7

Differential Algebraic Equations

Preview

In this chapter, we shall analyze simulation problems that don't present themselves initially in an explicit state–space form. For many physical systems, it is quite easy to formulate a model where the state derivatives show up implicitly and possibly even in a nonlinear fashion anywhere within the equations. We call system descriptions that consist of a mixture of implicitly formulated algebraic and differential equations *Differential Algebraic Equations (DAEs).* Since these cases constitute a substantial and important portion of the models encountered in science and engineering, they deserve our attention. In this chapter, we shall discuss the question, how sets of DAEs can be converted symbolically in an automated fashion to equivalent sets of ODEs.

7.1 Introduction

In the companion book *Continuous System Modeling* [7.5], we have demonstrated that object–oriented modeling of physical systems invariably leads to implicit DAE descriptions. Some of these can be converted to ODE descriptions quite easily by simple sorting algorithms, whereas others contain big *algebraic loops* or even *structural singularities.*

We shall now revisit these issues and present a set of symbolic formulae manipulation algorithms that allow us to convert implicit DAE descriptions to equivalent explicit ODE descriptions.

Let us once again begin with a simple electrical RLC circuit. Its schematic is shown in Fig.7.1.

As there are five circuit elements defining two variables each, namely the voltage across and the current through that element, we need 10 equations to describe the model, e.g. the five element equations, defining the relation between voltage across and current through the element, plus three mesh equations in the mesh voltages, plus two node equations in the node currents. A possible set of equations is:

$$u_0 = f(t) \tag{7.1a}$$

$$u_1 = R_1 \cdot i_1 \tag{7.1b}$$

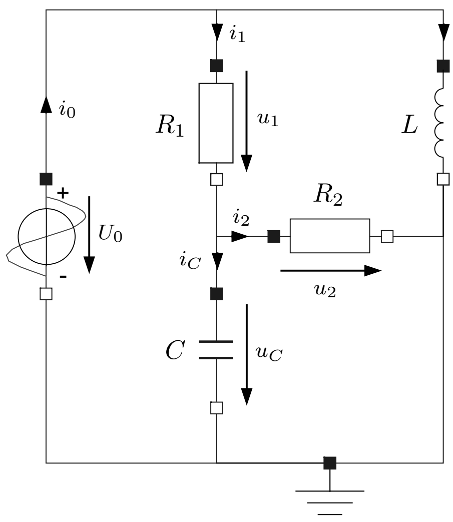

FIGURE 7.1. Schematic of electrical RLC circuit.

$$u_2 = R_2 \cdot i_2 \tag{7.1c}$$

$$u_L = L \cdot \frac{di_L}{dt} \tag{7.1d}$$

$$i_C = C \cdot \frac{du_C}{dt} \tag{7.1e}$$

$$u_0 = u_1 + u_C \tag{7.1f}$$

$$u_L = u_1 + u_2 \tag{7.1g}$$

$$u_C = u_2 \tag{7.1h}$$

$$i_0 = i_1 + i_L \tag{7.1i}$$

$$i_1 = i_2 + i_C \tag{7.1j}$$

As we wish to generate a state–space model, we define the outputs of the integrators, u_C and i_L, as our state variables. These can thus be considered known variables, for which no equations need to be found. In contrast, the inputs of the integrators, du_C/dt and di_L/dt, are unknowns, for which equations must be found. These are the state equations of the state–space description.

The structure of these equations can be captured in the so–called *structure incidence matrix*. The structure incidence matrix lists the equations in any order as rows, and the unknowns in any order as columns. If the i^{th} equation contains the j^{th} variable, the element $< i, j >$ of the structure incidence matrix assumes a value of 1, otherwise it is set to 0. The structure incidence matrix for the above set of equations could e.g. be written as:

$$\mathbf{S} = \begin{array}{r} \\ Eq.(7.1a) \\ Eq.(7.1b) \\ Eq.(7.1c) \\ Eq.(7.1d) \\ Eq.(7.1e) \\ Eq.(7.1f) \\ Eq.(7.1g) \\ Eq.(7.1h) \\ Eq.(7.1i) \\ Eq.(7.1j) \end{array} \begin{array}{c} \begin{array}{cccccccccc} u_0 & i_0 & u_1 & i_1 & u_2 & i_2 & u_L & \frac{di_L}{dt} & \frac{du_C}{dt} & i_C \end{array} \\ \left(\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array} \right) \end{array} \tag{7.2}$$

Initially, all of these equations are *acausal*, meaning that the equal sign has to be interpreted in the sense of an equality, rather than in the sense of an assignment. For example, the above set of equations contains two equations that list u_0 to the left of the equal sign. Evidently, only one of those can be used to solve for u_0.

Two simple rules can be formulated that help us decide, which variables to solve for from which of the equations:

1. If an equation contains only a single unknown, i.e., one variable for which no solving equation has been found yet, we need to use that equation to solve for this variable. For example, Eq.(7.1a) contains only one unknown, u_0, hence that equation must be used to solve for u_0, and consequently, Eq.(7.1a) has now become a *causal equation*, and u_0 can henceforth be considered a known variable in all remaining equations.

2. If an unknown only appears in a single equation, that equation must be used to solve for it. For example, i_0 only appears in Eq.(7.1i). Hence we must use Eq.(7.1i) to solve for i_0.

These rules can be easily visualized in the structure incidence matrix. If a row contains a single element with a value of 1, that equation needs to be solved for the corresponding variable, and both the row and the column can be eliminated from the structure incidence matrix. If a column contains a single element with a value of 1, that variable must be solved for using the corresponding equation. Once again, both the column and the row can be eliminated from the structure incidence matrix. The algorithm proceeds iteratively, until no more rows and columns can be eliminated from the structure incidence matrix.

7.2 Causalization of Equations

Although the algorithm based on the structure incidence matrix will work, another algorithm has become more popular in recent years that is based on graph theory. It is an algorithm proposed first by Tarjan[1] [7.21]. Rather than capturing the structure of the set of DAEs in the form of a structure incidence matrix, it captures the same information in a graphical data structure, called the *structure digraph.*

The structure digraph depicts on the left hand side the equations as a column of nodes. On the right hand side, the unknowns are displayed also as a column of nodes. Since the number of equations must always equal the number of unknowns, the two column vectors are of equal length. A straight line connects an equation with an unknown, if that unknown appears in the equation.

The structure digraph of the above set of equations could be drawn e.g. as shown in Fig.7.2.

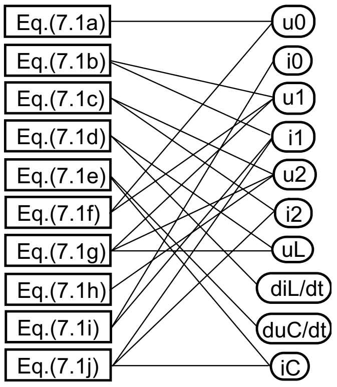

FIGURE 7.2. Structure digraph of electrical circuit.

We now implement our two rules for selecting which variable is to be solved for from which equation using the structure digraph.

When we select a variable to be solved for using a given equation, we color the connection between the equation and the variable in "red." Since this book is printed in black and white, we shall simulate the coloring by dashing it.

When we declare that a previously unknown variable is now known, because we already found an equation to solve for it, or because the equation,

[1]The algorithm presented in this section is not exactly the one originally proposed by Tarjan, but rather a somewhat modified algorithm, applied furthermore in a different context. Tarjan, in his original article, did not concern himself at all with the causalization of equations, but rather with detecting loops in a directed graph.

in which it occurs, is being used to solve for another variable, we color that connection in "blue." In this book, we shall simulate the coloring by dotting it.

A causal equation is an equation that has exactly one red (dashed) line attached to it. Acausal equations are equations that have only black (solid) and blue (dotted) lines attached to them. Known variables are variables that have exactly one red (dashed) line ending in them. An unknown variable has only black (solid) and blue (dotted) lines attached to it. No equation or variable has ever more than one red (dashed) line connecting to it.

We are now ready to implement our two rules.

1. For all acausal equations, if an equation has only one black (solid) line attached to it, color that line red (dash it), follow it to the variable it points at, and color all other connections ending in that variable in blue (dot the connections). Renumber the equation using the lowest free number starting from 1.

2. For all unknown variables, if a variable has only one black (solid) line attached to it, color that line red (dash it), follow it back to the equation it points at, and color all other connections emanating from that equation in blue (dot the connections). Renumber the equation using the highest free number starting from n, where n is the number of equations.

Figure 7.3 shows the structure digraph of the electrical circuit after one iteration through these two rules.

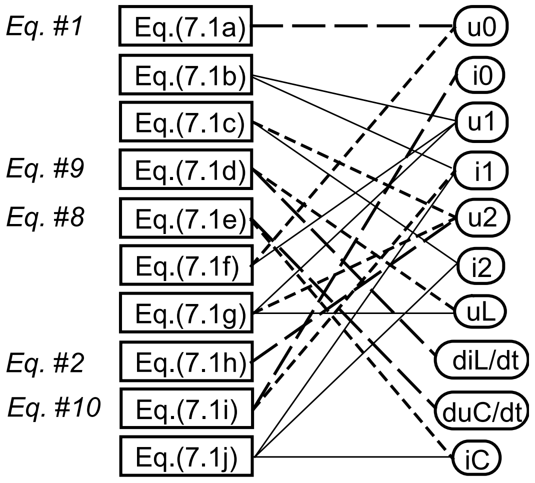

FIGURE 7.3. Structure digraph of electrical circuit after partial coloring.

In the first iteration, five of the 10 equations were made causal, two using rule #1, and three using rule #2. However, the algorithm doesn't end here, since these rules can be applied recursively.

Figure 7.4 shows the structure digraph of the electrical circuit after all equations have been made causal.

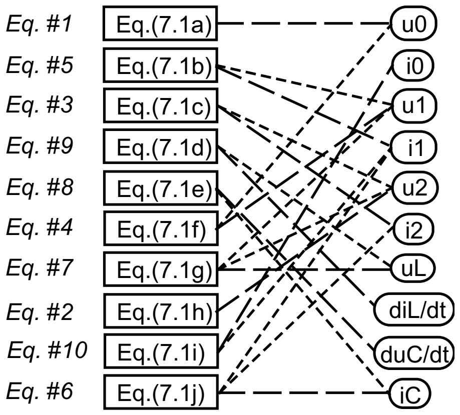

FIGURE 7.4. Structure digraph of electrical circuit after complete coloring.

We were able to complete the causalization of the equations. We can now read out the 10 equations in their causal form.

$$u_0 = f(t) \tag{7.3a}$$

$$u_2 = u_C \tag{7.3b}$$

$$i_2 = u_2/R_2 \tag{7.3c}$$

$$u_1 = u_0 - u_C \tag{7.3d}$$

$$i_1 = u_1/R_1 \tag{7.3e}$$

$$i_C = i_1 - i_2 \tag{7.3f}$$

$$u_L = u_1 + u_2 \tag{7.3g}$$

$$\frac{du_C}{dt} = i_C/C \tag{7.3h}$$

$$\frac{di_L}{dt} = u_L/L \tag{7.3i}$$

$$i_0 = i_1 + i_L \tag{7.3j}$$

By now, the equal signs have become true assignments. In addition, no variable is being used, before it has been computed. Consequently, the equations have not only been *sorted horizontally*, i.e., made causal. They have also been *sorted vertically*, i.e., the equations have been sorted into an executable sequence.

Let us write down the structure incidence matrix of the horizontally and vertically sorted set of equations.

$$\mathbf{S} = \begin{array}{r} \\ Eq.(7.3a) \\ Eq.(7.3b) \\ Eq.(7.3c) \\ Eq.(7.3d) \\ Eq.(7.3e) \\ Eq.(7.3f) \\ Eq.(7.3g) \\ Eq.(7.3h) \\ Eq.(7.3i) \\ Eq.(7.3j) \end{array} \begin{array}{cccccccccc} u_0 & u_2 & i_2 & u_1 & i_1 & i_C & u_L & \frac{du_C}{dt} & \frac{di_L}{dt} & i_0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \tag{7.4}$$

The structure incidence matrix of the sorted set of equations is now in lower–triangular form. Hence sorting the equations both horizontally and vertically is identical to finding permutation matrices that reduce the structure incidence matrix to lower–triangular form.

There exist algorithms to find these permutation matrices directly. This would be yet another approach to sorting the equations.

Variants of the Tarjan algorithm have become the most popular among all the available sorting algorithms for their efficiency, as their *computational effort* grows linearly with the size of the DAE system[1]. This is the best performance that can be expected of any algorithm.

A variant of the causalization algorithm, called *output set assignment*, can be found in a paper by Pantelides [7.20], who presented the algorithm once again in a somewhat different context. The Pantelides variant of the causalization algorithm has become the most popular of these algorithms, as it has the advantage that it can be implemented using a very compact and elegant recursive procedure.

7.3 Algebraic Loops

The previous section may leave the impression with you, the reader, that all DAE systems can be sorted as easily as the example system, by means of which the causalization algorithm has been demonstrated. Nothing could be farther from the truth.

Let us now look at a slightly modified circuit. Its schematic is shown in Fig.7.5. The capacitor has been replaced by a third resistor.

[1]It was shown in [7.6] that the computational complexity of the Tarjan algorithm grows in the worst case with $o(n \cdot m)$, where n is the number of equations, and m is the number of non–zero elements in the structure incidence matrix.

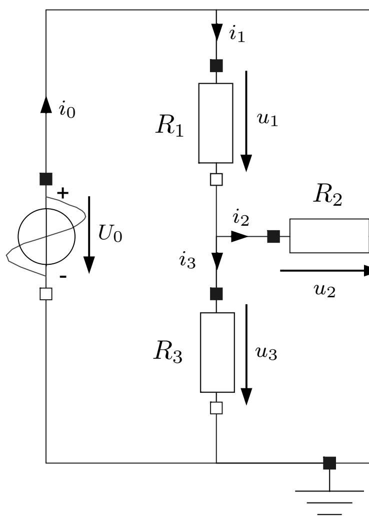

FIGURE 7.5. Schematic of modified electrical RLC circuit.

The resulting equations are almost the same as before. Only the element equation for the capacitor was replaced by a third element equation for a resistor.

$$u_0 = f(t) \tag{7.5a}$$
$$u_1 = R_1 \cdot i_1 \tag{7.5b}$$
$$u_2 = R_2 \cdot i_2 \tag{7.5c}$$
$$u_3 = R_3 \cdot i_3 \tag{7.5d}$$
$$u_L = L \cdot \frac{di_L}{dt} \tag{7.5e}$$
$$u_0 = u_1 + u_3 \tag{7.5f}$$
$$u_L = u_1 + u_2 \tag{7.5g}$$
$$u_3 = u_2 \tag{7.5h}$$
$$i_0 = i_1 + i_L \tag{7.5i}$$
$$i_1 = i_2 + i_3 \tag{7.5j}$$

The structure digraph for this new set of equations is presented in Fig.7.6.

Let us now apply the Tarjan algorithm to this structure digraph. Figure 7.7 shows the partially causalized structure digraph.

Unfortunately, the Tarjan algorithm stalls at this point. Every one of the remaining acausal equations and every one of the remaining unknowns has at least two black (solid) lines attached to it. Consequently, the DAE system cannot be sorted entirely.

Let us read out the partially sorted equations. We shall only list on the

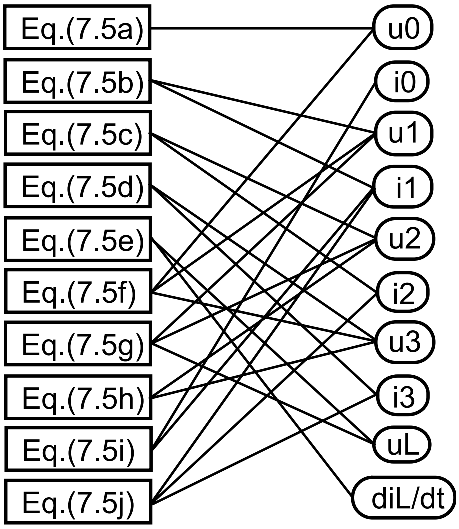

FIGURE 7.6. Structure digraph of modified electrical circuit.

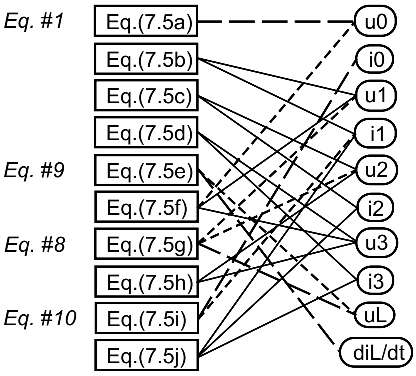

FIGURE 7.7. Structure digraph of partially causalized modified electrical circuit.

right side of the equal sign those variables that have already been computed.

$$u_0 = f(t) \tag{7.6a}$$
$$u_1 - R_1 \cdot i_1 = 0 \tag{7.6b}$$
$$u_2 - R_2 \cdot i_2 = 0 \tag{7.6c}$$
$$u_3 - R_3 \cdot i_3 = 0 \tag{7.6d}$$
$$u_1 + u_3 = u_0 \tag{7.6e}$$
$$u_2 - u_3 = 0 \tag{7.6f}$$
$$i_1 - i_2 - i_3 = 0 \tag{7.6g}$$

$$u_L = u_1 + u_2 \tag{7.6h}$$

$$\frac{di_L}{dt} = u_L / L \tag{7.6i}$$

$$i_0 = i_1 + i_L \tag{7.6j}$$

The six remaining acausal equations form an *algebraic loop.* They need to be solved together. The structure incidence matrix of the partially causalized equation system takes the form:

$$\mathbf{S} = \begin{array}{c|c|cccccc|c|c|c}
 & u_0 & u_1 & i_1 & u_2 & i_2 & u_3 & i_3 & u_L & \frac{di_L}{dt} & i_0 \\
\hline
Eq.(7.6a) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
Eq.(7.6b) & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Eq.(7.6c) & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
Eq.(7.6d) & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
Eq.(7.6e) & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
Eq.(7.6f) & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
Eq.(7.6g) & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
\hline
Eq.(7.6h) & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
Eq.(7.6i) & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
\hline
Eq.(7.6j) & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array} \tag{7.7}$$

Although the causalization algorithm has been unable to convert the structure incidence matrix to a true lower–triangular form, it was at least able to reduce it to a *Block–Lower–Triangular (BLT)* form. Furthermore, the algorithm generates diagonal blocks of minimal sizes.

How can we deal with the algebraic loop? Since the model is linear, we can write the loop equations in a matrix–vector form, and solve for the six unknowns by a *Gaussian elimination* in six equations and six unknowns.

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix} \tag{7.8}$$

Had the model been nonlinear in the loop equations, we would have had to use a *Newton iteration.*

Are algebraic loops a rarity in physical system modeling? Unfortunately, DAE systems containing algebraic loops are much more common than those that can be sorted completely by the Tarjan algorithm. Furthermore, the algebraic loops can be of frightening dimensions. For example when modeling mechanical *Multi–Body Systems (MBS)* [7.16, 7.18] containing closed kinematic loops, there immediately result highly nonlinear algebraic loops in hundreds if not thousands of unknowns and equations.

7.4 The Tearing Algorithm

We have now been haunted by large algebraic equation systems long enough. It is time that we do something about them.

Let us look once again at the system of six algebraic equations in six unknowns that we had met in the last section.

$$u_1 - R_1 \cdot i_1 = 0 \tag{7.9a}$$
$$u_2 - R_2 \cdot i_2 = 0 \tag{7.9b}$$
$$u_3 - R_3 \cdot i_3 = 0 \tag{7.9c}$$
$$u_1 + u_3 = u_0 \tag{7.9d}$$
$$u_2 - u_3 = 0 \tag{7.9e}$$
$$i_1 - i_2 - i_3 = 0 \tag{7.9f}$$

Its structure digraph is shown in Fig.7.8

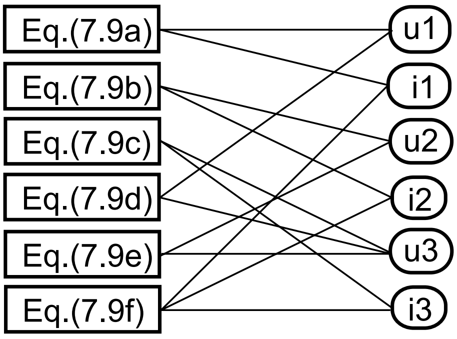

FIGURE 7.8. Structure digraph of algebraic equation system.

Clearly, every equation contains at least two unknowns, and every unknown appears in at least two equations. Yet, if only we could e.g. solve Eg.(7.9f) for the unknown i_3, as shown in Fig.7.9, then the entire set of equations could be made causal, as shown in Fig.7.10.

Now that the equation system has been causalized, we can write down the causal equations:

$$i_3 = i_1 - i_2 \tag{7.10a}$$
$$u_3 = R_3 \cdot i_3 \tag{7.10b}$$
$$u_1 = u_0 - u_3 \tag{7.10c}$$
$$i_1 = u_1 / R_1 \tag{7.10d}$$
$$u_2 = u_3 \tag{7.10e}$$
$$i_2 = u_2 / R_2 \tag{7.10f}$$

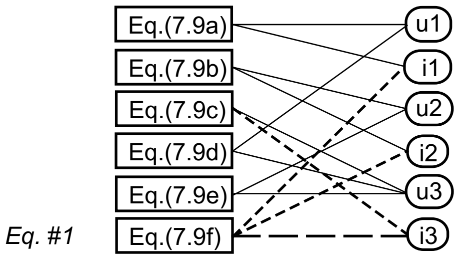

FIGURE 7.9. Structure digraph of partially causalized algebraic equation system.

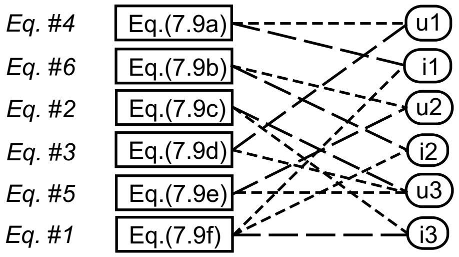

FIGURE 7.10. Structure digraph of completely causalized algebraic equation system.

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 Eq.(7.10a).

$$\begin{aligned}
i_3 &= i_1 - i_2 && (7.11a)\\
&= \frac{1}{R_1} \cdot u_1 - \frac{1}{R_2} \cdot u_2 && (7.11b)\\
&= \frac{1}{R_1} \cdot u_0 - \frac{1}{R_1} \cdot u_3 - \frac{1}{R_2} \cdot u_3 && (7.11c)\\
&= \frac{1}{R_1} \cdot u_0 - \frac{R_1 + R_2}{R_1 \cdot R_2} \cdot u_3 && (7.11d)\\
&= \frac{1}{R_1} \cdot u_0 - \frac{R_3 \cdot (R_1 + R_2)}{R_1 \cdot R_2} \cdot i_3 && (7.11e)
\end{aligned}$$

and thus:

$$\left[1 + \frac{R_3 \cdot (R_1 + R_2)}{R_1 \cdot R_2}\right] \cdot i_3 = \frac{1}{R_1} \cdot u_0 \tag{7.12}$$

or:

$$\frac{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3}{R_2} \cdot i_3 = u_0 \tag{7.13}$$

Since the equation is linear in i_3, we can solve it explicitly for the unknown, and obtain:

$$i_3 = \frac{R_2}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0 \tag{7.14}$$

Now, we can plug this equation back into the causalized equation system, replacing Eq.(7.10a) by it, and obtain the perfectly causal set of equations:

$$\begin{aligned}
i_3 &= \frac{R_2}{R_1 \cdot R_2 + R_1 \cdot R_3 + R_2 \cdot R_3} \cdot u_0 && \text{(7.15a)}\\
u_3 &= R_3 \cdot i_3 && \text{(7.15b)}\\
u_1 &= u_0 - u_3 && \text{(7.15c)}\\
i_1 &= u_1/R_1 && \text{(7.15d)}\\
u_2 &= u_3 && \text{(7.15e)}\\
i_2 &= u_2/R_2 && \text{(7.15f)}
\end{aligned}$$

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

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

Had the equations been nonlinear in the variable i_3, everything would have worked exactly the same way, except for the very last step, where we would have to involve a Newton iteration to solve for i_3, rather than solving for i_3 explicitly.

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.

Given the set of equations:

$$\begin{aligned}
u_3 &= R_3 \cdot i_3 && \text{(7.16a)}\\
u_1 &= u_0 - u_3 && \text{(7.16b)}\\
i_1 &= u_1/R_1 && \text{(7.16c)}\\
u_2 &= u_3 && \text{(7.16d)}\\
i_2 &= u_2/R_2 && \text{(7.16e)}\\
i_{3_{new}} &= i_1 - i_2 && \text{(7.16f)}
\end{aligned}$$

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:

$$\mathcal{F} = i_{3_{new}} - i_3 = 0.0 \tag{7.17}$$

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

$$\mathcal{H} = \frac{\partial \mathcal{F}}{\partial i_3} \tag{7.18}$$

A convenient way to compute the Hessian $\mathcal{H}$ is by means of *algebraic differentiation* [7.11].

$$du_3 = R_3 \tag{7.19a}$$
$$du_1 = -du_3 \tag{7.19b}$$
$$di_1 = du_1/R_1 \tag{7.19c}$$
$$du_2 = du_3 \tag{7.19d}$$
$$di_2 = du_2/R_2 \tag{7.19e}$$
$$di_{3_{new}} = di_1 - di_2 \tag{7.19f}$$
$$\mathcal{H} = di_{3_{new}} - 1 \tag{7.19g}$$

We can then compute the next version of i_3 as:

$$i_3 = i_3 - \mathcal{H} \backslash \mathcal{F} \tag{7.20}$$

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 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 given example, are called *tearing variables*, whereas the equations, from which the tearing variables are to be computed, such as Eq.(7.9f) in the given example, are called the *residual equations.*

Equation tearing is not exactly a new concept. The idea had originally been introduced by Gabriel Kron [7.12]. By now, many variations of different tearing algorithms have been reported in the literature. Some of the techniques are generally applicable, whereas others exploit particular matrix structures as they occur in special types of physical systems. Tearing has become most popular in chemical process engineering applications [7.13].

A version of tearing similar to the one described in this chapter has been implemented in Dymola [7.7] to accompany the Tarjan algorithm in the efficient solution of algebraic equation systems resulting from the automated symbolic conversion of DAE systems to ODE form [7.4].

How did we know to choose i_3 as tearing variable and Eq.(7.9f) as residual equation? What would have happened if we had chosen i_1 as the tearing variable and Eq.(7.9a) as the residual equation? The initial situation is depicted in Fig.7.11.

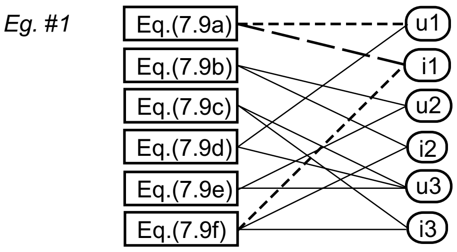

FIGURE 7.11. Structure digraph of partially causalized algebraic equation system.

We apply the Tarjan algorithm to the structure digraph. Unfortunately, the algorithm stalls once again after only one more step, as shown in Fig.7.12.

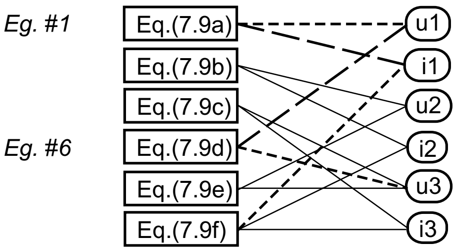

FIGURE 7.12. Structure digraph of partially causalized algebraic equation system.

We have been able to causalize only two of the six equations. Once again, we are faced with an algebraic loop in four equations and four unknowns, and therefore have to choose a second tearing variable and a second residual equation.

Let us proceed with the example to demonstrate, how the tearing algorithm can deal with multiple residual equations in multiple tearing variables. Let us select u_2 as the second tearing variable, and Eq.(7.9b) as the second residual equation. Now, we can complete the causalization of the equations. The completely colored structure digraph is shown in Fig.7.13.

We can read out the causal equations from the structure digraph of Fig.7.13.

$$i_1 = u_1/R_1 \tag{7.21a}$$
$$u_2 = R_2 \cdot i_2 \tag{7.21b}$$
$$u_3 = u_2 \tag{7.21c}$$
$$i_3 = u_3/R_3 \tag{7.21d}$$
$$i_2 = i_1 - i_3 \tag{7.21e}$$

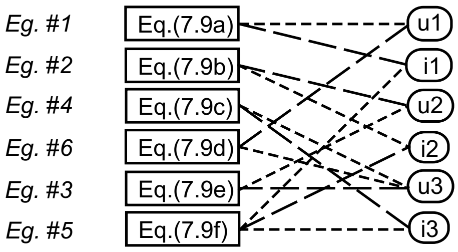

FIGURE 7.13. Structure digraph of completely causalized algebraic equation system.

$$u_1 = u_0 - u_3 \tag{7.21f}$$

Using the substitution technique, we can come up with two linearly independent equations in the two unknowns i_1 and u_2, i.e., in the two tearing variables. We begin with the first residual equation.

$$\begin{aligned}
i_1 &= u_1/R_1 && (7.22a)\\
&= \frac{1}{R_1} \cdot u_0 - \frac{1}{R_1} \cdot u_3 && (7.22b)\\
&= \frac{1}{R_1} \cdot u_0 - \frac{1}{R_1} \cdot u_2 && (7.22c)
\end{aligned}$$

Hence:

$$R_1 \cdot i_1 + u_2 = u_0 \tag{7.23}$$

We proceed with the second residual equation.

$$\begin{aligned}
u_2 &= R_2 \cdot i_2 && (7.24a)\\
&= R_2 \cdot i_1 - R_2 \cdot i_3 && (7.24b)\\
&= R_2 \cdot i_1 - \frac{R_2}{R_3} \cdot u_3 && (7.24c)\\
&= R_2 \cdot i_1 - \frac{R_2}{R_3} \cdot u_2 && (7.24d)\\
& && (7.24e)
\end{aligned}$$

Thus:

$$\left[1 + \frac{R_2}{R_3}\right] \cdot u_2 = R_2 \cdot i_1 \tag{7.25}$$

or:

$$R_2 \cdot R_3 \cdot i_1 - (R_2 + R_3) \cdot u_2 = 0 \tag{7.26}$$

We can write Eq.(7.23) and Eq.(7.26) in a matrix–vector form:

$$\begin{pmatrix} R_1 & 1 \\ R_2 \cdot R_3 & -(R_2+R_3) \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \end{pmatrix} \tag{7.27}$$

which can be solved for the two unknowns 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*.

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:

$$\begin{aligned}
u_3 &= u_2 && (7.28a)\\
i_3 &= u_3/R_3 && (7.28b)\\
i_2 &= i_1 - i_3 && (7.28c)\\
u_{2_{new}} &= R_2 \cdot i_2 && (7.28d)\\
u_1 &= u_0 - u_3 && (7.28e)\\
i_{1_{new}} &= u_1/R_1 && (7.28f)
\end{aligned}$$

We can formulate the following set of zero functions:

$$\mathcal{F} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} i_{1_{new}} - i_1 \\ u_{2_{new}} - u_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{7.29}$$

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

$$\mathcal{H} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} = \begin{pmatrix} \partial f_1/\partial i_1 & \partial f_1/\partial u_2 \\ \partial f_2/\partial i_1 & \partial f_2/\partial u_2 \end{pmatrix} \tag{7.30}$$

Using algebraic differentiation, we get:

$$\begin{aligned}
d_1u_3 &= 0 && (7.31a)\\
d_1i_3 &= d_1u_3/R_3 && (7.31b)\\
d_1i_2 &= 1 - d_1i_3 && (7.31c)\\
d_1u_{2_{new}} &= R_2 \cdot d_1i_2 && (7.31d)\\
d_1u_1 &= -d_1u_3 && (7.31e)\\
d_1i_{1_{new}} &= d_1u_1/R_1 && (7.31f)\\
d_2u_3 &= 1 && (7.31g)\\
d_2i_3 &= d_2u_3/R_3 && (7.31h)\\
d_2i_2 &= -d_2i_3 && (7.31i)\\
d_2u_{2_{new}} &= R_2 \cdot d_2i_2 && (7.31j)
\end{aligned}$$

$$d_2u_1 = -d_2u_3 \tag{7.31k}$$

$$d_2i_{1_{new}} = d_2u_1/R_1 \tag{7.31l}$$

$$h_{11} = d_1i_{1_{new}} - 1 \tag{7.31m}$$

$$h_{12} = d_2i_{1_{new}} \tag{7.31n}$$

$$h_{21} = d_1u_{2_{new}} \tag{7.31o}$$

$$h_{22} = d_2u_{2_{new}} - 1 \tag{7.31p}$$

where the prefix d_1 stands for the partial derivative with respect to i_1, and d_2 stands for the partial derivative with respect to u_2. Since i_1 and u_2 are mutually independent, 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 repeated 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.

We have seen by now that the selection of tearing variables and residual equations is not arbitrary. Our first choice led to a single residual equation in a single tearing variable, whereas our second choice led to two residual equations in two tearing variables.

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 heuristic procedure is described in the sequel.

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

2. For every one of these equations, follow its black (solid) lines, and determine those variables with the largest number of black (solid) 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.

Looking at the structure digraph of Fig.7.7, we see that only Eq.(7.5j) has three black (solid) lines attached to it. All other acausal equations have only two black solid lines attached to them. Consequently, Eq.(7.5j) will be chosen as the first residual equation.

Following each of the three black (solid) lines to the right side, we notice that each of the variables, i_1, i_2, and i_3 has exactly two black (solid) lines attached to it. Consequently, we need to check for each one of them, what happens if it were chosen as the first tearing variable.

It turns out that each of these three variables could have been chosen as the first tearing variable, since all of them lead to a complete causalization of the equation system.

We shall see in the next chapter that the simple heuristic algorithm described in this section sometimes maneuvers itself into a corner. The heuristic algorithm implemented in Dymola has been refined in several respects. On the one hand, it never gets stuck. The algorithm may become slow at times, but it will always find a legal tearing structure. On the other hand, the tearing algorithm implemented in Dymola guarantees that the selection of tearing variables never leads to a division by zero at run time. This is a rather tricky demand, because parameter values can change after compilation.

The complete tearing algorithm, as implemented in Dymola, has not been published. It is a company secret, designed to give Dynasim a competitive edge over its competitors.

7.5 The Relaxation Algorithm

There is yet another symbolic algorithm for the solution of algebraic systems of equations to be discussed, which is called the *relaxation algorithm* [7.17].

Contrary to the tearing algorithm, which is 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. One reason for this claim is the following. Within each Newton iteration of a nonlinear algebraic equation system, there is always a linear algebraic equation system to be solved. When we write the Newton iteration as:

$$\mathbf{x_{new}} = \mathbf{x_{old}} - \mathcal{H} \backslash \mathcal{F} \tag{7.32}$$

we are effectively saying that:

$$\mathbf{x_{new}} = \mathbf{x_{old}} - \mathbf{dx} \tag{7.33}$$

where $\mathbf{dx}$ is the solution of the linear algebraic equation system:

$$\mathcal{H} \cdot \mathbf{dx} = \mathcal{F} \tag{7.34}$$

Hence indeed, there is to be solved a linear algebraic equation system within each Newton iteration of the original nonlinear algebraic equation system.

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 in the previous section of this book.

We start out with the linear algebraic equation system in matrix–vector form, as presented in Eq.(7.8).

$$\begin{pmatrix} 1 & -R_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -R_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -R_3 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ u_3 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ u_0 \\ 0 \\ 0 \end{pmatrix} \tag{7.35}$$

However, we wish to minimize the number of non–zero elements above the diagonal. To this end, we causalize the equations in the same way that we used in the tearing algorithm, but write the residual equation as the last equation in the set.

$$u_3 = R_3 \cdot i_3 \tag{7.36a}$$
$$u_1 = u_0 - u_3 \tag{7.36b}$$
$$i_1 = \frac{u_1}{R_1} \tag{7.36c}$$
$$u_2 = u_3 \tag{7.36d}$$
$$i_2 = \frac{u_2}{R_2} \tag{7.36e}$$
$$i_3 = i_1 - i_2 \tag{7.36f}$$

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.

$$u_3 - R_3 \cdot i_3 = 0 \tag{7.37a}$$
$$u_1 + u_3 = u_0 \tag{7.37b}$$
$$R_1 \cdot i_1 - u_1 = 0 \tag{7.37c}$$
$$u_2 - u_3 = 0 \tag{7.37d}$$
$$R_2 \cdot i_2 - u_2 = 0 \tag{7.37e}$$
$$i_3 - i_1 + i_2 = 0 \tag{7.37f}$$

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:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -R_3 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & R_1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & R_2 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{7.38}$$

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

We can now apply Gaussian elimination without pivoting to this set of equations. Remember how Gaussian elimination works:

$$A_{ij}^{(n+1)} = A_{ij}^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot A_{kj}^{(n)} \tag{7.39a}$$

$$b_i^{(n+1)} = b_i^{(n)} - A_{ik}^{(n)} \cdot {A_{kk}^{(n)}}^{-1} \cdot b_k^{(n)} \tag{7.39b}$$

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.

Since we constantly eliminate rows and columns, the index k in the above equations is always 1. Thus, in the n plus first iteration of the algorithm, the element in row $i-1$ and column $j-1$ of the matrix is equal to the element in the row i and column j of the n^{th} iteration minus the product of the element at the very left end of the matrix (in row i) times the element at the very top end of the matrix (in column j) divided by the element in the top left corner.

For this reason, if an element in the top row is zero, the elements underneath it don't change at all during the iteration. Similarly, if an element in the leftmost column is zero, the elements to the right of it don't change.

Therefore, the only elements in the above equation system that change during the first iteration are the elements in the positions $< 2, 6 >$ and $< 4, 6 >$. Let us call the new elements c_1 and c_2. Thus, the second version of the equation system takes the form:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & c_1 \\ -1 & R_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & c_2 \\ 0 & 0 & -1 & R_2 & 0 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} u_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{7.40}$$

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

The only elements that can change in the next iteration are the element in the position $< 2, 5 >$ of the matrix, as well as the element in the position $< 2 >$ of the vector. Let us call those c_3 and c_4, respectively.

Thus, the third version of the equation system takes the form:

$$\begin{pmatrix} R_1 & 0 & 0 & c_3 \\ 0 & 1 & 0 & c_2 \\ 0 & -1 & R_2 & 0 \\ -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i_1 \\ u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} c_4 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{7.41}$$

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

In the next iteration, the only elements that can change are in the position $< 4, 4 >$ of the matrix, and in the position $< 4 >$ of the vector. Let us call these c_5, and c_6, respectively.

The fourth version of the equation system takes the form:

$$\begin{pmatrix} 1 & 0 & c_2 \\ -1 & R_2 & 0 \\ 0 & 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} u_2 \\ i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ c_6 \end{pmatrix} \tag{7.42}$$

where $c_5 = 1 + c_3/R_1$, and $c_6 = c_4/R_1$.

In the next iteration, the only element that can change is in the position $< 2, 3 >$ of the matrix. Let us call the new element c_7.

The fifth version of the equation system takes the form:

$$\begin{pmatrix} R_2 & c_7 \\ 1 & c_5 \end{pmatrix} \cdot \begin{pmatrix} i_2 \\ i_3 \end{pmatrix} = \begin{pmatrix} 0 \\ c_6 \end{pmatrix} \tag{7.43}$$

where $c_7 = c_2$.

In the final iteration, the only element that can change is in the position $< 2, 2 >$ of the matrix. Let us call the new element c_8.

The sixth and last version of the equation system takes the form:

$$\begin{pmatrix} c_8 \end{pmatrix} \cdot \begin{pmatrix} i_3 \end{pmatrix} = \begin{pmatrix} c_6 \end{pmatrix} \tag{7.44}$$

where $c_8 = c_5 - c_7/R_2$.

This equation system can be solved at once for the unknown i_3:

$$i_3 = \frac{c_6}{c_8} \tag{7.45}$$

¿From the previous set of equations, we can subsequently compute:

$$i_2 = -\frac{c_7 \cdot i_3}{R_2} \tag{7.46}$$

and so forth.

Thus, the overall equation system can be replaced by the following set of symbolic scalar equations:

$$c_1 = R_3 \tag{7.47a}$$
$$c_2 = -R_3 \tag{7.47b}$$
$$c_3 = c_1 \tag{7.47c}$$
$$c_4 = u_0 \tag{7.47d}$$
$$c_5 = 1 + \frac{c_3}{R_1} \tag{7.47e}$$
$$c_6 = \frac{c_4}{R_1} \tag{7.47f}$$
$$c_7 = c_2 \tag{7.47g}$$
$$c_8 = c_5 - \frac{c_7}{R_2} \tag{7.47h}$$
$$i_3 = \frac{c_6}{c_8} \tag{7.47i}$$
$$i_2 = -\frac{c_7 \cdot i_3}{R_2} \tag{7.47j}$$
$$u_2 = -c_2 \cdot i_3 \tag{7.47k}$$
$$i_1 = \frac{c_4 - c_3 \cdot i_3}{R_1} \tag{7.47l}$$
$$u_1 = u_0 - c_1 \cdot i_3 \tag{7.47m}$$
$$u_3 = R_3 \cdot i_3 \tag{7.47n}$$

Of course, we can also combine the relaxation approach with tearing. Once an expression for the tearing variable i_3 has been found, the remaining variables can be computed from the original set of equations instead of using those from the back–substitution:

$$c_1 = R_3 \tag{7.48a}$$
$$c_2 = -R_3 \tag{7.48b}$$
$$c_3 = c_1 \tag{7.48c}$$
$$c_4 = u_0 \tag{7.48d}$$
$$c_5 = 1 + \frac{c_3}{R_1} \tag{7.48e}$$
$$c_6 = \frac{c_4}{R_1} \tag{7.48f}$$
$$c_7 = c_2 \tag{7.48g}$$
$$c_8 = c_5 - \frac{c_7}{R_2} \tag{7.48h}$$
$$i_3 = \frac{c_6}{c_8} \tag{7.48i}$$
$$u_3 = R_3 \cdot i_3 \tag{7.48j}$$

$$u_1 = u_0 - u_3 \tag{7.48k}$$
$$i_1 = \frac{u_1}{R_1} \tag{7.48l}$$
$$u_2 = u_3 \tag{7.48m}$$
$$i_2 = \frac{u_2}{R_2} \tag{7.48n}$$

What would have happened if we had started out with the second set of causal equations, i.e., the one derived involving two tearing variables. The causal equations present themselves as follows:

$$u_3 = u_2 \tag{7.49a}$$
$$i_3 = \frac{u_3}{R_3} \tag{7.49b}$$
$$i_2 = i_1 - i_3 \tag{7.49c}$$
$$u_2 = R_2 \cdot i_2 \tag{7.49d}$$
$$u_1 = u_0 - u_3 \tag{7.49e}$$
$$i_1 = \frac{u_1}{R_1} \tag{7.49f}$$

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

$$u_3 - u_2 = 0 \tag{7.50a}$$
$$R_3 \cdot i_3 - u_3 = 0 \tag{7.50b}$$
$$i_2 - i_1 + i_3 = 0 \tag{7.50c}$$
$$u_2 - R_2 \cdot i_2 = 0 \tag{7.50d}$$
$$u_1 + u_3 = u_0 \tag{7.50e}$$
$$R_1 \cdot i_1 - u_1 = 0 \tag{7.50f}$$

This set of equations can be written in a matrix–vector form as follows:

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ -1 & R_3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & -R_2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & R_1 \end{pmatrix} \cdot \begin{pmatrix} u_3 \\ i_3 \\ i_2 \\ u_2 \\ u_1 \\ i_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ u_0 \\ 0 \end{pmatrix} \tag{7.51}$$

Just as in the previous case, all the diagonal elements of the matrix are non–zero, allowing a Gaussian elimination without pivoting to be performed. However 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.

Finding a minimal set of non–zero elements above the diagonal of the matrix is thus 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.

Why were we interested in minimizing the number of non–zero elements above the diagonal? Remember that we only need to introduce new auxiliary variables c_i in the symbolic Gaussian elimination, if both the elements in the top row and the elements in the leftmost column are non–zero. If an element in the top row is zero, then all the elements beneath it don't change in the next version of the system equations, i.e., in the next step of the Gaussian elimination algorithm. Thus by minimizing the number of non–zero elements above the diagonal, we also minimize the number of new auxiliary variables that need to be introduced, and for which expressions have to be evaluated.

Hence also the symbolic Gaussian elimination algorithm exploits the number and positions of the zero elements in the linear equation system, and therefore can be interpreted as a symbolic *sparse matrix technique.*

7.6 Structural Singularities

Let us now look at yet another circuit problem. This time, we shall exchange the capacitor and the inductor, as shown in Fig.7.14.

The set of differential and algebraic equations thus presents itself as:

$$u_0 = f(t) \tag{7.52a}$$
$$u_1 = R_1 \cdot i_1 \tag{7.52b}$$
$$u_2 = R_2 \cdot i_2 \tag{7.52c}$$
$$u_L = L \cdot \frac{di_L}{dt} \tag{7.52d}$$
$$i_C = C \cdot \frac{du_C}{dt} \tag{7.52e}$$
$$u_0 = u_1 + u_L \tag{7.52f}$$
$$u_C = u_1 + u_2 \tag{7.52g}$$
$$u_L = u_2 \tag{7.52h}$$
$$i_0 = i_1 + i_C \tag{7.52i}$$
$$i_1 = i_2 + i_L \tag{7.52j}$$

with the structure digraph as shown in Fig.7.15.

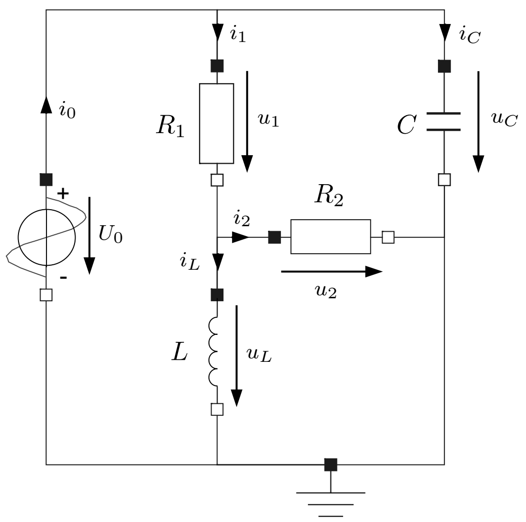

FIGURE 7.14. Schematic of once more modified electrical RLC circuit.

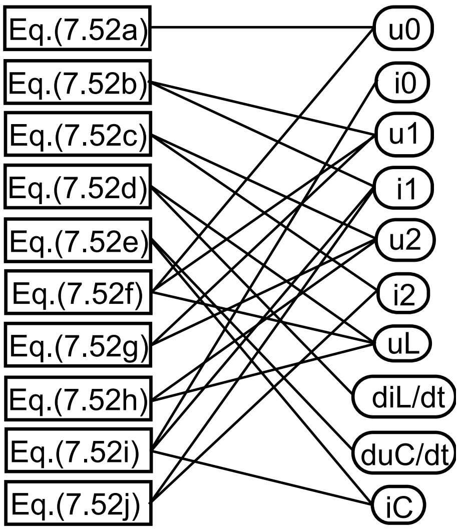

FIGURE 7.15. Structure digraph of once more modified electrical circuit.

Let us start to color the digraph. Figure 7.16 shows the partially colored digraph after the first iteration.

We notice that we are in trouble. The variable i_c has two blue (dotted) lines attached to it, and nothing else. Consequently, we no longer have an equation to compute the value of i_c.

Let us try another approach. We can introduce the node potentials, v_i, as additional variables, write down for each branch the relationship be-

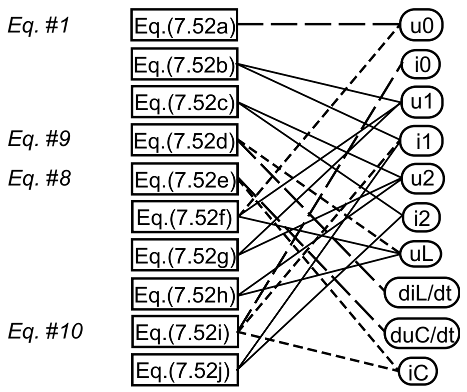

FIGURE 7.16. Structure digraph of partially causalized electrical circuit.

tween the branch voltage and the two neighboring node potentials, and eliminate the mesh equations instead. Figure 7.17 shows the schematic of the electrical circuit with node potentials.

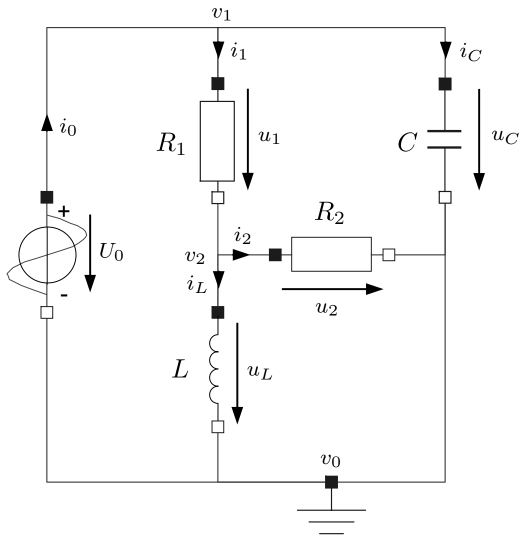

FIGURE 7.17. Schematic of electrical RLC circuit with node potentials.

The new set of equations can be written as follows:

$$u_0 = f(t) \tag{7.53a}$$
$$u_0 = v_1 - v_0 \tag{7.53b}$$
$$u_1 = R_1 \cdot i_1 \tag{7.53c}$$
$$u_1 = v_1 - v_2 \tag{7.53d}$$

$$u_2 = R_2 \cdot i_2 \tag{7.53e}$$

$$u_2 = v_2 - v_0 \tag{7.53f}$$

$$u_L = L \cdot \frac{di_L}{dt} \tag{7.53g}$$

$$u_L = v_2 - v_0 \tag{7.53h}$$

$$i_C = C \cdot \frac{du_C}{dt} \tag{7.53i}$$

$$u_C = v_1 - v_0 \tag{7.53j}$$

$$v_0 = 0 \tag{7.53k}$$

$$i_0 = i_1 + i_C \tag{7.53l}$$

$$i_1 = i_2 + i_L \tag{7.53m}$$

This time, we ended up with 13 equations in 13 unknowns. Its structure digraph is shown in Fig.7.18.

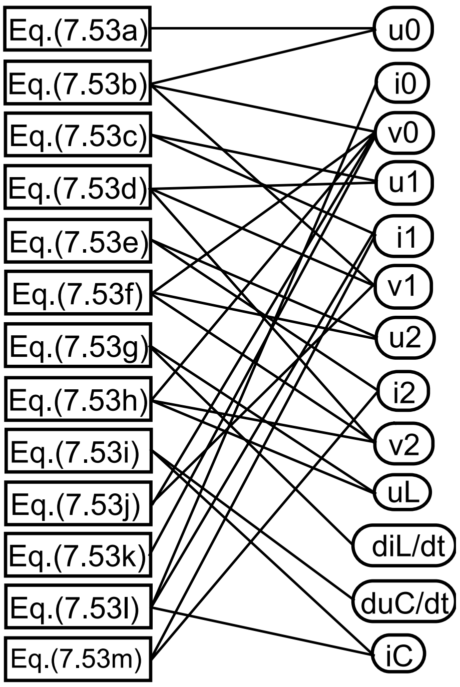

FIGURE 7.18. Structure digraph of electrical circuit with node potentials.

Figure 7.19 shows the partially colored digraph.

We again got stuck after two iterations. However, it seems that we made the problem worse rather than better. Just like last time, we again are left without an equation to compute i_C, but this time, we are also left with an equation, Eq.(7.53j), which has no unknowns left in it, although it has not

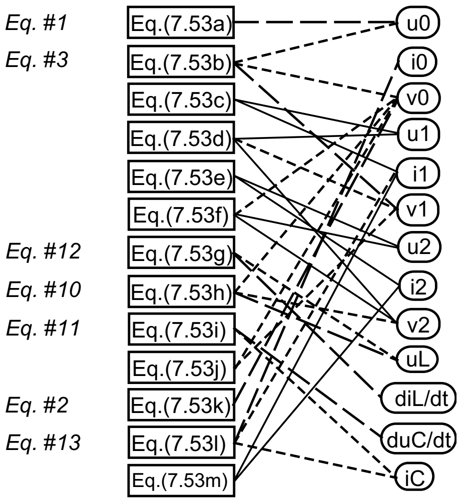

FIGURE 7.19. Partially colored structure digraph of electrical circuit with node potentials.

yet been made causal. We won't be able to use it for anything. Such an equation is called a *constraint equation*.

What has happened in this circuit? The capacitor has been placed in parallel with a voltage source. Consequently, the voltage across the capacitor cannot be chosen as an independent state variable. There is no freedom in choosing its initial condition.

7.7 Structural Singularity Elimination

Before we deal with the above circuit, let us choose a much simpler circuit that exhibits the same problems. Figure 7.20 shows the schematic of an electrical circuit with two capacitors in parallel.

Since the two capacitive voltages, u_1 and u_2, are the same, they don't qualify as independent state variables, as we cannot choose initial conditions for them independently. Hence we expect problems that are similar to those observed in the previous circuit.

The equations describing this circuit can be written as:

$$u_0 = f(t) \tag{7.54a}$$

$$u_R = R \cdot i_0 \tag{7.54b}$$

$$i_1 = C_1 \cdot \frac{du_1}{dt} \tag{7.54c}$$

$$i_2 = C_2 \cdot \frac{du_2}{dt} \tag{7.54d}$$

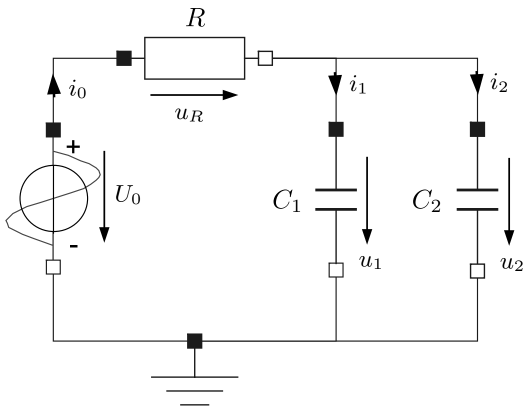

FIGURE 7.20. Schematic of electrical circuit with two capacitors in parallel.

$$u_0 = u_R + u_1 \tag{7.54e}$$
$$u_2 = u_1 \tag{7.54f}$$
$$i_0 = i_1 + i_2 \tag{7.54g}$$

If we choose u_1 and u_2 as state variables, then both u_1 and u_2 are considered known variables, and Eq.(7.54f) has no unknown left. Thus, it must be considered a *constraint equation.*

There are several different ways, how this problem can be solved [7.4]. We can turn the causality around on one of the capacitive equations, solving e.g. for the variable i_2, instead of du_2/dt. Consequently, the solver has to solve for du_2/dt instead of u_2, thus the *integrator* has been turned into a *differentiator.*

In the model equations, u_2 must be considered an unknown, whereas du_2/dt is considered a known variable. The equations can now easily be brought into causal form:

$$u_0 = f(t) \tag{7.55a}$$
$$i_2 = C_2 \cdot \frac{du_2}{dt} \tag{7.55b}$$
$$u_2 = u_1 \tag{7.55c}$$
$$u_R = u_0 - u_1 \tag{7.55d}$$
$$i_0 = \frac{1}{R} \cdot u_R \tag{7.55e}$$
$$i_1 = i_0 - i_2 \tag{7.55f}$$
$$\frac{du_1}{dt} = \frac{1}{C_1} \cdot i_1 \tag{7.55g}$$

with the block diagram as shown in Fig.7.21.

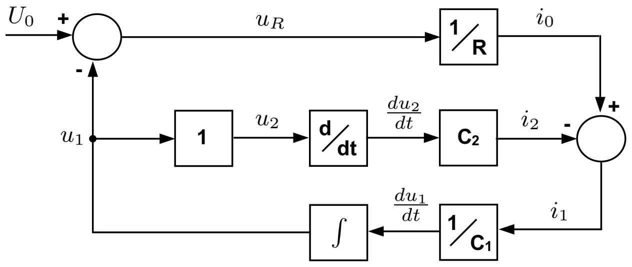

FIGURE 7.21. Block diagram of electrical circuit with parallel capacitors.

The solver generates u_1 out of du_1/dt by *numerical integration*, whereas it generates du_2/dt out of u_2 by *numerical differentiation.*

Numerical differentiation is a bad idea, at least if explicit formulae are being used. Using implicit formulae, numerical integration and differentiation are essentially the same, but as we already know, implicit formulae call for an iteration at every step.

Costas Pantelides [7.20] had a better idea. How about modifying the equation system such that the constraint equation disappears? In the above example, if:

$$u_2 = u_1 \tag{7.56}$$

at all times, then obviously, it must also be true that:

$$\frac{du_2}{dt} = \frac{du_1}{dt} \tag{7.57}$$

at all times. Thus, we can symbolically differentiate the constraint equation, and replace the constraint equation by its derivative. The new set of acausal equations takes the form:

$$u_0 = f(t) \tag{7.58a}$$
$$u_R = R \cdot i_0 \tag{7.58b}$$
$$i_1 = C_1 \cdot \frac{du_1}{dt} \tag{7.58c}$$
$$i_2 = C_2 \cdot \frac{du_2}{dt} \tag{7.58d}$$
$$u_0 = u_R + u_1 \tag{7.58e}$$
$$\frac{du_2}{dt} = \frac{du_1}{dt} \tag{7.58f}$$
$$i_0 = i_1 + i_2 \tag{7.58g}$$

with the partially colored structure digraph as shown in Fig.7.22.

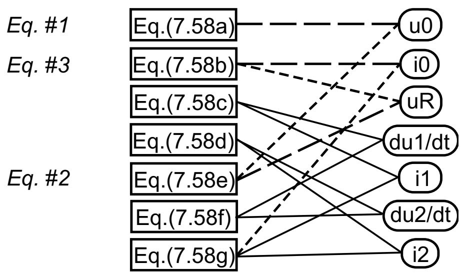

FIGURE 7.22. Partially colored structure digraph of electrical circuit with parallel capacitors after differentiation of the constraint equation.

The constraint equation has indeed disappeared. After partial causalization of the equations, we are now faced with an algebraic loop in four equations and four unknowns, a situation that we already know how to deal with.

Miraculously, we seem to have gotten rid of the constraint between the two capacitors. After the symbolic differentiation of the constraint equation, we seem to again have two integrators that we can integrate separately and independently.

Evidently, this cannot be true. The constraint on the capacitive voltages has not disappeared. It has only been hidden. It is true that we can now numerically integrate du_1/dt into u_1, and du_2/dt into u_2. However, we still must satisfy the original constraint equation when choosing the initial conditions for the two integrators.

The second integrator does not represent a true state variable. In fact, it is wasteful. We don't need two integrators, since the system has only one *degree of freedom*, i.e., one energy storage.

Let us thus modify the Pantelides algorithm once more. Instead of replacing the constraint equation by its derivative, we add the differentiated constraint equation as an additional equation to the set.

Hence we now have eight equations in seven unknowns. We have one equation too many, and consequently, we need to throw away one of them. We shall throw away one of the integrators, for example, the one that integrates du_2/dt into u_2.

We symbolize this by renaming the variable du_2/dt as du_2. du_2 is no longer a state derivative. It is simply an algebraic variable with a funny name. Hence both u_2 and du_2 are now unknowns, and we are thus faced with eight equations in eight unknowns. These are:

$$u_0 = f(t) \tag{7.59a}$$

$$u_R = R \cdot i_0 \tag{7.59b}$$

$$i_1 = C_1 \cdot \frac{du_1}{dt} \tag{7.59c}$$

$$i_2 = C_2 \cdot du_2 \tag{7.59d}$$
$$u_0 = u_R + u_1 \tag{7.59e}$$
$$u_2 = u_1 \tag{7.59f}$$
$$du_2 = \frac{du_1}{dt} \tag{7.59g}$$
$$i_0 = i_1 + i_2 \tag{7.59h}$$

with the partially colored structure digraph as shown in Fig.7.23.

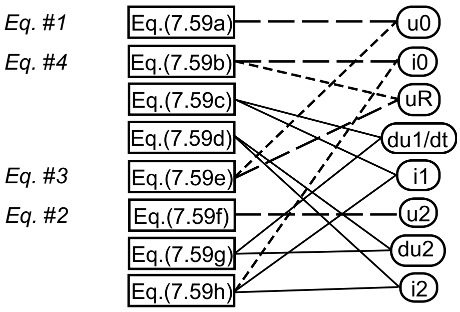

FIGURE 7.23. Partially colored structure digraph of electrical circuit with parallel capacitors after differentiation of the constraint equation.

Once again, we end up with an algebraic loop in four equations and four unknowns.

In the mathematical literature, *structurally singular* systems are called *higher–index problems*, or more precisely, structurally singular physical systems lead to mathematical descriptions that present themselves in the form of higher–index DAEs [7.1, 7.2, 7.19].

The *perturbation index* is a measure of the constraints among equations [7.10]. An *index–0 DAE* contains neither algebraic loops nor structural singularities. An *index–1 DAE* contains algebraic loops, but no structural singularities. A DAE with a perturbation index > 1, a so–called *higher–index DAE*, contains structural singularities[1].

The algorithm by Pantelides is a *symbolic index reduction* algorithm[2]. It reduces the perturbation index by one. Hence it may be necessary to

[1]A number of different definitions of structure indices are provided in the mathematical literature. A paper by Campbell and Gear [7.3] offers a good survey of this somewhat exotic issue. The different definitions all agree on the index of a *linear* DAE system, but sometimes disagree in the case of nonlinear DAE systems.

[2]The original paper by Pantelides did not concern itself with index reduction at all. It described an algorithm that could find, in a procedural fashion, a complete and consistent set of initial conditions for a DAE system. It was shown later in [7.4, 7.15] that the Pantelides algorithm can also be used as a symbolic index reduction algorithm.

apply the Pantelides algorithm more than once. For example, a mechanical system with constraints among positions or angles, such as a motor with a load, whereby the motor and the load are described separately by differential equations, leads to an index–3 DAE system. By applying the Pantelides algorithm once, the constraint gets reduced to a constraint between velocities or angular velocities, which are still state variables. By applying the Pantelides algorithm a second time, the constraint gets reduced to a constraint between accelerations or angular accelerations, which are no longer outputs of integrators, and therefore, are no longer state variables.

It is thus not surprising that, after applying the Pantelides algorithm, we ended up with an algebraic loop. This is usually the case.

Let us now return to the more complex circuit. We shall start with the version that contains a constraint equation, i.e., the version making use of the node potentials. The partially causalized set of equations can be written as follows:

$$u_0 = f(t) \tag{7.60a}$$
$$v_0 = 0 \tag{7.60b}$$
$$v_1 = u_0 - v_0 \tag{7.60c}$$
$$u_1 - R_1 \cdot i_1 = 0 \tag{7.60d}$$
$$u_1 + v_2 = v_1 \tag{7.60e}$$
$$u_2 - R_2 \cdot i_2 = 0 \tag{7.60f}$$
$$v_2 - u_2 = v_0 \tag{7.60g}$$
$$0 = u_C - v_1 + v_0 \tag{7.60h}$$
$$i_1 - i_2 = i_L \tag{7.60i}$$
$$u_L = v_2 - v_0 \tag{7.60j}$$
$$\frac{du_C}{dt} = \frac{1}{C} \cdot i_C \tag{7.60k}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.60l}$$
$$i_0 = i_1 + i_C \tag{7.60m}$$

The unknowns are written on the left side of the equal sign, thus equations with only one variable to the left of the equal sign are causal equations, those with more than one variable to the left of the equal sign are acausal equations, and those with zero variables to the left of the equal sign are constraint equations.

We differentiate the constraint equation, add it to the DAE system, and let go of an integrator associated with the constraint.

$$u_0 = f(t) \tag{7.61a}$$

$$v_0 = 0 \tag{7.61b}$$
$$v_1 = u_0 - v_0 \tag{7.61c}$$
$$u_1 - R_1 \cdot i_1 = 0 \tag{7.61d}$$
$$u_1 + v_2 = v_1 \tag{7.61e}$$
$$u_2 - R_2 \cdot i_2 = 0 \tag{7.61f}$$
$$v_2 - u_2 = v_0 \tag{7.61g}$$
$$0 = u_C - v_1 + v_0 \tag{7.61h}$$
$$0 = du_C - dv_1 + dv_0 \tag{7.61i}$$
$$i_1 - i_2 = i_L \tag{7.61j}$$
$$u_L = v_2 - v_0 \tag{7.61k}$$
$$du_C = \frac{1}{C} \cdot i_C \tag{7.61l}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.61m}$$
$$i_0 = i_1 + i_C \tag{7.61n}$$

In the process of differentiation, we introduced two new pseudo–derivatives[1], dv_1 and dv_0, for which we are lacking equations.

We now differentiate the causal equations that define v_1 and v_0, and add those to the set as well.

$$u_0 = f(t) \tag{7.62a}$$
$$v_0 = 0 \tag{7.62b}$$
$$dv_0 = 0 \tag{7.62c}$$
$$v_1 = u_0 - v_0 \tag{7.62d}$$
$$dv_1 = du_0 - dv_0 \tag{7.62e}$$
$$u_1 - R_1 \cdot i_1 = 0 \tag{7.62f}$$
$$u_1 + v_2 = v_1 \tag{7.62g}$$
$$u_2 - R_2 \cdot i_2 = 0 \tag{7.62h}$$
$$v_2 - u_2 = v_0 \tag{7.62i}$$
$$0 = u_C - v_1 + v_0 \tag{7.62j}$$
$$0 = du_C - dv_1 + dv_0 \tag{7.62k}$$
$$i_1 - i_2 = i_L \tag{7.62l}$$
$$u_L = v_2 - v_0 \tag{7.62m}$$
$$du_C = \frac{1}{C} \cdot i_C \tag{7.62n}$$

[1]The pseudo–derivatives are sometimes also called *dummy–derivatives* in the literature [7.15].

$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.62o}$$

$$i_0 = i_1 + i_C \tag{7.62p}$$

We again introduced one additional pseudo–derivative, du_0. Thus, the final set of equations can be written as:

$$u_0 = f(t) \tag{7.63a}$$

$$du_0 = \frac{df(t)}{dt} \tag{7.63b}$$

$$v_0 = 0 \tag{7.63c}$$

$$dv_0 = 0 \tag{7.63d}$$

$$v_1 = u_0 - v_0 \tag{7.63e}$$

$$dv_1 = du_0 - dv_0 \tag{7.63f}$$

$$u_1 - R_1 \cdot i_1 = 0 \tag{7.63g}$$

$$u_1 + v_2 = v_1 \tag{7.63h}$$

$$u_2 - R_2 \cdot i_2 = 0 \tag{7.63i}$$

$$v_2 - u_2 = v_0 \tag{7.63j}$$

$$0 = u_C - v_1 + v_0 \tag{7.63k}$$

$$0 = du_C - dv_1 + dv_0 \tag{7.63l}$$

$$i_1 - i_2 = i_L \tag{7.63m}$$

$$u_L = v_2 - v_0 \tag{7.63n}$$

$$du_C = \frac{1}{C} \cdot i_C \tag{7.63o}$$

$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.63p}$$

$$i_0 = i_1 + i_C \tag{7.63q}$$

If $f(t)$ is a known function of time, we can symbolically compute its derivative. On the other hand, if $f(t)$ stands for an input signal in a real–time simulation with hardware in the loop, we have a problem. In that case, we may need an additional sensor somewhere in the system that measures $df(t)/dt$, and add this signal as an additional real–time input to the simulation.

By now, we have 17 equations in 17 unknowns. The partially colored structure digraph is shown in Fig.7.24.

It worked. The Pantelides algorithm was able to reduce the DAE system to index–1. We ended up with an algebraic loop in five equations and five unknowns that can be tackled using any one among the techniques described in the previous sections of this chapter.

Let us now return to the original description of the circuit without node potentials. In that formulation, no constraint equation was visible. We only

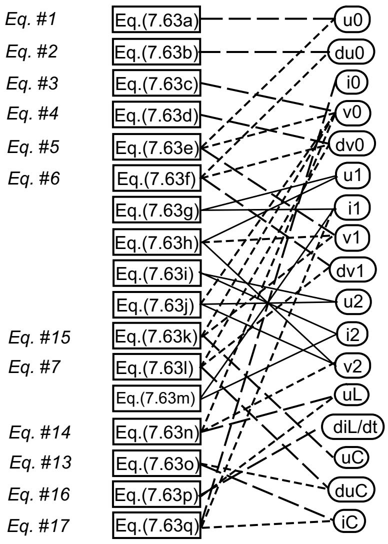

FIGURE 7.24. Partially colored structure digraph of electrical RLC circuit with node potentials after differentiation of the constraint equation.

knew that we had run into difficulties, because we recognized that we were left without an equation to compute the value of the variable i_c.

Let us write down the equations in their partially causalized form.

$$u_0 = f(t) \tag{7.64a}$$

$$u_1 - R_1 \cdot i_1 = 0 \tag{7.64b}$$

$$u_2 - R_2 \cdot i_2 = 0 \tag{7.64c}$$

$$u_1 + u_L = u_0 \tag{7.64d}$$

$$u_1 + u_2 = u_C \tag{7.64e}$$

$$u_L - u_2 = 0 \tag{7.64f}$$

$$i_1 - i_2 = i_L \tag{7.64g}$$

$$\frac{du_C}{dt} = \frac{1}{C} \cdot i_C \tag{7.64h}$$

$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.64i}$$

$$i_0 = i_1 + i_C \tag{7.64j}$$

We are left with six acausal equations in only five unknowns, since i_c doesn't show up anywhere in them. Thus, there still exists a constraint equation. However, it is hidden inside an algebraic system.

Let us draw the structure digraph of the algebraic system, and let us choose a residual equation and a tearing variable. The structure digraph is shown in Fig.7.25.

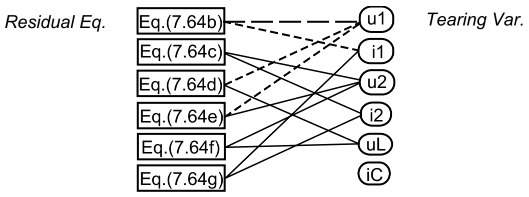

FIGURE 7.25. Structure digraph of algebraic subsystem of electrical RLC circuit without node potentials after a tearing variable has been selected.

We now complete the causalization of the algebraic equation system. The completely colored structure digraph is shown in Fig.7.26.

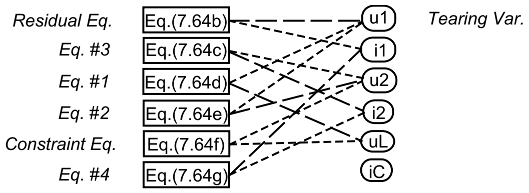

FIGURE 7.26. Completely colored structure digraph of algebraic subsystem of electrical RLC circuit without node potentials.

The constraint equation, Eq.(7.64f) has become clearly visible. We can now write down the equation system in its completely causalized form:

$$u_0 = f(t) \tag{7.65a}$$
$$u_L = u_0 - u_1 \tag{7.65b}$$
$$u_2 = u_C - u_1 \tag{7.65c}$$
$$i_2 = \frac{1}{R_2} \cdot u_2 \tag{7.65d}$$
$$i_1 = i_2 + i_L \tag{7.65e}$$
$$0 = u_L - u_2 \tag{7.65f}$$
$$u_1 = R_1 \cdot i_1 \tag{7.65g}$$
$$\frac{du_C}{dt} = \frac{1}{C} \cdot i_C \tag{7.65h}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.65i}$$
$$i_0 = i_1 + i_C \tag{7.65j}$$

We now apply the Pantelides algorithm. We start by differentiating the constraint equation, adding it to the equation system.

$$u_0 = f(t) \tag{7.66a}$$
$$u_L = u_0 - u_1 \tag{7.66b}$$
$$u_2 = u_C - u_1 \tag{7.66c}$$
$$i_2 = \frac{1}{R_2} \cdot u_2 \tag{7.66d}$$
$$i_1 = i_2 + i_L \tag{7.66e}$$
$$0 = u_L - u_2 \tag{7.66f}$$
$$0 = du_L - du_2 \tag{7.66g}$$
$$u_1 = R_1 \cdot i_1 \tag{7.66h}$$
$$\frac{du_C}{dt} = \frac{1}{C} \cdot i_C \tag{7.66i}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.66j}$$
$$i_0 = i_1 + i_C \tag{7.66k}$$

We introduced two new pseudo–derivatives, du_L and du_2. Hence we differentiate the equations defining u_L and u_2.

$$u_0 = f(t) \tag{7.67a}$$
$$u_L = u_0 - u_1 \tag{7.67b}$$
$$du_L = du_0 - du_1 \tag{7.67c}$$
$$u_2 = u_C - u_1 \tag{7.67d}$$
$$du_2 = du_C - du_1 \tag{7.67e}$$
$$i_2 = \frac{1}{R_2} \cdot u_2 \tag{7.67f}$$
$$i_1 = i_2 + i_L \tag{7.67g}$$
$$0 = u_L - u_2 \tag{7.67h}$$
$$0 = du_L - du_2 \tag{7.67i}$$
$$u_1 = R_1 \cdot i_1 \tag{7.67j}$$
$$\frac{du_C}{dt} = \frac{1}{C} \cdot i_C \tag{7.67k}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.67l}$$
$$i_0 = i_1 + i_C \tag{7.67m}$$

We introduced three new pseudo–derivatives, du_0, du_1, and du_C. We differentiate the equations defining u_0 and u_1, and we throw the integrator away that defines u_C.

$$u_0 = f(t) \tag{7.68a}$$
$$du_0 = \frac{df(t)}{dt} \tag{7.68b}$$
$$u_L = u_0 - u_1 \tag{7.68c}$$
$$du_L = du_0 - du_1 \tag{7.68d}$$
$$u_2 = u_C - u_1 \tag{7.68e}$$
$$du_2 = du_C - du_1 \tag{7.68f}$$
$$i_2 = \frac{1}{R_2} \cdot u_2 \tag{7.68g}$$
$$i_1 = i_2 + i_L \tag{7.68h}$$
$$0 = u_L - u_2 \tag{7.68i}$$
$$0 = du_L - du_2 \tag{7.68j}$$
$$u_1 = R_1 \cdot i_1 \tag{7.68k}$$
$$du_1 = R_1 \cdot di_1 \tag{7.68l}$$
$$du_C = \frac{1}{C} \cdot i_C \tag{7.68m}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.68n}$$
$$i_0 = i_1 + i_C \tag{7.68o}$$

We introduced another pseudo–derivative, di_1. Thus, we need to differentiate the equation defining i_1 as well.

$$u_0 = f(t) \tag{7.69a}$$
$$du_0 = \frac{df(t)}{dt} \tag{7.69b}$$
$$u_L = u_0 - u_1 \tag{7.69c}$$
$$du_L = du_0 - du_1 \tag{7.69d}$$
$$u_2 = u_C - u_1 \tag{7.69e}$$
$$du_2 = du_C - du_1 \tag{7.69f}$$
$$i_2 = \frac{1}{R_2} \cdot u_2 \tag{7.69g}$$
$$i_1 = i_2 + i_L \tag{7.69h}$$
$$di_1 = di_2 + \frac{di_L}{dt} \tag{7.69i}$$
$$0 = u_L - u_2 \tag{7.69j}$$
$$0 = du_L - du_2 \tag{7.69k}$$

$$u_1 = R_1 \cdot i_1 \tag{7.69l}$$
$$du_1 = R_1 \cdot di_1 \tag{7.69m}$$
$$du_C = \frac{1}{C} \cdot i_C \tag{7.69n}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.69o}$$
$$i_0 = i_1 + i_C \tag{7.69p}$$

We now introduced yet a new pseudo–derivative, di_2, and also a true derivative, di_L/dt.

As the constraint equation was hidden in one solid algebraic loop, we had to differentiate every single equation of that loop. We ended up with the following set of 17 equations in 17 unknowns:

$$u_0 = f(t) \tag{7.70a}$$
$$du_0 = \frac{df(t)}{dt} \tag{7.70b}$$
$$u_L = u_0 - u_1 \tag{7.70c}$$
$$du_L = du_0 - du_1 \tag{7.70d}$$
$$u_2 = u_C - u_1 \tag{7.70e}$$
$$du_2 = du_C - du_1 \tag{7.70f}$$
$$i_2 = \frac{1}{R_2} \cdot u_2 \tag{7.70g}$$
$$di_2 = \frac{1}{R_2} \cdot du_2 \tag{7.70h}$$
$$i_1 = i_2 + i_L \tag{7.70i}$$
$$di_1 = di_2 + \frac{di_L}{dt} \tag{7.70j}$$
$$0 = u_L - u_2 \tag{7.70k}$$
$$0 = du_L - du_2 \tag{7.70l}$$
$$u_1 = R_1 \cdot i_1 \tag{7.70m}$$
$$du_1 = R_1 \cdot di_1 \tag{7.70n}$$
$$du_C = \frac{1}{C} \cdot i_C \tag{7.70o}$$
$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.70p}$$
$$i_0 = i_1 + i_C \tag{7.70q}$$

Let us start from scratch with the causalization of this DAE system. Figure 7.27 shows the partially colored structure digraph of this set of equations.

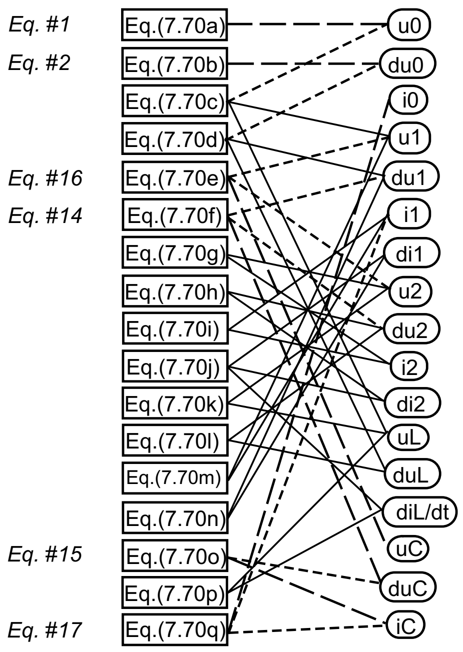

FIGURE 7.27. Partially colored structure digraph of electrical RLC circuit without node potentials.

It worked. The constraint equation has indeed disappeared. Instead we are now facing an algebraic loop in 11 equations and 11 unknowns.

Let us analyze this algebraic loop further, as this analysis will unveil yet another difficulty. The partially causalized equations can be written as follows:

$$u_0 = f(t) \tag{7.71a}$$

$$du_0 = \frac{df(t)}{dt} \tag{7.71b}$$

$$u_L + u_1 = u_0 \tag{7.71c}$$

$$du_L + du_1 = du_0 \tag{7.71d}$$

$$u_2 - R_2 \cdot i_2 = 0 \tag{7.71e}$$

$$du_2 - R_2 \cdot di_2 = 0 \tag{7.71f}$$

$$i_1 - i_2 = i_L \tag{7.71g}$$

$$di_1 - di_2 - \frac{di_L}{dt} = 0 \tag{7.71h}$$

$$u_L - u_2 = 0 \tag{7.71i}$$

$$du_L - du_2 = 0 \tag{7.71j}$$

$$u_1 - R_1 \cdot i_1 = 0 \tag{7.71k}$$
$$du_1 - R_1 \cdot di_1 = 0 \tag{7.71l}$$
$$u_L - l \cdot \frac{di_L}{dt} = 0 \tag{7.71m}$$
$$du_C = du_2 + du_1 \tag{7.71n}$$
$$i_C = C \cdot du_C \tag{7.71o}$$
$$u_C = u_2 + u_1 \tag{7.71p}$$
$$i_0 = i_1 + i_C \tag{7.71q}$$

Figure 7.28 shows the structure digraph of the algebraic subsystem after selecting a tearing variable and a residual equation.

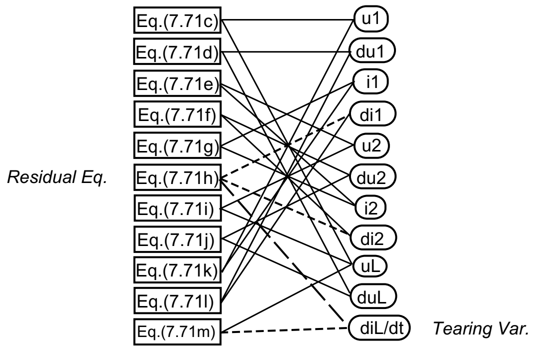

FIGURE 7.28. Structure digraph of algebraic subsystem of electrical RLC circuit after choosing a tearing variable and a residual equation.

We proceed with the usual graph–coloring algorithm. The partially colored structure digraph is shown in Fig.7.29.

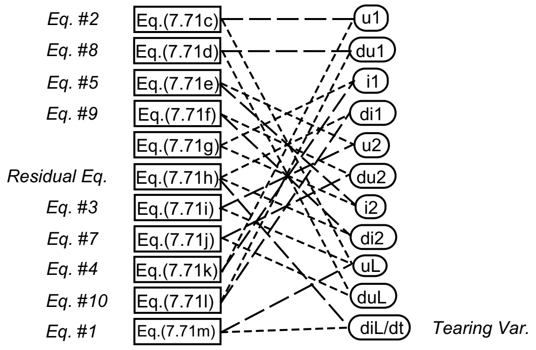

FIGURE 7.29. Partially colored structure digraph of algebraic subsystem of electrical RLC circuit.

We seem to again have ended up with a structural singularity. Equation (7.71g) is a constraint equation, whereas we are lacking an equation to compute the variable du_L.

Yet, this is a very different problem from the one discussed before. This constraint was caused by a poor selection of a tearing variable. Had we chosen a different tearing variable or a different residual equation, this problem would not have occurred. For this reason, we cannot simplify the heuristic procedure further. It is insufficient to look at the number of black (solid) lines attached to equations and the number of black (solid) lines attached to variables when selecting the residual equation and the tearing variable. For each proposed selection, we must pursue the consequences of that selection all the way to the end and be prepared to backtrack if we end up with a conflict.

Let us select a different tearing variable. The new selection is shown in Fig.7.30.

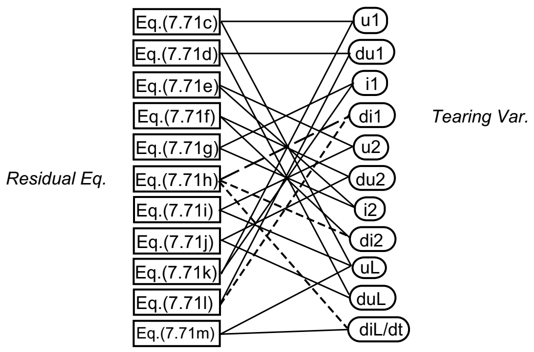

FIGURE 7.30. Structure digraph of algebraic subsystem of electrical RLC circuit after choosing a tearing variable and a residual equation.

The partially colored structure digraph is shown in Fig.7.31.

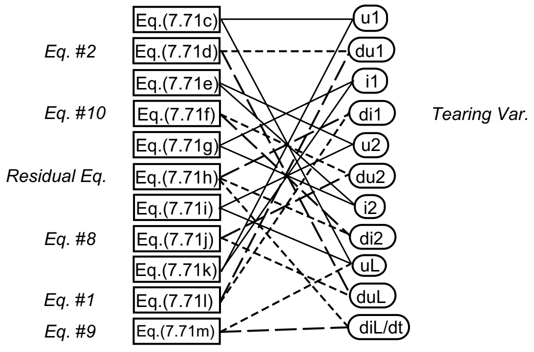

FIGURE 7.31. Partially colored structure digraph of algebraic subsystem of electrical RLC circuit.

We were able to causalize six of the eleven equations. We thus need to select a second residual equation and a second tearing variable, in order to complete the causalization of the algebraic equation system.

Dymola implements the Pantelides algorithm essentially in the form explained in this section. However as almost always, the devil is in the detail. For didactic reasons, we explained the algorithm by starting out with an individual constraint equation, which we differentiated and added to the set of equations. We then chose a pseudo–derivative, in order to ensure that we once again had the same number of unknowns as equations. We then checked, whether additional equations needed to be differentiated as well, since new pseudo–derivatives had been introduced in the process. Yet, this procedure already got us into trouble in one of the examples. For this reason, Dymola first determines *all* equations that need to be differentiated, and chooses the dummy derivative only in the very end.

Furthermore, a fixed choice of a pseudo–derivative may occasionally lead to a division by zero at run time. In fact, it can happen that no fixed choice of a pseudo–derivative avoids divisions by zero. In those cases, Dymola makes the choice of the state variables *dynamic*, switching from one selection to another during the course of the simulation run [7.14].

7.8 The Solvability Issue

The DAE literature talks about yet another issue, namely that of *solvability* [7.1]. Take for example the following DAE:

$$x - \dot{x}^2 = 0.0 \tag{7.72}$$

Converting Eq.(7.72) to ODE form, we obtain:

$$\dot{x} = \pm\sqrt{x} \tag{7.73}$$

Evidently, this ODE has only a real–valued solution as long as the initial value of x is positive. This constraint existed even in the DAE case. However, in the DAE formulation, the situation has become worse. The DAE formulation does not give us any hint, which of the two roots we should select. If we choose the positive root, $\dot{x}$ will also be positive, and x will keep growing. However, if we choose the negative root, $\dot{x}$ is negative, and x will decrease. Both solutions satisfy the DAE, and if the only information we have is the DAE, we can't tell which solution is for real. Even worse, it could happen that we should choose the positive root during some period of time, and the negative root during another. Thus, at any moment in time, we obtain a potential bifurcation in the solution depending on whether we choose the positive or the negative root.

To us, solvability is a non–issue. It is the typical worry of a mathematician who puts the mathematical formulation first, and then tries to interpret the ramifications of that formulation. Remember what we wrote earlier: mathematics is simply the language of physics. The reason why we are interested in differential equations and solving them is that we wish to gain a better understanding of physical phenomena in this universe. Consequently, the origin of our interest is always physics, not mathematics. Physics does not provide us with unsolvable riddles. Saying that a DAE is unsolvable is equivalent to saying that the phenomenon described by it is "defying causality" in the sense that the outcome of an experiment is non–deterministic, which in turn is almost equivalent to saying that the phenomenon is non–physical. True, chaos is for real [7.5]. We can observe chaotic phenomena in physics every day. However, chaotic phenomena are not described through unsolvable differential equations. Chaos only means that the solution in time is undecidable without infinite precision. However, the differential equation that produces a chaotic solution is perfectly deterministic [7.5]. Thus, philosophizing about the implications of solvability or non–solvability of DAEs is like discussing how many angels can dance on the tip of a needle. Or is it not?

Let us look at a simple pendulum as shown on Fig.7.32.

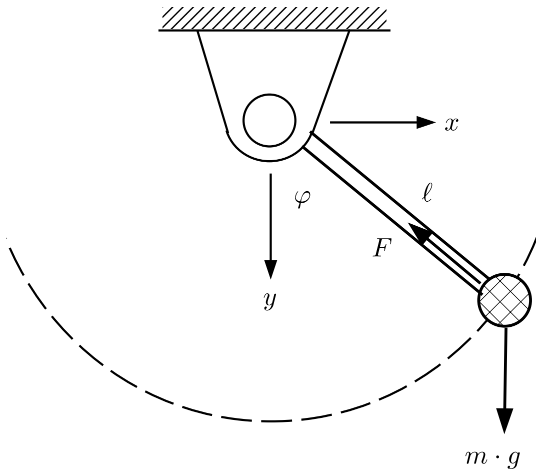

FIGURE 7.32. Mechanical pendulum.

The equations of motion for this pendulum can be described easily in DAE form:

$$m \cdot \frac{dv_x}{dt} = -\frac{F \cdot x}{\ell} \tag{7.74a}$$
$$m \cdot \frac{dv_y}{dt} = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.74b}$$
$$\frac{dx}{dt} = v_x \tag{7.74c}$$
$$\frac{dy}{dt} = v_y \tag{7.74d}$$
$$x^2 + y^2 = \ell^2 \tag{7.74e}$$

These are five equations in the five unknowns dv_x/dt, dv_y/dt, dx/dt, dy/dt, and F. The four natural state variables: v_x, v_y, x, and y are assumed known.

We notice at once that Eq.(7.74e) is a constraint equation, since it doesn't contain any of the unknowns. We apply the Pantelides algorithm, and obtain the following set of six equations in six unknowns:

$$m \cdot \frac{dv_x}{dt} = -\frac{F \cdot x}{\ell} \tag{7.75a}$$
$$m \cdot \frac{dv_y}{dt} = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.75b}$$
$$dx = v_x \tag{7.75c}$$
$$\frac{dy}{dt} = v_y \tag{7.75d}$$
$$x^2 + y^2 = \ell^2 \tag{7.75e}$$
$$2 \cdot x \cdot dx + 2 \cdot y \cdot \frac{dy}{dt} = 0 \tag{7.75f}$$

We decided to let go of the integrator for x, thus the six unknowns are dv_x/dt, dv_y/dt, dx, dy/dt, F, and x.

Eq.(7.75e) is no longer a constraint equation, as it can be solved for the new unknown x. Eq.(7.75f) can be solved for the unknown dx, but this leaves Eq.(7.75c) as a new constraint equation.

Evidently, the original problem was an index–3 problem, and the Pantelides algorithm needs to be applied a second time. We obtain the following set of nine equations in nine unknowns:

$$m \cdot dv_x = -\frac{F \cdot x}{\ell} \tag{7.76a}$$
$$m \cdot \frac{dv_y}{dt} = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.76b}$$
$$dx = v_x \tag{7.76c}$$
$$d2x = dv_x \tag{7.76d}$$

$$\frac{dy}{dt} = v_y \tag{7.76e}$$

$$d2y = \frac{dv_y}{dt} \tag{7.76f}$$

$$x^2 + y^2 = \ell^2 \tag{7.76g}$$

$$x \cdot dx + y \cdot \frac{dy}{dt} = 0 \tag{7.76h}$$

$$dx^2 + x \cdot d2x + \left(\frac{dy}{dt}\right)^2 + y \cdot d2y = 0 \tag{7.76i}$$

In the differentiation of Eq.(7.76c), a new variable, $d2x$, was introduced. Thus, the equation defining dx, i.e., Eq.(7.76h), had to be differentiated as well. In that differentiation, again one more new variable, $d2y$, was introduced. Hence the equation defining dy/dt, i.e., Eq.(7.76e), had to be differentiated also. Finally, another integrator had to be eliminated, namely the one defining the variable v_x. The nine unknowns of this equation system are dv_x, x, F, dv_y/dt, dx, v_x, $d2x$, dy/dt, and $d2y$.

This set of equations represents an index–1 DAE problem that can be causalized using the tearing method. Figure 7.33 shows the partially causalized structure digraph of this DAE system.

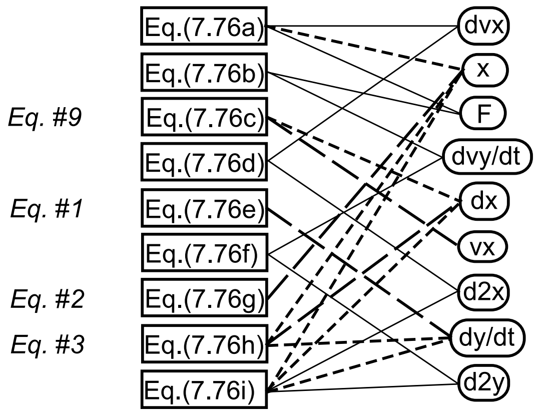

FIGURE 7.33. Partially causalized structure digraph of mechanical pendulum.

An algebraic loop in five equations and five unknowns remains. Figure 7.34 shows the completely causalized structure digraph after a residual equation and a tearing variable have been chosen. Since we have a choice, we decided to select a tearing variable that appears linearly in the residual equation.

We can read out the causal equations from the completely causalized structure digraph of Fig.7.34. They are:

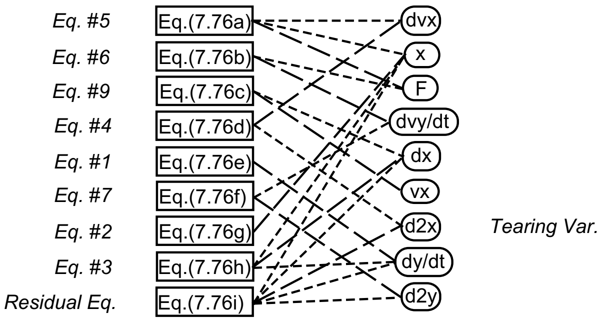

FIGURE 7.34. Completely causalized structure digraph of mechanical pendulum.

$$\frac{dy}{dt} = v_y \tag{7.77a}$$

$$x = \pm\sqrt{\ell^2 - y^2} \tag{7.77b}$$

$$dx = -\frac{y}{x} \cdot \frac{dy}{dt} \tag{7.77c}$$

$$dv_x = d2x \tag{7.77d}$$

$$F = -m \cdot \ell \cdot \frac{dv_x}{x} \tag{7.77e}$$

$$\frac{dv_y}{dt} = g - \frac{F \cdot y}{m \cdot \ell} \tag{7.77f}$$

$$d2y = \frac{dv_y}{dt} \tag{7.77g}$$

$$d2x = -\frac{dx^2 + \left(\frac{dy}{dt}\right)^2 + y \cdot d2y}{x} \tag{7.77h}$$

$$v_x = dx \tag{7.77i}$$

We are facing a new problem. We seem to have come across a *solvability* issue. At any point in time, there are two solutions to Eq.(7.77b), one of which is positive, whereas the other is negative. Yet, the physics behind the pendulum motion doesn't exhibit any ambiguity at all. The pendulum knows exactly, how to swing. It knows that we have to choose the positive root, whenever the pendulum is to the right of the joint, whereas we must choose the negative root otherwise.

Applying the Pantelides algorithm actually made the problem worse. The original index–3 DAE model at least knew that the position of the pendulum cannot jump, since x is the output of an integrator. The reduced index–1 DAE system no longer contains that information.

Evidently, the issue that we are facing here is not related to the physics of the pendulum, but only to the mathematical description thereof, i.e., to the DAE system describing the pendulum motion. Evidently, the simulation model, i.e., the index–1 DAE system, and to a lesser extent even the original index–3 DAE system, offers only an incomplete description of the physical reality.

Physics doesn't know anything about Newton's law. The physical reality of this universe of ours was created long before Mr. Newton was born. What physics cares about are the *conservation principles*: conservation of mass, conservation of energy, and conservation of momentum. Newton's law is one way of indirectly satisfying the conservation of energy principle. Yet for the problem at hand, we also need to conserve the momentum. The DAE system, as specified, does not capture, either directly or indirectly, the need for conserving the momentum.

For the example at hand, the problem can be solved easily by selecting a different set of state variables. Since the pendulum has one mechanical degree of freedom, we need two state variables. It turns out that φ and $\dot{\varphi}$ are a considerably smarter choice of a set of state variables than y and $\dot{y}$.

Unfortunately, the original model does not even contain φ as a variable. We need to add a description of the relationship between the variables currently captured and φ to the model. The easiest may be to replace the original constraint equation, Eq.(7.74e), by a set of two different equations:

$$m \cdot \frac{dv_x}{dt} = -\frac{F \cdot x}{\ell} \tag{7.78a}$$
$$m \cdot \frac{dv_y}{dt} = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.78b}$$
$$\frac{dx}{dt} = v_x \tag{7.78c}$$
$$\frac{dy}{dt} = v_y \tag{7.78d}$$
$$x = \ell \cdot \sin(\varphi) \tag{7.78e}$$
$$y = \ell \cdot \cos(\varphi) \tag{7.78f}$$

These are six equations in the six unknowns dv_x/dt, dv_y/dt, dx/dt, dy/dt, F, and φ.

Since x and y are initially known variables, we can solve Eq.(7.78e) for φ, which then makes Eq.(7.78f) a constraint equation. Left to its own devices, the Pantelides algorithm will differentiate the constraint equation, while letting go of the integrator for y. In the process of differentiation, a new *algebraic* variable, $d\varphi$, is created, and therefore, Eq.(7.78e) needs to be differentiated as well:

$$m \cdot \frac{dv_x}{dt} = -\frac{F \cdot x}{\ell} \tag{7.79a}$$
$$m \cdot \frac{dv_y}{dt} = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.79b}$$

$$\frac{dx}{dt} = v_x \tag{7.79c}$$
$$dy = v_y \tag{7.79d}$$
$$x = \ell \cdot \sin(\varphi) \tag{7.79e}$$
$$\frac{dx}{dt} = \ell \cdot \cos(\varphi) \cdot d\varphi \tag{7.79f}$$
$$y = \ell \cdot \cos(\varphi) \tag{7.79g}$$
$$dy = -\ell \cdot \sin(\varphi) \cdot d\varphi \tag{7.79h}$$

The Pantelides algorithm has no reason to select φ as a state variable on its own. It needs help. In Dymola [7.9], we can offer a *choice of preferred state variables* to the Pantelides algorithm. If we tell the algorithm that we wish to keep φ as a state variable, a true state derivative, $d\varphi/dt$, will be generated in the process of differentiation in place of the algebraic variable, $d\varphi$. The result of the operation will be:

$$m \cdot \frac{dv_x}{dt} = -\frac{F \cdot x}{\ell} \tag{7.80a}$$
$$m \cdot \frac{dv_y}{dt} = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.80b}$$
$$\frac{dx}{dt} = v_x \tag{7.80c}$$
$$dy = v_y \tag{7.80d}$$
$$x = \ell \cdot \sin(\varphi) \tag{7.80e}$$
$$y = \ell \cdot \cos(\varphi) \tag{7.80f}$$
$$dy = -\ell \cdot \sin(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.80g}$$

These are now seven equations in the seven unknowns dv_x/dt, dv_y/dt, dx/dt, dy, F, y, and $d\varphi/dt$. Since the integrator for y was eliminated, y is now an additional unknown. $d\varphi/dt$ was added as another unknown, but φ is no longer an unknown, since it is now the output of an integrator.

Since φ is now a known variable, Eq.(7.80e) has become a new constraint equation that needs to be differentiated. The result of this operation is:

$$m \cdot \frac{dv_x}{dt} = -\frac{F \cdot x}{\ell} \tag{7.81a}$$
$$m \cdot \frac{dv_y}{dt} = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.81b}$$
$$dx = v_x \tag{7.81c}$$
$$dy = v_y \tag{7.81d}$$
$$x = \ell \cdot \sin(\varphi) \tag{7.81e}$$

$$dx = \ell \cdot \cos(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.81f}$$
$$y = \ell \cdot \cos(\varphi) \tag{7.81g}$$
$$dy = -\ell \cdot \sin(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.81h}$$

We now have eight equations in the eight unknowns dv_x/dt, dv_y/dt, dx, dy, F, x, y, and $d\varphi/dt$. A second integrator, the one defining variable x was thrown out in the process.

Since v_x and v_y are still known variables, Eqs.(7.81c–d) need to be solved for dx and dy, respectively. We can then solve Eq.(7.81f) for $d\varphi/dt$, and consequently, Eq.(7.81h) has become a new constraint equation that needs to be differentiated. Since we told the Pantelides algorithm that we wish to preserve $d\varphi/dt$ as a state variable, a true state derivative, $d^2\varphi/dt^2$ is generated in the process of differentiation. The result of the operation is:

$$m \cdot \frac{dv_x}{dt} = -\frac{F \cdot x}{\ell} \tag{7.82a}$$
$$m \cdot dv_y = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.82b}$$
$$dx = v_x \tag{7.82c}$$
$$dy = v_y \tag{7.82d}$$
$$d2y = dv_y \tag{7.82e}$$
$$x = \ell \cdot \sin(\varphi) \tag{7.82f}$$
$$dx = \ell \cdot \cos(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.82g}$$
$$y = \ell \cdot \cos(\varphi) \tag{7.82h}$$
$$dy = -\ell \cdot \sin(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.82i}$$
$$d2y = -\ell \cdot \sin(\varphi) \cdot \frac{d^2\varphi}{dt^2} - \ell \cdot \cos(\varphi) \cdot \left(\frac{d\varphi}{dt}\right)^2 \tag{7.82j}$$

By now, we have 10 equations in the 10 unknowns dv_x/dt, dv_y, dx, dy, F, x, y, v_y, $d^2\varphi/dt^2$, and $d2y$. While differentiating the constraint equation, a new algebraic variable, $d2y$, was introduced. Hence the equation defining dy, Eq.(7.82d) had to be differentiated as well. This pointed to the integrator to be thrown out. It is the integrator defining v_y. Hence variable v_y has now also become an unknown. $d^2\varphi/dt^2$ was added as another unknown replacing the former unknown $d\varphi/dt$, which has now become a known variable, since it is the output of an integrator.

Since $d\varphi/dt$ is now a known variable, yet another constraint equation was introduced. It is Eq.(7.82g). This equation needs to be differentiated as well. The result of the operation is:

$$m \cdot dv_x = -\frac{F \cdot x}{\ell} \tag{7.83a}$$

$$m \cdot dv_y = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.83b}$$

$$dx = v_x \tag{7.83c}$$

$$d2x = dv_x \tag{7.83d}$$

$$dy = v_y \tag{7.83e}$$

$$d2y = dv_y \tag{7.83f}$$

$$x = \ell \cdot \sin(\varphi) \tag{7.83g}$$

$$dx = \ell \cdot \cos(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.83h}$$

$$d2x = \ell \cdot \cos(\varphi) \cdot \frac{d^2\varphi}{dt^2} - \ell \cdot \sin(\varphi) \cdot \left(\frac{d\varphi}{dt}\right)^2 \tag{7.83i}$$

$$y = \ell \cdot \cos(\varphi) \tag{7.83j}$$

$$dy = -\ell \cdot \sin(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.83k}$$

$$d2y = -\ell \cdot \sin(\varphi) \cdot \frac{d^2\varphi}{dt^2} - \ell \cdot \cos(\varphi) \cdot \left(\frac{d\varphi}{dt}\right)^2 \tag{7.83l}$$

This is the final set of 12 equations in the 12 unknowns dv_x, dv_y, dx, dy, F, x, y, v_x, v_y, $d^2\varphi/dt^2$, $d2x$, and $d2y$. It constitutes an implicit index–1 DAE system.

Dymola [7.9] performs one more level of symbolic preprocessing. If it finds a *trivial equation* of the type $a = b$, it throws it out, keeps only one of the variables in the model, and replaces all occurrences of the other by the former. This operation results in:

$$m \cdot dv_x = -\frac{F \cdot x}{\ell} \tag{7.84a}$$

$$m \cdot dv_y = m \cdot g - \frac{F \cdot y}{\ell} \tag{7.84b}$$

$$x = \ell \cdot \sin(\varphi) \tag{7.84c}$$

$$v_x = \ell \cdot \cos(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.84d}$$

$$dv_x = \ell \cdot \cos(\varphi) \cdot \frac{d^2\varphi}{dt^2} - \ell \cdot \sin(\varphi) \cdot \left(\frac{d\varphi}{dt}\right)^2 \tag{7.84e}$$

$$y = \ell \cdot \cos(\varphi) \tag{7.84f}$$

$$v_y = -\ell \cdot \sin(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.84g}$$

$$dv_y = -\ell \cdot \sin(\varphi) \cdot \frac{d^2\varphi}{dt^2} - \ell \cdot \cos(\varphi) \cdot \left(\frac{d\varphi}{dt}\right)^2 \tag{7.84h}$$

Hence we end up with a set of eight equations in the eight unknowns dv_x, dv_y, F, x, y, v_x, v_y, and $d^2\varphi/dt^2$.

Figure 7.35 shows the partially causalized structure diagram of this system.

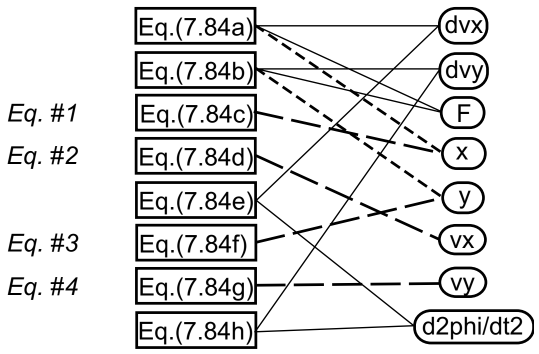

FIGURE 7.35. Partially causalized structure digraph of mechanical pendulum.

An algebraic loop in four equations and four unknowns remains. Figure 7.36 shows the completely causalized structure digraph after a suitable residual equation and tearing variable have been chosen.

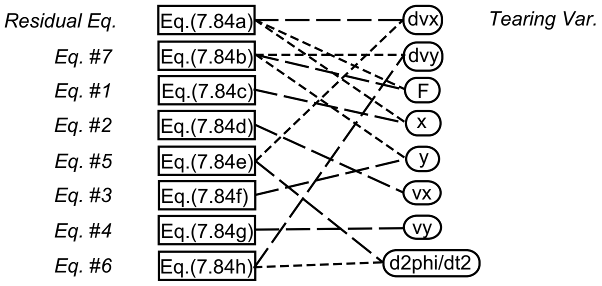

FIGURE 7.36. Completely causalized structure digraph of mechanical pendulum.

The causal equations can be read out of the structure digraph of Fig.7.36. They are:

$$x = \ell \cdot \sin(\varphi) \tag{7.85a}$$

$$v_x = \ell \cdot \cos(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.85b}$$

$$y = \ell \cdot \cos(\varphi) \tag{7.85c}$$

$$v_y = -\ell \cdot \sin(\varphi) \cdot \frac{d\varphi}{dt} \tag{7.85d}$$

$$\frac{d^2\varphi}{dt^2} = \frac{dv_x}{\ell \cdot \cos(\varphi)} + \frac{\sin(\varphi)}{\cos(\varphi)} \cdot \left(\frac{d\varphi}{dt}\right)^2 \tag{7.85e}$$

$$dv_y = -\ell \cdot \sin(\varphi) \cdot \frac{d^2\varphi}{dt^2} - \ell \cdot \cos(\varphi) \cdot \left(\frac{d\varphi}{dt}\right)^2 \tag{7.85f}$$

$$F = \frac{m \cdot g \cdot \ell}{y} - \frac{m \cdot \ell \cdot dv_y}{y} \tag{7.85g}$$

$$dv_x = -\frac{F \cdot x}{m \cdot \ell} \tag{7.85h}$$

With this choice of the set of state variables, all of the equations are linear in the variables they are being solved for. Consequently, there is no ambiguity, and the solvability problem has disappeared. This model can be simulated without difficulties for all values of φ and $\dot{\varphi}$. Clearly, the solvability issue was not related to the physics of the pendulum motion at all. It was purely a mathematical artifact caused by an unfortunate selection of state variables.

Of course, it would have been a yet better idea to formulate Newton's law directly in rotational coordinates. In that case, the resulting model would have been of index 1 or lower right from the beginning, and we would not have had to invoke the Pantelides algorithm at all.

Does this approach resolve all solvability issues in modeling mechanical systems? Unfortunately, this question must be answered in the negative. For multibody systems without closed kinematic loops, i.e., for tree–structured robots, it is always possible to avoid all solvability issues by choosing the relative positions and velocities of the joints as state variables. However, the same does no longer hold true for multibody systems with closed kinematic loops. The kinematic loops lead to large and highly nonlinear algebraic loops that must be solved by tearing. It is not always possible to choose the residual equations and tearing variables of these loops such that all loop equations are linear in the variables that they need to be solved for.

In fact, there exist fairly simple mechanical devices with closed kinematic loops, for which it can be shown that there does not exist a minimal set of state variables, in which all solvability issues can be avoided. One way how such problems have been dealt with in the past is by selecting redundant state variables together with some switching mechanisms that decide when to use which variables during the simulation.

This is precisely, what Dymola now does on its own. Whenever there is a potential problem with a fixed selection of state variables, Dymola postpones the decision until run time [7.14]. For the same reason, newer versions of Dymola will be perfectly capable of simulating the pendulum problem in its original formulation without any help from the user. Dymola recognizes the potential solvability issue, postpones the selection of states, and toggles between x and y at run time as needed.

We shall deal with switching models in Chapter 9 of this book. In Chapter 8, we shall look at these problems from yet another angle.

7.9 Summary

In this chapter, we have presented a number of interlinked algorithms that can be used to convert even higher–index DAE systems to ODE form.

The most central among these algorithms is the algorithm by Tarjan, an algorithm based on graph theory to partially sort a DAE system both horizontally and vertically. The algorithm also finds minimal subsets of algebraically coupled equation systems that need to be solved simultaneously. Although the algorithm is based on graph theory, it can be easily implemented algebraically using linked lists. The algorithm furthermore discovers constraint equations, i.e., can be used to detect higher–index problems.

If a higher–index problem has been detected, the algorithm by Pantelides can be employed to reduce the perturbation index, until all structural singularities have been resolved.

A heuristic procedure has been presented that allows to find suitable tearing variables for the algebraically coupled subsystems.

The algorithms presented in this chapter are similar to those that have been implemented in the model compiler of *Dymola* [7.8, 7.9], an object–oriented physical system modeling and simulation environment.

The algorithms are highly computationally efficient and well tested. Dymola is capable of converting DAE systems consisting of tens of thousands of equations to ODE form within seconds on a modern PC, while applying these algorithms.

7.10 References

[7.1] Kathryn E. Brenan, Stephen L. Campbell, and Linda R. Petzold. *Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations.* North–Holland, New York, 1989. 256p.

[7.2] Pawel Bujakiewicz. *Maximum Weighted Matching for High Index Differential Algebraic Equations.* PhD thesis, Delft Institute of Technology, The Netherlands, 1995.

[7.3] Stephen L. Campbell and C. William Gear. The Index of General Nonlinear DAEs. *Numerische Mathematik*, 72:173–196, 1995.

[7.4] François E. Cellier and Hilding Elmqvist. Automated Formula Manipulation Supports Object–oriented Continuous System Modeling. *IEEE Control Systems*, 13(2):28–38, 1993.

[7.5] François E. Cellier. *Continuous System Modeling.* Springer Verlag, New York, 1991. 755p.

[7.6] Iain S. Duff, Albert M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, Oxford, United Kingdom, 1986. 341p.

[7.7] Hilding Elmqvist and Martin Otter. Methods for Tearing Systems of Equations in Object–oriented Modeling. In *Proceedings European Simulation Multiconference*, pages 326–332, Barcelona, Spain, 1994.

[7.8] Hilding Elmqvist. *A Structured Model Language for Large Continuous Systems.* PhD thesis, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978.

[7.9] Hilding Elmqvist. *Dymola — Dynamic Modeling Language, User's Manual.* DynaSim AB, Research Park Ideon, Lund, Sweden, 2004.

[7.10] Ernst Hairer, Christian Lubich, and Michel Roche. *The Numerical Solution of Differential–Algebraic Systems by Runge–Kutta Methods.* Springer–Verlag, Berlin, Germany, 1989. 139p.

[7.11] Johann Joss. *Algorithmisches Differenzieren.* PhD thesis, Diss ETH 5757, Swiss Federal Institute of Technology, Zürich, Switzerland, 1976. 69p.

[7.12] Gabriel Kron. *Diakoptics: The Piecewise Solution of Large–Scale Systems.* Macdonald Publishing, London, United Kingdom, 1963. 166p.

[7.13] Richard S. H. Mah. *Chemical Process Structures and Information Flows.* Butterworth Publishing, London, United Kingdom, 1990. 500p.

[7.14] Sven Erik Mattsson, Hans Olsson, and Hilding Elmqvist. Dynamic Selection of States in Dymola. In *Proceedings Modelica Workshop*, pages 61–67, Lund, Sweden, 2000.

[7.15] Sven Erik Mattsson and Gustaf Söderlind. Index Reduction in Differential–Algebraic Equations Using Dummy Derivatives. *SIAM Journal on Scientific Computing*, 14(3):677–692, 1993.

[7.16] Martin Otter, Hilding Elmqvist, and François E. Cellier. Modeling of Multibody Systems with the Object–Oriented Modeling Language Dymola. *J. Nonlinear Dynamics*, 9(1):91–112, 1996.

[7.17] Martin Otter, Hilding Elmqvist, and François E. Cellier. 'Relaxing' – A Symbolic Sparse Matrix Method Exploiting the Model Structure in Generating Efficient Simulation Code. In *Proceedings Symposium on Modeling, Analysis, and Simulation, CESA'96, IMACS Multi-Conference on Computational Engineering in Systems Applications*, volume 1, pages 1–12, Lille, France, 1996.

[7.18] Martin Otter and Clemens Schlegel. Symbolic generation of efficient simulation codes for robots. In *Proceedings Second European Simulation Multi–Conference*, pages 119–122, Nice, France, 1988.

[7.19] Martin Otter. *Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter.* PhD thesis, Dept. of Mech. Engr., Ruhr–University Bochum, Germany, 1994.

[7.20] Constantinos Pantelides. The Consistent Initialization of of Differential–Algebraic Systems. *SIAM Journal of Scientific and Statistical Computing*, 9(2):213–231, 1988.

[7.21] Robert Tarjan. Depth–first search and linear graph algorithms. *SIAM Journal of Computation*, 1(2):146–160, 1972.

7.11 Homework Problems

[H7.1] Electrical Circuit, Horizontal and Vertical Sorting

Given the electrical circuit shown in Fig.H7.1a.

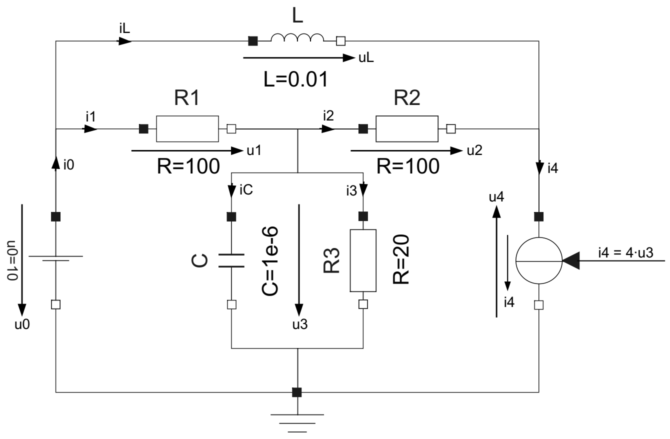

FIGURE H7.1a. Electrical circuit.

The circuit contains a constant voltage source, u_0, and a dependent current source, i_4, that depends on the voltage across the capacitor, C, and the resistor, R_3.

Write down the element equations for the seven circuit elements. Since the voltage u_3 is common to two circuit elements, these equations contain 13 rather than 14 unknowns. Add the voltage equations for the three meshes and the current equations for three of the four nodes. One current equation is redundant. Usually, the current equation for the ground node is therefore omitted.

In this way, you end up with 13 equations in the 13 unknowns. Draw the structure digraph of the DAE system, and apply the Tarjan algorithm to sort the equations both horizontally and vertically. Write down the causal equations, i.e., the resulting ODE system.

Simulate the ODE system across 50 μsec using RKF4/5 with zero initial conditions on both the capacitor and the inductor.

Plot the voltage u_3 and the current i_C on two separate subplots as functions of time.

[H7.2] Horizontal and Vertical Sorting, Newton Iteration

Given the following model in three nonlinear equations and three unknowns:

$$\mathcal{F}_1(x_1, x_3) = 0.0 \tag{H7.2a}$$
$$\mathcal{F}_2(x_2) = 0.0 \tag{H7.2b}$$
$$\mathcal{F}_3(x_1, x_2) = 0.0 \tag{H7.2c}$$

Write down the structure incidence matrix, $\mathbf{S}$, of this nonlinear model.

Draw the structure digraph, and sort the equations both horizontally and vertically using the Tarjan algorithm.

Write down the causal equations and their structure incidence matrix, $\hat{\mathbf{S}}$, which should now be in lower–triangular form.

Find two permutation matrices, $\mathbf{P}$ and $\mathbf{Q}$, such that:

$$\hat{\mathbf{S}} = \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{Q} \tag{H7.2d}$$

A *permutation matrix* is a matrix, in which every row and column contains exactly one element with a value of 1, whereas all other elements have values of 0.

As the structure incidence matrix, $\hat{\mathbf{S}}$, is in lower–triangular form, we can set the simulation up by specifying three Newton iterations in one variable each, rather than one Newton iteration in three variables. This is much more economical.

Set up the Newton iterations by introducing symbolic functions denoting the Hessians.

[H7.3] Hydraulic System, Algebraic Differentiation

Given the hydraulic system shown in Fig.H7.3a.

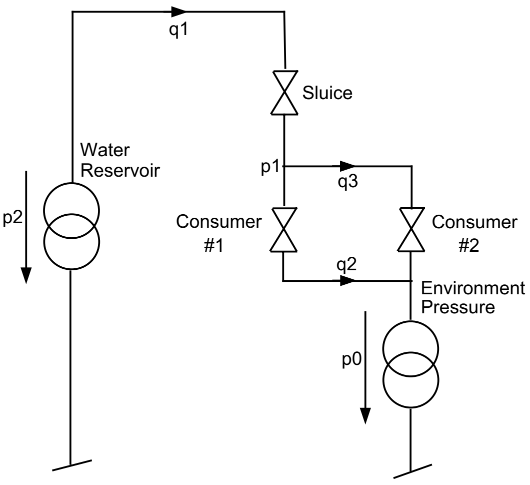

FIGURE H7.3a. Hydraulic system.

A water reservoir generates a pressure of p_2. A sluice reduces the pressure to p_1, which is the water pressure that the consumers see. p_0 is the pressure of the environment, i.e., the air pressure.

The sluice and the consumers can be represented by nonlinear turbulent resistance elements. The turbulent hydraulic resistance characteristic is shown in Fig.H7.3b.

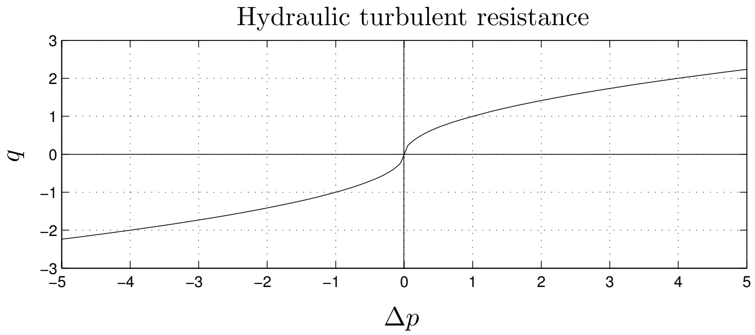

FIGURE H7.3b. Turbulent hydraulic resistance.

It shows the relationship between the pressure drop, Δp, and the flow rate, q. Mathematically, the relationship can be described by the formula:

$$q = k \cdot \text{sign}(\Delta p) \cdot \sqrt{|\Delta p|} \tag{H7.3a}$$

or if the inverse computational causality is required:

$$\Delta p = \frac{1}{k} \cdot \text{sign}(q) \cdot q^2 \tag{H7.3b}$$

Write down the nonlinear equations describing this system. You need six equations in the six unknowns p_0, p_1, p_2, q_1, q_2, and q_3.

Draw the structure digraph and isolate the nonlinear algebraic loop. You'll find an algebraic loop in four equations and four unknowns.

We shall first solve this equation system directly, i.e., without tearing. Set up a vector zero function, $\mathcal{F}(\mathbf{x})$ of length four, where the vector $\mathbf{x}$ stands for the four unknowns of the algebraic equation system. Find a symbolic expression for the Hessian, $\mathcal{H}(\mathbf{x})$, which is a matrix of dimensions 4×4. Write down the linear equation system that needs to be solved once per iteration step of the Newton iteration.

We now repeat the problem, this time with a tearing approach. Choose an appropriate tearing variable and residual equation, and causalize the equation system. Set up an appropriate scalar zero function in the tearing variable, and set up the Newton iteration such that it iterates over all equations, yet only uses the tearing variable as an iteration variable. Find a symbolic expression for the Hessian, which is now also a scalar. Use algebraic differentiation to compute the Hessian. Since the Newton iteration is scalar, you can come up with a closed–form symbolic expression for the next iteration of the tearing variable.

[H7.4] Linear System, Newton Iteration

Given the linear equation system:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{H7.4a}$$

$\mathbf{A}$ is assumed to be a nonsingular square matrix.

We wish to solve for the unknown vector $\mathbf{x}$ by Newton iteration. Set up the Newton iteration using symbolic expressions for the Jacobian and the Hessian. Prove that the Newton iteration indeed converges to the correct solution within a single step for arbitrary initial conditions.

[H7.5] Electrical Circuit, Tearing

Given the electrical circuit shown in Fig.H7.5a.

We wish to find a symbolic expression for the current i_3 as a function of the input voltage u_0 and the five resistance values.

Write down all the equations governing this circuit. Draw the structure digraph. You end up with an algebraic equation system in 10 equations and 10 unknowns. Use the heuristic procedure presented in this chapter to find appropriate tearing variables. You'll need two of them.

Use the substitution technique to come up with two symbolic expressions in the two tearing variables. These can be solved symbolically by matrix inversion. If one of the tearing variables was the current i_3, you are done. Otherwise, find a symbolic expression for i_3 in function of the two tearing variables, and substitute the previously found expressions once more into that new expression.

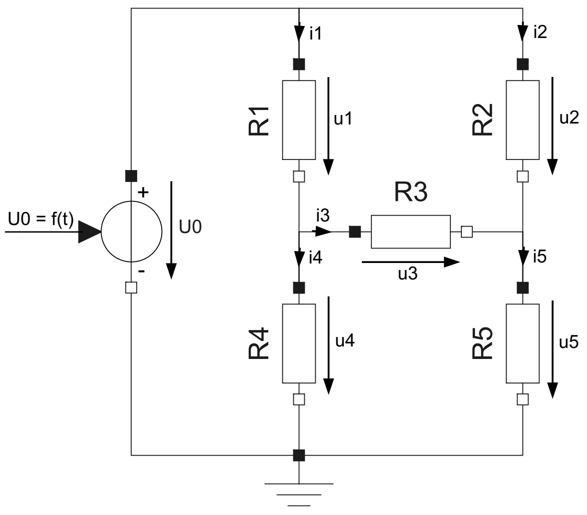

FIGURE H7.5a. Electrical resistance circuit.

[H7.6] Electrical Circuit, Relaxation

We wish to solve Problem [H7.5] once more, but this time using the relaxation algorithm.

Using the tearing structure found in Problem [H7.5], write the 10 equations in 10 unknowns in a matrix–vector form, such that you obtain only two non–zero elements above the diagonal of the matrix.

Apply symbolic Gaussian elimination without pivoting to this system to come up with a sequence of expressions to compute the tearing variables. At the end, use substitution to reduce this sequence of symbolic expressions to two expressions for the two tearing variables.

[H7.7] Electrical Circuit, Structural Singularity

Given the circuit shown in Fig.H7.7a containing three sinusoidal current sources.

Write down the complete set of equations describing this circuit. Draw the structure digraph and begin causalizing the equations. Determine a constraint equation.

Apply the Pantelides algorithm to reduce the perturbation index to 1. Then apply the tearing algorithm with substitution to bring the perturbation index down to 0.

Write down the structure incidence matrices of the index–1 DAE and the index–0 ODE systems, and show that they are in BLT form, and in LT form, respectively.

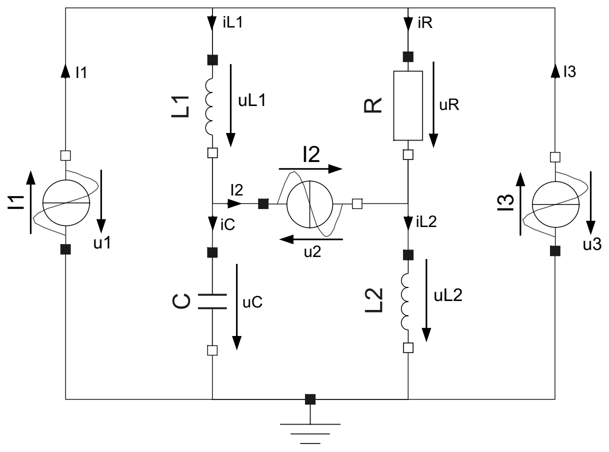

FIGURE H7.7a. Electrical structurally singular circuit.

[H7.8] Chemical Reactions, Pantelides Algorithm

The following set of DAEs:

$$\frac{dC}{dt} = K_1(C_0 - C) - R \tag{H7.8a}$$

$$\frac{dT}{dt} = K_1(T_0 - T) + K_2 R - K_3(T - T_C) \tag{H7.8b}$$

$$0 = R - K_3 \exp\left(\frac{-K_4}{T}\right) C \tag{H7.8c}$$

$$0 = C - u \tag{H7.8d}$$

describes a chemical isomerization reaction. C is the reactant concentration, T is the reactant temperature, and R is the reactant rate per unit volume. C_0 is the feed reactant concentration, and T_0 is the feed reactant temperature. u is the desired concentration, and T_C is the control temperature that we need to produce u. We want to turn the problem around (inverse model control) and determine the necessary control temperature T_C as a function of the desired concentration u. Thus, u will be an input to our model, and T_C is the output. The problem formulation was taken right out of [7.1].

Draw the structure digraph. You shall notice at once that one of the equations, Eq.(H7.8d), has no connections to it. Thus, it is a constraint equation that needs to be differentiated, while an integrator associated with the constraint equation needs to be thrown out.

We now have five equations in five unknowns. Draw the enhanced structure digraph, and start causalizing the equations. You shall notice that a second constraint equation appears. Hence the original DAE system had been an index–3 DAE system. Differentiate that constraint equation as well, and throw out the second integrator. In the process, new pseudo–derivatives are introduced that call for additional differentiations.

This time around, you end up with eight equations in eight unknowns. Draw the once more enhanced structure digraph, and causalize the equations. This is an example, in which (by accident) the Pantelides algorithm reduces the perturbation index in one step from 2 to 0, i.e., the final set of equations does not contain an algebraic loop.

Draw a block diagram that shows how the output T_C can be computed from the three inputs u, du/dt, and d^2u/dt^2.

7.12 Projects

[P7.1] Heuristic Procedures for Finding Tearing Variables

Study alternate strategies for finding small sets of tearing variables and residual equations. As the size of a DAE system generated by an object–oriented physical system modeling tool, such as Dymola [7.8, 7.9], can be very large, often containing thousands if not tens of thousands of equations, the computational efficiency of the heuristic procedure is very important.

[P7.2] Computation of Inverse Hessian

In Chapter 6, we have discussed approaches to numerically approximate the Hessian matrix. In this chapter, we have looked at an alternate approach making use of algebraic differentiation.

Study under what conditions it is more economical to approximate the Hessian numerically, and when a symbolic computation using algebraic differentiation should be used.

[P7.3] Solution of Linear Equation Systems

We have presented two different approaches to dealing with the solution of linear equation systems. On the one hand, we have presented a tearing approach, on the other, we have looked at a relaxation technique. Both techniques can be interpreted as symbolic sparse matrix algorithms. They have furthermore much in common. It was shown that the problem of finding small sets of tearing variables is identical to that of finding a small number of non–zero elements above the diagonal of the matrix in the relaxation approach.

Although we have shown by means of a few examples that the remaining linear systems in the tearing variables can be solved by substitution, this technique is not recommendable, as it invariably leads to an explosion in the size of the formulae. An alternate technique was also presented. It may make more sense to iterate over the entire set of equations, while using a Newton iteration on the tearing variables only.

In the case of linear systems, this requires an iteration rather than a closed–form solution, but the iteration may be acceptable as it converges in a single step if the Jacobian and Hessian are computed exactly, e.g. by means of algebraic differentiation.

The relaxation approach, on the other hand, leads to a closed–form solution of the linear equation system without requiring substitution. Hence this approach may be preferable at times.

Study under which conditions Newton iteration of a linear system is more economical, and when a relaxation approach may be cheaper.

7.13 Research

[R7.1] Pantelides and Small Equation Systems

In the modeling of multi–body systems (MBS), extensive research has focused on the generation of small sets of simulation equations. If the state variables are chosen carelessly in modeling a tree–structured robot, the number of simulation equations grows with the fourth power of the number of degrees of freedom (i.e., the number of articulations) of the robot. Yet, it is possible to choose the state variables such that the number of simulation equations grows only linearly in the number of articulations. Algorithms that behave in this fashion are called *order–n* algorithms in the literature [7.18, 7.19].

For this reason, the MBS library of Dymola [7.8, 7.9], which was developed by Martin Otter, does not make use of the Pantelides algorithm to resolve structural singularities. Instead, the model equations are formulated such that the structural singularities are resolved manually already at the time of the model formulation.

This places a heavy burden on the modeler. It would be better if the Pantelides algorithm could be made smart enough so that it would select the integrators to be thrown out such that an equation system is generated that is as small as possible.

Furthermore, the approach described by Otter only works in the case of tree–structured robots. If the MBS contains kinematic loops, the approach needs to be modified.

Study ways to automate the generation of efficient simulation code when using the Pantelides algorithm for index reduction.

[R7.2] Symbolic Model Compilation and Run–Time Errors

One of the biggest drawbacks of heavy symbolic preprocessing of the model equations in the generation of efficiently executable simulation code lies in the problem of tracing back run–time exceptions to original model equations.

When compiling a Dymola [7.8, 7.9] object–oriented model of a physical system into explicit ODE form, it happens frequently that the user receives an error message at the end of the model compilation of the type: “There are 3724 equations in 3725 unknowns.” Of course, such a model cannot be simulated.

Unfortunately, it may be quite difficult to trace the error message back to the original model. Usually, a connection has been omitted somewhere. Yet, the simulation code no longer contains the information, where the error might be located.

When the equations are made causal, the compiler will tell the user, which is the variable, for which no equation was left over. However, that information may be quite arbitrary.

Similarly, when the simulation dies with a division by zero, it may no longer be easy for the user to recognize, which equation was responsible for the problem, as the error message will point at the simulation code, not at the original model equations. By that time, the equations may have changed their appearance so drastically that they have become unrecognizable.

Furthermore, many of the equations in the final model were not even explicitly present among the original model equations. They were automatically generated from the topological connections among submodels.

Study how the algorithms presented in this chapter can be enhanced so that they preserve as much information as possible about the original model equations for the purpose of presenting the user with error messages in terms that he or she can relate to.