Numerical Simulation of Dynamic Systems XV

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

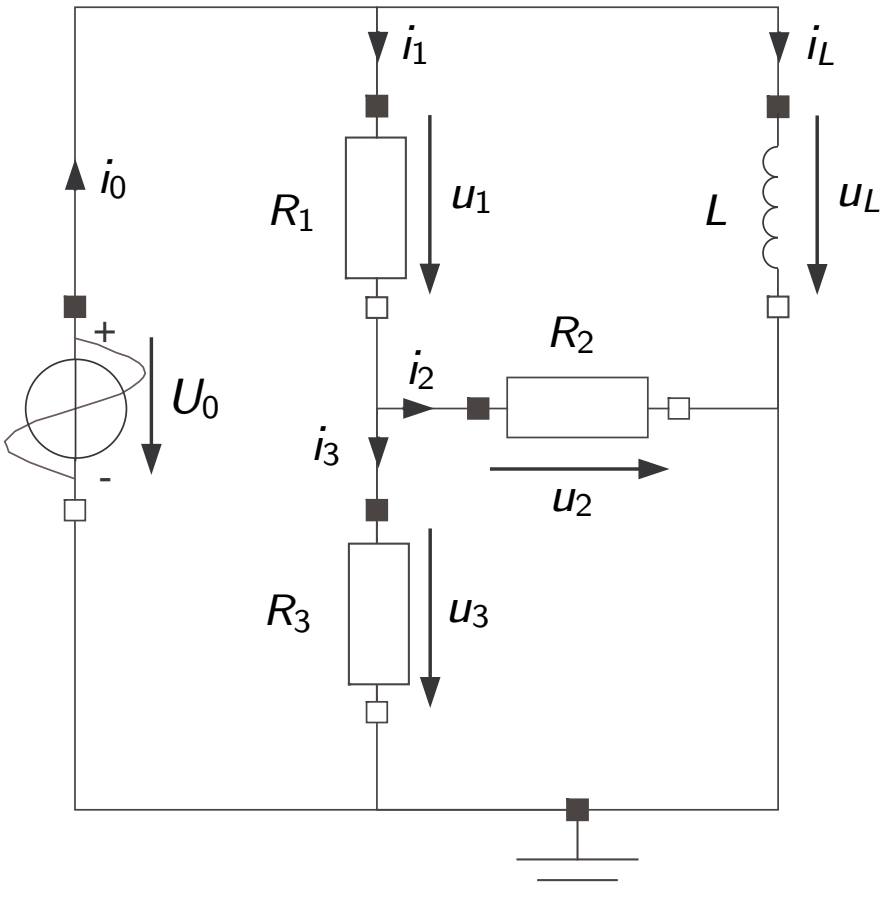
April 16, 2013

Algebraic Loops

Unfortunately, the approach proposed in the previous presentation doesn't always work:

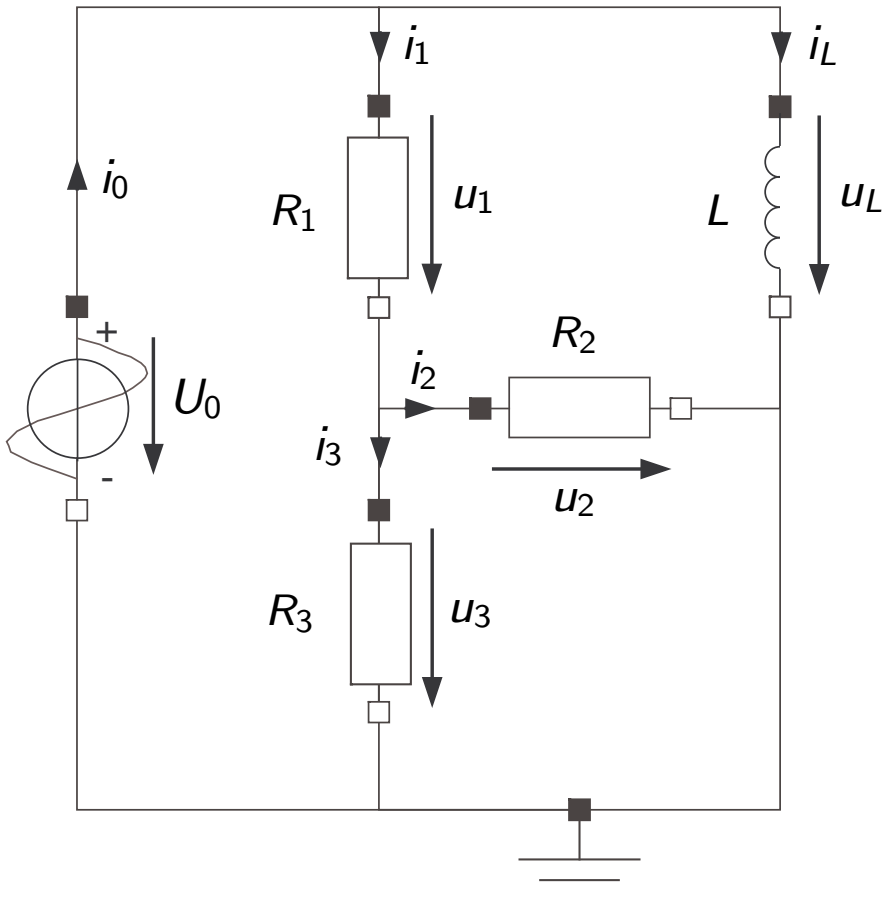
Algebraic Loops

Unfortunately, the approach proposed in the previous presentation doesn't always work:



Algebraic Loops

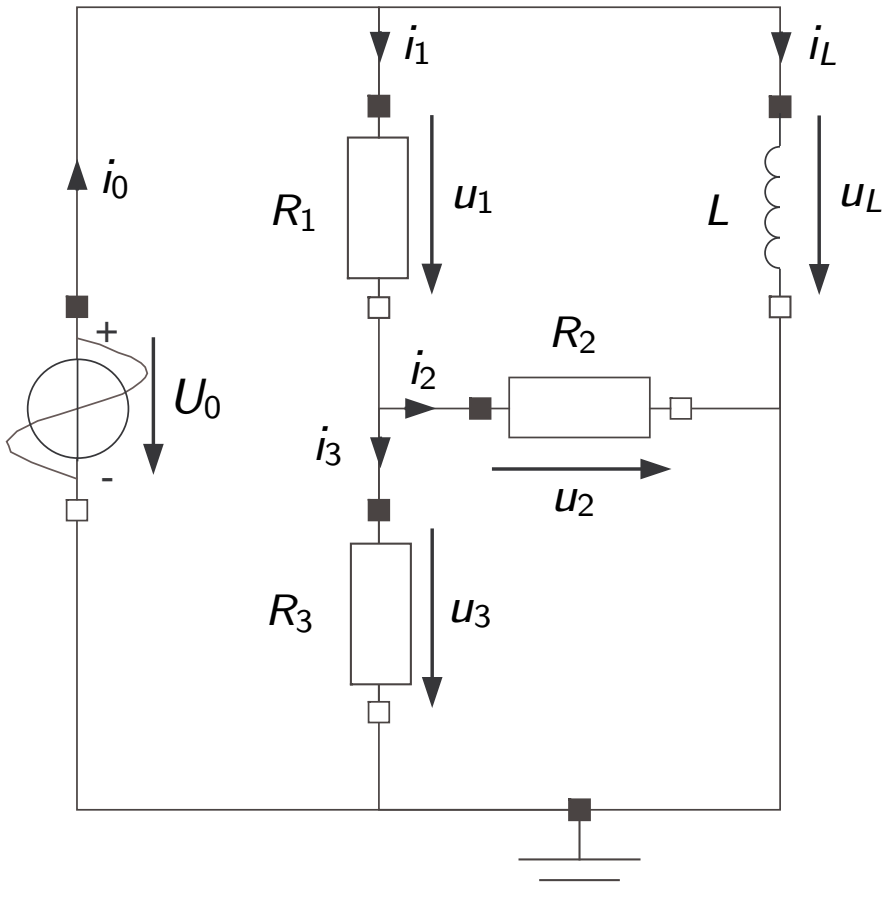
Unfortunately, the approach proposed in the previous presentation doesn't always work:



$$
\begin{aligned}
1:&\quad u_0 = f(t)\\
2:&\quad u_1 = R_1 \cdot i_1\\
3:&\quad u_2 = R_2 \cdot i_2\\
4:&\quad u_3 = R_3 \cdot i_3\\
5:&\quad u_L = L \cdot \frac{di_L}{dt}\\
\\
6:&\quad u_0 = u_1 + u_3\\
7:&\quad u_L = u_1 + u_2\\
8:&\quad u_3 = u_2\\
\\
9:&\quad i_0 = i_1 + i_L\\
10:&\quad i_1 = i_2 + i_3
\end{aligned}
$$

Algebraic Loops

Unfortunately, the approach proposed in the previous presentation doesn't always work:

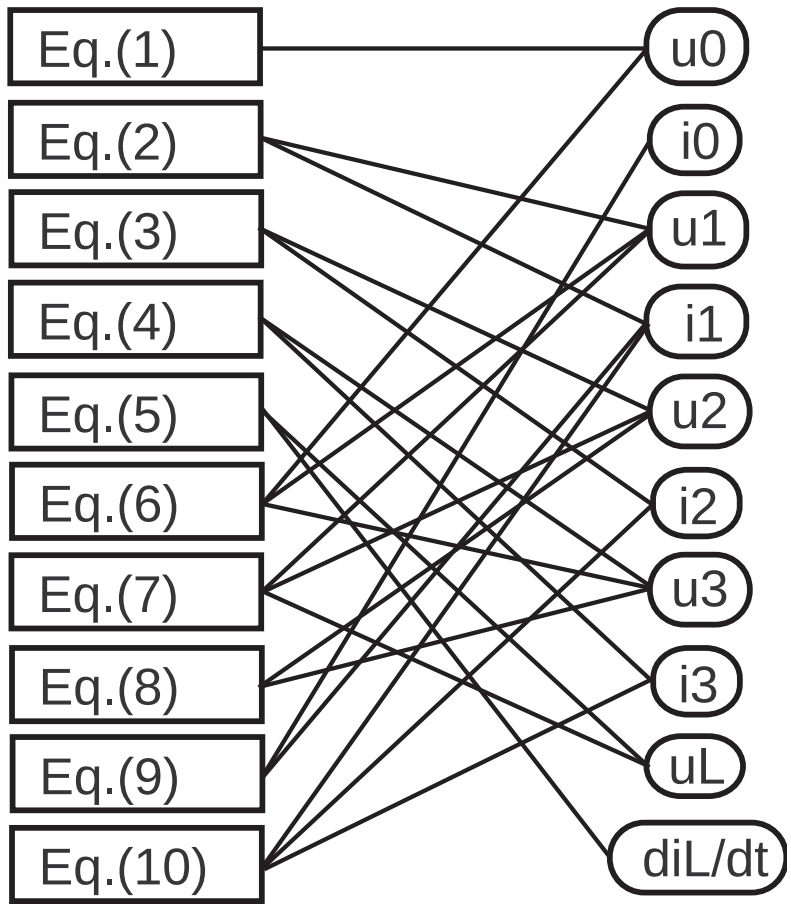


$$
\begin{aligned}
1: \quad & u_0 = f(t) \\
2: \quad & u_1 = R_1 \cdot i_1 \\
3: \quad & u_2 = R_2 \cdot i_2 \\
4: \quad & u_3 = R_3 \cdot i_3 \\
5: \quad & u_L = L \cdot \frac{di_L}{dt} \\
6: \quad & u_0 = u_1 + u_3 \\
7: \quad & u_L = u_1 + u_2 \\
8: \quad & u_3 = u_2 \\
9: \quad & i_0 = i_1 + i_L \\
10: \quad & i_1 = i_2 + i_3
\end{aligned}
$$

$\Rightarrow$ **We got again 10 implicitly formulated DAEs in 10 unknowns.**

Algebraic Loops II

Let us try the same approach. The structure digraph of the DAE system can be drawn as follows:

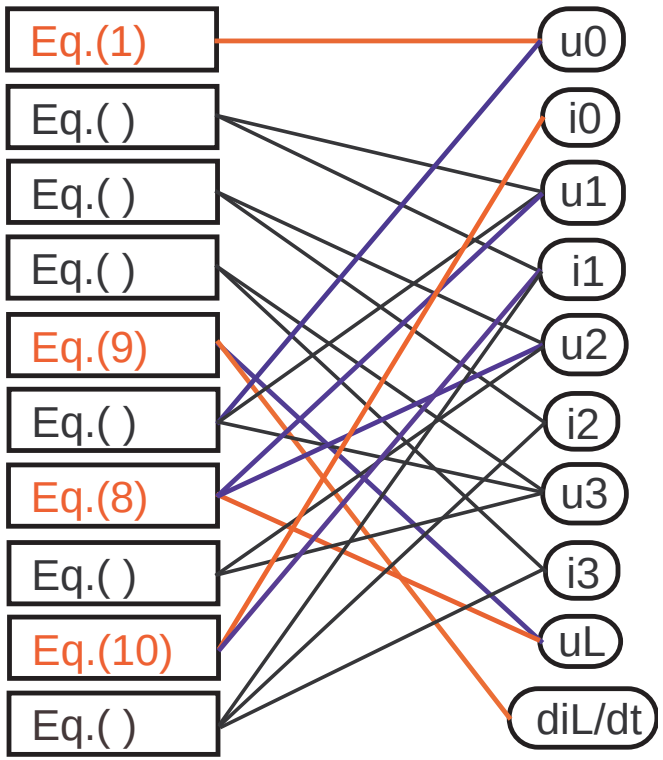


Algebraic Loops III

After a few steps of causalization:

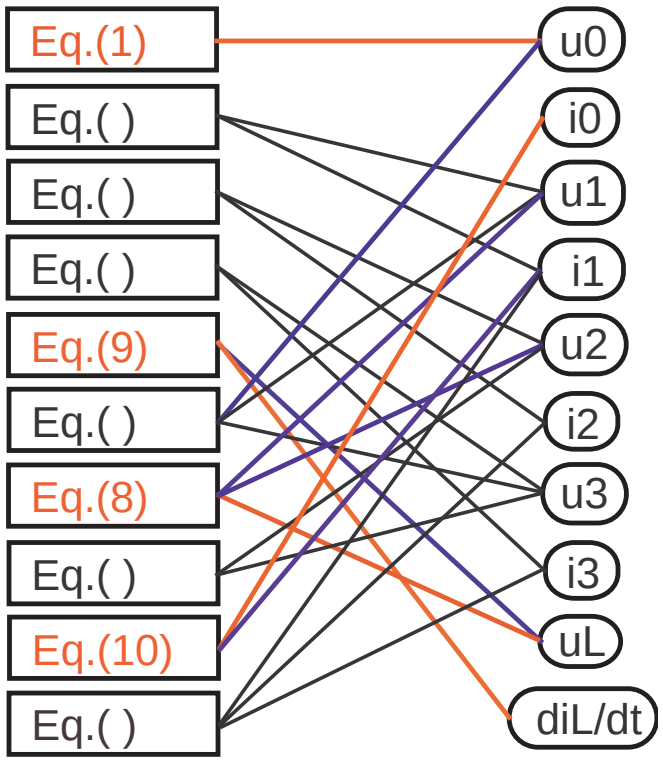
Algebraic Loops III

After a few steps of causalization:



Algebraic Loops III

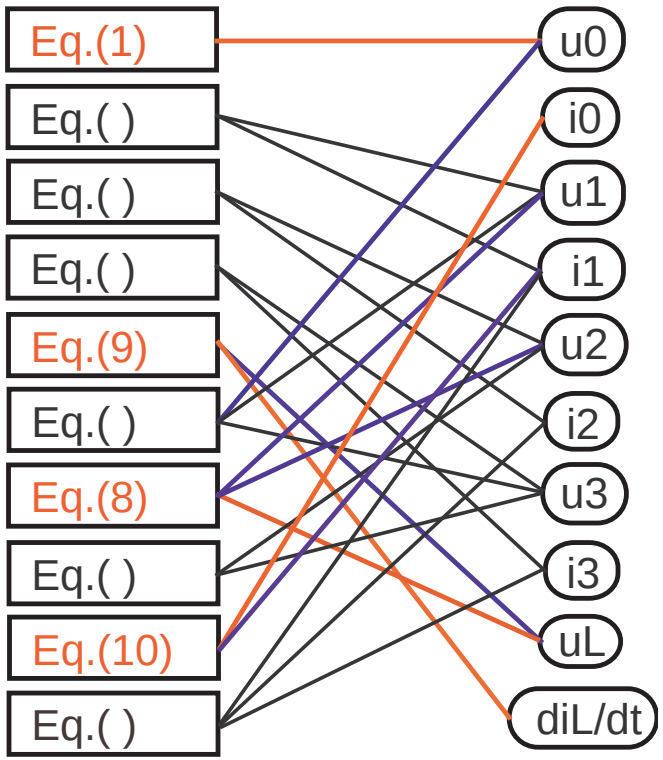
After a few steps of causalization:



- After four causalization steps, the algorithm stalls.

Algebraic Loops III

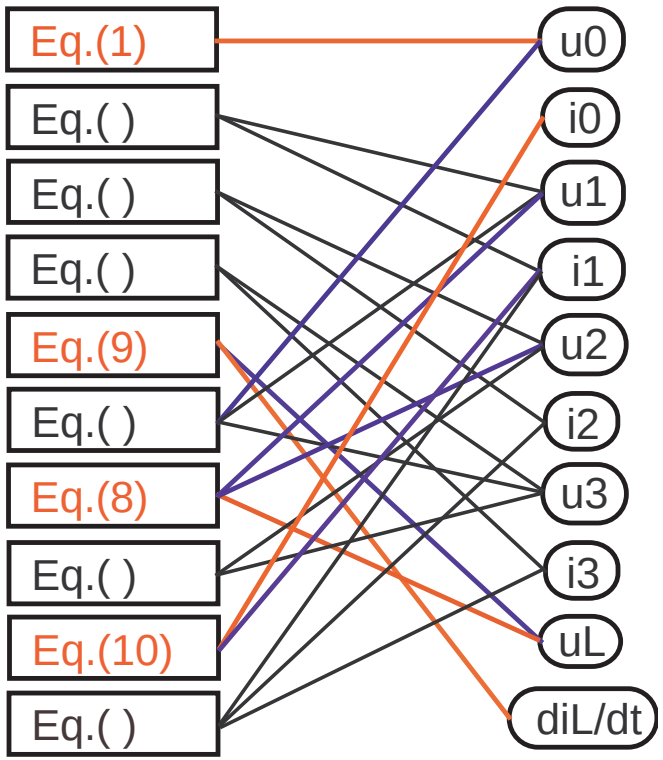
After a few steps of causalization:



- After four causalization steps, the algorithm stalls.
- Every remaining acausal equation has at least two black lines attached, i.e., contains at least two unknowns.

Algebraic Loops III

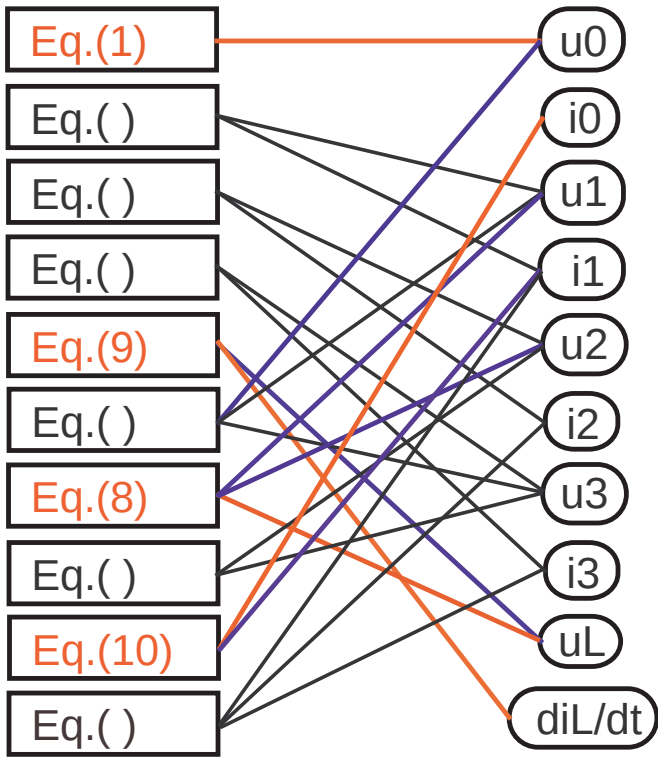
After a few steps of causalization:



- After four causalization steps, the algorithm stalls.
- Every remaining acausal equation has at least two black lines attached, i.e., contains at least two unknowns.
- Every remaining non-causalized variable has at least two black lines attached, i.e., appears in at least two different equations.

Algebraic Loops III

After a few steps of causalization:



- After four causalization steps, the algorithm stalls.
- Every remaining acausal equation has at least two black lines attached, i.e., contains at least two unknowns.
- Every remaining non-causalized variable has at least two black lines attached, i.e., appears in at least two different equations.
- **The remaining acausal equations form an algebraic loop.**

Algebraic Loops IV

Let us read out the partially causalized equations from the structure digraph. I placed all unknowns of the acausal equations to the left of the equal sign:

$$
\begin{aligned}
u_0 &= f(t) \\
u_1 - R_1 \cdot i_1 &= 0 \\
u_2 - R_2 \cdot i_2 &= 0 \\
u_3 - R_3 \cdot i_3 &= 0 \\
u_1 + u_3 &= u_0 \\
u_2 - u_3 &= 0 \\
i_1 - i_2 - i_3 &= 0 \\
u_L &= u_1 + u_2 \\
\frac{di_L}{dt} &= u_L / L \\
i_0 &= i_1 + i_L
\end{aligned}
$$

Algebraic Loops IV

Let us read out the partially causalized equations from the structure digraph. I placed all unknowns of the acausal equations to the left of the equal sign:

$$
\begin{aligned}
u_0 &= f(t) \\
u_1 - R_1 \cdot i_1 &= 0 \\
u_2 - R_2 \cdot i_2 &= 0 \\
u_3 - R_3 \cdot i_3 &= 0 \\
u_1 + u_3 &= u_0 \\
u_2 - u_3 &= 0 \\
i_1 - i_2 - i_3 &= 0 \\
u_L &= u_1 + u_2 \\
\frac{di_L}{dt} &= u_L / L \\
i_0 &= i_1 + i_L
\end{aligned}
$$

There is a group of six equations in six unknowns that have not yet been causalized and that need to be solved together, because they form an *algebraic loop*.

Algebraic Loops V

The structure incidence matrix of the partially sorted equations is in *block lower triangular (BLT) form*:

$$\mathbf{S} = \begin{array}{c} \\ 1: \\ \\ 2: \\ 3: \\ 4: \\ 5: \\ 6: \\ 7: \\ \\ 8: \\ \\ 9: \\ \\ 10: \end{array} \left[\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} u_0 & & u_1 & i_1 & u_2 & i_2 & u_3 & i_3 & & u_L & & \frac{di_L}{dt} & & i_0 \\ 1 & | & 0 & 0 & 0 & 0 & 0 & 0 & & 0 & & 0 & & 0 \\ - & + & - & - & - & - & - & - & . & & & & & \\ 0 & | & 1 & 1 & 0 & 0 & 0 & 0 & | & 0 & & 0 & & 0 \\ 0 & | & 0 & 0 & 1 & 1 & 0 & 0 & | & 0 & & 0 & & 0 \\ 0 & | & 0 & 0 & 0 & 0 & 1 & 1 & | & 0 & & 0 & & 0 \\ 1 & | & 1 & 0 & 0 & 0 & 1 & 0 & | & 0 & & 0 & & 0 \\ 0 & | & 0 & 0 & 1 & 0 & 1 & 0 & | & 0 & & 0 & & 0 \\ 0 & | & 0 & 1 & 0 & 1 & 0 & 1 & | & 0 & & 0 & & 0 \\ & . & - & - & - & - & - & - & + & - & . & & & \\ 0 & & 1 & 0 & 1 & 0 & 0 & 0 & | & 1 & | & 0 & & 0 \\ & & & & & & & & . & - & + & - & . & \\ 0 & & 0 & 0 & 0 & 1 & 0 & 0 & & 1 & | & 1 & | & 0 \\ & & & & & & & & & & . & - & + & - \\ 0 & & 0 & 1 & 0 & 0 & 0 & 0 & & 0 & & 0 & | & 1 \end{array}\right]$$

Algebraic Loops VI

How do we solve the loop equations?

Algebraic Loops VI

How do we solve the loop equations?

- If the equations are *linear in the unknown variables*, we can use matrix techniques. In the above example:

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix}$$

Algebraic Loops VI

How do we solve the loop equations?

- If the equations are *linear in the unknown variables*, we can use matrix techniques. In the above example:

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix}$$

- If the equations are *non-linear in the unknown variables*, we can use Newton iteration.

Algebraic Loops VI

How do we solve the loop equations?

- If the equations are *linear in the unknown variables*, we can use matrix techniques. In the above example:

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix}$$

- If the equations are *non-linear in the unknown variables*, we can use Newton iteration.

However in either case, we are still dealing with *too many unnecessary loop variables*.

Algebraic Loops VI

How do we solve the loop equations?

- If the equations are *linear in the unknown variables*, we can use matrix techniques. In the above example:

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix}$$

- If the equations are *non-linear in the unknown variables*, we can use Newton iteration.

However in either case, we are still dealing with *too many unnecessary loop variables*.

We shall discuss next, how the number of loop (iteration) variables can be reduced.

The Tearing Algorithm

Let us look more closely at the algebraic loop found earlier. The equations were:

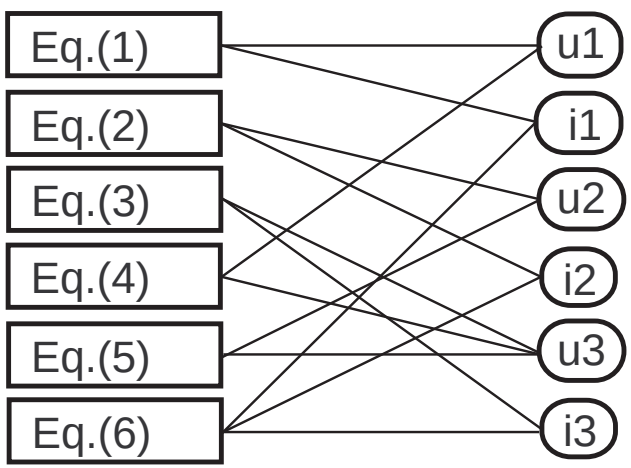
$$
\begin{aligned}
u_1 - R_1 \cdot i_1 &= 0 \\
u_2 - R_2 \cdot i_2 &= 0 \\
u_3 - R_3 \cdot i_3 &= 0 \\
u_1 + u_3 &= u_0 \\
u_2 - u_3 &= 0 \\
i_1 - i_2 - i_3 &= 0
\end{aligned}
$$

The Tearing Algorithm

Let us look more closely at the algebraic loop found earlier. The equations were:

$$
\begin{aligned}
u_1 - R_1 \cdot i_1 &= 0 \\
u_2 - R_2 \cdot i_2 &= 0 \\
u_3 - R_3 \cdot i_3 &= 0 \\
u_1 + u_3 &= u_0 \\
u_2 - u_3 &= 0 \\
i_1 - i_2 - i_3 &= 0
\end{aligned}
$$

with the structure digraph:



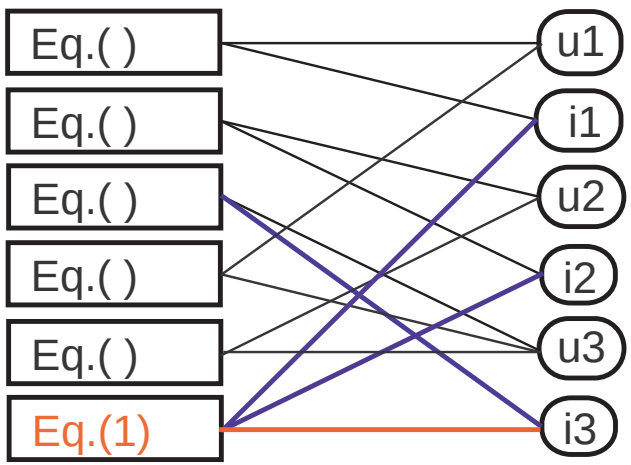
The Tearing Algorithm II

Clearly, every equation contains at least two unknowns, and every unknown appears in at least two equations.

The Tearing Algorithm II

Clearly, every equation contains at least two unknowns, and every unknown appears in at least two equations.

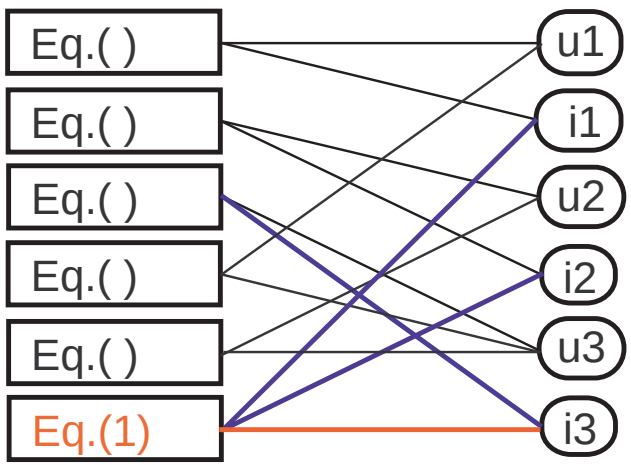
Let me assume, we can solve the former Eq.(6) for the unknown i_3. This assumption is reflected in the partially causalized structure digraph below:



The Tearing Algorithm II

Clearly, every equation contains at least two unknowns, and every unknown appears in at least two equations.

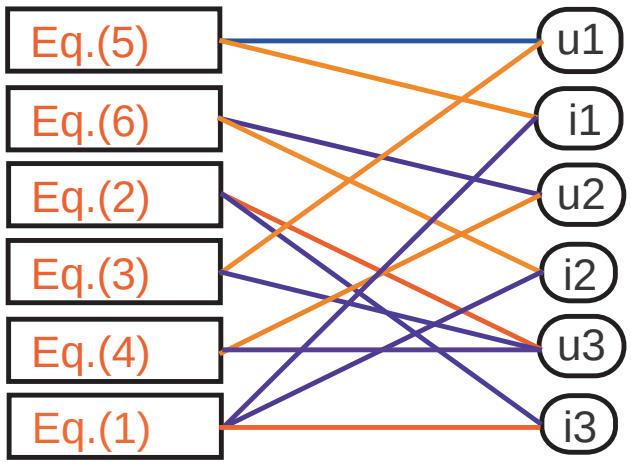
Let me assume, we can solve the former Eq.(6) for the unknown i_3. This assumption is reflected in the partially causalized structure digraph below:



We cannot use that same equation to also compute one of the other two unknowns, i_1 or i_2, and we can also not compute i_3 from the former Eq.(3). Hence the blue lines in the structure digraph.

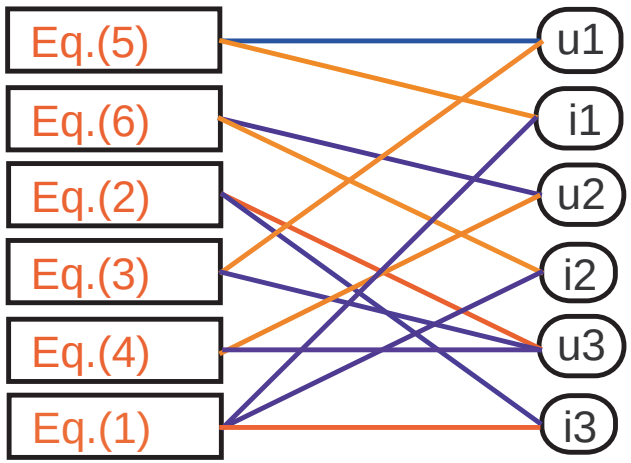
The Tearing Algorithm III

Now, everything is settled, and we can causalize the remaining equations without any difficulties:



The Tearing Algorithm III

Now, everything is settled, and we can causalize the remaining equations without any difficulties:



$$
\begin{aligned}
\Rightarrow \quad i_3 &= i_1 - i_2 \\
u_3 &= R_3 \cdot i_3 \\
u_1 &= u_0 - u_3 \\
u_2 &= u_3 \\
i_1 &= u_1 / R_1 \\
i_2 &= u_2 / R_2
\end{aligned}
$$

The Tearing Algorithm IV

Of course, it is all only a pipe dream, because in reality, we do not know either i_1 or i_2, and therefore, we cannot compute i_3. Or is it not?
Let us substitute the equations into each other, starting with the equation that defines i_3:

$$\begin{aligned} i_3 &= i_1 - i_2 \\ &= \frac{1}{R_1} \cdot u_1 - \frac{1}{R_2} \cdot u_2 \\ &= \frac{1}{R_1} \cdot u_0 - \frac{1}{R_1} \cdot u_3 - \frac{1}{R_2} \cdot u_3 \\ &= \frac{1}{R_1} \cdot u_0 - \frac{R_1 + R_2}{R_1 \cdot R_2} \cdot u_3 \\ &= \frac{1}{R_1} \cdot u_0 - \frac{R_3 \cdot (R_1 + R_2)}{R_1 \cdot R_2} \cdot i_3 \end{aligned}$$

and thus:

$$\left[1 + \frac{R_3 \cdot (R_1 + R_2)}{R_1 \cdot R_2}\right] \cdot i_3 = \frac{1}{R_1} \cdot u_0$$

or:

$$\frac{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3}{R_2} \cdot i_3 = u_0$$

The Tearing Algorithm V

We can solve for i_3:

$$i_3 = \frac{R_2}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0$$

The Tearing Algorithm V

We can solve for i_3:

$$i_3 = \frac{R_2}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0$$

We can plug this equation back into the original equation system, replacing the original Eq.(6) by it, and obtain the perfectly causal set of equations:

$$\begin{aligned}
i_3 &= \frac{R_2}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0 \\
u_3 &= R_3 \cdot i_3 \\
u_1 &= u_0 - u_3 \\
u_2 &= u_3 \\
i_1 &= u_1 / R_1 \\
i_2 &= u_2 / R_2
\end{aligned}$$

The Tearing Algorithm V

We can solve for i_3:

$$i_3 = \frac{R_2}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0$$

We can plug this equation back into the original equation system, replacing the original Eq.(6) by it, and obtain the perfectly causal set of equations:

$$\begin{aligned}
i_3 &= \frac{R_2}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0 \\
u_3 &= R_3 \cdot i_3 \\
u_1 &= u_0 - u_3 \\
u_2 &= u_3 \\
i_1 &= u_1 / R_1 \\
i_2 &= u_2 / R_2
\end{aligned}$$

Evidently, it hadn't been a pipe dream after all.

The Tearing Algorithm VI

- After substituting the equations into each other in the proposed form, we ended up with one equation in one unknown, instead of six equations in six unknowns. This is clearly much more economical.

The Tearing Algorithm VI

- After substituting the equations into each other in the proposed form, we ended up with one equation in one unknown, instead of six equations in six unknowns. This is clearly much more economical.
- We call i_3 in the above example a *tearing variable*, and we call the equation from which the tearing variable is being solved a *residual equation*.

The Tearing Algorithm VI

- After substituting the equations into each other in the proposed form, we ended up with one equation in one unknown, instead of six equations in six unknowns. This is clearly much more economical.
- We call i_3 in the above example a *tearing variable*, and we call the equation from which the tearing variable is being solved a *residual equation*.
- Had the equations been non-linear in the variable i_3, everything would have worked exactly the same way, except for the very last step, where we would have had to involve a Newton iteration to solve for i_3, rather than solving for i_3 explicitly.

The Tearing Algorithm VI

- After substituting the equations into each other in the proposed form, we ended up with one equation in one unknown, instead of six equations in six unknowns. This is clearly much more economical.
- We call i_3 in the above example a *tearing variable*, and we call the equation from which the tearing variable is being solved a *residual equation*.
- Had the equations been non-linear in the variable i_3, everything would have worked exactly the same way, except for the very last step, where we would have had to involve a Newton iteration to solve for i_3, rather than solving for i_3 explicitly.
- Only the tearing variables are included in the set of iteration variables.

The Tearing Algorithm VI

- After substituting the equations into each other in the proposed form, we ended up with one equation in one unknown, instead of six equations in six unknowns. This is clearly much more economical.
- We call i_3 in the above example a *tearing variable*, and we call the equation from which the tearing variable is being solved a *residual equation*.
- Had the equations been non-linear in the variable i_3, everything would have worked exactly the same way, except for the very last step, where we would have had to involve a Newton iteration to solve for i_3, rather than solving for i_3 explicitly.
- Only the tearing variables are included in the set of iteration variables.
- Substituting equations into each other may actually be a bad idea. The substituted equations may grow in size, and the same expressions may appear in them multiple times. It may be a better idea to iterate over the entire set of equations, but treat only i_3 as an iteration variable in the Newton iteration algorithm.

The Tearing Algorithm VII

Given the set of equations:

$$
\begin{aligned}
u_3 &= R_3 \cdot i_3 \\
u_1 &= u_0 - u_3 \\
i_1 &= u_1 / R_1 \\
u_2 &= u_3 \\
i_2 &= u_2 / R_2 \\
i_{3_{new}} &= i_1 - i_2
\end{aligned}
$$

where i_3 is an initial guess, and $i_{3_{new}}$ is an improved version of that same variable.

The Tearing Algorithm VII

Given the set of equations:

$$\begin{aligned} u_3 &= R_3 \cdot i_3 \\ u_1 &= u_0 - u_3 \\ i_1 &= u_1 / R_1 \\ u_2 &= u_3 \\ i_2 &= u_2 / R_2 \\ i_{3_{new}} &= i_1 - i_2 \end{aligned}$$

where i_3 is an initial guess, and $i_{3_{new}}$ is an improved version of that same variable.

We can set up the following zero function:

$$\mathcal{F} = i_{3_{new}} - i_3 = 0.0$$

The Tearing Algorithm VII

Given the set of equations:

$$\begin{aligned} u_3 &= R_3 \cdot i_3 \\ u_1 &= u_0 - u_3 \\ i_1 &= u_1 / R_1 \\ u_2 &= u_3 \\ i_2 &= u_2 / R_2 \\ i_{3_{new}} &= i_1 - i_2 \end{aligned}$$

where i_3 is an initial guess, and $i_{3_{new}}$ is an improved version of that same variable.

We can set up the following zero function:

$$\mathcal{F} = i_{3_{new}} - i_3 = 0.0$$

Since $\mathcal{F}$ is a scalar, also the Hessian is a scalar:

$$\mathcal{H} = \frac{\partial \mathcal{F}}{\partial i_3}$$

The Tearing Algorithm VIII

A convenient way to compute the Hessian $\mathcal{H}$ is by means of *algebraic differentiation*:

$$
\begin{aligned}
du_3 &= R_3 \\
du_1 &= -du_3 \\
di_1 &= du_1 / R_1 \\
du_2 &= du_3 \\
di_2 &= du_2 / R_2 \\
di_{3_{new}} &= di_1 - di_2 \\
\mathcal{H} &= di_{3_{new}} - 1
\end{aligned}
$$

The Tearing Algorithm VIII

A convenient way to compute the Hessian $\mathcal{H}$ is by means of *algebraic differentiation*:

$$\begin{aligned} du_3 &= R_3 \\ du_1 &= -du_3 \\ di_1 &= du_1 / R_1 \\ du_2 &= du_3 \\ di_2 &= du_2 / R_2 \\ di_{3_{new}} &= di_1 - di_2 \\ \mathcal{H} &= di_{3_{new}} - 1 \end{aligned}$$

We can then compute the next version of i_3 as:

$$i_3 = i_3 - \mathcal{H} \backslash \mathcal{F}$$

The Tearing Algorithm VIII

A convenient way to compute the Hessian $\mathcal{H}$ is by means of *algebraic differentiation*:

$$\begin{aligned} du_3 &= R_3 \\ du_1 &= -du_3 \\ di_1 &= du_1 / R_1 \\ du_2 &= du_3 \\ di_2 &= du_2 / R_2 \\ di_{3_{new}} &= di_1 - di_2 \\ \mathcal{H} &= di_{3_{new}} - 1 \end{aligned}$$

We can then compute the next version of i_3 as:

$$i_3 = i_3 - \mathcal{H} \backslash \mathcal{F}$$

If the set of equations is linear, the Newton iteration converges in a single step. Hence it will not be terribly inefficient to employ Newton iteration even in the linear case.

The Tearing Algorithm IX

- The algorithm that we just described is a so-called *tearing algorithm*, as the set of equations is torn apart by making an assumption about one variable or possibly several variables to be known.

The Tearing Algorithm IX

- The algorithm that we just described is a so-called *tearing algorithm*, as the set of equations is torn apart by making an assumption about one variable or possibly several variables to be known.
- The variables that are assumed known, such as i_3 in the above example, are called *tearing variables*.

The Tearing Algorithm IX

- The algorithm that we just described is a so-called *tearing algorithm*, as the set of equations is torn apart by making an assumption about one variable or possibly several variables to be known.
- The variables that are assumed known, such as i_3 in the above example, are called *tearing variables*.
- The equations, from which the tearing variables are to be computed are called *residual equations*.

The Tearing Algorithm IX

- The algorithm that we just described is a so-called *tearing algorithm*, as the set of equations is torn apart by making an assumption about one variable or possibly several variables to be known.
- The variables that are assumed known, such as i_3 in the above example, are called *tearing variables*.
- The equations, from which the tearing variables are to be computed are called *residual equations*.
- A version of tearing similar to the one described in this presentation has been implemented in **Dymola** to accompany the *Tarjan algorithm* used in the efficient solution of algebraic equation systems resulting from the automated symbolic conversion of DAE systems to ODE form. The Tarjan algorithm, a graph-theoretical algorithm, had been introduced in the previous presentation for the causalization of equation systems.

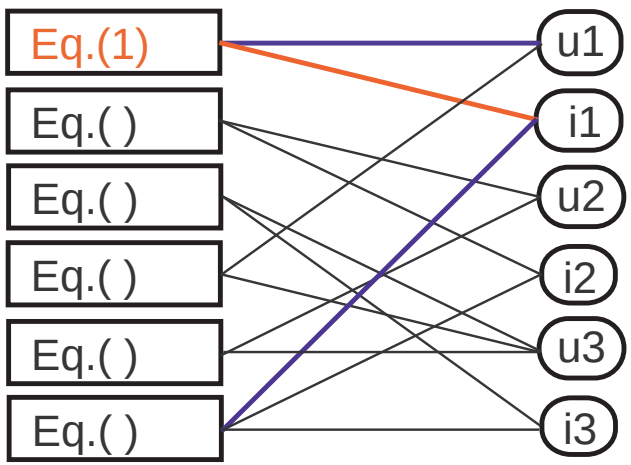
The Tearing Algorithm X

How did we know to choose i_3 as tearing variable and Eq.(6) as residual equation? What would have happened if we had chosen i_1 as the tearing variable and Eq.(1) as the residual equation?

The Tearing Algorithm X

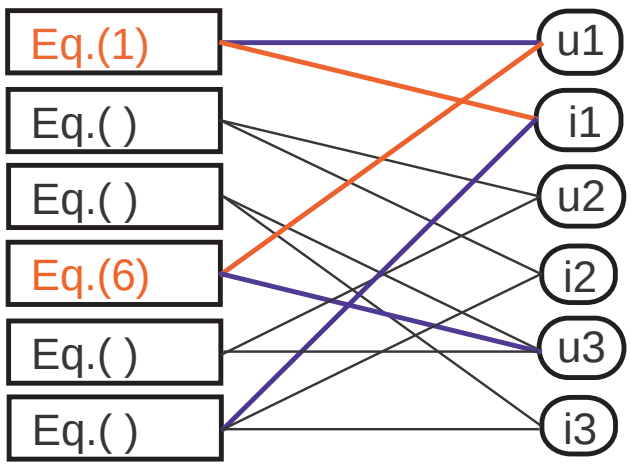
How did we know to choose i_3 as tearing variable and Eq.(6) as residual equation? What would have happened if we had chosen i_1 as the tearing variable and Eq.(1) as the residual equation?

The initial situation is depicted below:



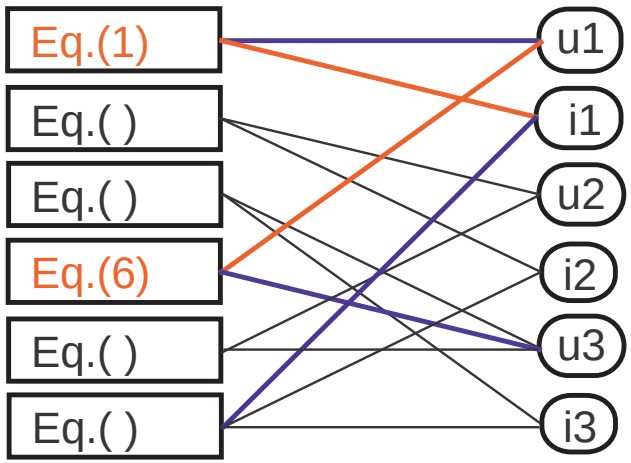
The Tearing Algorithm XI

We apply the Tarjan algorithm to the structure digraph. Unfortunately, the algorithm stalls once again after only one more step:



The Tearing Algorithm XI

We apply the Tarjan algorithm to the structure digraph. Unfortunately, the algorithm stalls once again after only one more step:



Once again, we are faced with an algebraic loop, this time in four equations and four unknowns, and therefore have to choose a second tearing variable and a second residual equation.

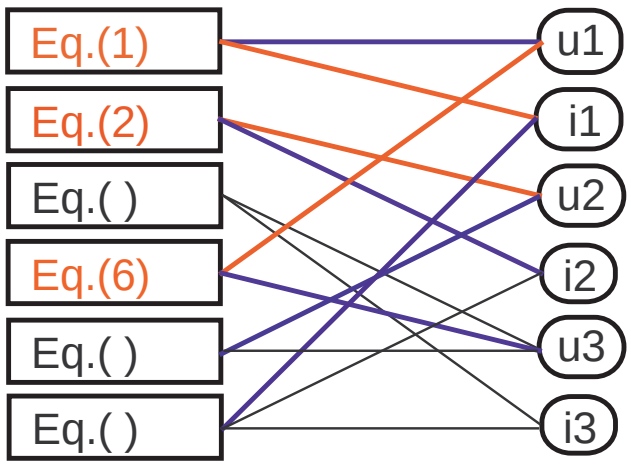
The Tearing Algorithm XII

Let us chose u_2 as the second tearing variable and the former Eq.(2) as the second residual equation.

The Tearing Algorithm XII

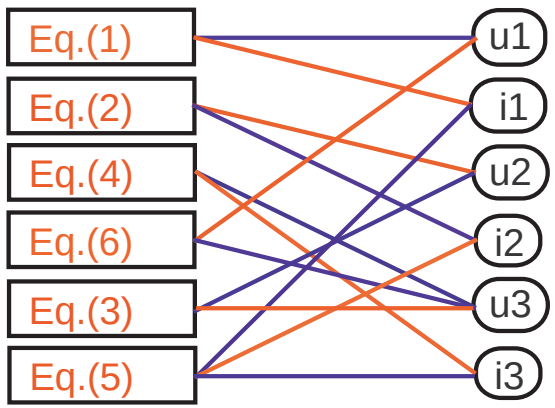
Let us chose u_2 as the second tearing variable and the former Eq.(2) as the second residual equation.

The new situation is depicted below:



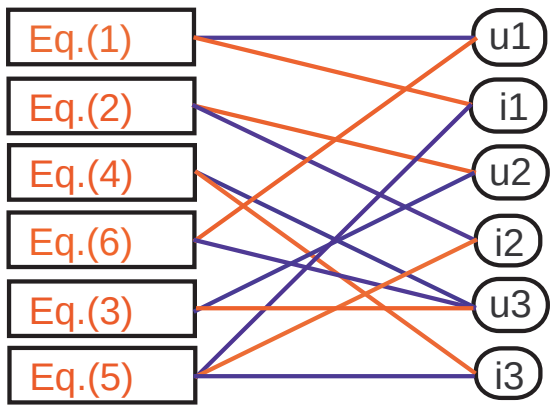
The Tearing Algorithm XIII

We once more apply the Tarjan algorithm to the structure digraph. This time around, we can complete the causalization:



The Tearing Algorithm XIII

We once more apply the Tarjan algorithm to the structure digraph. This time around, we can complete the causalization:



The completely causalized equations are:

$$
\begin{aligned}
i_1 &= u_1 / R_1 \\
u_2 &= R_2 \cdot i_2 \\
u_3 &= u_2 \\
i_3 &= u_3 / R_3 \\
i_2 &= i_1 - i_3 \\
u_1 &= u_0 - u_3
\end{aligned}
$$

The Tearing Algorithm XIV

Let us apply substitution. We begin with the first residual equation. We substitute all variables except for the tearing variables.

$$\begin{aligned} i_1 &= u_1 / R_1 \\ &= \frac{1}{R_1} \cdot u_0 - \frac{1}{R_1} \cdot u_3 \\ &= \frac{1}{R_1} \cdot u_0 - \frac{1}{R_1} \cdot u_2 \end{aligned}$$

and consequently:

$$R_1 \cdot i_1 + u_2 = u_0$$

The Tearing Algorithm XV

For the second residual equation, we find:

$$\begin{aligned} u_2 &= R_2 \cdot i_2 \\ &= R_2 \cdot i_1 - R_2 \cdot i_3 \\ &= R_2 \cdot i_1 - \frac{R_2}{R_3} \cdot u_3 \\ &= R_2 \cdot i_1 - \frac{R_2}{R_3} \cdot u_2 \end{aligned}$$

Thus:

$$\left[1 + \frac{R_2}{R_3}\right] \cdot u_2 = R_2 \cdot i_1$$

or:

$$R_2 \cdot R_3 \cdot i_1 - (R_2 + R_3) \cdot u_2 = 0$$

The Tearing Algorithm XVI

We ended up with two equations in two unknowns, i.e., the two tearing variables:

$$\begin{pmatrix} R_1 & 1 \\ R_2 \cdot R_3 & -(R_2 + R_3) \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \end{pmatrix}$$

which can be solved for i_1 and u_2.

The Tearing Algorithm XVI

We ended up with two equations in two unknowns, i.e., the two tearing variables:

$$\begin{pmatrix} R_1 & 1 \\ R_2 \cdot R_3 & -(R_2 + R_3) \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \end{pmatrix}$$

which can be solved for i_1 and u_2.

- Instead of solving six linear equations in six unknowns, we have "pushed the zeros out of the matrix," and ended up with two equations in two unknowns.

The Tearing Algorithm XVI

We ended up with two equations in two unknowns, i.e., the two tearing variables:

$$\begin{pmatrix} R_1 & 1 \\ R_2 \cdot R_3 & -(R_2 + R_3) \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \end{pmatrix}$$

which can be solved for i_1 and u_2.

- Instead of solving six linear equations in six unknowns, we have "pushed the zeros out of the matrix," and ended up with two equations in two unknowns.
- In this sense, tearing can be considered a *symbolic sparse matrix technique*.

The Tearing Algorithm XVII

If we use Newton iteration instead of equation substitution, we need to place the residual equations at the end of each set, rather than at the beginning. The set of equations now takes the form:

$$
\begin{aligned}
u_3 &= u_2 \\
i_3 &= u_3 / R_3 \\
i_2 &= i_1 - i_3 \\
u_{2_{new}} &= R_2 \cdot i_2 \\
u_1 &= u_0 - u_3 \\
i_{1_{new}} &= u_1 / R_1
\end{aligned}
$$

The Tearing Algorithm XVII

If we use Newton iteration instead of equation substitution, we need to place the residual equations at the end of each set, rather than at the beginning. The set of equations now takes the form:

$$\begin{aligned} u_3 &= u_2 \\ i_3 &= u_3 / R_3 \\ i_2 &= i_1 - i_3 \\ u_{2_{new}} &= R_2 \cdot i_2 \\ u_1 &= u_0 - u_3 \\ i_{1_{new}} &= u_1 / R_1 \end{aligned}$$

We can formulate the following set of zero functions:

$$\mathcal{F} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} i_{1_{new}} - i_1 \\ u_{2_{new}} - u_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The Tearing Algorithm XVII

If we use Newton iteration instead of equation substitution, we need to place the residual equations at the end of each set, rather than at the beginning. The set of equations now takes the form:

$$\begin{aligned} u_3 &= u_2 \\ i_3 &= u_3 / R_3 \\ i_2 &= i_1 - i_3 \\ u_{2_{new}} &= R_2 \cdot i_2 \\ u_1 &= u_0 - u_3 \\ i_{1_{new}} &= u_1 / R_1 \end{aligned}$$

We can formulate the following set of zero functions:

$$\mathcal{F} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} i_{1_{new}} - i_1 \\ u_{2_{new}} - u_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Hence the Hessian is a matrix of size 2×2:

$$\mathcal{H} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} = \begin{pmatrix} \partial f_1/\partial i_1 & \partial f_1/\partial u_2 \\ \partial f_2/\partial i_1 & \partial f_2/\partial u_2 \end{pmatrix}$$

The Tearing Algorithm XVIII

Using algebraic differentiation:

$$
\begin{aligned}
d_1 u_3 &= 0 \\
d_1 i_3 &= d_1 u_3 / R_3 \\
d_1 i_2 &= 1 - d_1 i_3 \\
d_1 u_{2_{new}} &= R_2 \cdot d_1 i_2 \\
d_1 u_1 &= -d_1 u_3 \\
d_1 i_{1_{new}} &= d_1 u_1 / R_1 \\
d_2 u_3 &= 1 \\
d_2 i_3 &= d_2 u_3 / R_3 \\
d_2 i_2 &= -d_2 i_3 \\
d_2 u_{2_{new}} &= R_2 \cdot d_2 i_2 \\
d_2 u_1 &= -d_2 u_3 \\
d_2 i_{1_{new}} &= d_2 u_1 / R_1 \\
h_{11} &= d_1 i_{1_{new}} - 1 \\
h_{12} &= d_2 i_{1_{new}} \\
h_{21} &= d_1 u_{2_{new}} \\
h_{22} &= d_2 u_{2_{new}} - 1
\end{aligned}
$$

The Tearing Algorithm XVIII

Using algebraic differentiation:

$$
\begin{aligned}
d_1 u_3 &= 0 \\
d_1 i_3 &= d_1 u_3 / R_3 \\
d_1 i_2 &= 1 - d_1 i_3 \\
d_1 u_{2_{new}} &= R_2 \cdot d_1 i_2 \\
d_1 u_1 &= -d_1 u_3 \\
d_1 i_{1_{new}} &= d_1 u_1 / R_1 \\
d_2 u_3 &= 1 \\
d_2 i_3 &= d_2 u_3 / R_3 \\
d_2 i_2 &= -d_2 i_3 \\
d_2 u_{2_{new}} &= R_2 \cdot d_2 i_2 \\
d_2 u_1 &= -d_2 u_3 \\
d_2 i_{1_{new}} &= d_2 u_1 / R_1 \\
h_{11} &= d_1 i_{1_{new}} - 1 \\
h_{12} &= d_2 i_{1_{new}} \\
h_{21} &= d_1 u_{2_{new}} \\
h_{22} &= d_2 u_{2_{new}} - 1
\end{aligned}
$$

- The prefix d_1 stands for the partial derivative with respect to the first tearing variable, i_1, and d_2 stands for the partial derivative with respect to the second tearing variable, u_2. Since i_1 and u_2 are mutually independent in this context, the partial derivative of i_1 with respect to u_2 is zero, and vice-versa.

The Tearing Algorithm XVIII

Using algebraic differentiation:

$$
\begin{aligned}
d_1 u_3 &= 0 \\
d_1 i_3 &= d_1 u_3 / R_3 \\
d_1 i_2 &= 1 - d_1 i_3 \\
d_1 u_{2_{new}} &= R_2 \cdot d_1 i_2 \\
d_1 u_1 &= -d_1 u_3 \\
d_1 i_{1_{new}} &= d_1 u_1 / R_1 \\
d_2 u_3 &= 1 \\
d_2 i_3 &= d_2 u_3 / R_3 \\
d_2 i_2 &= -d_2 i_3 \\
d_2 u_{2_{new}} &= R_2 \cdot d_2 i_2 \\
d_2 u_1 &= -d_2 u_3 \\
d_2 i_{1_{new}} &= d_2 u_1 / R_1 \\
h_{11} &= d_1 i_{1_{new}} - 1 \\
h_{12} &= d_2 i_{1_{new}} \\
h_{21} &= d_1 u_{2_{new}} \\
h_{22} &= d_2 u_{2_{new}} - 1
\end{aligned}
$$

- The prefix d_1 stands for the partial derivative with respect to the first tearing variable, i_1, and d_2 stands for the partial derivative with respect to the second tearing variable, u_2. Since i_1 and u_2 are mutually independent in this context, the partial derivative of i_1 with respect to u_2 is zero, and vice-versa.
- For each additional tearing variable, the causal model equations are replicated once in the computation of the Hessian. Hence given a system of n algebraic equations in $k < n$ tearing variables, we require $n \cdot k + k^2$ equations to explicitly compute the Hessian in symbolic form.

The Tearing Algorithm XIX

How can we determine the minimum number of tearing variables required?

The Tearing Algorithm XIX

How can we determine the minimum number of tearing variables required?

- Unfortunately, this is a hard problem. It can be shown that this problem is *np-complete*, i.e., the computational effort grows exponentially in the number of equations forming the algebraic loop. Consequently, finding the minimal number of tearing variables is not practical.

The Tearing Algorithm XIX

How can we determine the minimum number of tearing variables required?

- Unfortunately, this is a hard problem. It can be shown that this problem is *np-complete*, i.e., the computational effort grows exponentially in the number of equations forming the algebraic loop. Consequently, finding the minimal number of tearing variables is not practical.
- Yet, it is possible to design a *heuristic procedure* that always results in a small number of tearing variables. It often results in the minimal number, but this cannot be guaranteed. The advantage of this heuristic procedure is that its computational effort grows quadratically rather than exponentially in the size of the algebraic system for most applications.

The Tearing Algorithm XIX

How can we determine the minimum number of tearing variables required?

- Unfortunately, this is a hard problem. It can be shown that this problem is *np-complete*, i.e., the computational effort grows exponentially in the number of equations forming the algebraic loop. Consequently, finding the minimal number of tearing variables is not practical.
- Yet, it is possible to design a *heuristic procedure* that always results in a small number of tearing variables. It often results in the minimal number, but this cannot be guaranteed. The advantage of this heuristic procedure is that its computational effort grows quadratically rather than exponentially in the size of the algebraic system for most applications.
- **Dymola** implemented a set of heuristics for the efficient selection of tearing variables as part of its model compiler. Yet, their heuristics have never been published. These heuristics are treated as a trade secret, as this is considered to offer a commercial advantage to the company.

The Tearing Algorithm XIX

How can we determine the minimum number of tearing variables required?

- Unfortunately, this is a hard problem. It can be shown that this problem is *np-complete*, i.e., the computational effort grows exponentially in the number of equations forming the algebraic loop. Consequently, finding the minimal number of tearing variables is not practical.
- Yet, it is possible to design a *heuristic procedure* that always results in a small number of tearing variables. It often results in the minimal number, but this cannot be guaranteed. The advantage of this heuristic procedure is that its computational effort grows quadratically rather than exponentially in the size of the algebraic system for most applications.
- **Dymola** implemented a set of heuristics for the efficient selection of tearing variables as part of its model compiler. Yet, their heuristics have never been published. These heuristics are treated as a trade secret, as this is considered to offer a commercial advantage to the company.
- Dirk Zimmer has implemented an alternative set of heuristics in the model compiler of his experimental **Sol** language as part of his Ph.D. dissertation. These heuristics have been disclosed in his dissertation and are thus openly available to all modeling and simulation researchers.

The Tearing Algorithm XX

Here is a very simple set of heuristics that works most of the time but not always. The algorithm sometimes maneuvers itself into a corner.

The Tearing Algorithm XX

Here is a very simple set of heuristics that works most of the time but not always. The algorithm sometimes maneuvers itself into a corner.

1. Using the structure digraph, determine the equations with the largest number of black lines attached to them.

The Tearing Algorithm XX

Here is a very simple set of heuristics that works most of the time but not always. The algorithm sometimes maneuvers itself into a corner.

1. Using the structure digraph, determine the equations with the largest number of black lines attached to them.
2. For every one of these equations, follow its black lines, and determine those variables with the largest number of black lines attached to them.

The Tearing Algorithm XX

Here is a very simple set of heuristics that works most of the time but not always. The algorithm sometimes maneuvers itself into a corner.

1. Using the structure digraph, determine the equations with the largest number of black lines attached to them.
2. For every one of these equations, follow its black lines, and determine those variables with the largest number of black lines attached to them.
3. For every one of these variables, determine how many additional equations can be made causal if that variable is assumed to be known.

The Tearing Algorithm XX

Here is a very simple set of heuristics that works most of the time but not always. The algorithm sometimes maneuvers itself into a corner.

1. Using the structure digraph, determine the equations with the largest number of black lines attached to them.
2. For every one of these equations, follow its black lines, and determine those variables with the largest number of black lines attached to them.
3. For every one of these variables, determine how many additional equations can be made causal if that variable is assumed to be known.
4. Choose one of those variables as the next tearing variable that allows the largest number of additional equations to be made causal.

The Relaxation Algorithm

There is a second symbolic algorithm for the solution of algebraic systems of equations to be discussed that we call *relaxation algorithm*.

The Relaxation Algorithm

There is a second symbolic algorithm for the solution of algebraic systems of equations to be discussed that we call *relaxation algorithm*.

Contrary to the tearing algorithm, a general algorithm that can be applied to all algebraic equation structures, the relaxation algorithm is limited to the solution of *linear algebraic equation systems* only.

The Relaxation Algorithm

There is a second symbolic algorithm for the solution of algebraic systems of equations to be discussed that we call *relaxation algorithm*.

Contrary to the tearing algorithm, a general algorithm that can be applied to all algebraic equation structures, the relaxation algorithm is limited to the solution of *linear algebraic equation systems* only.

Yet, linear algebraic systems assume a special role within the set of algebraic equation systems, and deserve special attention.

The Relaxation Algorithm

There is a second symbolic algorithm for the solution of algebraic systems of equations to be discussed that we call *relaxation algorithm*.

Contrary to the tearing algorithm, a general algorithm that can be applied to all algebraic equation structures, the relaxation algorithm is limited to the solution of *linear algebraic equation systems* only.

Yet, linear algebraic systems assume a special role within the set of algebraic equation systems, and deserve special attention.

Within each Newton iteration of a non-linear algebraic equation system, there is always a linear algebraic equation system to be solved.

The Relaxation Algorithm II

When we write the Newton iteration as:

$$\mathbf{x}_{\mathbf{new}} = \mathbf{x}_{\mathbf{old}} - \mathcal{H} \backslash \mathcal{F}$$

we are effectively saying that:

$$\mathbf{x}_{\mathbf{new}} = \mathbf{x}_{\mathbf{old}} - \mathbf{dx}$$

where **dx** is the solution of the linear algebraic equation system:

$$\mathcal{H} \cdot \mathbf{dx} = \mathcal{F}$$

The Relaxation Algorithm II

When we write the Newton iteration as:

$$\mathbf{x_{new}} = \mathbf{x_{old}} - \mathcal{H} \backslash \mathcal{F}$$

we are effectively saying that:

$$\mathbf{x_{new}} = \mathbf{x_{old}} - \mathbf{dx}$$

where **dx** is the solution of the linear algebraic equation system:

$$\mathcal{H} \cdot \mathbf{dx} = \mathcal{F}$$

Hence indeed, a linear algebraic equation system must be solved within each Newton iteration step of the original non-linear algebraic equation system.

The Relaxation Algorithm III

Relaxation is a symbolic implementation of the Gaussian elimination algorithm without pivoting.

The Relaxation Algorithm III

Relaxation is a symbolic implementation of the Gaussian elimination algorithm without pivoting.

Let us demonstrate how the relaxation algorithm works by means of the same example of a linear algebraic equation system in six equations and six unknowns that we had used previously:

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix}$$

The Relaxation Algorithm III

Relaxation is a symbolic implementation of the Gaussian elimination algorithm without pivoting.

Let us demonstrate how the relaxation algorithm works by means of the same example of a linear algebraic equation system in six equations and six unknowns that we had used previously:

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix}$$

It is our goal to minimize the number of non-zero elements in the matrix above the diagonal.

The Relaxation Algorithm IV

To this end, we sort the equations in the same way as we did for the tearing algorithm with Newton iteration:

$$
\begin{aligned}
u_3 &= R_3 \cdot i_3 \\
u_1 &= u_0 - u_3 \\
i_1 &= \frac{u_1}{R_1} \\
u_2 &= u_3 \\
i_2 &= \frac{u_2}{R_2} \\
i_3 &= i_1 - i_2
\end{aligned}
$$

The Relaxation Algorithm IV

To this end, we sort the equations in the same way as we did for the tearing algorithm with Newton iteration:

$$
\begin{aligned}
u_3 &= R_3 \cdot i_3 \\
u_1 &= u_0 - u_3 \\
i_1 &= \frac{u_1}{R_1} \\
u_2 &= u_3 \\
i_2 &= \frac{u_2}{R_2} \\
i_3 &= i_1 - i_2
\end{aligned}
$$

We now move all the unknowns to the left side of the equal sign and all the knows to the right side. At the same time, we eliminate the denominators:

$$
\begin{aligned}
u_3 - R_3 \cdot i_3 &= 0 \\
u_1 + u_3 &= u_0 \\
R_1 \cdot i_1 - u_1 &= 0 \\
u_2 - u_3 &= 0 \\
R_2 \cdot i_2 - u_2 &= 0 \\
i_3 - i_1 + i_2 &= 0
\end{aligned}
$$

The Relaxation Algorithm V

We now rewrite these equations in a matrix-vector form, whereby we number the equations in the same order as above and list the variables in the same order as in the causal equations:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -R_3 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & R_1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & R_2 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The Relaxation Algorithm V

We now rewrite these equations in a matrix-vector form, whereby we number the equations in the same order as above and list the variables in the same order as in the causal equations:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -R_3 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & R_1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & R_2 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

There is now only a single non-zero element above the diagonal, and none of the diagonal elements are zero.

The Relaxation Algorithm VI

We can now apply Gaussian elimination without pivoting:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot A_{kj}^{(n)}$$
$$b_{i}^{(n+1)} = b_{i}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot b_{k}^{(n)}$$

The Relaxation Algorithm VI

We can now apply Gaussian elimination without pivoting:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot A_{kj}^{(n)}$$
$$b_{i}^{(n+1)} = b_{i}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot b_{k}^{(n)}$$

We can apply this algorithm symbolically. After each step, we eliminate the first row and the first column, i.e., the pivot row and the pivot column. Rather than substituting expressions into the matrix, we introduce auxiliary variables where needed.

The Relaxation Algorithm VI

We can now apply Gaussian elimination without pivoting:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot A_{kj}^{(n)}$$
$$b_{i}^{(n+1)} = b_{i}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot b_{k}^{(n)}$$

We can apply this algorithm symbolically. After each step, we eliminate the first row and the first column, i.e., the pivot row and the pivot column. Rather than substituting expressions into the matrix, we introduce auxiliary variables where needed.

- If an element in the top row is zero, the elements underneath it don't change at all during the iteration.

The Relaxation Algorithm VI

We can now apply Gaussian elimination without pivoting:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot A_{kj}^{(n)}$$
$$b_{i}^{(n+1)} = b_{i}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot b_{k}^{(n)}$$

We can apply this algorithm symbolically. After each step, we eliminate the first row and the first column, i.e., the pivot row and the pivot column. Rather than substituting expressions into the matrix, we introduce auxiliary variables where needed.

- If an element in the top row is zero, the elements underneath it don't change at all during the iteration.
- If an element in the leftmost column is zero, the elements to the right of it don't change.

The Relaxation Algorithm VI

We can now apply Gaussian elimination without pivoting:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot A_{kj}^{(n)}$$
$$b_{i}^{(n+1)} = b_{i}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot b_{k}^{(n)}$$

We can apply this algorithm symbolically. After each step, we eliminate the first row and the first column, i.e., the pivot row and the pivot column. Rather than substituting expressions into the matrix, we introduce auxiliary variables where needed.

- If an element in the top row is zero, the elements underneath it don't change at all during the iteration.
- If an element in the leftmost column is zero, the elements to the right of it don't change.
- For all other elements, we introduce new symbolic constants.

The Relaxation Algorithm VII

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -R_3 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & R_1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & R_2 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The Relaxation Algorithm VII

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -R_3 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & R_1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & R_2 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & c_1 \\ -1 & R_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & c_2 \\ 0 & 0 & -1 & R_2 & 0 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

where $c_1 = R_3$, and $c_2 = -R_3$.

The Relaxation Algorithm VIII

$$\begin{pmatrix} 1 & 0 & 0 & 0 & c_1 \\ -1 & R_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & c_2 \\ 0 & 0 & -1 & R_2 & 0 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The Relaxation Algorithm VIII

$$\begin{pmatrix} 1 & 0 & 0 & 0 & c_1 \\ -1 & R_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & c_2 \\ 0 & 0 & -1 & R_2 & 0 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} R_1 & 0 & 0 & c_3 \\ 0 & 1 & 0 & c_2 \\ 0 & -1 & R_2 & 0 \\ -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} c_4 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

where $c_3 = c_1$, and $c_4 = u_0$.

The Relaxation Algorithm IX

$$\begin{pmatrix} R_1 & 0 & 0 & c_3 \\ 0 & 1 & 0 & c_2 \\ 0 & -1 & R_2 & 0 \\ -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} c_4 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The Relaxation Algorithm IX

$$\begin{pmatrix} R_1 & 0 & 0 & c_3 \\ 0 & 1 & 0 & c_2 \\ 0 & -1 & R_2 & 0 \\ -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} c_4 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} 1 & 0 & c_2 \\ -1 & R_2 & 0 \\ 0 & 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ c_6 \end{pmatrix}$$

where $c_5 = 1 + \frac{c_3}{R_1}$, and $c_6 = \frac{c_4}{R_1}$.

The Relaxation Algorithm X

$$\begin{pmatrix} 1 & 0 & c_2 \\ -1 & R_2 & 0 \\ 0 & 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ c_6 \end{pmatrix}$$

The Relaxation Algorithm X

$$\begin{pmatrix} 1 & 0 & c_2 \\ -1 & R_2 & 0 \\ 0 & 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ c_6 \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} R_2 & c_7 \\ 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ c_6 \end{pmatrix}$$

where $c_7 = c_2$.

The Relaxation Algorithm X

$$\begin{pmatrix} 1 & 0 & c_2 \\ -1 & R_2 & 0 \\ 0 & 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ c_6 \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} R_2 & c_7 \\ 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ c_6 \end{pmatrix}$$

where $c_7 = c_2$.

$$\Rightarrow \quad (c_8) \cdot (i_3) = (c_6)$$

where $c_8 = c_5 - \frac{c_7}{R_2}$.

The Relaxation Algorithm XI

Hence we can replace the original set of equations by:

$$
\begin{aligned}
c_1 &= R_3 \\
c_2 &= -R_3 \\
c_3 &= c_1 \\
c_4 &= u_0 \\
c_5 &= 1 + \frac{c_3}{R_1} \\
c_6 &= \frac{c_4}{R_1} \\
c_7 &= c_2 \\
c_8 &= c_5 - \frac{c_7}{R_2}
\end{aligned}
$$

The Relaxation Algorithm XI

Hence we can replace the original set of equations by:

$$
\begin{aligned}
c_1 &= R_3 \\
c_2 &= -R_3 \\
c_3 &= c_1 \\
c_4 &= u_0 \\
c_5 &= 1 + \frac{c_3}{R_1} \\
c_6 &= \frac{c_4}{R_1} \\
c_7 &= c_2 \\
c_8 &= c_5 - \frac{c_7}{R_2}
\end{aligned}
$$

$$
\begin{aligned}
i_3 &= \frac{c_6}{c_8} \\
i_2 &= -\frac{c_7 \cdot i_3}{R_2} \\
u_2 &= -c_2 \cdot i_3 \\
i_1 &= \frac{c_4 - c_3 \cdot i_3}{R_1} \\
u_1 &= u_0 - c_1 \cdot i_3 \\
u_3 &= R_3 \cdot i_3
\end{aligned}
$$

The Relaxation Algorithm XI

Hence we can replace the original set of equations by:

$$
\begin{aligned}
c_1 &= R_3 \\
c_2 &= -R_3 \\
c_3 &= c_1 \\
c_4 &= u_0 \\
c_5 &= 1 + \frac{c_3}{R_1} \\
c_6 &= \frac{c_4}{R_1} \\
c_7 &= c_2 \\
c_8 &= c_5 - \frac{c_7}{R_2}
\end{aligned}
\qquad
\begin{aligned}
i_3 &= \frac{c_6}{c_8} \\
i_2 &= -\frac{c_7 \cdot i_3}{R_2} \\
u_2 &= -c_2 \cdot i_3 \\
i_1 &= \frac{c_4 - c_3 \cdot i_3}{R_1} \\
u_1 &= u_0 - c_1 \cdot i_3 \\
u_3 &= R_3 \cdot i_3
\end{aligned}
$$

which can be easily coded in **Matlab**.

The Relaxation Algorithm XII

However, there is no need to proceed with back-substitution beyond the determination of the tearing variables. The remaining variables can be taken just as easily from the original set of equations:

$$
\begin{aligned}
c_1 &= R_3 \\
c_2 &= -R_3 \\
c_3 &= c_1 \\
c_4 &= u_0 \\
c_5 &= 1 + \frac{c_3}{R_1} \\
c_6 &= \frac{c_4}{R_1} \\
c_7 &= c_2 \\
c_8 &= c_5 - \frac{c_7}{R_2}
\end{aligned}
$$

The Relaxation Algorithm XII

However, there is no need to proceed with back-substitution beyond the determination of the tearing variables. The remaining variables can be taken just as easily from the original set of equations:

$$
\begin{aligned}
c_1 &= R_3 \\
c_2 &= -R_3 \\
c_3 &= c_1 \\
c_4 &= u_0 \\
c_5 &= 1 + \frac{c_3}{R_1} \\
c_6 &= \frac{c_4}{R_1} \\
c_7 &= c_2 \\
c_8 &= c_5 - \frac{c_7}{R_2}
\end{aligned}
$$

$$
\begin{aligned}
i_3 &= \frac{c_6}{c_8} \\
u_3 &= R_3 \cdot i_3 \\
u_1 &= u_0 - u_3 \\
i_1 &= \frac{u_1}{R_1} \\
u_2 &= u_3 \\
i_2 &= \frac{u_2}{R_2}
\end{aligned}
$$

The Relaxation Algorithm XIII

What would have happened if we had started out with the second set of causal equations, i.e., those involving two tearing variables?

$$
\begin{aligned}
u_3 &= u_2 \\
i_3 &= \frac{u_3}{R_3} \\
i_2 &= i_1 - i_3 \\
u_2 &= R_2 \cdot i_2 \\
u_1 &= u_0 - u_3 \\
i_1 &= \frac{u_1}{R_1}
\end{aligned}
$$

The Relaxation Algorithm XIII

What would have happened if we had started out with the second set of causal equations, i.e., those involving two tearing variables?

$$
\begin{aligned}
u_3 &= u_2 \\
i_3 &= \frac{u_3}{R_3} \\
i_2 &= i_1 - i_3 \\
u_2 &= R_2 \cdot i_2 \\
u_1 &= u_0 - u_3 \\
i_1 &= \frac{u_1}{R_1}
\end{aligned}
$$

Moving all unknowns to the left side of the equal sign, we obtain:

$$
\begin{aligned}
u_3 - u_2 &= 0 \\
R_3 \cdot i_3 - u_3 &= 0 \\
i_2 - i_1 + i_3 &= 0 \\
u_2 - R_2 \cdot i_2 &= 0 \\
u_1 + u_3 &= u_0 \\
R_1 \cdot i_1 - u_1 &= 0
\end{aligned}
$$

The Relaxation Algorithm XIV

In matrix-vector form:

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ -1 & R_3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & -R_2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ i_3 \\ i_2 \\ u_2 \\ u_1 \\ i_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ u_0 \\ 0 \end{pmatrix}$$

The Relaxation Algorithm XIV

In matrix-vector form:

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ -1 & R_3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & -R_2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ i_3 \\ i_2 \\ u_2 \\ u_1 \\ i_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ u_0 \\ 0 \end{pmatrix}$$

- This time around, there are two non-zero elements above the diagonal, one involving the tearing variable u_2, the other involving the tearing variable i_1.

The Relaxation Algorithm XIV

In matrix-vector form:

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ -1 & R_3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & -R_2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ i_3 \\ i_2 \\ u_2 \\ u_1 \\ i_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ u_0 \\ 0 \end{pmatrix}$$

- This time around, there are two non-zero elements above the diagonal, one involving the tearing variable u_2, the other involving the tearing variable i_1.
- As there are more non-zero elements above the diagonal, we shall in all likelihood have to introduce more new constants into the matrix in the process of symbolic Gaussian elimination.

The Relaxation Algorithm XIV

In matrix-vector form:

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ -1 & R_3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & -R_2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ i_3 \\ i_2 \\ u_2 \\ u_1 \\ i_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ u_0 \\ 0 \end{pmatrix}$$

- This time around, there are two non-zero elements above the diagonal, one involving the tearing variable u_2, the other involving the tearing variable i_1.
- As there are more non-zero elements above the diagonal, we shall in all likelihood have to introduce more new constants into the matrix in the process of symbolic Gaussian elimination.
- Consequently, the number of resulting equations will grow, and the solution will be less efficient.

The Relaxation Algorithm XV

- Finding a minimal set of non-zero elements above the diagonal of the matrix is identical to finding a minimal set of tearing variables.

The Relaxation Algorithm XV

- Finding a minimal set of non-zero elements above the diagonal of the matrix is identical to finding a minimal set of tearing variables.
- Hence also this problem is *np-complete*.

The Relaxation Algorithm XV

- Finding a minimal set of non-zero elements above the diagonal of the matrix is identical to finding a minimal set of tearing variables.
- Hence also this problem is *np-complete*.
- The same heuristic procedure that was proposed for tackling the problem of finding a small (though not necessarily the minimal) set of tearing variables can also be used to find a small (though not necessarily the smallest) set of non-zero elements above the diagonal of the linear equation matrix for the relaxation algorithm.

Conclusions

- In this presentation, we looked at the problem of *algebraic loops* contained in the set of DAEs extracted from an object-oriented description of the system to be simulated.

Conclusions

- In this presentation, we looked at the problem of *algebraic loops* contained in the set of DAEs extracted from an object-oriented description of the system to be simulated.
- We demonstrated that the *Tarjan algorithm* for the causalization of the equations will get stuck in this case. The equations forming the loop need to be solved together.

Conclusions

- In this presentation, we looked at the problem of *algebraic loops* contained in the set of DAEs extracted from an object-oriented description of the system to be simulated.
- We demonstrated that the *Tarjan algorithm* for the causalization of the equations will get stuck in this case. The equations forming the loop need to be solved together.
- We then introduced two separate *symbolic formula manipulation algorithms* that can be used for "squeezing the zeros out of the loop equations." These are two different *symbolic sparse system solvers*.

Conclusions

- In this presentation, we looked at the problem of *algebraic loops* contained in the set of DAEs extracted from an object-oriented description of the system to be simulated.
- We demonstrated that the *Tarjan algorithm* for the causalization of the equations will get stuck in this case. The equations forming the loop need to be solved together.
- We then introduced two separate *symbolic formula manipulation algorithms* that can be used for "squeezing the zeros out of the loop equations." These are two different *symbolic sparse system solvers*.
- One of these algorithms, the *relaxation algorithm*, can only be applied in the case of linear systems, whereas the other, the *tearing algorithm*, is applicable also for non-linear systems.

Conclusions

- In this presentation, we looked at the problem of *algebraic loops* contained in the set of DAEs extracted from an object-oriented description of the system to be simulated.
- We demonstrated that the *Tarjan algorithm* for the causalization of the equations will get stuck in this case. The equations forming the loop need to be solved together.
- We then introduced two separate *symbolic formula manipulation algorithms* that can be used for "squeezing the zeros out of the loop equations." These are two different *symbolic sparse system solvers*.
- One of these algorithms, the *relaxation algorithm*, can only be applied in the case of linear systems, whereas the other, the *tearing algorithm*, is applicable also for non-linear systems.
- The relaxation algorithm is more efficient in the case of linear systems, and for this reason, it has its place. Yet due to the greater generality of the tearing approach, **Dymola** and **Sol** only make use of tearing.

References

1. Otter, M., H. Elmqvist, and F.E. Cellier (1996), " 'Relaxing' - A Symbolic Sparse Matrix Method Exploiting the Model Structure in Generating Efficient Simulation Code," *Proc. Symposium on Modelling, Analysis, and Simulation, CESA'96, IMACS MultiConference on Computational Engineering in Systems Applications*, Lille, France, vol.1, pp.1-12.

2. Zimmer, Dirk (2010), *Equation-based Modeling of Variable-structure Systems*, Ph.D. Dissertation, Dept. of Computer Science, ETH Zurich, Switzerland.