University of Twente

EEMCS / Electrical Engineering
Control Engineering

Simulation with Hamiltonian mechanics

Creating a module for 20-sim

Eric Staal

MSc Report

Supervisors:
prof.dr.ir. J. van Amerongen
dr.ir. P.C. Breedveld
dr. N. Ligterink
ir. B.P.T. Weustink

January 2007

Report nr. 002CE2007
Control Engineering
EE-Math-CS
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

Summary

The simulation program used and developed at the university of Twente, 20-sim, is using simple spatial models for simulation. The spatial models used in 20-sim do not allow simulation of realistic effects, like resonating frames, bending and breaking of beams. To model these structures 20-sim need to be extended to simulate these effects.
This can be realized by making a module for 20-sim which is able to model these effects. Modeling these structures and effects can easily be modeled using Hamiltonian mechanics. Hamiltonian mechanics have a different approach in comparison to classical, Newtonian mechanics. How Hamiltonian mechanics are used is described in this report.
The assignment is to create a module for the simulation program 20-sim which is able to model realistic effects using a continuous spatial description with Hamiltonian mechanics. This module consists of two parts. The main application is necessary to input models, change models and to draw the model. This application is used to set up the simulation parameters. The actual simulation is done by second part of the program. This part is called by 20-sim to do the simulation with the model made with the main application.
Several models are tested to see if the add-on module could simulate Hamiltonian models correctly. The results gave useful information. Most of the testing results are as expected and most results could be explained. However some results are unexpected. Further testing has to be done to see where these unexpected effects come from.
The assignment has several recommendations. The interface which is used in the main application is only able to model simple Hamiltonian models, models which consist of point masses and stiffnesses. To make the application work with other types of Hamiltonian models some changes need to be made. Secondly the integration method used is quite simple. This resulted in simulations becoming quickly unstable. This can be solved with small integration steps to stop the model from getting unstable.

Samenvatting

Het simulatie programma wat de universiteit van Twente gebruikt, 20-sim, gebruikt eenvoudige ruimtelijke modellen om de werkelijkheid te simuleren. Deze modellen zijn vaak voldoende om alles correct te simuleren. Echter zijn er sommige aspecten die wel in werkelijkheid voorkomen maar dan niet gemodelleerd kunnen worden. Hierbij kan gedacht worden aan resonerende frames, doorbuigende balken, et cetera. Dit is met 20-sim moeilijk te modelleren. Om dit soort realistische effecten wel te kunnen modeleren moet 20-sim uitgebreid worden zodat deze realistische elementen wel gebruikt kunnen worden.
Dit kan gerealiseerd worden door het simulatieprogramma uit te breiden met een module om deze ruimtelijke modellen te simuleren. Dit kan eenvoudig worden opgelost door Hamiltoniaanse mechanica te gebruiken. Hamiltoniaanse mechanica heeft een andere aanpak dan klassieke, Newtoniaanse mechanica. De Hamiltoniaanse mechanica zal worden uitgelegd in dit verslag.
De opdracht is een module te maken om het simulatieprogramma 20-sim te laten werken met deze Hamiltoniaanse mechanica. De module moet in staat zijn om modellen te laden en te simuleren op basis van Hamiltoniaanse mechanica. Deze module bestaat uit een tweedelige applicatie. Het hoofdprogramma wat noodzakelijk is om modellen in te voeren, te wijzigen en te tekenen. Het andere deel wordt gebruikt om de simulatie uit te voeren met 20-sim.
Enkele modellen zijn getest om te kijken of het resultaat voldoende was. Het resultaat bleek veel aspecten te hebben zoals deze ook in de realiteit voorkomen, echter waren er ook dingen die onverklaarbaar waren. Deze onverklaarbare aspecten moeten nog nader onderzocht worden om te kijken waar deze vandaan komen en of ze te verklaren zijn. Hieruit kan dan de conclusie getrokken worden of het een simulatie- of modelfout is.
Het programma bevat nog een redelijk simpele interface, er zijn maar een aantal basis Hamiltoniaanse modellen die gebruikt kunnen worden. Om het programma met meer modellen te laten werken zal het programma aangepast moeten worden zodat ook andere soorten modellen, dan modellen met puntmassa’s en stijfheden, gebruikt kunnen worden. Verder wordt er momenteel gebruikt gemaakt van een simpele integratiemethode. Dit heeft als gevolg dat het model snel onstabiel wordt, waardoor de simulatie met een kleinere stapgrote gedaan moet worden.

Table of contents

1 Introduction

When trying to model a real world spatial situation, restrictions are made to get a simplified version of the real world. However, sometimes the restrictions which are made have a bad influence on the model so it does not match the real world sufficient enough. To make a more realistic spatial model, the model will become more complex. Models with a continuous spatial description are therefore harder to simulate.
With conventional simulation techniques it is hard to simulate when a piece of material goes from bending to folding or even breaking. When a piece of material is modeled as a linear model it only works for a maximum deflection, when the force and so the deflection gets too big the linear model is not sufficient enough and another model have to be chosen. Therefore a solution needs to be found which makes it possible to simulate these non linear effects and not only the linear effects for small deflection.

A solution can be found in the form of models based on Hamiltonian mechanics. These mechanics allow creating non linear models. 20-sim, a simulation program developed by the University of Twente, does not have an out of the box solution for creating more realistic models for spatial situations with non linear effects, as bending, stretch and shear, therefore a module needs to be written to make these more realistic models for spatial situations work with 20-sim.

1.1 Assignment

The assignment is to write a module for 20-sim which describes a non-linear Hamilton function. A Hamiltonian describes the energy function of a system. This system can have single or multiple in- and outputs. Each input and output is connected to other parts of the model without generating or wasting energy, this is called power continuity. The transportation of energy to and from the Hamiltonian model is realized with two power ports. Each power port is described by an effort and a flow. The total transferred energy per port is the product of these two terms.

The begin situation is the code written by N.E. Ligterink. This code is written without an user interface as a stand alone application. The assignment is to make this code work with 20-sim and to give it an user interface. To solve this assignment some background information about Hamiltonian mechanics is needed, this is described in chapter 2.

The assignment consists of several subparts:

The construction of the Hamiltonian should be done automatically. The Hamiltonian must be constructed for several basic parts, these parts are a bar, a plate and a block. These parts can have different kinds of energy: bending, stretch, and shear.

Constructing the state vector. The state vector must be initialized at the beginning of the simulation and it must be possible to read this state vector in and out. This makes it possible to change the state vector to a desired position and to read out the current states.

An interface with 20-sim is also necessary. This interface must set down the port variables and the causality, so other models can be connected to the Hamiltonian model. The interface influences the state vector, so it is possible to change the state vector while it is operating.

The last part is the simulation of the system dynamics. The system dynamics are shown by doing a time integration which cooperates with the 20-sim simulation. The system dynamics have to be shown in a graphical interface which has to be time dependent.

The module itself has to be a very general module, so it must be possible to use this module which several kinds of dynamical systems: fluid dynamics, electrodynamics and etcetera. So each different dynamical system can be modeled with the module.

2 Hamilton function

In this section some background information is given about Hamiltonian mechanics. This information is needed to understand the assignment.

Each system has a specific energy function which describes its dynamic behavior. The energy of a system can be described with a Hamiltonian. First some explanation about the Lagrangian is given to understand the Hamiltonian. The Lagrangian describes the equations of motion of a system. The Lagrange equations are a reformulation of classical, Newtonian mechanics. The Lagrange equations are introduced in 1788 by Joseph Louis Lagrange. The trajectory of an object is derived by finding the path which minimizes the action, a quantity which is the integral of the Lagrangian over time. The Lagrangian for classical mechanics is taken by the difference between the kinetic co-energy and the potential energy.
The Lagrange equations simplifies many physical problems, because there are less equations since it is not directly calculating all the forces but only the force which minimizes the action. Therefore it would be possible to calculate more complex systems with less calculation.

Two examples of dynamic systems are shown below. The first system is a simple pendulum. The pendulum swings until all mechanical energy is disappeared due to friction. This pendulum is used in many mechanical systems, i.e. a clock and it easily can be modeled with Hamiltonian mechanics. The second picture, of a crane, must handle large masses without bending or breaking. The machine must be that strong to hold a heavy load. Bending of the crane can be modeled to test if the crane is strong enough to hold a certain load. Dynamic effects like a vibrating frame also can be modeled.

Figure 1: Pendulum

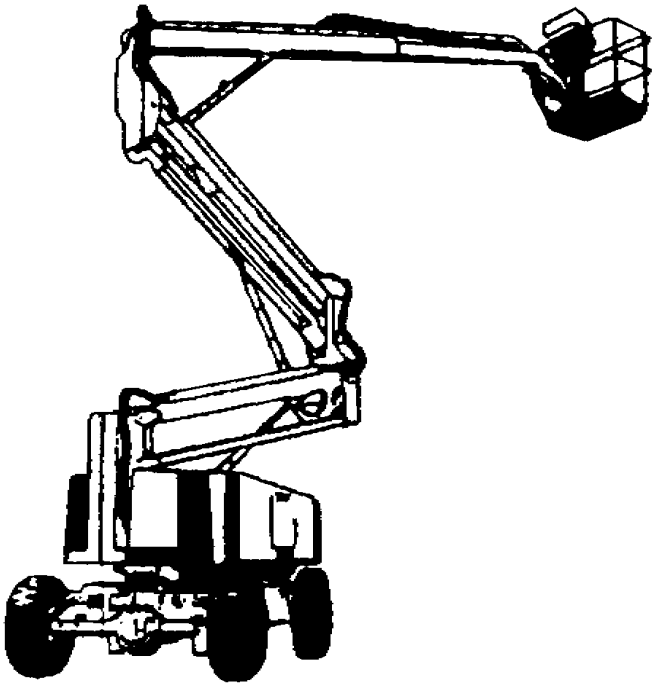

Figure 2: Drawing of a crane

When these systems are modeled with Lagrange mechanics the results can be obtained with less calculation than with classical mechanics, therefore the results are obtained quicker. Lagrange mechanics are used to calculate the path which minimizes the action.

The independent variables of a Lagrangian are q and $\dot{q}$. q are the generalized coordinates and $\dot{q}$ the generalized velocities. The Lagrangian is a function of q, $\dot{q}$ and the time which can be written as

$$L = T' - V$$

Where *L* is de Lagrangian, *T'* is the kinetic co-energy and *V* is the potential energy, the latter two expressed in the generalized coordinates. In order to express the kinetic co-energy and potential energy in generalized coordinates, commonly a coordinate transformation is required that reduces the number of generalized coordinates to the number of degrees of freedom. If this cannot be done straightforwardly, for instance by means of symmetry considerations, additional constraints have to be added by means of the so-called Lagrange multiplier.
If not all the forces acting on the system are derivable from a potential, then Lagrange's equations can be written in the form

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = Q_j,$$

where the first term represents the rate of change of momentum, the second term the conservative forces and the last term the non-conservative forces.

The generalized coordinates used in the Lagrangian can be useful, below an example is described where the use of these generalized coordinates is made clear. When a point mass is moving in a plane, it has two degrees of freedom, X and Y in Cartesian coordinates or angle and radius in polar coordinates. When the particle has a fixed radius, only one degree of freedom remains with polar coordinates, which is the angle of rotation. If this particle, with fixed radius, is described in Cartesian

coordinates, there are still two degrees of freedom, X and Y. This is a rather simple example where the use of generalized coordinates is useful. When a model gets more complex the generalized coordinates may have even more advantage.
Lagrangian mechanics result in a set of second order differential equations of which numerical integration is difficult. In a bond graph representation this is demonstrated by the fact that all kinetic storage ports are written in differentiated causality. This means that the equations are not written in a form that can be numerically integrated in straightforward manner. This will require additional processing power.

Another way to obtain the equations of motion of a model is with Hamiltonian mechanics. Hamiltonian mechanics look a lot like the Lagrangian mechanics but has a few differences. In a bond graph representation all kinetic ports remain their preferred, integral causality, which means that the true energy, i.e. the sum of kinetic and potential energy called Hamiltonian is a function of the generalized momenta and generalized coordinates. The Hamilton is thus defined as

$$H = T + V \,,$$

where H the Hamiltonian, T the kinetic energy and V the potential energy. Note that this is the expression for the total energy, $T+V$. This is no accident, but a general property of natural systems. The generalized momentum is related to the Lagrangian as follows:

$$p_j = \frac{\partial L}{\partial \dot{q}_j}.$$

This means that a Legendre transformation is required to change the generalized velocities as independent arguments of the Lagrangian into the conjugate generalized momenta as the independent arguments of the Hamiltonian:

$$H\left(q_j, p_j, t\right) = \sum_i \dot{q}_i p_i - L\left(q_j, \dot{q}_j, t\right) = T + V$$

In the example of the point mass in a plane mentioned earlier both the Lagrangian and the Hamiltonian are calculated and the role of the generalized coordinates is shown. First the model is described in Cartesian coordinates. The figure of this model is shown in Figure 3.

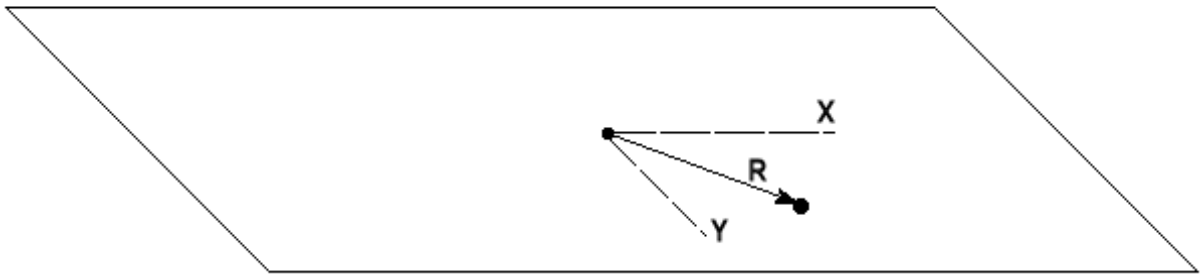

Figure 3: Moving particle in Cartesian coordinates

The particle is represented by the point p and has a mass m. Point p has an X and Y coordinate to represent to location of the point is the plane. The momentum of this particle is described with $p_x = m\dot{x}$ and $p_y = m\dot{y}$. The velocities are described by

$$\dot{x} = \frac{p_x}{m}, \dot{y} = \frac{p_y}{m}.$$

The Lagrangian for this system is

$$L = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}m\dot{y}^2 - V(x, y)$$

Thus the Hamiltonian

$$H = \left(p_x\dot{x} + p_y\dot{y}\right) - \left(\frac{1}{2}m\dot{x}^2 + \frac{1}{2}m\dot{y}^2 - V(x,y)\right) = \frac{p_x^2}{2m} + \frac{p_y^2}{2m} + V(x,y)$$

The Hamiltonian and Lagrangian are for an unconstrained model. When the radius is fixed, so $x^2 + y^2 = r^2$ with *r* a constant, and no potential energy (*V*) is stored, the Lagrangian will be

$$L = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}m\dot{y}^2$$

and the Hamiltonian

$$H = \frac{p_x^2}{2m} + \frac{p_y^2}{2m}.$$

Both functions still have two terms. The constraint has not resulted in a function with fewer terms. If the same model is described with generalized coordinates, polar coordinates in this case, it can be seen that the function will only have one term left, instead of two.

In Figure 4 the same model is shown with polar coordinates.

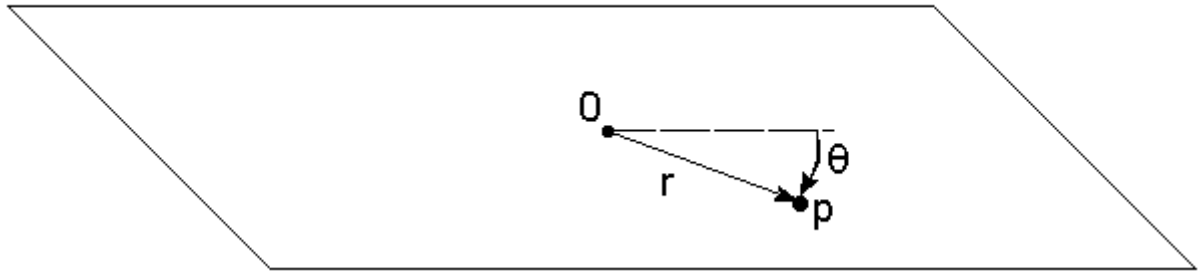

Figure 4: Moving particle described with polar coordinates

The polar coordinates are described by an angle, θ, and a radius, *r*. When using polar coordinates the generalized momenta are given by

$$\dot{r} = \frac{p_r}{m}, \quad \dot{\theta} = \frac{p_\theta}{mr^2}.$$

The Lagrangian for the unconstrained motion of *p* in the plane is

$$L = \frac{1}{2}m\dot{r}^2 + \frac{1}{2}mr^2\dot{\theta}^2 - V\left(r,\theta\right).$$

Thus the Hamiltonian for this motion is

$$H = \left(p_r\dot{r} + p_\theta\dot{\theta}\right) - \left(\frac{1}{2}m\dot{r}^2 + \frac{1}{2}mr^2\dot{\theta}^2 - V\left(r,\theta\right)\right).$$

This can be simplified into

$$H = \frac{p_r^2}{2m} + \frac{p_\theta^2}{2mr^2} + V\left(r,\theta\right).$$

When the particle only rotates around the center with a fixed radius, the momentum in the *r* direction, P_r, will be 0 and the velocity in the *r* direction also will be 0. *r* itself will be a constant. So the Lagrangian will look like:

$$L = \frac{1}{2}mr^2\dot{\theta}$$

And the Hamiltonian

$$H = \frac{p_\theta^2}{2mr^2}$$

The constraint resulted in a Lagrangian and Hamiltonian with only one term. Both functions became simpler because of the generalized coordinates. So when the generalized coordinates are chosen correctly it can result in fewer terms in the Lagrangian and Hamiltonian.

The Hamiltonian is used to describe the Hamilton's equations. These equations are a set of 2n first-order equations. Lagrange's equations are a set of n second-order equations. n is the number of variables in a model.

The first pair of Hamilton's equations for the model described in polar coordinates is

$$\dot{r} = \frac{\partial H}{\partial p_r} = \frac{p_r}{m}, \quad \dot{\theta} = \frac{\partial H}{\partial p_\theta} = \frac{p_\theta}{mr^2}.$$

They simply reproduce the relations between velocities and moment. The second pair, is

$$-\dot{p}_r = \frac{\partial H}{\partial r} = -\frac{p_\theta^2}{mr^3} + \frac{dV}{dr}, \quad -\dot{p}_\theta = \frac{\partial H}{\partial \theta} = \frac{dV}{d\theta}.$$

These relations combine Newton's second law with the fact that the conservative forces are the partial derivatives of the total energy with respect to the displacements. Because the Hamilton's equations correspond to an integral causality in a bond graph representation, these equations can be solved numerically in a more straightforward manner than Lagrange's equations.
In the constrained model where the radius is fixed and the potential energy is zero, so P_r is 0 and the velocity in the *r* direction also is 0, the Hamilton's equations are

$$\dot{r} = 0, \quad \dot{\theta} = \frac{\partial H}{\partial p_\theta} = \frac{p_\theta}{mr^2}$$

$$-\dot{p}_r = \frac{\partial H}{\partial r} = -\frac{p_\theta^2}{mr^3}, \quad -\dot{p}_\theta = \frac{\partial H}{\partial \theta} = 0$$

The generalized coordinates caused Hamilton's equations to be simplified. The term $\dot{p}_r$ is not zero because there is still centrifugal force working on the rotating mass.

With this example the advantage of generalized coordinates is given. The module which is written for 20-sim, must model all kinds of models. One set of coordinates is used, because it is very hard to change the coordinate system for each model. Cartesian coordinates are the most common type of coordinates and therefore they will be used in the module. This can result in large equations, but this should not be a problem because the equations are solved with a computer.

Another example is a girder modeled with several nodes. A normal girder will look like the picture shown below.

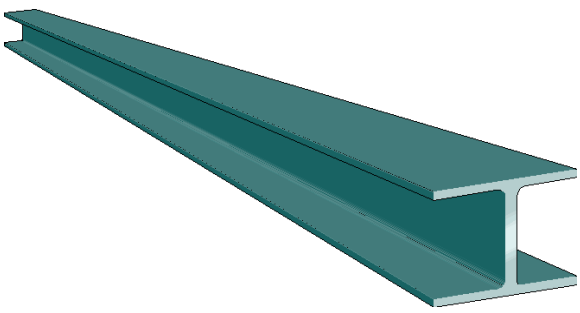

Figure 5: Girder

This girder can be modeled in a 2-dimensional way by making several pieces with a mass and a stiffness between each other. If this girder is modeled as point masses with stiffnesses between them a model as shown in Figure 6 is obtained.

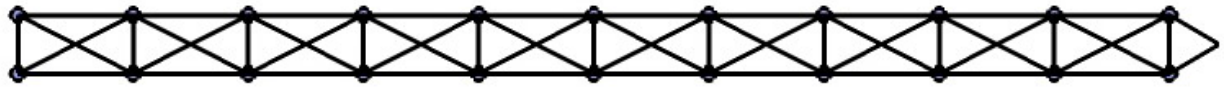

Figure 6: Girder modeled with nodes

Each node is on a fixed position and is held together with the stiffnesses. This girder is modeled as a 2-dimensional model, therefore it is a simple model of a girder.

Each dot is modeled as a node and each line as a stiffness. In this configuration the Hamiltonian (without gravity) looks like:

$$H = \sum_{i=1}^{N} \frac{\vec{p}_i^{\,2}}{2m_i} + \sum_{<i,j>} V_{ij}$$

And the Lagrangian looks like

$$L = \sum_{i=1}^{N} \frac{1}{2} m \dot{\vec{x}}^2 - \sum_{<i,j>} V_{ij}$$

with N the number of nodes, m the mass of each node and, V the potential energy.

V is defined as

$$V_{ij} = \frac{1}{2} k \left(\left(\vec{x}_i - \vec{x}_j \right)^2 - d^2 \right)^2$$

Each X represents the coordinates of each node, k is the stiffness between the nodes and d is the rest distance for the stiffness. Each node has $2n$ states, where n is the dimension. n of them are for the position and the other n for the momenta. With these $2n$ states it is possible to calculate the Hamiltonian.

This kind of model is the point of departure for further models. Each model will be described as point masses and virtual stiffnesses. This makes it easy to create models which are sufficiently competent within the problem context.

The model is not a stand-alone model but has connections to other parts. All the parts make one complete dynamic system, e.g. a crane, a printer or a car. To connect a subpart of a model with other parts of the model the so-called port concept is used. The port concept uses a connection between energy storage elements which has power continuity. This means that no energy is generated or wasted in the connection. Each connection, power port, transfers an energy P. P is represented by the product of effort and flow.

Every subpart of the model is an energy storage element which can be filled or drawn. Each storage element can be classified in 2 types of storage elements, a p-type storage element or a q-type storage element. The p-type storage element is a storage element which stores momentum, a mass, and q-type storage element stores a movement, a spring.

Both types of storage elements have another definition of effort and flow. For a q-type buffer these definitions are:

$$\text{flow} = v = \frac{dx}{dt} \quad \text{effort} = F = kx\,.$$

A p-type of storage element has the effort and flow defined as:

$$\text{flow} = v = \frac{p}{m} \quad \text{effort} = F = \frac{dp}{dt}\,.$$

With the effort and flow defined the energy for both storage elements can be defined:

$$\text{p-type: } E = \int P dt = \int \frac{p}{m}\frac{dp}{dt} dt = \frac{p^2}{2m}$$

$$\text{q-type: } E = \int P dt = \int kx\frac{dx}{dt} dt = \frac{1}{2}kx^2$$

The Hamiltonian model can be defined as a p-type or q-type storage element. This depends on if a force or velocity is applied to the model. When an effort (force) is applied to the system a velocity is returned. The force or velocity can be practiced on one separate node or on the whole system. This can be seen as a bowl where the model is lying in. This bowl is represented by a mass less point which has a stiffness connected to each point mass in the system. So when this bowl, the mass less point, is moved all other nodes undergo a force or a velocity.
More about the coupling of the Hamiltonian to other parts can be found in section 4.

All the information about the Hamiltonian and Hamilton's equations is necessary to understand the problem context and to make correct design decisions. In the next chapter the design of the module is given.

More information about the Lagrangian and Hamiltonian can be found in (Lagrange, 2006), (Hamilton, 2006) and (Fundamentals of multi-body dynamics, 2006) and (Classical mechanics, 2004). More about port concepts can be read in (Dynamical systems, 2003).

3 Creating a plug-in for 20-sim

To make 20-sim (20-sim, 2006) work with models with a continuous spatial description a plug-in needs to be written. This plug-in makes it possible to model these spatial models within 20-sim. This chapter describes all steps which are taken to make 20-sim work with Hamiltonian mechanics and how the module is designed and created.

3.1 Choices for tooling

This part describes several choices which led to the actual program. Each step is described carefully and all decisions are explained.

3.1.1 Connecting the module to 20-sim

There are two ways to connect an external simulation module to 20-sim. Both methods are described and the conclusion is mentioned below.

3.1.1.1 Static DLL

The first method of connecting a third party simulation application to 20-sim is by using the ‘static DLL’ functionality of 20-sim. The static DLL (Dynamic Link Library) functionality uses an external DLL, written in any source code, to obtain a result from a given value. The DLL can be written in any programming language if there is a simply input-output function in it.

An example of the static DLL functionality is shown below:

```
parameters
  string filename = 'example.dll';
  string function = 'myFunction';
variables
  real x[2],y[2];
equations
  x = [ramp(1);ramp(2)];
  y = dll(filename,function,x);
```

In this example it can be seen that the external DLL has the name ‘example.dll’. The function which is called every integration step is ‘myFunction’. So the DLL should have a method which has the name ‘myFunction’ which must return a value when two variables are given. The input in this example is a one-dimensional array with two values, ramp(1) and ramp(2).

The user-function in the DLL must have certain arguments. The function prototype is like this:

```
int myFunction(double *inarr, int inputs, double *outarr, int outputs, int major)
```

where:

inarr	pointer to an input array of doubles. The size of this array is given by the second argument.
inputs	size of the input array of doubles.
outarr	pointer to an output array of doubles. The size of this array is given by the fourth argument.
outputs	size of the output array of doubles.
major	boolean which is 1 if the integration method is performing a major integration step, and 0 in the other cases. For example Runge-Kutta 4 method only has one in four model evaluations a major step.

When a simulation starts with an external DLL several functions are called, if they exist, to do the initialization and termination of the DLL.

The first function 20-sim searches for is the ‘int Initialize()’. This function is to initialize the DLL and it must return a 1 for success and a 0 for an error. After the DLL has been initialized, 20-sim will look for the function ‘int InitializeRun()’. This function is for initializing a simulation run. This function also must return a 1 for success and a 0 for an error.
When the simulation is done, 20-sim will look for the function ‘int TerminateRun()’. This function is called to do some cleaning after a simulation. At last 20-sim searches for ‘int Terminate()’. This function is called when the DLL is unlinked. Both of the termination functions must return a 1 for success and a 0 for an error.
An example of a static DLL can be found in appendix B. More information about writing static DLLs can be found in (20-sim help, 2006) or in the help function of 20-sim.

3.1.1.2 Dynamic DLL

The dynamic DLL, or dlldynamic as 20-sim names it, looks a lot like the static DLL mentioned in section 3.1.1.1. The major difference between those two is the fact that the dynamic DLL uses 20-sim to store all states. In case a component described with different nodes, as in the example of the girder, the number of states depends on the number of nodes which are used for the model. If the model is tetrahedron you only have four nodes as shown in Figure 7: Tetrahedron.

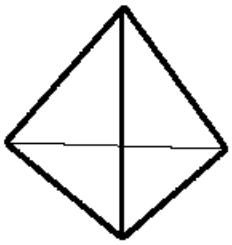

Figure 7: Tetrahedron

So two times the number of dimensions, times the number of nodes, 4 in this case is the number of states which need to be stored. In case of the tetrahedron the number of states are 2x3x4, 24. Each different model has a different number of states which needs to be stored. The advantage when storing the states in 20-sim itself, is that 20-sim uses its own integration method and the DLL does not have to do this.

Because the dynamic DLL stores the states it also has a different function prototype than the static DLL. This function prototype uses a simulating struct. This struct contains information about the simulation, start and stops times and the number of states. The complete struct is shown in appendix C.

The struct is used to give the correct information for initialization and simulation. The dynamic DLL has the function ‘int initialize()’ as the static DLL has. Further is has the function ‘int SFunctionInit(SimulatorSFunctionStruct *s)’ which is used to initialize the simulation run. Next there is a function to set the initial values, ‘int SFunctionGetInitialStates(double *initialIndepStates, double *initialDepRates, double *initialAlgloopIn, SimulatorSFunctionStruct *simStruct)’. A return value of 0 means an error, every other value means success. The initial value for the independent states, dependent rates and algebraic loop variables can be specified by the DLL in this function. This function is just called before the initial output calculation function. If all the initial values are zero, nothing has to be specified.
The function which is called every simulation step is the SFunction. This function has several cases which each case representing different situation. All the situations are described in appendix D. After the simulation the DLL is terminated with the function ‘int Teminate()’. A working example of the dynamic DLL can be found in appendix E.

3.1.1.3 The choice

Both ways of connecting a DLL to 20-sim has their advantages and disadvantages. In Table 1 all the advantages and disadvantages of each method are mentioned.

Static DLL	**Dynamic DLL**
+ Easy to implement	+ 20-sim does the integration
+ Integration method can be different than 20-sim	+ 20-sim can use the states for plotting, etc.
- Integration must be done by the DLL	- All states are stored in 20-sim
- 20-sim cannot read the states and cannot interpret the states	- Implementation is slightly more difficult than the static DLL

Table 1: Comparison table between different DLL writing methods

The dynamic DLL is very useful when an integration method is used, which is also used in 20-sim. It is not possible to use an integration method which is not available in 20-sim. The dynamic DLL also stores each state. Most of the time only several states of the model are useful to the user, therefore a lot of results may be stored unnecessary which results in unnecessary memory usage.
The static DLL is sufficient enough to work with Hamiltonian mechanics and has as advantage that better integration methods can be used which are not available in 20-sim, therefore the static DLL is chosen. The drawing however must be done by the DLL or another application because 20-sim cannot reach the states.

3.1.2 Communication between the DLL and the host application

The DLL, written for 20-sim, can only simulate; drawing and changing of the model is not possible within 20-sim. Therefore a host application needs to be written which can change the Hamiltonian and model which is used and draw the current states of the model. This section describes the communication method which is chosen to let the main application work with the DLL and the other way around.

3.1.2.1 Data copy

Data copy is an IPC (Interprocess Communications) which uses the ‘WM_COPYDATA’ message to send information to another process. This method requires cooperation between the sending process and the receiving process. The receiving process must know the format of the information and must be able to identify the sender. The sending process cannot modify the memory referenced by any pointers. This method is a single way communication, if the other process wants to send data back it must make a new message. The advantage of this system is that it can quickly send information to other processes using Windows messaging. More information about data copy can be found in (Interprocess communications, 2006).

3.1.2.2 DDE for IPC

DDE (Dynamic Data Exchange) makes it possible to exchange information in different formats. DDE use shared memory to exchange the information. Several messages are send between processes that share this data. These messages handle which process has which rights. This is for synchronization, so that only a single process can write at the time.
The data formats are the same as those used by the clipboard in Microsoft Windows. DDE can exchange data between processes on different computers in a network. The disadvantage of DDE is that it is not as efficient as the newer technologies. More information about DDE can be found in (Interprocess communications, 2006).

3.1.2.3 File mapping for IPC

File mapping creates a file which is treated as a block of memory in the process itself. The process gets a pointer where simple pointer operations can be used on to examine and modify information. If more processes want to access this shared memory each process gets its own different pointer. With this pointer each process can independently read and write in this memory space.
File mapping is efficient system which can only be used on a single computer. The only problem with file mapping that there is no synchronization between multiple processes. Therefore separate synchronization functionality must be added. More information about file mapping can be found in (Interprocess communications, 2006).

3.1.2.4 Pipes for IPC

There are two types of pipes for two-way communication. The first is an anonymous pipe and the second a named pipe. The anonymous pipe is used to broadcast information to other processes. Anonymous pipes are for one-way communication, if two-way communication is required a second anonymous pipe must be created.
The named pipe is to read and write information strictly to another process. This process can even be on other a computer on the network. The first process creates a pipe with a known name. The second process opens this pipe using this same name, then there is a connection and data can be exchanged. The anonymous pipes are an efficient way to redirect in- and output to child processes on the same computer. Named pipes provide a simply communication method for two different processes on the same computer or over a network. More information about pipes can be found in (Interprocess communications,2006).

3.1.2.5 RPC for IPC

RPC (Remote Procedure Calls) makes it possible to call remote functions directly. IPC with RPC is therefore just as easy as calling a regular function. Data can easily be transfer with this method. RPC is an interface which supports automatic data conversion with other operating systems than Windows. Therefore RPC is extremely useful for communications between different operating systems.
More information about RPC can be found in (Interprocess communications, 2006).

3.1.2.6 The choice

The biggest problem with the communication between the two processes is that they must operate independently of each other. So when only one process is active the simulation must run as it should be. So 20-sim must be able to run a simulation without the host application to be active. This requirement makes it possible to run a pre-defined simulation without the need of starting the host application. The host application is not needed because the model does not need to be changed during the simulation.
This requirement causes lots of IPCs to drop out. The only one which did not drop out is the file mapping.
The advantage of the file mapping is that all the information about the model is stored in a piece of memory which is available to more processes and therefore storing a model is just as easy as storing this shared memory in a file.
Because there is no synchronization with the file mapping method a mutex is introduced to do the synchronization. A mutex is an object which is used for synchronization of a shared object between several threads. The mutex keeps track which process has the write access to the shared memory, so only one process can write at the time.
This is shown in Figure 8: Schematic of shared memory with mutex.

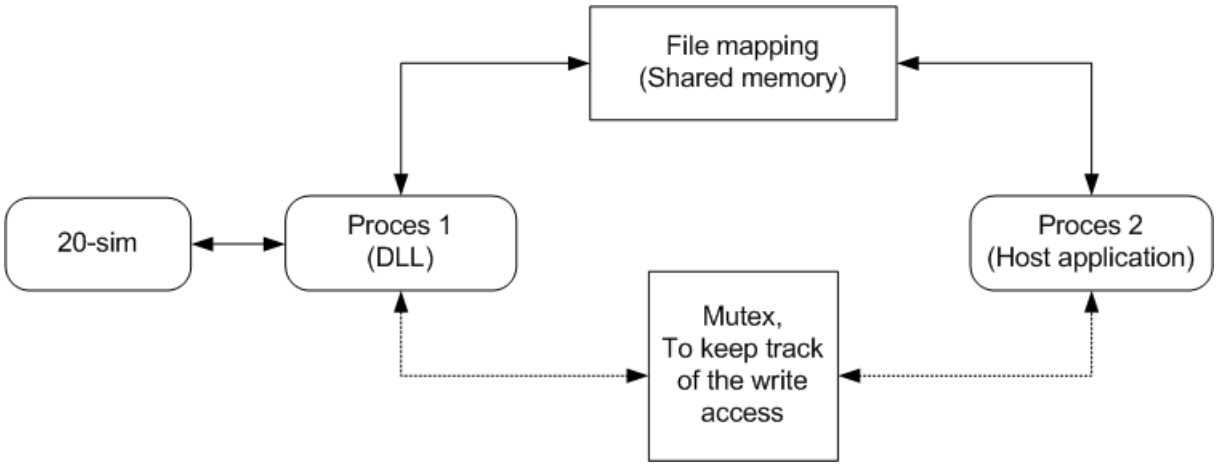

Figure 8: Schematic of shared memory with mutex

3.1.3 The integration method

The Hamiltonian needs to be integrated to calculate the next step. This can be done in several ways. Some ways are quite complex and other quite simple. This section describes a few first-order integration methods and which one is used. The simpler the integration method, the larger the error will be.

3.1.3.1 Euler

The easiest way is with the Euler integration method, shown below.

$$x(t+h) = x(t) + h\frac{dx(t)}{dt}$$

This method only requires a first derivative and only one past value of the derivative.

3.1.3.2 Adams-Bashforth, 2nd order

This is a 2nd order integration method with uses Euler for the first step. This method has a much better results than Euler and does not require much more processor power.

$$x(t+h) = x(t) + h\left(\frac{3}{2}\frac{dx(t)}{dt} - \frac{1}{2}\frac{dx(t-h)}{dt}\right)$$

This method uses two steps and requires the derivative of a step earlier.

3.1.3.3 Adams-Bashforth, 3rd order

This method uses three steps to obtain the result and gives a little better result than the 2nd order Adams-Bashforth method.

$$x(t+h) = x(t) + h\left(\frac{23}{12}\frac{dx(t)}{dt} - \frac{16}{12}\frac{dx(t-h)}{dt} + \frac{5}{12}\frac{dx(t-2h)}{dt}\right)$$

3.1.3.4 Leapfrog

The Leapfrog integration method does not use the velocity of the current time to get the next step, but uses the velocity of a half time period further. This can be seen in the figure below.

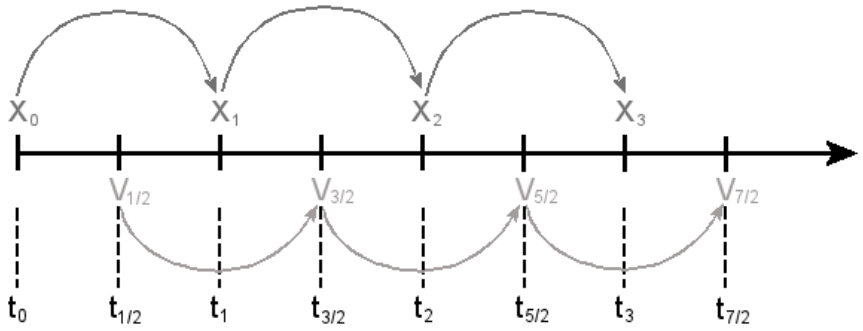

Figure 9: Leapfrog integration method

The integration steps are defined as:

$$x(t+h) = x(t) + h\frac{dx(t+0.5h)}{dt}.$$

Because the Hamiltonian has different state vectors for position and velocity (generalized momentum) this method can easily be used. The velocities are calculated each half step and the position will be calculated each whole step. This integration method is time-reversible therefore good results are obtained.

3.1.3.5 The choice

The Euler method is very unstable but has as advantage that is very easy to implement. The 2nd order Adams-Bashforth method also uses only the first derivative but because it also uses the derivative of a step earlier it has much better results. The 3rd order Adams-Bashforth method has no significant improvement in comparison to the 2nd order Adams-Bashforth method. The leapfrog method has the best results but is a bit harder to implement than a method which does not uses half time periods.
The 2nd order Adams-Bashforth method is chosen because this is an easy method which is sufficient to see if the module is working correctly. More information about integration methods can be found (Integration methods, 2006) and (Leapfrog, 2006).

3.2 Design

This section describes the design of the module to make 20-sim work continuous spatial models. In the schematic below the software design is drawn. This design is made before the implementation was started.

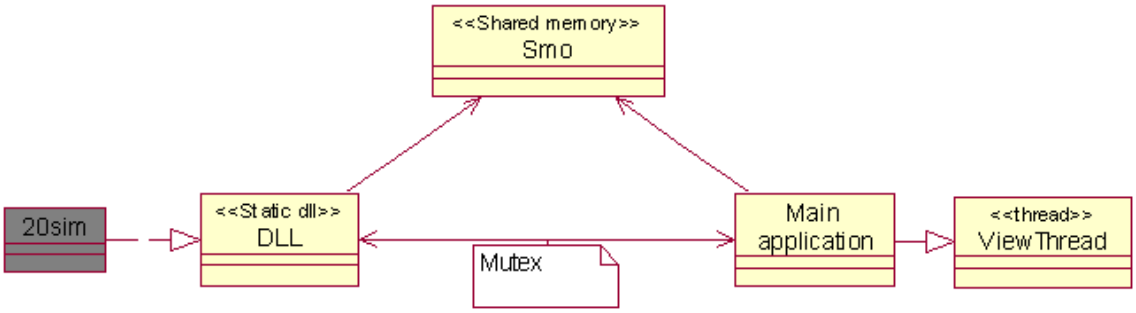

Figure 10: Schematic design

This schematic is finally worked out to a class diagram which can be found in appendix F. Each part is described in the section below.

20-sim

The 20-sim block represents the 20-sim simulation program. This block already exists and therefore it has been drawn darker than the other blocks. The 20-sim program is using the DLL block to communicate with the external module. This module simulates models with a continuous spatial description based on Hamiltonian mechanics.

DLL

The DLL block communicates with 20-sim by DLL calls. These calls are predefined as mentioned in section 3.1.1.1. The DLL gets its information of the model from the shared memory which can be altered by the ‘Main application’. When a simulation is running the DLL will block writing to the shared memory. The shared memory only can be changed if the simulation is stopped.
Because static DLL functionality is used, is the integration done by the DLL itself.

Smo

The ‘Smo’ represents a struct which is the shared memory. The contents of this struct can be found in appendix G. The struct contains all information which is needed for the DLL to simulate a model and all information to create a model with the ‘Main application’. Some information is stored two times in a different way. First the information is stored to create a model and secondly it is stored to simulate the model. By storing this information two times the simulation and model creation are much quicker than when storing it in one way.

Mutex

The mutex block represents the mutex between both processes to keep track of which process has write access to the shared memory. This prevents that two different processes write data at the same time.

Main application

The ‘Main application’ is the program which can change the model and create a Hamiltonian which is used by the DLL for simulation. This application is necessary to create a working model for 20-sim. The application shows how the model will behave during a simulation. The ‘Main application’ can also store and load existing models.

ViewThread

Drawing a model real time takes as much processing power as available, therefore a second thread is created to stop the process from locking. The ‘ViewThread’ handles the drawing of the model during a simulation. When the user disables the drawing, this thread will be suspended.

3.3 Implementation

The implementation went as described in the design part. Some parts are lift out because they require some special attention. These parts are described in this section.

3.3.1 Storing the Hamilton

A Hamiltonian used in the module looks like

$$H = \sum_{i=1}^{N} \frac{p_i^2}{2m_i} + \sum_{<i,j>} V_{ij}$$

This can be written out to a function which looks like

$$H = \frac{1}{2m} p_x p_x + \frac{1}{2m} p_y p_y + \frac{1}{2m} p_z p_z$$

This is a simple Hamiltonian which represents a moving mass without potential energy. When more nodes are added and potential energy is added the Hamiltonian can become very large. This also can be written out. Each term of the Hamiltonian has the same layout: first a coefficient and secondly a number of states. In the Hamiltonian shown above the coefficient is 0.5, if the mass is 1, and than there a two terms, the impulses in the x, y or z direction.

This can easily be stored in three arrays. The first array is an array of doubles which contains the coefficient of each term. The second array is an array with integers of how many variables there will come. The third array is the offset to obtain the correct state.

For example there are 6 states, position and impulses in 3-dimensional. The state array is described as double dStates={‘x’,’y’,’z’,’px’,’py’,’pz’}. So dStates[0] is the x coordinate of the mass.
The first array of the coefficients looks like: double dCoefficient={0.5,0.5,0.5}. The second array is int iNumberOfVariables={2,2,2} because each term has 2 variables. The third array is int iVariable={3,3,4,4,5,5}. The numbers in the third array are the offset for the states ‘3’ represents the third offset and dStates[3] equals ‘px’.

The dCoefficient and iNumberOfVariables array have a size which matches the number of terms. The size of iVariables is the sum of the iNumberOfVariables array and is a lot bigger.

So you can store the Hamiltonian with three arrays and one state array. When using this method the Hamiltonian can easy be extended with more terms, just add the terms in the arrays.

To calculate the whole energy of the system, the Hamiltonian itself, only the three arrays need to be walked through. An example of how it is done can be found in appendix H.
The advantage of calculating the Hamiltonian, when it is stored in this way is that only a few floating point operations are needed and therefore the calculating can be done very quickly.

When the Hamiltonian is stored in this way it is very hard to see what the model does look like. Therefore the model is also stored as a matrix of nodes, masses and stiffnesses. With this matrix is easier to see how the model looks like. The values in the matrix are used to create the Hamiltonian and store it in the way described above.

An example of this matrix is shown below.

K D	1	2	3
1	1	2	0
2	1	2	5
3	0	10	3

Figure 11: An example matrix

This matrix represents 3 nodes with a mass of 1, 2 and 3 shown in the diagonal. In the bottom left part the stiffnesses are shown. The stiffness between node 1 and 2 is 1Nm and between node 2 and 3 is 10Nm. In the top right part are the lengths of the stiffnesses in rest. The length of the stiffness between node 1 and 2 is 2m and between node 2 and 3 is 5m.

The matrix represents the model shown below.

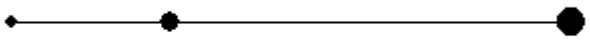

Figure 12: The representation of the matrix

So the actual Hamiltonian is stored in two ways. The first way is for the program and the calculation. The second way is for the user so he can easily see how the model looks like and change it quickly.

3.3.2 The derivative of the Hamiltonian

When using the integration method as described in 3.1.3, the derivative of the Hamiltonian is required to calculate the next step. The derivative of a Hamiltonian is:

$$\begin{pmatrix} \dot{x} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial H}{\partial p} \\ -\dfrac{\partial H}{\partial x} \end{pmatrix}$$

So the partial derivates of the Hamiltonian are required to obtain the derivative. Because the Hamiltonian is stored in three arrays the partial derivates are easily obtained. This is shown in the following example.

Suppose the Hamiltonian is $H = 0.5x^4 + 2xy + x^2y^3 + p_x + 10p_y$. This can be written as $H = 0.5xxxx + 2xy + xxyyy + p_x + 10p_y$ and can be stored in three arrays as follows:

```
dCoefficient={0.5,2,1,1,10}
iNumberOfVariables={4,2,5,1,1}
iVariables={1,1,1,1,1,2,1,1,2,2,2,3,4}
```

The partial derivatives of H are:

$$\frac{\partial H}{\partial x} = 4 * 0.5xxx + 2y + 2xyyy$$
$$\frac{\partial H}{\partial y} = 2x + 3xxyy$$
$$\frac{\partial H}{\partial p_x} = 1$$
$$\frac{\partial H}{\partial p_y} = 10$$

This is calculated as follows, as example the first term of $\frac{\partial H}{\partial p_x}$ is taken.

Each term is searched for a term which contains p_x. If a term is found the term is calculated with one term p_x less. So the term $1p_x$ is calculated to 1. The result is added to an array which contain all the partial derivates under the $\frac{\partial H}{\partial p_x}$.

The integration of $0.5xxxx$ to $\frac{\partial H}{\partial x}$ will take 4 steps. Each term is calculated separately. The first step is differentiating to x this results in $0.5xxx$. The next three steps do exactly the same thing for the other terms and also return a $0.5xxx$. When added together you get $4*0.5xxx$.

Integration of a term with more kinds of variables, like $2xy$, will work in the same way. First the variable x is found. When removing the x from the term, $2y$ is the result. So this is added to array which contains the partial derivates under $\frac{\partial H}{\partial x}$. Second variable found is the y variable. Removing this from the term results in a $2x$. So $2x$ is added to the term $\frac{\partial H}{\partial y}$.

The program walks thru the iVariables array and calculates the partial derivate on the fly. Instead of storing the variables in the array, the actual data is stored. The result is a partial derivate array containing only doubles, which are the partial derivates at the current time.

These partial derivates are used to obtain the derivative mentioned above and the derivative is used to calculate the next step with the integration method described in section 3.1.3.

3.3.3 Synchronizing the drawing

The program is able to draw the current states of the model in a simple plot. To draw the model a thread is used because the drawing takes as much processing power as available. The thread will prevent the system from locking.
The simulation does not take as much processing power as the drawing. Therefore synchronizing is needed. When there is no synchronizing it can be possible that an interchange between two states is drawn, during a simulation step. The synchronizing must also take care of this problem.

Every time a simulation step is taken a flag is set in the shared memory. This flag is the boolean bSync in the shared memory struct, the struct is described in appendix G. The thread waits until the flag is set, when it detects the flag it will copy the contents of the states and unsets the flag. Then the copied data is used to draw the model.

When simulation steps gets very small the changes can be minimal in the drawing. Therefore in can be chosen not to draw each step but after a predefined number of steps, for example after 100 steps. The simulation program will set the flag after 100 steps instead after each step. This will speed up the simulation, because the drawing is not required to do a proper simulation.

3.3.4 Storing the model

When a complex model is created it is useful to make it possible to store it, so it can be reused. As mention in section 3.3.1 the Hamiltonian is stored in two ways. One method is needed to simulate and the other to see how the Hamilton is made. It is impossible to create the matrix as shown in Figure 11 from the Hamiltonian describes in section 3.3.1. On the other hand it would be unnecessary to calculate the Hamiltonian from the matrix each time a simulation is started. Therefore the model and Hamiltonian are both stored.

The Hamiltonian has also other parameters which are not stored in the three arrays. The parameters are fixation of a point and if gravity is enabled. These extra parameters are also needed for the model and are stored in the struct. All the data in the struct is stored in a file.
This file can be reopened and copied to the shared memory. When the file is copied the whole model, Hamiltonian and settings are restored.

3.4 Results

As mentioned before the program consists of two parts, the main application and the DLL. A screenshot of the main application is shown in Figure 13: Screenshot of main application (1).

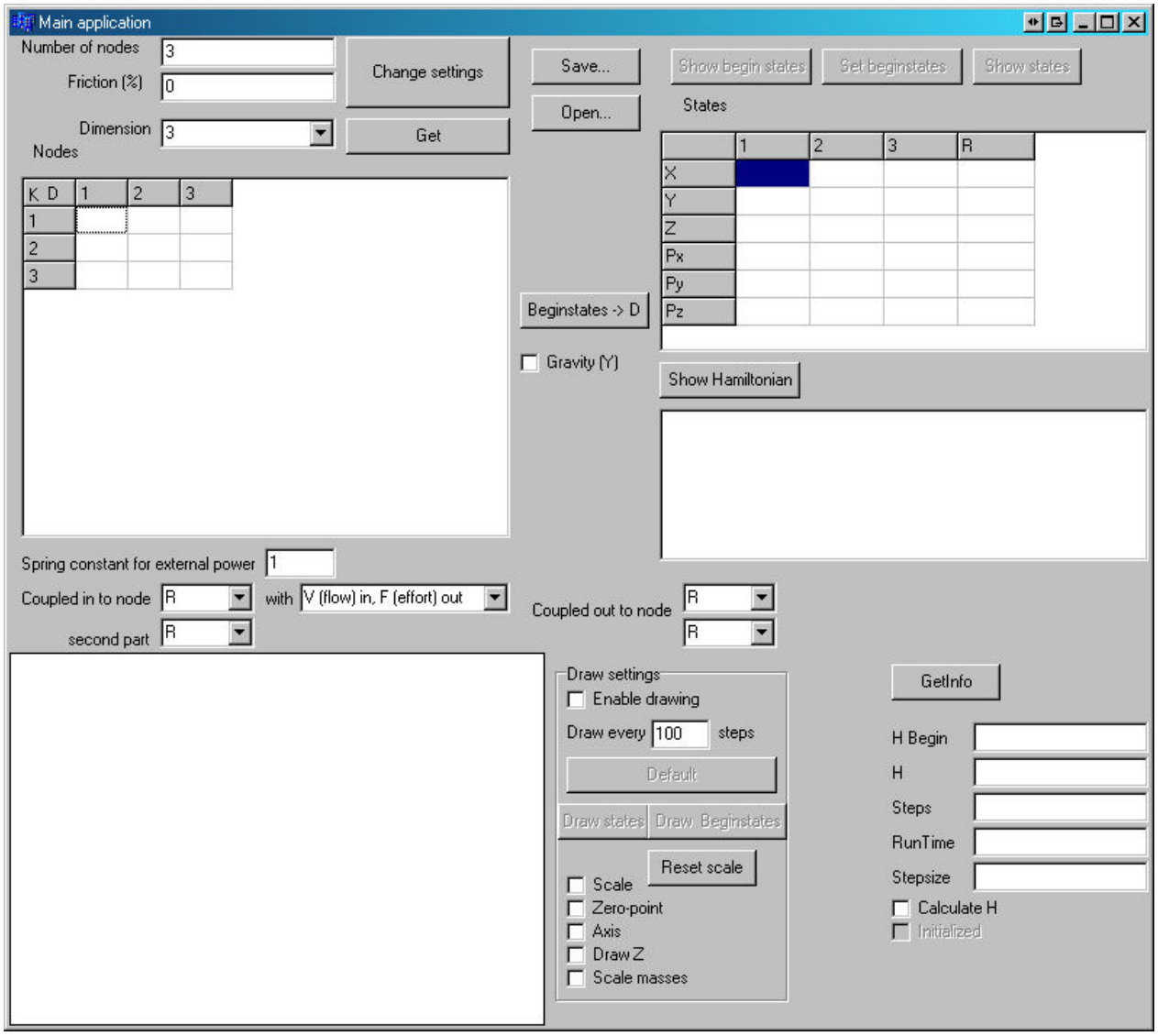

Figure 13: Screenshot of main application (1)

The main application makes it possible to create a model and with this model a Hamiltonian which is used for the simulation. The main application uses nodes and stiffnesses to create a model. This is shown in the upper left table of the main application. With this matrix a model can be created and this model is used to create the Hamiltonian. The variables in the upper left corner are used to change the basics of the model, number of nodes, friction and dimension. The button 'change settings' is used to create the model and the Hamiltonian. When the program is started it is also able to read the information from the shared memory if the shared memory exists with the button 'Get'.
The buttons 'Open…' and 'Save…' are used to open and save a model. The button below 'Beginstates -> D' is used to calculate the distances for the stiffnesses in rest when the begin states are given. So when a node is created at location (0,0,0) and a second node at location (1,1,1) with a stiffness between them, this button will calculate the rest distance. In this case this is $\sqrt{3}$. This value is changed in the matrix so you do not have to calculate this by hand. Below this button a checkbox is shown to allow the user to enable or disable the gravity force. If this box is unchecked no gravity force is added and the model will not 'fall' in the Y-direction.

The upper right table shows the begin states and states when the buttons above are pressed. With the button ‘Set beginstates’ the beginstates of the model can be altered. The states of ‘R’ are the states of a zero mass node which has a spring connected to each node. This can be seen as a ‘bowl’ where the whole model is lying in. The stiffness of each node to this bowl is shown in ‘Spring constant for external power’.
Below the table of the states a textbox is shown to show the Hamiltonian when the ‘Show Hamiltonian’ button is pressed.

The lower left corner is used to draw the model. The Y-direction is vertical, the X-direction is horizontal and the Z-direction is in depth of the screen. On the right of the drawing the draw settings can be altered. The buttons ‘Default’, ‘Draw states’ and ‘Draw Beginstates’ are used to change the model which will be drawn. ‘Default’ draws only when simulating. ‘Draw states’ will draw the current states and ‘Draw Beginstates’ is used to draw the initial states. When the checkbox ‘Enable drawing’ is unchecked the viewthread is suspended and nothing is drawn. If checked the viewthread is resumed and the drawing will continue.

Above the drawing and below the matrix to change the model, the settings are shown for power coupling. The power is coupled to other 20-sim parts with two ports, each port is defined with an effort and a flow. The effort is the force and the flow is the velocity, as mentioned in chapter 2. The model has two ports. Each input is connected to a separate node or to the ‘bowl’, R. The R is a zero mass node which has a stiffness attached to each node. So when as force is applied on R all the nodes receive a force.
The way the inputs are used is shown in the pull down menu on the left. Two options can be chosen. The flow in and effort out or the effort in and flow out. The outputs are defined next to it. The outputs are also represented by a node or R.
So when effort in, flow out is selected the inputs are forces which acts on the nodes. The output is the velocity of the nodes when simulating. More information about the models can be found in the chapter 4.

The last part is in the bottom right corner. This shows all common information about the simulation. The checkbox ‘Calculate H’ is used to calculate the H each simulation step and can be disabled to speed up the simulation.

If the main application has a model loaded it will looks like Figure 14: Screenshot of main application (2).

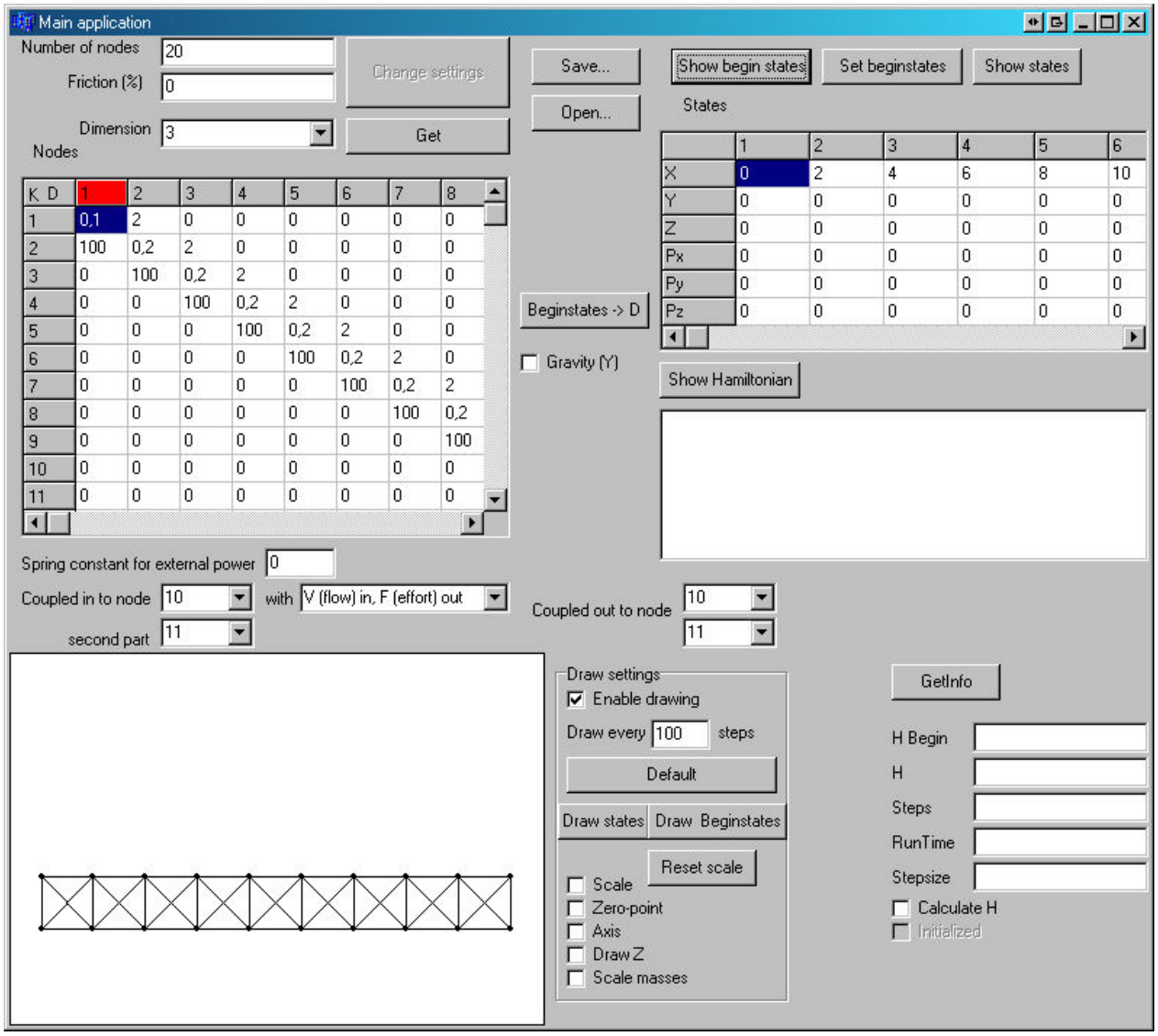

Figure 14: Screenshot of main application (2)

In this figure the node parameters are shown in the matrix upper left. The first node has another color. This color represents the fact that this is a fixed node. The node cannot move. The first node has a mass of 0.1kg, the nodes 2 till 8 have a mass of 0.2kg. There are two powers coupled to the system, one to node 10 and one to node 11. These are the two most right nodes shown in the figure on the lower left. The table top right shows the begin states.

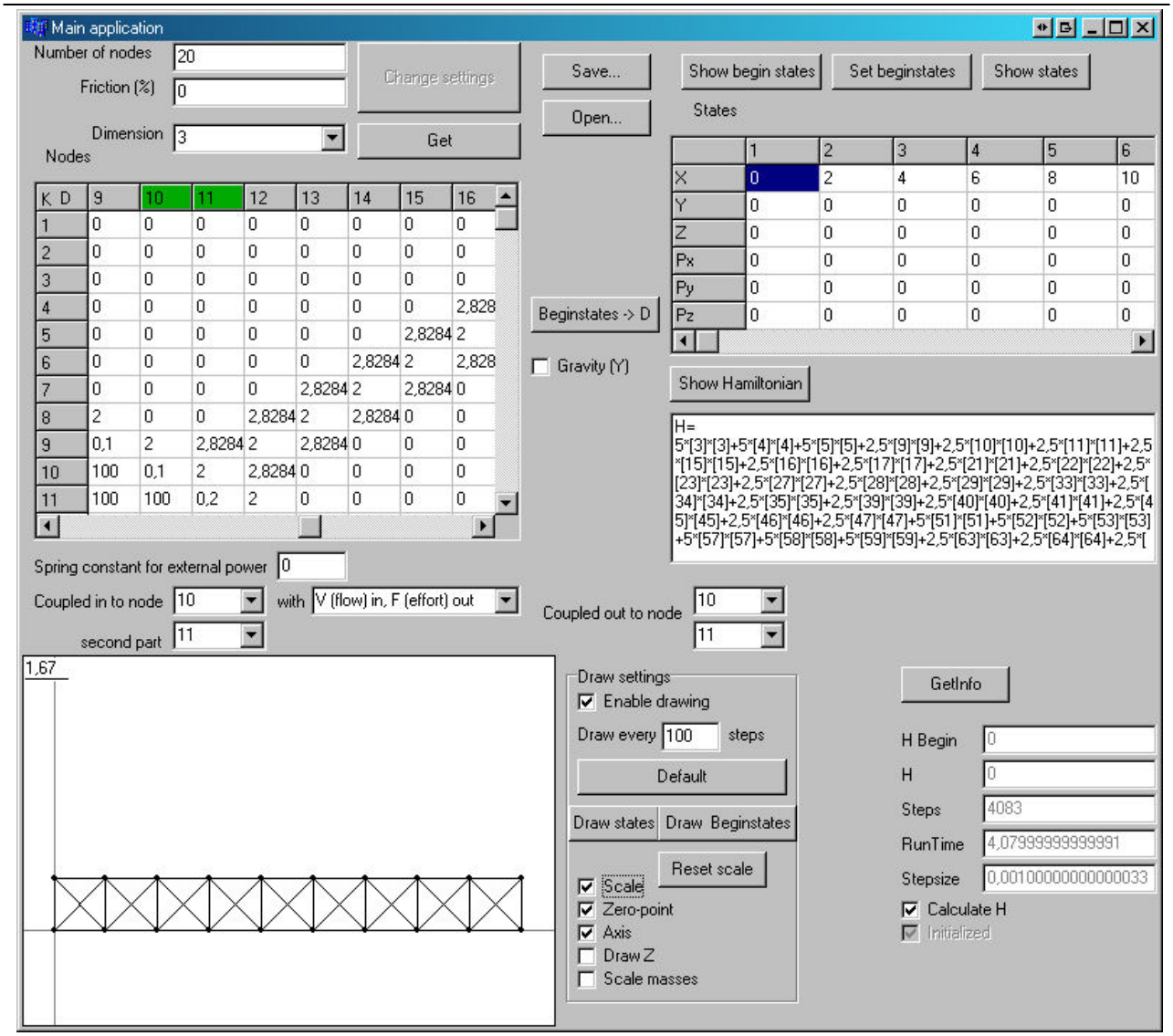

Figure 15: Screenshot of main application (3)

The same model is shown in Figure 15: Screenshot of main application (3). But this screenshot shows the nodes 9 till 16. Node 10 and 11 are fixed in the Y-direction. So they can only move horizontally. The Hamiltonian is also shown. The values between de braces ([,]) is the offset in the state array, so [3] represents the momentum in the X-direction and [6] represent the x position of the second node. The model which has been drawn in the lower leftalso has the scale, zero-point and axis drawn. This is useful to make the drawing clearer.

4 Model testing

To test the module and its functionality several models where tested to see if the whole idea worked. All models which where tested are described in this section.

4.1 Preferred causality

This section is used to explain what kind of storage element the written module is, and how it can be used in a natural way.

As mentioned before the written module uses the port concept to transfer energy to and from the system. Each energy storage element has a preferred causality. For example, a mass is a p-type storage element, which has the integral of effort (force), i.e. momentum as preserved quantity. The p-type storage element uses the integral of flow (velocity) to define the equilibrium and the effort to adjust the equilibrium. The preferred causality is effort in and flow out. The q-type storage element has flow as preserved quantity, effort to define the equilibrium and flow to adjust the equilibrium. This preferred causality is flow in and effort out. An example of a q-type storage element is a spring. More information about the port concept can be found in (Dynamical systems, 2003).

Each storage element is a p-type of q-type storage element, the preferred causality is the type of buffer the storage element wants to be. So a mass storage element prefers effort in and flow out because it is a p-type storage element. The module for 20-sim, which calculates the Hamiltonian models for 20-sim consists of springs and masses, because the springs are q-type storage elements and masses are p-type storage elements it is hard to determine if the module is a p-type or q-type storage element. Therefore some analyzing must be done to determine this.

Each Hamiltonian model consists of masses and stiffnesses. The masses are always at the end of the model. The energy, which is coupled to the model, is always attached to a node. For example if a simple model is taken with 3 masses and 2 springs, as shown in Figure 12. When energy is coupled to this system, it is applied on one of the masses. If the energy is applied on the most right mass, the other two masses will interact with the first mass which has the energy applied to it. Because the energy is always coupled to a mass the Hamiltonian acts as a p-type storage element, but there is an exception. If the energy is coupled to the R the energy is applied straight to a stiffness, to a spring. So when energy is applied on the R the module acts as a q-type storage element.

When the module is used as a q-type storage element it can be seen how a model responds when it is shaken. It is more natural to use the module as a p-type storage element. This is when a beam is modeled and the user wants to know how it will react when a force is applied on the beam. So the user can see how strong the beam is and when it starts to bend or fold.
It is very useless to define a velocity of a node and to see which force is required to obtain this velocity, this is not a natural way. The natural causality of the module is as a p-type buffer, with force in and velocity out.

4.2 Static modelsl

4.2.1 Poisson's ratio

This model is to test how the stiffnesses of a model influences the Poisson ratio. The Poisson ratio is the ratio between the contraction strain divided by the relative extension strain. So the ratio between of bar shortens by a force and the ticker it will become. This is shown in the picture below.

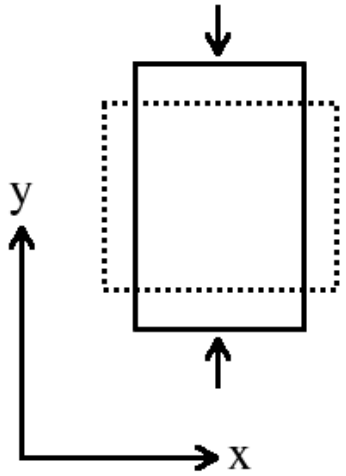

Figure 16: Poisson ratio example

The bar gets shorter and thicker when a uniform force is applied on the bar. The Poisson formula for a 3-dimensional bar is given by:

$$v = -\frac{\varepsilon_x}{\varepsilon_y}$$

ε_x is the transverse strain and ε_y is the axial strain.

The model which is used to test the relation between the stiffnesses and the Poisson ratio is shown in Figure 17: The model for testing the Poisson's ratio.

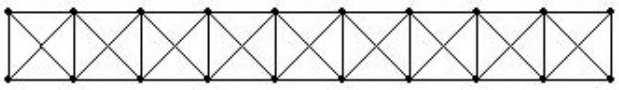

Figure 17: The model for testing the Poisson's ratio

The model represents a beam. The two left nodes are fixed and the two right nodes have a fixed *Y* position, so they only can move horizontally. The model is a 2-dimensional model. Therefore the Poisson ratio formula shown above cannot be used. The strain in z-direction, ε_z ,is fixed to 0 when a 2-dimensional model is used. The Poisson ratio formula for a 2-dimensional situation is:

$$\varepsilon_y = -\frac{v}{1-v}\varepsilon_x .$$

The strain in the y-direction is bigger due to the fact that the strain in z-direction is fixed to zero. This formula is used to determine the Poisson ratio, *v*. More information about theory of plates and the 2-dimensional Poisson ratio can be found in (Theory of plates and shells, 1959).

Each node has a mass of 0.2kg. The mass of the four corner nodes is 0.1kg. All the stiffnesses are 100Nm. A force of 300N per node is applied on the two right nodes to the right. The model is tested with the Euler integration method in 20-sim and the results are after 8 seconds of simulation, when the model was stable.

The first test is how the vertical stiffness influences the Poisson ratio. Results are shown in Table 2.

	K vertical	Height	Length	Y5	Y6	Y15	Y16	X10	X11		Height change	Length change	Passionratio
1	100	2	18	-0,110	-0,130	1,97	1,99	16,63	16,58		105%	92%	0,39
2	200	2	18	-0,046	-0,049	2,02	2,02	16,78	16,77		103%	93%	0,33
3	250	2	18	-0,027	-0,026	2,03	2,03	16,82	16,83		103%	93%	0,30
4	300	2	18	-0,011	-0,008	2,04	2,04	16,85	16,86		102%	94%	0,28
5	350	2	18	0,001	0,007	2,05	2,05	16,88	16,89		102%	94%	0,27
6	400	2	18	0,011	0,020	2,06	2,05	16,89	16,92		102%	94%	0,25
7	450	2	18	0,020	0,031	2,07	2,06	16,91	16,94		102%	94%	0,25
8	500	2	18	0,028	0,040	2,07	2,06	16,92	16,95		102%	94%	0,21
9	550	2	18	0,034	0,047	2,08	2,06	16,93	16,97		101%	94%	0,20
10	600	2	18	0,039	0,054	2,08	2,07	16,94	16,98		101%	94%	0,20
11	700	2	18	0,048	0,064	2,09	2,07	16,95	17,00		101%	94%	0,17
12	800	2	18	0,054	0,072	2,09	2,08	16,96	17,01		101%	94%	0,16
13	900	2	18	0,059	0,078	2,10	2,08	16,97	17,02		101%	94%	0,16
14	1000	2	18	0,063	0,083	2,10	2,08	16,98	17,03		101%	94%	0,13

Table 2: Results when changing K vertical

The *Y5* and *Y6* column represents the *Y* value of point 5 and 6 and *Y15* and *Y16* the *Y* value of point 15 and 16. These are the four points in the middle of the model and they are used to determine the height of the model. The *X10* and *X11* represents the *X* value of the most right nodes, they are used to determine the length of the model.

Poisson ratios above 0.5 are not realistic, because these materials do not exist. Rubber has the highest Poisson ratio of 0.5 and cork the lowest of 0. Materials like metal have a Poisson ratio around 0.3 and concrete about 0.2. A Poisson ratio of 0 is hard to model because the stiffness is infinite in vertical direction, all other materials can be modeled.
There are also materials with a negative Poisson ratio. This means when a bar is compressed is gets thicker instead of thinner. These materials are known as auxetics and they cannot be modeled with the written module.

A graph of the Poisson ratio as function of *K* vertical is shown in Figure 18.

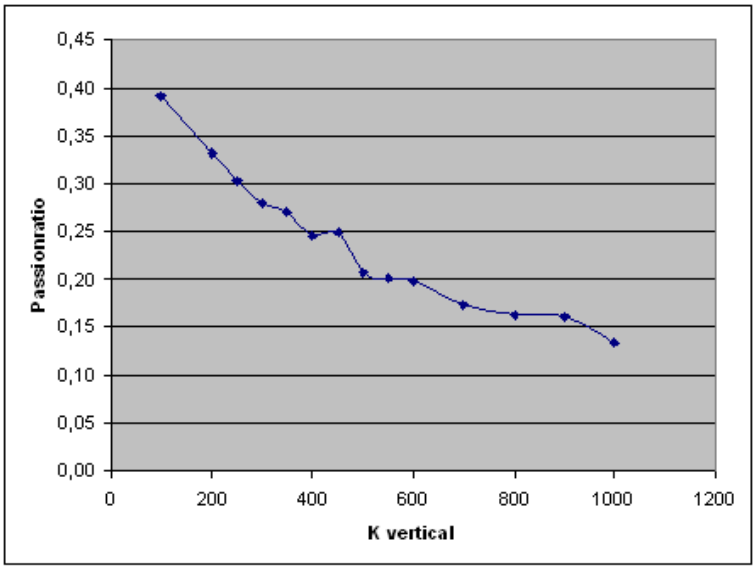

Figure 18: Graph when changing *K* vertically

The model is also tested when changing *K* diagonally. All other stiffnesses remained 100Nm. These results are shown below.

	K Vertical	Height	Length	Y5	Y8	Y15	Y18	X10	X11	Height change	Length change	Passionratio
1	40	2	18	0,068	0,102	2,18	2,14	16,39	16,47	104%	91%	0,30
1	50	2	18	0,014-	0,006-	2,07	2,07	16,47	16,49	104%	92%	0,32
1	60	2	18	0,055-	0,059-	2,03	2,03	16,52	16,51	104%	92%	0,35
1	70	2	18	0,079-	0,089-	2,00	2,01	16,56	16,53	104%	92%	0,36
1	80	2	18	0,095-	0,107-	1,99	2,00	16,59	16,55	105%	92%	0,38
1	100	2	18	0,110-	0,130-	1,97	1,99	16,63	16,58	105%	92%	0,39

Table 3: Results when changing K diagonal

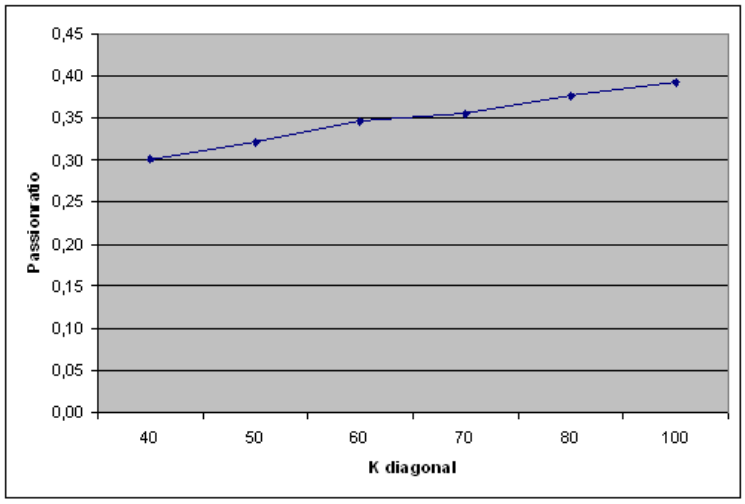

Figure 19: Graph when changing K diagonal

In the second test the model became unstable when the stiffness in diagonal direction was smaller than 40Nm, the result of this simulation is shown in Figure 20. It can be seen that the beam is not as it should be, and therefore it is unstable. Then most right nodes starts to move further left than the second most right nodes. This is impossible with a normal beam, therefore it can be concluded that the model is unsuitable for this situation.

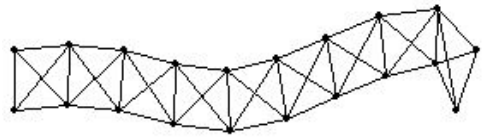

Figure 20: Unstable model

The model only has values for *K* which are useful and *K*'s which causes the Poisson ratio to be smaller than 0.5.

From this test it can be concluded that the Poisson ratio can be altered when changing the vertical or diagonal stiffness of the model. The vertical stiffness has a greater influence on the Poisson ratio but has an unwanted result, the stiffness in y-direction will also change. If the diagonal stiffness is changed the stiffness in y-direction stays more constant because the vertical stiffness maintains the same value.

4.2.2 Clamped beam with a force

When a beam is clamped and a force is applied on the beam the beam starts to bend. Bending of a beam is tested in two ways, the first situation is a single clamped beam and the second situation is a beam with both sides supported. The first situation is shown in the figure below.

Figure 21: Single clamped beam

The single clamped beam has a formula to calculate the deflection of a beam. This formula is taken from (Mechanics of materials, 2001).

This formula is:

$$v = -\frac{Px^2}{6EI}(3L - x)$$

with v the deflection in the y (vertical) direction (positive upward), P the applied force, L the length of the beam, x the position of deflection and EI the material constant. This formula applies only for small deflections.

So when a model is made with the written module, the EI variable must remain constant at every position of the beam even when the force is changed. The model used is shown in Figure 22: Single clamped beam.

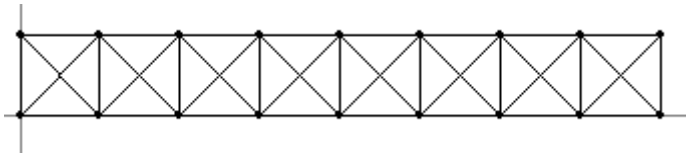

Figure 22: Single clamped beam model

The model has the two left nodes clamped. A force is pressing on the top-right node. Each stiffness is 200Nm and each point mass is 0.2kg, the length between the nodes is 2m. So when a force is applied the beam should bend. The total deflection is depended of the force, so for each force and each position the same material constant EI must be obtained. EI is defined as

$$EI = \frac{\frac{Px^2}{6}(3L - x)}{v}.$$

The deflection is measured at 4m, 8m, 12m and 16m. Each measure point is used to calculate the material constant EI, with the formula shown above. The model is simulated with an Euler integration method with a step time of 0.001 second. The results are obtained after 60seconds of simulation, so the model has a steady state. A friction of 10% is used to stabilize to model. There is no gravity force working on the model, so only the applied force is working on the model. The results of the first test are shown in Table 4.

Force (N)	Deflection at (mtr)					EI at (mtr)				EI difference
	4	8	12	16		4	8	12	16	between 4m and 16m
1	0,009	0,034	0,068	0,107		13037	12549	12706	12760	-2%
2	0,019	0,067	0,136	0,214		12351	12736	12706	12760	3%
4	0,037	0,135	0,271	0,428		12685	12642	12753	12760	1%
8	0,074	0,266	0,537	0,849		12685	12832	12872	12865	1%
9	0,083	0,299	0,603	0,953		12723	12843	12896	12894	1%
10	0,092	0,331	0,669	1,056		12754	12890	12915	12929	1%
20	0,179	0,646	1,304	2,058		13110	13209	13252	13269	1%
40	0,338	1,211	2,438	3,840		13886	14093	14176	14222	2%
60	0,473	1,688	3,377	5,299		14884	15166	15351	15460	4%
80	0,588	2,082	4,139	6,465		15964	16394	16700	16895	6%
100	0,685	2,409	4,757	7,394		17129	17711	18163	18465	8%
120	0,769	2,684	5,262	8,141		18309	19076	19704	20125	10%
150	0,876	3,021	5,865	9,011		20091	21185	22097	22728	13%
200	1,017	3,447	6,593	10,031		23074	24756	26210	27222	18%

Table 4: Results from first measurement

The comparison of *EI* at 4 meter and EI at 16 meter is shown below.

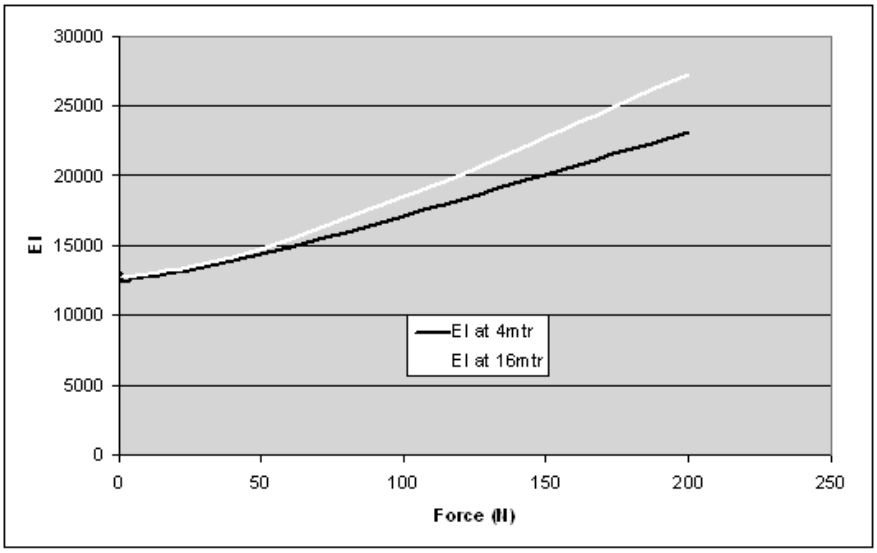

Figure 23: Graph of *EI* at 4 meter and at 16 meter

The *EI* should be constant for each force and at both positions. The results should be a horizontal line instead of a curve. When the force is rather small, less than 40N, the model is sufficient because it does not differ a lot from the results obtained by the formula. If the force gets larger this formula is not sufficient enough and the results differ a lot.
Secondly it can be seen that the *EI* constant is different at the beginning of the beam compared to the one at the end. The difference between the first point, at 4m, and the last point, at 16m, is shown in the last column. The difference is small, less than 5%, when the force is less than 80N. If the force gets too large the difference between the two material constants gets very big, a difference of 18% is obtained.

In the next plot the difference in terms of percentage between *EI* at 4m is compared to the *EI* at 16m.

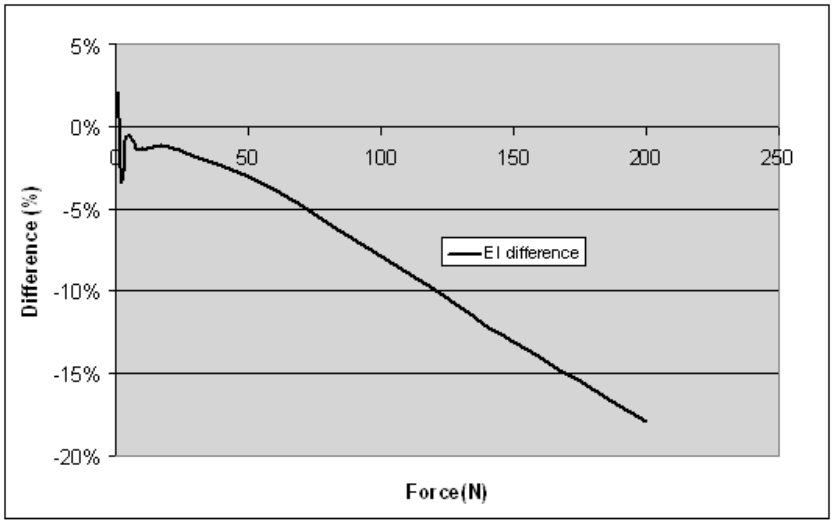

Figure 24: *EI* difference between 4m and 16m

In this graph it clearly can be seen that the formula is sufficient for small forces, less than 40N, because the difference between *EI* is less than 5%. There are two common reasons why the *EI* difference in terms of percentage increases while the force increases. The most likely reason for this is that the formula used to calculate the *EI* constant only works for small deflections, when the deflections are too big the formula cannot be used which results in incorrect results. The second reason is that the model is not good enough to represent a beam. The model only uses point masses and stiffnesses. And a real beam has more aspects than point masses and stiffnesses.

In Figure 25 the model is shown after 60seconds when 200N of force is applied. It can be seen that the beam has a large deflection, therefore the formula from (Mechanics of materials, 2001) is not sufficient enough for this kind model with 200N of force.

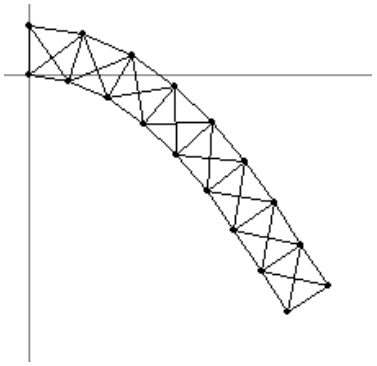

Figure 25: Single clamped beam with 200N of force

The bending mostly occurs in the beginning of the beam. At the end the beam is more stretched than bended. Therefore the beam looks stronger at the end than at the beginning. A beam is stronger when stretched than bended. This explains the large differences of the *EI* constant between the point at 4m and at 16m, shown in Figure 24.

The same test is done with another model. This model is the same except the diagonal stiffnesses are 300Nm and the horizontal and vertical stiffnesses are 100Nm. Results are shown in Table 5.

Force (N)	Deflection at (mtr)					EI at (mtr)				EI difference
	4	8	12	16		4	8	12	16	between 4m and 16m
1	0,018	0,066	0,134	0,212		6519	6465	6448	6440	-1%
2	0,036	0,132	0,267	0,422		6519	6465	6472	6471	-1%
4	0,072	0,262	0,530	0,839		6519	6514	6521	6509	0%
8	0,141	0,514	1,041	1,647		6657	6641	6640	6632	0%
9	0,158	0,575	1,165	1,842		6684	6678	6675	6671	0%
10	0,175	0,635	1,287	2,036		6705	6719	6713	6706	0%
20	0,329	1,192	2,407	3,798		7133	7159	7179	7190	1%
40	0,569	2,042	4,081	6,390		8248	8358	8469	8547	4%
60	0,741	2,624	5,179	8,038		9501	9756	10010	10192	7%
80	0,869	3,039	5,929	9,013		10802	11232	11658	12119	12%
100	0,968	3,352	6,470	9,888		12121	12729	13354	13808	14%
120	1,050	3,597	6,880	10,451		13410	14234	15070	15677	17%
150	1,148	3,885	7,344	11,069		15331	16474	17647	18502	21%
200	1,274	4,237	7,886	11,770		18420	20140	21912	23200	26%

Table 5: Results from second measurement

A graph of the *EI* difference between 4m and 16m is shown below.

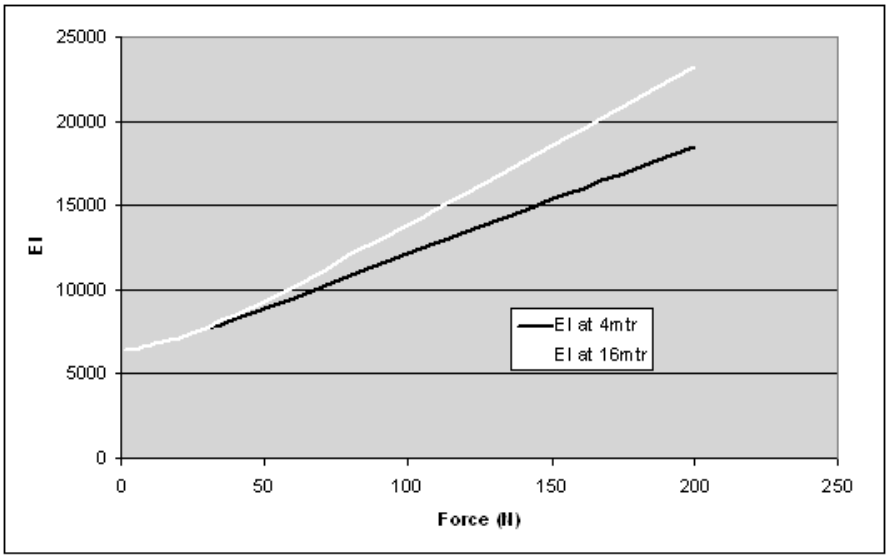

Figure 26: Graph of EI difference between 4 and 16 meter from second measurement

This model has the same kind of results as the first measurement, only it can be seen that the *EI* constant overall is smaller than the first model. It also can be seen that the *EI* constant increases when more force is applied. *EI* should be the same for each force, which is not the case. This beam is also weaker than the first model.

The difference in terms of percentage is also plotted, shown below.

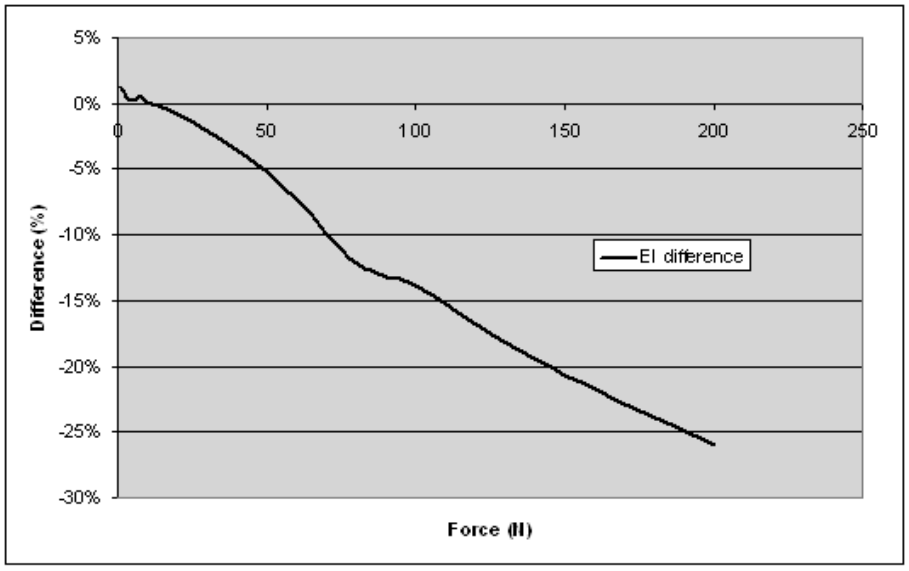

Figure 27: *EI* difference at 4m compared to EI at 16m from second measurement

This graph shows also that the linear formula is sufficient for a small force, less than 40N. When this force gets larger the differences of the *EI* constants become larger. This happens because the beam bends in the beginning and is only stretched at the end.
At the end the difference in terms of percentage is larger because the beam is weaker than the beam used in the first measurement, so the difference is also greater.

The conclusion for this test is that the model is not sufficient to model a beam, because the *EI* constant is not the same at each point of the beam and the *EI* constant changes when the force changes. The linear formula is sufficient when small forces are used but for larger forces the formula is not sufficient enough. For small deflections the *EI* should be constant using the deflection formula, this is not the case, therefore the model is not sufficient enough. When a weaker beam is modeled the differences between *EI* become even larger with large forces. With a stiffer beam these differences are not that large. So the model works for stiff, strong beams with a small force. The difference of the *EI* constant between the beginning of the beam and the end is caused by a large force. The bend point is in the beginning of the beam, the end is therefore more stretched than bended and this causes the *EI* constant to become smaller at the beginning of the beam than at the end of the beam, this effect cannot be described with the deflection formula.
The overall conclusion is that the model is too simple to model a real beam. A beam therefore cannot be modeled with simple point masses and stiffnesses, but has other aspects. This model is also a 2-dimensional model for a 3-dimensional beam. This also could lead to differences in the *EI* constant for larger forces.

4.2.3 Dual supported beam

Secondly a model is tested with both sides supported. One side is supported and the other side can only move in horizontal direction. A figure of this situation is shown below.

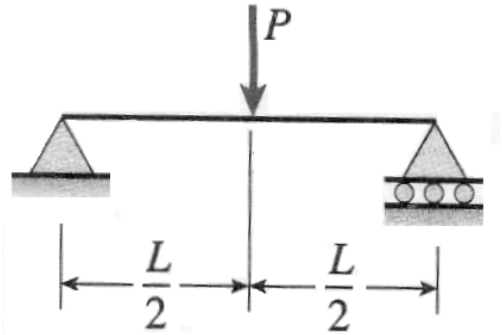

Figure 28: Dual supported beam

It can be seen that the left side can move horizontally. The model is carried on the bottom right node and the bottom left node. The model which is used is the same as the test with a single clamped beam, shown in Figure 22: Single clamped beam model.
The deflection for this situation can be calculated with:

$$v = \frac{Px}{48EI}\left(3L^2 - 4x^2\right) \quad 0 \le x \le \frac{L}{2}$$

with v the deflection in the y (vertical) direction (positive upward), P the applied force, L the length of the beam, x the position of the deflection and EI the material constant. The formula only works for small deflections. This formula can be rewritten to calculate the material constant, EI, to

$$EI = \frac{\frac{Px}{48}\left(3L^2 - 4x^2\right)}{v}$$

With this formula the EI is calculated at several points of the beam. The figure below shows the points where EI is calculated.

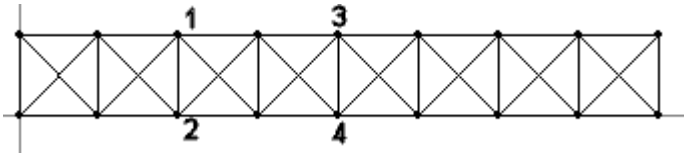

Figure 29: Points where EI is calculated

The force is applied downwards on point 4 of the beam. At point 1,2,3 and 4 the EI constants are determined. The results of the measurement are shown in Table 6. In this table the relative difference between point 3 and 4 is shown. This difference represents the EI difference between those points which is caused by the force which is applied on point 4. In the table the relative difference between point 1 and 3 is also shown. This difference represents the uniformity of the beam.

Force (N)	Deflection at (mtr)					EI at (mtr)				EI difference between	
	1	3	2	4		1	3	2	4	3 and 4	1 and 3
5	0,024	0,034	0,024	0,035		12222	12549	12222	12190	3%	-3%
10	0,048	0,069	0,047	0,070		12222	12367	12482	12190	1%	-1%
20	0,095	0,138	0,095	0,140		12351	12367	12351	12190	1%	0%
40	0,192	0,278	0,191	0,282		12222	12278	12286	12104	1%	0%
60	0,291	0,419	0,288	0,425		12096	12220	12222	12047	1%	-1%
80	0,392	0,561	0,386	0,569		11973	12169	12159	11998	1%	-2%
100	0,493	0,705	0,484	0,715		11900	12104	12121	11935	1%	-2%
120	0,596	0,849	0,583	0,861		11812	12061	12075	11893	1%	-2%
150	0,752	1,066	0,731	1,081		11702	12008	12038	11841	1%	-3%
200	1,015	1,424	0,978	1,448		11560	11985	11997	11786	2%	-4%
500	2,511	3,425	2,298	3,465		11682	12457	12765	12314	1%	-6%
1000	4,159	5,553	3,580	5,613		14106	15367	16387	15203	1%	-8%
1500	5,011	6,667	4,152	6,729		17561	19199	21195	19022	1%	-9%

Table 6: Dual clamped beam results (1)

The graph of these results are shown below.

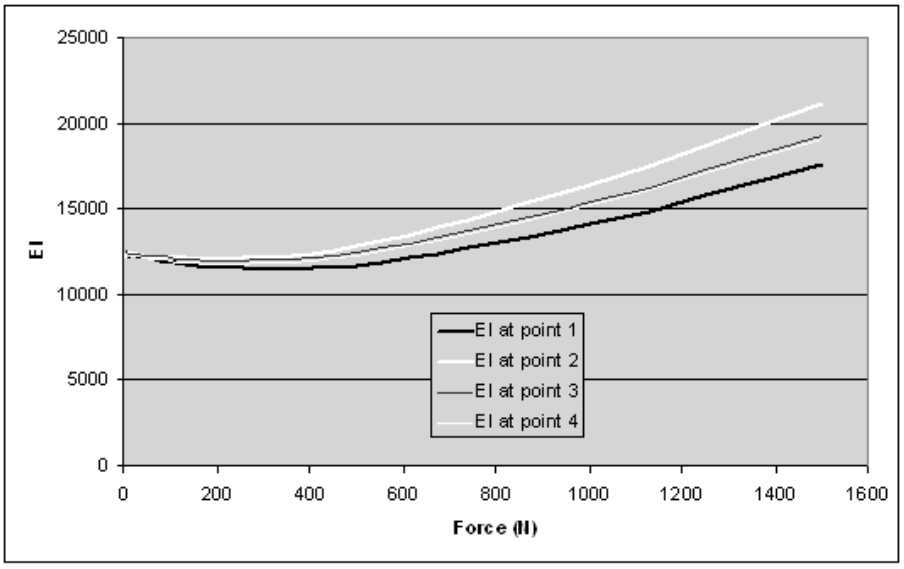

Figure 30: Graph of first results

Second the *EI* differences between point 2 compared to point 1 and point 3 compared to point 4 are plotted. The difference between these values should be zero.

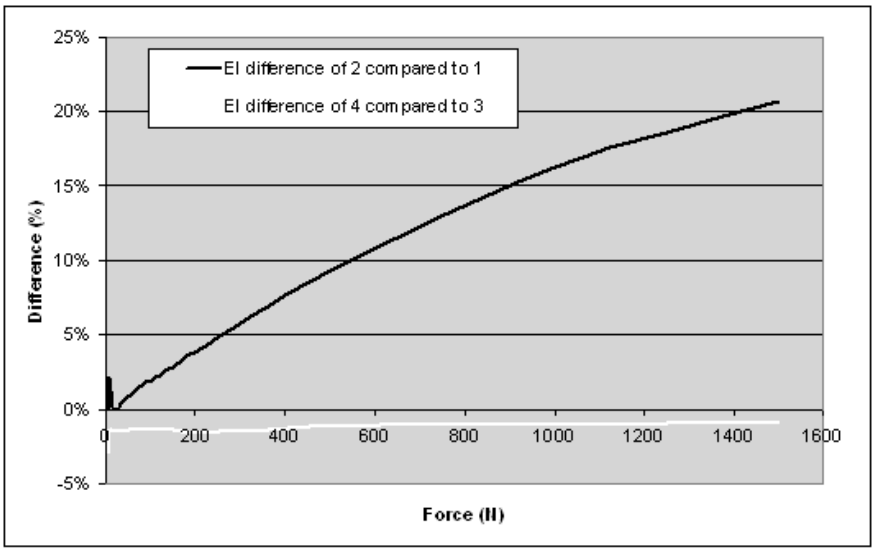

Figure 31: *EI* Difference between 1 & 2 and 3 & 4

The difference of *EI* between point 4 and 3 is minimal, therefore the force is not deforming the beam much in vertical direction. The difference of *EI* between point 1 and point 2 gets larger while the force gets larger. This happens because point 1 and point 2 do not only down but they also rotate around the most left point, where the beam is supported. This is shown in Figure 33. The difference is caused by this rotation.

Next the *EI* difference between point 1 and point 3 is plotted.

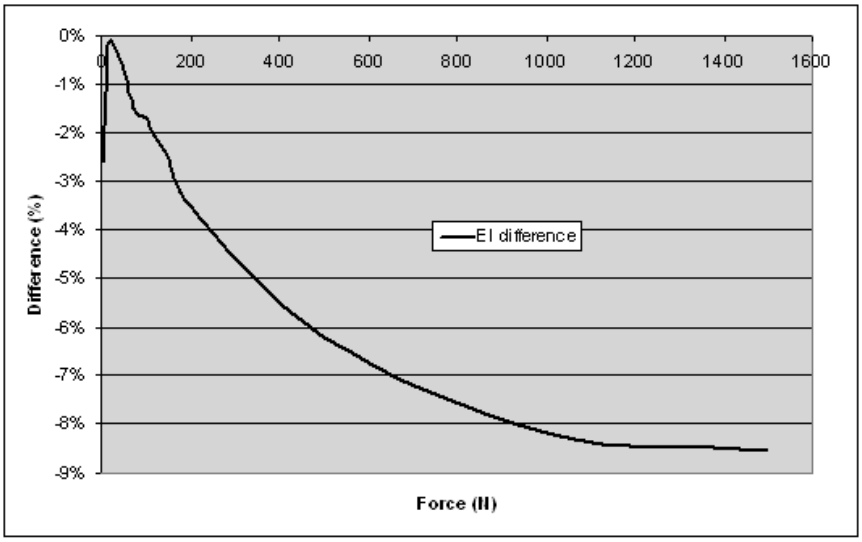

Figure 32: *EI* difference between point 1 and 3

The *EI* difference between point 1 and point 3 gets larger when the force gets larger. This is the same as what happened to the single clamped beam. The model is also stronger when it is stretched than bended, therefore the *EI* constant is larger at point 3 than at point 1. The bending happens mostly at point 1, point 3 is not bended that much. This is shown below.

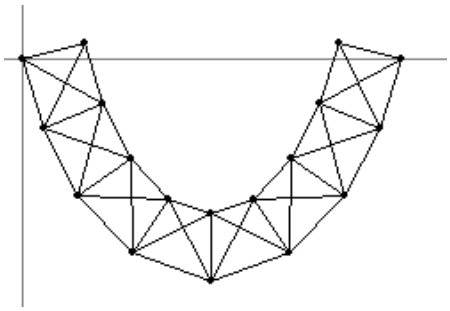

Figure 33: dual clamped beam with 1500N force

Point 1 is the weakest point of the beam due to the bending, therefore the material constant *EI* will relatively decrease compared to the *EI* constant at point 3.

This model is also tested with the force applied to point 3 instead point 4. These results are shown below.

Force (N)	Deflection at (mtr)					EI at (mtr)				EI difference between	
	1	3	2	4		1	3	2	4	3 and 4	1 and 3
5	0,024	0,035	0,024	0,034		12222	12190	12222	12549	-3%	0%
10	0,047	0,070	0,047	0,069		12482	12190	12482	12367	-1%	2%
20	0,095	0,140	0,095	0,138		12351	12190	12351	12367	-1%	1%
40	0,193	0,282	0,191	0,278		12159	12104	12286	12278	-1%	0%
60	0,292	0,426	0,288	0,419		12055	12019	12222	12220	-2%	0%
80	0,392	0,571	0,386	0,562		11973	11956	12159	12147	-2%	0%
100	0,494	0,717	0,484	0,705		11876	11901	12121	12104	-2%	0%
120	0,597	0,864	0,583	0,849		11792	11852	12075	12061	-2%	-1%
150	0,753	1,086	0,731	1,068		11687	11786	12038	11985	-2%	-1%
200	1,017	1,457	0,978	1,431		11537	11714	11997	11926	-2%	-2%
500	2,53	3,533	2,308	3,447		11594	12077	12709	12378	-2%	-4%
1000	4,299	6,155	3,657	5,778		13647	13864	16042	14769	-6%	-2%
1500	INSTABLE										

Table 7: Dual supported beam results (2)

The graph of the second measurement results is shown below.

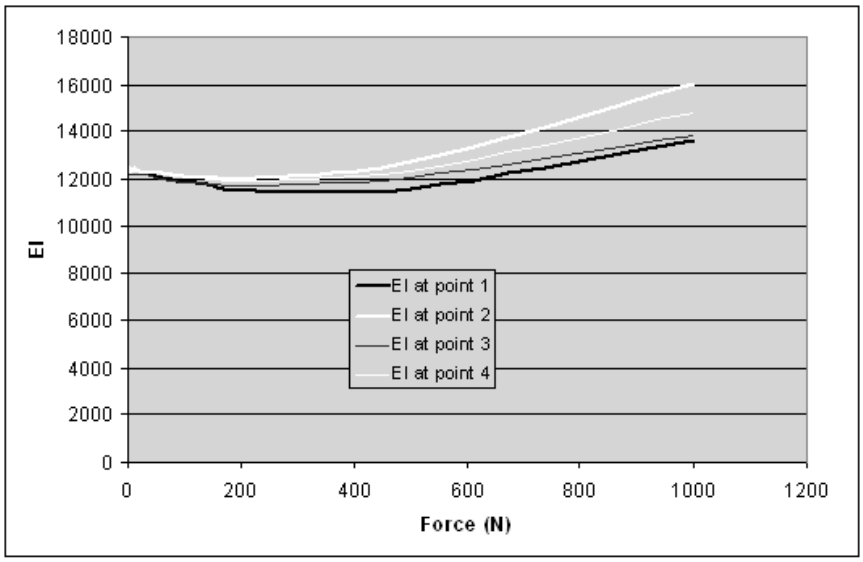

Figure 34: Results of second measurement

The beam responds more or less the same. The applied force on node 3 causes that point 3 deflects more than point 4. If the force gets too large, point 3 will pushed thru point 4, the model will become unstable. The difference between point 1 and point 3 is not as large as in the first test because the force is applied on point 3, therefore the beam will change a bit.

Next the *EI* difference between point 2 and point 1 and the difference between point 4 and point 3 is plotted. This is shown below.

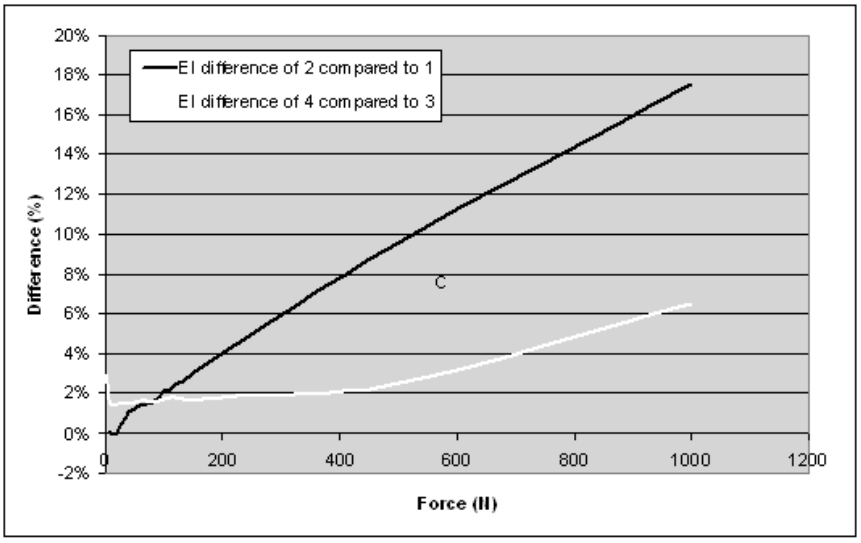

Figure 35: *EI* Difference between 1 & 2 and 3 & 4 for second measurement

The difference between point 4 and point 3 is increasing because the force is not applied on point 4 but on point 3. Therefore the *EI* difference between those points gets larger compared to the first measurement. The difference between point 2 and point 1 looks the same compared to the first measurement. This difference has also the same explanation.

The difference between point 1 and point 3 is also drawn.

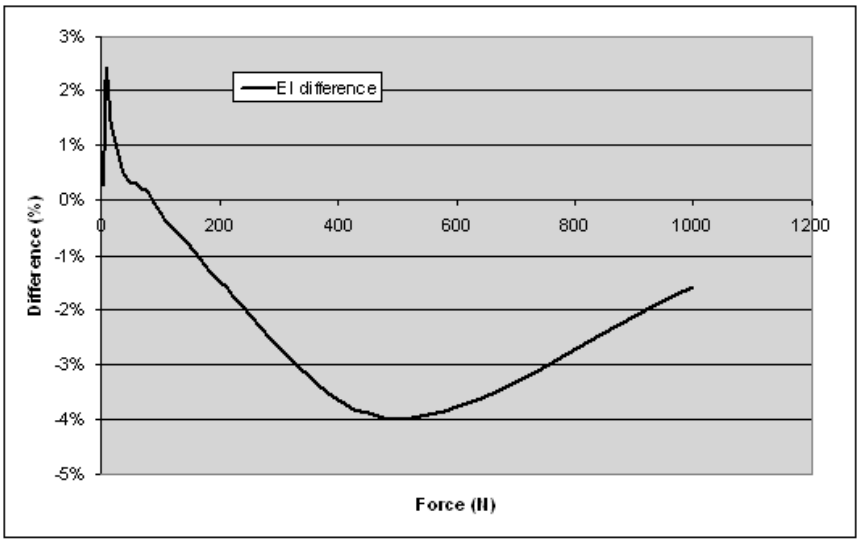

Figure 36 *EI* difference between point 1 and 3 for second measurement

This figure shows that the *EI* difference gets larger when the force gets larger. After 500N the difference decreases again. This happens because the force is applies on point 3 instead of point 4. The first part is the same as the first measurement. Because the force is applies on point 3 is pushed further down than in the first measurement. When point 3 is pushed further down the calculated *EI* will become smaller, therefore the difference become smaller after 500N. With a force less than 500N this difference cannot be seen because the beam is stiff enough to compensate this.

The result is that the model is useful as a real model. The *EI* constant changes about 11% from 5N till 1000N. The model however is not prefect and need to be altered to obtain more competent results.

4.3 Dynamic models

4.3.1 Mass on a spring

The first dynamic model is a simple mass on a spring, shown in the picture below.

Figure 37: Mass on a spring

The model has gravity enabled and the spring is in rest. So when the simulation starts the mass must start to resonate and at the end it stops resonating with the spring stretched out.
The results are shown in the graph below.

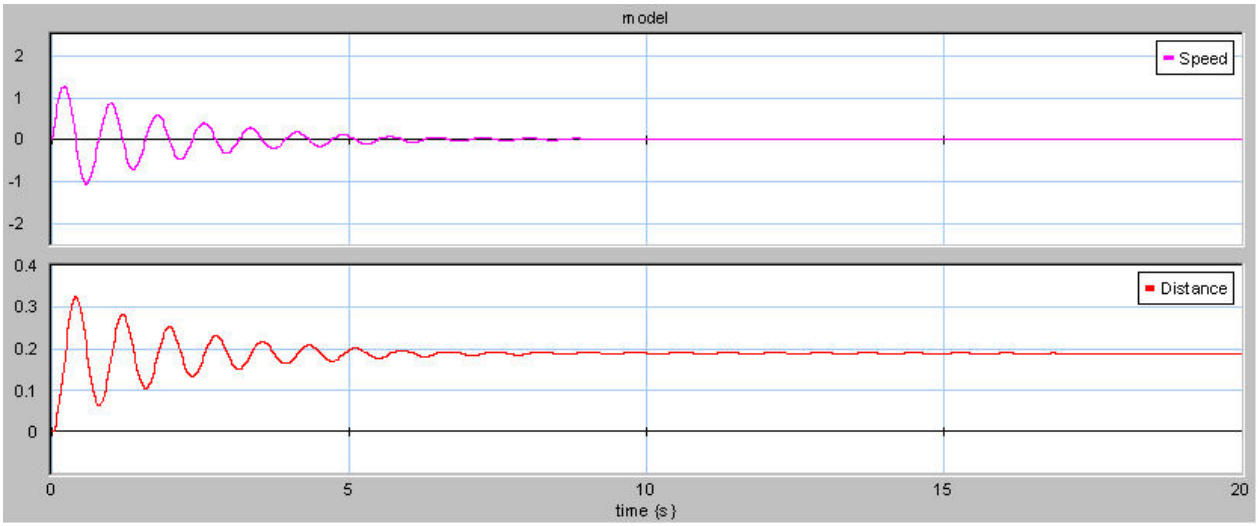

Figure 38: Results of mass-spring

The main application shows that the spring is stretched by 0.189m. To see if the results are correct the model can be calculated.
The force on the mass is

$$F = ma = 9.81*10 = 98.1$$

The force on the spring is defined as

$$F = 2k(x^2 - d^2)x = 200(x^2 - 1)x$$

So the stand at ease is

$$100(x^2 - 1)x = 98.1$$

$$x = 1.89$$

The spring force is not the force as expected, $F = kx$, because the spring used has rest length. So if the spring gets smaller it also requires a force. Therefore the 4th order spring force is used. This function is plotted Figure 39.

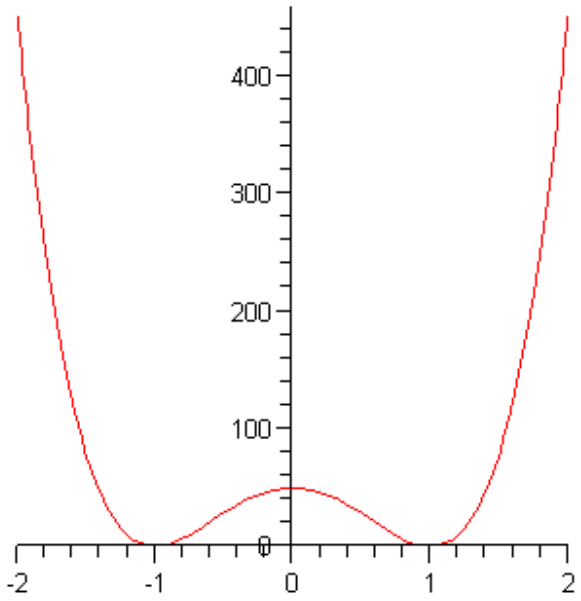

Figure 39: Graph of 4th order spring energy, *d*=1

The plot has horizontal the length of the spring, with 1 as rest length. The 'spring constant' *k* is 100Nm. The function plotted therefore is $V = \frac{1}{2}k\left(l^2 - d^2\right)^2 = 50\left(l^2 - 1\right)^2$. This clearly shows energy is needed to push the spring in. Because the energy function is a 4th order function, the function changes a lot which results in a much higher 'spring constant'. This is shown in the plot below where the *d* is increased to 2 and the function become: $V = 50\left(l^2 - 4\right)^2$

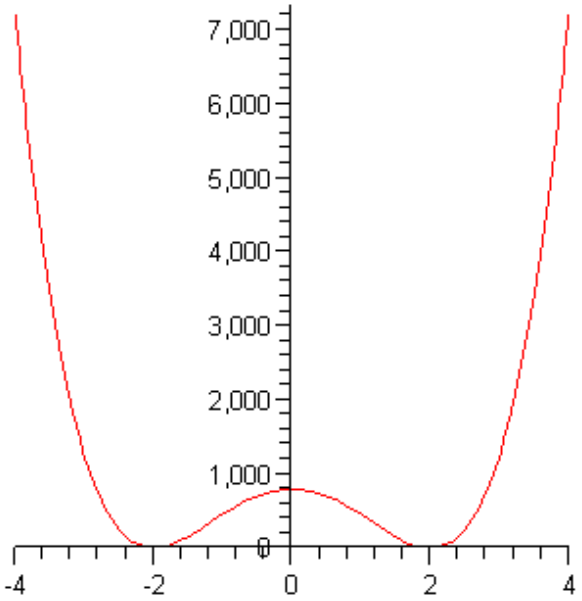

Figure 40: Graph of 4th order spring energy, d=2

There is much more energy required so the spring will not stretch 2 times more than a spring with length 1 but only a little bit. The result when using the spring with *d* is 2 is shown in Figure 41: results when *d* is 2.

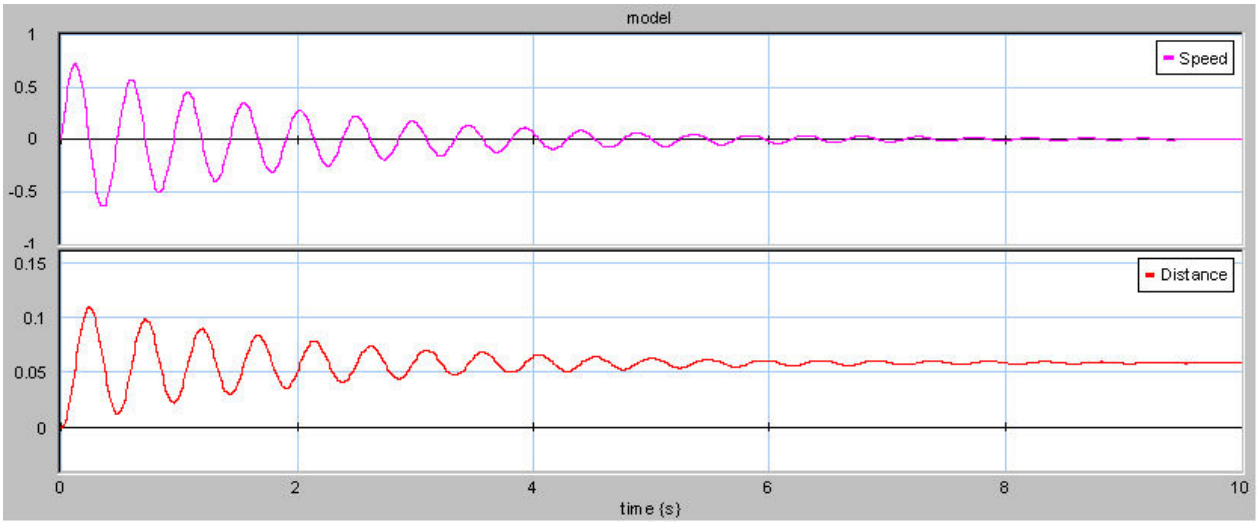

Figure 41: results when *d* is 2

The spring is lengthened by 0.059m and therefore the actual spring constant is much higher.

This can be shown with the following equation $V = \frac{1}{2}k\left(l^2 - d^2\right)^2 = \frac{1}{2}k\left(l+d\right)^2(l-d)^2$. The actual spring constant will increase by a factor $\left(l+d\right)^2$ in comparison with a one dimensional model, the spring will not lengthen more but less. So the results are as the model would expect.

4.3.2 Mass on a chain

The next model which is tested is a mass on a chain. The chain consists of 11 nodes, the space between each node is 2m. Each node has a mass of 0.2kg and at the end a mass of 10kg is mounted. The chain gets a momentum by a force of 1000N which is applied to the left side for 0.3seconds at the end mass. The model of the beam is shown below. The top node is located at the location (x,y) (0,22) and is fixed. The model is simulated with the Euler integration method with a step size of 0.001 second.

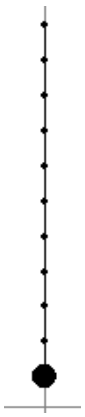

Figure 42: Begin state of mass of a chain

Due to the gravity force the chain will extend a little, this rest position is the begin position when the force is applied. The chain is lengthened by 0.33m, so the total length will be 20.33m.

After the momentum was applied on the beam the states of the bottom mass, kinetic and potential energy where measured at a several time intervals. A lot of results were obtained, because there are many results, these are shown in appendix I.

First the path of the moving mass is plotted. This is shown below.

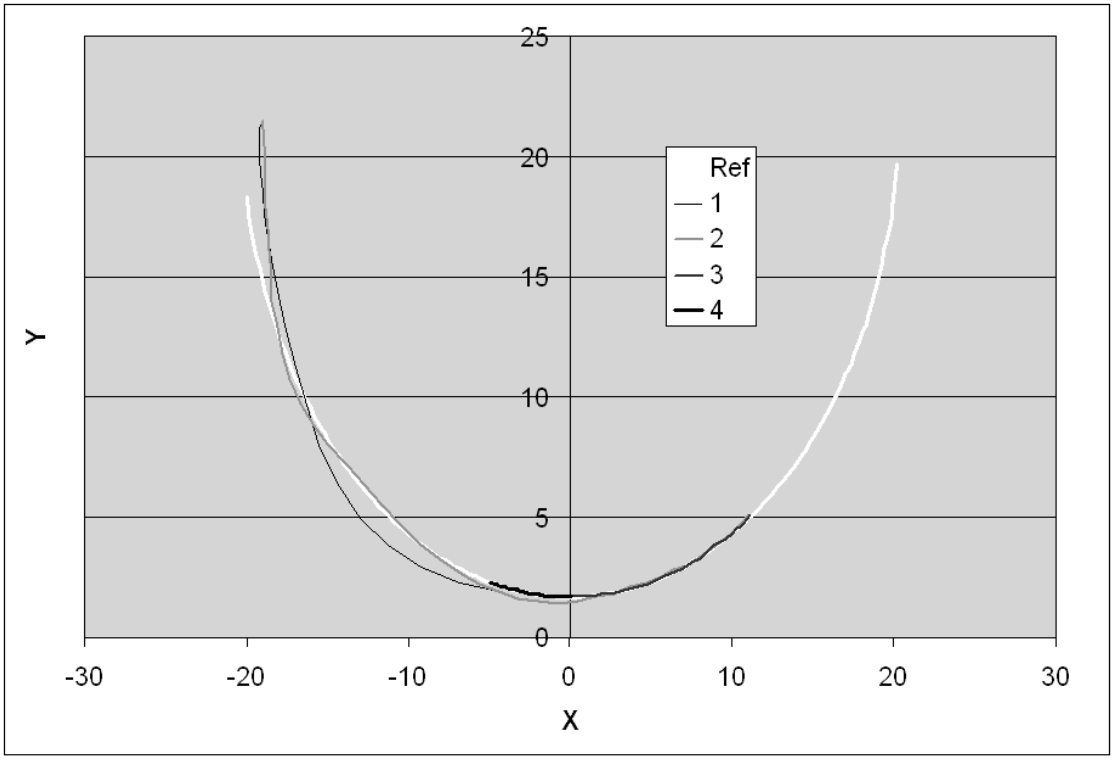

Figure 43: Chained mass, path

The thick white line is a reference path. This reference path is part of a circle with a center at (0,22) with a radius of 20.33.
The thin black line is the first half swing, from bottom to top. The measurement is started right after the force has stopped. The mass gets of course between x is -5 and 15. This happens because the force is applied in x-direction. The momentum causes the chain to stretch a bit and starts to resonate in axial direction. After a while it will shorten due to the stiffnesses in the chain. This result is shown between x is 15 and 18. A picture was also taken at this part. This picture was taken after 1.475 seconds of simulation and is shown below.

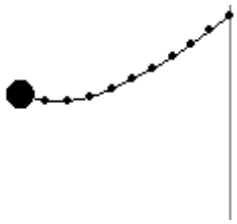

Figure 44: Chained mass during simulation

This picture shows that the chain is shortened due to momentum it was given at the beginning. The chain in this figure does not have the same kinetic energy as the end mass and it is a little slower than the end mass, therefore the chain is curved.

The second swing, from far left to far right, starts with a slightly shortened chain due to the first swing. Some resonating can be seen in axial direction around x is 18 and x is 14. At x is 0 it can be seen that the chain is a bit longer than the reference path. This happened due to the centrifugal force. The kinetic energy of the mass causes the chain to stretch at x is 0. When the energy is decreased the reference path is followed correctly.

The forth swing has so less energy that the chain will not deform anymore and the reference path is followed precisely. The forth swing stops at x is 0 and is slightly thicker than the third swing.

Next the radius of the chain is plotted during simulation. This is shown in Figure 45.

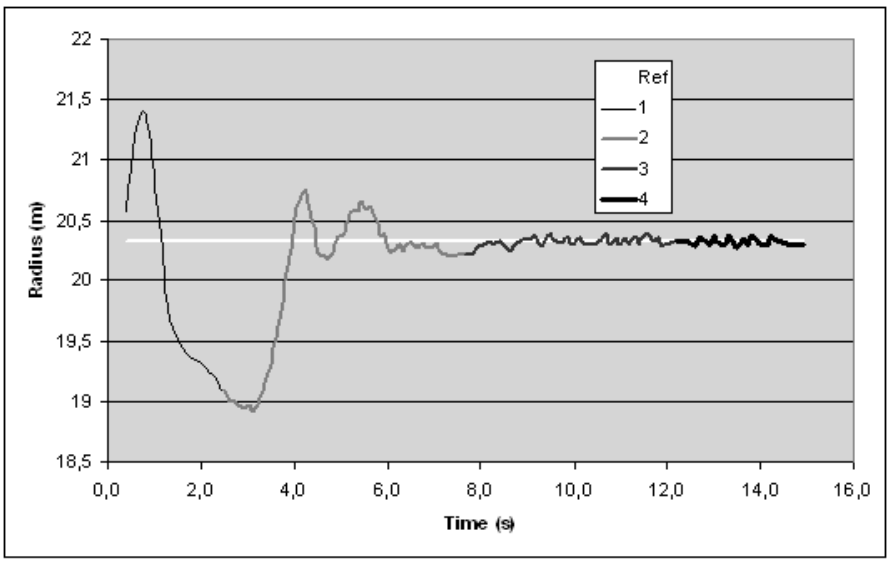

Figure 45: Radius of chain

The reference signal is the radius in rest, 20.33m. In the first period the resonation can be seen clearly. The resonation does not have a steady frequency. The frequency changes with the time. This is mainly caused by the dynamics of the chain and by the resonating of the nodes between themselves.

After 8 seconds the major resonation has stopped. The little vibrations after 8seconds are caused by a resonating between the nodes themselves which has not been damped by friction yet.

To determine the period of the swing the angle over time is plotted.

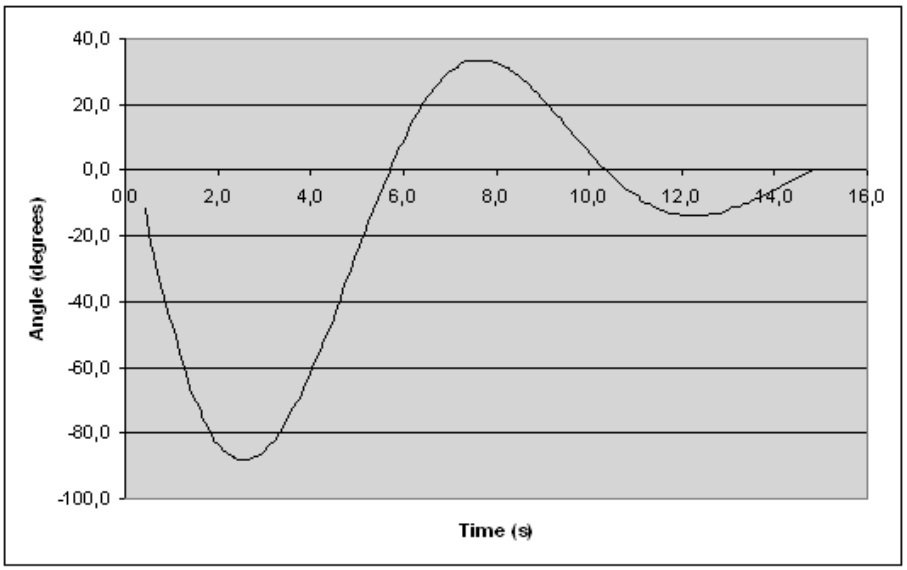

Figure 46: Angle over time

With this figure the swing period can be determined. The swing period is the period from the minimum at 2.6seconds and the minimum at 12.3seconds. This period is 9.7seconds. This period can also be determined by plotting the kinetic energy over time. This is shown below.

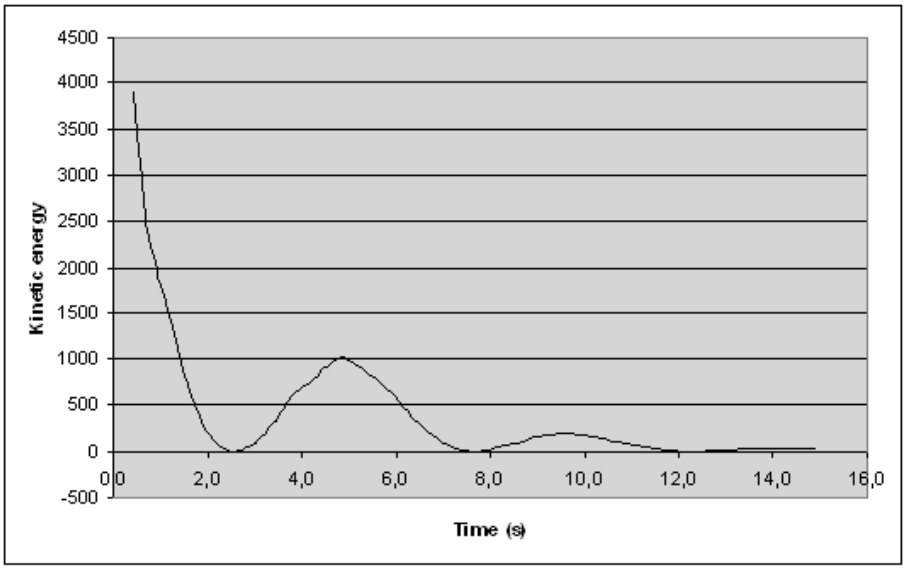

Figure 47: Kinetic energy over time

The period of a swing is the time between the first zero point and the third. These are located at 2.6seconds and 12.3seconds. This also gives a period of 9.7seconds. Both graphs gave the same result, as they should be. The second graph is only to check if this is the case.

Next the kinetic energy is plotted as function of the angle velocity. The velocity is calculated with the measured values of dx/dt and dy/dt. The velocity should be with squared onto the radius. But this is not the case due resonating in axial direction. So the actual velocity is measured and projected on a component squared on the radius. The difference between the original velocity and the one projected squared onto the radius is plotted in Figure 48.

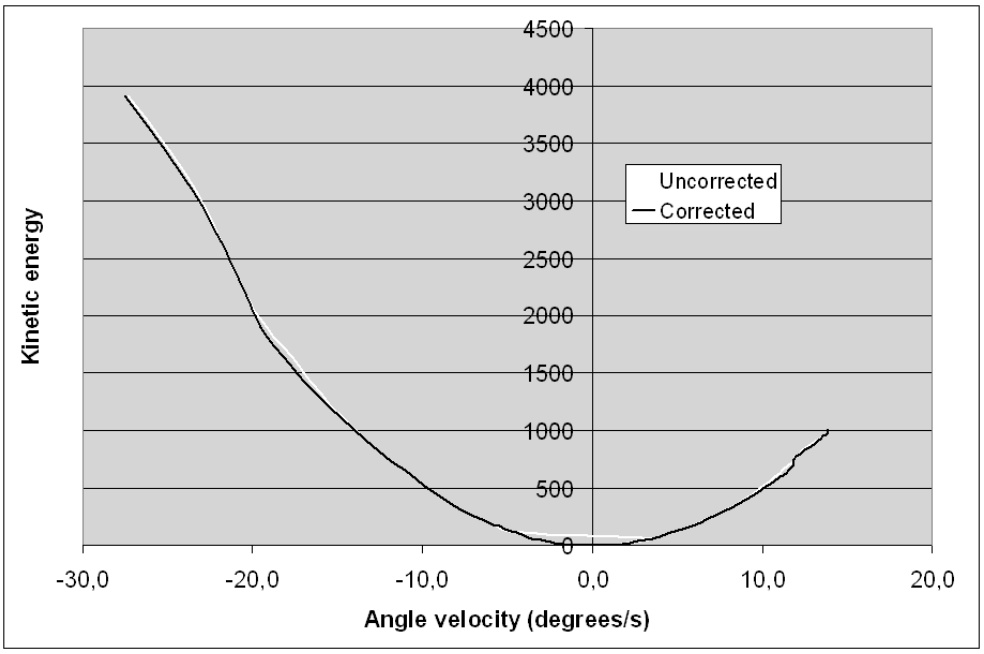

Figure 48: Corrected angle velocity

The angle velocity is positive when moving to the right and negative when moving to the left. There is a slightly difference between the corrected and uncorrected velocity around x is 0 and between x is -20 and -15. The corrected angle velocity is used for the next plot, which shows the kinetic energy for all swings.

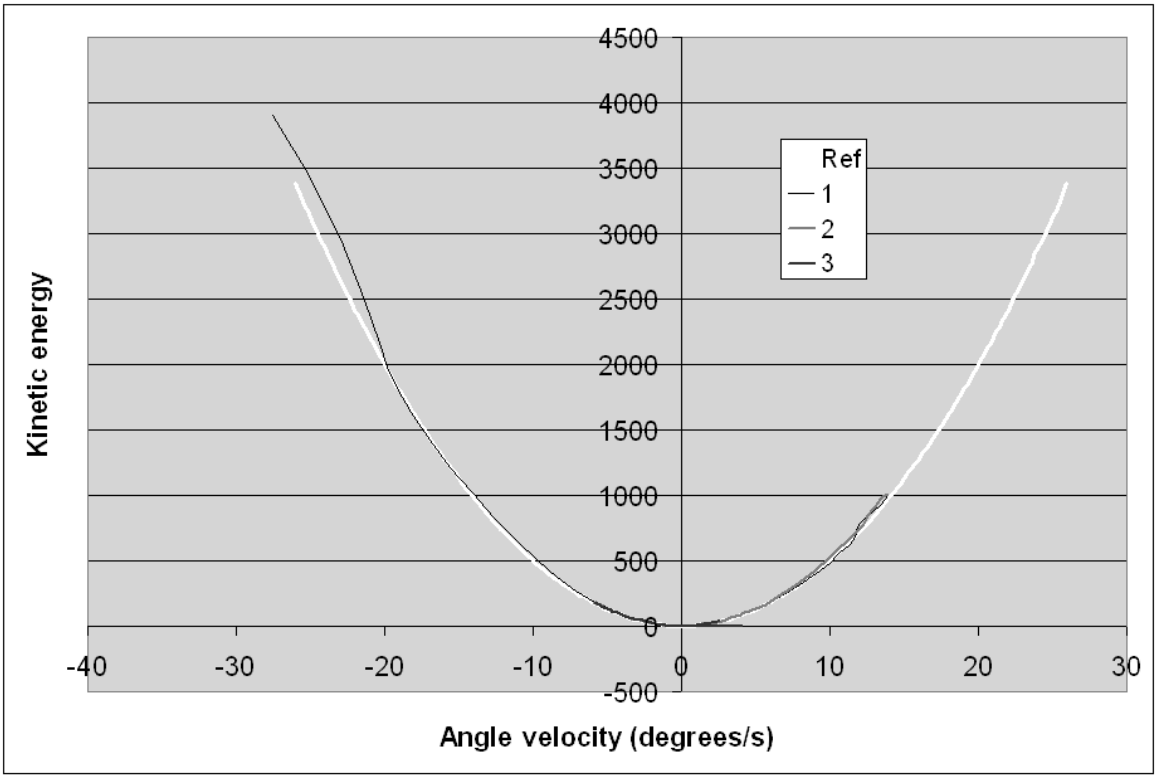

Figure 49: Kinetic energy versus angle velocity

The plot also shows a reference signal. This reference signal is defined as: $E_k = \frac{1}{2}10\dot{\theta}^2$.

The results were expected to be a parabola as the reference signal, which is the kinetic energy of a pendulum with a mass on 10kg. The kinetic energy is the largest when the angle velocity is the highest. The reference signal is the signal which was expected.
The plot shows a little bump in the first swing between the angle velocity of -20 and -27. This bump can be explained by the chain. When the angle velocity is high in the first swing, the whole chain starts to move. The whole chain has a mass greater than 10kg due to the nodes. This results in a kinetic energy which is larger than the reference signal at a high velocity. When the angle velocity is less this effect can be neglected. On the other side between the angle velocity from 5 till 12 the same effect can be seen.
The second swing is steadier than the first swing and follows the reference signal almost correctly.
The last swing shows a decreasing kinetic energy. This is the straight piece between the angle velocity 0 and 5. This happens due the friction of 1% in the model. At the end when the chain stops swinging, is the kinetic energy 0. This is shown after three swings. After three swings the model does not swing any more, the kinetic energy is gone due friction.

The reference signal represents the kinetic energy quite well. This model has a mass of 10kg. The sum of all masses in the system is 12kg. The dynamics of the chain has caused the mass will look a bit different than the reference model.

When the kinetic energy and angle velocity are known the mass also can be calculated by

$$m = \frac{2E_k}{\dot{\theta}^2}$$

The result of calculating the mass is shown in Figure 50.

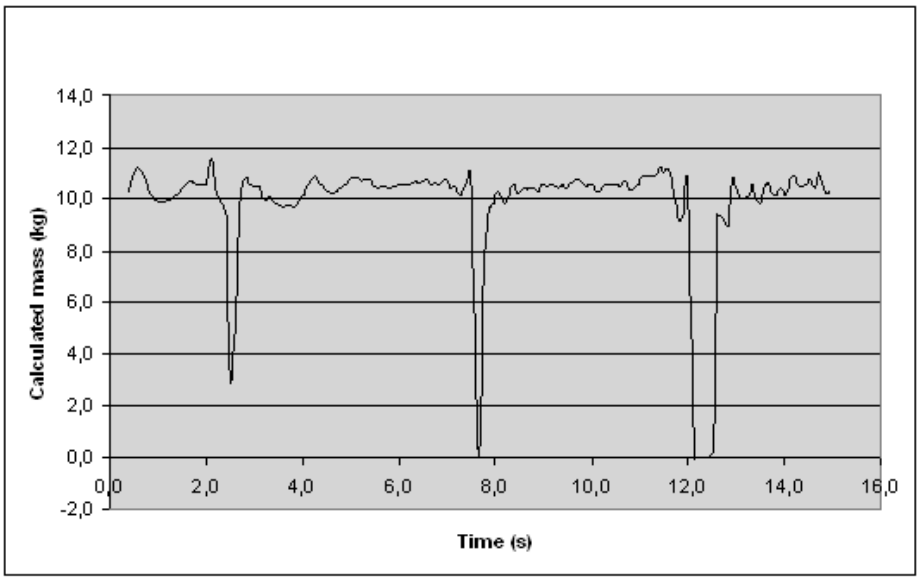

Figure 50: Mass calculated from the kinetic energy

The peaks down are caused by a kinetic energy which is 0 or almost 0, therefore the mass is not calculated correctly. The calculated mass should be a constant value, which must lie between 10kg and 12kg. The 12kg is the total mass of the system and the 10kg is only the end mass. Because not all mass is located at the end of the chain, the calculated mass must have a value between 10kg and 12kg.
The vibrations of the calculated mass are caused by the dynamics of the chain and the friction. The vibrations do not have a clear frequency and therefore they cannot be calculated.
The mass is calculated with the corrected velocity, the one squared on the radius, and therefore some extra wiggling can occur.

Next the potential energy is plotted as function of the angle.

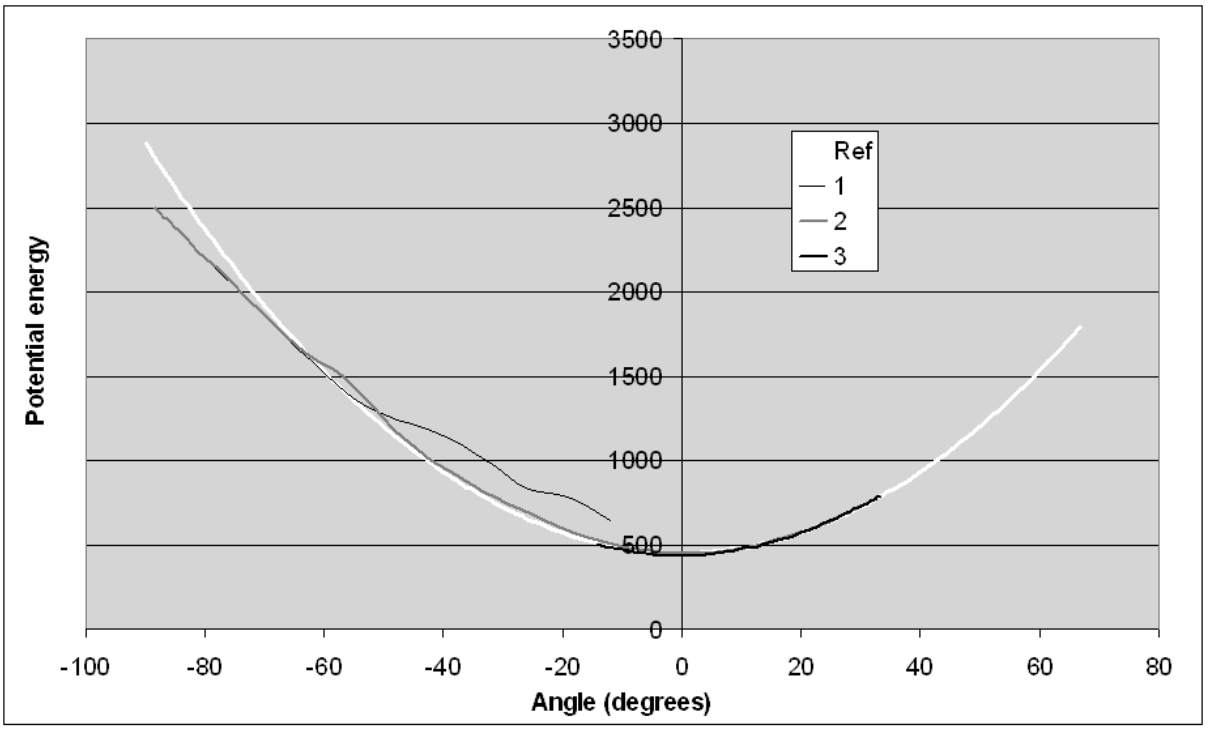

Figure 51: Potential energy versus angle

This plot has a reference signal plotted which is represented by: $E_p = 0.3\theta^2 + 450$. The potential energy of the model depends on the height of each node and the tension in the stiffnesses. If the tension is neglected the potential energy as function of the angle must be a parabola. The reference signal drawn is the parabola which matches the potential energy the most. The reference signal has a constant of 450 added because the chain contains potential energy in steady state due gravity force. The bottom is at zero degrees, top left at -90 degrees and top right at 90 degrees.

The first half swing, from bottom to top, is a bit wiggly in the beginning. The most common reason for this is because the momentum is given in negative x-direction which causes a wiggle in the beginning. After a while it stabilizes itself. But is does not follow the reference signal. At the end, the potential energy is less than expected. This can be explained by the chain. If the mass is at its highest point the chain will bend al little, so the potential energy is a bit less as expected. The chain will also shorten a bit due centrifugal forces. When the angle is small the chain is lengthened a little due the centrifugal force and therefore more potential energy is stored when is angle is small, from -40 till 40 degrees. The second swing, from top left to top right, starts to follow the reference signal. The third swing also follows the reference signal correctly.
The second swing does not follow the reference signal precisely; it is wider at the top and smaller at the bottom. A cause for this can be found in the chain itself, but the exact cause remains unknown.

4.3.2.1 Conclusion

The model which is used represents a mass on a chain quite good. Most of the results were as expected. However the model is not tested in more situations so it cannot be concluded if the model will work in other situations.
The bump in the kinetic energy plot was not expected, but because the chain had several nodes it could be explained. Some further testing have to be done to see how this effects exactly works. The calculated mass shows a wiggle which cannot be explained easily. To understand where this wiggle come form some further testing has to be done.
The potential energy is a bit wiggly in the beginning. A precise reason was not found. The starting impulse could have something to do with this. It could be an option to apply a ramp function on the mass instead of a step function. The ramp will smoothly increase the force and therefore the result should be smoother.

4.3.3 Mass on a beam

Next a beam is modeled with a mass at the end. The beam is build of segments with 4 nodes and 6 stiffnesses. All these segments are put together and at the end a big mass is mounted. All the stiffnesses between the nodes are 200Nm. The point masses are 0.2kg. The triangle, where the big mass is, is made with 400Nm stiffnesses. The mass at the bottom is 25kg. The model used is shown in Figure 52.

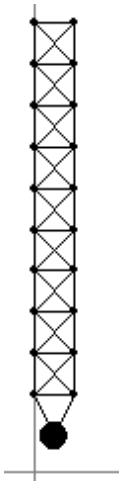

Figure 52: Mass on a beam, begin state

The top left node is located at (0, 22) and the top right at (2, 22). This model also works with gravity force. Therefore the model will lengthen a little due to the gravity force. The states of the lengthened model were taken as begin states for testing. The gravity force caused the end mass to be at (1, 1.7) instead of (1, 2), so the model is lengthened by 0.3meter.

A force of 2000N is applied on the model from 0.1 till 0.4seconds. This force is applied on the bottom mass in the left direction. After 0.4seconds the measurement is started. The simulation is done with an Euler integration method with a step size of 0.001seconds and a friction of 4% to stabilize the model and to reduce instabilities.

The results of the measurement which were taken are shown in appendix J, because a lot of results were obtained.

First the path of the end mass is plotted. This is shown below.

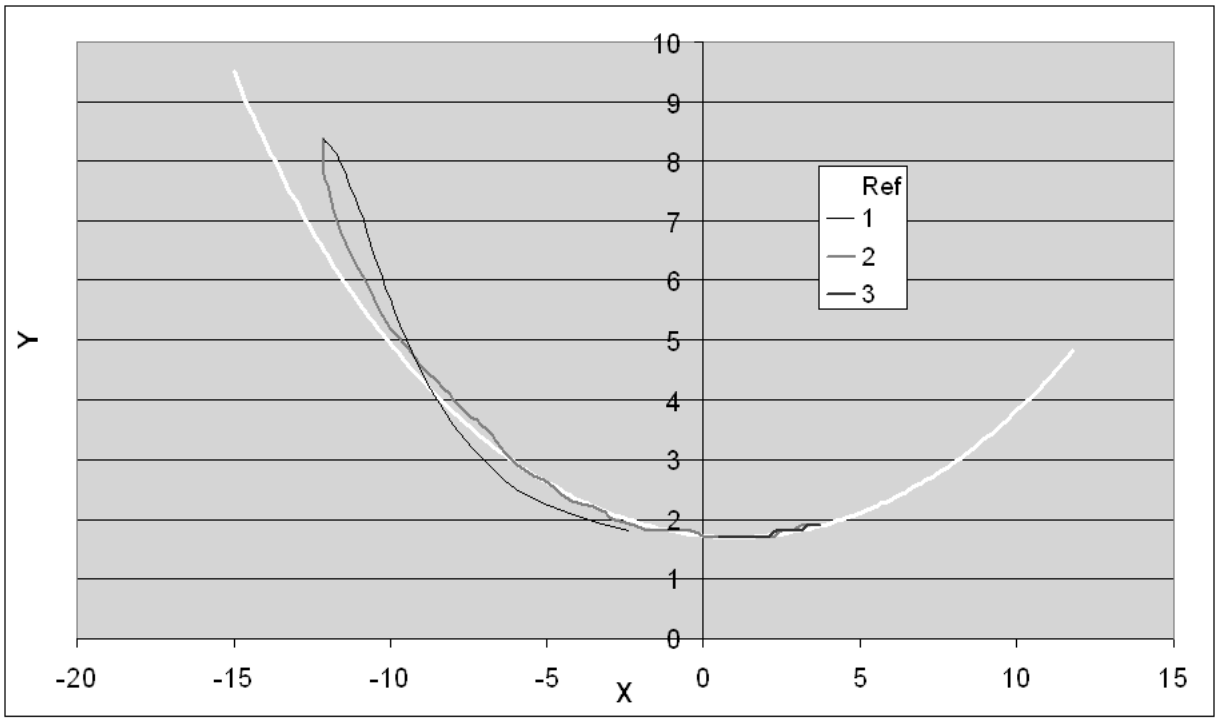

Figure 53: Path of moving mass onto a beam

The white line is a reference signal which is a circle with a radius of 20.30 and a center at (1, 22). The thin black line represents the first half swing, when the mass swings up. The radius will lengthen a bit, between *x* is -4 and -10. This happens because the mass gets a momentum in the negative x-direction and due to the inertia the radius gets a bit longer and the mass moves in x-direction instead of following the reference signal.
After a while the radius shortens a bit and starts to resonate in axial direction. This resonation is caused by the starting momentum. The resonating will stop on the swing back at *x* is -7. It also can be seen that the radius will shorten a lot in the highest point, at x is -12. This happens because the beam bends and this bending causes the radius to shorten.
The jerky plot on the left half of the figure is caused by big step between the measurement results, when the interval between the measuring was smaller this would be a smooth line.

The second swing, from left to right, has also some resonation in axial direction, this stops at x is -7. After x is -7 the swing will follow the reference signal. The beam is lengthened a little at the bottom, this is caused by the centrifugal force. A picture was taken at this moment.

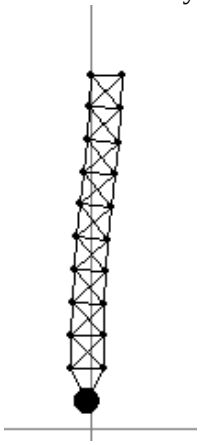

Figure 54: Mass on a beam after 4.137seconds

A little bending can be seen in the picture. The radius is also shortened at the most right, this happens due to the bending. This effect also occurred in the first swing.

The last two swings follow the reference signal well and the bending of the beam has only little effect on the radius of the beam.

Next the radius over time is plotted, this shows how the beam is resonating axially. This is shown below.

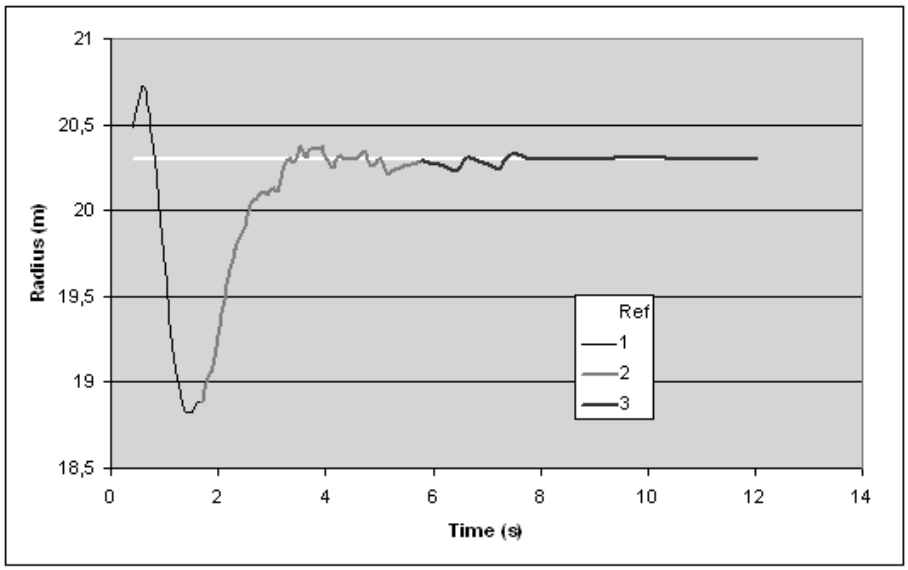

Figure 55: Radius over time

The reference is the radius of the mass of the beam in steady state. This reference signal must be reached when the model stops swinging. The beam is stretched a bit due to the momentum given at the beginning. After the second swing the beam will shorten and starts to resonate in axial direction. Between 3 and 8seconds the beam wiggles a little, this is caused by the dynamics of the beam. After 8seconds the beam has stabilized itself and the radius remains constant, the steady state has reached. The axial resonating is damped quickly because the model has a friction of 4%.

Next the angle over time is plotted to determine the swing period. This is shown in Figure 56.

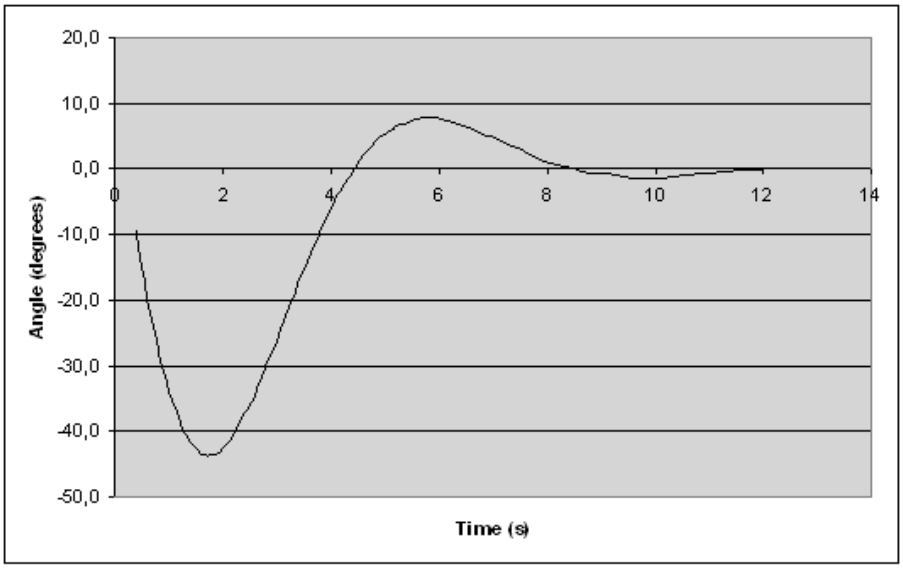

Figure 56: Angle over time for the mass on a beam

The swing period of the mass on a beam is determined from the zero crossing at 4.4seconds and at 12seconds. The swing period therefore is 7.6seconds. The damping has caused the beam to stop swinging after 12seconds.

Next a plot of the kinetic energy versus the angle velocity is made. This model has the same problem as the chained mass model, the velocity was not always in the same direction as the one squared onto the radius. Therefore the velocity was projected on a vector squared to the radius to get the correct angle velocity. The difference between the corrected and uncorrected angle velocity is shown below.

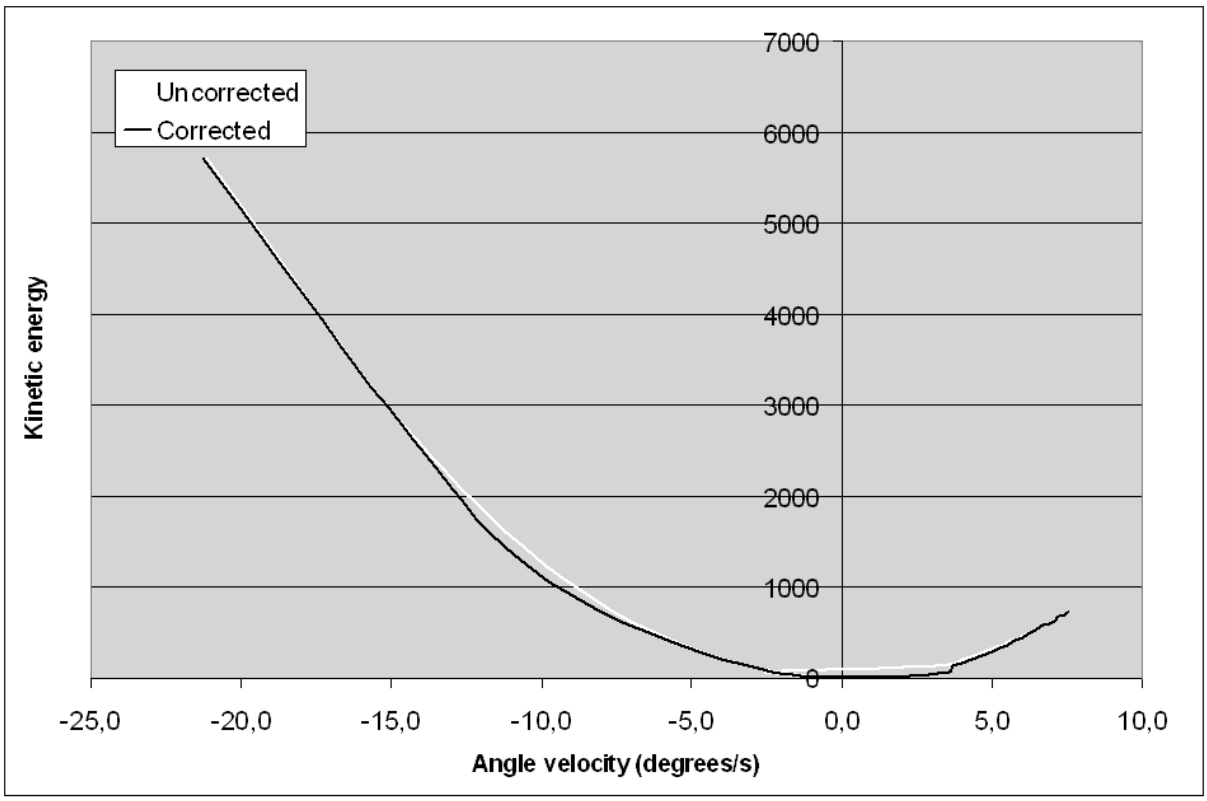

Figure 57: Corrected angle velocity with a mass on a beam

Both signals are most of the time the same. This happens because the velocity is most of the time squared on the radius. The corrected angle velocity is used to make a plot of the kinetic energy versus the angle velocity, which is shown in Figure 58.

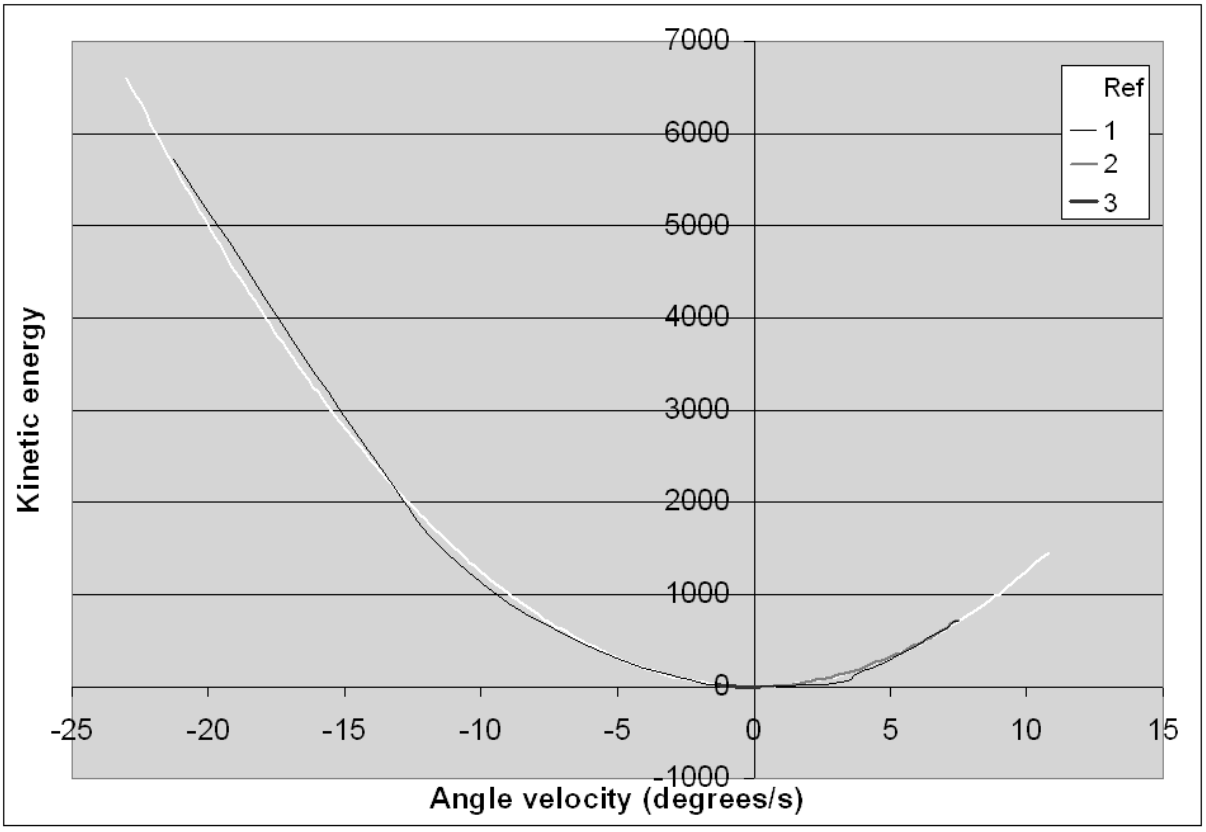

Figure 58: Kinetic energy of the mass on a beam

A parabola is expected and the result also shows a parabola. The reference signal is defined as $E_k = \frac{1}{2} 25 \dot{\theta}^2$, which represents the kinetic energy of a pendulum with a mass of 25kg.

The first half swing, the thin black line from an angle velocity of -20 till 0 degrees/s, shows some similarities with the mass on a chain. First the kinetic energy will increase while the mass slows down. It does not follow the reference signal clearly due the dynamics of the beam. The beam resonates axially a little, therefore it does not follow the line correctly.
At an angle velocity of 2 degrees/s a wiggle occurs. The exact reason for this wiggle is hard to understand. A common reason for this wiggle is the axial resonation. The axial resonation can be caused by the velocity, which it not the actual velocity, but the velocity which is projected squared on the radius. These velocities are a little different which can cause the resonation. This resonation can also be caused by the dynamics of the beam. This effect must be tested further to see where these results come from.

The second swing shows what is expected; the kinetic energy increases when the angle velocity increases.

When the kinetic energy and the angle velocity are known the mass also can be calculated with:

$$m = \frac{2E_k}{\dot{\theta}^2}$$

This is plotted in Figure 59.

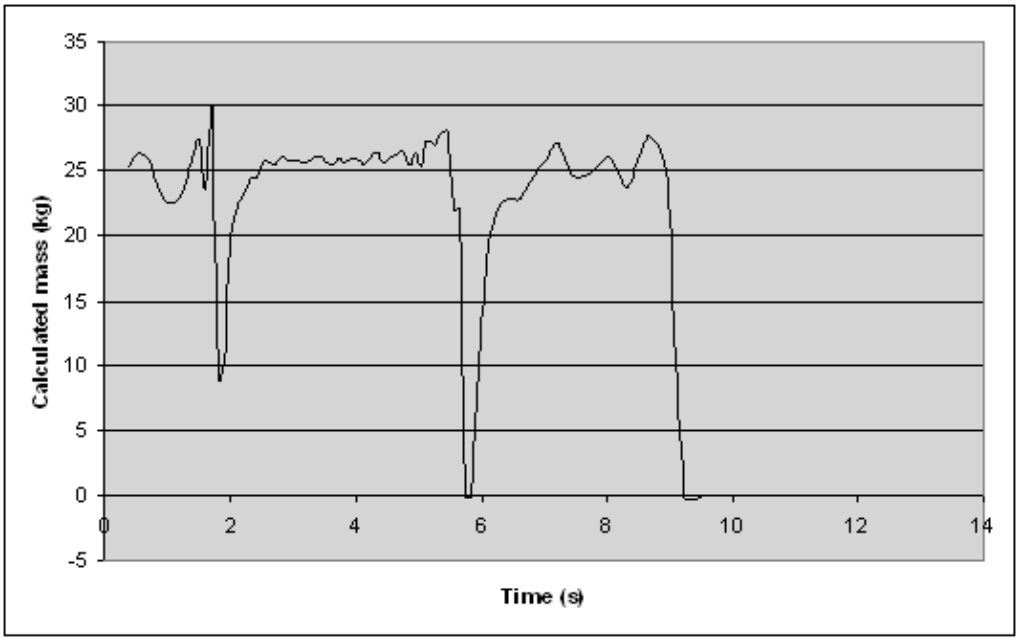

Figure 59: Mass calculated from kinetic energy

The peaks down are caused by the kinetic energy which is 0 or almost 0, therefore the wrong mass is calculated. After 9seconds the kinetic energy remains 0, the beam has stopped swinging. Between 0.4 and 0.8seconds a periodic resonation seems to appear. At 0.8seconds this resonation frequency is increased. This effect is caused by the dynamics of the beam. The momentum, which is given at the beginning, caused an axial resonation. The resonation causes the kinetic energy to change a little which result in a vibration in the calculated mass.
The vibrations after 2seconds are caused by the dynamics of the model, the friction and the projection of the actual velocity squared onto the radius. This corrected velocity, which is used to calculate the mass, also has some influences in these vibrations.
Finally the mass should reach a value between 25kg and 29kg. 25kg is only the mass at the end of the beam and 29kg is the sum of all point masses in the beam.
The small vibrations must be investigated further to understand where these came from.

Next the potential energy versus the angle is plotted. This plot is shown below.

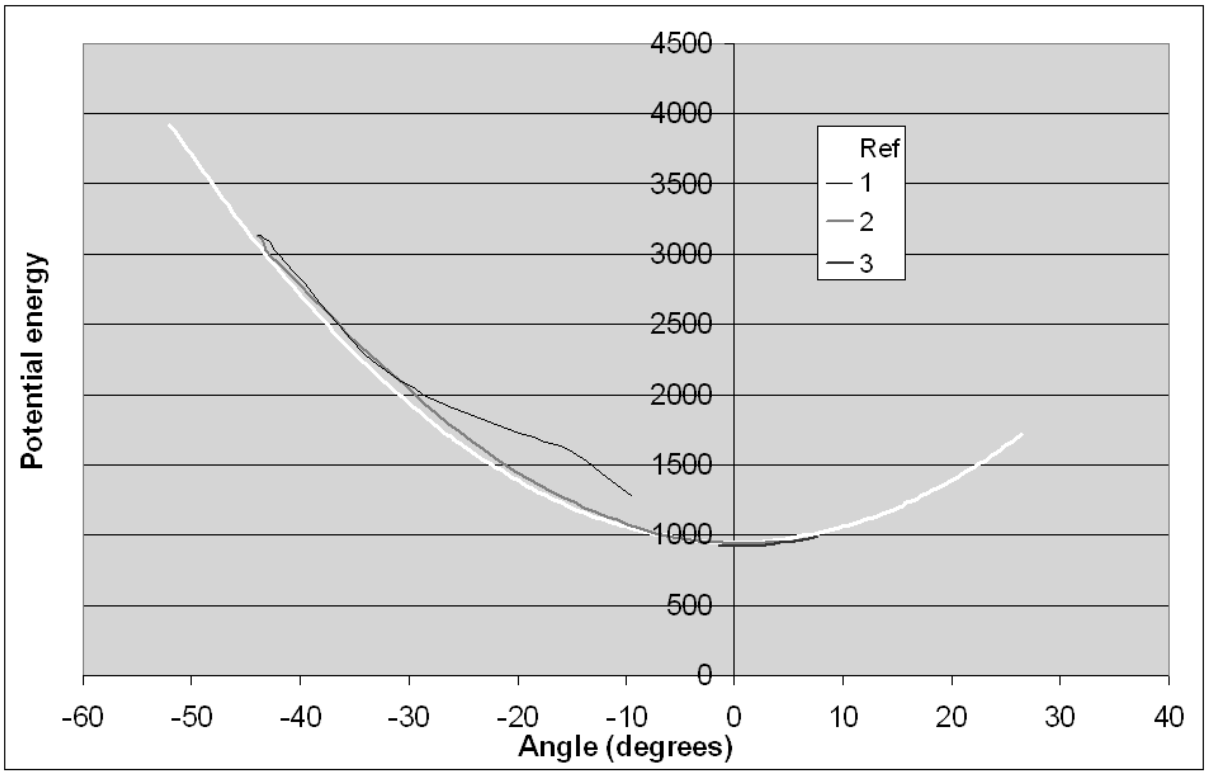

Figure 60: Potential energy of a mass on a beam

The expected result is a parabola. The parabola which matches the potential energy the best is described by the function $E_p = 1.1\theta^2 + 950$. The constant of 950 is added due to the gravity force, which causes a potential which is height depended. The potential energy is depended of the height of each node and the tension between the nodes. In this case the tension plays a much larger role than with the mass on a chain, because the beam is bending which causes the tension between the nodes.

The first swing does not follow the reference signal clearly. The potential energy is higher than the reference signal. This happens because a momentum is given to the mass at the bottom in the beginning. The mass starts to swing, which causes the beam to swing with the mass. This swing cause the beam to bend and this bending will increase the potential energy.

After the first swing the model will follow the reference signal well.

4.3.3.1 Conclusion

It can be concluded that the model works when a momentum is applied on a mass which is mounted to the beam. The reference signal is followed almost correctly, some differences occur due to the beam dynamics.
The other plots show results which also can be expected only the wiggle in the calculated mass and the potential energy must be investigated further to understand where the effects come from. The model also must be tested further to see if the whole model is corrected, even with another mass of stiffnesses.

4.3.4 Wave conducting in a beam

In this section a momentum is applied on a beam to obtain the wave conductance of the material. The model which is used is shown below.

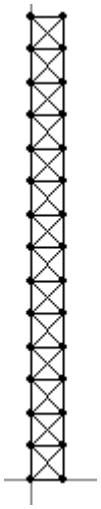

Figure 61: Wave conducting model

This model has two fixed nodes at the bottom. The two top nodes can only move vertically. Each stiffness is 200Nm and each point mass is 2kg except for the four corners, they have a point mass of 1kg. In the beginning a force is applied for 50ms. This results in a longitudinal wave. A picture of the model after a force is applied is shown in Figure 62: Wave conducting model with a wave.

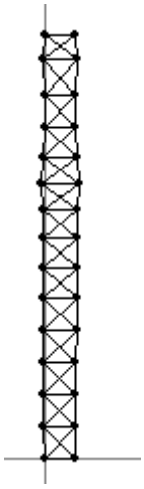

Figure 62: Wave conducting model with a wave

The wave is going downward. At the end it returns and goes back to the beginning. The model is tested by applying a force of 2200N for 50ms downwards, this force is applied after 0.01second. The speed and the distance in the y-direction of the two top nodes are monitored. The top-left node is node 15 and the top-right is node 16.

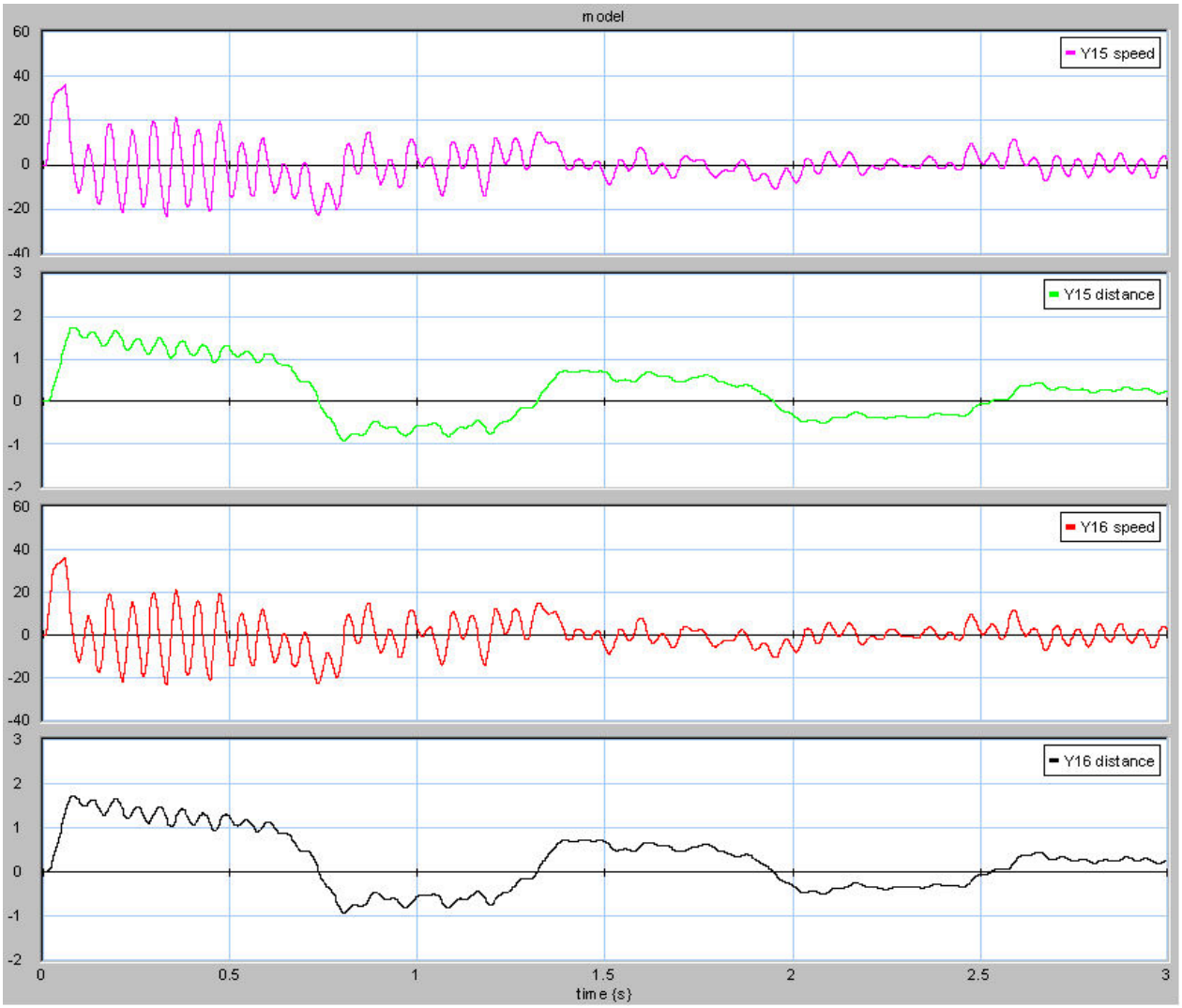

Figure 63: Wave conducting results

The result shows a high frequent resonation. These are the smaller waves in the distance curve. These are caused by two nodes that are resonating at a higher frequency with each other. This is not the longitudinal wave and this resonation must be ignored. The low frequent part is the longitudinal wave which has returned. The model is also tested with point masses of 4kg and the corners with 2kg. This result is shown in Figure 64.

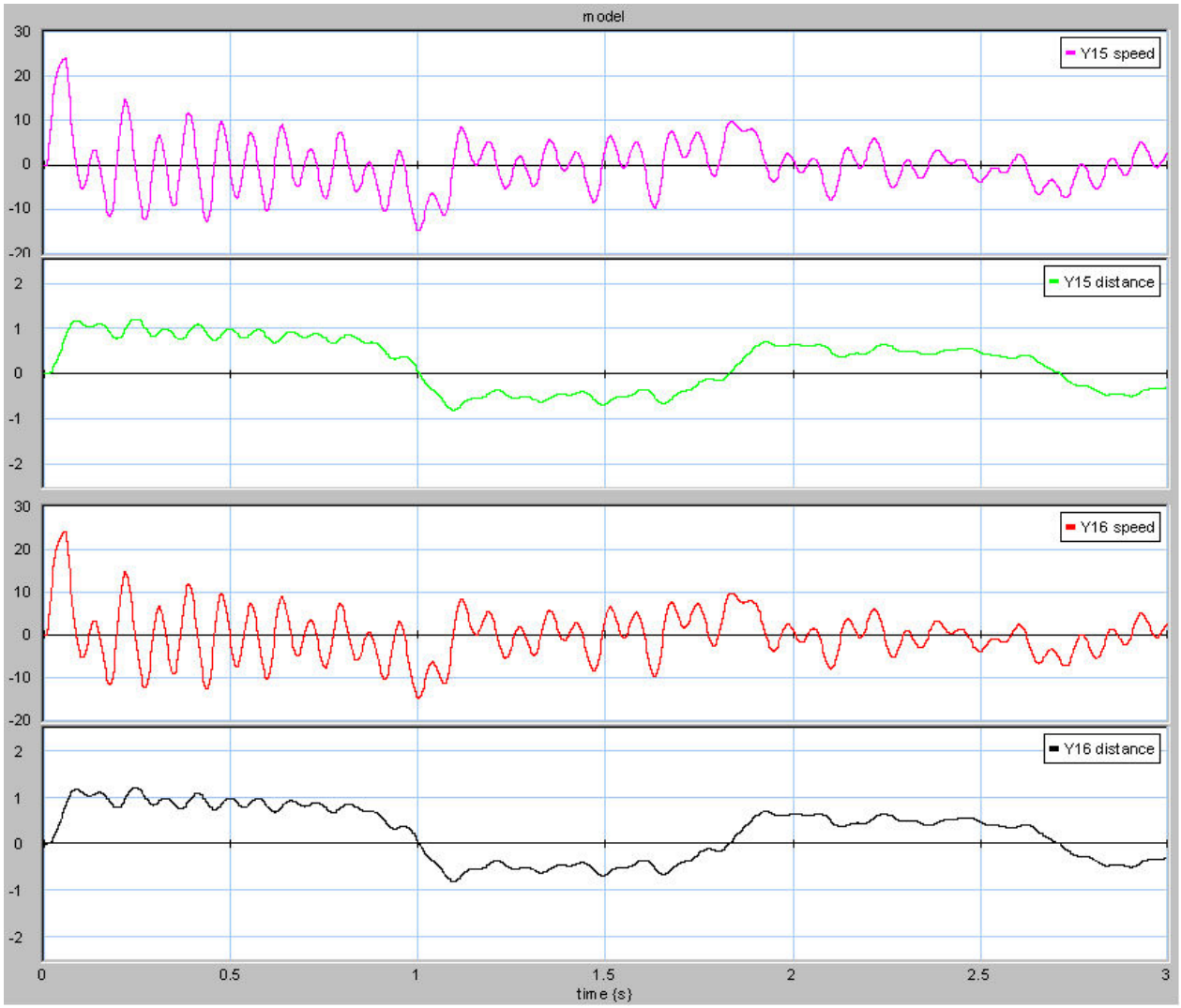

Figure 64: Results with 4kg masses

The frequency is much lower and the wave velocity is slower as expected. The wave velocity is calculated by dividing the length of the beam, which is 14m, by a half period. The results which where obtained from the model where compared to the wave equation:

$$c = \sqrt{\frac{EA}{m}}.$$

c is the wave propagation velocity, *EA* is a material constant and *m* the mass of the beam per meter. With all terms known, except the material constant EA, the formula can be rewritten as:

$$EA = mc^2.$$

So when the mass of the nodes changes, the wave propagation velocity will change but the *EA* term must remain the same. The results are shown in the table below. The *EA* column is calculated with the formula shown above.

Point mass(kg)	Total mass (kg)	Mass per length	Period(s)	Frequency(Hz)	Wave velocity (m/s)	EA
1	28	1	0,955	1,047	29,3	860
2	56	2	1,31	0,763	21,4	914
4	112	4	1,82	0,549	15,4	947
6	168	6	2,21	0,452	12,7	963

Table 8: Wave conducting results

It can be seen that the *EA* term changes a little bit. The changes can be caused by friction or a fault in the model. However the *EA* term does not change a lot, the model is sufficient enough.

When the point mass is less than 1kg the longitudinal wave cannot be seen and the wave is not conducted but damped. The results are as expected, when the mass of the beam increases, the inertia increases and the wave propagation velocity decreases, and therefore the frequency decreases.

The wave propagation velocities are very low. Rubber has a wave velocity of 50m/s, which has a very low wave propagation velocity, therefore this model cannot be used to model real materials. The stiffnesses of the model are too low in comparison to a real, existing material, but due to simulation instabilities the model could not be tested with much higher stiffnesses.

5 Conclusion and recommendations

5.1 Conclusion

The module works are required. The module makes it possible to simulate models based on Hamiltonian mechanics. Each model can be made with the main application. When the model is made it can be saved for reusing it.
The connection between 20-sim and the written module works without errors. 20-sim can use the module as an ordinary part in simulations. The model can draw during the simulation to see how it changes during simulation. The models which were tested do not match real world materials because real materials are too stiff which causes instability. To test real materials another integration method must be used or smaller steps have to be taken. Basic models are tested and the results gave a good first impression. The models need to test further to see if the model is also able to test real materials. The conclusion therefore is that the assignment is done successfully.

5.2 Recommendations

Integration method

The module uses a simple integration method. This method has as advantage that is very quick but as disadvantage that the system can quickly become unstable. To solve this problem a different integration method must be used. Other integration methods were not implemented due complexity.
It is useful to make use of more kinds of integration methods to get better results. It must be tested which method has the best results compare to processor time. A better integration method can result in more accurate results and less instabilities.

Model testing

The models which are tested are showing useful results, but some models also have results which cannot be explained easily. The models need to be tested further to see if model is correct, so the dynamic effects can be explained. The used models are a simple way to model a piece of material. This can have as results the dynamics of the model are not correctly modeled. Some further testing must be done with different kinds of models.
The models also need to be changed so they match a real material and not only a non-existing material. To make this possible the integration method needs to be changed first or the integration must be doen with very small steps.

Models which can be used

The current module only supports Hamiltonian models which are made of stiffnesses and point masses. With this kind of model only several kinds of spatial structures can be modeled. Several more options need to be added. Options like surface tension, which is needed for blowing balloons.
With point masses and stiffnesses a lot of models can be made but sometimes this is not sufficient enough. To extend the program with this functionality the user interface which is currently used is not sufficient enough. It is very hard to create a model with point masses, stiffnesses en surface tension without losing track on what you are doing. Therefore a whole new user interface needs must be designed.

The user interface of the main application

The user interface is a non-user-friendly interface. The user can change each node and stiffness manually, but when there are too many nodes it will be quite difficult to keep track of the model. If a lot nodes are used the user must work very neat to make the model without mistakes. If a mistake occurs, it is very hard to correct this, because it is hard to see where the mistake is.
An alternative user interface is to use pre-made models which can be used to create a big model. For example the user selects a girder and a pre-made girder will shown. Only some variables of this girder must be changed. With this method complex models can easily made with pre-made models.
The user interface currently used is very useful to test new models and materials properties, because each value can be altered but it cannot work with pre-made models.

Data exchange between DLL and main application

The data between the DLL and the main application is exchanged with shared memory and a mutex to keep track of which process may write to the shared memory. This method is very easy to implement but has as disadvantage that the shared memory easily can be altered by a 3rd party software program, so simulation errors may occur.
The data exchange can be much saver. Alternative ways are described in section 3.1.2. These methods are harder too implement but can assure the data is transmitted without another program to corrupt it.

Model storage

The storage works in an easy way. The whole shared memory is copied in a file. This method assures all the data which is needed is saved, but saves too much. The shared memory can hold up to 1000 nodes. If only 10 nodes are used, 990 other nodes are saved without reason. This has as result that the files are quite large. Each model, no matter how complex they are, has the same size. And most models can be a lot smaller without losing data.
The data stored must only contain useful information. Only the nodes which are used must be stored, not all nodes.

Integration in 20-sim

If this extension is used often in 20-sim, it can be chosen to integrate this module in 20-sim itself, so no external program is required. The only disadvantage about this, is that a part of the 20-sim program needs to be rewritten which requires a lot of work. The big advantage is that the communication with 20-sim and the DLL can be made more efficient which has as result that the simulations can be done quicker. On the other hand a lot of testing has to be done to create a flawless integration between those two parts.

Appendix A: Glossary

Several abbreviations and technical jargon is used in this report. The description of these is mentioned in this section.

DLL	Dynamic Link Library. A library y with functions which can be used by several applications.
DDE	Dynamic Data Exchange.
IPC	Interprocess Communications. Communications between different processes
RPC	Remote Procedure Calls.

Appendix B: Example static DLL

The following example is a static DLL written in Visual C++, which takes the cosinus of each input. This DLL can be used in the example on page 9.

```
#include <windows.h>
#include <math.h>
#include <fstream.h>
#define DllExport __declspec( dllexport )
ofstream outputStream;

extern "C"
{
DllExport int myFunction(double *inarr, int inputs, double *outarr, int outputs,
int major)
{
if( inputs != outputs )
return 1;
for( int i = 0; i < inputs; i++ )
  {
    outarr[i] = cos( inarr[i] );
    outputStream << inarr[i] << " " << outarr[i] << " ";
  }
  outputStream << endl;
  return 0;
}

DllExport int Initialize()
{
outputStream.open("c:\\data.log");
  return 1; // Indicate that the dll was initialized successfully.
}

DllExport int Terminate()
{
outputStream.close();
return 1; // Indicate that the dll was terminated successfully.
}
```

Appendix C: Dynamic DLL struct

The struct shown below is the struct which is used to send information to the DLL. The struct is used for initialization and the simulation itself.

```
struct SimulatorSFunctionStruct
{
double versionNumber;          // Version number of 20-sim
int nrInputs;                  // Number of inputs which are required for the DLL
int nrOutputs;                 // Number of outputs given to 20-sim
int nrIndepStates;             // Number of independent states
int nrDepStates;               // Number of dependent states
int nrAlgLoops;                // Number of algebraic loops
double simulationStartTime;    // Start time of the simulation
double simulationFinishTime;   // Finish time of the simulation
double simulationCurrentTime;  // Current time, actually the start time at the
                                  moment  of initialization
bool major;                    // Major step?
bool initialOutputCalculation; // Can the DLL give an initial output
};
```

Appendix D: SFunction cases

Because the SFunction has several cases for each different situation, each situation has its own case as shown below.

```
case: no dependent states, no algebraic loop variables
int sFunctionName(double *inputArray,
double *stateArray,
double *outputArray,
double *rateArray,
SimulatorSFunctionStruct *simStruct);

case: dependent states, no algebraic loop variables
int sFunctionName(double *inputArray,
double *stateArray,
double *dependentRateArray,
double *outputArray,
double *rateArray,
double *dependentStateArray,
SimulatorSFunctionStruct *simStruct);

case: no dependent states, algebraic loop variables
int sFunctionName(double *inputArray,
double *stateArray,
double *algLoopInArray,
double *outputArray,
double *rateArray,
double *algLoopOutrray,
SimulatorSFunctionStruct *simStruct);

case: dependent states, algebraic loop variables
int sFunctionName(double *inputArray,
double *stateArray,
double *dependentRateArray,
double *algLoopInArray,
double *outputArray,
double *rateArray,
double *dependentStateArray,
double *algLoopOutrray,
SimulatorSFunctionStruct *simStruct);
```

Appendix E: Example dynamic DLL

Below is the source code for a dynamic DLL, written in Visual C++

```
#include <windows.h>
#include "SimulatorSFunctionStruct.h"

/*********************************************************************************
 * in this source file we are gonna describe a linear system which is defined by
 * the following transfer function description:

        34
Y = ---------------- * U
     s^2 + 6s + 34

or A, B, C, D system:

A = [ 0, -3.4;
  10, -6];
B = [ -3.4;
  0];
C = [0, -1];
D = 0
which has two poles on (-3 + 5i) and (-3 -5i)

  steady state = 1

  *********************************************************************************/

#define DllExport __declspec( dllexport )

extern "C"
{

// called at begin of the simulation run
DllExport int Initialize()
{
// you can perform your own initialization here.

// success
return TRUE;
}

// called at end of the simulation run
DllExport int Terminate()
{
// do some cleaning here

// success
return TRUE;
}

DllExport int SFunctionInit (SimulatorSFunctionStruct *s)
{
// tell our caller what kind of dll we are
s->nrIndepStates = 2;
s->nrDepStates = 0;
s->nrAlgLoops = 0;

// dubious information, since 20-sim itself does not check and need this info
s->nrInputs = 1;
s->nrOutputs = 1;

// return 1, which means TRUE
return 1;
}
```

```
DllExport int SFunctionGetInitialStates (double *x0, double *xd0, double *xa0,
SimulatorSFunctionStruct *s)
{
// fill in the x0 array here. Since we specified no Dependent states, and No
algebraic loop variables
// the xd0 and xa0 may not be used.

// initial value is zero.
x0[0] = 0;
x0[1] = 0;

// return 1, which means TRUE
return 1;
}

DllExport int SFunctionCalculate(double *u, double *x, double *y, double *dx,
SimulatorSFunctionStruct *s)
{
// we could check the SimulatorSFunctionStruct here if we are in an initialization
state and/or we are
// in a major integration step.
#if 0
if( s->initialialOutputCalculation )
; // do something

// possibly do some explicit action when we are in a major step.
if( s->major == TRUE )
; // do something
#endif
dx[0] = -3.4 * x[1] -3.4 * u[0];
dx[1] = 10 * x[0] -6 * x[1];


y[0] = -x[1];

// return 1, which means TRUE
return 1;
}

}// extern "C"

BOOL APIENTRY DllMain( HANDLE hModule,
  DWORD ul_reason_for_call,
  LPVOID lpReserved
  )
{
  return TRUE;
}
```

This example can be run from 20-sim with the following code:

```
parameters
  string dllName = 'demoDynamicDll.dll';
  string functionName = 'SFunctionCalculate';
equations
  output = dlldynamic (dllName, functionName, input);
```

Appendix F: Class diagram

This class diagram contains all the attributes and method for each object. All the graphical functions in the main application are left behind due to the complexity of this part.

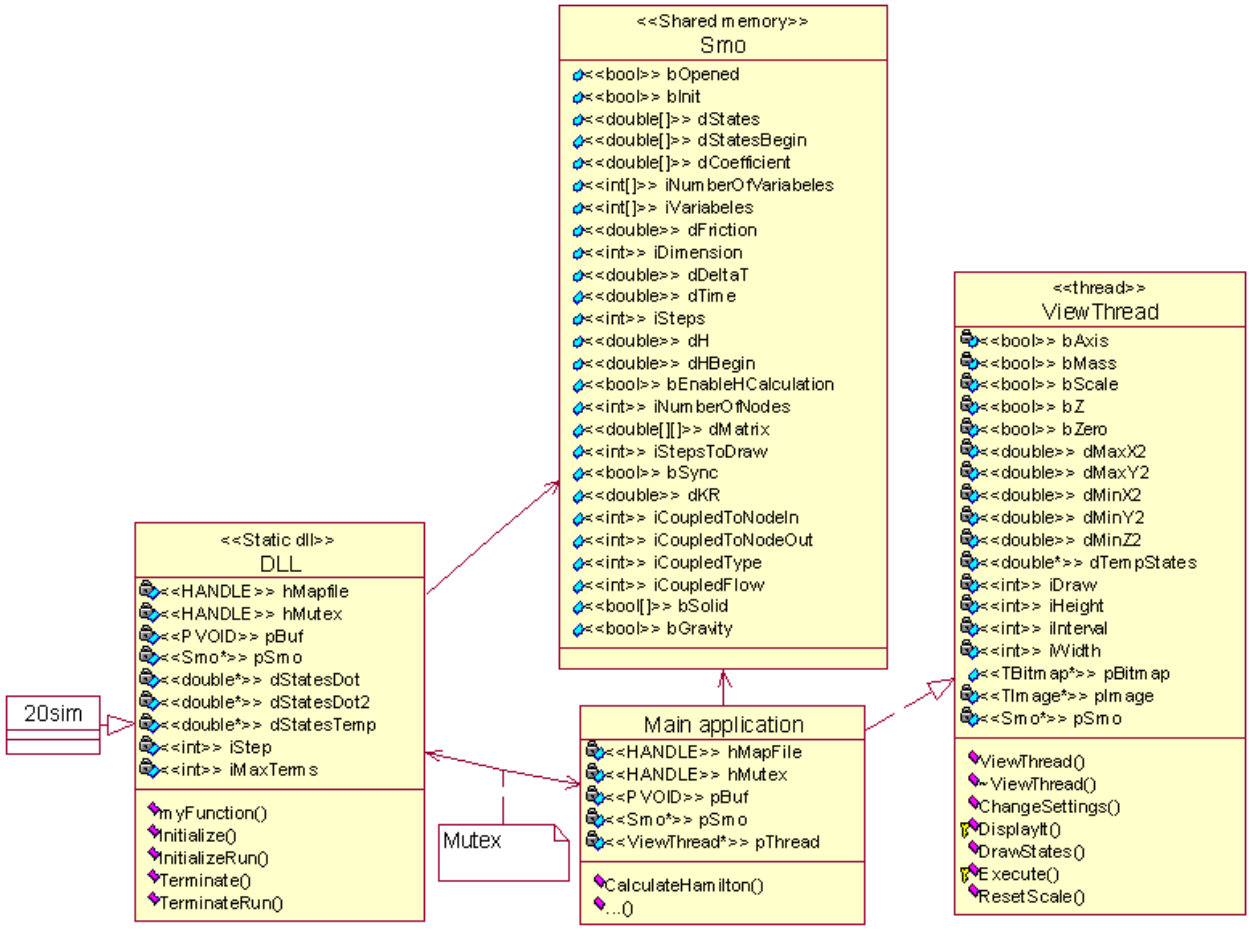

Figure 65: Class diagram

Appendix G: Shared memory struct

Below the shared memory struct ‘Smo’ is described. This struct is used to exchange information between the DLL and the application.

```
#ifndef SmoStruct
#define SmoStruct
struct Smo {
    bool bOpened;                 // Is not used anymore (replaced by mutex)
    bool bInit;                   // true if smo is ready for use; hamilton is ok
    double dStates[6000];         // 2*dimension*(nodes+1)  +1 is for the R state
    double dStatesBegin[6000];    // 2*dimension*(nodes+1)  +1 is for the R state
for the begin
    double dCoefficient[6000];    // Number of coeffiencts for the Hamilton
    int iNumberOfVariabeles[6000]; // number of varabeles per term
    int iVariabeles[24000];        // which states;
    double dFriction;             // Friction per integration step
    int iDimension;               // Number of dimension (1,2,3)
    double dDeltaT;               // stepsize
    double dTime;                 // runtime
    int iSteps;                   // number of steps taken
    double dH;                    // current H
    double dHBegin;               // H when it starts
    bool bEnableHCalculation;     // Calculate H each step?
    int iNumberOfNodes;           // Number of nodes (ex. R)
    double dMatrix[1000][1000];   // start matrix
    int iStepsToDraw;             // number of steps
    bool bSync;                   // true=draw, false=calculate
    double dKR;                   // Spring constant for R term
    int iCoupledToNodeIn;         // which node is used for connection IN   R=0
    int iCoupledToNodeOut;        // which node is used for connection Out  R=0
    int iCoupledType;             // 0: Add, 1 desired
    int iCoupledFlow;             // 1=F in, v out; 0=v in, F out
    bool bSolid[1000];            // solid node?
    bool bGravity;                // Gravity in the Hamilton

};
#endif
```

Appendix H: Calculating total energy

Calculating the total energy of a Hamiltonian can be done by

```
while (iWhichCoeff<iMaxTerms){
  dTemp=pSmo->dCoefficient[iWhichCoeff];
  for (int iLoop=0;iLoop<iNumberOfVariabeles[iWhichCoeff];iLoop++){
    dTemp=dTemp*dStates[iVariabeles[iWhichVar+iLoop]];
  }
  dH+=dTemp;
  iWhichVar+= iNumberOfVariabeles[iWhichCoeff];
  iWhichCoeff++;
}
```

In this example iWhichCoeff is an integer with references to the place in the dCoefficient array. iWhichVar is a reference to the place in the iVariables array. First dTemp will be filled with the coefficient and then it will be multiplied by each state. After each term this dTemp is added to the total energy (dH).

Appendix I: Results for chained mass measurement

Time(s)	X	Y	dx/dt	dy/dt	Ekinetic	Epotential	H (total energy)
0,4	-4,3	1,9	-27,2	2,5	3907	643	4550
0,5	-6,9	2,2	-24,4	5,1	3439	784	4223
0,6	-9,2	2,9	-21,5	7,8	2967	826	3793
0,7	-11,2	3,8	-18,7	10,3	2488	976	3464
0,8	-13,0	5,0	-15,7	12,9	2167	1100	3267
0,9	-14,4	6,4	-12,5	15,1	1980	1179	3159
1,0	-15,5	8,0	-9,4	16,2	1818	1244	3062
1,1	-16,3	9,6	-7,1	16,2	1636	1318	2954
1,2	-17,0	11,2	-5,8	15,5	1424	1437	2861
1,3	-17,5	12,7	-5,1	14,5	1210	1586	2796
1,4	-18,0	14,2	-4,4	13,4	1014	1735	2749
1,5	-18,4	15,4	-3,5	12,1	825	1874	2699
1,6	-18,7	16,6	-2,5	10,8	653	1993	2646
1,7	-18,9	17,6	-1,8	9,5	501	2096	2597
1,8	-19,0	18,5	-1,3	8,3	373	2190	2563
1,9	-19,2	19,3	-0,8	7,1	268	2273	2541
2,0	-19,2	20,0	-0,4	5,9	183	2343	2526
2,1	-19,2	20,5	0,0	4,5	116	2398	2514
2,2	-19,2	20,9	0,2	3,5	66	2441	2507
2,3	-19,2	21,2	0,4	2,4	31	2473	2504
2,4	-19,1	21,4	0,5	1,4	11	2492	2503
2,5	-19,1	21,5	0,5	0,3	2	2501	2503
2,6	-19,0	21,5	0,5	-0,7	4	2498	2502
2,7	-19,0	21,4	0,5	-1,6	16	2485	2501
2,8	-18,9	21,1	0,4	-2,6	37	2462	2499
2,9	-18,9	20,8	0,4	-3,5	65	2429	2494
3,0	-18,9	20,4	0,3	-4,4	102	2387	2489
3,1	-18,8	19,9	0,2	-5,3	147	2335	2482
3,2	-18,8	19,4	0,2	-6,3	200	2274	2474
3,3	-18,8	18,7	0,3	-7,2	264	2205	2469
3,4	-18,8	17,9	0,4	-8,1	330	2127	2457
3,5	-18,7	17,0	0,6	-8,9	404	2041	2445
3,6	-18,6	16,1	0,9	-9,7	481	1947	2428
3,7	-18,5	15,1	1,3	-10,3	557	1847	2404
3,8	-18,4	14,0	1,8	-10,9	629	1743	2372
3,9	-18,1	12,9	2,6	-11,1	686	1645	2331
4,0	-17,8	11,8	3,8	-11,0	713	1571	2284
4,1	-17,3	10,7	5,7	-10,3	732	1501	2233
4,2	-16,7	9,7	7,5	-9,3	779	1387	2166
4,3	-15,8	8,8	9,0	-8,5	833	1254	2087
4,4	-14,9	8,0	10,0	-8,1	883	1128	2011
4,5	-13,8	7,2	10,6	-8,1	927	1022	1949
4,6	-12,7	6,3	11,0	-8,2	967	931	1898
4,7	-11,6	5,5	11,2	-8,2	998	851	1849
4,8	-10,5	4,7	11,5	-7,8	1010	781	1791
4,9	-9,3	3,9	11,7	-7,1	1000	715	1715
5,0	-8,1	3,3	11,9	-6,2	976	650	1626
5,1	-6,9	2,7	12,1	-5,1	942	592	1534
5,2	-5,6	2,2	12,3	-4,1	901	546	1447
5,3	-4,4	1,9	12,3	-2,9	858	510	1368
5,5	-3,2	1,6	12,3	-1,8	818	483	1301
5,6	-1,9	1,5	12,2	-0,7	779	464	1243
5,7	-0,7	1,4	11,9	0,2	741	453	1194
5,8	0,5	1,5	11,5	1,0	698	449	1147
5,9	1,6	1,7	11,0	1,6	649	452	1101
6,0	2,7	1,8	10,5	1,9	596	464	1060
6,1	3,7	2,1	9,9	2,1	540	481	1021
6,2	4,7	2,3	9,3	2,3	482	504	986
6,3	5,6	2,5	8,6	2,4	424	529	953
6,4	6,4	2,8	7,9	2,5	367	556	923
6,5	7,2	3,0	7,2	2,6	311	583	894
6,6	7,9	3,3	6,4	2,7	259	611	870
6,7	8,5	3,6	5,7	2,7	211	638	849
6,8	9,0	3,8	5,0	2,6	169	664	833
6,9	9,5	4,1	4,3	2,5	131	689	820
7,0	9,9	4,3	3,6	2,3	98	712	810
7,1	10,3	4,5	3,1	2,0	71	733	804
7,2	10,5	4,7	2,5	1,7	48	751	799
7,3	10,8	4,9	2,0	1,3	29	766	795
7,4	10,9	5,0	1,5	0,9	16	777	793
7,5	11,1	5,1	1,0	0,5	7	786	793
7,6	11,1	5,1	0,5	0,1	1	790	791

Time(s)	X	Y	dx/dt	dy/dt	Ekinetic	Epotential	H (total energy)
7,7	11,1	5,1	0,0	-0,2	0	791	791
7,8	11,1	5,1	-0,5	-0,5	2	788	790
7,9	11,0	5,0	-1,0	-0,8	8	783	791
8,0	10,9	4,9	-1,5	-1,0	16	774	790
8,1	10,8	4,8	-1,9	-1,2	26	763	789
8,2	10,5	4,6	-2,4	-1,4	38	749	787
8,3	10,3	4,5	-2,8	-1,6	52	734	786
8,4	10,0	4,3	-3,1	-1,7	66	716	782
8,5	9,6	4,1	-3,5	-1,9	81	697	778
8,6	9,3	4,0	-3,8	-2,0	96	677	773
8,7	8,9	3,8	-4,1	-2,1	110	657	767
8,8	8,4	3,5	-4,4	-2,1	124	635	759
8,9	8,0	3,3	-4,7	-2,1	136	614	750
9,0	7,5	3,1	-4,9	-2,0	148	593	741
9,1	7,0	2,9	-5,1	-2,0	157	572	729
9,2	6,5	2,7	-5,3	-1,8	165	552	717
9,3	5,9	2,6	-5,5	-1,7	172	534	706
9,4	5,4	2,4	-5,6	-1,5	177	516	693
9,5	4,8	2,2	-5,7	-1,4	179	501	680
9,6	4,2	2,1	-5,7	-1,2	180	486	666
9,7	3,6	2,0	-5,7	-1,0	180	474	654
9,8	3,1	1,9	-5,7	-0,9	177	464	641
9,9	2,5	1,8	-5,7	-0,7	173	456	629
10,0	1,9	1,8	-5,6	-0,5	167	448	615
10,1	1,4	1,7	-5,5	0,4	160	443	603
10,2	0,8	1,7	-5,4	0,2	152	440	592
10,3	0,3	1,7	-5,2	-0,1	142	438	580
10,4	-0,2	1,7	-5,0	0,1	132	437	569
10,5	-0,7	1,7	-4,8	0,2	122	438	560
10,6	-1,2	1,7	-4,6	0,3	111	440	551
10,7	-1,7	1,7	-4,3	0,4	100	444	544
10,8	-2,1	1,8	-4,1	0,4	88	447	535
10,9	-2,5	1,8	-3,8	0,5	77	452	529
11,0	-2,8	1,9	-3,5	0,5	67	457	524
11,1	-3,2	1,9	-3,2	0,5	57	462	519
11,2	-3,5	2,0	-2,9	0,5	47	467	514
11,3	-3,8	2,0	-2,6	0,5	38	472	510
11,4	-4,0	2,1	-2,3	0,5	31	477	508
11,5	-4,3	2,1	-2,0	0,4	23	482	505
11,6	-4,4	2,1	-1,7	0,4	17	486	503
11,7	-4,6	2,2	-1,5	0,3	12	490	502
11,8	-4,7	2,2	-1,2	0,3	7	493	500
11,9	-4,8	2,3	-0,9	0,2	4	496	500
12,0	-4,9	2,3	-0,6	0,1	2	498	500
12,1	-5,0	2,3	-0,3	0,1	0	499	499
12,2	-5,0	2,3	-0,1	0,0	0	499	499
12,3	-5,0	2,3	0,2	-0,1	0	499	499
12,4	-5,0	2,3	0,4	-0,1	0	499	499
12,5	-4,9	2,3	0,6	-2,0	2	497	499
12,6	-4,8	2,3	0,9	-0,2	4	495	499
12,7	-4,7	2,2	1,1	-0,3	6	493	499
12,8	-4,6	2,2	1,3	-0,3	8	490	498
12,9	-4,5	2,2	1,4	-0,3	11	487	498
13,0	-4,3	2,1	1,6	-0,4	14	483	497
13,1	-4,1	2,1	1,8	-0,4	17	479	496
13,2	-4,0	2,1	1,9	-0,4	19	476	495
13,3	-3,8	2,0	2,0	-0,4	22	472	494
13,4	-3,6	2,0	2,2	-0,4	25	468	493
13,5	-3,3	2,0	2,3	-0,4	27	463	490
13,6	-3,1	1,9	2,3	-0,4	29	460	489
13,7	-2,9	1,9	2,4	-0,3	30	456	486
13,8	-2,6	1,8	2,5	-0,3	32	453	485
13,9	-2,4	1,8	2,5	-0,3	33	449	482
14,0	-2,1	1,8	2,5	0,3	34	447	481
14,1	-1,9	1,8	2,5	-0,2	34	444	478
14,2	-1,6	1,7	2,5	-0,2	34	441	475
14,3	-1,3	1,7	2,5	-0,2	33	439	472
14,4	-1,1	1,7	2,5	-0,1	33	438	471
14,5	-0,8	1,7	2,4	-0,1	31	436	467
14,6	-0,6	1,7	2,4	-0,1	30	435	465
14,7	-0,4	1,7	2,3	0,0	29	435	464
14,8	-0,1	1,7	2,3	0,0	27	434	461
14,9	0,1	1,7	2,2	0,0	25	434	459

Appendix J: Results for mass on a beam

Time(s)	X	Y	dx/dt	dy/dt	Ekinetic	Epotential	H (total energy)
0,4	-2,4	1,8	-21,1	1,5	5719	1282	7001
0,5	-4,3	2,1	-18,2	3,5	4523	1584	6107
0,6	-6,0	2,5	-15,1	5,7	3418	1727	5145
0,7	-7,4	3,2	-12,1	7,3	2617	1848	4465
0,8	-8,5	4,0	-9,5	8,3	2066	1967	4033
0,9	-9,3	4,8	-7,4	8,5	1634	2106	3740
1,0	-10,0	5,7	-5,8	8,0	1252	2288	3540
1,1	-10,5	6,4	-4,7	7,0	909	2496	3405
1,2	-10,9	7,1	-3,9	5,8	617	2697	3314
1,3	-11,3	7,6	-3,2	4,4	383	2867	3250
1,4	-11,6	8,0	-2,5	3,1	209	2997	3206
1,5	-11,8	8,2	-1,9	1,8	94	3083	3177
1,6	-12,0	8,3	-1,4	0,6	31	3128	3159
1,7	-12,1	8,4	-0,8	0,4	13	3135	3148
1,8	-12,1	8,2	-0,2	-1,5	30	3110	3140
1,9	-12,1	8,1	0,0	-2,4	74	3059	3133
2,0	-12,1	7,8	1,1	-3,1	136	2986	3122
2,1	-11,9	7,4	1,8	-3,6	209	2898	3107
2,2	-11,7	7,0	2,6	-4,0	287	2796	3083
2,3	-11,4	6,6	3,3	-4,2	366	2684	3050
2,4	-11,0	6,2	4,1	-4,3	443	2561	3004
2,5	-10,6	5,8	4,7	-4,2	515	2430	2945
2,6	-10,1	5,3	5,3	-4,1	579	2290	2869
2,7	-9,5	4,9	5,8	-4,0	633	2147	2780
2,8	-8,9	4,5	6,1	-3,8	675	2004	2679
2,9	-8,3	4,2	6,4	-3,7	705	1866	2571
3,0	-7,6	3,8	6,6	-3,5	722	1735	2457
3,1	-6,9	3,5	6,7	-3,4	727	1613	2340
3,2	-6,3	3,1	6,8	-3,1	720	1502	2222
3,3	-5,6	2,8	6,8	-2,8	703	1402	2105
3,4	-4,9	2,6	6,8	-2,4	678	1313	1991
3,5	-4,2	2,3	6,8	-2,0	645	1234	1879
3,6	-3,5	2,2	6,7	-1,6	607	1165	1772
3,7	-2,8	2,0	6,5	-1,2	565	1106	1671
3,8	-2,2	1,9	6,3	-0,9	519	1057	1576
3,9	-1,6	1,8	6,0	-0,6	472	1018	1490
4,0	-1,0	1,8	5,7	-0,5	423	988	1411
4,1	-0,4	1,8	5,4	-0,3	374	966	1340
4,2	0,1	1,7	5,0	-0,2	326	951	1277
4,3	0,6	1,7	4,6	-0,1	280	942	1222
4,4	1,0	1,7	4,3	0,0	237	937	1174
4,5	1,4	1,7	3,9	0,1	197	936	1133
4,6	1,8	1,7	3,5	0,2	161	938	1099
4,7	2,2	1,7	3,1	0,3	129	942	1071
4,8	2,5	1,8	2,8	0,3	101	947	1048
4,9	2,7	1,8	2,4	0,3	77	953	1030
5,0	2,9	1,8	2,1	0,3	57	960	1017
5,1	3,1	1,9	1,7	0,2	40	966	1006
5,2	3,3	1,9	1,4	0,2	27	972	999
5,3	3,4	1,9	1,1	0,1	17	977	994
5,5	3,5	1,9	0,8	0,1	9	981	990
5,6	3,6	1,9	0,6	0,0	4	984	988
5,7	3,6	1,9	0,3	0,0	1	986	987
5,8	3,7	1,9	0,1	0,0	0	987	987
5,8	3,7	1,9	-0,1	0,0	0	987	987
6,0	3,6	1,9	-0,5	0,0	2	984	986
6,2	3,5	1,9	-0,8	-0,1	7	978	985
6,4	3,3	1,9	-1,0	-0,2	12	970	982
6,6	3,1	1,8	-1,2	-0,2	17	961	978
6,8	2,8	1,8	-1,3	-0,1	21	952	973
7,0	2,6	1,8	-1,3	-0,1	22	944	966
7,2	2,3	1,8	-1,3	-0,1	23	936	959
7,4	2,0	1,7	-1,3	-0,1	21	930	951
7,7	1,7	1,7	-1,2	-0,1	18	923	941
8,0	1,3	1,7	-1,0	0,0	13	920	933
8,3	1,1	1,7	-0,8	0,0	8	919	927
8,6	0,8	1,7	-0,6	0,0	5	919	924
8,9	0,7	1,7	-0,4	0,0	2	919	921
9,2	0,6	1,7	-0,3	0,0	0	920	920
9,5	0,5	1,7	-0,1	0,0	0	920	920
9,8	0,5	1,7	0,0	0,0	0	920	920
10,1	0,5	1,7	0,1	0,0	0	920	920

Time(s)	X	Y	dx/dt	dy/dt	Ekinetic	Epotential	H (total energy)
10,5	0,6	1,7	0,2	0,0	0	919	919
10,9	0,7	1,7	0,2	0,0	0	919	919
11,3	0,8	1,7	0,2	0,0	0	918	918
11,7	0,9	1,7	0,2	0,0	0	918	918
12,0	0,9	1,7	0,2	0,0	0	918	918

References

Dynamical systems (2003), *dr.ir. P.C. Breedveld, prof.dr.ir. J. van Amerongen, Dynamische systemen: modelvorming en simulatie met bondgrafen*

Lagrange (2006), *Lagrangian – Wikipedia, the free encyclopedia*, http://en.wikipedia.org/wiki/Lagrangian

Hamilton (2006), *Hamiltonian mechanices – Wikipedia, the free encyclopedia*, http://en.wikipedia.org/wiki/Hamiltonian_mechanics

Fundamentals of multibody dynamics (2006), *Farid Amirouche, Fundamentals of multibody dynamics – Theory and applications*, 0-8176-4236-6

Applied dynamics (1998), *Francis C. Moon, Applied dynamics. With applications to multibody and mechatronic systems*, 0-471-13828-2

Classical mechanics (2004), *Tom W.B. Kibble, Frank H. Berkshire, Classical mechanics*, 1-86094-435-3

Mechanics of materials (2001), *James M. Gere, Mechanics of materials 5th SI edition*, 0-7487-6675-8

Technisch handboek werktuigbouw (1997), *G. Groenedijk, Technisch handboek werktuigbouw*, 90-5576-072-2

20-sim (2006), *Welcome to 20-sim, the software for modeling dynamic systems*, http://www.20sim.com/

20-sim help (2006), *20-sim Help*, http://www.20sim.com/webhelp4/20sim/index20sim.htm

Interprocess communications (2006), *Interprocess Communications*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ipc/base/interprocess_communications.asp

Integration methods (2006), *Methodes voor het oplossen van beginwaardeproblemen*, http://www.cs.kuleuven.ac.be/~ade/WWW/NW/NW/wagenslinger/methodes/index.html

Leapfrog (2006), *Leapfrog integrator*, http://einstein.drexel.edu/courses/CompPhys/Integrators/leapfrog/

Theory of plates and shells (1959), *Stephen P. Timoshenko, S. Woinowsky-Krieger, Theory of plates and shells, 2nd edition*, 0-07-085820-9

Fundamentals of vibrations (2001), *Leonard Meirovitch, Fundamentals of vibrations*, 0-07-118174-1