Python Framework – For structural dynamics

User´s Guide

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Softwaretechnik

ABSTRACT

This document represents the user guide that illustrates how to install and run the Python Framework for modeling and simualtion of variable-structure models under Windows OS. The aim is to enable the user to create his own models and simulate them with the given Framework. An introduction to the Framework is given and examples explain how to use the Framework.

Contents

1 Introduction

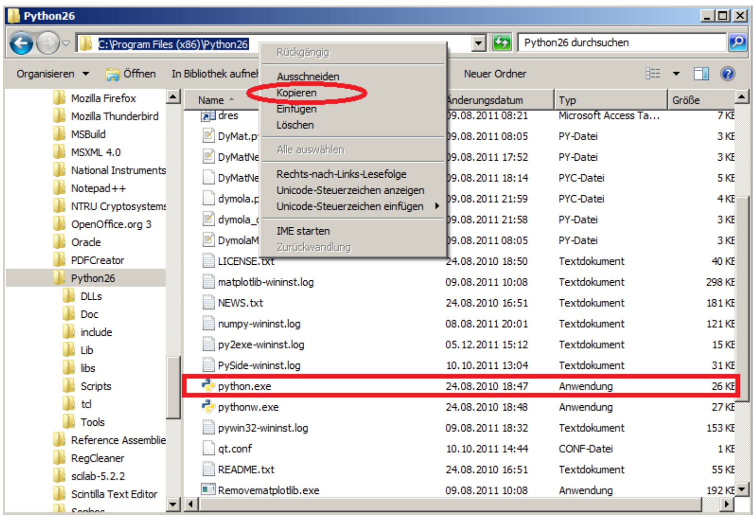
In this section we show you how to set up DySMo. If you have any questions or remarks please write an email to a.mehlhase@tu-berlin.de or leave a comment at `https://bitbucket.org/amehlhase/dysmo`.

1.1 Before you start

Save the install folder 'DySMo' to your desired destination path. The directory references are relative paths for the rest of the document. If there is a need to change some configurations «yourDirectory» is used. For example you copied the folder to $C : \backslash simulation \backslash modelica \backslash pythonFramework$ then you have to replace «yourDirectory» with $C : \backslash simulation \backslash modelica \backslash$.

1.2 Install Python

To install the *python-interpreter* under WindowsOS you can use the link in the software.txt file. It is strongly recommended to use the **Version 2.6**, the source code is not tested with other version. Therefore there is no guarantee that everything will work as expected. For further information about the python-interpreter, refer to the official Python projectpage `www.python.org`. Please also install all necessary packages from the software.txt. Please use the recommended versions, otherwise the framework might not work.

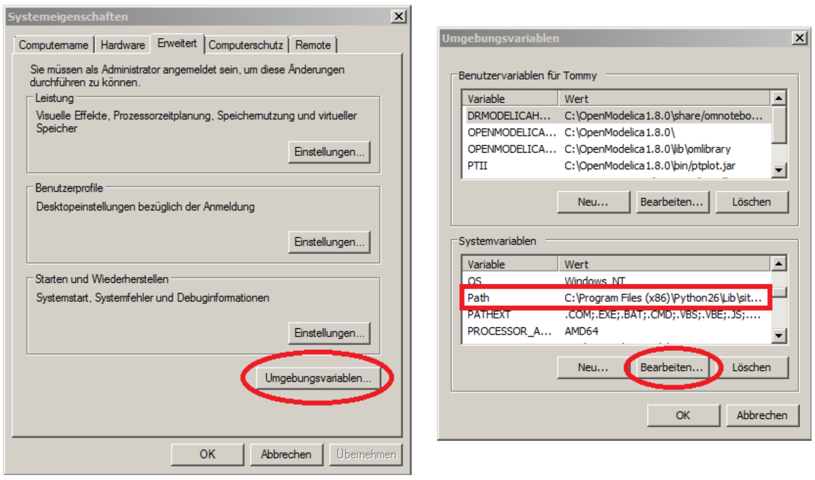


C:\Program Files (x86)\Python26

Name	Änderungsdatum	Typ	Größe
dres	09.08.2011 08:21	Microsoft Access Ta...	7 KB
DyMat.p	09.08.2011 08:05	PY-Datei	3 KB
DyMatNe	09.08.2011 17:52	PY-Datei	3 KB
DyMatNe	09.08.2011 18:14	PYC-Datei	5 KB
dymola.p	09.08.2011 21:59	PYC-Datei	4 KB
dymola_	09.08.2011 21:58	PY-Datei	3 KB
DymolaM	09.08.2011 08:05	PY-Datei	3 KB
LICENSE.txt	24.08.2010 18:50	Textdokument	40 KB
matplotlib-wininst.log	09.08.2011 10:08	Textdokument	298 KB
NEWS.txt	24.08.2010 16:51	Textdokument	181 KB
numpy-wininst.log	08.08.2011 20:01	Textdokument	121 KB
py2exe-wininst.log	05.12.2011 15:12	Textdokument	15 KB
PySide-wininst.log	10.10.2011 13:04	Textdokument	31 KB
python.exe	24.08.2010 18:47	Anwendung	26 KB
pythonw.exe	24.08.2010 18:48	Anwendung	27 KB
pywin32-wininst.log	09.08.2011 18:32	Textdokument	153 KB
qt.conf	10.10.2011 14:44	CONF-Datei	1 KB
README.txt	24.08.2010 16:51	Textdokument	55 KB
Removematplotlib.exe	09.08.2011 10:08	Anwendung	192 KB

Figure 1: Copy this **path** and paste it to your environment Variables

After the installation you have to set an environment variable, so that it is possible to run the python interpreter from any path. Therefore locate the file *python.exe*. If Python is installed in the default directory, you can find the file under $C : \backslash ProgramFiles(x86) \backslash Python26$, refer to figure 2. Copy this path and open the *environment variables setup (Systemcontrol->System->advanced->environment variables) (Systemsteuerung->System->Erweiterte Systemeigenschaften->Umgebungsvariablen)*, refer to figure 3. Find the variable **Path** and add the copied path from figure 2 separated by a semi-colon. If everything works, you can now open the command line (Eingabeaufforderung) and start the python interpreter. To test it, goto *Start->programs->accessories->execute (Start->Programme-*



Sie müssen als Administrator angemeldet sein, um diese Änderungen durchführen zu können.

Visuelle Effekte, Prozessorzeitplanung, Speichernutzung und virtueller Speicher

Desktopeinstellungen bezüglich der Anmeldung

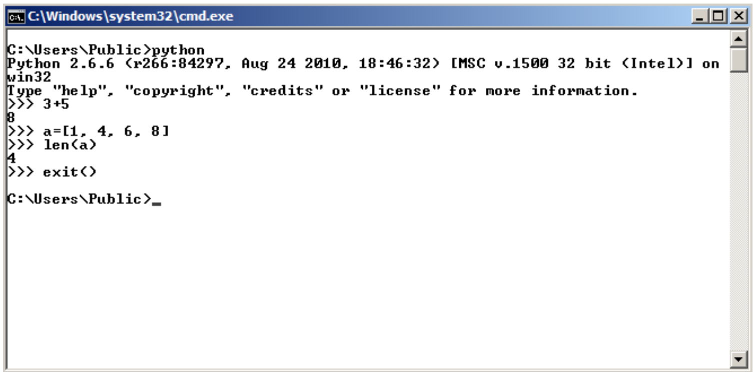
Systemstart, Systemfehler und Debuginformationen

Variable	Wert
DRMODELICAH...	C:\OpenModelica1.8.0\share/omnotebo...
OPENMODELICA...	C:\OpenModelica1.8.0\
OPENMODELICA...	C:\OpenModelica1.8.0\lib\omlibrary
PTII	C:\OpenModelica1.8.0\bin/ptplot.jar

Variable	Wert
OS	Windows_NT
Path	C:\Program Files (x86)\Python26\Lib\sit...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Figure 2: Edit the environment variable **Path**

>Zubehör->Ausführen) and type in **cmd**. You can now type in **python**, the interpreter should start and accept your commands, figure 4. So the framework works as expected all libaries from the *addOns* folder have to be installed, refer to figure 1.



```
C:\Users\Public>python
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+5
8
>>> a=[1, 4, 6, 8]
>>> len(a)
4
>>> exit()

C:\Users\Public>_
```

Figure 3: Open your console and test some python commands

1.3 Install Dymola

To install the modeling environment Dymola, please refer to the Dymola user´s manual («your-Directory»\ Documentation\ Dymola5Manual.pdf). Make sure you have a fully licensed version, with the demo version the following examples do not work. Furthermore the examples do mostly not work with a borrowed server license. For more information about Dymola visit `www.dymola.com`.

1.4 Install OpenModelica

You can also use OpenModelica to simulate your models. DySMo works with OpenModelica Version 1.8. A link to the installation for OpenModelica on Windows is also given in the software.txt, figure 1. For further information about OpenModelica, refer to the official project page `www.openmodelica.org`.

1.5 Install Matlab/Simulink

The Framework does also support Matlab/Simulink. To use Matlab/Simulink in the Framework Matlab/Simulink has to be installed on your computer. The Framework was tested with Matlab 2011a and was not fully tested with other versions, so the Framework might not work with other versions.

2 Getting started with the Framework

The Framework supports different modeling tools and an arbitrary number of mode switches. This chapter gives you a design overview and shows you how you have to prepare your models so that they can be used as variable-structure models in the given Framework. In the second part the focus is on how to run the delivered examples.

2.1 Design

The Framework is implemented through different objects which describe the variable-structure model. Figure 5 presents an oberview. The main files are located in «yourDirectory» \ python-Package \ source. For every new variable-structure-model you have to create a parameter file, <modelName.py>. The body of such a file is shown in chapter 2.3.

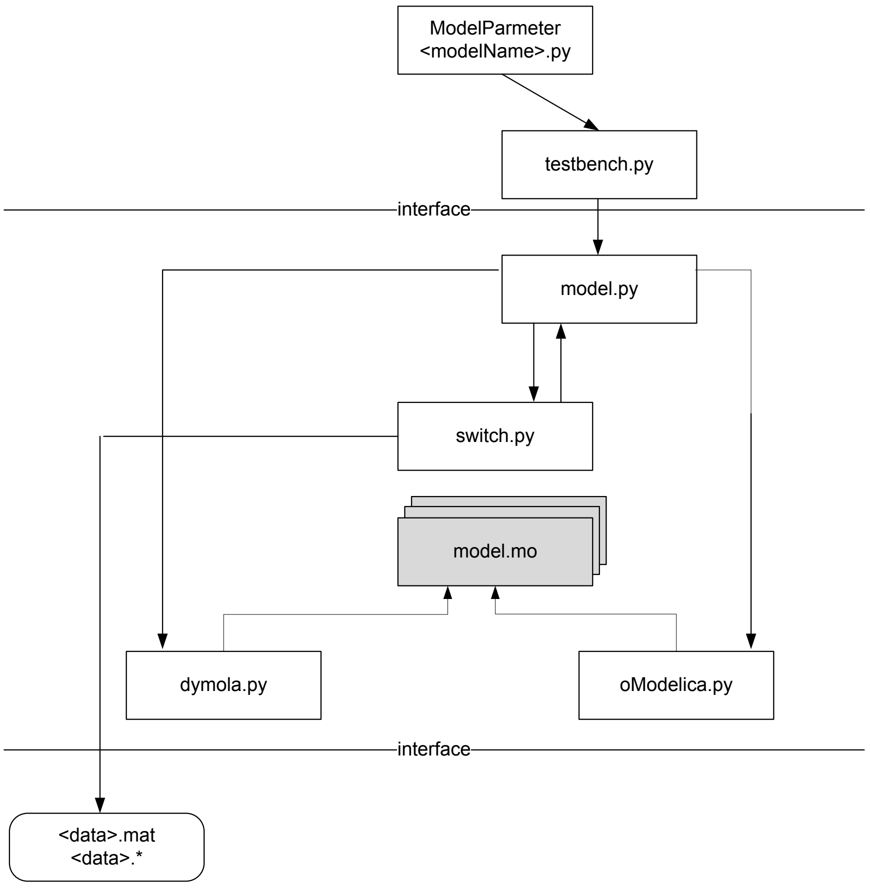


Figure 4: Static framework overview

This parameter file is given to the *testbench.py* which is the executable Python file which needs to be started. The actual variable-structure model inside the Python Framework is then build by the *testbench.py*. After building the variable-structure model the model is handed to the method *switch.py*. This method handles the actual mode switch, the initialization of the new mode and the saving of the simulaiton resultss of each simulation. In the following a few examples are introduced to show how the Framework can be used.

2.2 Case study: The extended bouncing ball

In this section a case study is designed to illustrate a simulation cycle using the Framework. You can find the same model implemented under «yourDirectory»\sample \ MechStruk.

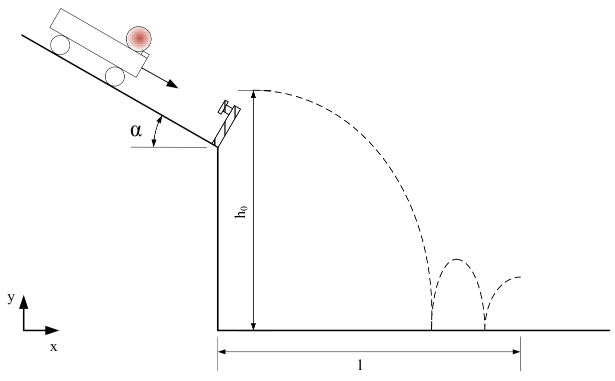


Figure 5: Physical representation: The extended bouncing ball

For the given model, three modes are needed (wagen_struc ID=1, contact_struc ID=2, ball_struc ID=3). Every mode has a terminate condition, for example the model terminates from wagon to ball at t_0 (mode 1->2). These conditions have to be specified in the corresponding .mo file (refer to «yourDirectory»\sample \ MechStruk \ mechanik.mo). Figure 6 shows the physical representation. In the model you also have to specify the number of the mode which follows the simulation of the actual mode. Therefore it is possible to have two transitions leading to different modes if necessary. A sequence diagram which shows the sequence of the simulation is shown in figure 7. (You can switch translation on / off in *testbench.py*, refer to chapter 2.4).

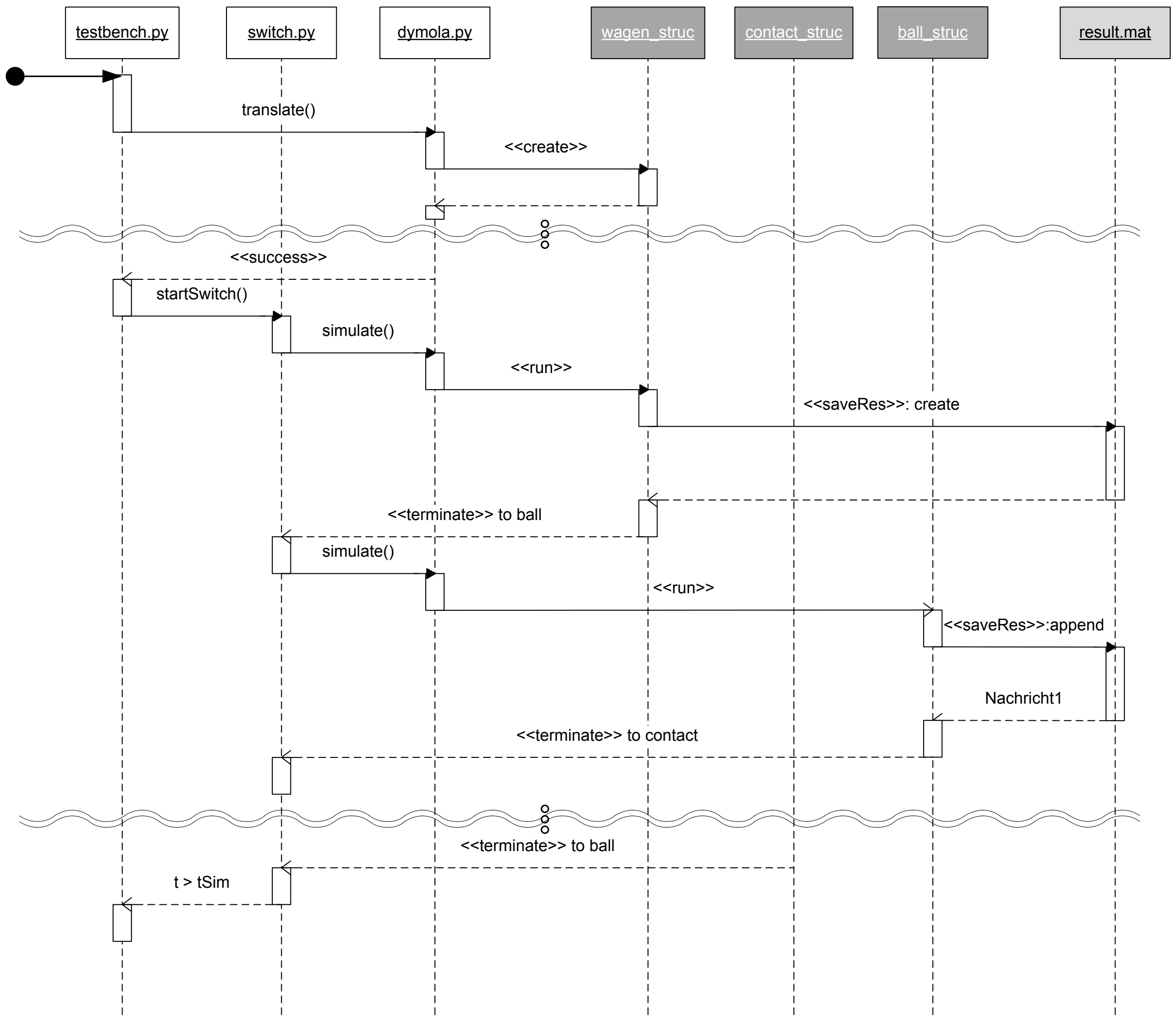


Figure 6: Sequence Diagram: The extended bouncing ball

2.3 Body of the parameter file

In this subsection the idea of the model parameter file is described. You can find the complete file in your folder *sample\mechStruk*. First you have to set your simulation settings. Parameters like Interval length, Number of Intervals, Tolerance, fixed stepsize and simulation time can be chosen in the first block, refer to listing 1. If you need further information for these parameters, refer to the Modelica users guide.

```
INT_LENGTH = 0     # interval lenght for saved data
INT_NUM = 500      # number of saved data
TOL = 3e-05        # tolerance of solver
FIXED = 0.01       # fixed step size
SIMTIME = 10       # simulation time
```

Listing 1: Simulation parameters

In the following block the path settings and the mode names have to be specified.

```
# Type in path and filename for the model here
FILEDIR = "..\Beispiele\MechStruk"
MOFILE = ["mechanik.mo", "mechanikOM.mo"]
RESULTFOLDER = "resultDy"

MODE1 = "mechanik.wagen_struc"
MODE2 = "mechanikOM.ball_struc"
MODE3 = "mechanik.contact_struc"
MODELNAMELIST = [MODE1, MODE2, MODE3]
TOOLS = [Environment.DYMOLA, Environment.OMODELICA, Environment.DYMOLA]
```

Listing 2: Path settings

Listing 2 shows the needed information. If you have more than one *.mo file with Modelica code (like in this example), separate them by a coma. The result folder can be freely chosen it will be generated during the simulation. Every mode needs a unique name. You can separate the names by points, like Modelica does it to disclose sub models for example. The modelnamelist must include every modename above (so that it is possible to recognize automatically how many modes are needed in the actual model). The Toollist has the same length as the modenamelist. You must specify which tool you want to use for which mode. In our example we simulate the wagon and the ground contact with Dymola. The free fall mode is simulated with open Modelica.
Now you have to specify, which variables from the simulation you like to save or plot. These two options are independent of each other. For the plot option you need a lists with pairs [x over y]. It is possible to plot more than one pair, for each list a separate plot will be generated.

```
#Which variables are you interested in?
# for saving use structures like: ['h', 'u']
outputVariablesToSave = []
#append for every mode the individual names to save
outputVariablesToSave.append(['x', 'y'])
# for plotting use: [['t','h'],['v','t'],['...','...']]
outputVariablesToPlot = [['t', 'y']]
```

Listing 3: Variables to plot and save

In listing 3 the variables x and y are saved and one plot y *over* t will be generated. Be aware that there is another option to rename the variable names in the different modes (Described in the following paragraph). For plotting and saving you have to use the variable name like in the first mode!
In the last step you have to specify each transition. In listing 4 first a relationship between the variable names in mode 1 and 2 was compounded. You have to do this for each variable you like to save or plot! Even if the name is the same (e.g. x) in each modes.

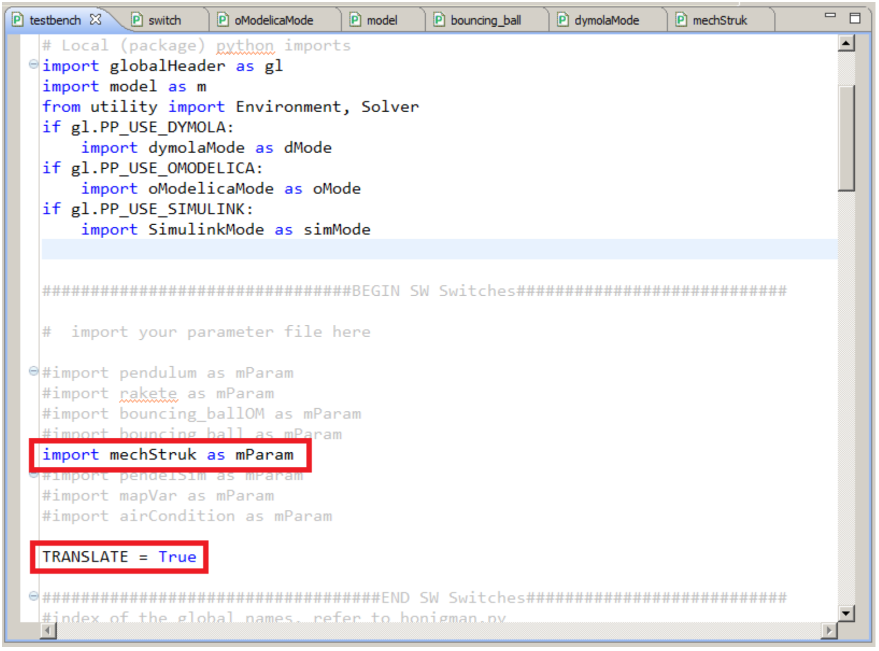
```
#mapping list for the variables
outList = ['x', 'y', 'der(x)', 'der(y)']
inList = ['x', 'h', 'vx', 'vy']

#create modes for testing
translist1_2 = mode.transition(2, 'conditionToSimulate', outList, inList)
outputVariablesToSave.append([])
```

Listing 4: Map Variables and create transistions

Then a transition (translist1_2) has to be specified. It consists the aim mode (in our case mode 2), a user defined string (you can type in everything here, to find your transition later), and the previous generated in- and outlist. Now you have to generate an entry in the *outputVariablesToSave* list with the local (mode-based) variable name from your variables you want to save. If the name is the same as in the previous mode an empty entry can be appended.

2.4 Simulate a given example

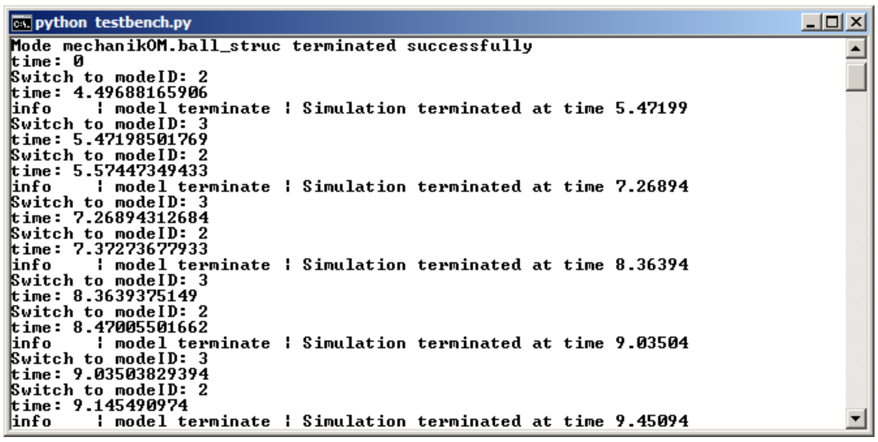
Now that all preconditions are fulfilled, we can devote to the main simulation. This subsection explains the last necessary settings before you can run python and get the results. Note there are more possibilities to start the scripting via the interface, refer to figure 5. This subsection shows how to start it via the *testbench.py*.
Open the file *testbench.py* and uncomment the parameter file mechStruk, figure 8. Make sure that all other entries are commented. You can uncomment them later to get familiar with the framework. Set the *TRANSLATE*-flag to *True*, this ensures that your model will be compiled and all necessary folders will be created.



```
# Local (package) python imports
import globalHeader as gl
import model as m
from utility import Environment, Solver
if gl.PP_USE_DYMOLA:
    import dymolaMode as dMode
if gl.PP_USE_OMODELICA:
    import oModelicaMode as oMode
if gl.PP_USE_SIMULINK:
    import SimulinkMode as simMode


################################BEGIN SW Switches############################

#  import your parameter file here

#import pendulum as mParam
#import rakete as mParam
#import bouncing_ballOM as mParam
#import bouncing_ball as mParam
import mechStruk as mParam
#import pendelSim as mParam
#import mapVar as mParam
#import airCondition as mParam

TRANSLATE = True

###################################END SW Switches###########################
#index of the global names, refer to honigman.py
```

Figure 7: Settings in testbench.py

Now open the console, (as shown in figure 4) and start the scripting with **python testbench.py**. If everything is alright, Dymola starts and translate your model. Also open Modelica translate a mode in background. After few seconds you should be able to pursuit the simulation process, as shown in figure 9.
Another few seconds later a plot with the results should open, like in figure 10. The different colour indicates the origin mode from the data.
Now you can try to change the simulation parameters, maybe plot *x over y*. Now you are also able to simulate the other delivered examples.



```
Mode mechanikOM.ball_struc terminated successfully
time: 0
Switch to modeID: 2
time: 4.49688165906
info     | model terminate | Simulation terminated at time 5.47199
Switch to modeID: 3
time: 5.47198501769
Switch to modeID: 2
time: 5.57447349433
info     | model terminate | Simulation terminated at time 7.26894
Switch to modeID: 3
time: 7.26894312684
Switch to modeID: 2
time: 7.37273677933
info     | model terminate | Simulation terminated at time 8.36394
Switch to modeID: 3
time: 8.3639375149
Switch to modeID: 2
time: 8.47005501662
info     | model terminate | Simulation terminated at time 9.03504
Switch to modeID: 3
time: 9.03503829394
Switch to modeID: 2
time: 9.145490974
info     | model terminate | Simulation terminated at time 9.45094
```

Figure 8: Information during the simulation process

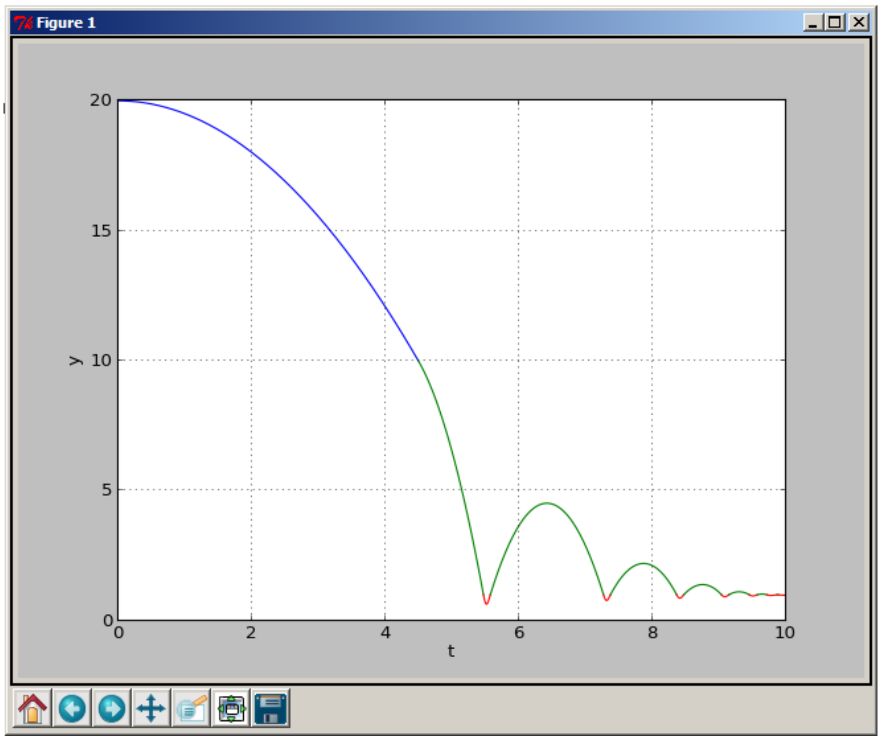


Figure 9: Graphical representation of the simulation results

List of Figures