CAGeM on Python
User guide

March 1, 2017

1 CAGeM: Computer-Aided Generation of Motion

The utility called CAGeM (*Computer-Aided Generation of Motion*) automatically generates the kinematics of a multibody system only from the position matrices. CAGeM is actually a Python script, which uses the symbolic features of Sympy, which is a Python library, to derive the position matrices. Sympy is a full-featured computer algebra system which is free for non commercial applications.

Besides computing the symbolic expressions of velocities and accelerations, CAGeM generates a `C++` draft, containing the main components of an EasyDyn program, which is intended to be modified by the user later on (cf. EasyDyn user guide). Based on the choice of the user, CAGeM is also able to output some scripts to plot the results obtained by EasyDyn and to automatically write a LaTeX report of the simulations.

Another Python script must be written in order to use the functionalities of CAGeM. This document thus describes all CAGeM functions by giving, for each of them, a definition, their syntax and an example. The last section of this document shows one complete example integrating most of the basic functions.

2 Basic functions

In order to use all CAGeM functionalities, the related Python script must start by importing the CAGeM functions. It is specified by the following syntax: `from cagem import *`. The CAGeM import is then followed by the call of the functions described in this document.
The CAGeM script (`CAGeM.py`) containing the source code of the functions should remain accessible either from the directory where the Python script is executed or from the directory where Python was installed:
(`C:\Users\xxxx\AppData\Local\Programs\Python\Python35`).

2.1 MBS Class

Definition:
The purpose of this class is to define the considered multibody system in terms of number of bodies (`nbrbody`), number of degrees of freedom (`nbrdof`) and number of dependent

configuration parameters (`nbrdep`). The title of the application to be created is also defined by this class as well as the name(s) of the several files that CAGeM outputs. This multibody class gathers different functions described in this document.

Syntax:

```
MBS_obj=MBSClass(nbrbody,nbrdof,nbrdep,ApplicationTitle,ApplicationFileName)
```

Example:

```
robot=MBSClass(nbrbody=10, nbrdof=6, nbrdep=0, ApplicationTitle="Simulation
of a robot", ApplicationFileName="Robot")
```

2.2 Definition of q, p, pexpr and time

Definition:

CAGeM uses the following notation:

- q: vector of configuration parameters;
- p: vector of dependent parameters;
- pexpr: vector of expressions for dependent parameters;
- t: time.
  These notations are defined with the function `Getqpt()` which is intended to be systematically called by the user Python script.

Syntax:

```
q,p,pexpr,t=MBS_obj.Getqpt()
```

Example:

- `q[0]` stands for the first configuration parameter;
- `p[5]` stands for the sixth dependent parameter;
- `pexpr[2]` stands for the third expression of the dependent parameter;

2.3 Definition of gravity

Definition:

Gravity can be defined through the function `SetGravity()`.

Syntax:

```
MBS_obj.SetGravity(gx,gy,gz)
```

Example:

```
robot.SetGravity(0,0,-9.81)
```

2.4 Definition of some constants

Definition:

Constants may be defined in the user Python script and are intended to be used within the script for an easier readability. Since it is a Python script, the user can benefit from the Python environment to define Python variables such as *numbers*, *strings*, *lists*, ...

Syntax:
Cf. Python programming language.
Example:
```
mass0=0.7
l0=1.1
Izz=l0*l0/12*mass0
```

2.5 Definition of inertia characteristics

Definition:
Inertia properties are defined for each body through the function `Set()`. The user then defines the mass (m) and the inertia tensor (`Ixx, Iyy, Izz, Ixy, Ixz, Iyz`) of the considered body, assumed to be projected in the body coordinate system. Default values for `Ixy`, `Ixz` and `Iyz` are set to zero.
Syntax:
```
MBS_obj.body[i].Set(mass,Ixx,Iyy,Izz)
MBS_obj.body[i].Set(mass,Ixx,Iyy,Izz,Ixy=0,Ixz=0,Iyz=0)
```
Example:
```
robot.body[0].Set(mass=1,Ixx=1,Iyy=1,Izz=0.0833)
robot.body[0].Set(mass=1,Ixx=1,Iyy=1,Izz=0.0833,Ixy=0.022)
```

2.6 Definition of the homogeneous transformation matrices

Definition:
The elementary homogeneous transformation matrices are defined by the functions `Tdisp`, `Trotx`, `Troty` and `Trotz`:

- `Tdisp` returns a homogeneous transformation matrix corresponding to a displacement without rotation by coordinates x, y and z:

$$Tdisp(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- `Trotx` returns a homogeneous transformation matrix corresponding to a rotation around x-axis of an angle θ:

$$Trotx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,\theta & -sin\,\theta & 0 \\ 0 & sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- `Troty` returns a homogeneous transformation matrix corresponding to a rotation around y-axis of an angle θ:

$$Troty(\theta) = \begin{bmatrix} cos\,\theta & 0 & sin\,\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\,\theta & 0 & cos\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- `Trotz` returns a homogeneous transformation matrix corresponding to a rotation around z-axis of an angle θ:

$$Trotz(\theta) = \begin{bmatrix} cos\,\theta & -sin\,\theta & 0 & 0 \\ sin\,\theta & cos\,\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Combination of elementary homogeneous transformation matrices:
Then, elementary homogeneous transformation matrices may be combined in order to obtain the complete homogeneous transformation matrix of the considered body with respect to the global reference frame by using the class variable `TOF`, in terms of system geometry, configuration parameters and possibly dependent parameters.

Syntax:

MBS_obj.body[i].TOF=T_1 * T_2 * ... * T_n

Example:

```
robot.body[0].TOF=Trotz(q[0]) * Tdisp(2.0,0,0)
```

Relative motions with respect to other bodies:
Instead of defining the complete homogeneous transformation matrix of a body with respect to the global reference frame, it can be defined relatively with respect to the coordinate system of another body by using the class variable `TrefF` instead of `TOF`. The user must then specify from which body the complete homogeneous transformation matrix should be computed by the function `ReferenceFrame()`. Using relative motions improves dramatically the simulation duration.

Syntax:

MBS_obj.body[i].TrefF=Trotz(θ) * Tdisp(x,y,z)
MBS_obj.body[i].ReferenceFrame(number of the reference body)

Example:

```
robot.body[1].TrefF=Trotz(q[3])*Tdisp(0,q[2],0)
robot.body[1].ReferenceFrame(0)
```

2.7 Definition of the initial conditions

Definition:
User may define an initial value for each configuration parameter q_k as well as for its first derivative $\dot{q}_k$ through the class variables `qini` and `qdini` whose default values are set to zero.

Syntax:
```
MBS_obj.qini[k]=value
MBS_obj.qdini[k]=value
```
Example:
```
robot.qini[0]=12.2
robot.qdini[0]=2.5
```

2.8 Definition of blocked configuration parameters

Definition:
User may block some configuration parameter with `qdoflocked[k]`. It is sometimes required to compute a static equilibrium or the poles of the system. To block configuration parameter k, user must set the variable `qdoflocked[k]` to 1. The default value of `qdoflocked[k]` is set to 0, meaning that the considered configuration parameter is unlocked.

Syntax:
```
MBS_obj.qdoflocked[k]=0 or 1
```
Example:
```
robot.qdoflocked[0]=0 or 1
```

If one of the `qdoflocked` variables is different from zero, user may preserved the blocked configuration parameters during the integration by setting the variable `lockintegration` to one. As a result, the configuration parameter q_k will be frozen during the dynamic motion of the system.

Syntax:
```
MBS_obj.lockintegration=0 or 1
```
Example:
```
robot.lockintegration=0 or 1
```

2.9 Definition of applied forces

Definition:
User may define forces applied to a specific body through the function `Force()`. Inside the brackets, the user defines the applied forces in `C++` format (cf. `EasyDyn` user guide).

Syntax:
```
MBS_obj.Force(’’’
body[0].R += 6348 * body[0].TOG.R.uz() ;
body[0].MG += (673*t-508) * body[0].TOG.R.uz() ;
’’’)
```
Example:
```
robot.Force(’’’
body[0].R += 6348 * body[0].TOG.R.uz() ;
body[0].MG += (673*t-508) * body[0].TOG.R.uz() ;
''')
```

2.10 Print the kinematics

Definition:
User may call the functions `PrintSystemSize()` or `PrintKinematics()` in order to display the kinematics through the Python dialogue window.
Syntax:
`MBS_obj.PrintSystemSize()`: displays the number of bodies, the number of degrees of freedom and the number of dependent variables.
`MBS_obj.PrintKinematics()`: displays the kinematics in terms of transformation homogeneous matrices, velocities and accelerations.
Example:

```
robot.PrintSystemSize()
robot.PrintKinematics()
```

2.11 Compute kinematics

Definition:
The function used to compute the kinematics (velocities and accelerations) is `ComputeKinematics()`.
Syntax:

```
MBS_obj.ComputeKinematics()
```

Example:

```
robot.ComputeKinematics()
```

2.12 Set the integration parameters

Definition:
User can define the integration parameters such as the duration of the simulation (`tfinal`), the time step used to save data (`hsave`), and the maximum allowed time step during numerical integration (`hmax`). Decreasing the maximum allowed time step `hmax` can solve convergence issues during integration.
Syntax:

```
MBS_obj.SetIntegrationParameters(tfinal,hsave,hmax)
```

Example:

```
robot.SetIntegrationParameters(tfinal=2,hsave=0.01,hmax=0.005)
```

2.13 User's flags

Definition:
There are three flags in CAGeM to be set either to 0 (disable) or 1 (active):

- `STATIC` is set to 1 in case the user wants CAGeM to generate the code to search for static equilibrium before integration;

- `POLE` is set to 1 if the user wants to perform an eigen value analysis after a static equilibrium;
- `TEST` is set to 1 if the user wants to perform the efficiency tests. The results of the efficiency tests are then stored in a .perf file after the execution of the `EasyDyn` simulation.

Then, the function `EasyDynFlags(STATIC,POLE,TEST)` transfers the flag values into MBS Class. By default, their values are set to zero.

Syntax:

- `MBS_obj.EasyDynFlags(STATIC=0,POLE=0,TEST=0).`

Example:

`robot.EasyDynFlags(STATIC=1,POLE=0,TEST=1)`

2.14 Export the `EasyDyn` program

Definition:

User may call the function `ExportEasyDynProgram()` to generate the `EasyDyn` program in `C++` format.

Syntax:

`MBS_obj.ExportEasyDynProgram()`

Example:

`robot.ExportEasyDynProgram()`

2.15 Graphics

Definition:

User may call the function `ExportGnuplotScript()` to generate a .plt file which will be read by `GnuPlot` to plot the graphics of displacement, velocity and acceleration of each configuration parameter, once the `EasyDyn` simulation has been successfully performed. The GnuPlot file then creates .eps figures which will be used in the LaTeX report.

Syntax:

`MBS_obj.ExportGnuplotScript()`

Example:

`robot.ExportGnuplotScript()`

2.16 Write a LaTeX report in French or in English

Definition:

User may call the function (`ExportFR_Latex_Report()`) or (`ExportUK_Latex_Report()`) either for the French version of the report or for the English version. Plots of graphics in .eps format must be generated by GnuPlot prior compiling the LaTeX file.

Syntax:

- `MBS_obj.ExportFR_Latex_Report()`
- `MBS_obj.ExportUK_Latex_Report()`

Example:

```
robot.ExportFR_Latex_Report()
robot.ExportUK_Latex_Report()
```

2.17 Summary of the basic functions

Function	Syntax
CAGeM import	from cagem import *
MBS Class	MBS_obj=MBSClass(nbrbody,nbrdof,nbrdep, ApplicationTitle,ApplicationFileName)
q, p, pexpr, t	q,p,pexpr,t=MBS_obj.Getqpt()
Gravity	MBS_obj.SetGravity(gx,gy,gz)
Constants	constant=x
Mass and Inertia	MBS_obj.body[i].Set(mass=m,Ixx=Ixx,Iyy=Iyy,Izz=Izz)

Homogeneous transformation matrices	Tdisp(x,y,z), Trotx(θ), Troty(θ), Trotz(θ)
	MBS_obj.body[i].TOF=Trotz(θ) * Tdisp(x,y,z)
	MBS_obj.body[i].TrefF=Trotz(θ) * Tdisp(x,y,z)
	MBS_obj.body[i].ReferenceFrame(body of reference)

Initial conditions	MBS_obj.qini[k]=value and MBS_obj.qdini[k]=value
Blocked configuration parameters	MBS_obj.qdoflocked[k]=0 or 1
Applied force	MBS_obj.Force('''Force in C++ format''')
Print kinematics	MBS_obj.PrintSystemSize() and MBS_obj.PrintKinematics()
Compute kinematics	MBS_obj.ComputeKinematics()
Set integration parameters	MBS_obj.SetIntegrationParameters(tfinal,hsave,hmax)
User's flags	MBS_obj.EasyDynFlags(STATIC=0,POLE=0,TEST=0)
Export the EasyDyn program	MBS_obj.ExportEasyDynProgram()
Graphics	MBS_obj.ExportGnuplotScript()
Export the LaTeX report	MBS_obj.ExportFR_Latex_Report() and MBS_obj.ExportUK_Latex_Report()

3 Full example

3.1 Double pendulum: EasyDyn classic

```
#!/usr/bin/python3

from cagem import *

dp3=MBSClass(nbrbody=2, nbrdof=2, nbrdep=0,\
ApplicationTitle="Simulation of a double pendulum",\
    ApplicationFileName="dprel")
q,p,pexpr,t=dp3.Getqpt()

dp3.SetGravity(0,-9.81,0)

l0=1.2
l1=1.1

# Inertia characteristics
dp3.body[0].Set(mass=1.1,Ixx=1,Iyy=1,Izz=l0*l0/12*1.1)
dp3.body[1].Set(mass=0.9,Ixx=1,Iyy=1,Izz=l1*l1/12*0.9)

# Definition of the position matrices
dp3.body[0].T0F=Trotz(q[0]) * Tdisp(0,-l0/2,0)
dp3.body[1].TrefF=Tdisp(0,-l0/2,0) * Trotz(q[1]) * Tdisp(0,-l1/2,0)
dp3.body[1].ReferenceFrame(mainframe=0)

# Initial conditions
dp3.qini[1]=1

# Simulation parameters
dp3.SetIntegrationParameters(tfinal=5, hsave=0.01, hmax=0.005)

dp3.PrintSystemSize()

dp3.ComputeKinematics()
dp3.PrintKinematics()

# Set STATIC to 1 in case you want CAGeM to generate the code
# to search for static equilibrium before integration
STATIC=0
# Set POLE to 1 if you want to perform an eigen value analysis
POLE=0
#Set TEST to 1 if you want to perform the efficiency tests
TEST=0
```

```
dp3.EasyDynFlags(STATIC,POLE,TEST)
dp3.ExportEasyDynProgram()

# Uncomment if you want the LaTeX report in French
dp3.ExportFR_Latex_Report()
# Uncomment if you want the LaTeX report in English
dp3.ExportUK_Latex_Report()
# Uncomment to plot the evolution of position, velocity and acceleration
dp3.ExportGnuplotScript()
```