Direct Implementation of ARGESIM Benchmark C7 'Constrained Pendulum' in MATLAB and EXCEL

Anna E. Stockinger[1], Elisabeth Gütl[1], Stefan A. Rath[1], Dominik Strasser[1], Martin Bicher[2*], Andreas Körner[1], Horst Ecker[3*]

[1]Mathematical Modelling and Simulation Group, Inst. of Analysis and Scientific Computing
[2]Inst. of Information System Engineering, [3]Inst. of Mechanics and Mechatronics
TU Wien, Wiedner Hauptstrasse 8-10, 1040 Vienna, Austria; *horst.ecker@tuwien.ac.at*

SNE 29(2), 2019, 105 - 110, DOI: 10.11128/sne.29.bne07.10478
Received: October 25, 2018; Revised: January 13, 2019;
Revised: May 10, 2019; Accepted: May 30, 2019
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna
ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. This Benchmark Report with educational aspects presents a straightforward and direct implementation of ARGESIM Benchmark C7 'Constrained Pendulum' in MATLAB and in EXCEL: the given models are implemented without any change, and the tasks are directly simulated without any rearrangement. Central issue of this benchmark is the detection and handling of a state event: when the pendulum hits or releases a pin, pendulum length and angular velocity are changing discontinuously.
The MATLAB approach makes use of the event termination feature of the ODE solvers, and a MATLAB script loops between long and short pendulum and handles the event changes. The EXCEL approach solves the overall ODEs by Euler algorithm (a simple EXCEL recursion). The events are synchronized with the chosen time step (detection with delay), and handled by distinction of cases in any state update (no event, hit, release, velocity jump). The MATLAB implementation is straightforward but makes use of different models necessary. The EXCEL implementation shows that a spreadsheet tool – not really designed for simulation - can do simulation by direct implementation of ODE algorithms, but event-handling causes elaborate case-by-case analysis.

Introduction

ARGESIM Benchmark C07 'Constrained Pendulum' requires the simulation of a pendulum that hits and leaves a pin at a certain angular position. Figure 1 shows the schematic structure of the pendulum system. The description of the pendulum is generally given by a nonlinear ODE or a linear (approximating) ODE of second order:

$$m \cdot l \cdot \ddot{\varphi}(t) = -m \cdot g \cdot \sin\varphi(t) - d \cdot l \cdot \dot{\varphi}(t)$$

$$m \cdot l \cdot \ddot{\varphi}(t) = -m \cdot g \cdot \varphi(t) - d \cdot l \cdot \dot{\varphi}(t)$$

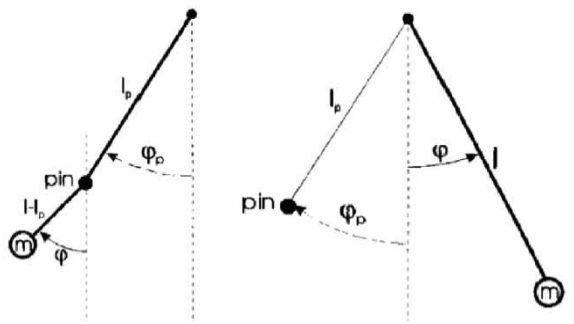

Figure 1: Pendulum hitting a pin.

The angular movement $\varphi(t)$ (measured in radians) is positively counted from the vertical position counterclockwise. Given parameters are pendulum length l, pendulum mass m, damping factor d, the angular position of the pin φ_{pin}, and initial values for angular movement φ_0 and angular velocity $\dot{\varphi}_0$.

If φ_0 or $\dot{\varphi}_0$ is big enough, the pendulum hits the pin if the event function $e(t)$ becomes zero:

$$e(t) = \varphi(t) - \varphi_{pin} = 0.$$

The hit is modelled as state event, changing pendulum length l and angular velocity $\dot{\varphi}$ discontinously.

At hit, the pendulum length shortens to $l_s = l - l_p$, and the angular velocity jumps (increase):

$$\dot{\varphi} \rightarrow \frac{l}{l_s} \cdot \dot{\varphi}$$

After some time the pendulum leaves the pin, causing the 'reverse' event: the pendulum length changes back to l, and the angular velocity jumps at release again (decrease):

$$\dot{\varphi} \rightarrow \frac{l_s}{l} \cdot \dot{\varphi}$$

The tasks of the benchmark are i) time domain simulations with proper detection and handling of the hit and release events, with varying parameters, ii) comparison of nonlinear and linear model description and impact on hit and release events, and iii) a boundary value problem.

These tasks provide different challenges for the MATLAB implementation, and for the EXCEL implementation. Because the solution approach follows a straightforward implementation, the detection and the handling of the hit and release events require special consideration.

1 MATLAB Implementation

The MATLAB ODE solvers require an explicit state space description $\dot{\vec{y}} = \vec{f}(\vec{y}, t)$, which gives for the pendulum description with $y_1(t) = \varphi(t), y_2(t) = \dot{\varphi}(t)$ the following equations:

$$\dot{y}_1 = y_2 = yder_1$$
$$\dot{y}_2 = -\frac{g}{l}\sin y_1 - \frac{d}{m}y_2 = yder_2$$

The MATLAB ODE solvers need these equations as function:

```
function yder = ODE_pendulum(t,y,m,l,d,g)
yder=[y(2);-g*sin(y(1))/l-d/m*y(2)]; end
```

or in linear version

```
function yder = ODE_pendulum_lin(t,y,m,l,d,g)
yder=[y(2);-g*sin(y(1))/l-d/m*y(2)]; end
```

1.1 Task a: Time Domain Analysis – MATLAB Implementation

Since the pendulum swings because of hit and release of the pin in the original length *l* and in the shortened length l_s, a permanent change between two differential equations must take place. This was solved by a *while loop* in combination with an *event function* characterized by the position of the pin, which terminates the ODE solver at hit or at release. With the *event function* it is possible to toggle between the differential equations by interrogating the momentary angle of the pendulum.

In the next listing the *event function* is shown as code: the event occurs when the current angle equals the angle of the pin (**position** equals zero); **isterminal = 1** means that the differential equation is only executed until the first occurrence of the event; **direction = 0** means that it does not matter which side of the pin angle the event originates from:

```
function [position,isterminal,direction] =
event_pendelum(t,y,pinangle)
position=y(1)-pinangle;
isterminal=1;
direction=0; end
```

For solving the differential equations, MATLAB's *ode45* solver (standard) is used. The solver is called in a loop (terminating with given simulation time), which toggles between long and short pendulum. After the *event function* has terminated the ODE solver at the event time, the event is handled: change of length, change of angular velocity; then the solution is concatenated to the previous ones. The MATLAB code is:

```
stopevent = odeset('Events', @(t, y) Event_pendulum
                                    (t, y, pinangle));
if(startingangle>pinangle)
  %Pendulum starts in the long state
  [t,y,te,ye,ie]=ode45(@(t,y) DGL_pendulum(t,y,m,l,d,g),
   [0 10],[startingangle;Initialangularvelocity],stopevent);
else
  %Pendulum starts in the short state
  [t,y,te,ye,ie]=ode45(@(t,y) DGL_pendulum(t,y,m,l2,d,g),
  [0 10],[startingangle;Initialangularvelocity],stopevent);
  i=2;
end
while t(end) < 10;  %as long as the end time not reached
 if(mod(i,2) ~= 0);  %i = odd -> short pendulum
  [t2,y2,te,ye,ie]=ode45(@(t2,y2)  ODE_pendulum (t2,y2,m,l2,
          d,g), [t(end) 10],[y(end,1);y(end,2)*l/l2],stopevent);
 else       %i  = even -> long pendulum
  [t2,y2,te,ye,ie]=ode45(@(t2,y2) DGL_pendulum(t2,y2,m,l
         ,d,g),[t(end) 10],[y(end,1);y(end,2)*l2/l],stopevent);
  end
  y=vertcat(y,y2);    t=vertcat(t,t2);    i=i+1;
end
```

This implementation works independent of position of pin and of initial angle, so that simulations for the two different initial angles can be performed (*Task a1* and *Task a2*). Section 3 shows results of the simulations, in comparison with the results from the EXCEL implementation results.

1.2 Task b: Comparison of Nonlinear and Linear Model – MATLAB Implementation

For *Task b: Comparison of Nonlinear and Linear Model*, simply the simulation is repeated with the linear model, using the same MATLAB script as for *Task a: Time Domain Analysis* but with changed derivative function.

The *ode45* solver performs stepsize control, so that results for nonlinear and linear model are calculated at different grid points, and the number of grid points differs. A comparison can be done simply graphically – for a numerical comparison of results, interpolation of both results must be used (graphical results in Section 3).

1.3 Task c: Boundary Value Problem – MATLAB Implementation

Task c: Boundary Value Problem requires to reach a target angle of the pendulum ($-\pi/2$), by proper choice of the initial angular velocity $\dot{\varphi}_0$. For this task a line search is implemented, which varies the initial angular velocity with variable increments, until the desired angular position is reached.

For controlling the search, a deviation is defined, the difference between the target angle and the maximally reached angle of the shortened pendulum. Depending on the current deviation, the initial angular velocity is increased or decreased with respect to the deviation size, unless the deviation is smaller than allowed. The following code is self-explanatory:

```
while(perdeviation<dev)
 if(startingangle>pinangle); % starts in  long state
  [t,y,te,ye,ie]=ode45(@(t,y) DGL_pendulum(t,y,m,l,d,g),
    [0 10],[startingangle;Initialangularvelocity],stopevent);
    [t2,y2,te,ye,ie]=ode45(@(t,y) DGL_pendulum(t,y,m,l2 ,d,g),
    [t(end) 10],[y(end,1);y(end,2)*l/l2],stopevent);
      y=vertcat(y,y2);  t=vertcat(t,t2);
 else   %Pendulum starts in short state
    [t,y,te,ye,ie]=ode45(@(t,y) DGL_pendulum(t,y,m,l2,d,g),
   [0 10],[startingangle;Initialangularvelocity*l/l2],stopevent);
 end
 maxangle= min(y(:,1));%until pendulum max turns
   dev= abs(targetangle-maxangle);
   if(dev>0.5)   delta=0.1;
   elseif (dev<=0.5 & dev>0.1)  delta=0.01;
   elseif (dev<=0.1 & dev>0.01) delta= 0.001;
   else  delta=0.0001;  end
   if(targetangle<maxangle)
      Initialangularvelocity=Initialangularvelocity-delta;
   else
      Initialangularvelocity=Initialangularvelocity+delta;
   end;    steps=steps+1; end
```

This MATLAB script results in an initial angular velocity of $\dot{\varphi}_0 = -2{,}187$ rad/s in order to reach the angle $-\pi/2$ after one hit. Graphical results are given in Section 3.

2 EXCEL Implementation

EXCEL is not really a simulator, it is mainly used in the area of simulation in economics, also for dynamic processes. 'Basic' EXCEL does not offer ODE solvers - but the spreadsheet structure allows an easy implementation of simpler ODE solvers, as Euler solver, or Heun solver with fixed appropriate small step size: columns calculate time advance and state advance in a recursive manner.

Because of the fixed stepsize, state events cannot be localized – they can only be detected at the next timestep after the event has happened. The handling of the event causes complicated if -then – else constructs, depending on the quality of the event – in this case two actions with the change of the parameter length – easy – and with the jump of the angular velocity – complicated.

2.1 Euler Solver for Pendulum Equations

This contribution makes use of the EULER ODE solver, in state space notation $\dot{\vec{y}} = \vec{f}(\vec{y}, t)$ generally given by

$$\vec{y}_{n+1} = \vec{y}_n + h \cdot \vec{f}(\vec{y}_n, t_n), \;\; t_{n+1} = t_n + h$$

With $y_1(t) = \varphi(t), y_2(t) = \dot{\varphi}(t)$ the state space

$$\dot{y}_1 = f_1(y_1, y_2) = y_2$$
$$\dot{y}_2 = f_2(y_1, y_2) = -\frac{g}{l}\sin y_1 - \frac{d}{m} y_2$$

results in the following Euler algorithm for solving the ODEs:

$$y_{1,n+1} = y_{1,n} + h \cdot f_1(y_{1,n}, y_{2,n}) = y_{1,n} + h \cdot y_{2,n}$$
$$y_{2,n+1} = y_{2,n} + h \cdot f_2(y_{1,n}, y_{2,n}) =$$
$$= y_{2,n} + h \cdot \left(-\frac{g}{l}\sin y_{1,n} - \frac{d}{m} y_{2,n}\right)$$

2.2 Euler Solver with Event Handling

Together with the state space update due to Euler algorithm, in each time step t_{n+1} also the event function $e(t) = \varphi(t) - \varphi_{pin}$ must be checked and compared with the previous time step:

$$e_n = \varphi_n - \varphi_{pin} = y_{1,n} - \varphi_{pin}$$
$$e_{n+1} = \varphi_{n+1} - \varphi_{pin} = y_{1,n+1} - \varphi_{pin}$$

If the event function has changed sign, then the event has happened between t_n and t_{n+1}, and the quality of the sign change determines the event: hit or release.

The event must be handled now at t_{n+1}. As for t_{n+1} all time and state values are already calculated, the event changes must be considered at the next integration step

$$t_{n+2}, y_{1,n+2}, y_{2,n+2}$$

which now makes uses of a changed length (and also the further steps) and which must make use (only once) of the jump in the angular velocity, i.e. in case of hit

$$\dot{\varphi}_{n+1} = y_{2,n+1} \rightarrow \frac{l}{l_s} \cdot \dot{\varphi}_{n+1} = \frac{\dot{l}}{l_s} \cdot y_{2,n+1}$$

As consequence, the handling of the event requires actions within three timesteps – resulting in complicated case-by-case analysis of the status of the states in each time step. The update equation for the angular velocity

$$y_{2,n+1} = y_{2,n} + h \cdot \left(-\frac{g}{l}\sin y_{1,n} - \frac{d}{m} y_{2,n}\right)$$

is extended by nested *if – then- else* calculations, especially because of the singular jump of the angular velocity.

2.3 Event Handling in EXCEL

There are several ways to implement the complicated case-by-case analysis for the events *hit* and *release*. In any case, the implementation is based on EXCEL's *if-then-else* formula for calculating the value in a cell:

cellvalue **=** **IF(***condition***, THEN-***formula***, ELSE-***formula***);**

For proper implementation of the changes at events *hit* and *release*, these *if-then-else* formulas must be nested, i.e. **THEN-***formula* and **ELSE-***formula* are themselves *if-then-else* formulas. For documentation, the order of the *if*-formulas is identified by a number, i.e. **IF1**, **IF2**, **and IF3**.

The first check **IF1** asks if the angle of the pendulum φ at time t is greater than the pin angle φ_{pin}. If *true* – **THEN-***formula* of **IF1** – another *if*-formula **IF2** is necessary. *if*-formula **IF2** decides, that if the angle of the pendulum φ is less than the pin angle φ_{pin} at the time $t - h$ (checking **IF2**), then the extended update equation for the angular velocity (with change of length to l and with jump of angular velocity with factor l_s/l) is used; if this is not the case (φ greater than φ_{pin} at $t - h$) the standard update equation for the Euler update of the angular velocity with length l is used. The **ELSE-***formula* of **IF1** is another *if*-formula **IF3**. It decides that if the angle of the pendulum φ is greater than the pin angle φ_{pin} at the time $t - h$ (checking **IF3**), then the extended update equation for the angular velocity with shortened length l_s and with jump of angular velocity with factor l/l_s) is used (**THEN-***formula* of **IF3**); if this is not the case, the standard update equation for the Euler update of the angular velocity with length l_s is used **ELSE-***formula* of **IF3**.

The combined *if*-formulas **IF1**, **IF2**, and **IF3** now setup the overall update formula for the angular velocity:

```
= IF ( H16 > phi_p, IF ( H15 < phi_p,
        ls/l * G16 * (1-h*d/m) – h * g/l* SIN(H16),
        G16*(1-h*d/m)-h*g/l*SIN(H16)],
    IF ( H15  >phi_p,
        l/ls * G16  *(1-h*d/m) - h*g/ls * SIN(H16),
       G16 * (1-h*d/m) – h * g/ls * SIN(H16)))
```

Here, the angular velocity at the time step t is calculated in cell **G17**; **H16** is the angle value of the pendulum at the time step $t - h$ and **H15** the angle value at $t - 2h$. **G16** is the value of the angular velocity at the time step before $(t - h)$.

2.4 Task a: Time Domain Analysis – EXCEL Implementation

The implementation follows the classical spreadsheet usage – time update and update of states in columns defined by recursive formulas, and parameters are defined by names, for better understanding of formulas.

Figure 2 shows definitions for the parameters, and Figure 3 sketches the structure of the spreadsheet with columns for time – step size 0.001 –, angle, and angular velocity.

EXCEL allows graphical representations of various kinds; result graphs can be easily produced from the result columns – results see Section 3.

	A	B	C
1	**CONSTRAINED PENDULUM**		
2			
3	DGL:		
4	phi''=	=-g*sin(phi)/l-d/m*phi '	
7	PARAMETER:		
8	g	gravity	9,81
9	m	mass of the pendulum	1,02
10	l	length of the pendulum	1
11	ls	shortend length of the pendulum	0,3
12	d	damping factor	0,2

7	INITIAL CONDITIONS:		
8	phi_0	starting angle	0,5235988
9	phi'_0	starting angular velocity	0
10	phi_P	angle of the pin	-0,262

Figure 2: Definition of parameters – EXCEL implementation.

	D	E	F	G	H
13					
14					Expl. Euler
15		time		phi'	phi
16		0		0	0,5235988
17		0,001		-0,004905	0,5235988
18		0,002		-0,00980904	0,5235939
19		0,003		-0,01471207	0,5235841
20		0,004		-0,01961406	0,5235693
21		0,005		-0,02451497	0,5235497
22		0,006		-0,02941474	0,5235252
23		0,007		-0,03431335	0,5234958
24		0,008		-0,03921075	0,5234615
25		0,009		-0,04410689	0,5234223
26		0,01		-0,04900175	0,5233782

Figure 3: EXCEL spreadsheet structure for state updates.

2.5 Task b: Comparison of Nonlinear and Linear Model – EXCEL Implementation

For *Task b: Comparison of Nonlinear and Linear Model*, simply two further columns are added, calculating the linear equations, using same case-by-case analysis as with the nonlinear equations.

Figure 4 sketches the structure of the spreadsheet with –for time – step size 0.001 –, nonlinear results (angle, angular velocity), and linear results (angle, angular velocity). The fixed step size allows a direct calculation of the deviation between nonlinear and linear results, shown in Figure 4 with columns for deviations of angle and angular velocity.

			Expl. Euler						
time		phi'	phi		phi'_lin	phi_lin		dev phi'	dev phi
0		0	0,26179939		0	0,26179939		0	0
0,001		-0,002539015	0,26179939		-0,00256825	0,26179939		-0,00253901	0
0,002		-0,005077781	0,26179685		-0,00513625	0,26179682		-0,00507778	2,9237E-08
0,003		-0,007616274	0,26179177		-0,00770398	0,26179168		-0,00761627	8,7709E-08
0,004		-0,01015447	0,26178415		-0,0102714	0,26178398		-0,01015447	1,7541E-07
0,005		-0,012692345	0,261774		-0,01283849	0,26177371		-0,01269234	2,9234E-07
0,006		-0,015229875	0,26176131		-0,01540523	0,26176087		-0,01522987	4,3848E-07
0,007		-0,017767035	0,26174608		-0,0179716	0,26174546		-0,01776704	6,1384E-07
0,008		-0,020303803	0,26172831		-0,02053756	0,26172749		-0,0203038	8,184E-07
0,009		-0,022840154	0,26170801		-0,02310309	0,26170696		-0,02284015	1,0522E-06
0,01		-0,025376064	0,26168517		-0,02566817	0,26168385		-0,02537606	1,3151E-06
0,011		-0,027911508	0,26165979		-0,02823277	0,26165818		-0,02791151	1,6072E-06
0,012		-0,030446464	0,26163188		-0,03079687	0,26162995		-0,03044646	1,9285E-06

Figure 4: EXCEL spreadsheet structure for state updates: results nonlinear model, results linear model, and deviation.

2.6 Task c: Boundary Value Problem – MATLAB Implementation

EXCEL provides as standard feature the *Goal Seeking Function* in the *What If Analysis* – suitable for approximating the initial value for angular velocity $\dot{\varphi}_0$, with goal reaching an angle of pendulum ($\pi/2$).

The *Goal Seeking Function* needs as input the cell of the parameter to be iterated – here $\dot{\varphi}_0$ in cell **I9**, the goal function evaluation – here the maximal angle after one hit in cell **I14**, and the goal value – here given with $\pi/2$ in cell **I13**. Additionally an accuracy parameter can be given – here 0.01 in cell **I14** (Figure 5).

	E	F	G	H	I
6					
7	INITIAL CONDITIONS:				
8	phi_0	starting angle			0,5235988
9	phi'_0	starting angular velocity			-2,17
10	phi_P	angle of the pin			-0,262
11					
12	maximum allowable deviation				0,01
13	required maximum angle of the pendulum				**-1,5707963**
14	measured maximum angle of the pendulum				-1,5684524

Figure 5: EXCEL goal seeking function – parameters for boundary value problem for initial angular velocity.

After start of the *What If Analysis,* EXCEL performs an optimizing search for the initial angular velocity, performing several simulation runs with changing values for the initial angular velocity.

Figure 5 shows the input cells for the *What If Analysis* and the results for initial angular velocity $\dot{\varphi}_0 = -2.17$ in cell **I9** and the reached goal angle φ_{end} in cell **I14** ($\varphi_{end} = -1.5684524 \sim -\frac{\pi}{2} = -1.5707963$). Graphical results for the solution are shown in Section 3.

An alternative is use of an EXCEL macro, which changes the initial velocity similar to the controlled line search in the MATLAB implementation.

3 Results – MATLAB and EXCEL

In the following graphical results from the three tasks for the MATLAB implementation and for the EXCEL implementation are shown and commented.

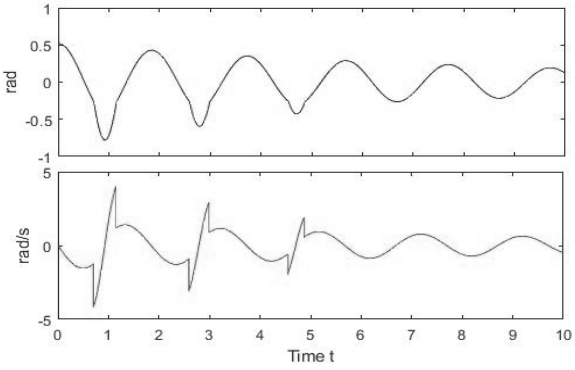

Figure 6: Simulation results for Task a1 – MATLAB.

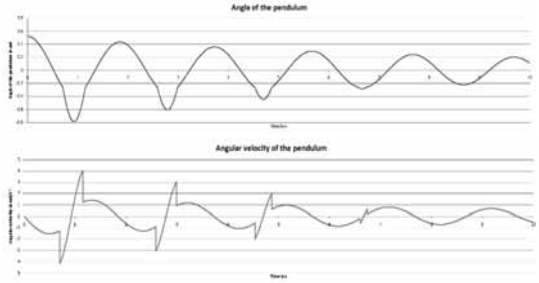

Figure 7: Simulation results for Task a1 – EXCEL.

At first glance, MATLAB results and EXCEL results for *Task a1* look the same. But the EXCEL result comes along with four hits, whereas the MATLAB implementation detects only three hits. It is evident from other solutions, that a fourth event pair *hit – release* exists.

Curiously MATLAB fails, although MATLAB makes use of a much more accurate ODE solver with step size control. Here MATLAB outmanoeuvres itself: the step size control choses because of high order a relatively big step size, so that the very close fourth event pair *hit – release* simply is not recognized (between one step both

events take place). As prevention, the ODE solver parameters must be better tuned.

For *Task b: Comparison of Nonlinear and Linear Model*, the MATLAB implementation repeats the simulation with the linear model. The *ode45* solver performs step size control, so that results for nonlinear and linear model are calculated at different grid points, additionally the total number of grid points differs.

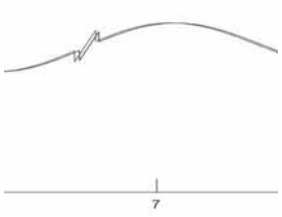

Figure 8: Comparison of linear and nonlinear model with zoom – MATLAB.

But both results can be plotted over the same time scale and look the same; only a zoom-in allows to recognize differences for later time values (Figure 8).

For a numerical comparison of results, interpolation of both results must be used.

The EXCEL implementation calculates nonlinear and linear model in parallel. The fixed step size allows a direct calculation of the deviation between nonlinear and linear results, shown in Figure 4 with columns for deviations of angle and angular velocity.

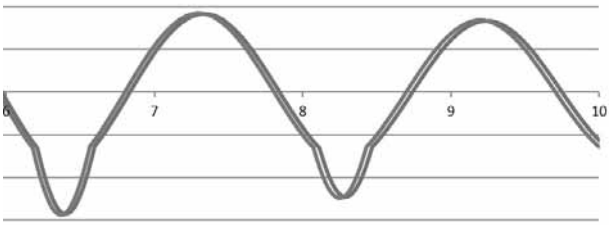

Figure 9: Comparison of linear and nonlinear model with zoom – EXCEL.

The EXCEL implementation shows bigger differences, because of less accuracy – see Figure 9.

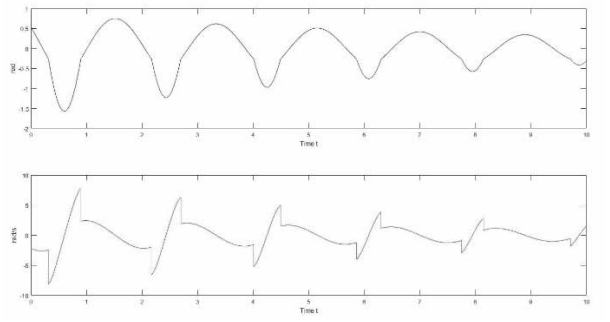

Figure 10: Time course of angle and angular velocity for solution of Task c: Boundary Value Problem - MATLAB.

Task c: Boundary Value Problem requires to reach a target angle of the pendulum ($-\pi/2$), by proper choice of the initial angular velocity $\dot{\varphi}_0$. Both implementations work with an iterative approach to determine the initial angular velocity, with sufficient similar results: MATLAB gives $\dot{\varphi}_0 = -2.187$, and EXCEL gives $\dot{\varphi}_0 = -2.17$. For completeness, Figure 10 shows the time course of angle and angular velocity for solution of the boundary value problem

4 Conclusion

A spreadsheet tool as EXCEL is definitely not a simulator – modelling features for ODEs, processes, events, etc. are missing. But spreadsheet programs are an excellent experiment environment with statistical analysis, optimisation, what-if analysis, date handling, etc. Of course, macros and external programming could be used, but to some extent the standard features allow to implement this benchmark with sufficient accuracy, using explicit Euler integration.

The crucial task in the EXCEL implementation is the handling of events. Events must be realized by elaborate nesting IF-formulas. As a result, the entire algorithmic model is complicated and lacks clarity.

MATLAB is a classical programming and simulation tool, and allows quick solutions in a standardized and comfortable manner. The MATLAB ODE solvers allow event functions, which terminate the integration, the overall model must be put together in a loop. Nevertheless, parameters for ODE solver and for event functions must be properly tuned.

Both implementations produce quite similar results. MATLAB allows an increase of accuracy (parameters of ODE solvers), EXCEL is limited in choice of step size, because each integration step adds a now row into the spreadsheet (here about 12000 rows).

It was generally the intention to compose a direct implementation, without model reformulation, without toolboxes or macros. On the other side, simple reformulation would allow much easier event handling, especially in EXCEL. If instead of the angular velocity $\dot{\varphi}$ the tangential velocity $v = l \cdot \dot{\varphi}$ is used as state variable, at the events *hit* and *release* the (tangential) velocity remains unchanged; event handling is then simply switching between different values for pendulum length. And also, the boundary value problem can be avoided. Reaching exactly the angle $-\pi/2$ implies, that the pendulum must swing back; this happens only, if the angular velocity is zero for angle $-\pi/2$. As consequence, an initial value problem with reverse time can replace the boundary value problem.