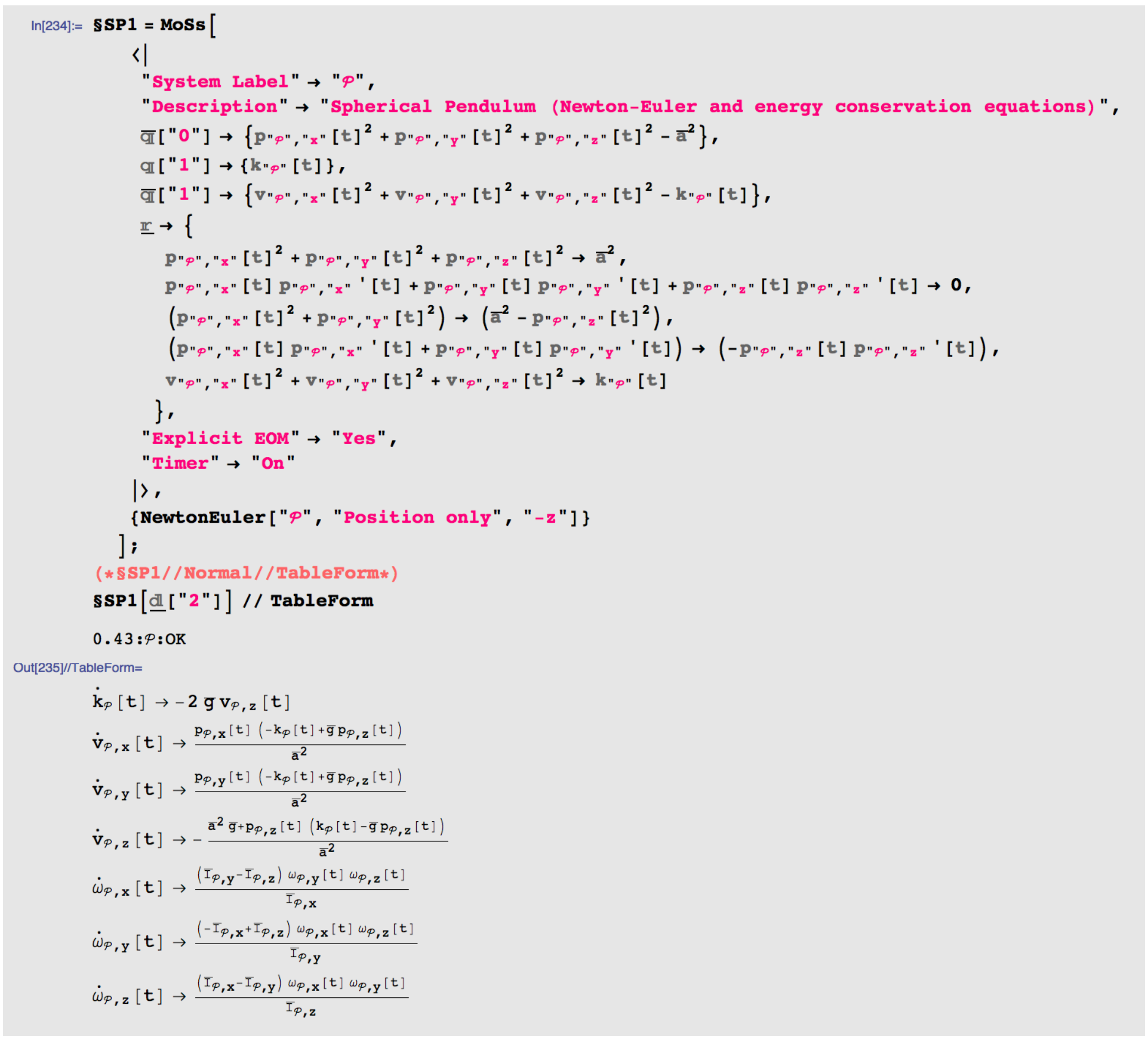
```
In[209]:= §SPO = MoSs[
             <|
              "System Label" → "𝒫",
              "Description" → "Spherical Pendulum (Newton-Euler equations)",
              q̄["0"] → {p"𝒫","x"[t]² + p"𝒫","y"[t]² + p"𝒫","z"[t]² - ā²},
              r̄ → {p"𝒫","x"[t]² + p"𝒫","y"[t]² + p"𝒫","z"[t]² → ā²},
              "Explicit EOM" → "Yes",
              "Timer" → "On"
             |>,
             {NewtonEuler["𝒫", "Position only", "-z"]}
            ];
          (*§SPO//Normal//TableForm*)
          §SPO[d̄["2"]] // TableForm

          0.18:𝒫:OK
Out[210]//TableForm=
```

$$
\begin{aligned}
\dot{v}_{\mathcal{P},x}[t] &\to \frac{p_{\mathcal{P},x}[t]\left(\bar{g}\,p_{\mathcal{P},z}[t]-v_{\mathcal{P},x}[t]^2-v_{\mathcal{P},y}[t]^2-v_{\mathcal{P},z}[t]^2\right)}{\bar{a}^2}\\
\dot{v}_{\mathcal{P},y}[t] &\to \frac{p_{\mathcal{P},y}[t]\left(\bar{g}\,p_{\mathcal{P},z}[t]-v_{\mathcal{P},x}[t]^2-v_{\mathcal{P},y}[t]^2-v_{\mathcal{P},z}[t]^2\right)}{\bar{a}^2}\\
\dot{v}_{\mathcal{P},z}[t] &\to -\frac{\bar{g}\left(p_{\mathcal{P},x}[t]^2+p_{\mathcal{P},y}[t]^2\right)+p_{\mathcal{P},z}[t]\left(v_{\mathcal{P},x}[t]^2+v_{\mathcal{P},y}[t]^2+v_{\mathcal{P},z}[t]^2\right)}{\bar{a}^2}\\
\dot{\omega}_{\mathcal{P},x}[t] &\to \frac{\left(\bar{I}_{\mathcal{P},y}-\bar{I}_{\mathcal{P},z}\right)\omega_{\mathcal{P},y}[t]\,\omega_{\mathcal{P},z}[t]}{\bar{I}_{\mathcal{P},x}}\\
\dot{\omega}_{\mathcal{P},y}[t] &\to \frac{\left(-\bar{I}_{\mathcal{P},x}+\bar{I}_{\mathcal{P},z}\right)\omega_{\mathcal{P},x}[t]\,\omega_{\mathcal{P},z}[t]}{\bar{I}_{\mathcal{P},y}}\\
\dot{\omega}_{\mathcal{P},z}[t] &\to \frac{\left(\bar{I}_{\mathcal{P},x}-\bar{I}_{\mathcal{P},y}\right)\omega_{\mathcal{P},x}[t]\,\omega_{\mathcal{P},y}[t]}{\bar{I}_{\mathcal{P},z}}
\end{aligned}
$$

Figure 2: Model of a spherical pendulum obtained using MoSs package

Basically, a spherical pendulum $\mathscr{P}$ can be conceived as a rigid body $\mathcal{P}$ whose centre of mass is constrained to move in a spherical surface. Consider that the centre of the sphere remains fixed with respect to an inertial reference frame $\mathcal{N}$. In order to use the modular modelling algorithm for this system, consider it as composed by two subsystems: $\mathscr{N}$ consisting of the inertial reference frame $\mathcal{N}$ and $\mathscr{P}^*$ consisting of the rigid body $\mathcal{P}$.



```
In[234]:= §SP1 = MoSs[
      <|
       "System Label" → "𝒫",
       "Description" → "Spherical Pendulum (Newton-Euler and energy conservation equations)",
       q̄["0"] → {p"𝒫","x"[t]^2 + p"𝒫","y"[t]^2 + p"𝒫","z"[t]^2 - ā^2},
       q["1"] → {k"𝒫"[t]},
       q̄["1"] → {v"𝒫","x"[t]^2 + v"𝒫","y"[t]^2 + v"𝒫","z"[t]^2 - k"𝒫"[t]},
       r → {
         p"𝒫","x"[t]^2 + p"𝒫","y"[t]^2 + p"𝒫","z"[t]^2 → ā^2,
         p"𝒫","x"[t] p"𝒫","x"'[t] + p"𝒫","y"[t] p"𝒫","y"'[t] + p"𝒫","z"[t] p"𝒫","z"'[t] → 0,
         (p"𝒫","x"[t]^2 + p"𝒫","y"[t]^2) → (ā^2 - p"𝒫","z"[t]^2),
         (p"𝒫","x"[t] p"𝒫","x"'[t] + p"𝒫","y"[t] p"𝒫","y"'[t]) → (-p"𝒫","z"[t] p"𝒫","z"'[t]),
         v"𝒫","x"[t]^2 + v"𝒫","y"[t]^2 + v"𝒫","z"[t]^2 → k"𝒫"[t]
        },
       "Explicit EOM" → "Yes",
       "Timer" → "On"
      |>,
      {NewtonEuler["𝒫", "Position only", "-z"]}
     ];
    (*§SP1//Normal//TableForm*)
    §SP1[d["2"]] // TableForm

    0.43:𝒫:OK

Out[235]//TableForm=
```

$$\dot{k}_{\mathcal{P}}[t] \to -2\,\bar{g}\,v_{\mathcal{P},z}[t]$$

$$\dot{v}_{\mathcal{P},x}[t] \to \frac{p_{\mathcal{P},x}[t]\left(-k_{\mathcal{P}}[t]+\bar{g}\,p_{\mathcal{P},z}[t]\right)}{\bar{a}^2}$$

$$\dot{v}_{\mathcal{P},y}[t] \to \frac{p_{\mathcal{P},y}[t]\left(-k_{\mathcal{P}}[t]+\bar{g}\,p_{\mathcal{P},z}[t]\right)}{\bar{a}^2}$$

$$\dot{v}_{\mathcal{P},z}[t] \to -\frac{\bar{a}^2\,\bar{g}+p_{\mathcal{P},z}[t]\left(k_{\mathcal{P}}[t]-\bar{g}\,p_{\mathcal{P},z}[t]\right)}{\bar{a}^2}$$

$$\dot{\omega}_{\mathcal{P},x}[t] \to \frac{\left(\bar{I}_{\mathcal{P},y}-\bar{I}_{\mathcal{P},z}\right)\omega_{\mathcal{P},y}[t]\,\omega_{\mathcal{P},z}[t]}{\bar{I}_{\mathcal{P},x}}$$

$$\dot{\omega}_{\mathcal{P},y}[t] \to \frac{\left(-\bar{I}_{\mathcal{P},x}+\bar{I}_{\mathcal{P},z}\right)\omega_{\mathcal{P},x}[t]\,\omega_{\mathcal{P},z}[t]}{\bar{I}_{\mathcal{P},y}}$$

$$\dot{\omega}_{\mathcal{P},z}[t] \to \frac{\left(\bar{I}_{\mathcal{P},x}-\bar{I}_{\mathcal{P},y}\right)\omega_{\mathcal{P},x}[t]\,\omega_{\mathcal{P},y}[t]}{\bar{I}_{\mathcal{P},z}}$$

Figure 3: Model of a spherical pendulum obtained using MoSs package: definition of a new quasi-velocity

Define a coordinate system fixed to $\mathcal{N}$ such that its origin is the centre of the spherical surface and the z-axis is vertical pointing upwards. Let $(p_{\mathcal{P},\mathrm{x}}, p_{\mathcal{P},\mathrm{y}}, p_{\mathcal{P},\mathrm{z}})$ denote the co-ordinates of the centre of mass of $\mathcal{P}$ in this coordinate system. The (external) constraint between systems $\mathscr{N}$ and $\mathscr{P}$ can be expressed by the following equation:

$$p_{\mathcal{P},\mathrm{x}}^2 + p_{\mathcal{P},\mathrm{y}}^2 + p_{\mathcal{P},\mathrm{z}}^2 - \bar{a}^2 = 0$$

Therefore, the mathematical model of $\mathscr{P}^*$ can be given by:

```
NewtonEuler["𝒫", "Position␣Only", "-z"]
```

and $\mathscr{P}$ can be defined as a system composed by the subsystems $\mathscr{N}$ (included by default) and $\mathscr{P}^*$ whose external constraints are defined by the following order 0 invariant:

```
q̄["0"]->{Subscript[p, "𝒫", "x"][t]^2 + Subscript[p, "𝒫", "y"][t]^2
  + Subscript[p, "𝒫", "z"][t]^2 - ā^2}
```

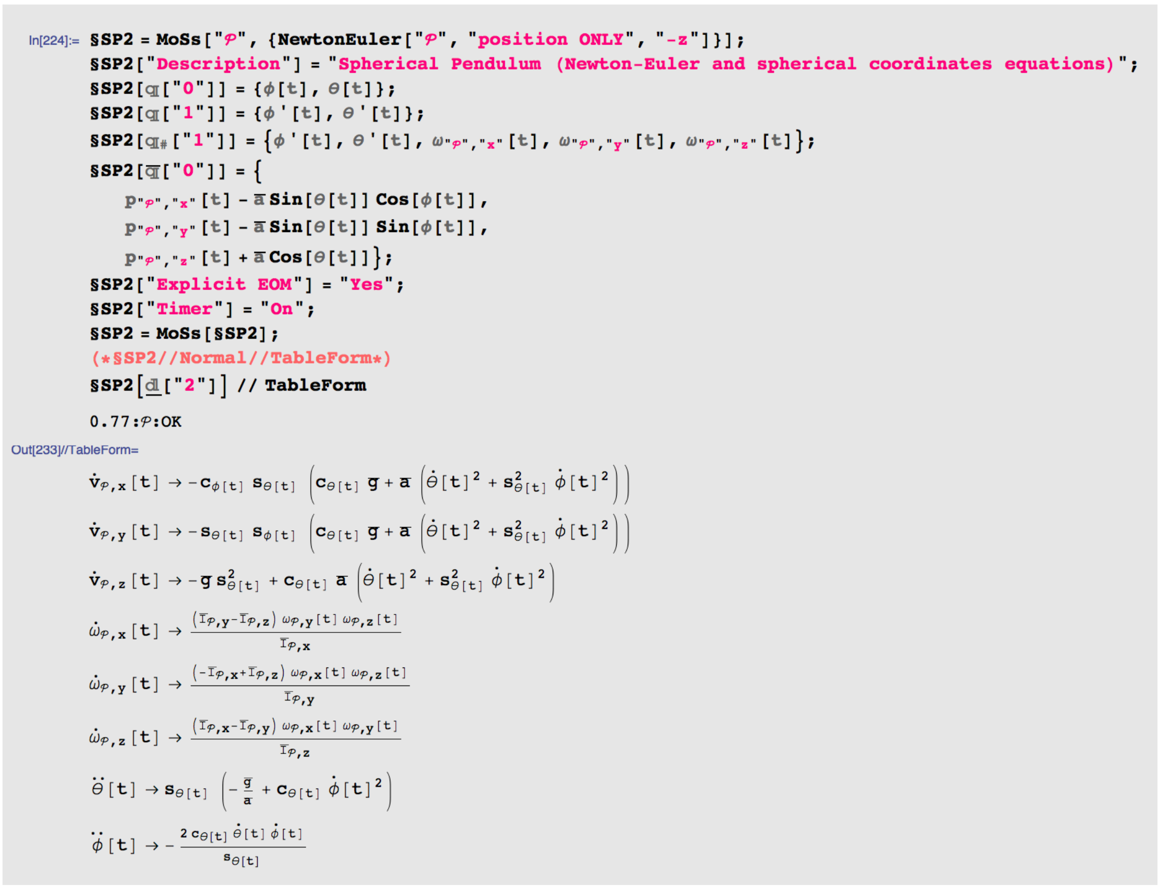
This is the strategy for modelling $\mathscr{P}$ presented in Figure 2.

However, noticing that the term $v^2_{\mathcal{P},\mathrm{x}} + v^2_{\mathcal{P},\mathrm{y}} + v^2_{\mathcal{P},\mathrm{z}}$ appears in all the dynamic equations related to the translational motion, a new quasi-velocity $k_{\mathcal{P}}$ can be defined, such that:

$$v^2_{\mathcal{P},\mathrm{x}} + v^2_{\mathcal{P},\mathrm{y}} + v^2_{\mathcal{P},\mathrm{z}} - k_{\mathcal{P}} = 0$$

This is done in the example shown in Figure 3.

Another variant of the model can be obtained when $(p_{\mathcal{P},\mathrm{x}}, p_{\mathcal{P},\mathrm{y}}, p_{\mathcal{P},\mathrm{z}})$ are parametrized in terms of spherical coordinates, leading to the most conventional version of the spherical pendulum equations of motion. This is done in the example shown in Figure 4.



```
In[224]:= §SP2 = MoSs["𝒫", {NewtonEuler["𝒫", "position ONLY", "-z"]}];
          §SP2["Description"] = "Spherical Pendulum (Newton-Euler and spherical coordinates equations)";
          §SP2[q["0"]] = {ϕ[t], θ[t]};
          §SP2[q["1"]] = {ϕ'[t], θ'[t]};
          §SP2[q#["1"]] = {ϕ'[t], θ'[t], ω"𝒫","x"[t], ω"𝒫","y"[t], ω"𝒫","z"[t]};
          §SP2[q̄["0"]] = {
              p"𝒫","x"[t] - ā Sin[θ[t]] Cos[ϕ[t]],
              p"𝒫","y"[t] - ā Sin[θ[t]] Sin[ϕ[t]],
              p"𝒫","z"[t] + ā Cos[θ[t]]};
          §SP2["Explicit EOM"] = "Yes";
          §SP2["Timer"] = "On";
          §SP2 = MoSs[§SP2];
          (*§SP2//Normal//TableForm*)
          §SP2[d["2"]] // TableForm

          0.77:𝒫:OK
Out[233]//TableForm=
```

$$\dot{v}_{\mathcal{P},x}[t] \to -c_{\phi[t]}\, s_{\theta[t]} \left(c_{\theta[t]}\, \bar{g} + \bar{a}\left(\dot{\theta}[t]^2 + s^2_{\theta[t]}\, \dot{\phi}[t]^2\right)\right)$$

$$\dot{v}_{\mathcal{P},y}[t] \to -s_{\theta[t]}\, s_{\phi[t]} \left(c_{\theta[t]}\, \bar{g} + \bar{a}\left(\dot{\theta}[t]^2 + s^2_{\theta[t]}\, \dot{\phi}[t]^2\right)\right)$$

$$\dot{v}_{\mathcal{P},z}[t] \to -\bar{g}\, s^2_{\theta[t]} + c_{\theta[t]}\, \bar{a}\left(\dot{\theta}[t]^2 + s^2_{\theta[t]}\, \dot{\phi}[t]^2\right)$$

$$\dot{\omega}_{\mathcal{P},x}[t] \to \frac{\left(\bar{I}_{\mathcal{P},y} - \bar{I}_{\mathcal{P},z}\right)\omega_{\mathcal{P},y}[t]\,\omega_{\mathcal{P},z}[t]}{\bar{I}_{\mathcal{P},x}}$$

$$\dot{\omega}_{\mathcal{P},y}[t] \to \frac{\left(-\bar{I}_{\mathcal{P},x} + \bar{I}_{\mathcal{P},z}\right)\omega_{\mathcal{P},x}[t]\,\omega_{\mathcal{P},z}[t]}{\bar{I}_{\mathcal{P},y}}$$

$$\dot{\omega}_{\mathcal{P},z}[t] \to \frac{\left(\bar{I}_{\mathcal{P},x} - \bar{I}_{\mathcal{P},y}\right)\omega_{\mathcal{P},x}[t]\,\omega_{\mathcal{P},y}[t]}{\bar{I}_{\mathcal{P},z}}$$

$$\ddot{\theta}[t] \to s_{\theta[t]}\left(-\frac{\bar{g}}{\bar{a}} + c_{\theta[t]}\, \dot{\phi}[t]^2\right)$$

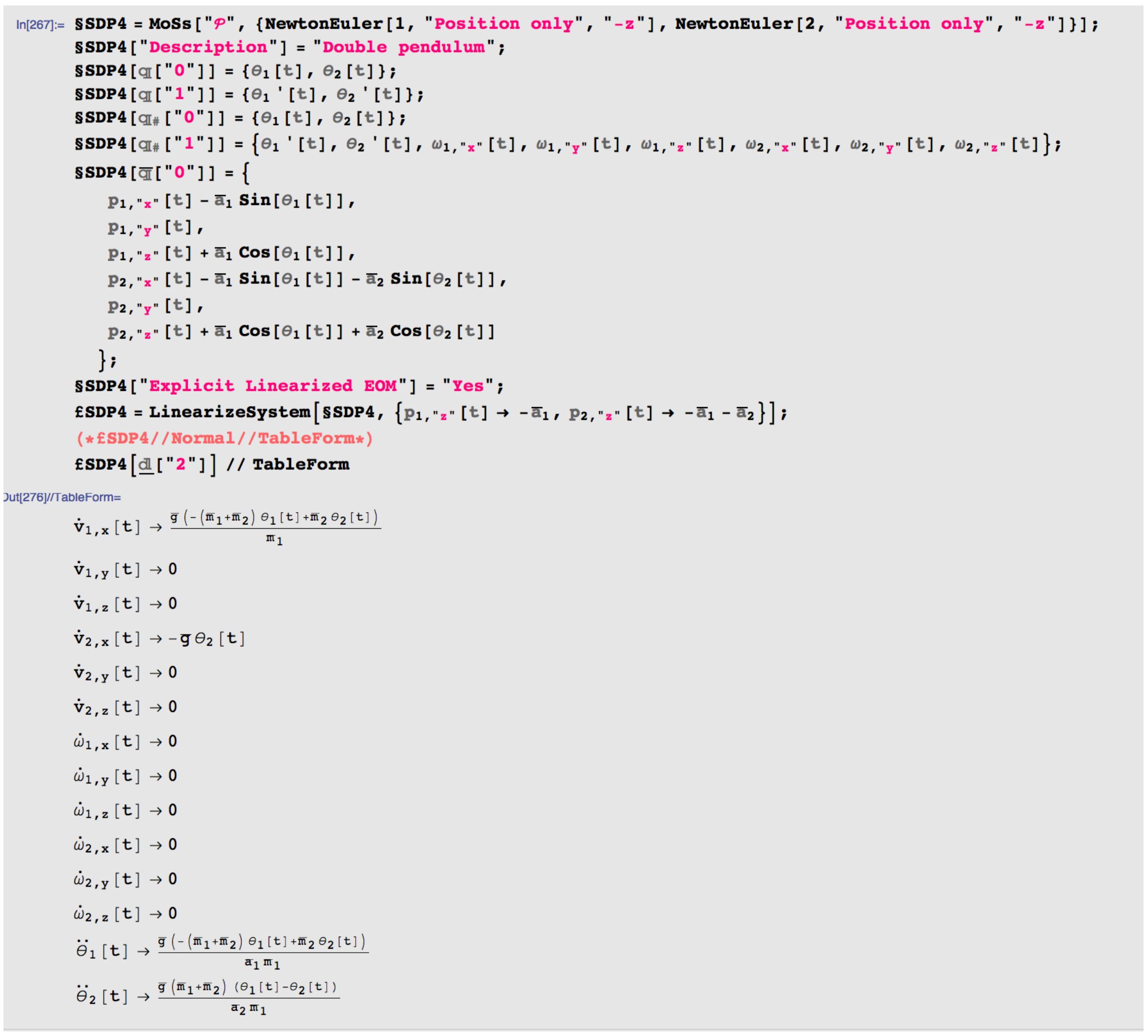
$$\ddot{\phi}[t] \to -\frac{2\, c_{\theta[t]}\, \dot{\theta}[t]\, \dot{\phi}[t]}{s_{\theta[t]}}$$

Figure 4: Model of a spherical pendulum obtained using MoSs package: model in spherical coordinates

4.2 Double pendulum modelling

The example shown in Figure 5 explores an alternative use of the syntax of the function `MoSs` and the to model a planar double pendulum. Also linearized equations of motion are obtained by the use of the function `LinearizeSystem`.

The strategy consists of defining a multibody system $\mathscr{P}$ consisting of two subsystems, 1 and 2, each one consisting of a free rigid body in a gravitational field (that has "$-$z" direction). New angular coordinates θ_1 and θ_2, as well as the quasi-velocities $\dot{\theta}_1$ and $\dot{\theta}_2$, are defined to parametrize the description of the position coordinates of the centres of mass of each of these rigid bodies. Such parametrical descriptions lead to order 0 invariants. Finally the reference state of the system is defined and the linearization procedure can be applied, leading to the linearized explicit equations of motion shown in Figure 5.



```
In[267]:= §SDP4 = MoSs["𝒫", {NewtonEuler[1, "Position only", "-z"], NewtonEuler[2, "Position only", "-z"]}];
          §SDP4["Description"] = "Double pendulum";
          §SDP4[q["0"]] = {θ₁[t], θ₂[t]};
          §SDP4[q["1"]] = {θ₁'[t], θ₂'[t]};
          §SDP4[q#["0"]] = {θ₁[t], θ₂[t]};
          §SDP4[q#["1"]] = {θ₁'[t], θ₂'[t], ω1,"x"[t], ω1,"y"[t], ω1,"z"[t], ω2,"x"[t], ω2,"y"[t], ω2,"z"[t]};
          §SDP4[q̄["0"]] = {
              p1,"x"[t] - ā₁ Sin[θ₁[t]],
              p1,"y"[t],
              p1,"z"[t] + ā₁ Cos[θ₁[t]],
              p2,"x"[t] - ā₁ Sin[θ₁[t]] - ā₂ Sin[θ₂[t]],
              p2,"y"[t],
              p2,"z"[t] + ā₁ Cos[θ₁[t]] + ā₂ Cos[θ₂[t]]
            };
          §SDP4["Explicit Linearized EOM"] = "Yes";
          £SDP4 = LinearizeSystem[§SDP4, {p1,"z"[t] → -ā₁, p2,"z"[t] → -ā₁ - ā₂}];
          (*£SDP4//Normal//TableForm*)
          £SDP4[d["2"]] // TableForm
```

Out[276]//TableForm=

$$\dot{v}_{1,x}[t] \to \frac{\bar{g}\left(-\left(\bar{m}_1+\bar{m}_2\right)\theta_1[t]+\bar{m}_2\,\theta_2[t]\right)}{\bar{m}_1}$$

$$\dot{v}_{1,y}[t] \to 0$$

$$\dot{v}_{1,z}[t] \to 0$$

$$\dot{v}_{2,x}[t] \to -\bar{g}\,\theta_2[t]$$

$$\dot{v}_{2,y}[t] \to 0$$

$$\dot{v}_{2,z}[t] \to 0$$

$$\dot{\omega}_{1,x}[t] \to 0$$

$$\dot{\omega}_{1,y}[t] \to 0$$

$$\dot{\omega}_{1,z}[t] \to 0$$

$$\dot{\omega}_{2,x}[t] \to 0$$

$$\dot{\omega}_{2,y}[t] \to 0$$

$$\dot{\omega}_{2,z}[t] \to 0$$

$$\ddot{\theta}_1[t] \to \frac{\bar{g}\left(-\left(\bar{m}_1+\bar{m}_2\right)\theta_1[t]+\bar{m}_2\,\theta_2[t]\right)}{\bar{a}_1\,\bar{m}_1}$$

$$\ddot{\theta}_2[t] \to \frac{\bar{g}\left(\bar{m}_1+\bar{m}_2\right)\left(\theta_1[t]-\theta_2[t]\right)}{\bar{a}_2\,\bar{m}_1}$$

Figure 5: Model of a double pendulum obtained using MoSs package

References

[1] R. M. M. Orsino and T. A. Hess-Coelho. A contribution on modular modelling of multibody systems. *Submitted*, 2015.

Index