# FMNN05: Simulation tools

Ludwig Jacobsson, Julia Nilsson, Fredrik Norrman Brange, Robin Somers

December 2013

## 1 Introduction

Many engineering problems has for a long time and will continue to require advanced simulation tools in order to obtain accurate enough solutions. For some problems it is as easy as entering the problem into the software tool and press solve. Many problems, however, requires a more careful approach. Unfortunately it is not often very clear when that is the case and even when such a case is identified it might be difficult to know what needs to be done. It is therefore important to have experience when confronted with such problems which is what this course has given through three different projects.
This report details the results of all three projects which consisted of dealing with a highly oscillatory systems, differential algebraic equations for a multibody system and a discontinuous system. The simulation tools used and studied were Assimulo using Python and the commercial tool Dymola. The projects are presented disjointly in order with a final discussion at the end of the report spanning all projects.

## 2 Project 1 - Highly oscillatory problems

### 2.1 Problem Introduction

For the first part of the project the elastic spring pendulum example will be examined for which the ordinary differential equations are unconstrained. Different methods and softwares are used to primarily investigate the notion of order and stability. The problem is given by the following equations with $\lambda(y_1, y_2) = k\frac{\sqrt{y_1^2+y_2^2}-1}{\sqrt{y_1^2+y_2^2}}$:

$$\dot{y_1} = y_3 \quad (1)$$
$$\dot{y_2} = y_4 \quad (2)$$
$$\dot{y_3} = -y_1\lambda(y_1, y_2) \quad (3)$$
$$\dot{y_4} = -y_2\lambda(y_1, y_2) - 1 \quad (4)$$

Here $k$ is the spring constant and is of interest to this study. Small values of $k$ will correspond to an elastic spring for which we expect to see slow oscillations in the radial direction. As $k$ is increased the spring should become more and more stiff which in turn should give higher frequency oscillations. Therefore, for really large values of $k$ the problem itself will become stiff in the numerical sense

which should become hard to solve for some choices of integrators, in particular for explicit methods.
The goal of this part of the project is to study the performance of different methods and softwares ranging from simple implementations of Eulers method to commercial software. The main parameter of interest is the spring constant $k$ but other things will be considered as well such as the tolerance parameters for CVODE.

## 2.2 Results

The softwares used for the tests were the Assimulo package for Python using both own code and its included methods (CVODE in particular) as well as the commercial program Dymola.

### 2.2.1 BDF + Euler (Assimulo)

The first methods that we consider are simple implementations of the Euler method and 2:nd, 3:rd and 4:th order BDF. They all use a constant stepsize and are coded to be run using *simulate* in the Assimulo package for Python. Below are plots with increasing $k$ to visualize the decreasing accuracy of the integrators.

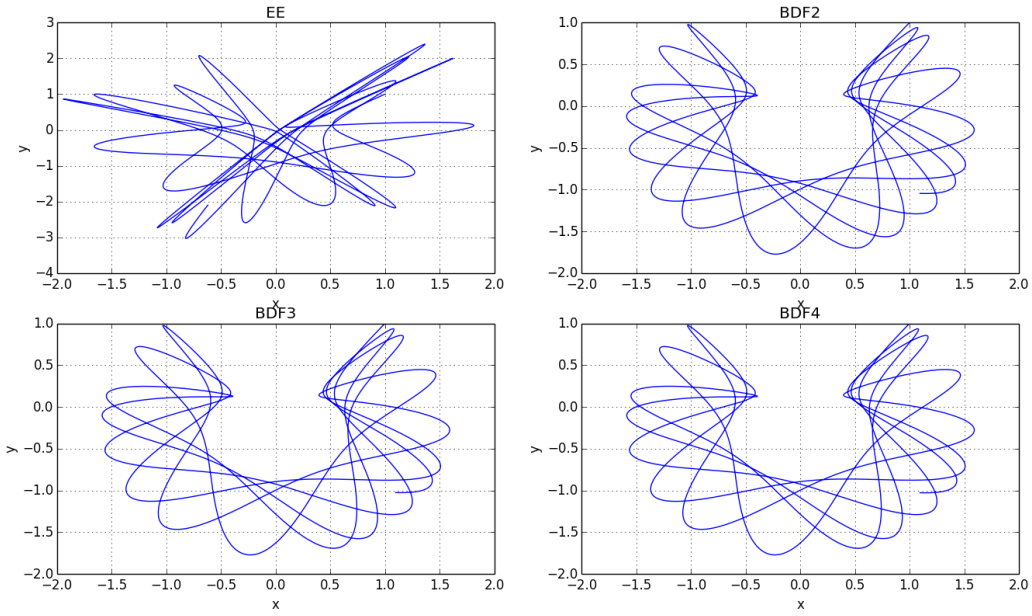



Figure 1: $k = 10, t = 30, init = 1, 1, 0, 0$, stepsize $= 0.01$

In *figure 1* the BDF-methods seems to handle themselves fairly well while Eulers method is already clearly struggling. It should be noted that for considerably lower values for $k$ even Eulers method manages to produce a good solution. In addition with decreasing stepsize Euler produces more accurate simulations even for moderately sized $k$'s. However it is of more interest to increase the frequency of the oscillations by increasing $k$ as in *figure 2 - 4*.

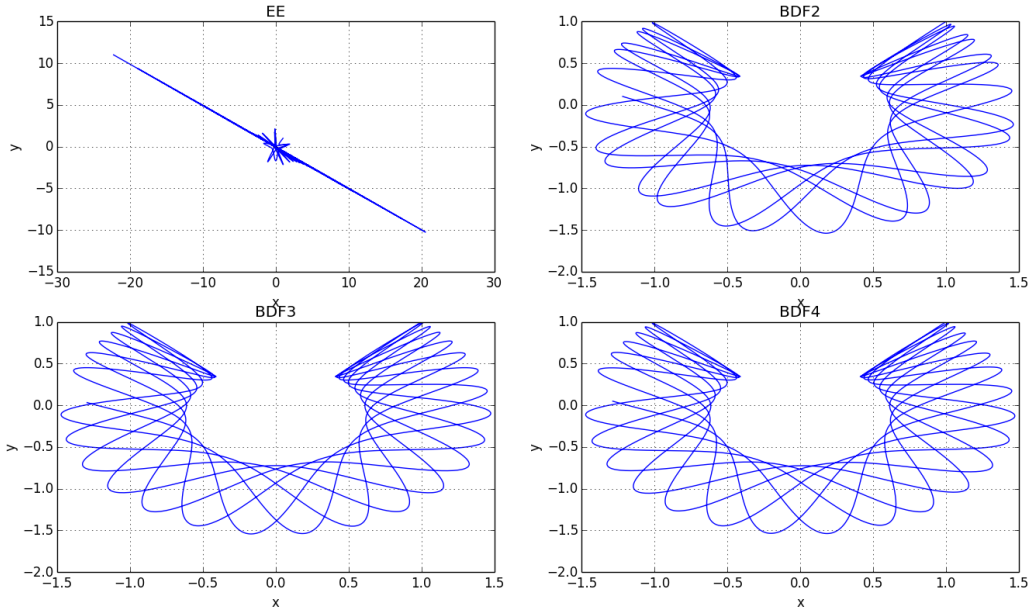



Figure 2: $k = 30, t = 30, init = 1, 1, 0, 0$, stepsize $= 0.01$

At $k = 30$ *figure 2* Euler performs very badly unless the stepsize is significantly decreased. In addition the BDF-2 method gives a slightly different result compared to BDF-3 and BDF-4. Most likely the lower accuracy of the former is affecting the solution compared to the latter methods.

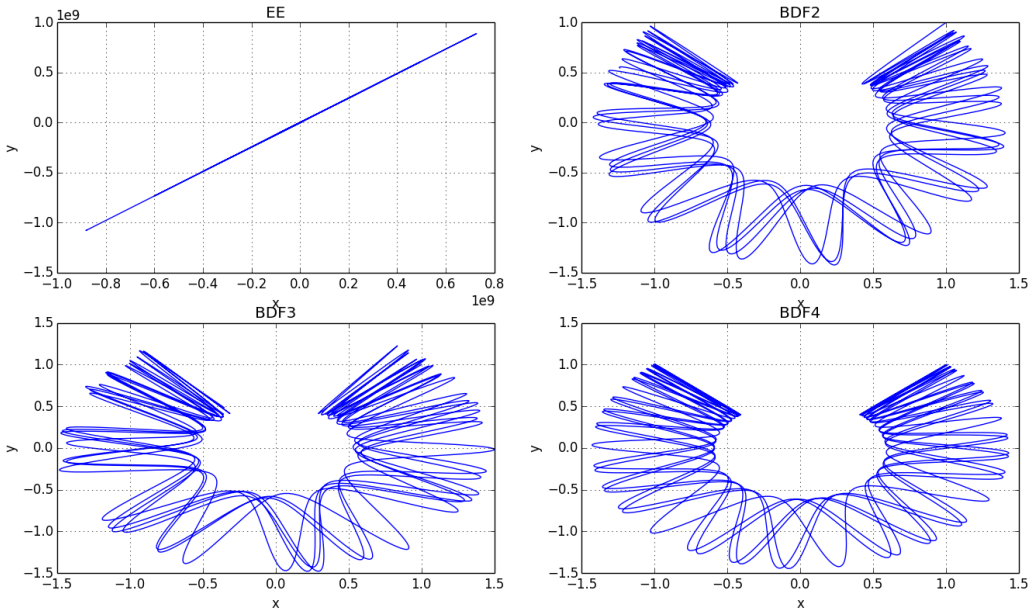



Figure 3: $k = 150, t = 30, init = 1, 1, 0, 0$, stepsize $= 0.01$

In *figure 3* BDF-4 still seems to perform well while the others are having

issues. In fact BDF-3 looks worse than BDF-2 which is peculiar since BDF-4 seems to be working fine. Because of how the stability diagrams looks it could theoretically be possible to find a region for which BDF-2 and 4 are stable but not BDF-3. Could this be such an effect?

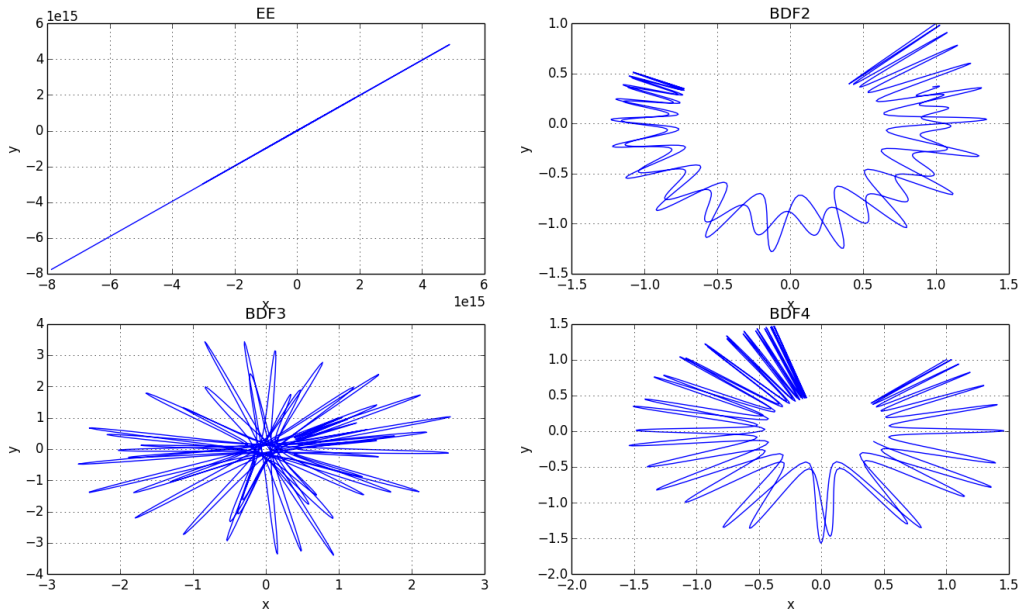



Figure 4: $k = 1000, t = 8, init = 1, 1, 0, 0$, stepsize $= 0.01$. Plots for the function evaluation comparison table.

In *figure 4* it is obvious that Eulers method and BDF-3 fails. The solution to BDF-4 does not seem completely reasonable either although the behaviour is still mostly correct. What is of more interest, however, is that BDF-2 performs much better than BDF-3 (but not correct either). There is clearly a numerical damping effect for BDF-2 showing signs of it being deep in the stability region. The other two BDF methods seem to be outside the stability region, BDF-3 still more so than BDF-4. It would seem that at least one eigenvalue of the system (for high values of $k$) is such that the BDF-3 method fails badly for the current stepsize.

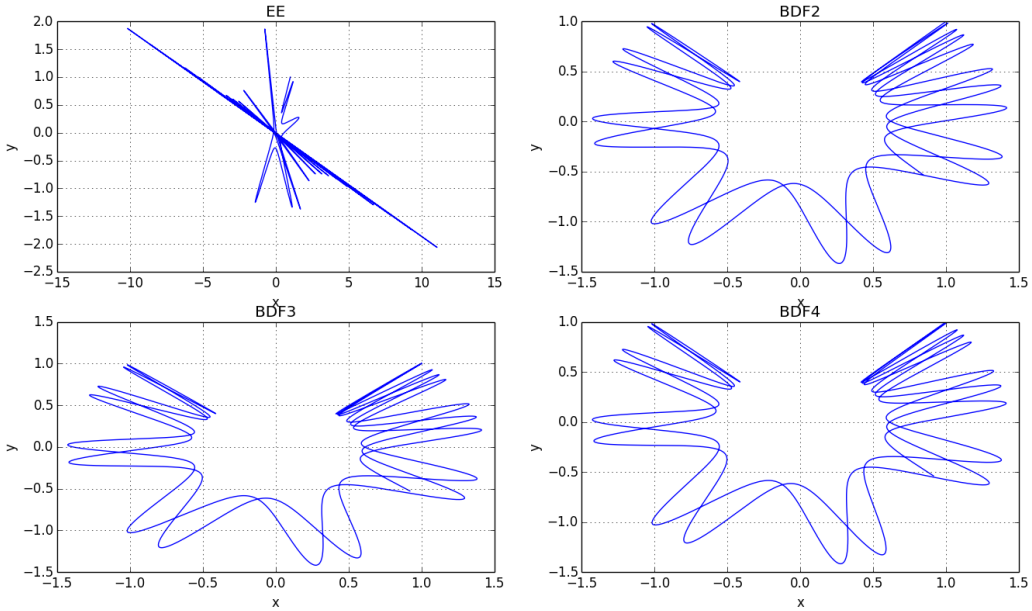



Figure 5: $k = 150, t = 10, init = 1, 1, 0, 0$, stepsize $= 0.005$

Table 1: Comparison between different BDF integrators in Assimulo

| Integrator | BDF2 | BDF3 | BDF4 |
|---|---|---|---|
| CPU-time (s) | 2.340264 | 2.471802 | 2.609666 |
| F-evaluations | 16214 | 16112 | 16061 |
| $\Delta$ F-evaluations | 0 | 102 | 133 |

Increasing the order of the BDF affects the number of function evaluations. A higher order means more function evaluations. This can be seen in *table 1*. Higher order is also more demanding for the CPU . The data for the explicit euler is not included in the table since the solution is not correct.

### 2.2.2 CVODE

For low k-values ($k = 10$) CVode yields the same result for different discretization methods (Adams-Moulton and BDF) and different maximum order of the method. This indicates that the problem is correctly solved by all methods and shows that they can handle low frequency oscillatory systems. The simulation time for different methods is however different. For low order methods the simulation time gets longer due to the necessity of taking shorter steps.

Both Adams-Moulton and BDF with highest possible maximum order (12 and 5 respectively) gives the same reasonable result for $k = 10^7$, see figure 6.

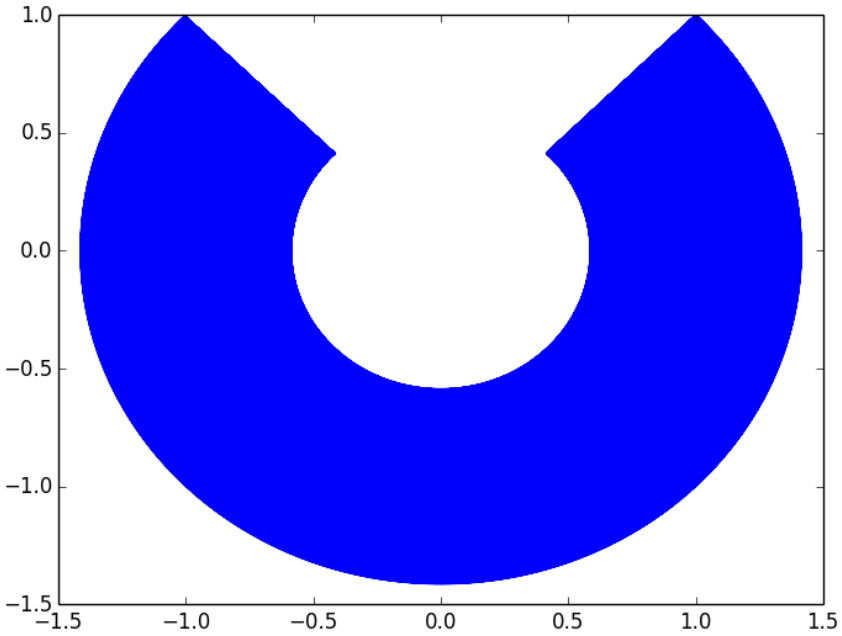
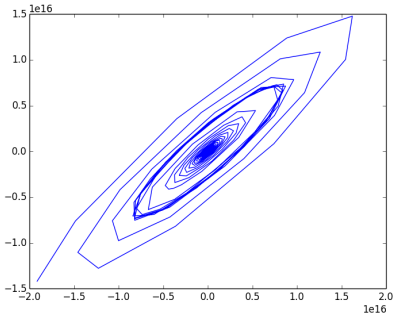


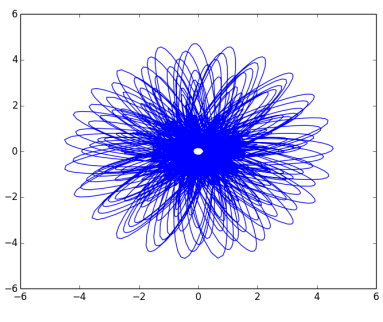

Figure 6: Adams-Moulton discretization, $k = 10^7$, simulation time $t = 30$, $atol = 10^{-8}[1, 1, 1, 1]$, $rtol = 10^{-8}$, maxorder= 12.

For low order methods signs of numerical damping can be observed, see figure 7. It is therefore essential to use high enough order if this needs to be avoided.

Decreasing *rtol* and *atol* gives obviously erronous solutions, see figure 8a and 8b. It is therefore important to choose a reasonably small value of the tolerances. It should be noticed that when decreasing *rtol* the error is escalating more rapidly than when decreasing *atol*. This seems reasonable since the relative error is dependent on the value of the solution.




(a) $atol = 10^{-6}[1, 1, 1, 1], rtol = 10^{-1}$




(b) $atol = 10^{-1}[1, 1, 1, 1]$, $rtol = 10^{-6}$

Figure 8: BDF discretization, $k = 10^3$, simulation time $t = 20$, maxorder= 5.

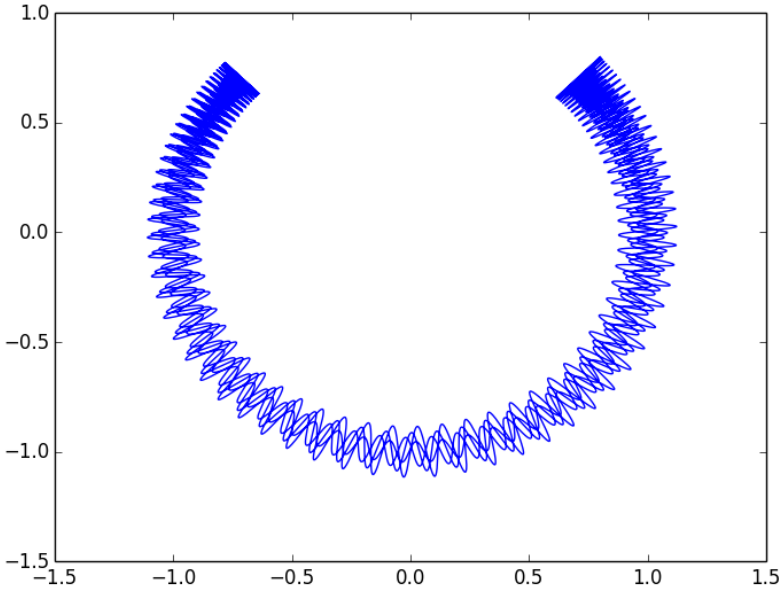



Figure 7: Adams-Moulton discretization, $k = 10^3$, simulation time $t = 20$, $atol = 10^{-6}[1, 1, 1, 1]$, $rtol = 10^{-6}$, maxorder= 1.

### 2.2.3 Dymola

Several different integrators were tested in Dymola. For high $k$ values, integrators like Euler and Rkfix fail to give any reasonable solution due to the stiffness of the problem. Integrators like Lsodar and Dassl are not limited by this aspect since they are adapted to stiff systems.

In figure 9, the results of four different simulations using two different methods are shown. For $k = 5000$, both Rkfix3 and Dassl manage to return a reasonable solution. However, for increasing $k$ values, the problem becomes stiffer and Rkfix3 fails to produce a reasonable solution for $k = 50000$. Even though the solution is still bounded, it is clearly damped, which is a characteristic numerical artifact. Dassl, on the other hand, produces a solution with conserved amplitude for $k = 50000$ and is able to do this even for much higher values of $k$.

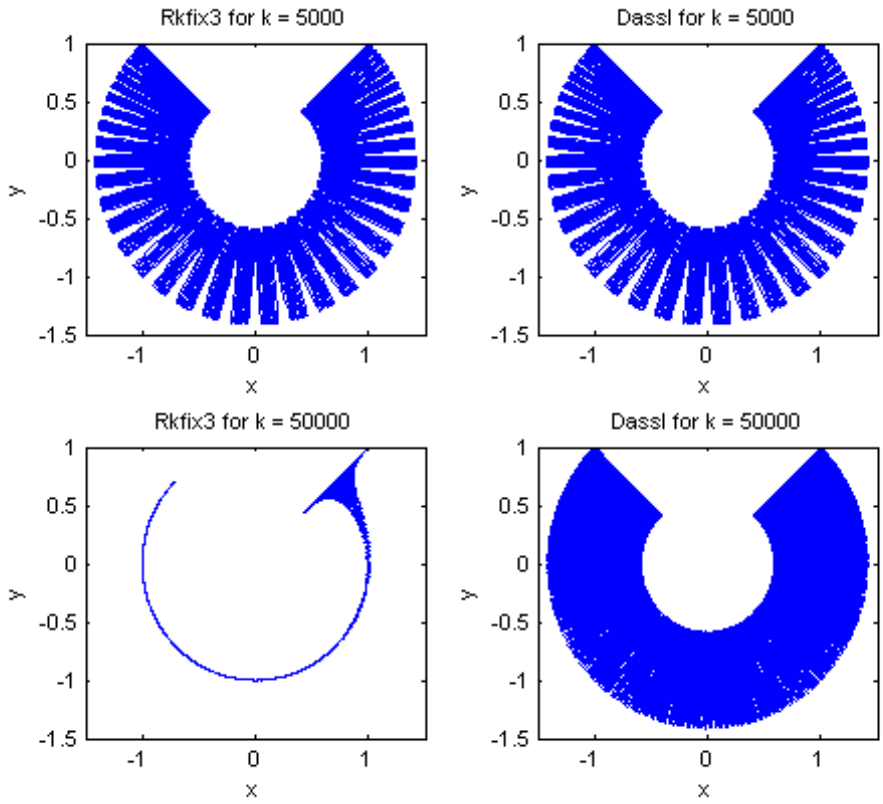



Figure 9: Simulation results for two different integrators in Dymola (Rkfix3 and Dassl, respectively) for two different $k$ values and $t = 30$, $tol = e - 8$, number of intervals = 10000 and $init = 1, 1, 0, 0$

In *table 2*, a quantitative comparison is shown between four different integrators in Dymola. The integrators have been used for solving the problem with $t = 30s$, number of points = 10000 and $k = 1000000$. Note that each of these integrators is adapted to stiff systems and can therefore solve the problem for this high $k$ value.

Table 2: Comparison between different integrators in Dymola for k = 1000000

| Integrator | Lsodar | Dassl | Dopri45 | Sdirk34 |
|---|---|---|---|---|
| CPU-time (s) | 3.17 | 7.34 | 14.97 | 41.28 |
| Successfull steps | 391591 | 698252 | 314896 | 603762 |
| Rejected steps | - | - | 935 | 14570 |
| F-evaluations | 809798 | 1422260 | 1894983 | 6215245 |
| Jacobians | 3706 | 7651 | 0 | 74847 |

The results in *table 2* clearly show a huge difference in CPU-time for the different integrators. The fastest of the integrators, Lsodar, is more than ten times faster than the slowest of the integrators, Sdirk34. The speed of an integrator is dependent on several factors, such as the number of steps, the number of function calls and the number of Jacobian evaluations. Lsodar uses the lowest number of steps and function calls. Of the three methods using Jacobians, Lso-

dar is also the one that uses the lowest number of such evaluations. All in all, Lsodar turns out to be the most well-suited integrator for our highly oscillatory system.

For lower values of $k$, e.g. $k = 10$, the CPU-time is reduced substantially and reaches values below 1 s. It is difficult to say anything about which method is the fastest one, since the short CPU-time makes the results very sensible for simultaneous computations done by other programs on the computer.

### 2.2.4 Comparison between different methods

In *table 3*, a comparison is shown between different integrators using Assimulo, CVODE and Dymola with $k = 100$, $t = 10s$ and $tol = 1e - 8$. The number of function calls is much lower for CVODE and Lsodar than for the BDF methods. While the BDF methods use a constant step size, CVODE and Lsodar use an adaptive step size. This makes it possible for CVODE and Lsodar to solve the problem with a lower number of steps and still keep the same tolerance level.

Table 3: Comparison between different integrators

| Integrator | BDF2 | BDF3 | BDF4 | CVODE(Adams) | Lsodar |
|---|---|---|---|---|---|
| No. steps | 1001 | 1001 | 1001 | 1007 | 1106 |
| F-evaluations | 8469 | 8390 | 8349 | 1227 | 2309 |

## 2.3 Discussion

As our results show, different integrators are suitable for different problems. For low $k$ most integrators yield a reasonable solution. The important differences are the CPU-time and the number of function calls as well as the order of the method, since it will determine the accuracy.

For higher values of $k$, the problem becomes stiff. Hence it is not possible to use integrators like Euler and Rkfix, since they are not adapted to stiff systems. A high $k$ requires a larger number of steps and thereby a longer CPU-time. Some methods shows clear signs of damping for high $k$ values as well. Increasing the order of the method can fix that but at the risk of it becoming unstable.

Of the four integrators tested in Dymola, Lsodar turned out to be the best integrator with respect to CPU-time. Based on the result it is also our personal recommendation for problems of this kind.

Finally *table 3* shows a comparison between the BDF-methods, CVODE and Lsodar in Dymola. They use roughly the same amount of steps for the given problem but CVODE and Lsodar manages to solve it with considerably fewer function calls. Clearly adaptive stepsize and the careful considerations done within those two methods manages to decrease the amount of work significantly. In general though for a problem several integrators should always be used if possible. By doing so the risk of choosing a bad solution (which might look convincing enough) is greatly lowered. It is also easier to feel confident about a solution if several integrators agree on it.

# 3 Project 2: Andrews's squeezer mechanism

## 3.1 Problem Introduction

For the second project the system of interest is "Andrews squeezer mechanism" which has been heavily studied since it was suggested as a test example for numerical codes by Giles and Manning.
The system is shown in figure10. It consists of 7 connected rigid bodies in the plane. Friction is not taken into account. For a complete description of the problem, including the values of the numerical constants and initial values we refer to a book by E.Hairer and G.Wanner[1, p.530-537].
The problem is described by a differential algebraic equation (DAE). The index 3 formulation of the system is given by:

$$\begin{aligned} \dot{p} &= v && (5)\\ M(p)\dot{v} &= f(t,p,v) - G^T(p)\lambda && (6)\\ 0 &= g(p) && (7) \end{aligned}$$

Here $p$ are the angles of the system, $v$ the angular velocities, $M(p)$ the mass matrix, $f(t,p,v)$ the forces in the system, $G(p)$ the Jacobian matrix, $\lambda$ are the Lagrange multipliers and $g(p)$ the algebraic constraints[1].

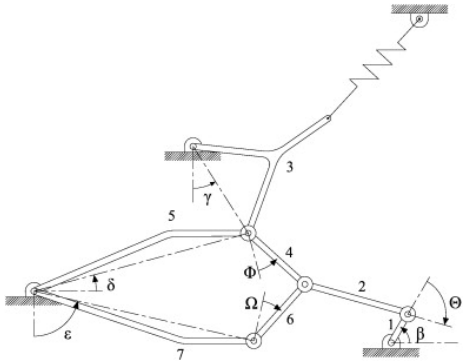



Figure 10: Seven body mechanism (Schiehlen 1990)[2]

The index 2 formulation is obtained by replacing equation 7 with the following:

$$0 = G(p)v \tag{8}$$

Likewise for the index 1 formulation equation 7 should be replaced by:

$$0 = (\frac{d}{dt}G(p))v + G(p)\dot{v} \tag{9}$$

For the report these equations are studied and solved using DAE solvers in Assiumlo and Dymola. Additionally equation 6 can be solved for $\lambda$ using equation 8 and 9. This creates an ODE solvable by, for example, an explicit Runge-Kutta method which is done as well in the report.

## 3.2 Results

### 3.2.1 Assimulo

In the book by Hairer and Wanner a possible choice of initial conditions is given (equation 7.7)[1]. These initial conditions are used for this report as well but they were calculated again to for control purposes. Setting $\Theta = 0$ and using Newtons method for the equations given by (7.7) in [1] gives the results in table 4. The results are exacly the same as those in [1].

| Angle | Initial value |
|---|---|
| $\beta$ | -0.06171389 |
| $\gamma$ | 0.45527982 |
| $\Phi$ | 0.22266839 |
| $\delta$ | 0.48736498 |
| $\Omega$ | -0.22266839 |
| $\varepsilon$ | 1.23054744 |

Table 4: Initial values for angles

Figure 11 shows the result using the IDA-solver in Assimulo for the index 3 formulation of the problem. In order to be able to produce a result the algebraic varables needed to be suppressed. In fact, every lagrange multiplier and velocity had to be suppressed by setting their respective ATOL-values very high. Using the built-in command *"suppress_alg"* did not produce a result. Suppressing that many variables will naturally lead to larger uncertainties with the result which is examined below.

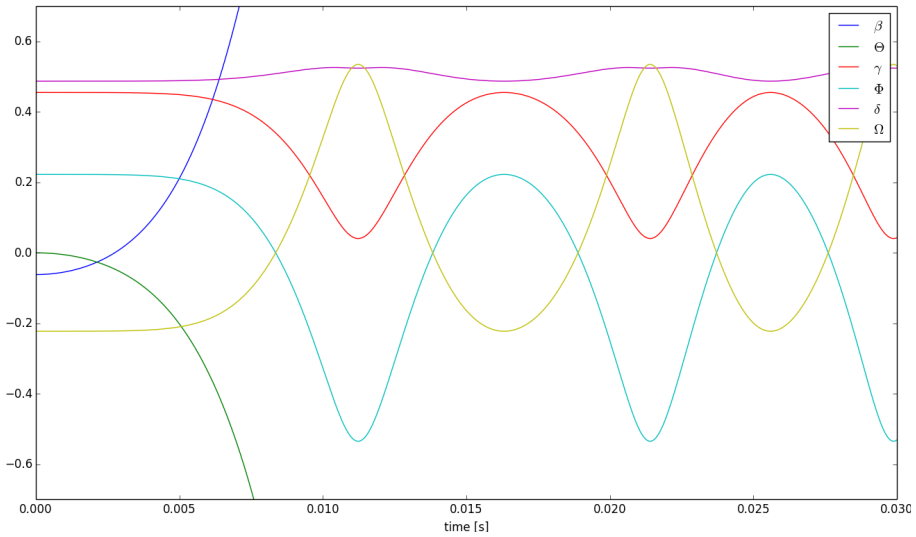



Figure 11: Simulation of Index-3 formulation

The index 2 plot, again using the IDA-solver, is shown in figure 12. The result is clearly very similar to the index 3 result. It is hard to tell a difference. However, the result can be produced without suppressing the velocities (but still the lagrange multipliers) and it works by either setting ATOL high or using *"suppress_alg"* which makes it more preferable to use.

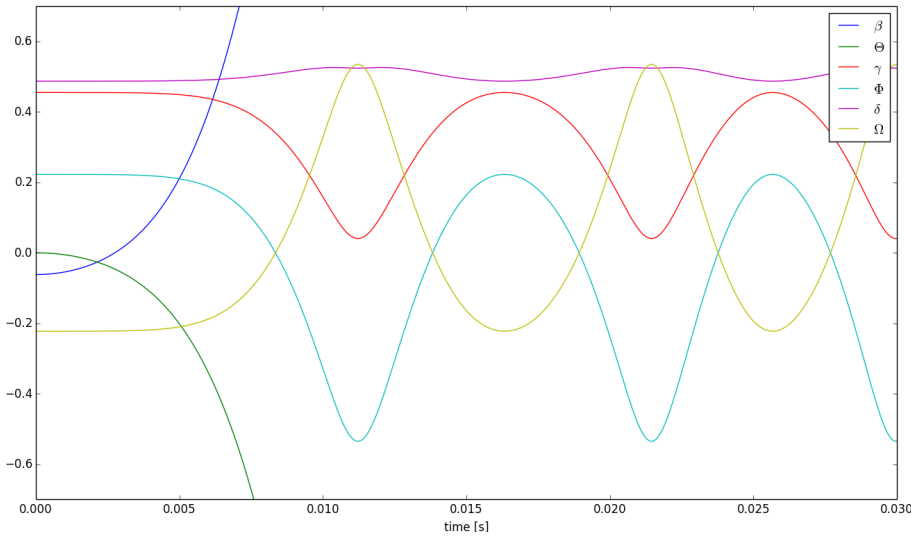



Figure 12: Simulation of Index-2 formulation

To compare the results in figure 11 and 12 in more detail the lagrange multipliers are compared in figure 13 as well as the difference between the actual angles. The lagrange multipliers are clearly behaving worse for the index 3 formulation. They seem slightly unstable with sudden spikes appearing from time to time. With the index 2 formulation this behaviour is not observed.
The bottom plot also indicates that the values differ increasingly as the simulation goes on. Presumably this means that the index 3 values become less and less correct. However, note that the difference is still fairly small in the studied interval but if the simulation would go on and this trend continued it would become significant eventually.

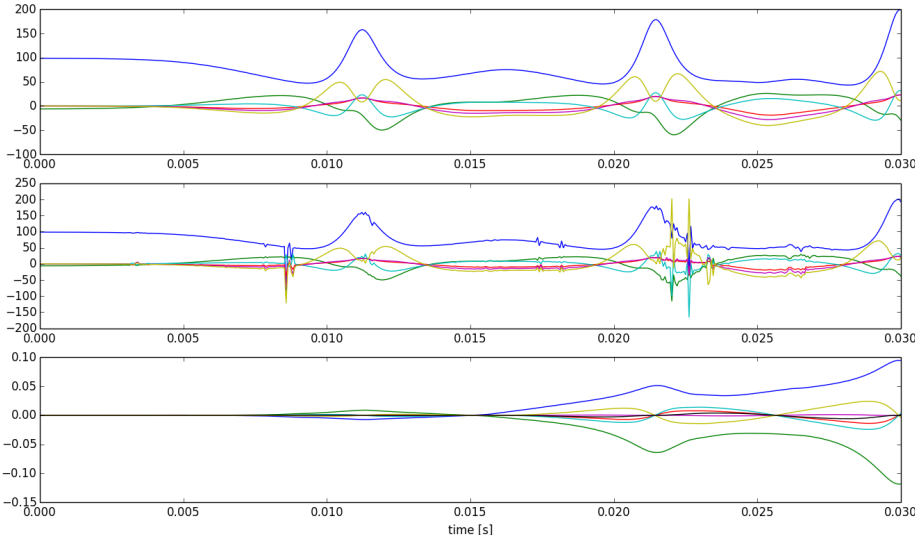



Figure 13: top: Lagrange multipliers Index-2, middle: Lagrange multipliers Index-3, bottom: $y_{index3} - y_{index2}$

| | Index 2 - ATOL | Index 2 -algvar | Index 3 ATOL |
|---|---|---|---|
| Time | 1.379 s | 1.793 s | 8.537 s |
| Steps | 910 | 617 | 707 |
| Function eval. | 1511 | 1613 | 1496 |
| Jacobian eval. | 75 | 120 | 784 |
| Newton conv. fail | 0 | 0 | 120 |

Table 5: Run-time statistics

Table 5 shows some run-time statistics for the cases studied so far. It is easily seen that using the index 3 formulation gives the worst performance by far. The simulation time is considerably longer, the number of Jacobian evaluations is much larger and it has many Newton convergence failures.
Comparing the index 2 cases (suppression using ATOL or *"suppress_alg"*) is harder based only on statistics but it is clear that they perform differently at least with the ATOL version seeming slightly faster. However, using the command rather than ATOL should give a more accurate answer since it does not affect the newton method tolerances. Therefore, if possible, it should be the preferred method to use.

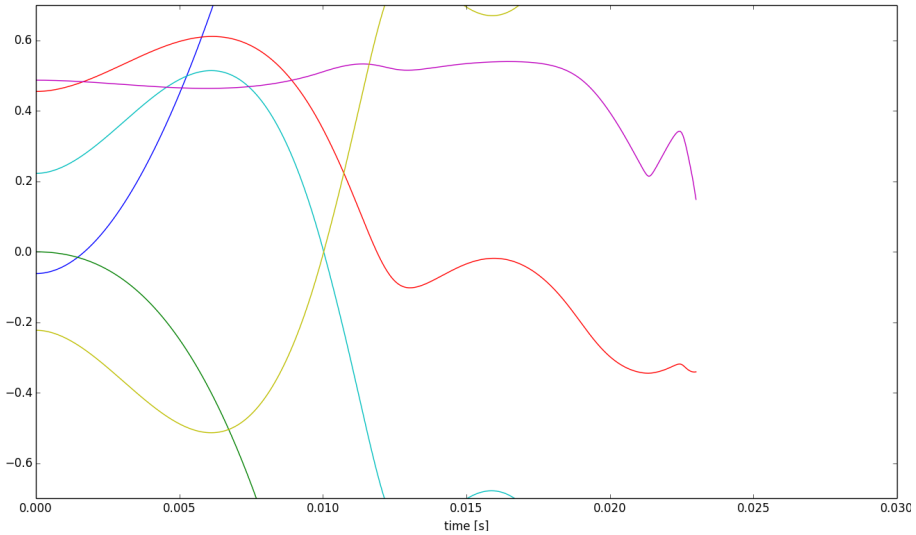



Figure 14: Simulation of Index-1 formulation using RungeKutta34 (variable step size)

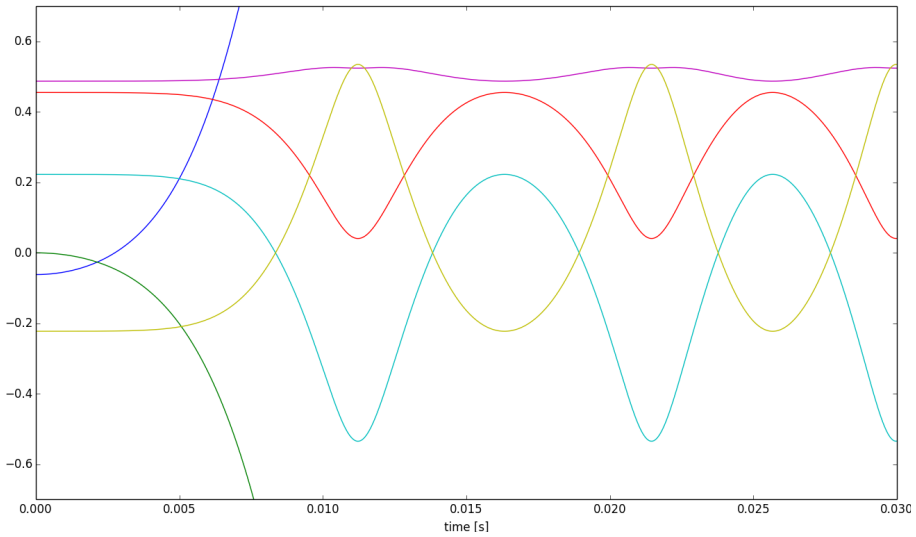



Figure 15: Simulation of Index-1 formulation using CVode

As shown above in figure9, reducing the DAE to a Index-1 formulation presents the possibility to simulate the system using an explicit solver. Figure 14 shows an attempt to use a runge kutta method with a variable stepsize (RungeKutta34). After approximately 0.023 seconds the solver becomes unstable an does not reach the final time 0.03. This despite the fact that the solver has a variable stepsize. Using a multi-step solver (CVode) instead provides a better simulation and does not struggle to reach final time as shown in figure 15.

Figure 16 shows the same simulations as above, CVode resp. RungeKutta34, with angles, angular-velocities and another y-axis. We can clearly see when the Runge Kutta method becomes unstable shortly after the 0.02 mark. It is also visible that the solver is struggling throughout the simulation.

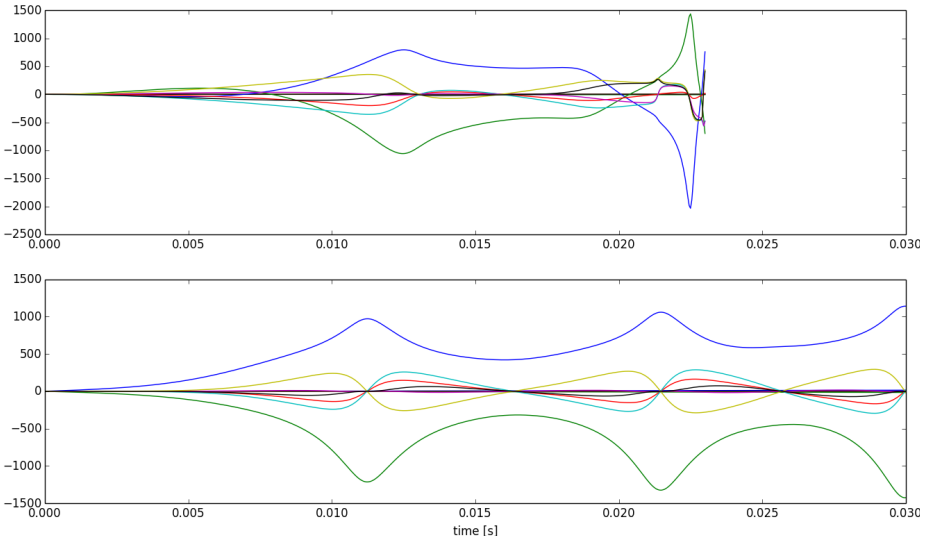



Figure 16: Simulation of Index-1 formulation, top: CVode, bottom: RungeKutta34

### 3.2.2 Dymola

In Dymola, the model was set up by mainly using bodyshapes and revolutes, see figure 17. The initial conditions for the different angles were specified in the revolutes, while the lengths were specified in the bodyshapes. Great care was taken in order to ensure that the relative angles were specified correctly.

The model was slightly simplified by replacing the bent elements by straight ones. By specifying the correct center-of-mass positions, this did not change the model's general behaviour. The resulting animation graphics is displayed in figure 18

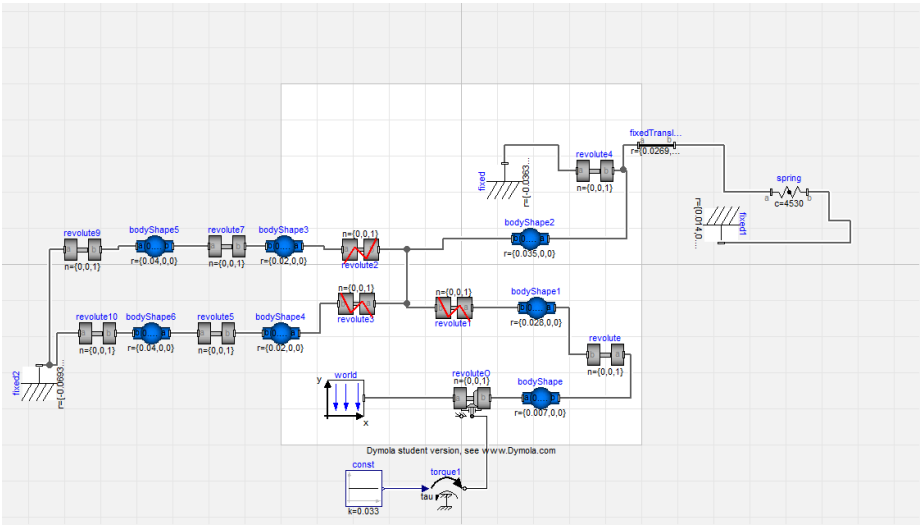



Figure 17: The model of Andrews squeezer mechanism in Dymola.

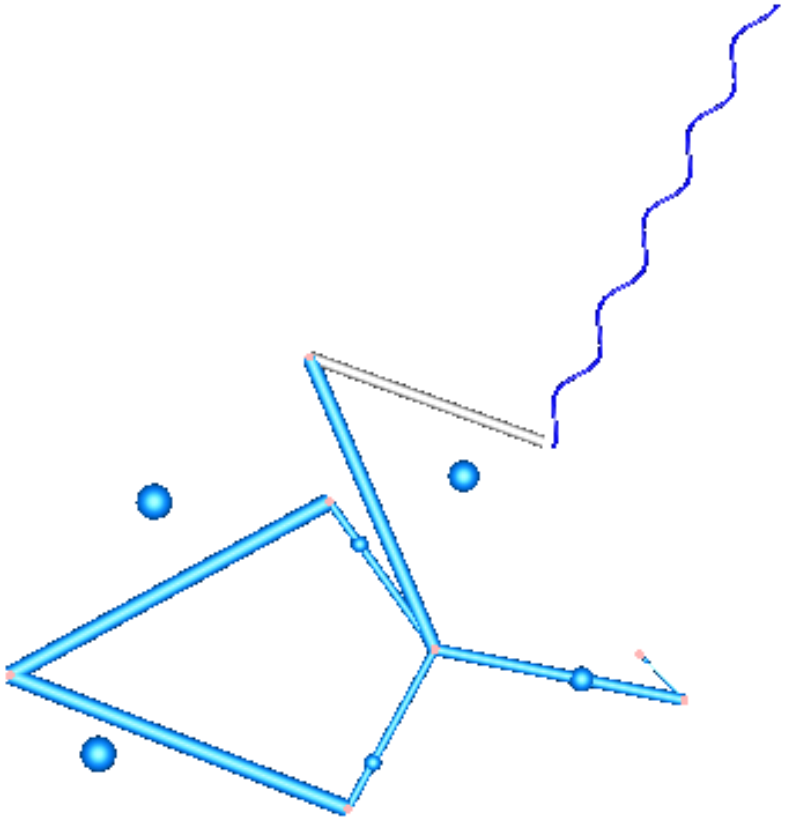


Figure 18: The model animation of Andrews squeezer mechanism in Dymola.

The result of the Dymola simultion is shown in figure 19. As can be seen the result is similar to the ones obtained with Assimulo (this result was regardless of integration algorithm).

In Dymola the reduction of index is taken care of by the software in the graphical mode and did not constitute an obstacle. Since the indexing is solved by the software there was no need to increase the tolerances.

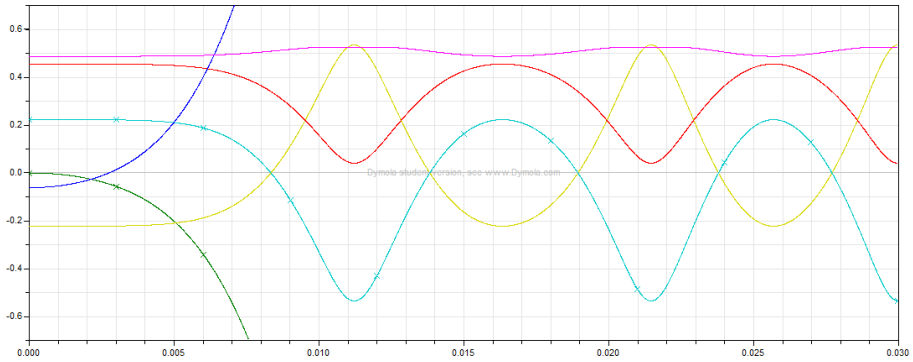



Figure 19: Simulation of Andrews squeezer mechanism in Dymola.

### 3.3 Discussion

Project 2 has primarily shown the difficulties when working with a DAE regarding the index of the formulation. Clearly an index-3 formulation is hard, but not necessarily impossible to use when wanting to produce an accurate solution. Many variables need to be suppressed for it to work which shows that it definitely is preferable to work with a lower index model.
With index-2 the results are noticably better and transforming the problem to an explicit ODE, if possible, can perhaps be a worthwile solution but only if the solver is good enough to handle the problem.
Dymola performs well and removes the issue of having to consider the index value of a model since it automatically reduces it. The animation it provides is also helpful when trying to understand the results.

## 4 Project 3: The woodpecker toy

### 4.1 Problem Introduction

For the third part a woodpecker toy as shown in figure 20 is studied. The system is primarily of interest because it exhibits discontinuous behavior. The model has two bodies, the sleeve and the woodpecker itself. The sleeve is connected to the bar which is represented by a number of constraints which can change under certain conditions. When the sleeve does not block the downward motion it has two degrees of freedom (vertical position and a rotation) and when the sleeve is in a blocking state the degrees of freedom are blocked as well.

Figure 20: Woodpecker toy[3]

The bird is connected to the sleeve by a rotational spring without damping and it has one degree of freedom (rotation relative to sleeve). Note that in this model friction is not taken into account. The model is split into four parts corresponding to four different states:

I The system is free

II The sleeve blocks at lower right and upper left corner.

III The sleeve blocks at lower left and upper right corner.

IV The beak hits the bar.

The model changes depending on which state the system is in and the states will change depending on certain conditions. Note that the system will never be in state IV for an extended period of time. State IV can only be reached from state III and when the conditions are met then no change of state will occur, only a change in inital values. For the exact equations of motion for every state, the physical constants used and a complete description of the conditions for the state changes we refer to the project description given by [4].
Important to note is that the system is given by a DAE formulation with extra constraints appearing in state II and III. Equation (2d) and (3d) are index-3 constraints while (2e) and (3e) are index-2 constraints. By differentiating (2d) and (2e) with respect to time both equations become the index-2 constraint:

$$\dot{\varphi_S} = 0 \tag{i-d}$$

Yet another differentiation along with a differentiation of (2e) and (3e) gives the index-1 constraints:

$$\ddot{\varphi_S} = 0 \tag{ii-d}$$

$$\ddot{z} + r_S \ddot{\varphi_S} = 0 \tag{ii-e}$$

Using the equations given by [4] and the ones above the woodpecker model was implemented in python using Assimulo and in Dymola.

## 4.2 Results

### 4.2.1 Assimulo

Simulating discontious DAE in assimulo makes use of two solver class functions `state_events` and `handle_events` . In each iteration step `state_events` checks whether a discontinouity has occured, ie. an event. If so, the iterator stops and `handle_events` makes the correct adjustments to the problem according to the nature of the discontinuity for the simulation to be able to continue. Our choice of events in `state_events` is when the following equations changes sign:

$$h_S\varphi_S + (r_S - r_0) \tag{10}$$
$$h_S\varphi_S - (r_S - r_0) \tag{11}$$
$$\lambda_1 \tag{12}$$
$$h_B\varphi_b - (l_S + l_G - l_B - r_0) \tag{13}$$

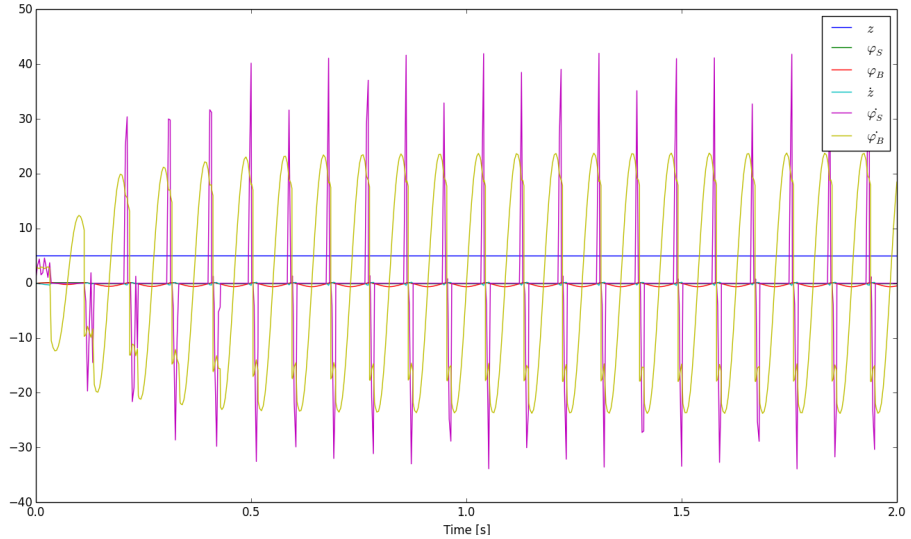



Figure 21: Simulation of woodpecker for 2 seconds.

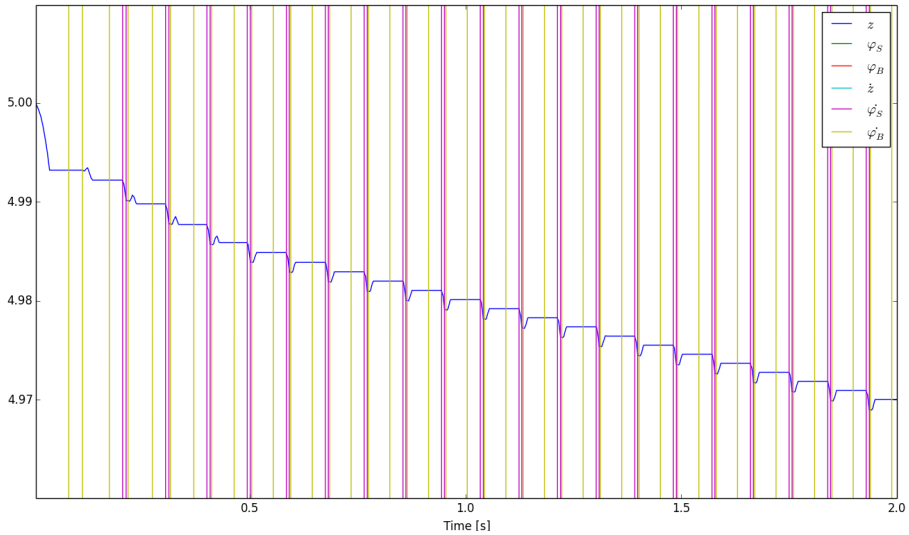



Figure 22: Simulation from figure 21 with zoom on z

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| y0 | 5.0 | 0.0 | 0.0 | 0.0 | 1.0 | 3.0 | 0.0 | 0.0 |
| yd0 | 0.0 | 1.0 | 3.0 | -9.81 | 0.0 | 0.0 | 0.0 | 0.0 |

Table 6: Initial conditions for simulation in figure 21, 22

In figure 21 the woodpecker is simulated using the initial conditions given by table 6. For these initial conditions the woodpecker starts i state I and transtitions into every state several times. The woodpeckers beak hits the bar 21 times in total during the first two seconds of simulation. The hits are visible in the graph as $\dot{\varphi}$ suddenly changing its sign while keeping its absolute value. Figure 22 is the same as the previous plot but zoomed in to see the behaviour of $z$ which decreases slowly but surely. The woodpecker jumps down in steps. These occur when it reaches state I (in other words, when in free fall). It also seems to lurch upward quite frequently after dropping. This is perhaps due to the birds movement when getting stuck in state II or III.

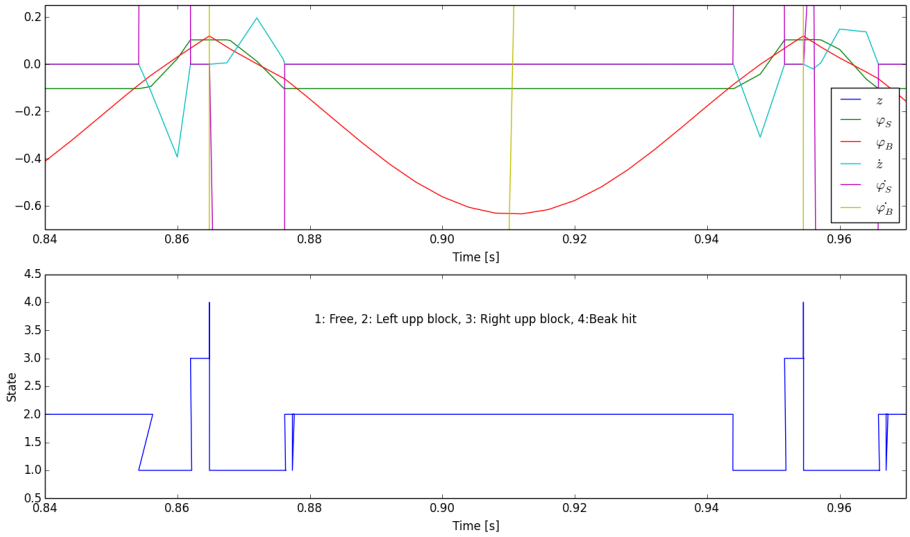



Figure 23: One period of the system with a plot of the different system states.

In figure 23 the systems passage through different states is vizualised. After some careful thought it is clear how the birds movement affects the sleeve's angle and further its $z$ position. During the first change from state II to state I the system seems to be in both states at the same time. This is most likely an artifact from how assimulos handles discontinuity.

## 4.3 Dymola

The model was implemented in equation mode. The equations were put in the "equation" part and the shifts inbetween states were taken care of by when statements in the "algorithm" part. Dymola requires an index-1 formulation and therefore the constraints had to be reduced by differentiation.

In Dymola, we did not manage to get a completely well functioning model. The model was unable to detect when the beak of the bird hit the bar. In other words the simulation never reached state IV. In figure 24 the result of this simulation is shown. As can be seen $\dot{\varphi_B}$ has no discontinuities as a result of the erronous model. However the derivative, $\dot{\varphi_S}$, has peaks which is a characteristic feature of the system.

The problems with this simulation could all be traced back to the project description where two parameters are given the wrong value. When the correct parameter values were found in the py-file and incorporated in the model the same result was obtained as for Assimulo as can be seen in figure 25.

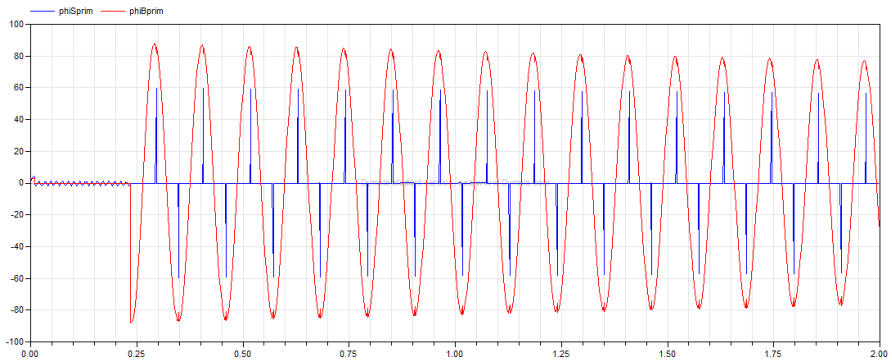



Figure 24: The unsuccessful result of the Dymola simulation.

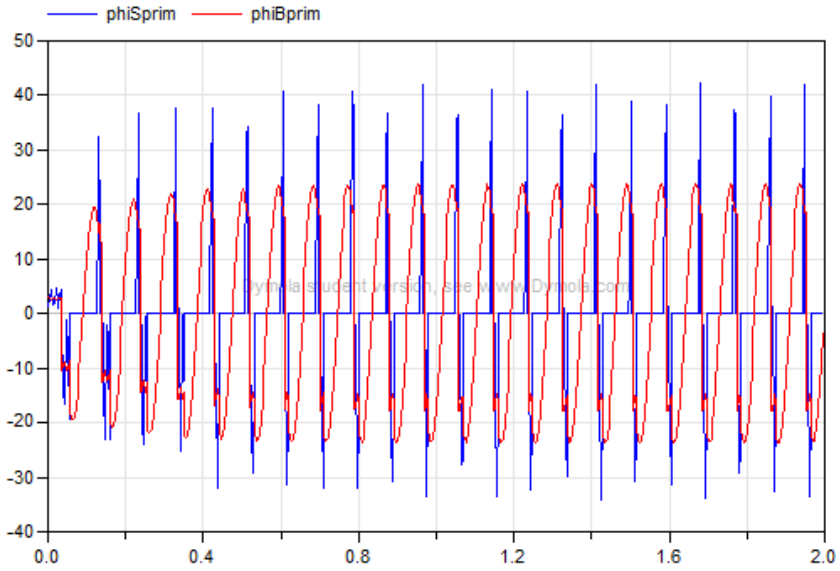



Figure 25: The successful result of the Dymola simulation.

## 4.4 Discussion

Assimulo proves to have a well functioning system for solving discontinous DAE. The methods `state_events` and `handle_events` are intuitive and easy to implement. The Woodpecker DAE's does not have any numerical difficulties apart from the discontinuity and Index III formulations. Using DAE index knowledge from project 2 and discontinuity handling in Assimulo is enough to solve the problem.

Dymola has a good structure to handle this type of problems. Given the correct problem description it this model is easily implemented. The fact that the formulation needed to be index-1 we regard as a disadvantage.

# 5 Overall Discussion

Throughout the project many challenges have appeared, both in form of numerical computation difficulties and understanding how to use the given tools. In the end though, both Assimulo and Dymola have proven themselves to be powerful simulation tools able to handle difficult problems if just given the correct information.
Project 1 showed the importance of choosing a suitable solver. A wrong choice can lead to obviously faulty solutions or, even worse, to solution with a nice and even reasonable behaviour while actually being wrong. Project 2 introduced DAE:s which can have alternative formulations for some constraints (leading to different index formulations). Analysing the model before implementation to maybe reduce its index value proved itself to be of much importance. Project 3 presented difficulties due to it being discontinuous. This required much care to be taken when formulating the problem in Assimulo and Dymola.
Dymola proved itself to be a very capable tool and easily used for certain problems. Assimulo was not as nicely packaged and did not have as many nice features such as animations and pre-programmed parts and models. However, it still provided powerful frameworks, especially when working with the discontinuous problem for which Dymola was not as easy to use.
As for the course it has provided valuable experience working with different tools to study problems that are difficult or problematic in different ways. However, better planning for the time spent on the different problems is needed. Project 1 especially became a disproportionally large part of the course at the expense of the time given to the other two projects. This has affected the understanding of the projects and the theory behind them in a negative way. With that said it has still been interesting and the experience gained can most certainly be of value in the future.

# References


[1] March 2010, *Solving Ordinary Differential Equation II* - By Ernst Hairer, Gerhard Wanner - Springer - 2010.03.08 - Paperback - 614 pages - ISBN 3642052207

[2] I.M. Khan, K.S. Anderson, Performance investigation and constraint stabilization approach for the orthogonal complement-based divide-and-conquer algorithm, Mechanism and Machine Theory, Volume 67, September 2013, Pages 111-121, ISSN 0094-114X, (http://www.sciencedirect.com/science/article/pii/S0094114X13000815)

[3] April 2001, *Nonlinear Dynamics of Wooden Toys*, Remco Leine, Christoph Glocker (http://www.zfm.ethz.ch/ leine/toys.htm)

[4] October 2011 *Project 3,* FMNN05, Claus Führer, Christian Andersson, Lund University (http://www.maths.lth.se/na/courses/FMNN05/media/material/project03___.pdf)