Realtime Simulations with Inline Integration and Mixed Mode Integration

Mikael Gustafsson

Master's thesis
2021:E74

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

Abstract

In this thesis we discuss and test ways to improve the performance of inline integration.
Inline integration is a tool used for real-time simulations of complex dynamical systems. The idea is to relate information about the system to the numerical method at a model level.
Real-time simulations of complex models have become standard in industry. This puts high demands on simulation tools with regard to computational speed.
Mixed mode integration is treating stiff variables and non-stiff variables with different numerical methods, in doing so we expect to reduce the dimensions of the non-linear systems that needs to be solved and hence increase the computational speed. We investigate what mixed mode integration means with inline integration and elaborate on the idea using an electrical circuit as an example.

Acknowledgement

I would like to thank my supervisors Claus Führer and Christian Winther for criticism and advice during the process of writing my masters thesis.

Contents

1 Introduction

In real time simulations one has a predetermined run-time for how long the numerical solver has to update the solution. One example of when real time simulations is useful is when connecting a real plant to a simulation of a controller. The simulation of the plants controller is used to rapidly develop a prototype of the plants real controller. This setting is called rapid controller prototyping (RCP). In industry RCP is used to reduce the cost and time required in order to devolop a plants controller [2]. Such settings puts high demands on computational speed for the simulation tool. This thesis is done in collaboration with Modelon AB which is a software company based in Lund. They have developed a powerful tool for real time simulations called inline integration where the idea is to mix symbolic and iterative methods.

With the aim to increase the computational speed we investigate an idea called mixed mode integration that originates from ordinary differential equations (ODE). Mixed mode integration is treating different state variables with different numerical solvers, namely explicit or implicit solvers. The symbolic methods behind inline integration is trying to reduce the dimensions of the equations that needs to be solved to update the solution. By introducing mixed mode integration into inline integration we expect to aid the symbolic solver in doing so and hence increase the computational speed.

Mixed mode integration requires splitting the state variables that appear in differentiated form into those which will be solved by an explicit method, we call these "slow" state variables and those which will be solved by an implicit method, we call these "fast" state variables. In Section 3 we look at such a partitioning on ODE's.

In Section 4 inline integration is used on the spring pendulum with explicit Euler's method and implicit Euler's method. The symbolic methods generate a set of equations that needs to be solved to update the solution. We use the example to get a better understanding of how inline integration could benefit from mixed mode integration.

The real time simulation tool is used on a special case of implicit ODEs, ie

$$F(\dot{x}, x, w, t) = 0 \tag{1}$$

where $F : \mathbb{R}^{n_x+n_w} \mapsto \mathbb{R}^{n_x+n_w}$ and with $\frac{\partial F}{\partial \dot{x}}$ being singular. These equations are called differential algebraic equations (DAE) and model dynamical systems that arise in various field such as multibody dynamics, chemical process control and electrical circuit design. In Section 5 we discuss the idea of mixed mode integration using graph theory and present a proposition that gives us an idea of what changes in terms of the dimensions of the systems of equations when we mix explicit and implicit solvers. A simple electrical circuit is then used as an example to build upon the proposition and to investigate if we can introduce what we call 'a delay of information' such that we split the system of equations that needs to be solved into two parts. We formulate the delay of information using Newton's method. The delay of information is seen as a change to the Jacobian. We then combine Woodbury's matrix identity [7] with Banach's fixed point theorem to investigate the convergence of the split system.

1.1 Task description

During simulation, time moves forward in discrete steps of equal duration, meaning that the time step h is fixed. The clock time required for a simulation tool to update the solution from time t_n to t_{n+1} may be shorter or longer then h. In real time simulations the simulation tool must give a sufficiently accurate update of the outputs within the time step duration. As a matter of fact the time required to update the outputs must be shorter then the time step, this is called 'the wall time'. With a sufficiently accurate solution within wall time we can permit the needed input and output operations to and from external devices. If a sufficiently accurate update isn't achieved within wall time it is considered erroneous. Such an update is called an overrun [2].

The task can be described as Modelons real time simulation tool, called inline integration, needs to update the solution of (1) without overruns. We will in the next section explain the general idea of inline integration.

2 Inline integration

2.1 Insert and extend with the numerical method

Consider a differential algebraic equation.

$$F(\dot{x}, x, w, t) = 0 \tag{2}$$

where $F : \mathbb{R}^{n_x+n_w} \mapsto \mathbb{R}^{n_x+n_w}$ and with $\frac{\partial F}{\partial \dot{x}}$ being singular. The idea behind inline integration is to closer relate information about the DAE with the numerical methods. The first step in doing so is to insert the numerical method at a model level. To explain what this means we need to introduce some notation. Consider the ODE

$$\dot{x} = f(x, t) \tag{3}$$

and discretize (3) using for example by implicit Euler's method

$$x_{n+1} = h \cdot f(x_{n+1}, t_{n+1}) + x_n \tag{4}$$

in our context we replace $f(x_{n+1}, t_{n+1})$ by $\dot{x}_{n+1}$ because the variable only exists in the model, then for a large class of implicit methods we would have the same structure for the discretization, namely

$$\mathbf{x} = h \cdot \dot{\mathbf{x}} + \text{old}(\mathbf{x}) \tag{5}$$

where for convenient notation we replace $\dot{x}(t_{n+1})$ by $\dot{\mathbf{x}}$, $x(t_{n+1})$ by $\mathbf{x}$ and $\text{old}(\mathbf{x})$ is a function depending on the previous values and the numerical method. For example, any backward differentiation method (BDF) belongs to this class of implicit methods. This notation was introduced in [5]. A general formula for a BDF method is

$$\sum_{k=0}^{s} \alpha_k \cdot x_{n+k} = h \cdot \beta f(x_{n+s}, t_{n+s}) \tag{6}$$

h denotes the step size and $t_n = t_0 + n \cdot h$. The coefficients β and a_k is chosen such that the maximum order s is achieved. Rewriting (5) such that it agrees with the formulation of (4) we put x_{n+s} on one side

$$x_{n+s} = h \cdot \frac{\beta}{\alpha_s} f(x_{n+s}, t_{n+s}) - \frac{1}{\alpha_s} \sum_{k=0}^{s-1} \alpha_k \cdot x_{n+k} \tag{7}$$

Where old($\mathbf{x}$) = - $\frac{1}{\alpha_s} \sum_{k=0}^{s-1} \alpha_k \cdot x_{n+k}$, is a known value that depends on the method and previous values.

Now if we insert the numerical method into the original DAE and extend the system by the numerical method, we get the augmented system

$$F_\Delta(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) = \begin{bmatrix} F(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) \\ -\mathbf{x} + h \cdot \dot{\mathbf{x}} + old(\mathbf{x}) \end{bmatrix} = 0 \tag{8}$$

The numerical integration algorithm is now applied "inline" with the model, this gives rise to its name. Think of this as solving

$$F(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) = 0$$

under the constraint that the equation for the numerical method is fulfilled.

Next we treat $\{\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}\}$ as algebraic variables and preform partitioning of the system.

2.2 Partitioning

Let us consider the DAE in augmented form

$$F_\Delta(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) = 0$$

and let us think of this as a system of equations with $z = (\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w})$,

$$h(z) = 0$$

The aim of partitioning is to reduce the work carried out by the solver. We want to know if, given that $h(z)$ is very large and sparse, is it possible to find subgroups of equations such that if the variables in one subgroup are known, those variables can be treated as known in the next subgroup. Finding such subgroups is called *partitioning* and the subgroups will be refered to as algebraic loops. Notice that the algebraic loops are systems equations, and even though we will reduce the work carried out by the solver the most time requiring part of updating the solution will still be solving the algebraic loops. The idea of this thesis is to investigate ways to further reduce the algebraic loops.

To give the reader an understanding of the general idea of partitioning we now explain the principle by an example. Consider a system of equations $h(z) = 0$ of the form

$$h(z) = \begin{pmatrix} h_1(z_1) \\ h_2(z_1, z_3) \\ h_3(z_1, z_2, z_3) \\ h_4(z_2, z_3, z_4) \end{pmatrix}$$

We represent a system by a structural matrix S, defined by

$$S_{ji} = \begin{cases} 0 & \text{if } \frac{\partial h_i}{\partial z_j} \equiv 0 \\ 1 & \text{otherwise} \end{cases}$$

For the example above,

$$\mathbf{S} = \begin{array}{c} \begin{matrix} z_1 & z_2 & z_3 & z_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{matrix}$$

The operations we have at our disposal are interchanging rows or interchanging columns. The hope of doing this is to get a strictly lower triangular matrix. For example if we would interchange column 2 by column 3 we would get

$$\widetilde{\mathbf{S}} = \begin{array}{c} \begin{matrix} z_1 & z_3 & z_2 & z_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{matrix}$$

If the structural matrix can be put into strictly lower triangular form we say that we have found an explicit sequence in which we can solve the system. Reordering to lower triangular form cannot be done in general. To

see this just imagine that we have a system with the structural matrix

$$\mathbf{S_1} = \begin{array}{c} \begin{array}{cccc} z_1 & z_2 & z_3 & z_4 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \\ h_4 \end{array}$$

$\mathbf{S_1}$ cannot be put into a strict lower triangular form, but we can present one triangular block along the diagonal such that equations in that block can be solved in an explicit sequence. Such a matrix is called block lower triangular (BLT) and in this case it looks like

$$\widetilde{\mathbf{S}}_\mathbf{1} = \begin{array}{c} \begin{array}{cccc} z_1 & z_3 & z_2 & z_4 \end{array} \\ \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \\ h_4 \end{array}$$

We can now solve $h_1(z_1) = 0$ for z_1, then we use z_1 as known in $h_2(z_1, z_3) = 0$ and solve for z_3, then we use the values for z_1 and z_3 as known in

$$h_3(z_1, z_2, z_3, z_4) = 0 \tag{9}$$
$$h_4(z_1, z_2, z_3, z_4) = 0 \tag{10}$$

to solve for z_2, z_4. By doing the partitioning we have consequently gone from solving one 4×4 system to solving two 1×1 systems and then one 2×2 system.
Modelon's inline integration tool is used on DAEs that model complex dynamical systems using Modelica. Since Modelica is an object oriented descriptive language constructed to easily assemble models by reusing parts and connecting them by special connector functions, we will often end up with a sparse structural matrix. The sparsity makes partitioning very efficient.

Next we look at the equations that are possibly rearranged and we will use that information to further reduce the work carried out by preforming tearing on the algebraic loops.

2.2.1 Tearing

The algebraic loops are put into a tearing algorithm, with the aim to reduce the dimensions of the system of equations that needs to be solved. The tearing algorithm takes a system of equations

$$h(z) = 0$$

where $h : \mathbb{R}^k \mapsto \mathbb{R}^k$ and aim to find a subset $z_1 \in \mathbb{R}^{k_1}$ of z and a subset $h_1 \in \mathbb{R}^{k_1}$ of h such that the remaining variables $z_2 \in \mathbb{R}^{k-k_1}$ can be calculated using the remaining equations $h_2 \in \mathbb{R}^{k-k_1}$ as a function of z_1, i.e,

$$\begin{aligned} z_2 &= h_2(z_1) \\ 0 &= h_1(z_1, z_2) \end{aligned}$$

notice that we can just solve $h_1(z_1, h_2(z_1)) = 0$ and then explicitly plug in for z_2. The variable z_1 that we need to solve for is called iteration variables, the function h_1 that we need to solve over is called the residual equations. The variables z_2 that gets solved for when knowing z_1 are torn away from the system and therefore called tearing variables.

The tearing algorithm has consequently changed the problem of solving

$$h(z) = 0$$

which is a system of equations of dimension k to solving the system

$$h_1(z_1, h_2(z_1)) = 0$$

which is a system of equations of dimension k_1 and then declaring the remaining variables z_2 by

$$z_2 = h_2(z_1)$$

.We will in Section 4 see that the tearing algorithm reduces the dimension of the system of equations that needs to be solved when inlining the spring pendulum using implicit Euler's method.

3 Mixed mode on ODEs

Consider the ODE

$$\dot{x} = f(x,t) \in \mathbb{R}^N, \quad x(t_0) = x_0 \tag{11}$$

To solve such an equation numerically we can apply either an explicit or an implicit method. The simplest explicit method is the explicit Euler's method

$$x_{n+1} = x_n + h \cdot f(x_n, t_n) \tag{12}$$

and the simplest implicit method is the implicit Euler's method.

$$x_{n+1} = x_n + h \cdot f(x_{n+1}, t_{n+1}) \tag{13}$$

The main implementation difference between an explicit and implicit method is that, using an explicit method we can directly calculate x_{n+1} from x_n and using an implicit method we need to solve a system of equations to get the value of x_{n+1}. Solving a system of equations can be expensive especially when N becomes large. One might thereforee choose to just use explicit solvers. However some special problems would require for explicit methods the step size to be very small. Such special problems are called stiff and one usually uses implicit methods to solve these.

Making a mathematical definition of stiffness has proven difficult, but many statements about the phenomena have been made. One example is, "stiffness occur when stability requirement rather then accuracy constrains the step length". Such systems are dynamically very fast or have highly damped components .

Systems might contain only few very fast or highly damped components, this leads to the unsatisfactory situation of having to use either very small step sizes with an explicit method or, to use an implicit method and, to solve a system of equations [9].

For the sake of argument let us say that we use an implicit method, this means that we need to solve a system of equations of size N even though the system only contain, $k_2 << N$ very fast or highly damped components. To reduce the work carried out by the numerical solver we suggest doing

a row-wise partitioning of the left-hand side into "fast" and "slow" state-variables. Let us denote the "fast" variables by x^F and the "slow" by x^S. We then split the system into

$$\dot{x}^S = f^S(x^S, x^F, t) \in \mathbb{R}^{k_1} \tag{14}$$
$$\dot{x}^F = f^F(x^S, x^F, t) \in \mathbb{R}^{k_2} \tag{15}$$

We apply explicit Euler's method on (14) to get

$$x^S_{n+1} = x^S_n + h \cdot f^S(x^S_n, x^F_n, t_{n+1}) \tag{16}$$

The information about x^S_{n+1} is then used when implicit Euler's method is used on (15)

$$x^F_{n+1} = x^F_n + h \cdot f^F(x^S_{n+1}, x^F_{n+1}, t_{n+1}) \tag{17}$$

To update (17) we only need to solve a system of equations of dimension k_2 instead of N.

3.1 Stability of a numerical method

Absolute stability of a numerical method is an important concept in dealing with stiff ODEs. Absolute stability is concerned with stability as $t \to \infty$ for a fixed step-size h. Let us look at the initial value problem.

$$\dot{x}(t) = \lambda x(t), \quad x(0) = x_0, \quad \lambda \in \mathbb{C} \tag{18}$$

Equation (18) with $\mathbf{Re}(\lambda) < 0$ is called the test equation and is used to study the stability of numerical methods. An *absolute stable* numerical method is one for which the numerical solution of the test equation also exhibits the same controlled decay as the exact solution.
For example explicit Euler's method gives

$$x_{n+1} = x_n + h \cdot x_n = (1 + h\lambda) \cdot x_n = (1 + h\lambda)^n x_0 \tag{19}$$

Since the test equation for $\mathbf{Re}(\lambda) < 0$ has a decaying solution we require the numerical method to exhibit the same behavior. This means that the condition $\mid 1 + h\lambda \mid < 1$ must be fulfilled, hence the step size must satisfy $\mid h \mid < \frac{-2}{\lambda}$. When the numerical method has such a condition on the step size

it is called *conditionally stable*. If we also study the behavior of implicit Euler's method on (18) we find that

$$x_{n+1} = x_n + h \cdot x_{n+1} = \frac{1}{(1-h\lambda)} \cdot x_n = \frac{1}{(1-h\lambda)^n} x_0 \tag{20}$$

The requirement for this method is that it also must exhibit the same decreasing behavior as the solution as $t \to \infty$, which holds for any $h > 0$. Such methods are called *unconditionally stable*. Unconditionally stable solvers are usually all that is needed for stiff problems. Such methods are also called A-stable.

The notion of controlled decay of the numerical method is what motivates the suggested partitioning of a linear ODE

$$\dot{x} = Ax \in \mathbb{R}^n \tag{21}$$

We note that the solution of such a system is asymtotically stable if and only if the eigenvalues of A fullfill $\mathbf{Re}(\lambda) < 0$. The idea behind doing a partitioning on a linear ODE comes from that the mixed numerical method must exhibit the same behavior as $t \to \infty$ as the solution of (21) with $\mathbf{Re}(\lambda) < 0$.

3.1.1 Partitioning a linear ODE

The partitioning can be done manually by the user or we can automatically generate a set of state variables that could be considered as "slow". To see how an automatic partitioning of (22) can be done we partition the linear ODE.

$$\dot{x} = Ax \in \mathbb{R}^n \tag{22}$$

We would like to do a row-wise partitioning of (22) into "fast" and "slow" variables, to select rows we multiply A by a diagonal projection matrix, $P = \text{diag}(\delta_1, ..., \delta_n)$ with $\delta_i \in \{0, 1\}$. To select row i as "slow" we set $\delta_i = 1$.

$$\dot{x}^S = PAx \tag{23}$$

The remaining rows are selected as "fast" by multiplying A from the left by $(I - P)$, since

$$A = PA + (I - P)A \tag{24}$$

then we get the system

$$\dot{x}^S = PAx \tag{25}$$
$$\dot{x}^F = (I - P)Ax \tag{26}$$

or

$$\dot{x} = PAx + (I - P)Ax \tag{27}$$

Now perform explicit Euler on (25) and implicit Euler on (26). This type of method is a mix of an implicit Runge-Kutta method and an explicit Runge-Kutta method, these methods are suitable for problems containing stiff and non stiff terms [8].

$$x^S_{n+1} = Px_n + h \cdot APx_n \tag{28}$$
$$x^F_{n+1} = (I - P)x_n + h \cdot (I - P)Ax_{n+1} \tag{29}$$

or

$$x_{n+1} = x_n + h \cdot PAx_n + h \cdot (I - P)Ax_{n+1} \tag{30}$$

We can then solve for x_{n+1}

$$x_{n+1} = (I - h \cdot (I - P)A)^{-1}(I + h \cdot PA)x_n \tag{31}$$

lets call $I - h \cdot (I - P)A)^{-1}(I + h \cdot PA) = R(h, P)$ then

$$x_{n+1} = R(h, P)x_n \tag{32}$$

Hence, to ensure stability for the given step size we require that the eigenvalues in absolute value of $R(h, P)$ are smaller then 1. This means that the numerical method exhibits the same asymptotic stability as the exact solution.

Before a simulation is started a test should be preformed, the test consists of choosing δ_i. All δ_i start as 0, if $\delta_i = 0$, then the i'th component of x is considered as "fast". Then in order we set $\delta_i = 1$ and calculate the eigenvalues of the iteration matrix $R(h, P)$. If the eigenvalues of $R(h, P)$ are in absolute value smaller then 1 then δ_i is kept as 1 otherwise it is set to 0. If δ_i is kept as 1 then the i'th component of x is considered as "slow". This is done for all i.

3.1.2 Partitioning a non - linear ODE

For a linear ODE with $\mathbf{Re}(\lambda) < 0$

$$\dot{x} = Ax \tag{33}$$

we know how to do a partition into "fast" and "slow" state variables such that the mixed method exhibits the same decay as the exact solution. In the case of a non linear ODE

$$\dot{x} = f(x, t) \tag{34}$$

the idea is to make a linear approximation of f along its trajectory. At the points where the linear approximation satisfies the condition with $\mathbf{Re}(\lambda) < 0$ we preform the suggested partitioning for the linear problem. This would result in a set of "slow" and a set of "fast" state variables. The union of all state variables that were ever considered as "fast" is chosen as fast, the rest is chosen as "slow". In practice, since we are trying to help a user that is performing a real-time simulation of their dynamical system, we assume that the user has enough information about the system such that the user can provide a set of points along the solutions trajectory.

Below we perform a simulation of a spring pendulum using a formulation and spring constant such that we before hand have a idea of which variables will be "fast" and which variables will be "slow".

3.2 Example spring pendulum and a linear ODE

To get a better understanding of mixed mode integration we test it on the spring pendulum. The partitioning is tried on a linear ODE that satisfies the assumption that $\mathbf{Re}(\lambda) < 0$.

A formulation of the spring pendulum is chosen such that we before hand have an idea of which state variables are "slow" and which are "fast". The purpose of the example is to investigate the suggested methods strengths and weaknesses.

Let $\dot{x}, \dot{y}$ denote the velocity and acceleration with which the length of the pendulum changes, and let $\dot{\theta}, \dot{\phi}$ be the angular velocity and acceleration respectively. The constants are m, k, l_0 and g, where m is the mass of the object attached to the spring, k is the stiffens coefficient of the spring, l_0

is the length of the pendulum in equilibrium and $g = 9.81$ is the gravity constant. The equations of the spring pendulum are given by

$$\dot{x} = y \tag{35}$$

$$\dot{y} = (l_0 + x) \cdot \phi^2 - \frac{k}{m} \cdot x + g \cdot \cos(\theta) \tag{36}$$

$$\dot{\theta} = \phi \tag{37}$$

$$\dot{\phi} = \frac{-g}{l_0 + x} \cdot \sin(\theta) - \frac{2 \cdot y}{l_0 + x} \cdot \phi \tag{38}$$

We initialize the system with values such that we have "fast" components present. Let the constants be $m = 3$, $l_0 = 1$, $k = 35$ with initial conditions $X_0 = [\ 1.5\ ,\ 0\ ,\ \frac{\pi}{4},\ 0\ ]^T$. Since $l_0 = 1$, the initial condition of $x_0 = 1.5$ means that the spring is stretched out to 1.5 times its resting length. The high k means that the spring is stiff.

Below we preform simulations of the spring pendulum for $t \in$ [0,20] using mixed mode integration with $\dot{x}, \dot{y}$ as "fast" and $\dot{\theta}, \dot{\phi}$ as "slow" for different step-sizes and compare the error with explicit Euler's method and implicit Euler's method where the solution generated using CVode with step-size $h = 10^{-6}$ and rtol $= 10^{-9}$ as solver is considered the exact solution.

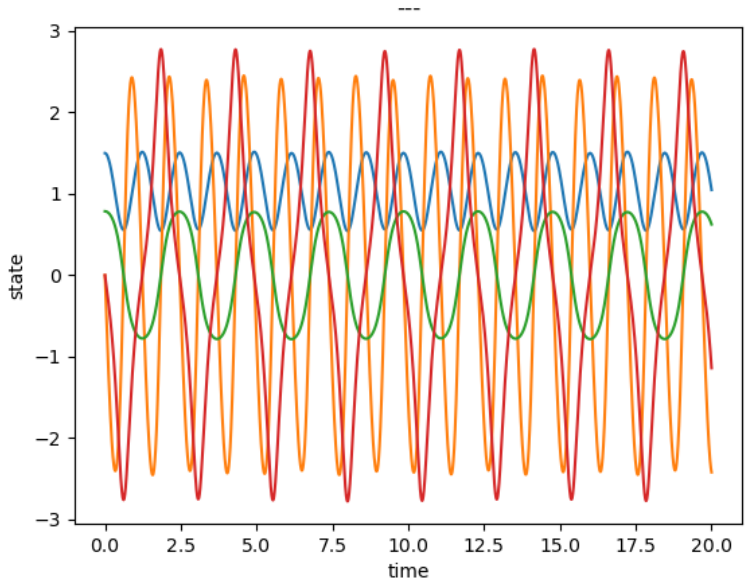

Figure 1: Solution using CVode, $h = 10^{-6}$, rtol $= 10^{-9}$.

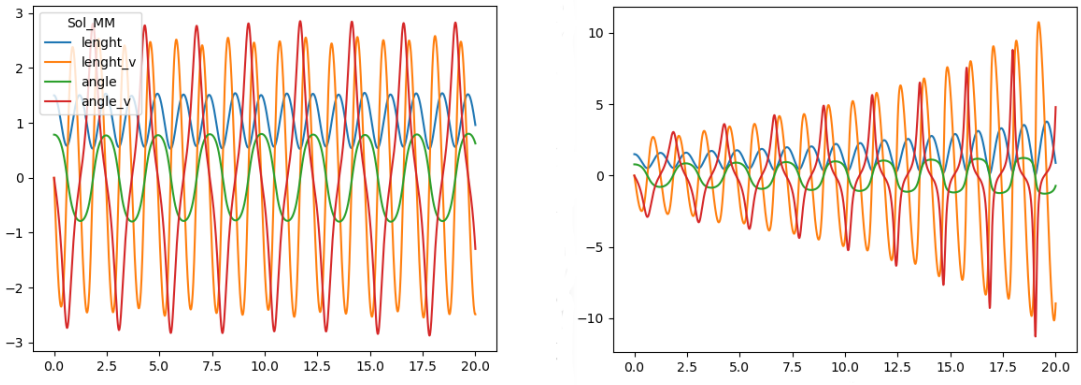

Figure 2: Solution for mixed mode to the left and explicit Euler's method to the right, $h = 10^{-2}$.

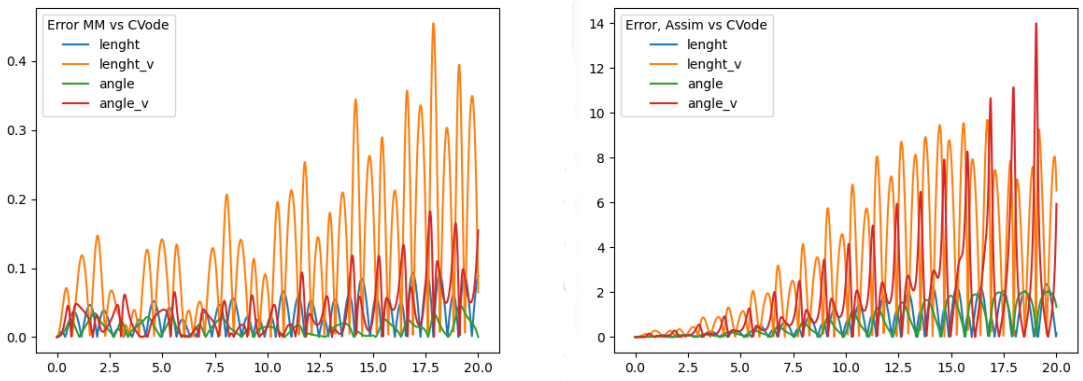

Figure 3: Error for mixed mode to the left and explicit Euler's method to the right, $h = 10^{-2}$.

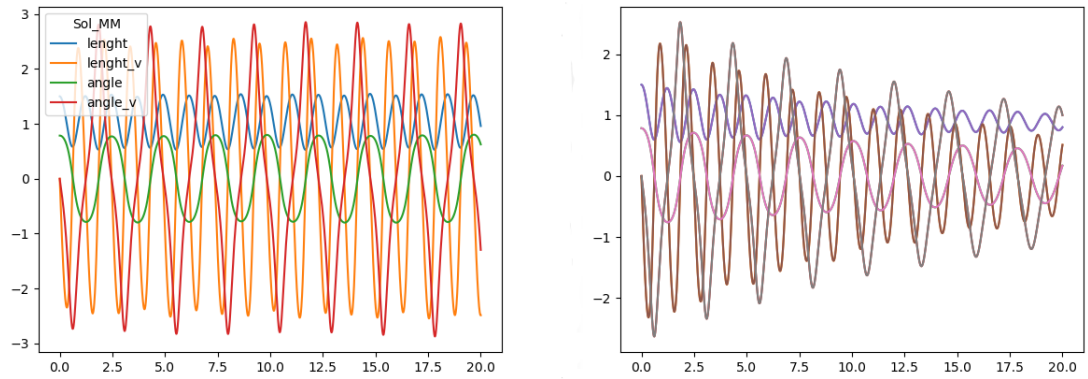

Figure 4: Solution for mixed mode to the left and implicit Euler's method to the right, $h = 10^{-2}$.

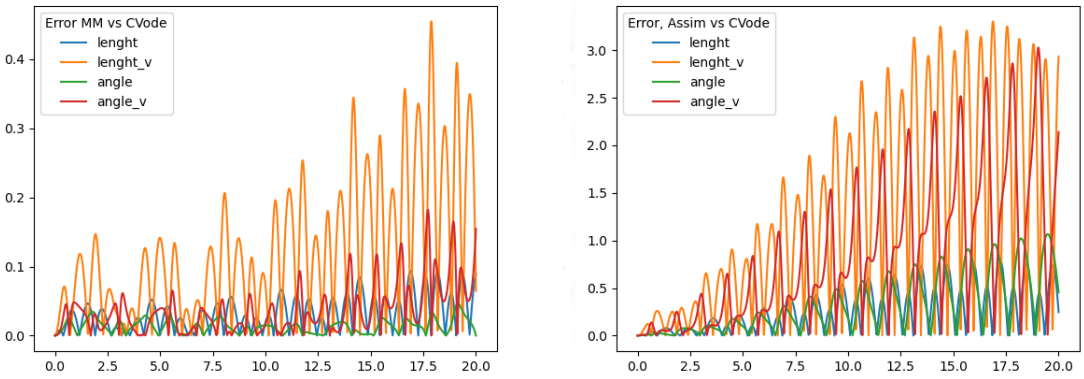

Figure 5: Error for mixed mode to the left and implicit Euler's method to the right, $h = 10^{-2}$.

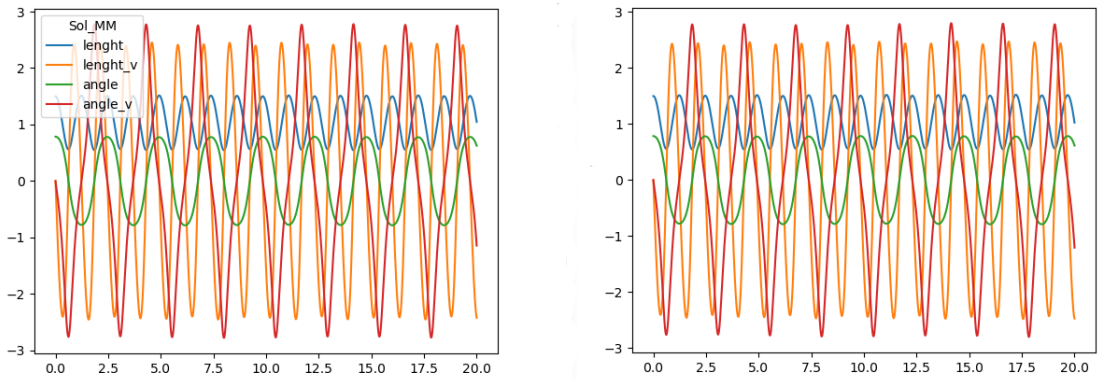

Figure 6: Solution for mixed mode to the left and explicit Euler's method to the right, $h = 10^{-4}$.

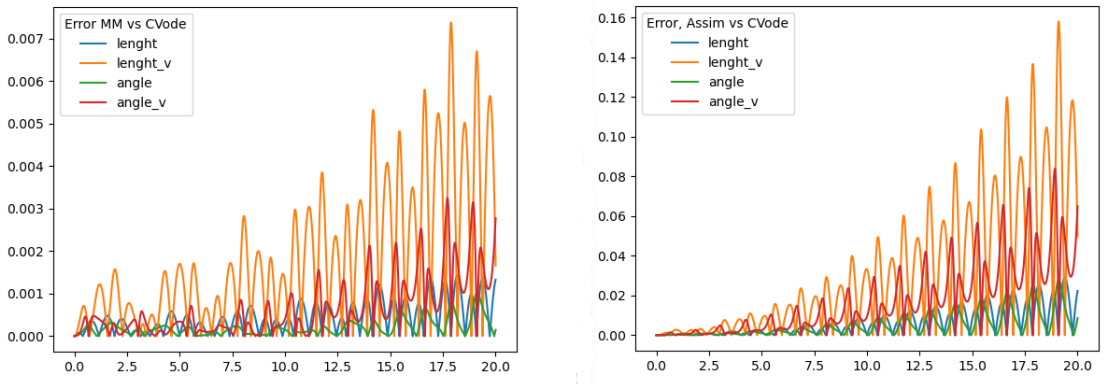

Figure 7: Error for mixed mode to the left and explicit Euler's method to the right, $h = 10^{-4}$.

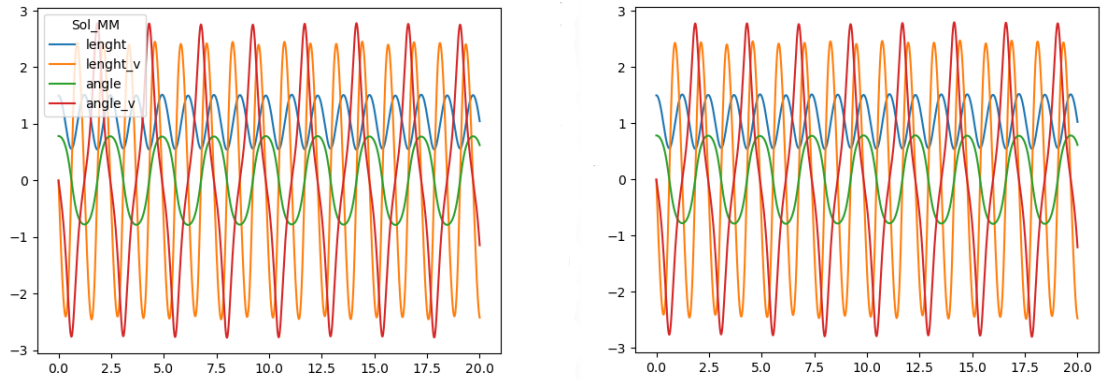

Figure 8: Solution for mixed mode to the left and implicit Euler's method to the right, $h = 10^{-4}$.

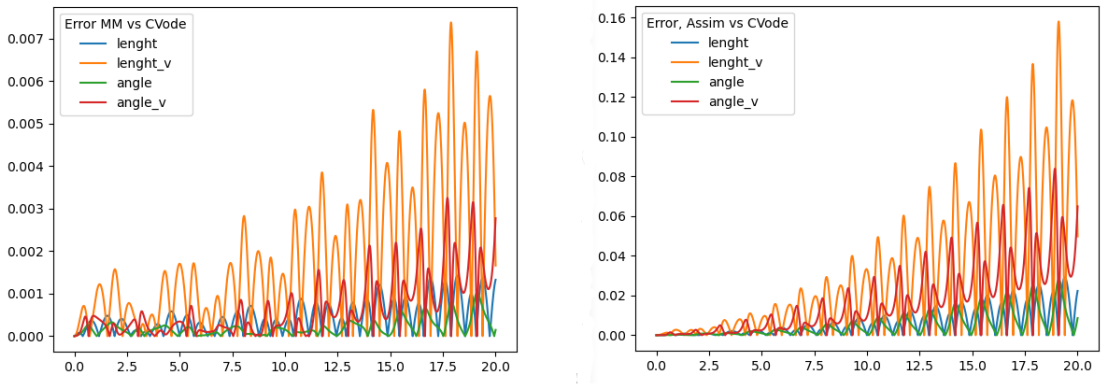

Figure 9: Error for mixed mode to the left and implicit Euler's method to the right, $h = 10^{-4}$.

The above shows the unexpected result of mixed mode integration actually out-preforming explicit and implicit Euler's method in terms of accuracy. The following code should give the reader the important information about what happens in every mixed mode step. Two things should be noticed, y_{n+1}^S might act like a predictor and the result can be a consequence of different nonlinear solvers. In mixed mode we used Scipy's [6] fsolve as a non-linear solver, this could outperform Assimulos [1] non-linear solver. A short argument below is given on why the results are not a consequence of the two facts described above.

```
def Explicit_step( self , y_S_n , y_n ):

    y_S_np1 = y_S_n + self.h * self.f_S(y_n)
    return y_S_np1

def Implicit_step( self , y_S_np1 , y_F_n ):
    def g( y_F ):
        "Collect makes sure f get the variable in correct order."
        f_F = lambda y_F: f( 1 , self.Collect( y_S_np1 , y_F ) )
        "We need to evaluate the whole f and pick out the" "fast" "parts."
        yf = np.array( [ f_F(y_F)[ i ] for i in range( len( self.E ) ) \
        if self.E[ i ] == 0 ] )

        return - y_F + y_F_n + self.h * yf
    y_F_np1 = optimize.fsolve( g , y_F_n )

    return y_F_np1
```

Scipy's optimizing function fsolve uses xtol = 1.49012e-08 as default. This means that a zero to the function g is assumed to be found if

$$| x_{n+1} - x_n | < 1.49012e - 08$$

. In Assimulo's implicit Euler a relative tolerance of 1e-08 was given to the Newton solver together with a high number of allowed Newton updates, this was done to prevent the non-linear solver in mixed mode to outperform the one used in Assimulo. These precautions prevent the results being a function of the non-linear solver.

As for y_n^S acts like a predictor, we checked using a predictor/corrector code with explicit Euler's method as a predictor for all variables and then implicit Euler's method as a corrector and for the above example the mixed mode integration code outperformed this code also with almost similar results. In the simulations above we used the fact that we beforehand had an idea of which variables to consider as "slow" and "fast".
Next we take a look at the following IVP to investigate the suggested partitioning. Let

$$m \cdot \ddot{x} + c \cdot \dot{x} + k \cdot x = 0, \quad x(0) = 1, \;\; \dot{x}(0) = 0 \tag{39}$$

and lets set $m = 1$, $c = 1001$ and $k = 1000$. and lets rewrite this to form

(18). Let $x = \dot{y}$ and $\dot{y} = -1001 \cdot y - 1000 \cdot x$. This gives the system

$$\dot{z} = Az \tag{40}$$

with $z = (x, y)^T$ and $A = \begin{pmatrix} 0 & 1 \\ -1001 & -1000 \end{pmatrix}$. The matrix A has eigenvalues $\lambda_1 = -1000$ and $\lambda_2 = -1$, that is, they both have negative real part. We can therefore preform the suggested partition of this. Notice that Explicit Euler' method on (40) is unstable for a step-size larger than 10^{-3}. The idea is therefore to use the partitioning suggested in section 3.1.1 to find a mixed method that is stable for a step-size larger then 10^{-3}.

Below we present the partitioning code for (40), and investigate which variables gets put as "fast" and which are kept as "slow" using different step-sizes. Then mixed mode is applied to (40) using the suggested partitioning and the results is compared to explicit Euler's method and implicit Euler's method in the same way as we did with the spring pendulum.

```
" The function make E starts as a list of 1's, if variable i is" "fast"
"element i in E is changed to 0"
def Make_E( A ):
    E = [ 1 for i in range( 2 ) ]
    Set  = [ i for i in range( 2 ) ]
    for i in range( 2 ) :
            R = Make_R( E , A )
            S_1 = np.linalg.eigvals( R )
            if all( abs( S_1 ) < 1 ):
                pass
            else:
                for k in Set:
                    if abs( S_1[ k ] ) > 1:
                        E[ k ] = 0
                        Set.remove( k )
    return E
```

We preform a partitioning using $h_1 = 10^{-1}, h_2 = 10^{-2}, h_3 = 10^{-3}$. For h_1 and h_2 the function gives us the set E = [1,0] and for h_3 we generate the set [1,1].

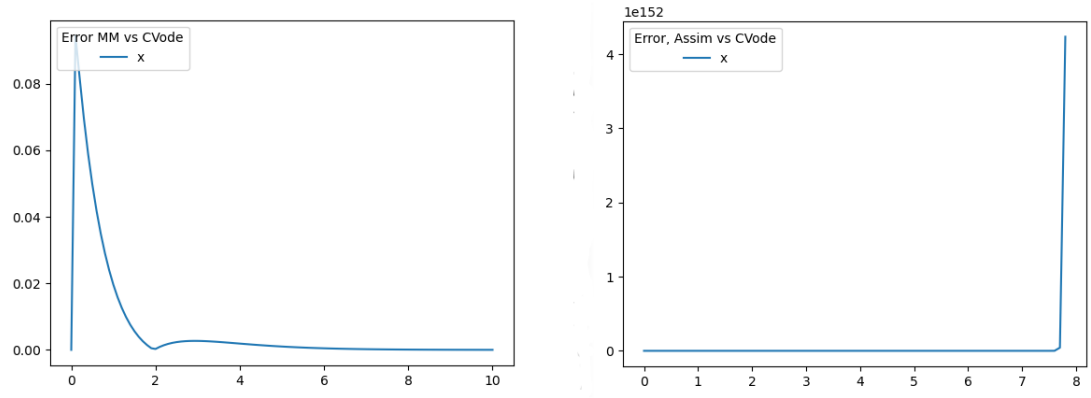

Figure 10: Error for mixed mode to the left and implicit Euler's method to the right, $h = 10^{-1}$.

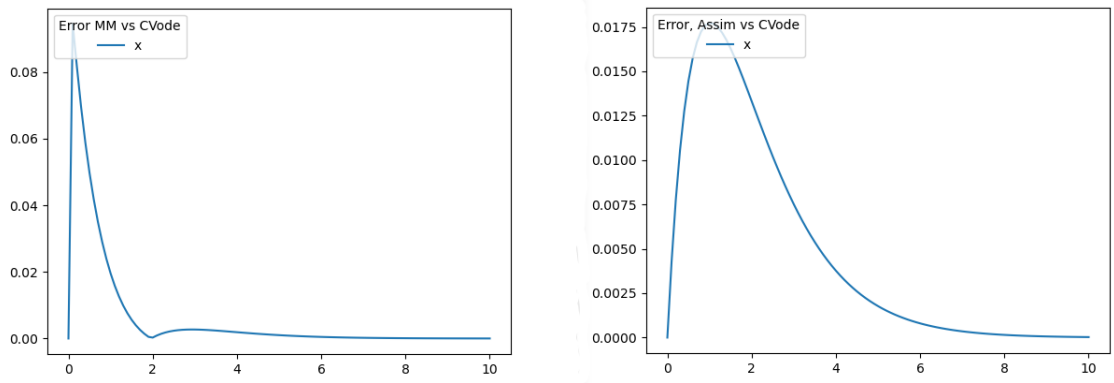

Figure 11: Error for mixed mode to the left and implicit Euler's method to the right, $h = 10^{-1}$.

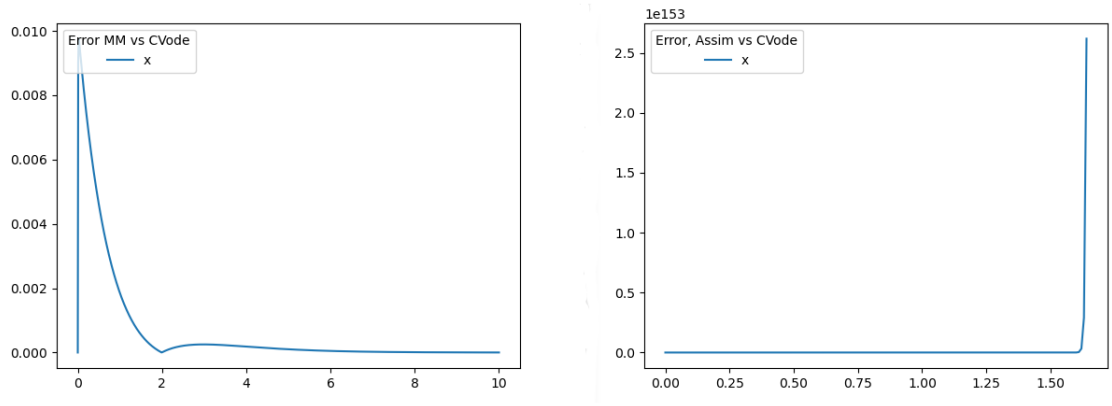

Figure 12: Error for mixed mode to the left and explicit Euler's method to the right, $h = 10^{-2}$.

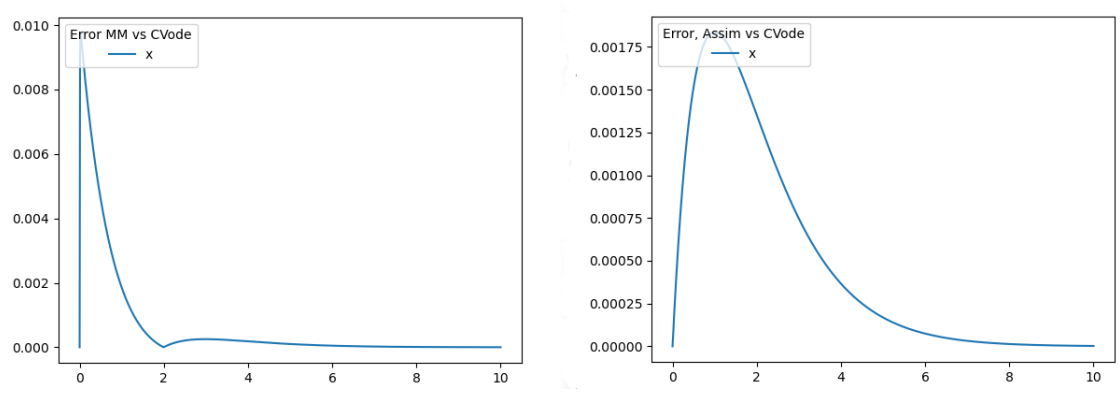

Figure 13: Error for mixed mode to the left and implicit Euler's method to the right, $h = 10^{-2}$.

3.3 Remarks on the suggested method

When coding the explicit Euler step and implicit Euler step we noticed the difference between the theoretical idea of going from

$$\dot{x} = f(x,t) \in \mathbb{R}^N \tag{41}$$

to

$$\dot{x}^S = f^S(x^S, x^F, t) \in \mathbb{R}^{k_1} \tag{42}$$

$$\dot{x}^F = f^F(x^S, x^F, t) \in \mathbb{R}^{k_2} \tag{43}$$

and coding a mixed mode step. The unsatisfactory difference is that when a user gives an arbitrary function $f(x, t)$ we cannot just evaluate parts of it, but we need to evaluate the whole function and pick out the "slow" and "fast" parts of the function. This makes a step very slow. A mixed mode step using inline integration will however work differently.

4 Mixed mode and inline integration

To investigate how inline integration can benefit from mixed mode integration we need a better understanding of how inline integration works using an explicit or implicit numerical solver. We will investigate this using two examples.

4.1 Two test models

Consider the following equation

$$\dot{x} = \sin(x) \tag{44}$$

and let us investigate this using inline integration with both explicit Euler's method and implicit Euler's method.

Modelon's inline integration tool generates the set of equations that needs to be solved together with a block lower triangular (BLT) matrix to update the solution. By comparing the structure of these we hope to give the reader a better understanding of how inline integration works with explicit and implicit solvers. Since it is not possible to use inline integration with mixed mode integration we will by hand generate the set of equations when inlining the spring pendulum. The speed and acceleration of the length is treated as "fast" and therefore solved using implicit Euler's method while the speed and acceleration of the angle is treated as "slow" and therefore solved using explicit Euler's method. The set of equations is compared with the set of equations when inlining was done using implicit Euler's method.

The set of equations that gets generated for (44) using explicit Euler's method is

$$\dot{\mathbf{x}} = \sin(\mathbf{x}) \tag{45}$$
$$t_0 = \text{old}(t) \tag{46}$$
$$\text{time} = \text{old}(t) + \text{h} \tag{47}$$
$$x_{x0} = \text{old}(\mathbf{x}) \tag{48}$$
$$x_{k0} = \sin(x_{x0}) \tag{49}$$
$$\mathbf{x} = \text{old}(\mathbf{x}) + \text{h} \cdot x_{k0} \tag{50}$$

The generated BLT matrix is

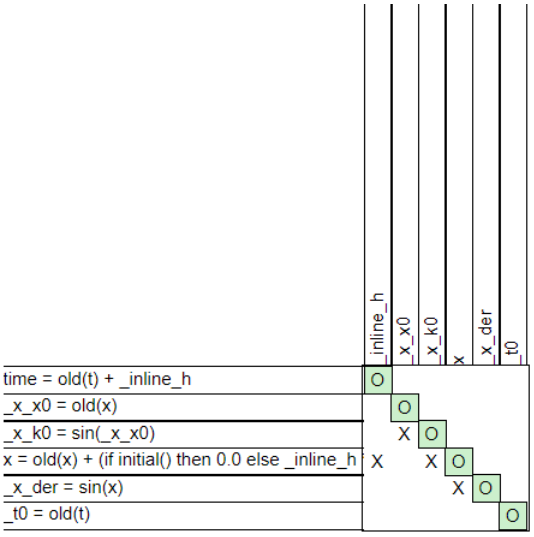

Figure 14: 'o' means that the variable can be solved analytically if the other variables are known, 'x' means that the variable cannot be solved for analytically even if the other variables are known. The green color indicates that the equation is a solved equation.

The set of equations that gets generated for (44) using implicit Euler's method is

$$\dot{\mathbf{x}} = \sin(\mathbf{x}) \tag{51}$$

$$t_0 = \text{old}(t) + \text{h} \tag{52}$$

$$\text{time} = \text{old}(t) + \text{h} \tag{53}$$

$$\mathbf{x} = \text{old}(\mathbf{x}) + \text{h} \cdot \dot{\mathbf{x}} \tag{54}$$

The generated BLT matrix is

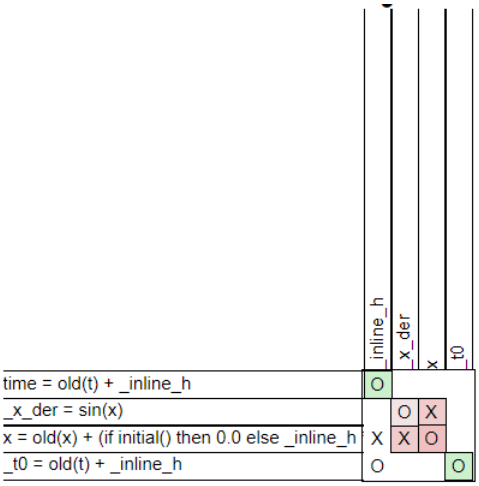

Figure 15: 'o' means that the variable can be solved analytically if the other variables are known, 'x' means that the variable cannot be solved for analytically even if the other variables are known. The green color indicates that the equation is a solved equation. The pink indicates an algebraic loop and the dark pink shows the iteration variables and residual equations of the pink block.

The main difference here is that in the explicit Euler's case all the equations can be solved in a explicit sequence while in the implicit Euler's case there is an algebraic loop. This algebraic loop has 2 unknowns, $\dot{\mathbf{x}}$ and $\mathbf{x}$. This algebraic loop can be torn, using $\mathbf{x}$ as iteration variable. The residual equation needs then to be solved for that variable. This difference leads us to expect that if both explicit and implicit numerical solvers are used we can reduce the dimensions of the algebraic loops.

Let us look at the equation and BLT generated when doing inlining on the equations for the spring pendulum using implicit Euler's method as a numerical solver.

$$\dot{\mathbf{y}} = (l_0 + \mathbf{x}) \cdot \mathbf{z}^2 - \frac{\mathrm{k}}{\mathrm{m}} \cdot \mathbf{x} + g \cdot \cos(\boldsymbol{\theta}) \tag{55}$$

$$\dot{\mathbf{z}} = \frac{-9.81}{l_0 + x} \cdot \sin(\boldsymbol{\theta}) - \frac{2 \cdot \mathbf{y}}{l_0 + \mathbf{x}} \cdot \mathbf{z} \tag{56}$$

$$t_0 = \mathrm{old}(t) + \mathrm{h} \tag{57}$$

$$\mathrm{time} = \mathrm{old}(t) + \mathrm{h} \tag{58}$$

$$\mathbf{x} = \mathrm{old}(\mathbf{x}) + \mathrm{h} \cdot \mathbf{y} \tag{59}$$

$$\mathbf{y} = \mathrm{old}(\mathbf{y}) + \mathrm{h} \cdot \dot{\mathbf{y}} \tag{60}$$

$$\boldsymbol{\theta} = \mathrm{old}(\boldsymbol{\theta}) + \mathrm{h} \cdot \mathbf{z} \tag{61}$$

$$\mathbf{z} = \mathrm{old}(\mathbf{z}) + \mathrm{h} \cdot \dot{\mathbf{z}} \tag{62}$$

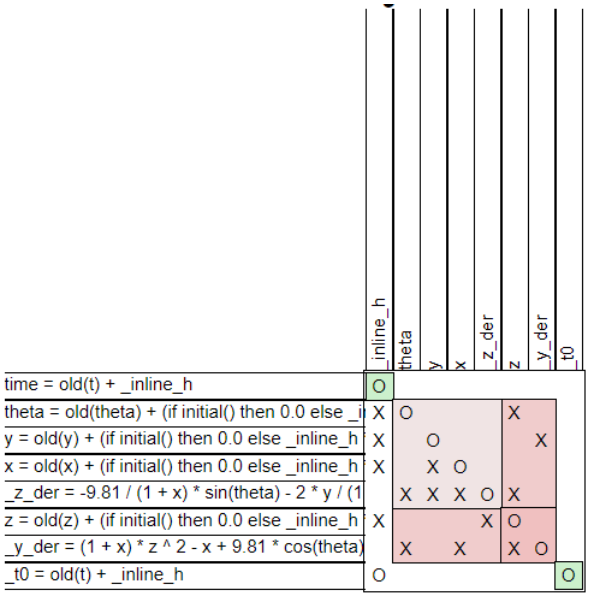

Figure 16: 'o' means that the variable can be solved analytically if the other variables are known, 'x' means that the variable cannot be solved for analytically even if the other variables are known. The green color indicates that the equation is a solved equation. The pink indicates an algebraic loop and the dark pink shows the iteration variables and residual equations of the pink block.

If we look at the BLT matrix that is generated we can see that we have a 6×6 algebraic block indicated by the pink. The dark pink shows that 4

of the unknown variables can be solved if $\mathbf{z}$ and $\dot{\mathbf{y}}$ are known. This means that if we use implicit Euler's method on the spring pendulum we need to solve a system of equations with 2 unknowns.

We stated in the beginning of this section that we would consider $\dot{z}$ and $\dot{\theta}$ as "slow" and therefore apply explicit Euler's method on them. If we would do that we would change (62) to

$$\mathbf{z} = \text{old}(\mathbf{z}) + \text{h} \cdot \dot{\mathbf{z}}(t_n) \tag{63}$$

This would mean that if $\dot{z}$ could be considered a "slow" variable we would reduce the iteration variables to just $\dot{\mathbf{y}}$. We also said that we would consider $\dot{\theta}$ as "slow". This would replace

$$\boldsymbol{\theta} = \text{old}(\boldsymbol{\theta}) + h \cdot \mathbf{z} \tag{64}$$

by

$$\boldsymbol{\theta} = \text{old}(\boldsymbol{\theta}) + h \cdot \mathbf{z}(t_n) \tag{65}$$

Since using $\dot{z}$ as "slow" $\mathbf{z}$ is defined by equation (63), hence there is no use of also using $\dot{\theta}$ as "slow", because it would not reduce the dimensions of the system of equations that needs to be solved.

This example suggest to us that we could suggest to the user variables as "slow" which have more or less impact in terms of reducing the algebraic loop.

4.2 Reducing algebraic loops and iteration variables

We would like to reduce the size of the algebraic loops and the iteration variables and residual equations that needs to be solved in order to update the solution from t_n to t_{n+1} while preserving accuracy.

In the algebraic loop we have variables of $\{\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}\}$. This means that either of those variables can be chosen by the tearing algorithm as iteration variables. The tool at our disposal is that we can declare a variable as "slow". We want to know if there exists a set of variables that we could suggest the user to declare as "slow" so that this set would guarantee a significantly smaller set of iteration variables. To get an idea of how we can suggest such a set we present a proposition in Chapter 5.

The aim is breaking an algebraic loop by introducing "slow" variables while preserving accuracy. However, reducing the size of the algebraic loop also result in a reduced number of equations that needs to be solved.

5 Graph theory

In this section we talk about the augmented DAE from the perspective of graph theory. We also look closer at an idea for reducing the algebraic loops, but first we need to introduce some graph theoretical concepts.

Definition 1. A graph is a ordered triple $G = (N, E, \Psi)$ with a set of nodes N and a set of edges E disjoint from N and a incidence function Ψ that associates with each edge a unordered pair of nodes of the graph. If $\Psi(e) = uv$ then e joins u, v.

Definition 2. Consider the graph G. If for every $u, v \in N(G)$ there exists paths $p_1 : v \Rightarrow u$ and $p_2 : u \Rightarrow v$ we say the G is *strongly connected*.

Definition 3. Consider the graph G and let $\widetilde{G}$ be a sub-graph of G.
If $\widetilde{G}$ is *strongly connected* then $\widetilde{G}$ is said to be a *strongly connected component* (SCC) of G.

The strongly connected components is a representation of the algebraic loops. It is our goal to reduce or break the strongly connected components since this would reduce the dimensions for the systems of equations that needs be solved.

5.1 The structural matrix as a graph

Recall the augmented DAE

$$F_\Delta(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t) = \begin{bmatrix} F(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t) \\ -\mathbf{x} + h \cdot \dot{\mathbf{x}} + old(\mathbf{x}) \end{bmatrix} = 0 \tag{66}$$

In section 2.2 we mentioned that the augmented system

$$F_\Delta(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t) = 0$$

is considered as a system of equations on which we preform a partitioning on its structural matrix. The elements of the structural matrix is either 0 or 1 with the i'th row representing the i'th equation and the j'th column representing the j'th variable, we recall that if variable j appear in equation

i element $S_{ji} = 1$ otherwise it is 0.

The structural matrix is a representation of a graph, the variables and equations are nodes and if $S_{ji} = 1$ there is an edge between the node representing equation j and the node representing variable i.

The important thing to notice is that if a variable that is in an equation is removed we remove the edge between that variable and the equation.

5.2 Breaking strongly connected components

We recall that the variables appearing as nodes are $\{\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}\}$ and as discussed in section 2.1 we use the notation that $\dot{\mathbf{x}} = \dot{x}_{n+1}$, $\mathbf{x} = x_{n+1}$ and $\mathbf{w} = w_{n+1}$. If a variable is in an equation there is an edge between the variable and the equation, but if we would allow the previous value to be used we would actually remove that edge.

Using the previous value for a variable in an equation in (66) will henceforth be referred to as *a delay of information*.

For example, when we extend the DAE with the implicit Euler's method we add equations

$$\mathbf{x} = h \cdot \mathbf{x}(t_n) + \dot{\mathbf{x}} \tag{67}$$

and when we say that we allow for a delay of information for $\dot{\mathbf{x}}$ in such a equation we replace $\dot{\mathbf{x}} = \dot{x}_{n+1}$ by $\dot{x}_n$. This is the explicit Euler's method.

Let us represent the algebraic loop for the spring pendulum generated by Modelons inlining tool as a graph when using the implicit Euler's method as the numerical solver.

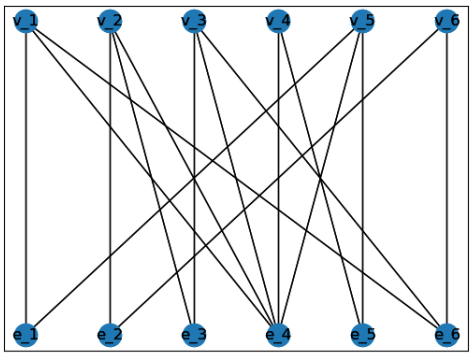

Figure 17: v_i correspond to the i'th variable and e_i correspond to the i'th equation.

We suggest that we could break the algebraic loop by allowing for a delay of information in some variables. The variables that we suggest is z and y in equation 5 and 2 respectively. The reason for choosing these variables was disucssed in section 3 and 4. In terms of the graph doing so would remove edges in the graph, namely the edge (v_4, e_5) and (v_6, e_2), this gives us the new graph.

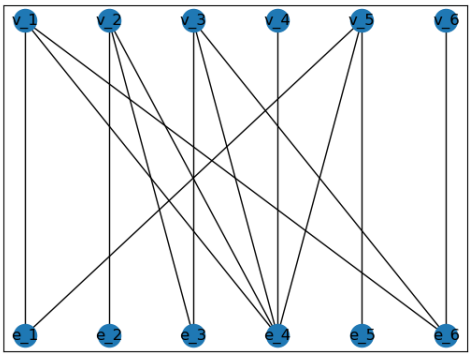

Figure 18: v_i correspond to the i'th variable and e_i correspond to the i'th equation.

Notice that in equation 2 and equation 5 there is only 1 variable respectively, this means that these equations define those variables, hence the node e_2, e_5, v_2 and v_5 is removed, together with the edges coming from those nodes. This gives us the new graph.

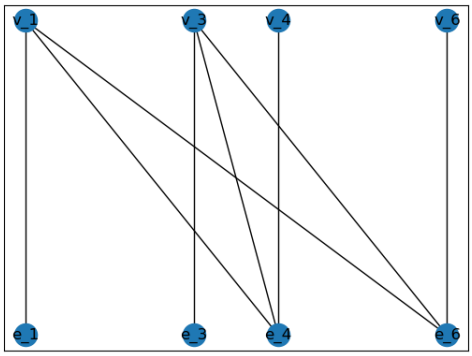

Figure 19: v_i correspond to the i'th variable and e_i correspond to the i'th equation.

Notice that there is only variable v_1 in equation 1 and only variable v_3 in equation 3, this means that that those equations defines those variables, hence the nodes e_1, e_3, v_1 and v_3 is removed, together with the edges coming from those nodes. This gives the new graph.

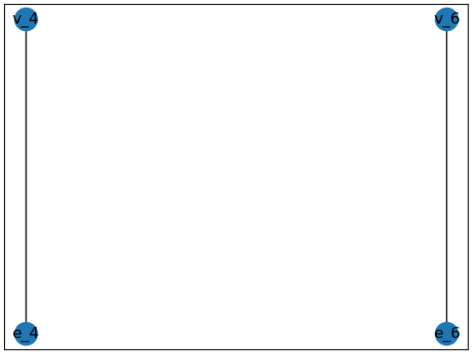

Figure 20: v_i correspond to the i'th variable and e_i correspond to the i'th equation.

Notice that equation 4 only contains variable v_4 hence this is now defined. We can therefore remove node v_4. The only variable appearing in equation 6 is variable v_6 therefore the algebraic loop is broken.

In this example we have used a very simple fact to check what changes in the algebraic loop if an edge is removed. If an equation only contains one

variable, then that equation defines that variable. In terms of graph theory we could present it as a proposition

Proposition 1. If a node representing an equation only have one edge, then the node on the other side of the edge is the only variable in that equation. Meaning that the equation defines that variable.
Consequently, the nodes representing the equation and the variable is removed together with all the edges coming from those nodes.

In the above example we have iteratively applied proposition 1 on the graph with the removed edges to see what the consequences would be in the SCC. Important to notice is that we have not mentioned how convergence is affected when the value of a variable at t_{n+1} is changed to the value at t_n. We will try to deal with that later. The code that we used to check proposition 1 is as follows.

```
def Graph_Check(G):
    List_to_remove = []
    for v in G:  ## goes through Nodes in graph
        List = []
        for nbr in G[v]:
            List.append(nbr)
        if len(List) == 1 and ('v' in nbr):
            List_to_remove.append(v)
            if nbr not in List_to_remove:
                List_to_remove.append(nbr)

    return G, List_to_remove


def Remove_Edges(G,N):
    # G is the Graph
    # N is the list of edges that we remove

    for n in N:
        G.remove_edge(n[0],n[1])
    G_new , N_new = Graph_Check(G)

    while len(N_new) != 0:

        G_new , N_new = Graph_Check(G_new)

        for n in N_new:

            G_new.remove_node(n)

    Last_remove = []
    for n in G_new:
        if G_new.adj[n] == {}:
            Last_remove.append(n)
    for n in Last_remove:
        G_new.remove_node(n)

    return G_new
```

The function *Remove edges* takes the original algebraic loop as a graph

G and a set of edges N that we remove from the graph. Then it iterativly checks proposition 1 in the function *Graph check*. Next we want to focus on the fact that going from implicit Euler's method to explicit Euler's method can be seen as a delay of information and that proposition 1 dose not care what edge is removed by introducing such a delay.

5.3 Delay of information

We have seen in the previous section that introducing a delay of information can reduce the size of the algebraic loop. The aim of this section is to motivate why we can allow for a delay of information for variables in certain equations. We also show that a system can be split into smaller subsystems by introducing a delay of information at the right places. The example we use to demonstrate the argued method is a simple electrical circuit.

5.3.1 A simple electrical circuit

We investigate a simple electrical system inlined using implicit Euler's method with the goal of showing that a system can be broken into parts by introducing a delay of information in some equations. Proposition 1 is applied on the graph with the removed edges. The new graph tells us the complexity of the algebraic loop after the edges is removed. The system we have chosen is the electrical circuit shown below.

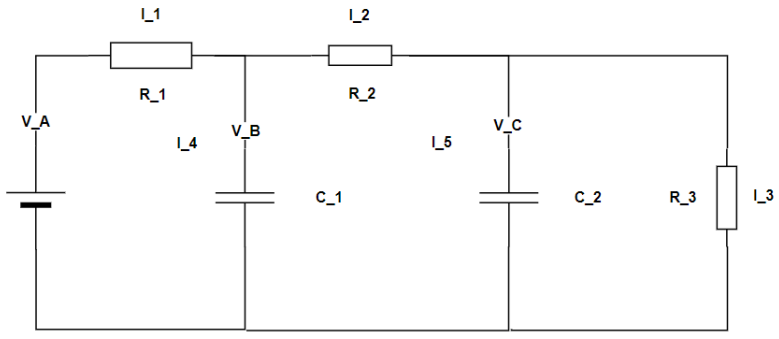

Figure 21: Electrical circuit

This system can be described by the ODE

$$E(t) - V_A(t) = 0 \tag{68}$$
$$V_A(t) - V_B(t) = R_1 \cdot I_1(t) \tag{69}$$
$$C_1 \cdot \dot{V}_B(t) = I_4(t) \tag{70}$$
$$V_B(t) - V_C(t) = R_2 \cdot I_2(t) \tag{71}$$
$$C_2 \cdot \dot{V}_C(t) = I_5(t) \tag{72}$$
$$V_C(t) = R_3 \cdot I_3(t) \tag{73}$$
$$I_1(t) - I_4(t) - I_2(t) = 0 \tag{74}$$
$$I_2(t) - I_5(t) - I_3(t) = 0 \tag{75}$$

E is an outside voltage source. Equations (69), (71) and (73) are for the 3 resistors. Equations (70) and (72) are for the capacitors in the circuit. Equations (74) and (75) are Kirchoff's current conservation law. After extending with the numerical method, which is implicit Euler for the variable appearing in differentiated form in the capacitor equations we get the following BLT matrix.

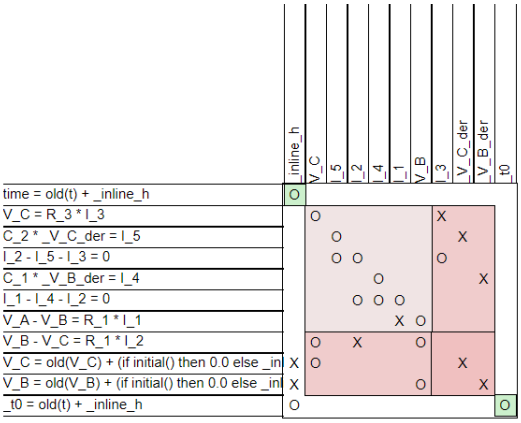

Figure 22: The electrical circuits BLT matrix when inlined using implicit Euler's method.

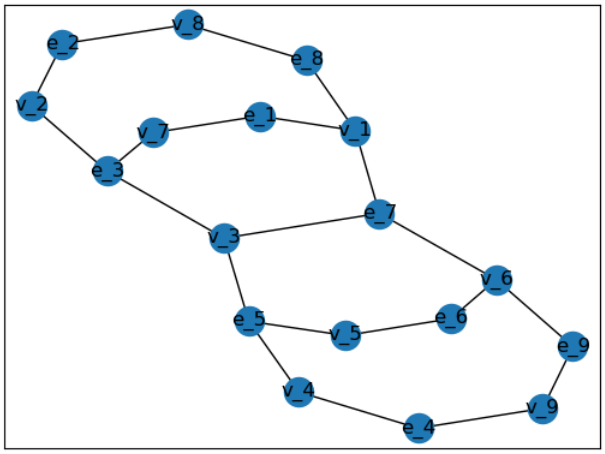

Figure 23: The original algebraic loop is here represented using a graph.

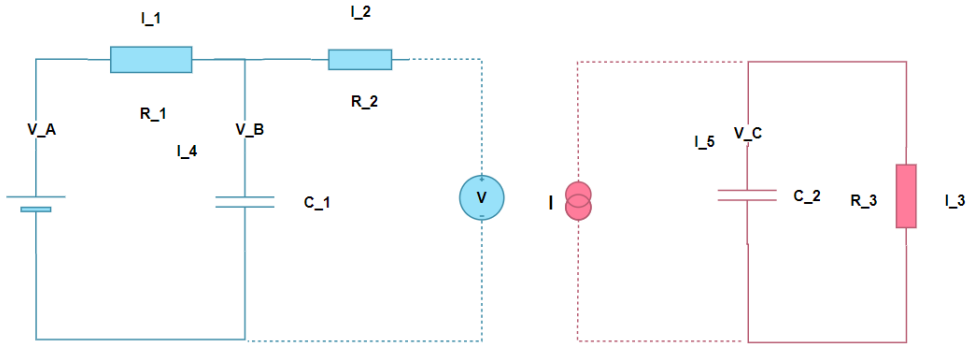

Figure 24: The electrical circuit is split into two subsystems, by connecting the left part with a fictitious voltage generator V and the right part is connected by a fictitious current generator I.

This is done by using the previous value for V_C in Equation (71) and using the previous value for I_2 in equation (75). This removes 2 edges.

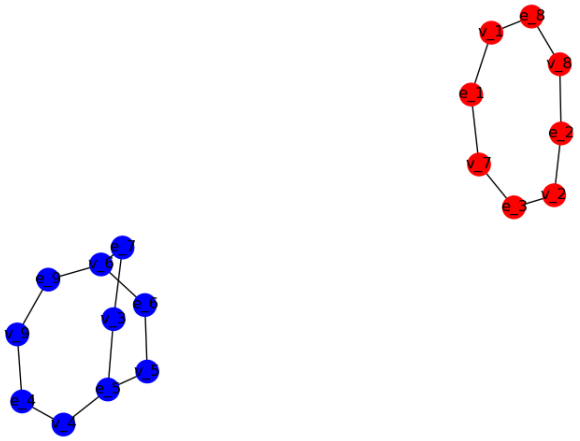

Figure 25: The algebraic loop is split into two parts. The blue loop represents the left part of the electrical circuit and the red part represent the right part of the electrical circuit.

We can see from the graph representing the system that we have managed to split the algebraic loop into two smaller parts.

To simulate this with the notation used in inlining we suggest that we stay at every time-step and update the values that makes the systems talk to each other. To illustrate what we mean by this we write up the simulation using implicit Euler's method.

5.3.2 Simulating the detached system

Let us think of the connected circuit as an autonomous implicit ODE

$$F(\dot{x}, x, w, \omega) = 0$$

We call the algebraic variables in the left system for $w = (I_1, I_2, I_4)$ and the algebraic variables in the right system for $\omega = (I_3, I_5)$. The variable that appear in differentiated form is V_B in the left system and V_C in the right system. They come from the conductors in the respective parts.

The differential equation for the left and right part of the circuit is non autonomous. This is because the behavior of the systems is dependent on a time dependent outside sources that we call $u_L(t)$ and $u_R(t)$.

The left part of the circuit becomes

$$F_1(\dot{V}_B, V_B, w, u_L(t)) = \begin{bmatrix} V_A - V_B - R_1 \cdot I_1 \\ C_1 \cdot \dot{V}_B - I_4 \\ V_B - u_L(t) - R_2 \cdot I_2 \\ I_1 - I_4 - I_2 \end{bmatrix} = 0$$

notice $u_L(t)$ in equation 3, this is the fictitious voltage generator.

The right part of the circuit becomes

$$F_2(\dot{V}_C, V_C, \omega, u_R(t)) = \begin{bmatrix} C_2 \cdot \dot{V}_C - I_5 \\ V_C - R_3 \cdot I_3 \\ u_R(t) - I_5 - I_3 \end{bmatrix} = 0$$

notice $u_R(t)$ in equation 3, this is the fictitious current generator.

We insert implicit Euler's method into our differential equations and extend with the numerical method to get the augmented differential equations. We then add a constraint equation. We now treat the system as a system of equations.

$$F_{\Delta_1}(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{V}_C, \mathbf{w}, \boldsymbol{u_L}) = \begin{bmatrix} F_1(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{w}, \boldsymbol{u_L}) \\ h \cdot \dot{\mathbf{V}}_B + V_B(t_n) - \mathbf{V}_B \\ \boldsymbol{u_L} - \mathbf{V}_C \end{bmatrix} = 0$$

$$F_{\Delta_2}(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{I}_2, \boldsymbol{u_R}) = \begin{bmatrix} F_2(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{u_R}) \\ h \cdot \dot{\mathbf{V}}_C + V_C(t_n) - \mathbf{V}_C \\ \boldsymbol{u_R} - \boldsymbol{I}_2 \end{bmatrix} = 0$$

Let

$$F_\Delta(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, \boldsymbol{\omega}, \boldsymbol{u_L}, \boldsymbol{u_R}) = \begin{bmatrix} F_{\Delta_1}(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{V}_C, \mathbf{w}, \boldsymbol{u_L}) \\ F_{\Delta_2}(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{I}_2, \boldsymbol{u_R}) \end{bmatrix} = 0$$

Using this formulation we hope to give the reader a better understanding of what happens from Newton's perspective when one introduces a delay of information such that the system of equations is split into parts. Let us use Newton's method with $\mathbf{y} = (\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, \boldsymbol{\omega}, \boldsymbol{u_L}, \boldsymbol{u_R})$

$$F'_\Delta(\mathbf{y}^{(i)})\Delta\mathbf{y} = -F_\Delta(\mathbf{y}^{(i)}) \tag{76}$$

$$\mathbf{y}^{(i+1)} = \Delta\mathbf{y} + \mathbf{y}^{(i)} \tag{77}$$

Let us write up the Jacobian. Notice that the first 6 rows are from the left system and the last 5 rows is from the right system. Since the left system do not depend on variables in the right system except at where we want to introduce a delay and vice versa for the right part, we will have this structure.

$$F'_\Delta = \begin{pmatrix}
\frac{\partial f_{11}}{\partial \dot{V}_B} & 0 & \frac{\partial f_{11}}{\partial V_B} & 0 & \frac{\partial f_{11}}{\partial I_1} & \frac{\partial f_{11}}{\partial I_2} & 0 & \frac{\partial f_{11}}{\partial I_4} & 0 & \frac{\partial f_{11}}{\partial u_L} & 0 \\
\frac{\partial f_{12}}{\partial \dot{V}_B} & 0 & \frac{\partial f_{12}}{\partial V_B} & 0 & \frac{\partial f_{12}}{\partial I_1} & \frac{\partial f_{12}}{\partial I_2} & 0 & \frac{\partial f_{12}}{\partial I_4} & 0 & \frac{\partial f_{12}}{\partial u_L} & 0 \\
\frac{\partial f_{13}}{\partial \dot{V}_B} & 0 & \frac{\partial f_{13}}{\partial V_B} & 0 & \frac{\partial f_{13}}{\partial I_1} & \frac{\partial f_{13}}{\partial I_2} & 0 & \frac{\partial f_{13}}{\partial I_4} & 0 & \frac{\partial f_{13}}{\partial u_L} & 0 \\
\frac{\partial f_{14}}{\partial \dot{V}_B} & 0 & \frac{\partial f_{14}}{\partial V_B} & 0 & \frac{\partial f_{14}}{\partial I_1} & \frac{\partial f_{14}}{\partial I_2} & 0 & \frac{\partial f_{14}}{\partial I_4} & 0 & \frac{\partial f_{14}}{\partial u_L} & 0 \\
\frac{\partial f_{15}}{\partial \dot{V}_B} & 0 & \frac{\partial f_{15}}{\partial V_B} & 0 & \frac{\partial f_{15}}{\partial I_1} & \frac{\partial f_{15}}{\partial I_2} & 0 & \frac{\partial f_{15}}{\partial I_4} & 0 & \frac{\partial f_{15}}{\partial u_L} & 0 \\
\frac{\partial f_{16}}{\partial \dot{V}_B} & 0 & \frac{\partial f_{16}}{\partial V_B} & \frac{\partial f_{16}}{\partial V_C} & \frac{\partial f_{16}}{\partial I_1} & \frac{\partial f_{16}}{\partial I_2} & 0 & \frac{\partial f_{16}}{\partial I_4} & 0 & \frac{\partial f_{16}}{\partial u_L} & 0 \\
0 & \frac{\partial f_{21}}{\partial \dot{V}_C} & 0 & \frac{\partial f_{21}}{\partial V_C} & 0 & 0 & \frac{\partial f_{21}}{\partial I_3} & 0 & \frac{\partial f_{21}}{\partial I_5} & 0 & \frac{\partial f_{21}}{\partial u_R} \\
0 & \frac{\partial f_{22}}{\partial \dot{V}_C} & 0 & \frac{\partial f_{22}}{\partial V_C} & 0 & 0 & \frac{\partial f_{22}}{\partial I_3} & 0 & \frac{\partial f_{22}}{\partial I_5} & 0 & \frac{\partial f_{22}}{\partial u_R} \\
0 & \frac{\partial f_{23}}{\partial \dot{V}_C} & 0 & \frac{\partial f_{23}}{\partial V_C} & 0 & 0 & \frac{\partial f_{23}}{\partial I_3} & 0 & \frac{\partial f_{23}}{\partial I_5} & 0 & \frac{\partial f_{23}}{\partial u_R} \\
0 & \frac{\partial f_{24}}{\partial \dot{V}_C} & 0 & \frac{\partial f_{24}}{\partial V_C} & 0 & 0 & \frac{\partial f_{24}}{\partial I_3} & 0 & \frac{\partial f_{24}}{\partial I_5} & 0 & \frac{\partial f_{24}}{\partial u_R} \\
0 & \frac{\partial f_{25}}{\partial \dot{V}_C} & 0 & \frac{\partial f_{25}}{\partial V_C} & 0 & \frac{\partial f_{25}}{\partial I_2} & \frac{\partial f_{25}}{\partial I_3} & 0 & \frac{\partial f_{25}}{\partial I_5} & 0 & \frac{\partial f_{25}}{\partial u_R}
\end{pmatrix}$$

Rearanging the columns we get

$$F'_\Delta = \begin{pmatrix}
\frac{\partial f_{11}}{\partial \dot{V}_B} & \frac{\partial f_{11}}{\partial V_B} & \frac{\partial f_{11}}{\partial I_1} & \frac{\partial f_{11}}{\partial I_2} & \frac{\partial f_{11}}{\partial I_4} & \frac{\partial f_{11}}{\partial u_L} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{12}}{\partial \dot{V}_B} & \frac{\partial f_{12}}{\partial V_B} & \frac{\partial f_{12}}{\partial I_1} & \frac{\partial f_{12}}{\partial I_2} & \frac{\partial f_{12}}{\partial I_4} & \frac{\partial f_{12}}{\partial u_L} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{13}}{\partial \dot{V}_B} & \frac{\partial f_{13}}{\partial V_B} & \frac{\partial f_{13}}{\partial I_1} & \frac{\partial f_{13}}{\partial I_2} & \frac{\partial f_{13}}{\partial I_4} & \frac{\partial f_{13}}{\partial u_L} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{14}}{\partial \dot{V}_B} & \frac{\partial f_{14}}{\partial V_B} & \frac{\partial f_{14}}{\partial I_1} & \frac{\partial f_{14}}{\partial I_2} & \frac{\partial f_{14}}{\partial I_4} & \frac{\partial f_{14}}{\partial u_L} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{15}}{\partial \dot{V}_B} & \frac{\partial f_{15}}{\partial V_B} & \frac{\partial f_{15}}{\partial I_1} & \frac{\partial f_{15}}{\partial I_2} & \frac{\partial f_{15}}{\partial I_4} & \frac{\partial f_{15}}{\partial u_L} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{16}}{\partial \dot{V}_B} & \frac{\partial f_{16}}{\partial V_B} & \frac{\partial f_{16}}{\partial I_1} & \frac{\partial f_{16}}{\partial I_2} & \frac{\partial f_{16}}{\partial I_4} & \frac{\partial f_{16}}{\partial u_L} & 0 & \frac{\partial f_{16}}{\partial V_C} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{21}}{\partial \dot{V}_C} & \frac{\partial f_{21}}{\partial V_C} & \frac{\partial f_{21}}{\partial I_3} & \frac{\partial f_{21}}{\partial I_5} & \frac{\partial f_{21}}{\partial u_R} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{22}}{\partial \dot{V}_C} & \frac{\partial f_{22}}{\partial V_C} & \frac{\partial f_{22}}{\partial I_3} & \frac{\partial f_{22}}{\partial I_5} & \frac{\partial f_{22}}{\partial u_R} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{23}}{\partial \dot{V}_C} & \frac{\partial f_{23}}{\partial V_C} & \frac{\partial f_{23}}{\partial I_3} & \frac{\partial f_{23}}{\partial I_5} & \frac{\partial f_{23}}{\partial u_R} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{24}}{\partial \dot{V}_C} & \frac{\partial f_{24}}{\partial V_C} & \frac{\partial f_{24}}{\partial I_3} & \frac{\partial f_{24}}{\partial I_5} & \frac{\partial f_{24}}{\partial u_R} \\
0 & 0 & 0 & \frac{\partial f_{25}}{\partial I_2} & 0 & 0 & \frac{\partial f_{25}}{\partial \dot{V}_C} & \frac{\partial f_{25}}{\partial V_C} & \frac{\partial f_{25}}{\partial I_3} & \frac{\partial f_{25}}{\partial I_5} & \frac{\partial f_{25}}{\partial u_R}
\end{pmatrix}$$

$$F'_\Delta = \begin{pmatrix}
\frac{\partial f_{11}}{\partial \dot{V}_B} & \frac{\partial f_{11}}{\partial V_B} & \frac{\partial f_{11}}{\partial I_1} & \frac{\partial f_{11}}{\partial I_2} & \frac{\partial f_{11}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{12}}{\partial \dot{V}_B} & \frac{\partial f_{12}}{\partial V_B} & \frac{\partial f_{12}}{\partial I_1} & \frac{\partial f_{12}}{\partial I_2} & \frac{\partial f_{12}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{13}}{\partial \dot{V}_B} & \frac{\partial f_{13}}{\partial V_B} & \frac{\partial f_{13}}{\partial I_1} & \frac{\partial f_{13}}{\partial I_2} & \frac{\partial f_{13}}{\partial I_4} & \frac{\partial f_{13}}{\partial u_L} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{14}}{\partial \dot{V}_B} & \frac{\partial f_{14}}{\partial V_B} & \frac{\partial f_{14}}{\partial I_1} & \frac{\partial f_{14}}{\partial I_2} & \frac{\partial f_{14}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{15}}{\partial \dot{V}_B} & \frac{\partial f_{15}}{\partial V_B} & \frac{\partial f_{15}}{\partial I_1} & \frac{\partial f_{15}}{\partial I_2} & \frac{\partial f_{15}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{\partial f_{16}}{\partial V_C} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{21}}{\partial \dot{V}_C} & \frac{\partial f_{21}}{\partial V_C} & \frac{\partial f_{21}}{\partial I_3} & \frac{\partial f_{21}}{\partial I_5} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{22}}{\partial \dot{V}_C} & \frac{\partial f_{22}}{\partial V_C} & \frac{\partial f_{22}}{\partial I_3} & \frac{\partial f_{22}}{\partial I_5} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{23}}{\partial \dot{V}_C} & \frac{\partial f_{23}}{\partial V_C} & \frac{\partial f_{23}}{\partial I_3} & \frac{\partial f_{23}}{\partial I_5} & \frac{\partial f_{23}}{\partial u_R} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{24}}{\partial \dot{V}_C} & \frac{\partial f_{24}}{\partial V_C} & \frac{\partial f_{24}}{\partial I_3} & \frac{\partial f_{24}}{\partial I_5} & 0 \\
0 & 0 & 0 & \frac{\partial f_{25}}{\partial I_2} & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Notice what changes when we instead introduce a delay in the corresponding constraint equations, i.e,

$$\tilde{F}_{\Delta_1}(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{w}, \boldsymbol{u_L}) = \begin{bmatrix} F_1(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{w}, \boldsymbol{u_L}) \\ h \cdot \dot{\mathbf{V}}_B + V_B(t_n) - \mathbf{V}_B \\ \boldsymbol{u_L} - V_C(t_n) \end{bmatrix} = 0$$

$$\tilde{F}_{\Delta_2}(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{u_R}) = \begin{bmatrix} F_2(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{u_R}) \\ h \cdot \dot{\mathbf{V}}_C + V_C(t_n) - \mathbf{V}_C \\ \boldsymbol{u_R} - I_2(t_n) \end{bmatrix} = 0$$

$$\tilde{F}_\Delta(\boldsymbol{y}) = \begin{bmatrix} \tilde{F}_{\Delta_1}(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{w}, \boldsymbol{u_L}) \\ \tilde{F}_{\Delta_2}(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{u_R}) \end{bmatrix} = 0$$

$$\tilde{F}'_\Delta(\mathbf{y}^{(i)})\Delta\mathbf{y} = -\tilde{F}_\Delta(\mathbf{y}^{(i)}) \tag{78}$$

$$\mathbf{y}^{(i+1)} = \Delta\mathbf{y} + \mathbf{y}^{(i)} \tag{79}$$

the difference in $\tilde{F}'_\Delta$ is that we have forced two elements to be zero. Consequently, all vectors in the blue block are linearly independent to all the vectors in the red block.

$$\tilde{F}'_\Delta = \begin{pmatrix}
\frac{\partial f_{11}}{\partial \dot{V}_B} & \frac{\partial f_{11}}{\partial V_B} & \frac{\partial f_{11}}{\partial I_1} & \frac{\partial f_{11}}{\partial I_2} & \frac{\partial f_{11}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{12}}{\partial \dot{V}_B} & \frac{\partial f_{12}}{\partial V_B} & \frac{\partial f_{12}}{\partial I_1} & \frac{\partial f_{12}}{\partial I_2} & \frac{\partial f_{12}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{13}}{\partial \dot{V}_B} & \frac{\partial f_{13}}{\partial V_B} & \frac{\partial f_{13}}{\partial I_1} & \frac{\partial f_{13}}{\partial I_2} & \frac{\partial f_{13}}{\partial I_4} & \frac{\partial f_{13}}{\partial u_L} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{14}}{\partial \dot{V}_B} & \frac{\partial f_{14}}{\partial V_B} & \frac{\partial f_{14}}{\partial I_1} & \frac{\partial f_{14}}{\partial I_2} & \frac{\partial f_{14}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial f_{15}}{\partial \dot{V}_B} & \frac{\partial f_{15}}{\partial V_B} & \frac{\partial f_{15}}{\partial I_1} & \frac{\partial f_{15}}{\partial I_2} & \frac{\partial f_{15}}{\partial I_4} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{21}}{\partial \dot{V}_C} & \frac{\partial f_{21}}{\partial V_C} & \frac{\partial f_{21}}{\partial I_3} & \frac{\partial f_{21}}{\partial I_5} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{22}}{\partial \dot{V}_C} & \frac{\partial f_{22}}{\partial V_C} & \frac{\partial f_{22}}{\partial I_3} & \frac{\partial f_{22}}{\partial I_5} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{23}}{\partial \dot{V}_C} & \frac{\partial f_{23}}{\partial V_C} & \frac{\partial f_{23}}{\partial I_3} & \frac{\partial f_{23}}{\partial I_5} & \frac{\partial f_{23}}{\partial u_R} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial f_{24}}{\partial \dot{V}_C} & \frac{\partial f_{24}}{\partial V_C} & \frac{\partial f_{24}}{\partial I_3} & \frac{\partial f_{24}}{\partial I_5} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Rewriting the respective Newton iterations in fixed point form we get

$$\varphi(\mathbf{y}) = \mathbf{y} - F'_\Delta(\mathbf{y})^{-1} F_\Delta(\mathbf{y}) \tag{80}$$

$$\tilde{\varphi}(\mathbf{y}) = \mathbf{y} - \tilde{F}'_\Delta(\mathbf{y})^{-1} \tilde{F}_\Delta(\mathbf{y}) \tag{81}$$

Iterations based on φ and $\tilde{\varphi}$ might converge to a fixed point, but not necessarily to the same fixed point. The delay of information that was introduced has an exchange of information between the time steps. Let us change the exchange to an exchange between the Newton iterations.

5.4 On fixed point form

Notice that

$$\tilde{F}'_{\Delta}(\mathbf{y}^{(i)})\Delta\mathbf{y} = -\tilde{F}_{\Delta}(\mathbf{y}^{(i)}) \tag{82}$$

$$\mathbf{y}^{(i+1)} = \Delta\mathbf{y} + \mathbf{y}^{(i)} \tag{83}$$

is the same as

$$\tilde{F}'_{\Delta_1}(\mathbf{y}_1^{(i)})\Delta\mathbf{y}_1 = -\tilde{F}_{\Delta_1}(\mathbf{y}_1^{(i)}) \tag{84}$$

$$\mathbf{y}_1^{(i+1)} = \Delta\mathbf{y}_1 + \mathbf{y}_1^{(i)} \tag{85}$$

$$\tilde{F}'_{\Delta_2}(\mathbf{y}_2^{(i)})\Delta\mathbf{y}_2 = -\tilde{F}_{\Delta_2}(\mathbf{y}_2^{(i)}) \tag{86}$$

$$\mathbf{y}_2^{(i+1)} = \Delta\mathbf{y}_2 + \mathbf{y}_2^{(i)} \tag{87}$$

where $\mathbf{y}_1 = (\dot{\boldsymbol{V}}_B, \boldsymbol{V}_B, \boldsymbol{w}, \boldsymbol{u_L})$ and $\mathbf{y}_2 = (\dot{\boldsymbol{V}}_C, \boldsymbol{V}_C, \boldsymbol{\omega}, \boldsymbol{u_R})$ and

$$\tilde{F}'_{\Delta_1} = \begin{pmatrix} \frac{\partial f_{11}}{\partial \dot{V}_B} & \frac{\partial f_{11}}{\partial V_B} & \frac{\partial f_{11}}{\partial I_1} & \frac{\partial f_{11}}{\partial I_2} & \frac{\partial f_{11}}{\partial I_4} & 0 \\ \frac{\partial f_{12}}{\partial \dot{V}_B} & \frac{\partial f_{12}}{\partial V_B} & \frac{\partial f_{12}}{\partial I_1} & \frac{\partial f_{12}}{\partial I_2} & \frac{\partial f_{12}}{\partial I_4} & 0 \\ \frac{\partial f_{13}}{\partial \dot{V}_B} & \frac{\partial f_{13}}{\partial V_B} & \frac{\partial f_{13}}{\partial I_1} & \frac{\partial f_{13}}{\partial I_2} & \frac{\partial f_{13}}{\partial I_4} & \frac{\partial f_{13}}{\partial u_L} \\ \frac{\partial f_{14}}{\partial \dot{V}_B} & \frac{\partial f_{14}}{\partial V_B} & \frac{\partial f_{14}}{\partial I_1} & \frac{\partial f_{14}}{\partial I_2} & \frac{\partial f_{14}}{\partial I_4} & 0 \\ \frac{\partial f_{15}}{\partial \dot{V}_B} & \frac{\partial f_{15}}{\partial V_B} & \frac{\partial f_{15}}{\partial I_1} & \frac{\partial f_{15}}{\partial I_2} & \frac{\partial f_{15}}{\partial I_4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\tilde{F}'_{\Delta_2} = \begin{pmatrix} \frac{\partial f_{21}}{\partial \dot{V}_C} & \frac{\partial f_{21}}{\partial V_C} & \frac{\partial f_{21}}{\partial I_3} & \frac{\partial f_{21}}{\partial I_5} & 0 \\ \frac{\partial f_{22}}{\partial \dot{V}_C} & \frac{\partial f_{22}}{\partial V_C} & \frac{\partial f_{22}}{\partial I_3} & \frac{\partial f_{22}}{\partial I_5} & 0 \\ \frac{\partial f_{23}}{\partial \dot{V}_C} & \frac{\partial f_{23}}{\partial V_C} & \frac{\partial f_{23}}{\partial I_3} & \frac{\partial f_{23}}{\partial I_5} & \frac{\partial f_{23}}{\partial u_R} \\ \frac{\partial f_{24}}{\partial \dot{V}_C} & \frac{\partial f_{24}}{\partial V_C} & \frac{\partial f_{24}}{\partial I_3} & \frac{\partial f_{24}}{\partial I_5} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The first system updates the variables $(\dot{\boldsymbol{V}}_B, \boldsymbol{V}_B, \boldsymbol{w})$ and it used $\boldsymbol{u}_L$ instead of $\boldsymbol{V}_C$ in equation 3 because of the delay of information. In the second system we update the variables $(\dot{\boldsymbol{V}}_C, \boldsymbol{V}_C, \boldsymbol{\omega})$ and we used the value $\boldsymbol{u}_R$ instead of $\boldsymbol{I}_2$ in equation 3. What we did in the first systems is to estimate $\mathbf{V}_C$ by $V_C(t_n)$ and we estimated $\mathbf{I}_2$ by $I_2(t_n)$ in the second system.

We make one change to this, we change the right hand side in equation (84) from $\tilde{F}_{\Delta_1}(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{w}, \boldsymbol{u_L})$ to $F_{\Delta_1}(\dot{\mathbf{V}}_B, \mathbf{V}_B, \mathbf{w}, \boldsymbol{u_L}, \boldsymbol{V}_C)$

and the right hand side in equation (86) from $\tilde{F}_{\Delta_2}(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{u_R})$ to $F_{\Delta_2}(\dot{\mathbf{V}}_C, \mathbf{V}_C, \boldsymbol{\omega}, \boldsymbol{u_R}, \boldsymbol{I}_2)$ then we write this Newton iteration as

$$\tilde{F}'_{\Delta_1}(\boldsymbol{y}_1^{(i)})\Delta\boldsymbol{y}_1 = -F_{\Delta_1}(\boldsymbol{y}_1^{(i)}) \tag{88}$$

$$\boldsymbol{y}_1^{(i+1)} = \Delta\boldsymbol{y}_1 + \boldsymbol{y}_1^{(i)} \tag{89}$$

$$\tilde{F}'_{\Delta_2}(\boldsymbol{y}_2^{(i)})\Delta\boldsymbol{y}_2 = -F_{\Delta_2}(\boldsymbol{y}_2^{(i)}) \tag{90}$$

$$\boldsymbol{y}_2^{(i+1)} = \Delta\boldsymbol{y}_2 + \boldsymbol{y}_2^{(i)} \tag{91}$$

rewriting this to get it on fixed point form

$$\boldsymbol{y}_1^{(i+1)} = \boldsymbol{y}_1^{(i)} - \tilde{F}'_{\Delta_1}(\boldsymbol{y}_1^{(i)})^{-1} F_{\Delta_1}(\boldsymbol{y}_1^{(i)}) \tag{92}$$

$$\boldsymbol{y}_2^{(i+1)} = \boldsymbol{y}_2^{(i)} - \tilde{F}'_{\Delta_2}(\boldsymbol{y}_2^{(i)})^{-1} F_{\Delta_2}(\boldsymbol{y}_2^{(i)}) \tag{93}$$

Let $\boldsymbol{x} = (\boldsymbol{y}_1, \boldsymbol{y}_2)$ and let

$$\varphi(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{y}_1 - \tilde{F}'_1(\boldsymbol{y}_1)^{-1} F_{\Delta_1}(\boldsymbol{y}_1) \\ \boldsymbol{y}_2 - \tilde{F}'_2(\boldsymbol{y}_2)^{-1} F_{\Delta_2}(\boldsymbol{y}_2) \end{bmatrix} = \boldsymbol{x} \tag{94}$$

The benefit of the introducing the delay of information is that instead of solving a linear system of 9×9, we solve two systems in parallel which are 5×5 and 4×4 respectively. Next we will try to get control over the convergence of the new method.

6 Convergence analysis

In this section we use Banach's fixed point theorem [4] to investigate the convergence of Newton's method when forcing very specific elements in the Jacobian to be zero.

Assume that the iteration matrix $B(x)$ is invertable, then the fixed point x^* of

$$\varphi(x) = x - B(x)^{-1}F(x)$$

is a root of $F(x)$. If we define the sequence

$$x^{(i+1)} = x^{(i)} - B(x^{(i)})^{-1}F(x^{(i)})$$

we can use Banach fixed point theorem to see if this sequence converges to x^*.

Notice that when $B(x) = F'(x)$ we have Newton's method. In practice we often use an approximation $B(x) \approx F'(x)$. One such example is $B(x) = F'(x^{(0)})$, this is known as simplified Newton's method.

Introducing a delay of information such that we split the electrical circuit into two parts can be seen as changing $B(x) = F'(x)$ to a very specific approximation $B(x) \approx F'(x)$. The aim of this section is to investigate when such a delay can be justified by Banach's fixed point theorem.

Definition 4. A function $\varphi : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$. φ is called Lipschitz continuous on $D_0 \subseteq D$. If there exists a constant L such that

$$||\varphi(x) - \varphi(y)|| \leq L \cdot ||x - y|| \quad \forall x, y \in D_0$$

if L $\in$ [0,1) we say φ is a contraction.

Theorem 1 (Banach's fixed point)**.** Let $\varphi : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be a contraction on a closed set $D_0 \subseteq D$ and suppose that $\varphi(D_0) \subset D_0$, then φ admits a unique fixed point $x^* \in D_0$, i.e, $\varphi(x^*) = x^*$.
Furthermore, for every $x^{(0)} \in D_0$ the sequence $\{x^{(k)}\}_{k=0}^{\infty}$ defined by $\varphi(x^{(k)}) = x^{(k+1)}$ converges to this fixed point and the a priori error estimate

$$||x^{(k)} - x^*|| \leq \frac{L^k}{1-L}||x^{(1)} - x^{(0)}||$$

holds.

6.1 The linear case

Assume that $F(x) = Ax + b$, then Newton's method on $F(x)$ converges in one step. Changing elements in $B(x)$ and then taking an exact inverse can be generalized using

Theorem 2. Woodbury's matrix identity

Let $A \in \mathbb{R}^{n\times n}$ be a invertable matrix and let $U \in \mathbb{R}^{n\times k}$, $V \in \mathbb{R}^{k\times n}$ and let $C \in \mathbb{R}^{k\times k}$ be invertable. Then if

$$(C^{-1} + VA^{-1}U)$$

is invertable, we have

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

Proof. see [7]. □

Let us denote the Newton iteration by

$$\varphi(\mathbf{x}) = \mathbf{x} - F'(\mathbf{x})^{-1}F(\mathbf{x}) \tag{95}$$

notice that

$$\varphi(\mathbf{x}) - \varphi(\mathbf{y}) = \mathbf{x} - A^{-1}(A\mathbf{x} + b) - \mathbf{y} + A^{-1}(A\mathbf{y} + b) = 0 \tag{96}$$

The new method where we force elements in the Jacobian to be zero is a mix of Newton's method and a fixed-point method. We define our fixed point map by

$$\tilde{\varphi}(\mathbf{x}) = \mathbf{x} - (A + UCV)^{-1}F(\mathbf{x}) \tag{97}$$

Let us estimate the Lipschitz constant of $\tilde{\varphi}$

$$||\tilde{\varphi}(\mathbf{x}) - \tilde{\varphi}(\mathbf{y})|| = ||\mathbf{x} - (A + UCV)^{-1}F(\mathbf{x}) - \mathbf{y} + (A + UCV)^{-1}F(\mathbf{y})|| = \tag{98}$$

$$||\varphi_1(\mathbf{x}) - \varphi_1(\mathbf{y}) + A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}F(\mathbf{x}) - \tag{99}$$

$$A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}F(\mathbf{y})|| = \tag{100}$$

$$||A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}F(\mathbf{x}) - \tag{101}$$

$$A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}F(\mathbf{y})|| = \tag{102}$$

$$||A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}(A\mathbf{x}+b)-A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}(A\mathbf{y}+b)||= \tag{103}$$

$$||A^{-1}U(C^{-1}+VA^{-1}U)^{-1}V(\mathbf{x}-\mathbf{y})|| \leq ||A^{-1}U(C^{-1}+VA^{-1}U)^{-1}V|| \cdot ||\mathbf{x}-\mathbf{y}|| \tag{104}$$

hence the new method converges if

$$||A^{-1}U(C^{-1}+VA^{-1}U)^{-1}V|| < 1 \tag{105}$$

The goal of adding UCV to A is to produce blocks in $A+UCV$ where all the vectors in one block are linearly independent to all the vectors in the other blocks. This results in solving smaller linear systems but using more Newton iterations.
Let us look at how this condition can be applied on a mechanical example and if the physical interpretation agrees with the results.

6.2 A mechanical example

We leave the electrical system for a moment and look at a mechanical system. Mechanical systems are intuitively easier to get an idea if we can decouple. The example we look at is two identical pendulums connected by a spring. The aim is to investigate if we can decouple the pendulums by introducing a delay of information while still maintaining convergence.

Let L be the length between the pendulums pivot point and the center of mass , let m be the mass of the object attached to the rod, let k be the spring constant and let g be the gravity constant. The angular displacement is denoted x_A and x_B for the right and left pendulums respectively. The equations of motion for the spring coupled pendulums is linear if x_A and x_B is "small" and non linear if they are large. We want to investigate the case when x_A and x_B is "small" and use (105) to see when we can justify a decoupling. The equations of motion is given by

$$\ddot{x}_A = -\frac{g}{L}x_A - \frac{k}{m}(x_A - x_B) \tag{106}$$

$$\ddot{x}_B = -\frac{g}{L}x_B + \frac{k}{m}(x_A - x_B) \tag{107}$$

Notice that the equation for one pendulum with small enough angles is

$$\ddot{\theta} = -\frac{g}{L}\theta \tag{108}$$

the term $\frac{k}{m}(x_A - x_B)$ in (106) and $\frac{k}{m}(x_A - x_B)$ in (107) comes from the potential energy of the spring connecting the pendulums. When either the spring constant $k \to 0$ or the mass $m \to \infty$ the term from the potential energy disappears and the pendulums behave more and more like two detached pendulums. Without loss of generality we will henceforth use $m = 1$. We will introduce a delay of information of x_B in (110) and x_A in (112). We will check condition (105) to see when this is allowed. Our expectation is that when the spring constant becomes small enough a decoupling is possible

By order reduction we get

$$\dot{x}_A = y_A \tag{109}$$
$$\dot{y}_A = -\frac{g}{L}x_A - k(x_A - x_B) \tag{110}$$
$$\dot{x}_B = y_B \tag{111}$$
$$\dot{y}_B = -\frac{g}{L}x_B + k(x_A - x_B) \tag{112}$$

We insert implicit Euler's method into our differential equations and extend with the numerical method to get the augmented differential equations. We now treat the system as a system of equations.

$$F_\Delta(\dot{\mathbf{x}}_A, \dot{\mathbf{y}}_A, \dot{\mathbf{x}}_B, \dot{\mathbf{y}}_B, \mathbf{x}_A, \mathbf{y}_A, \mathbf{x}_B, \mathbf{y}_B) = \begin{bmatrix} \dot{\mathbf{x}}_A - \mathbf{y}_A \\ \dot{\mathbf{y}}_A + \frac{g}{L}\mathbf{x}_A - k(\mathbf{x}_A - \mathbf{x}_B) \\ h\dot{\mathbf{x}}_A + x_A(t_n) - \mathbf{x}_A \\ h\dot{\mathbf{y}}_A + y_A(t_n) - \mathbf{y}_A \\ \dot{\mathbf{x}}_B - \mathbf{y}_B \\ \dot{\mathbf{y}}_B + \frac{g}{L}\mathbf{x}_B - k(\mathbf{x}_B - \mathbf{x}_A) \\ h\dot{\mathbf{x}}_B + x_B(t_n) - \mathbf{x}_B \\ h\dot{\mathbf{y}}_B + y_B(t_n) - \mathbf{y}_B \end{bmatrix} = 0$$

Decoupling the linear system of equations is done by using the previous value for $\mathbf{x}_B$ in equation 2 and the previous value for $\mathbf{x}_A$ in equation 6. Then the first 4 equations only depend on variables describing the right pendulum, i.e, $\dot{\mathbf{x}}_A, \dot{\mathbf{y}}_A, \mathbf{x}_A,\ \mathbf{y}_A$ and the last 4 equations only depend on variables describing the left pendulum, i.e, $\dot{\mathbf{x}}_B, \dot{\mathbf{y}}_B, \mathbf{x}_B,\ \mathbf{y}_B$

$$\tilde{F}_\Delta(\dot{\mathbf{x}}_A, \dot{\mathbf{y}}_A, \dot{\mathbf{x}}_B, \dot{\mathbf{y}}_B, \mathbf{x}_A, \mathbf{y}_A, \mathbf{x}_B, \mathbf{y}_B) = \begin{bmatrix} \dot{\mathbf{x}}_A - \mathbf{y}_A \\ \dot{\mathbf{y}}_A + \frac{g}{L}\mathbf{x}_A - k(\mathbf{x}_A - x_B(t_n)) \\ h\dot{\mathbf{x}}_A + x_A(t_n) - \mathbf{x}_A \\ h\dot{\mathbf{y}}_A + y_A(t_n) - \mathbf{y}_A \\ \dot{\mathbf{x}}_B - \mathbf{y}_B \\ \dot{\mathbf{y}}_B + \frac{g}{L}\mathbf{x}_B - k(\mathbf{x}_B - x_A(t_n)) \\ h\dot{\mathbf{x}}_B + x_B(t_n) - \mathbf{x}_B \\ h\dot{\mathbf{y}}_B + y_B(t_n) - \mathbf{y}_B \end{bmatrix} = 0$$

We will not solve $F_\Delta(\mathbf{x}^*) = 0$ using Newton's method, i.e,

$$\varphi(\mathbf{x}) = \mathbf{x} - F'_\Delta(\mathbf{x})^{-1}F(\mathbf{x}) \tag{113}$$

but we use the new method which is a mix of Newton's method and a fixed-point method. We define our fixed point map by

$$\tilde{\varphi}(\mathbf{x}) = \mathbf{x} - \tilde{F}'_{\Delta}(\mathbf{x})^{-1} F(\mathbf{x}) \tag{114}$$

where

$$\tilde{F}'_{\Delta}(\mathbf{x})^{-1} = (A + UCV)^{-1}$$

and $C = \mathrm{I} \in \mathbb{R}^{2\times 2}$, $U = (e_2, e_6)^T \in \mathbb{R}^{7\times 2}$ and $V = k(e_7, e_5) \in \mathbb{R}^{2\times 7}$.

Since $F_{\Delta}(\mathbf{x}) = A\mathbf{x} + b$ is linear we can use the condition

$$||A^{-1}U(C^{-1} + VA^{-1}U)^{-1}V|| < 1 \tag{115}$$

to see if $\tilde{\varphi}$ convergence to the same fixed point as φ. We plot

$$||A^{-1}U(C^{-1} + VA^{-1}U)^{-1}V||_2 \tag{116}$$

for different step sizes when we varying the spring constant.

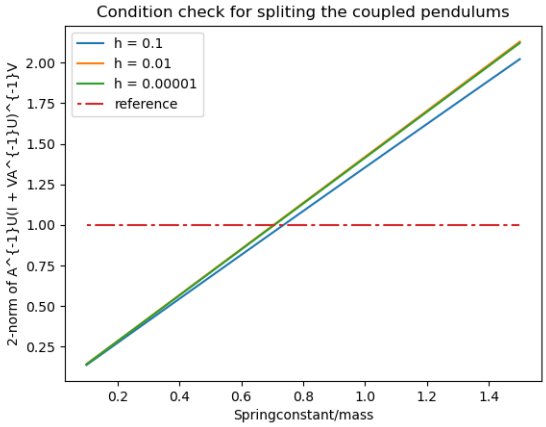

Figure 26: For values below the reference value the pendulums can be decoupled.

Figure 26 show the expected result, we can decoupled the pendulums when the spring constant is small enough.

7 Mixed mode integration from Newton's perspective

Next we present an alternative way of mixing explicit and implicit methods using the ideas discussed in the previous section. Let

$$\dot{x} = f(x,t) \in \mathbb{R}^N \tag{117}$$

and let us use implicit Euler method to solve this

$$x_{n+1} = x_n + h \cdot f(x_{n+1}, t_{n+1}) \tag{118}$$

we rewrite this to

$$F(x_{n+1},) = x_{n+1} - x_n - h \cdot f(x_{n+1}, t_{n+1}) = 0 \tag{119}$$

and apply Newton's method on (118) to solve for x_{n+1}. We write Newton's method on fixed point form and define the sequence $\{x_n^{(i)}\}_{i=0}^{\infty}$ by

$$\varphi(x_n^{(i)}) = x_n^{(i)} - F'(x_n^{(i)})^{-1} F(x_n^{(i)}) = x_n^{(i+1)} \tag{120}$$

then under the conditions in Banach's fixed point theorem this sequence converges to x_{n+1}.

Let us change the iteration matrix $F'(x)^{-1}$ to an approximation. The approximation we use is $B(x)^{-1} = \mathrm{I}$. Define the fixed point

$$\tilde{\varphi}(x_n^{(i)}) = x_n^{(i)} - F(x_n^{(i)}) \tag{121}$$

if we just use a single iteration we get

$$\tilde{\varphi}(x_n^{(0)}) = x_n - (x_n - x_n) + h \cdot f(x_n, t_{n+1}) \tag{122}$$

The idea then becomes, mix explicit and implicit methods using the iteration matrix. In inline integration it would work like this.

7.1 Changing the iteration matrix

Let us think of mixed mode integration in inline integration in terms of changing the iteration matrix. Notice that if we inline a DAE using implicit Euler's method the partitioning algorithm will present us with algebraic loops, i.e, system of equations. These systems have in general the form

$$G_\Delta(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) = \begin{bmatrix} G(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) \\ -\mathbf{x} + h \cdot \dot{\mathbf{x}} + \mathbf{x}(t_n) \end{bmatrix} = 0 \tag{123}$$

This means that some of the equations comes from implicit Euler's method. Let us assume that we solve this system using Newton's method, i.e,

$$\varphi(\mathbf{y}^{(i)}) = \mathbf{y}^{(i)} - G'_\Delta(\mathbf{y}^{(i)})^{-1} G_\Delta(\mathbf{y}^{(i)}) = \mathbf{y}^{(i+1)} \tag{124}$$

we change one variable $\dot{z}$ of $\dot{x}$ from being solved using implicit Euler's method

$$-\mathbf{z} + h \cdot \dot{\mathbf{z}} + \mathbf{z}(t_n) = 0$$

to being solved using explicit Euler's method.

$$-\mathbf{z} + h \cdot \dot{\mathbf{z}}(t_n) + \mathbf{z}(t_n) = 0$$

This would change one element in

$$G'_\Delta(\mathbf{y})$$

namely

$$\frac{\partial G_\Delta(\mathbf{y})}{\partial \dot{\mathbf{z}}} = h$$

would change to zero. This is a rank-one change to the Jacobian, we call this approximation B($\mathbf{x}$). Then define the sequence

$$\tilde{\varphi}(\mathbf{y}^{(i)}) = \mathbf{y}^{(i)} - B(\mathbf{y}^{(i)})^{-1} G_\Delta(\mathbf{y}^{(i)}) = \mathbf{y}^{(i+1)} \tag{125}$$

then by Banach's fixed point theorem we have that, if this sequence converges to $\mathbf{y}^*$, then $G_\Delta(\mathbf{y}^*) = 0$. We used G_Δ from the implicit Euler's method, but we changed the iteration matrix to where $\dot{z}$ is solved using explicit Euler's method. The idea is to zero out very specific elements in the Jacobian and introduce a variable as "slow" using the iteration matrix. This gives the user control over the convergence and the method will not

introduce an error if it converges.

Thus we have introduced a quantifiable way of declaring a variable as "slow" or "fast" using the iteration matrix in Newton's method when inline integration is used on

$$F(\dot{x}, x, w, t) = 0 \tag{126}$$

8 Discussion and further work

We used a strongly connected component to represent the algebraic loops and a delay of information was used to removed edges in the strongly connected component. In the electrical circuit we introduced a delay of information at two places which removed two edges such that the strongly connected component was split into two smaller strongly connected components. Finding edges that would split the strongly connected component into parts should perhaps not be done by hand. We would therefore like to explore a graph theoretical idea for which there exists algorithms for. The idea is based on the following: an edge e of a graph G is a bridge if its deletion increases the number of strongly connected components of G.

To clarify, a strongly connected component is a representation of the system of equations that needs to be solved. Removing a bridge would separate the system of equations that needs to be solved into two parts of lower dimensions. One algorithm for finding bridges is Tarjan's bridge search algorithm.

In Section 5.3 we used an electrical circuit as an example to show that we could split a system of equations into two parts by introducing a delay of information. We wrote Newton's method on fixed point form for the connected electrical circuit as

$$\varphi(\mathbf{y}) = \mathbf{y} - F'_{\Delta}(\mathbf{y})^{-1} F_{\Delta}(\mathbf{y}) \tag{127}$$

and for the detached electrical circuit as

$$\tilde{\varphi}(\mathbf{y}) = \mathbf{y} - \tilde{F}'_{\Delta}(\mathbf{y})^{-1} \tilde{F}_{\Delta}(\mathbf{y}) \tag{128}$$

Notice that this is a delay of information together with a exchange of information between time steps. We mentioned that φ and $\tilde{\varphi}$ might converge

to a fixed point, but not necessarily to the same fixed point. The idea that we would like to further explore is that, if $\tilde{\varphi}$ converges to a fixed point $\tilde{y}^*$ could we in practice test if $\tilde{F}_\Delta(\tilde{\mathbf{y}}^*) = 0$.

9 Summary

The task was formulated as, Modelon's real time simulation tool, called inline integration, must operate without overruns on the DAE

$$F(\dot{x}, x, w, t) = 0 \in \mathbb{R}^{n_x+n_w} \tag{129}$$

The general idea of inline integration was explained in section 2. We treat

$$F_\Delta(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) = \begin{bmatrix} F(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t_{n+1}) \\ -\mathbf{x} + h \cdot \dot{\mathbf{x}} + old(\mathbf{x}) \end{bmatrix} = 0 \in \mathbb{R}^{2n_x+n_w} \tag{130}$$

as a system of equations, on which we preform partitioning and tearing. This is done to reduce the dimensions of the system of equations that needs to be solved. These symbolic methods presents us with algebraic loops, i.e, systems of equations, that needs to be solved. Solving these is the most time requiring part of updating the solution from t_n to t_{n+1}. The aim was therefore to further reduce the dimensions of the algebraic loops while still preserving the accuracy. We presented two ideas, mixed mode integration [9] and *a delay of information*.

In Section 3 the idea of mixed mode integration was introduced on ODE's. The idea was to split

$$\dot{x} = f(x, t) \in \mathbb{R}^N \tag{131}$$

into a "fast" and a "slow" part, i.e,

$$\dot{x}^S = f^S(x^S, x^F, t) \in \mathbb{R}^{k_1} \tag{132}$$

$$\dot{x}^F = f^F(x^S, x^F, t) \in \mathbb{R}^{k_2} \tag{133}$$

and apply an explicit solver on the "slow" part and an implicit solver on the "fast part.

In Section 4 we used two examples to get a better understanding of inline integration and how it works with explicit and implicit numerical methods. The main example was a spring pendulum, where a formulation was chosen such that we before hand had an idea of what state variables should be "slow". We noticed that declaring variables as "slow" could have more or less impact in terms of reducing the dimensions of the system of equations that needs to be solved. We therefore asked, how to suggest to the user a set of variables that could be declared as slow such that accuracy was preserved and the dimensions of the system of equations was reduced. The tool we used to explore the idea came from graph theory.

In Section 5 the algebraic loops was discussed using graph theory, where a graph was used to represent the algebraic loop. We noticed that introducing a delay of information removes an edge in the graph. A simple proposition gave us a tool to see what happened to the graph after an edge was removed. We used a simple electrical circuit as an example to further explore the idea and noticed that the algebraic loop could be split into two parts by introducing a delay of information at two places. We formulated the delay using Newton's method. We used this formulation to introduce two types of information exchanges, exchange between time steps and exchange between the Newton iterations. The second type was viewed as using a very specific approximation of the Jacobian where certain elements where put to zero.

In Section 6 we used Banach's fixed point theorem to look at the convergence of Newton's method when forcing very specific elements in the Jacobian to be zero and then taking an exact inverse. Using Woodbury's matrix identity we got an upper bound for the Lipschitz constant when the modified Newton's method where used to find the root of the linear function $F(x) = Ax + b$. The purpose of the modified Newton's method on the linear problem was to produce linearly independent blocks in A by adding a matrix UCV to A. By doing this we could perhaps solve smaller systems to the expense of using more then one iteration. Two pendulums connected by a spring modeled as a differential equation when the angles was "small" was used as an example to show that we could make a physical interpretation of when a delay of information could be used and that this interpretation actually agreed with the upper bound for Lipschitz constant of the modified Newton's method.

In Section 7 we explained the idea of mixing explicit and implicit methods using the iteration matrix. Using this approach an additional error would

not be introduced if Newton's method converged.

References

[1] Andersson, C and Führer, C and Åkesson, J. Assimulo: A unified framework for ODE solvers. Mathematics and Computers in Simulation. Volume 116. (2015).

[2] Bélanger, J., P. Venne and J. Paquin. "The What , Where and Why of Real-Time Simulation." (2010).

[3] Casella, F. "Exploiting Weak Dynamic Interactions in Modelica." (2005).

[4] Eich-Soellner, Edda and C. Führer. "Numerical Methods in Multibody Dynamics." (2013).

[5] Elmqvist, H., M. Otter and F. Cellier. "Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equations Systems." (1995).

[6] Jones,E Oliphant,T Peterson, P and others.SciPy: Open Source Scientific Tools for Python. (2001–)

[7] Max, A. W. Inverting modified matrices. In Memorandum Rept. 42, Statistical Research Group. (1950).

[8] Pareschi, L Giovanni, R. Implicit–Explicit Runge–Kutta schemes for stiff systems of differential equations. (2001).

[9] Schiela, A. Olsson, Hans. Mixed-mode Integration for Real-time Simulation. 69-75. (2000).

Master's Theses in Mathematical Sciences 2021:E74
ISSN 1404-6342

LUNFNA-3041-2021

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/