TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

A survey to the multibody program mechanica motion

Citation for published version (APA):
Termeer, M. K. (1996). *A survey to the multibody program mechanica motion*. (DCT rapporten; Vol. 1996.119). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1996

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

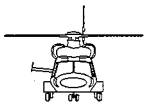
providing details and we will investigate your claim.

Download date: 04. Oct. 2023

A SURVEY TO THE MULTIBODY PROGRAM MECHANICA MOTION

M.K. TERMEER

WFW-REPORT 96.119
JULY 1996



A SURVEY TO THE MULTIBODY PROGRAM

PRO-MECHANICA MOTION

Martijn Termeer

Daf Special Products

Report TN-056
WFW report 96.119
July 1996

Coordination:
Ronald Schut
Bert Verbeek

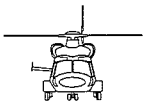
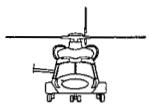


Table of contents



SUMMARY.

This report is a study to the multibody package MECHANICA MOTION. The first part contains theoretical and numerical aspects of multibody dynamics in general. The choice of a set of coordinates, kinematic description, and the dynamic formulation used in multibody packages, are discussed in chapter 2. Chapter 3 reviews numerical methods for integration of ODE's. In chapter 4 attention is paid to the routines and measures needed to solve DAE's, a typical set of equations commonly found in multibody dynamics. In chapter 5 a brief overview is given of the methods and routines used in MOTION. The last part of this report discusses small tests, run with MOTION in order to find answers to questions posed in chapter 6. Following questions are discussed: What makes a system stiff? What integration routines must be used best for what kind of systems? Can discontinuities be modeled by smooth functions, so they form less problems with numerical integration? How are flexible beams modeled in MOTION, and what is the influence of the internal damping value? Chapter 7 contains a brief discussion, conclusions and further recommendations.

1. INTRODUCTION.

DAF Special Products uses the package Pro-MECHANICA MOTION as a tool for multibody dynamics. It is mostly used for the dynamic analysis of reconnaissance vehicles and helicopter landing gear. The experiences with MOTION are generally positive, especially the user-friendly environment. However with complex models, unpredictable large computation times can be expected. The objective of this report is to get a better insight in the routines and methods used in MECHANICA MOTION and in multibody dynamics in general.

With a better insight in the backgrounds it might be possible to pay more attention to the models with respect to numerical efficiency. Especially discontinuities in the dynamic response seem to be an important problem. For example crash-simulations of landing gear have characteristically a lot of discontinuities, fast changing forces and geometries leading to an abruptly changing solution. This leads to problems with numerical integration and huge CPU times can be faced.

The first part of this report, chapter 1 to 4, will review theoretical and numerical aspects of multibody dynamics in general. However, sometimes will be referred to MECHANICA MOTION or other commercially available packages. The second part will discuss specifically the MOTION models used at DAF Special Products. Small problems will be modeled in MOTION in order to distillate different aspects of the larger models. By varying design parameters, integration methods and input tables in these small models, it will be easier to find relations between model-changes and effects in numerical efficiency. It might be possible to find some general applicable rules that can be used in the realistic models.

2. MULTIBODY DYNAMICS AND FORMULATIONS

2.1 The choice of a set of coordinates.

An important issue in multibody programs is the choice of a set of coordinates used to describe the motion. The most important sets are absolute (or Cartesian) coordinates, and relative coordinates (or state variables). The absolute coordinates define the system with respect to an inertial frame, this results in a maximal set of Cartesian coordinates for each body. The relative coordinates describe the position and velocities of a body relative to that of a connected body, this results in a minimal set of variables.

2.2 The choice of the dynamic formulation.

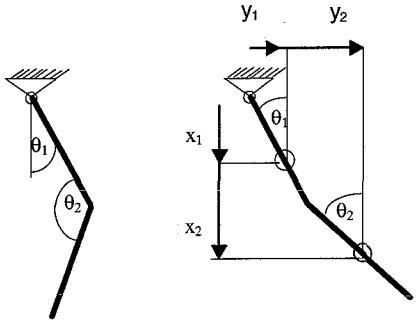
2.2.1 Augmentation- and elimination method.

Closely related to the choice of a set of coordinates is the choice of the dynamic formulation. In general there are two approaches, the augmentation method and the elimination method. In the augmentation method the equations of motion are augmented with the constraint equations, in the elimination method the constraint equations are eliminated as much as possible. In the first case a set of absolute coordinates is chosen, resulting in a maximal number of equations of motion and constraint equations, the so called descriptor form. In the second approach relative coordinates are chosen and this leads to a minimal set of equations, the so called elimination form.

Both approaches have advantages and disadvantages. The generation of equations of motion in the descriptor form is straight forward and therefore easy to automate and implement in computer software. The matrices are large, on the other hand quite sparse. Special 'sparse oriented' solvers are used. The descriptor form is used in the packages DADS and ADAMS.

In the elimination approach the equations of motion are more complex. The equations are highly non-linear, the matrices are full. On the other hand the number of equations are smaller, so only small matrices have to be inverted. This approach is found in several packages such as MADYMO and MECHANICA MOTION.

This issue will be explained with an 2-D example of a double pendulum. See Figure 2.1.



Relative Coordinates. Absolute Coordinates

Figure 2.1

The double pendulum has two degrees of freedom and can be described in either the descriptor form or the elimination form.

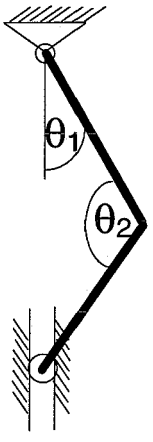
In the *descriptor* form the center of mass of each body is described with respect to a reference frame by translational x-y- coordinates and a rotational coordinate φ. This results in 6 Cartesian coordinates for the two bodies together. These coordinates are not independent, they do not satisfy the kinematic constraints automatically. Therefore the 6 equations of motion have to be augmented with 4 constraint equations, two constraints for each rotational joint. Of course this gives 6 - 4 = 2 degrees of freedom. Unknown Lagrange multipliers account for the constraint loads. Hence the resulting equations form a combined system of differential equations (the equations of motion) and algebraic equations (the constraint equations). Together this is called a set of Differential Algebraic Equations or DAE's.

In the *elimination form* there are two relative rotational coordinates φ_1 and φ_2, each describing the position of the body relative to a connected body. These independent coordinates form degrees of freedom so they do not need to be augmented with constraint equations. There are no constraints that can be violated. The results is a minimal set of Ordinary Differential Equations or ODE's, the system is kinematically unconstraint.

2.2.2 Closed-loop and Open-loop systems.

In fact relative coordinates only result in ODE's for open loop systems. For closed loop system relative description will result in DAE's. For closed loop systems the relative coordinates are not independent anymore, they do not form degrees of freedom.

This is demonstrated in the following example, a crank-slider mechanism, see Figure 2.2. This is a simple closed loop system and in fact it is a double pendulum with an extra constraint. Therefore it has only 1 degree of freedom. The two relative coordinates θ_1 and θ_2 are not independent, the system is kinematically constraint.

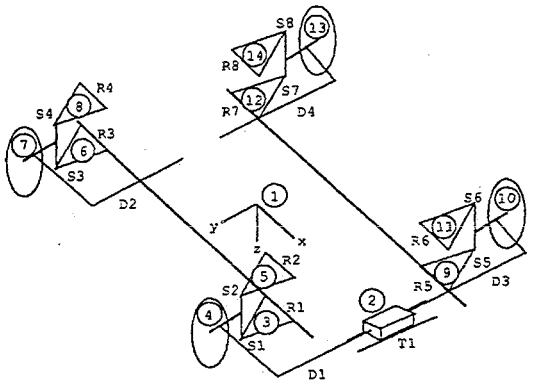


Closed-loop

In theory this system can be modeled with only one coordinate, the degree of freedom. This will lead to only one very complex equation of motion without any constraint equations. This way is not used in multibody dynamics. The generation of the equations of motion described in degrees of freedom would be far too complex for real systems.

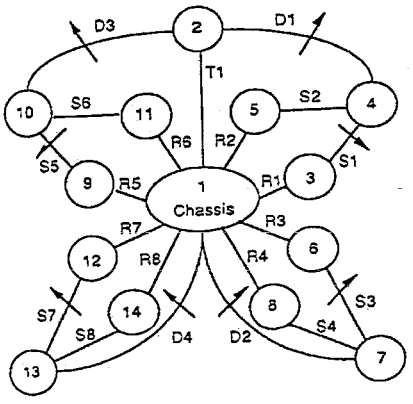
Figure 2.2

A more realistic example of a closed-loop system is the model of a reconnaissance vehicle, see Figure 2.3.



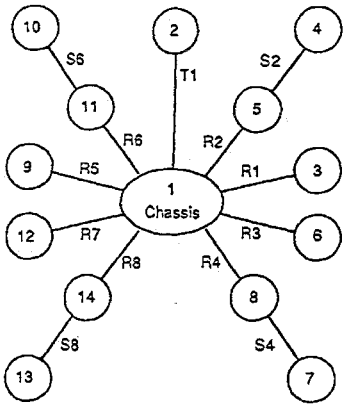
Vehicle Model

Figure 2.3



Graph Representation

Figure 2.4



Spanning Tree

Wittenburg developed the idea of opening closed chains. The system is transformed in a tree-configuration by cutting the closed chains, see Figure 2.4. Which joints in have to be cut, is decided with the aid of graph theory [2,5]. The best candidate for base-body is determined by graph theory as well. This way results in a tree-topology in which the least numerical error will accumulate. At the cut-joints it is necessary to introduce constraints and Lagrange multipliers, to keep the original configuration intact*. The result is a Spanning-tree, see Figure 2.5.

Figure 2.5

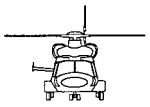
2.2.3 Equations of kinematic unconstraint systems.

So an open loop system can be described with ODE's when relative coordinates are used. In that case the system is kinematically unconstraint. The topology is a tree-structure with only open chains. The number of coordinates (n) equals the number of degrees of freedom. The equations of motion can be described as follows:

$$M \cdot \ddot{q} = g(q, \dot{q}, t),$$

with M the matrix containing the mass and inertia terms, $\ddot{q}$ the vector with translational and rotational accelerations, and g the force vector containing all internal and external forces and moments. Gravity is considered an external force, whereas force elements within the system, such as springs and dampers, are considered

* In the package MADYMO these constraints are guaranteed by stiff springs and dampers, see also paragraph 4.2.4.



internal forces. Of course, since it is an unconstraint system, there are no constraint forces. For numerical integration of ODE's see chapter 3.

2.2.4 Equations of kinematic constraint systems.

If the number of coordinates is exceeds the number of degrees of freedom, the system is said to be kinematically constraint. In paragraph 2.2.2, it was found that these systems are open-loop systems described in relative coordinates, or closed loop-systems. This kind of systems can't be solved with ordinary numerical integration routines since the equations are mixed differential and algebraic equations (DAE's). The complete set of equations is given by

Equations of motion, the differential part:

$$M\ddot{q} - \Phi_q^T \lambda = g$$

With

- *M,* the mass matrix containing masses inertia terms
- $\ddot{q}$, a vector containing the accelerations
- g, a vector containing external and internal forces
- λ, a vector containing the unknown Lagrange multipliers, they account for the reaction forces in the constraints.
- Φ_q, the Jacobian matrix, containing the derivatives of the constraint equations to the coordinates.

and *M, g,* λ can be a function of $q, \dot{q},$ and an explicit function of *t.*

Constraint equations for the positions, the algebraic part:

$$\Phi \equiv \Phi(q)$$

with Φ the constraint equations.

Together this is a semi implicit DAE of index 3. The index of a DAE is a measure of 'how algebraic' the DAE is [5]. Since most numerical methods are only convergent for index 1 DAE's, the DAE must be reduced to index 1 by differentiating the position constraints twice. This results in the set:

$$\Phi \equiv \Phi(q) = 0$$
$$\dot{\Phi} \equiv \Phi_q \dot{q} = 0$$

The constraint acceleration equations are written together with the equations of motion in matrix notation:

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma \end{bmatrix}, \text{ with}$$

$$\gamma \equiv -(\Phi_q \dot{q})_q \dot{q} - 2\Phi_{qt}\dot{q} - \Phi_{tt},$$

the righthand side of the acceleration equations. Chapter 3 will deal with solving such a set of differential algebraic equations of index 1.

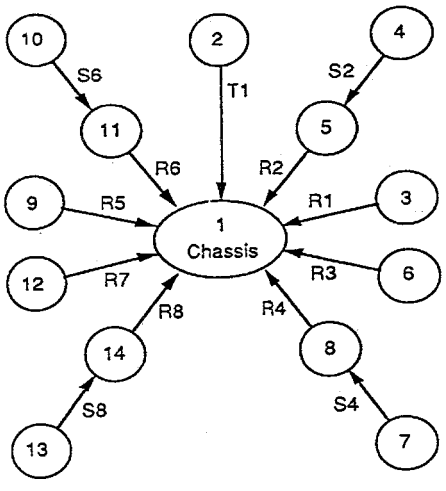
2.2.5 Kane's Method.

In paragraph 2.2.1 the advantages and disadvantages of the formulations in the descriptor form with absolute coordinates and the elimination form, with relative coordinates were discussed. The Kane's Method [7] tries to the combine the advantages of both formulations. It starts with defining the systems in global dependent coordinates. These Cartesian coordinates are transformed by means of a 'partial velocity matrix' to relative coordinates. For open-loop systems this will result in a set of ODE's, however for a closed-loop system the resulting equations will still be DAE's.

The original Kane's method is $O(n^3)$, which means that the number of numerical operations will increase with the power three of the number of bodies (n) in the model. This is due to the fact that the sizes of the matrices increase linearly with the number of degrees of freedom. Increase in model-size will simply increase the set of equations and increase the size of the matrices. The number of operations to invert a full matrix depends on the power three of the size of the matrix. In MECHANICA MOTION a modified Kane's method is used, this will be reviewed in the next paragraph.

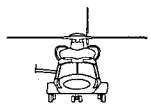
2.2.6 Recursive Formulations.

Description of a system in relative coordinates, makes it possible to use recursive formulation to generate the equations of motion. Recursive formulation [5,6] is a procedure in which elementary relationships between two arbitrary bodies, can be used all along the system. The equations of motion are not generated for the whole system. In a tree-branch the motion of one body is expressed in terms of the motion of the adjacent body and the relative motion of the connecting joint. The equations of motion are generated by starting at an end-point of a tree-branch, moving recursively back to a base body, see Figure 2.6.



Backward Path Dynamics Sequence

Figure 2.6



This recursive approach is of O-(n) because the number of operations increases only linearly with the number of bodies in the system. The sizes of the matrices stay small. When a system becomes twice as large, only two times the same recursive algorithm has to be used. This makes this description highly compact and efficient.

A recursive O(n) algorithm is used in MECHANICA MOTION and is called modified Kane's Rosenthal method. MOTION uses this algorithm if the number of coordinates is more than 15, otherwise the original $O(n^3)$ Kane's method is used. MADYMO also uses a recursive O(n) algorithm.

Recursive formulation is highly suitable for parallel processing, since all the tree-branches can be handled independently, at the same time. There is a lot of recent development in this parallel processing algorithms for multibody dynamics, especially in the field of real-time driving- and flight-simulations [5,6].

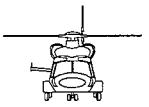
2.2.7 Symbolic equations in MECHANICA MOTION.

MECHANICA MOTION, as some other packages, formulates all equations symbolically. This means that in the equations the 'design information' is kept in symbols. An advantage of this approach is the fact that the equations can be reused in a new job with for example other forces.

A design study can be done, design parameters can be varied over a certain range, without new generation of equations. Another advantage is the availability of explicit derivatives of the equations to design-variables. These derivatives can be used in optimization analysis.

A disadvantage of the symbolic description is the memory space required for such computations. The 'design information' has to be stored in the matrices. Compared to symbolic description, in numerical description no 'design information' is found in the matrices. Then the matrices are only functions of positions and velocities (and sometimes explicit functions of time).

To simplify the symbolic equations as much as possible, MOTION uses a 'Symbolic Equation Manipulator'. Simplifications in the equations are possible in case of a force working at a center of gravity, bodies that have inertia in a direction that is fixed, plane motion, etc. After this manipulation subroutines are written, compiled and linked.



3. NUMERICAL INTEGRATION METHODS FOR SOLVING ODE'S.

As explained in the previous chapter the equations of motion, for planar and spatial systems, are either a set of ordinary differential equations (ODE's) or a set of mixed differential and algebraic equations (DAE's). In general these equations have to be solved numerically, although it might be possible to obtain a closed form solution for highly simplified systems. In this chapter the standard numerical integration routines for ODE's are briefly reviewed.

3.1 Initial value problems

The equations found in dynamics are often sets of second order ODE's. Such a set of *n* equations needs to be converted to a set of *2n* first order ODE's, because almost all numerical integration routines are developed for first order ODE's. This conversion can always be done by introducing extra state variables.

The original equations of motion are a set of *n* second order differential equations:

$$\ddot{q} = f(q, \dot{q}, t).$$

After introducing

$$y = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \quad and \quad \dot{y} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix},$$

this results in the form

$$\dot{y} = f(y, t).$$

This is a set of *2n* first order differential equations.

Then the numerical integration at time $t=t^i$ can be interpreted by the following diagram:

$$\dot{y}(t^i) \xrightarrow{\text{integration}} y(t^{i+1})$$

In other words, velocities and accelerations at $t=t^i$ yield coordinates and velocities at $t=t^{i+1}$. So these methods involve a step by step process with discrete points in time. To start this process, initial conditions have to be specified at $t=t_0$ for q_0 and $\dot{q}_0$. The stepsize *h* between the discrete points can be constant or variable.

Numerical integration routines can be roughly classified by issues like:

- Singlestep vs. multistep
- Implicit vs. explicit
- The order of the method
- Variable or constant stepsize

3.2 Euler methods.

To explain the principle of numerical integration it is convenient to start with Euler methods. These algorithms are not suitable for practical use but they illustrate the principle clearly. Euler Forward is the basis for explicit integration methods. Euler

backward is the basic implicit method. Both methods are single-step methods. This means that in each timestep only knowledge of y_i is used and no points in history like y_{i-1}.

3.2.1 Euler forward.

The Scheme can be formulated as

$$y_{i+1} = y_i + h \cdot f(y_i, t_i) \ .$$

This can be viewed graphically, see figure 3.1. Here it is shown that at $t=t_i$ the function $f(y_i, t_i)$ is evaluated and used to determine the slope. With this slope a step h in time is made. At $t=t_{i+1}$ the point y_{i+1} is found as an approximation of the real solution at that point. This method is a so called explicit method because at the time $t=t_i$ there is an explicit formulation of y_{i+1}, or in other words, in the right hand side of the equation only variables at $t=t_i$ occur.

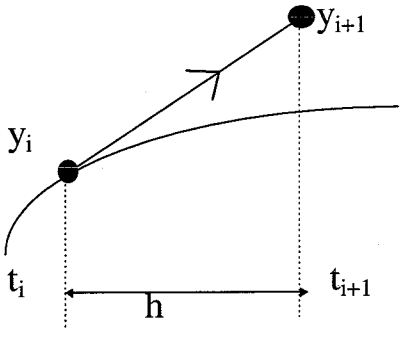


Figure 3.1

Explicit methods are conditionally stable. This means that the time-step h must be within a stability area. As will be seen in paragraph 3.4, this stability interval will only be a problem when explicit methods are used for stiff differential equations.

3.2.2 Euler backward

The formulation of the Euler backward algorithm can be written as

$$y_{i+1} = y_i + h \cdot f(y_{i+1}, t_{i+1}) \ .$$

Here y_{i+1} is a function of y_{i+1}, and t_{i+1} . This is a implicit description and needs to be solved iteratively by a Newton-Raphson proces every timestep. This makes a timestep for this method more expensive than a step with Euler forward. On the other hand, this method is unconditionally stable. The Newton-Raphson process is 'searching for' that slope at $t=t_{i+1}$ so that the tangent points through the point y_i. See figure 3.2.

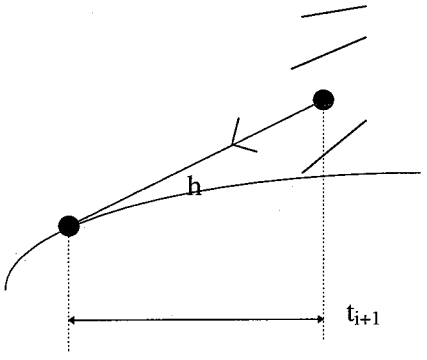
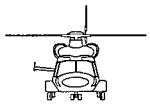


Figure 3.2

3.2.3 Runge-Kutta methods.

Runge-Kutta methods are widely used in the numerical solution of the nonlinear differential equations of mechanical systems. The order of these methods is usually higher than one. They use an average slope, calculated at several points. The number of slopes or function evaluations each time



step is the order of the method. Widely used is Runge-Kutta-4 method, which calculates 4 slopes each timestep. The slopes are averaged and then, similar to Euler forward, this average slope is used to make a step in time. The use of Runge-Kutta methods needs more function evaluations each timestep, as compared to Euler forward. This price must be paid for the higher accuracy. However, as a result of this accuracy, timesteps can be larger. The algorithm of a Runge-Kutta-4 method can be written as

$y^{i+1} = y^i + h \cdot g$ where g is the average slope,

$$g = \frac{1}{6}(f_1 + 2f_2 + 2f_3 + f_4)$$

$$f_1 = f(y^i, t^i)$$

$$f_2 = f(y^i + \frac{h}{2} f_1, t^i + \frac{h}{2})$$

$$f_3 = f(y^i + \frac{h}{2} f_2, t^i + \frac{h}{2})$$

$$f_4 = f(y^i + hf_3, t^i + h)$$

It can be seen that a Runge Kutta method is an explicit method, y^{i+1} is not a function of y^{i+1} itself. This is an advantage since an implicit method has to be solved iteratively with a Newton-Raphson algorithm. On the other hand an explicit method is conditionally stable, this means that there is a maximum time step, called the stability area, in which the numerical errors stay bounded and stability is guaranteed.

Runge-Kutta methods are single step methods, no historical information of y_{i-1} or so is used in the computations of the next timesteps. Hence this method is not as computationally efficient as some multistep methods. Most used Runge-Kutta methods do have a variable time-step selector for efficiency. This is explained in paragraph 3.5.2. MOTION uses a Runge-Kutta-45 method with variable timestep.

3.3 Multistep methods.

In contrast to the Euler and Runge-Kutta methods, multi-step methods do use information of previous computations. It can be more efficient to use this passed information. The higher the order of the method the more historical information is taken into account. In practice up to ten past values are used. Multistep methods can be both explicit and implicit. To start a multistep method a special start routine has to be used since there are no previous computations. This can either be done with an explicit Runge-Kutta method, or starting with order one and small time-steps, increasing the order until enough points are known. Multistep methods can be both explicit or implicit.

3.3.1 Adams methods.

A general formulation of Adams methods [see 1,2] is

$$y^{i+1} = y^i + h \cdot \left[b_{-1} f\left(y^{i+1}, t^{i+1}\right) + b_0 f\left(y^i, y^i\right) + \ldots\ldots + b_p f\left(y^{i-p}, t^{i-p}\right) \right]$$

It can be seen that in case of $b_{-1} = 0$ the method is explicit since in that case y^{i+1} does not occur on the right side. This class of methods are known as Adams-Bashforth algorithms.

In the case of $b_{-1} \neq 0$ the above equation becomes an implicit formulation and these methods are called Adams-Moulton Algorithms. Of course, this implicit formulation has to be solved iteratively until convergence, each integration time-step. This provides again the unconditional stability of an implicit method.

A practical often an efficient combination of Adams-Bashforth and Adams-Moulton is used. This is called a predictor-corrector-method. The explicit Bashforth part provides an estimation-step for the corrector part, an Adams moulton algorithm. The differences between these steps are used for error estimation.

3.3.2 Methods for stiff systems.

A stiff system is referred to as any initial-value problem for which the solution contains fast, rapidly varying components that damp out to become smooth. In other words it contains different time-scales along the solution, or high and low frequency components, for which the high frequency component will damp out completely. A stiff system is sometimes referred to as a system with widely spread eigenvalues. This can be a little confusing, since mathematically spoken, a single differential equation can be stiff. The difference between stiff and non-stiff systems will be reviewed in paragraph 6.2. An example of a stiff DE, for solution and direction field see figure 3.4.

$$\dot{y} = \lambda\left(e^{-t} - y\right); \quad \lambda >> 1; \quad t > 0 \ .$$

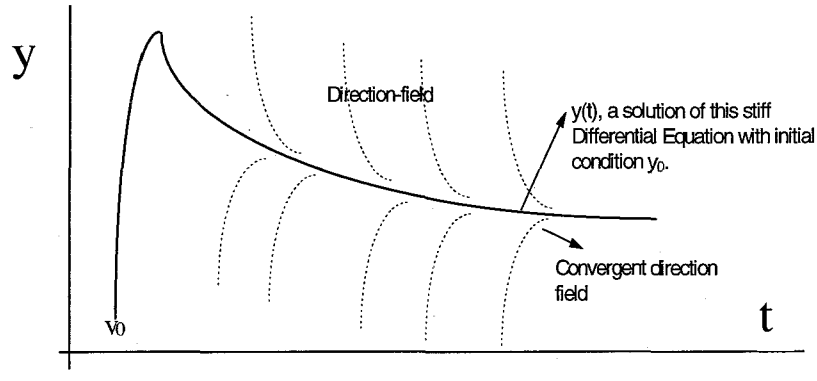
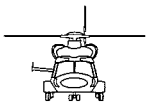


Figure 3.3

For a numerical solution, integration has to be performed over a long time interval in order to capture the slow components. On the other hand the time steps need to be small to capture the high frequency components. It is obvious that carrying out integration over a long time period with small time steps can make computation times unrealisticly large.



For stiff systems it would be convenient to have the possibility to take large timesteps when the solution is smooth. What you need then is a method that is numerically stable for large time steps. Explicit methods have a small stability interval when the differential equation is stiff. Implicit multistep methods will do well in this situation since they are unconditionally stable. A family of implicit methods called Backward Difference Formulae (BDF), firstly published by Gear, are specially designed for stiff systems.

A general scheme can be written as:

$$y^{i+1} = a_0(k)y^i + a_1(k)y^{i-1} + \ldots + a_{k-1}(k)y^{i-k+1} + h \cdot b_{-1}(k)f(y^{i+1}, t^{i+1}),$$

where $a_j(k)$ indicates the dependence of each coefficient on the order k.

The stiff character of the system is still present in the smooth part of the solution. This can be seen from the direction field near the solution in the smooth part. This direction field is sharply convergent close to the exact solution, it forms a kind of narrow wedge. BDF methods benefit from this convergent direction field, since this implicit method searches that slope for which the tangent will 'point back through the previous point'. This slope will only be found when the function is evaluated in a point close to the exact solution, in other words, the Newton-Raphson process will only find a matching y_{i+1} in a small band around the exact solution. A small distance of the exact solution the direction is completely different and the tangent will not point back through y_i.

For the same reasons an explicit method has a disadvantage from this convergent direction-field. Such a method would blindly step forward in time with a computed derivative. One step *h* further in time it will find an approximation for y_{i+1} but, since the convergent direction field, it will likely find a false derivative. The error between for example 4-th and 5-th order computations will be high, and therefore steps have to be small. This explains the small stability interval when explicit methods are used for stiff DE.

Although BDF methods are convenient for solving stiff differential equations it is of course possible to integrate a stiff DE with for example a explicit Runge-Kutta method. But in that case a lot of small time steps are needed even in the smooth part of the solution because of the already mentioned small stability interval.

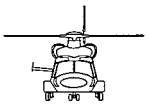
Mechanica MOTION uses a BDF method from DASSL, a package with six families of BDF methods for solving DAE's. The method has an automatic order- step-size control mechanism.

The philosophy of Mechanica MOTION is (told by Customer Support Germany) to make their integrators (both explicit and implicit) accurate in the first place, not necessarily fast.

3.4 Error estimation accuracy and efficiency.

3.4.1 Error estimation.

An important aspect in numerical integration is the error estimation. Each time step an error will be made, called the local error. All these local errors together form a total



error at the end, called the global error. The order of an integration method is a measure of the order of the local error. When a method is of order *k*, then the local error is of order *k+1*, then the global error is of order *k* again.

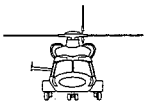
Since the exact solution is not known, how is it possible to compute an error estimation? This can be done by calculating the solution with different orders. A Runge-Kutta-4,5 for example uses the difference between a fourth order and a fifth order computation. This is efficient because the fifth order calculation uses the same points as the fourth order, only one new function evaluation is needed. The error estimation is used to determine that stepsize for which the error on the exact solution stays smaller than the user requested tolerance. Adams methods and BDF methods work in a similar way. Besides a relative error indication, good integrator-algorithms use a combination of relative and absolute error tolerances. Otherwise for example an oscillating motion around zero can form a problem.

3.4.2 Variable timestep and order.

The stepsize *h* for each timestep may be optimized by choosing the largest possible timestep for which the local error remains bounded below the user specified maximum allowable error and for which the algorithm stays numerically stable. When a function is smooth the step-size may be increased, when it is changing abruptly, it will be decreased.

With Runge-Kutta methods it is easy to make time steps variable, since only the knowledge of one timestep is used. With multistep methods this is somewhat more complex because several points in history are used in each time step and varying timesteps is only possible after interpolation between the points calculated in history.

Multistep methods usually can vary their order by taking into account more or less points in history. Increase the order k can be efficient when a function is smooth, in that case larger step-sizes are possible. When a function is changing rapidly the order can be decreased since older information is then of less use than the recent information. To make it possible to increase the order enough information must be kept in memory. A lot of advanced multistep methods vary both order and time-step.



4. NUMERICAL METHODS IN MULTIBODY DYNAMICS

This chapter deals with methods for solving DAE's. When system equations form DAE's is found in paragraph 2.2.2 and 2.2.4. For a thorough knowledge of DAE's in multibody dynamics see reference [5].

4.1 Direct integration.

A simple but crude method for obtaining the dynamic response of a system represented by the equations in paragraph 2.2.4, is direct time integration. This method simply integrates the matrix part of the equations, without taking into account the position- and velocity- constraints. Only the acceleration constraint will be taken into account. Accelerations appear in both the equations of motion and in the constraint equations. So when numerically integrated, the accelerations will always obey the constraints within the given accuracy. This means that the position and velocity constraints can be violated, especially when integration-times are longer. For this reason this method is very crude and not practically useful.

4.2 Constraint violation stabilization method

The Constraint Violation Stabilization method by Baumgarte is an extension of feedback control theory applied to the dynamics of mechanical systems. One of the goals in designing a feedback controller is to reduce the error growth and achieve a stable response [2]. Baumgarte introduced the stabilization control terms into the index one DAE. Instead of the pure acceleration constraints a linear combination of acceleration, velocities and position constraints is used with parameters α and β. In this way the constraint violations of the position, Φ and velocity, $\dot{\Phi}$ are fed back in a way that the response will approach to the exact solution in an oscillating way.

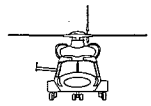
$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma - 2\alpha\dot{\Phi} - \beta^2\Phi \end{bmatrix}$$

The choice of the parameters is critical and has to be so that the solution for $\Phi = 0$ is stable. Often one chooses $\alpha = \beta$. Amplitude and frequency of the oscillation around the exact solution depend on these parameters.

4.3 Coordinate partitioning method.

It was explained in 4.2.1 that direct integration of the equations of motion will lead to constraint violations in the positions and velocities. This method makes use of the fact that the n coordinates are not independent. The n coordinates can be partitioned into m dependent coordinates u and k independent coordinates v. Also the velocity vector q can be partitioned in $\dot{u}$ and $\dot{v}$. Only the independent variables are integrated each time-step. So the integration arrays are:

$$y = \begin{bmatrix} v \\ \dot{v} \end{bmatrix}; \quad \text{and } \dot{y} = \begin{bmatrix} \dot{v} \\ \ddot{v} \end{bmatrix}$$



Integration of the array $\dot{y}$ yields the array y, each array with dimension *2n.* With these independent coordinates and velocities $(v, \dot{v})$, the dependent coordinates and velocities $(u, \dot{u})$ can be solved at the end of each time-step, out of the algebraic constraint equations:

$$\Phi(u, v) = 0$$
$$\Phi_u \dot{u} = -\Phi_v \dot{v}$$

Now that the vectors q and $\dot{q}$ are completely known, $\ddot{q}$ and λ can be solved from equation

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma \end{bmatrix}.$$

From $\ddot{q}$, $\dot{v}$ can be transferred to the next integration array $\dot{y}$. A new step can begin.

The important issue here is that the position constraints and the velocity constraints can't accumulate numerical error, since the dependent variables are not integrated at all* . They are only solved with a Newton-Raphson process out of the algebraic constraint equations (both positions and velocities) at the end of a time-step.

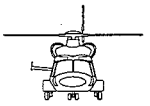
The only problem is to find the right partitioning of dependent and independent coordinates. This is possible with Gauss elimination or LU decomposition of the constraint Jacobian Φ_q. For a mechanical system with m constraints the Jacobian is a m*n matrix. After Gauss elimination the columns of the reordered Jacobian correspond to the order of elements in vector q. The first m elements of q can be used as the dependent coordinates u, the remaining k elements can be used as the independent coordinates v. A specific partitioning can hold until numerical ill conditioning of the Jacobian occurs. Then a new set of dependent and independent coordinates must be specified. This process can be done fully automatically.

The package DADS uses a combination of the Coordinate Partitioning method and Constrained Stabilization Method. The hybrid method takes advantage of both methods, the speed of Constraint stabilization and the reliability of the Coordinate Partitioning Method. Not every step the dependent variables are solved. Small numerical errors are stabilized with Baumgarte, the constraint violations are monitored. Only when violations become too big a Newton-Raphson process will solve the dependent algebraic variables.

4.4 Handle DAE's as stiff ODE's.

This method is completely different from the methods discussed above. This method considers the algebraic constraint equations as a special form of differential equations in which the time derivatives do not appear. When normally the DAE can be written as:

* Sometimes the dependent coordinates are integrated, then they are estimations for the Newton-Raphson process.



$$\dot{y} = f(y,t)$$
$$0 = g(y,t)$$

Now the DAE will be written as a stiff ODE:

$$\begin{aligned} \dot{y} &= f(y,t) \\ \varepsilon\ \dot{y} &= g(y,t) \end{aligned} \quad \text{with } \varepsilon << 1.$$

This approach is used in MADYMO. This package describes the system in relative coordinates as can be found in paragraph 2.2.1. and 2.2.5. In the case of open loop systems, this description forms spanning trees leading to a minimal set of ODE's. For closed loop systems the loops are cut. At the cut joints there must be done something to prevent the parts from tearing apart. This usually can be done by introducing Lagrange multipliers, resulting in a set of DAE's. See also paragraph 2.2.2. In MADYMO this is slightly different, the cut loops are kept together by introducing stiff springs and dampers at the cut joints. These elements keep the joints together and in this way the kinematic constraint forces are handled in fact as internal forces. The result is that the algebraic constraints disappear, therefor artificial internal forces appear. This way results in stiff ODE's.

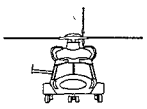
Normally such stiff ODE's would be solved with implicit BDF methods, highly optimized for stiff ODE's, but in MADYMO these stiff ODE's are solved explicitly with a Runge-Kutta method. Reason for this is the fact that crash simulations form a lot of discontinuities. Often there will be a discontinuity somewhere in the system, asking for small stepsizes. Therefore timesteps have to be small all the time and the advantage of implicit BDF methods, with the possibility of large timesteps, is unnecessary. In fact solving an implicit formulation for lots of degrees of freedom is costly. Another aspect is the fact that if an implicit method approaches a discontinuity with a larger stepsize, that discontinuity is relatively larger compared to the situation when an explicit method, with a conservative small stepsize, approaches the discontinuity.

4.5 Projection method.

Drift off effects of the constraints can be avoided by numerically solving a DAE with reduced index, combined with a projection of velocities and positions, such that the original constraints are satisfied. This projection can be done each timestep or, for efficiency, each few timesteps. This method resembles the Coordinate Partitioning Method but here there is no partitioning of the coordinates. After every time step the coordinates are projected at the original constraints. For more detailed information see [5].

4.6 Overdetermined DAE.

Another approach is developed by Fuhrer. To prevent the drift from the constraints, not only the position constraints but also the velocity and acceleration constraints are used. This forms a set of Overdetermined Differential Algebraic Equations. If this overdetermined set is discretized by a BDF method, there is no unique solution. Therefore a solution in the least square sense has to be found. This approach is used in Mechanica MOTION when implicit integration is chosen. See for further detail [5].



5. OVERVIEW OF MOTION.

A brief technical overview of the methods and algorithms used in MECHANICA MOTION:

- Coordinate formulation and equations:
  - * Kane's method: transformation from Cartesian coordinates to relative coordinates, an order-(n^3) method.
  - * Tree-branch coordinates makes recursive formulation possible, an order-(n) method. This formulation is used if the number of coordinates exceeds 15.
- Symbolic formulation of equations
  - * Sensitivity analysis or design study possible with same equations
  - * Optimization possible without finit difference methods (explicit derivatives available)
  - * Re-use of equations in new jobs
  - * Large memory requests
- Integrators:
  - * Plot-interval is maximum stepsize for integrators
  - * Mechanica's philosophy is to make their integrators primarily accurate and conservative, not necessarily fast.
- Explicit: Runge-Kutta-45
  - * variable stepsize
  - * for non-stiff systems
  - * error estimation by difference in 4-th /5-th order
- Implicit: BDF method for solving DAE's (from DASSL)
  - * unconditionally stability, developed for stiff DE's
  - * history information is used, variable order and stepsize
  - * need for solving implicit set of equations with Newton Raphson
- Preventing constraint violation:
- Baumgarte constraint stabilization for explicit method
- Overdetermined DAE for implicit method
- Flexible beams
  - * Handled as six internal loads (in the right hand side of the equations).
  - * Internal damping is a factor on the six stiffnesses of the beam, proportional damping.

6. TESTS WITH MECHANICA MOTION.

6.1 Introduction.

At DAF Special Products the multibody package Mechanica MOTION is used for the dynamic analysis of complex systems. These systems are mostly landing gear and reconnaissance vehicles. Characteristic for these multibody models, especially the crash-models for landing gear, is the appearance of discontinuities. Some examples are abruptly opening valves, shearing safety pins, impact of tires and wheels and Coulomb friction.

The following questions were explicitly asked by the MOTION users at DAF Special Products, before the start of this project:

- What's the difference between explicit and implicit integration?
- What integration methods are most suitable for what kind of models?

After discussion the following questions were found to be interesting too:

- What's the influence of discontinuities on the numerical efficiency?
- What are the consequences of stiff elements like flexible beams?
- What are the effects of introducing damping?
- What is the effect of tolerances and accuracies?
- What is the influence of Coulomb friction in models on CPU times?

Answers to these questions will lead to more insight in MOTION, and in that way, to better defined models. Maybe CPU times can be decreased, and if not, the need for a certain amount of CPU time is at least better argumented.

To find answers to the questions above, simple tests have been done with MOTION. These tests are described in the following paragraph, results can be found in the appendices. Because of the limited time for this project a selection is made, only the first five mentioned questions will be investigated in tests.

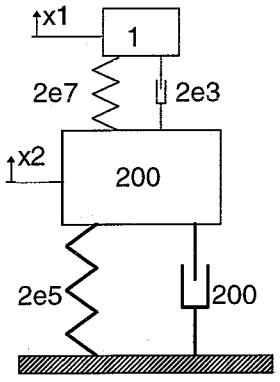
A reason for skipping the question about accuracies and tolerances is the badly description of the meaning of these parameters. The MOTION Reference Guide is not very clear with respect to this topic. Information conflicting with this manual, is recieved from customer support. In other words experimenting with parameters that have a badly described function is not very useful. The question about Coulomb friction was considered less important for DAF Special Products.

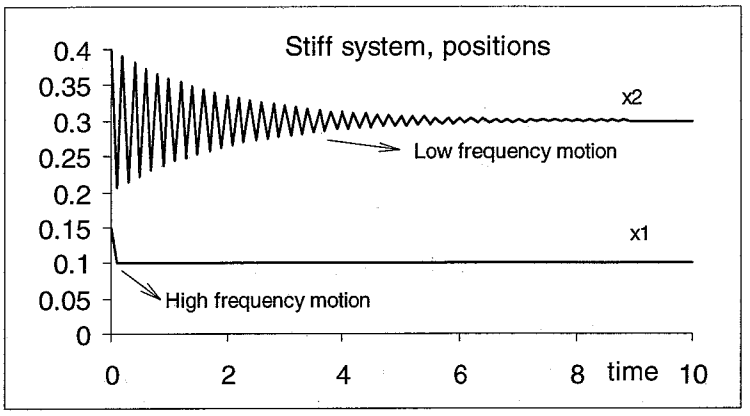
6.2 Stiff and non-stiff systems.

In order to investigate whether sytems can be classified as stiff, some simple tests are worked out. It will be investigated what integration method is used best in what kind of situation. Several double mass-spring-damper systems will be regarded. In all cases the system will be given a free (damped) motion. The springs have unstreched lenghts of l_{01} =0.10 m and l_{02} =0.3 m. The masses will be given the initial positions x_0 = [0.15; 0.40 m]. The response will be determined by implicit and explicit integration. The used CPU time and integration stepsizes will be regarded. Because of the large plotinterval, the grafical representation of the motions can be misleading (aliasing).

Test 1a.

This system has widely spread eigenvalues, the high frequency component will damp out completely during the simulation time. This system can be regarded as stiff.





Eig. Freq. Hz.	Damp ξ [-]
714	0.112
5.02	0.058

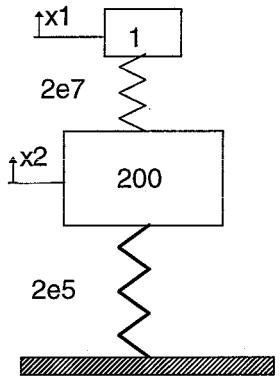
The high frequency component will damp out quickly and the solution quickly shows only the low frequency component. So during the simulation, different step sizes are 'asked' for integration. This is tested in MOTION and numerically integrated with the explicit and the implicit methods. As was to be expected for a stiff system, the solution is much faster with the implicit BDF method. See appendix A for the results.

The explicit method shows a relatively constant increasing CPU-time as function of simulated time. The method has to take small conservative time steps, even when the high frequency component is damped out, because otherwise the explicit Runge-Kutta routine would become numerically instable. The stiffness causes the method to have a small stability interval (see paragraph 3.4).

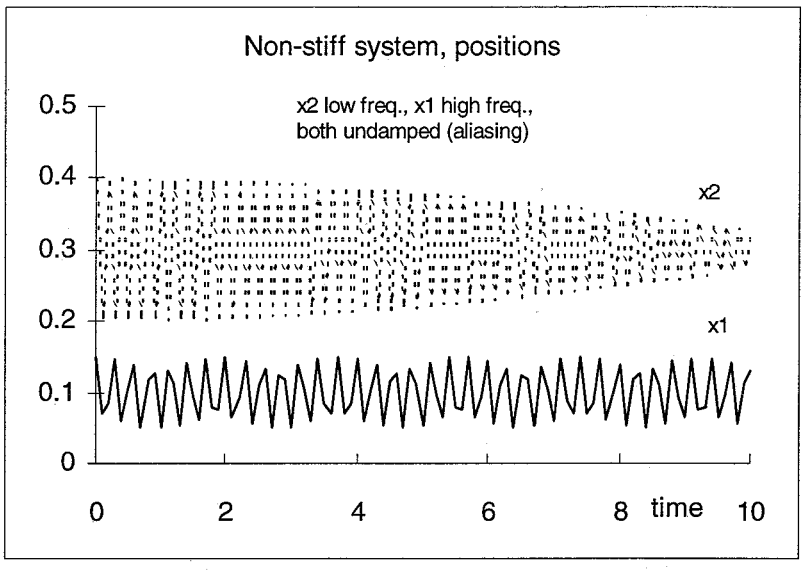
It has to be taken into account that, whether a system is stiff or not, is not only determined by the physical system itself but also by the simulation time. If this test-example would be integrated over just the time period where the high frequency component is present it would not be a stiff system.

The implicit method is much quicker. In the area of high frequency motion this method takes small timesteps too, to follow the rapid varying motion. But it's unconditional stability makes larger timesteps possible when the high-frequency component is damped out. This is where the method gains it's advantage. It is clear that an increase in stepsize only can be done when the high frequency components are damped out completely. The following test demonstrates this.

Test 1b:



Eig. Freq. Hz.	Damp ξ [-]
714	0
5.02	0



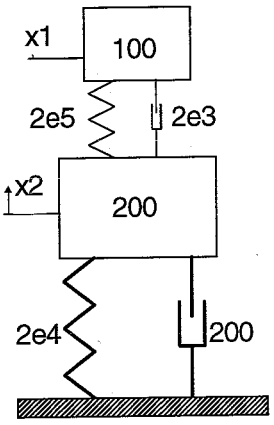
In this case the the high-frequency component does not damp out and both implicit and explicit integrators have to take small timesteps to follow the solution. Although the solution contains fast and slow components, for the numerical integration process in fact only the high frequency component plays a role since it is present everywhere. By definition this systems is not stiff and therefore is best integrated with the Runge-Kutta method.

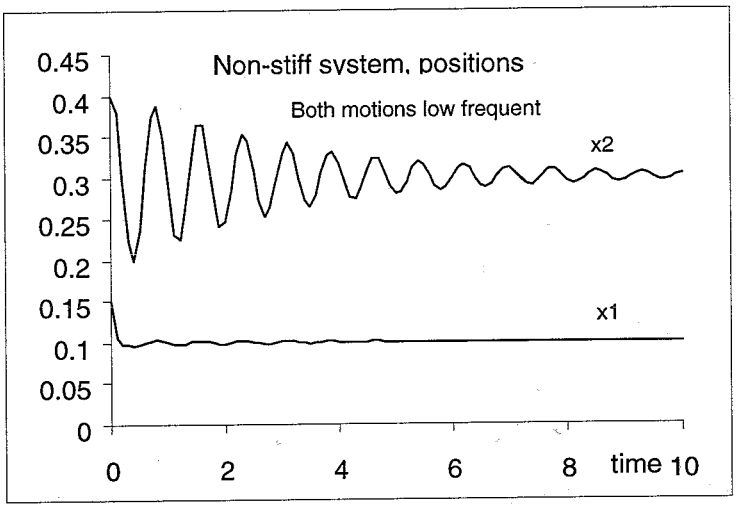
The advantage of the implicit method is totally lost, no stepsize increase is possible. In fact now the explicit method is a bit faster. An implicit step is more expensive than an explicit step (solving an implicit set of equations), so when the timesteps are of the same order over the whole solution-time for both explicit and implicit methods, explicit integration will be cheaper.

By adding the right amount of damping, this system can be made stiff and it will be like test 1a. In practice this is an important issue. When a system has widely spread eigenfrequencies and is badly damped, carefully chosen damping on the high frequency components will make the system stiff and CPU times can be decreased by using implicit integration. Of course the added damping must match physical reality. It will be clear that the advantage is only gained when all high frequency modes are damped.

Test 1c:

Non-stiff systems, systems with time-scales of the same order along the solution, are usually efficiently integrated with an explicit integrator. Again a two mass-spring-damper system will be reviewed, but in this case the eigenvalues will be close together. Both eigenfrequencies will be relatively low.



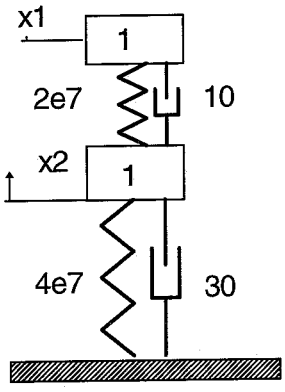


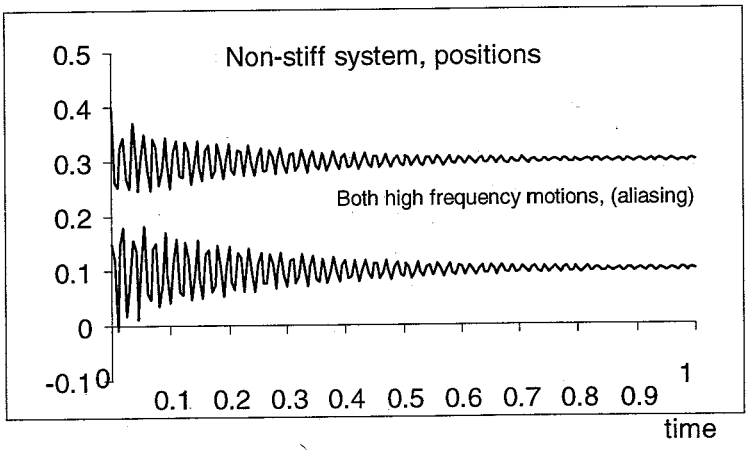
Eig. Freq. Hz.	Damp ξ [-]
8.38	0.288
0.41	0.13

There are no parts in the solution that ask different stepsizes, timescales are of the same order all along the solution. In this case the explicit integrator shows its advantages. As was to be expected it is faster than the implicit method (see appendix A).

Tests 1d:

This system has also eigenvalues of the same order, now both frequencies are high. For the same reason this is not a stiff system. Just like in test 1c, there are no parts in the solution that ask different stepsizes. The explicit integrator shows small advantage.





Eig. Freq. Hz.	Damp ξ [-]
1315	0.0026
545	0.0011

Compared to test 1c, more CPU time is needed for both explicit and implicit integration, because of the presence of high frequencies. In order to capture the fast motion the overall stepsizes are smaller than in test 1c.

Conclusions Test 1:

Tests 1a to 1d show the differences with respect to numerical integration of stiff and non-stiff systems. In a stiff system the solution has different timescales along the simulated time. At some parts, the fast components ask small timesteps, at other areas the high frequencies are damped out completely, only slow components are present. A non-stiff system has a solution that contains the same timescale all along the solution time. In fact only the fastest component dictates the timestep.

A stiff system is best integrated with an implicit method for efficiency. Because of the unconditional stability of such methods, they obey increasing stepsizes if the solution becomes smooth. Non-stiff systems with widely spread eigenfrequencies can be made stiff by adding a right amount of damping in order to lose the high-frequency components and make an increase in stepsize possible. The importance of damping will be reviewed again in the next chapter. Non-stiff systems are best integrated with the explicit Runge-Kutta method.

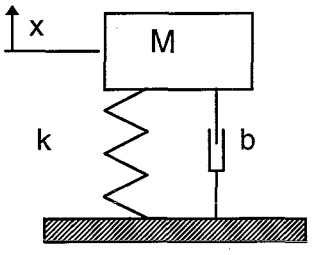
6.3 Discontinuities.

Discontinuities in non-stiff systems do not seem to be the biggest problem, explicit Runge-Kutta methods are used, and for non-stiff systems these methods usually outperform the other methods [10]. Discontinuities are handled efficiently for different reasons. No history information is used, only recent information is used so sudden changes in the solution can be followed. No implicit set of equations needs to be solved, so there is no Jacobian possibly getting singular.

For this reason attention will be paid to stiff systems with discontinuities, since implicit BDF methods seem to have more problems with discontinuities. This is also the experience at DAF Special Products. Sometimes stiff systems are integrated explicit because of the problems with discontinuities in combination with implicit integration. This is also one of the reasons for the approach of MADYMO to use only explicit integration: crash-simulation is a sum of discontinuities (see paragraph 4.2.4).

Test 2:

The following model is a one mass-spring-damper system. Although there is only one eigenfrequency, the system can be regarded as stiff when motion in the solution completely damps out. An implicit method is able to take large timesteps in the rest-situation (after motion has decayed). An explicit method still needs to take small steps in the rest-situation, because the derivative field is convergent (see paragraph 3.4). When this system would be integrated over a time period in which no rest-situation occurs (rest-situation can be compared with slow component), this would not be a stiff system.



M= 1 kg
k= 1e8 N/m
b= 2e2, 2e3 and 2e4 Ns/m
l_0= 0.5 m
freq.= 1592 Hz
ξ= 0.1 [-]

At t=0 the system is in rest. In different tests a variety of discontinuous forces will be put on M at t=2 in positive x direction. In all cases the force will reach a maximum value of 1e7 N, but by different functions. Due to the external force, M will move to a new equilibrium position at x=0.6 m.

It is interesting to look at the influences of the type of force-functions on the CPU time needed to overcome the discontinuity. The types of functions that will be reviewed are:

- Step-function
- Arctan-function with different slopes
- Cycloidal-sine-curve*

It is interesting to look at the effect of damping too, since an implicit method only can take advantage of it's possibility to increase step-size when high-frequency motion is decayed. With normal plot intervals it is not possible to see whether there may be a lot of undamped high-frequency motion immediately after the discontinuity.

Test 2a: Step-function.

In this test step-functions will be tested on systems with different damping (see appendix B). At first the damping is chosen ξ=0.01 and with $\xi = \frac{b}{2 \cdot \sqrt{m \cdot k}}$ and k=1e8, m=1, this yields b=200.

Two other damping situations are tested: ξ=0.1 with b=2e3 Ns/m and ξ=1 with b=2e4 Ns/m. A decrease of needed CPU time can be seen when the system is more damped (in the appendix B). Over critical damping leads to an increase in CPU-time again (this is not printed in the appendix).

It can be concluded that when a system with a high frequency eigenvalue is badly damped, a discontinuity will excitate this vibration and the (implicit) integrator has to decrease stepsize for a long time (compared to the period of the vibration) in order to capture this badly damped motion. In practice this situation can occur without recognition, a complex system passes a discontinuity and somewhere in the system a badly damped highfrequency component is excitated. Since the plot-interval can be much larger than the period-time of this vibration, it will be totally invisible.

* In Dutch: 'scheve sinus'.

Test 2b: Arctan-function.

The discontinuity is now an arctan function with the 'vertical part' at t=2. The rest of the system is the same as the critical damped system of test-2a (ξ=1).

The arctan function is:

$$F = 3.18 \cdot 10^6 \left(\arctan\left(A \cdot (t-2)\right) + \frac{\pi}{2} \right)$$

By the parameter A the slope of the arctan can be controlled, the smaller A, the smoother the arctan function. With this function the total force of 1e7 N is reached too, but since this function forms a less abrupt discontinuity one would expect the integrator to pass it faster. For A=1e4 this is not the case, the CPU-time needed for the discontinuity is the same as with a step function (see also appendix B). Other slopes are tried but no faster calculations are found than the one with the step-forces (therefore this is not plotted in the appendix).

For a less steep arctan-function, A=1e3, even an increase in CPU time, compared to a step-function, is found. This can be explained by the fact that it will take longer for this force to get a constant value and so it takes longer to reach equilibrium position and obey step increase of the integrator.

Only in the weakly damped case for ξ=0.01 a decrease in CPU-time is found compared with step-functions for an arctan-function with A=1e3. This lead to a CPU time of 16.5 sec compared to 19.5 sec for a step-function.

So the arctan-function, at least in this simple test, does not show much numerical advantage. A clear disadvantage of arctan-functions is the fact that they reach only a constant value at infinity. This is very inconvenient to the user in practice, one has to specify for example the time for the function to reach 99% of its end-value. A smooth function that does not have this disadvantage is the cycloidal-sine-curve, discussed in the next test.

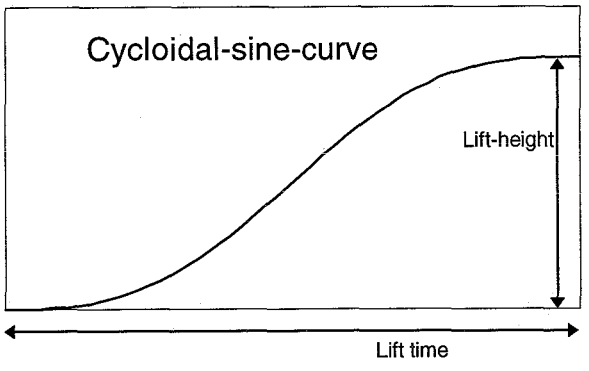
Test 2c: Cycloidal-sine-curve.

The cycloidal-sine-curve is a smooth function widely used as a profile for cam-mechanisms because of its convenient mechanical properties. When used as a cam-profile the cam-follower has smooth accelerations and little vibrations are excited. For the 'same' reasons this function is convenient for numerical integration.

A cycloidal-sine-curve can be written as:

$$y(t) = \frac{h_l}{T_l} \cdot t - \frac{h_l}{2\pi} \sin\left(\frac{2\pi \cdot t}{T_l} \right)$$

with h_l the 'lift-height' and T_l the 'lift-time'.



The lift-height is exactly reached with a horizontal slope at the beginning and end of the lift-time. Of course in stead of the lift-time a cam-angle or a spring-displacement can be read.

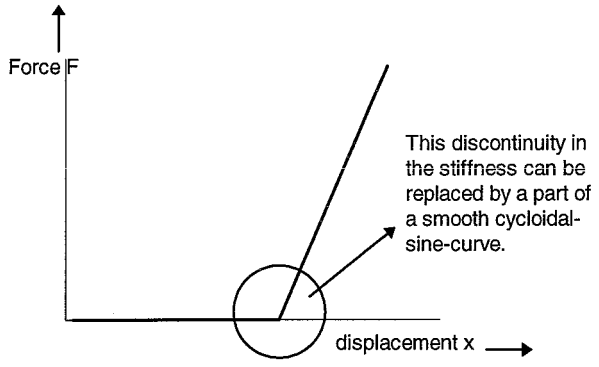
The same system is tested again with the same maximal force of 1e7 (lift-height) and a series of lift-times, this for two damping situations ξ=1 and ξ=0.01. The cycloidal-sine-curve is used with lift-times of 0.001, 0.005 and 0.01 sec. For exact comparison only the 'vertical' part of the CPU-time-curve is regarded, in other words, only the CPU-time needed to integrate the discontinuity.

For the critical damped situation ξ=1, with a lift-time of 0.001 sec, the cycloidal-sine-curve reduces CPU-time from 0.79 sec, the situation with the stepforce, to 0.43 sec. (see appendix B). For a lift time of 0.01 sec even 0.35 sec is reached, this is 44% of .79. These results were not expected since the arctan function showed no advantages at all. For the less damped system with ξ=0.01 the CPU time is reduced from 16.2 to 13.8 sec. for a fast lift-time of 0.001 sec. For a bit slower lift-time of 0.01 sec this is even reduced to 3.82 sec, a reduction to 24% of the value with the step-function. One test is done with a longer lift-time of 0.02 sec but in that case a small increase in CPU-time is found for the same reasons as with the arctan-test: it takes longer for the system to reach a stable situation, the time area where small stepsizes have to be take is longer.

It can be concluded that the cycloidal sine curve shows numerical advantage compared to step-functions. Two effects play a role:

- The discontinuity itself is smoother, so it forms less problems for the integrator.
- The smoother discontinuity excitates less (high-frequency) vibrations asking for small timesteps.

This knowledge may be used in real models. The function can be used for input tables with forces as function of displacement, explicit time dependent forces and springs that are conditionally active etc. see figure.



Smoothening by means of a cycloidal-sine-curve will also match better the physical reality. In practice 'sharp edges' do not occur. By the parameters lift-height and lift-time, and for example combinations of lineair functions with (parts of) cycloidal-sine-curves, desired functions can be made.

6.4 Flexible Beams in MOTION.

A flexible beam in MOTION is in fact a six-degree-of-freedom load. A 'beam-load' can be placed between two joints and the load will consist of six moments and forces all function of the six internal degrees of freedom of the beam. Since the beam in MOTION is modeled as a load the beam-element itself is massless. The mass of the beam in the system has to be divided and put on the two part-joints. When flexible beams are used in a model, MOTION gives the advice to use a stiff-system-integrator, in other words the implicit BDF method.

Systems with flexible beams can become stiff, since the six degrees of freedom of the load experience very different stiffnesses. The masses at the ends of the beam are the same and for slender beams the axial stiffness is much higher than bending and torsion stiffness. This results in a system of six mass-spring-damper systems with totally different eigenfrequencies. When the high frequency axial vibration is damped out slow components stay in the solution and this can be regarded as a stiff system.

The internal damping of the beam-element can be input in MOTION with the beam-load input table, default this damping coefficient is 0.001. This damping value is multiplied with the six stiffnesses• . So for slender beam geometries, the axial translational degree of freedom will have the heaviest damping. Mass terms are equal for all six degrees of freedom so with $\xi = \frac{b}{2 \cdot \sqrt{m \cdot k}}$ it is trivial that the axial motion has the largest ξ too and will damp out faster than the other motions.

• It can be found in the MOTION Help: The damping value is multiplied with the 'stiffness matrix' but there is no real stiffness matrix in the equations since the six stiffness terms appear on the right hand side of six equations, so they are handeled as external loads as function of positions.

Test 3: Stiffness and damping.

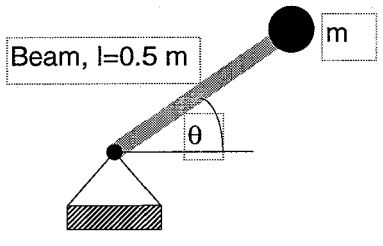
In this test the stiff character of the beam element will be reviewed. The effect of the damping value on the vibrations will be regarded.

A standard flexible beam in MOTION, with a length of 0.5 m, height of 0.05 m and width of 0.02 m, is fixed to the ground with one end, the other end is attached to a mass of 50 kg. Then this beam is given initial conditions on all six degrees of freedom (rotations and positions). A free-damped motion will result. This is integrated with a very small plot interval (i.e. max. integration time step). As was to be expected the axial motion is of another frequency-order than the other five motions.

For different damping values. on the left hand side (appendix C) the detailed motion of the axial translational vibration is showed, on the right hand side the other five degrees of freedom are found. Indeed the axial vibration is of a higher frequency than the other five. It can be found that the axial motion is critically damped, when the damping value of 6.8e-4 is used. The other vibrations are damped too by the internal damping, but they are far from critically damped (slender beam). The damping value for which critical damping occurs depends on the geometry of the beam, the attached mass etc. If for example this test is done with the same beam but with a mass of 150 kg (3 times more) critical damping will occur for 1.18e-3 ($\sqrt{3}$ times more). Also for a mass of 10 kg (5 times less) critical damping occurs for 3.04e-4 ($\sqrt{5}$ times less). This shows again that a flexible beam in MOTION is a set of six mass-spring damper systems.

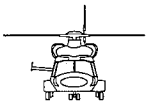
Test 4a: Influence of damping on CPU-times.

What is the influence of this damping value on the CPU time? The following test is made: A flexible beam (with different geometry and mass as with test 3) is connected to the ground by a rotational pin-joint. The beam is given an initial rotation of 1 rad. It can freely rotate by gravity until the angle θ between the beam and the horizontal axis is zero. Then a stiff rotational spring becomes active.



m=	20 kg
l=	0.5 m
Ixx=	1e-9 m^4
Iyy=	1e-9 m^4
A=	0.001 m^2
E=	1.9e11N/m^2

In this case the fastest CPU-time is found with the default damping value of 0.001: 160 sec. (see appendix D). When the axial motion is studied in detail again, it is found that for this system the motion is critically damped for 0.001. It is difficult to find the amount of damping for which the system is exactly critical damped, but in these tests it was not very sensitive: 5 % more or less damping than critical damping is hardly found in the CPU-curve.



In practice it would be interesting to test each flexible beam with a small plot-interval (maybe isolated out of the system with a realistic mass term) for the right amount of damping, in order to achieve critical damping and optimal performance.

Test 4b: Locking axial motion.

Now the effect of locking the axial translational d.o.f. by means of a constant length driver is investigated. The high-frequency axial vibration will disappear. The five left d.o.f. will form a non-stiff system since their eigenfrequencies are of the same order. Indeed when this system is integrated explicitly a CPU time of 90 sec. is found (appendix D).

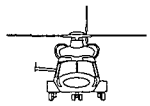
Now even the implicit method will be a bit faster than in test 4a, since no high-frequency motion needs to be integrated. Implicit integration of this non-stiff system takes 150 sec. That is slower than explicit, but a little faster than implicit with axial vibration (test 4a).

When a system is stiff only because of flexible beams and one is not interested in the high-frequency axial motion, then it may be interesting to lock this motion. In that case explicit integration can be used (compare the CPU-time of 90 sec. in this test with the 160 sec. of test 4a).

Even when the system is stiff because of flexible beams in combination with other reasons, it may be an advantage too to make the beams non-stiff by this method. Then still implicit integration must be used (since the rest of the system is stiff) but the beams do not ask small stepsizes (compare the CPU-time of 150 sec. in this test with the 160 sec. of 4a).

Test 4c: Explicit integration of flexible beam.

This tests (appendix D) shows that a normal flexible beam, with free axial stiffness, must not be integrated with an explicit integrator. A CPU-time of 1200 sec. must be faced, compared to test 160 sec. with an implicit integrator in test 4a. This situation can be compared with explicit integration in test 1a.



7. DISCUSSION, CONCLUSIONS AND RECOMMENDATIONS

Discussion

In the scope of this project, relevant theory of multibody dynamics is reviewed. This has led to a clearer insight in backgrounds and methods in general, and especially in methods used in the multibody package MECHANICA MOTION. Some of the black-box character of MOTION has been taken away. Practical tests have explained the questions stated in chapter 6. Results of these simple tests can be used in more complex models. The philosophy of Pro/Mechanica is to make user-friendly software, not to give the user insight in methods and backgrounds.

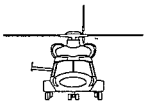
Conclusions

- Integrators in MOTION are more conservative and accurate than fast.
- A stiff system has different timescales along the simulation-time, in other words, during integration high-frequency motion is present at some part of the solution, and is completely decayed at another part.
- Stiff systems are best integrated with the implicit BDF method. During integration of the solution of a stiff system, timesteps of different magnitude are required. The BDF method allows stepsize increase in a convergent direction field without losing stability.
- Effectively damping of unnecessary high-frequency motion, can lead to a situation in which implicit integration is faster. In fact by making the system stiff it is more suitable for implicit integration.
- A flexible beam is modeled in MOTION as a six degree of freedom load. These six loads depend on the translations and rotations between the two end-points of the beam. For slender beams the axial motion has a frequency of higher magnitude than the other five degree of freedoms. When this motion damps out during solution, the beam can be considerd as stiff.
- Internal damping of flexible beams is presented by a factor on the six stiffnesses (proportional damping). The internal damping can have great influence on needed CPU times. Badly damping of the (axial) motion leads to unrealistic small timesteps and large CPU times.
- Cycloidal-sine-curves lead to less discontinue systems. Numerical advantages are gained in a test with cycloidal-sine-curves functions as external forces. These smooth functions can be used in a lot of practical situations.

Recommendations:

- Test the influence on integration of accuracy, assembly and velocity tolerance* .
- Try to extrapolate results from the small tests to more complex realistic models.
    - look for hidden, unnecessary badly damped high-frequency vibrations.
    - try out the use of smooth cycloidal-sine-curves in real models.
- Compare a representative model run with MOTION, with another multibody package.

* A received fax from Customer Support Germany can be of use with this.



References

[1] E.J. Haug — Computer-Aided Kinematics and Dynamics of Mechanical Systems, Vol.1. Allyn and Bacon, 1989.

[2] P.E. Nikravesh — Computer-Aided Analysis of Mechanical Systems. Prentice Hall, 1988.

[3] A.A. Shabana — Computational Dynamics. Wiley-Interscience, 1994.

[4] A.Sauren — Multibody Dynamica, Dictaat TUE 4659.

[5] P.M.E.J. Wijckmans — Conditioning of Differential Algebraic Equations and Numerical Solution of Multibody Dynamics. Ph.D. thesis. TUE 1996.

[6] W. Schielen — Advanced Multibody System Dynamics. Kluwer Academic, 1993.

[7] T.R. Kane, D.A. Levinson — Dynamics: Theory and Applications. McGraw Hill, 1985.

[8] R.M.M. Mattheij — Numerieke Analyse van gewone Differentiaal vergelijkingen, Dictaat TUE 2466.

[9] Reusken — Numerieke Methoden en programmatuur, Dictaat TUE 2434.

[10] The MathWorks — MATLAB SIMULINK User's Guide.

Appendix

Test results:

Test 1: Stiff and non-stiff systems.

Test 2: Discontinuities.

Test 3: Flexible beams.

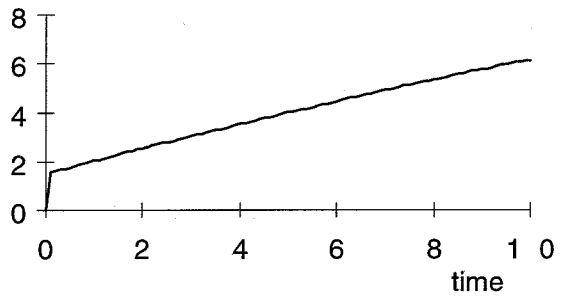
Test 4: Flexible beam with discontinuity.

Test1: Stiff and non-stiff systems.

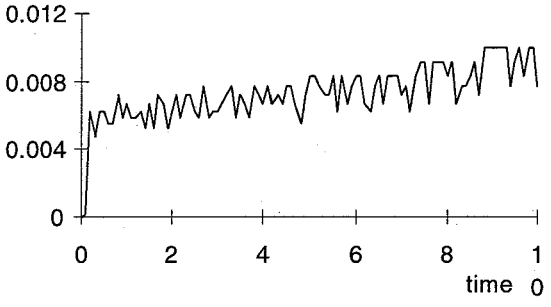
This serie of tests studies double mass-spring-damper systems, both stiff and non-stiff. See chapter 6.2 for descriptions.

Test 1a,
A stiff system is integrated both implicitly and explicitly. Mention the difference in CPU time and required stepsize. Implicit integration (above), is much faster than explicit integration (below).

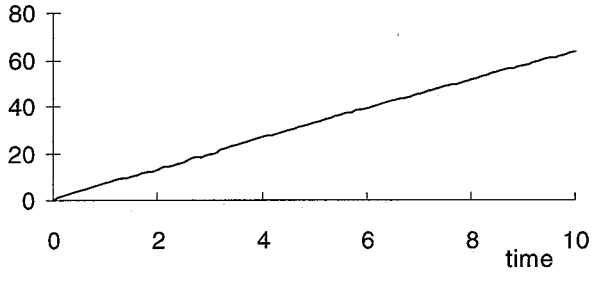
Test1a, implicit, CPU time



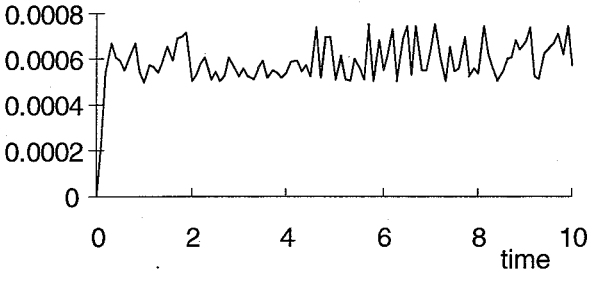
Test1a, implicit, average step size



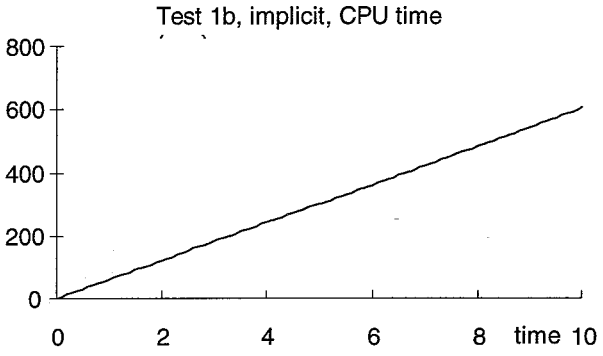
Test 1a, explicit, CPU time

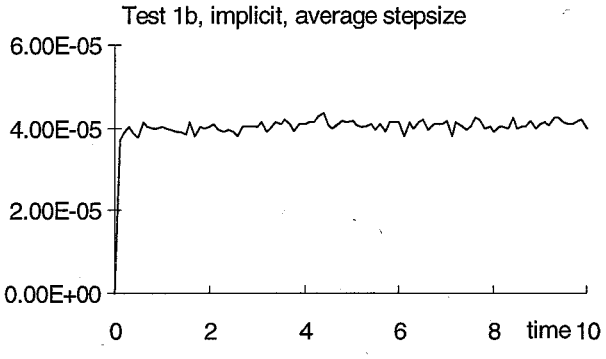


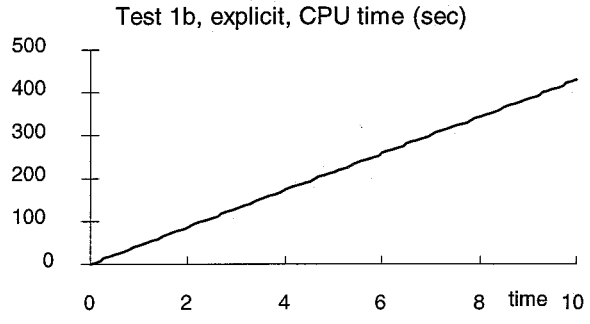
Test 1a, explicit, average stepsize

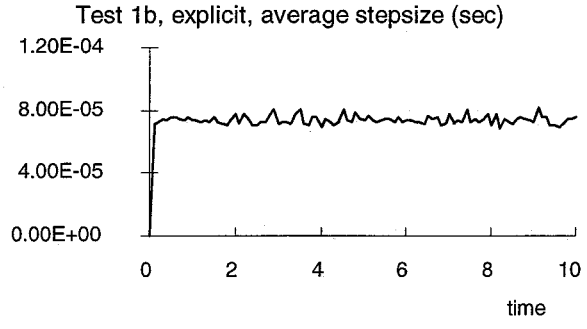


Test 1b.
This test is almost the same as test 1a, in this case without any damping. The implicit integration loses its advantage, high-frequency motion is present during the whole simulation time, stepsize increase can not occur. This is a non-stiff system.

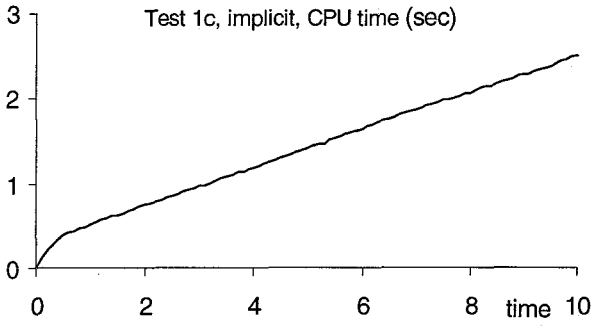


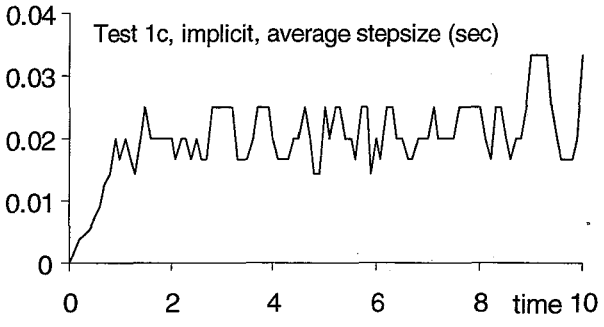


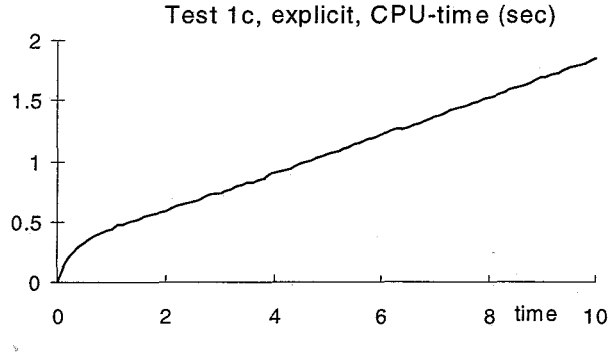


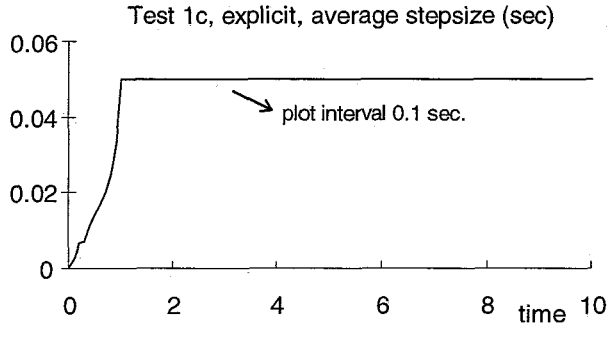


Test 1c.
This is a non-stiff system, both eigenfrequencies are of the same low magnitude. Again an explicit method shows to be slightly faster.

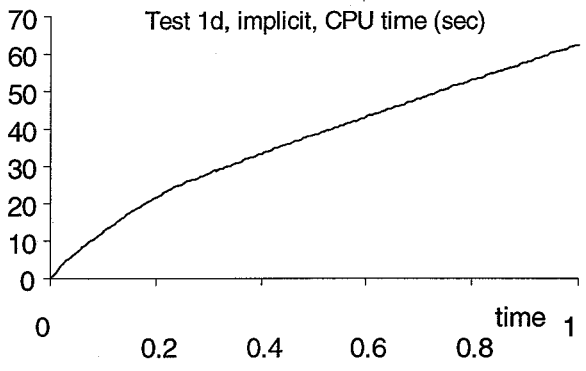


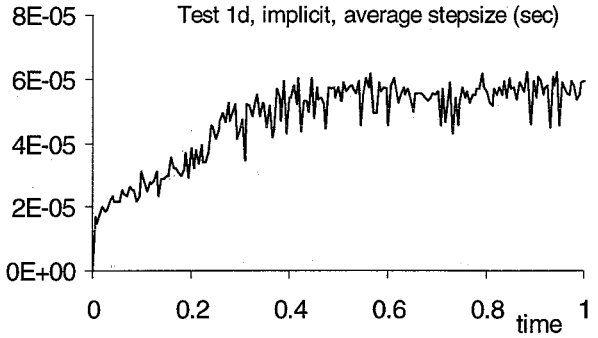


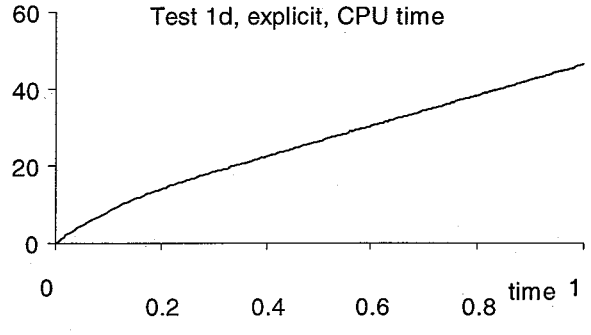


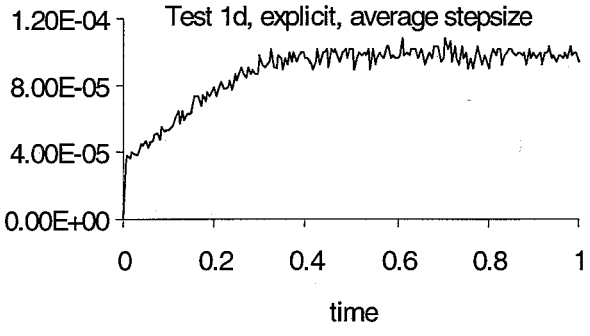


Test 1d.
Here both eigenfrequencies are of the same high magnitude. Absolutely spoken this test takes more CPU time than test 1c. The reason for this is the need for small timesteps in order to capture the fast motion. Again for a non-stiff system, explicit integration is a bit faster







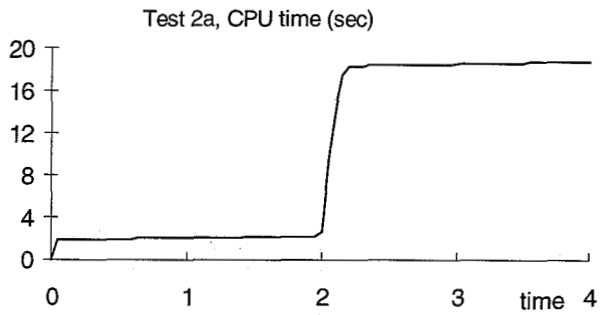


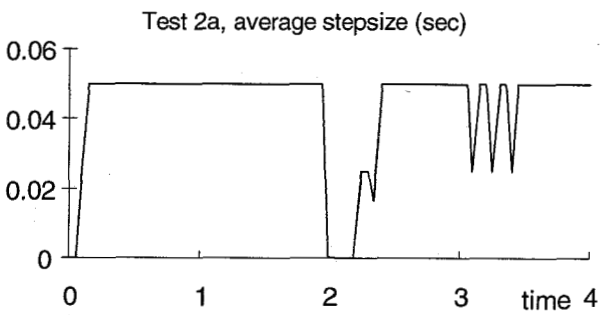
Test 2: Discontinuities

Test 2 studies the behavior of a one-mass-spring-damper system with respect to different discontinue external forces. See chapter 6.3 for description.

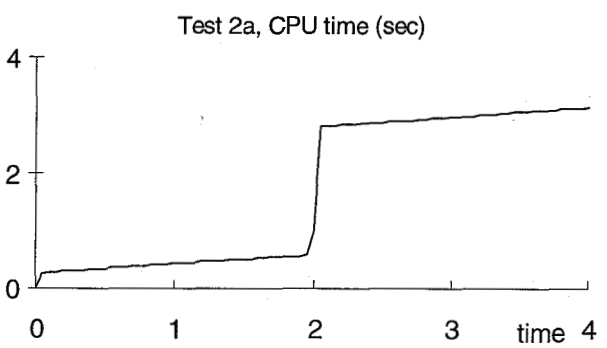
Test 2a.
The discontinuity is a step-force at t=2 of 1e7 N. Different damping values are tested, from underdamped to critically damped situations. All situations are integrated implicitly. Mention the influence of damping on the CPU time. The discontinuity resulted by the step-force is of less influence if the system is critically damped. In the badly damped case the high frequecy motion, excitated by the step-force, requires small timesteps and leads to long CPU times.

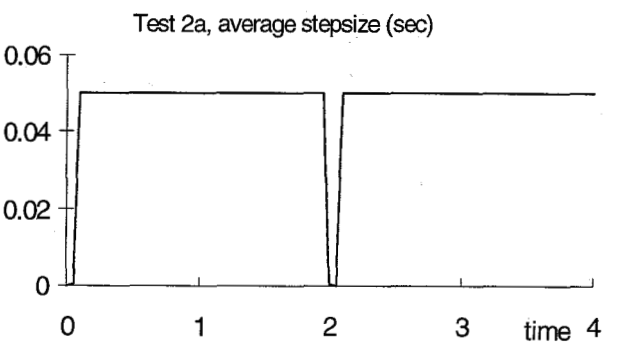
ξ=0.01



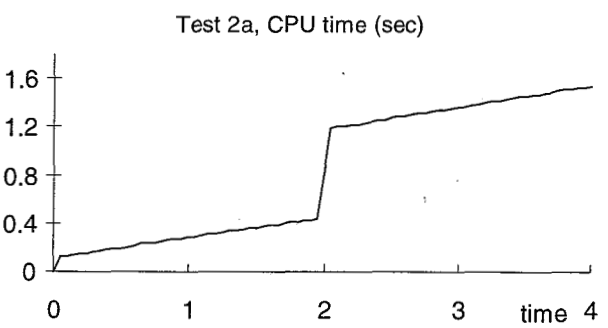


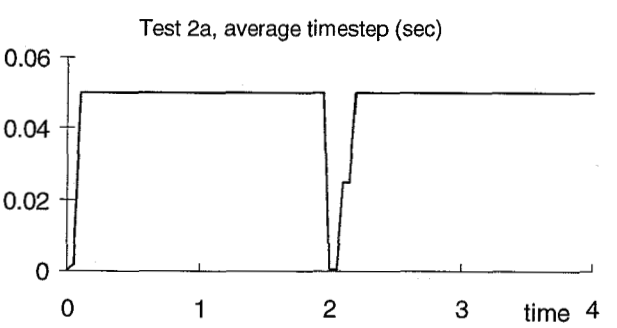
ξ=0.1





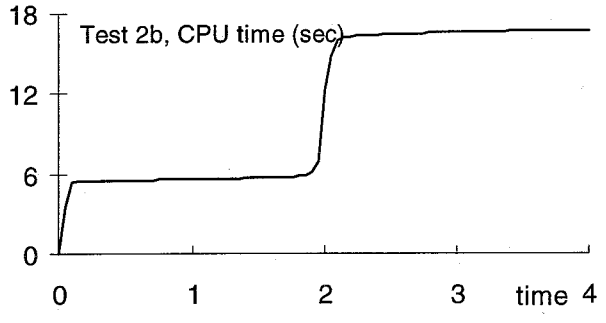
ξ=1

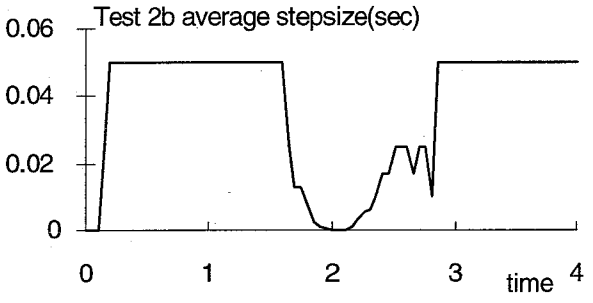




Test 2b.

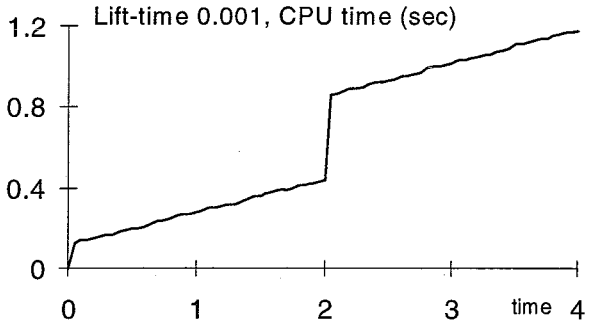
No numerical advantages are found with an 'arctan-function shaped' discontinuity. Other arctan-functions with different slopes even increase required CPU times. Mention the large dip in the stepsize curve, this is due to the fact that the function is still changing for a long time (in fact till infinity).The fact that the mass does not reach an equilibrium state, is not very convenient for practical use. The same problem is found at the start, where does the function 'start growing'?

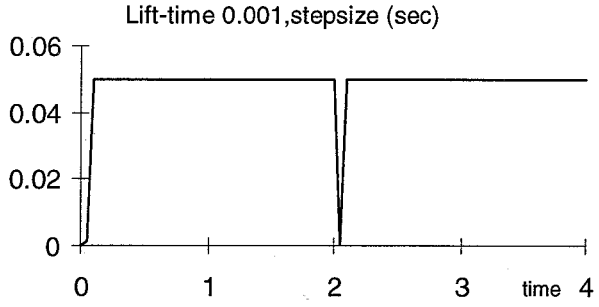


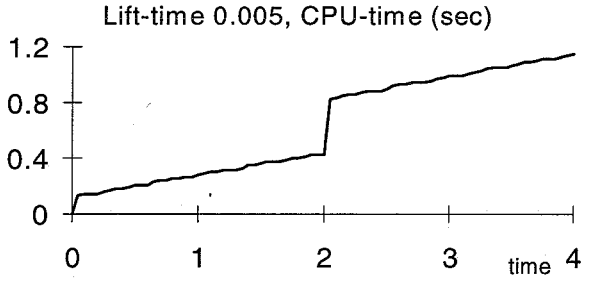


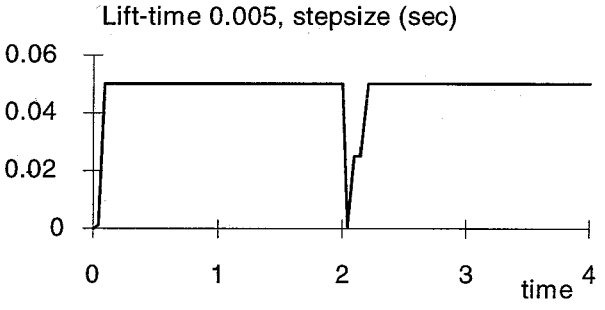
Test 2c. ξ=1

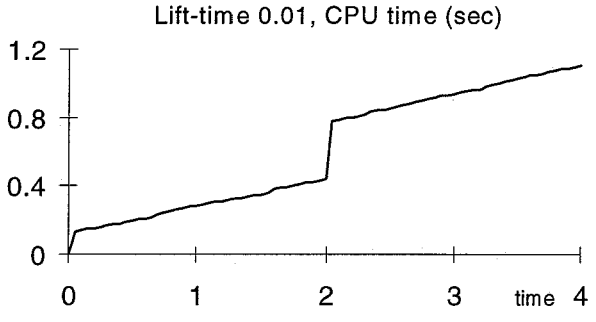
A cycloidal-sine-curve is used as external force. The lift-height is always the same: 1e7 N, lift-times are varied from 0.001 to 0.01 sec (for definitions, see page 9 of this appendix). On this page damping is critical: ξ=1. Compared to the CPU-times found in test 2a, numerical advantage is gained. These smooth functions form a smaller discontinuity for the integrator.

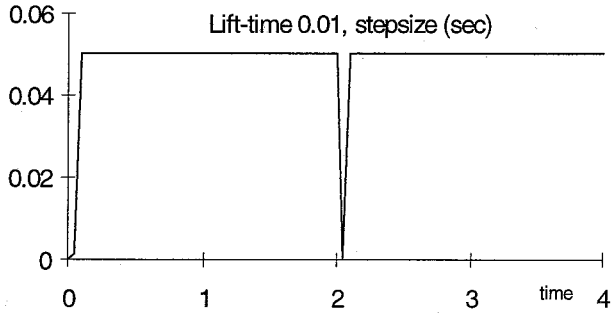




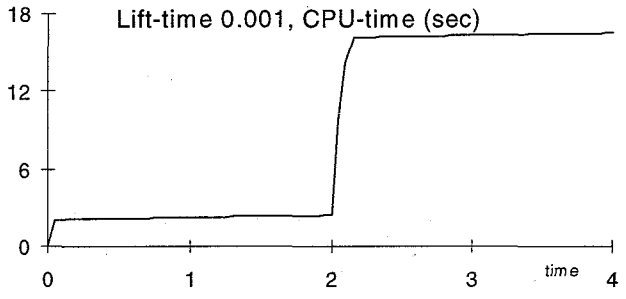


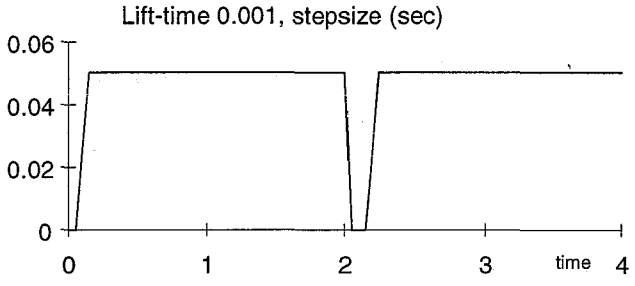


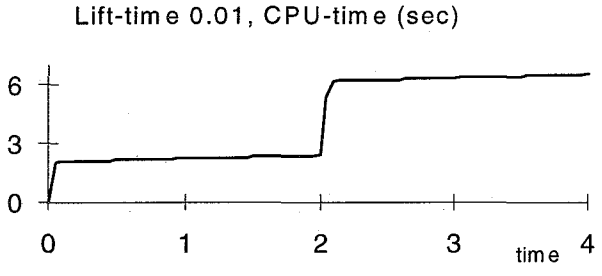


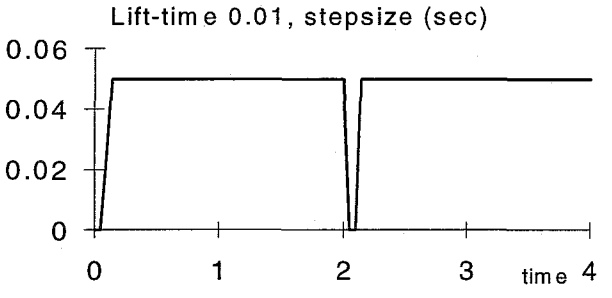


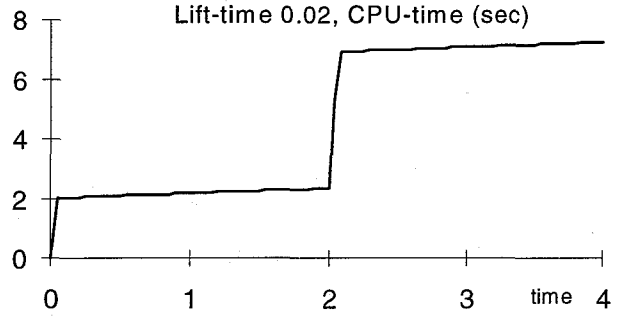
test 2c. ξ=0.01
Again cycloidal-sine-curves with different lift-times, now for a weak damped situation: ξ=0.01. A lift-times of 0.01 sec reduces the CPU time remarkably (compared to 2a). The smooth function excitates less high frequency motion.

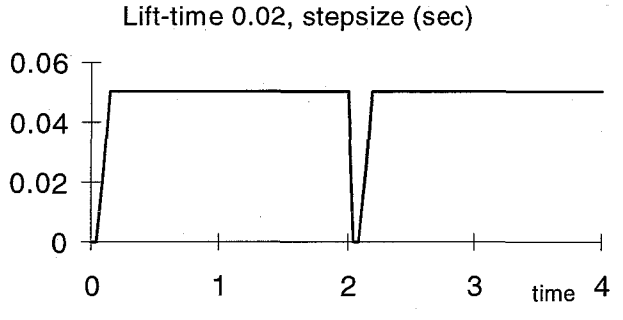




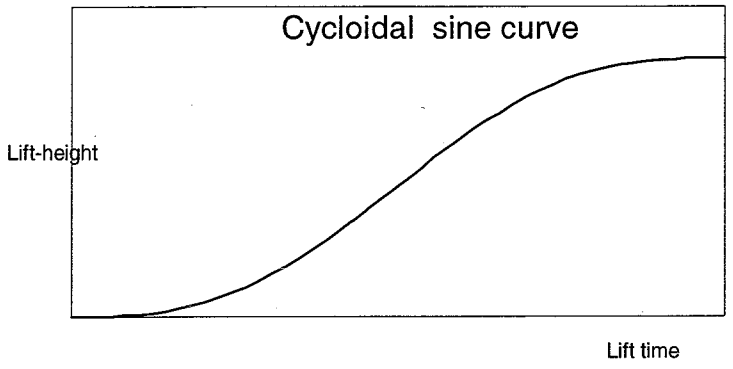








Mention the slight increase in CPU time for the lift-time of 0.02 sec: it takes longer to reach the equilibrium state.



An overview of the CPU-times found in tests 2c. This must be compared with the table below: results of the step-function from test 2a.

Cycloidal-sine-curve		
	CPU time for discontinuous part (sec)	
Lift-time	damping ξ=1	damping ξ=0.01
0.001	0.43	13.8
0.005 [1]	0.42	8.8
0.01	0.35	3.82

CPU times found in test 2a with a step-force of 1e7 N:

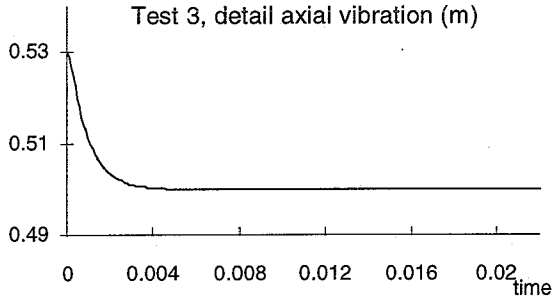
CPU time for discontinuous part (sec)	
damping ξ=1	damping ξ=0.01
0.79	16.2

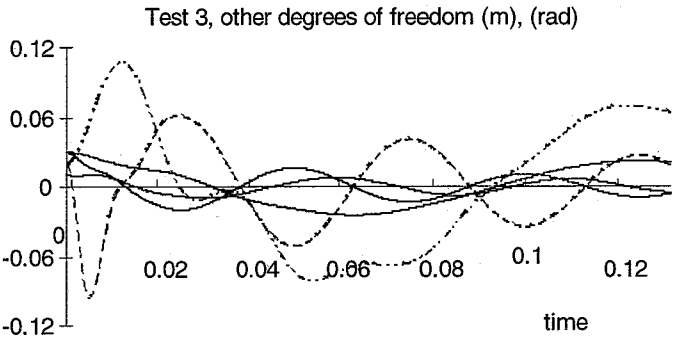
[1] There are no plots of this lift-time

Test 3: Flexible beams

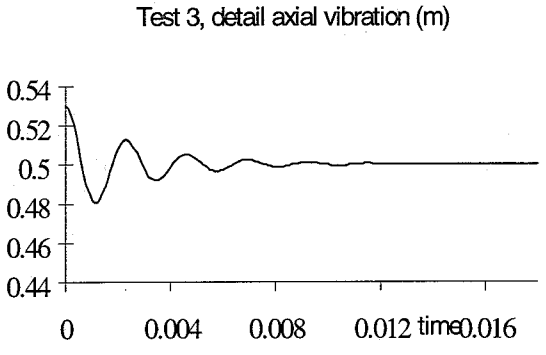
Mention the different frequency magnitudes of the axial motion and the other five d.o.f. of a slender beam.

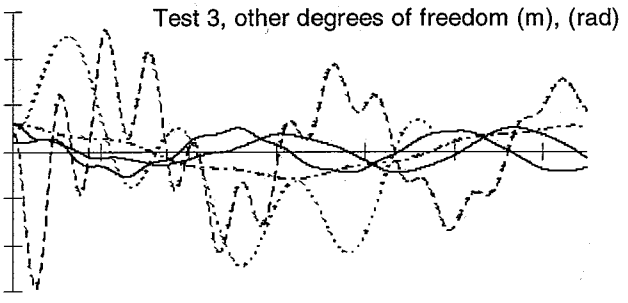
damping value=0.001 (default)



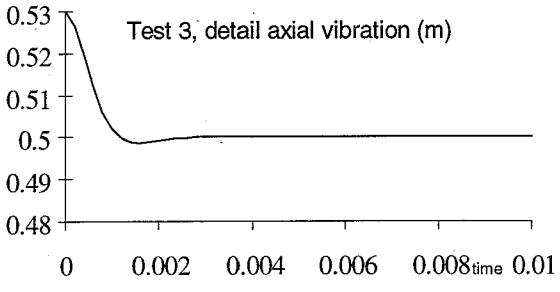


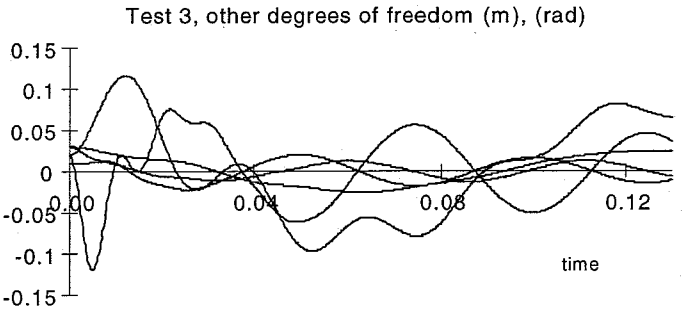
damping value=0.0001





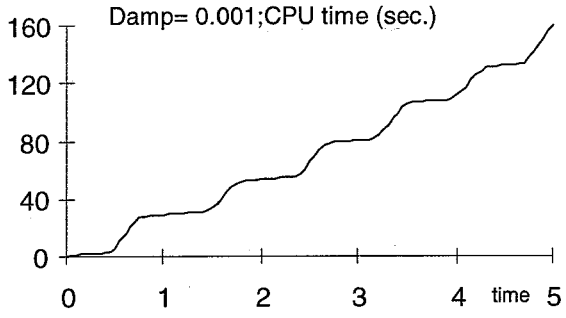
damping value=0.0005

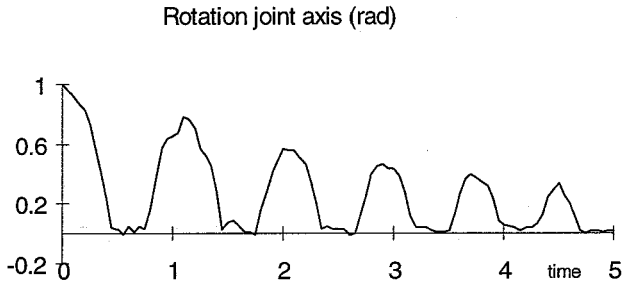


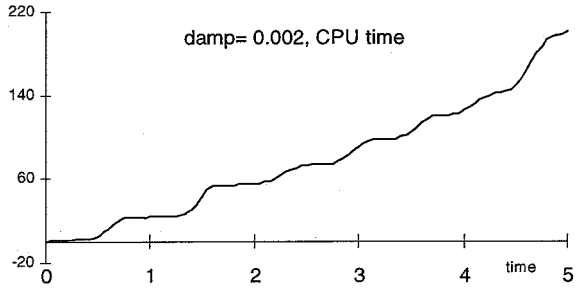


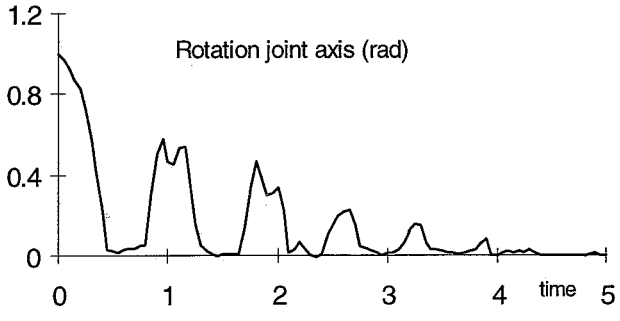
Test 4: Flexible beam with discontinuity.

Test 4a.
The influence of the damping value on the CPU time is tested. Fastest integration is found with a damping value of 0.001. On the righthand side the rotational motion of the beam can be viewed, the damping in the motion is only caused by the internal damping of the beam load.



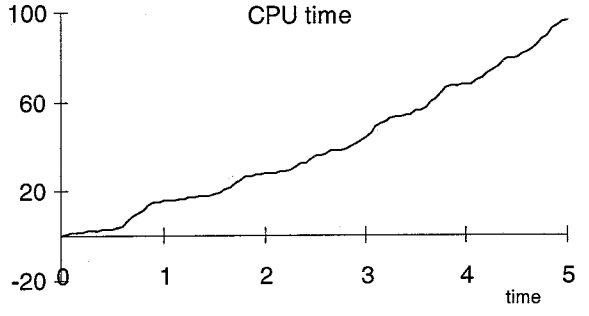


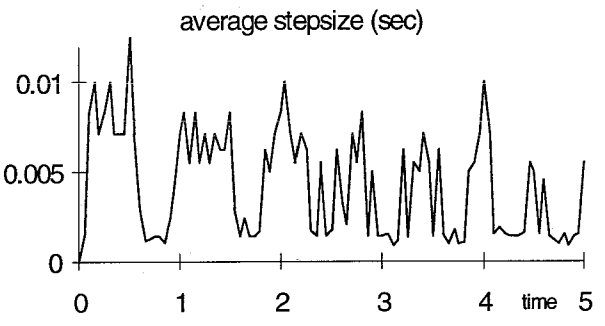




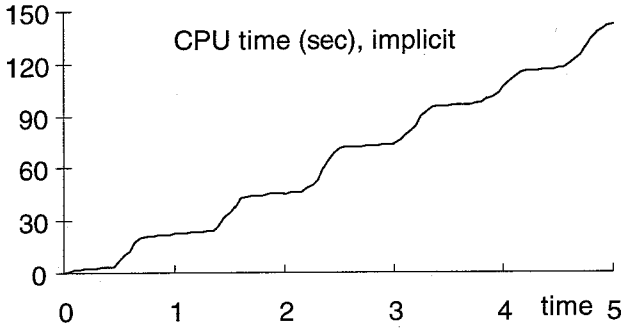
test 4b.
This test studies the effect of locking the axial motion by a constant lenght-driver. In that case the beam is no longer a stiff element, and so is best integrated explicitly:

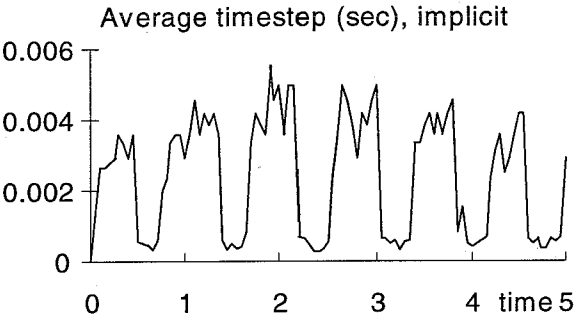
Explicit integration:





Implicit integration:





test 4c
This trivial test shows that a standard slender beam (with the same configuration as test 4a), is better not integrated explicitly. Compare these CPU times and time-steps with 4a.